

Б А К А Л А В Р И А Т

*Р.А. Жуков*

**ЯЗЫК  
ПРОГРАММИРОВАНИЯ  
PYTHON  
ПРАКТИКУМ**

У Ч Е Б Н О Е   П О С О Б И Е



Электронно-  
Библиотечная  
Система  
**znanium.com**



Уважаемый читатель!  
Вы держите в руках книгу,  
дополнительные материалы которой  
доступны Вам **БЕСПЛАТНО**  
в интернете на [www.znaniium.com](http://www.znaniium.com)  
Специального программного  
обеспечения не требуется

ВЫСШЕЕ ОБРАЗОВАНИЕ – БАКАЛАВРИАТ

серия основана в 1996 г.



**Р.А. ЖУКОВ**

# ЯЗЫК ПРОГРАММИРОВАНИЯ PYTHON ПРАКТИКУМ

**УЧЕБНОЕ ПОСОБИЕ**

*Рекомендовано Межрегиональным учебно-методическим советом профессионального образования в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки 38.03.05 «Бизнес-информатика» (квалификация (степень) «бакалавр») (протокол № 6 от 25.03.2019)*

Электронно-  
Библиотечная  
Система  
znanium.com

Москва  
ИНФРА-М  
2019

**УДК 004.43(075.8)**  
**ББК 32.973.26-018.1я73**  
**Ж86**

*Рекомендовано к изданию Ученым советом Тульского филиала  
Финансового университета при Правительстве Российской Федерации*

**Рецензенты:**

*Двоенко С.Д.*, доктор физико-математических наук, профессор  
Тульского государственного университета;

*Привалов А.Н.*, доктор технических наук, профессор Тульского го-  
сударственного педагогического университета имени Л.Н. Толстого

**Жуков Р.А.**

**Ж86** Язык программирования Python: практикум : учеб. пособие /  
Р.А. Жуков. — М.: ИНФРА-М, 2019. — 216 с. + Доп. материалы [Элект-  
ронный ресурс; Режим доступа: <http://www.znaniium.com>]. — (Высшее  
образование: Бакалавриат). — [www.dx.doi.org/10.12737/textbook\\_5cb5ca35aaa7f5.89424805](http://www.dx.doi.org/10.12737/textbook_5cb5ca35aaa7f5.89424805).

ISBN 978-5-16-014701-7 (print)


ISBN 978-5-16-107207-3 (online)

Учебное пособие посвящено теоретическому и практическому изу-  
чению современного широко используемого языка программирования  
Python. Состоит из пяти глав, в которых последовательно рассмотрены  
такие вопросы, как история языков программирования, особенности и ос-  
новные элементы языка программирования Python (типы данных; ин-  
струкции, функции, модули; объектно-ориентированное программирова-  
ние; разработка графических интерфейсов). Материал изложен компактно  
с сохранением строгости, алгоритмичности и детальной проработанности  
основных понятий в соответствии с рабочей программой дисциплины  
«Компьютерный практикум».

Соответствует требованиям федеральных государственных образова-  
тельных стандартов высшего образования последнего поколения.

Для студентов бакалавриата направления подготовки «Бизнес-инфор-  
матика», а также всех, кто интересуется программированием.

**УДК 004.43(075.8)**  
**ББК 32.973.26-018.1я73**

Материалы, отмеченные знаком , доступны  
в электронно-библиотечной системе [Znaniium.com](http://Znaniium.com)

ISBN 978-5-16-014701-7 (print)  
ISBN 978-5-16-107207-3 (online)

© Жуков Р.А., 2019

## Предисловие

В современном обществе практически ни в одной сфере деятельности человека невозможно обойтись без использования информационных технологий (ИТ), которые позволяют упростить производственно-хозяйственную деятельность организаций в части совершенствования информационного обмена, обработки структурированных и неструктурированных данных, решения сложных задач экономики, математики, управления и т.п. На рынке труда все больше востребованы ИТ-специалисты, которым предлагают достойный уровень заработной платы (практически самый высокий по сравнению с другими специальностями). При этом работодатели предъявляют серьезные требования к своим работникам, одним из которых является способность к самостоятельной разработке ИТ-приложений, связанных со специальными знаниями в области алгоритмизации и программирования. Именно поэтому актуальным остается вопрос подготовки квалифицированных специалистов, которые будут иметь теоретические и практические знания в области программирования.

Язык Python — один из современных объектно-ориентированных языков программирования, который используют такие ИТ-гиганты, как, например, *Google* и *Yandex*. К тому же, простота и универсальность Python делают его одним из лучших языков программирования. Поэтому неслучайно его изучение включено в основную образовательную программу бакалавриата по направлению 38.03.05 «Бизнес-информатика», профиля «ИТ-менеджмент в бизнесе» в рамках учебной дисциплины «Компьютерный практикум». В результате освоения учебной дисциплины студенты бакалавриата будут:

### **знать**

- типы и структуры данных, используемые в языке Python, технологии обработки, анализа и интерпретации данных различной природы;
- инструкции и конструкции языка программирования Python;
- основные понятия объектно-ориентированного и событийного программирования;
- возможности языков программирования для решения математических и научных задач;
- технологии создания программных решений на современных языках программирования;



### ***уметь***

- выбирать структуры данных и алгоритмы, позволяющие решить поставленную задачу оптимальным способом, применять алгоритмы для поиска и выявления зависимостей в данных;
- создавать собственные функции и классы;
- создавать приложения с графическим интерфейсом;
- использовать библиотеки для решения поставленной задачи;
- формализовывать постановку прикладных задач исследования с целью программирования решения;

### ***владеть***

- навыками решения практических задач с использованием высокоуровневых структур данных;
- навыками использования интегрированных сред разработки для создания программ;
- навыками работы с математическими библиотеками языка Python;
- практическими навыками управления данными, включая различные преобразования данных.

Учебное пособие соответствует рабочей программе дисциплины. Пособие содержит в себе теоретический и практический материал, упражнения и задания для самостоятельной работы.

Первая глава посвящена истории развития языков программирования и основным подходам к алгоритмизации. Рассмотрены особенности и преимущества языка Python, изучен вопрос его установки и представлены базовые инструменты работы в режиме интерпретатора и среде IDLE.

Следующие главы включают изучение основных элементов языка программирования Python: типов данных (глава 2); инструкций, функций, модулей (глава 3); объектно-ориентированного программирования (глава 4) и разработки графических интерфейсов (глава 5).

Учебное пособие может быть интересно широкому кругу читателей.

# Глава 1

## ЯЗЫКИ ПРОГРАММИРОВАНИЯ

### 1.1. ПОНЯТИЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ

*Язык программирования* (ЯП) — это формальная знаковая система, предназначенная для описания команд (инструкций) и данных, которые могут быть обработаны электронно-вычислительной машиной (ЭВМ).

Языки программирования являются *формальными*, или искусственными языками. Они обеспечивают взаимодействие пользователя и ЭВМ.

Формальный язык определен формальной грамматикой. По А.Н. Хомскому<sup>1</sup>, формальные языки классифицируются в соответствии с типами грамматик, которыми они задаются. Языки программирования определяются контекстно-свободными грамматиками при условии, что символами алфавита являются токены (объекты, создающиеся из лексем<sup>2</sup> в процессе лексического анализа), образованные по правилам регулярной грамматики.

*Естественные языки* (ЕЯ) определяются грамматиками общего вида и в этом смысле отличаются от ЯП, хотя языки программирования высокого уровня внешне похожи на ЕЯ, например английский язык [3]. Как и все языки, ЯП имеют собственный алфавит, синтаксис и семантику.

*Алфавит* — это конечный набор символов для конкретного языка программирования.

*Синтаксис* определяет правила образования токенов и правила формирования последовательностей токенов.

*Семантика* — это формальное содержание (смысл) последовательности токенов.

Если в формальных языках построенная фраза грамматически правильна, то это означает, что семантически она тоже правильна.

---

<sup>1</sup> Авраам Ноам Хомский (род. 1928) — американский лингвист, автор классификации формальных языков (иерархия Хомского). Предложил четыре типа формальных грамматик: неограниченные, контекстно-зависимые, контекстно-свободные и регулярные (самые простые).

<sup>2</sup> *Лексема* — минимальная смысловая единица для языка программирования (например: константа, ключевые слова и т.п.). В прикладном программировании токены и лексемы могут не различаться.

## 1.2. РАЗВИТИЕ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Развитие языков программирования неразрывно связано с развитием вычислительной техники и определенными задачами, которые было необходимо решать.

Основателем программирования считают английского математика Чарльза Бэббиджа (1791–1871), который изобрел первую в мире аналитическую вычислительную машину (прототип современных ЭВМ), состоящую из валиков и шестерней, вращающихся с помощью рычага (рис. 1.1).

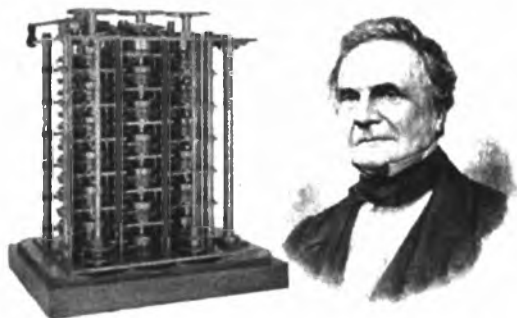


Рис. 1.1. Чарльз Бэббидж и его изобретение

Принципы работы машины были основаны на вычислении таблиц разностным методом. При этом использовалась только операция сложения.

Следующий шаг в области программирования сделал Жозеф Мари Жаккар (1752–1854), который стал использовать перфокарты для алгоритмизации действий ткацких станков (рис. 1.2).

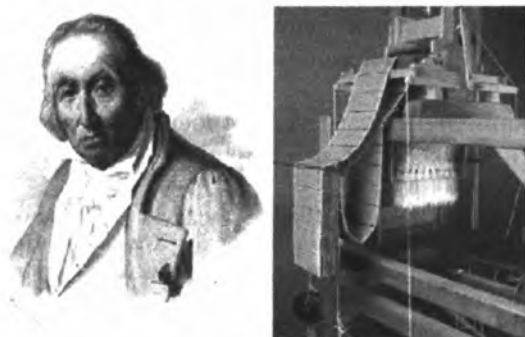
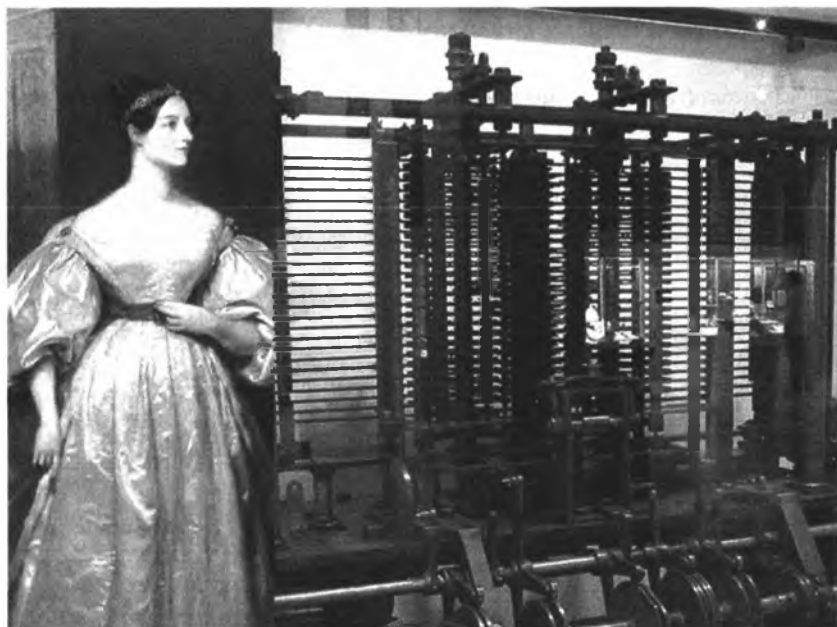


Рис. 1.2. Жозеф Мари Жаккар и ткацкий станок

Первым в истории программистом мировое сообщество считает друга и соратника Бэббиджа Аду Лавлейс (1815–1852). Она разработала описание большой разностной вычислительной машины<sup>1</sup>, составила для нее первую программу, а также ввела такие термины, как «рабочая ячейка» и «цикл» (рис. 1.3). Первая программа включала алгоритм вычисления чисел Бернулли, и ее текст был опубликован в комментариях к запискам о вычислительной машине итальянского инженера Луиджи Манабреа.



**Рис. 1.3.** Ада Лавлейс и большая разностная вычислительная машина

Для того чтобы осуществить программирование таких машин, использовались особые комбинации цифр, которые «понимала» только данная машина. Эти цифровые комбинации называли машинными кодами [2]. Процесс программирования был очень сложным занятием, такое трудоемкое дело могли осилить только специалисты, количество которых было невелико.

---

<sup>1</sup> Технологии того времени не позволили создать машину в таком виде, в котором она задумывалась. Полностью собранную большую разностную машину удалось сконструировать только в 1906 г. компании *Monroe Calculating Machine Co.*

В конце 1940-х гг., одновременно с созданием первых ЭВМ, появились языки программирования первого поколения (языки машинно-ориентированной парадигмы программирования<sup>1</sup>). Они представляли собой машинные инструкции-коды в двоичной системе счисления. Программисту необходимо было знать уникальные для каждой машины инструкции (набор команд), чтобы заставить ЭВМ решить требуемую задачу.

В 1949 г. появляется машинно-ориентированный язык Ассемблера (от англ. *assembler* — сборщик (рис. 1.4)), команды которого строго соответствовали командам машины.

<pre>function Max1(var Arr; N: Integer): Integer; inline( \$59/ \$5E/ \$1F/ \$33/\$C0/ \$BB/\$8001/ \$3B/\$C8/ \$7E/\$09/ \$AD/ \$3B/\$C3/ \$7E/\$02/ \$8B/\$D8/ \$E2/\$F7/ \$8B/\$C3 );</pre>	<pre>function Max2(var Arr; N: Integer): Integer; begin asm LDS SI,Arr адрес Arr в SI MOV CX,N значение N в CX LODSW слово в AX, адрес DS:SI, и SI-2 MOV BX,AX AX в BX MOV AX,11 в AX CMP CX,AX сравнить, AX - BX ( флаг SF ) JLE @3 перейти, если &lt;= 0 SUB CX,1 CX - 1 @1: LODSW слово в AX ... CMP AX,BX сравнить ... JLE @2 перейти..</pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Рис. 1.4.** Фрагмент кода на машинном языке (слева) и языке Ассемблер (справа)

Программировать стало легче за счет введения символьной формы записи команд — мнемокодов. Язык Ассемблера активно применялся в эпоху третьего поколения ЭВМ.

**Справка.** Выделяют пять поколений ЭВМ, которые связывают с элементной базой (составом электронно-вычислительных машин). Первое поколение ЭВМ — на базе электровакуумных ламп (1940-е – 1950-е гг.); второе — транзисторная элементная база (1950-е – 1960-е гг.); третье —

<sup>1</sup> Термин «парадигма программирования» был введен позже, в 1978 г., Робертом Флойдом. Парадигма программирования — это подход к программированию, включающий совокупность идей и понятий, определяющих стиль написания компьютерных программ. Машинно-ориентированную парадигму программирования иногда называют допарадигмальной. Некоторые современные ЯП поддерживают реализацию нескольких парадигм.

интегральные схемы (1960-е – 1970-е гг.); четвертое поколение (микро-ЭВМ) – микропроцессорная база (1970-е гг. – настоящее время); пятое поколение – принципиально новая элементная база, основанная на искусственном интеллекте (начало – 1981 г.).

Язык Ассемблера имеет различные вариации. Из них наиболее известны Ассемблер А86, Microsoft Macro Assembler (MASM), Borland Turbo Assembler (TASM), Watcom Assembler (WASM). Именно на Ассемблере А86 писались программы в бывшем СССР, когда не было персональных ЭВМ.

Языку Ассемблера соответствовала своя среда разработки (рис. 1.5).

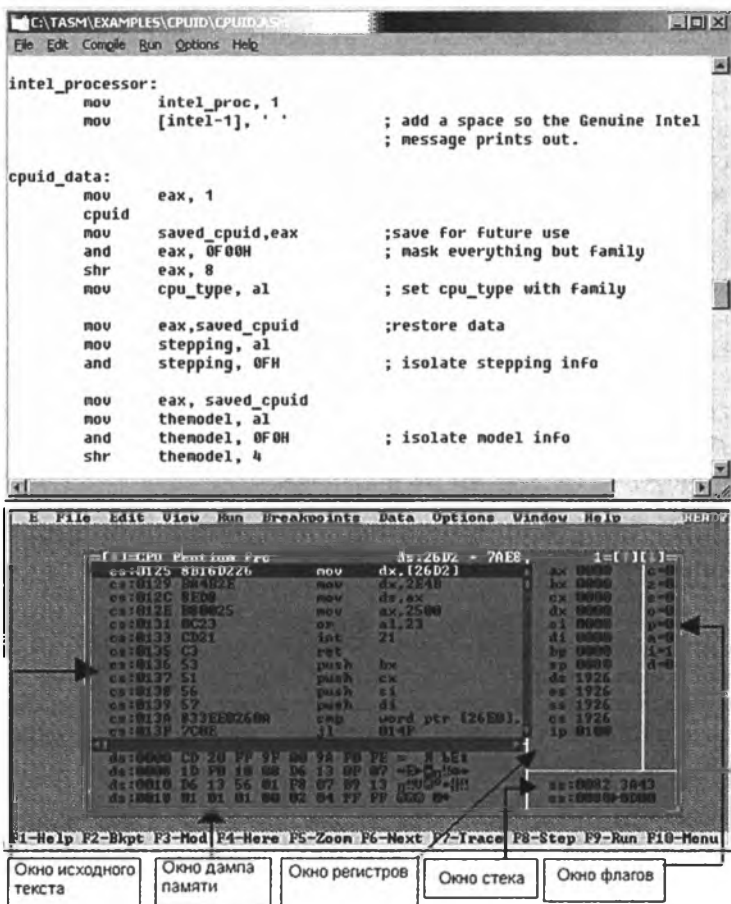


Рис. 1.5. Среда разработки языка Ассемблер

В том же 1949 г. сотрудник Пенсильванского университета США Джон Моучли, участвуя в проекте создания первого компьютера общего назначения ЭНИАК и на начальном этапе проекта создания более совершенного компьютера EDVAC, разработал особую систему кодирования машинных команд *Short Code*, которые вводились непосредственно в память машины. *Short Code* — это интерпретатор-переводчик с языка программирования на машинный код. Интерпретатор использовал примитивный язык программирования высокого уровня. Программист, применяя этот язык, писал задачу математическими формулами, затем пользовался специальной таблицей, где каждому символу формул соответствовали двухлитерные коды. Завершающим этапом работы было выполнение специальной программы компьютера, которая переводила двухлитерные коды в двоичные машинные коды. Таким образом, уже в то время программисты заставили компьютер переводить информацию с языка программирования на машинный код. В настоящее время система Джона Моучли *Short Code* считается началом развития современных языков программирования, реализующих процедурную парадигму<sup>1</sup>.

В 1951 г. Грейс Хоппер (1906–1992) (рис. 1.6), которая была сотрудницей компании Джона Моучли, создала первый компилятор для компьютера МАРК-II<sup>2</sup>. Компилятором Хоппер осуществлялась функция объединения команд. Компилятор выделял память компьютера, преобразовывал команды высокого уровня (в то время — псевдокоды) в комплекс машинных команд (рис. 1.7).

С середины 1950-х гг. в области программирования начинаются стремительные изменения. Специалисты теряют интерес к программированию в машинных командах и переходят к разработке и использованию языков — посредников между программистом и машиной, отдавая предпочтение императивной парадигме программирования.

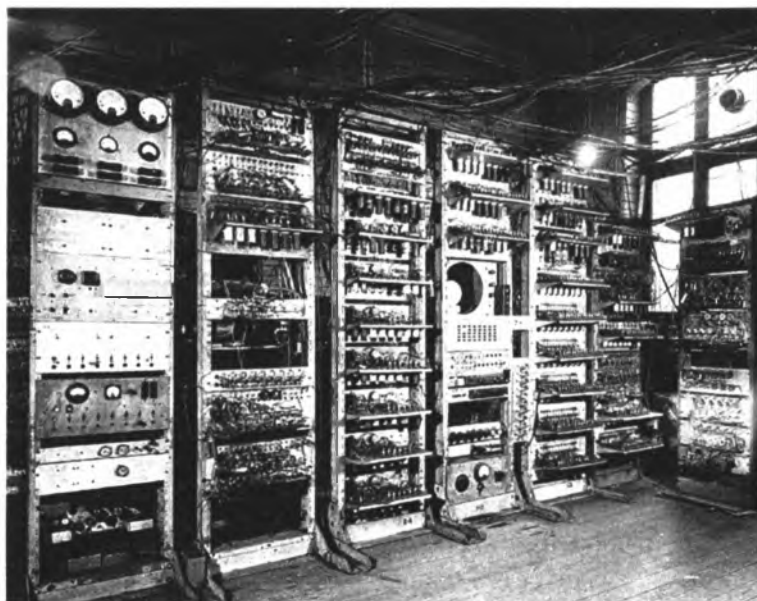
**Справка.** Императивное программирование — это парадигма программирования, которая имеет следующие особенности. Исходный код программы содержит команды (инструкции), которые выполняются последовательно. Данные, необходимые для выполнения команды, могут читаться из памяти, в которую они были занесены при выполнении предыдущей инструкции.

<sup>1</sup> Процедурная (императивная, директивная, модульная) парадигма — одна из вычислительных парадигм, реализуемых в ЯП.

<sup>2</sup> МАРК-II — усовершенствованный компьютер МАРК-I (автоматический вычислитель), который состоял из электромеханических реле и переключателей. Первоначально использовался для военных нужд.



**Рис. 1.6.** Грейс Хоппер



**Рис. 1.7.** Первый американский программируемый компьютер MARK-I (1941)



К наиболее известным ЯП, реализующим императивную парадигму, относят языки FORTRAN, ALGOL, PL/1, BASIC, COBOL. Последний из них разрабатывался при участии Грейс Хоппер (1959).

**Язык FORTRAN.** Первым и впоследствии наиболее распространенным языком стал Фортран (рис. 1.8). Язык Фортран (Fortran — сокращение от FORMula TRANslator, или транслятор формул) был создан в 1954 г. программистами компании *IBM*.

```
DIMENSION K(20)
READ (1,10) K
10 FORMAT (20I5)
DO 20 I=1,19
M=I+1
DO 20 J=M,20
IF (K(I)-K(J)) 20,20,30
30 N=K(I)
K(I)=K(J)
20 K(J)=N
WRITE (2,50) K
50 FORMAT ('МАССИВ K ', 20I5/3X)
STOP
END
```

**Рис. 1.8.** Фрагмент кода на языке FORTRAN.  
Упорядочивание элементов массива

**Язык ALGOL.** В конце 1950-х гг. появляется язык Алгол-60. Алгол-60 (ALGOL-60) — ALGOrythmic Language, «алгоритмический язык», версия 1960 г., был разработан международным коллективом специалистов. Этот язык широко применялся в то время, иногда используется и сейчас. С помощью языка Алгол записываются алгоритмы, строящиеся как последовательность процедур. Алгол был неоднозначно воспринят специалистами, но все же его влияние на теорию и практику современного программирования было значительным. На основе Алгола было создано целое семейство новых языков.

Algol-68 — один из языков программирования, являющийся потомком Алгола-60. Для этого языка не успели сделать полноценный компилятор. Развитие его приостановилось, поскольку начали появляться первые персональные компьютеры (ПК), для которых он был слишком сложным.

**Язык PL/1 — Programming Language One** (рис. 1.9). Язык PL/1 был создан в 1963 г. фирмой *IBM*. Широкое распространение этого универсального языка программирования не удалось. Причиной тому можно считать следующие недостатки: его сильная привязанность к системе *IBM 360/370* и плохая семантика — чересчур многое разрешается делать по умолчанию. Этим языком пользовались профессионалы. В современном программировании он почти не употребляется.

```
BINOM: PROC OPTIONS (MAIN);  
DO N=0 TO 10;  
KB, KW=1; NW=N;  
DO K=0 TO N/2;  
PUT SKIP DATA (K);  
IF K*N $\neq$ 0 THEN  
DO; KB=KB*NW/KW;  
KW=KW+1; NW=NW-1; END;  
PUT DATA (KB);  
END BINOM;
```

**Рис. 1.9.** Фрагмент кода на языке PL/1

**Язык BASIC.** Аббревиатура BASIC расшифровывается как *Beginners All-purpose Symbolic Instruction Code*, что в переводе означает «универсальный символический код для начинающих». BASIC был разработан сотрудниками Дартмутского колледжа Джоном Кемени и Томасом Курцом в 1964 г. (рис. 1.10).



**Рис. 1.10.** Томас Курц и Джон Кемени

Этот язык предназначался для того, чтобы обучать начинающих программистов, и стал широко известным в силу своей простоты. Первые ПК компании *IBM* получили BASIC-интерпретатор.

**Справка.** У первых ПК программы записывались на дискеты, объем которых составлял 360 Кб. В конце 1980-х — начале 1990-х гг. при появлении таких компьютеров, как *Pentagon*, программы можно было считывать с аудиокассет. При этом объем оперативной памяти (ОЗУ) составлял всего 64 Кб, его можно было увеличить до 640 Кб. Интерпретатор Бейсика размещался в зоне ROM (read only memory, или постоянное запоминающее устройство (ОЗУ)), который был доступен MS DOS — операционной системе ПК.

После создания базового языка Basic появилось множество его версий, например Visual Basic, разработанный компанией *Microsoft*, в основе которого лежал объектно-ориентированный подход к программированию. Его основное назначение на первоначальном этапе заключалось в упрощении разработки пользовательских приложений в операционной среде *Windows* (одна из его версий — язык VBA, Visual Basic Application, предназначенный для офисного программирования, в основном для MS Excel и MS Access) (рис. 1.11).

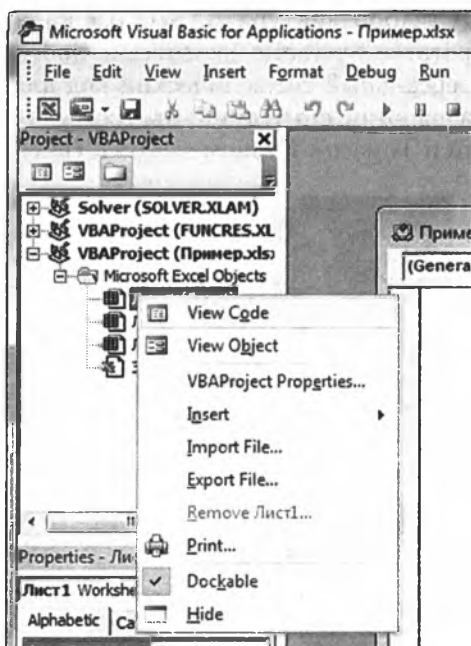


Рис. 1.11. Среда разработки языка VBA

В бывшем СССР также разрабатывались языки программирования, реализующие императивный подход к программированию. Примером такого языка является АП (автоматическое программирование), который был разработан в 1970-е гг. для использования на мини-ЭВМ «Наири» (рис. 1.12) [6].

```
1 допустим a=1 b=-3 c=2
2 вычислим d= $\sqrt{b^2-4ac}$ 
3 вычислим  $x=(-b-d)/(2a)$ 
4 вычислим  $y=(-b+d)/(2a)$ 
5 печатаем с 3 знаками x y
6 кончаем
исполним 1
```

Рис. 1.12. Фрагмент кода на языке АП

Начиная с 1950-х гг. языки программирования, реализующие императивную парадигму, не могли успешно справляться с существующими на тот момент задачами, связанными, например, с необходимостью проведения быстрых математических вычислений, что привело к развитию другой парадигмы — декларативной.

**Справка.** Декларативное программирование — это парадигма программирования, представляющая собой спецификацию задачи, т.е. описание проблемы и ожидаемый результат. Подразделяется на функциональное и логическое программирование.

Наиболее известными языками программирования, реализующими декларативное программирование, являются LISP, LOGO, PROLOG, Scheme, ML, Haskell.

**Язык LISP.** LISP — LISt Processing (англ.), буквально переводится как «обработка списков». Язык LISP, разработанный Джоном Маккарти<sup>1</sup>, появился в конце 1950-х гг. в США и был предназначен для решения задач в области исследований искусственного интеллекта (рис. 1.13).

Данные в LISP представляются в виде системы линейных списков символов, которые образуют иерархии, если элементом списка является также список. В языке были реализованы элементы функционального исчисления. В LISP и данные, и процедуры — это функции, а функция в качестве аргументов принимает другие функции, образуя модульную структуру программы.

<sup>1</sup> Джон Маккарти — американский ученый, внесший огромный вклад в область исследований искусственного интеллекта, основоположник функционального программирования.

```

1  lisp
2  >; Guardar valores como una lista de caracteres
3  >(define (SumarSiguiente V)
4      (cond ((null V) (progn (print "Suma=") 0))
5            (T (+ ( SumarSiguiente (cdr V)) (car V) ) ))
6  SUMARSIGUIENTE
7  >; Craar vector de valores de entrada
8  (defun ObtenerEntrada(f c)
9      (cond ((eq c 0) nil)
10           (T (cons (read f) (ObtenerEntrada f (- c 1))))))
11 OBTENERENTRADA
12 >(defun Hazlo()
13     (progn
14       (setq archivoent (open "lisp.data"))
15       (setq arreglo (ObtenerEntrada archivoent (read archivoent)))
16       (print arreglo)
17       (print (SumarSiguiente arreglo))))
18 HAZLO
19 >Hazlo
20
21 (1 2 3 4)
22 "Suma="
23 10
24 10

```

Рис. 1.13. Фрагмент кода на языке LISP

**Язык LOGO.** Язык ЛОГО (от греч. *logos* — слово) (функциональное программирование) был создан в 1967 г. Сеймуром Пейпертом, преподавателем математики и педагогики (Массачусетский технологический институт). Язык ЛОГО предназначался для того, чтобы обучать школьников основам программирования.

Дети учатся программированию на языке ЛОГО, задавая простые команды и составляя программы из них. Эти программы взаимодействуют с условным объектом — «черепашкой», который передвигается по экрану, оставляя след. Ребенок заинтересован в обучении, поскольку язык ЛОГО позволяет школьнику сделать простой мультфильм с фоном и движущимся персонажем и дает простор для творчества (рис. 1.14).

**Язык Prolog.** Язык Prolog (сокращение от PROgramming in LOGic) был разработан в 1970-х гг. с целью создания систем искусственного интеллекта. В его основу заложено логическое программирование.

**Справка.** Логическое программирование — парадигма, основанная на получении логических выводов из имеющихся фактов. Опирается на теорию и аппарат математической логики.

**Язык Scheme.** Язык Scheme является функциональным языком программирования, был разработан в 1975 г. Гаем Стилом и Джеральдом Сассменом. Он стал развитием языка LISP.

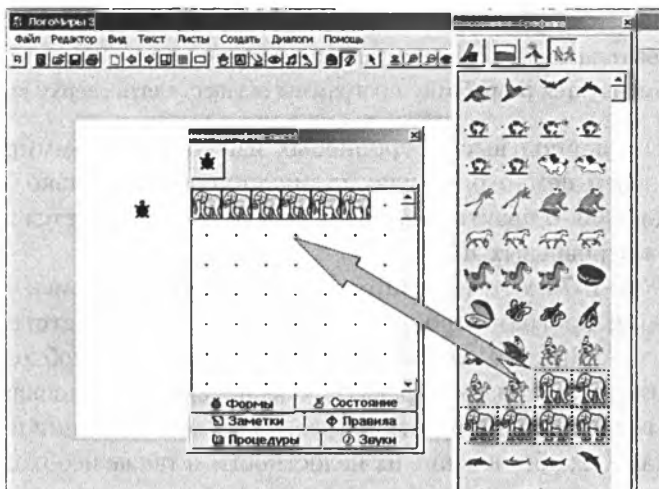


Рис. 1.14. Среда разработки языка LOGO

**Семейство языков ML.** ML (MetaLanguage) – семейство языков программирования, реализующих функциональную парадигму, было представлено в 1973 г. разработчиками из Эдинбургского университета во главе с Робинем Милнером.

**Язык Haskell.** Язык Haskell (1990) является строго типизированным функциональным языком, относящимся к семейству ML. Основная структура – функция.

В конце 1960-х – начале 1970-х гг. в связи с расширением области применения ЭВМ, увеличением мощности вычислительной техники и появлением высокоуровневых (близких к естественным) языков программирования наметился «кризис программного обеспечения». Он обсуждался на знаменательной международной конференции в 1968 г., где Чарльз Хоаэр и Никлаус Вирт высказали мысль о необходимости придания ЯП свойств, позволяющих обеспечивать создание сложного и надежного ПО. До этого периода большинство программ были «штучными» продуктами и в программном коде разобраться было очень сложно, поскольку применение операторов языка, порядок их следования никак не регламентировались. Такой подход позже был назван хаотическим программированием. В том же 1968 г. вышла статья Эдсгера Дейкстры «Доводы против оператора Go To», которая положила начало новой парадигмы – структурной.

**Справка.** Структурная парадигма основана на следующих принципах. Каждый программный блок (модуль, процедура) должен иметь один

вход и выход. Рекомендуется применять четыре вида конструкций: последовательность, ветвление, цикл, выбор из нескольких альтернатив. Рекомендуется разработку программы осуществлять сверху вниз.

В этот период высокоуровневых языков программирования было совсем немного, однако их число через несколько лет возросло до 3000. В практической деятельности используется не более 20 высокоуровневых ЯП.

В 1960–1970-х гг. начали появляться объектно-ориентированные языки программирования, реализующие соответствующую парадигму<sup>1</sup>. Предпосылками их создания явились: необходимость разрабатывать сложные проекты; обеспечение взаимосвязи множества встроженных алгоритмов; требование минимизации избыточности данных; обеспечение их целостности, а также необходимость повышения управляемости процесса разработки программ.

Одним из первых объектно-ориентированных языков программирования стал Simula67, разработанный в 1967 г. сотрудниками Норвежского вычислительного центра. Simula67 – первый язык, поддерживающий семантику современных языков ООП.

**Язык SmallTalk.** Язык SmallTalk разработан в 1969 г. в компании *Xerox Park* группой ученых. Идея заключалась в описании предметной области как совокупности объектов, а взаимодействие между объектами осуществлялось посредством посылки сообщений. В языке были реализованы основные принципы ООП.

**Язык Pascal.** В начале 1970-х гг. появился получивший впоследствии большую популярность язык программирования Pascal (Паскаль), в котором использовалась идея структурной разработки алгоритмов, перекочевавшая из языка Алгол. Его создателем стал ученый из Швейцарии Никлаус Вирт (род. 1934) (рис. 1.15).

Паскаль предполагалось использовать в качестве учебного языка программирования, что практикуется и сейчас в некоторых вузах и школах. Он оказался настолько понятным и удобным, что его взяли на вооружение и профессиональные программисты (рис. 1.16).

---

<sup>1</sup> Объектно-ориентированное программирование (ООП) – парадигма программирования, в основе которой лежит понятие объекта, имеющего определенные свойства, над которым можно осуществлять определенные действия (применять соответствующие методы), изменяющие его состояние.



Рис. 1.15. Никлаус Вирт

```
Program Massivel;  
  Uses WinCrt;  
  Var A:array[1..10] of integer;  
      i,x: integer;  
Begin  
  ClrScr;  
  Randomize;  
  For i:=1 to 10 do  
    begin  
      A[i]:=Random(99);  
      Write(A[i],',');  
    end;  
  WriteLn;  
  Write('Конец программы');  
End.
```

Рис. 1.16. Среда разработки языка Pascal

**Turbo Pascal.** Развитием языка Паскаль стал Turbo Pascal, разработчиком которого выступила фирма *Borland International* еще в 1983 г. Последняя его версия 7.0 вышла в 1992 г. Незначительный объем компилятора (всего 30 Кб) привлек многих программистов, стесненных в объеме пространства памяти на своих компьютерах. В версии 5.5, вышедшей в 1989 г., были реализованы средства для объектного программирования (рис. 1.17).



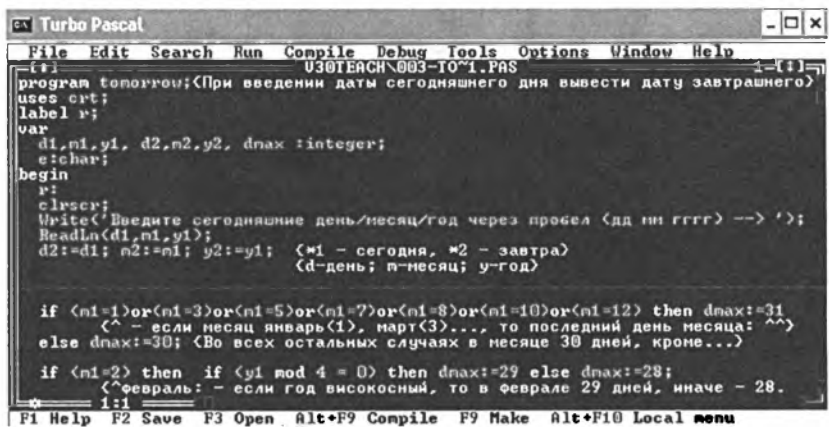


Рис. 1.17. Среда разработки языка Turbo Pascal

В тот период существовала проблема хранения все возрастающего объема данных. Turbo Pascal при наличии всего лишь одного компилятора trc.exe и стандартной библиотеки модулей turbo.tpr общим объемом 120 Кб позволил решать довольно серьезные задачи. Также можно было подключить библиотеку graph.tpu и драйвер egavga.bgi, чтобы создавать графические представления данных. При этом программист мог получить доступ ко всем ресурсам машины, работая в режиме эмуляции MS-DOS (рис. 1.18).

```

uses DOS,CRT;
var r: Registers;
BEGIN
Randomize;
r.ax:=$13;
Intr($10,r);
while not KeyPressed do
begin
Delay(1);
mem[$A000:320*Random(199)+Random(319)]:=Random(255)
end;
TextMode(CO80)
END.
  
```

Рис. 1.18. Фрагмент кода на языке Turbo Pascal

**Язык Delphi.** В 1995 г. появилась среда разработки Borland Delphi, в основе которой лежал тот же Turbo Pascal, расширенный до Object Pascal, с возможностью работы в ОС *Windows* и разработки соответствующих приложений. Используемые case-средства (инструменты разработки) позволяли генерировать часть кода, что ускоряло работу программиста (рис. 1.19).



**Рис. 1.19.** Среда разработки языка Delphi

Однако в последнее время использование этого языка несколько потеряло свою актуальность.

**Язык ADA.** Нельзя обойти стороной и язык ADA (мультипарадигменный ЯП), который был назван в честь Августы Ады Лавлейс и разработан группой ученых из фирмы *Honeywell Bull* и ее одноименного исследовательского центра. По заказу Министерства обороны США в 1975 г. был организован конкурс для создания универсального языка программирования, который обеспечил бы высокую надежность и многолетний срок службы разработанных программ. Язык предусматривал возможность параллельного программирования для компьютеров, имеющих несколько процессоров (рис. 1.20).

```
with Ada.Text_IO;  
  
procedure Hello is  
  use Ada.Text_IO;  
begin  
  Put_Line("Hello, world!");  
end Hello;
```

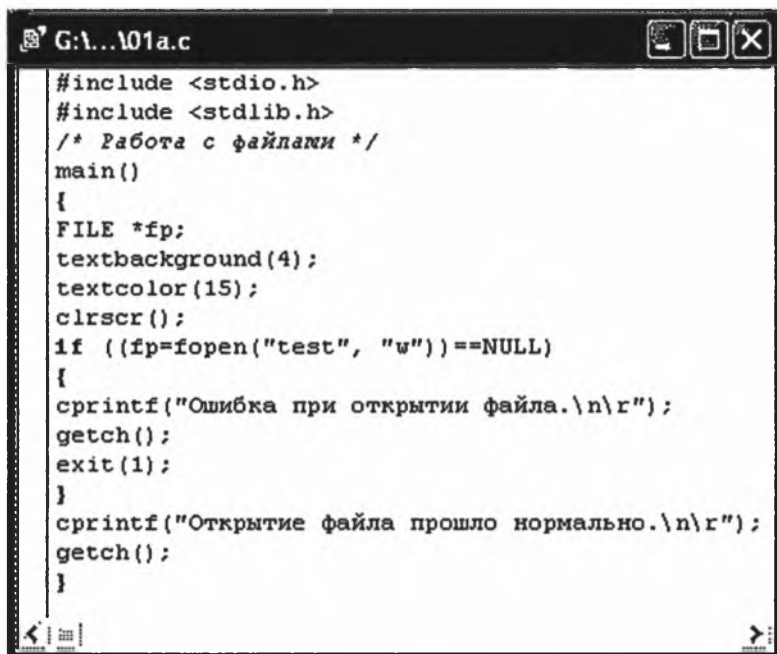
**Рис. 1.20.** Фрагмент кода на языке ADA

**Язык Си (C).** Язык Си (класс языка – процедурный) появился в начале 1970-х гг., автором его стал Деннис Ритчи (1941–2011), *Bell Laboratories* (рис. 1.21).



**Рис. 1.21.** Деннис Ритчи

Язык Си был разработан для реализации операционной системы *Unix*. Его преимуществами являлись компактность и объектный подход, однако сложность синтаксиса вызывала легкое «непонимание», граничащее с «неприятием» со стороны программистов, начинавших с Pascal и работающих под ОС *Windows*. Программирование на Си требует хорошей подготовки, при этом довольно сложно разбираться с текстом чужих программ (рис. 1.22).

A screenshot of a Windows Notepad window titled "G:\...\01a.c". The window contains the following C code:

```
#include <stdio.h>
#include <stdlib.h>
/* Работа с файлами */
main()
{
FILE *fp;
textbackground(4);
textcolor(15);
clrscr();
if ((fp=fopen("test", "w"))==NULL)
{
printf("Ошибка при открытии файла.\n\r");
getch();
exit(1);
}
printf("Открытие файла прошло нормально.\n\r");
getch();
}
```

Рис. 1.22. Фрагмент кода на языке Си

**Язык С++.** Развитием языка С стал С++, взявший за основу средства объектно-ориентированного программирования языка Simula67. Его создателем в начале 1980-х гг. стал Бьярн Страуструп (род. 1950) (рис. 1.23).



Рис. 1.23. Бьярн Страуструп

В качестве базового приоритета ученый определил необходимость обеспечения эффективности программирования, включающую различные представления, в том числе графику, пользовательские интерфейсы, более понятную семантику. По сравнению с С программный код на С++ стал «читаться» гораздо лучше (рис. 1.24).



Рис. 1.24. Среда разработки и фрагмент кода на языке С++

**Язык С#.** Язык С# (читается как «Си шарп») также стал «потомком» языка С, предназначенным для разработки платформ .NET (конкретно MS.NET Framework). Над его созданием в 1998–2001 гг. трудилась группа ученых, включая Андерса Хейлсберга, Скотта Уилтамута и Питера Гоудла (рис. 1.25).

По сути, платформа .NET является средой, позволяющей управлять выполнением программ корпорации *Microsoft* и обеспечивать безопасность кода и совместимость различных языков программирования (рис. 1.26). Главной идеей стало использование промежуточного машинно-независимого языка CIL (Common Intermediate Language (IL)).

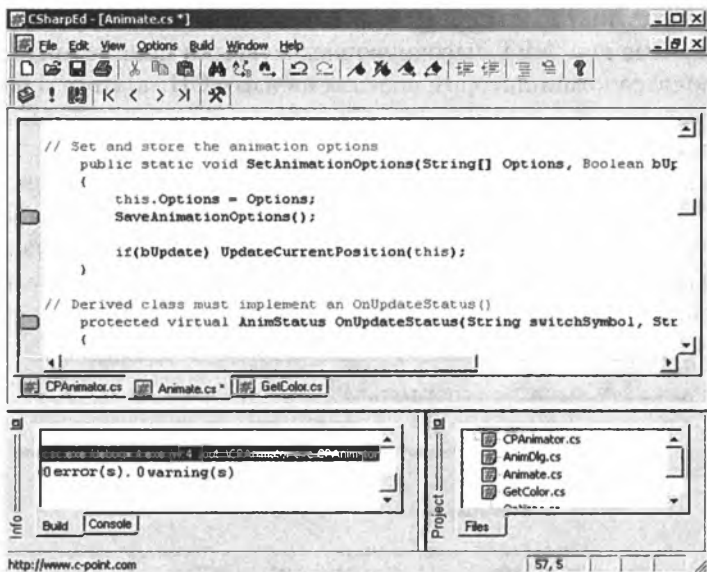


Рис. 1.25. Среда разработки и фрагмент кода на языке C#

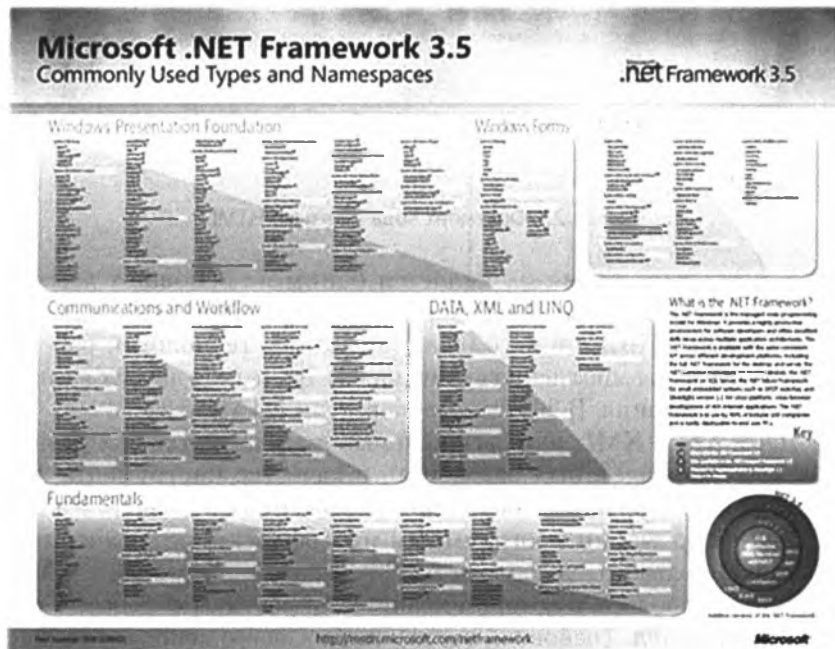


Рис. 1.26. Платформа .NET

Технология работы с платформой заключается в том, что адаптированные под .NET высокоуровневые языки программирования переводятся компилятором сначала на язык CIL, а затем в режиме JIT (Just-In-Time, «по мере надобности») сама среда компилирует машинный код.

**Язык HTML.** Язык HTML (Hyper Text Markup Language – язык разметки гипертекста) в 1986–1991 гг. разрабатывался Тимом Бернерсом в ЦЕРН в Женеве (Швейцария) и был предназначен для обмена информацией различного типа между людьми, не знакомыми с технологиями верстки документов (рис. 1.27).

```
1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
2  "http://www.w3.org/TR/html4/strict.dtd">
3  <html>
4  <head>
5  <meta http-equiv="Content-Type" content="text/html; charset=windows-
6  1251">
7
8  <title>Брошюрная панель навигации в HTML5</title>
9
10 <link rel="stylesheet" href="css/layout.css" type="text/css" media="screen">
11 <link rel="stylesheet" href="css/menue.css" type="text/css" media="screen">
12 <!--[if lt IE 9]>
13 <script src="http://html5shav.googlecode.com/svn/trunk/html5.js"></script>
14 <![endif]-->
15 <script type="text/javascript" src="js/jquery-1.5.2.min.js"></script>
16 <script type="text/javascript" src="js/script.js"></script>
17 </head>
18 <body>
19 <br></br>
20 <p align="center">&copy; Все права защищены</p>
21
22 </body>
23 </html>
```

Рис. 1.27. Фрагмент кода на языке HTML

В настоящее время он является одним из основных языков программирования, которые формируют интернет-представления информации. Позже в него были добавлены технологии работы с мультимедиа, выполнения скриптов и встраивания других языков программирования. В 2017 г. вышла версия HTML 5.2.

**Язык XML.** XML, или расширяемый язык разметки (eXtensible Markup Language), был разработан в качестве инструмента для универсального (понятного всем приложениям) хранения и передачи информации программным интерфейсам, объединенным в единую пользовательскую сеть. Документ, написанный в XML, состоит из пролога, в котором хранятся директивы обработки информации, и тела (основной части), содержащего непосредственно саму информацию. Интерпретатор XML, пользуясь директивами,

обрабатывает информацию, находящуюся в теле (основной части) документа, и передает ее получателю в заданном виде (рис. 1.28).

```
<?xml version="1.0" encoding="windows-1251"?>
<TEST>
<NODE>
<ENG>HELLO World!</ENG>
<RU>Привет мир!</RU>
</NODE>
</TEST>
```

Рис. 1.28. Фрагмент кода на языке XML

**Язык Java.** Язык Java, официальной датой создания которого считают 23 мая 1995 г., является объектно-ориентированным языком программирования, разработанным в компании *Sun Microsystems* и приобретенным впоследствии известной компанией *Oracle*. Отличительной особенностью Java является возможность работать вне зависимости от архитектуры компьютера на любой виртуальной Java-машине, переводя высокоуровневое представление в байт-код. Язык Java является неотъемлемой частью масштабных ERP (от англ. enterprise resource planning — планирование ресурсов предприятия) систем, например, ERP от компании *Oracle* (рис. 1.29).

```
public static void main(String [] args) {
    // TODO code application logic here
    try
    {

        int l1 = 11;
        int l2 = 11;
        int [] n1 = new int [l1];
        int [] n2 = new int [l2];
        Random r = new Random();
        for(int i = 0; i<n1.length; i++) {
            n1 [i] = r.nextInt(10);
        }
    }
}
```

Рис. 1.29. Фрагмент кода на языке Java

**Язык PHP.** Язык PHP (Hypertext Preprocessor, что в переводе означает «гипертекстовый препроцессор») предназначен для



разработки веб-приложений. Он является одним из скриптовых (пошаговых) языков программирования с открытым кодом, обеспечивающих клиент-серверное взаимодействие, и поддерживается практически всеми хостинг-провайдерами. Также он используется для создания динамических веб-страниц, осуществляя обработку действий пользователя на стороне сервера в сочетании с другим языком Javascript, обрабатывающим на стороне клиента через веб-браузер (рис. 1.30).

```
1 <?php
2 // Хост (обычно localhost)
3 $db_host = "192.168.0.1";
4 $db_name = "crm";
5 $db_user = "z1";
6 $db_pass = "1";
7 $db = mysql_connect("192.168.0.1", "z1", "1") or die
8 mysql_select_db("crm", $db);
9 mysql_query("SET NAMES cp1251");
10 mysql_query("SET CHARACTER SET cp1251");
11 mysql_query("SET character_set_client = cp1251");
12 mysql_query("SET character_set_connection = cp1251");
13 mysql_query("SET character_set_results = cp1251");
14 if ( isset( $_GET['id_im'] ) ) {
```

Рис. 1.30. Фрагмент кода на языке PHP

**Язык SQL.** Язык SQL (Structured query language), или язык структурированных запросов, является непроцедурным языком программирования, используемым в системах управления реляционными базами данных, такими как MS SQLServer и MySQL. Он имеет всего пять основных команд, которые позволяют манипулировать хранящимися структурированными данными. Также его поддерживают постреляционные и многомерные СУБД (системы управления базами данных) в качестве инструмента обработки и извлечения данных. Его расширением является T-SQL (Transact SQL), осуществляющий обработку транзактов (рис. 1.31).

```
USE OOO_Ромашка;
SELECT Id_number, Surname, Name
FROM Staff
WHERE (Id_number = 103 AND Surname = 'Медведев')
OR (Name = 'Дмитрий' AND LastName = 'Романович');
```

Рис. 1.31. Фрагмент кода на языке SQL

Имеются и другие языки программирования, которые решают те или иные задачи (например, CSS (каскадные таблицы стилей),

JavaScript, Ajax, jQuery и т.п.). Как можно заметить, история развития языков программирования демонстрирует эволюцию алфавита, синтаксиса и семантики, приближая их к естественным языкам. Скорее всего, в недалеком будущем кодирование можно будет осуществлять посредством простых голосовых сообщений.

### 1.3. КЛАССИФИКАЦИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Классификация языков программирования осуществляется по ряду критериев в зависимости от класса решаемых задач или парадигм (принципов), заложенных в их фундамент.

**Классификация по близости ЯП к естественным языкам.** Разделяют низкоуровневые (близкие к машинному коду) и высокоуровневые (близкие к естественному) языки программирования.

Поскольку процессор компьютера работает с машинными кодами, а писать на машинном коде довольно трудно, сама программа пишется на высокоуровневом языке и переводится на машинный уровень через специальную программу, называемую транслятором.

Следующая классификация определяется **принципами написания программного кода**.

**Императивные языки программирования:** программа включает последовательность операторов или команд (инструкций). Каждая инструкция меняет содержимое памяти ЭВМ согласно архитектуре фон Неймана. Наиболее известными представителями императивных языков являются C++, C, C#, Basic, Pascal, Fortran, Algol.

**Справка. Принципы фон Неймана<sup>1</sup>. Использование двоичной системы счисления в вычислительных машинах.** Преимущество перед десятичной системой счисления заключается в том, что устройства можно делать достаточно простыми, арифметические и логические операции в двоичной системе счисления также выполняются довольно просто.

**Программное управление ЭВМ.** Работа ЭВМ контролируется программой, состоящей из набора команд. Команды выполняются последовательно друг за другом. Созданием машины с хранимой в памяти программой было положено начало тому, что мы сегодня называем программированием.

**Память компьютера используется не только для хранения данных, но и для программ.** При этом и команды программ, и данные кодируются в двоичной системе счисления, т.е. их способ записи одинаков.

---

<sup>1</sup> Принципы фон Неймана. URL: <https://infl.info/machineneumann> (дата обращения: 25.03.2018).

Поэтому в определенных ситуациях над командами можно выполнять те же действия, что и над данными.

**Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы.** В любой момент можно обратиться к любой ячейке памяти по ее адресу. Этот принцип открыл возможность использовать переменные в программировании.

**Возможность условного перехода в процессе выполнения программы.** Несмотря на то, что команды выполняются последовательно, в программах можно реализовать возможность перехода к любому участку кода.

К следующему классу языков программирования относят *аппликативные, или функциональные языки программирования* (ФЯП).

Блок программы состоит из вложенных друг в друга последовательностей функций:

```
function_1(function_2(... function_N(data)))
```

Однако использование ФЯП на ЭВМ, имеющих архитектуру фон Неймана, считается неэффективным.

Типичными представителями таких ЯП являются *Hascel, Miranda, MetaLanguage (ML), Scheme, LISP.*

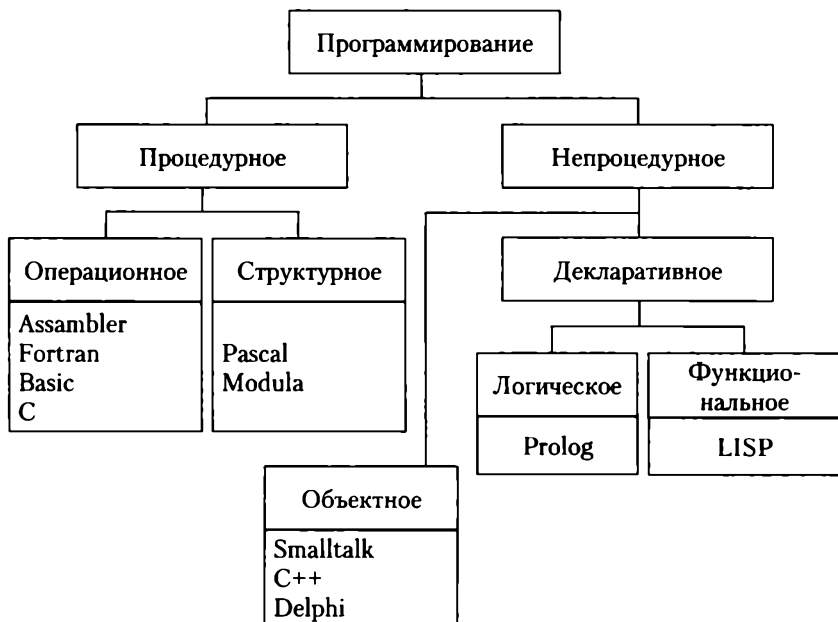
Также широко распространено использование *процедурных языков программирования*, которые представляют собой блоки или подпрограммы решения задачи или ее части.

Типичным представителем *логических или декларативных языков* считается язык программирования Prolog (Programming in Logic), предложенный Аланом Кольмероз еще в 1973 г. Программа представляет собой набор фактов и правил. При формировании пользовательского запроса к встроенной базе знаний инициируются вычисления, результатом которых является генерирование ответа в виде «истина» или «ложь». Логический вывод осуществляется посредством использования методов математической логики.

Еще к одному отдельному классу относят *объектные или объектно-ориентированные языки программирования*, суть которых заключается в описании объектов в виде классов, имеющих свои свойства, атрибуты, и к которым можно применять собственные методы для изменения этих свойств. Наиболее известными такими языками являются Delphi, Java, Ada и ряд других. Например, Python, которому и посвящено настоящее учебное пособие, совмещает в себе возможность использования объектно-ориентированного подхода и обычного алгоритмического подхода и включает как процедурное, так и функциональное программирование.

Также выделяют специализированные языки, предназначенные для манипулирования данными (базы данных); организации компьютерных сетей, а также моделирующие языки, обычно используемые для программирования бизнес-процессов или при решении задач имитационного моделирования (GPSS).

**Классификация по характеру выполнения кода программы.** В зависимости от подхода к программированию выделяют процедурное и непроцедурное программирование (рис. 1.32).



**Рис. 1.32.** Классификация ЯП по подходу к программированию

В зависимости от задач, поставленных программистом, выбор языков программирования, которых, по оценкам специалистов, более 3000, должен осуществляться по следующим критериям [2]:

- относительная простота – предполагает удобство использования и легкость изучения;
- гибкость – подразумевает реализацию алгоритмов с помощью простых средств;
- надежность – определяется наличием в трансляторе обработчика ошибок, что позволяет выявить недостатки программы во время ее отладки;

- мобильность — включает в себя возможность переноса программы с одного компьютера или компьютеров на другие вычислительные средства;
- естественность — подразумевает возможность описания предметной области на близком к ней языке.

Немаловажной при выборе языка программирования является и его стоимость. Хотя в настоящее время существует множество бесплатных языковых средств, и этот критерий понемногу отходит на второй план.

Можно предложить и другие критерии выбора языка программирования, которые разработчик может выбрать самостоятельно, руководствуясь своими приоритетами.

#### **1.4. СИНТАКСИС И СЕМАНТИКА ЯЗЫКА. ОБЩИЕ КОНСТРУКЦИИ**

Напомним следующие определения, которые были даны ранее [2].

Алфавит — конечный набор символов для конкретного языка программирования.

Синтаксис определяет правила образования токенов и правила формирования последовательностей токенов.

Семантика — формальное содержание (смысл) последовательности токенов.

Взаимодействие синтаксических и семантических правил определяют те или иные понятия языка, например операторы, идентификаторы, переменные, функции и процедуры, модули и т.д. В отличие от естественных языков, правила грамматики и семантики для языков программирования, как и для всех формальных языков, должны быть явно, однозначно и четко сформулированы.

Для строгого и точного описания синтаксиса языка программирования, как правило, используют специальные метаязыки (языки для описания других языков).

Наиболее распространенными метаязыками являются металингвистические формулы Бэкуса — Наура (язык БНФ) и синтаксические диаграммы Вирта.

*Язык БНФ* (язык нормальных форм) — способ записи конструкций языка программирования с помощью формул, похожих на математические.

Для каждого понятия языка существует единственная метаформула (нормальная форма).

*Состав метаформулы:* левая и правая части.

*Левая часть:* определяемое понятие.

*Правая часть:* множество допустимых конструкций языка, которые объединяются в это понятие.

Используются специальные метасимволы в виде  $\langle \rangle$ , которые обозначают определяемое понятие (в левой части формулы) или ранее определенное понятие (в ее правой части).

Левая и правая части формулы разделяются метасимволом «::=», имеющим смысл «по определению есть». Знак «|» читается как «или» (примеры взяты из [2]).

### **Пример**

$\langle \text{переменная} \rangle ::= A | B$

$\langle \text{выражение} \rangle ::= \langle \text{переменная} \rangle | \langle \text{переменная} \rangle + \langle \text{переменная} \rangle | \langle \text{переменная} \rangle - \langle \text{переменная} \rangle$

### **Означает:**

$\langle \text{переменная} \rangle$  это одна из букв, А или В, а  $\langle \text{выражение} \rangle$  – любая из следующих десяти записей: А; В; А+А; А+В; В+А; В+В; А-А; А-В; В-А; В-В.

## **Рекурсивная форма**

### **Пример**

Пусть необходимо ввести понятие  $\langle \text{двоичный код} \rangle$ , под которым понимается любая непустая последовательность цифр 0 и 1. Тогда простое и компактное рекурсивное определение с помощью метаформул выглядит так:

$\langle \text{двоичная цифра} \rangle ::= 0 | 1$

$\langle \text{двоичный код} \rangle ::= \langle \text{двоичная цифра} \rangle | \langle \text{двоичная цифра} \rangle \langle \text{двоичный код} \rangle$

### **Означает:**

конструкция 001101

является двоичным кодом, поскольку, последовательно применяя рекурсию, мы имеем следующие 5 шагов рекурсии:

1. 0 – двоичная цифра, а 01101 – двоичный код, поскольку
2. 1 – двоичная цифра, а 1101 – двоичный код, поскольку
3. 1 – двоичная цифра, а 101 – двоичный код, поскольку
4. 1 – двоичная цифра, а 01 – двоичный код, поскольку
5. 0 – двоичная цифра, а 1 – также двоичная цифра, рекурсия завершена.

Синтаксические диаграммы позволяют графически отобразить значения метапеременных метаязыка. Диаграмма состоит из основных символов или понятий языка (рис. 1.33).

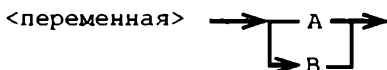


Рис. 1.33. Пример синтаксической диаграммы

Диаграмма на рис. 1.33 эквивалентна метаформуле:

`<переменная> ::= A|B.`

Другие примеры синтаксических диаграмм представлены на рис. 1.34.

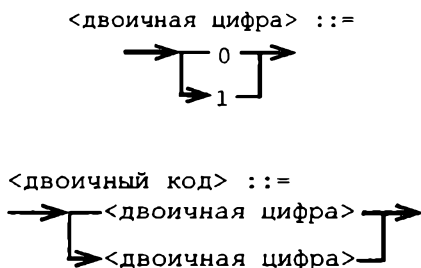


Рис. 1.34. Примеры синтаксических диаграмм

В большинстве своем алфавиты языков программирования типичны, в основе их лежат буквы латинского алфавита, арабские цифры и специальные символы.

Примером сочетаний элементов алфавита, цифр и символов, которые составляют синтаксис и семантику языка, могут быть:

`<буква> ::= a|b|c|d|e|f|g|h|i|j|k|l|m и т.п.`

`<цифра> ::= 0|1|2|3|4|5|6|7|8|9`

`<знак арифметической операции> ::= +|-|*|/`

`<разделитель> ::= . | , | ; | : | ( | ) | [ | ] | { | } | ' | : =`

`<служебное слово> ::= begin | end | if | then | else | for | next и т.п.`

`<спецсимвол> ::= <знак арифметической операции> | <разделитель> |`

`<служебное слово>`

`<основной символ> ::= <буква> | <цифра> | <спецсимвол>`

`<комментарий> ::= {любая последовательность символов}`

Для высокоуровневых языков представленные конструкции схожи. Например, операция присваивания в Delphi пишется как «:=», для VBA и Python — «=».

## 1.5. СПОСОБЫ РЕАЛИЗАЦИИ ЯЗЫКОВ: КОМПИЛЯЦИЯ, ИНТЕРПРЕТАЦИЯ, СМЕШАННЫЙ ПОДХОД

Языки программирования могут быть реализованы как компилируемые и интерпретируемые.

*Компилятор* — это программа, которая предназначена для перевода (трансляции) высокоуровневого языка программирования в машинный код или на язык, близкий к машинному (например, Ассемблер). Входной информацией для компилятора является текст программы<sup>1</sup>. На выходе получаем машинно-ориентированный язык.

*Компиляция* — трансляция программы на язык, близкий к машинному, либо трансляция программы, составленной на исходном языке, в объектный модуль. Иными словами, формируется программа, обычно с расширением .exe, которая обрабатывается непосредственно ЭВМ.

*Интерпретатор* — языковый процессор, который построчно анализирует исходную программу и одновременно выполняет предписанные действия, а не формирует на машинном языке скомпилированную программу, которая выполняется впоследствии.

Некоторые языки, например Java и C#, являются смешанными, т.е. реализуются как компиляторы и интерпретаторы. Другими словами, программа компилируется не в машинный язык, а в машинно-независимый код низкого уровня, байт-код. Далее байт-код выполняется виртуальной машиной. Для выполнения байт-кода обычно используется интерпретация, хотя отдельные его части для ускорения работы программы могут быть транслированы в машинный код непосредственно во время выполнения программы по технологии компиляции «на лету» (Just-in-time compilation, JIT). Для Java байт-код исполняется виртуальной машиной Java (Java Virtual Machine, JVM), для C# — Common Language Runtime.

---

<sup>1</sup> Текст программы — это текст, написанный на одном из языков программирования. Его также называют исходным кодом (программным кодом или просто кодом). Процесс написания текста программы называют кодированием.



## 1.6. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ В ПРОГРАММИРОВАНИИ

Компьютерная программа (здесь и далее — программа) — это совокупность данных и алгоритмов, представленных в виде текста на одном из языков программирования.

Никлаус Вирт сформулировал следующую формулу для понимания сущности программы:

Программа = алгоритм + данные

**Этапы проектирования программы.**

*Постановка задачи.* Установить цель решения задачи и раскрыть ее содержание.

*Построение математической модели.* Определить, какие математические соотношения (формулы) и методы подходят для решения задачи.

*Разработка алгоритма:*

- 1) выделение автономных этапов вычислительного процесса;
- 2) формальная запись каждого этапа;
- 3) установка порядка выполнения этапов;
- 4) проверка правильности выбранного алгоритма;
- 5) реализация алгоритма. Запись алгоритма на одном из языков программирования. Создание программы;
- 6) отладка программы. Выявление и устранение ошибок. Общий метод устранения ошибок: прослеживание хода выполнения программы, вывод промежуточных результатов и сравнение их с контрольными результатами, полученными вручную;
- 7) анализ и обработка результатов;
- 8) создание документации на программу.

Следование представленным этапам проектирования позволяет программисту избежать ряда ошибок при создании любого проекта, в том числе при написании программы или разработке информационной системы (ИС).

В качестве методов разработки программ используют метод «сверху вниз» или «снизу вверх», последний из которых применяется в случае, если в будущем потребуется расширить функциональные возможности ИС.

**Проектирование «снизу вверх».** Метод основан на создании базовых простейших элементов, на основе которых строятся более сложные структуры.

**Проектирование «сверху вниз».** В этом методе сначала создается структура программы. Каждый элемент представлен моделью «черного ящика». Далее детально прорабатывается каждый элемент.

Также существуют различные формы представления алгоритма программы, под которым понимают набор инструкций или последовательность действий, направленных на достижение определенной цели.

1. Словесная запись. Недостаток: отсутствие строгой формализации и наглядности вычислительного процесса.

2. Табличная форма записи алгоритма.

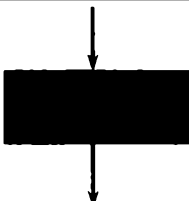
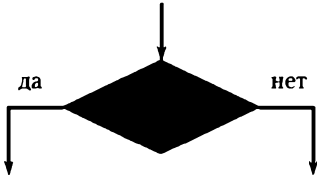
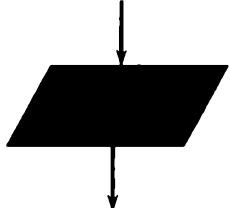

3. Логические схемы алгоритмов.

4. Граф-схемы алгоритмов (блок-схемы).

*Блок-схема* — графическое изображение программы, дополненное элементами словесной записи. Каждый этап программы представляется определенной графической фигурой (табл. 1.1).

Таблица 1.1

**Графические элементы алгоритма**

№ п/п	Обозначение	Описание
1		<i>Инструкция</i> , например присвоение
2		<i>Условие</i> . Выбирается дальнейшее направление выполнения программы в зависимости от выполнения условия
3		<i>Ввод и вывод данных</i> из программы
4		<i>Начало и конец</i> программы

## Пример

Функцию следующего вида

$$y = \begin{cases} 2x^2 + 3, & x \leq 0 \\ 2x^2 - 3, & x > 0 \end{cases}$$

можно представить как блок-схему (рис. 1.35).

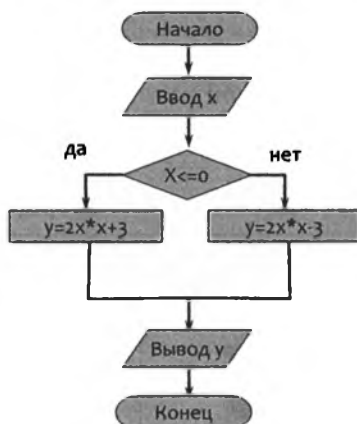


Рис. 1.35. Блок-схема вычисления функции

## Упражнение 1.1

Проведите интерпретацию алгоритма, представленного в виде блок-схемы, показанной на рис. 1.36.

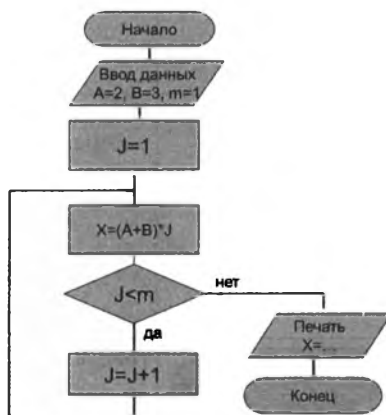


Рис. 1.36. Блок-схема алгоритма

Запись алгоритма в виде программы представлена на рис. 1.37.

```
Program Sqrt_eq;  
Var x, y: real;  
Begin  
Writeln('Введите x')  
Readln(x);  
y:=(x^2-2*x*(sqrt(x)+2));  
writeln('y= ', y:5:2);  
End.
```

**Рис. 1.37.** Программа для реализации алгоритма

Читателю, немного знакомому с основами алгоритмизации и английским языком, станет ясно, что программа вычисляет корни квадратного уравнения. Данная программа написана на языке Pascal. Ознакомившись с основными понятиями в программировании, можно переходить непосредственно к изучению языка Python.

## **1.7. ЯЗЫК ПРОГРАММИРОВАНИЯ PYTHON И ЕГО МЕСТО СРЕДИ ДРУГИХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ**

Язык программирования Python (читается как «пайтон», но обычно в русской версии произносят «питон») — это высокоуровневый язык программирования, который может работать в режиме интерпретатора. Его простота, удобство пользования, относительно простые синтаксис и семантика, возможность использования объектно-ориентированного подхода привлекли множество программистов со всего мира. Он является бесплатным, что также служит весомым преимуществом.

Вот только несколько возможностей Python:

- работа с xml/html файлами;
- работа с http-запросами;
- GUI (графический интерфейс);
- создание веб-сценариев;
- работа с базами данных;
- работа с FTP (англ. *File Transfer Protocol* — протокол передачи файлов);
- работа с изображениями, аудио- и видеофайлами;
- робототехника;
- программирование математических и научных вычислений и т.д.

Python подходит для решения большинства повседневных задач, будь то резервное копирование, чтение электронной почты или какое-нибудь игровое приложение. Язык программирования Python практически ничем не ограничен, поэтому также может использоваться в крупных проектах. Например, Python интенсивно используется такими IT-гигантами, как *Google* и *Yandex*. К тому же простота и универсальность Python делают его одним из лучших языков программирования.

## 1.8. УСТАНОВКА PYTHON

Установка Python осуществляется в следующей последовательности.

1. Для установки Python необходимо его скачать с официального сайта (<http://www.python.org>) и выбрать соответствующую версию, лучше последнюю (рис. 1.38).

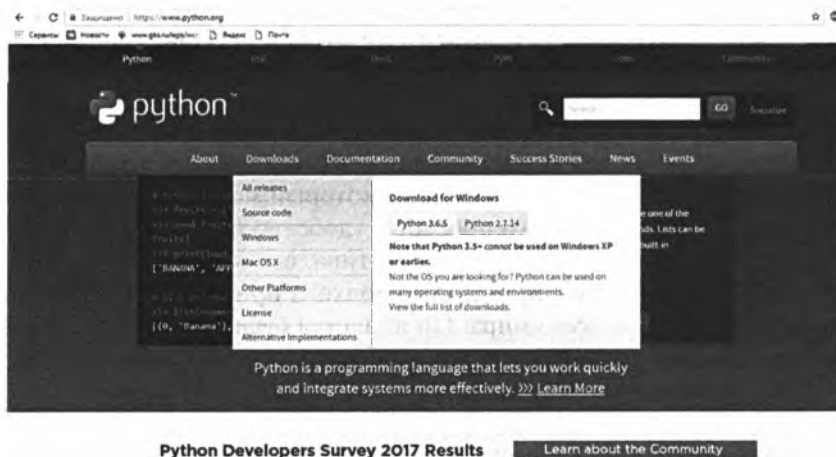


Рис. 1.38. Страница загрузки Python

**Внимание.** Версия выбирается в соответствии с типом операционной системы и ее разрядности, которая установлена на вашем компьютере. Если у вас стоит Windows и ее разрядность – 64 бит, то, соответственно, скачивайте эту версию (рис. 1.39).

## Python Releases for Windows

- Latest Python 3 Release - Python 3.6.5
- Latest Python 2 Release - Python 2.7.14
- Python 2.7.15rc1 - 2018-04-15
  - Download Windows x86 MSI installer
  - Download Windows x86-64 MSI installer
  - Download Windows help file
  - Download Windows debug information files for 64-bit binaries
  - Download Windows debug information files
- Python 3.7.0b3 - 2018-03-29
  - Download Windows x86 web-based installer
  - Download Windows x86 executable installer
  - Download Windows x86 embeddable zip file
  - Download Windows x86-64 web-based installer

Рис. 1.39. Версии Python

2. После загрузки файла .exe дважды кликаем на него и нажимаем «Выполнить» (рис. 1.40).

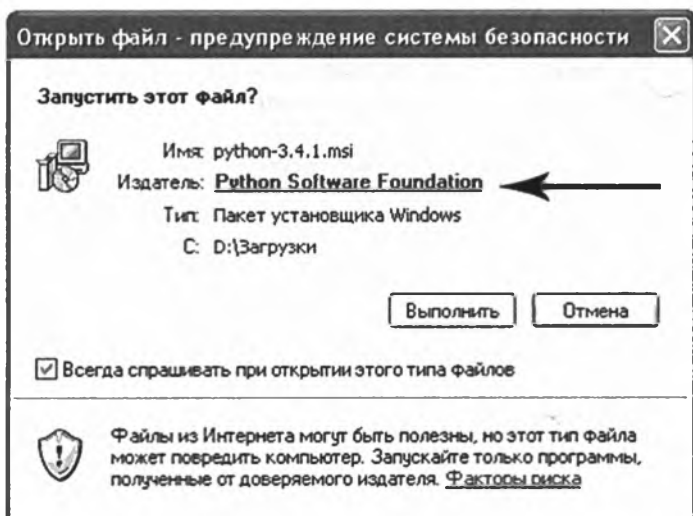


Рис. 1.40. Запуск файла Python

3. Далее необходимо выбрать тип установки (инсталляции) – только для одного пользователя или для всех (1.41).



Рис. 1.41. Выбор типа установки Python

4. Выбираем каталог установки (рис. 1.42).



Рис. 1.42. Выбор папки для установки Python

5. Следующим этапом является выбор модулей и компонент для установки (лучше оставить по умолчанию) (рис. 1.43).

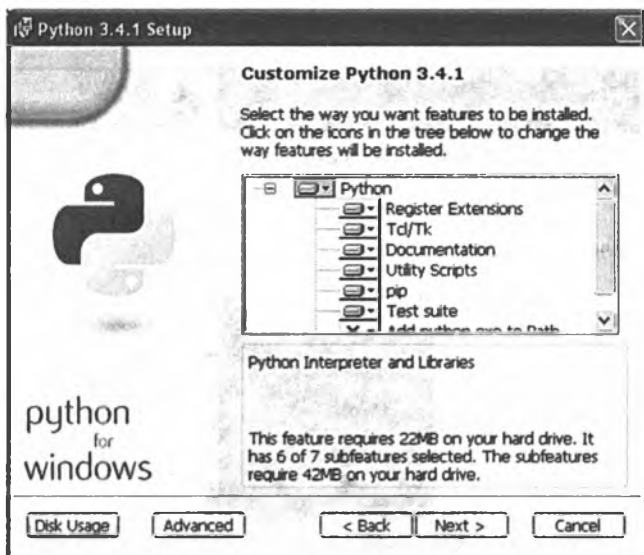


Рис. 1.43. Выбор папки для установки Python

6. Нажимаем Next и ожидаем окончания установки.

**Справка.** В операционной системе (ОС) Windows вместе с установкой Python устанавливается интегрированная среда разработки IDLE (Integrated Development Environment), в которой можно писать программы, в отличие от обычного интерпретатора, где последовательно выполняются введенные инструкции.

### Упражнение 1.2

Самостоятельно установите язык программирования Python на свой компьютер, следуя алгоритму в данном параграфе.

## 1.9. РАБОТА В ИНТЕРАКТИВНОМ РЕЖИМЕ ИНТЕРПРЕТАТОРА

Интерпретатор анализирует и тут же выполняет (собственно интерпретация) программу покомандно (или построчно) по мере поступления ее исходного кода на вход интерпретатора. Достоинством такого подхода является мгновенная реакция. Недостаток: такой интерпретатор обнаруживает ошибки в тексте программы только при попытке выполнения команды (или строки) с ошибкой.



Для запуска интерпретатора достаточно нажать на иконку, которая была установлена на ваш рабочий стол (рис. 1.44).

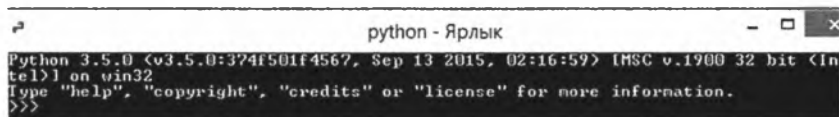


Рис. 1.44. Интерпретатор Python

Для работы в режиме интерпретатора команды вводятся с клавиатуры построчно (рис. 1.45).

```
>>> print("hi")
hi
>>> 2+2
4
>>> 2*4
8
>>> 2**2
4
>>> 2**3
8
>>>
```

Рис. 1.45. Ввод команд с клавиатуры

При работе с числами интерпретатор ведет себя как обычный калькулятор (рис. 1.46).

```
>>> 2**2
4
>>> 2**3
8
>>> print("завтра будет 15 февраля")
завтра будет 15 февраля
>>> print("завтра", 14+2, "пороза")
завтра 16 пороза
>>> 2+3*2
8
>>> 12.5/2
(12, 2.5)
>>> 12.5/2
6.25
>>> 14%3
2
>>> 13*5+4*(5+6)**3
5389
>>> --5
5384
>>> _/33
161.15151515151516
>>> _*365
59050.3030303030303
>>>
```

Рис. 1.46. Ввод команд с клавиатуры

Анализ рис. 1.46 позволяет сделать вывод о назначении символов, осуществляющих операции с числами.

### Упражнение 1.3

Проанализируйте рис. 1.46. Определите символы, осуществляющие операции с числами. Составьте таблицу, где первый столбец представляет собой набор символов, а второй — его назначение.

### Упражнение 1.4

*Ознакомление с Python.* Для диалога с Python запустите интерпретатор Python. Нажмите на соответствующий ярлык на рабочем столе (см. рис. 1.44).

**Справка.** Диалоговое окно позволяет вводить команды построчно.

Напечатайте:

```
print('Доброе утро')
```

В результате должно получиться следующее (рис. 1.47).

```
>>> print <'доброе утро'>
доброе утро
>>> _
```

Рис. 1.47. Результат выполнения команды (инструкции)

Замените апострофы на кавычки и тройные кавычки, в результате ничего не изменится (рис. 1.48).

```
>>> print <'доброе утро'>
доброе утро
>>> print <"Доброе утро">
Доброе утро
>>> print <""""Доброе утро"""">
Доброе утро
>>>
```

Рис. 1.48. Результат выполнения команды (инструкции)

Введите следующие строки (рис. 1.49).

```
print ("Сегодня", 8, "марта")
print ("Завтра "+"будет", 8+1, "марта")
print ("Послезавтра будет", 8+2, "марта")
```

Рис. 1.49. Инструкции Python

Посмотрите, что будет в интерпретаторе. Сделайте вывод о проведенных операциях.

### Упражнение 1.5

Сложите  $n$  столов и  $2n$  стульев ( $n$  – № по журналу или последняя цифра зачетки).

В интерпретаторе должно получиться в одной строке:  $n$  столов +  $2n$  стульев равняется  $X$  предметов мебели.

Основные арифметические операции приведены в табл. 1.2.

Таблица 1.2

Основные операции над числами

$x+y$	Операция сложения
$x-y$	Операция вычитания
$x*y$	Умножение чисел
$x/y$	Деление
$x//y$	Целая часть от деления
$x\%y$	Получение остатка от деления. Остаток от деления
$-x$	Замена знака числа
$x^{**}y$	Операция возведения в степень. При этом степень не обязательно должна быть целым числом

### Упражнение 1.6

Сконструируйте алгоритм, вычисляющий стоимость покупки бананов, мандаринов и яблок при известном весе и цене за килограмм.

Результат нужно представить в виде скриншота, где печатается соответствующая цена.

**Справка.** Чтобы осуществить диалог с пользователем, используется инструкция `input`.

Например, инструкция:

```
x=int(input('Введите целое число '))
```

присвоит переменной  $x$  значение, введенное пользователем с клавиатуры.

Тогда введенное вами выражение  $x+5$ , и если перед этим пользователь ввел 2, то интерпретатор Python выдаст значение 7.

### Упражнение 1.7

Вычислите выражения, сделайте скриншот, результат представьте в виде табл. 1.3.

Таблица отчета

Выражение	Результат выполнения

$$\frac{2^{-9} \cdot 2^7}{2^{-4}}, (\sqrt{17} - \sqrt{12})(\sqrt{17} + \sqrt{12}), \frac{(2\sqrt{6})^2}{25}, \left( \sqrt{2\frac{4}{7}} - \sqrt{7\frac{1}{7}} \right) : \sqrt{\frac{2}{63}};$$

$$h(10 + x) + h(10 - x), \text{ если } h(x) = \sqrt[3]{x} + \sqrt[3]{x - 20} \quad (x = n);$$

$$1 + \frac{2}{3 + \frac{4}{5 + \frac{6}{7 + x}}};$$

$$1 + \sin^2(n) \cdot 3^3 + \arctg(8 \cdot \pi / 3);$$

$$\frac{\ln^2(n^3)}{1 + n^2} \bmod(2n) + 4 \cos^2(n) + \arctg(9 \cdot \pi / 2);$$

$$\frac{2^n(n^3)}{n^3 + n^2} \operatorname{div}(2n) + 4 \cos^2(n) \operatorname{tg}(n + 6), \text{ округлить до целых};$$

$$\sqrt{n^3 + n^2} / \frac{2^n(n^3)}{3^4} \cdot (3n) + 4 \cos^2(n) \operatorname{tg}(n + 6) \bmod(2), \text{ взять остаток.}$$

### 1.10. СРЕДА ПРОГРАММИРОВАНИЯ. ИСПОЛЬЗОВАНИЕ ДОКУМЕНТАЦИИ

Среда программирования IDLE для Python вызывается посредством установленной иконки на рабочем столе. Однако для того чтобы создать свою первую программу, нужно проделать следующее.

Создайте текстовый файл, откройте его в блокноте, сохраните с расширением .py и закройте.

Нажмите на него правой кнопкой мыши (далее – ПКМ) и выберите «Редактировать с IDLE» (рис. 1.50).

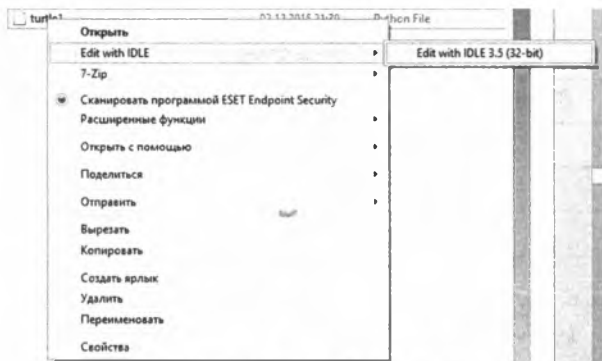


Рис. 1.50. Вызов IDLE Python

В результате файл будет открыт в IDLE, где можно будет писать уже соответствующую программу.

**Справка.** Совсем не обязательно текст программы писать в IDLE. Ее можно писать в любом текстовом редакторе или любой другой среде разработки, например MSVisio. Главное, чтобы у файла было расширение .py

Для запуска программы необходимо в меню выбрать Run и RunModule или просто нажать F5 (рис. 1.51).

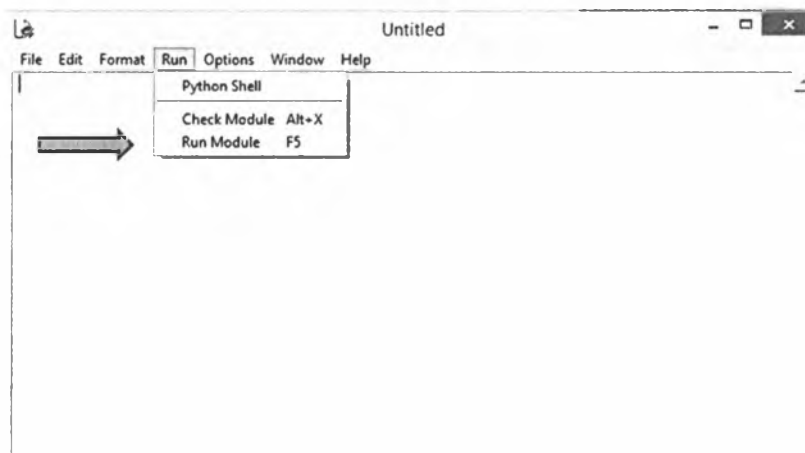


Рис. 1.51. Запуск программы в IDLE Python

В разделе «опции» можно настроить форму отображения инструкций, переменных, констант, шрифтов и т.п.

Документация отрывается через меню Help (рис. 1.52).

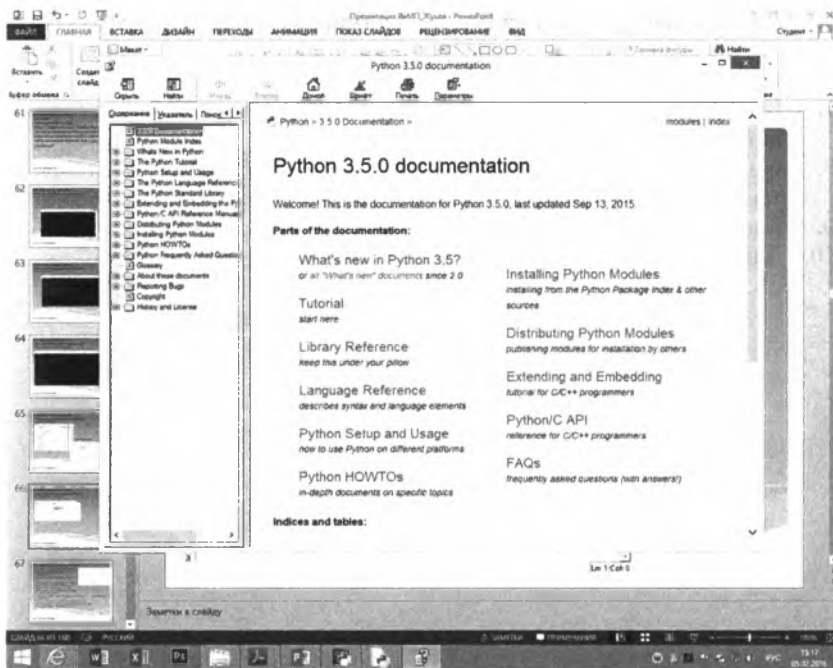


Рис. 1.52. Окно документации Python

### Упражнение 1.8

Создайте файл с расширением `турprogram1.py` через текстовый редактор. Откройте его через среду IDLE.

### Упражнение 1.9

Введите следующий код программы (рис. 1.53).

```
print('Здравствуй, мир!')
print('Как Ваше настроение?')
input('Напишите в этой строке |')
print('Могло бы быть и хуже')
```

Рис. 1.53. Код программы Python

Проявите творчество. Сделайте скриншот.

### Контрольные вопросы и задания

1. Опишите историю развития языков программирования.
2. Объясните понятие языка программирования.

3. Что такое алфавит, синтаксис и семантика языка?
4. Какие есть способы реализации языков программирования?
5. Дайте характеристику языка программирования Python и его особенностей.
6. Как осуществляется установка Python?
7. Какие проблемы (некорректная работа) могут возникнуть после установки Python?
8. Опишите особенности работы в интерактивном режиме интерпретатора.
9. Как запустить среду программирования IDLE?
10. Как вызвать и использовать документацию Python?

## Глава 2

# ТИПЫ ДАННЫХ И ОПЕРАЦИИ ЯЗЫКА PYTHON

### 2.1. РАБОТА С ЧИСЛАМИ

#### 2.1.1. Общие сведения

Как и во всех языках программирования, в Python реализованы возможность работы с числами и осуществления операций над ними. Наиболее известными операциями являются арифметические операции, которые были использованы в упражнениях в главе 1. Основные операции приведены в табл. 1.2. Помимо арифметических операций, над целыми числами можно проводить побитовые преобразования (табл. 2.1).

Таблица 2.1

Побитовые операции над целыми числами

$x   y$	Побитовое <i>или</i>
$x \wedge y$	Побитовое <i>исключающее или</i>
$x \& y$	Побитовое <i>и</i>
$x \ll n$	Битовый сдвиг влево
$x \gg y$	Битовый сдвиг вправо
$\sim x$	Инверсия битов

#### Пример

Необходимо вывести на экран результат вычисления арифметического выражения четных величин.

Программа и результат ее выполнения представлены на рис. 2.1.

Из кода программы видно, что встречаются две инструкции: инструкция `input()`, которая обеспечивает диалог с пользователем, и `print()`, которая выводит на экран результаты выполнения. Более подробно они будут рассмотрены в соответствующем параграфе. Также здесь представлен оператор присвоения. К сожалению, программирование отличается тем, что невозможно изучить отдельный вопрос или тему без использования других тем. Поэтому в примерах мы будем применять ряд простейших и очевидных инструкций, подробное рассмотрение которых будет отражено далее.



```
File Edit Format Run Options Window Help
import math
print ('Программа вычисления n-го четного числа')
print ('Введите число')
n=int(input())
a=2*n
print ("Четное число с номером ",n," составит ",a, " единиц")

Ln: 9 Col: 0

Программа вычисления n-го четного числа
Введите число
5
Четное число с номером 5 составит 10 единиц
>>>
----- RESTART: D:/Рама/Работа с python/upr6.py -----
Программа вычисления n-го четного числа
Введите число
10
Четное число с номером 10 составит 20 единиц
>>>
```

Рис. 2.1. Код программы Python

## Упражнение 2.1

Выведите на экран результат вычисления арифметического выражения нечетных величин.

### 2.1.2. Базовые числовые типы `int` и `float`

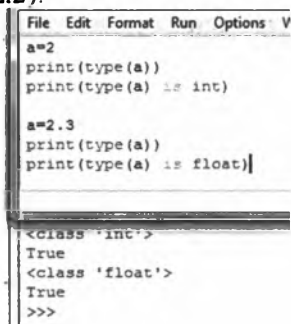
В любых языках программирования используют целые и вещественные типы данных для работы с числами. Python также поддерживает базовые `int` (целые) и `float` (вещественные, числа с плавающей точкой) числовые типы данных. Диапазон целых чисел составляет  $[-2\ 147\ 483\ 647; 2\ 147\ 483\ 647]$ , а вещественных —  $[4.9406564584234654E-324; 1.7976931348623157E308]$ .

**Справка.** Такие диапазоны связаны с особенностями хранения чисел в памяти компьютера в битовом формате, где на каждый знак, порядок или мантиссу отводится бит памяти, в который можно записать только 0 или 1.

Однако совсем не обязательно для переменных (еще одно понятие, которое рассматривается позже), которые являются в Python ссылками на объекты, задавать тип данных, так же как и декларировать их. Это отличает Python от большинства других высокоуровневых языков. Python сам «понимает», какой тип данных у объекта. Это, во-первых, уменьшает код программы, а во-вторых, позволяет программисту не задумываться о типе переменных. При этом в процессе выполнения программы переменная может менять свой тип в зависимости от примененной к ней операции.

## Упражнение 2.2

Введите следующий код программы и посмотрите на результат ее выполнения (рис. 2.2).



```
File Edit Format Run Options V
a=2
print(type(a))
print(type(a) is int)

a=2.3
print(type(a))
print(type(a) is float)

<class 'int'>
True
<class 'float'>
True
>>>
```

Рис. 2.2. Код программы Python

Здесь можно заметить, что для определения типа переменной можно использовать инструкцию `type()`. К тому же ее можно «обернуть» (вложить) в другую функцию, чего некоторые языки не позволяют сделать. Обратим внимание, что для задания целого и вещественного числа (целая и дробная часть отделяется точкой) используется одна и та же переменная, которая в процессе исполнения меняет свой тип. В других языках, например, VBA или Delphi, компилятор выдал бы ошибку «type mismatch» — «несоответствие типов». Это также преимущество Python. Инструкция `is` позволяет проверить истинность утверждения.

Помимо базовых числовых типов `int` и `float`, Python поддерживает следующие типы данных:

- `long` — длинное целое ограничивается объемом оперативной памяти;
- `complex` — комплексный тип, содержащий два числа, которые определяют вещественную и мнимую части. Такой тип редко встречается в других языках программирования.

### 2.1.3. Числовые литералы

*Числовые литералы* — это не что иное как задание чисел в программе или интерпретаторе. Например, 5, 7 или 2.23 являются числовыми литералами, в других языках они называются константами. Однако простое задание литералов, если только Python не используется как калькулятор, не имеет смысла. Обычно в программе значение присваивается переменной. В более общем понимании числовой литерал — это числовая запись. Различают целые, дробные литералы, литералы, представленные в различных системах счисления, различной точности и т.п.

В Python под литералом понимают запись, которая создает объект, в данном случае число.

На рис. 2.2 было задано два числовых литерала целого и вещественного типа, которые были присвоены переменной `a` в коде программы.

### Упражнение 2.3

В режиме интерпретатора введите литерал `2` и прибавьте к нему литерал `3.43`. Сравните полученный результат с результатом, представленным на рис. 2.3.

```
>>> 2
2
>>> _+3.43
5.43
>>> |
```

Рис. 2.3. Работа с числовыми литералами Python

На рис. 2.3 была использована операция «`_`», означающая работу с предыдущим результатом обработки данных, т.е.  $2+3.43=5.43$ .

### 2.1.4. Операторы для работы с числовыми объектами

*Операторы* — это простейшие конструкции языка программирования, которые позволяют выполнить ту или иную команду. Для работы с числовыми объектами в Python могут применяться следующие типы операторов:

- 1) арифметические операторы (см. табл. 1.2);
- 2) побитовые операторы (см. табл. 2.1);
- 3) операторы сравнения (табл. 2.2). Дают логический результат True, если выражение истинно, или False, если выражение ложно.

Таблица 2.2

Операторы сравнения<sup>1</sup>

Оператор	Описание	Пример
<code>==</code>	Условие становится истинным, если литерал, стоящий слева от оператора, равен литералу, стоящему справа	<code>7 == 7 - True</code>
<code>!=</code>	Условие становится истинным, если литерал, стоящий слева от оператора, не равен литералу, стоящему справа	<code>11 != 7 - True</code>

<sup>1</sup> Образовательный портал Pythonicway. URL: <http://pythonicway.com/python-operators>

Оператор	Описание	Пример
<>	Условие становится истинным, если литерал, стоящий слева от оператора, не равен литералу, стоящему справа	11<>7 – True
>	Условие становится истинным, если литерал, стоящий слева от оператора, больше литерала, стоящего справа	7>3 – True
<	Условие становится истинным, если литерал, стоящий слева от оператора, меньше литерала, стоящего справа	3 <7 – True
>=	Условие становится истинным, если литерал, стоящий слева от оператора, больше и равен литералу, стоящему справа	5>= 5 – True.
<=	Условие становится истинным, если литерал, стоящий слева от оператора, меньше и равен литералу, стоящему справа	4 <= 6 – True.

#### Упражнение 2.4

В режиме интерпретатора проверьте истинность выражений:  $3=2$ ,  $3>2$ ,  $3>=2$ ,  $5!=3$ ,  $6<=3$ . Сравните с результатом, представленным на рис. 2.4.

```

>>> 3==2, 3>2, 3>=2, 5!=3, 6<=3
(False, True, True, True, False)
>>>

```

Рис. 2.4. Использование операторов сравнения Python

#### 2.1.5. Форматы чисел

Обычно под *форматом чисел* понимают форму их представления для пользователя, например число знаков до и после запятой, центрирование на экране и т.п. Поэтому инструкция `format()` используется в качестве метода инструкции `print()`, которая была упомянута ранее, а более детально будет рассмотрена позже.

Синтаксис инструкции `format` можно представить в следующем виде:

```
print("{: опции}".format(число))
```

Часто используемые опции, которые не являются обязательными, приведены ниже:

- символ заполнителя (например «\*»);

- символ выравнивания "<" по левому краю, ">" по правому краю, "^" по центру, "=" требование заполнять пространство между знаком числа и его первой значащей цифрой;
- "+" — обязательный вывод знака числа;
- "-" — вывод знака только для отрицательных чисел;
- " " — пробел для положительного числа и "-" — для отрицательного числа.

Далее указывается число цифр, которые будут выводиться на экран.

Для вещественных чисел можно указать общее число цифр и число знаков после запятой в формате  $x.f$ . Если требуется вывести число в виде мантииссы, то формат будет  $x.fg$ . Здесь после точки указывается число значащих цифр. Если использовать символ  $e$ , то данные будут выводиться в экспоненциальном формате  $x.fе$ . В этом случае после точки ставится количество знаков в дробной части числа.

Можно указать тип системы исчисления:  $b$  — двоичная,  $o$  — восьмеричная,  $x$  — 16-ричная в нижнем регистре,  $X$  — 16-ричная в верхнем регистре,  $d$  — десятичная системы счисления.

Можно применять форматирование для нескольких чисел. Тогда необходимо использовать "[ ]":

```
print("{: опции1} {: опции2}").format(число1,
число2))
```

Реализацию метода `format()` удобней рассмотреть на примерах:

```
print("{:*+12}").format(3269204.23)
print("{:*=12}").format(-8756703.233)
```

Даст следующий результат:

```
+*3269204.23
-8756703.233
```

В первом случае в качестве заполнителя был использован символ "\*", обязателен вывод знака и общее количество разрядов 12:

```
print("{:011}").format(1756703)
print("{:011}").format(-27567203)
```

Даст следующий результат:

```
00001756703
-0027567203
```

В этом случае указано общее количество разрядов, при этом заполнение осуществляется нулями до значащих цифр:

```
print("{:*^+15}").format(34512345)
```

Результат:

```
***+34512345***
```

Представление в различных числовых типа:

```
print("{:10.2}".format(2345.123))
print("{:10.2f}".format(2345.123))
print("{:10.2g}".format(2345.123))
print("{:10.2e}".format(2345.123))
```

Результат:

```
2.3e+03
      2345.12
      2.3e+03
      2.35e+03
```

Представление в различных системах счисления:

```
print("{0: b} {0: o} {0: x} {0: X} {0: d}".
format(2345996))
print("{0: #b} {0: #o} {0: #x} {0: #X} {0: #d}".
format(2345996))
```

Результат:

```
1000111100110000001100 10746014 23cc0c 23CC0C 2345996
0b1000111100110000001100 0o10746014 0x23cc0c 0X23CC0C
2345996
```

Во втором случае указывается и тип системы счисления. Достаточно добавить символ "#".

Применение формата для нескольких чисел:

```
print("[{:010.2e}] [{:010.2f}]".format(2345.123,
32112.456))
```

Результат:

```
[002.35e+03] [0032112.46]
```

Как видно из примеров, Python предоставляет обширные возможности по представлению форматированных чисел.

### Упражнение 2.5

В режиме IDLE выберите два произвольных числа (целое и вещественное) и выведите их в следующих форматах:

- 1) используйте заполнитель "?" с центрированием по правому краю;
- 2) формат данных вещественный, в виде мантиссы и экспоненциальный;
- 3) для целого числа представьте его в различных системах счисления с указанием ее типа.

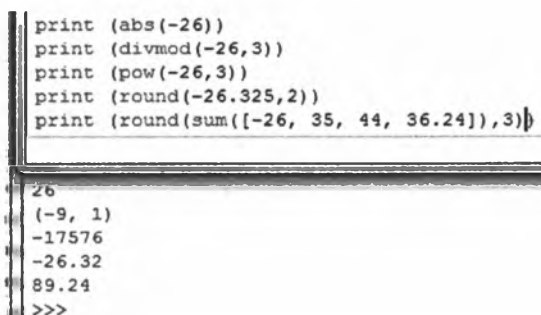
## 2.1.6. Встроенные функции и модули для работы с числами

Помимо стандартных арифметических операций и операций сравнения есть еще встроенные непосредственно в Python функции:

- `abs()` — абсолютное значение числа;
- `divmod(a, b)` — выводит целую часть и остаток от деления числа `a` на число `b`;
- `pow(a, b)` — `a` возводит в степень `b`;
- `round(a, b)` — округляет `a` с точностью до `b` знаков после запятой;
- `sum([a1, a2, ..., an])` — считает сумму последовательности.

### Упражнение 2.6

В режиме IDLE выберите два произвольных числа (целое и вещественное) и примените к ним соответствующие встроенные функции по примеру, представленному на рис. 2.5. Проверьте полученный результат.



```
print (abs(-26))
print (divmod(-26,3))
print (pow(-26,3))
print (round(-26.325,2))
print (round(sum([-26, 35, 44, 36.24]),3))

26
(-9, 1)
-17576
-26.32
89.24
>>>
```

Рис. 2.5. Использование встроенных функций Python

При использовании стандартных математических функций необходимо подключить модуль `math`:

```
from math import * или просто import math
```

При вызове соответствующей функции необходимо указать <имя модуля>.<имя функции>.

Основные функции представлены ниже [13]:

- `math.ceil(X)` — округление до ближайшего большего числа;
- `math.copysign(X, Y)` — возвращает число, имеющее модуль такой же, как и у числа `X`, а знак — как у числа `Y`;
- `math.fabs(X)` — модуль `X`;
- `math.factorial(X)` — факториал числа `X`;
- `math.floor(X)` — округление вниз;
- `math.fmod(X, Y)` — остаток от деления `X` на `Y`;
- `math.frexp(X)` — возвращает мантиссу и экспоненту числа;

- **math.ldexp** (X, I) —  $X * 2^I$ . Функция, обратная функции **math.frexp()**;
- **math.fsum** (последовательность) — сумма всех членов последовательности. Эквивалент встроеной функции **sum()**, но **math.fsum()** более точна для чисел с плавающей точкой;
- **math.isfinite** (X) — является ли X числом;
- **math.isnan** (X) — является ли X NaN (Not a Number — не число);
- **math.modf** (X) — возвращает дробную и целую часть числа X. Оба числа имеют тот же знак, что и X;
- **math.trunc** (X) — отсекает значение X до целого;
- **math.exp** (X) —  $e^X$ ;
- **math.expm1** (X) —  $e^X - 1$ . При  $X \rightarrow 0$  точнее, чем **math.exp** (X) - 1;
- **math.log** (X, [base]) — логарифм X по основанию base. Если base не указан, то вычисляется натуральный логарифм;
- **math.log1p** (X) — натуральный логарифм (1+X). При  $X \rightarrow 0$  точнее, чем **math.log** (1+X);
- **math.log10** (X) — логарифм X по основанию 10;
- **math.log2** (X) — логарифм X по основанию 2;
- **math.pow** (X, Y) —  $X^Y$ ;
- **math.sqrt** (X) — квадратный корень из X;
- **math.acos** (X) — арккосинус X. В радианах;
- **math.asin** (X) — арксинус X. В радианах;
- **math.atan** (X) — арктангенс X. В радианах;
- **math.atan2** (Y, X) — арктангенс Y/X. В радианах. С учетом четверти, в которой находится точка (X, Y);
- **math.cos** (X) — косинус X (X указывается в радианах);
- **math.sin** (X) — синус X (X указывается в радианах);
- **math.tan** (X) — тангенс X (X указывается в радианах);
- **math.hypot** (X, Y) — вычисляет гипотенузу треугольника с катетами X и Y (**math.sqrt** (x \* x + y \* y));
- **math.degrees** (X) — конвертирует радианы в градусы;
- **math.radians** (X) — конвертирует градусы в радианы;
- **math.cosh** (X) — вычисляет гиперболический косинус;
- **math.sinh** (X) — вычисляет гиперболический синус;
- **math.tanh** (X) — вычисляет гиперболический тангенс;
- **math.acosh** (X) — вычисляет обратный гиперболический косинус;
- **math.asinh** (X) — вычисляет обратный гиперболический синус;
- **math.atanh** (X) — вычисляет обратный гиперболический тангенс;



**math.erf** (X) – функция ошибок;

**math.erfc** (X) – дополнительная функция ошибок (1 – math.erf (X));

**math.gamma** (X) – гамма-функция X;

**math.lgamma** (X) – натуральный логарифм гамма-функции X;

**math.pi** –  $\pi = 3,1415926\dots$ ;

**math.e** –  $e = 2,718281\dots$

### Упражнение 2.7

Используя стандартные функции модуля math, вычислите выражения, сделайте скриншот, результат представьте в виде табл. 2.3.

Таблица 2.3

Таблица отчета

Выражение	Результат выполнения

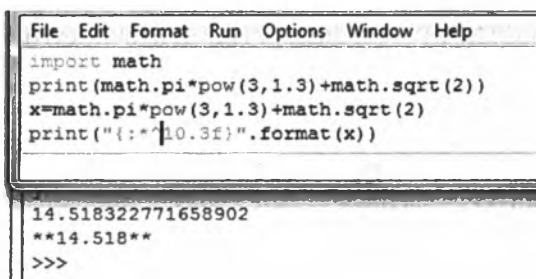
$$1 + \sin^2(n) \cdot 3^3 + \arctg(8 \cdot \pi / 3);$$

$$\frac{\ln^2(n^3)}{1 + n^2} \bmod(2n) + 4 \cos^2(n) + \operatorname{arccctg}(9 \cdot \pi / 2);$$

$$\frac{2^n(n^3)}{n^3 + n^2} \operatorname{div}(2n) + 4 \cos^2(n) \operatorname{tg}(n + 6), \text{ округлить до целых};$$

$$\sqrt{n^3 + n^2} / \frac{2^n(n^3)}{3^4} \cdot (3n) + 4 \cos^2(n) \operatorname{tg}(n + 6) \bmod(2), \text{ взять остаток.}$$

Пример решения аналогичной задачи представлен на рис. 2.6.



```
File Edit Format Run Options Window Help
import math
print(math.pi*pow(3,1.3)+math.sqrt(2))
x=math.pi*pow(3,1.3)+math.sqrt(2)
print("{:.*f}".format(x))

14.518322771658902
**14.518**
>>>
```

Рис. 2.6. Использование функций модуля math Python

Также широко известен модуль numpy, который позволяет работать с массивами чисел.

### 2.1.7. Преобразование и смешивание в выражениях значений разных типов

Иногда возникает необходимость преобразовать значения чисел и представлять их в едином формате и единой системе счисления с целью получения корректного результата при проведении операций с числами различного типа. Для этого в Python предусмотрены различные операции.

Преобразование типов представлено в табл. 2.4.

Таблица 2.4

Операции с числами. Преобразование типов

int (x)	Преобразование к целому
float (x)	Преобразование к числу с плавающей точкой
complex (x, y)	Преобразование к комплексному типу $x+i*y$ (по умолчанию $y=0$ )
z.conjugate()	Сопряженное комплексному числу z

Преобразование систем счисления представлено ниже:

- **int** ([object], [основание системы счисления]) — преобразование к целому числу в десятичной системе счисления. По умолчанию система счисления десятичная, но можно задать любое основание от 2 до 36 включительно;
- **bin** (x) — преобразование целого числа в двоичную строку;
- **hex** (x) — преобразование целого числа в 16-ричную строку;
- **oct** (x) — преобразование целого числа в восьмеричную строку;
- **float.as\_integer\_ratio()** — пара целых чисел, чье отношение равно этому числу;
- **float.is\_integer()** — является ли значение целым числом;
- **float.hex()** — переводит float в hex (16-ричную систему счисления).

#### Упражнение 2.8

Используя произвольно выбранное число, осуществите преобразование типов, как показано на рис. 2.7.

```
a=int(123)
print (type(a))
a=float(123)
print (type(a))
a=complex(123, 321)
print (a)
print (type(a))
a1=a.conjugate()
print (a1)
```

```
<class 'int'>
<class 'float'>
(123+321j)
<class 'complex'>
(123-321j)
<class 'complex'>
>>>
```

Рис. 2.7. Использование операций преобразования типов Python

## Упражнение 2.9

Используя произвольно выбранное число, осуществите преобразование систем счисления, как показано на рис. 2.8.

```
a=int(123)
print (a)
print(bin(123))
print(hex(123))
print(oct(123))
print(float.as_integer_ratio(2.5))
print(float.is_integer(2.5))
print(float.hex(2.5))
```

```
123
0b1111011
0x7b
0o173
(5, 2)
False
0x1.4000000000000p+1
>>>
```

Рис. 2.8. Использование операций преобразования систем счисления Python

### Задания для самостоятельного выполнения по теме «Работа с числами»

1. Петр Петрович все утро простоял в очереди к нотариусу и от скуки пересчитывал людей. В очереди стояло  $n$  человек. Сколько человек находилось в очереди между  $i$ -м и  $k$ -м?
2. Сколько нечетных чисел на отрезке  $(a, b)$ , если  $a$  и  $b$  – четные? Если  $a$  и  $b$  – нечетные? Если  $a$  – четное,  $b$  – нечетное?
3. Сколько полных минут и часов содержится в  $x$  секундах?
4. В доме девять этажей, на каждом этаже одного подъезда по четыре квартиры. В каком подъезде и на каком этаже находится  $n$ -я квартира?
5. Старинными русскими денежными единицами являются: 1 рубль – 100 копеек, 1 гривна – 10 копеек, 1 алтын – 3 копейки, 1 полушка – 0,25 копейки. Имеется  $A$  копеек. Запишите выражения для представления имеющейся суммы в рублях, гривнах, алтынах и полушках.
6. Стрелка прибора вращается с постоянной скоростью, совершая  $w$  оборотов в секунду (не обязательно стрелка прибора, может быть, это волчок в игре «Что? Где? Когда?» и т.п.) Угол поворота стрелки в нулевой момент времени примем за ноль. Каков будет угол поворота через  $t$  секунд?
7. Вы стоите на краю дороги, и от вас до ближайшего фонарного столба –  $x$  метров. Расстояние между столбами –  $y$  метров. На каком расстоянии от вас находится  $n$ -й столб?
8. Та же ситуация, что и в предыдущей задаче. Длина вашего шага  $z$  метров. Мимо скольких столбов вы пройдете, сделав  $n$  шагов?

## 2.2. ПОСЛЕДОВАТЕЛЬНОСТИ. РАБОТА СО СТРОКАМИ

В Python используется такое понятие, как последовательности, т.е. упорядоченный или неупорядоченный набор символов, над которыми предусмотрено выполнение определенных операций. Наиболее распространенные последовательности в Python — это строки и кортежи (неизменяемые последовательности) и списки (изменяемые последовательности). По сути, последовательности — это те же массивы, которые широко используются в большинстве языков программирования. Вообще в Python предусмотрено шесть типов последовательностей [5]: это строки (string), юникод-строки (unicode), списки (list), кортежи (tuple), буферы (buffer) и x-range объекты.

### 2.2.1. Строки. Литералы строк. Специальные символы

*Строка* в общем понимании — это набор символов. При этом строку нельзя изменять, а возможно только создать новую строку из части предыдущей строки. *Литералы строк* в Python — это представление упорядоченного набора символов, содержащего в себе текстовую информацию о чем-либо. Литералы строк могут быть записаны в апострофах или кавычках, в тройных кавычках или с помощью специальных символов.

Апострофы и кавычки выполняют одну и ту же функцию, и интерпретатор Python воспринимает заключенный между ними текст как строку.

#### Упражнение 2.10

Введите следующий код:

```
print('Python это язык программирования')
print("Python это язык программирования")
```

Сравните полученный результат.

Введите следующий код:

```
print('Python это"язык программирования"')
print("Python это 'язык программирования'")
```

Данный код не выдает ошибки.

Введите следующий код:

```
print("Python это"язык программирования"")
print('Python это 'язык программирования''')
```

После запуска инструкций вы получите сообщение: «Syntax Error: invalid syntax». Для того чтобы избежать такой ошибки, для вывода апострофа или кавычки используется левый слэш «\».

Введите код:

```
print("Python это \"язык программирования\"")  
print('Python это \'язык программирования\')
```

В результате апостроф и кавычка отображается.

Для того чтобы вывести символ, можно применить этот способ.

Обычно апострофы и одинарные кавычки используются для экранирования одной строки.

Чтобы вывести текст в несколько строк, употребляются тройные апострофы или кавычки. Последний способ используется для написания комментариев в «теле» программы, а также при создании документации.

Знак «\» чаще всего применяется при необходимости вывода специальных символов, а также при представлении текста в заданной форме (табл. 2.5).

Таблица 2.5

Специальные символы при работе со строками

Символ	Описание
\n	Перевод строки
\r	Возврат каретки
\a	Звонок
\b	Забой
\f	Перевод страницы
\v	Вертикальная табуляция
\t	Горизонтальная табуляция

На рис. 2.9 представлен пример использования служебных символов.

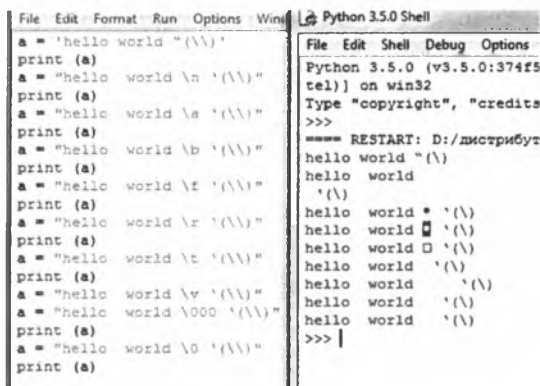


Рис. 2.9. Использование служебных символов при работе со строками Python

Также возможно представление символов в различных системах счисления.

Еще один способ — это подавление экранирования, когда перед строкой ставится символ «r»:

```
>>> print(r'd:\\ROMAN\SQLEXPRESS')
d:\\ROMAN\SQLEXPRESS
```

Как видно из примера, знак экранирования «\» подавляется и слэш выводится на экран. Иными словами, выводится вся строка.

### 2.2.2. Операции над строками

*Конкатенация* — это сложение строк. При сложении строк формируется новая строка. Для этого используется обычный символ сложения:

```
>>>print('hello'+ ' world')
hello world
>>>print('hello'+ 'beautiful'+ ' world')
hello beautiful world
```

Как видно из примера, можно складывать несколько строк одновременно.

Иногда возникает необходимость повторять строки. Для этого используется символ умножения «\*».

#### Пример

Необходимо вывести фразу «Много много много очень очень много данных». Введем следующий код:

```
>>> print('много '*3, 'очень '*2, 'много ', 'данных')
```

Результат выполнения:

```
много много много  очень очень  много  данных
```

Иногда требуется вывести отдельный символ из строки. Для этого используется схема доступа по индексу.

#### Пример

Вывести третий символ слова Python:

```
>>> s1='Python'
>>> print(s1 [2])
```

Результат:

```
t
```

Как и в большинстве языков программирования, нумерация начинается с «0».

### Упражнение 2.11

По заданному пользователем номеру символа строки «бизнес-информатика» — одно из современных направлений подготовки вывести его на экран.

**Справка.** Ранее была представлена инструкция `input()`, позволяющая вводить данные пользователем, а также инструкция `int` — преобразование строки в число. Совместив эти инструкции, а также инструкцию `print`, можно обеспечить ввод данных.

Рассмотрим два способа.

1. Способ в две строки:

```
print('Введите номер символа ')
i=int(input())
```

2. Способ в одну строку:

```
i=int(input('Введите номер символа: '))
```

Как видно из примера, Python позволяет совмещать несколько операций в одно выражение, что также является одним из его преимуществ.

Возникают задачи, когда необходимо «вычлени́ть» часть текста из строки или получить подстроку. Для этого используется следующая конструкция:

```
st[i: j: step].
```

Здесь `st` — строка, `i` — индекс входа, `j` — индекс выхода, а `step` — шаг выборки начиная с первого символа.

### Пример

Необходимо вывести из строки «Python это язык программирования» подстроку начиная с 16 символа и заканчивая 30. Вывести из подстроки каждый второй символ начиная с первого.

Введем следующий код:

```
st='Python это язык программирования'
print(st[16:30])
print(st[16:30:2])
```

В результате получим:

```
программирован
пормиоа
```

Данная операция также называется срезом строки.

Еще одна операция — это проверка вхождения подстроки в строку. Для этого в Python предусмотрена инструкция `find(str, [start], [end])`.

Здесь `str` — искомая подстрока, `[start]` — индекс входа, `[end]` — индекс выхода.

**Справка.** В языках программирования принято необязательные атрибуты инструкции заключать в `[]`.

Данная инструкция выводит номер вхождения заданной подстроки.

### Пример

Данный код выводит номер вхождения подстроки «программ» в строку «Python это язык программирования»:

```
st='Python это язык программирования'  
print(st.find('программ', 10,30))  
print('Python это язык программирования'.find('программ',  
10,30))
```

Как видно из примера, в первом случае создается строковая переменная, которая является ссылкой на объект, и к ней применяется операция (метод). Во втором случае записывается исходная строка. Более подробно о переменных речь пойдет ниже. Результат получается один и тот же, равный 16. Если бы подстроки не существовало, то обработчик выдал бы значение -1.

**Справка.** В Python используется объектно-ориентированный подход интерпретации кода программы. Все созданные переменные по сути являются ссылками на объекты, к которым можно применять методы.

Конструкция применения метода к объекту такова:

<имя объекта>.<имя метода>.

В дополнение к представленной инструкции Python позволяет получить номер последнего вхождения, если подстрок несколько (`rfind(str, [start], [end])`).

Альтернативой этих инструкций являются инструкции `index(str, [start], [end])` и `rindex(str, [start], [end])` с тем отличием, что при отсутствии подстроки выводится ошибка `ValueError`.

Еще одна полезная инструкция — это определение длины строки, которая часто используется при организации циклов:

```
len(str).
```

### 2.2.3. Функции и методы для работы со строками

Метод в Python — это функция, которая применяется к созданному объекту. Если мы не создаем объект, а применяем его непосредственно к литералу, то это функция. Если переменная создана, то его преобразование осуществляется посредством метода.

Наиболее известные методы и функции представлены ниже.

`st.replace(шаблон, замена)` — заменяет шаблон в строке на замену.

### Пример

Необходимо заменить в строке «Python это язык программирования» все буквы «o» на «a».



Введем следующий код:

```
print(st)
print(st.replace('o','a'))
```

Результат:

Python это язык программирования

Python эта язык праграммиравания

### Упражнение 2.12

Дан следующий текст. «Я, ФИО настоящим удостоверяю, что... Также ФИО обязуется исполнять... ФИО / Подпись». Замените ФИО своей фамилией и инициалами. Текст после «...» должен начинаться с новой строки.

Метод `st.split` (разделитель) позволяет разделять строки на подстроки по разделителю.

#### Пример

```
s1='123;24;45;16;74;32;98;09;4'
```

```
s2=s1.split(';')
```

```
print(s2)
```

Результат:

```
['123', '24', '45', '16', '74', '32', '98', '09', '4']
```

Таким образом, получилась последовательность, состоящая из подстрок, представленная в виде списка, о котором речь пойдет позже.

Метод `st.join([подстрока1, подстрока2,...])` является обратным предыдущему методу и собирает соответствующие подстроки в строку с разделителем `st`.

#### Пример

```
db_con=';'.join(['BG', 'MSSQLServer', '1431', 'user', 'pswd'])
```

```
print(db_con)
```

Результат:

```
BG; MSSQLServer; 1431; user; pswd
```

Данный пример демонстрирует передачу переменной данных о подключении к серверу.

Ниже приводится список функций и методов работы со строками:

- `isdigit()` — проверка того, что строка состоит из цифр;
- `isalpha()` — то же, только из букв;
- `isalnum()` — проверка состава из букв и цифр;
- `islower()` — проверка того, что все символы строки в нижнем регистре;

- `isupper()` — проверка того, что все символы строки в верхнем регистре;
- `istitle()` — проверка того, что строка начинается с прописного символа;
- `isspace()` — проверка того, что строка содержит специальные символы.

### Пример

```
print('\n Привет'.isspace())
```

```
print('\n\f'.isspace())
```

Первое выражение выведет `False`, второе — `True`.

Методы `lower()` и `upper()` переводят строку в нижний и верхний регистр соответственно.

- `startswith(шаблон)` — проверка того, что строка начинается с шаблона;
- `endswith(шаблон)` — проверка того, что строка заканчивается шаблоном.

`ord(символ)` и `chr(символ)` переводят символ в код ASCII и наоборот соответственно.

Например, код

```
print(ord('w')); print(chr(119))
```

даст соответственно 119 и `w`.

**Справка.** Заметим, что использование знака «`>`» дает возможность инструкции записывать в одну строку.

- `capitalize()` — перевод первого символа строки в верхний, а всех остальных — в нижний регистр;
- `title()` — перевод первого символа каждого слова строки в верхний регистр;
- `swapcase()` — перевод символов верхнего регистра в нижний регистр, а нижнего — в верхний;
- `center(длина [, символ])` — выводит по центру строку заданной длины, где по краям стоят символы (по умолчанию стоит знак пробела).

### Пример

```
print('Python'.center(10, '*'))
```

выведет `**Python**`

- `count(шаблон [, start, [end]])` — считает количество вхождений шаблона в строке;
- `lstrip([символ])`, `rstrip([символ])`, `strip([символ])` — удаляет символы в начале, в конце, с начала и конца строки соответственно (по умолчанию — пробел);

- `ljust(width, fillchar='символ')` — создает строку длины не меньшей, чем `width`, заполняя остальное заданным символом;
- `rjust(width, fillchar='символ')` — создает строку длины не меньшей, чем `width`, заполняя первые позиции заданным символом;
- `format()` — форматирует строки. Действует так же, как и с числами. Подробнее будет рассмотрено ниже.

#### 2.2.4. Форматирование строк

Форматирование строк является одним из мощных инструментов представления текстовой информации. Все другие типы могут быть приведены к строке посредством оператора `str()`.

Существует два подхода: стандартный и с использованием шаблонов [10].

Стандартный подход предполагает использование стандартного оператора «%». «Операция форматирования заменяет в строке форматирования все записи, начинающиеся с процента и оканчивающиеся буквой, на данные, указанные справа от операции, по очереди. Эта операция позволяет выводить текст с включениями чисел, строк и других данных в поля определенной ширины, выровненные по левому или правому краю. Для чисел строка форматирования дает возможность указывать количество цифр после запятой» [8]. Другими словами, оператор «%» используется, когда в строку нужно добавить некоторые элементы строки, которые принимают определенные значения в процессе выполнения программы.

Структура оператора имеет вид

'% <набор спецификаторов>%' <что нужно форматировать>.

В качестве *примера* рассмотрим следующий код:

```
print('%s%' 'Привет')
print('Привет%s%' 'мир')
```

Даст результат:

Привет

Привет мир

Здесь спецификатор `s` сообщает обработчику, что в качестве подставляемого значения будет передана строка.

Если в качестве спецификатора используется `%d`, `%i` или `%u`, то будет передано десятичное число, `%f` — число с плавающей точкой, `%%` — будет выведен процент. Также можно указать `%o` — число в восьмеричной системе, `%x`, `%X` — 16-ричная система в нижнем и верхнем регистре, `%e`, `%E` — экспоненциальное представление числа с плавающей точкой, `%c` — код символа.

Если перед спецификатором `d`, `f` или `e` значения указать `0X.X`, где `X` — цифра, то первая `X` будет обозначать число разрядов, вторая `X` — число знаков дробной части, `0` — заполняет незанятые разряды. `0` в данном случае называется флагом.

Используются следующие флаги: «`#`» — альтернативная форма, «`-`» или «`+`» — свободное место заполняется пробелами справа, «`+`» — пробелы слева.

Такое представление очень похоже на форматирование чисел, рассмотренное выше.

Для лучшего понимания форматирования рассмотрим следующий пример, где используются переменные, описание которых также будет представлено далее:

```
s1='Привет'  
s2='Мираполис'  
d=123.456  
f=4/3  
print('%s'%s1, '%.3s'%s2)  
print('%s%.3s'% (s1, s2))  
print('%%.4u%+6.3f'% (d, f))
```

В результате выполнения программы получим:

```
Привет Мир  
Привет Мир  
0123 +1.333
```

Аналогичный результат получится, если на место переменных вставить их значения:

```
print('%s'% 'Привет', '%.3s'% 'Мираполис')  
print('%s%.3s'% ('Привет', 'Мираполис'))  
print('%%.4u%+6.3f'% (123.456, 4/3))
```

В первой строке мы присваиваем значение 'Привет', а потом добавляем первые три знака второго значения 'Мираполис'.

Во второй строке задается форматирование каждого из передаваемых значений, а после знака «`%`» в скобках через запятую передаются сами значения.

В третьей строке мы задаем число знаков для десятичного числа, а для второго числа задаем число разрядов и число знаков после запятой.

Можно использовать любой вариант, при этом результат будет одинаков.

Так же как и для чисел, форматирование строк можно осуществить с помощью специального оператора `.format()`.

Его спецификация и результаты выполнения представлены на рис. 2.10–2.13.

```
поле замены ::= "[" [имя поля] ["(" преобразование) [":" спецификация] "]"
имя поля   ::= arg_name ("." имя атрибута | "[" индекс "]" ) *
преобразование ::= "(" (внутреннее представление) | "(" (человеческое представление)
спецификация ::= см. ниже
```

Рис. 2.10. Форматирование строк — структура

```
спецификация ::= [[fill]align][sign][#][0][width][,][.precision][type]
заполнитель ::= символ кроме '[' или '"'
выравнивание ::= "<" | ">" | "=" | ""
знак         ::= "+" | "-" | ""
ширина      ::= integer
точность     ::= integer
тип         ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" |
              "n" | "o" | "s" | "x" | "X" | "a"
```

Рис. 2.11. Форматирование строк — спецификация

```
>>> 'Hello, {}'.format('Vasya') # вставить между
'Hello, Vasya'
>>> '{0}, {1}, {2}'.format('a', 'b', 'c') # вставить по порядку
'a, b, c'
>>> '{}, {2}, {}'.format('a', 'b', 'c') # вставить по порядку
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c') # вставить по позиции
'c, b, a'
>>> '{2}, {1}, {0}'.format('*abc') # вставить по позиции и разделить текст
'c, b, a'
>>> '{0}({1}{0})'.format('abra', 'cad') # вставить по позиции 2 слова
'abracadabra'
>>> 'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-115.81W')
# присвоить координаты
'Coordinates: 37.24N, -115.81W'
>>> coord = {'latitude': '37.24N', 'longitude': '-115.81W'}
>>> 'Coordinates: {latitude}, {longitude}'.format(**coord)
# присвоить координаты
'Coordinates: 37.24N, -115.81W'
```

Рис. 2.12. Операции форматирования строк Python

```
>>> coord = (3, 5)
>>> 'X: {0[0]}; Y: {0[1]}'.format(coord)
'X: 3; Y: 5'
>>> 'repr() shows quotes: {r}; str() doesn't: {s}'.format('test1', 'test2')
'repr() shows quotes: 'test1'; str() doesn't: test2'
>>> '{:<30}'.format('left aligned')
'left aligned'
>>> '{:>30}'.format('right aligned')
'right aligned'
>>> '{:^30}'.format('centered')
'centered'
>>> '{:~30}'.format('centered') # use '~' as a fill char
'~centered~'
>>> '{:+E}; {:-f}'.format(3.14, -3.14) # show it always
'+3.140000; -3.140000'
>>> '{:E}; {:-f}'.format(3.14, -3.14) # show a space for positive numbers
' 3.140000; -3.140000'
>>> '{:-E}; {:-f}'.format(3.14, -3.14) # show only the minus -- same as '{:f}; {:-f}'
'-3.140000; -3.140000'
>>> # format also supports binary numbers
>>> 'int: {0:d}; hex: {0:x}; oct: {0:o}; bin: {0:b}'.format(42)
'int: 42; hex: 2a; oct: 52; bin: 101010'
>>> # with 0x, 0o, or 0b as prefix
>>> 'int: {0:d}; hex: {0:#x}; oct: {0:#o}; bin: {0:#b}'.format(42)
'int: 42; hex: 0x2a; oct: 0o52; bin: 0b101010'
>>> points = 19.5
>>> total = 22
>>> 'Correct answers: {:.2%}'.format(points/total)
'Correct answers: 88.64%'
```

Рис. 2.13. Операции форматирования строк Python

В простейших случаях программисты используют знак «%». В последних разработках принято применять `.format()`.

### Упражнение 2.13

Дан следующий текст: «Сегодня в “а” будет проходить внеочередное заседание совета по проблемам “b”, организованное “с”. Было подано “d” заявок по проектам на общую сумму “е” руб. Средняя стоимость проекта составила “g” руб. Здесь а — 12 часов 30 минут; b — информатизация образования; с — ооо “мега; информ; инвест”; d — 213; e — 1230456.25;  $g=e/d$ ».

Используя арифметические операции, операции работы со строками и операции форматирования «%» и `.format`, вывести следующий текст: «Сегодня в 12.30 будет проходить внеочередное заседание совета по проблемам «Информатизации образования», организованное ООО «МегаИнформИнвест». Было подано 213 заявок по проектам на общую сумму 1.230 тыс. руб. Средняя стоимость проекта составила 5776.79 руб.».

## 2.2.5. Регулярные выражения

*Регулярные выражения* (regular expression) — это специальные шаблоны, которые позволяют искать заданный текст в строках, заменять его, удалять, форматировать, т.е. осуществлять манипуляцию с элементами строк.

По сути регулярные выражения представляют собой обычные строки, которые могут содержать в себе обычные символы (буквы и цифры) и специальные символы (\*, ^, ?, [], {} и т.п.).

Для работы с регулярными выражениями Python содержит в себе специальный модуль `re`, который можно подключить так же, как и модуль `math`, рассмотренный ранее:

```
import re
```

При работе с регулярными выражениями используется правило конкатенации. Если А и В есть регулярные выражения, то и АВ есть также регулярное выражение.

Для работы с регулярными выражениями наиболее часто применяются следующие функции и методы модуля `re`.

**`re.match` (шаблон, строка)**. Проверяет, соответствует ли начало строки «строка» регулярному выражению «шаблон». Например, выражению `'test'` соответствует строка `'test1'`, но не соответствует строка `'1test'`.

Возвращает объект `MatchObject`, если строка найдена, или `None`, если не найдена.

Также может быть найдена пустая строка, если она соответствует регулярному выражению.

### Упражнение 2.14

Напишите следующий код программы:

```
import re
str1='Первая программа на Python это программа,
позволяющая сделать первый шаг к изучению языка Python'
pattern1='программа'
pattern2='Первая'
print(re.match(pattern1, str1))
print(re.match(pattern2, str1))
```

Запустите ее, и у вас должен получиться следующий результат:  
None

```
<_sre.SRE_Match object; span=(0, 6), match='Первая'>
```

**re.search (шаблон, строка).** Сканирует всю строку до полного совпадения с шаблоном.

### Упражнение 2.15

Добавьте к следующему коду функцию `re.search` и посмотрите на результат:

```
print(re.search(pattern1, str1))
print(re.search(pattern2, str1))
```

В результате должно получиться:

```
<_sre.SRE_Match object; span=(7, 16), match='программа'>
<_sre.SRE_Match object; span=(0, 6), match='Первая'>
```

**Справка.** `Span` – показывает номер вхождения и выхода шаблона.

Следующая функция **re.compile (шаблон)**. Создает объект – шаблон для дальнейшего использования, что ускоряет процесс выполнения программы.

### Пример

```
pat1=re.compile('программа')
print(pat1.match(str1))
print(pat1.search(str1))
```

Выдаст аналогичный результат, что и в упражнениях 2.14 и 2.15.

**re.findall (шаблон, строка).** Сканирует всю строку до полного совпадения с шаблоном и выдает полный список найденных шаблонов.

Если добавить к предыдущему примеру строку

```
print(re.findall(pattern1, str1)),
```

то обработчик выведет:

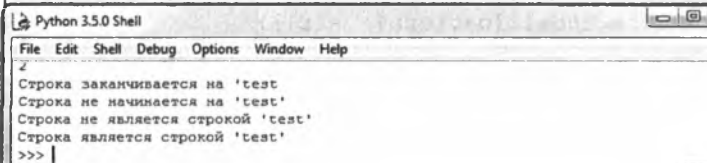
```
['программа', 'программа']
```

**re.finditer (шаблон, строка).** Выводит итератор и действует аналогично предыдущей функции:

<callable\_iterator object at 0x000000002C5B438>

Пример использования специальных символов приведен на рис. 2.14.

```
import re
pattern = 'test'
string = 'This is a simple test message for test'
found = re.findall(pattern, string)
len(found) == string.count('test')
print (len(found))
string = 'This is a simple test message for test'
string2 = 'test'
pattern1 = 'test$'
pattern2 = '^test'
pattern3 = '^test$'
if re.search(pattern1, string) == None: #Строка заканчивается на 'test'
    print()
else: print ("Строка заканчивается на 'test'")
if re.match(pattern2, string) == None: #Строка не начинается на 'test'
    print("Строка не начинается на 'test'")
if re.match(pattern3, string) == None: #Строка не является строкой 'test'
    print("Строка не является строкой 'test'")
if re.match(pattern3, string2) == None: #Строка является строкой 'test'
    print()
else: print ("Строка является строкой 'test'")
```



**Рис. 2.14.** Использование специальных символов в шаблоне

Использование специального символа «\$» в шаблоне в конце шаблона указывает, что строка должна оканчиваться на шаблон. «^» в начале шаблона указывает, что строка должна начинаться с шаблона. Если данный оператор стоит не в начале выражения, то он будет оператором отрицания.

### Пример

Необходимо вывести все трехзначные числа, начинающиеся на 5 и оканчивающиеся на 2, из строки «Число строк программного кода у студентов группы варьировалось от 502 до 652, что в среднем составило 592 строки»:

```
str1='Число строк программного кода у студентов группы  
варьировалось от 502 до 652, что в среднем составило 592  
строки'
```

```
pattern3='5.2'  
print (re.findall (pattern3, str1))
```

Получим следующее:

```
['502', '592']
```



Чтобы указать произвольный символ на соответствующей позиции, ставится точка «.».

#### Шаблон

```
pattern3='..2'  
выведет все три числа  
['502', '652', '592']
```

Для того чтобы указать конкретные символы, используются квадратные скобки []: [д] — ищет символы д в строке; [0-9] — все цифры; [a-z] — все буквы; [321bds] — ищет любой символ, указанный в скобке.

Также после шаблона можно указать следующие символы:

- «\*» — шаблон может повторяться 0 или больше раз (может найтись пустая строка);
- «+» — шаблон может повторяться 1 или больше раз;
- «?» — шаблон может повторяться 0 или 1 раз.

#### Пример

Код:

```
pattern4=' [0-9] '  
print(re.findall(pattern4, str1))
```

Результат:

```
['5', '0', '2', '6', '5', '2', '5', '9', '2'],
```

а при шаблоне

```
pattern4=' [0-9] +'
```

получатся все числа:

```
['502', '652', '592']
```

Также можно формировать составные шаблоны. Например, шаблон ' [\d] + \*- \* [\d] +' позволяет найти сначала группу цифр, потом — произвольное число пробелов, затем — окончание группой цифр.

Подробнее о конструкциях и об обозначениях групп можно прочитать в документации к модулю re.

Использование шаблонов позволяет проводить контекстный анализ текста, что обычно используется при анализе сайтов.

#### *Задания для самостоятельного выполнения по теме «Последовательности. Работа со строками»*

1. Составьте любую строку, содержащую не менее 10 символов. Извлеките из нее последний символ, первый символ, третий с конца и с начала строки. Выведите строку, составленную из извлеченных символов, и определите ее длину.
2. Составьте любую строку, содержащую не менее 10 символов. Извлеките из нее срезы: восемь символов с начала строки, шесть символов

из середины строки, а также символы, индексы которых кратны трем. Выведите строку, составленную из извлеченных символов, и определите ее длину.

3. Создайте произвольную строку, содержащую цифры. Составьте шаблон и код, который бы выводил эти цифры.
4. Строка содержит следующее выражение: «Все смешалось в доме Облонских». Выведите все слова, в которых встречается буква «о».
5. Создайте шаблон для поиска e-mail в тексте.

## 2.3. ПОСЛЕДОВАТЕЛЬНОСТИ. СПИСКИ

*Список* (`list`) — это последовательность, которая может изменяться и содержать разнородные элементы, в том числе другие списки. В других языках программирования аналогом списков являются массивы, но в большинстве своем они строго типизированы, а в Python такого ограничения нет. Как было видно из предыдущей темы, когда осуществлялся поиск по шаблону или с помощью `%`, интерпретатор выводил последовательность вида ['значение1', 'значение2', ...], что есть не что иное, как список. Иными словами, интерпретатор в зависимости от кода программы сам создает список.

### 2.3.1. Создание списка

Создать список можно несколькими способами. Один из них — это использование инструкции `list()`, которая преобразует строку в список.

#### Пример

Преобразовать строку в список

```
>>>print(list('spisok'))  
['s', 'p', 'i', 's', 'o', 'k']
```

Как видно из примера, список заключается в квадратные скобки `[]`.

Второй способ — это непосредственное задание литерала списка:

```
s= [1,2,3,'Python',5, [2,'a']]  
print(s)
```

выдаст

```
[1, 2, 3, 'Python', 5, [2, 'a']]
```

Как видно из примера, список может содержать символы, цифры и даже вложенные списки. Список может быть и пустым:

```
s= []
```

### 2.3.2. Генераторы списков

Генераторы списков позволяют создать новые списки из элементов исходного списка. Для этого используется конструкция `for`, как и для организации цикла.

#### Пример

Создать из слова Python список, содержащий утроенные буквы.

Код:

```
s = [s * 3 for s in 'Python']
```

```
print(s)
```

Результат:

```
['PPP', 'yyy', 'ttt', 'hhh', 'ooo', 'nnn']
```

#### Пример

Создать последовательность из 10 натуральных чисел.

Код:

```
s = [i for i in range(1,11)]
```

```
print(s)
```

```
s = []
```

```
for i in range(1,11):
```

```
    s.append(i)
```

```
print(s)
```

Результат:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

В первом случае используется конструкция цикла `for` и область (`range()`), во втором — непосредственно цикл `for` и метод добавления элементов списка `append()`, которые подробнее будут рассмотрены далее.

Таким образом, существует довольно много способов создания списков. При этом, организуя вложенные циклы, можно создавать и вложенные списки.

#### Упражнение 2.16

Создайте список из выражения «Список — особая форма представления данных».

#### Упражнение 2.17

Создайте список, содержащий пять произвольных букв и пять цифр.

#### Упражнение 2.18

Создайте список из удвоенных натуральных чисел от 1 до 9.

### 2.3.3. Создание копии списка, полная и поверхностная копии списка

Прежде всего, для создания копии списка можно воспользоваться оператором присваивания «=». Однако в Python оператор присваивания создает объект, являющийся ссылкой на объект, созданный ранее.

#### Пример

```
s = ['s', 'p', 'i', 's', 'o', 'k']
new_s = s
print(new_s)
```

Выдаст тот же список

```
['s', 'p', 'i', 's', 'o', 'k']
```

Можно использовать и операцию list():

```
new_s=list(s)
```

Также различают поверхностное и глубокое копирование, которое связано с технологией создания нового списка с разным временем выполнения. Для этого используется встроенный модуль Python copy.

#### Пример

Требуется осуществить поверхностное и глубокое копирование списка ['s', 'p', 'i', 's', 'o', 'k'].

Код:

```
import copy
new_pov=copy.copy(s)
new_gl=copy.deepcopy(s)
print(new_pov)
print(new_gl)
```

Результат один и тот же:

```
['s', 'p', 'i', 's', 'o', 'k']
```

В первом случае создается новый объект, и по мере добавления (изменения) вставляются ссылки на исходный объект (поверхностное копирование). Во втором случае создается новый объект, и по мере необходимости в него вставляются копии исходного объекта. Различие проявляется только при операции изменения списка. Различие проявляется только при операции изменения списков, когда имеются составные изменяемые объекты (например, список списков). Первый способ быстрее, чем второй.

### 2.3.4. Операции над списками

Для доступа к элементу списка по индексу достаточно указать имя списка и в квадратных скобках — значение индекса. Отметим, что индексация элементов списка начинается с нуля.

### Пример

Вывести третий элемент списка.

Код:

```
s= ['s', 'p', 'i', 's', 'o', 'k']  
print(s [2])
```

Результат:

```
i
```

Извлечение среза осуществляется так же, как и для строк, которые были рассмотрены ранее.

### Пример

Извлечь срезы списка.

Код:

```
s= ['s', 'p', 'i', 's', 'o', 'k']  
print(s [2:5]) # извлекает срез со 2-го по 5-й элемент  
print(s [:4]) # извлекает срез до 4-го элемента  
print(s [2:]) # извлекает срез со 2-го элемента  
print(s[:-1]) # извлекает срез до предпоследнего
```

элемента

Результат:

```
['i', 's', 'o']  
['s', 'p', 'i', 's']  
['i', 's', 'o', 'k']  
['s', 'p', 'i', 's', 'o']
```

**Справка.** Если в строке кода указать «#», то все символы после него интерпретатор воспринимает как комментарий и они не обрабатываются. Считается хорошим тоном оставлять комментарии. При этом они очень сильно помогают программисту, если программа содержит довольно много строк кода (сотни или тысячи).

*Операция конкатенации или сложения списков аналогична операции со строками.*

### Пример

Код:

```
s3= [1,2,3]  
s4= [5,6,7]  
s5= []  
print(s3+s4)  
for i in range(0, len(s3)):  
    s5.append(s3 [i] +s4 [i])  
print(s5)
```

Результат:

```
[1, 2, 3, 5, 6, 7]  
[6, 8, 10]
```

В первом случае создается список, содержащий шесть элементов, во втором — три элемента, где в алгоритме был использован цикл `for`, а также функция `len()`, которая возвращает длину списка, т.е. число его элементов.

*Повторение списков* аналогично операции со строками.

### Пример

Повторить список `[1, 2, 3]` 2 раза.

Код:

```
print([1,2,3] *2)
```

Результат:

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Еще один пример работы со списками представлен на рис. 2.15.

```
import math
print ("Программа работы со списками")
print ("Введите слово из 8 символов")
a=(input('слово:')) # Ввод слова
while len(a)!=8: # Проверка числа символов
    print("число символов не равно 8")
    a=input('Введите слово заново:')# Ввод слова заново
b=list(a) # Преобразование строки в список
c1=a[0] # Извлечение первого символа
print(c1) # Печать новой строки
c2=a[-1] # Извлечение последнего символа
print(c2) # Печать новой строки
c3=a[2] # Извлечение третьего символа
print(c3) # Печать новой строки
c4=a[-3] # Извлечение третьего символа с конца
print(c4) # Печать новой строки
c5=c1+c2+c3+c4 # Что получилось
print(c5) # Печать новой строки
print('Длина новой строки составит: ',len(c5), 'символов') # Печать новой строки
```

Ln:9 Col: 7

```
Программа работы со списками
Введите слово из 8 символов
слово:василиса
в
а
с
и
васи
Длина новой строки составит: 4 символов
>>>
```

**Рис. 2.15.** Пример работы со списками

### Упражнение 2.19

Напишите код программы, позволяющий из слов «абра» и «кад» составить слово «абракадабра». При этом необходимо использовать списки.

Напишите следующий код:

```
s1=list('абра')
s2=list('кад')
print(''.join(s1+s2+s1))
```

Посмотрите на результат.

*Замечание.* Конечно, было бы легче просто сложить строки.

Для проверки вхождения элемента в список используется несколько функций. Так, для определения количества вхождений элемента в список используется функция `count()`.

**Пример**

Код:

```
s= [1,2,3,1,1,4,1,5,6]
print(s.count(0))
print(s.count(1))
```

Результат:

```
0
4
```

Для определения первого вхождения используется функция `index(s, [start, [end]])`.

**Пример**

Код:

```
s= [1,2,3,1,4,4,1,5,6]
print(s.index(4))
print(s.index(1,5,8))
```

Результат:

```
4
6
```

В данном примере индекс первого вхождения числа 4 соответствует четырем начиная с 0, т.е. 4 находится на пятой позиции. 6 — это индекс вхождения числа 1 начиная с шестой позиции и заканчивая восьмой.

В случае если элемента нет в списке, интерпретатор выдает `Value Error`. Поэтому, для проверки наличия лучше использовать функцию `count()` или создать обработчик ошибок с помощью конструкции `try... except`.

Перебор элементов списка можно осуществлять с помощью циклов `for` или `while`, подробнее о которых будет рассказано в соответствующей теме.

Также можно обратиться непосредственно к элементу списка, указав соответствующий индекс.

**Упражнение 2.20**

Напишите следующий код программы, соблюдая соответствующие отступы (об отступах подробнее ниже), объясните результат ее выполнения:

```
s=list('Python')
```

```

for i in range(0, len(s)):
    print(s [i])
i=0
while i<len(s):
print(s [i])
i=i+1

```

### 2.3.5. Методы списков

Ниже представлены основные методы списков.

**s.append (x)** — добавляет элемент списка s со значением x. Ранее в примерах этот метод уже был рассмотрен.

**s.extend (x)** — расширяет список s, добавляя к нему список x.

#### Пример

Код:

```

s= [1,2,3]
s.append(4)
print(s)
s.extend(s)
print(s)

```

Результат:

```

[1, 2, 3, 4]
[1, 2, 3, 4, 1, 2, 3, 4]

```

В данном примере список расширил сам себя, что аналогично операции «\*».

**s.insert (i, x)** — заменяет (вставляет) на i-ю позицию элемент x.

**s.remove (x)** — удаляет элемент x, который был обнаружен в списке первым. Если такого элемента не существует, то выводится ошибка Value Error.

**s.clear()** — список очищается.

**s.pop ([i])** — удаляет i-й элемент, по умолчанию удаляет последний элемент списка.

**s.sort ([key=function])** — сортирует элементы списка в соответствии с заданной функцией (по умолчанию сортировка осуществляется по возрастанию). Если в скобках указать `s.sort(reverse=1)`, то сортировка будет осуществляться по убыванию (по умолчанию `reverse=0`).

**s.reverse()** — список разворачивается.

Для преобразования списка в строку можно использовать метод `join()`, который был подробно рассмотрен в теме «Работа со строками».



## Пример

Код:

```
s= ['a','b','c']
d=' '.join(s)
print('—', d)
s= [1,2,3]
for i in range(0, len(s)):
    s [i] =str(s [i])
d=''.join(s)
print('—', d)
```

Результат:

```
— a b c
— 123
```

Во втором варианте необходимо сначала числа преобразовать в строку, а затем применять метод сборки join().

## Упражнение 2.21

Введите в интерпретаторе следующую последовательность команд, представленную на рис. 2.16. Прокомментируйте полученные результаты.

```
File Edit Format Run Options Window Help
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print(a.count(333), a.count(66.25), a.count('x'))
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

Рис. 2.16. Работа с одномерными списками

## 2.3.6. Многомерные списки

Работа с многомерными списками ничем не отличается от обычных списков, только при обращении к ним можно указать еще одно или несколько измерений.

## Пример

Код:

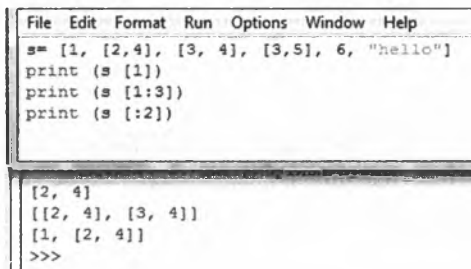
```
s= [1,2,3,'Python',5, [2,'a']]
```

```
s= [1,2,3,'Python',5, [2,'a']]
print(s [1], ' ', s[5] [1])
```

Результат:

2 a

Над многомерными списками также можно осуществлять операции и применять методы, описанные выше. Еще один пример представлен на рис. 2.17.



```
File Edit Format Run Options Window Help
s= [1, [2,4], [3, 4], [3,5], 6, "hello"]
print (s [1])
print (s [1:3])
print (s [:2])

[2, 4]
[[2, 4], [3, 4]]
[1, [2, 4]]
>>>
```

Рис. 2.17. Работа с многомерными списками

### Упражнение 2.22

Список содержит следующие данные двух пользователей: логин, пароль, ФИО и e-mail. Организуйте хранение данных в многомерном списке и выведите только фамилии пользователей.

Сформируем двумерный список, а код программы будем писать последовательно для понимания кода программы.

Запишите следующий код:

```
log= [['Ivanov', '123', 'Иванов Петр Петрович', 'ivanov@
mail.ru'], ['Sidorov', '123', 'Сидоров Иван Иванович',
'sidorov@yandex.ru']]
fiol=log [0] [2]
fio2=log [1] [2]
family1=fiol.split()
family2=fio2.split()
print('Фамилии пользователей: ', family1 [0], ' и ',
family2 [0])
```

В результате получим:

Фамилии пользователей: Иванов и Сидоров

Можно уменьшить размер кода за счет использования цикла for и записи в одну строку, без использования новых объектов:

```
i=0
print('Список фамилий пользователей:')
for i in range(2):
    print(log [i] [2].split() [0])
```

Результат:

Иванов

Сидоров

Как видно, Python позволяет решать задачу разными способами, все зависит от предпочтений разработчика.

### *Задания для самостоятельного выполнения по теме «Последовательности. Списки»*

1. Сформируйте два произвольных списка с количеством элементов списка не менее пяти.
2. Выведите из списка третий элемент.
3. Замените последний элемент из второго списка и выведите его.
4. Проведите конкатенацию списков, выведите получившийся список на экран и оцените длину списка.
5. Выведите часть нового списка так, чтобы в него попали элементы первых двух списков, а также определите длину среза.
6. Добавьте в получившийся список еще три новых элемента и выведите его.

## **2.4. КОРТЕЖИ**

*Кортеж* — это список, элементы которого нельзя менять. Обычно используется в случаях, когда данные постоянны на всем протяжении выполнения программы. К ним можно отнести различные константы, данные о совершенных сделках и т.п. Кортеж заключается в круглые скобки (...) и также может быть многомерным.

**Создание кортежа.** Так же как и для списков, кортеж можно задать в виде литерала.

### **Пример**

```
k = (1, 2, 3, 'Python')
```

```
k = (1, 2, 3, ('Python', 'C#', 'Delphi'), 'plus@mail.ru')
```

Кортеж можно создать с помощью функции `tuple()`.

### **Пример**

Код:

```
print(tuple('tuple'))
```

```
s = (1, 2, 3, 'Python', 5, (2, 'a'))
```

```
print(s)
```

Результат:

```
('t', 'u', 'p', 'l', 'e')
```

```
(1, 2, 3, 'Python', 5, (2, 'a'))
```

**Операции над кортежами. Методы кортежей.** Операции над кортежами и методы кортежей аналогичны средствам работы со строками при условии, что они не изменяют их содержимое. К ним можно отнести конкатенацию кортежей, повторение и ряд других.

### *Задания для самостоятельного выполнения по теме «Кортежи»*

Проанализируйте возможность использования операций над строками и их методов для работы с кортежами. Составьте отчет.

Пример приведен на рис. 2.18.



```
File Edit Format Run Options Window Help
print(tuple('tuple'))
s=(1,2,3,'Python',5, (2,'a'))
print(s)

s1=s
s2=tuple('tuple')
print(s1+s2)
print(s1*2)

Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
('t', 'u', 'p', 'l', 'e')
(1, 2, 3, 'Python', 5, (2, 'a'))
(1, 2, 3, 'Python', 5, (2, 'a'), 't', 'u', 'p', 'l', 'e')
(1, 2, 3, 'Python', 5, (2, 'a'), 1, 2, 3, 'Python', 5, (2, 'a'))
```

Рис. 2.18. Работа с кортежами.

## 2.5. СЛОВАРИ

*Словарь*, или Dictionary — отображение между ключами (keys) и значениями (values), при котором ключу однозначно соответствует значение. В других языках программирования словари могут называться другими именами: хэши, ассоциативные массивы. В словаре не может быть двух одинаковых ключей, тогда как на значения таких ограничений не накладывается [8].

Его синтаксис следующий:

словарь: {ключ1: значение1, ключ2: значение2, ..., ключN: значениеN, }

### 2.5.1. Создание словаря

Словарь чаще всего создается с помощью литерала или функции dict(). Конструкция аналогична конструкции при создании строк, списков или кортежей.

**Пример.** Создание словаря.

Код:

```
d= {1:'Google',2:'Mail', 3:'Yandex', 4:'Yahoo'}
print(d [1])
```

```

d1=dict(g='Google', m='Mail', y='Yandex', yl='Yahoo')
print(d1 ['y'])
d2=dict([(1, 'Google'), (2, 'Mail'), (3, 'Yandex'), (4,
'Yahoo')])
print(d2 [2])
d3=dict.fromkeys(['Google', 'Mail', 'Yandex', 'Yahoo'],
[1,2,3,4])
print(d2 [4])

```

Результат:

Google  
Yandex  
Mail  
Yahoo

Как видно из примера, задать словарь с помощью dict() можно двумя способами, а также это можно сделать посредством метода fromkeys(). Доступ к элементу словаря осуществляется путем указания после имени словаря в квадратных скобках ключа элемента, что аналогично индексу для элемента списка или кортежа.

### Упражнение 2.23

Создайте таблицу квадратов чисел от 1 до 10, пользуясь словарем, выведите значение 5<sup>2</sup>.

Для выполнения данного упражнения лучше воспользоваться генератором словаря.

Введите код, представленный на рис. 2.19, и сравните с полученным результатом.

```

d={}
for i in range(1,11): # создает цикл
    d=dict([(i,i*i)]) # создание словаря
    d.update(d1) # объединение словарей
# будет рассмотрено позже
print(d) # вывод словаря
print(d[5]) # вывод 5^2

```

---

```

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
25
>>>

```

Рис. 2.19. Код создания таблицы квадратов

## 2.5.2. Операции над словарями

При работе со словарями нельзя осуществить сложение словарей, как со списками или кортежами, однако можно объединить словари посредством метода update().

### Пример

Код:

```
d4= {1:'Google', 2:'Mail', 3:'Yandex', 4:'Yahoo'}
```

```
d5= {5:'Rambler', 6:'Instagram', 7:'Facebook', 8:'Vk'}
```

```
d4.update(d5)
```

```
print(d4)
```

**Результат:**

```
{1: 'Google', 2: 'Mail', 3: 'Yandex', 4: 'Yahoo', 5: 'Rambler', 6: 'Instagram', 7: 'Facebook', 8: 'Vk'}
```

Также можно очистить словарь посредством метода `clear()`.

### **Пример**

**Код:**

```
d= {1:'Google', 2:'Mail', 3:'Yandex', 4:'Yahoo'}
```

```
d.clear()
```

```
print(d)
```

**Результат:**

```
{}
```

Для преобразования двумерного списка в словарь можно также использовать `dict()`.

### **Пример**

**Код:**

```
s= [[1,'Google'], [2,'Mail'], [3,'Yandex'], [4,'Yahoo']]
```

```
d=dict(s)
```

```
print(d)
```

**Результат:**

```
{1: 'Google', 2: 'Mail', 3: 'Yandex', 4: 'Yahoo'}
```

Аналогичным образом словарь можно создать из кортежа.

Также можно сделать новый словарь, созданный посредством копирования исходного словаря методом `copy()` из модуля `copy`.

### **Пример**

**Код:**

```
d= {1:'Google', 2:'Mail', 3:'Yandex', 4:'Yahoo'}
```

```
d1=d.copy()
```

```
print(d1)
```

**Результат тот же.**

Можно удалить соответствующий элемент словаря, используя встроенный оператор `del`.

### **Пример**

**Код:**

```
del d [2]
```

```
print(d)
```

**Результат:**

```
{1: 'Google', 3: 'Yandex', 4: 'Yahoo'}
```

При переборе элементов словаря также используется инструкция `for`.

#### Пример

Код:

```
d= {1:'Google', 2:'Mail', 3:'Yandex', 4:'Yahoo'}
for i in d:
    print(i, 'это', d [i])
```

Результат:

1 это Google 2 это Mail 3 это Yandex 4 это Yahoo

#### Упражнение 2.24

Есть два пользователя, имеющие идентификаторы 1 и 2. Данные каждого пользователя содержат информацию о логине, пароле, ФИО и e-mail. Требуется составить программу, которая выводит данные пользователя по идентификатору.

Очевидно, что идентификатор определяет список или кортеж для каждого пользователя. Поэтому нужно связать ключ со списком, т.е. необходимо организовать сложный словарь.

Запишите следующий код и проверьте результат:

```
log= [['Ivanov', '123', 'Иванов Петр Петрович',
'ivanov@mail.ru'], ['Sidorov', '123', 'Сидоров Иван
Иванович', 'sidorov@yandex.ru']]
d= {1: log [0],2: log [1]}
print('Сотрудник с id 1: ', d [1])
print('Сотрудник с id 2: ', d [2])
```

Результат:

Сотрудник с id 1: ['Ivanov', '123', 'Иванов Петр Петрович', 'ivanov@mail.ru']

Сотрудник с id 2: ['Sidorov', '123', 'Сидоров Иван Иванович', 'sidorov@yandex.ru']

#### Упражнение 2.25

Используя результаты предыдущего упражнения, осуществите вывод e-mail пользователя по идентификатору.

### 2.5.3. Методы для работы со словарями

Методы `clear()`, `copy()`, `fromkeys()` и `update()` были рассмотрены в предыдущих параграфах. Приведем другие методы, которые позволяют организовать работу со словарями.

**get (ключ, [default])** – возвращает значение ключа (default (по умолчанию) выдает None).

#### Пример

Код:

```
d= {1:'Иванов', 2:'Петров', 3:'Сидоров', 4:'Жуков'}
```

```
print(d.get(4))
print(d.get(5, 'Такого сотрудника нет'))
```

Результат:

```
Сотрудник с id 1:  ['Ivanov', '123', 'Иванов Петр
Петрович', 'ivanov@mail.ru']
```

```
Сотрудник с id 2:  ['Sidorov', '123', 'Сидоров Иван
Иванович', 'sidorov@yandex.ru']
```

**items()** — возвращает пару «ключ — значение».

**keys()** — возвращает только ключ.

**Пример**

Код:

```
d= {1:'Иванов', 2:'Петров', 3:'Сидоров', 4:'Жуков'}
print(d.items())
print(d.keys)
```

Результат:

```
dict_items([(1, 'Иванов'), (2, 'Петров'), (3,
'Сидоров'), (4, 'Жуков')])
dict_keys([1, 2, 3, 4])
```

**pop (ключ, [default])** — удаляет и возвращает ключ, по умолчанию вызывает исключение.

**Пример**

Код:

```
d= {1:'Иванов', 2:'Петров', 3:'Сидоров', 4:'Жуков'}
print('Удален пользователь ', d.pop(2))
print(d)
print(d.pop(2, 'Такого пользователя нет'))
```

Результат:

```
Удален пользователь  Петров
{1: 'Иванов', 3: 'Сидоров', 4: 'Жуков'}
Такого пользователя нет
```

Если бы не было установлено значение по умолчанию, то интерпретатор выдал бы ошибку:

```
print(d.pop(2))
```

KeyError: 2

**popitem()** — возвращает и удаляет последнюю в списке пару «ключ — значение», в случае отсутствия выдает KeyError.

**Пример**

Код:

```
d= {1:'Иванов', 2:'Петров', 7:'Сидоров', 5:'Жуков'}
print(d.popitem())
print(d)
```



Результат:

```
(5, 'Жуков')
```

```
{1: 'Иванов', 2: 'Петров', 7: 'Сидоров'}
```

**values()** — выводит значения в словаре.

Используя данные предыдущего примера, интерпретатор выведет:

```
dict_values(['Иванов', 'Петров', 'Сидоров', 'Жуков'])
```

**setdefault (ключ, [default])** — возвращает значение ключа в словаре, или None по умолчанию

**Пример**

Код:

```
d= {1:'Иванов', 2:'Петров', 3:'Сидоров', 4:'Жуков'}
```

```
print(d.setdefault(2))
```

Результат:

Петров

*Задания для самостоятельного выполнения по теме «Словари»*

1. Создайте словарь, отражающий количество сотрудников в пяти подразделениях.
2. Определите число сотрудников в произвольном подразделении.
3. В результате реорганизации изменилось количество сотрудников в одном из подразделений, а также было добавлено два дополнительных подразделения. Измените словарь в соответствии с произошедшими преобразованиями.
4. Выведите данные о подразделениях на экран.
5. Можно ли преобразовать словарь в список?

## 2.6. МНОЖЕСТВА

*Множество* — это совокупность неповторяющихся элементов, расположенных в случайном порядке. При этом они могут быть как числовыми, так и символьными. По сути, множество является аналогом множества, используемого в математике. Множество также заключается в фигурные скобки, что иногда приводит к путанице между обозначением множества и словарем. Множество может быть и пустым.

**Пример**

```
{1, 3, 4}
```

```
{'Мир', 'Здравствуй', 'прекрасный', ',',''}
```

### 2.6.1. Создание множества

Множества можно создать посредством литерала (пустое множество с помощью литерала создать нельзя). Также можно использовать инструкцию `set()` или `frozenset()`. В первом случае созда-

ется изменяемое множество, во втором — неизменяемое. Еще один способ заключается в генерации элементов множества, например чисел.

### Пример

Код:

```
s= {1,2,3} # множество через литерал
print(s)
s=set(['1','2','3']) # множество из списка
print(s)
s=set(('1','2','3')) # множество из кортежа
print(s)
s=set(['1','2','Здравствуй', 'мир']) # разный тип данных
print(s)
s=set('Здравствуй, мир') # множество из строки
print(s)
s= {i for i in range(11)} # генератор множества
print(s)
```

Результат:

```
{1, 2, 3}
{'2', '1', '3'}
{'2', '1', '3'}
{'2', 'Здравствуй', '1', 'мир'}
{'д', 'с', 'и', ' ', 'з', 'м', 'т', 'в', ',', 'й',
'р', 'а', 'у'}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Также множество можно использовать для удаления повторяющихся элементов.

### Пример

Код:

```
s= ['Иванов', 'Петров', 'Петров']
print(set(s))
```

Результат:

```
{'Петров', 'Иванов'}
```

### Упражнение 2.26

Создайте множество неупорядоченных чисел от 1 до 10.

## 2.6.2. Операции над множествами

К операциям над множествами относят объединение, пересечение, разность и симметрическую разность (исключающее «или»), а также добавление элементов и их удаление из множества.

Пример использования первых четырех операций представлен на рис. 2.20.

```

# Операции над множествами
s1=set('12345') # 1-е множество
s2=set('6789345') # 2-е множество
print(s1, ' ', s2) # вывод множеств
s=s1|s2; print(s) # объединение множеств
s=s1&s2; print(s) # пересечение множеств
s=s1-s2; print(s) # разность множеств
s=s1^s2; print(s) # симметрическая разность
# (исключающее или) множеств
{'5', '2', '4', '3', '1'} {'5', '7', '4', '3', '6', '8', '9'}
{'5', '7', '2', '4', '3', '1', '6', '8', '9'}
{'3', '5', '4'}
{'2', '1'}
{'1', '6', '9', '7', '2', '8'}
>>>

```

**Рис. 2.20.** Операции над множествами

Добавление элемента осуществляется с помощью метода `add()`, а удаление — посредством `remove()` (при отсутствии элемента выводится ошибка `KeyError`) или `discard()`. Метод `pop()` удаляет первый элемент множества, но поскольку элементы множества не упорядочены, нельзя сказать, какой именно элемент будет удален. Чтобы удалить все элементы множества, применяют метод `clear()`.

**Пример**

Код:

```

s1=set('1234'); print(s1)
s1.add('5'); print(s1)
s1.remove('5'); print(s1)
s1.discard('4'); print(s1)
s1.pop(); print(s1)
s1.clear(); print(s1)

```

Результат:

```

{'3', '4', '2', '1'}
{'5', '1', '3', '4', '2'}
{'1', '3', '4', '2'}
{'1', '3', '2'}
{'3', '2'}
set()

```

Для того чтобы осуществить *проверку наличия элемента* в множестве, используется конструкция `x in set`.

**Пример**

Код:

```

s1=set('1234'); print(s1)

```

```
print('4' in s1)
```

```
print('5' in s1)
```

Результат:

```
True
```

```
False
```

Также можно проверить, что элемент не принадлежит данному множеству: `x not in set`.

*Сравнение множеств* осуществляется с помощью стандартных операций больше (`>`), меньше (`<`) или тождественно равно (`==`).

*Замечание.* Не путайте операцию присваивания (`=`) и операцию сравнения (`==`).

### Пример

Код:

```
s1=set('1234');
```

```
s2=set('123');
```

```
s3=set('3456')
```

```
print(s1>s2, s1<s2, s1==s2)
```

```
print(s1>s3, s1<s3, s1==s3)
```

Результат:

```
True False False
```

```
False False False
```

Для пересекающихся и непересекающихся множеств операция сравнения всегда будет давать `False`. `True` будет в случае, если одно из множеств есть точно такое же множество или оно является подмножеством (надмножеством) другого множества.

### Упражнение 2.27

Сравните множество, составленное из букв вашей фамилии, с множеством, составленным из букв ваших фамилии и имени.

## 2.6.3. Методы для работы с множествами

Существуют аналогичные методы, позволяющие осуществлять объединение (`union()`), пересечение (`intersection()`), разность (`difference()`) и симметрическую разность (`symmetric_difference()`).

### Пример

Код:

```
s1=set('12345') # 1-е множество
```

```
s2=set('6789345') # 2-е множество
```

```
s3=set('Петров')
```

```
s=s1.union(s2, s3); print(s) # объединение множеств
```

```
s=s1.intersection(s2); print(s) # пересечение множеств
s=s1.difference(s2); print(s) # разность множеств
s=s1.symmetric_difference(s2); print(s) # симметрическая
разность (исключающее или) множеств
```

Результат:

```
{'2', '4', 'п', '6', 'e', 'o', '5', '3', 'в', '7',
'8', 'т', '9', '1', 'р'}
{'5', '4', '3'}
{'2', '1'}
{'8', '2', '6', '9', '1', '7'}
```

Как видно из примера, операции можно производить над несколькими множествами.

Также в Python предусмотрены методы сравнения:

- `issubset(x)` — возвращает истину, если `x` является надмножеством множества;
- `issuperset(x)` — возвращает истину, если `x` является подмножеством множества.

**Пример**

Код:

```
s1=set('1234');
s2=set('123');
print(s1.issubset(s2), s1.issuperset(s2))
```

Результат:

```
False True
```

**Упражнение 2.28**

Аналогично упражнению 2.27, только необходимо использовать методы работы со словарями.

Для определения количества элементов множества, так же как и для последовательностей, используется метод `len()`.

**Пример**

Код:

```
s1=set('1234');
s2=set('123');
print(len(s1), ' ', len(s2))
for i in s1:
    print(i)
    for j in range(len(s2)):
        print('s2 [' , j, ']' )
```

Результат (частично):

```
4 3
4
```

```
s2 [0]
s2 [1]
s2 [2]
1
s2 [0] ...
```

**Справка.** Поскольку элементы множества не упорядочены, обращение к элементу множества не имеет смысла, можно лишь осуществить перебор элементов множества, как показано в примере.

Можно создать поверхностную копию множества, воспользовавшись аналогичным методом работы с последовательностями — `copy()`.

### *Задания для самостоятельного выполнения по теме «Множества»*

Первокурсники трех групп по направлениям «Бизнес-информатика», «Экономика» и «Менеджмент» имели следующие баллы ЕГЭ по математике: в первой группе — 79, 86, 88, 95, 94, 86, 89, 92, 78, 78; во второй — 92, 63, 74, 75, 74, 70, 69, 82, 82, 78; в третьей — 55, 56, 58, 67, 76, 65, 67, 67, 63, 60. Напишите код программы, которая позволила бы ответить на следующие вопросы.

1. Каково количество студентов в каждой группе и общее их количество?
2. Каковы уникальные значения баллов для первой; второй и третьей групп; первой и второй; второй и третьей; первой и третьей; и первой, и второй, и третьей групп?
3. Какие баллы имеются во второй группе, которые не содержатся в первой группе?
4. Баллы, которые имеются во второй группе, содержатся ли в первой группе?
5. Какие баллы одинаково содержатся и в первой, и в третьей группе?

## **2.7. РАБОТА С ДАТОЙ И ВРЕМЕНЕМ**

Очень часто разработчикам приходится сталкиваться с необходимостью использовать данные типа «дата и время», например при записи входа пользователей в систему, идентификации платежных транзакций и т.п. Поэтому работа с данными типа дата и время выделена в отдельную тему.

Дата и время имеют особый формат, над ними могут проводиться различные операции.

Для работы с датой и временем в Python разработаны специальные модули `datetime` и `calendar`, которые подробнее будут описаны ниже.

### 2.7.1. Получение текущей даты и времени

Для получения текущей даты используется метод `today()`, который принадлежит классу `date` из модуля `datetime`.

#### Пример

Код:

```
import datetime
print(datetime.date.today())
```

Результат:

```
2018-05-01
```

Как видно из примера, сначала записывается имя модуля, затем после точки — имя класса, а потом после точки — имя метода. Такое представление характерно для всех высокоуровневых языков программирования, в которых реализован объектно-ориентированный подход.

Для получения текущего времени используется метод `now().time()`, который принадлежит классу `datetime` из модуля `datetime`.

#### Пример

Код:

```
import datetime
print(datetime.datetime.now().time())
```

Результат:

```
13:30:42.175385
```

Для получения текущей даты и времени используется метод `now()`, который принадлежит классу `datetime` из модуля `datetime`.

#### Пример

Код:

```
import datetime
print(datetime.datetime.now())
```

Результат:

```
2018-05-01 13:34:30.574810
```

Как видно из примеров, можно получить текущее время, дату, а также дату и время одновременно.

#### Упражнение 2.29

Выведите текущее время, дату, а также дату и время одновременно.

Еще может быть использован специальный модуль `time`.

#### Упражнение 2.30

Используя данные скриншота, приведенного на рис. 2.21, прокомментируйте результаты выполнения представленных методов.

```
File Edit Format Run Options Window Help
import datetime
print (datetime.date.today())
print (datetime.datetime.now().time())
print (datetime.datetime.now())

import time
t=time.time() # количество секунд с 1 января 1970 года 00:00:00
print(t)
print(time.ctime())
time.sleep(2)
print('i sleep 2 sec')
```

**Рис. 2.21.** Использование модуля time

### 2.7.2. Форматирование даты и времени

Как видно из предыдущих примеров, дата и время выводятся в заданном по умолчанию формате. Российский формат (дата, месяц, год) отличается от европейского или американского представлений. Именно поэтому форматированию даты и времени уделяется особое внимание.

Первый вариант — это использование стандартного метода `format()` инструкции `print()`.

#### Пример

Код:

```
import datetime
t_now=datetime.datetime.now()
print(' {}. {}. {} {}: {} '.format(t_now.day, t_now.
month, t_now.year, t_now.hour, t_now.minute))
```

Результат:

1.5.2018 15:5

Здесь же можно рассмотреть метод, преобразующую строку с датой и временем в формат даты и времени.

Синтаксис:

```
strptime(string, format)
```

Формат может содержать следующие значения: `%d` — день в виде числа, `%m` — номер месяца, `%y` — два числа года, `%Y` — четыре числа года, `%H` — час в 24-часовом формате, `%M` — минуты, `%S` — секунды, `%A` — название дня недели, `%B` — название месяца.

#### Пример

Код:

```
from datetime import datetime
t = datetime.strptime('01/05/2018', '%d/%m/%Y')
print(t)
t = datetime.strptime('01/05/2018 15:30',
'%d/%m/%Y%H:%M')
```



```

print(t)
t = datetime.strptime('05-01-2018 15:30', '%m-%d-%Y%H:%M')
print(t)

```

Результат:

```

2018-05-01 00:00:00
2018-05-01 15:30:00
2018-05-01 15:30:00

```

Форматирование можно применить и к текущей дате и времени. Для этого лучше использовать метод `strftime()`, который преобразует дату в строку.

### Пример

Код:

```

import datetime
t_now=datetime.datetime.now()
print(t_now)
t_now=t_now.strftime('%d.%m.%y%H:%M:%S')
print(t_now)

```

Результат:

```

2018-05-01 15:40:12.055683
01.05.18 15:40:12

```

### Упражнение 2.31

Используя предыдущий пример, извлеките текущие дату и время и представьте их в формате чч-мм-сс-дд-мм-гггг.

### Упражнение 2.32

Введите следующий код и проанализируйте назначение символов форматирования `%A`, `%a`, `%B` и `%b`:

```

import datetime
t_now=datetime.datetime.now()
t_now1=t_now.strftime('%A%d%B%y')
print(t_now1)
t_now1=t_now.strftime('%a%d%b%y')
print(t_now1)

```

Более подробную информацию о средствах форматирования можно получить в документации по Python на официальном сайте<sup>1</sup>.

## 2.7.3. Модули `datetime` и `calendar`

Модуль `datetime` является основным модулем для работы с датами и временем. Методы `now()`, `time()`, `today()`, `strftime()`,

<sup>1</sup> URL: <https://docs.python.org/2/library/datetime.html#strftime-and-strptime-behavior>

`strftime()` из данного модуля были рассмотрены в подпараграфах 2.7.1 и 2.7.2.

Вообще модуль `datetime` содержит в себе три класса методов `date`, `time` и `datetime`. Из самих названий классов методов ясно их назначение.

Ниже приведены основные методы из этих классов для работы с датой и временем и некоторые примеры.

**`date (год, месяц, день)`** — выводит дату с тремя параметрами

**Пример**

Код:

```
import datetime
t=datetime.date(2018,5,1)
print(t)
print(datetime.date(2018,5,1).strftime('%m.%y'))
```

Результат:

```
2018-05-01
05.18
```

Данный метод принадлежит к классу `date`.

К классу `time` относится следующий метод.

**`time ([час [, минута [, секунда [, микросекунда]]]])`** — выводит время, причем можно задать формат представления с использованием необязательных параметров.

**Пример**

Код:

```
import datetime
t=datetime.time(); t1=datetime.time(12)
t2=datetime.time(12,1); t3=datetime.time(12,1,5)
t4=datetime.time(12,1,5,23)
print(t1, ' ', t2, ' ', t3, ' ', t4)
```

Результат:

```
12:00:00 12:01:00 12:01:05 12:01:05.000023
```

**`Datetime (год, месяц, день [час [, минута [, секунда [, микро-секунда]]]])`** — выводит дату и время.

**Упражнение 2.33**

Используя предыдущий пример и метод `datetime()`, выведите текущую дату и время, которое указана на ваших часах (телефоне или компьютере).

**`weekday()`** — возвращает день недели от 0 до 6.

**`isoweekday()`** — возвращает день недели от 1 до 7.

**`isocalendar()`** — возвращает дату в формате ISO 8601<sup>1</sup>.

<sup>1</sup> ISO 8601 — международный стандарт, выданный организацией ISO (International Organization for Standardization), который описывает форматы дат и времени и дает рекомендации для его использования в международном контексте (год, неделя, день).

**isoformat()** — возвращает дату в формате YYYY-MM-DDTHH:MM:SS.mmmmmm (см. пример ниже).

### Пример

Код:

```
import datetime
t=datetime.datetime.now()
print(t.weekday(), t.isoweekday())
print(t.isocalendar(), t.isoformat())
```

Результат:

1 2

(2018, 18, 2) 2018-05-01T17:06:00.336769

Модуль `calendar` позволяет формировать собственный календарь. Подробнее о свойствах календаря можно посмотреть в руководстве по модулю<sup>1</sup>.

### Упражнение 2.34

Введите код программы, представленный на рис. 2.22. Запустите программу. В результате у вас в папке с программой будет создан файл `calendar.html`. Откройте его в любом браузере и сравните с результатом, представленным на рис. 2.23.

```
import calendar
a = calendar.LocaleHTMLCalendar(locale='Russian_Russia')
with open('calendar.html', 'w') as g:
    print(a.formatyear(2018, width=4), file=g)
```

Рис. 2.22. Код программы для создания календаря

2018		
Январь	Февраль	Март
ПнВтСрЧтПтСбВс	ПнВтСрЧтПтСбВс	ПнВтСрЧтПтСбВс
1 2 3 4 5 6 7	1 2 3 4	1 2 3 4
8 9 10 11 12 13 14 5 6 7 8 9	10 11 12 13 14 5 6 7 8 9 10 11	6 7 8 9 10 11
15 16 17 18 19 20 21 12 13 14 15 16 17 18 12 13 14 15 16 17 18	19 20 21 22 23 24 25 19 20 21 22 23 24 25 19 20 21 22 23 24 25	29 30 31
26 27 28	26 27 28	29 30 31
Апрель	Май	Июнь
ПнВтСрЧтПтСбВс	ПнВтСрЧтПтСбВс	ПнВтСрЧтПтСбВс
1	1 2 3 4 5 6	1 2 3
2 3 4 5 6 7 8	8 9 10 11 12 13 4 5 6 7 8 9 10	4 5 6 7 8 9 10
9 10 11 12 13 14 15	14 15 16 17 18 19 20 11 12 13 14 15 16 17 18	19 20 11 12 13 14 15 16 17 18
16 17 18 19 20 21 22	21 22 23 24 25 26 27 18 19 20 21 22 23 24	25 26 27 28 29 30
23 24 25 26 27 28 29	28 29 30 31	25 26 27 28 29 30
30		
Июль	Август	Сентябрь
ПнВтСрЧтПтСбВс	ПнВтСрЧтПтСбВс	ПнВтСрЧтПтСбВс
1	1 2 3 4 5	1 2
2 3 4 5 6 7 8	6 7 8 9 10 11 12 3 4 5 6 7 8 9	3 4 5 6 7 8 9
9 10 11 12 13 14 15	13 14 15 16 17 18 19 10 11 12 13 14 15 16	10 11 12 13 14 15 16
16 17 18 19 20 21 22	20 21 22 23 24 25 26 17 18 19 20 21 22 23	17 18 19 20 21 22 23
23 24 25 26 27 28 29	27 28 29 30 31	24 25 26 27 28 29 30
30 31		
Октябрь	Ноябрь	Декабрь
ПнВтСрЧтПтСбВс	ПнВтСрЧтПтСбВс	ПнВтСрЧтПтСбВс
1 2	1 2 3 4	1 2
3 4 5 6 7 8 9	3 4 5 6 7 8 9 10 11 12 13 14 15 16	3 4 5 6 7 8 9 10 11 12 13 14 15 16
10 11 12 13 14 15 16	16 17 18 19 20 21 12 13 14 15 16 17 18 19 20 21 22 23	17 18 19 20 21 22 23
15 16 17 18 19 20 21 12 13 14 15 16 17 18 19 20 21 22 23 24 25	24 25 26 27 28 29 30	24 25 26 27 28 29 30
22 23 24 25 26 27 28 19 20 21 22 23 24 25	26 27 28 29 30	31
29 30 31		

Рис. 2.23. Результат выполнения кода

<sup>1</sup> URL: <https://docs.python.org/3/library/calendar.html>

**Задания для самостоятельного выполнения по теме  
«Работа с датой и временем»**

1. Определите текущую дату, время, а также одновременно дату и время, пользуясь методами модуля `datetime`.
2. Отформатируйте данные, пользуясь методом `format()` и `strftime()`.
3. Задайте дату своего рождения и представьте ее в формате ISO.
4. Используя методы модуля `datetime` для задания даты и времени, составьте программу, которая выводит следующее сообщение: «Экзамен по дисциплине “Компьютерный практикум” будет проходить 26.06.2018 в 209 ауд. Начало экзамена в 10.00».
5. Определите и выведите день недели в числовом и текстовом формате для мероприятия из предыдущего задания.

**Контрольные вопросы и задания**

1. Перечислите основные типы данных, используемых в Python.
2. Как задаются числовые типы данных? Какие операции применимы к ним?
3. Чем отличается список от кортежа?
4. Какие операции можно осуществлять над строками?
5. Что такое множества? Какие методы используются при работе с ними?
6. В чем заключаются особенности создания словаря? Приведите способы создания словарей.
7. Каким способом можно преобразовать строку в список (кортеж) и обратно?
8. Каким образом с помощью метода `split` можно получить список из строки?
9. Опишите классы методов, содержащихся в модуле `datetime`.
10. Зачем нужен модуль `calendar`?

# Глава 3

## ИНСТРУКЦИИ, ФУНКЦИИ, МОДУЛИ В ЯЗЫКЕ PYTHON

### 3.1. ПЕРЕМЕННЫЕ

*Переменная* — это величина, которая в процессе выполнения кода программы может принимать разные значения в соответствии с его логикой и особенностями реализации алгоритмов. Например, общая сумма нескольких сделок определяется последовательным добавлением к изначальной сумме каждой суммы сделки. По мере их добавления общая сумма меняется.

#### 3.1.1. Правила именования переменных

Все переменные на языке Python должны начинаться с буквы или символа нижнего подчеркивания. Если переменная начинается с нижнего подчеркивания, то она используется внутри локального модуля; если два подчеркивания стоят в начале именования переменной, то переменная или уже объект имеют другой смысл, например, при перегрузке операторов (`__call()`), инициализации класса объектов (`__init()`) и т.п. В любом случае, алфавита достаточно, чтобы объявить сколь угодно сложные переменные. В качестве хорошего тона, что также помогает в понимании значения переменных, в их название стоит вкладывать определенный смысл. Например, для описания сотрудников лучше использовать, например, `employee`, для обозначения студентов — `students`, фамилии студентов обозначить как `Students_Surname`, хотя проще создать класс студентов, у которого одним из свойств будет фамилия (`students.surname()`). Об этом речь пойдет в главе 4, посвященной объектно-ориентированному программированию.

Переменные могут принадлежать трем различным пространствам имен:

- 1) глобальные переменные видны всей программе;
- 2) локальные переменные видны только подпрограмме (функции, в которой они определены);
- 3) встраиваемые переменные или идентификаторы (переменные специального назначения), такие как ключевые или зарезервированные слова, например `help`, `print`, `input`, `error` и т.п.

При объявлении переменных лучше избегать использования ключевых слов. Для просмотра списка зарезервированных имен или идентификаторов нужно вызвать функцию `keyword.kwlist` из модуля `keyword`, которая выведет список всех зарезервированных сочетаний.

### Пример

Код:

```
import keyword
print(keyword.kwlist)
```

Результат:

```
['False', 'None', 'True', 'and', 'as', 'assert',
'break', 'class', 'continue', 'def', 'del', 'elif',
'else', 'except', 'finally', 'for', 'from', 'global',
'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not',
'or', 'pass', 'raise', 'return', 'try', 'while', 'with',
'yield']
```

### Упражнение 3.1

Для получения полного списка встроенных идентификаторов введите следующий код:

```
import builtins
print(dir(builtins))
```

Подсчитайте их общее количество.

## 3.1.2. Присваивание значений переменным

Для того чтобы присвоить значение переменной, что ранее было продемонстрировано в примерах, используют оператор присваивания `←=→`, где слева от него стоит переменная, а справа — значение или объект.

*Замечание.* Не путайте оператор присваивания `←=→` с оператором сравнения `←==→`, который обычно используется при проверке условий.

### Пример

Код:

```
a1=1 # присвоение числа
a2='P; y; t; h; o; n' # присвоение строки
a3=a2.split(';') # присвоение объекта(списка)
a4, a5 = 4,5 # присвоение нескольким переменным
a6= [i*iforiinrange(1,6)] # итеративное присвоение
print(a1, ' ', a2, ' ', a3, ' ', a4, ' ', a5, ' ', a6)
```

Результат:

```
1 P; y; t; h; o; n ['P', 'y', 't', 'h', 'o', 'n']
4 5 [1, 4, 9, 16, 25]
```

Практически любая программа содержит в себе одну или несколько переменных.

### Упражнение 3.2

Создайте переменные, содержащие словарь, кортеж, множество размером не менее пяти символов, и выведите их на экран.

### 3.1.3. Динамическая типизация

Основным отличием языка Python от других языков программирования является то, что не нужно заранее объявлять тип переменных, их тип определяет сам «движок», причем переменные могут изменять свой тип по ходу выполнения программы. Иными словами, в нем реализована технология динамической типизации объектов.

**Справка.** В языках, производных от Pascal, для объявления переменных необходимо указать ключевое слово `var` и указать тип данных, в VBA — `Dim`. В случае несоответствия типов переменных, используемых в одном и том же выражении, выводится ошибка `Type Mismatch`. При этом тип переменной не может быть изменен в процессе выполнения программы. Такая строгая типизация реализована во многих языках программирования, например C++, Java, Perl, FORTRAN и ряде других.

Основными свойствами динамической типизации объектов являются следующие: каждый созданный объект (в Python переменная ведет себя как объект и, по сути, является объектом) есть ссылка; тип объекта определяется типом объекта, на который идет ссылка; объект может изменять свой тип при изменении ссылки, которая указывает на другой объект. Меняя ссылку, объект изменяет свой тип.

#### Пример

Код:

```
a='1'; print(type(a))
a=1; print(type(a))
a=1.125; print(type(a))
a= [1,2,3]; print(type(a))
```

Результат:

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'list'>
```

Как видно из примера, в процессе строчного выполнения программы переменная меняется свой тип.

### 3.1.4. Понятие о счетчике ссылок и сборке мусора

Как упоминалось ранее, при присвоении переменной значения другой переменной, поскольку каждая переменная является ссылкой на объект, осуществляется копирование ссылки на исходный объект. Каждый из объектов Python включает в себя счетчик ссылок, и при появлении новой ссылки значение счетчика увеличивается на единицу. Если ссылка на объект перестала существовать, то значение счетчика ссылок уменьшается на единицу. Как показано в предыдущем примере, когда переменной *a* присваивалось новое значение, счетчик на объект увеличивался на единицу, а счетчик предыдущего объекта уменьшался на единицу. В случае если значение счетчика становится равным нулю, Python считает, что объект является неиспользуемым, и память, отведенная для него, освобождается посредством автоматического вызова деструктора.

Такая технология (механизм) называется **garbage collection**, или **сборкой мусора**. Другими словами, если в программе больше нет ссылок на объект, то память, выделенная для хранения информации об объекте, очищается.

#### Пример

Код:

```
1. a='1'  
2. b=a  
3. a=1  
4. a=1.125  
5. a= [1,2,3]
```

Результат:

1. Счетчик увеличивается для '1' (значение 1)
2. Счетчик увеличивается для '1' (значение 2)
3. Счетчик увеличивается для 1 (значение 1) и уменьшается для '1' (значение 1)
4. Счетчик увеличивается для 1.125 (значение 1) и уменьшается для 1 (значение 0, память для хранения 1 освобождается)
5. Счетчик увеличивается для [1,2,3] (значение 1) и уменьшается для 1.125 (значение 0, память для хранения 1.125 освобождается)

Таким образом, хранятся значения только 2 объекта '1' и [1,2,3].

**Справка.** Структура объекта Python имеет два поля: в первом из них хранится информация о типе данных, а второе содержит значение счет-



чика. При этом в процессе выполнения программы Python сам определяет, когда можно удалить объект, т.е. очистить память, и разработчику не нужно следить за необходимостью освобождать память для новых переменных.

### 3.1.5. Проверка и преобразование типов данных. Удаление переменных

Для проверки типа данных ранее мы использовали функцию `type()`. В сочетании с инструкцией `print()` она позволяет вывести тип переменной на экран (см. пример в подпараграфе 3.1.3).

Также существует еще одна функция, позволяющая проверить тип данных: `isinstance` (переменная, тип переменной).

#### Пример

Код:

```
a='1'; print(isinstance(a, int))
a=1; print(isinstance(a, int))
a=1.125; print(isinstance(a, int))
a= [1,2,3]; print(isinstance(a, list))
```

Результат:

```
False
True
False
True
```

Другими словами, данная функция выдает истину, если переменная соответствует указанному в качестве второго аргумента типу, и ложь — в противном случае.

Для преобразования типов данных в одном из предыдущих примеров была использована функция `int()`, которая преобразовывала строку в целое число.

#### Упражнение 3.3

Напишите следующий код программы, запустите его и объясните назначение функций, осуществляющих преобразование типов переменных.

Код:

```
question='На каком курсе вы учитесь? '
a=input(question); print(type(a))
a=int(a); print(type(a))
a=17+a-1; print(type(a))
a=str(a); print(type(a))
a= ['Если вы поступили в 17 лет, то вам сейчас должно
быть около', a, ' лет']
```

```

print(a[0], a[1], a[2])
print(type(a))
a=tuple(a); print(type(a))
a=list(a); print(type(a))
a=set(a); print(type(a))
q1='Введите 2 числа'; print(q1)
a1=float(input('Первое число: '))
a2=float(input('Второе число: '))
print(a1, ' ', type(a1), ' ', a2, ' ', type(a2))
a=int(input('Введите целое число'))
d1=ord(str(a)); print(d1, ' ', type(d1)) # возвращает
значение байта
d2=chr(a); print(d2, ' ', type(d2)) # возвращает символ
Результат:
На каком курсе вы учитесь? 2
<class 'str'>
<class 'int'>
<class 'int'>
<class 'str'>
Если вы поступили в 17 лет, то вам сейчас должно быть
около 18 лет
<class 'list'>
<class 'tuple'>
<class 'list'>
<class 'set'>
Введите 2 числа
Первое число: 3
Второе число: 4
3.0 <class 'float'> 4.0 <class 'float'>
Введите целое число 4
52 <class 'int'>


---


<class 'str'>
>>>

```

Также используется следующие преобразования: `bytes()` — преобразование в байты; `bytearray()` — массив байтов.

### Упражнение 3.4

Составьте программу, которая использует функции `bytes()` и `bytearray()`.

Для удаления переменной используется инструкция `del`. При этом ее можно применять к нескольким переменным.

## Пример

Код:

```
del a
del d1, d2
print(a, d1, d2)
```

Результат:

```
переменные.py", line 47, in<module>
print(a, d1, d2)
NameError: name 'a' is not defined
```

После удаления при обращении к переменной интерпретатор выдаст ошибку, причем остановится сразу на первой переменной (как видно из примера).

### *Задания для самостоятельного выполнения по теме «Переменные»*

1. Создайте переменную `student` и присвойте ей строку, содержащую фамилию, имя и отчество, разделенные «;».
2. Преобразуйте ее в список с помощью метода `split()` и выведите его на экран.
3. Преобразуйте полученный список сначала в кортеж, а потом в множество. Выведите их на экран.
4. Пользователь вводит целое число из командной строки. Создайте переменную, которая является суммой числа, введенного пользователем, и 3. Выведите полученное значение переменной и ее тип.
5. Пользователь вводит вещественное число из командной строки. Создайте переменную, которая является частным числа, введенного пользователем, и 3. Выведите полученное значение переменной и ее тип.

## 3.2. ПРОГРАММА. СВОЙСТВА И ОСОБЕННОСТИ ПОСТРОЕНИЯ

*Программа* — это текст (последовательный код), написанный на одном из языков программирования, который может включать модули, инструкции, функции, операторы и т.п. В Python особое внимание уделяется структуре программы, комментариям, блокам и правилам оформления отступов. Для передачи интерпретатору программы используют три способа.

Первый из них, который уже был использован в предыдущих темах, — это файловый ввод. Иными словами, создается файл с расширением `*.ру` и вызывается через IDLE с помощью команды `Run` (удобнее нажать `F5`) либо двойным нажатием на сам файл в проводнике, если у вас расширение `*.ру` ассоциировано (при установке Python эта ассоциация осуществляется автоматически) с интерпретатором Python.

Второй способ — это непосредственный ввод кода в интерпретатор, который был рассмотрен в параграфе 1.9.

Третий способ реализуется в интерактивном режиме посредством ввода выражений с использованием функций `eval()` и `input()` (рис. 3.1).

```
>>> input()
"2*2"
! "2*2"!
>>> eval("2+2, 3**6")
(4, 729)
>>> 2*4
8
>>> 2**3, 2*3
(8, 6)
>>> |
```

Рис. 3.1. Результат выполнения кода

При этом выражения можно писать через запятую.

### 3.2.1. Структура программы

Python построен на объектах, которые являются каркасом всей структуры программы. Все представления, данные есть объекты, которые имеют свойства или методы. При этом свойства также могут быть объектами, но более низкого уровня. Другими словами, формируется иерархическая структура программы, верхним уровнем которой является модуль, содержащий инструкции. Инструкции включают выражения, которые создают объекты и манипулируют ими.

**Справка.** Программа Python может быть написана в любом текстовом редакторе. Единственным условием для программы является то, что файл должен иметь расширение \*.py.

Атомарным элементом программы является символ в соответствии с принятым алфавитом. Совокупность символов формирует физические строки.

Логическая строка состоит из нескольких физических строк, которые собственно интерпретатор Python и обрабатывает. При этом обработчик может «понимать» только одну логическую строку, за исключением некоторых конструкций, например обработчика ошибок `try ... except`. Иногда логические строки бывают довольно длинными, и для удобства переноса части кода на другую физическую строку используется оператор «\». В данном случае интерпретатор воспринимает следующую строку как продолжение предыдущей записи.

Разбиение может быть внутри строковых литералов, внутри апострофов, кавычек, тройных кавычек или любых скобок [], {}, () (рис. 3.2).

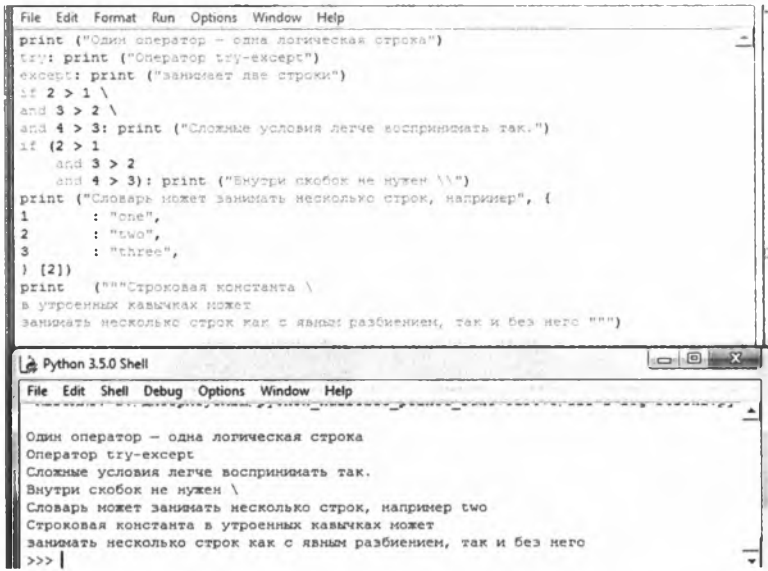


Рис. 3.2. Пример переноса логических строк

Как видно из примера, символ «\» можно использовать в различных случаях. Он играет важную роль при задании отображения данных, которое было рассмотрено ранее. Например, «\n» обозначает перенос строки.

### 3.2.2. Комментарии

*Комментарии* служат для лучшего понимания кода программы. Они записываются после основного текста в одну строку. При этом чтобы Python «понимал», что это комментарий, перед его началом ставится знак #. Все, что идет после этого знака, будет восприниматься как комментарий.

#### Пример

Код:

```
del a # удаляет переменную a
del d1, d2 # удаляет переменные d1, d2
print(a, d1, d2) # печатает переменные a, d1 и d2
```

#### Упражнение 3.5

Составьте программу, которая запрашивает у пользователя число и выводит его на экран. Снабдите код комментариями.

### 3.2.3. Блок. Правила оформления отступов

*Блок* — это часть программы, которая рассматривается как единое целое и направлена на выполнение определенных действий, причем нескольких. В других языках блок называют подпрограммой. В качестве блока в Python могут выступать функции, условные операторы, циклы и набор команд.

Перед блоком команд ставится «:».

#### Пример

Код:

```
a=float(input('Введите первое число: '))
b=float(input('Введите второе число: '))
if a<b: print(a); print(b)
else: print(b); print(a)
```

Результат (при двух прогонах):

```
Введите первое число: 1
Введите второе число: 2
1.0
2.0
>>>
Введите первое число: 2
Введите второе число: 1
1.0
2.0
>>>
```

В данном примере использовался блок, определяемый условным оператором `if`, который подробнее будет рассмотрен позже.

Второй пример демонстрирует, как можно и нужно использовать отступы при организации блоков.

#### Пример

Код:

```
a=float(input('Введите первое число: '))
b=float(input('Введите второе число: '))
if a<b:
    print(a)
    print(b)
else:
    print(b)
    print(a)
```

Результат:

```
Введите первое число: 1
Введите второе число: 2
1.0
```

2.0

>>>

Если отступ не соответствует принятому синтаксису в Python, то интерпретатор выдаст сообщение (рис. 3.3).



**Рис. 3.3.** Сообщение интерпретатора в случае неправильного отступа: «непредвиденный отступ»

Из примера видно, что оформление отступов является одним из важнейших синтаксических процедур при формировании программы. В Python отступ равен четырем пробелам (восьми пробелам на пишущей машинке) или табуляции. Лучший способ создавать правильные отступы — использовать IDLE Python, который автоматически устанавливает требуемый отступ при нажатии клавиши Enter, если курсор установлен после начала блока и символа «:».

Отступы играют роль объединения части текста программы в единый блок, и Python воспринимает его как единое целое. Например, в Delphi для организации блока его необходимо «окружить» ключевыми словами begin... end, в php или JavaScript (JS) используются фигурные скобки при описании тела функции. В VBA условие оканчивается End If, чтобы обработчик «понимал», что данные команды, находящиеся между If и End If, должны выполняться только в этом блоке. Отступы обладают рядом преимуществ, основными из которых являются сокращение кода (не нужно объявлять начало и окончание блока) и наглядность представления. Последнее позволяет упростить восприятие программы. Подавляющее большинство программистов для удобного прочтения кода и идентификации блоков используют искусственное табулирование. В IDLE Python этого делать не нужно — все осуществляется автоматически. Это, по сути, стиль программы.

Подробнее об этом можно прочитать в книге Романа Сузи [8, с. 7]. Негласно принятые правила оформления упростят чтение кода другими программистами.

### Упражнение 3.6

Запишите следующий код программы, используя правила стилей, и проверьте результат ее выполнения. Опишите полученные результаты. Измените отступы и проверьте работоспособность программы.

Код:

```
a=float(input('Введите первое число: '))
b=float(input('Введите второе число: '))
if a<b:
    for i in range(1,6):
        print(a)
        a=a+i
    print(b)
else:
    for i in range(1,6):
        print(b)
b=b*i
print(a)
```

*Задания для самостоятельного выполнения по теме «Программа. Свойства и особенности построения»*

1. Используя книгу Р. Сузи или иные источники, составьте перечень стилей, принятых при формировании кода программы на языке Python.
2. Составьте справочник, который содержит в себе эти правила и выводит их на экран.

## 3.3. ИНСТРУКЦИИ

Инструкция в языке Python — это команда, наименьшая исполняемая часть программы. Их совокупность формирует общую структуру программы. В большинстве языков программирования вместо инструкции принято использовать слово «команда».

В Python несколько типов инструкций: инструкции присваивания, ввод и вывод данных, арифметические операции (были рассмотрены ранее), логические операторы и операторы сравнения, ветвления (`if`, `elif`, `else`), повторное исполнение (циклы с пред- и постусловиями). Этот набор характерен и для большинства современных высокоуровневых языков программирования.

### 3.3.1. Инструкция присваивания

Практически во всех программах используется инструкция присваивания, которая также затрагивалась в той или иной форме для понимания рассматриваемых тем в предыдущих примерах, по-



сколькx без нее практически невозможно продемонстрировать основные возможности языка Python. Присвоение переменной осуществляется с помощью оператора присваивания «=», например, литералов строк или чисел, продемонстрированных в главе 2.

Инструкция присваивания создает ссылку на объект или обновляет уже существующую переменную. Создание переменной обычно называется инициализацией и определяется присвоением литерала определенного типа данных (число, строка, список и т.п.).

### Пример

Код:

```
a='1' # присваивание числа
a=int(a) +2 # обновление переменной
print(a) # вывод числа
```

Результат:

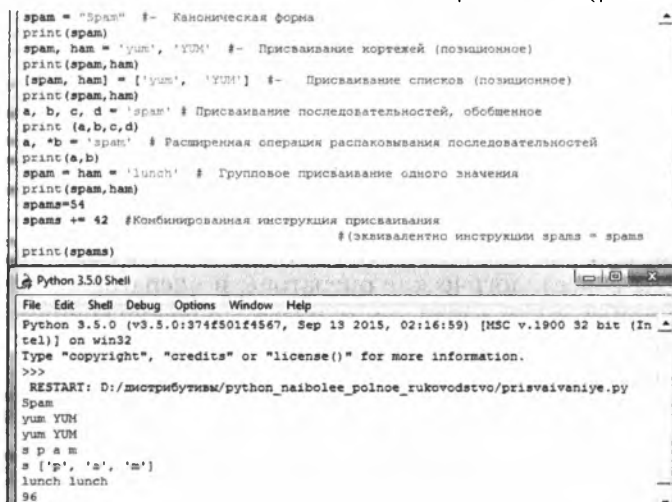
3

Если сразу обновить переменную, то интерпретатор выведет сообщение об ошибке, что переменная не определена. Поэтому сначала нужно присвоить ей какое-то значение, например 0.

Поскольку переменная есть ссылка на объект, то инструкция присваивания формирует или меняет ссылку на объект, эта технология подробно рассмотрена в подпараграфе 3.1.4.

Также затрагивалась инструкция группового присваивания, которая одновременно может присвоить одинаковое или разное значение нескольким переменным.

Комбинированное присваивание позволяет сочетать групповое и присваивание и обновление значений переменных (рис. 3.4).



```
spam = "spam" #- Каноническая форма
print(spam)
spam, ham = 'yum', 'YUM' #- Присваивание кортежей (позиционное)
print(spam,ham)
[spam, ham] = ['yum', 'YUM'] #- Присваивание списков (позиционное)
print(spam,ham)
a, b, c, d = 'spam' # Присваивание последовательностей, обобщенное
print (a,b,c,d)
a, *b = 'spam' # Расширенная операция распаковывания последовательностей
print(a,b)
spam = ham = 'lunch' # Групповое присваивание одного значения
print(spam,ham)
spams=54
spams += 42 #Комбинированная инструкция присваивания
#(эквивалентно инструкции spams = spams
print(spams)
```

Python 3.5.0 Shell

File Edit Shell Debug Options Window Help

Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:16:59) [MSC v.1900 32 bit (Intel)] on win32

Type "copyright", "credits" or "license()" for more information.

>>>

RESTART: D:\дистрибутивм\python\_naibolee\_polnoe\_rukovodstvo/prisvaivaniye.py

spam

yum YUM

yum YUM

s p a m

s ['p', 'a', 'm']

lunch lunch

96

Рис. 3.4. Групповое и комбинированное присваивание

При групповом присваивании числа или строки можно воспользоваться конструкцией

```
a=b=число (строка)
```

### Пример

Код:

```
a=b=2; print(a, ' ', b)
a, b, c=1,2,'Иванов'; print(a, ' ', b, c)
c=d='Студент'; print(c, ' ', d)
c='Аспирант'; print(c, ' ', d)
```

Результат:

```
2 2
1 2 Иванов
Студент Студент
Аспирант Студент
```

Такое комбинированное присвоение является одним из преимуществ Python по сравнению с другими языками программирования.

Можно присваивать последовательности и одновременно проводить операции над ними.

Основные операторы присваивания представлены в табл. 3.1.

Таблица 3.1

### Операторы присваивания

Оператор	Описание
=	Присваивает значение переменной, стоящей справа, переменной, стоящей слева (a=5; >>>5)
+=	Складывает значение левого и правого операнда (переменной) (a=3; b=4; b+=a; >>>7)
-=	Аналогичен предыдущему, только осуществляет операцию вычитания
*=	Включает операцию умножения
/=	Деление
//=	Целочисленное деление
%=	Остаток от деления
**=	Возведение в степень

Как видно из таблицы, данные операторы позволяют совмещать стандартную операцию присваивания с арифметическими операциями, что также уменьшает размер кода.

### Упражнение 3.7

Студенты Иванов, Петров и Сидоров получили одинаковые оценки по дисциплинам «Математика» и «ИТ-инфраструктура организации» — удовлетворительно, хорошо и отлично соответственно, по дисциплине «Базы данных» — хорошо, хорошо и отлично. Сформируйте список оценок для каждого студента и вычислите средний балл для всех и для каждого студента по отдельности. Выведите результаты на экран.

#### 3.3.2. Ввод и вывод данных. Функции `input()` и `print()`

Для обеспечения взаимодействия пользователя и приложения практически в каждом языке программирования организованы ввод и вывод данных. В Python такими инструкциями являются `input()` и `print()`.

Синтаксис:

`input(['текст приглашения'])` — при выполнении запроса программа ждет ввода данных от пользователя с клавиатуры. После нажатия клавиши `Enter` интерпретатор обрабатывает полученные данные. Обычно значение, введенное пользователем, присваивается переменной, которая будет использована в процессе исполнения программы.

#### Пример

Код:

```
input()
input('Введите число')
n=input('Введите текст')
print('Вы ввели: ', n)
```

Результат:

Введите число1

Введите текст Бизнес-информатика

Вы ввели: Бизнес-информатика

#### Упражнение 3.8

Составьте программу, где студент вводит оценки, полученные им по трем изученным дисциплинам. В итоге должен быть выведен на экран средний балл.

**Справка.** В случае работы с числовыми литералами, когда требуется дальнейшая обработка этих чисел, используется преобразование типов данных. Инструкция `input` допускает сочетание с операторами преобразование типов. Например, выражение `n=int(input())` передаст в переменную `n` значение целого типа.

Функция `print()` является мощным инструментом для вывода данных как на экран, так и в файл. Она содержит в себе набор параметров (опций), которые могут создавать представления в различном формате. В предыдущих параграфах для понимания соответствующих тем она уже использовалась. Здесь она будет рассмотрена более подробно.

Синтаксис:

```
print([элемент вывода1,...] [, sep='разделитель'] [, end='чем должен быть закончен вывод элемента'] [, file=sys.stdout])
```

Здесь элемент `вывода1` — собственно то, что должно выводиться. Это может быть как значение переменной, так и литерал.

`sep='разделитель'` — определяет, как будут разделены элементы вывода при их отображении на экране или в файле. По умолчанию — пробел.

`end=''` — определяет, чем будет оканчиваться строка. По умолчанию стоит `'\n'` — перевод каретки на новую строку, если `'!\n'` — то порядок вывода не важен.

`file=sys.stdout` — определяет поток вывода. Поток вывода и его перенаправление определяются методами модуля `sys`, который является стандартным модулем Python, о чем подробнее будет рассказано в параграфе 3.6.

### Пример

Код:

```
a=1; b=2; c=3
print(a, b, c)
print(a, b, c, sep=';')
print(a, b, c, sep=';', end='!')
print(a, b, c, sep=';', end='\n')
print(a, b, c, sep=';', end='!\n')
print(a, b, c, sep=';', end='')
print("{:*+12}".format(3269204.23))
print("{:*=12}".format(-8756703.233))
```

Результат:

```
1 2 3
1;2;3
1;2;3!1;2;3
1;2;3!
1;2;3
+*3269204.23
-8756703.233
```

Ранее рассмотренная функция форматирования `.format()` — удобный способ представления данных.

### Упражнение 3.9

Модифицируйте программу, которая была написана в упражнении 3.8, так, чтобы вывод содержал «Оценка по первой дисциплине: X; оценка по второй дисциплине: X; оценка по третьей дисциплине: X; средний балл X.XX»). Здесь X — формат представления.

### 3.3.3. Операторы сравнения

Операторы сравнения обычно используются при проверке каких-либо условий, и в зависимости от результата выполнения той или иной инструкции, функции, метода и т.п. ключевые операторы сравнения были представлены в табл. 2.2.

### Упражнение 3.10

Не пользуясь табл. 2.2, выведите на экран форму операторов и их назначение примерно так, как показано на рис. 3.5.

\*\*\*\*\*Таблица 2.2 - Операторы сравнения\*\*\*\*\*

Оператор	Описание
<code>==</code>	Тождественно равно
<code>&lt;</code>	Значение истинно, если левый операнд меньше правого
<code>&gt;</code>	Значение истинно, если левый операнд больше правого

>>>

Рис. 3.5. Таблица операторов сравнения

### Упражнение 3.11

Используя навыки, полученные в предыдущем упражнении, составьте таблицу студентов вашей группы, причем данные о них должны храниться в словаре.

### 3.3.4. Логические операторы `and`, `or`, `not`

Логические операторы обычно используются при проверке условий, когда требуется выполнение одного из условий (оператор *или* (`or`)), всех условий (оператор *и* (`and`)) или невыполнение условий (оператор отрицания *нет* (`not`)).

### Пример

Код:

`a=3`

`b=4`

`c=a+b`

```
print(c>3, end=' '); print(c<3)
print(a>=3 and b>=4, end=' ')
print(a>3 and b>=4, end=' ')
print(a>3 or b>=4, end=' ')
print(a>=3 and not b>=4, end=' ')
```

Результат:

True False

True False True False

Еще один пример демонстрирует возможность использования не только числовых, но и строковых переменных, которые широко применяются в разделах математической логики. Объясните верность последнего высказывания (попробуйте объяснить с философской точки зрения).

### Пример

Код:

```
a = {'Преподаватель прав': 'Истина', 'Преподаватель
неправ': 'Ложь'}
print(a); print(a['Преподаватель прав'],',', a['Препо-
даватель неправ'])
print(a.items()); print(a.keys())
print(a['Преподаватель прав'] == 'Истина' and \
a['Преподаватель неправ'] == 'Ложь')
```

Результат:

```
{'Преподаватель прав': 'Истина', 'Преподаватель
неправ': 'Ложь'}
Истина Ложь
dict_items([('Преподаватель прав', 'Истина'), ('Препо-
даватель неправ', 'Ложь')])
dict_keys(['Преподаватель прав', 'Преподаватель
неправ'])
True
```

### 3.3.5. Инструкция ветвления if ... else. Проверка нескольких условий

Инструкция ветвления позволяет в зависимости от условия или условий выполнить определенный алгоритм, который может включать как атомарную операцию присвоения, так и блок программы.

Синтаксис:

If условие:

инструкция 1 (блок программы)

[elif условие:

инструкция n (блок программы n)]

# elif может быть несколько

```
[else:  
инструкция <в противном случае> (блок программы <в  
противном случае>)]
```

Как видно из синтаксиса, можно проверять несколько условий (множественный выбор), при этом в других языках необходимо использовать другие конструкции. Например, в Delphi оператор ветвления такой же — `if ... else`, а для множественного выбора используется `case <переменная> of`.

### Упражнение 3.12

Введите следующий код:

```
a=3; b=4  
if a>2:  
    print('a>2')  
    a=a-1  
    if a>2:  
        print('a снова>2')  
    else:  
        print(a)  
elif b==4:  
    print('b=', b)  
else:  
    print('a is not more than 2')
```

Запустите программу и объясните результат и логику ветвления.

### Упражнение 3.13

Прокомментируйте код и результат выполнения программы, представленный на рис. 3.6.

```
import math  
print ('Программа упорядочивания 2-х чисел')  
print ('Введите 2 числа')  
a=int(input('a='))  
b=int(input('b='))  
if a>b:  
    print ("Числа представлены в порядке возрастания ",b,a,end=" ")  
else:  
    print ("Числа представлены в порядке возрастания ",a,b,end=" ")  
  
Программа упорядочивания 2-х чисел  
Введите 2 числа  
a=2  
b=5  
Числа представлены в порядке возрастания 2 5,  
>>>  
----- RESTART: D:/Рама/Работа с python/upr7_1.py -----  
Программа упорядочивания 2-х чисел  
Введите 2 числа  
a=6  
b=2  
Числа представлены в порядке возрастания 2 6,  
>>>
```

Рис. 3.6. Код и результат выполнения программы

### Упражнение 3.14

Напишите код программы, которая запрашивает у пользователя три числа, а потом выводит их по возрастанию.

### Упражнение 3.15

Напишите программу, которая запрашивает число у пользователя. Если пользователь ввел не число, то выдается сообщение, что введено не число. В противном случае умножьте это число на 2 и напечатайте результат.

*Подсказка.* Воспользуйтесь, например, методом `isdigit()`.

### Упражнение 3.16

Прокомментируйте код и результат выполнения программы, представленный на рис. 3.7.

```
print ('Введите количество баксов вашем кармане')
a = int(input())

if (a/10) %10 == 1:
    print ("У вас на счете ",a," баксов")
elif a % 10 == 1:
    print ("У вас на счете ",a," бакс")
elif 2 <= a % 10 <= 4:
    print ("У вас на счете ",a," бакса")
else:
    print ("У вас на счете ",a," баксов")

Введите количество баксов вашем кармане
38
У вас на счете  38  баксов
>>>
```

**Рис. 3.7.** Код и результат выполнения программы

### Упражнение 3.17

Используя результаты предыдущего упражнения, вставьте правильное склонение при выводе количества лет, введенных пользователем. Например, если пользователь ввел 5, то должно выводиться: «Вам 5 лет», если он ввел 24, то должно выводиться: «Вам 24 года».

### 3.3.6. Инструкция цикла `while`

*Цикл `while`* — это цикл с предусловием, когда цикл повторяется до тех пор, пока это условие не будет нарушено.

Синтаксис:

```
while условие:
    инструкция (блок инструкций)
```

### Упражнение 3.18

Введите следующий код программы и прокомментируйте полученный результат:



```
import time
i=0
while i<10:
    time.sleep(1)
    print(i+1, 'сек.')
i=i+1
print('Прошло ', i, ' сек')
```

### Упражнение 3.19

Найдите минимальное число, которое больше 200 и делится без остатка на 17.

### Упражнение 3.20

Известно, что вложение 100 руб. приносит 10 руб. чистой прибыли. Используя цикл `while`, подсчитайте сколько раз необходимо инвестировать по 100 руб., чтобы получить чистую прибыль в 1000 руб. Проверьте полученный результат простым расчетом.

### Упражнение 3.21

Используя решение предыдущей задачи, включите этап, когда сумму инвестиций задает пользователь, причем ее размер должен варьироваться от 10 до 100 руб. В противном случае должны выводиться сообщение, что введенная сумма не принята, и просьба ввести сумму еще раз.

## 3.3.7. Инструкция цикла `for`. Функция `range`

Цикл `for` — это стандартный цикл, применяемый в большинстве языков программирования, когда задается конечное число циклов.

Синтаксис:

```
for <счетчик> in <область изменения>:
```

инструкция (блок инструкций, называется телом цикла)

Счетчик — это переменная, которая увеличивается на единицу после каждого выполнения тела цикла.

Область изменения может быть представлена в виде строки или списка, а также специальной функцией `range` (начало, конец+1, шаг счетчика).

### Пример

Код:

```
for i in range(1,6,2):
    print(i, end=' ')
print('')
for i in '1 2 3':
    print(i*2, end='')
print('')
s= [1,2,3,4,5]
```

```

for i in s:
    s[i-1] *=s[i-1]
print(s)

```

Результат:

```

1 3 5
11 22 33
[1, 4, 9, 16, 25]
>>>

```

### Упражнение 3.22

Запустите и прокомментируйте код, представленный на рис. 3.8.

```

File Edit Format Run Options Window Help
from random import randint
for i in range(1, 8):
    for j in range(1, 8):
        print (randint(0, 1), |end=" ")
    print ()

0 0 1 1 0 1 0
1 0 1 1 0 1
1 1 1 0 1
1 1 1 1
0 1 0
0 1
1
>>>

```

Рис. 3.8. Код и результат выполнения программы

Что обозначает первая строка, что делает функция randint ()?

Из данного упражнения видно, что Python позволяет делать вложенные циклы.

### Упражнение 3.23

Выведите таблицу умножения чисел от 1 до 9 на 7. При этом должно выводиться как  $1 \times 7 = 7$ ,  $2 \times 7 = 14$  и т.п.

### Упражнение 3.24

Вычислите сумму квадратов чисел от 1 до 9.

Для выполнения этого упражнения необходимо использовать технологию накопления суммы. Введите следующий код:

```

s=0
for i in range(1,10):
    s=s+i*i
print('Сумма квадратов чисел от 1 до 9 равна: ', s)

```

Запустите программу и проверьте правильность вычисления.

### Упражнение 3.25

Пользователь вводит  $x$  и  $n$ . Напишите код, который бы вычислял и выводил на экран следующее выражение:

$$s = 1 + x + x^2 + \dots + x^n.$$

### 3.3.8. Инструкции `break`, `continue`, `pass`

Функция `break` позволяет прервать выполнение цикла. Особенно это важно в случае использования цикла `while`.

#### Пример

Код:

```
print('Для выхода из цикла напишите да')
while True:
    a=input('Напишите здесь: ')
    if a=='да':
        break
print('Для выхода из цикла напишите да')
```

Результат:

```
Для выхода из цикла напишите да
Напишите здесь: 123
Для выхода из цикла напишите да
Напишите здесь: да
>>>
```

Как видно, при вводе «да» программа выходит из цикла.

Инструкция `continue` позволяет пропустить текущий блок и перейти к следующей итерации цикла.

#### Пример

Код:

```
print('Для выхода из цикла напишите да')
while True:
    s=input('Придумайте пароль (не менее 5 символов): ')
    if s=='да':
        break
    if len(s) < 5:
        print('Ненадежный пароль')
    continue
print('Вы ввели пароль достаточной длины')
```

Результат:

```
Для входа из цикла напишите да
Придумайте пароль (не менее 5 символов): 123
Ненадежный пароль
```

Придумайте пароль (не менее 5 символов): 12345

Вы ввели пароль достаточной длины

Придумайте пароль (не менее 5 символов): да

>>>

### Упражнение 3.26

Пользователю предлагается придумывать пароль. Программа запрашивает повторить пароль. Если пароли совпадают, то значение пароля записывается в переменную, в противном случае процедура повторяется заново.

Инструкция `pass` является функцией, которая ничего не выполняет и обычно используется для формирования логики программы, когда впоследствии в тело цикла будет вставлен блок программы.

#### Пример

Код:

```
for i in range(1,11): pass
```

Результат:

>>>

#### Задания для самостоятельного выполнения по теме «Инструкции»

1. Пользователь вводит коэффициенты квадратного уравнения. Программа должна вывести эти корни. В случае отрицательного дискриминанта программа должна вывести сообщение «Корней нет».

2. Имеется квадратное отверстие размером  $a \times b$  см и параллелепипед размером  $c \times d \times e$  см. Требуется проверить, что объект пройдет через отверстие с гарантированным зазором между стенками отверстия и объектом в 1 см. Результат должен быть выведен как «Пройдет» или «Не пройдет».

3. Напишите программу, которая возводит в квадрат введенные пользователем числа и считает их сумму до тех пор, пока не будет введено слово «Выход».

4. Напишите программу, вычисляющую биномиальный коэффициент, который считается как:  $C(m, n) = n! / (m! \cdot (n-m)!)$ , где  $!$  — факториал числа (последовательное перемножение чисел  $1 \cdot 2 \cdot \dots \cdot n$ ).  $n$  и  $m$  вводятся пользователем.

5. Напишите код программы для игрового автомата, который выдает выигрыш в 45% случаев. При этом выдает удвоенную ставку, введенную пользователем. Выведите сумму выигрыша (проигрыша) при сериях из 10, 100 и 1000 игр.

*Рекомендация.* Используйте, например, функцию `randint` из модуля `random`.

## 3.4. ФУНКЦИИ

*Функция* — это операция, которая после выполнения возвращает результат, хотя в Python для некоторых из них это необязательно. С математической точки зрения — это отображение одного множества на другое.

Разделяют функции на три типа.

Чистые функции возвращают однозначный результат по известным входным параметрам.

Встроенные функции Python — стандартные функции, которые также возвращают результат. Например, функции работы с числами из модуля `math`, рассмотренные в соответствующей главе.

### Пример

Код:

```
import math
f= [1,2,3,4,5]
s=set(f)
d=math.pi*min(s)
print(' {:.10.4f} '.format(d))
```

Результат:

```
3.1416
```

```
>>>
```

Наиболее интересен третий тип функций — пользовательские функции, создаваемые разработчиком программы в соответствии с поставленными задачами. О них и пойдет речь в дальнейшем.

### 3.4.1. Создание функции. Инструкция `return`

Для создания функции используется инструкция `def`.

Синтаксис:

```
def имя_функции([переменные]):
    тело функции
    [return возвращаемый результат]
```

Функция может передавать не только числа, но и строку, список и даже другую функцию.

### Пример

Код:

```
def spis():
    print(list('список'))
```

Результат:

```
['с', 'п', 'и', 'с', 'о', 'к']
>>>
```

### Упражнение 3.27

Составьте функцию, которая из вашей фамилии создает множество 1, а из имени — множество 2. Затем выводит на экран объединение этих множеств и возвращает их пересечение в качестве аргумента.

*Рекомендации.* Не забывайте формировать правильные отступы.

#### 3.4.2. Вызов функции

В случае если функция выполняет какое-либо действие, например, печать, и не возвращает ни одного значения, то для ее вывода достаточно указать ее. Если же она возвращает значение, то это значение необходимо передать переменной.

##### Пример

Код:

```
def spis():
    print(list('список'))
    return tuple('кортеж')
```

spis()

z=spis(); print(z)

Результат:

```
['с', 'п', 'и', 'с', 'о', 'к']
['с', 'п', 'и', 'с', 'о', 'к']
('к', 'о', 'р', 'т', 'е', 'ж')
```

### Упражнение 3.28

Составьте функцию, которая вычисляет сумму квадратов чисел от 1 до 10. Выведите ее на экран.

#### 3.4.3. Передача аргументов в функцию. Необязательные аргументы. Функции с переменным числом аргументов

Для передачи аргументов в функцию в скобках указываются переменные, которые должны быть переданы.

##### Пример

Код:

```
def fun1(a, b):
    if a>=b: z=a+b
    else: z=a*b
    return z
z1=fun1(3,4); z2=fun1(4,3)
print(z1,',' , z2)
```

Результат:

12; 7

### Упражнение 3.29

Составьте функцию, которая вычисляет корни квадратного уравнения, если  $a=3$ ,  $b=4$ ,  $z=5$ , и выведите результат ее выполнения.

Функция может содержать необязательные аргументы. Чтобы указать, что данный аргумент необязательный, ему задается значение по умолчанию.

#### Пример

Код:

```
def fio(a, b, c='не указано'):
    print('Фамилия: ', a)
        print('Имя: ', b)
            print('Отчество: ', c)
fio('Петров', 'Иван')
fio('Петров', 'Иван', 'Петрович')
```

Результат:

```
Фамилия: Петров
Имя: Иван
Отчество: не указано
Фамилия: Петров
Имя: Иван
Отчество: Петрович
```

### Упражнение 3.30

Составьте функцию, которая в зависимости от введенных пользователем числа  $n$  и своей фамилии выводит на экран его фамилию  $n$  раз.

Также в зависимости от назначения в функцию можно передать разное количество переменных. Для этого используется символ «\*».

#### Пример

Код:

```
def ret(*s):
    returns*2
print(ret()); print(ret(2)); print(ret(2, 'a', 555))
```

Результат:

```
()
(2, 2)
(2, 'a', 555, 2, 'a', 555)
```

### Упражнение 3.31

Напишите следующий код, представленный на рис. 3.9.

**Справка.** В Python введены специальные лямбда-функции, которые выполняют простейшие операции, о них подробнее – в подпараграфе 3.4.5 «Анонимные функции».

```
File Edit Format Run Options Window Help
1 in range(0,a):
    print('hello')
print ("Программа вывода слова hello n раз")
print ("Введите количество копий слова hello")
a=int(input('n='))
hello(a)
print ('Вывод с помощью функции lambda')
x='hello '
f=lambda x,a:x*a
print(f(x,a))

Программа вывода слова hello n раз
Введите количество копий слова hello
n=5
hello
hello
hello
hello
hello
Вывод с помощью функции lambda
hello hello hello hello hello
```

Рис. 3.9. Код и результат выполнения программы

### 3.4.4. Глобальные и локальные переменные

В Python предусмотрены две области видимости переменных: *локальная* и *глобальная*.

Глобальные переменные видны всей программе, а локальные — только тем блокам (функциям) программы, в котором они определены. При этом если локальные и глобальные переменные названы одним и тем же именем, то после выхода из блока (тела функции), интерпретатор будет «помнить» только глобальную переменную. Чтобы программа видела введенную в теле отдельного блока переменную, после его исполнения перед переменной ставится ключевое слово `global<имя переменной>`.

#### Пример

Код:

```
f= [1,2,3,4,5] # глобальная видимость
def gl():
    for i in f:
        print(f[i-1], '', end='')

gl()
print('Проверка значения f', f)
def gl_loc():
    f= [i for i in range(6,11)] # локальная видимость
    for i in range(0, len(f)):
        print(f[i], '', end='')
```



```

gl_loc()
print('Проверка значения f', f)
def gl_glob():
    global f # глобальная видимость
    f= [i for i in range(12,17)]
        for i in range(0, len(f)):
            print(f[i], '', end='')
gl_glob()
print('Проверка значения f', f)

```

Результат:

```

1 2 3 4 5 Проверка значения f [1, 2, 3, 4, 5]
6 7 8 9 10 Проверка значения f [1, 2, 3, 4, 5]
12 13 14 15 16 Проверка значения f [12, 13, 14, 15,
16]

```

Как видно из примера, после отработки первой функции `gl_loc()` и вывода значения переменной `f` ее величина осталась прежней. При отработке функции `gl_glob()` за счет объявления в ее теле переменной `global` глобальная переменная изменилась.

### Упражнение 3.32

Составьте функцию, которая переводит студентов на следующий курс. Студент (тип «список») содержит в себе поля: ФИО, Направление подготовки, Уровень (Бакалавр, Магистр), Курс. Эти данные вводит пользователь. Далее у пользователя запрашивается: «Перевести студента на следующий курс (Да/Нет)». Если «Да», то к текущему значению курса добавляется 1 при условии, что значение не может быть больше 4 для бакалавра и больше 2 для магистра (для магистров должно выводиться сообщение, что это последний курс магистратуры). Для студентов четвертого курса должна быть осуществлена проверка с запросом: «Перевести студента в магистратуру (Да/Нет)». После отработки функции должно быть выведено, что студент переведен на `x` курс направления `xx` уровня подготовки (бакалавр или магистр). Переменная Студент обновляется и выводится на экран.

### 3.4.5. Анонимные функции

Анонимные функции — это функции, которые могут содержать только одно выражение. Часто их называются лямбда-функции, поскольку их объявление определяется ключевым словом `lambda`.

Синтаксис функции `lambda`.

`lambda` [входные переменные]: выражение.

### Пример

Код:

```
f=lambda s1, s2, s3=' не задано': 'Фамилия: '+s1+'
Имя: '+s2+' Отчество: '+s3
fio=f('Жуков', 'Илья')
print(fio)
fio=f('Жуков', 'Петр', 'Романович')
print(fio)
```

Результат:

Фамилия: Жуков Имя: Илья Отчество: не задано  
Фамилия: Жуков Имя: Петр Отчество: Романович

### Упражнение 3.33

Составьте лямбда-функцию, которая вычисляет налог на доход физических лиц (НДФЛ) в размере 13% от начисленной заработной платы, которую вводит пользователь.

### 3.4.6. Функции-генераторы

Функция-генератор отдает значение с помощью ключевого слова `yield`, при этом обновляя свое значение. По сути функция является генератором итератора, поскольку обновляет свое состояние при получении новых значений.

### Пример

Код:

```
def fib():
    x1, x2=0,1
    while True:
        yield x2
        x1, x2=x2, x1+x2
n=int(input('Введите необходимое количество чисел
Фиббоначи: '))
fib1=fib()
for i in range(n):
    print(next(fib1), '', end='')
```

Результат:

Введите необходимое количество чисел Фиббоначи: 10  
1 1 2 3 5 8 13 21 34 55  
>>>

Как видно из примера, функция каждый раз обновляет свое значение при каждом новом вызове. Функция `next` вызывает объект-генератор. Без него при печати выдается сам созданный объект:

```
<generator object fib at 0x0000000002CF72B0>
```

### Упражнение 3.34

Составьте функцию «счетчик посещений», которая при каждом приглашении «Войти на сайт» добавляет к нему единицу и выводит общее количество входов (посещений).

*Рекомендации.* Приглашение «Войти на сайт» возникает постоянно, если только не введено слово «нет».

### 3.4.7. Декораторы функций. Вложенные функции. Рекурсивные функции

Декораторы функций — это те же функции, внутри которых может исполняться другая функция. По сути, они декорируют («оборачивают») исполняемые функции.

#### Пример

Код:

```
def dec():
    print('Фамилия ', end='')
        f(z)
    print(' успешно зарегистрирована', end='')
def f(z):
    print(z, end='')
z=input('Введите Вашу фамилию: ')
f(z) # вызов функции
print()
dec() # вызов функции декоратора
```

Результат:

Введите Вашу фамилию: Петров

Петров

Фамилия Петров успешно зарегистрирована

Как видно из примера, декоратор «обрамляет» результат выполнения функции посредством дополнительного описания результата выполнения обычной (простой) функции.

Также декоратор можно применить для любой функции, которая будет передана для декорирования. Для того чтобы функцию «обернуть», перед ее описанием необходимо ввести: @имя\_функции.

### Упражнение 3.35

Запишите следующий код программы. Посмотрите результаты выполнения. Прокомментируйте каждую строку кода:

```
def dec(fam):
    def ob():
        global z
```

```

        if a==0:
            z=input('Введите свою фамилию: ')
            return fio[0] +fam() +skl[0] +skl[1]
elif a==1:
            z=input('Введите свое имя: ')
            return fio[1] +fam() +skl[0] +skl[2]
else:
            z=input('Введите свое отчество: ')
            return fio[2] +fam() +skl[0] +skl[2]
return ob
@dec # Объявление декоратора
def fam(): # функция, которая будет декорироваться
    return z
a=int(input('Если вы хотите ввести фамилию, то нажмите
0, Имя - 1, Отчество - 2: '))
fio= ['Фамилия: ', 'Имя: ', 'Отчество: ']
skl= [' успешно ', 'зарегистрирована', ' зарегистрировано']
print(fam())

```

Из предыдущего упражнения видно, что функция может содержать и встроенные (вложенные) функции.

### Упражнение 3.36

Запишите следующий код программы. Посмотрите результаты выполнения. Прокомментируйте каждую строку кода:

```

def def1(z):
    def act(x):
        if z=='1': z1=x*.17
        elif z=='2': z1=x*.035
        elif z=='3': z1=x*0.01
        else:
            print('Вы не выбрали банковскую услугу')
            z1=0
        return z1
    return act
z=input('Выберите банковскую услугу: \n Кредит - 1,
Вклад - 2, Депозит - 3: ')
x=float(input('Введите сумму: '))
k=def1(z)
print('Ожидаемая сумма% - ', ' {:.10.2f} '.format(k(x)),
' рублей')

```

*Рекурсивные функции* — это функции, которые возвращают сами себя, т.е. возвращают эту же функцию. Основная проблема —

определить, когда именно необходимо закончить рекурсию, иначе функция будет исполняться бесконечное число раз.

### Пример

```
Код:  
def fact(n):  
    if n==0:  
        return 1  
    else:  
        return n*fact(n-1)  
print(fact(10))  
Результат:  
3628800  
>>>
```

### Упражнение 3.37

Запишите с помощью рекурсии «На колу мочало, не начать ли сказку сначала».

#### *Задания для самостоятельного выполнения по теме «Функции»*

1. Создайте функцию, которая упорядочивает три числа по убыванию, причем она должна включать функцию, которая упорядочивает два числа по убыванию.

2. Создайте функцию — примитивный счетчик посещений, который запрашивает в цикле (до тех пор, пока не будет набрано слово «да» после запроса «Вы хотите выйти из программы») логин. Если логин новый, то счетчик увеличивается на единицу, если логин уже был, то счетчик посещений также увеличивается на единицу, а число уникальных посетителей остается неизменным. После очередной отработки цикла на экран должно выводиться количество посещений и количество уникальных пользователей.

3. Для преобразования координат в виде поворота используют следующее соотношение:

$$\begin{cases} x' = x \cos \varphi - y \sin \varphi \\ y' = x \sin \varphi + y \cos \varphi \end{cases}$$

Составьте функцию, которая осуществляет такое преобразование. Пользователь вводит координаты точки  $(x_1, x_2)$  и угол поворота системы координат. В результате отработки программы должны выводиться координаты точки в новой системе координат.

4. Составьте функцию, которая создает двумерный список студентов. Каждый студент представляет собой запись, включающую

в себя ФИО, направление подготовки, курс и номер зачетной книжки.

Программа в цикле (до тех пор, пока не будет набрано слово «да» после запроса «Вы хотите закончить ввод данных о студентах») запрашивает у пользователя «Введите данные о студенте», затем проверяет, нет ли такого уже в списке, и если нет, то заносит его в список с соответствующим сообщением «Данные о студенте введены». После окончания ввода студентов программа запрашивает: «Вывести полный список студентов (да/нет)?». Если да, то выводит его в виде таблицы, где заголовками являются поля для каждого студента | Фамилия | Имя | Отчество | Напр. подг. | Курс | № зач. книжки |.

5. Декорируйте вывод списка студентов с помощью функции-декоратора, которая для каждого студента пишет: Фамилия: ... Имя: ..., Отчество: ..., Направление подготовки: ..., Курс: ..., номер зачетной книжки: ...

## 3.5. МОДУЛИ

*Модули* — это программы Python, в которых содержатся объекты, функции, методы, классы, позволяющие программисту не писать свой код, а использовать уже существующий и отработанный (протестированный) код разработчиков и энтузиастов. В понимании Python модулем считается любая программа, которую можно использовать в создаваемом приложении. Разделяют стандартные модули (библиотеки) и пользовательские модули. Каждая программа может получить доступ к содержимому модуля посредством его импорта. Также модуль может быть написан не только на Python, но и с помощью других языков программирования, например C++.

### 3.5.1. Инструкции `import` и `from`

Существует два способа использования содержимого модуля. Первый — подключить модуль полностью с помощью ключевого слова `import`. Ранее такой подход был продемонстрирован при работе с числами, когда требовалось использовать стандартные математические функции.

Синтаксис:

```
import math
```

В этом случае все объекты, инструкции, методы или функции импортированного модуля будут доступны программе.

Импортируемому модулю может быть присвоен псевдоним, который может быть использован для обращения к атрибутам модуля.

Синтаксис:

```
import math as m
```

Можно подключать несколько модулей в отдельных строках или через запятую.

### Пример

Код.

```
import math, datetime
```

или

```
import math
```

```
import datetime
```

Стилистически рекомендуется подключать модули в нескольких строках с целью «читабельности» кода.

Второй способ предполагает вызов конкретного метода или методов (атрибутов) модуля, которые необходимы в программе. Для этого используется инструкция `from (из)`.

Синтаксис:

```
from модуль import атрибут1 [as псевдоним1] [, атрибут2  
[as ...]]
```

или

```
from модуль import *
```

Последний предполагает вызов всех атрибутов модуля, которые включены в переменную `__all__` (атрибуты). Если такой переменной нет, то будут подключены все атрибуты, которые не начинаются с нижнего подчеркивания.

При этом, если подключать модули полностью, то может возникнуть перезапись переменных, имена которых совпадают с именами, определенными в основной программе.

Для вызова атрибута из модуля достаточно указать его имя или псевдоним, а после точки — имя атрибута, сохранив требуемый синтаксис.

### Пример

Код:

```
import math
```

```
r=float(input('Введите радиус окружности: '))
```

```
l=2*math.pi*r
```

```
s=math.pi*r**2
```

```
print('Длина окружности равна: ', ' {:.10.4f}'  
.format(l))
```

```
print('Площадь окружности равна: ', ' {:.10.4f}'  
      '.format(s))
```

Результат:

Введите радиус окружности: 3

Длина окружности равна: 18.8496

Площадь окружности равна: 28.2743

Справка. Если использовать `from math import pi`, то в выражениях `l=2*math.pi*r` и `s=math.pi*r**2` необходимо убрать `math.`, иначе произойдет исключение.

### Упражнение 3.38

Используя атрибуты ранее рассмотренных модулей, вычислите

$$s_n = 1 + x + \frac{x^2}{2!} + \dots + \frac{x^n}{n!}.$$

При этом  $x$  — случайная величина на интервале  $(0;1)$ ,  $n$  задается пользователем.

#### 3.5.2. Создание и использование собственных модулей

Для создания собственного модуля необходимо создать обычный файл, куда записать, например, требуемые функции. Для того чтобы вызвать модуль, его необходимо разместить в той же папке, что и программы, либо в папке `.../lib Python`.

#### Упражнение 3.39

Создайте программу `mymod.py` со следующим кодом:

```
def fun1(x, y): # сумма чисел от x до y  
    s=0  
    for i in range(x, y+1):  
        s=s+i  
    return s  
def fun2(x, y): # произведение чисел от x до y  
    s=1  
    for i in range(x, y+1):  
        s=s*i  
    return s
```

Создайте код основной программы и подключите файл `mymod`.

py:

```
import mymod  
x=int(input('Введите первое число: '))  
y=int(input('Введите второе число: '))
```



```

s1=mymod.fun1(x, y)
s2=mymod.fun2(x, y)
print('Сумма чисел от', x,'до', y,'равна', s1, sep='
')
print('Произведение чисел от', x,'до', y,'равно', s2,
sep=' ')

```

Запустите программу и сравните с представленным результатом:

Введите первое число: 6

Введите второе число: 9

Сумма чисел от 6 до 9 равна 30

Произведение чисел от 6 до 9 равно 3024

>>>

### 3.5.3. Обзор стандартной библиотеки Python

В данном параграфе представлены не все модули Python, а только наиболее известные и часто используемые.

**Модуль os.** Содержит атрибуты для работы с операционной системой.

#### Пример

Код и описание:

```

import os
os.getcwd() # выводит путь к текущему каталогу
os.system('dir *.txt') # выполнить указанную команду ОС
# выводит список текстовых файлов, выведенных командой
os.chdir('d:/python/пособие') # меняет текущий
каталог
dir(os) # выводит список всех атрибутов модуля...
help(os) # выводит руководство по модулю

```

Подробнее об этом модуле и других можно прочитать на официальном сайте Python<sup>1</sup>.

**Модуль shutil** — мощный инструмент для работы с файлами.

Например, `shutil.copyfile()` и `shutil.move()` позволяют копировать файл и перемещать каталог.

**Модуль glob** — на основе заданного шаблона выводит список файлов.

Например, `glob.glob('*.*js')` выведет файлы с расширением `.js` (javascript).

**Модуль sys** — обеспечивает доступ к переменным и функциям интерпретатора. Он имеет три основных класса: `stdin` (стандартный ввод), `stdout` (стандартный вывод), `stderr` (обработчик ошибок).

<sup>1</sup> URL: <https://docs.python.org/3/library/>

Последний из классов позволяет заменить стандартное сообщение Python на пользовательское сообщение.

Например, `sys.stderr.write('Не найден файл')`.

**Модуль `re`.** Модуль `re`, который ранее был рассмотрен при изучении регулярных выражений, предназначен именно для работы с ними.

**Модуль `math`** также был подробно рассмотрен в соответствующем параграфе.

Для работы с сервисами и протоколами Интернет чаще всего используются модули `urllib.request` (работа с получаемыми данными по URL) и `smtplib` (для работы с сообщениями). Пример работы представлен на рис. 3.10.



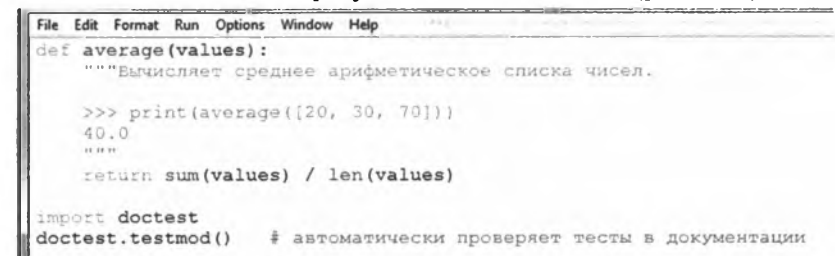
```
File Edit Format Run Options Window Help
from urllib.request import urlopen
import smtplib
for line in urlopen('http://tycho.usno.navy.mil/cgi-bin/timer.pl'):
    if 'EST' in line or 'EDT' in line: # временные зоны
        print(line)
server = smtplib.SMTP('localhost')
server.sendmail('soothsayer@example.org', 'jcaesar@example.org',
... """To: jcaesar@example.org
... From: soothsayer@example.org
...
... Beware the Ides of March.
... """)
server.quit()
```

Рис. 3.10. Пример работы с Интернет [8]

**Модули `datetime`, `time`, `timeit`, `date`** были подробно рассмотрены в параграфе 2.7.

Для сжатия данных и их архивации предусмотрены модули `gzip`, `zlib`, `zipfile`, `tarfile`.

**Модуль `doctest`** позволяет проверять код программы, заложенный в документации. Иными словами, в документацию для понимания алгоритма работы той или иной функции часто записывают тестовое задание с результатом выполнения (рис. 3.11).



```
File Edit Format Run Options Window Help
def average(values):
    """Вычисляет среднее арифметическое списка чисел.

    >>> print(average([20, 30, 70]))
    40.0
    """
    return sum(values) / len(values)

import doctest
doctest.testmod() # автоматически проверяет тесты в документации
```

Рис. 3.11. Пример работы с модулем `doctest` [8]



При этом нужно прописать путь для python.exe c:\users\...\python.exe setup.py install.

Потом можно воспользоваться командами easy\_install или pip <имя скачанного архива> easy\_install -h – информация о команде. Данные команды находятся в папке ...\\Python\\Scripts.

В последних версиях Python pip и easy\_install уже находятся в папке Scripts.

Для работы с базами данных удобнее работать с модулем PYODBC. Для этого скачаем файл последней версии с официального сайта <https://pypi.python.org/pypi/pyodbc/>, соответствующей вашей ОС и ее разрядности, а также версии Python с расширением .whl. Скопируем файл в папку lib python. Запустим программу cmd (вызов консоли Windows) (рис. 3.13).

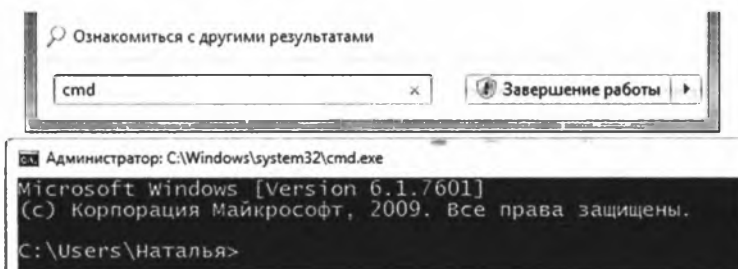


Рис. 3.13. Запуск консоли Windows

Сменим директорию для доступа к файлу в библиотеке Lib (рис. 3.14).

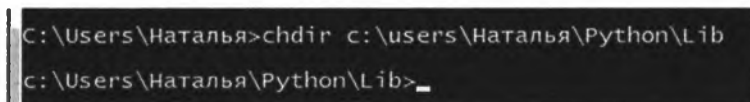


Рис. 3.14. Смена директории для доступа к файлу

Введем команду pip install (рис. 3.15).

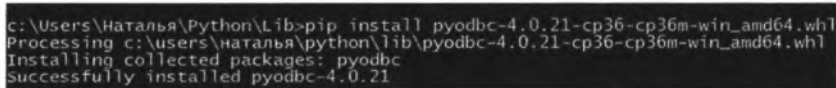


Рис. 3.15. Команда и результат ее выполнения

Аналогично можно установить pymssql для работы с MS SQL Server, хотя модуль pyodbc может быть достаточно.

### *Задания для самостоятельного выполнения по теме «Модули»*

1. Используя методы модуля `ге`, создайте программу, которая при вводе пользователем текста к каждой двузначной цифре, найденной в тексте, добавляет еще один разряд слева, которому присваивает значение 1.
2. Составьте программу, которая бы вычисляла производную функции  $\cos(2\pi x/360) * \sin(2\pi x/360)$ , где  $x$  задается пользователем.
3. Создайте пользовательский модуль, который содержит в себе функции преобразования координат поворота и смещения центра координат. В основной программе предусматриваются ввод пользователем угла поворота, новый центр координат, а также координата точки.
4. Решите предыдущую задачу для трехмерного случая.
5. Составьте модуль, который содержит функции вычисления чистой заработной платы (заработная плата на руки после вычета налога 13%), а также начисления на фонд оплаты труда (ФОТ) (определяется так, чтобы после вычета 30% получалась начисленная заработная плата, от которой вычитается 13% и остается заработная плата на руки). В основной программе пользователь вводит число сотрудников, для каждого вводит начисленную заработную плату. Программа выводит для каждого сотрудника заработную плату на руки, а также начисления на ФОТ и сумму общего размера ФОТ.

### **3.6. ФАЙЛЫ**

Значительная часть данных хранится в базах данных или файлах, из которых они потом извлекаются, добавляются в них или модифицируются. При этом удобнее хранить информацию именно в отдельных файлах и не хранить ее в программе. Сама программа является обработчиком и манипулятором данными. Такое представление характерно для большинства современных информационных систем.

**Методы для работы с текстовыми файлами.** Для работы с текстовыми файлами, как и для других типов файлов, используются такие методы, как запись данных, редактирование, удаление данных, чтение информации из файла.

Эти методы реализуются через функции работы с файлами.

**Функции для работы с файлами.** Первая из таких функций — это функция открытия файла `open()`.

Создается переменная, которой передается файл с заданными правами доступа.

Синтаксис:

```
open('имя файла', 'режим', [encoding])
```

Режим — это, собственно, то, что можно делать с данными из файла. Последний аргумент указывает на кодировку данных (например 'utf-8' или 'cp1251').

Режимы представлены в табл. 3.2.

Таблица 3.2

**Режимы открытия файлов**

Режим	Описание
r	Открывает файл для чтения (задано по умолчанию)
w	Открывает файл для записи, причем содержимое удаляется, а в случае отсутствия такого файла создается новый
x	Открывает файл для записи, причем, если файл отсутствует, то интерпретатор выдаст ошибку
a	Открывает файл для записи, причем содержимое не удаляется, а записываемые данные будут добавлены в конец файла (a — append)
b	Открывает файл в двоичном режиме
+	Открывает файл и для чтения, и для записи
t	Данный режим установлен по умолчанию, т.е. файл открывается в текстовом режиме

Также можно использовать сочетания этих режимов: например, `rt` (задано по умолчанию) означает, что файл будет открыт для чтения в текстовом режиме.

Для чтения из файла используется функция `read([n])`.

Необязательный аргумент `n` (целое) указывает, какое количество символов необходимо прочитать из файла. Если этот аргумент опущен, то в переменную заносится все содержимое файла.

Для записи используется функция `write` (переменная [список операторов]).

Список операторов позволяет записать данные в заданном формате.

Следующая функция, которая является обязательной, — это `close()`. Она закрывает файл и освобождает переменную.

Пример реализации функций представлен на рис. 3.16.

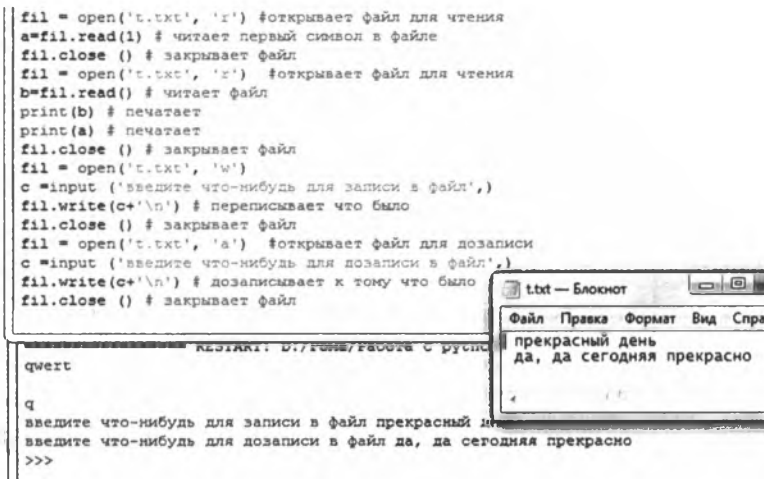


Рис. 3.16. Команда и результат ее выполнения

### Упражнение 3.41

Составьте программу, которая формирует список (квадратную матрицу произвольного порядка). Ввод элементов матрицы может осуществляться с клавиатуры либо из файла (можно файл .txt) (ветвление осуществляется через диалог с пользователем).

Python позволяет работать не только с текстовыми файлами, но и с другими, например файлами Excel, являющимися удобным средством хранения числовой информации и не только.

Для работы с файлами Excel необходимо загрузить соответствующие библиотеки с официального сайта <http://www.python-excel.org/>.

При этом потребуется библиотека **XlsxWriter**. Для ее установки нужно зайти на страницу <https://pypi.python.org/pypi/XlsxWriter>, где из таблицы скачать и затем распаковать соответствующий архив (рис. 3.17).

File	Type	Py Version	Uploaded on	Size
XlsxWriter-0.8.4-py2.py3-none-any.whl (md5)	Python Wheel	2.7	2016-01-16	131KB
XlsxWriter-0.8.4.tar.gz (md5)	Source		2016-01-16	229KB

Рис. 3.17. Архив модулей для работы с Excel

В распакованном архиве выбрать папку `xlsxwriter` и скопировать ее в папку библиотек Python (рис. 3.18).

Теперь в программу можно подключить соответствующий модуль `import xlsxwriter`.

Документацию можно скачать на сайте <http://xlsxwriter.readthedocs.org>.

Пример реализации работы с файлом .xls представлен на рис. 3.19.

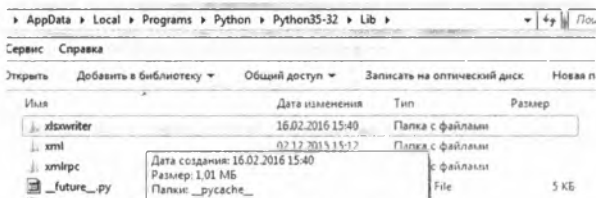


Рис. 3.18. Копирование модуля



Рис. 3.19. Работа с xls-файлами

Для чтения из файла необходим модуль `openpyxl` <https://pypi.python.org/pypi/openpyxl>, который также скачайте с сайта и распакуйте в библиотеку с Python, документация <http://openpyxl.readthedocs.org/en/2.3.3/> (рис. 3.20).

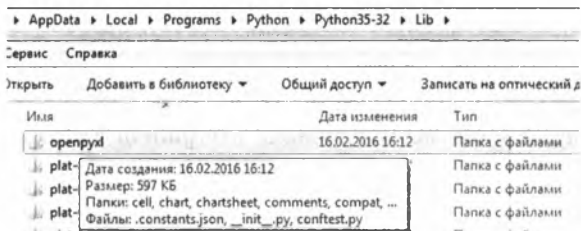


Рис. 3.20. Копирование модуля

Далее необходимо скачать дополнительный модуль `jdcal` <https://pypi.python.org/simple/jdcal/>, распаковываем его в папку `lib` и `et_xmlfile` [https://pypi.python.org/pypi/et\\_xmlfile](https://pypi.python.org/pypi/et_xmlfile). Нужно запустить в каждой папке файл `setup.py`

### Упражнение 3.42

Создайте файл `t.xlsx`, как показано на рис. 3.21.



	A	B	C	D
1	1	2	3	
2	4	5	6	
3	7	8	9	
4				
5				

Рис. 3.21. Работа с xls-файлами (чтение)

Напишите следующий код программы на Python (рис. 3.22) и посмотрите на результат выполнения, дополните программу комментариями.

```

File Edit Format Run Options Window Help
from openpyxl import load_workbook
wb2 = load_workbook('3.xlsx')
print (wb2.get_sheet_names())
sheet_ranges = wb2['Первенец']
s2=''
s5=''
s6=''
for i in range (1,4):
    s=chr(i)
    s1='A'+s
    s3='B'+s
    s4='C'+s
    print(sheet_ranges[s1].value)
    s2=s2+str(sheet_ranges[s1].value)
    s5=s5+str(sheet_ranges[s3].value)
    s6=s6+str(sheet_ranges[s4].value)
print(list(s2))
print(list(s5))
print(list(s6))
s7=list(s2)+list(s5)+list(s6)
print(s7)
ss=[int(s2[0]),int(s2[1]),int(s2[2])],[int(s5[0]),int(s5[1]),int(s5[2])],[int(s
print ("Введенный список", ss)
def dterm():
    d1 = int(s2[0])*int(s5[1])*int(s6[2])
    d2 = int(s2[2])*int(s5[0])*int(s6[1])
    d3 = int(s2[1])*int(s5[2])*int(s6[0])
    d4 = int(s2[2])*int(s5[1])*int(s6[0])
    d5 = int(s2[0])*int(s5[2])*int(s6[1])
    d6 = int(s2[1])*int(s5[0])*int(s6[2])
    d = d1+d2-d3-d4-d5-d6
    print ("Определитель 3х3 равен ",d)
    return d
dterm()

```

```

Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
Python 3.5.0 (v3.5.0:374e501f4567, Sep 13 2015, 02:16
tel]) on win32
Type "copyright", "credits" or "license()" for more i
>>>
===== RESTART: D:\Раба\Раба с python\upr12_
['Первенец']
1
4
7
['1', '4', '7']
['2', '5', '8']
['3', '6', '9']
['1', '4', '7', '2', '5', '8', '3', '6', '9']
Введенный список [[1, 4, 7], [2, 5, 8], [3, 6, 9]]
Определитель 3х3 равен 0
>>>

```

Рис. 3.22. Код программы и его реализация

Другой вариант создания массива представлен на рис. 3.23.

Как видно из рисунка, результат такой же.

**Перенаправление ввода/вывода. Сохранение объектов в файл.**

Для перенаправления ввода/вывода используется, как и ранее, функция print(), которая может перенаправить поток вывода не на экран, а в файл. При этом рекомендуется подключить модуль sys, который и отвечает за перенаправление.

Соответствующий синтаксис представлен для удобства повторно:

```

print([элемент вывода1,...] [, sep='разделитель']
[, end='чем должен быть закончен вывод элемента'] [,
file=sys.stdout)

```

sys.stdout — переменная модуля sys, в которой хранится запись, куда, собственно, будет направлен поток вывода.

```

File Edit Format Run Options Window Help
from openpyxl import load_workbook
wb2 = load_workbook('t.xlsx')
print (wb2.get_sheet_names())
sheet_ranges = wb2['Первенец']
s2=''
s5=''
s6=''
for i in range (1,4): # читает данные из файла
    s=str(i)
    s1='A'+s
    s3='B'+s
    s4='C'+s
    s2=s2+str(sheet_ranges[s1].value)
    s5=s5+str(sheet_ranges[s3].value)
    s6=s6+str(sheet_ranges[s4].value)
mass33 =list(zip(map(float,s2),map(float,s5),map(float,s6)))
# map(float,...) переводит элементы списка в вещественный тип
# zip(a,b,c) создает двумерный массив zip объект
# list() создает двумерный массив (последовательность списка и кортежа)
print ('mass33',mass33)
def dterm3x3(): # вычисляет определитель
    d1 = (mass33[0][0]*(mass33[1][1]*(mass33[2][2]))[2]
    d2 = (mass33[0][2]*(mass33[1][1]*[0]*(mass33[2][1]))[1]
    d3 = (mass33[0][1]*(mass33[1][2]*(mass33[2][0]))[0]
    d4 = (mass33[0][2]*(mass33[1][1]*[1]*(mass33[2][0]))[0]
    d5 = (mass33[0][0]*(mass33[1][2]*(mass33[2][1]))[1]
    d6 = (mass33[0][1]*(mass33[1][0]*(mass33[2][2]))[2]
    d = d1+d2+d3-d4-d5-d6
    print('Определитель 3x3 равен ',d)
    return d
dterm3x3()
|
Ln: 32 Col: 0
-----
RESTART: D:/Формы/Работа с русским/upriz_xls_open_1.py
['Первенец']
mass33 [(1.0, 2.0, 3.0), (4.0, 5.0, 6.0), (7.0, 8.0, 9.0)]
Определитель 3x3 равен 0.0
>>>

```

Рис. 3.23. Код программы и его реализация

## Пример

Код:

```

import sys
dir1='D:/.../упражнения/'
file='output.txt'
old_stdout=sys.stdout # переменная, отвечающая
за вывод(по умолчанию на экран)
sys.stdout=open(dir1+file,'w') # перенаправляет вывод
в файл
print('Вывод данных в файл')
sys.stdout=old_stdout # перенаправление потока из файла
на экран(консоль)
print('Вывод данных на экран')

```

Результат:

Вывод данных на экран

>>>

Теперь необходимо зайти в папку, где был перезаписан (создан) файл `output.txt`, и открыть его. В результате там будет записано: «Вывод данных в файл».

Для сохранения сложных объектов используется модуль `pickle`, который преобразует сложные объекты в поток байтов и обратно (сериализация и десериализация).

В данном модуле реализованы следующие функции:

- `dump()` — записывает объект в файл;
- `load()` — загружает объект из файла.

### Упражнение 3.43

Создайте программу, которая сериализует три записи о студентах, включающие ФИО, дата рождения, направление подготовки, курс, баллы по ЕГЭ, в файл, а затем выводит их в обычный файл и на экран.

#### *Задания для самостоятельного выполнения по теме «Файлы»*

1. Составьте программу, которая вводит данные о студенте в файл. Программа должна поддерживать ввод информации о нескольких студентах. Окончание цикла ввода должно быть организовано через диалог с пользователем.
2. Организуйте вывод из файла, созданного в задании 1, данных о студенте. При этом должен быть организован вывод как отдельного студента, например, по номеру зачетной книжки или ФИО, так и полного списка.
3. Создайте программу, которая выводит в файл или на экран элементы матрицы, а также считает определитель (ветвление осуществляется через диалог с пользователем).
4. Создайте файл Excel, который решает систему трех линейных уравнений методом Крамера. Создайте программу, через которую пользователь вводит коэффициенты уравнения и свободные члены. Эти параметры записываются в созданный файл Excel, а программа выводит решение этой системы на экран.
5. Модифицируйте предыдущую программу, чтобы результаты выводились в текстовый файл и из него считывались данные и выводились на экран.

## 3.7. ИСКЛЮЧЕНИЯ

*Исключения* — это особый тип данных в Python, и они создаются на события, возникающие в процессе исполнения программы, что приводит к остановке ее выполнения. Такими исключениями могут

быть, например, деление на ноль или строку, отсутствие переменной, к которой идет обращение, и т.п. Другими словами, это сообщения в ответ на ошибки, возникающие в процессе выполнения программы.

### 3.7.1. Основные исключения

Основные исключения, предусмотренные в Python, представлены ниже.

**Exception** — базовое исключение, на котором строятся остальные исключения.

**Attribute Error** — возникает при невозможности присвоить значение или создать ссылку на атрибут.

**IO Error** — ошибка ввода/вывода (Input/Output)

**Import Error** — связана с операцией импортирования, например, не найден модуль.

**Index Error** — ошибка выхода индекса последовательности за пределы диапазона (Out of range).

**Key Error** — ошибка при отсутствии ключа в словаре.

**Name Error** — не найдено запрашиваемое имя (например, variable 'i' is not defined).

**Syntax Error** — ошибка синтаксиса.

**Value Error** — тип переменной правильный, но передаваемое значение — нет.

**Type Error** — возникает, когда операция или функция применяется к объекту несоответствующего типа.

**Zero Division Error** — ошибка деления на ноль.

#### Пример

Код:

```
import math
a='Строка'
z=math.sin(a)
print(z)
```

Результат:

```
Traceback(most recent call last):
```

```
File "D:/... /упражнения/3.7 исключения.py", line 3, in
<module>
```

```
z=math.sin(a)
```

```
TypeError: must be real number, not str
```

```
>>>
```

#### Упражнение 3.44

Создайте программу, которая бы вызывала исключение Index Error и Key Error.

### 3.7.2. Обработка исключений. Инструкция try ... except

*Обработка исключений* — это замена стандартного вывода ошибки своим сообщением так, чтобы программа не «вылетала» при появлении исключения.

Традиционным способом обработки исключений является инструкция try ... except:

```
try:
<что должно выполняться>
except <имя исключения> [as <псевдоним>]:
<что должно выполняться при возникновении ошибки>
[except <имя исключения> [as <псевдоним>]:
<что должно выполняться при возникновении ошибки> ...]
# except может быть несколько
[else:
<что должно выполняться, если исключение не произошло>]
[finally:
<выполняется в любом случае>]
```

#### Упражнение 3.45

Заключите предыдущий пример в обработчик исключений в соответствии с кодом, представленным ниже:

```
import math
a='Строка'
try:
    z=math.sin(a)
    print(z)
except TypeError:
    print('Вычислить синус от строки невозможно')
finally:
    print('В любом случае этот текст будет выведен
на экран ')
```

Запустите программу и прокомментируйте полученный результат.

Если неизвестно, какая ошибка может быть выведена, то после except необходимо указать Exception. Иными словами, обработка будет осуществляться при любом типе исключения.

#### Пример

```
Traceback(most recent call last):
  File "D:/... /упражнения/3.7 исключения.py", line 3,
in <module>
  z=math.sin(a)
TypeError: must be real number, not str
```

### 3.7.3. Получение информации об исключении. Создание новых исключений

Информация об исключении выдается при возникновении ошибки интерпретатором. Также можно получить информацию о типе ошибки непосредственно в обработчике исключений посредством использования псевдонима.

#### Упражнение 3.46

Напишите следующий код и введите сначала текст, а затем 0. Проанализируйте полученные исключения:

```
try:
    z=int(input('Введите число: '))
    z=3/z
except Exception as err:
    print('Возникло исключение ', err)
```

Для создания собственного исключения обычно создают класс исключений, т.е. используют средства объектно-ориентированного программирования, которые будут рассмотрены в следующей главе.

При этом используется инструкция raise() для создания собственного исключения.

#### Пример

Код:

```
z=input('Введите целое число больше 0: ')
try:
    z=int(z)
    if z<0:
        raise MError('Вы ввели число меньше 0')
except ValueError:
    print('Вы ввели не число')
except MError:
    print(MError.txt)
```

Результат:

Введите целое число больше 0: -2

Вы ввели число меньше 0

>>>

Если бы была введена строка, то результат был бы следующим:

Введите целое число больше 0: wer

Вы ввели не число

>>>

#### Задания для самостоятельного выполнения по теме «Исключения»

1. Необходимо решить квадратное уравнение. Коэффициенты уравнения вводит пользователь. Предусмотрите возможные ошибки при вводе

данных и при решении уравнения (дискриминант  $<0$ ), используя инструменты обработки исключений.

2. Решите аналогичную задачу для решения системы линейных уравнений методом Крамера (с помощью определителей) (два или три уравнения определяются пользователем).
3. Необходимо ввести данные о студенте: ФИО, направление подготовки, дату рождения в формате date, номер зачетной книжки. Предусмотрите возможные ошибки при вводе данных с помощью инструментов обработки исключений.
4. Пользователь вводит свою электронную почту в формате name@mail.ru. Почта должна относиться к почтовому серверу yandex.ru, gmail.com, mail.ru. Создайте программу, которая записывает данные об электронной почте в файл, предусмотрев обработку исключений при неправильном вводе почтового сервера.
5. Модифицируйте предыдущую программу, где настройте обработчик так, что имя почты должно содержать только буквы и цифры и состоять не менее чем из шести символов.

### *Контрольные вопросы и задания*

1. Каковы основные особенности структуры программы на Python?
2. Опишите правила именования переменных, а также объясните отличия объявления переменных в Python от других языков программирования.
3. Что такое инструкции в Python?
4. Опишите особенности использования функций print() и input().
5. Каков синтаксис организации ветвления алгоритма программы?
6. Как организуются циклы в Python? Перечислите и опишите основные способы.
7. Как создать пользовательскую функцию и вызвать ее в теле программы?
8. Что такое модули? Перечислите основные модули стандартной библиотеки Python.
9. Как организовать работу с файлами?
10. Что такое исключения? Каковы способы их обработки?

## Глава 4

# ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ В ЯЗЫКЕ PYTHON

Современные языки программирования не обходятся без использования объектно-ориентированного подхода. Из содержания предыдущих глав можно сделать вывод, что Python является объектно-ориентированным языком. Сама его структура представлена в виде объектов, которые имеют свои атрибуты, к ним могут применяться характерные только для них методы. Другими словами, объекты — это не что иное, как сущности заданной предметной области, и именно с ними работает Python.

### 4.1. БАЗОВЫЕ ПРИНЦИПЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Представим следующие понятия, которые широко известны программистам, использующим объектно-ориентированный подход (ООП).

*Абстракция* — придание объекту характеристик (в объектно-ориентированном программировании), свойственных только ему, отличных от других объектов, формируя тем самым границы его концептуального представления.

*Абстракция данных* — разделение несущественных деталей реализации подпрограммы и характеристик, которые являются существенными при корректном их использовании.

*Объект* — сущность, которая принадлежит некоторой области исследования и имеет определенный набор свойств и методов, которые позволяют манипулировать объектом: изменять его, создавать новые свойства, создавать новые объекты или удалять их.

Также основными базовыми принципами ООП являются инкапсуляция, наследование и полиморфизм.

#### 4.1.1. Инкапсуляция

*Инкапсуляция* — это одно из важнейших свойств языков программирования, которое позволяет взаимодействовать с абстракциями (объектами) посредством специального внешнего представления (интерфейса). Причем нет необходимости знать, как объект создан (что у него внутри). Интерфейс представляет набор пуб-



личных (public) свойств и методов, к которым пользователь может обращаться. При этом ему предоставляется спецификация, т.е. набор доступных атрибутов объекта. Сущность объекта и его методы защищены от взаимодействия с пользователем посредством специального (private) объявления сущности при создании объекта.

#### **Упражнение 4.1**

Необходимо описать сущность объекта «студент». Какие свойства и методы не должны быть доступны для изменения, а какие могут изменяться извне?

#### **4.1.2. Наследование**

*Наследование* является принципом, который позволяет создать новый набор объектов (новый класс, абстракция, сущность) на базе существующей абстракции (родительская сущность, базовый класс). Объекты такой абстракции называют наследниками или потомками. Наряду с доставшимися им от родителей свойствами они могут свои специфические свойства. При этом нет необходимости описывать свойства родительского класса, в новом классе достаточно указать, что наследник является потомком класса более высокого уровня. Такое наследование называется простым. При этом иерархию можно устанавливать сколь угодно большого уровня сложности. В большинстве своем используют двухуровневую вложенность.

Встречается понятие абстрактного класса, на основе которого не имеет смысла создавать объекты, но можно создавать классы-потомки.

#### **Пример**

Можно создать абстрактный класс «сотрудник университета», который имеет свойства ФИО, год рождения и т.п.

От него можно создать классы-потомки: научные работники, профессорско-преподавательский состав, административные работники и т.п.

#### **Упражнение 4.2**

Определите абстрактный класс и классы-потомки для организации, занимающейся поставками сантехники.

#### **4.1.3. Полиморфизм**

*Полиморфизм* — это принцип ООП, подразумевающий, что объекты, которые имеют одинаковые атрибуты (спецификацию), могут быть реализованы по-разному.

### **Пример**

Имеется класс геометрических фигур, которые могут быть прямоугольниками, эллипсами и т.п. При этом методы, которые используются для преобразования этих фигур, разные. Каждое преобразование требует различного представления и вполне конкретного объема данных. Например, метод масштабирования для многоугольника и окружности одинаков, требует задания коэффициента масштаба. Однако чтобы осуществить такое преобразование для прямоугольника, требуется знать координаты двух диагональных углов либо координаты одного угла и длины сторон, их угол поворота относительно осей координат, а для окружности достаточно знать ее центр и радиус.

### **Упражнение 4.3**

Приведите примеры классов объектов, элементы которых могут иметь разную реализацию.

#### *Задания для самостоятельного выполнения по теме «Базовые принципы ООП»*

1. Пусть имеется поток передачи данных, который определяется как базовый. Основываясь на принципе полиморфизма, определите объекты данного класса, которые могут быть в него включены.
2. Пусть имеется предметная область «склад». Перечислите возможные классы (сущности) данной предметной области (не менее трех).
3. Используя результаты, полученные в предыдущем задании, выделите классы, которые могут иметь доступные и недоступные атрибуты. Опишите их интерфейсы.
4. Основываясь на предыдущем задании, сформируйте классы-потомки (не менее трех).
5. Дополните интерфейсы классов из предыдущего задания новыми свойствами и методами (не менее трех).

## **4.2. КЛАССЫ В ЯЗЫКЕ PYTHON**

Классы на языке Python – это абстрактные классы, которые описывают сущности предметной области, содержат атрибуты (свойства), методы и предназначены для создания экземпляров (объектов) класса. Как подчеркивалось ранее, в Python любое значение (данные) – это объект. Списки, числа, кортежи и т.п. являются все теми же объектами, а переменные, которым присваиваются объекты, являются ссылками на них. Создание классов и присвоение их определенным переменным в ООП считается созданием экземпляра класса, о котором речь пойдет ниже.

Класс в Python — это сложная структура, которая может содержать в себе как атрибуты, так и процедуры и функции, которые могут быть применены к представителям класса — объектам. По сути, класс — тоже объект, который выступает в качестве абстрактного (абстрактная сущность).

### 4.2.1. Инструкция `class`

Для создания класса используется инструкция `class`.

Синтаксис:

```
class имя_класса:  
    переменная=значение
```

```
        def имя_метода(self, [список передаваемых  
переменных]):  
            self.переменная=значение  
            ...
```

Как видно из синтаксиса, класс содержит свойства, определяемые внутренними переменными и функциями (методами), которые могут быть использованы для манипуляций с экземплярами класса.

#### Упражнение 4.4

Напишите следующий код программы. Запустите ее и прокомментируйте результат.

```
class имя_класса:  
    переменная1=3  
        def имя_метода(self, переменная2,  
переменная3=переменная1):  
            self.переменная4=переменная2*переменная3  
            print(self.переменная4)  
obj1=имя_класса()  
obj1.имя_метода(2)
```

Для создания класса можно использовать конструктор класса, при обращении которого создается экземпляр класса с уже заданными свойствами. Конструктор — это та же функция, имя которой `__init__` (self, [список передаваемых аргументов]) обрамляется двойным нижним подчеркиванием. Отличие заключается в форме создания экземпляра класса, речь о котором пойдет далее.

#### Упражнение 4.5

Напишите следующий код программы. Запустите ее и прокомментируйте результат и укажите отличия от предыдущего упражнения:

```
class имя_класса2:
    def __init__(self, переменная):
        self.переменная1=переменная
obj2=имя_класса2(4)
print(obj2.переменная1)
```

#### 4.2.2. Создание экземпляра класса

Как видно из упражнений 4.4 и 4.5, для создания экземпляра класса достаточно присвоить переменной имя класса. В результате будет создан объект, который обладает свойствами (атрибутами) класса (абстракции), а также к нему могут быть применены соответствующие методы класса.

##### Упражнение 4.6

Создайте класс «студент», имеющий один атрибут «курс». Создайте два экземпляра класса 2-го курса, используя первый способ. Для выполнения упражнения напишите следующий код. Запустите его и сравните с результатом, представленным ниже.

Код:

```
class student:
    kurs=2
student1=student() #создание экземпляра класса
student2=student() #создание экземпляра класса
print('Студент 1 учится на', student1.kurs, '-м курсе')
print('Студент 2 учится на', student2.kurs, '-м курсе')
```

Результат:

```
Студент 1 учится на 2-м курсе
Студент 2 учится на 2-м курсе
```

В данном упражнении были созданы два объекта (экземпляры класса) — «студенты», которые принадлежат базовому классу — student и имеют свойство (атрибут), равный 2, т.е. соответствующий номеру курса.

##### Упражнение 4.7

Создайте, как и в предыдущем примере, двух студентов, только учащихся на втором и третьем курсе.

**Справка.** Для выполнения этого упражнения необходимо воспользоваться конструктором класса (рассмотрен в подпараграфе 4.2.1), который автоматически создает ему атрибуты.

Код:

```
class student:
    def __init__(self, kurs):
        self.kurs=kurs
```

```
student1=student(2) #создание экземпляра класса
student2=student(3) #создание экземпляра класса
print('Студент 1 учится на', student1.kurs, '-м курсе')
print('Студент 2 учится на', student2.kurs, '-м курсе')
```

Результат:

Студент 1 учится на 2-м курсе

Студент 2 учится на 3-м курсе

Как и в предыдущем упражнении, были созданы два студента, причем их свойства были определены в процессе их создания за счет использования конструктора класса `__init__()`. Как видно, это гораздо удобнее — не обращаясь к методам класса, создавать экземпляры класса с уже заданным набором свойств.

### 4.2.3. Атрибуты класса и экземпляра класса.

#### Закрытые атрибуты

По сути, атрибуты класса — это не что иное как свойства объектов, которыми будут обладать вновь созданные экземпляры класса.

Для задания свойств объекта достаточно присвоить переменной, находящейся внутри класса, заданное значение или передать его через конструктор класса. Также атрибутами класса считаются не только имена переменных внутри класса, но и имена функций, определенных в классе.

Определенные внутри класса имена наследуются всеми экземплярами класса. Атрибуты определяют свойства и поведение объекта, причем атрибуты могут создаваться и внутри метода класса.

Синтаксис:

переменная=значение

Для доступа к значению экземпляра класса необходимо указать имя экземпляра класса и через точку имя атрибута.

Синтаксис:

имя\_экземпляра\_класса.переменная

Для изменения свойства (атрибута) достаточно присвоить ему новое значение.

#### Упражнение 4.8

Выполните задание, как в упражнении 4.7, не используя конструктор класса. Напишите следующий код программы, запустите его и сравните результат.

Код:

```
class student1:
    kurs=0
```

```
student3=student1() # создается объект 0-го курса
student3.kurs=2 # атрибут курс меняется на 2-й
student4=student1() # создается объект 0-го курса
student4.kurs=3 # атрибут курс меняется на 2-й
print('Студент 3 учится на', student3.kurs, '-м курсе')
print('Студент 4 учится на', student4.kurs, '-м курсе')
```

Результат:

Студент 3 учится на 2-м курсе

Студент 4 учится на 3-м курсе

Как видно из этого упражнения, код программы увеличился на две строки по сравнению с предыдущим кодом. Другими словами, сначала создается экземпляр класса, потом его атрибуту присваивается некоторое значение.

*Закрытые аргументы* — это аргументы, которые не могут быть изменены после инициализации объекта, что очень похоже на свойство кортежа.

Существует два типа аргументов — это *защищаемый аргумент (атрибут)* (одно нижнее подчеркивание) и *частный аргумент (атрибут)* (два нижних подчеркивания).

Прежде всего, такое разделение сделано для разработчиков, чтобы понимать, что атрибут первого типа не должен быть изменен, а аргумент второго типа не должен быть доступен извне. Такой аргумент в других языках программирования называют *приватным (private)*. Продемонстрируем это на примере.

### Пример

Код:

```
class student:
    def __init__(self, kurs, sex):
        self.kurs=kurs # обычный аргумент
        self._sex=sex # защищаемый аргумент
sex=input('Введите пол 1-го студента: ')
student1=student(2, sex) # создание экземпляра
print(student1.sex) # печать атрибута экземпляра
```

Результат:

Введите пол 1-го студента: мужской

Traceback(most recent call last):

File "D:/.../классы\_init\_закр.py", line 7, in <module>

print(student1.sex) # печать атрибута экземпляра

AttributeError: 'student' object has no attribute

'sex'

>>>

Иными словами, нет возможности прочитать, что такой атрибут содержится в данном классе. Однако если заменить атрибут `sex` на `_sex`, то результат получится следующим:

Код:

...

```
print(student1._sex) # печать атрибута экземпляра
```

Результат:

Введите пол 1-го студента: мужской

мужской

Пусть необходимо в процессе программы изменить значение атрибута. Для этого дополним код строчками.

Код:

```
student1.sex='средний' # изменение значения атрибута
```

```
print(student1.sex) # печать атрибута экземпляра
```

```
print(student1._sex) # печать атрибута экземпляра
```

Результат:

Введите пол 1-го студента: мужской

мужской

средний

мужской

>>>

При этом значение атрибута изменилось без ошибок, поскольку ограничение вводилось только на инициализацию. Изменилось значение у экземпляра, а у класса значение не изменилось. Добавим строчки.

Код:

```
student2=student(2, sex)
```

```
print(student2.sex)
```

Результат:

Обработчик выдает ошибку ...

При замене `sex` на `_sex` выйдет значение мужской.

Для того чтобы защитить атрибут от изменений, используется конструкция, представленная в следующем примере описания класса `student`.

**Пример**

Код:

```
class student:
```

```
    def __init__(self, kurs, sex):
```

```
        self.kurs=kurs # обычный аргумент
```

```
        self._sex=sex # защищаемый аргумент
```

```
@property # декоратор – защита от изменений
    #функция после – защита от изменений
def sex(self):
    return self._sex
sex=input('Введите пол 1-го студента: ')
student1=student(2, sex) # создание экземпляра
student1.sex='средний' # попытка заменить значение
```

атрибута

Результат:

Введите пол 1-го студента: мужской

Traceback(most recent call last):

```
File "D:/.../классы_init_закр.py", line 10, in <module>
student1.sex='средний' # попытка заменить значение
```

атрибута

```
AttributeError: can't set attribute
```

```
>>>
```

Как видно, выводится исключение «Ошибка атрибута», которое свидетельствует, что атрибут не может быть изменен.

#### Упражнение 4.9

Создайте класс «студент» (название можно написать на русском языке), который будет содержать неизменяемые атрибуты: ФИО, направление подготовки и год поступления. Курс является изменяемым атрибутом. Создайте два экземпляра класса и протестируйте программу.

Как было сказано ранее, *частные аргументы* не могут быть доступны при обращении к ним, например, идентификатор студента.

#### Пример

Код:

```
class student:
    def __init__(self, kurs, id1):
        self.kurs=kurs # обычный аргумент
        self.__id=id1 # частный аргумент
id1=input('Введите идентификатор студента: ')
student1=student(1, id1) # создание экземпляра
print(student1.id1) # обращение к аргументу(атрибуту)
```

Результат:

Введите идентификатор студента: 1

Traceback(most recent call last):

```
File "D:/.../классы_init_закр_private.py", line 8, in
<module>
```

```
print(student1.id1)
```



```
AttributeError: 'student' object has no attribute 'idl'
```

Иными словами, интерпретатор сообщает, что такого атрибута нет у экземпляра «студент».

Замена последней строки кода на `print(student1.__id)` также не дает ожидаемого результата, который был возможен при работе с защищенными аргументами.

Однако это можно обойти, если записать сначала класс объекта, а затем его свойство.

### Пример

Код:

```
class student:
    def __init__(self, kurs, idl):
        self.kurs=kurs # обычный аргумент
        self.__id=idl # частный аргумент
idl=input('Введите идентификатор студента: ')
student1=student(1, idl) # создание экземпляра
print(student1._student__id)
```

Результат:

```
Введите идентификатор студента: 1
1
```

Другими словами, и эта защита может быть снята, если знать внутреннее имя атрибута (аргумента).

### Упражнение 4.10

Используя результаты упражнения 4.9, добавьте идентификационный номер и e-mail студента, которые должны быть частными.

Аналогичный подход можно реализовать и для методов класса, которые будут рассмотрены ниже. Методы класса довольно разнообразны, и поэтому они выделены в отдельную тему.

### Задания для самостоятельного выполнения по теме «Классы в языке Python»

1. Создайте класс «сотрудник склада», который имеет следующие атрибуты: Код сотрудника, Фамилия, Имя, Отчество, Должность, Дата приема на работу, Уровень заработной платы. Создайте двух сотрудников и выведите данные о них на экран.
2. Определите защищенные и частные атрибуты и модифицируйте программу из предыдущего задания.
3. Модифицируйте предыдущую программу, когда ввод сотрудников осуществляется в цикле с запросом: «Вы хотите продолжить ввод данных? (да/нет)». После окончания ввода должен выводиться список (это подсказка) всех введенных сотрудников.

4. Добавьте обработчик исключений к предыдущей задаче.
5. Модифицируйте программу так, чтобы вывод сотрудников и защищенной информации осуществлялся по паролю. Пароли хранятся в словаре, где ключ – имя сотрудника, а значение – пароль.

### 4.3. МЕТОДЫ КЛАССА

Методы класса создаются точно так же, как и функции, посредством ключевого слова `def` `имя_метода(self, передаваемые аргументы)`.

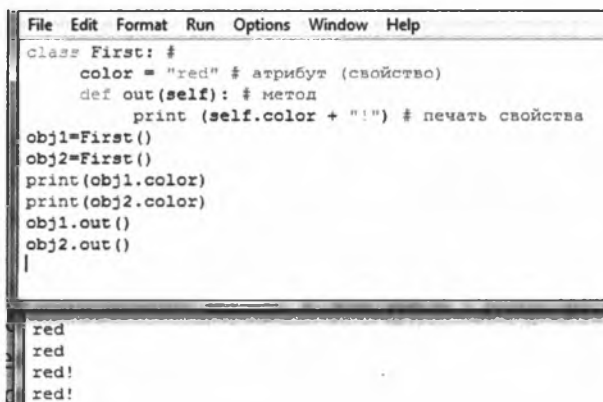
При этом главным отличием от обычной функции является наличие в скобках, стоящих после имени метода, ключевого слова `self`, куда передается имя объекта, к которому будет применен соответствующий метод.

Такое представление было использовано ранее, когда необходимо было запретить изменение атрибута.

Для вызова метода используется конструкция:

`имя_объекта.имя_метода([передаваемые аргументы])`

Пример реализации метода представлен на рис. 4.1.



```
File Edit Format Run Options Window Help
class First: #
    color = "red" # атрибут (свойство)
    def out(self): # метод
        print (self.color + "!") # печать свойства
obj1=First()
obj2=First()
print(obj1.color)
print(obj2.color)
obj1.out()
obj2.out()

red
red
red!
red!
```

Рис. 4.1. Пример реализации метода класса

Здесь создается два объекта класса `First` и к ним применяется метод `out()`, который выводит цвет объекта, добавляя к нему знак восклицания.

Как было показано ранее, для создания класса лучше использовать специальный метод `__init__(self, [список передаваемых аргументов])`, который позволяет создать экземпляр класса с уже заданными свойствами. В предыдущих примерах и упражнениях

он был использован, но ввиду его особой важности рассмотрим его подробнее.

#### 4.3.1. Конструктор класса `__init__()`

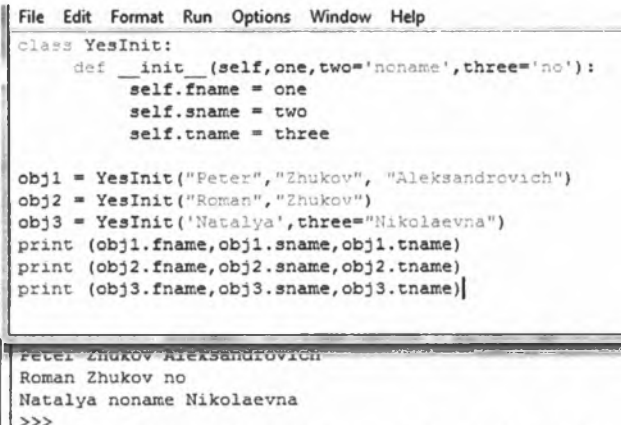
Конструктор класса `__init__()` — специальный метод, который при создании экземпляра класса автоматически создает ему атрибуты (указывается два нижних подчеркивания слева и справа).

Синтаксис:

```
def __init__(self [, список передаваемых аргументов]):  
    self.имя_аргумента=передаваемый аргумент(значение)  
    ...
```

При этом некоторые аргументы можно задать по умолчанию. Тогда в случае отсутствия значения аргумента его значение будет установлено по умолчанию.

Пример такой реализации представлен на рис. 4.2.



```
File Edit Format Run Options Window Help  
class YesInit:  
    def __init__(self, one, two='noname', three='no'):  
        self.fname = one  
        self.sname = two  
        self.tname = three  
  
obj1 = YesInit("Peter", "Zhukov", "Aleksandrovich")  
obj2 = YesInit("Roman", "Zhukov")  
obj3 = YesInit("Natalya", three="Nikolaevna")  
print (obj1.fname, obj1.sname, obj1.tname)  
print (obj2.fname, obj2.sname, obj2.tname)  
print (obj3.fname, obj3.sname, obj3.tname)  
  
Peter Zhukov Aleksandrovich  
Roman Zhukov no  
Natalya noname Nikolaevna  
>>>
```

Рис. 4.2. Пример использования конструктора класса

Отличие такого конструктора класса от обычного метода заключается в форме вызова метода при создании экземпляра класса (рис. 4.3).

Как видно из рисунка, созданы два класса, первый из которых содержит конструктор, а второй — обычную функцию (метод). При этом в первом случае при создании объекта — экземпляра класса атрибуты создаются автоматически. Во втором случае необходимо сначала присвоить `obj1` класс, а затем для задания свойств вызвать метод `names()`, в котором задать значения необходимых аргументов. Последний способ увеличивает код на одну строчку, т.е. создание экземпляра класса осуществляется в два этапа.

```

File Edit Format Run Options Window Help
class YesInit:
    def __init__(self, one, two, three):
        self.fname = one
        self.sname = two
        self.tname = three

obj1 = YesInit("Peter", "Zhukov", "Aleksandrovich")

print (obj1.fname, obj1.sname, obj1.tname)
class NoInit:
    def names(self, one, two, three):
        self.fname = one
        self.sname = two
        self.tname = three

obj1 = NoInit()
obj1.names("Peter", "Zhukov", "Aleksandrovich") # Отличается вызовом метода

print (obj1.fname, obj1.sname, obj1.tname)

RESTART: D:\POMA\Работа с python\upriz_cof_init.py
Peter Zhukov Aleksandrovich
Peter Zhukov Aleksandrovich
>>>

```

**Рис. 4.3.** Отличие реализации конструктора класса от обычного метода

Также при использовании конструктора класса свойства созданных объектов можно менять (рис. 4.4).

В данном примере создается класс «товар», одним из атрибутов которого является количество. В программе таких экземпляров три: «кукла пластиковая — 10 шт.», «мишка плюшевый — 200 шт.» и «ружье охотничье — 0 шт. (по умолчанию)». Данные экземпляры класса создаются автоматически посредством конструктора `__init__()`. Свойства (значения атрибутов) указываются непосредственно при создании экземпляра класса.

Метод `mwhere()` определяет, где должен находиться товар: в магазине, на складе или отсутствует в зависимости от количества.

Методы `plus()`, `minus()` позволяют изменять значение переменной `n`, которая отвечает за количество товара и его размещение (на складе или в магазине). Обращаясь к данным методам и указывая приход (расход) определенного экземпляра класса (передаваемое значение `p` или `m`), можно изменять (увеличивать или уменьшать) значение атрибута (`n`).

Таким образом, можно имитировать процесс изменения объема товаров организации и формировать их учет.

```

class Place:
    def __init__(self, na, t, n=0):
        self.name = na
        self.type = t
        self.numbers = n
        self.mwhere(n)

    def mwhere(self, n):
        if n <= 0:
            self.where = "отсутствуют"
        elif 0 < n < 100:
            self.where = "магазин"
        else:
            self.where = "склад"

    def plus(self, p):
        self.numbers = self.numbers + p
        self.mwhere(self.numbers)
    def minus(self, m):
        self.numbers = self.numbers - m
        self.mwhere(self.numbers)

p1 = Place("кукла", "пластиковая", 10)
p2 = Place("мешка", "плхшевый", 200)
p3 = Place("ружье", "охотничье")

print (p1.name, p1.type, p1.where)
print (p2.name, p2.type, p2.where)
print (p3.name, p3.type, p3.where)

p2.minus(20)
p3.plus(5)

print (p2.name, p2.numbers, p2.where)
print (p3.name, p3.numbers, p3.where)

```

---

```

кукла пластиковая магазин
мешка плхшевый склад
ружье охотничье отсутствуют
мешка 180 склад
ружье 5 магазин
>>>

```

**Рис. 4.4.** Замена значений атрибутов экземпляра класса

### Упражнение 4.11

Создайте класс студентов, имеющих следующие атрибуты: ФИО, номер группы, оценку на последнем экзамене, которая влияет на свойство, принимающее значения: отличник, хорошист, середнячок, двоечник. Организуйте ввод и вывод информации о студентах (четыре объекта).

#### 4.3.2. Использование ссылки на экземпляр класса

Можно на основе уже существующего экземпляра создать новый экземпляр класса простым присвоением.

#### Пример

Код:

```

class student:
    def __init__(self, kurs, id1):

```

```

        self.kurs=kurs
        self.id=id1
    id1=input('Введите идентификатор студента: ')
    kurs=input('Введите курс студента: ')
    student1=student(kurs, id1) # создание экземпляра
    print('Курс:', student1.kurs, student1.id, sep=' ')
    student2=student1
    id1=input('Введите идентификатор 2-го студента: ')
    student2.id=id1
    print('Курс:', student1.kurs, student1.id, sep=' ')
    kurs=input('Введите новый курс студентов: ')
    student1.kurs=kurs
    print('Новый курс:', student1.kurs, 'Идентификатор:',
student1.id, sep=' ')
    print('Новый курс:', student2.kurs, 'Идентификатор:',
student2.id, sep=' ')

```

Результат:

```

Введите идентификатор студента: 35
Введите курс студента: 2
Курс: 2 35
Введите идентификатор 2-го студента: 36
Курс: 2 36
Введите новый курс студентов: 3
Новый курс: 3 Идентификатор: 36
Новый курс: 3 Идентификатор: 36
>>>

```

В данном примере показано, что при изменении свойства одного экземпляра изменяются свойства и другого экземпляра, поскольку они являются копиями друг друга и, по сути, являются ссылками на базовый класс.

#### Упражнение 4.12

Модифицируйте код программы из предыдущего примера так, чтобы студенты с разными идентификаторами переводились одновременно на другой курс. При этом идентификаторы не должны меняться.

### 4.3.3. Статические методы

Статические методы отличаются тем, что при их использовании не нужно создавать экземпляр класса. Для их объявления необходимо записать специальный декоратор `@staticmethod`. После декоратора в тексте программы (в блоке класса) необходимо описать непосредственно статический метод (рис. 4.5).

```

class Class1(object):
    @staticmethod
    def sum1(x, y):                # Статический метод
        return x + y
    def sum2(self, x, y):         # Обычный метод в классе
        return x + y
    def sum3(self, x, y):
        return Class1.sum1(x, y) # Вызов из метода класса

print Class1.sum1(10, 20)       # Вызываем статический метод
c1 = Class1()
print (c1.sum2(15, 6))          # Вызываем метод класса
print (c1.sum1(50, 12))        # Вызываем статический метод
                                # через экземпляр класса
print (c1.sum3(23, 5))         # Вызываем статический метод
                                # внутри класса

```

**Рис. 4.5.** Использование статического метода

Из рисунка видно, что статический метод не содержит ключевого слова `self`, т.е. можно сказать, что он определяется вне класса.

#### **Упражнение 4.13**

Создайте статический метод, который переводит студентов на следующий курс.

#### **4.3.4. Закрытые методы**

*Закрытые методы* — это методы, которые не могут быть вызваны непосредственным обращением к ним. Их описание очень похоже на описание защищенных и частных атрибутов, рассмотренных ранее. Для их создания перед именем метода необходимо добавить одно или два нижних подчеркивания.

#### **Пример**

Код:

```

class student:
    def __init__(self, kurs, id1):
        self.kurs=kurs
        self.id1=id1
    def _perevod(self, kurs, id1):
        self.kurs=kurs+1
        print('Студент с id', id1,'переведен на',
self.kurs,'-й курс', sep=' ')
    id1=input('Введите идентификатор студента: ')
    kurs=int(input('Введите курс студента: '))
    student1=student(kurs, id1) # создание экземпляра
    z=input('Перевести студента на следующий курс?(да/нет) ')
    ifz=='да':
        student1._perevod(kurs, id1)

```

```

elif z=='нет':
    pass
else:
    print('Введено неправильное значение')
    print('Студент cid', studentl.idl,'учится на', studentl.kurs,'-м курсе', sep=' ')
Результат:
Введите идентификатор студента: 1
Введите курс студента: 1
Перевести студента на следующий курс?(да/нет) да
Студент с id 1 переведен на 2-й курс
Студент с id 1 учится на 2-м курсе
>>>

```

#### Упражнение 4.14

Основываясь на результатах предыдущего примера, создайте приватный метод и добавьте в код обработчик исключений на ввод некорректных данных.

### 4.3.5. Специальные методы

Под специальными методами понимают встроенный класс методов Python, которые работают с экземплярами классов. Они начинаются и заканчиваются двумя нижними подчеркиваниями.

Наиболее часто используемые методы приведены ниже.

Метод `__init__()` — подробно рассмотрен в подпараграфе 4.3.1.

Метод `__del__()` — это деструктор или «уничтожитель» объекта, который вызывается автоматически в Python. Пример работы с деструктором представлен на рис. 4.6.

```

class Class1:
    def __init__(self): # Конструктор класса
        print "Вызван метод __init__()"
    def __del__(self): # Деструктор класса
        print "Вызван метод __del__()"
c1 = Class1()          # Выведет: Вызван метод __init__()
del c1                # Выведет: Вызван метод __del__()
c2 = Class1()          # Выведет: Вызван метод __init__()
c3 = c2               # Создаем ссылку на экземпляр класса
del c2                # Ничего не выведет, т.к. существует ссылка
del c3                # Выведет: Вызван метод __del__()

```

Рис. 4.6. Использование деструктора

Данный метод не будет вызван, если на экземпляр класса существует хотя бы одна ссылка. Поэтому его использование в Python особого смысла не имеет.



Следующий метод `__doc__()` — строки документации. При его вызове будет выведена документация по соответствующему классу объектов.

Для его вызова используется следующий синтаксис:

```
print(имя_объекта.__doc__)
```

Для определения состава соответствующего класса можно использовать функцию `dir()` (рис. 4.7).

```
>>> import math
>>> print(math.__doc__)
This module is always available. It provides access to the
mathematical functions defined by the C standard.
>>> dir
<built-in function dir>
>>> dir(math)
['_doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fm
od', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'is
inf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan',
'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
>>> |
```

Рис. 4.7. Использование `__doc__()`.

`__name__` — определяет имя класса.

`__dict__` — выводит словарь атрибутов класса.

`__bases__` — выводит в порядке следования базовых классов кортеж.

`__call__` — всегда вызывается при обращении к классу.

`__cmp__` — данный метод вызывается при любой операции сравнения.

`__getattr__` — позволяет получить значение атрибута класс, который не доступен при обычном вызове.

`__setattr__` — аналогичен предыдущему методу, только присваивает значение недоступному аргументу (атрибуту).

`__delattr__` — осуществляет удаление атрибута.

Подробнее о других специальных методах можно узнать на официальном сайте Python<sup>1</sup>.

### Упражнение 4.15

Получите документацию на модули `re`, `random` и `sys`.

### Упражнение 4.16

Создайте для программы из упражнения 4.14 собственную документацию и вызовите ее с помощью `__doc__`.

---

<sup>1</sup> URL: <https://docs.python.org/3/>

### 4.3.6. Перегрузка операторов

*Перегрузка операторов* — это один из возможных способов реализации полиморфизма, когда специальные методы заменяются пользовательскими методами. Иными словами, для созданного пользовательского класса метод реализуется по-своему.

В следующем примере, представленном на рис. 4.8, осуществляется перегрузка операторов сложения (`__add__`), строковое представление (`__str__`).

```
class Newclass:
    def __init__(self, base):
        self.base = base
    def __add__(self, a):
        self.base = self.base + a
    def __str__(self):
        return "%s !!!" % self.base

a = Newclass(10)
a + 20
print (a)

b = Newclass("yes")
b + "terday"
print (b)

c = Newclass([2,6,3])
c + [7, 1]
print (c)

30 !!!
yesterday !!!
[2, 6, 3, 7, 1] !!!
>>>
```

**Рис. 4.8.** Перегрузка операторов

На следующем рисунке представлена перегрузка оператора `__call__` и `__str__` (рис. 4.9).

```
class Changeable:
    def __init__(self, color):
        self.color = color
    def __call__(self, newcolor):
        self.color = newcolor
    def __str__(self):
        return "%s" % self.color

canvas = Changeable("green")
frame = Changeable("blue")

canvas("red")
frame("yellow")

print (canvas, frame)

red yellow
>>>
```

**Рис. 4.9.** Перегрузка операторов (`__call__`, `__str__`)

Данный пример демонстрирует возможность замены атрибута (в данном случае «цвет») при обращении к данному объекту. В результате вызывается метод `__call__`, который и устанавливает новый цвет экземпляра класса. Метод `__str__` позволяет вывести результат в виде строкового выражения.

#### Упражнение 4.17

Дополните код программы, представленной на рисунке, дополнительными методами класса — методами `__mul__` (вызывается при использовании объекта в операциях умножения) и `__sub__` (вычитание). Вызовите данные методы с помощью соответствующих операций с объектами. Для каких объектов невозможно использовать метод `__sub__`?

*Задания для самостоятельного выполнения по теме «Методы класса»*

1. Создайте для класса «студент» функцию, которая считает средний балл по трем дисциплинам, оценки по которым вводятся пользователями. Реализуйте программу двумя способами: без использования конструктора `__init__` и с ним.
2. Создайте функцию, которая считает и выводит в файл заработную плату сотрудника склада в зависимости от количества отработанных дней в месяце. Оклад является защищаемым атрибутом класса «сотрудник».
3. Используя результат предыдущего задания, создайте функцию начисления премии сотруднику, вычисляемую как 1% от уровня заработной платы.
4. Модифицируйте предыдущую программу, позволяющую вычислять заработную плату, премию для сотрудников и определять величину общего фонда заработной платы как сумму заработной платы и премий, деленную на коэффициент 0,7. Данные о сотрудниках вводятся через цикл `while` и записываются в соответствующий файл `.txt` или `.xlsx`.
5. Используя технологию перегрузки операторов, измените `__truediv__` (деление) так, чтобы числителем был знаменатель и наоборот. Аналогичные процедуры осуществите для операторов `__floordiv__` (целочисленное деление), `__mod__` (остаток от деления). Числа вводят пользователи. Предусмотрите обработку исключений.

## 4.4. НАСЛЕДОВАНИЕ

*Наследование* — это не что иное, как использование свойств родительского класса в классе-наследнике. Это понятие было подробно рассмотрено в параграфе 4.1.

### 4.4.1. Простое наследование

Рассмотрим простой пример, когда на основе существующего класса экземпляру этого класса придается новое свойство (рис. 4.10).

```
File Edit Format Run Options Window Help
class Things:
    def __init__(self,n,t):
        self.namething = n
        self.total = t

th1 = Things("стол", 5)
th2 = Things("стулья", 7)

print (th1.namething,th1.total) # Вывод: столов 5
print (th2.namething,th2.total) # Вывод: стульев 7

th1.color = "ольха" # новое свойство объекта th1

print (th1.color) # Вывод: ольха
print (th2.color) # ОШИБКА: у объекта th2 нет свойства color!
```

**Рис. 4.10.** Добавление новых свойств

Как видно из примера, у второго объекта нет свойства «цвет», которое было создано только для первого экземпляра.

Следующий пример демонстрирует создание нового класса (кухонный стол) на базе родительского класса (стол) (рис. 4.11).

```
File Edit Format Run Options Window Help
class Table:
    def __init__(self,l,w,h):
        self.long = l
        self.width = w
        self.height = h
    def outing(self):
        print (self.long,self.width,self.height)

class Kitchen(Table):
    def howplaces(self,n):
        if n < 2:
            print ("Это не кухонный стол")
        else:
            self.places = n
    def outplaces(self):
        print (self.places)

t_1 = Kitchen(2,1,0.5)
t_1.outing()
t_1.howplaces(5)
t_1.outplaces()

t_2 = Table(1,3,0.7)
t_2.outing()
t_2.howplaces(8) # ОШИБКА
```

**Рис. 4.11.** Добавление новых свойств

### Упражнение 4.18

Создайте новые классы компьютерных и рабочих столов с дополнительными методами, отличающимися от предыдущего примера, представленного на рисунке.

## 4.4.2. Множественное наследование

Множественное наследование предполагает использование методов от разных классов (рис. 4.12).

```
class Class1: # Базовый класс для класса Class2
    def f_func1(self):
        print ("Метод f_func1() класса Class1")

class Class2(Class1): # Класс Class2 наследует класс Class1
    def f_func2(self):
        print ("Метод f_func2() класса Class2")

class Class3(Class1): # Класс Class3 наследует класс Class1
    def f_func1(self):
        print ("Метод f_func1() класса Class3")
    def f_func2(self):
        print ("Метод f_func2() класса Class3")
    def f_func3(self):
        print ("Метод f_func3() класса Class3")
    def f_func4(self):
        print ("Метод f_func4() класса Class3")

class Class4(Class2, Class3): # Множественное наследование
    def f_func4(self):
        print ("Метод f_func4() класса Class4")

c1 = Class4() # Создаем экземпляр класса Class4
c1.f_func1() # Выведет: Метод f_func1() класса Class1
c1.f_func2() # Выведет: Метод f_func2() класса Class2
c1.f_func3() # Выведет: Метод f_func3() класса Class3
c1.f_func4() # Выведет: Метод f_func4() класса Class4
```

Рис. 4.12. Множественное наследование<sup>1</sup>

### Упражнение 4.19

Имеется класс «сотрудники», в котором определяется заработная плата, и класс «студенты», где вычисляется средний оценочный балл предыдущей сессии. Составьте новый класс студентов, получающих стипендию в зависимости от среднего балла.

## 4.4.3. Абстрактные методы

Абстрактные методы используют в случае, когда этот метод должен быть переопределен для классов-наследников. По сути, такой подход также реализует полиморфизм: один метод — множество реализаций. Абстрактный метод создается в базовом родительском классе и переопределяется в классах-наследниках (рис. 4.13).

Также для объявления абстрактного метода используют встроенный декоратор Python @abstractmethod (рис. 4.14).

### Упражнение 4.20

Напишите следующий код, представленный на рис. 4.15. Запустите программу и прокомментируйте результат ее выполнения.

<sup>1</sup> URL: <http://python-3.ru/page/mnozhestvennoe-nasledovanie-v-python>

```

class Class1(object):
    def test(self, x): # Абстрактный метод
        # Возбуждаем исключение с помощью raise
        raise NotImplementedError("Необходимо переопределить метод")
class Class2(Class1): # Наследуем абстрактный метод
    def test(self, x): # Переопределяем метод
        print (x)
class Class3(Class1): # Класс не переопределяет метод
    pass
c2 = Class2()
c2.test(50) # Выведет: 50
c3 = Class3()
try: # Перехватываем исключения
    c3.test(50) # Ошибка. Метод test() не переопределен
except NotImplementedError as msg:
    print (msg) # Выведет: Необходимо переопределить метод
50
Необходимо переопределить метод
>>>

```

Рис. 4.13. Абстрактный метод<sup>1</sup>

```

from abc import abstractmethod #подключение декоратора
class Class1(object):
    @abstractmethod # декоратор
    def pr(self, x): # Абстрактный метод
        pass
class Class2(Class1): # Наследуем абстрактный метод
    def __init__(self,x):
        self.x=x
    def pr(self, x): # Переопределяем метод
        print ('Вы ввели ',self.x, '!')
class Class3(Class1): # Класс не переопределяет метод
    pass
x=int(input('Введите значение x:'))
c2 = Class2(x)
c2.pr(x) # Выведет: x
try:
    c3 = Class3() # Ошибка. Метод test() не переопределен
    c3.pr(x)
except TypeError as msg:
    print (msg) # Can't instantiate abstract class Class3
# with abstract methods test

```

Рис. 4.14. Абстрактный метод<sup>2</sup>

```

class T1:
    n=10
    def total(self,N):
        self.total = int(self.n) + int(N)

class T2:
    def total(self,s):
        self.total = len(str(s))

t1 = T1()
t2 = T2()
t1.total(45)
t2.total(45)
print (t1.total) # Вывод: 55
print (t2.total) # Вывод: 4

```

Рис. 4.15. Переопределение метода класса

<sup>1</sup> URL: <http://python-3.ru/page/abstraktnye-metody>

<sup>2</sup> Там же.

Также можно переопределять классы в классе-наследнике. Если какого-либо метода нет в классе-наследнике, то интерпретатор ищет его базовом классе.

### Упражнение 4.21

Напишите следующий код, представленный на рис. 4.16. Запустите программу и прокомментируйте результат ее выполнения.

```
File Edit Format Run Options Window Help
class Base:
    def __init__(self,n):
        self.numb = n
    def out(self):
        print (self.numb)

class One(Base):
    def multi(self,m):
        self.numb *= m

class Two(Base):
    def inlist(self):
        self.inlist = list(str(self.numb))
    def out(self):
        i = 0
        while i < len(self.inlist):
            print (self.inlist[i])
            i += 1

obj1 = One(45)
obj2 = Two('abc')

obj1.multi(2)
obj1.out() # Вывод числа 90

obj2.inlist()
obj2.out() # Вывод в столбик букв a, b, c
```

Рис. 4.16. Переопределение метода класса

### Задания для самостоятельного выполнения по теме «Наследование»

1. Создайте класс «зарплата», который вычисляет сумму зарплаты в зависимости от тарифной ставки и числа отработанных дней. Расширьте метод начислением процентов в зависимости от объема продаж.
2. Переопределите метод начисления зарплаты: в случае, если объем продаж в месяц превысил 1 000 000, проценты добавляются к зарплате, в противном случае – вычитаются.
3. Создайте для класса «сотрудники» абстрактный метод начисления зарплаты. Создайте два класса-наследника: сотрудник отдела продаж и сотрудник склада. Для первого класса начисление происходит аналогично заданию 1, а для второго – в зависимости от количества отработанных дней.
4. Решите предыдущую задачу, используя декоратор @abstractmethod.

5. Имеется класс «мебель». Создайте класс «корпусная мебель», «мягкая мебель» и «кухонная мебель». Определите атрибуты и методы родительского класса и классов-наследников.

### *Контрольные вопросы и задания*

1. Что такое класс в Python? Каковы его основные характеристики?
2. Опишите базовые принципы ООП.
3. Что такое экземпляр класса? Каким образом можно осуществить его создание?
4. Дайте определение атрибута класса и опишите его основные особенности.
5. Что такое методы класса? Каковы особенности создания и вызова метода?
6. Каковы отличия закрытых методов от обычных?
7. В чем заключается преимущество использования конструктора `__init__()` при создании класса?
8. Как осуществляется перегрузка специальных методов класса в Python?
9. Как реализуется принцип наследования в Python? Приведите примеры.
10. В чем смысл использования абстрактного метода в Python?



## Глава 5

# РАЗРАБОТКА ГРАФИЧЕСКИХ ИНТЕРФЕЙСОВ В ПРОГРАММЕ НА ЯЗЫКЕ PYTHON

### 5.1. СОБЫТИЙНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Под *событийно-ориентированным программированием* (СОП) понимают организацию диалога с пользователем или некоторым приложением, когда на их действия программа запускает тот или иной код с целью обработки этих действий (фактов или событий). Все программы, которые связаны с использованием графического интерфейса, являются событийно-ориентированными, поскольку обеспечивают взаимодействие программы с пользователем.

#### 5.1.1. Событие

Под *событием* понимают факт совершения каких-либо действий со стороны пользователя или приложения (в большинстве своем — пользователя).

Такими действиями могут быть ввод данных в форму, нажатие на форму одной из кнопок мыши, закрытие окна и т.п. Главным отличительным свойством событийно-ориентированного программирования является то, что программа, состоящая, главным образом, из основного цикла приложения, который включает выборку события и его обработку, в зависимости от события запускает то или иное приложение (подпрограмму или блок программы). При этом события могут быть вызваны, например, действиями со стороны веб-сервера на запрос о загрузке интернет-страницы в браузер и т.п.

#### Упражнение 5.1

Укажите возможные события (не менее семи), которые могут возникнуть при диалоге пользователя с программой.

#### 5.1.2. Обработчик события

*Обработчик события* — это некоторая функция или процедура (тело подпрограммы, метод), которая вызывается или срабатывает при наступлении определенного пользователем или внешним приложением события.

Например, при нажатии левой кнопкой мыши (ЛКМ) на кнопку «Открыть файл» вызывается диалоговое окно для выбора из каталога требуемого файла; нажатие «крестика» диалогового окна приводит к его закрытию и т.п. Очевидно, что обработчик события —

это мощный инструмент обеспечения релевантного взаимодействия пользователя и приложения, а событийно-ориентированное программирование в настоящее время является основной парадигмой для разработчиков современных информационных систем.

### Упражнение 5.2

Составьте концептуальную модель (описательная модель на естественном языке) обработчика нажатия ЛКМ на кнопку панели инструментов «Сохранить» в MS Office.

#### 5.1.3. Цикл обработки событий

*Цикл обработки событий* — это последовательность (алгоритм) действий программы, предназначенной для «прослушивания» действий пользователя (сторонних программ) и при наступлении определенных событий вызова соответствующего метода. При вызове метода или запуска основной программы реализуется ее главный цикл, который получает события и передает их определенным объектам.

Модель цикла включает в себя трех участников: источник события, объект события и цель события.

*Источник события* — это объект, у которого меняется состояние и который вызывает событие. При этом событие порождает изменение в источнике события.

*Цель события* — это не что иное, как объект, который ожидает сообщения, что событие произошло и необходимо совершить определенные действия.

*Объект события* — это объект, который создается при совершении события, содержит детальную информацию о событии и передает ее цели события.

*Главный цикл обработки событий* — это бесконечный цикл, который постоянно ожидает следующего события. На рис. 5.1 представлена структурная схема данного цикла.



Рис. 5.1. Структурная схема главного цикла обработки событий

### Упражнение 5.3

Составьте алгоритм и представьте его в виде блок-схемы обработчика запроса на загрузку интернет-страницы в браузер, предполагая, что модель взаимодействия основана на клиент-серверной архитектуре.

#### *Задания для самостоятельного выполнения по теме «Событийно-ориентированное программирование»*

1. Студент регистрируется на информационно-образовательном портале вуза. Составьте концептуальную модель главного цикла обработчика событий, который включает основные события, обработчик событий, вызываемые методы.
2. Сотрудник склада вводит информацию о поступивших товарах в информационную систему организации. Составьте концептуальную модель главного цикла обработчика событий, который включает основные события, обработчик событий, вызываемые методы.
3. Схематически представьте обработчик формирования заказа в магазине.
4. Основываясь на предыдущем задании, включите в него обработчик исключений. Опишите основные возможные исключения.
5. Производится оплата заказа через терминал. Составьте концептуальную модель главного цикла обработчика событий, которая включает основные события, обработчик событий, вызываемые методы.

## **5.2. ИНСТРУМЕНТЫ ДЛЯ СОЗДАНИЯ ГРАФИЧЕСКИХ ИНТЕРФЕЙСОВ ПОЛЬЗОВАТЕЛЯ (GUI)**

### **5.2.1. Общие сведения о GUI Python**

*Графический интерфейс пользователя* (Graphical User Interface, GUI) – это представление, которое обеспечивает наглядное и интуитивно понятное взаимодействие пользователя с программой. Интерфейс может содержать в себе базовое окно (другие всплывающие окна) и набор инструментов (в Python они называются виджетами) (рис. 5.2).

Раскроем понятия окна и виджета, хотя каждому интуитивно ясно, что они собой представляют.

*Окно* (window) – это область экрана (традиционно прямоугольной формы), которая обеспечивает взаимодействие пользователя с программой. В нем размещены основные (требуемые) инструменты организации такого взаимодействия. Они называются визуальными компонентами, или виджетами.

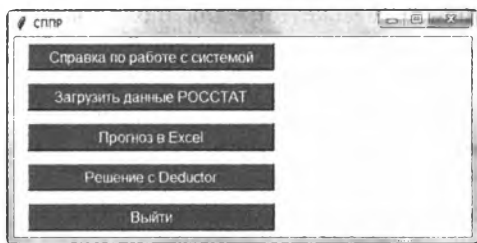


Рис. 5.2. Пример графического интерфейса Python

*Виджет* (от англ. *widget* – штука) – это организованный специальным образом визуальный компонент, цель которого – обеспечить взаимодействие пользователя с программой в рамках выполняемых ею задач. Виджеты могут быть вложенными друг в друга, что определяется концепцией любого GUI, и формируют тем самым иерархическую структуру графического пользовательского интерфейса. Например, виджет `frame` (рамка) может содержать в себе кнопку (`button`) и надпись (`label`).

Иерархическая структура виджетов называется деревом виджетов, которая состоит из родительских и дочерних виджетов.

Для создания графического интерфейса в Python имеется встроенный модуль `tkinter`, на изучение возможности использования которого ориентирован данный параграф. Также имеется модуль `PyQt5` от компании *Digia*, который представляет собой набор библиотек для создания графических интерфейсов. Скачать его для Windows можно с официального сайта<sup>1</sup>, а с основами работы ознакомиться на сайте <https://pythonworld.ru/gui>.

Модуль `tkinter` содержит следующие классы виджетов.

**Button (кнопка)** – это обычная кнопка, при нажатии на которую программа выполняет какие-либо действия.

**Label (надпись)** – обычный текст, который может быть размещен в любом месте главного окна.

**Message (сообщение)** – аналогичен `Label`, но позволяет по размеру области менять и «заворачивать» длинные строки.

**Entry (поле ввода)** – специальная область, где пользователь может ввести однострочный текст.

**Text (текст)** – позволяет вводить многострочный текст заданного формата и стиля, а также встраивать изображения.

<sup>1</sup> URL: <https://riverbankcomputing.com/software/pyqt/download5>

**Frame (рамка)** — виджет, предназначенный для группировки других виджетов в заданной области фрейма.

**Radiobutton (радиокнопка)** — виджет-переключатель, который позволяет сделать выбор из нескольких вариантов.

**Checkbutton (флаговая кнопка, флажок)** — позволяет осуществлять множественный выбор.

**Listbox (список)** — область, в которой пользователь может выбрать требуемую запись (одну или несколько) из представленных в виде списка записей.

**Menu (меню)** — виджет, стандартное меню, которое может быть реализовано в виде выпадающего (popup) или ниспадающего (pulldown) представления.

**Menubutton (кнопка меню)** — кнопка, при нажатии на которую выводится ниспадающее меню.

**Canvas (рисунок)** — используется для вывода различных изображений, геометрических фигур и графиков.

**Scrollbar (полоса прокрутки)** служит для перемещения содержимого окна, если оно не умещается полностью. Бывает горизонтальной и вертикальной.

**Scale (шкала)** бывает горизонтальной и вертикальной. Служит для задания числового значения путем перемещения ползунка по шкале.

**Toplevel (окно верхнего уровня)** — отдельное (дополнительное) окно графического интерфейса, где также могут быть размещены различные виджеты.

#### **Упражнение 5.4**

Основываясь на GUI, представленном на рис. 5.2, предложите свой вариант интерфейса СППР (система поддержки принятия решений). Как можно задействовать радиокнопки?

### **5.2.2. Отслеживание событий**

Наличие и размещение виджетов в главном или дочернем окне — это просто макет интерфейса, который будет «неживым», если он не может идентифицировать действия, совершаемые пользователем. Именно поэтому в модуле tkinter предусмотрена возможность отслеживания событий пользователя, т.е. совершение им тех или иных действий. Все события в tkinter описываются в виде событийного шаблона — текстовой строки, содержащей модификаторы, тип события и детализацию события.

Основные события представлены в табл. 5.1.

События в tkinter

Событие	Описание события
Activate	Активизация выбранного окна
ButtonPress	Нажатие кнопки мыши (различают <Button-1>, <Button-2>, <Button-3> – левая, средняя и правая кнопки мыши соответственно)
Button Release	Отжатие соответствующей кнопки мыши
Deactivate	Деактивация активного окна
Destroy	Закрытие окна
Enter	Событие входа курсора в область заданного виджета
Focus In	Получение фокуса
Focus Out	Потеря фокуса
Key Press	Нажатие кнопки на клавиатуре <Название кнопки>
Key Release	Отжатие кнопки
Leave	Выход курсора мыши за виджет
Motion	Движение мыши в пределах области виджета
MouseWheel	Прокрутка колеса мыши
Reparent	Изменение окна-родителя
Vizable	Изменение видимости окна

В зависимости от действий пользователя программа может их обрабатывать.

### Упражнение 5.5

Основываясь на GUI, представленном на рис. 5.2, предложите, какие события должна отслеживать программа.

#### 5.2.3. Создание базового окна

Для создания базового окна, как, впрочем, и дочерних окон, используется следующая схема:

- 1) импорт библиотеки;
- 2) непосредственное создание окна;
- 3) создание виджетов и настройка их свойств;
- 4) идентификация событий;
- 5) идентификация обработчиков событий;
- 6) расположение виджетов в главном окне и упаковка;
- 7) отображение главного окна.

## Пример

Код:

```
from tkinter import * # импорт библиотеки
pr = Tk() # создание объекта tk
pr.title("Главное окно") # название окна
pr.geometry('500x200') # размер окна
pr.mainloop() # отображение окна
```

Результат показан на рис. 5.3.



Рис. 5.3. Создание базового окна

## Упражнение 5.6

Пользуясь предыдущим примером, создайте окно размером 300×300 пикселей с названием «Студенты».

*Задания для самостоятельного выполнения по теме  
«Инструменты для создания графических интерфейсов  
пользователя (GUI)»*

1. Создайте концептуальную модель GUI регистрации студента на образовательном портале. Определите размер главного окна.
2. Определите основные виджеты, которые должны быть размещены в главном окне.
3. Определите основные события, которые должен идентифицировать обработчик.
4. Какие действия должен совершить обработчик для событий, которые определены в предыдущем задании?
5. Создайте главное окно в соответствии с вашей концепцией из задания 1. Добавьте свойства окна следующими строчками `pr ["bg"] = "peachpuff"` и `pr.iconbitmap('star.ico')`.

**Справка.** Первая инструкция задает фон, вторая – иконку (файл с иконкой должен быть в папке с программой или необходимо прописать полный путь).

## 5.3. ЭЛЕМЕНТЫ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА (ВИДЖЕТЫ)

### 5.3.1. Создание и конфигурирование виджета. Менеджер размещения

Для создания и конфигурирования виджета используется объектная переменная.

Синтаксис:

```
переменная=имя_виджета(родительское_окно [, свойство1=значение1 [, свойство2=значение2 [...]]])
```

Иными словами, создается переменная — ссылка на виджет с определенными свойствами.

Далее вызывается менеджер размещения, который упаковывает виджет в окно.

Синтаксис:

```
переменная.pack()
```

Таким образом, для работы с виджетами используется объектно-ориентированный подход, когда к созданному объекту применяется метод, характерный для его класса.

После упаковки можно отображать главное окно с помощью метода `имя_окна.mainloop()`

В этом случае, если верно указаны параметры виджета, он будет размещен в главном окне.

Однако для размещения совсем необязательно упаковывать объект в окно, поскольку это делается автоматически в двух следующих случаях.

Например, когда размещение относительно главного окна задается непосредственно в свойствах виджета в абсолютных или относительных координатах. Для этого используется метод `place()`.

#### Пример

Фрагмент кода:

```
from tkinter import * # импорт библиотеки
...
but2=Button(pr,
             text="Решение с Deductor",
background="#555", foreground="#ccc",
             padx="20", pady="8", font="16", com-
mand='')
but2.place(relx=.3, rely=.1, anchor="c", height=30,
width=270, bordermode=OUTSIDE)
pr.mainloop() # отображение окна
```



Результат представлен на рис. 5.4.

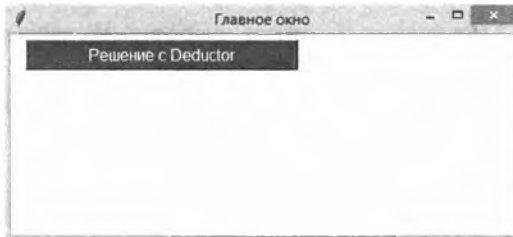


Рис. 5.4. Размещение виджета кнопки

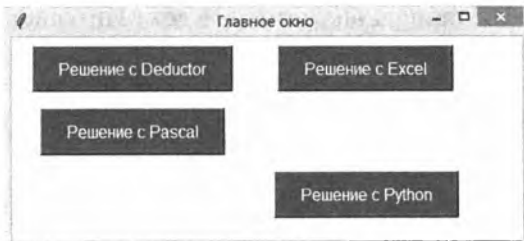
Как видно из примера, в качестве размещения был использованы относительные координаты `relx` и `rely`, которые задаются в долях 0.0 — левый верхний угол главного окна, а 1. — правый нижний угол. Также можно использовать абсолютные координаты `x` и `y`, задаваемые в пикселах. Опция `anchor` определяет привязку по розе ветров или направлениям компаса `n` (норд), `s` (зюйд), `w` (вест), `e` (ост), а также `c` (`center` — центр). Следующие параметры задают высоту и ширину виджета, последняя опция — границы виджета.

Еще один способ размещения — это размещение на двумерной сетке `grid`, где каждый виджет размещается в соответствующей ячейке сетки.

### Пример

Фрагмент кода:

```
from tkinter import * # импорт библиотеки
...
but2=Button(pr,
              text="Решение с Deductor",
background="#555", foreground="#ccc",
              padx="20", pady="8", font="16", com-
mand='')
...
but1.grid(row=0, column=0, padx="20", pady="8")
but2.grid(row=0, column=1, padx="20", pady="8")
but3.grid(row=1, column=0, padx="20", pady="8")
but4.grid(row=3, column=1, padx="20", pady="8")
pr.mainloop() # отображение окна
Результат показан на рис. 5.5.
```



**Рис. 5.5.** Размещение виджетов кнопок с помощью метода `grid()`

Как видно из примера, данный метод разбивает окно на части одинаковых размеров, считая их общее количество. При этом если не задать размер кнопки, то он будет автоматически определен в соответствии с размером текста.

### **Упражнение 5.7**

Пользуясь предыдущим примером, создайте окно с названием «Студенты», где разместите кнопки «Вывести список», «Ввести данные», «Создать запрос», «Выйти» тремя рассмотренными способами. Сравните результат и прокомментируйте преимущества и недостатки использования данных методов.

### **5.3.2. Использование элементов `Button` (Кнопка)**

В предыдущем подпараграфе для демонстрации работы с главным окном и менеджером размещения виджетов был использован виджет `Button` (кнопка), который предназначен для того, чтобы при его нажатии вызывалась функция или метод. Например, при нажатии на кнопку «Вывести список» должен быть сформирован список студентов.

Синтаксис:

```
переменная=Button(родительское_окно [, свойство1=значение1 [, свойство2=значение2 [...]]])
```

Основные опции виджета представлены ниже:

- `text` — задает надпись на кнопке;
- `width, height` — ширина и высота виджета;
- `background` — цвет фона (можно использовать сокращение — `bg`);
- `foreground` — цвет надписи (также допустимо — `fg`);
- `font` ('Тип', Размер, 'стиль') — параметры шрифта;
- `padx, pady` — расстояние между содержимым и границами виджета.

Подробнее об опциях виджетов — в сети Интернет<sup>1</sup>.

### Упражнение 5.7

Введите код программы, представленный на рис. 5.6.

Запустите программу и прокомментируйте результат ее выполнения.

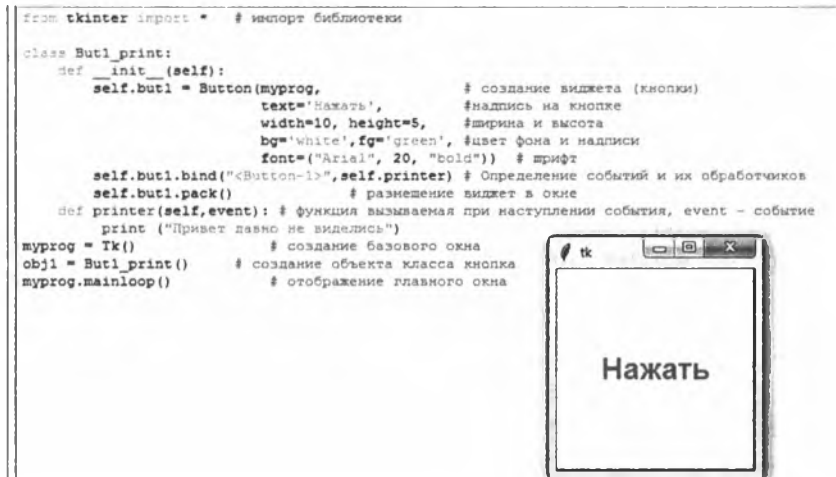


Рис. 5.6. Использование элемента «Кнопка»

В данном упражнении реализован объектно-ориентированный подход, в котором объект создается на базе класса с использованием конструктора объектов `__init__()`, который был подробно рассмотрен в главе 4.

### 5.3.3. Label (Надпись)

Виджет *Label* (надпись, метка) используется для информирования пользователя о чем-либо. Например, метка для ввода логина или пароля.

Синтаксис:

```
переменная=Label(родительское_окно [, свойство1=значение1 [, свойство2=значение2 [...]])
```

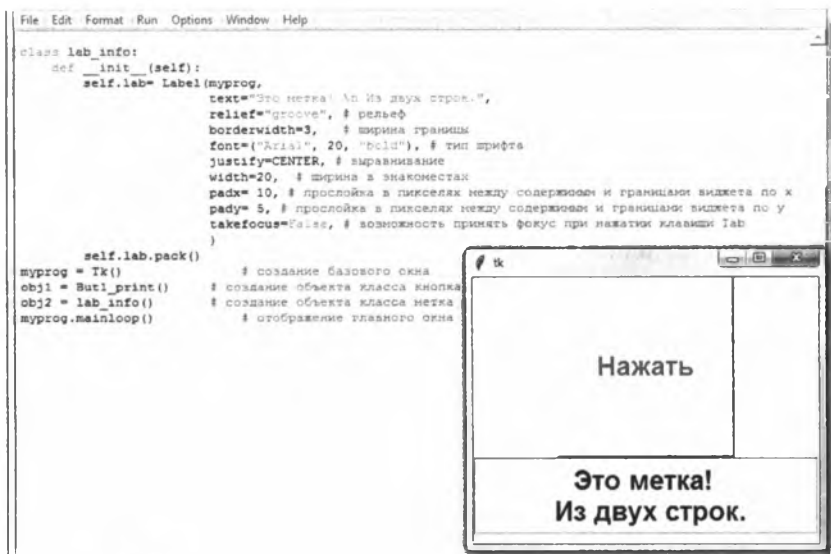
Основные опции данного виджета аналогичны виджету *Button*. Подробнее о всех возможных опциях и применяемых методах можно узнать на том же сайте.

### Упражнение 5.8

Введите код программы, представленный на рис. 5.7.

Запустите программу и прокомментируйте результат ее выполнения.

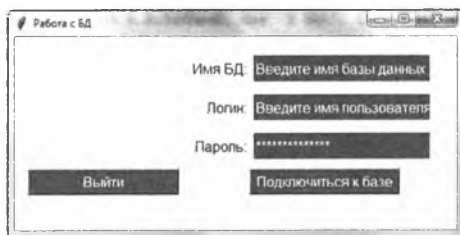
<sup>1</sup> URL: <http://russianlutheran.org/python/life/life.htm>



 **Рис. 5.7.** Использование элемента «Надпись (метка)»

### Упражнение 5.9

Создайте программу, которая реализует виджет Label в графическом интерфейсе, представленном на рис. 5.8.



**Рис. 5.8.** Графический интерфейс

В последнем упражнении для ввода данных используется виджет Entry, который будет рассмотрен далее.

### 5.3.4. Entry (Поле ввода)

*Entry* (поле ввода) — это виджет, позволяющий считывать данные, введенные пользователем, и передавать их управляющей подпрограмме или функции, что обеспечивает взаимодействие пользователя и программы в форме диалога.

Синтаксис:

```

переменная=Entry(родительское_окно [, свойство1=
значение1 [, свойство2=значение2 [...]])

```

Особое внимание следует уделить именно передачи переменной.

Сначала в основном блоке программы создается текстовая переменная, являющаяся объектом `StringVar()` из стандартного модуля `string`. Поэтому в самом начале программы необходимо его подключить с помощью строки

```
import string
```

Далее в виджете `Entry` необходимо ввести следующий аргумент и присвоить ему значение этой переменной. Для получения значения введенной переменной используется следующая конструкция:

```
переменная=str(объект_типа_Entry.ent.get())
```

### Пример

Фрагмент кода:

```
import string
```

```
...
```

```
tv=stringVar()
```

```
obj1=Entry(..., textvariable=tv, ...)
```

```
...
```

```
tv.set('Введите текст') # задает начальный текст
```

```
...
```

```
dbl=str(obj1.ent.get()) # присваивает переменной значение из поля ввода объекта obj1 и переводит его в строковый формат
```

```
...
```

Пример реализации виджета представлен на рис. 5.9.

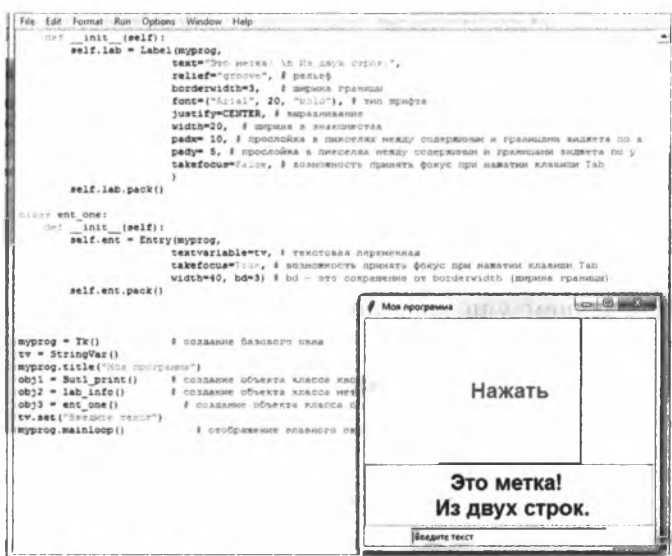


Рис. 5.9. Использование элемента «Поле ввода»

## Упражнение 5.10

Разместите виджеты Label, Button и Entry, как показано на рис. 5.8.

**Справка.** Для того чтобы в поле ввода отображалась «\*» или какой-либо другой символ, в виджете Entry необходимо задать аргумент show='заполнитель'. В данном случае в качестве заполнителя использован символ «\*».

### 5.3.5. Checkbutton (Флажок)

*Checkbutton* (Флажок) — данный виджет позволяет сделать множественный выбор.

## Упражнение 5.11

Создайте программу, в которую включите следующий код, и сравните его с полученным результатом, представленным на рис. 5.10. Объясните назначение следующих опций: justify, variable, onvalue, offvalue; объектной переменной IntVar(); методов c1.set (0), c2.set (0), c3.set (0). Что будет, если для всех последних методов вместо 0 поставить 1?

```
File Edit Format Run Options Window Help
wrap=WORD) # тип переноса слов
self.tex.pack()

class rad_but:
    def __init__(self):
        rv = IntVar() # переменная для выбора радиокнопки
        rv.set(1)
        self.rad0 = Radiobutton(myprog, text="Решить квадратное уравнение",
                                variable=rv, value=0)
        self.rad1 = Radiobutton(myprog, text="Вычислить выражение",
                                variable=rv, value=1)
        self.rad2 = Radiobutton(myprog, text="Вычислить определитель",
                                variable=rv, value=2)

        self.rad0.pack()
        self.rad1.pack()
        self.rad2.pack()

class check_but:
    def __init__(self):
        c1=IntVar()
        c2=IntVar()
        c3=IntVar()
        c1.set(0)
        c2.set(0)
        c3.set(0)
        self.che1 = Checkbutton(myprog, text="Вывести результат", justify=LEFT,
                                variable=c1, onvalue=1, offvalue=0)
        self.che2 = Checkbutton(myprog, text="Показать решение", justify=LEFT,
                                variable=c2, onvalue=2, offvalue=0)
        self.che3 = Checkbutton(myprog, text="Сделать вывод ", justify=LEFT,
                                variable=c3, onvalue=3, offvalue=0)
        self.che1.pack()
        self.che2.pack()
        self.che3.pack()
```

Рис. 5.10. Использование элемента «Флажок (Множественный выбор)»

### 5.3.6. Radiobutton (Переключатель)

*Radiobutton* (Переключатель), или радиокнопка, позволяет делать единственный выбор из представленных вариантов.

#### Упражнение 5.12

Создайте программу, в которую включите следующий код и сравните его с полученным результатом, представленным на рис. 5.11.

**Справка.** Представленная группа переключателей определяет значение одной переменной. Если в группе будет выбрано значение 2, то переменной *rv* (*radiovariable*) будет передано значение 2. В зависимости от значения переданной переменной можно организовать ветвление алгоритма в основном блоке программы. Метод *rv.set()* устанавливает первоначальное значение переменной. При этом нумерация начинается с 0. В данном скриншоте первоначальное значение установлено для среднего варианта *rv.set (1)*.

```
File Edit Format Run Options Window Help
class tex_one:
    def __init__(self):
        self.tex = Text(myprog,
            takefocus=True, # возможность принять фокус при нажатии клавиши Tab
            width=40, height=10, bd=3, # bd - это сокращение от borderwidth (ширина гра
            font='Arial 14', # шрифт
            wrap=WORD) # тип переноса слов
        self.tex.pack()

class rad_but:
    def __init__(self):
        self.rad0 = Radiobutton(myprog, text="Решить квадратное уравнение",
            variable=rv, value=0)
        self.rad1 = Radiobutton(myprog, text="Вычислить выражение",
            variable=rv, value=1)
        self.rad2 = Radiobutton(myprog, text="Вычислить определитель",
            variable=rv, value=2)

        self.rad0.pack()
        self.rad1.pack()
        self.rad2.pack()

myprog = Tk() # создание базового окна
tv = StringVar() # переменная для однострочного текста
rv = IntVar() # переменная для выбора радиокнопки
myprog.title("Моя программа")
obj1 = But1_print() # создание объекта класса кнопка
obj2 = lab_info() # создание объекта класса метка
obj3 = ent_one() # создание объекта класса однострочный текст
obj4 = tex_one() # создание объекта класса многострочный текст
obj5 = rad_but() # создание объекта класса радиокнопка
rv.set("Введите текст")
rv.set(1)
myprog.mainloop() # отображение главного окна
```

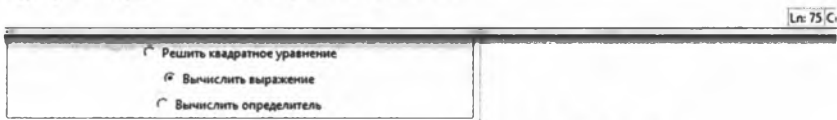


Рис. 5.11. Использование элемента Radiobutton

### 5.3.7. Другие виджеты

Далее приведены примеры использования других виджетов и упражнения для получения практических навыков их применения.

**Text** — многострочное текстовое поле.

Синтаксис:

```
переменная=Text(родительское_окно [, свойство1=значение1 [, свойство2=значение2 [...]])
```

Пример кода и его реализация представлены на рис. 5.12.



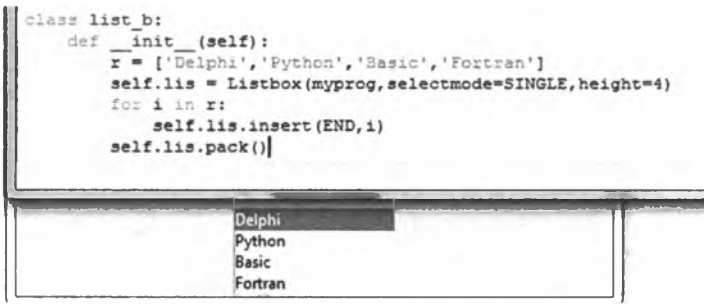
Рис. 5.12. Использование элемента Text

*Listbox* (список) позволяет из представленных записей выбрать тот или иной вариант.

#### Упражнение 5.13

Создайте программу, в которую включите следующий код, и сравните его с полученным результатом, представленным на рис. 5.13. Прокомментируйте назначение соответствующих опций (аргументов).





**Рис. 5.13.** Использование элемента Listbox

*Frame* (рамки) используются для группировки тех или иных виджетов (рис. 5.14). В данном примере создаются три фрейма различного размера и цвета. В один из них (второй фрейм) размещается виджет Entry (поле ввода), в котором можно ввести необходимый текст. Здесь используется аргумент (свойство) *bd* (*borderwidth*), которое задает расстояние от границы рамки до размещенного в нем виджета.



**Рис. 5.14.** Использование элемента Frame (ООП)

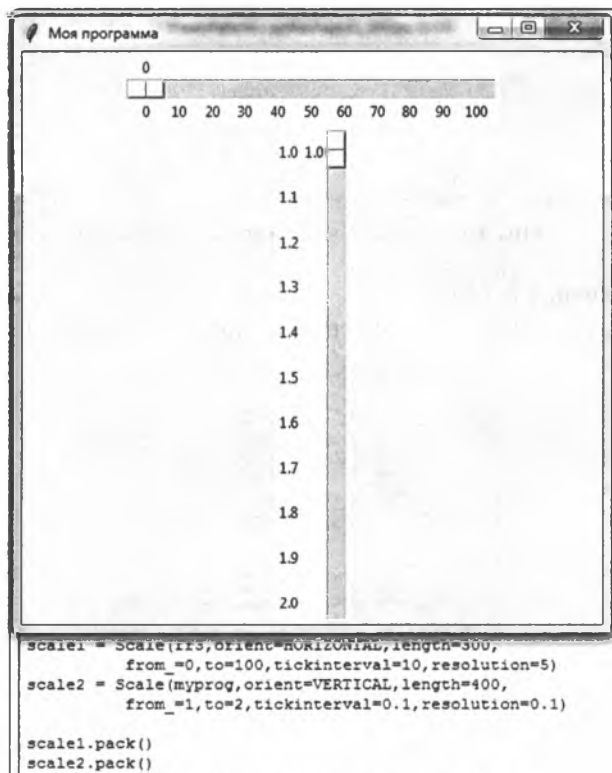
Предыдущий пример демонстрирует использование ООП. Следующий пример — это операторная реализация, причем результат оказывается тем же (рис. 5.15).

```
myprog = Tk() # создание базового окна
myprog.title("Моя программа") # Название окна
myprog["bg"]="peachpuff"

fr1 = Frame(myprog,width=500,height=100,bg="darkred")
fr2 = Frame(myprog,width=300,height=200,bg="green",bd=20)
fr3 = Frame(myprog,width=500,height=150,bg="darkblue")
fr1.pack()
fr2.pack()
fr3.pack()
entri = Entry(fr2,width=20)
entri.pack()
```

**Рис. 5.15.** Использование элемента Frame

*Scale* (шкала) — рис. 5.16.



**Рис. 5.16.** Использование элемента Scale

Свойства:

- `orient` — определяет, является ли шкала горизонтальной или вертикальной;
- `length` — длина шкалы, определяемая в пикселах;
- `from_to` (от и до) — позволяет задать начальное и конечное значение шкалы;
- `resolution` (разрешение) — задает шаг перемещения ползунка.

**ScrollBar** (полоса прокрутки) — виджет, позволяющий перемещать отображение текста, который не умещается в размере заданного текстового поля (рис. 5.17).

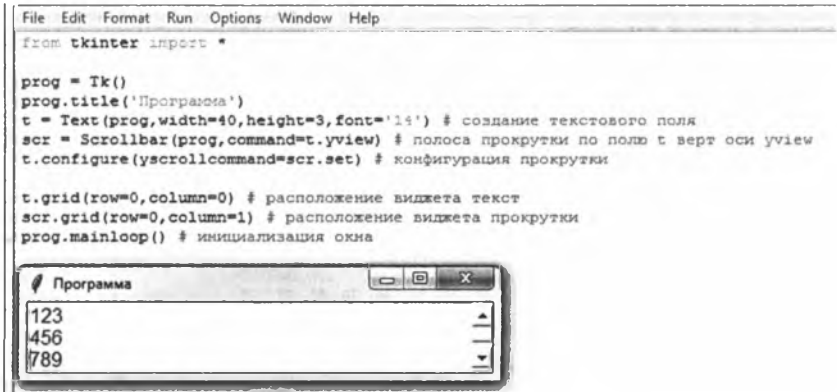


Рис. 5.17. Использование элемента ScrollBar

### Упражнение 5.14

Создайте графический интерфейс, представленный на рис. 5.18.

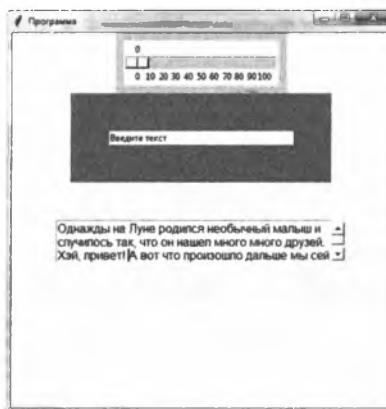
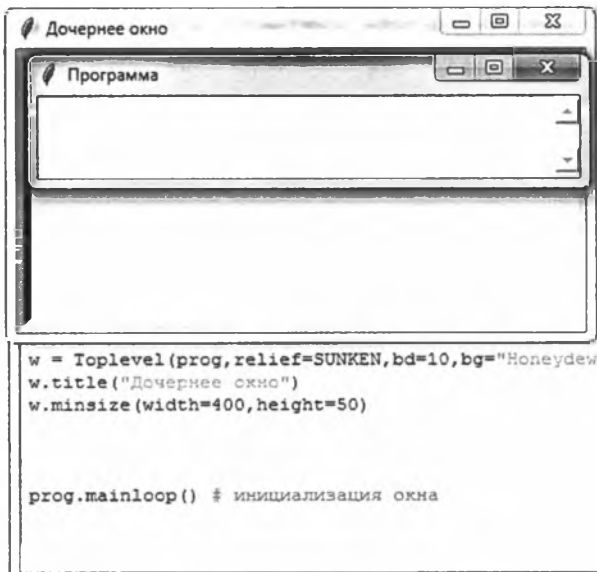


Рис. 5.18. Графический интерфейс

**Справка.** Первоначальный размер окна можно задать как `prog.minsize (500,500)` либо `prog.geometry ('500x500')`, цвет `prog [«bg»] =>цвет». Для того чтобы фрейм не удалялся, необходимо в его опции добавить bd=10 (граница).`

*Toplevel Window* (дочернее окно) — окно верхнего уровня, вызываемое для перехода пользователя для работы в новом окне (рис. 5.19).



**Рис. 5.19.** Использование элемента `Toplevel`

### Упражнение 5.15

Используя пример на рис. 5.19, создайте приложение, состоящее из главного и двух дочерних окон. На каждом из трех окон должны располагаться один или два любых графических объекта.

*Menu* (меню) — позволяет сделать стандартное выпадающее меню, которое размещается в верхней части окна.

### Упражнение 5.16

Введите следующий код программы:

```
from tkinter import * # импорт библиотеки
from tkinter.filedialog import *
import string
import fileinput
pr = Tk()
```

```

pr.title('Пример с меню')
def new_win():
    win = Toplevel(pr)

def close_win():
    pr.destroy()

def about():
    win = Toplevel(pr)
    lab = Label(win, text="Это просто программа - пример
\n с меню в Tkinter")
    lab.pack()

def _open():
    op = askopenfilename()
    for l in fileinput.input(op):
        txt.insert(END, l)

def _save():
    sa = asksaveasfilename()
    letter = txt.get(1.0, END)
    f = open(sa, "w")
    f.write(letter)
    f.close()

men = Menu(pr) #создается объект Меню на главном окне
pr.config(menu=men) #окно конфигурируется с указанием
меню для него

fmen = Menu(men) #создается пункт меню с размещением
на основном меню(men)
men.add_cascade(label="Файл", menu=fmen) #пункту
располагается на основном меню(men)
fmen.add_command(label="Новый", command=new_win)
#формируется список команд пункта меню
fmen.add_command(label="Открыть", command=_open)
fmen.add_command(label="Сохранить", command=_save)
fmen.add_command(label="Выход", command=close_win)

smen = Menu(men) #второй пункт меню
men.add_cascade(label="Помощь", menu=smen)

```

```

smen.add_command(label="Помощь F1")
smen.add_command(label="О программе", command=about)
nfmén = Menu(fmén)
fmén.add_cascade(label="Импорт", menu=nfmén)
nfmén.add_command(label="Картинка")
nfmén.add_command(label="Текст")

txt = Text(pr, width=40, height=15, font="12")
txt.pack()
pr.mainloop()

```

Запустите ее и посмотрите на результат выполнения. Прокомментируйте используемые аргументы. Что позволяет сделать аргумент `command`?

### Упражнение 5.17

Пользуясь результатами предыдущего упражнения, добавьте условие: после нажатия пункта `Exit` пользователю выводится окно с вопросом «сохранить или нет». В случае положительного ответа должна вызываться функция `_save` и только затем — завершаться программа.

Если в текстовом поле что-то содержится и при открытии файла не удаляется, а содержимое файла просто дописывается, то исправьте этот недостаток (перед открытием файла содержимое текстового поля должно удаляться).

### 5.3.8. Метод `bind`

Метод `bind` является одним из ключевых методов организации диалога пользователя и программы через графический интерфейс. Он позволяет обрабатывать события, совершенные пользователем, с вызовом соответствующего обработчика — функции или метода.

Синтаксис:

```
имя_виджета.bind(event, function)
```

`event` — событие, которое совершает пользователь; `function` — функция, которая будет вызвана при появлении события. При этом при объявлении функции в скобках должно быть указано ключевое слово `event`. В этом случае функция будет обрабатывать событие.

### Упражнение 5.18

Напишите код, представленный на рис. 5.20. Протестируйте программу и прокомментируйте результат ее выполнения.

```

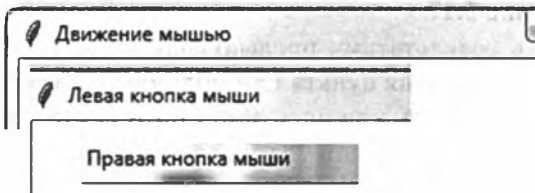
import string
def b1(event):
    pr.title("Левая кнопка мыши")
def b3(event):
    pr.title("Правая кнопка мыши")
def move(event):
    pr.title("Движение мышью")

pr = Tk()
pr.minsize(width = 400, height=100)

pr.bind('<Button-1>',b1)
pr.bind('<Button-3>',b3)
pr.bind('<Motion>',move)

pr.mainloop()

```



**Рис. 5.20.** Использование метода bind

Для каждого события создается своя функция-обработчик.

Описание событий, обрабатываемых tkinter, представлено в соответствующей теме.

Буквенные клавиши записываются без квадратных скобок (например 'A'). Также существуют зарезервированные слова <Return> – клавиша ввода, <Escape> – отмена, <Space> – пробел, <Control-Shift> – сочетание клавиш и ряд других.

На рис. 5.21 представлен код программы, который меняет цвет фрейма при нажатии пользователем клавиши ввода, а при нажатии левой кнопки мыши (ЛКМ) на кнопке «Выход» закрывает окно. Как видно из рисунка, созданы две функции, первая из которых обрабатывает клавишу «Ввод», а вторая – «ЛКМ». Для обработки смены цвета используется список, содержащий их название. При нажатии <Return> происходит смена соответствующего цвета. При этом используется конфигурактор, определяющий первоначальный цвет, configure(bg='цвет') . Здесь опция bg задает цвет фона (background).

```

from tkinter import * # импорт библиотеки
import string
li = ["red", "green"]
def color(event):
    fra.configure(bg=li[0])
    li[0], li[1] = li[1], li[0]

def outgo(event):
    pr.destroy()

pr = Tk()

fra = Frame(pr, width=100, height=100, bd=20)
but = Button(pr, text="Выход")

fra.pack()
but.pack()

pr.bind("<Return>", color)
but.bind("<Button-1>", outgo)

pr.mainloop()

```



 **Рис. 5.21.** Использование метода bind

Таким образом можно обеспечить связь действий пользователя и работы приложения (программы).

### 5.3.9. Canvas (изображение)

*Canvas* (изображение, холст) — это виджет, который позволяет создавать изображения, рисовать графики и т.п. Ввиду его важности его рассмотрение выделено в отдельный подпараграф.

Синтаксис:

```
переменная=Canvas(родительское_окно [, свойство1=
значение1 [, свойство2=значение2 [...]])
```

После создания области холста можно использовать следующие методы.

Создание примитива:

```
имя_объекта.create_имя_примитива(имя_примитива [,
свойство1=значение1 [, свойство2=значение2 [...]])
```

Перемещение примитива:

```
имя_объекта.move(имя_примитива [свойство1=значение1 [,
свойство2=значение2 [...]])
```



**Изменение свойств примитива:**

```
имя_объекта.itemconfig(имя_примитива [свойство1=
значение1 [, свойство2=значение2 [...]])
```

**Изменение координат примитива:**

```
имя_объекта.coord(имя_примитива [свойство1=значение1
[, свойство2=значение2 [...]])
```

**Удаление примитива:**

```
имя_объекта.delete(имя_примитива)
```

### **Упражнение 5.19**

Введите следующий код программы. Запустите ее и посмотрите на результат выполнения. Прокомментируйте используемые аргументы:

```
from tkinter import *
import time
import datetime
import threading
import turtle
myprog=Tk()
def moove(event):
    can.move(rect,0,150) # модификатор
    can.itemconfig(trian, outline="red", width=3) #
модификатор
    can.coords(oval,300,200,450,450) # модификатор
def color(event):
    can.itemconfig('group1', fill="yellow", width=5)
def clean(event):
    can.delete(oval)
def animate(event):
    x=5
    y=5
    i=0
    tr=trian
    while i<10:
        can.delete(tr)
        for j in range(2):
            time.sleep(0.25)
            tr=can.create_polygon(330+x,80+y,380+x,10+y,
430+x,80+y, fill='grey80', outline="black")
            x+=10
            y+=10
            i+=1
```

```

can = Canvas(width=500, height=460, bg='Honeydew') #
идентификатор
can.pack() # размещение
oval = can.create_oval(30,10,130,80, tag='group1') #
идентификатор
rect = can.create_rectangle(180,10,280,80,
tag='group1') # идентификатор
trian = can.create_polygon(330,80,380,10,430,80,
fill='grey80', outline="black")
# идентификатор
can.bind('<Button-1>', moove) # реакция на событие
can.bind('<Button-3>', color) # реакция на событие
can.bind('<Double Button-1>', clean) # реакция на
событие
can.bind('<Button-2>', animate) # реакция на событие
myprog.mainloop() #инициация

```

Следующее упражнение позволит вам приобрести навыки работы при построении графиков.

### Упражнение 5.20

Напишите код, представленный на рис. 5.22. Протестируйте программу и прокомментируйте результат ее выполнения. Как формируется график? Как задать тип и координаты линии? Как нарисовать оси? Что позволяет осуществить метод `append()`?

### Упражнение 5.21

Постройте следующие графики. Диапазон выберите произвольно. Здесь  $n$  — переменная:

$$1 + \sin^2(n) \cdot 3^3 + \arctg(8 \cdot \pi / 3);$$

$$\frac{\ln^2(n^3)}{1 + n^2} \bmod(2n) + 4 \cos^2(n) + \text{arccctg}(9 \cdot \pi / 2).$$

### Задания для самостоятельного выполнения по теме «Элементы графического интерфейса (виджеты)»

1. Создайте программу, которая бы формировала окно (рис. 5.23). Используйте объектно-ориентированный стиль. Таблицу цветовых кодов можно найти в открытом доступе<sup>1</sup>.
2. Создайте программу, которая бы формировала следующее окно (рис. 5.24). После ввода цифры и нажатия на кнопку «вывести» в соответствующем окне должна отобразиться справочная информация.

<sup>1</sup> URL: <http://www.realcoding.net/article/view/530>

```

File Edit Format Run Options Window Help
# -*- coding: utf-8 -*-
from tkinter import *
import math
#
tk=Tk()
tk.title("Graph")
#
button=Button(tk,text='Закрыть', command=quit)
button.pack()
#
canvas=Canvas(tk)
canvas["height"]=360
canvas["width"]=480
canvas["background"]="#eeeeff"
canvas["borderwidth"]=2
canvas.pack()
#
canvas.create_text(20,10,text="20,10")
canvas.create_text(460,350,text="460,350")
#
points=[]
ay=150
y0=150
x0=50
x1=470
dx=10
#
for n in range(x0,x1,dx):
    y=y0-ay*math.cos(n*dx)
    pp=(n,y)
    points.append(pp)
#
canvas.create_line(points,fill="blue",smooth=1)
#
y_ave=[]
yy=(x0,0)
y_ave.append(yy)
yy=(x0,y0+ay)
y_ave.append(yy)
canvas.create_line(y_ave,fill="black",width=2)
#
x_ave=[]
xx=(x0,y0)
x_ave.append(xx)
xx=(x1,y0)
x_ave.append(xx)
canvas.create_line(x_ave,fill="black",width=2)
#
tk.mainloop()

```

**Рис. 5.22.** Построение графика

Моя программа

**Ваш адрес:**

Введите адрес доставки

**Комментарий:**

**Отправить**

Сколько штук?

0-10

11-20

21-30

31-40

Какого цвета?

Красный(Red)

Синий(Blue)

Зеленый(Green)

Желтый(Yellow)

☁ **Рис. 5.23.** Графический интерфейс

**Вывести**

**Вывести** Введите 1 или 2 в поле слева

**Вывести** Обслуживание клиентов на втором этаже

**Вывести** Пластиковые карты выдают в соседнем здании

**Рис. 5.24.** Графический интерфейс

3. Создайте приложение, в котором меняется размер фрейма в зависимости от того, какая из трех объектов-кнопок была нажата.
4. Создайте программу с меню, содержащим два пункта: Цвет и Размер. Пункт «Цвет» должен содержать три команды (Красный, Синий и Зеленый), меняющие цвет рамки на главном окне. Пункт «Размер» должен содержать две команды (500×500 и 700×400), изменяющие размер рамки.
5. Разработайте программу с графическим интерфейсом, которая формирует список студентов, выводит размер их стипендии, считает среднее значение стипендии и средний балл за три дисциплины. Должно быть три группы разных направлений, не менее пяти студентов в каждой группе. Должна быть реализована выборка по группам, дисциплине и наличию либо отсутствию стипендии.

### *Контрольные вопросы и задания*

1. Что такое событийно-ориентированное программирование?
2. Дайте определение понятий «событие», «обработчик событий» и «цикл обработки событий».
3. Перечислите основные инструменты для создания графического интерфейса.
4. Опишите алгоритм построения интерфейса на базе главного окна и способы размещения виджетов.
5. Каковы синтаксис создания главного окна и его основные атрибуты?
6. Опишите особенности построения текстовых виджетов.
7. Каковы особенности создания управляющих кнопок?
8. Как создать меню?
9. Как реализовать связывание событий, инициированных пользователем, с обработчиком этих событий? Опишите синтаксис соответствующего метода.
10. Опишите особенности построения виджета canvas и работу с основными графическими примитивами.

## Библиографический список

1. *Берри П.* Изучаем программирование на Python [Текст] / П. Берри. — М.: Эксмо, 2017. — 624 с.
2. *Могилев А.В.* Методы программирования. Компьютерные вычисления [Текст] / А.В. Могилев, Л.В. Листрова. — СПб.: БХВ-Петербург, 2012. — 320 с.
3. *Плещинский Н.Б.* Языки программирования и методы трансляции (конспект лекций для студентов 3-го курса ИВМиИТ КФУ, весна 2013) [Электронный ресурс] / Н.Б. Плещинский. — URL: <http://www.abcrnb.ru/LEC/PLTM01.pdf>
4. *Ромальо Л.* Python. К вершинам мастерства [Текст] / Л. Ромальо; пер. с англ. А.А. Слинкина. — М.: ДМК-Пресс, 2016. — 768 с.
5. *Россум Г.* Язык программирования Python [Электронный ресурс] / Г. Россум, Ф.Л.Дж. Дрейк, Д.С. Откидач. — URL: <http://prosmart.by/programming/895-yazyk-programmirovaniya-python-rossum-g-drejk.html>
6. *Светозарова Г.И.* Практикум по программированию на алгоритмических языках [Текст] / Г.И. Светозарова, Е.В. Сигитов, А.В. Козловский. — М.: Наука, 1980. — 320 с.
7. *Слаткин Б.* Секреты Python. 59 рекомендаций по написанию эффективного кода [Текст] / Б. Слаткин. — М.: Вильямс, 2017. — 272 с.
8. *Сузи Р.А.* Python [Текст] / Р.А. Сузи. — СПб.: БХВ-Петербург, 2006. — 768 с.
9. *Хеллман Д.* Стандартная библиотека Python 3. Справочник с примерами [Текст] / Д. Хеллман. — М.: Вильямс, 2018. — 1376 с.
10. *Яковлев С.* Программирование на Python. Ч. 2 [Электронный ресурс] / С. Яковлев. — URL: [https://www.ibm.com/developerworks/ru/library/l-python\\_part\\_2/1-python\\_part\\_2-pdf.pdf](https://www.ibm.com/developerworks/ru/library/l-python_part_2/1-python_part_2-pdf.pdf)

### Интернет-источники

1. Официальный сайт Python. URL: <http://www.python.org>
2. Образовательный портал Pythonicway. URL: <http://pythonicway.com>
3. Python 3 для начинающих. URL: <https://pythonworld.ru/moduli/modul-math.html>

## Перечень тем для проверки знаний по дисциплине

1. История развития языков программирования.
2. Понятие языка программирования. Синтаксис и семантика языка. Способы реализации языков: компиляция, интерпретация, смешанный подход.
3. Язык программирования Python и его место среди других языков программирования. Установка Python.
4. Работа в интерактивном режиме интерпретатора. Среда программирования. Использование документации.
5. Работа с числами. Базовые числовые типы `int` и `float`. Числовые литералы. Операторы для работы с числовыми объектами. Форматы чисел. Встроенные функции и модули для работы с числами. Преобразование и смешивание в выражениях значений разных типов.
6. Строки. Литералы строк. Типы `str`, `bytes`, `bytearray`. Операции над строками: конкатенация, повторение, доступ по индексу, получение подстроки, проверка вхождения.
7. Форматирование строк. Функции и методы для работы со строками. Регулярные выражения.
8. Списки. Создание списка. Операции над списками. Перебор элементов списка. Многомерные списки. Методы списков. Кортежи.
9. Словари. Создание словаря. Операции над словарями. Перебор элементов словаря. Методы для работы со словарями. Множества.
10. Переменные. Правила именования переменных. Присваивание значения переменным. Динамическая типизация.
11. Понятие о счетчике ссылок и сборке мусора. Проверка и преобразование типов данных. Удаление переменной.
12. Структура программы. Комментарии. Блок. Правила оформления отступов.
13. Инструкция присваивания. Групповое присваивание. Комбинированные инструкции присваивания. Инструкции выражений. Функция `print`.
14. Операторы сравнения. Логические операторы. Инструкция ветвления `if ... else`. Инструкция цикла `while`. Инструкция цикла `for`.
15. Инструкции `break`, `continue`, `pass`, `else`.
16. Функции. Создание функций. Область видимости переменной. Передача аргументов в функцию. Инструкция `return`. Вызов функции.
17. Лямбда-функции. Функции-генераторы. Инструкция `yield`.
18. Модули. Инструкции `import` и `from`. Обзор стандартной библиотеки Python.
19. Базовые принципы объектно-ориентированного программирования: инкапсуляция, наследование, полиморфизм.

20. Классы в языке Python. Инструкция class. Создание экземпляра класса. Атрибуты класса и экземпляра класса. Методы класса.
21. Python. Конструктор и деструктор. Статические методы. Закрытые методы и атрибуты. Свойства. Наследование. Множественное наследование. Перегрузка операторов.
22. Событийно-ориентированное программирование. Событие. Обработчик события. Цикл обработки событий.
23. Инструменты для создания графических интерфейсов пользователя (GUI).
24. Создание базового окна. Элементы графического интерфейса (виджеты).
25. Создание и конфигурирование виджета. Менеджер размещения.
26. Использование элементов «Кнопка», «Надпись», «Текстовое поле», «Флажок», «Переключатель».
27. Связывание виджетов по событиям с функциями.
28. Создание графических объектов. Поле Canvas. Редактирование объектов. Перемещение, изменение размера, удаление.



# Оглавление

<b>Предисловие</b> .....	<b>3</b>
<b>Глава 1. Языки программирования</b> .....	<b>5</b>
1.1. Понятие языка программирования .....	5
1.2. Развитие языков программирования .....	6
1.3. Классификация языков программирования.....	29
1.4. Синтаксис и семантика языка. Общие конструкции .....	32
1.5. Способы реализации языков: компиляция, интерпретация, смешанный подход ...	35
1.6. Основные понятия и определения в программировании.....	36
1.7. Язык программирования Python и его место среди других языков программирования .....	39
1.8. Установка Python.....	40
1.9. Работа в интерактивном режиме интерпретатора .....	43
1.10. Среда программирования. Использование документации .....	47
<i>Контрольные вопросы и задания</i> .....	49
<b>Глава 2. Типы данных и операции языка Python</b> .....	<b>51</b>
2.1. Работа с числами.....	51
2.1.1. Общие сведения.....	51
2.1.2. Базовые числовые типы int и float.....	52
2.1.3. Числовые литералы.....	53
2.1.4. Операторы для работы с числовыми объектами .....	54
2.1.5. Форматы чисел.....	55
2.1.6. Встроенные функции и модули для работы с числами .....	58
2.1.7. Преобразование и смешивание в выражениях значений разных типов .....	61
<i>Задания для самостоятельного выполнения по теме «Работа с числами»</i> .....	62
2.2. Последовательности. Работа со строками.....	63
2.2.1. Строки. Литералы строк. Специальные символы .....	63
2.2.2. Операции над строками.....	65
2.2.3. Функции и методы для работы со строками.....	67
2.2.4. Форматирование строк .....	70
2.2.5. Регулярные выражения .....	73
<i>Задания для самостоятельного выполнения по теме «Последовательности. Работа со строками»</i> .....	76
2.3. Последовательности. Списки .....	77
2.3.1. Создание списка.....	77
2.3.2. Генераторы списков.....	78
2.3.3. Создание копии списка, полная и поверхностная копии списка .....	79
2.3.4. Операции над списками.....	79
2.3.5. Методы списков .....	83
2.3.6. Многомерные списки .....	84
<i>Задания для самостоятельного выполнения по теме «Последовательности. Списки»</i> .....	86
2.4. Кортежи .....	86
<i>Задания для самостоятельного выполнения по теме «Кортежи»</i> .....	87

2.5. Словари.....	87
2.5.1. Создание словаря.....	87
2.5.2. Операции над словарями.....	88
2.5.3. Методы для работы со словарями.....	90
<i>Задания для самостоятельного выполнения по теме «Словари»</i> .....	92
2.6. Множества.....	92
2.6.1. Создание множества.....	92
2.6.2. Операции над множествами.....	93
2.6.3. Методы для работы с множествами.....	95
<i>Задания для самостоятельного выполнения по теме «Множества»</i> .....	97
2.7. Работа с датой и временем.....	97
2.7.1. Получение текущей даты и времени.....	98
2.7.2. Форматирование даты и времени.....	99
2.7.3. Модули datetime и calendar.....	100
<i>Задания для самостоятельного выполнения по теме «Работа с датой и временем»</i> ...	103
<i>Контрольные вопросы и задания</i> .....	103
<b>Глава 3. Инструкции, функции, модули в языке Python</b> .....	<b>104</b>
3.1. Переменные.....	104
3.1.1. Правила именования переменных.....	104
3.1.2. Присваивание значений переменным.....	105
3.1.3. Динамическая типизация.....	106
3.1.4. Понятие о счетчике ссылок и сборке мусора.....	107
3.1.5. Проверка и преобразование типов данных. Удаление переменных.....	108
<i>Задания для самостоятельного выполнения по теме «Переменные»</i> .....	110
3.2. Программа. Свойства и особенности построения.....	110
3.2.1. Структура программы.....	111
3.2.2. Комментарии.....	112
3.2.3. Блок. Правила оформления отступов.....	113
<i>Задания для самостоятельного выполнения по теме «Программа. Свойства и особенности построения»</i> .....	115
3.3. Инструкции.....	115
3.3.1. Инструкция присваивания.....	115
3.3.2. Ввод и вывод данных. Функции input() и print().....	118
3.3.3. Операторы сравнения.....	120
3.3.4. Логические операторы and, or, not.....	120
3.3.5. Инструкция ветвления if ... else. Проверка нескольких условий.....	121
3.3.6. Инструкция цикла while.....	123
3.3.7. Инструкция цикла for. Функция range.....	124
3.3.8. Инструкции break, continue, pass.....	126
<i>Задания для самостоятельного выполнения по теме «Инструкции»</i> .....	127
3.4. Функции.....	128
3.4.1. Создание функции. Инструкция return.....	128
3.4.2. Вызов функции.....	129
3.4.3. Передача аргументов в функцию. Необязательные аргументы. Функции с переменным числом аргументов.....	129
3.4.4. Глобальные и локальные переменные.....	131
3.4.5. Анонимные функции.....	132
3.4.6. Функции-генераторы.....	133
3.4.7. Декораторы функций. Вложенные функции. Рекурсивные функции.....	134
<i>Задания для самостоятельного выполнения по теме «Функции»</i> .....	136

3.5. Модули .....	137
3.5.1. Инструкции import и from .....	137
3.5.2. Создание и использование собственных модулей .....	139
3.5.3. Обзор стандартной библиотеки Python .....	140
<i>Задания для самостоятельного выполнения по теме «Модули» .....</i>	<i>144</i>
3.6. Файлы .....	144
<i>Задания для самостоятельного выполнения по теме «Файлы» .....</i>	<i>150</i>
3.7. Исключения .....	150
3.7.1. Основные исключения .....	151
3.7.2. Обработка исключений. Инструкция try ... except .....	152
3.7.3. Получение информации об исключении. Создание новых исключений ...	153
<i>Задания для самостоятельного выполнения по теме «Исключения» .....</i>	<i>153</i>
<i>Контрольные вопросы и задания .....</i>	<i>154</i>
<b>Глава 4. Объектно-ориентированное программирование</b>	
<b>в языке Python .....</b>	<b>155</b>
4.1. Базовые принципы объектно-ориентированного программирования .....	155
4.1.1. Инкапсуляция .....	155
4.1.2. Наследование .....	156
4.1.3. Полиморфизм .....	156
<i>Задания для самостоятельного выполнения по теме «Базовые принципы ООП» .....</i>	<i>157</i>
4.2. Классы в языке Python .....	157
4.2.1. Инструкция class .....	158
4.2.2. Создание экземпляра класса .....	159
4.2.3. Атрибуты класса и экземпляра класса. Закрытые атрибуты .....	160
<i>Задания для самостоятельного выполнения по теме «Классы в языке Python» .....</i>	<i>164</i>
4.3. Методы класса .....	165
4.3.1. Конструктор класса __init__() .....	166
4.3.2. Использование ссылки на экземпляр класса .....	168
4.3.3. Статические методы .....	169
4.3.4. Закрытые методы .....	170
4.3.5. Специальные методы .....	171
4.3.6. Перегрузка операторов .....	173
<i>Задания для самостоятельного выполнения по теме «Методы класса» .....</i>	<i>174</i>
4.4. Наследование .....	174
4.4.1. Простое наследование .....	174
4.4.2. Множественное наследование .....	176
4.4.3. Абстрактные методы .....	176
<i>Задания для самостоятельного выполнения по теме «Наследование» .....</i>	<i>178</i>
<i>Контрольные вопросы и задания .....</i>	<i>179</i>
<b>Глава 5. Разработка графических интерфейсов в программе</b>	
<b>на языке Python .....</b>	<b>180</b>
5.1. Событийно-ориентированное программирование .....	180
5.1.1. Событие .....	180
5.1.2. Обработчик события .....	180
5.1.3. Цикл обработки событий .....	181
<i>Задания для самостоятельного выполнения по теме «Событийно-ориентированное</i>	<i>182</i>
<i>программирование» .....</i>	<i>182</i>

5.2. Инструменты для создания графических интерфейсов пользователя (GUI) .....	182
5.2.1. Общие сведения о GUI Python .....	182
5.2.2. Отслеживание событий .....	184
5.2.3. Создание базового окна .....	185
<i>Задания для самостоятельного выполнения по теме «Инструменты для создания графических интерфейсов пользователя (GUI)» .....</i>	<i>186</i>
5.3. Элементы графического интерфейса (виджеты) .....	187
5.3.1. Создание и конфигурирование виджета. Менеджер размещения .....	187
5.3.2. Использование элементов Button (Кнопка).....	189
5.3.3. Label (Надпись) .....	190
5.3.4. Entry (Поле ввода) .....	191
5.3.5. Checkbutton (Флажок) .....	193
5.3.6. Radiobutton (Переключатель).....	194
5.3.7. Другие виджеты .....	195
5.3.8. Метод bind .....	201
5.3.9. Canvas (изображение) .....	203
<i>Задания для самостоятельного выполнения по теме «Элементы графического интерфейса (виджеты)» .....</i>	<i>205</i>
<i>Контрольные вопросы и задания .....</i>	<i>208</i>
<b>Библиографический список .....</b>	<b>209</b>
<b>Перечень тем для проверки знаний по дисциплине.....</b>	<b>210</b>

*По вопросам приобретения книг обращайтесь:*  
**Отдел продаж «ИНФРА-М» (оптовая продажа):**  
127282, Москва, ул. Полярная, д. 31В, стр. 1  
Тел. (495) 280-15-96; факс (495) 280-36-29  
E-mail: books@infra-m.ru

•  
**Отдел «Книга—почтой»:**  
тел. (495) 280-15-96 (доб. 246)

---

ФЗ      Издание не подлежит маркировке  
№ 436-ФЗ      в соответствии с п. 1 ч. 4 ст. 11

*Учебное издание*

**Жуков Роман Александрович**

# **ЯЗЫК ПРОГРАММИРОВАНИЯ PYTHON: ПРАКТИКУМ**

**УЧЕБНОЕ ПОСОБИЕ**

Оригинал-макет подготовлен в НИЦ ИНФРА-М  
ООО «Научно-издательский центр ИНФРА-М»  
127282, Москва, ул. Полярная, д. 31В, стр. 1  
Тел.: (495) 280-15-96, 280-33-86. Факс: (495) 280-36-29  
E-mail: books@infra-m.ru      <http://www.infra-m.ru>

Подписано в печать 16.04.2019.  
Формат 60×90/16. Бумага офсетная. Гарнитура Newton.  
Печать цифровая. Усл. печ. л. 13,5.  
Тираж 500 экз. (1 – 50). Заказ № 04504  
ТК 681551-1000002-160419

Отпечатано в типографии ООО «Научно-издательский центр ИНФРА-М»  
127282, Москва, ул. Полярная, д. 31В, стр. 1  
Тел.: (495) 280-15-96, 280-33-86. Факс: (495) 280-36-29



**ЖУКОВ**  
**Роман Александрович**

Кандидат физико-математических наук, доцент, доцент Тульского филиала Финансового университета при Правительстве Российской Федерации.

Сфера интересов: моделирование, программирование, математика, экономика, сложные системы.

Автор более 50 учебно-методических и научных работ, учебных пособий.

ISBN: 978-5-16-014701-7



9 785160 147017