Дальневосточный федеральный университет
Политехнический институт

Г.П. Озерова

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON В ПРИМЕРАХ И ЗАДАЧАХ

Учебное электронное издание
Учебное пособие для вузов

Владивосток
2022

Дальневосточный федеральный университет
Политехнический институт

Г.П. Озерова

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON В ПРИМЕРАХ И ЗАДАЧАХ

Учебное электронное издание
Учебное пособие для вузов

Владивосток
Издательство Дальневосточного федерального университета
2022

УДК 004.42
ББК 32.973.26-018.1
О-46

Рецензенты: *И.Е. Воронина*, д.т.н, профессор кафедры программного обеспечения и администрирования информационных систем (Воронежский государственный университет, Воронеж); *О.Н. Любимова,* д.ф.-м.н., профессор отделения машиностроения, морской техники и транспорта Инженерного департамента (Дальневосточный федеральный университет, Владивосток).

Автор: **Озерова Галина Павловна**, к.т.н, доцент
отделения машиностроения, морской техники и транспорта
Инженерного департамента Политехнического института
Дальневосточный федеральный университет, Владивосток

Озерова Г.П. Основы программирования на языке Python в примерах и задачах: учебное пособие для вузов / Политехнический институт ДВФУ. – Владивосток: Изд-во Дальневост. федерал. ун-та, 2022. – 1 CD. [128 с.]. – Систем. требования: Acrobat Reader, Foxit Reader либо другой их аналог. – ISBN 978-5-7444-5217-9. – Текст: электронный.

В учебном пособии доступно изложены конструкции языка программирования Python и их применение на примерах решения задач из различных сфер деятельности. При этом студентам предлагаются минимальные теоретические аспекты программирования, необходимые для решения конкретной задачи. Кроме того, в научно-популярной форме объясняется физическая, математическая или экономическая суть каждой задачи. Использование возможностей языка проиллюстрировано большим числом примеров. Предусмотрены варианты заданий для самостоятельного решения.

Пособие предназначено для студентов младших курсов инженерных специальностей, изучающих информационные технологии и программирование.

Ключевые слова: алгоритм, программа, операторы, графика, символьные вычисления, модули math, matplotlib, numpy, sympy.

Keywords: algorithm, program, operators, graphics, symbolic calculations, modules math, matplotlib, numpy, sympy.

Редактор Т.В. Рябкова
Компьютерная верстка Г.П. Озеровой
Дизайн CD Г.П. Писаревой

Опубликовано: 14.04.2022
Объем 3,7 МБ [Усл. печ. л. 14,9]
Тираж 10 экз.

Издание подготовлено редакционно-издательским отделом
Политехнического института ДВФУ
[Кампус ДВФУ, корп. С, каб. 714]

Дальневосточный федеральный университет
690022, Владивосток, о. Русский, пос. Аякс, 10

Изготовитель CD: Дальневосточный федеральный университет
(типография Издательства ДВФУ
690091, Владивосток, ул. Пушкинская, 10)

Защищено от копирования

ISBN 978-5-7444-5217-9 © ФГАОУ ВО «ДВФУ», 2022

Оглавление

ВВЕДЕНИЕ 5
1. ОСНОВЫ ВЫЧИСЛЕНИЙ НА ЯЗЫКЕ PYTHON 6
1.1. Переменные и типы 6
1.2. Основные конструкции языка 7
1.3. Решение простой вычислительной задачи 12
1.4. Математические вычисления, модуль math 17
1.5. Обработка исключительных ситуаций 20
1.6. Функции 22
1.7. Форматный вывод 24
1.8. Задача, вычисления на плоскости 25
1.9. Варианты заданий для самостоятельного решения 31
2. ОБРАБОТКА ТАБЛИЧНЫХ ДАННЫХ И ИХ ВИЗУАЛИЗАЦИЯ 34
2.1. Списки 34
2.2. Задача, формирование таблицы значений функции 41
2.3. Построение графиков и диаграмм 45
2.4. Задача, построение графиков функций одной переменной 50
2.5. Задача: построение графиков функций одной переменной с точками разрыва 52
2.6. Рисование плоских фигур 55
2.7. Задача, рисование фигуры из геометрических примитивов 59
2.8. Варианты заданий для самостоятельного решения 62
3. МАТРИЦЫ И ВЕКТОРЫ 72
3.1. Массивы 72
3.2. Задача о движении по различным участкам дороги 78
3.3. Многомерные массивы 80
3.4. Задача, матричные вычисления 86
3.5. Задача: решение системы линейных уравнений 89
3.6. Тренды 90
3.7. Задача о выстреле из пушки 92
3.8. Варианты заданий для самостоятельного решения 95
4. СИМВОЛЬНЫЕ ВЫЧИСЛЕНИЯ 101
4.1. Символьные преобразования 102
4.2. Решение уравнений 104
4.3. Дифференцирование и интегрирование 109
4.4. Предел функции 115
4.5. Задача: исследование функции и построение ее графика 117
4.6. Варианты заданий для самостоятельного решения 122
ЗАКЛЮЧЕНИЕ 125
СПИСОК ЛИТЕРАТУРЫ 126
Приложение А. Описание работы в среде IDLE(Python) 127
Приложение Б. Установка библиотек Python 128
Приложение В. Стили библиотеки matplotlib.pyplot 129

ВВЕДЕНИЕ

Python – это универсальный современный язык программирования высокого уровня, к преимуществам которого относят высокую производительность программных решений и структурированный хорошо читаемый код. Синтаксис Питона максимально облегчен, что позволяет изучить его за сравнительно короткое время. Ядро имеет очень удобную структуру, а широкий перечень встроенных библиотек позволяет применять широкий набор полезных функций и возможностей. Язык Python может использоваться для реализации прикладных приложений, а также для разработки WEB-сервисов. Правильное русское произношение названия языка программирования – «Пайтон», но чаще используется искаженное – «Питон».

История языка программирования Python началась в конце 1980-х. Создатель языка Гвидо ван Россум задумал Python в 1980-х годах, а приступил к его созданию в декабре 1989 года в центре математики и информатики в Нидерландах. Ван Россум является основным автором Python и продолжал выполнять центральную роль в принятии решений относительно развития языка вплоть до 12 июля 2018 года.

Язык программирования позаимствовал название у популярного в 70-е годы шоу Monty Python's Flying Circus. Но большая часть пользователей об этом не знают и ассоциируют название со змеей. Одна из целей создателей Python – сделать программирование простым и забавным.

Питон – не самый «молодой» язык программирования, но и не слишком старый. К моменту его создания уже существовали такие «монстры», как Паскаль или Си. И поэтому при создании языка авторы старались взять лучшее из различных платформ для разработчиков. Фактически Python представляет собой своеобразное сочетание удачных решений более чем из 8 различных языков.

В настоящее время Python является одним из самых востребованных языков программирования. В соответствии с индексом TIOBE (это один из самых известных рейтингов языков программирования) на ноябрь 2020 года Python занимал второе место в рейтинге самых популярных языков программирования в мире. Его обогнал только C.

После изучения языка Python легко освоить другие языки программирования.

Существует классическая техника изучения языков программирования: сначала типы данных, потом конструкции, затем сложные структуры, классы, объекты и т.д. Как следствие, решаемые в качестве примеров задачи демонстрируют возможности определенного элемента языка и не имеют никакого прикладного значения. Но в жизни все совсем не так: есть задача, ее нужно решить, желательно – средствами, которые не потребуют глубоких знаний по программированию.

В данном учебном пособии минимальные теоретические аспекты программирования предлагаются читателю, только если этого потребует решаемая задача. Кроме того, в научно-популярной форме объясняется физическая, математическая или экономическая суть каждой задачи. В пособии рассматриваются:

- решение вычислительных задач с использованием библиотеки **`math`**;
- представление и обработка табличных данных с помощью списков;
- визуализация данных средствами библиотеки **`matplotlib`**;
- решение задач с массивами с помощью библиотеки **`numpy`**;
- символьные вычисления и преобразования средствами модуля **`sympy`**.

Каждый параграф завершается заданием для самостоятельного решения, в конце каждой главы предлагаются варианты индивидуальных заданий.

1. ОСНОВЫ ВЫЧИСЛЕНИЙ НА ЯЗЫКЕ PYTHON

1.1. Переменные и типы

Для хранения данных в программе используются переменные. Простыми словами, переменная – это некоторый объект, который имеет имя, тип и значение.

Правила для выбора имен переменных:

- имя должно состоять из букв, цифр и знаков подчеркивания, начинаться с буквы, не может совпадать с ключевыми словами языка;
- Python различает большие и маленькие буквы в имени (переменные **x** и **X** – разные переменные).

Также есть несколько рекомендаций, которые позволяют сделать программу более читабельной:

- имя переменной должно быть записано строчными буквами;
- имя переменной должно максимально точно соответствовать хранимым в ней данным;
- если имя переменной состоит из нескольких слов, то слова разделяются подчеркиванием.

Например, следующие имена переменных являются верными и могут использоваться для описания возраста человека:

`age, user_age, person_age, user_1_age.`

Имена следующих переменных записаны неверно:

`12_age` – начинается с цифры;

`user-age` – содержит недопустимый символ «-»;

`f(x)` – содержит недопустимые символы «(» и «)»;

`if` – является ключевым словом языка.

В Python не предусмотрено явное описание типа переменных, тип определяется в процессе выполнения программы в зависимости от данных, которые заносятся в переменную. В переменную можно занести:

- строки текста, которые в программе заключаются в двойные или одинарные кавычки (`"Python", 'Я изучаю Python'`);
- целые значения `(17, -4, 0)`;
- вещественные числа; при вводе значений этого типа в качестве разделителя целой и дробной части используется точка (`-15.4, 12.0, 318.5302`).

Задания для самопроверки

1. Выберите верные имена переменных:

```
name_use
name+user
1_user
user_1_1
user
```

2. Укажите тип, который соответствует следующим значениям (для строковых данных – **string**, для целых чисел – **integer**, для вещественных **float**, если значение записано неверно – укажите **error**):

```
"Привет"
1,67
-7.90
0
24
12/09/2022
```

1.2. Основные конструкции языка

1.2.1. Операторы ввода и вывода

Для ввода данных используется оператор **input()**, который считывает строку текста, введенную пользователем с консоли, и заносит ее в переменную. Следующий оператор вводит строку текста и заносит ее в переменную **name**:

```
name = input()
```

Для вывода значения переменной используется оператор **print()**, оператор выведет на экран значение переменной **name**:

```
print(name)
```

С помощью оператора **print()** можно выводить строки в кавычках и значения переменных одновременно, при выводе они разделяются запятой.

```
print("Привет, ", name, "!")
```

Для ввода числовых значений в переменные необходимо перед оператором ввода **input()** указать тип значения (**int** или **float**). Следующий оператор вводит целое значение в переменную **age**:

```
age = int(input())
```

Кроме того, Python допускает использование подсказки пользователю непосредственно в операторе **input()**. При использовании такой подсказки ввод данных пользователем при выполнении программы будет осуществляться сразу после текста подсказки на той же строке. Следующий оператор вводит целое значение в переменную **age**, выдавая пользователю подсказку:

```
age = int(input("Сколько Вам полных лет? "))
```

Пример. Спросить у пользователя его имя и сколько ему лет. Вывести приветствие пользователю.

Код:

```
name = input("Ваше имя? ")
age = int(input("Сколько Вам полных лет? "))
print("Привет,", name, "! Ваш возраст", age, ".")
```

Результат:

```
Ваше имя? Алина
Сколько Вам полных лет? 20
Привет, Алина ! Ваш возраст 20 .
```

Задания для самопроверки

1. Все перечисленные ниже операторы являются неверными, укажите ошибку и исправьте каждый оператор:

```
name = print("Ваше имя? ")
name = input("Ваше имя? ')
x = input(int("x="))
print("x=" x)
```

2. Напишите программу, которая спрашивает у пользователя его имя, возраст и рост (в метрах). Вывести приветствие пользователю, указав его возраст и рост.

1.2.2. Оператор присваивания

Вычисление выражения и занесение его значения в некоторую переменную осуществляется с помощью оператора присваивания. Например, следующий оператор заносит в переменную name значение «Алина»:

```
name = "Алина"
```

В выражениях допустимо использовать арифметические операции:

- сложение «+»,
- вычитание «-»,
- умножение «*»,
- деление «/»,
- целочисленное деление «//»,
- остаток от деления «%», возведение в степень «**».

Python поддерживает привычный порядок выполнения операций: сначала возведение в степень, затем умножение, деление, целочисленное деление и остаток (все они имеют одинаковый приоритет, применяются последовательно слева направо), последним выполняется сложение или вычитание. Для изменения приоритета используются круглые скобки.

Примеры выражений:

```
3 + 2 * 5 ** 2  - 1 = 52
(3 + 2) * 5 ** 2  - 1 = 124
5 ** 2  / 2 = 12.5
63 % 10 = 3
63 % 10 + 4 = 7
63 % (10 + 4) = 7
63 // 10 = 6
63 // 10 + 4 = 10
63 // (10 + 4) = 4
```

Пример. Спросить у пользователя его имя и его год рождения. Вычислить сколько лет сейчас пользователю и вычислить результат (считаем, что текущим является 2022 год).

Код:

```
name = input("Ваше имя? ")
year_birth = int(input("В каком году Вы родились? "))
age = 2022 - year_birth
print("Привет, ", name, "! Ваш возраст ", age, ".")
```

Результат:

```
Ваше имя? Алина
В каком году Вы родились? 2002
Привет, Алина ! Ваш возраст 20 .
```

Задания для самопроверки

1. Вычислите значения следующих выражений для заданных **x** и **y**:

```
x = 5
y = 3
x + y ** 2 =
y ** 2 * x =
x ** y - x =
x ** (y - 2) =
x / y + 3 =
x / y * 3 =
x // y =
x % y =
```

2. Запишите операторы присваивания, которые описывают следующие выражения:

$$y = 4 \cdot x^2 - 5$$

$$z = \frac{(x-2)^2 - 1}{x + 4}$$

1.2.3. Условный оператор

Для реализации возможности выполнения различных действий в зависимости от условия в Python используется условный оператор **if**…**elif**…**else**, синтаксис которого следующий (разделы **elif** и **else** могут отсутствовать):

```
if условие_1:
    операторы_1
elif условие_2:
    операторы_2
...
else:
    операторы
```

Выполняется этот оператор так:

- проверяется `условие_1` после **if**, если условие истинно, то выполняются `операторы_1`;
- в противном случае проверяется `условие_2` после первого **elif**, если оно истинно, то выполняются `операторы_2`;
- аналогично обрабатываются все **elif**;
- в случае если все условия ложны, то выполняются `операторы` после **else**.

Особенности синтаксиса оператора if:

1. ключевые слова (**if**, **elif**, **else**) записываются строчными буквами;
2. после условия в операторах **if** и **elif**, а также после **else** обязательно ставится знак двоеточия «:», который означает наличие вложенных конструкций;
3. все операторы после знака «:» считаются вложенными и записываются с отступом (ставится два или четыре пробела перед каждым вложенным оператором).

Вложенные инструкции в Python записываются в соответствии с одним и тем же шаблоном, когда основная инструкция завершается двоеточием, вслед за которым располагается вложенный блок кода, с отступом под строкой основной конструкции.

Вложенные инструкции объединяются в блоки по величине отступов. Отступ может быть любым, главное, чтобы в пределах одного вложенного блока отступ был одинаков (обычно это 4 пробела).

В следующем примере оператор 1 и оператор 2 относятся к вложенному блоку основной конструкции, а оператор 3 находится на уровне основной конструкции.

```
основная_конструкция:
    оператор_1
    оператор_2
оператор_3
```

Условия в операторе **if** являются логическими выражениями и могут включать:

- знаки отношений (равно «==» (два подряд равенства), не равно «!=», больше «>», меньше «<», больше или равно«>=», меньше или равно «<=»),
- логические операции (И «`and`», ИЛИ «`or`», НЕ «`not`»);

Результатом вычисления логических выражений является либо Истина (`True`), другими словами, в этом случае выражение «верно», либо Ложь (`False`) – выражение «неверно».

При использовании логической операции И (`and`) выражение верно, если все его элементы верны. При использовании логической операции ИЛИ (`or`) выражение верно, если хотя бы один его элемент верен.

Примеры:

```
3 > 5 результат False
3 == (1 + 2) результат True
3 >= (1 + 2) результат True
3 != (1 + 2) результат False
```

```
3 < 5 and 3 >= 4 результат False
3 < 5 or 3 >= 4 результат True
3 < 5 and not (3 >= 4) результат True
```

Пример. Дана сторона квадрата **a** (вещественное число). Если введено положительное значение **a**, то посчитать площадь и периметр квадрата. В противном случае вывести сообщение об ошибке.

Код:

```
a = float(input("a = "))
if a <= 0:
    print("Длина должна быть положительной")
else:
    p = 4 * a
    s = a * a
    print("p =", p, "s =", s)
```

Результат:

Тест 1

```
a = 4
p = 16.0 s = 16.0
```

Тест 2

```
a = -2
Длина должна быть положительной
```

1.2.4. Комментарии в Python

Комментарии – это способ описания того, что делает программа на самом высоком уровне. Это специальные отмеченные строчки, которые комментируют код. Это неисполняемые, но все равно важные части кода. Комментарии делают код более понятым и читаемым.

В Python строка с комментарием начинается с символа «#». Все, что написано после этого символа, при выполнении программа пропускает.

Например, если в программе встречается следующий оператор:

```
income = x *(1 + k /(12*100)) ** n - x
```

совершенно непонятно, что именно он делает. Если же добавить комментарий, то его назначение становится прозрачным:

```
#доход от вклада x на n месяцев под k% годовых с капитализацией процентов
income = x *(1 + k /(12*100)) ** n - x
```

1.2.5. Встроенные функции Python

В языке Python реализовано много встроенных функций, использующихся для различных целей, вот некоторые из них:

`abs(x)` – вычисляет модуль числа x;

`round(x, n)` - округляет число x до n знаков после запятой;

`min(x, y, z, …)` – находит минимальное из списка значений;

`max(x, y, z, …)` – находит максимальное из списка значений.

Примеры:

```
abs(-4) = 4
abs(5) = 5
round(3.141569) = 3
round(3.141569, 4) =3.1416
min(3, 4, 6.8, -3, -1.89) = -3
max(3, 4, 6.8, -3, -1.89) = 6.8
```

Важно помнить, что в языке Python для округления чисел используется банковский способ округления, который отличается от математического, если в следующей позиции после позиции округления стоит 5. При математическом округлении в этом случае число

округляется в большую по модулю сторону, при банковском – к ближайшему чётному. Например:

```
round(2.65, 1) = 2.6
round(2.75, 1) = 2.8
```

Пример. Всего в группе учится **n** студентов, из них проживает в общежитии **k** человек. Проверить входные данные и вывести процент студентов, проживающих в общежитии. При выводе результат округлить до двух знаков после запятой.

Код:

```
n = int(input("n = "))
k = int(input("k = "))
# количества студентов - положительное число,  также количество
# проживающих в общежитии не может быть больше общего количества
if n <= 0 or k <= 0 or k > n:
   print("Ошибочные входные данные")
else:
   percent = k / n * 100
   print("проживает в общежитии:", round(percent, 2),"%")
```

Результат:

тест 1

```
n = 23
k = 17
проживает в общежитии: 73.91 %
```

тест 2

```
n = 20
k = 21
Ошибочные входные данные
```

Задания для самопроверки

1. Какое значение получит переменная **y** после программы:

```
x = 3
if x <= 0:
    y = x
elif x <= 5:
    y = x ** 2
else:
    y = -x
```

2. Вычислите значения следующих логических выражений:

```
x == y or (x - 4) >= 1
x != 2 and y <= 5
```

3. Какие значения получатся после выполнения следующих встроенных функций?

```
round(3.89) =
round(-2.49) =
round(4.999,2) =
abs(-5) =
abs(6.1) =
max(1.9, -2, -3.1, 0.2) =
abs(min(3, -2, -3.1, 0.2)) =
min(3, abs(-2), abs(-3.1), 0.2) =
```

4. Напишите программу для решения следующей задачи:

Ракета запускается с точки на экваторе и развивает скорость v км/с. Каков результат запуска?

***Указание**: если $v \le 7.8$ км/с, то ракета упадет на Землю (вывести 0), если $7.8 < v < 11.2$, то ракета станет спутником Земли (вывести 1), если $11.2 \le v \le 16.4$, то ракета*

станет спутником Солнца (вывести 2), если v > 16.4, то ракета покинет Солнечную Систему (вывести 3).

Если будет введено значение, меньшее или равное 0, то вывести "error".

1.3. Решение простой вычислительной задачи

Задача. Определить индекс массы тела человека по его росту и весу, а затем интерпретировать результат в соответствии с рекомендациями Всемирной организации здравоохранения (таблица 1).

Таблица 1 – Интерпретация индекса массы тела

Индекс массы тела	Описание
Меньше 18,5	недостаточная масса тела
От 18,5 до 24,99	нормальная масса тела
От 25 до 29,99	избыточная масса тела
Больше или равно 30	ожирение

Для решения любой задачи средствами языка программирования обязательно выполняется следующая последовательность шагов (или этапов).

1. **Неформальная постановка задачи**. На этом этапе определяется, что и как будет делать наша программа, описываются входные и выходные данные, указываются необходимые формулы.
2. **Проектирование и реализация программы**. На этом этапе программист определяет, на каком языке он будет писать, какие программные средства ему понадобятся, выбирает тип данных и имена для входных и выходных данных, проектирует и реализует код программы.
3. **Отладка и тестирование**. На этом этапе необходимо найти и устранить ошибки программы, а также проверить ее работоспособность для различных входных данных.

Подробно рассмотрим каждый этап для решения задачи вычисления и интерпретации массы тела.

1.3.1. Неформальная постановка задачи

Прежде всего, необходимо ответить на вопрос: «Какая информация нам нужна, чтобы решить поставленную задачу?» Мы должны знать вес человека, точнее, его массу в килограммах и его рост в метрах. Иногда при расчете индекса массы тела учитывается возраст. Это **входные данные** программы. Для всех входных данных нужно указать, какие значения считаются в данной задаче недопустимыми.

При вводе возраста, так как в задаче учитывается количество полных лет, верным будет целое число, например 23 или 45. Однако представить человека в возрасте −10 лет мы не можем, также считаем, что тем, кто еще не достиг 10-летнего возраста, знание индекса массы тела совсем необязательно. Поэтому, если пользователь введет число меньше 10, считаем этот ввод неверным. Для роста верным считаем интервал от 0 до 2.5 метров, для веса – от 0 до 300 килограмм. Это условные значения и они могут быть пересмотрены разработчиком программы. В таблице 2 приведено описание входных данных.

Таблица 2 – Входные данные

Описание	Пример	Недопустимые значения
Возраст (количество полных лет)	23, 45, 16	<10
Рост (м)	1.78, 1.65, 1.85	0 < или > 2.5
Вес (кг)	65, 58.7, 80.3, 75	<0 или > 300

Следующим шагом неформальной постановки задачи является описание результата выполнения программы или **выходных данных**. В нашем случае мы должны посчитать индекс массы тела, а также разъяснить пользователю, что этот индекс означает. В таблице 3 приведено описание выходных данных программы.

Таблица 3 – Выходные данные

Описание	Пример
Индекс массы тела	18.45, 24.12, 27.1
Заключение	недостаточная масса тела нормальная масса тела избыточная масса тела ожирение

На следующем этапе неформальной постановки задачи необходимо определиться с **алгоритмами и формулами**, которые нам понадобятся в процессе реализации. Индекс массы тела вычисляется как отношение массы в килограммах к квадрату роста человека, выраженного в метрах:

$$bmi = \frac{m}{h^2},$$

где bmi– индекс массы тела в кг/м²,

m – масса тела в килограммах,

h – рост в метрах.

Последним шагом неформальной постановки задачи является описание **функциональных возможностей** программы. Наша программа должна:

- спросить у пользователя его рост и вес;
- вычислить индекс массы тела;
- вывести индекс массы тела на экран;
- вывести заключение по посчитанному индексу.

1.3.2. Проектирование и реализация программы

Для реализации программы мы **выбираем язык** Python, и не только потому, что он стоит в названии этого пособия. В соответствии с индексом TIOBE (это один из самых известных рейтингов языков программирования) на ноябрь 2020 года Python занимал второе место в рейтинге самых популярных языков программирования в мире. Его обогнал только C.

Также на этом этапе нужно выбрать среду, в которой будет разрабатываться программа. Среду можно выбрать любую, например в Приложении А приведено описание работы в среде IDLE(Python), которая устанавливается автоматически при установке Python с официального сайта.

Следующий шаг – **выбор имен переменных** и их типов для входных и выходных данных в терминах выбранного языка программирования. В таблице 4 приведено их формальное описание.

Таблица 4 – Описание переменных

Описание	Имя	Тип
Возраст (количество полных лет)	`age`	`int`
Рост (м)	`height`	`float`
Вес (кг)	`weight`	`float`
Индекс массы тела	`bmi`	`float`
Заключение	`description`	`string`

На этапе проектирования и реализации разрабатывается **проект программы**. Для этого используются различные средства, в том числе для простых программ можно использовать естественный язык. Для нашей задачи проект будет выглядеть так:

```
ввести входные данные
если входные данные неверные
то
    вывести сообщение об ошибке
иначе
    вычислить индекс массы тела
    выполнить интерпретацию индекса массы тела
    вывести результаты
```

Дальше можно переходить к реализации (кодированию) программы, то есть к записи ее на выбранном языке программирования. При этом рекомендуется писать программу по частям, постоянно проверяя ее работоспособность. Последовательно рассмотрим шаги реализации программы для вычисления индекса массы тела в соответствии с ее проектом.

1. Написать фрагмент программы для ввода входных данных. Для проверки правильности ввода вывести результат. Эту т.н. промежуточную печать после запуска и проверки программы удаляют.

Код:

```
age = int(input("Ваш возраст? "))
height = float(input("Ваш рост? "))
weight = float(input("Ваш вес? "))
# вывод значений для проверки
print("age =", age, "height =", height, "weight =", weight)
```

Результат:

```
Ваш возраст? 22
Ваш рост? 1.72
Ваш вес? 65
age = 22 height = 1.72 weight = 65.0
```

2. Выполнить проверку входных данных, если они ошибочные – вывести сообщение об ошибке (какие данные считаются ошибочными, описано в таблице 2).

Код:

```
age = int(input("Ваш возраст? "))
height = float(input("Ваш рост? "))
weight = float(input("Ваш вес? "))
if age < 10 or height <= 0 or height > 2.5 or weight <= 0 or weight > 300:
    print("Ошибочные входные данные")
else:
    print("Верные входные данные")
```

Результат:

Тест 1

```
Ваш возраст? 20
Ваш рост? 1.72
Ваш вес? 65
Верные входные данные
```

Тест 2

```
Ваш возраст? 20
Ваш рост? 172
Ваш вес? 65
Ошибочные входные данные
```

3. Вычислить значение индекса массы тела и округлить результат до 2 знаков после запятой. Обычно округление значений осуществляется при выводе, но в таблице для интерпретации результатов нашей задачи используются вещественные числа с двумя знаками после запятой, значит необходимо округлить результат в программе.

Код:

```
age = int(input("Ваш возраст? "))
height = float(input("Ваш рост? "))
weight = float(input("Ваш вес? "))
if age < 10 or height <= 0 or height > 2.5 or weight <= 0 or weight > 300:
    print("Ошибочные входные данные")
else:
    bmi = weight / height ** 2
    bmi = round(bmi, 2)
    print("Ваш индекс массы тела:", bmi)
```

Результат:

Тест 1

```
Ваш возраст? 20
Ваш рост? 1.72
Ваш вес? 65
Ваш индекс массы тела: 21.97
```

Тест 2

```
Ваш возраст? 25
Ваш рост? 1.65
Ваш вес? 70
Ваш индекс массы тела: 25.71
```

4. Объяснить пользователю результат вычисления в соответствии с таблицей 1. При этом на экран предполагается вывести сообщение типа «Вы относитесь к группе с нормальной массой тела», поэтому в переменную `description` словосочетание заносится в творительном падеже.

Код:

```
age = int(input("Ваш возраст? "))
height = float(input("Ваш рост? "))
weight = float(input("Ваш вес? "))
if age < 10 or height <= 0 or height > 2.5 or weight <= 0 or weight > 300:
    print("Ошибочные входные данные")
else:
    bmi = weight / height ** 2
    bmi = round(bmi, 2)
    print("Ваш индекс массы тела:", bmi)
    if bmi < 18.5:
        description = "недостаточной массой тела."
    elif bmi < 25:
        description = "нормальной массой тела."
    elif bmi < 30:
        description = "избыточной массой тела."
    else:
        description = "ожирением."
    print("Вы относитесь к группе людей с", description)
```

Результат:

Тест 1

```
Ваш возраст? 20
Ваш рост? 1.72
Ваш вес? 65
Ваш индекс массы тела: 21.97
Вы относитесь к группе людей с нормальной массой тела.
```

Тест 2

```
Ваш возраст? 25
Ваш рост? 1.65
Ваш вес? 70
Ваш индекс массы тела: 25.71
Вы относитесь к группе людей с избыточной массой тела.
```

1.3.3. Отладка и тестирование

На этом этапе после того, как найдены и исправлены все ошибки (то есть программа запускается и выдает результат), необходимо проверить работоспособность программы для различных входных данных. Естественно, невозможно перебрать все возможные варианты ввода пользователя, но необходимо их систематизировать и выделить те, которые позволяют проверить все вычисления и условные переходы.

Описание процесса тестирования можно оформить в виде таблицы (таблица 5). В ней привести вариант входных данных, правильный результат, который должна выдать программа, а также предусмотреть поле, в котором отметить совпадает ли реальный результат с верным.

Таблица 5 – Тестирование программы для различных входных данных

Набор входных данных	Правильный результат	Верно?

`age : -4`
`height :1.5`
`weight : 6`
	Ошибочные входные данные	✔

`age : 9`
`height :-1.5`
`weight : 65`
	Ошибочные входные данные	✔

`age : 25`
`height :3.8`
`weight : 60`
	Ошибочные входные данные	✔

`age : 40`
`height :1.8`
`weight : -5`
	Ошибочные входные данные	✔

`age : 30`
`height :1.8`
`weight : 90`
	Ваш индекс массы тела: 27.78
	Вы относитесь к группе людей с избыточной массой тела.
		✔

`age : 25`
`height :1.6`
`weight : 45`
	Ваш индекс массы тела: 17.58
	Вы относитесь к группе людей с недостаточной массой тела.
		✔

`age : 30`
`height :1.7`
`weight : 63`
	Ваш индекс массы тела: 21.8
	Вы относитесь к группе людей с нормальной массой тела.
		✔

`age : 16;`
`height :1.7`
`weight : 162`
	Ваш индекс массы тела: 56.06
	Вы относитесь к группе людей с ожирением.
		✔

Задание для самопроверки

Реализовать и протестировать программу для вычисления индекса массы тела в зависимости от роста, веса и возраста, а затем проинтерпретировать результат в соответствии с рекомендациями Всемирной организации здравоохранения (таблица 6):

Таблица 6 – Интерпретация индекса массы тела с учетом возраста

Индекс массы тела, возраст < 45	Индекс массы тела, возраст >= 45	Описание
Меньше 18,5	Меньше 22	недостаточная масса тела
От 18,5 до 24,99	От 22 до 26,99	нормальная масса тела
От 25 – 29.99	От 27до 31.99	избыточная масса тела
Больше или равно 30	Больше или равно 32	ожирение

Тест 1

```
Возраст: 35
Рост: 1.8
```

```
Вес: 120.71
bmi = 37.26 Вы относитесь к группе людей с ожирением.
```
Тест 2
```
Возраст: 18
Рост: 1.68
Вес: 82.79
bmi = 29.33 Вы относитесь к группе людей с избыточной массой тела.
```
Тест 3
```
Возраст: 50
Рост: 1.97
Вес: 98.67
bmi = 25.42 Вы относитесь к группе людей с нормальной массой тела.
```

1.4. Математические вычисления, модуль math

В программах можно выполнять достаточно сложные вычисления, в которых используются математические функции, такие как логарифм, экспонента, синус, косинус и т.д. Для вычисления этих и многих других математических функций в Python предусмотрена библиотека **`math`** (или по-другому модуль **`math`**). Библиотека является встроенным модулем Python, поэтому для ее использования никакой дополнительной установки не требуется.

1.4.1. Модуль math

Модуль **`math`** содержит множество функций для работы с числовыми данными, а также математические константы. В таблице 7 приведены некоторые из них.

Таблица 7 – Функции модуля math

Функция	Описание	Пример
`ceil(x)`	Округление до ближайшего большего числа	`ceil(4.2)=5` `ceil(-5.8)=-5`
`floor(x)`	Округление до ближайшего меньшего числа	`floor(4.2)=4` `floor(-5.8)=-6`
`exp(x)`	Экспонента от **x** (**e^x**)	`exp(3)=20.08...`
`log(x, base)`	Логарифм **x** по основанию **base,** если **base** не указан, вычисляется натуральный логарифм	`log(125, 5)=3.0`
`log10(x)`	Логарифм **x** по основанию 10	`log10(100)=2.0` `log10(0.01)=-2.0`
`log2(x)`	Логарифм x по основанию 2	`log2(64)=6.0` `log2(0.125)=-3.0`
`pow(x, y)`	**x^y**	`pow(3, 4)=81.0`
`sqrt(x)`	Квадратный корень из **x**	`sqrt(4)=2.0` `sqrt(2)=1.41...`
`acos(x)`	Арккосинус **x** (**x** в интервале от -1 до 1), Результат – значение в радианах	`acos(0)=1.57...` `acos(1)=0.0`
`asin(x)`	Арксинус **x** (**x** в интервале от -1 до 1), Результат – значение в радианах	`asin(0)=0.0` `asin(1)=1.57...`
`atan(x)`	Арктангенс **x**, Результат – значение в радианах	`atan(0)=0.0` `atan(1)=0.78...`
`cos(x)`	Косинус **x** (**x** задается в радианах)	`cos(0)=1.0`
`sin(x)`	Синус **x** (**x** задается в радианах)	`sin(0)=0.0`
`tan(x)`	Тангенс **x** (**x** задается в радианах)	`tan(0)=0.0`
`degrees(x)`	Конвертирует радианы в градусы	`degrees(3) = 171.8...`
`radians(x)`	Конвертирует градусы в радианы	`radians(180)=3.14...`
`pi`	pi = 3.1415926...	
`e`	e = 2.718281...	
`inf`	∞	

Практически все функции модуля **math** возвращают значения в вещественном формате. В некоторых случаях может получиться такой результат:

```
cos(1.5707963) = 2.6794896585e-08
```

Число вида `2.6794896585e-08` является записью вещественного числа в экспоненциальной форме, переписать его можно так:

$$2.6794896585e-08 = 2.6794896585 \cdot 10^{-8} = 0.000000026794896585,$$

то есть это практически ноль или, точнее, число равно нулю с точностью 7 знаков после запятой.

В программе, где предполагается использование модуля, применяется оператор:

```
import math
```

Этот оператор должен располагаться до первого обращения к функциям модуля. Как правило, все импортируемые модули перечисляются в начале программы.

В выражениях при использовании функций из этого модуля перед именем функции вставляется префикс **math** и точка. В таблице 8 приведены примеры выражений, для вычисления которых необходимы функции из модуля **math**.

Таблица 8 – Примеры описания выражений с явным указанием имени модуля

Выражение	Код программы
$sin(x) + cos(x)$	`math.sin(x) + math.cos(x)`
$\log_5(x \cdot e^{2 \cdot x})$	`math.log(x * math.exp(2 * x), 5)`
$\sqrt{b^2 - 4 \cdot a^2}$	`math.sqrt(b ** 2 - 4 * a ** 2)` `math.sqrt(math.pow(b, 2) - 4 * math.pow(a,2))`
$\sin(40^o)$	`math.sin(math.radians(40))`
π	`math.pi`

В случае, если точно известно, какие функции из модуля понадобятся в программе, можно их перечислить при подключении модуля и уже в тексте программы не использовать префикс **math**:

```
from math import функция_1, функция_2, ...
```

В таблице 9 приведены примеры описания выражений в программе, если все необходимые функции перечислены при импорте:

```
from math import sin, cos, log, exp, pow, radians, pi
```

Таблица 9 – Примеры описания выражений без явного указания имени модуля

Выражение	Код программы
$sin(x) + cos(x)$	`sin(x) + cos(x)`
$\log_5(x \cdot e^{2 \cdot x})$	`log(x * exp(2 * x), 5)`
$\sqrt{b^2 - 4 \cdot a^2}$	`sqrt(b ** 2 - 4 * a ** 2)` `sqrt(pow(b, 2) - 4 * pow(a,2))`
$\sin(40^o)$	`sin(radians(40))`
π	`pi`

Задания для самопроверки

1. Какие функции из модуля math записаны неверно? Исправьте ошибки в неверных выражениях.

```
math.log(10, 2)
math.exp(3, 5)
math.sin(-16)
math.degrees(16)
math sin(3)
math.radian(3)
```

2. Используя функции модуля **math**, запишите следующие выражения:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \qquad \pi r^2.$$

3. Написать программу, которая вычисляет расстояние между двумя точками на плоскости $A(x_0, y_0)$ и $B(x_1, y_1)$. Координаты точек ввести с клавиатуры, результат округлить до 4 знаков после запятой. Формула для вычисления расстояния:

$$r = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2},$$

Проверить программу для следующих входных данных:

Тест 1

```
x_0 = -4.5
y_0 = 2.7
x_1 = 1
y_1 = 3.1
Результат: 5.5145
```

Тест 2

```
x_0 = 1
y_0 = 1
x_1 = 2
y_1 = 2
Результат: 1.4142
```

1.4.2. Рекомендации по записи сложных выражений

Рекомендуется **перечислить** все используемые **функции** для вычисления выражения при подключении модуля **math**, чтобы запись формул была короче и понятнее.

Пример. Вычислить значение выражения:

$$y = \sqrt{e^{\sin(x)} + \ln(2 + \cos(x))}$$

при x = 4.5.

Код:

Вариант 1

```
import math
x = 4.5
y = math.sqrt(math.exp(math.sin(x)) + math.log(2 + math.cos(x)))
print(y)
```

Вариант 2

```
from math import sqrt, exp, sin, log, cos
x = 4.5
y = sqrt(exp(sin(x)) + log(2 + cos(x)))
print(y)
```

Результат:

```
0.9787799339565816
```

Сравните два варианта программы, обе программы выдают одинаковый результат, но способ с перечислением функций выглядит более удобочитаемым ~~читабельным.~~

При записи сложных математических выражений можно допустить ошибки, например, забыть поставить скобку или пропустить операцию. Поэтому рекомендуется **разбить выражение на части**, занести их в отдельные переменные, а затем вычислить полное выражение.

Пример. Вычислить значение выражения:

$$\ln\left(\left|y - \sqrt{sin^2\left(x + \frac{\pi}{3}\right)}\right| + cos\left(tg^2\left(\frac{1}{\sqrt[3]{z+1}}\right)\right)\right)$$

при x = 3.6 , y = −4.1 , z =5.

Решение

Из данного выражения целесообразно выделить две части и вычислить их отдельно (рисунок 1), сохранив результаты в переменные ***eq_1*** и ***eq_2***, а затем получить результат, используя эти переменные.

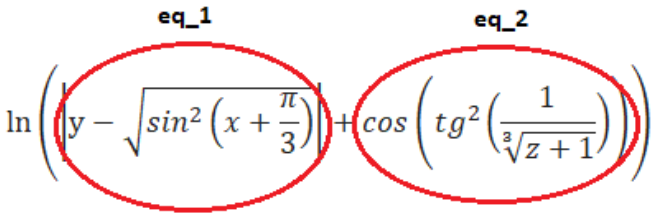


Рисунок 1. Пример выделения частей выражения

Код:

```
from math import log, sqrt, sin, pi, cos, tan

x = 3.6
y = -4.1
z = 5
eq_1 = sqrt(sin(x + pi / 3) ** 2)
eq_2 = cos(tan(1/((z+1) ** (1/3))) ** 2)
eq = log(abs(y - eq_1) + eq_2)
print("eq =", eq)
```

Результат:

```
eq = 1.796391214837781
```

Задание для самопроверки

Написать программу для вычисления следующего выражения, значения **x** и **y** ввести с клавиатуры, результат округлить до 5 знаков после запятой:

$$z=\frac{arcsin\left(\cos\left(x+\frac{\sqrt{3}}{2}\pi\right)\right)+1.2\sqrt{2-cos^2(y)}}{x^2+y^2+1}.$$

Проверить программу для следующих входных данных:

Тест 1

```
x = -2.3
y = 8.72
Результат: 0.03134
```

Тест 2

```
x = -7.1
y = 12.8
Результат: 0.00417
```

1.5. Обработка исключительных ситуаций

При вычислении выражений для некоторых значений могут возникать ошибки при выполнении программы, такие как деление на ноль, переполнение и пр. Например, при вычислении выражения:

```
y = 1 / x
```

если **x** будет иметь значение 0, то возникнет ошибка «Деление на ноль» и программа закончит свое выполнение.

Подобные ошибки необходимо обработать при реализации программы. Можно найти область определения выражения (в нашем случае **x** не должно равняться 0), а в программу включить условный оператор, в котором проверить значение **x**, если значение равно 0 – вывести ошибку, в противном случае выполнить вычисление:

```
if x == 0:
    print("Деление на ноль")
else:
    y = 1 / x
```

Однако, если выражение сложное, описать область ее определения часто не представляется возможным. Тогда можно использовать блок обработки исключений **try** … **except**. С его помощью осуществляется отслеживание и обработка ошибок, возникающих в процессе вычисления выражений. Блок обработки исключений состоит из двух частей:

- раздел **try**, где размещаются операторы, в которых может возникнуть ошибка;
- раздел **except**, в нем размещаются операторы, выполняемые при возникновении ошибок в разделе **try**, также в нем можно реализовать различные действия для различных типов ошибок.

Выполняется блок обработки исключений следующим образом:

- вычисляются операторы, размещенные в разделе **try**, если в каком-то из них возникает ошибка, управление передается блоку **except**, в противном случае выполняется первый оператор после блока обработки исключений;
- если управление передается в раздел **except**, то выполняются операторы в этом блоке, затем управление передается первому оператору после блока обработки исключений.

С использованием блока обработки исключений наш пример будет выглядеть следующим образом:

```
x = int(input("x = "))
try:
    y = 1 / x
except:
    print("Деление на ноль")
print("y =", y)
```

Протестируем нашу программу на различных входных данных:

Тест 1

```
x = 6
y = 0.16666666666666666
```

Тест 2

```
x = 0
Деление на ноль
Traceback (most recent call last):
  File "test.py", line 6, in <module>
    print("y =", y)
NameError: name 'y' is not defined
```

Программа выдает ошибку при вводе значения 0, так как при таком использовании блока обработки исключений после него осуществляется вывод вычисленного значения **y**, которое не определено.

Исправить ошибку можно, поместив вывод в разделе **try**:

```
x = int(input("x = "))
try:
    y = 1 / x
    print("y =", y)
except:
    print("Деление на ноль")
```

Тогда оба теста проходят.

Тест 1

```
x = 6
y = 0.16666666666666666
```

Тест 2

```
x = 0
Деление на ноль
```

Такой вариант подходит, если в дальнейшем переменную **y** нигде не нужно использовать. Если же результат (бесконечность), который получается при делении на ноль, важен для вычислений, можно занести в переменную константу **inf**, которая определена в модуле **math**. Наша программа в этом случае будет выглядеть следующим образом:

```
import math
x = int(input("x = "))
try:
    y = 1 / x
except:
    y = math.inf
print("y =", y)
```

Протестируем программу на различных входных данных:

Тест 1

```
x = 6
y = 0.16666666666666666
```

Тест 2

```
x = 0
y = inf
```

Задание для самопроверки

Написать программу для вычисления следующего выражения, значения **x** и **y** ввести с клавиатуры. В случае возникновения ошибок вычисления вывести **inf**.

$$z = \frac{\arccos(x) + 1.2\sqrt{y}}{x^2 - (y+1)^2}$$

Проверить программу для следующих входных данных:

Тест 1

```
x = 0.5
y =6
z = -0.08177610753920843
```

Тест 2

```
x = 0.5
y =-5
z = inf
```

Тест 3

```
x = 1
y =0
z = inf
```

Тест 4

```
x = 2
y = 1
z = inf
```

1.6. Функции

Программа – это последовательность операторов. Часто случается, что какая-то часть программы неоднократно повторяется. Например, если в задаче требуется вычислить попарное расстояние между тремя точками на плоскости $A(x_A, y_A)$, $B(x_B, y_B)$ и $C(x_C, y_C)$, то одна и та же формула применяется для вычисления трех расстояний.

Код:

```
from math import sqrt

# здесь нужно ввести или задать координаты трех точек

r_ab = sqrt((x_a - x_b) ** 2 + (y_a - y_b) ** 2)
r_bc = sqrt((x_b - x_c) ** 2 + (y_b - y_c) ** 2)
r_ac = sqrt((x_c - x_a) ** 2 + (y_c - y_a) ** 2)
```

Чтобы устранить подобного рода избыточность программного кода, используют понятие **функции**. Повторяющийся (да и не только) блок программного кода обозначают некоторым уникальным именем, чтобы потом при необходимости обратиться к нему по этому имени.

Для определения функции используется ключевое слово **def**, после которого указывается ее имя, круглые скобки и двоеточие.

Функции, вычисляющей расстояние между двумя точками, дадим имя **compute_len**, ее объявление имеет вид:

```
def compute_len():
```

Внутри функции можно использовать любые операторы языка, в нашем случае должна вычисляться длина стороны:

```
def compute_len():
    len_line = sqrt((x_b - x_a) ** 2 + (y_b - y_a) ** 2)
```

Но реализованная таким образом функция вычисляет только расстояние между точками **A** и **B**. Для того чтобы вычислить расстояние между двумя произвольными точками с координатами (x_0, y_0) и (x_1, y_1), необходимо в заголовке функции указать параметры (аргументы), с которыми она будет работать, а внутри функции использовать эти параметры как обычные переменные:

```
def compute_len(x_0,y_0,x_1,y_1):
    len_line = sqrt((x_1 - x_0) ** 2 + (y_1 - y_0) ** 2)
```

Расстояние между точками вычислено. Для того чтобы результат стал известен вне функции, используется оператор **return**, после которого указывается переменная (или выражение), значение которой является результатом выполнения функции (говорят, что это значение «возвращается» из функции):

```
def compute_len(x_0,y_0,x_1,y_1):
    len_line = sqrt((x_1 - x_0) ** 2 + (y_1 - y_0) ** 2)
    return len_line
```

Данная функция реализована и может быть использована для вычисления расстояния между двумя точками на плоскости, заданными своими координатами. Для этого к ней обращаются (вызывают ее) с конкретными значениями координат, а результат заносят в некоторую переменную. Например, чтобы вычислить расстояние между точками $A(x_A, y_A)$ и $B(x_B, y_B)$:

С помощью функции **compute_len** можно вычислить расстояние между тремя точками на плоскости $A(x_A, y_A)$, $B(x_B, y_B)$ и $C(x_C, y_C)$.

Код:

```
from math import sqrt

def compute_len(x_0,y_0,x_1,y_1):
    len_line = sqrt((x_1 - x_0) ** 2 + (y_1 - y_0) ** 2)
    return len_line

x_a = float(input("x_a = "))
y_a = float(input("y_a = "))
x_b = float(input("x_b = "))
y_b = float(input("y_b = "))
x_c = float(input("x_c = "))
y_c = float(input("y_c = "))

r_ab = compute_len(x_a,y_a,x_b,y_b)
r_bc = compute_len(x_b,y_b,x_c,y_c)
r_ac = compute_len(x_a,y_a,x_c,y_c)

print("r_ab =", round(r_ab, 4))
print("r_bc =", round(r_bc, 4))
print("r_ac =", round(r_ac, 4))
```

Результат:

```
x_a = 4
y_a = 1
x_b = 8.2
y_b = -3
x_c = 4.5
y_c = -1
r_ab = 5.8
r_bc = 4.2059
r_ac = 2.0616
```

Задания для самопроверки

1. Какое значение получит переменная **res** после выполнения следующего фрагмента программы:

```
def compute_resist(r_1, r_2):
    r = r_1 * r_2 / (r_1 + r_2)
    return r
res = compute_resist(15.0, 11.0)
```

2. Описать функцию, которая вычисляет прибыль от вложения суммы **x** руб на **n** месяцев под **k** процентов годовых с капитализацией процентов. Формула для вычисления прибыли:

$$income = x \cdot \left(1 + \frac{k}{12 \cdot 100}\right)^n - x$$

Заголовок функции должен иметь вид:

```
# deposit - сумма вклада, interest_rate -процентная ставка,
# amount_months - количество месяцев
def compute_income(deposit, interest_rate, amount_months):
```

С помощью этой функции посчитать прибыль от вложения для различных входных данных:

Тест 1

```
x = 100000
k = 4
n = 18
Прибыль: 6173.06
```

Тест 2

```
x = 50600
k = 4.5
n = 36
Прибыль: 7298.94
```

Тест 3

```
x = 1000000
k = 5
n = 6
Прибыль: 25261.868
```

1.7. Форматный вывод

Для наглядного представления выходных данных в Python используется форматный вывод.

Форматный вывод осуществляется с помощью оператора **%**, который используется в операторе **print**():

```
print("шаблон вывода" %(список вывода))
```

Список вывода – это перечисленные через запятую переменные, которые выводятся на экран. **Шаблон вывода** – строка текста со вставленными туда элементами форматирования для отображения каждого элемента списка вывода. Допускается форматный

вывод как числовых значений, так и строк. В таблице 10 приведены форматы для вывода наиболее распространённых типов данных.

Таблица 10 – Описание форматов

Тип данных	Формат	Описание
Целое число	`%8d`	Целое число размещается в 8 позициях, прижато к правому краю
Целое число	`%-8d`	Целое число размещается в 8 позициях, прижато к левому краю
Вещественное число	`%8.4f`	Вещественное число размещается в 8 позициях, выводится 4 знака после запятой, прижато к правому краю, незаполненные позиции после запятой заполняются 0
Вещественное число	`%-8.4f`	Вещественное число размещается в 8 позициях, выводится 4 знака после запятой, прижато левому краю, незаполненные позиции после запятой заполняются 0
Строка текста	`%8s`	Если длина строки меньше 8 символов, то она размещается в 8 позициях, прижата к правому краю, в противном случае формат не учитывается
Строка текста	`%-8s`	Если длина строки меньше 8 символов, то она размещается в 8 позициях, прижата к левому краю, в противном случае формат не учитывается

На рисунке 2 показано соответствие между форматом и переменной. Так, первый встретившийся в шаблоне вывода формат определяет способ вывода первой переменной в списке вывода, второй формат используется для второй переменной.

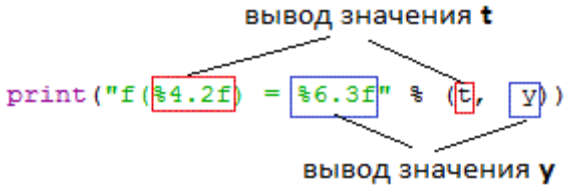


Рисунок 2. Форматный вывод

В строке шаблонов на рисунке 2 заданы строковые константы, которые должны быть выведены на экран `("f(", ") = ")`, а также форматы для двух переменных:

`%4.2f` – вещественное число, выводится в 4 позициях, количество знаков после запятой – 2, используется как формат для вывода переменной **`t`**;

`%6.3f` – вещественное число выводится в 6 позициях, количество знаков после запятой – 3, используется как формат для вывода значения **`y`**.

Задание для самопроверки

Дан фрагмент программы:

```
r = 5.9247836
v = 1.9747316
print("r = %___, v = %___" % (r, v))
```

В операторе **`print`** укажите форматы для каждой переменной так, чтобы в консоли был выведен следующий результат:

```
r = 5.925, v = 1.9747
```

1.8. Задача, вычисления на плоскости

Задача. Дан треугольник на плоскости, заданный координатами своих вершин. Для этого треугольника вычислить:

- длины сторон;
- периметр;

- площадь;
- величины углов в градусах.

1.8.1. Неформальная постановка задачи

Для наглядности нарисуем треугольник на плоскости (рисунок 3). Каждая его вершина представляет собой точку. Для описания точки на плоскости используются две координаты, например, точка A на рисунке 3 имеет координаты 8 по оси OX и 4 по оси OY.

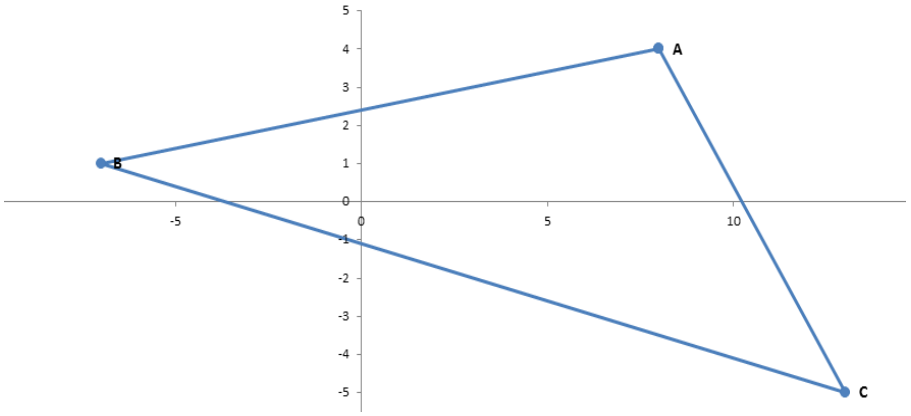


Рисунок 3. Треугольник ABC

Следовательно, входными данными нашей задачи будут 3 пары переменных, каждая пара – это координаты **x** и **y** вершины треугольника. Примерами координат являются числа 3 или −4.98. Ограничений на ввод координат треугольника нет.

Но есть ограничения на расположение точек. Так треугольник построить нельзя, если:

- или все его вершины совпадают,
- или две любые вершины совпадают,
- или все вершины лежат на одной прямой.

В этом случае никакие характеристики треугольника посчитать нельзя, поскольку это либо точка, либо отрезок.

Все эти случаи можно учесть с помощью признака существования треугольника: «треугольник существует, если сумма длин двух любых его сторон больше третей».

В таблице 11 приведены входные данные задачи. Также (для сокращения записи, чтобы не повторять таблицу в разделе проектирования) в этой таблице укажем имена переменных и их тип. Например, координату **x** точки **A** назовем **x_a**, координату точки **A** – **y_a**. Все переменные имеют тип **float**.

Таблица 11 – Входные данные

Описание	Имя	Тип	Пример	Недопустимые значения
Координата **x** точки A	x_a	float	5, -4.5	Нет
Координата **y** точки A	y_a	float	-15.1, 6.5	Нет
Координата **x** точки B	x_b	float	0, 1	Нет
Координата **y** точки B	y_b	float	-1.1, 9.45	Нет
Координата **x** точки C	x_c	float	2.55, 5.1	Нет
Координата **y** точки C	y_c	float	-2, -6.7	Нет
Условие существования треугольника: сумма двух любых его сторон больше третьей				

Результатом выполнения нашей программы (выходными данными) должны стать различные характеристики треугольника: длины его сторон, площадь и периметр, величины углов в градусах. Имена переменным выберем, как это принято в геометрии. Длину стороны назовем маленькой буквой, которая соответствует вершине, лежащей напротив стороны. Например, сторона **AB**, ее длина **c**, так как эта сторона лежит напротив вершины **C**. Площадь

и периметр назовем **`s`** и **`p`**. Величины узлов обозначаются специальным знаком угла «∠», после которого пишется имя вершины. Использовать символ угла в имени мы не можем, поэтому вместо него напишем **`angle`**, для соединения с именем вершины воспользуемся подчеркиванием. Типы результатов – вещественные числа. В таблице 12 приведены выходные данные задачи.

Таблица 12 – Выходные данные

Описание	Имя	Тип	Пример
Длина стороны AB	`c`	`float`	2.704, 16.6623
Длина стороны BC	`a`	`float`	13.327, 16.3218
Длина стороны AC	`b`	`float`	14.866, 6.2412
Периметр ABC	`p`	`float`	30.897, 39.2253
Площадь ABC	`s`	`float`	15.6, 50.46
Величина угла A	`angle_A`	`float`	9.061, 82.1801
Величина угла B	`angle_B`	`float`	50.906, 76.0373
Величина угла C	`angle_C`	`float`	120.033, 21.7826

На следующем этапе неформальной постановки задачи необходимо определиться с алгоритмами и формулами, которые нам понадобятся в процессе реализации. Эти формулы можно найти в справочнике по геометрии или на любом информационном ресурсе, посвященном аналитической геометрии.

1. *Длины сторон*:

$$AB = c = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2},$$
$$BC = a = \sqrt{(x_C - x_B)^2 + (y_C - y_B)^2},$$
$$AC = b = \sqrt{(x_C - x_A)^2 + (y_C - y_A)^2}.$$

2. *Периметр:*

$$p = a + b + c.$$

3. *Площадь (формула Герона)*:

$$s = \sqrt{\frac{p}{2} \cdot \left(\frac{p}{2} - a\right) \cdot \left(\frac{p}{2} - b\right) \cdot \left(\frac{p}{2} - c\right)}.$$

4. Для вычисления *углов* используется теорема косинусов:

$$c^2 = a^2 + b^2 - 2 \cdot a \cdot b \cdot \cos(\angle C).$$

Выразим величины углы (значения будут получены в радианах):

$$\angle A = arccos\left(\frac{c^2 + b^2 - a^2}{2 \cdot c \cdot b}\right),$$
$$\angle B = arccos\left(\frac{c^2 + a^2 - b^2}{2 \cdot c \cdot a}\right),$$
$$\angle C = arccos\left(\frac{a^2 + b^2 - c^2}{2 \cdot a \cdot b}\right).$$

Последним шагом неформальной постановки задачи является описание **функциональных возможностей** программы. Наша программа должна:

- ввести координаты точки A;
- ввести координаты точки B;
- ввести координаты точки C;
- вычислить длины сторон треугольника;
- проверить условие существования треугольника; если треугольник не существует, вывести сообщение;

- вычислить периметр треугольника;
- вычислить площадь треугольника;
- вычислить величины углов треугольника в радианах;
- перевести значения величин углов в градусы;
- вывести длины сторон, периметр, площадь и углы.

1.8.2. Проектирование и реализация программы

Для реализации программы мы по-прежнему выбираем язык Python, именование входных и выходных данных, выбор типов данных уже выполнено на этапе неформальной постановки задачи. Сразу перейдем к описанию проекта программы:

```
описать функцию вычисления длины стороны по координатам ее вершин
описать функцию вычисления площади по длинам трех сторон
описать функцию вычисления угла по длинам трех сторон
ввести входные данные
вычислить длины сторон треугольника
если входные данные неверные
то
    вывести сообщение об ошибке
иначе
    вычислить площадь
    вычислить периметр
    вычислить углы
    вывести результаты
```

Ниже приведен код программы, последовательность реализации программы описана в виде комментариев.

Код:

```
# Импортировать функцию для перевода радиан в градусы,
# арккосинуса и квадратного корня.
from math import sqrt, acos, degrees

#Реализовать функцию для вычисления длины отрезка
#с координатами концов (x0, y0) и (x1, y1).
def compute_len(x_0,y_0,x_1,y_1):
    len_line = sqrt((x_1 - x_0) ** 2 + (y_1 - y_0) ** 2)
    return len_line

#Реализовать функцию для вычисления площади треугольника
# по трем сторонам a_1, a_2, a_3.
def compute_area(a_1, a_2, a_3):
    p = (a_1 + a_2 + a_3) / 2
    area = sqrt(p * (p - a_1) * (p - a_2) * (p - a_3))
    return area

#Реализовать функцию для вычисления угла треугольника в градусах,
# если известны длины сторон a_1, a_2, a_3.
def compute_angle(a_1, a_2, a_3):
    angle_rad = acos((a_1 ** 2 + a_2 ** 2 - a_3 ** 2)/(2 * a_1 * a_2))
    return degrees(angle_rad)

#Ввести координаты вершин треугольника с подсказкой пользователю.
x_a = float(input("x_a = "))
y_a = float(input("y_a = "))
x_b = float(input("x_b = "))
y_b = float(input("y_b = "))
x_c = float(input("x_c = "))
y_c = float(input("y_c = "))
```

```
# Вычислить длины сторон треугольника.
c = compute_len(x_a, y_a, x_b, y_b)
a = compute_len(x_b, y_b, x_c, y_c)
b = compute_len(x_a, y_a, x_c, y_c)

#Проверить, существует ли треугольник, построенный по заданным точкам,
# если нет - выдать сообщение пользователю.
if a + b <= c or b + c <= a or a +c <= b:
    print("Треугольник не существует")
# В противном случае вычислить площадь, периметр, величины углов.
else:
    s = compute_area(a, b,c)
    p = a + b + c
    angle_A = compute_angle(c, b, a)
    angle_B = compute_angle(c, a, b)
    angle_C = compute_angle(a, b, c)

# Вывести результаты вычисления на экран,
# округлив их до 3 знаков после запятой.
    print("Стороны:", round(a, 3), round(b, 3), round(c, 3))
    print("Площадь:", round(s,3))
    print("Периметр:", round(p,3))
    print("Углы:", round(angle_A, 3),round(angle_B, 3),round(angle_C, 3))
```

Результат:

тест 1

```
x_a = 5
y_a = -15.1
x_b = 0
y_b = -1.1
x_c = 2.55
y_c = -2
Стороны: 2.704 13.327 14.866
Площадь: 15.6
Периметр: 30.897
Углы : 9.061 50.906 120.033
```

тест 2

```
x_a = 1
y_a = 1
x_b = 2
y_b = 2
x_c = 3
y_c = 3
Треугольник не существует
```

1.8.3. Отладка и тестирование

На этом этапе, после того как найдены и исправлены все ошибки (то есть программа запускается и выдает результат), проверяется работоспособность программы на различных входных данных. В таблице 13 приведены результаты тестирования.

Задание для самопроверки

Решить следующую задачу:

Дан треугольник ABC на плоскости, заданный координатами своих вершин. Для этого треугольника вычислить:

- *радиус вписанной в треугольник окружности;*
- *радиус описанной вокруг треугольника окружности;*
- *сумму длин трех медиан треугольника.*

Результаты округлить до 4 знаков после запятой.

Таблица 13 – Тестирование программы для различных входных данных

Набор входных данных	Правильный результат	Верно?
x_a = 0 y_a = 0 x_b = 9 y_b = 1 x_c = 9 y_c = 1	Треугольник не существует	✔
x_a = 0 y_a = 1 x_b = 0 y_b = 2 x_c = 0 y_c = 5	Треугольник не существует	✔
x_a = 0 y_a = 10 x_b = -5 y_b = 0 x_c = 5 y_c = 0	Стороны: 10.0 11.18 11.18 Площадь: 50.0 Периметр: 32.361 Углы: 53.13 63.435 63.435	✔
x_a = 0 y_a = 1 x_b = 0 y_b = 0 x_c = 1 y_c = 0	Стороны: 1.0 1.414 1.0 Площадь: 0.5 Периметр: 3.414 Углы: 45.0 90.0 45.0	✔
x_a = 1 y_a = -5 x_b = 5 y_b = -1 x_c = 6 y_c = 8	Стороны: 9.055 14.318 8.062 Площадь: 3.5 Периметр: 31.435 Углы: 35.48 113.405 31.115	✔

Проверить программу для следующих входных данных:

Тест 1

```
x_a = 1
y_a = 1
x_b = 0
y_b = 0
x_c = 1
y_c = 1
Результат: Треугольник не существует
```

Тест 2

```
x_a = -12.8
y_a = 3.4
x_b = -7.7
y_b = 8.6
x_c = -14.6
y_c = -3.5
Результат: 0.9113 14.0042 22.8319
```

Формулы для вычислений

1. Радиус вписанной окружности вычисляется:

$$r = \sqrt{\frac{\left(\frac{p}{2} - a\right) \cdot \left(\frac{p}{2} - b\right) \cdot \left(\frac{p}{2} - c\right)}{\frac{p}{2}}}.$$

2. Радиус описанной вокруг треугольника окружности:

$$R = \frac{a \cdot b \cdot c}{4 \cdot s}.$$

3. Длины медиан вычисляются по формулам:

$$M_a = \frac{1}{2} \cdot \sqrt{2 \cdot (c^2 + b^2) - a^2},$$

$$M_b = \frac{1}{2} \cdot \sqrt{2 \cdot (c^2 + a^2) - b^2},$$

$$M_c = \frac{1}{2} \cdot \sqrt{2 \cdot (a^2 + b^2) - c^2}.$$

1.9. Варианты заданий для самостоятельного решения

Вариант 1

1. Найти уравнение прямой, проходящей через две различные точки, заданные своими координатами.
2. Определить, имеет ли функция **y = sin (x)** единственный корень на отрезке **[x,x+1]**. Если функция имеет единственный корень на данном интервале, то ее значения на его границах имеют разные знаки.

Вариант 2

1. Дан катет и гипотенуза прямоугольного треугольника. Найти величины острых углов треугольника (в градусах).
2. Даны два вещественных числа. Вычислить коэффициенты приведенного квадратного уравнения, корнями которого являются эти числа.

Вариант 3

1. Вычислить длину окружности, площадь круга и объем шара одного и того же заданного радиуса.
2. Идет **k**-ая секунда суток. Определить сколько полных часов и полных минут прошло к этому моменту суток.

Вариант 4

1. Дана окружность радиуса **r** и величина центрального угла (в градусах). Найти длину дуги и площадь сектора, ограниченных этим углом.
2. Составить программу для решения уравнения **a·x = b**, где **a** и **b** – заданные действительные числа.

Вариант 5

1. Дан цилиндр, заданный радиусом основания и высотой. Найти объем, площадь боковой и полной поверхностей цилиндра.
2. Даны три действительных числа **a, b, c.** Определить, являются ли они последовательными членами арифметической прогрессии.

Вариант 6

1. Дана равнобедренная трапеция, заданная длинами оснований и высотой. Найти площадь и периметр этой трапеции.
2. Составьте программу, которая для целого числа k (от 1 до 99), введенного вами, напечатает фразу «Мне k лет», при этом в нужных случаях слово «лет» заменяя на «год» или «года».

Вариант 7

1. Дан ромб со стороной **a** и острым углом **α**. Найти площадь ромба и длины его диагоналей.
2. Составить программу для решения уравнения $\mathbf{x^2+b\cdot x+c=0}$, где **c** и **b** – заданные действительные числа. Определить сколько корней имеет уравнение.

Вариант 8

1. Даны координаты центров двух окружностей $\mathbf{(x_1,y_1)}$ и $\mathbf{(x_2,y_2)}$, а также их радиусы $\mathbf{r_1}$ и $\mathbf{r_2}$ $\mathbf{(r_1>r_2)}$. Определить взаимное расположение окружностей.
2. На поле **(k, p)** шахматной доски расположен слон. Угрожает ли он полю **(m, n)**?

Вариант 9

1. Даны координаты двух точек $\mathbf{A(x_1,y_1)}$ и $\mathbf{B(x_2,y_2)}$. Какая из этих точек находится дальше от начала координат?
2. На поле **(k, p)** шахматной доски расположена ладья. Угрожает ли она полю **(m, n)**?

Вариант 10

1. Даны координаты трех вершин треугольника. Определить тип этого треугольника (прямоугольный, остроугольный или тупоугольный).
2. Составить программу для решения неравенства **a · x < b**, где **a** и **b** – любые действительные числа.

Вариант 11

1. Найти координаты точек пересечения прямой **y=kx+b** и окружности радиуса **R** с центром в начале координат.
2. Составьте программу, которая по введенному вами **k** – числу грибов печатает фразу «Мы нашли в лесу **k** грибов», причем согласовывает окончание слова «гриб» с числом **k.**

Вариант 12

1. Даны координаты двух точек $\mathbf{A(x_1,y_1)}$ и $\mathbf{B(x_2,y_2)}$. Пересекает ли отрезок AB оси координат?
2. На поле **(k, p)** шахматной доски расположен ферзь. Угрожает ли он полю **(m, n)**?

Вариант 13

1. Выяснить, принадлежит ли точка с координатами **(x,y)** кольцу с центром в начале координат с внешним радиусом **a** и с внутренним радиусом **b**.
2. Вычислить значение выражения (x, y, z – вводятся с клавиатуры):

$$y = \frac{\max(x,y,z)^2 - 2\min(x,y,z)}{\sin(2) + \dfrac{\max(x,y,z)}{\min(x,y,z)}}.$$

Вариант 14

1. Даны координаты центров двух окружностей **(0, 0)** и $\mathbf{(x_2,y_2)}$, а также их радиусы $\mathbf{r_1}$ и $\mathbf{r_2}$. Найти точки пересечения этих окружностей.
2. Составить программу для решения уравнения **a · |x| = b**, где **a** и **b** – заданные действительные числа.

Вариант 15

1. Даны координаты двух точек $\mathbf{A(x_1,y_1)}$ и $\mathbf{B(x_2,y_2)}$. Определить, параллелен ли этот отрезок **AB** одной из осей координат, если параллелен – вывести какой именно.
2. Составить программу для решения неравенства **a · |x| < b**, где **a** и **b** – заданные действительные числа.

Вариант 16

1. Дана точка **O(x,y)** на плоскости и длина диагонали квадрата **a**. Считая, что точка O является точкой пересечения диагоналей, отрезок **a** параллелен оси OX, найти координаты вершин квадрата, площадь и периметр квадрата.

2. Даны координаты двух точек **$A(x_1,y_1)$** и **$B(x_2,y_2)$.** Найти уравнение прямой, являющейся серединным перпендикуляром к этому отрезку.

Вариант 17

1. Даны координаты двух точек **$A(x_1,y_1)$** и **$B(x_2,y_2)$.** Определить, проходит ли отрезок **AB** через начало координат.

2. Даны три действительных числа **a, b, c.** Определить, являются ли они последовательными членами геометрической прогрессии.

Вариант 18

1. Дана точка O(x,y) на плоскости и длины двух диагоналей ромба a и b. Считая, что точка O является точкой пересечения диагоналей, отрезок a параллелен оси OX, отрезок b параллелен оси OY, найти координаты вершин ромба, длину стороны ромба и величины углов ромба.

2. Даны координаты точки **A(x, y).** Определить, в какой координатной плоскости она расположена. Если точка лежит на осях координат – вывести, какой именно оси она принадлежит.

2. ОБРАБОТКА ТАБЛИЧНЫХ ДАННЫХ И ИХ ВИЗУАЛИЗАЦИЯ

2.1. Списки

Списки в Python – упорядоченные изменяемые коллекции объектов произвольных типов (рисунок 4).

Рисунок 4. Список

Чтобы использовать списки, их нужно **создать**. Например, создадим пустой список:

```
x_list = []
```

или сразу сформируем список из нескольких элементов:

```
x_list = [23, 34.4, -0.45, -3.1, 5]
```

Нумерация элементов в списке начинается с 0, для того чтобы **обратиться** ко второму элементу списка **x_list,** используется запись:

```
x_list[1]
```

значение этого элемента равно 34.4.

Чтобы **заменить** элемент списка **x_list** номером 1, используется оператор присваивания:

```
x_list[1] = -23.14
```

в результате наш список будет иметь вид:

```
[23, -23.14, -0.45, -3.1, 5]
```

Длину списка вычисляют с помощью встроенной функции **len()**:

```
n = len(x_list)
```

В данном случае в переменную **n** будет занесено значение 5.

2.1.1. Оператор цикла, диапазон

Для перебора элементов списка используется **оператор цикла**, с помощью которого осуществляется заданное число повторений вложенного в цикл блока:

```
for ... :
    операторы
```

Для управления перебором в цикле используется **диапазон**, который описывает неизменяемую последовательность целых чисел от начального значения до границы (не включая ее) с указанным шагом:

```
range(нач_знач, граница, шаг)
```

Формируется диапазон следующим образом:

1. первое число диапазона – это начальное значение;
2. второе (и последующие) вычисляется как предыдущее плюс шаг;
3. число большее или равное границе в диапазон не включается, заполнение диапазона заканчивается.

Формирование значений диапазона рассмотрим на следующем примере:

```
range(3, 11, 2)
```

В этом диапазоне начальное значение равно 3, граница 10, шаг 2:

- первое число диапазона – это начальное значение: 3

- второе число вычисляется как предыдущее плюс шаг `3+2 = 5`: **`3 5`**
- третье число вычисляется как предыдущее плюс шаг `5+2 = 7`: **`3 5 7`**
- четвертое число вычисляется как предыдущее плюс шаг `7+2 = 9`: **`3 5 7 9`**
- пятое число вычисляется как предыдущее плюс шаг `9+2 = 11`, получили число большее или равное границе, поэтому оно в диапазон не включается, заполнение диапазона заканчивается: **`3 5 7 9`**

Таким образом, получили диапазон, состоящий из четырех чисел.

Описание диапазона может включать 3, 2 и одно значение в качестве параметров. Принято считать:

- если **`range()`** содержит 2 значения – пропущен шаг, который по умолчанию равен 1;
- если **`range()`** содержит 1 значение – пропущен шаг, который по умолчанию равен 1, и начальное значение, которое по умолчанию равно 0.

Например:

```
range(0, 5, 2) результат: 0, 2, 4
range(1, 4) результат: 1, 2, 3
range(5) результат: 0, 1, 2, 3, 4
```

Диапазоны используются в цикле **for**, в котором на каждом шаге управляющей переменной цикла присваивается очередное значение диапазона.

Рассмотрим, как работает цикл на следующем примере:

```
for i in range(0, 4):
    print(i)
```

Диапазон генерирует следующую последовательность чисел:

```
range(0, 4) → 0, 1, 2, 3
```

Выполняется цикл следующим образом:

Шаг 1. Из диапазона выбирается первое значение и заносится в переменную `i (i = 0)`, это значение становится доступным в теле цикла, с помощью оператора в консоль выводится значение переменной `i`:

```
0
```

Шаг 2. Из диапазона выбирается второе значение и заносится в переменную `i (i = 1)`, это значение становится доступным в теле цикла, с помощью оператора в консоль выводится значение переменной `i`:

```
0
1
```

Шаг 3. Из диапазона выбирается третье значение и заносится в переменную `i (i = 2)`, с помощью оператора в консоль выводится значение переменной `i`:

```
0
1
```

Шаг 4. Из диапазона выбирается четвертое значение и заносится в переменную `i (i = 3)`, с помощью оператора в консоль выводится значение переменной `i`:

```
0
1
2
3
```

Шаг 5. Элементы диапазона исчерпаны, значит, цикл заканчивает свою работу.

Другими словами, в цикле последовательно перебираются числа от 0 до 3, составляющие диапазон, последовательно заносятся в переменную **`i`** и выводятся в консоль.

Пример. Арифметическая прогрессия – это последовательность, в которой каждый следующий член можно найти, прибавив к предыдущему одно и то же число **h**. Вывести члены арифметической прогрессии, первый элемент которой равен целому числу **a**, последний не превышает целого числа **b**, шаг – целое число **h**.

Код:

```
# вводим параметры арифметической прогрессии
a = int(input("a = "))
b = int(input("b = "))
h = int(input("h = "))

# формируем члены арифметической прогрессии и выводим их
for x in range(a, b + 1, h):
    print(x)
```

Результат:

Тест 1

```
a = 6
b = 12
h = 3
6
9
12
```

Тест 2

```
a = 5
b = 12
h = 2
5
7
9
11
```

Задания для самопроверки

1. Для каждой числовой последовательности напишите диапазон, который ее формирует.

- 0, 2, 4, 6, 8;
- -6, -7, -8, -9, -10;
- 3, 6, 9, 12;
- 0, 1, 2, 3, 4, 5.

2. Заполните пропущенные фрагменты программы:

```
a = _
for i in range(2, _):
    a = a + a % i
    print(a)
```

так, чтобы в консоль был выведен следующий результат:

```
10
11
14
18
18
22
```

2.1.2. Генератор списка

Кроме явного создания списков как, например:

```
x_list = [3, 6, 23]
```

их можно создавать с помощью генератора.

Генератор списков – способ построить новый список, применяя выражение к каждому элементу диапазона или другого списка.

Для создания списка, элементы которого формируются в зависимости от значения **переменной-диапазона i**, используется следующий оператор:

```
список = [формула_от_i for i in range(описание диапазона)]
```

Формирование списка с помощью генератора списка рассмотрим на примере:

```
x_list = [i / (i + 1) for i in range(4)]
```

Диапазон в генераторе имеет вид:

```
range(4) → 0, 1, 2, 3
```

Последовательность генерации списка **x_list**:

- для формирования первого числа в списке из диапазона выбирается первое значение и заносится в переменную `i (i = 0)`, затем для этой переменной вычисляется выражение `i / (i + 1)= 0 / (0 + 1) = 0`:

```
x_list = [0]
```

- для формирования второго числа в списке из диапазона выбирается второе значение и заносится в переменную `i (i = 1)`, затем для этой переменной вычисляется выражение `i / (i + 1)= 1 / (1 + 1) = 0.5`:

```
x_list = [0, 0.5]
```

- для формирования третьего числа в списке из диапазона выбирается третье значение и заносится в переменную `i (i = 2)`, затем для этой переменной вычисляется выражение `i /(i + 1)= 2 / (2 + 1) = 0.666667`:

```
x_list = [0, 0.5, 0.666667]
```

- для формирования четвертого числа в списке из диапазона выбирается четвертое значение и заносится в переменную `i (i = 3)`, затем для этой переменной вычисляется выражение `i /(i + 1)= 3 / (3 + 1) = 0.75`:

```
x_list = [0, 0.5, 0.666667, 0.75]
```

- элементы диапазона исчерпаны, это значит, что генерация списка заканчивается:

```
x_list = [0, 0.5, 0.666667, 0.75]
```

Таким образом, получили список, состоящий из четырех чисел.

Генерация списков осуществляется с использованием диапазонов не только, но и **других списков**:

```
новый_список = [формула_от_x for x in уже_существующий_список]
```

Формирование списка на основе другого списка рассмотрим на примере:

```
x_list = [5, 8, -5]
y_list = [x ** 2 for x in x_list]
```

Последовательность генерации списка **y_list**:

- для формирования первого числа в списке **y_list** из списка **x_list** выбирается первое значение и заносится в переменную `x (x = 5)`, затем для этой переменной вычисляется выражение `x ** 2 = 5 ** 2 = 25`:

```
y_list = [25]
```

- для формирования второго числа в списке **y_list** из списка **x_list** выбирается второе значение и заносится в переменную `x (x = 8)`, затем для этой переменной вычисляется выражение `x ** 2 = 8 ** 2 = 64`:

```
y_list = [25, 64]
```

- для формирования третьего числа в списке **y_list** из списка **x_list** выбирается третье значение и заносится в переменную `x (x = -5)`, затем для этой переменной вычисляется выражение `x ** 2 = -5 ** 2 = 25`:

```
y_list = [25, 64, 25]
```

- элементы списка **x_list** исчерпаны, значит генерация списка **y_list** заканчивается:

```
y_list = [25, 64, 25]
```

Таким образом, получили список, состоящий из трех чисел.

Примеры создания списков:

```
x_list = [2 * i + 1 for i in range(5)] → [1, 3, 5, 7, 9]
x_list = [i for i in range(5,10,2)] → [5, 7, 9]
x_list = [2*i for i in range(3,-3,-1)] → [6, 4, 2, 0, -2, -4]
x_list = [3, 2, 7]
y_list = [x % 2 for x in x_list] → [1, 0, 1]
```

Пример. Дан список, в который занесены различные года. В современном календаре каждый год, номер которого делится на 4, является високосным, за исключением тех, которые делятся на 100 и при этом не делятся на 400. Создать новый список, каждый элемент которого равен `True`, если соответствующий год из списка високосный, `False` – если нет.

Код:

```
# список лет
year_list = [1900, 1945, 1980, 2000,2019, 2020, 2300]

# по году определяем, високосный он или нет
leap_year = [year % 4 == 0 and not (year % 100 == 0 and year % 400 != 0)
             for year in year_list]

# выводим результаты
print(year_list)
print(leap_year)
```

Результат:

```
[1900, 1945, 1980, 2000, 2019, 2020, 2300]
[False, False, True, True, False, True, False]
```

Задания для самопроверки

1. Создайте следующие списки с помощью генератора:

- [0, 0.1, 0.001, 0.0001, 0.00001];
- [0.5, 0.25, 0.125, 0.0625, 0.03125];
- [2.1, 5.1, 8.1, 11.1 ,14.1].

2. Список **y_list** формируется на основе списка **x_list**. Заполните пропущенные фрагменты кода:

```
x_list = [________]
y_list = [i + i / 10 for i in x_list]
print(y_list)
```

так, чтобы в консоль был выведен следующий результат:

```
[16.5, 22.0, 19.8, 18.7, 35.2]
```

2.1.3. Основные операции со списками

В Python предусмотрено большое количество различных функций для работы со списками. В таблице 14 приведены некоторые из них. В примерах функции применяются к следующему списку:

```
x_list = [1, 2, 3, 4, 1, 3]
```

Таблица 14 – Функции работы со списком

Функция	Описание	Пример
`x_list.append(x)`	Добавляет элемент в конец списка	`x_list.append(-1)` `[1, 2, 3, 4, 1, 3, -1]`
`x_list.extend(L)`	Расширяет список **x_list**, добавляя в конец все элементы списка L	`x_list.append([0, 5])` `[1, 2, 3, 4, 1, 3, 0, 5]`
`x_list.insert(i, x)`	Вставляет в **i**-й элемент значение **x**	`x_list.insert(2, 7)` `[1, 2, 7, 4, 1, 3]`
`x_list.remove(x)`	Удаляет первый элемент в списке, имеющий значение **x**.	`x_list.remove(3)` `[1, 2, 4, 1, 3]`
`x_list.pop(i)`	Удаляет **i**-й элемент и возвращает его. Если индекс не указан, удаляется последний элемент	`x_list.pop(3)` `[1, 2, 3, 1, 3]`
`x_list.index(x, start, end)`	Возвращает положение первого элемента со значением **x** (при этом поиск ведется от **start** до **end**). Если **start** и **end** опущены, то поиск осуществляется по всему списку	`x_list.index(1, 3, 5)` `4`
`x_list.count(x)`	Возвращает количество элементов со значением **x**	`x_list.count(1)` `2`
`x_list.sort()`	Сортирует список по возрастанию	`x_list.sort()` `[1, 1, 2, 3, 3, 4]`
`x_list.reverse()`	Разворачивает список	`x_list.reverse()` `[3, 1, 4, 3, 2, 1]`
`x_list.clear()`	Очищает список	`x_list.clear()` `[]`

Пример. В список занесен рост студентов группы. Найти самый маленький и самый большой рост в группе. Посчитать, сколько студентов имеют самый маленький и самый большой рост.

Код:

```
# создадим список с ростом студентов
height_list = [ 1.67, 1.85, 1.58, 1.89, 1.76, 1.89, 1.7, 1.89]

# отсортируем список по возрастанию
height_list.sort()

# первый элемент этого списка - самый маленький рост
height_min = height_list[0]

# последний элемент отсортированного списка - самый большой рост
height_max = height_list[len(height_list) - 1]

#посчитаем количество студентов с самым маленьким ростом
min_count = height_list.count(height_min)

#посчитаем количество студентов с самым большим ростом
max_count = height_list.count(height_max)

# выведем результат:
print("минимальный рост:", height_min, "количество:", min_count)
print("максимальный рост:", height_max, "количество:", max_count)
```

Результат:

```
минимальный рост: 1.58 количество: 1
максимальный рост: 1.89 количество: 3
```

Задание для самопроверки

Решить следующую задачу:

В список занесены численности городов некоторого региона:

```
population = [300125, 50000, 600050, 212000, 120000, 50000, 17000]
```

Из этого списка удалить одно максимальное и одно минимальное значения.

2.1.4. Вывод списка

Пусть список заполнен следующим образом:

```
x_list = [23, -23.14, -0.45, 15]
```

Для вывода этого списка можно использовать несколько способов:

- вывести список с помощью оператора **`print()`**:

```
print(x_list)
```

список будет выведен в одну строку в квадратных скобках через запятую:

```
[23, -23.14, -0.45, 15]
```

- вывести список в цикле:

```
for x in x_list:
    print(x)
```

в этом случае каждый элемент будет размещен на новой строке:

```
23
-23.14
-0.45
15
```

- вывести список с использованием его индексов:

```
for i in range(len(x_list)):
    print("x[", i + 1,"] =", x_list[i])
```

результат:

```
x[ 1 ] = 23
x[ 2 ] = -23.14
x[ 3 ] = -0.45
x[ 4 ] = 15
```

Задание для самопроверки

Написать фрагмент программы вывода списка `[23, 76, 12, -6, 50]`, чтобы результат в консоли выглядел следующим образом.

```
Список:
23 - элемент с индексом 0
76 - элемент с индексом 1
12 - элемент с индексом 2
-6 - элемент с индексом 3
50 - элемент с индексом 4
```

2.1.5. Ввод списка

Ввод списка может быть осуществлен разными способами.

Первый способ – **поэлементный ввод**, который реализуется по следующему алгоритму:

- спросить у пользователя количество элементов списка, занести значение в переменную **`n`** (или просто занести в **`n`** нужное значение);
- сформировать пустой список;
- реализовать перебор по переменной **`i`** в цикле для **`n`** элементов;
- ввести очередной элемент списка и занести его в переменную **`x`**;
- добавить **`x`** в конец списка.

Пример. Введем с клавиатуры 5 вещественных чисел и занесем их в список:

```
# количество элементов в списке
n = 5

# формируем пустой список
x_list = []
```

```
# цикл перебора индексов списка от 0 до 4
for i in range(n):
    # вводим очередной элемент списка
    x = float(input("Введите элемент списка: "))
    # добавляем введенный элемент в конец списка
    x_list.append(x)

#выводим список на экран
print(x_list)
```

Результат:

```
Введите элемент списка: 5.9
Введите элемент списка: -7.2
Введите элемент списка: 2.1
Введите элемент списка: 12.45
Введите элемент списка: -2.12
[5.9, -7.2, 2.1, 12.45, -2.12]
```

Вторым способом является ввод элементов списка **в одну строку** через пробел, а затем формирование списка на основе введенной строки. Алгоритм формирования списка:

- попросить пользователя ввести элементы списка в одну строку через пробел;
- полученную строку преобразовать в список с помощью метода **`split()`**, при этом в список будет занесены введенные числа в виде строк;
- преобразовать элементы списка из строковых переменных в вещественные (или целые).

Пример. Введем с клавиатуры вещественные числа и занесем их в список.

```
# вводим строку с числами
line = input("Введите элементы списка через пробел: ")

# преобразуем строку в список, элементы которого - строки
str_list = line.split()
print("Список из строк:", str_list)

# создаем новый список на основе уже сформированного,
# переводя каждый элемент в вещественное число
x_list = [float(x) for x in str_list]
print("Список из чисел:", x_list)
```

Результат:

```
Введите элементы списка через пробел: 45.7 34.2 -32.2 0.12 -3.1
Список из строк: ['45.7', '34.2', '-32.2', '0.12', '-3.1']
Список из чисел: [45.7, 34.2, -32.2, 0.12, -3.1]
```

Задание для самопроверки

Необходимо ввести два списка: первый список **`int_list`** состоит из целых чисел, второй **`float_list`** из вещественных. Исправьте фрагмент программы, в котором эти списки вводятся:

```
str_int = int(input(""))
str_float = input("")
int_list = str_int.split()
float_list = str_float.split()
float_list = [int(x) for x in float_list]
print(int_list)
print(float_list)
```

2.2. Задача, формирование таблицы значений функции

Задача. Построить таблицу значений функции $f(x)$ на интервале [a, b], количество значений в таблице **n**.

$$f(x) = \frac{1}{x+1} - \frac{x}{x-3}.$$

2.2.1. *Описание алгоритма*

Алгоритм построения таблицы значений функции на интервале [a, b], количество значений n:

- вычислить шаг h по формуле:

$$h = \frac{b-a}{n-1};$$

- создать список **x_list**, содержащий все значения аргумента функции **x**, список должен включать следующие значения:

$$a, a + h, a + 2 \cdot h, a + 3 \cdot h, \dots, b;$$

- создать список **y_list**, содержащий все значения функции, вычисленные от сформированных в списке аргументов, список должен включать:

$$f(x_{list[0]}), f(x_{list[1]}), .. f(x_{list[n-1]});$$

- вывести таблицу значений функции, в первом столбце которой будут располагаться элементы списка **x_list**, во втором – соответствующие значения списка **y_list**.

2.2.2. *Реализация*

Для построения таблицы значений функции используется следующая программа, пояснения к операторам даны в комментариях:

```
import math
# Опишем функцию f(x):
def f_x(x):
    try:
        y = 1 / (x+1) + x / (x-3)
    except:
        y = math.inf
    return y

# Введем границы интервала построения a, b
# и количество значений на интервале n:
a = float(input("a = "))
b = float(input("b = "))
n = int(input("n = "))

# Выполним проверку правильности ввода:
# количество значений должно быть больше 0,
# нижняя граница должна быть меньше верхней
if n < 0 or a >= b:
    print("Ошибочные входные данные")

# Если все введено верно
else:
    # вычислим шаг изменения аргумента функции:
    h = (b - a) / (n - 1)

    # создадим список значений
    x_list = [a + i * h for i in range(n)]

    # вычислим значение функции для каждого значения из списка
    f_list = [f_x(x) for x in x_list]

    # Выведем таблицу значений функции:
    for i in range(n):
        print ("%4.1f : %6.3f" % (x_list[i], f_list[i]))
```

Результат:

```
a = -5
b = 5
n = 21
-5.0 :  0.375
-4.5 :  0.314
-4.0 :  0.238
-3.5 :  0.138
-3.0 :  0.000
-2.5 : -0.212
-2.0 : -0.600
-1.5 : -1.667
-1.0 :    inf
-0.5 :  2.143
 0.0 :  1.000
 0.5 :  0.467
 1.0 :  0.000
 1.5 : -0.600
 2.0 : -1.667
 2.5 : -4.714
 3.0 :    inf
 3.5 :  7.222
 4.0 :  4.200
 4.5 :  3.182
 5.0 :  2.667
```

Оформим вывод в виде привычной таблицы (знак * при форматном выводе означает, что строку "-" нужно повторить указанное количество раз). Для этого заменим вывод таблицы в программе на следующий код:

```
# вывод шапки таблицы
print("-" * 17)
print ("| %4s | %6s |" % ("x", "f(x)"))
print("-" * 17)
# вывод содержимого таблицы
for i in range(n):
    print ("| %4.1f | %6.3f |" % (x_list[i], f_list[i]))
# вывод подчеркивания
print("-" * 17)
```

Результат:

```
a = -5
b = 5
n = 21
-----------------
|    x |   f(x) |
-----------------
| -5.0 |  0.375 |
| -4.5 |  0.314 |
| -4.0 |  0.238 |
| -3.5 |  0.138 |
| -3.0 |  0.000 |
| -2.5 | -0.212 |
| -2.0 | -0.600 |
| -1.5 | -1.667 |
| -1.0 |    inf |
| -0.5 |  2.143 |
|  0.0 |  1.000 |
|  0.5 |  0.467 |
|  1.0 |  0.000 |
|  1.5 | -0.600 |
|  2.0 | -1.667 |
|  2.5 | -4.714 |
|  3.0 |    inf |
|  3.5 |  7.222 |
```

```
|  4.0 |  4.200 |
|  4.5 |  3.182 |
|  5.0 |  2.667 |
-----------------
```

Задание для самопроверки

Написать программу для решения следующей задачи:

Для вычисления и прогноза численности населения Земли С.П. Капица предложил следующую формулу:

$$N(t) = \frac{C}{\tau} \cdot arcctg\left(\frac{T_1 - t}{\tau}\right),$$

где *t – год, для которого вычисляется численность населения;*
C – 172 миллиарда человек/лет;
T_1 – 2000 год;
τ – 45 лет.
Вычислить численность населения в заданные годы.
Реализовать задачу на основе шаблона.

```
import math

def compute_population(t):
    #вычислить численность населения для года t по формуле


#ввести строку с перечисленными через пробел годами
line = input("Введите список лет через пробел: ")

# преобразовать строку в список из строковых значений годов
years_str_list = line.split()

# вычислить количество элементов в списке
n = len(years_str_list)

# сформировать список years_list на основе years_str_list,
#преобразовав строковые значения в целые


# создать список population_list, каждый элемент которого вычисляется
# функцией compute_population от соответствующего года из years_list


# в цикле для каждого года вывести численность населения,
# для вывода использовать формат "%5d - %6.3f миллиард(ов)"
```

Результат

```
Введите список лет через пробел: 1220 2369 2636 1032 2657 198 1118
 1220 -  0.220 миллиард(ов)
 2369 - 11.544 миллиард(ов)
 2636 - 11.738 миллиард(ов)
 1032 -  0.178 миллиард(ов)
 2657 - 11.746 миллиард(ов)
  198 -  0.095 миллиард(ов)
 1118 -  0.195 миллиард(ов)
```

Пояснение

1. В модуле **math** функции **arcctg()** нет, но ее можно выразить через **arctg()** следующим образом:

$$arcctg(x) = \frac{\pi}{2} - actg(x).$$

2. Для вычисления **arctg()** в модуле **math** используется **atan()**.

2.3. Построение графиков и диаграмм

Для построения графиков и диаграмм используется библиотека matplotlib.pyplot. Эта библиотека не является стандартной, в большинстве сред разработки (в том числе и IDLE(Python)) ее необходимо установить. В Приложении Б приведена инструкция по установке дополнительных модулей.

Перед первым применением библиотеки ее необходимо подключить. Чтобы упростить обращение к функциям библиотеки, можно присвоить ей имя, например, **plt**:

```
import matplotlib.pyplot as plt
```

С помощью этой библиотеки создается графическая область, на которой отображаются следующие элементы:

- линии;
- оси координат;
- заголовок;
- легенды линий;
- многое другое.

Графическая область представляет собой прямоугольную сетку, координаты узлов которой определяются либо автоматически, либо могут быть установлены пользователем. В графическую область можно вывести точку с координатами **(x, y)** (рисунок 5). Значение **x** увеличивается по горизонтали слева направо, значение **y** увеличивается по вертикали снизу вверх, как это принято в декартовой системе координат.

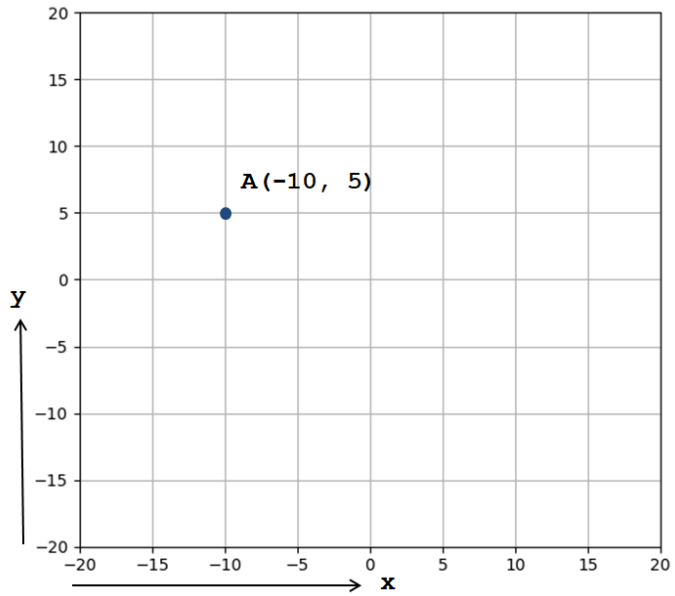


Рисунок 5. Область рисования

2.3.1. *Построение линий*

Для построения линий по точкам на плоскости используется метод **plot():**

```
plot(x, y, параметры),
```

где ***x*** – список значений по оси X,

y – список значений по оси Y;

параметры – формат отображения линии на графике (ее характеристики).

Например, построим график по точкам, координаты которых представлены в виде двух списков (последний оператор нужен, чтобы отобразить линию в области построения):

```
import matplotlib.pyplot as plt
plt.plot([1, 5, -3, -0.5], [1, 25, 9, 0.25 ])
plt.show()
```

В результате будет построена ломаная линия (рисунок 6).

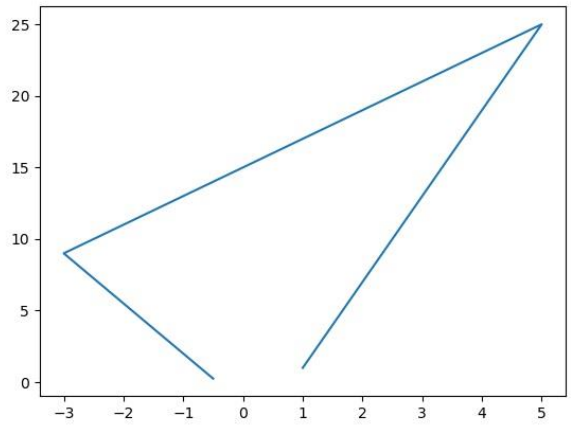


Рисунок 6. Линия, построенная по точкам

Библиотека matplotlib.pyplot позволяет устанавливать следующие характеристики линии (краткий справочник со значениями характеристик линий приведен в Приложении В):

- маркер точек, по которым строится линия, его размер;
- стиль (сплошная, пунктирная, отсутствует и пр);
- цвет (красный, зеленый и пр);
- толщина (в пикселях).

Это можно сделать, либо указав соответствующие параметры в методе **plot**(), как показано в таблице 15, либо использовать метод **setp**(), который устанавливает характеристики заданной первым параметром линии.

Таблица 15 – Установка характеристик линии в plot()

Метод с параметрами	Описание
`plot(x, y)`	Создает линию, в которой используются параметры отображения, установленные по умолчанию
`plot(x, y, "bo")`	Создает точки на графике, которые обозначаются маркерами круглой формы синего цвета
`plot(y)`	Создает линию, в качестве значения **x** используется интервал от 0 до количества значений в списке **y** минус 1
`plot(y, "r+")`	Создает точки на графике , которые обозначаются маркером «+» красного цвета
`plot(x, y, "go--", linewidth=2, markersize=12)`	Создает пунктирную линию зеленого цвета, толщиной 2, в качестве маркеров используются круги радиусом 12 пикселей
`plot(x, y, color="green", marker="o", linestyle="dashed",linewidth=2, markersize=12)`	Другой способ описания предыдущей линии

Для того чтобы использовать метод **`setp`**`()` для установки характеристик линий, необходимо

- дать линии имя:

```
line = plt.plot([1, 5, -3, -0.5], [1, 25, 9, 0.25 ])
```

- задать необходимые характеристики для линии, которая указывается первым параметром метода **`setp()`**:

```
plt.setp(line, color= "red", linewidth=2, marker="o" )
```

- показать линии в области построения:

```
plt.show()
```

Результат построения показан на рисунке 7.

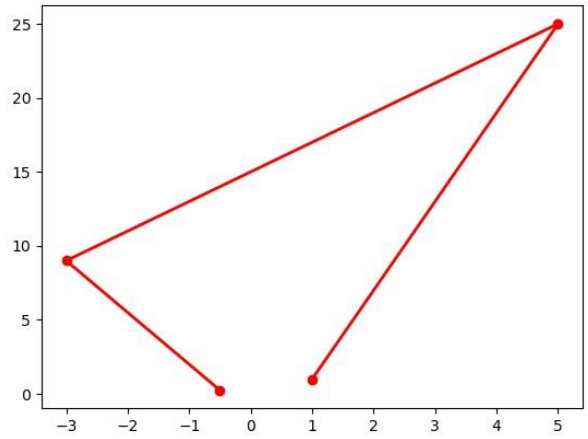


Рисунок 7. Красная линия с маркерами, построенная по точкам

2.3.2. Оси координат

Для построения и настройки координатных осей используется метод **gca**(). На графике могут быть отображены 4 оси, обращения к которым осуществляются следующим образом:

```
gca().spines["left"] – левая ось
gca().spines["bottom"] – нижняя ось
gca().spines["top"] – верхняя ось
gca().spines["right"] – правая ось
```

Для каждой из них можно указать ее **расположение** с помощью функции:

```
set_position("значение")
```

где значение – положение оси на графике, при этом можно задать:

zero – в точке с координатой 0 по соответствующей оси;

center – по центру области вывода графиков.

Управление видимостью оси осуществляется с помощью метода:

```
set_visible(значение)
```

в котором значение может быть либо **True** (если ось нужно показать, установлено по умолчанию), либо **False** (если ось нужно «спрятать»).

Например, нарисуем нашу ломаную в традиционной декартовой системе координат. Для этого уберем верхнюю и правую ось, а левую и нижнюю установим в позицию **zero**:

```
import matplotlib.pyplot as plt
# формируем линию
line = plt.plot([1, 5, -3, -0.5], [1, 25, 9, 0.25 ])

# задаем формат ее вывода
plt.setp(line, color="red", linewidth=2, marker="o" )

# устанавливаем две оси в положение zero
plt.gca().spines["left"].set_position("zero")
plt.gca().spines["bottom"].set_position("zero")

# скрываем остальные две
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)

# отображаем область построения
plt.show()
```

Результат выполнения программы показан на рисунке 8.

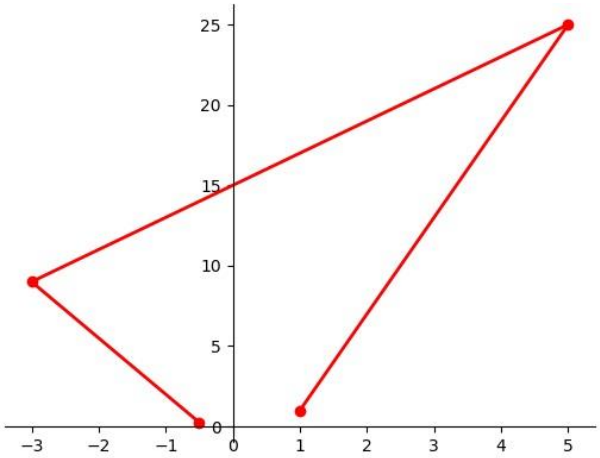


Рисунок 8. Линия в декартовой системе координат

2.3.3. Заголовок

В область построения можно вывести заголовок с помощью метода:

```
title("текст").
```

Если в нашу программу добавить оператор:

```
plt.title("Пример рисования линий")
```

то в области построения появится новый элемент – заголовок (рисунок 9)

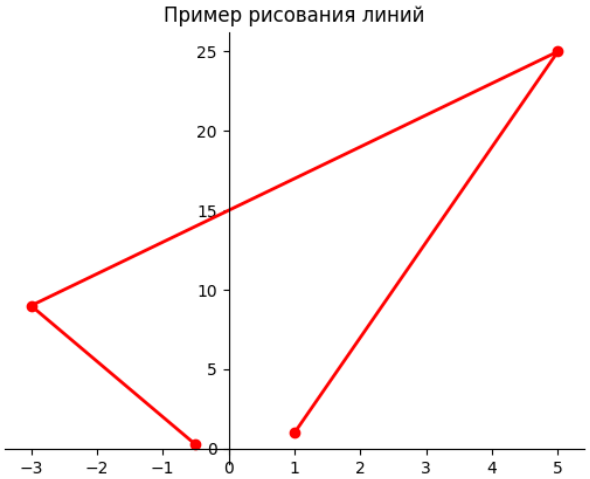


Рисунок 9. График с заголовком

2.3.4. Легенда

Легенда – это условное обозначение графиков. Легенда показывает названия и маркеры линий, используемых при их отображении в области построения.

Для каждой линии задается параметр – ее подпись, которая будет выводиться в качестве легенды. Это можно сделать двумя способами: либо задать параметр **`label`** при создании линии:

```
line_blue = plt.plot([1, 5, -3, -0.5], [1, 25, 9, 0.25], label='синяя линия')
```

либо задать этот параметр одновременно с установкой параметров стиля линии:

```
plt.setp(line_red, color="red", linewidth=2, label='красная линия')
```

Для вывода в область построения легенды используется метод **`legend()`**. В этом случае для расположения легенды выбирается «наилучшее» ее положение в области построения, то есть то, которое не перекрывает линии в области построения.

Если необходимо изменить положение вывода, используется метод:

```
legend(loc="значение")
```

Параметр `значение` определяет положение легенды на графике. Он может принимать следующие значения: `best`, `upper right`, `upper left`, `lower left`, `lower right`, `right`, `center left`, `center right`, `lower center`, `upper center`, `center`.

Например, построим две ломаные линии, зададим для них легенду.

```
import matplotlib.pyplot as plt

# создаем ломанные линии
line_blue = plt.plot([1, 5, -3, -0.5], [1, 25, 9, 0.25 ],
                     label='синяя линия')
line_red = plt.plot([-6, -5, 0, 8], [-5, -2, -3, 4 ])

# задаем стили для линий
plt.setp(line_blue, color="blue", linewidth=2, marker="v" )
plt.setp(line_red, color="red", linewidth=2, marker="o",
         label='красная линия')

# отображаем легенду
plt.legend()

# показываем область рисования
plt.show()
```

Результат выполнения программы показан на рисунке 10.

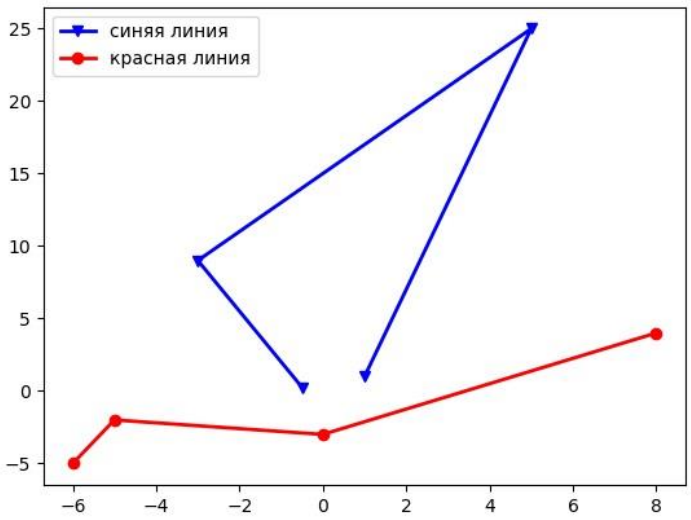


Рисунок 10. Вывод линий с легендой

Задания для самопроверки

1. Дан фрагмент программного кода:

```
import matplotlib.pyplot as plt
x_list = [1, 5, -3]
y_list = [-5, 6, 1 ]
#функция создания линии
plt.show()
```

Вставить оператор вместо комментария, чтобы была выведена курсивная ломаная линия зеленого цвета, маркеры отображаются в виде знака плюс.

2. Дан фрагмент программного кода, в котором строится линия по координатам точек, занесенных в списки **`x_list`** и **`y_list`**.

```
import matplotlib.pyplot as plt
plt.plot(x_list, y_list, 'ro--', linewidth=4, markersize=10)
```

Выберите фрагменты кода, с помощью которых можно построить линию с теми же характеристиками:

```
a.  plt.plot(x, y, color='red', marker='o',
```

```
            linestyle='dashed',linewidth=4, markersize=10)
b.  plt.plot(x, y, color='red', marker='o', linewidth=4, markersize=10)
c.  plt.plot(x, y, color='red', marker='o', linestyle='solid',
            linewidth=4, markersize=10)
d.  line =  plot(x, y)
   plt.setp(line, color='red', marker='o', linestyle='dashed',
            linewidth=4, markersize=10)
```

2.4. Задача, построение графиков функций одной переменной

Задача. Построить графики двух функций на интервале [a,b]:

$$f(x) = x^3 - 6x^2 + x + 5,$$
$$y(x) = (x-2)^2 - 6.$$

2.4.1. Описание алгоритма

Алгоритм построения графика функции $f(x)$ на интервале [a, b]:

- определить количество точек построения **n**;
- вычислить шаг **h** по формуле:

$$h = \frac{b-a}{n-1};$$

- создать список **x_list**, содержащий все значения аргумента функции **x**, список должен включать следующие значения:

$$a, a + h, a + 2 \cdot h, a + 3 \cdot h, \dots, b;$$

- создать список **y_list**, содержащий все значения функции, вычисленные от сформированных в списке аргументов, список должен включать:

$$f(x_{list[0]}), f(x_{list[1]}), .. f(x_{list[n-1]});$$

- построить линию графика функции, задать характеристики;
- вывести координатные оси;
- при необходимости добавить заголовок и легенду;
- отобразить область построения.

2.4.2. Реализация

Для построения графиков двух функций используется следующая программа, пояснения к операторам даны в комментариях:

```
# подключим модуль для построения графиков функций, дадим ему имя plt:
import matplotlib.pyplot as plt

# создадим две функции:
def f_x(x):
   y = x ** 3 - 6 * x ** 2 +  x + 5
   return y

def y_x(x):
    y = (x - 2) ** 2 -6
    return y

# Зададим интервал построения функции и количество точек построения:
Вычислим шаг:
a = -2
b = 6
n =100

# Вычислим шаг:
h = (b-a)/(n-1)
```

```
# Сформируем список со значениями аргумента x:
x_list = [a + h * i for i in range(n)]

# Сформируем списки со значениями функций f(x) и (x):
f_list = [f_x(x) for x in x_list]
y_list = [y_x(x) for x in x_list]

#Построим линии графиков функций, зададим подпись для вывода легенды:
line_f = plt.plot(x_list, f_list, label='f(x)')
line_y = plt.plot(x_list, y_list, label='y(x)')

#Зададим стили линий:
plt.setp(line_f, color="blue", linewidth=2)
plt.setp(line_y, color="red", linewidth=2)

#Выведем 2 оси, установим для них позицию zero:
plt.gca().spines["left"].set_position("zero")
plt.gca().spines["bottom"].set_position("zero")
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)

#Выведем легенду и заголовок в область построения:
plt.legend()
plt.title("Графики функций")

# Отобразим область построения:
plt.show()
```

На рисунке 11 показан результат выполнения программы.

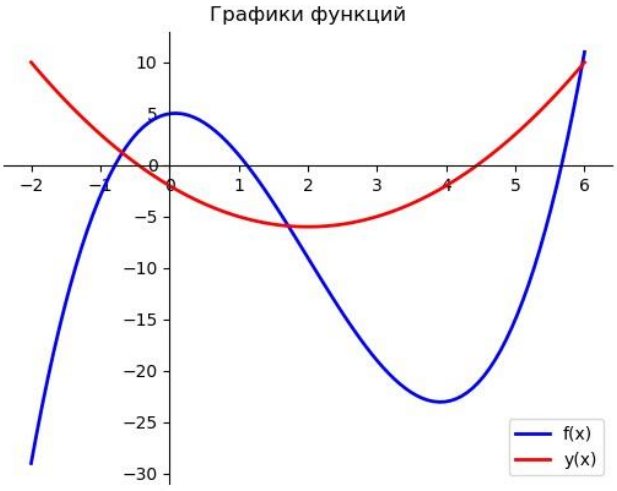


Рисунок 11. Результат выполнения программы

Задание для самопроверки

Постройте графики следующих функций на интервале от -240^o до 360^o:

$$f(x) = e^{\cos(x)} + ln(sin^2(0.8x) + 1) \cdot \cos(x),$$

$$y(x) = -\ln((\cos(x) + \sin(x))^2 + 1.7) + 2.$$

На оси OX должны быть отмечены значения **в градусах**.

Пояснение. Для вычисления функций **`cos()`** и **`sin()`** значения **`x`** нужно перевести в радианы.

По графику ответьте на вопросы.

- В точке x = 71^o сравните значения функции, вставьте знак: f(x) ___ y(x);
- Какая функция на этом интервале имеет наименьшее значение? Укажите название функции ___.

- Какая функция на данном интервале принимает только отрицательные значения? Укажите функцию или слово «Нет» ___.
- Сколько решений имеет уравнение f(x)-y(x) = 0 на данном интервале? Для этого посчитайте количество пересечений графиков f(x)и y(x): ___.

2.5. Задача: построение графиков функций одной переменной с точками разрыва

Задача. Построить график функции на интервале `[a,b]`:

$$f(x) = x + \frac{1}{x}.$$

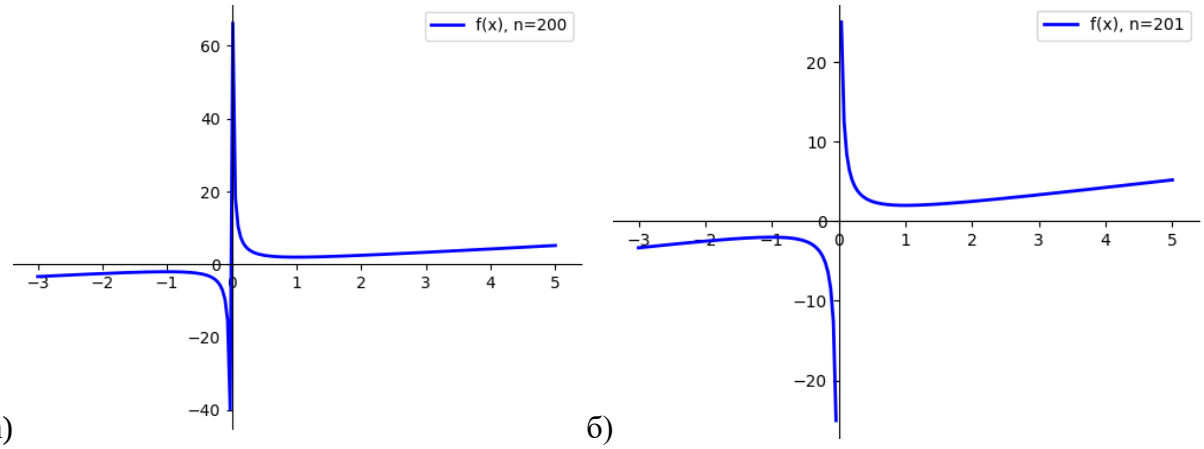
2.5.1. Описание алгоритма

Функция $f(x)$ имеет точку разрыва при $x_b = 0$, так как в этой точке знаменатель обращается в ноль. Если при описании функции вставить обработку исключений и при делении на ноль вернуть константу `math.inf`, то можно использовать алгоритм построения функции, описанный в предыдущем пункте. Описание функции в это случае будет иметь вид:

```
import math

def f_x(x):
    try:
        y = x + 1 / x
    except:
        y = math.inf
    return y
```

Но при некоторых значениях n график будет построен неверно. Сравните результат построения графика при n = 200 (рисунок 12а) и при n = 201 (рисунок 12б).



а) б)

Рисунок 12. Результат построения функции: а) n = 200; б) n = 201

Другим способом построения графика такой функции является построение двух частей графика на двух интервалах, если $x_b \in [a, b]$:

$[a, x_b - \varepsilon]$ и $[x_b - \varepsilon, b]$,

где ε – положительное число, которое выбирается произвольно, например, оно может быть равно шагу построения функции.

Алгоритм построения графика функции $f(x)$ на интервале [a, b] с точкой разрыва x_b, $x_b \in [a, b]$.

- определить количество точек построения **n**;
- вычислить шаг **h** по формуле:

$$h = \frac{b-a}{n-1};$$

- вычислить количество точек построения, расположенных до точки разрыва:

$$n_1 = int\left(\frac{x_b - a}{h}\right) - 1;$$

- вычислить количество точек построения, расположенных после точки разрыва:

$$n_2 = n - n_1 - 2;$$

- создать список **x_list_1**, содержащий значения аргумента функции **x**, на интервале $[a, x_b - h]$ список должен включать следующие значения, количество элементов в этом списке $\boldsymbol{n_1}$:

$$a, a + h, a + 2 \cdot h, a + 3 \cdot h, \dots, x_b - h;$$

- создать список **x_list_2**, содержащий значения аргумента функции **x**, на интервале $[x_b + h, b]$ список должен включать следующие значения, количество элементов в этом списке $\boldsymbol{n_2}$:

$$x_b + h, x_b + 2 \cdot h, x_b + 3 \cdot h, \dots, b;$$

- создать список **y_list_1**, содержащий все значения функции, вычисленные от сформированных в списке **x_list_1** аргументов, список должен включать:

$$f(x\_list\_1[0]), f(x\_list\_1[1]), .. f(x\_list\_1[n_1 - 1]);$$

- создать список **y_list_2**, содержащий все значения функции, вычисленные от сформированных в списке **x_list_2** аргументов, список должен включать:

$$f(x\_list\_2[0]), f(x\_list\_2[1]), .. f(x\_list\_2[n_2 - 1]);$$

- построить две линии по спискам (**x_list_1, f_list_1)** и (**x_list_2, f_list_2)**, задать одинаковые характеристики линиям;
- вывести координатные оси;
- при необходимости добавить заголовок и легенду для одной из линий;
- отобразить область построения.

2.5.2. Реализация

Для построения графиков двух функций используется следующая программа, пояснения к операторам даны в комментариях:

```
# подключим модуль для построения графиков функций, дадим ему имя plt:
import matplotlib.pyplot as plt

# создадим функцию:
def f_x(x):
    y = x + 1 / x
    return y

# Зададим интервал построения функции и количество точек построения
a = -6
b = 7

# Вычислим шаг:
n = 200

# Точка разрыва
x_b = 0

# Вычислим шаг:
h = (b-a)/(n-1)

# Вычислим n1
n_1 = int((x_b - a ) / h) - 1
```

```
# Вычислим n2
n_2 = n - n_1 - 2

# Сформируем список со значениями аргумента x до точки разрыва:
x_list_1 = [a + h * i for i in range(n_1)]

# сформируем список со значениями аргумента x до точки разрыва:
x_list_2 = [x_b +  h + h * i for i in range(n_2)]

# Сформируем списки со значениями двух частей функций f(x):
f_list_1 = [f_x(x) for x in x_list_1]
f_list_2 = [f_x(x) for x in x_list_2]

#Построим линии графиков функций, зададим подпись для вывода легенды:
line_f_1 = plt.plot(x_list_1, f_list_1, label='f(x)=x+1/x')
line_f_2 = plt.plot(x_list_2, f_list_2)

#Зададим стили линий:
plt.setp(line_f_1, color="red", linewidth=2)
plt.setp(line_f_2, color="red", linewidth=2)

#Выведем 2 оси, установим для них позицию zero:
plt.gca().spines["left"].set_position("zero")
plt.gca().spines["bottom"].set_position("zero")
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)

#Выведем легенду и заголовок в область построения:
plt.legend()
plt.title("График функции с разрывом")

# Отобразим область построения:
plt.show()
```

На рисунке 13 показан результат выполнения программы.

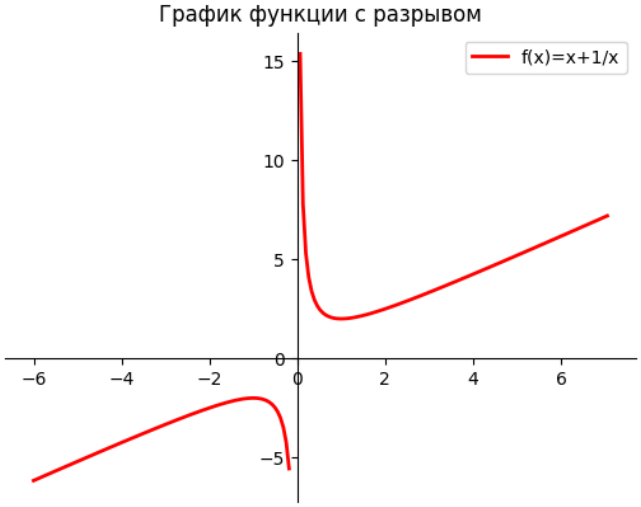


Рисунок 13. Результат выполнения программы

Задание для самопроверки

Постройте график функции $f(x)$ с двумя точками разрыва интервале от [−3, 3]:

$$f(x) = \frac{1}{x^2 - 1}.$$

Результат построения показан на рисунке 14.

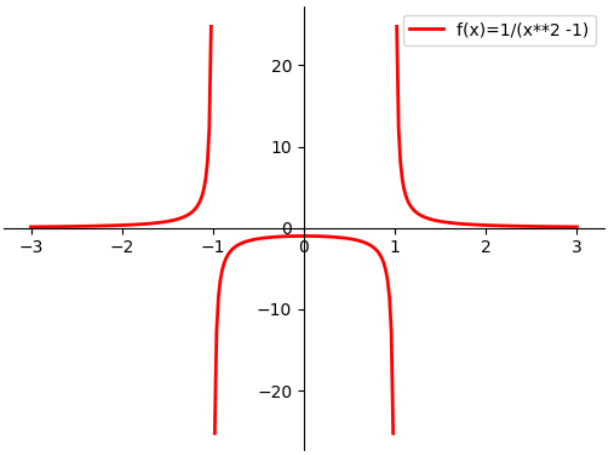


Рисунок 14. Результат построения функции

2.6. Рисование плоских фигур

Плоские геометрические фигуры создаются с помощью библиотеки matplotlib.patches, из которой можно импортировать шаблоны для различных геометрических фигур (все фигуры перечислить **на одной** строке):

```
from matplotlib.patches import Circle, Wedge, Polygon, Ellipse,
    Rectangle, Arc, Path, PathPatch
```

Для отображения фигур в области рисования используется библиотека matplotlib.pyplot:

```
import matplotlib.pyplot as plt
```

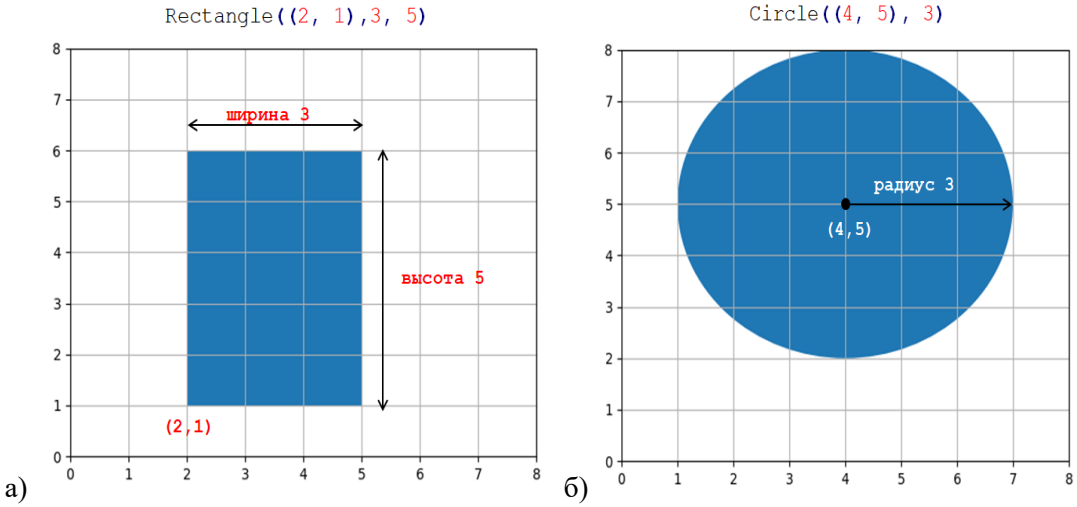
2.6.1. Создание геометрической фигуры

Каждая геометрическая фигура создается методом библиотеки matplotlib.patches, который имеет два типа параметров:

- положение фигуры;
- характеристики фигуры.

Каждая фигура имеет свой набор параметров, описывающих **положение фигуры** в области рисования. В зависимости от типа фигуры это могут быть координаты точек, радиусы, величины углов и пр. Рассмотрим методы создания наиболее часто используемых геометрических фигур.

`Rectangle((x, y), width, height)` – создает прямоугольник, левый верхний угол которого располагается в точке с координатами `(x, y)`, высота которого равна `width`, а длина – `height` (рисунок 15а).



а) б)

Рисунок 15. Фигуры: а) прямоугольник; б) окружность

`Circle((x, y), radius)` – создает круг с центром в точке `(x, y)`, радиуса `radius` (рисунок 15б).

`Ellipse((x, y), width, height)` – создает эллипс с центром в точке `(x, y)`, диаметр которого по горизонтальной оси равен `width`, а диаметр по вертикальной – `height` (рисунок 16а).

`Polygon([(x0, y0), (x1, y1), (x2, y2), ... ], closed)` – создает ломаную линию по точкам `(x0, y0), (x1, y1), (x2, y2), ...`, если параметр `closed` равен **`True`** (или он отсутствует), первая точка соединяется с последней, в противном случае `(closed=` **`False`**`)`, ломаная линия остается незамкнутой (рисунок 16а).

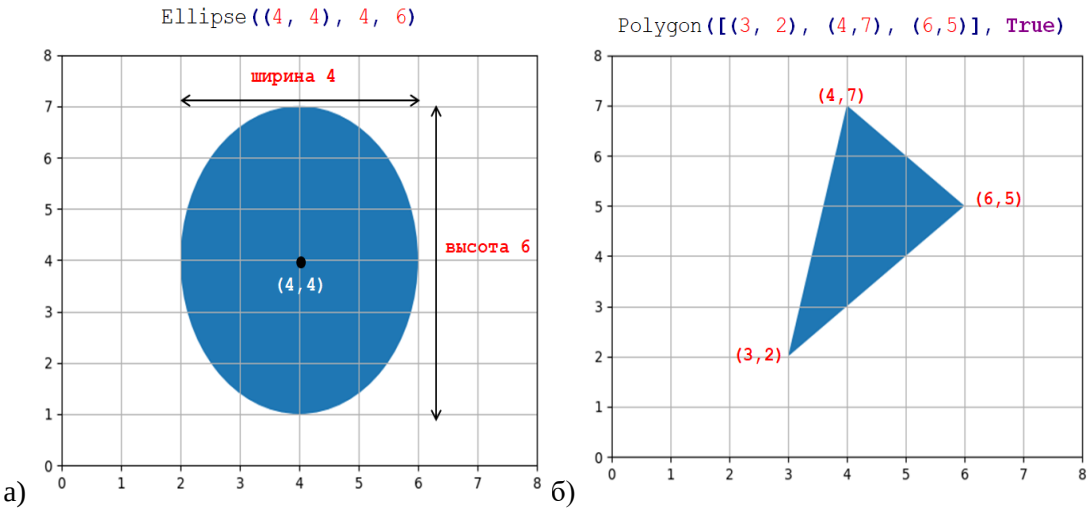


Рисунок 16. Фигуры: а) эллипс; б) полигон

`Wedge((x, y), radius, t1, t2)` – создает сектор с центром в точке `(x, y)`, радиуса `radius`, ограниченный линиями углов `t1 и t2` (рисунок 17а).

`Arc((x, y), width, height, angle, t1, t2)` – создает дугу с центром в точке `(x, y)`, диаметр которой по горизонтальной оси равен `width`, а диаметр по вертикальной – `height`, угол поворота дуги относительно центра `angle`, дуга ограничена линиями углов `t1 и t2` (рисунок 17б).

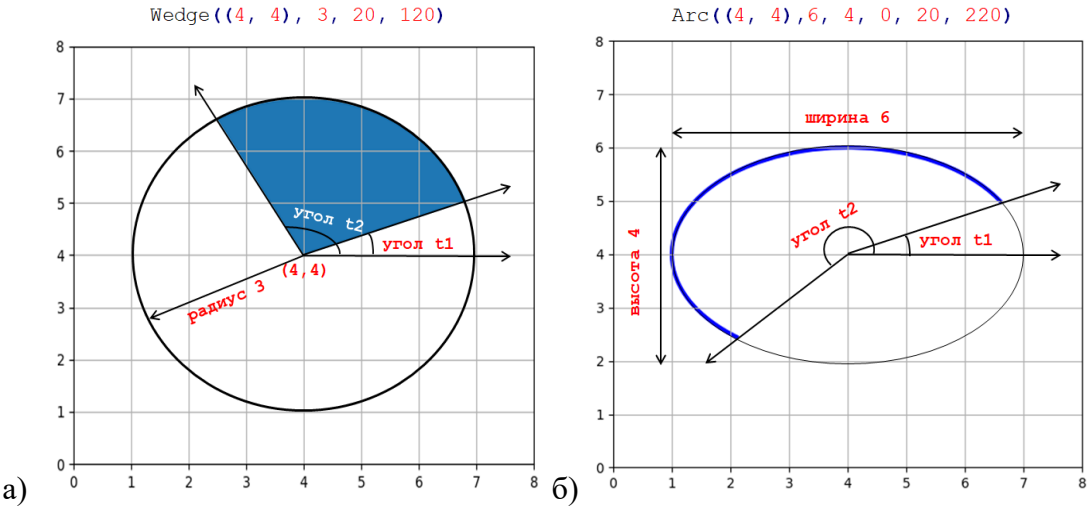


Рисунок 17. Фигуры: а) сектор; б) дуга

Совокупность линии можно рисовать с помощью так называемых путей (`Path`). Путь представляет собой список опорных точек (как при создании полигона) и список кодов, задающих, что с этими вершинами делать.

Коды – это числовые константы, определенные в модуле `Path`, которые могут принимать значения:

- `MOVETO` (переместиться в точку без рисования линии);
- `LINETO` (провести линию в заданную точку);
- другие.

Коды можно записывать двумя способами:

- полной ссылкой (`matplotlib.path.Path.MOVETO, matplotlib.path.Path.LINETO`);
- цифрой 1 или 2 соответственно.

Свойства, описывающие **стиль вывода** каждой фигуры, общие для всех фигур. Основные свойства представлены в таблице, большинство из них могут быть описаны в полной или сокращенной форме (таблица 16). Возможные значения характеристик приведены в Приложении В.

Таблица 16 – Стили вывода фигур

Характеристика	Описание
`facecolor="цвет"` `fc="цвет"`	Цвет заливки фигуры
`linewidth=значение` `lw=значение`	Толщина границы фигуры
`edgecolor="цвет"` `ec="цвет"`	Цвет границы фигуры
`fill=значение`	Фигура закрашенная (True, по умолчанию) или незакрашенная(False)

2.6.2. Алгоритм рисования

Все геометрические объекты рисуются в несколько шагов.

1. Импортируются все необходимые модули, функции, шаблоны:

```
from matplotlib.patches import  Circle
import matplotlib.pyplot as plt
```

2. Определяются координаты окна, в котором будет создаваться изображение, например, следующие операторы создают окно, координаты которого изменяются от 0 до 12 по горизонтали и вертикали:

```
plt.xlim(0, 12)
plt.ylim(0, 12)
```

3. Создается область для рисования, связанная с осями координат с помощью метода gca() :

```
ax = plt.gca()
```

4. Создается геометрическая фигура на основе описания из модуля matplotlib.patches. Например, следующий оператор создает круг с центром в точке (6, 7) и радиусом 5 и заносит результат в переменную **`circle`**:

```
circle = Circle((6, 7),5)
```

5. Созданная фигура добавляется в область **`ax`** с помощью метода **`add_patch()`**:

```
ax.add_patch(circle)
```

6. Рисунок отображается в графическом окне:

```
plt.show()
```

Результат выполнения программы показан на рисунке 18.

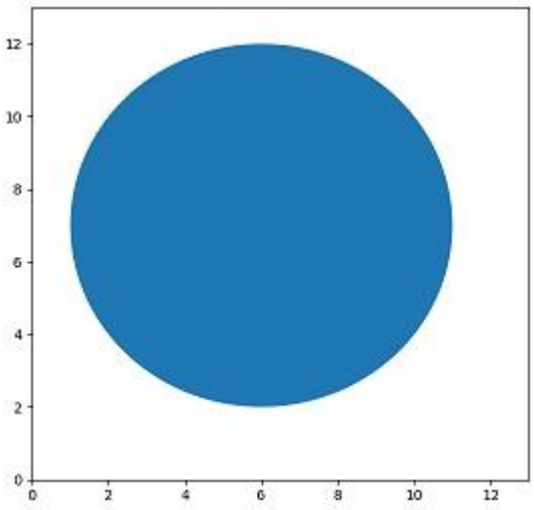


Рисунок 18. Результат выполнения программы

Пример. Создадим рисунок (рисунок 19) с помощью линий, объединенных в пути.

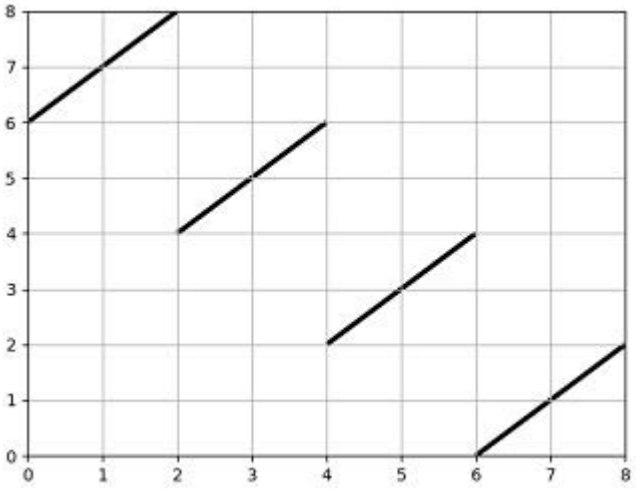


Рисунок 19. Шаблон рисунка

Для создания рисунка используется следующий код, пояснения даны в комментариях:

```
# Импортируем необходимые функции и библиотеки:
from matplotlib.patches import Path, PathPatch
import matplotlib.pyplot as plt

# Настроим область рисования:
n = 8
m  = 8
plt.xlim(0, n)
plt.ylim(0, m)
ax = plt.gca()

# Создадим список опорных точек:
vertices = [(0, 6), (2, 8), (2, 4), (4, 6), (4, 2), (6,4), (6, 0), (8,2)]

# Создадим список кодов (установить курсор в первую точку (0, 6) - MOVETO,
# провести линию во вторую (2, 8) - LINETO,
# установить курсор в третью (2, 4) - MOVETO,
# провести линию в четвертую (4, 6) - LINETO и т.д.):
codes = [1, 2, 1, 2, 1, 2, 1, 2]

# Создадим переменную path с помощью метода matplotlib.patches.Path,
# указав список вершин (vertices) и список кодов (codes):
```

```
path = Path(vertices, codes)

# Создадим фигуру, представляющую собой совокупность линий, толщиной 3 px:
path_patch = PathPatch(path, lw=3)

#Добавим созданную фигуру в область ax, удалим оси и покажем рисунок
ax.add_patch(path_patch)
ax.axes.set_axis_off()
plt.show()
```

Результат выполнения программы показан на рисунке 20.

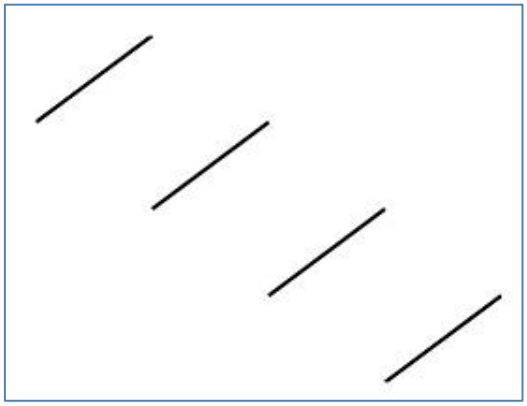


Рисунок 20. Результат выполнения программы

Задание для самопроверки

В программе создаются плоские фигуры с помощью методов библиотеки `matplotlib.patches`:

```
from matplotlib.patches import Circle, Wedge, Ellipse, Arc, Rectangle
```

Выберите **неверные** операторы:

```
a.  figure = wedge((4, 4), 2, -90, 90)
b.  figure = Wedge((4, 4), 2, -90, 90)
c.  figure = Wedge((4, 4), 2, -90)
d.  figure = Rectangle((10, 12), 5, 8)
e.  figure = Ellipse(5, 1, 2, 3)
```

2.7. Задача, рисование фигуры из геометрических примитивов

2.7.1. Алгоритм рисования изображения из геометрических фигур

1. Нарисовать изображение на клетчатом листе бумаге, пронумеровать клетки от 0 до **n** по горизонтали, от 0 на **m** вертикали (в нашем случае от 0 до 13 и от 0 до 15), нумерация клеток может начинаться с любых значений, но удобнее взять 0 (рисунок 21).

2. Обозначить координаты опорных точек, образующих плоские фигуры, при необходимости указать другие характеристики (рисунок 21):

- полигон – координаты опорных точек (на рисунке туловище кота);
- круг – координаты центра и радиус (глаза, зрачки и нос);
- эллипс – координаты центра, диаметр по горизонтали и вертикали (передние лапы);
- сектор – координаты центра, радиус, угол начала сектора, угол его завершения (задние лапы);
- дуга – координаты центра, радиус по горизонтали, радиус по вертикали, угол поворота, угол начала дуги, угол ее завершения (улыбка);
- набор линий – координаты опорных точек (усы, линия между носом и улыбкой).

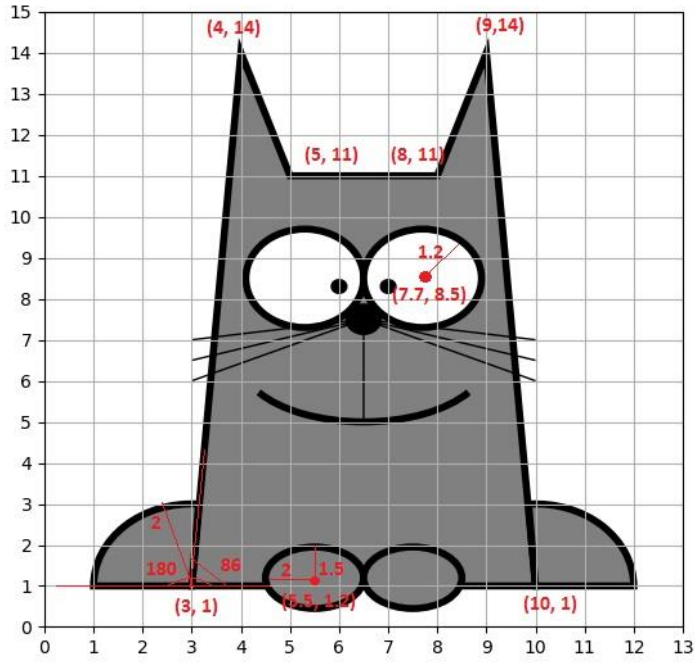


Рисунок 21. Рисунок с опорными точками

3. Определить стили вывода каждой фигуры:

- толщина границы набора линий – 1 пиксель, остальные – 4 пикселя;
- цвет границ всех фигур – черный;
- цвета закраски – черный (для зрачков и носа), белый – для глаз, серый – для туловища и лап;
- все фигуры, кроме дуги и набора линий, – замкнутые.

4. Реализовать каждую фигуру с помощью шаблонов плоских фигур из `matplotlib.patches`.

2.7.2. Реализация

Для создания рисунка используется следующий код, пояснения даны в комментариях:

```
# Импортировать необходимые фигуры рисования,
# подключить модуль для рисования:
from matplotlib.patches import Circle, Wedge, Polygon, Ellipse, Arc,
     Path, PathPatch
import matplotlib.pyplot as plt

# Реализовать функцию для рисования фигуры,
# параметром передать область рисования:
def draw_cat(ax):
    # туловище
    poly = [(3, 1), (4, 14), (5, 11), (8, 11), (9, 14), (10, 1)]
    polygon = Polygon(poly, fc="grey", ec="black", lw=4)
    ax.add_patch (polygon)

    # глаза
    circle = Circle((5.3, 8.5), 1.2, fc="white", ec="black", lw=4)
    ax.add_patch (circle)

    circle = Circle((7.7, 8.5), 1.2, fc="white", ec="black", lw=4)
    ax.add_patch (circle)

    # зрачки
    circle = Circle((6, 8.3), 0.1, fc="black", ec="black", lw=4)
    ax.add_patch (circle)

    circle = Circle((7, 8.3), 0.1, fc="black", ec="black", lw=4)
    ax.add_patch (circle)
```

```
    # нос
    circle = Circle((6.5, 7.5), 0.3, fc="black", ec="black", lw=4)
    ax.add_patch (circle)
    # задние лапы
    wedge = Wedge((3, 1), 2, 86, 180, fc="grey", ec="black", lw=4)
    ax.add_patch (wedge)

    wedge = Wedge((10, 1), 2, 0, 94, fc="grey", ec="black", lw=4)
    ax.add_patch (wedge)
    # передние лапы
    ellipse = Ellipse((5.5,1.2), 2, 1.5, fc="grey", ec="black", lw=4)
    ax.add_patch (ellipse)
    ellipse = Ellipse((7.5,1.2), 2, 1.5, fc = "grey", ec="black", lw=4)
    ax.add_patch (ellipse)
    # улыбка
    arc =  Arc((6.5, 6.5), 5, 3, 0, 200, 340, lw=4, fill=False)
    ax.add_patch(arc)
    # линия между носом и улыбкой, усы
    vertices = [(6.5, 5), (6.5, 7.5), (10, 6), (6.5, 7.5), (10, 6.5),
                (6.5, 7.5), (10, 7), (6.5, 7.5), (3, 6), (6.5, 7.5),
                (3, 6.5), (6.5, 7.5), (3, 7)]
    # число 1 соответствует команде matplotlib.path.Path.MOVETO
    # число 2 соответствует команде matplotlib.path.Path.LINETO
    codes = [1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
    path = Path(vertices, codes)
    path_patch = PathPatch(path, fill=False, lw=1)
    ax.add_patch(path_patch)

# Установить размер и координаты углов для области рисования
# в соответствии с рисунком 21:
n = 13
m  = 15
plt.xlim(0, n)
plt.ylim(0, m)

# Создать область, связанную с осями координат,
# куда будут выводиться плоские фигуры (ax):
ax = plt.gca()

# вызвать функцию рисования (draw_cat):
draw_cat(ax)

#Удалить оси координат и показать рисунок:
ax.axes.set_axis_off()
plt.show()
```

Результат выполнения программы показан на рисунке 22.

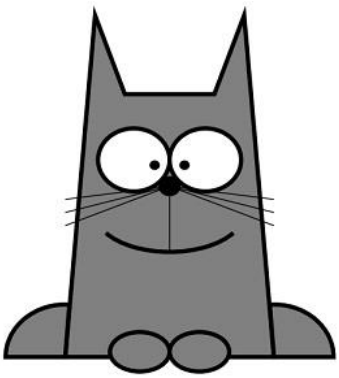


Рисунок 22. Результат выполнения программы

Задание для самопроверки

Допишите программу так, чтобы в результате ее выполнения получалось изображение, показанное на рисунке 23.

```
from matplotlib.patches import Path, PathPatch
import matplotlib.pyplot as plt

n = # указать размер области
m  = # указать размер области

plt.xlim(0, n)
plt.ylim(0, m)
ax = plt.gca()

# создать массив точек
vertices = [(1, 3), (7, 2), (6, 1), (3, 1), (1, 3), (4, 2.5), (4,6),
            (7, 3), (4,2.5)]

#создать список кодов для последовательности рисования:
codes = # записать список

#создать объект path
path = Path(vertices, codes)

#создать фигуру
path_patch = PathPatch(path, lw=3)

# Добавить созданную фигуру в область ax:

plt.show()
```

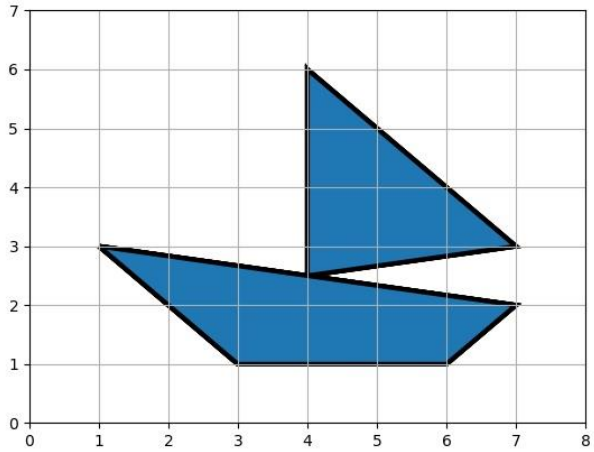


Рисунок 23. Изображение фигуры

2.8. Варианты заданий для самостоятельного решения

Вариант 1

1. Описать функцию $f(x, y, z)$, при создании включить обработку исключительных ситуаций:

$$f(x,y,z) = \left| \left(y - \sqrt{1 + \sqrt{\cos^2\left(4x + \frac{\pi}{\sqrt{3}}\right)}} \right) \cdot \left(z - \frac{1}{z + \frac{x}{y}} \right) \right| + \sin\left(\cos\sqrt[3]{(z+1)^2}\right)$$

а) посчитать значения функции:

$x = 4.5,\ y = \sqrt{\pi},\ z = 4;$

$x = 1,\ y = 0,\ z = 5.$

б) вывести таблицу значений функции $f(x, y, z)$, состоящей из N=20 строк, в точках $(x_i, y_i, z_i),\ i = 1..N$, где

$$x_i = \cos\left(i \cdot \frac{\pi}{3}\right), y_i = \sin\left(i \cdot \frac{\pi}{3}\right), z_i = \frac{\mathrm{i}}{e^i}.$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 4 знаками после запятой:

N пп	x	y	z	f(x,y,z)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x) = \frac{x^2 - 3}{x + 5}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 2

1. Описать функцию $f(x, y, a)$, при реализации включить обработку исключительных ситуаций:

$$f(x, y, a) = \frac{\sqrt{e^x - cos^4(x^2 a^5)} + arctg^4(a - x^5)}{\sqrt{|a + xy^4|}};$$

а) посчитать значения функции:

$x = 2\pi,\ y = 4.01,\ a = -1.6;$

$x = 1,\ y = 1,\ a = -1;$

б) вывести таблицу значений функции $f(x, y, a)$, состоящей из N=15 строк, в точках $(x_i, y_i, a_i),\ i = 1..N$, где

$$x_i = \frac{\tan\left(i \cdot \frac{\pi}{4}\right)}{i}, y_i = \ln\left(\frac{i}{i+1}\right), a_i = \frac{\mathrm{i}^2}{e^{i-1}}.$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 3 знаками после запятой:

N пп	x	y	a	f(x,y,a)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x) = \left(\frac{x-1}{x}\right)^2.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 3

1. Описать функцию $f(x, y, c)$ при реализации включить обработку исключительных ситуаций:

$$f(x, y, c) = \frac{tg(x^4 - 6) - cos^3\sqrt{c + xy}}{\cos^4|x^3 c|};$$

а) посчитать значения функции:

$x = -1.2,\ y = -3.41, c = 1.6;$

$x = 1,\ y = 1,\ c = \frac{\pi}{2};$

б) вывести таблицу значений функции $f(x, y, c)$, состоящей из N=25 строк, в точках $(x_i, y_i, c_i),\ i = 1..N$, где

$$x_i = \frac{i^3}{\sqrt{i}}, y_i = \tan\left(\frac{i}{i+1} \cdot \frac{\pi}{7}\right), c_i = \frac{\log_5(\mathrm{i}^2)}{i}.$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 5 знаками после запятой:

N пп	x	y	c	f(x,y,c)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x) = \frac{4 \cdot x}{x^2 - 4}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 4

1. Описать функцию $f(x, y, c)$, при реализации включить обработку исключительных ситуаций:

$$f(x, y, c) = \ln|\cos(x^2 + 2)| + \frac{3.5\left(\frac{x}{y}\right)^2 + e^c}{\sin^2(y^2 x^3)};$$

а) посчитать значения функции:

$x = \frac{\pi}{\sqrt{2}},\ y = -3.41,\ c = 1.6;$

$x = 1,\ y = 0,\ c = -10;$

б) вывести таблицу значений функции $f(x, y, c)$, состоящей из N=18 строк, в точках $(x_i, y_i, c_i),\ i = 1..N$, где

$$x_i = \sqrt[3]{\left|\sin\left(i \cdot \frac{\pi}{4}\right)\right|}, y_i = \sin\left(\sqrt{i \cdot \frac{\pi}{6}}\right), c_i = \frac{\log_2(\mathrm{i})}{\cos(i)}.$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 4 знаками после запятой:

N пп	x	y	c	f(x,y,c)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x) = \frac{x^2 - 5}{x}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 5

1. Описать функцию $f(x, y, b)$, при реализации включить обработку исключительных ситуаций:

$$f(x, y, b) = \frac{cos^6(bx^2) - (cos(x^3 + 6) - sin^3 y)}{b^3 arcsin(\frac{x}{\sqrt{y + 17}} + b)};$$

а) посчитать значения функции:

$x = \frac{\sqrt{3}\pi}{2},\ y = -1,\ b = 0.1;$

$x = 1,\ y = -17,\ b = -10;$

б) вывести таблицу значений функции $f(x, y, b)$, состоящей из N=23 строк, в точках $(x_i, y_i, b_i),\ i = 1..N$, где

$$x_i = \frac{\sqrt{(i+1)}}{i}, y_i = \cos\left(\sqrt[3]{\frac{i}{i+1}\cdot\frac{\pi}{6}}\right), b_i = \sin\left(\frac{\pi}{6}\cdot i\right).$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 5 знаками после запятой:

N пп	x	y	c	f(x,y,b)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x) = \frac{3\cdot x^3}{x^2-5} + 20.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 6

1. Описать функцию $f(x,y,b)$, (при реализации включить обработку исключительных ситуаций):

$$f(x,y,b) = \frac{cos^6(bx^2) - (cos(x^3+6) - sin^3 y)}{b^3 arcsin(\frac{x}{\sqrt{y+17}} + b)};$$

а) посчитать значения функции:

$x = \frac{\sqrt{3}\pi}{2},\ y = -1,\ b = 0.1;$

$x = 1,\ y = -17,\ b = -10;$

б) вывести таблицу значений функции $f(x,y,b)$, состоящей из N=23 строк, в точках $(x_i, y_i, b_i),\ i = 1..N$, где

$$x_i = \frac{\sqrt{i+1}}{i}, y_i = \cos\left(\sqrt[3]{\frac{i}{i+1}\cdot\frac{\pi}{6}}\right), b_i = \sin\left(\frac{\pi}{6}\cdot i\right).$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 5 знаками после запятой:

N пп	x	y	c	f(x,y,b)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x) = \frac{x^2+1}{x^2-1}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 7

1. Описать функцию $f(x,y,b)$, при реализации включить обработку исключительных ситуаций:

$$f(x,y,b) = \frac{sin^5(bx^2) - (sin\left(x^2\cdot\frac{\pi}{4}\right) - sin^3 y)}{b^2 arccos(\frac{x}{\sqrt{y+4}})};$$

а) посчитать значения функции:

$x = \frac{\sqrt{3}\pi}{2},\ y = -1,\ b = 0.1;$

$x = 1,\ y = -4,\ b = -10;$

б) вывести таблицу значений функции $f(x, y, b)$, состоящей из N=26 строк, в точках $(x_i, y_i, b_i),\ i = 1..N$, где

$$x_i = \frac{\sqrt{i+1}}{i}, y_i = \sin\left(i \cdot \frac{\sqrt{2} \cdot \pi}{2}\right), b_i = \sin e^{\pi \cdot i}.$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 3 знаками после запятой:

N пп	x	y	c	f(x,y,b)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x) = \frac{x^3 - 8}{2x + 6}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 8

1. Описать функцию $f(x, y, b)$, при реализации включить обработку исключительных ситуаций:

$$f(x, y, b) = \frac{2.33 + ln\sqrt{1 + cos^2\left(y + \frac{\sqrt{3}}{2}\pi\right)}}{sin^2\left(x + \frac{b^3}{y}\right)};$$

а) посчитать значения функции:

$x = 2,\ y = \frac{\pi}{\sqrt{2}},\ b = -2.9;$

$x = 4,\ y = 0,\ b = -3;$

б) вывести таблицу значений функции $f(x, y, b)$, состоящей из N=28 строк, в точках $(x_i, y_i, b_i),\ i = 1..N$, где

$$x_i = \sin\left(\sqrt[3]{i \cdot \frac{\pi}{6}}\right), y_i = \frac{\sqrt{i-1}}{i^3}, b_i = \cos\left(\frac{\pi}{\sqrt{6}} \cdot i\right).$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 5 знаками после запятой:

N пп	x	y	c	f(x,y,b)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x) = \sqrt[3]{\frac{x}{(x-1)^2}}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 9

1. Описать функцию $f(x, y, a)$, при реализации включить обработку исключительных ситуаций:

$$f(x,y,a)=\frac{\ln\left|a+\left(\frac{x}{y}\right)^3\right|-\sin^4(y-a)}{\sqrt{\sin^3\left(\frac{x+y}{x-y}\right)}};$$

а) посчитать значения функции:

$x=2,\ y=\sqrt{\pi},\ a=-6.1;$

$x=4,\ y=-4,\ a=-3;$

с) вывести таблицу значений функции $f(x,y,a)$, состоящей из N=24 строк, в точках $(x_i,y_i,a_i),\ i=1..N$, где

$$x_i=\mathrm{e}^{\sqrt[3]{i}}\quad ,y_i=\log_2\left(\frac{\pi}{\sqrt{6}}\cdot i\right),\quad a_i=\frac{\sqrt{i-1}}{i^3}.$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 4 знаками после запятой:

N пп	x	y	c	f(x,y,a)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x)=\frac{x-8}{x^2-3x+2}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 10

1. Описать функцию $f(x,y,a)$, при реализации включить обработку исключительных ситуаций:

$$f(x,y,a)=\frac{\arccos(a+x^3)-\sin^4(y-a)}{\sqrt[3]{|\sin(x+y)|}};$$

а) посчитать значения функции:

$x=2,\ y=\sqrt{\pi},\ a=-6.1;$

$x=4,\ y=-4,\ a=-3;$

с) вывести таблицу значений функции $f(x,y,a)$, состоящей из N=19 строк, в точках $(x_i,y_i,a_i),\ i=1..N$, где

$$x_i=\cos\sqrt[3]{i\cdot\frac{\pi}{3}},\qquad y_i=\tan\sqrt{i\cdot\frac{\pi}{5}},\qquad a_i=\frac{\mathrm{i}}{\log_5(i+1)}.$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 4 знаками после запятой:

N пп	x	y	c	f(x,y,a)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x)=\frac{e^x}{x^2-25}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 11

1. Описать функцию $f(x,y,a)$, при реализации включить обработку исключительных ситуаций:

$$f(x,y,a)=\frac{ctg^3(x^3-a)+arctg^2(y)}{\sqrt{e^{tg(x+y-a)}}};$$

а) посчитать значения функции:

$x=7,\ y=\sqrt{\pi},\ a=-1.1;$

$x=0,\ y=-4,\ a=-3.4;$

б) вывести таблицу значений функции $f(x,y,a)$, состоящей из N=33 строк, в точках $(x_i,y_i,a_i),\ i=1..N$, где

$$x_i=\frac{\sqrt{(i+1)}}{i},\ y_i=\cos\left(\sqrt[3]{\frac{i}{i+1}\cdot\frac{\pi}{6}}\right),\quad a_i=\sin\left(\frac{\pi}{6}\cdot i\right).$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 5 знаками после запятой:

N пп	x	y	c	f(x,y,a)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x)=\frac{4-x}{x^2-4}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 12

1. Описать функцию $f(x,y,a)$, при реализации включить обработку исключительных ситуаций:

$$f(x,y,a)=\frac{\sqrt{e^x-cos^2(x^2a^5)}+arctg^4(a-y^5)}{\sqrt{|a+xy^4|}};$$

а) посчитать значения функции:

$x=7,\ y=\sqrt{\pi^3},\ a=e^3$

$x=-10,\ y=4,\ a=5;$

б) вывести таблицу значений функции $f(x,y,a)$, состоящей из N=33 строк, в точках $(x_i,y_i,a_i),\ i=1..N$, где

$$x_i=\frac{\sqrt{\left|\sin(\frac{\pi}{3}\cdot i)\right|}}{i},\ y_i=\cos\left(\sqrt{\frac{log_3(i)}{i+1}}\right),\ a_i=\sin\left(\frac{\pi}{12}\cdot i\right).$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 5 знаками после запятой:

N пп	x	y	c	f(x,y,a)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x)=\frac{4\cdot x^3}{x^2-8}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 13

1. Описать функцию $f(x, y, c)$, при реализации включить обработку исключительных ситуаций:

$$f(x, y, z) = \frac{c^5 + sin^4(y - c)}{sin^3(x + y) + |x - y|};$$

а) посчитать значения функции:

$x = e^2,\ y = 5.01, c = 1.6;$

$x = 0,\ y = 0,\ c = \frac{\pi}{2};$

с) вывести таблицу значений функции $f(x, y, c)$, состоящей из N=27 строк, в точках $(x_i, y_i, c_i),\ i = 1..N$, где

$$x_i = 1 - \frac{i^2}{\sqrt{i}}, y_i = \tan\left(\frac{i - 1}{i + 1} \cdot \frac{\pi}{6}\right), c_i = \frac{\log_5(\mathrm{i}^2)}{\log_2(\mathrm{i})}.$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 5 знаками после запятой:

N пп	x	y	c	f(x,y,c)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x) = \frac{x - 3}{x^2 - 25}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 14

1. Описать функцию $f(x, y, c)$, при реализации включить обработку исключительных ситуаций:

$$f(x, y, t) = \frac{sin^3 x + \ln(2y + 3x)}{t^5 - \sqrt{x}};$$

а) посчитать значения функции:

$x = e^2,\ y = 5.01, t = 1.6;$

$x = 1024,\ y = 0,\ t = 2;$

б) вывести таблицу значений функции $f(x, y, c)$, состоящей из N=31 строк, в точках $(x_i, y_i, t_i),\ i = 1..N$, где

$$x_i = \frac{\log_3(i^2)}{\sqrt{i}}, \quad y_i = \mathrm{e}^{-\sqrt{i}}, \qquad t_i = \frac{\sin(\frac{\pi}{4}\mathrm{i}^2)}{\cos(\mathrm{i})}.$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 4 знаками после запятой:

N пп	x	y	c	f(x,y,t)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x) = \frac{\ln(x^2 + 1)}{x^2}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 15

1. Описать функцию $f(x, y, a)$ при реализации включить обработку исключительных ситуаций:

$$f(x,y,a) = \frac{\sqrt{4e^x + sin^2(x^3a^5)} + arctg^4(a - y^5)}{\sqrt{|a - xy^2|}};$$

а) посчитать значения функции:

$x = 7,\ y = \sqrt{\pi^5},\ a = e^2$

$x = 1,\ y = 1,\ a = 1;$

б) вывести таблицу значений функции $f(x, y, a)$, состоящей из N=31 строки, в точках $(x_i, y_i, a_i),\ i = 1..N$, где

$$x_i = \frac{\sqrt{\left|cos(\frac{\pi}{4} \cdot i)\right|}}{i},\ y_i = \cos\left(\sqrt{\frac{log_{10}(i)}{i^2 + 1}}\right),\ a_i = \cos\left(\frac{\pi}{3} \cdot i\right).$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 5 знаками после запятой:

N пп	x	y	c	f(x,y,a)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x) = \frac{e^{0.01 \cdot x}}{x^2 - 8}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 16

1. Описать функцию $f(x, y, c)$, при реализации включить обработку исключительных ситуаций:

$$f(x,y,z) = \frac{y \cdot c^3 + sin^2(y - c)}{cos^3(x + y^2) + |x^2 - y| - 1};$$

а) посчитать значения функции:

$x = e^2,\ y = 6.01, c = 2.6;$

$x = 0,\ y = 0,\ c = \frac{\pi}{2};$

б) вывести таблицу значений функции $f(x, y, c)$, состоящей из N=25 строк, в точках $(x_i, y_i, c_i),\ i = 1..N$, где

$$x_i = 1 - \frac{\cos(i^2)}{\sqrt{i}}, y_i = \operatorname{ctan}\left(\frac{i}{i+1} \cdot \frac{\pi}{3}\right), c_i = \frac{\log_5(\mathrm{i}^2)}{e^i}.$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 5 знаками после запятой:

N пп	x	y	c	f(x,y,c)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x) = \frac{x^3 - 1}{8x - 1}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 17

1. Описать функцию $f(x,y,c)$, при реализации включить обработку исключительных ситуаций:

$$f(x,y,c)=\frac{sin(x^4-6)-\tan(\sqrt[3]{c+xy})}{\cos^3|x^3c|}$$

а) посчитать значения функции:

$x=-0.2,\ y=-4.41, c=\ 1.2;$

$x=1,\ y=1,\ c=\frac{\pi}{2};$

с) вывести таблицу значений функции $f(x,y,c)$, состоящей из N=25 строк, в точках $(x_i,y_i,c_i),\ i=1..N$, где

$$x_i=\frac{(i-0.5)^2}{\sqrt{i+1}}, y_i=\mathrm{tg}\left(\frac{i}{i+1}\cdot\frac{\pi}{7}\right), c_i=\frac{\mathrm{i}}{e^{\sqrt{i}}}.$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 5 знаками после запятой:

N пп	x	y	c	f(x,y,c)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x)=\frac{5\cdot x}{x^3-8}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

Вариант 18

1. Описать функцию $f(x,y,a)$, при реализации включить обработку исключительных ситуаций:

$$f(x,y,a)=\frac{tg^3(xy^2-a)+arctg^2(y-x)}{\sqrt{e^{\sin(x+y-a)}}};$$

а) посчитать значения функции:

$x=7,\ y=\sqrt{\pi},\ a=\ -1.1;$

$x=0,\ y=-4,\ a=\frac{\pi}{2};$

с) вывести таблицу значений функции $f(x,y,a)$, состоящей из N=33 строк, в точках $(x_i,y_i,a_i),\ i=1..N$, где

$$x_i=\frac{\sqrt{i^2+1}}{i^2},\ y_i=\sin\left(\sqrt{\frac{i}{i+1}\cdot\frac{\pi}{6}}\right),\quad a_i=\mathrm{tg}\left(\frac{\pi}{6}\cdot i\right).$$

Таблицу вывести в следующем виде (использовать форматный вывод), все числа в таблице (кроме номера) вывести с 5 знаками после запятой:

N пп	x	y	c	f(x,y,a)
1	…	…	…	…

2. Построить график функции на интервале `[a, b]`:

$$f(x)=\frac{x+5}{x^2-9}.$$

3. Выбрать произвольный рисунок и нарисовать его средствами модуля `matplotlib.patches`.

3. МАТРИЦЫ И ВЕКТОРЫ

3.1. Массивы

NumPy — это библиотека языка Python, предназначенная для создания и обработки многомерных массивов числовых данных (векторов и матриц). Также NumPy включает большую библиотеку высокоуровневых (и очень быстрых) математических функций для операций с этими массивами. Подключение библиотеки:

```
import numpy as np
```

Для упрощения обращения к методам библиотеки модуль в программе будет именоваться **np**.

3.1.1. Понятие массива

Массив – это пронумерованная последовательность величин одинакового типа, обозначенная одним именем. Каждое из значений, составляющих массив, называется *компонентой* (или *элементом* массива). Для обращения к элементу массива используется его индекс (рисунок 24).

Рисунок 24. Структура массива

Для обращения к элементу по индексу 5 используется запись **a[5]**, изменить значение, хранящееся по этому индексу, можно с помощью оператора присваивания:

```
a[5] = 47
```

Создание массива

В NumPy существует много способов создать массив. Проще всего создать массив из списка. Для этого используется метод **`array()`**:

```
import numpy as np
a = np.array([-4, 5, 2])
```

При создании массива из списка можно переопределить тип его элементов, например, из списка, состоящего из строк, представляющих собой числа, можно создать массив из целых чисел.

```
a = np.array(['-4', '5', '2'], dtype=int)
```

Например, введем с клавиатуры строку, которая состоит из чисел, разделенных пробелом. А затем создадим массив из них:

```
line = input("Введите числа через пробел : ")
list_a = line.split()
a = np.array(list_a, dtype=int)
```

Вывод массива

Вывод массива осуществляется с помощью оператора **`print()`**:

```
a = np.array(['-4', '5', '2'], dtype=int)
print("a = ", a)
```

Результат:

```
a =  [-4  5  2]
```

Задание для самопроверки

Отметьте **неверные** операторы и объясните, в чем ошибка. В начале программы модуль numpy импортирован следующим образом:

```
import numpy as np
```

При анализе операторов вывода считайте, что в переменной **a** хранится массив чисел.

- `print a`
- `a = array([1, 2, 3], dtype=int)`
- `a = input(np.array())`
- `a = np.array([1, 2, 3], dType=int)`
- `a = np.array(1, 2, 3)`
- `print("a = %3d " % a)`

3.1.2. Методы массивов

Модуль **numpy** предоставляет множество методов работы с массивами. Для обращения к ним после имени массива ставится точка, затем записывается метод и круглые скобки. В таблице 17 приведены некоторые методы, примененные к массиву с именем **a**.

Таблица 17 – Методы массивов

Метод	Описание
`a.sum()`	Сумма элементов массива a
`a.prod()`	Произведение элементов массива a
`a.mean()`	Среднее значение элементов массива a
`a.max()`	Максимальное значение из элементов массива a
`a.min()`	Минимальное значение из элементов массива a

Например, в следующей программе методы массивов применяются для вычисления суммы, произведения элементов массива, поиска среднего, максимального и минимального элемента.

```
import numpy as np

a = np.array([-4, 5, 2])

print("a = ", a)
print("Сумма = ", a.sum())
print("Произведение = ", a.prod())
print("Среднее = ", a.mean())
print("Максимум = ", a.max())
print("Минимум = ", a.min())
```

Результат:

```
a =  [-4  5  2]
Сумма =  3
Произведение =  -40
Среднее =  1.0
Максимум =  5
Минимум =  -4
```

Задание для самопроверки

Вставьте в программу пропущенные фрагменты так, чтобы в результате выполнения программы получился следующий результат:

```
[200.    6.   -5.    1.]
```

Код программы:

```
import numpy as np

a = np.array([-5, 5, -1, 1, 2, 4])
```

```
b_0 = a.___
b_1 = a.___
b_2 = a.___
b_3 = a.___

b = np.array([b_0, b_1, b_2, b_3])
print(b)
```

3.1.3. Операции с массивами

Математические операции: сложение (+), умножение (*), вычитание (-) , деление(/), остаток от деления (%), целочисленное деление (//) – применяются непосредственно к массивам. При их использовании создается новый массив, который заполняется результатами действия математической операции с соответствующими элементами массива. Массивы при этом должны быть одинакового размера.

В следующей программе математические операции применяются к массивам **a** и **b**.

```
import numpy as np

a = np.array([-4, 5, 2])
b = np.array([3, 2, 1])

c = a + b
print("a + b =", c)

c = a - b
print("a - b =", c)

c = a * b
print("a * b =", c)

c = a / b
print("a / b =", c)

c = a % b
print("a % b =", c)

c = a // b
print("a // b =", c)
```

Результат:

```
a + b = [-1  7  3]
a - b = [-7  3  1]
a * b = [-12  10   2]
a / b = [-1.33333333  2.5         2.        ]
a % b = [2 1 0]
a // b = [-2  2  2]
```

Задание для самопроверки

Какую операцию нужно вставить в выражение для формирования массива **c**, чтобы получился следующий результат:

```
[3 7 2]
```

Код программы:

```
import numpy as np

a = np.array([5, 5, 2])
b = np.array([2, 7, 1])

c = a.min()+ a * (b + b.min()) __ b
print(c)
```

3.1.4. Функции массивов

В модуле **numpy** реализованы математические функции, выполняемые над массивами, в таблице приведены некоторые из них для массива с именем **a**. Перед вызовом функций в программе необходимо указать префикс «**np**» (имя модуля **numpy** в текущей программе) и точку.

Таблица 18 – Функции массивов

Функция	Описание
`round(a,k)`	Округляет значения элементов массива **a** до **k** знаков после запятой, результат – массив
`sin(a)`	Вычисляет синусы от каждого элемента массива **a**, результат – массив
`cos(a)`	Вычисляет косинусы от каждого элемента массива **a**, результат – массив
`tan(a)`	Вычисляет тангенсы от каждого элемента массива **a**, результат – массив
`arcos(a)`	Вычисляет арккосинусы от каждого элемента массива **a**, результат – массив
`arcsin(a)`	Вычисляет арксинусы от каждого элемента массива **a**, результат – массив
`arctan(a)`	Вычисляет арктангенсы от каждого элемента массива **a**, результат – массив
`degrees(a)`	Переводит каждый элемент массива **a** в градусы, результат – массив
`radians(a)`	Переводит каждый элемент массива **a** в радианы, результат – массив
`log(a)`	Вычисляет натуральный логарифм от каждого элемента массива **a**, результат – массив
`log10(a)`	Вычисляет десятичный логарифм от каждого элемента массива **a**, результат – массив
`exp(a)`	Вычисляет экспоненту от каждого элемента массива **a**, результат – массив
`sum(a)`	Вычисляет сумму элементов массива **a**, результат – число
`prod(a)`	Вычисляет произведение элементов массива **a**, результат – число
`mean(a)`	Вычисляет среднее значение элементов массива **a**, результат – число
`max(a)`	Вычисляет максимальное значение из элементов массива **a**, результат – число
`min(a)`	Вычисляет минимальное значение из элементов массива **a**, результат – число
`abs(a)`	Применяет функцию модуль для каждого элемента массива **a**, результат – массив
`pi`	Число **пи**, размноженное на все элементы массива.
`sqrt(a)`	Вычисляет корень от каждого элемента массива **a**, результат – массив

Пример. Решим задачу с вычислением численности населения по формуле Капицы, используя массивы из модуля **numpy**. Создадим массив с различными годами, а затем посчитаем численность населения Земли в эти года по формуле:

$$N(t) = \frac{C}{\tau} \cdot arcctg\left(\frac{T_1 - t}{\tau}\right),$$

где t – год, для которого вычисляется численность населения;
C – 172 миллиарда человек/лет;
T_1 – 2000 год;
τ – 45 лет.

Код программы:

```
import numpy as np

def compute_population(t):
    c = 172
    t_1 = 2000
    tau = 45
    y = c / tau * (np.pi / 2 - np.arctan((t_1-t)/tau))
    return y

t = np.array([1000, 1500, 1800, 1850, 1900, 1950, 1990, 2019, 2030])
```

```
print("Годы:", t)
print("Численность:", np.round(compute_population(t), 3))
```

Результат:

```
Годы: [1000 1500 1800 1850 1900 1950 1990 2019 2030]
Численность: [0.172 0.343 0.846 1.114 1.616 2.801 5.168 7.531 8.251]
```

Обратите внимание, что в программе не используются циклы, которые осуществляют поэлементный перебор, не используются конструкторы списков. Все операции выполняются **целиком** над массивами. Поэтому перед всеми функциями (**pi, arctan, round**) стоит префикс **np**.

Задание для самопроверки

Дан фрагмент программы:

```
import numpy as np
a = np.array([2, 5, -2, 0, 8])
b = np.array([5, -1, -3, 6, 2])
```

Выберите те операторы, в результате которых получается массив.

- `np.sum(a + b)`
- `np.sum(a) + np.mean(a)`
- `a + np.sum(b)`
- `np.pi - a`
- `np.round(a, 2)`
- `a + b`

3.1.5. Функция where и логические функции

Для того чтобы найти индексы элементов в массиве, отвечающие некоторому условию, используется метод **where**(), в качестве аргумента которого задается логическое выражение.

Это выражение может включать:

- знаки отношения – равно(==), не равно(!=), больше(>), больше или равно(>=), меньше(<), меньше или равно(<=);
- имена массивов;
- константы;
- выражения.

Результат выполнения функции – массив модуля **numpy**, чтобы получить список индексов в привычном виде, необходимо обратиться к результату по индексу 0 ([0]).

Рассмотрим, как работает функция **where()**, в качестве исходных данных будем использовать массивы:

```
import numpy as np
a = np.array([5, 5, 2, 6, -1])
b = np.array([2, 7, 2, 3, -1])
```

- если в логическом выражении используются массивы, то они должны быть одинаковой длины, при этом каждый элемент одного массива сравнивается с элементом с тем же индексом второго массива, если логическое выражение выполняется, то индекс этого элемента включается в итоговый список:

```
print(np.where(a == b)[0]) → [2 4]
print(np.where(a > b)[0]) → [0 3]
```

- если в логическом выражении используются константа, то каждый элемент массива сравнивается с этой константой, если логическое выражение выполняется, то индекс этого элемента включается в итоговый список:

```
print(np.where(a < 3)[0]) → [2 4]
print(np.where(b > 10)[0]) → []
```

- если в логическом выражении используются выражения, то их результатом может быть либо число, либо массив, но тогда его длина должна соответствовать длине других массивов, используемых в выражении:

```
print(np.where(a >= b.mean())[0]) → [0 1 3]
print(np.where(a + b >= 8)[0]) → [1 3]
```

Также в логическом выражении можно использовать логические операции, которые реализуются с помощью функций:

ИЛИ – для реализации этой операции используется функция **np.logical_or()**;

И – **np.logical_and()**;

НЕ – **np.logical_not()**.

В скобках через запятую перечисляются логические выражения, которые должны быть соединены соответствующей операцией. Например, логическое выражение **a > 0 И a <= 6** в методе **where**() записывается так:

```
np.logical_and(a > 0, a <= 6) результат [0 1 2 3]
```

Допускается использование вложенных функций, описывающих логические операции. Например, логическое выражение **(a > 0 И a < 6) ИЛИ (b < 0 ИЛИ b > 7)** в методе **where**() записывается так:

```
np.logical_or(
    np.logical_and(a > 0, a < 6),
    np.logical_or(b < 0, b > 7)
) результат [0 1 2 4]
```

Задание для самопроверки

Дан фрагмент программы:

```
import numpy as np
a = np.array([2, 5, 2, 0, 0])
b = np.array([5, -1, -1, 6, 2])
c = # выражение
print(c)
```

Что будет напечатано в консоли, если массив **c** будет сформирован с помощью следующих выражений:

- `np.where(a == b)[0]`
- `np.where(np.logical_not(b <= 0))[0]`
- `np.where(np.logical_or(a >= 0, b == 6))[0]`
- `np.where(a == 0)[0]`
- `np.where(np.logical_and(a > 1, b < 0))[0]`

3.1.6. Срезы и индексы

Для обращения к элементам или группе элементов массива используются индексы и срезы.

Индекс – это номер элемента в массиве, нумерация начинается с 0. Для обращения к элементу с заданным индексом используются квадратные скобки: **a[3]** – элемент массива **a** с индексом 3.

Срез – это часть массива, которую можно выделить по индексам:

```
массив[нижняя_граница : верхняя_граница : шаг]
```

Для выделения среза в квадратных скобках указывается нижняя граница среза, верхняя граница, которая в срез не включается, и шаг, с которым выделяются индексы. Части описания среза могут быть опущены.

Например, рассмотрим массив объявленный следующим образом:

```
import numpy as np
a = np.array([5, 5, 1, 6, 15, 3, 8, -1])
```

тогда:

`a[2:6]` – в срез включаются элементы с индексами 2, 3, 4, 5 – `[1 6 15 3]`

`a[2:]` – элементы с индексами, начиная со 2 до последнего – `[1 6 15 3 8 -1]`

`a[:6]` – элементы с индексами, начиная со 0 до 5 включительно – `[5 5 1 6 15 3]`
`a[1:6:2]` – элементы с индексами 1, 3, 5 – `[5 6 3]`
`a[1::2]` – элементы с индексами 1, 3, 5, 7 – `[5 6 3 -1]`
`a[:6:2]` – элементы с индексами 0, 2, 4 – `[5 1 15]`
`a[::2]` – элементы с индексами 0, 2, 4, 6 – `[5, 1, 15, 8]`

Срез можно рассматривать как новый массив. Оператор:

```
b = a[1:3]
```

выделяет элементы из массива **a**, начиная с элемента с индексом 1 и заканчивая элементом с индексом 2 (получается результат `[5 2]`), полученный массив записывается в переменную **b**.

Срезы позволяют с помощью оператора присваивания изменять значения нескольких элементов массива одновременно. Оператор:

```
a[1:4] = 0
```

присваивает значение 0 элементам массива **a**, начиная с индекса 1 по индекс 3, результат – `[5 0 0 0 -1]`.

Задание для самопроверки

Напишите программу для решения задачи:

Дан массив, в который занесены расходы человека на проезд по каждому месяцу в течение года. Сравнить, в какой период – зимний или летний – тратится больше денег на проезд. Вывести номера месяцев (начиная с 1), в которые расходы были наибольшими.

Тест 1

```
Расходы: 1200 1300 900 1450 1300 1000 900 1000 1450 1450 1300 1400
Зимой на проезд потрачено больше денег, сумма: 3900 руб.
Самая большая сумма:1450 руб., потрачена в следующих месяцах: [ 4  9 10]
```

Тест 2

```
Расходы: 1000 900 750 1050 1000 900 1200 1300 1050 1050 1300 1200
Зимой на проезд потрачено больше денег, сумма: 3400 руб.
Самая большая сумма:1300 руб., потрачена в следующих месяцах: [8 11]
```

Тест 3

```
Расходы: 1200 1300 900 1450 1300 1000 1500 1000 1450 1450 1300 1000
Зимой и летом на проезд потрачены одинаковые деньги, сумма: 3500 руб.
Самая большая сумма:1450 руб., потрачена в следующих месяцах: [ 7]
```

3.2. Задача о движении по различным участкам дороги

Задача. Дорога из пункта **A** в пункт **B** состоит из нескольких участков, известны их длины. Для автомобиля известна средняя скорость его движения на каждом участке. Результаты приведены в таблице 19.

Таблица 19 – Информация о движении автомобиля

Длина участка, км	15	5	12	2	21	17	21	3	10	5
Скорость, км/ч	60	30	60	45	50	60	50	40	60	40

Определить:

1) расстояние между пунктами **A** и **B**;
2) время прохождения каждого участка и общее время в пути;
3) среднюю скорость движения;
4) номера тех участков, при движении по которым затрачено максимальное время;
5) длину и время проезда первых четырех участков;
6) среднюю скорость движения по первым четырем участкам.

Код программы, пояснения приведены в комментариях:

```
# Импортируем модуль numpy, дадим ему имя np
import numpy as np
```

```
#Сформируем массивы длин участков и скорости автомобиля на них:
path = np.array([15, 5, 12, 2, 21, 17, 21, 3, 10, 5])
speed = np.array([60, 30, 60,45, 50, 60, 50, 40, 60, 40])

# Вычислим длину пути от A до B и выведем результат:
len_path = path.sum()

# Вычислим время прохождения автомобилем каждого участка,
# результат получится в виде массива, каждый элемент
# массива при выводе округлим до 2 знаков после запятой:
time = path / speed
print("Время на каждом участке:", np.round(time, 2))

# Вычислим общее время в пути,  при выводе округлим значения:
sum_time = time.sum()
print("Общее время в пути: ", round(sum_time, 2))

# Посчитаем среднюю скорость автомобиля (среднюю путевую скорость):
# Среднюю скорость нельзя вычислять как среднее значение скорости на всех
участках!
avg_speed = len_path / sum_time
print("Средняя скорость: ", round(avg_speed, 2))

# Вычислим максимальное время и выведем номера участков дороги,
# на проезд по которым потрачено больше всего времени:
max_time = time.max()
max_path = np.where(time == max_time)[0]
print("Участки, на проезд по которым потрачено больше всего времени:",
max_path)

# Посчитаем длину и время проезда по первым 4 участкам:
len_path_four = path[:4].sum()
print("Длина первых четырех участков:", len_path_four)

time_four = path[:4] / speed[:4]
sum_time_four = time_four.sum()
print("Время проезда:", round(sum_time_four, 2))

# Вычислим среднюю скорость движения по первым 4 участкам:
avg_speed_four = len_path_four / sum_time_four
print("Средняя скорость движения:", round(avg_speed_four, 2))
```

Результат:

```
Время на каждом участке: [0.25 0.17 0.2  0.04 0.42 0.28 0.42 0.08 0.17 0.12]
Общее время в пути:  2.15
Средняя скорость:  51.6
Участки, на проезд по которым потрачено больше всего времени: [4 6]
Длина первых четырех участков: 34
Время проезда: 0.66
Средняя скорость движения: 51.43
```

Задание для самопроверки

Напишите программу для решения задачи:

Дорога из пункта A в пункт B состоит из нескольких участков, известны их длины Для автомобиля известна средняя скорость его движения на каждом участке. Автомобиль въехал на дорогу в начале участка ***k*** *и выехал с нее после проезда по участку* ***p****. Найти сколько километров проехал автомобиль по дороге, время движения и среднюю скорость.*

Входные данные:

- строка, в которой через пробел перечислены длины всех участков дороги;
- строка, в которой через пробел перечислены средние скорости на участках;

- номер участка, на котором автомобиль въехал на дорогу;
- номер участка, после которого выехал.

Выходные данные:

- длина пути с **k** по **p** участок;
- время в пути;
- средняя скорость движения автомобиля по этим участкам.

Тест 1

```
Длины: 20 8 9 18 5 12 16 16 6 7
Скорости: 44 70 44 66 46 38 38 37 66 67
k = 4
p = 7
S =  49 км, T =  1.28 час, V = 38.34 км/ч
```

Тест 2

```
Длины: 7 13 20 9 8 17 13 6 7
45 63 70 59 38 51 45 60 41
k = 3
p = 7
S =  53 км, T =  1.09 час, V = 48.83 км/ч
```

Тест 3

```
Длины: 15 18 5 9 5 18 13 14 20 13 6 16 9 15
Скорости: 70 49 39 40 66 60 62 63 59 37 66 34 65 70
k = 3
p = 9
S =  92 км, T =  1.72 час, V = 53.40 км/ч
```

3.3. Многомерные массивы

Библиотека **NumPy** позволяет создавать и обрабатывать двухмерные массивы.

Двумерный массив – это упорядоченная совокупность однотипных данных, имеющих общее имя. Обращение к элементам двумерного массива осуществляется по двум индексам. На рисунке 25 массив с именем **a** состоит из трех строк и 4 столбцов, нумерация индексов строк и столбцов начинается с 0. Для обращения к элементу двумерного массива после имени массива указываются квадратные скобки, в которых через запятую сначала указывается индекс строки, а затем индекс столбца.

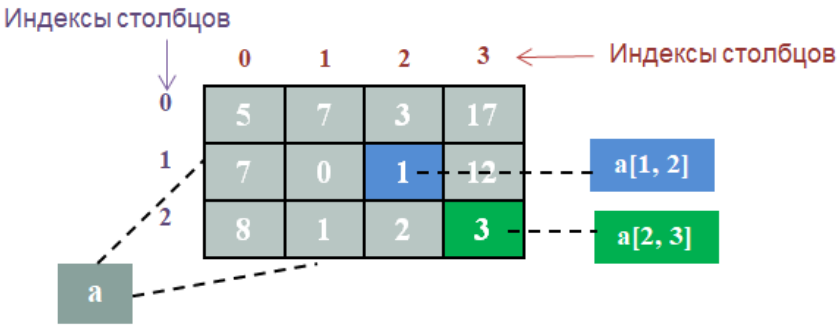


Рисунок 25. Структура двумерного массива

Двумерный массив можно использовать для описания числовых матриц.

3.3.1. Создание двумерного массива

В NumPy существует много способов создать массив. Проще всего создать массив из списка, каждый элемент которого представляет собой новый список. Для этого используется функция **`array().`** На основе списка будет создан массив, количество строк которого будет равно количеству вложенных списков, количество столбцов – количеству элементов в каждом.

```
import numpy as np
```

```
a = np.array([[3, 4, -2], [-2, -1, 4], [1, 2, 1]])
```

В нашем случае **a** будет представлять собой массив, состоящий из 3 строк и 3 столбцов.

При создании двумерного массива можно переопределить тип его элементов.

```
a = np.array([['3','4','-2'], ['-2','-1','4'], ['1','2','1']],
             dtype=int)
```

Другой способ – создать массив на основе данных, введенных пользователем с клавиатуры, по следующему алгоритму:

- сначала определим пустой список, в котором будут накапливаться списки для каждой строки массива, дадим ему имя **array_list**;
- затем в цикле для каждой строки:
  - введем строку, в которой числа разделяются пробелами;
  - элементы каждой строки занесем в список с помощью функции **split();**
  - созданный список добавляется к списку **array_list;**
- после ввода всех строк, полученный списо**к** преобразуется в двумерный массив **numpy**.

В примере вводится двумерный массив, состоящий из трех строк и трех столбцов.

```
import numpy as np
array_list = [ ]

for i in range(3):
    line = input("Введите строку чисел через пробел: ")
    array_str = line.split()
    array_list.append(array_str)

a = np.array(array_list, dtype=int)
```

3.3.2. Вывод массива

Вывод массива осуществляется с помощью оператора **print()**:

```
import numpy as np
a = np.array([[3, 4, -2], [-2, -1, 4], [1, 2, 1]])
print(a)
```

Результат:

```
[[ 3  4 -2]
 [-2 -1  4]
 [ 1  2  1]]
```

3.3.3. Операции с матрицами

Операции с матрицами реализованы в модулях numpy и numpy.linalg**.** Последний необходимо импортировать в начале программы и для простоты обращения к его функциям дать ему имя **alg**.

```
import numpy.linalg as alg
import numpy as np
```

Сложение/вычитание матриц

Данные операции выполняются для матриц одинакового размера **A(n, m)** и **B(n, m)**. Получается матрица размера **n** на **m**, каждый элемент которой является суммой/разностью соответствующих элементов матриц **A** и **B**.

Пример:

```
a = np.array([[3, 4, -1], [-2, 1, 3]])
b = np.array([[1, 0, -5], [2, 2, -1]])

c = a + b

print("a:\n", a)
```

```
print("b:\n", b)
print("c:\n", c)
```

Пояснение. В операторе **print()** в строковых константах можно использовать специальные символы, которые используются для оформления вывода. Так `"\n"` означает, что все последующие элементы вывода будут располагаться на новой строке.

Результат:

```
a:
 [[ 3  4 -1]
 [-2  1  3]]
b:
 [[ 1  0 -5]
 [ 2  2 -1]]
c:
 [[ 4  4 -6]
 [ 0  3  2]]
```

Умножение матриц

Операция умножения матрицы **A(n, m)** на матрицу **B(m, k)** выполняется с помощью функции модуля **`numpy dot()`**. В результате получается матрица размером **n** на **k.** Элемент с индексами **i** и **j** матрицы является скалярным произведением **i**-й строки матрицы **A** и **j**- го столбца матрицы **B** (рисунок 24).

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 7 & 8 \\ 9 & 1 \\ 2 & 3 \end{bmatrix},$$

$$C = A \cdot B,$$

$$C = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 7 & 8 \\ 9 & 1 \\ 2 & 3 \end{bmatrix} =$$

$$= \begin{bmatrix} 1 \cdot 7 + 2 \cdot 9 + 3 \cdot 2 & 1 \cdot 8 + 2 \cdot 1 + 3 \cdot 3 \\ 4 \cdot 7 + 5 \cdot 9 + 6 \cdot 2 & 4 \cdot 8 + 5 \cdot 1 + 6 \cdot 3 \end{bmatrix} = \begin{bmatrix} 31 & 19 \\ 85 & 55 \end{bmatrix}$$

Рисунок 26. Умножение двух матриц

Пример:

```
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([[7, 8], [9, 1], [2, 3]])

c = np.dot(a, b)

print("a:\n", a)
print("b:\n", b)
print("c:\n", c)
```

Результат:

```
a:
 [[1 2 3]
 [4 5 6]]
b:
 [[7 8]
 [9 1]
 [2 3]]
c:
 [[31 19]
 [85 55]]
```

Операции с матрицей и числом

Умножение и другие операции с числом осуществляются поэлементно, то есть операция выполняется для каждого элемента матрицы и числа. Например, при умножении матрицы на число, каждый элемент матрицы умножается на это число.

Пример:

```
a = np.array([[1, 2, 3], [4, 5, 6]])

c = 5 * a

print("a:\n", a)
print("c:\n", c)
```

Результат:

```
a:
 [[1 2 3]
 [4 5 6]]
c:
 [[ 5 10 15]
 [20 25 30]]
```

Вычисление определителя

Для квадратной матрицы, у которой одинаковое число строк и столбцов, можно вычислить определитель с помощью функции модуля **numpy.linalg det().**

Пример:

```
a = np.array([[1, 2, 3], [0, 1, -3], [2, 1, 4]])

det_a = alg.det(a)

print("a:\n", a)
print("определитель = ", det_a)
```

Результат:

```
a:
 [[ 1  2  3]
 [ 0  1 -3]
 [ 2  1  4]]
определитель =  -11.0
```

Обратная матрица

Для квадратной матрицы A можно вычислить обратную матрицу A^{-1} с помощью функции модуля **numpy.linalg inv().**

Пример:

```
a = np.array([[1, 2, 3], [0, 1, -3], [2, 1, 4]])

inv_a = alg.inv(a)

print("a:\n", a)
print("inv_a:\n", inv_a)
```

Результат:

```
a:
 [[ 1  2  3]
 [ 0  1 -3]
 [ 2  1  4]]
inv_a:
 [[-0.63636364  0.45454545  0.81818182]
 [ 0.54545455  0.18181818 -0.27272727]
 [ 0.18181818 -0.27272727 -0.09090909]]
```

Задание для самопроверки

Дан фрагмент программы:

```
import numpy as np
import numpy.linalg as alg

a = np.array([[3, 4, -2], [-2, -1, 4]])
b = np.array([[1, -3, -2, 1], [2, 4, -2, -1], [5, -2, 0, -2]])
c = np.array([[-1, 0, 2], [2, -2, 3], [0, 5, 1]])
```

Какие из следующих операторов записаны неверно? Объясните, в чем ошибки.

- `d = np.dot(a, b)`
- `d = np.dot(c, c)`
- `det_array = alg.det(b)`
- `d = 12 - b`
- `d =a + b`
- `d = np.dot(c, b)`

3.3.4. Срезы

Для обращения к частям матрицы используются индексы и срезы. С их помощью можно выделить:

- элемент;
- строку;
- столбец;
- прямоугольный блок элементов.

Рассмотрим матрицу:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

В таблице 20 приведены примеры выделения различных частей матрицы **a**.

Таблица 20 – Описание срезов и индексов

Срез/индекс	Результат	Описание
`a[1,2]`	`6`	Элемент массива **a,** расположенный на пересечении строки с индексом 1 и столбца с индексом 2 (нумерация индексов начинается с 0), результат – число
`a[2]`	`[7 8 9]`	Строка с номером 2 массива **a**, результат – одномерный массив
`a[:,1]`	`[2 5 8]`	Столбец с номером 1 массива **a**, результат – одномерный массив
`a[:2]`	`[[1, 2, 3],` `[4, 5, 6]]`	Первые две строки матрицы **a**, результат – двумерный массив
`a[1:3]`	`[[4, 5, 6],` `[7, 8, 9]]`	Строки матрицы **a** с номерами 1 и 2, результат – двумерный массив

`a[:,:2]`	`[[1, 2],`
	`[4, 5],`
	`[7,8]]`
		Первые два столбца матрицы **a**, результат – двумерный массив

`a[:,1:3]`	`[[2, 3],`
	`[5, 6],`
	`[8,9]]`
		Столбцы матрицы **a** с номерами 1 и 2, результат – двумерный массив

`a[0:2,1:3]`	`[[2,3],` `[5,6]]`	Блок матрицы **a**, результат – двумерный массив

Для обработки отдельных частей двумерных массивов используются функции модуля **`numpy`**, приведенные в таблице 21.

Пример. Координаты точек на плоскости представлены в виде двухмерного массива, каждая строка которого - координаты одной точки в декартовой системе координат. Найти «среднюю точку», координаты которой вычисляются как средние значения координат **x** и **y** всех точек. Показать точки на плоскости.

Таблица 21 – Функции модуля numpy

Функция	Описание
`max(a,0)`	Находит максимальные элементы в каждом столбце матрицы **a**, результат – вектор, размерность которого равна количеству столбцов матрицы
`max(a,1)`	Находит максимальные элементы в каждой строке матрицы **a**, результат – вектор, размерность которого равна количеству строк матрицы
`min(a,0)`	Находит минимальные элементы в каждом столбце матрицы **a**, результат – вектор, размерность которого равна количеству столбцов матрицы
`min(a,1)`	Находит минимальные элементы в каждой строке матрицы **a**, результат – вектор, размерность которого равна количеству строк матрицы
`mean(a,0)`	Находит среднее значение каждого столбца матрицы **a**, результат – вектор, размерность которого равна количеству столбцов матрицы
`mean(a,1)`	Находит среднее значение каждой строки матрицы **a**, результат – вектор, размерность которого равна количеству строк матрицы
`sum(a,0)`	Находит сумму каждого столбца матрицы **a**, результат – вектор, размерность которого равна количеству столбцов матрицы
`sum(a,1)`	Находит сумму каждой строки матрицы **a**, результат – вектор, размерность которого равна количеству строк матрицы
`prod(a,0)`	Находит произведение каждого столбца матрицы **a**, результат – вектор, размерность которого равна количеству столбцов матрицы
`prod(a,1)`	Находит произведение каждой строки матрицы **a**, результат – вектор, размерность которого равна количеству строк матрицы
`diag(a)`	Находит диагональные элементы квадратной матрицы, результат – вектор, размерностью равной количеству строк (или столбцов) матрицы **a**
`triu(a,i)`	Выделяет треугольную матрицу из матрицы **a**, **i** – целое число, которое означает «сдвиг» стороны (гипотенузы) треугольника относительно главной диагонали, результат – матрица, равная по размеру матрице **a**, элементы, не относящиеся к «треугольнику», заполняются нулями

Код программы, пояснения приведены в комментариях:

```
import numpy as np
import matplotlib.pyplot as plt

#задаем координаты точек
points = np.array([[1, 3], [3, -2], [2, 3], [-3, 4], [-2, -1]])

# вычисляем координаты средней точки
mean_point = np.mean(points, 0)
print("Средняя точка: ",np.round(mean_point, 2))

# выводим точки на график
plt.plot(points[:, 0], points[:, 1], "bo")
plt.plot(mean_point[0], mean_point[1], "ro")

# устанавливаем оси
plt.gca().spines["left"].set_position("zero")
plt.gca().spines["bottom"].set_position("zero")
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)

plt.show()
```

Результат:

```
Средняя точка:  [0.2 1.4]
```

Изображение точек на плоскости, полученное в результате выполнения программы, показано на рисунке 27.

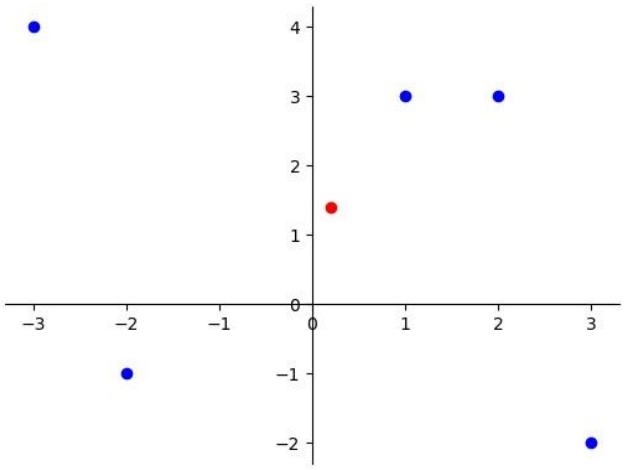


Рисунок 27. Результат выполнения программы

Задание для самопроверки

Дан фрагмент программы:

```
import numpy as np
b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

Что будет напечатано в консоли после выполнения каждого оператора:

- `print(b[: , 2])`
- `print(b[:2])`
- `print(b[2])`
- `print(b[:2, 2])`
- `print(b[:2,:2])`
- `print(np.sum(b, 1))`
- `print(np.max(b, 0))`

3.4. Задача, матричные вычисления

Задача. Даны две матрицы **A** размером 3 на 3 и **B** размером 3 на 4:

$$A = \begin{bmatrix} 3 & 4 & -2 \\ -2 & -1 & 4 \\ 1 & 2 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & -3 & -2 & 1 \\ 2 & 4 & -2 & -1 \\ 5 & -2 & 0 & -2 \end{bmatrix}.$$

Вычислить:

- определитель матрицы **A**;
- обратную матрицу $\mathbf{A}^{-1}$;
- матричное выражение:

$$\det(A) \cdot A^{-1} \cdot B - 10 \cdot A^3 \cdot B;$$

- определитель квадратного блока матрицы, состоящего из элементов на пересечении первых двух строк и последних двух столбцов матрицы **B**;
- сумму элементов главной диагонали матрицы **A**;
- максимальные элементы каждого столбца матрицы **B**;
- средние значения каждой строки матрицы **A**.

Код программы, пояснения к операторам даны в комментариях:

```
# Подключить модули  numpy и  numpy.linalg:
import numpy as np
import numpy.linalg as alg

# Создать матрицу A в виде двухмерного массива, вывести ее на экран
#(«\n» используется для перехода на следующую строку при выводе):
a = np.array([[3, 4, -2], [-2, -1, 4], [1, 2, 1]])
print("A :\n", a)
```

```
# Создать матрицу B , вывести ее на экран:
b = np.array([[1, -3, -2, 1], [2, 4, -2, -1], [5, -2, 0, -2]])
print("B :\n", b)

# Вычислить определитель матрицы A, вывести результат:
det_a = alg.det(a)
print("det(A): ", round(det_a, 1))

# Вычислить обратную матрицу A^(-1):
a_inv = alg.inv(a)
print("A^-1:\n", np.round(a_inv,1))

# Вычислить матричное выражение
# для упрощения A^3 вычислим отдельно:
a_3 = np.dot(a, np.dot(a,a))
result = det_a * np.dot(a_inv, b) - 10 * np.dot(a_3, b)
print("Матричное выражение:\n", result)

# Вычислить определитель квадратного блока  матрицы, состоящего
из элементов
# на пересечении первых двух строк  и столбцов с номером 1 и 2 матрицы B
det_bl = alg.det(b[:2,1:3])
print("det(block) = ", det_bl)

# Вычислить сумму элементов главной диагонали матрицы A, вывести
результат:
sum_diag = np.sum(np.diag(a))
print("Сумма главной диагонали A:", sum_diag)

# Найти максимальные элементы каждого столбца матрицы B:
max_col = np.max(a, 0)
print("Максимальные по столбцам, матрица B:", max_col)

# Найти средние значения каждой строки:
avg_row = np.mean(a, 1)
print("Средние по строкам, матрица A:", np.round(avg_row, 1))
```

Результат:

```
A :
 [[ 3  4 -2]
 [-2 -1  4]
 [ 1  2  1]]
B :
 [[ 1 -3 -2  1]
 [ 2  4 -2 -1]
 [ 5 -2  0 -2]]
det(A):  3.0
A^-1:
 [[-3.  -2.7  4.7]
 [ 2.   1.7 -2.7]
 [-1.  -0.7  1.7]]
Матричное выражение:
 [[-1385.    77.   134.   601.]
 [ -584.   -42.   158.   227.]
 [-1322.    21.   190.   559.]]
det(block) =  14.0
Сумма главной диагонали A: 3
Максимальные по столбцам, матрица B: [3 4 4]
Средние по строкам, матрица A: [1.7 0.3 1.3]
```

Задание для самопроверки

Решить следующую задачу:

Дан многоугольник на плоскости, заданный координатами своих вершин. Найти координаты вершин многоугольника, полученные поворотом каждой точки на заданный угол вокруг начала координат (рисунок 26).

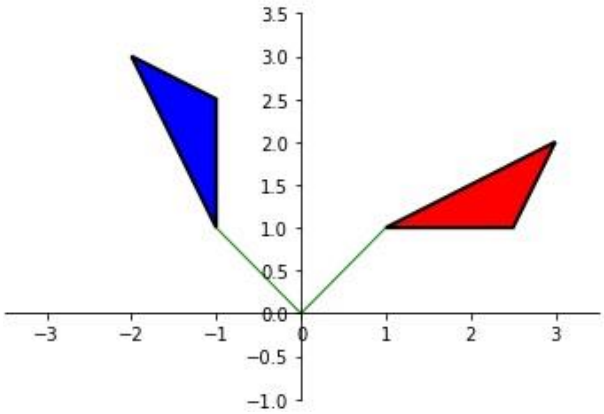


Рисунок 28. Поворот многоугольника вокруг центра координат

Входные данные:

- количество вершин многоугольника;
- список вершин многоугольника, каждая вершина на отдельной строке, строка состоит из двух чисел через пробел, определяющие координаты вершины по оси OX и OY;
- угол поворота в градусах.

Выходные данные:

- список вершин многоугольника, каждая вершина на отдельной строке.

Возможный алгоритм решения:

- Создать двухмерный массив из вершин многоугольника (количество строк – количество вершин, количество столбцов – 2, для координат **x** и **y** точки).
- Составить матрицу поворота вокруг центра координат (φ – значение угла поворота в радианах):

$$rotate = \begin{bmatrix} cos\varphi & sin\varphi \\ -sin\varphi & cos\varphi \end{bmatrix}.$$

- Умножить массив вершин на матрицу поворота. Полученная матрица – это координаты вершин многоугольника после поворота.

Результат:

Тест 1

```
n = 3
Вершины:
-7 -8
-11 4
-9 5
Угол поворота: 45
Координаты вершин повернутого многоугольника:
[[  0.707 -10.607]
 [-10.607  -4.95 ]
 [ -9.899  -2.828]]
```

Тест 2

```
n = 4
Вершины:
10 3
4 -3
-11 -3
-9 13
Угол поворота: 70
```

```
Координаты вершин повернутого многоугольника:
 [[  0.601  10.423]
 [  4.187   2.733]
 [ -0.943 -11.363]
 [-15.294  -4.011]]
```

Тест 3

```
n = 4
Вершины:
-9 5
11 9
10 -9
8 -11
Угол поворота: -20
Координаты вершин повернутого многоугольника:
 [[ -6.747   7.777]
 [ 13.415   4.695]
 [  6.319 -11.877]
 [  3.755 -13.073]]
```

3.5. Задача: решение системы линейных уравнений

Задача. Решить систему линейных уравнений матричным способом.

$$\begin{cases} -2x_1 - 8.5x_2 - 3.4x_3 + 3.5x_4 = -1.88, \\ 2.4x_2 + 8.2x_4 = -3.28, \\ 2.5x_1 + 1.6x_2 + 2.1x_3 + 3x_4 = -0.5, \\ 0.3x_1 - 0.4x_2 - 4.8x_3 + 4.6x_4 = -2.83. \end{cases}$$

Для этого:

- сформировать матрицу **A**, состоящую из коэффициентов левых частей уравнений (должна получиться матрица размером 4 на 4);
- сформировать вектор – столбец **B**, каждый элемент которого – соответствующие значения правых частей уравнений;
- вычислить вектор – решение системы уравнений по формуле:

$$X = A^{-1} \cdot B;$$

- вывести полученный вектор, округлив значения до одного знака после запятой.

Код программы, необходимые пояснения приведены в комментариях:

```
#подключаем модули numpy и numpy.linalg
import numpy as np
import numpy.linalg as alg

#формируем матрицу A, состоящую из коэффициентов левых частей уравнений
a = np.array([[-2, -8.5, -3.4, 3.5],
              [0, 2.4, 0, 8.2],
              [2.5, 1.6, 2.1, 3],
              [0.3, -0.4, -4.8, 4.6]])

#формируем матрицу B, состоящую из коэффициентов правых частей уравнений
b = np.array([-1.88, -3.28, -0.5, -2.83])

#вычисляем обратную матрицу A
a_inv = alg.inv(a)

#вычисляем произведение A и B
c = np.dot(a_inv, b)

#выводим полученный вектор, округлив значения
print(np.round(c,1))
```

Результат:

```
[ 0.1 -0.1  0.2 -0.4]
```

Задание для самопроверки

Решить следующую задачу:

Трое сотрудников получили премию в размере 2970 р., причем второй получил 1/3 того, что получил первый, и еще 180 р., а третий получил 1/3 денег второго и еще 130 р. Какую премию получил каждый?

Пояснение. Для решения задачи необходимо составить систему уравнений и решить ее матричным способом.

1. Обозначим:

x_1 – премия первого сотрудника;

x_2 – премия второго сотрудника;

x_3 – премия третьего сотрудника.

2. Составим систему трех уравнений по условию задачи:

- трое сотрудников получили премию в размере 2970 р, следовательно:

$$x_1 + x_2 + x_3 = 2970;$$

- второй получил 1/3 того, что получил первый, и еще 180:

$$\mathrm{x}_2 = \frac{1}{3} \cdot x_1 + 180;$$

- третье уравнение составьте самостоятельно.

3. Преобразуем уравнения так, чтобы слагаемые с переменными были расположены в правой части уравнения:

- первое остается без изменения:

$$x_1 + x_2 + x_3 = 2970;$$

- второе будет выглядеть так:

$$-\frac{1}{3} \cdot x_1 + \mathrm{x}_2 = 180;$$

- первые два уравнения системы имеют вид (третье добавьте самостоятельно):

$$\begin{cases} x_1 + x_2 + x_3 = 2970, \\ -\frac{1}{3} \cdot x_1 + \mathrm{x}_2 = 18, \\ \text{третье уравнение.} \end{cases}$$

4. Решим полученную систему уравнений матричным способом:

Результат:

```
Распределение премии:
первый сотрудник: 1800 р.
второй сотрудник: 780 р.,
третий сотрудник: 390 р.
```

3.6. Тренды

Тренд – это основная тенденция изменения чего-либо: например, в математике – временного ряда. Тренды могут быть описаны различными уравнениями: линейными, логарифмическими, степенными и так далее.

Рассмотрим пример. На рисунке 29 показано распределение температур по месяцам в некотором городе за 2021 год.

Для данных температур необходимо подобрать математическое описание (функцию) так, чтобы эти точки были как можно ближе к линии, построенной по выбранной функции. Эта линия называется линией **тренда**. Для наших точек линию тренда можно описать с помощью полинома второй степени:

$$f(x) = -0.78 \cdot x^2 + 9.03 \cdot x + 0.95.$$

Например, возьмем месяц май, который на графике имеет индекс 4, то есть функция тренда вычисляется для этого значения ($x = 4$):

$$f(4) = 24.5733.$$

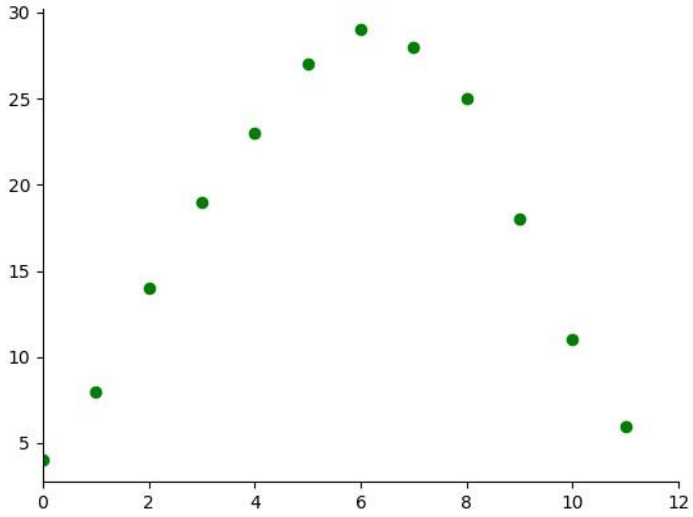


Рисунок 29. График распределения температур по месяцам

А точное значение температуры в мае – 23 градуса. Ошибка составляет 1.6 градус. Можно посчитать погрешность вычисления по формуле:

$$\delta = \left|\frac{\text{точное\_значение} - \text{приближенное}}{\text{точное\_значение}}\right| \cdot 100\%.$$

Для температуры в мае погрешность составит 6.84%. Если посчитать значения для других месяцев, погрешность вычисления будет другой.

Добавив линию тренда на график, можно проанализировать, насколько тренд хорошо отражает тенденцию изменения температур (рисунок 30):

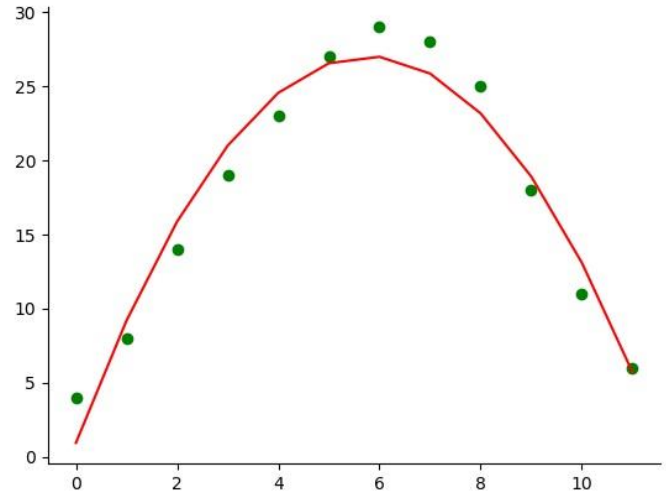


Рисунок 30. График распределения температур и тренд

В качестве функций при построении тренда можно использовать линейную функцию:

$$f(x) = a_0 \cdot x + a_1$$

и полиномиальную:

$$f(x) = a_0 \cdot x^n + a_1 \cdot x^{n-1} + \ldots + a_{n-1} \cdot x + a_n.$$

Линейную функцию можно рассматривать как частный случай полиномиальной, степень полинома $n = 1$.

В модуле **numpy** для построения линейного и полиномиального тренда используется функция:

```
polyfit(x_array, y_array, deg)
```

где `x_array, y_array` – массивы координат точек по оси OX и OУ должны иметь одинаковый размер;

deg – степень полинома (линейная функция – 1, квадратичная функция – 2 и т.д.)

Результатом функции является массив, в котором перечислены значения коэффициентов полинома $a_0, a_1, \ldots, a_n$.

Пример. Вычислить коэффициенты линии тренда для температур, вычислить погрешность в каждой точке. Погрешность округлить до 1 знака после запятой.

Код программы:

```
import numpy as np

month = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8 , 9, 10, 11])
temperature = np.array([4, 8, 14, 19, 23, 27, 29, 28, 25, 18, 11, 6])

k_poly = np.polyfit(month, temperature, 2)
print(k_poly)
```

Результат:

```
[-0.78171828  9.03246753  0.95054945]
```

Чтобы посчитать значение тренда в некоторой точке, необходимо создать функцию, параметрами которой будут искомая точка и коэффициенты полинома:

```
def get_trend(x, a):
    y = a[0] * x **2 + a[1]* x + a[2]
    return y
```

Теперь можно вычислить погрешность построения тренда для каждой точки:

```
import numpy as np

def get_trend(x, a):
    y = a[0] * x **2 + a[1]* x + a[2]
    return y

month = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8 , 9, 10, 11])
temperature = np.array([4, 8, 14, 19, 23, 27, 29, 28, 25, 18, 11, 6])

k_poly = np.polyfit(month, temperature, 2)
print(k_poly)
delta = np.abs((temperature - get_trend(month, k_poly))/temperature)*100
print("погрешность:", np.round(delta, 1))
```

Результат:

```
[-0.78171828  9.03246753  0.95054945]
погрешность: [76.2 15.  13.5 10.6  6.8  1.6  6.9  7.6  7.3  5.1 19.1  4.7]
```

Задание для самопроверки

Получить коэффициенты линий тренда 3 и 4 степени для задачи с температурами, вычислить погрешность тренда в каждой точке для полиномов 3 и 4 степени. Вывести средние значения погрешностей для полиномов 3 и четвертой степени.

Результат:

```
n = 3 [-0.03703704 -0.17060717  6.45839346  2.78388278]
средняя погрешность: 11.3
n = 4 [ 0.01343969 -0.33271011  1.86646513  1.93881928  4.30448718]
средняя погрешность: 4.33
```

3.7. Задача о выстреле из пушки

Задача. Установленная на горе пушка стреляет под углом к горизонту. С помощью интервальной съемки получили траекторию движения снаряда. В таблице 22 приведены его координаты, отснятые в некоторые моменты времени.

Таблица 22 – Положение снаряда в некоторые моменты времени

X, м	18.6	99.9	157.2	219.9	303.7	399.6	452.5	528.4	613.8	669.7	750.6	816.2	906.2
Высота, м	85.7	173.8	196.7	259.6	332.5	379.3	414.2	419.7	461.3	438.9	447.8	434.1	441.4

Определить:

1. На какой высоте установлена пушка?
2. Попадет ли снаряд в мишень, если ее центр расположен в точке **x = 1450** метров на высоте **h = 51** метр, а радиус мишени – **50 см**?

Показать траекторию движения снаряда на графике.

Ниже приведен код программы, необходимые пояснения даны в комментариях.

```
# Импортируем модуль numpy, дадим ему имя np
import numpy as np
# Импортируем  библиотеку для построения графиков:
import matplotlib.pyplot as plt

# Создадим функцию для вычисления значений
# полинома второй степени с коэффициентами a в точке x
def get_trend(x, a):
    y = a[0] * x **2 + a[1]* x + a[2]
    return y

# Сформируем массивы координат движения снаряда:
x_array = np.array([18.6, 99.9, 157.2, 219.9, 303.7, 399.6, 452.5, 528.4,
                    613.8, 669.7, 750.6, 816.2, 906.2])
h_array = np.array([85.7, 173.8, 196.7, 259.6, 332.5, 379.3, 414.2, 419.7,
                    461.3, 438.9, 447.8, 434.1, 441.4])

# Построим траекторию движения снаряда, используя в качестве линии тренда
# полином второй степени. Найдем его коэффициенты:
a = np.polyfit(x_array, h_array, 2)

# С помощью тренда вычислим высоту, на которой находилась пушка.
# Значение координаты по оси OX в этой точке равно 0.
h_zero = get_trend(0, a)
print("Высота, на которой стоит пушка: %6.2f м" % h_zero)

# Вычислим, на какой высоте будет находиться снаряд в точке по оси OX,
# где расположена мишень
x_target = 1450
h_target = get_trend(x_target, a)
print("Высота, в точке %4d м: %6.2f м" % (x_target, h_target))

# Определим, попадет ли снаряд в цель, если известно, что мишень
# расположена на высоте 51 метр, учитывая, что радиус мишени 50 см =0.5 м.
# Для этого найдем модуль разности между высотой,
# на которой расположена мишень, и положением снаряда, вычисленного
# с помощью линии тренда.
# Затем сравним полученное значение с радиусом мишени и выведем результат.

delta_h = abs(51 - h_target)
if delta_h <= 0.5:
    print("Снаряд попадет в мишень.")
else:
    print("Снаряд не попадет в мишень.")

# Построим исходные точки и линию тренда.
# Создадим списки, необходимые для построения линии тренда:

# Список со значениями координат по оси OX - целые числа от 0 до 1500
#(так как наша мишень находится в точке x = 1450), с шагом 10:
x_trend = [i for i in range(0,1500,10)]
```

```
# Список координат по оси OY, значения которого посчитаны
# с помощью функции тренда
y_trend = [get_trend(x, a) for x in x_trend]

# Сформируем линию для отображения точных координат положений снаряда:
plt.plot(x_array, h_array, 'go', label="положение снаряда")

# Сформируем график линии тренда:
plt.plot(x_trend, y_trend, 'r-', label="линия тренда")

# Отформатируем оси, выведем легенду и покажем область построения:
plt.gca().spines["left"].set_position("zero")
plt.gca().spines["bottom"].set_position("zero")

plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)

plt.legend(loc="lower center")

plt.show()
```

Результат:

```
Высота, на которой стоит пушка:  63.53 м
Высота, в точке 1450 м:  51.35 м
Снаряд попадет в мишень.
```

Также в результате выполнения программы будет построен график, приведенный на рисунке 31.

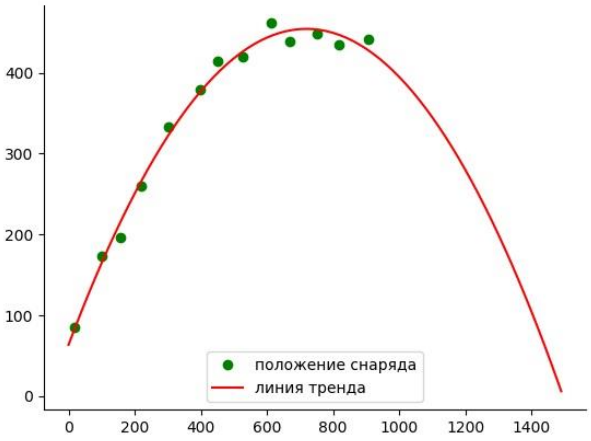


Рисунок 31. Траектория движения снаряда и линия тренд

Задание для самопроверки

Решить следующую задачу:

Даны точки на плоскости, координаты которых занесены в массивы ***x_array*** *и* ***y_array****. Постройте по этим точкам наиболее подходящий тренд: линейный (полином первой степени) или квадратичный (полином второй степени). Для этого для каждого типа тренда:*

- *найдите коэффициенты полинома;*
- *сформируйте массив значений, посчитанных по формуле тренда в точках* ***x_array****;*
- *вычислите среднюю ошибку между известными значениями* ***y_array*** *и посчитанными с помощью формулы тренда (сначала посчитать относительную погрешность между координатой точки по оси OY и значением тренда в этой точке, затем найти среднее значение погрешности).*

Далее необходимо сравнить среднюю погрешность двух трендов и вывести коэффициенты тренда с наименьшей средней ошибкой. Если ошибки одинаковы, то выводить коэффициенты полинома второй степени.

Отобразите исходные точки и линии обоих трендов на графике.

Входные данные:

- строка, в которой через пробел заданы координаты точек по оси OX;
- строка, в которой через пробел заданы координаты точек по оси OY.

Выходные данные:

- коэффициенты полинома, описывающего линию тренда;
- график с исходными функциями и линиями трендов.

Результат:

```
x_array: -0.8 0.2 0.3 0.6 0.6 0.8 1.0 1.3 1.3 1.5 2.3 2.5 2.9 2.9 3.2 4.2 4.2
y_array: 6.2 2.1 1.8 1.1 1.1 0.7 0.3 0.1 0.1 -0.1 0.2 0.5 1.3 1.3 2.1 6.2 6.2
Полином второй степени, коэффициенты: 1.012 -3.443 2.788
```

Также в результате выполнения программы будет построен график, приведенный на рисунке 32.

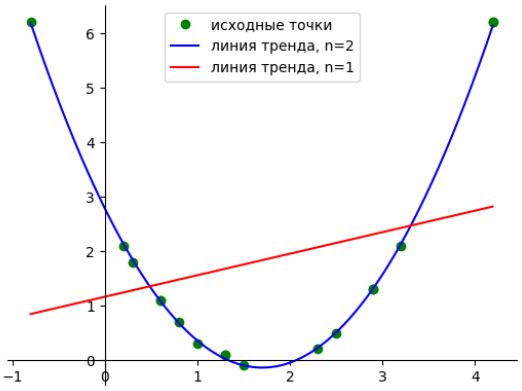


Рисунок 32. Исходные точки и линии тренда

3.8. Варианты заданий для самостоятельного решения

Вариант 1

1. Электрическая цепь состоит из **n** резисторов. В таблице приведены сопротивления на каждом резисторе **r**. Сила тока **I** в цепи, а следовательно и на каждом резисторе равна 3.3 А.

	1	2	3	4	5	6	7	8
r, Ом	0.7	0.6	1.0	0.7	1.1	1.5	1.6	1.2

Резисторы в цепи соединены последовательно.

Вычислить:

- напряжение $\mathbf{u_i}$ на каждом резисторе по формуле (сформировать вектор):
$$u_i = I \cdot r_i;$$
- общее сопротивление цепи по формуле:
$$R = r_1 + r_2 + \cdots + r_n;$$
- напряжение в цепи по формуле:
$$U = u_1 + u_2 + \cdots + u_n;$$
- найти напряжение на концах новой цепи, в которую включены резисторы с номерами с **1** по **5** (включительно), а сила тока в ней составляет **4.1 А**.
-

2. Даны квадратная матрица **A**, размером **N** на **N**, и вектор **B**, размером **N (N = 4)**, элементы которых определены следующим образом:

$$a_{ij} = sin\left(\frac{\sqrt{2} \cdot \pi \cdot j}{2} + 1\right) \cdot \ln\sqrt{i+2}\,, \qquad b_i = \sin\left(i^2 - \frac{\pi}{\sqrt{2}}\right).$$

Выполнить следующие действия:

- вычислить матричное выражение:

$$A^3 \cdot B + 5 \cdot A^{-1} \cdot B;$$

- вычислить сумму элементов матрицы **A** и сумму элементов вектора **B**;
- получить новый вектор **C**, каждый элемент которого определяется как:

$$c_i = b_i \cdot \max_{j=0..N-1} a_{ij};$$

- найти скалярное произведение векторов **B** и **C**;
- вывести индексы всех максимальных элементов матрицы **A**.

3. Баскетболист бросает мяч в корзину. С помощью интервальной съемки получили траекторию движения мяча. В список **x_list** и **h_list** занесены его координаты, отснятые в некоторые моменты времени:

```
x_list = np.array((1.3, 1.9, 2.7, 3.2, 4.0, 4.6, 5.3, 5.9, 6.6, 7.5))
h_list = np.array((3.2, 4.0, 4.5, 4.8, 4.8, 5.0, 4.9, 4.7, 4.5, 4.2))
```

Реализуйте программу, в которой:

- найдите траекторию движения мяча, используя в качестве линии тренда полином второй степени;
- ответьте на вопросы:

- с какой высоты баскетболист бросал мяч?
- попадет ли мяч в корзину, если ее центр расположен в точке **x = 8.0** на высоте **h = 3.0**, а радиус равен **0.27**?

- постройте график, на котором выведите исходные точки и линию тренда.

Вариант 2

1. Известны координаты **n** точек на плоскости:

	1	2	3	4	5	6	7	8
x	-7.3	2.7	-2.4	-6.0	-9.4	8.9	-9.2	3.4
y	-8.9	-5.3	-4.8	-3.4	-6.4	0.1	5.3	6.0

Вычислить:

- длины отрезков, соединяющих заданные точки с началом координат по формуле:

$$r_i = \sqrt{x_i^2 + y_i^2};$$

- величины углов в градусах, которые образуют эти отрезки с осью OX по формуле:

$$a_i = arctan\frac{y_i}{x_i};$$

- указать точки (точку), расположенные на максимальном расстоянии от центра координат;
- указать номер отрезка, который образует минимальный угол с осью OX.

2. Даны квадратная матрица **A** размером **N** на **N**, и вектор **B** размером **N (N = 4)**, элементы которых определены следующим образом:

$$a_{ij} = \sqrt{\left|ln\frac{i+2}{j+1}\right|}, \qquad b_i = \sin^2\left(\frac{i}{2}\right).$$

Выполнить следующие действия:

- вычислить матричное выражение:

$$\frac{1}{2} \cdot A^4 \cdot B - (A^{-1})^2 \cdot B;$$

- вычислить сумму элементов первых двух строк матрицы **A**, и произведение элементов вектора **B**;
- получить новый вектор **C**, каждый элемент которого определяется как:

$$c_i = b_i + \max_{j=0..N-1} a_{ij} + \min_{j=0..N-1} a_{ji};$$

- найти сумму векторов **B** и **C**;
- вывести количество минимальных элементов матрицы **A.**

3. Человек, стоящий на обрыве, бросает камень горизонтально. С помощью интервальной съемки получили траекторию движения камня. В список **x_list** и **y_list** занесены его координаты, отснятые в некоторые моменты времени:

```
x_list = np.array((1.1,1.7,2.4,3.0,3.5,4.3,4.8,5.5,6.0,6.6,7.3,8.1))
h_list = np.array((10.8,10.7,10.4,10.5,9.7,10.2,9.4,9.4,8.3,8.2,7.8, 7.1))
```

Реализуйте программу, в которой:

- найдите траекторию движения камня, используя в качестве линии тренда полином второй степени;
- ответьте на вопросы:

- с какой высоты человек бросал камень?
- попадет ли камень в мишень, если ее центр расположен в точке **x = 7.1** на высоте **h = 7.9**, а радиус равен **0.29**?

- постройте график, на котором выведите исходные точки и линию тренда.

Вариант 3

1. Даны **n** населенных пунктов, координаты которых в некоторой относительной системе координат представлены в таблице:

	1	2	3	4	5	6	7
x, км	71	79	12	45	88	53	43
y, км	91	10	48	43	45	3	19

В населенном пункте с номером **m = 4**, предполагается установка радиостанции с радиусом действия равным 100 км.

Вычислить:

- расстояние от выбранного населенного пункта до всех остальных пунктов по формуле:

$$r_i = \sqrt{(x_i - x_m)^2 + (y_i - y_m)^2};$$

- какие населенные пункты попадают в радиус действия радиостанции (расстояние от пункта **m** до этих пунктов меньше или равно 100 км;
- номер (номера) населенных пунктов, расположенные на максимальном расстоянии от пункта **m**;
- количество населенных пунктов, попадающих в радиус действия радиостанции.

2. Даны квадратная матрица **A**, размером **N** на **N**, и вектор **B**, размером **N (N = 4)**, элементы которых определены следующим образом:

$$a_{ij} = 10 \cdot cos\left(\frac{\pi \cdot j}{i+1}\right), \qquad b_i = \sin^2\left(i + \frac{\sqrt{3}\pi}{2}\right).$$

Выполнить следующие действия:

- вычислить матричное выражение:

$$(5 \cdot A^{-1})^2 \cdot B - A^3 \cdot B;$$

- вычислить сумму произведений элементов строк матрицы A и сумму элементов вектора B;
- получить новый вектор **C**, каждый элемент которого определяется как:

$$c_i = b_i \cdot \frac{\max\limits_{j=0..N-1} a_{ij}}{\operatorname{mean}\limits_{j=0..N-1} a_{ji}};$$

- найти скалярное произведение векторов **B** и третьего столбца матрицы **A**;
- вывести количество нулевых элементов матрицы **A.**

3. Баскетболист бросает мяч в корзину. С помощью интервальной съемки получили траекторию движения мяча. В список **x_list** и **h_list** занесены его координаты, отснятые в некоторые моменты времени:

```
x_list = np.array((1.641,3.03,4.11,5.6,6.8,8.261,9.362,10.49,12.5,13.287))
h_list = np.array((3.98,5.04,6.05,6.64,7.46,8.04,7.79,8.243,8.251,7.779))
```

Реализуйте программу, в которой:

- найдите траекторию движения мяча, используя в качестве линии тренда полином второй степени;
- ответьте на вопросы:

- с какой высоты баскетболист бросал мяч?

- попадет ли мяч в корзину, если ее центр расположен в точке **x = 16.573** на высоте **h = 3.05**, а радиус равен **0.24**?

- постройте график, на котором выведите исходные точки и линию тренда.

Вариант 4

1. Электрическая цепь состоит из **n** ветвей, каждая ветвь состоит из резистора и источника ЭДС включенных последовательно. Известны сопротивление резистора $\mathbf{r_i}$ каждой ветви и сила тока в каждой ветви $\mathbf{i_i}$:

	1	2	3	4	5	6	7	8	9	10
r, Ом	0.6	1.5	1.5	1.5	1.1	0.9	0.7	1.0	0.6	1.2
i, A	3.7	3.1	3.4	3.2	3.7	4.1	2.8	2.8	3.9	3.5

Все **n** ветвей соединены параллельно. Направления токов в ветвях одинаковые.

Вычислить:

- эквивалентное сопротивление цепи по формуле (сначала вычислите значение **1/R**, а потом получите **R** – единица, деленная на полученное значение):

$$\frac{1}{R} = \frac{1}{r_1} + \frac{1}{r_2} + \cdots + \frac{1}{r_n};$$

- силу тока в цепи по формуле:

$$I = i_1 + i_2 + \cdots + i_n;$$

- напряжение в цепи по формуле:

$$U = I \cdot R;$$

- в другую цепь включили резисторы с номерами **3** и **9**, соединили их последовательно, напряжения на концах каждого резистора равна **1.4В** и **1.0В** соответственно. Вычислить силу тока в цепи по формуле:

$$I = \frac{U}{R} = \frac{U_1 + U_2}{R_1 + R_2},$$

где $U_1,\ R_1$ – напряжение и сопротивление на первом резисторе;

$U_2,\ R_2$ – напряжение и сопротивление на втором резисторе.

2. Даны квадратная матрица **A**, размером **N** на **N**, и вектор **B**, размером **N (N = 4)**, элементы которых определены следующим образом:

$$a_{ij} = \ln \sqrt[3]{i + j + 3}, \qquad b_i = \cos\left(\frac{i^2}{\sqrt{3}}\right).$$

Выполнить следующие действия:

- вычислить матричное выражение:

$$A^2 \cdot B - 3 \cdot (A^{-1})^2 \cdot B;$$

- вычислить сумму элементов первых трех строк матрицы A и произведение элементов вектора B;

- получить новый вектор **C**, каждый элемент которого определяется как:

$$c_i = cos|b_i| + \max_{j=0..N-1} a_{ij};$$

- найти скалярное произведение вектора **C** на первый столбец матрицы A.

- вывести количество элементов вектора B, которые равны минимальному элементу матрицы **A**.

Задание 3

Человек, стоящий на обрыве, бросает камень горизонтально. С помощью интервальной съемки получили траекторию движения камня. В список **x_list** и **y_list** занесены его координаты, отснятые в некоторые моменты времени:

```
x_list = np.array((1.35,3.75,6.33,9.17,11.16,13.85,15.96,18.26,20.12))
h_list = np.array((22.14,21.25,21.13,18.72,18.49,15.38,12.41,10.17, 7.6))
```

Реализуйте программу, в которой:

- найдите траекторию движения камня, используя в качестве линии тренда полином второй степени.
- ответьте на вопросы:

- с какой высоты человек бросал камень?

- попадет ли камень в мишень, если ее центр расположен в точке в точке **x = 18.292** на высоте **h = 10.111**, а радиус равен **0.21**?

- постройте график, на котором выведите исходные точки и линию тренда.

Вариант 5

Известны координаты и массы **n** материальных точек:

	1	2	3	4	5	6	7	8	9	10	11	12	13
x, см	9.4	4.0	-4.2	3.6	-0.9	-7.1	-7.5	-6.0	6.9	8.5	5.4	-7.4	-5.0
y, см	2.8	6.4	-2.9	-7.4	-7.3	-6.3	7.3	8.4	-7.8	2.7	7.3	3.6	2.6
m, грамм	8	5	15	10	13	15	7	2	5	15	5	3	3

Вычислить:

- координаты центра масс для заданной системы точек по формуле:

$$x_m = \frac{x_1 \cdot m_1 + x_2 \cdot m_2 + \cdots x_n \cdot m_n}{m_1 + m_2 + \cdots + m_n};$$

$$y_m = \frac{y_1 \cdot m_1 + y_2 \cdot m_2 + \cdots y_n \cdot m_n}{m_1 + m_2 + \cdots + m_n};$$

- расстояние от каждой точки до центра масс по формуле:

$$r_i = \sqrt{(x_i - x_m)^2 + (y_i - y_m)^2};$$

- указать точки (точку), расположенные на максимальном расстоянии от центра масс.

2. Даны квадратная матрица **A**, размером **N** на **N**, и вектор **B**, размером **N (N = 4)**, элементы которых определены следующим образом:

$$a_{ij} = sin\left(\frac{(j+1)^2}{\pi \cdot (i+1)}\right), \qquad b_i = \sin^2\left(i^2 + \frac{\sqrt{3}\pi}{6}\right).$$

Выполнить следующие действия:

- вычислить матричное выражение:

$$A^2 \cdot (A^{-1})^2 \cdot B - \frac{3}{4} \cdot B;$$

- вычислить сумму элементов первых трех столбцов матрицы **A** и произведение элементов вектора **B**;

- получить новый вектор **C**, каждый элемент которого определяется как:

$$c_i = |b_i|^3 + \frac{\max\limits_{j=0..N-1} a_{ij}}{\operatorname{mean}\limits_{j=0..N-1} a_{ji}};$$

- найти скалярное произведение вектора **C** и первого столбца матрицы **A**.

- вывести номера минимальных элементов вектора **B**.

3. Баскетболист бросает мяч в корзину. С помощью интервальной съемки получили траекторию движения мяча. В список **x_list** и **h_list** занесены его координаты, отснятые в некоторые моменты времени:

```
x_list = np.array((1.25,2.17,2.83,3.54,4.58,5.43,5.98,7.04,7.79,8.55))
h_list = np.array((3.01,3.45,4.2,4.22,4.86,4.87,5.0,5.07,4.85, 4.92))
```

Реализуйте программу, в которой:

- найдите траекторию движения мяча, используя в качестве линии тренда полином второй степени;
- ответьте на вопросы:

- с какой высоты баскетболист бросал мяч?

- попадет ли мяч в корзину, если ее центр расположен в точке **x = 11.8** на высоте **h = 3.05**, а радиус равен **0.28**?

- постройте график, на котором выведите исходные точки и линию тренда.

4. СИМВОЛЬНЫЕ ВЫЧИСЛЕНИЯ

SymPy – это библиотека символьной математики языка Python. Она является реальной альтернативой таким математическим пакетам как Mathematica или Maple.

Библиотека поддерживает символьные вычисления. Символьные вычисления – это преобразования и работа с математическими формулами как с последовательностью символов, которые используются в традиционных ручных методах.

С помощью обычных выражений в языке Python осуществляются численные расчеты, которые оперируют приближёнными числовыми значениями.

Например, квадратный корень числа с помощью модуля **math** в Python вычисляется вот так:

```
import math
print(math.sqrt(3))
```

Результат:

```
1.7320508075688772
```

Полученное значение является приближенным значением корня из 3, посчитанное с некоторой точностью. Если возвести это значение в квадрат, мы не получим целое число 3, так как возводим во вторую степень неточный корень из 3:

```
print(math.sqrt(3) ** 2)
```

Результат:

```
2.9999999999999996
```

Модуль SymPy позволяет получать и использовать точные значения в символьном виде. Так, пример с извлечением квадратного корня и возведением его в квадрат будет выглядеть следующим образом:

```
import sympy
print(sympy.sqrt(3))
print(sympy.sqrt(3) ** 2)
```

Результат:

```
sqrt(3)
3
```

Значение корень из 3 сохраняется в символьном виде как точное значение (`sqrt(3)`), поэтому при возведении в квадрат получается целое число 3.

Модуль SymPy преобразует выражение так, как если бы мы это делали «вручную». Например, вычислим корень из 12 «вручную»:

$$\sqrt{12} = \sqrt{4 \cdot 3} = 2 \cdot \sqrt{3}.$$

SymPy это делает аналогично, в то время как с помощью модуля **math** можно получить только приближенное число:

```
print(math.sqrt(12))
print(math.sqrt(12) ** 2)
```

Результат:

```
3.4641016151377544
11.999999999999998
```

```
print(sympy.sqrt(12))
print(sympy.sqrt(12) ** 2)
```

Результат:

```
2*sqrt(3)
12
```

Чтобы преобразовывать математические выражения и выполнять различные операции, нужно явно объявить математический символ (в понятиях языка программирования – переменную) как символьную переменную. Для этого используется оператор:

```
x = sympy.Symbol("x")
```

Теперь с помощью этого оператора можно записать различные математические выражения:

```
y = x ** 2 - x - 6
print(y)
z =  (x - 1) ** 3
print(z)
```

Результат:

```
x**2 - x - 6
(x - 1)**3
```

Выражения сохраняются и обрабатываются в символьном виде. Разложим, например, выражение $x^2 - x - 6 = (x - x_1) \cdot (x - x_2)$ на множители:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{-(-1) + \sqrt{1^2 - 4 \cdot (-6)}}{2} = \frac{1 + \sqrt{25}}{2} = 3$$

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} = \frac{-(-1) - \sqrt{1^2 - 4 \cdot (-6)}}{2} = \frac{1 - \sqrt{25}}{2} = -2$$

Таким образом, имеем

$$x^2 - x - 6 = (x - 3) \cdot (x + 2).$$

Для разложения на множители в SymPy используется функция **`factor()`**, применяя ее, получаем результат аналогичный вычисленному «в ручную»:

```
print(sympy.factor(y))
```

Результат:

```
(x - 3)*(x + 2)
```

Символьные вычисления модуля SymPy подобным образом могут использоваться для символьного интегрирования и дифференцирования, подстановки одних выражений в другие, упрощения формул и т. д.

4.1. Символьные преобразования

С помощью функций модуля **`sympy`** можно выполнить большое количество преобразований математических выражений. В таблице 23 приведены некоторые из них. Все они выполняются только для символьных переменных.

Таблица 23 – Функции символьных преобразований

Оператор	Назначение	Результат

`pprint((x-1)/(x-2))`	Выводит выражение в несколько строк в математическом виде	`x - 1`
		`-----`
		`x - 2`

`fraction((x-1)/(x-2))`	Выделяет числитель и знаменатель, возвращает результат в виде списка	`(x - 1, x - 2)`
`expand((x-1)*(x-2))`	Раскрывает скобки в выражении	`x**2 - 3*x + 2`
`factor(x ** 2 - x - 6)`	Раскладывает выражение на множители	`(x - 3)*(x + 2)`
`apart((2-x)/(x**2-7*x+12))`	Раскладывает выражение на простые дроби	`1/(x-3)-2/(x-4)`
`solve((x**2 - 3*x + 2) < 0)`	Решает неравенство	`(1 < x) & (x < 2)`
`simplify()`	Упрощает выражение	

Пример. Для выражения:

$$\frac{x^4 + x^3 + 2x^2 + 4x - 8}{x^5 - 3x^4 + 9x^3 - 23x^2 + 36}$$

выполнить следующие математические преобразования:

- разложить числитель и знаменатель на множители;

- разложить выражение на элементарные дроби;

- выяснить, при каких значениях x выражение больше нуля.

Код программы, необходимые пояснения приведены в комментариях:

```
# Подключить модульsympy, указать для него имя sp:
import sympy as sp

# Объявить символьную переменную x, указать,
#что она может принимать только действительные значения.
x = sp.Symbol("x", real = True)

# Задать выражение (чтобы часть выражения можно было перенести
# на другую строку, необходимо заключить выражение в круглые скобки):
eq = (( x ** 4 + x ** 3 + 2 * x ** 2 + 4 * x - 8) /
     (x ** 5 - 3 * x ** 4 + 9 * x ** 3 - 23 * x ** 2 + 36))

# Вывести выражение в математическом виде:

print("Выражение:")
sp.pprint(eq)

# Выделить числитель и знаменатель, вывести результаты:
fract = sp.fraction(eq)
print("\nЧислитель и знаменатель:")
sp.pprint(fract)

# Занести числитель и знаменатель в переменные eq_1 и eq_2
eq_1 = fract[0]
eq_2 = fract[1]

# Разложить на множители числитель и знаменатель,
# разложить выражение на простые дроби:
f = sp.factor(eq_1)
print("\nРазложение числителя на множители: ", f)
f = sp.factor(eq_2)
print("Разложение знаменателя на множители: ",f )
f = sp.apart(eq)
print("Разложение выражения на простые дроби: ")
sp.pprint(f)

# Найти интервалы, на которых заданное выражение больше нуля.
result = sp.solve(eq > 0)
print("\nВыражение больше 0: ", result)
```

Результат:

```
Выражение:
    4    3      2
   x  + x  + 2⋅x  + 4⋅x - 8
─────────────────────────────
 5      4      3       2
x  - 3⋅x  + 9⋅x  - 23⋅x  + 36

Числитель и знаменатель:
 4    3      2              5      4      3       2
x  + x  + 2⋅x  + 4⋅x - 8, x  - 3⋅x  + 9⋅x  - 23⋅x  + 36

Разложение числителя на множители:  (x - 1)*(x + 2)*(x**2 + 4)
Разложение знаменателя на множители:  (x - 2)**2*(x + 1)*(x**2 + 9)
Разложение выражения на простые дроби:
 34⋅x + 211          1           1384              32
─────────────── - ───────── + ──────────── + ───────────
     ⎛ 2    ⎞     9⋅(x + 1)   1521⋅(x - 2)             2
169⋅⎝x  + 9⎠                                 39⋅(x - 2)
Выражение больше 0:  (2 < x) | ((-2 < x) & (x < -1)) | ((1 < x) & (x < 2))
```

Задание для самопроверки

Упростить выражение:

$$f(x) = \frac{x^2 - 1}{x^3 - 1}.$$

Результат:

```
(x + 1)/(x**2 + x + 1)
```

4.2. Решение уравнений

4.2.1. Точное решение уравнений

Модуль SymPy позволяет получить точное решение уравнения одной переменной в символьном виде. Для этого используется функция **`solve()`**. В скобках указывается выражение, записанное для символьной переменной. Функция **`solve()`** вычисляет корни уравнения, правая часть которого равна 0. Например, для решения уравнения:

$$x^2 - 2 = 0$$

используется следующий оператор (переменная x должна быть заранее объявлена как символьная):

```
sp.solve(x ** 2 - 2)
```

В выражении можно использовать стандартные математические функции (`sin()`, `cos()`, `log()`, `exp()` и др.), определенные в модуле `sympy` (таблица 24).

Таблица 24 – Математические функции для символьных вычислений

Функция	Описание
`sin(x)`	Синус от **`x`**, используется, если **`x`** – символьная переменная
`cos(x)`	Косинус от **`x`**, используется, если **`x`** – символьная переменная
`tan(x)`	Тангенс от **`x`**, используется, если **`x`** – символьная переменная
`cot(x)`	Котангенс от **`x`**, используется, если **`x`** – символьная переменная
`acos(x)`	Арксинус от **`x`**, используется, если **`x`** – символьная переменная
`asin(x)`	Арккосинус от **`x`**, используется, если **`x`** – символьная переменная
`atan(x)`	Арктангенс от **`x`**, используется, если **`x`** – символьная переменная
`acot(x)`	Арккотангенс от **`x`**, используется, если **`x`** – символьная переменная
`ln(x)`	Натуральный логарифм **`x`**, используется, если **`x`** – символьная переменная
`log(x,base)`	Логарифм **`x`**, по основанию **`base`**, основание **`base`** может быть опущено, тогда вычисляется натуральный логарифм, используется, если **`x`** – символьная переменная
`exp(x)`	Экспонента от **`x`**, используется, если **`x`** – символьная переменная
`abs(x)`	Модуль **`x`**, используется, если **`x`** – символьная переменная
`sqrt(x)`	Корень из **`x`**, используется, если **`x`** – символьная переменная

Результат выполнения функции **`solve()`** – список корней. Каждый корень вычисляется в символьном виде. Например, результат решением уравнения:

$$x^2 - 2 = 0$$

является список из двух элементов:

```
[-sqrt(2), sqrt(2)]
```

Решение уравнение целесообразно сохранить в виде отдельной переменной:

```
result = sp.solve(x ** 2 - 2)
```

Тогда в переменной **`result`** будет храниться список решений. К отдельному корню обращение осуществляются по его индексу, например, можно вывести в консоль каждый корень по отдельности:

```
print(result[0], result[1])
```

В результате получим:

```
-sqrt(2) sqrt(2)
```

Полученные корни не являются числами, чтобы преобразовать их к числовому формату используется функция модуля sympy **N()** или метод **evalf()**. Например, преобразуем полученные корни к числовому виду при выводе:

```
print("x0 =", sp.N(result[0]), "x1 =", result[1].evalf())
```

Результат:

```
x0 = -1.41421356237310 x1 = 1.41421356237310
```

В функции **N()** и методе **evalf()** можно указать количество знаков после запятой:

```
print("x0 =",sp.N(result[0], 4), "x1 =", result[1].evalf(4))
```

Результат (в результат округляется до количества знаков после запятой на единицу меньше, чем указано в скобках):

```
x0 = -1.414 x1 = 1.414
```

Пример. Решим уравнение $f(x) = 0$, если:

$$f(x) = (x-2)^2 - 6.$$

Код программы, необходимые пояснения приведены в комментарии:

```
# импортировать модуль символьных вычислений, дать ему имя
import sympy as sp

# опишем левую часть уравнения в виде функции:
def f(x):
    z = (x - 2) ** 2 - 6
    return z

# определить переменную, относительно которой решается уравнение,
# как символьную переменную:
x = sp.Symbol("x")

# решить уравнение в символьном виде с помощью функции solve():
result = sp.solve(f(x))

# вывести результат в виде списка:
print("Список корней:", result)

# вывести отдельные корни в символьном виде:
print("x0 =", result[0], "x1 =", result[1])

# вывести отдельные корни в числовом виде:
print("x0 =", sp.N(result[0]), "x1 =", sp.N(result[1]))
```

Результат:

```
Список корней: [2 - sqrt(6), 2 + sqrt(6)]
x0 = 2 - sqrt(6) x1 = 2 + sqrt(6)
x0 = -0.449489742783178 x1 = 4.44948974278318
```

Задания для самопроверки

1. Найти область определения функции:

$$f(x) = \frac{x^2+1}{x^2-3}.$$

Для этого необходимо:

- выделить знаменатель функции;
- приравнять знаменатель 0, решить уравнение, корни этого уравнения – точки, в которых функция не существует.

Результат:

```
Знаменатель: x**2 - 3
Точки, в которых функция не существует:  [-sqrt(3), sqrt(3)]
```

2. В символьном виде решить уравнение:

$$\ln(x^2 + 1) = 1.$$

Корни вывести в символьном виде и в виде чисел.

Результат:

```
Список корней: [-sqrt(-1 + E), sqrt(-1 + E)]
x0 = -1.31083249443209 x1 = 1.31083249443209
```

4.2.2. Приближенное решение уравнений

Другим способом решения уравнений является приближенное решение уравнений помощью функции **`nsolve()`**. Эта функция применяется в том случае, если решение точными методами в символьном виде либо невозможно, либо затруднительно. Функция для приближенного решения уравнений следующим образом:

`nsolve(выражение, переменная, начальное приближение)`

где **`выражение`** – это левая часть уравнения, которое нужно решить (правая должна быть равна нулю;

`переменная` – это символьная переменная, относительно которой находится решение;

`начальное_приближение` – начальное приближение корня (функция находит корень в окрестности этого значения).

Результат выполнения функции – число. Например, найдем приближенным методом корни уравнения:

$$x^2 - 2 = 0.$$

Начальные приближения корней обычно определяют по графику. На рисунке 33 приведен график функции $f(x) = x^2 - 2$. Как видно из графика, уравнение имеет два корня (две точки пересечения графика с осью OX), в качестве начальных приближений можно задать числа, расположенные «поблизости» от пересечений. Например, для отрицательного корня это могут быть числа: −2, −1.

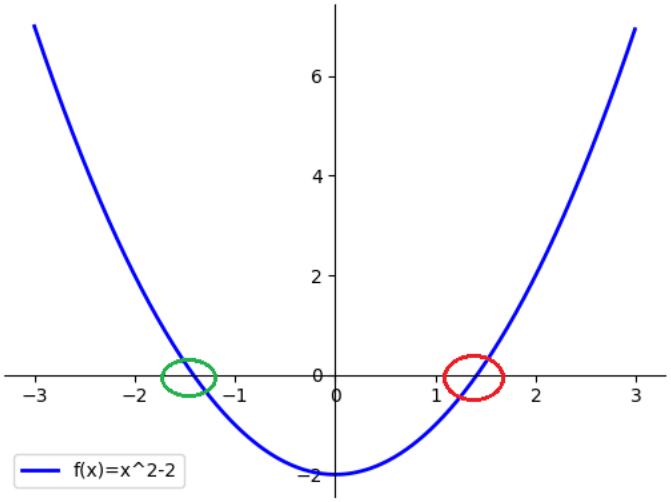


Рисунок 33. График функции

Код программы:

```
x = sp.Symbol("x")

# решаем уравнение относительно x, начальное приближение -1
x0 = sp.nsolve(x ** 2 - 2, x, -1)
print("x0 =", x0)

# решаем уравнение относительно x, начальное приближение 1
x1 = sp.nsolve(x ** 2 - 2, x, 1)
print("x1 =", x1)
```

Давайте сравним значения корня **x0**, полученные точным и приближенным методом:

```
x0_точное = -1.41421356237310
x0_приближенное = -1.41421356237310
```

Эти значения совпадают до 14 знака после запятой. Это говорит о том, что для решения уравнений с этой точностью можно использовать оба метода. При этом приближенное решение является менее ресурсоёмким.

Пример. Решим уравнение $f(x) = 0$, если:

$$f(x) = x^3 - 6x^2 + x + 5.$$

Код программы, необходимые пояснения приведены в комментариях:

```
# сначала необходимо построить график функции, по которому определить начальные приближения
# импортируем модуль для рисования, дать ему имя
import matplotlib.pyplot as plt

# импортировать модуль символьных вычислений, дать ему имя
import sympy as sp

# опишем левую часть уравнения в виде функции
# (если функция включает математические функции sin(), cos() и пр.
# использовать функции из модуля sympy:
def f(x):
    z = x ** 3 - 6 * x ** 2 + x + 5
    return z

# интервал построения, шаг, количество точек построения 100:
a = -2
b = 7
h = (b-a)/100
# список координат по оси OX
x_list = [a + i*h for i in range(0,101)]

#список координат по оси OY, значения которого посчитаны
y_list = [f(x) for x in x_list]

#Сформируем график функции:
plt.plot(x_list, y_list, 'b-', label="f(x)=x^3-6*x^2+x+5", lw=2)

#Отформатируем оси, выведем легенду и покажем область построения:
plt.gca().spines["left"].set_position("zero")
plt.gca().spines["bottom"].set_position("zero")

plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)

plt.legend(loc="upper center")

plt.show()
```

В результате выполнения программы будет построен график, показанный на рисунке 34. По графику видно, что уравнение имеет три корня, в качестве начальных приближений можно взять числа: −2, 2, 6 соответственно.

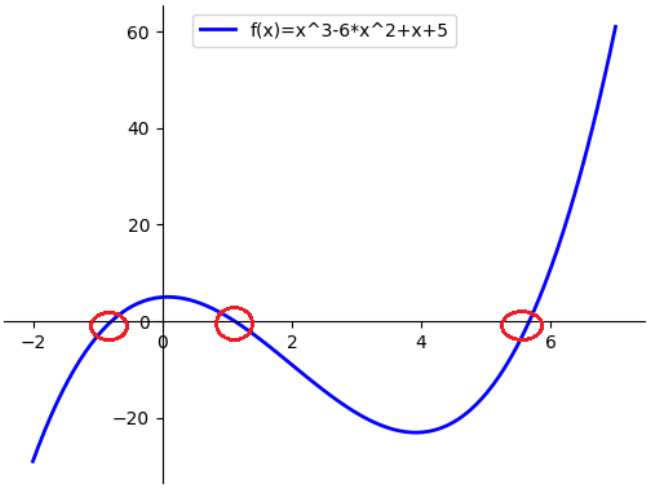


Рисунок 34. График функции

Код программы (продолжение):

```
# определить переменную, относительно которой решается уравнение,
# как символьную переменную:
x = sp.Symbol("x")

# решить уравнение с помощью функции nsolve():
x0 = sp.nsolve(f(x), x, -2)
x1 = sp.nsolve(f(x), x, 2)
x2 = sp.nsolve(f(x), x, 6)

print("x0 =", x0, "x1 =", x1, "x2 =", x2)
```

Результат:

```
x0 = -0.787759039037294 x1 = 1.11983034344540 x2 = 5.66792869559189
```

Задание для самопроверки

Приближенным методом решить уравнение, предварительно построить график:

$$\ln(x^2 + 1) = 1$$

На рисунке 35 показан график функции.

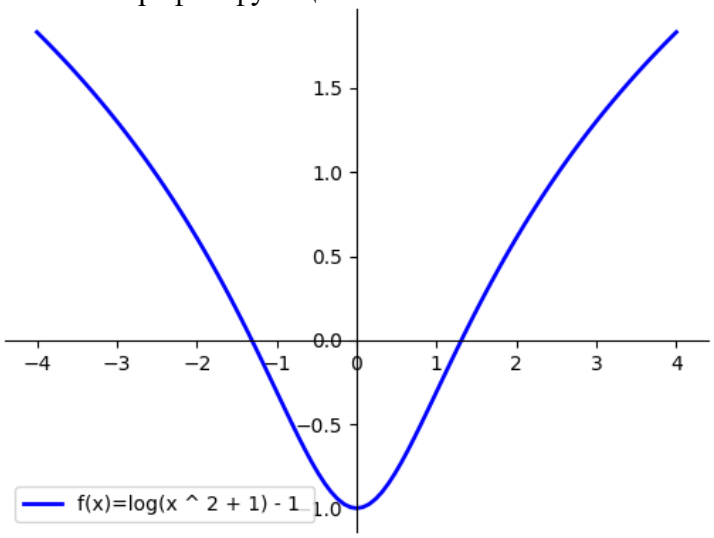


Рисунок 35. График функции

Результат:

```
x0 = -1.31083249443209 x1 = 1.31083249443209
```

4.3. Дифференцирование и интегрирование

В модуле **sympy** реализованы функции, с помощью которых можно выполнять дифференцирование и интегрирование функции. В таблице 25 приведено описание функций и примеры их использования.

Таблица 25 – Символьные операторы дифференцирования и интегрирования

Оператор	Назначение	Результат

`diff(4*x**2 -3*x +5)`
`diff((4*x**2 -3*x +5), x)`
`diff((4*x**2 -3*x +5), x, 1)`
	Находит производную символьного выражения по переменной **x**	`8*x - 3`

`diff((4*x**2 -3*x +5), x, 2)`	Находит вторую производную символьного выражения по переменной**x**	`8`
`integrate(x**2 + 2)` `integrate(x**2 + 2, x)`	Вычисляет неопределенный интеграл символьного выражения по переменной **x**	`x**3/3 + 2*x`
`integrate(x**2 + 2, (x, 1, 2))`	Вычисляет определенный интеграл символьного выражения по переменной **x**, после которой указывается нижний и верхний предел интегрирования	`13/3`

Для символьных выражений можно вычислить их значения в некоторой точке. Для этого используется функция `evalf()`, в качестве аргумента указывается переменная и значение, при каком необходимо вычислить выражение, например:

```
(x ** 2 + 2).evalf(subs={x : 3})
```

Выполняется эта функция следующим образом: вместо символьной переменной (в примере это x) в выражение подставляется значение (в нашем случае 3) и вычисляется результат (в примере результат – 11).

4.3.1. Задача: поиск экстремумов функции

Задача. Найти локальные экстремумы функции:

$$f(x) = x^3 - 6x^2 + x + 5.$$

Пояснение. Локальный экстремум – это точки, в которых первая производная функции равна 0. Эти точки могут быть локальным минимумом (в этом случае вторая производная больше нуля), локальным максимумом (в этом случае вторая производная меньше нуля), точкой перегиба (вторая производная равна 0).

Алгоритм:

- найти первую производную функции;
- приравнять производную нулю, решить уравнение;
- найти вторую производную функции;
- вычислить значения второй производной в точках экстремума;
- определить тип точки экстремума: минимум, максимум, перегиб.

Ниже приведен код программы, необходимые пояснения даны в комментариях.

```
# Подключить модуль sympy, указать для него имя sp:
import sympy as sp

# Описать функцию
def f(x):
    z = x ** 3 - 6 * x ** 2 + x + 5
    return z
```

```
# Объявить символьную переменную x
x = sp.Symbol("x")
# Вывод функции:
print("f(x) =", f(x))
# Найти производную функции f(x) и сохранить ее в переменной d_f:
d_f = sp.diff(f(x), x)
print("первая производная функции:", d_f)

# Решить уравнение d_f = 0:
x_extr = sp.solve(d_f)
print("точки экстремума", x_extr)

# Сохранить точки экстремума в переменные:
x_0 = x_extr[0]
x_1 = x_extr[1]

# Найти вторую производную функции:
d2_f = sp.diff(f(x), x, 2)
print("вторая производная функции:", d2_f)

# Найти значение второй производной в точках экстремума x_0 и x_1,
# сохранить их в переменных f_0_d2 и f_1_d2:
f_0_d2 = d2_f.evalf(subs = {x : x_0})
f_1_d2 = d2_f.evalf(subs = {x : x_1})

# Определить, является точка x_0 точкой
# локального минимума, максимума или перегиба
if f_0_d2 > 0:
    print(x_0, "- локальный минимум")
elif f_0_d2 < 0:
    print( x_0, "- локальный максимум")
else:
   print(x_0, "- точка перегиба")

# Определить, является точка x_1 точкой
# локального минимума, максимума или перегиба
if f_1_d2 > 0:
    print(x_1, "- локальный минимум")
elif f_1_d2 < 0:
    print( x_1, "- локальный максимум")
else:
   print(x_1, "- точка перегиба")
```

Результат:

```
f(x) = x**3 - 6*x**2 + x + 5
первая производная функции: 3*x**2 - 12*x + 1
точки экстремума [2 - sqrt(33)/3, sqrt(33)/3 + 2]
вторая производная функции: 6*(x - 2)
2 - sqrt(33)/3 - локальный максимум
sqrt(33)/3 + 2 - локальный минимум
```

Задание для самопроверки

Найти максимальное и минимальное значения функции $f(x)$ на интервале $[-5, 5]$:

$$f(x) = x^3 - 6x^2 + x + 5.$$

Для этого:

- найти точки экстремума функции;
- выбрать только те из них, которые принадлежат интервалу $[-5, 5]$;
- вычислить значения функции $f(x)$ в точках экстремума и на границах интервала (в точках -5, 5);
- выбрать наибольшее и наименьшее значения функции.

Результат:

```
f(x) = x**3 - 6*x**2 + x + 5
точки экстремума [2 - sqrt(33)/3, sqrt(33)/3 + 2]
f( 2 - sqrt(33)/3 )= 5.04226424709296
f( sqrt(33)/3 + 2 )= -23.0422642470930
f(-5)= -275
f(5)= -15
Наименьшее значение: -275 , в точке: -5
Наибольшее значение: 5.0423 , в точке: 2 - sqrt(33)/3
```

4.3.2. Построение касательной функции

Задача. Найти касательную функции $f(x)$ в точке $x_0 = 4$ и построить ее на графике, обозначить точку касания:

$$y(x) = (x-2)^2 - 6.$$

Пояснение. Уравнение касательной к функции$f(x)$ имеет вид:

$$y - f(x_0) = f'(x_0) \cdot (x - x_0).$$

Это прямая линия, которая описывается уравнением $y = k \cdot x + b$. В нашем случае:

$$y = f'(x_0) \cdot x + (f(x_0) - f'(x_0) \cdot x_0).$$

Следовательно, коэффициенты уравнения касательной вычисляются следующим образом:

$$k = f'(x_0), \qquad b = f(x_0) - f'^{(x_0)} \cdot x_0.$$

Алгоритм вычисления коэффициентов касательной:

- найти производную функции $f(x)$;
- вычислить значение производной в точке x_0;
- вычислить значение функции в точке x_0;
- вычислить коэффициенты k и b по формулам.

Код программы, необходимые пояснения приведены в комментариях:

```
import sympy as sp
import matplotlib.pyplot as plt

# описание функции для построения прямой
def y (x, k, b):
    z = k * x + b
    return z

# описание функции
def f (x):
    z = (x - 2) ** 2 - 6
    return z

x_0 = 3
x = sp.Symbol("x")

# вычислить производную
d_f = sp.diff(f(x), x)

# посчитать значение функции и производной в точке x_0
d_f_x_0 = d_f.evalf(subs = {x : x_0})
f_x_0 = f(x).evalf(subs = {x : x_0})

# вычислить коэффициенты касательной
k = d_f_x_0
b = f_x_0 - d_f_x_0 * x_0
print("уравнение касательной: y = %5.2f * x + %5.2f" % (k, b))
```

```
# построить график функции касательную
a = -2
a1 = 6
h = (a1 - a)/100

# список координат по оси OX
x_list = [a + i * h for i in range(0,101)]

# список координат по оси OY, значения которого посчитаны
f_list = [f(x) for x in x_list]
y_list = [y(x, k, b) for x in x_list]

# линия графика функции:
plt.plot(x_list, f_list, 'r-', label="f(x) = (x-2)^2-6", lw=2)

# линия касательной
plt.plot(x_list, y_list, 'b-', label="y = kx+b", lw=2)

# точка касания
plt.plot([x_0], [f(x_0)], 'go')

# оси, легенда
plt.gca().spines["left"].set_position("zero")
plt.gca().spines["bottom"].set_position("zero")
plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)

plt.legend()

plt.show()
```

Результат:

```
уравнение касательной: y =  2.00 * x + -11.00
```

Также в результате выполнения программы будет построен график функции, касательная и точка касания (рисунок 36).

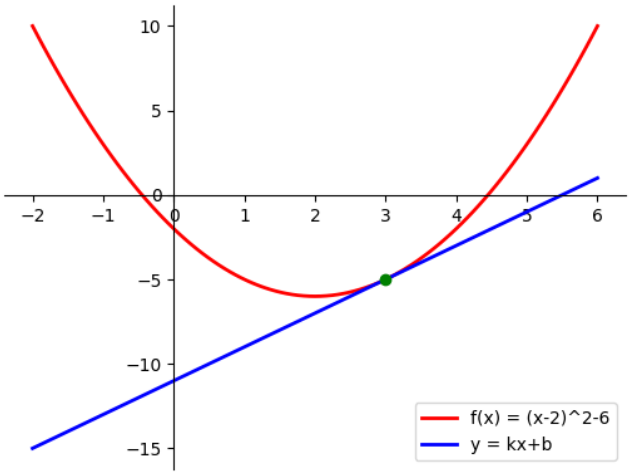


Рисунок 36. Результат выполнения программы

Задание для самопроверки

Найти уравнение касательной функции $f(x)$ в точках -2 и 3, построить график функции, линии касательных, обозначить точки касания:

$$f(x) = x^3 - 6x^2 + x + 5.$$

Результат:

```
уравнение касательной в точке -0.5: y =  7.75 * x +  6.75
уравнение касательной в точке 4: y =  1.00 * x + -27.00
```

Также в результате выполнения программы должны быть построены графики функций (рисунок 37).

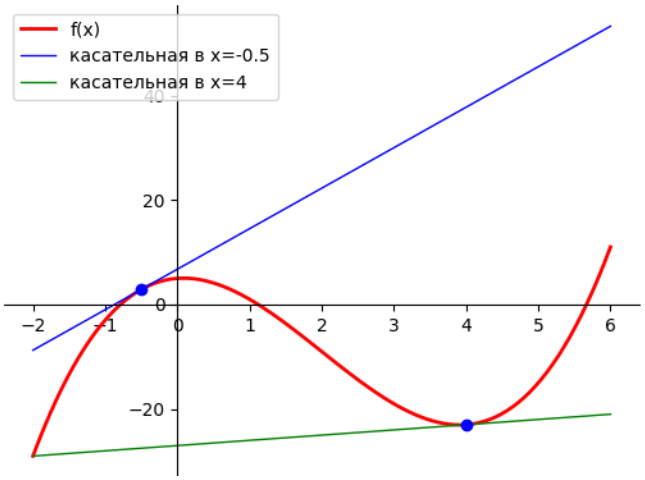


Рисунок 37. Результат выполнения программы

4.3.3. Вычисление неопределенных и определенных интегралов

Задача. Вычислить площадь области, ограниченной графиками функций$f(x)$ и $y(x)$ (рисунок 38):

$$y(x) = (x-2)^2 - 6,\ \ f(x) = x^3 - 6x^2 + x + 5.$$

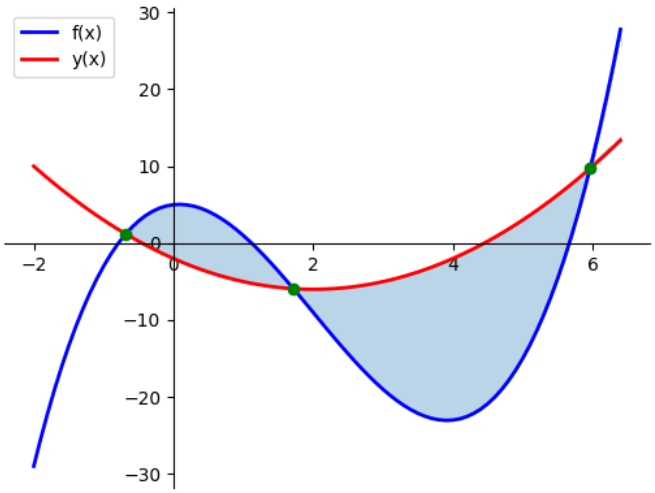


Рисунок 38. Область, ограниченная линиями графиков

Алгоритм вычисления площади:

- найти точки пересечения графиков функции (решить уравнение $f(x) = y(x)$);
- разбить область построения на две части, как показано на рисунке 38;
- найти площадь области I, здесь функция $f(x)$ больше, чем $y(x)$, область ограничена графиками функций и прямыми $x = k_0,\ x = k_1$, следовательно, площадь может быть найдена по формуле:

$$\int_{k_0}^{k_1} \big(f(x) - y(x)\big)dx\,;$$

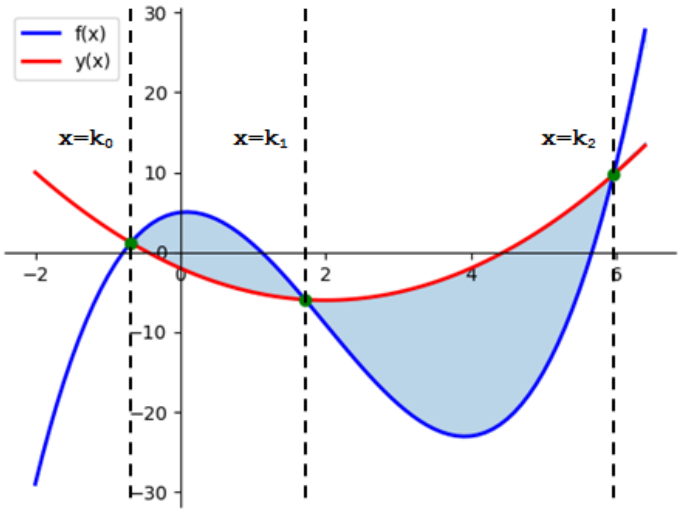


Рисунок 39. Разбиение области

- найти площадь области II, здесь функция $y(x)$ больше, чем $f(x)$, область ограничена графиками функций и прямыми $x = k_1$, $x = k_2$, следовательно, площадь может быть найдена по формуле:

$$\int_{k_1}^{k_2} \big(y(x) - f(x)\big)dx\,;$$

- найти общую площадь фигуры, сложив площади областей I и II.

Ниже приведен код программы, необходимые пояснения даны в комментариях:

```
import sympy as sp

# Описание функций
def y (x):
    z = (x - 2) ** 2 -6
    return z

def f (x):
    z = x ** 3 - 6 * x ** 2 + x + 5
    return z

x = sp.Symbol("x")
# Поиск корней уравнения f(x) - y(x) = 0 приближенным методом
# для поиска начальных значений построить графики функций на одной области
k_0 = sp.nsolve(f(x) - y(x), x, -1)
k_1 = sp.nsolve(f(x) - y(x), x, 2)
k_2 = sp.nsolve(f(x) - y(x), x, 6)
print("Точки пересечения графиков k_0 = %5.3f, k_1 = %5.3f, k_2 = %5.3f" %
      (k_0, k_1, k_2))

# Вычисление площади
s_1 = sp.integrate(f(x) - y(x), (x, k_0, k_1))
s_2 = sp.integrate(y(x) - f(x), (x, k_1, k_2))
s = s_1 + s_2
print("Площадь фигуры, ограниченной графиками функций: %6.3f" % s)
```

Результат:

```
Точки пересечения графиков k_0 = -0.683, k_1 = 1.718, k_2 = 5.965
Площадь фигуры, ограниченной графиками функций: 70.336
```

Задание для самопроверки

Вычислить площадь области, ограниченной графиками функций $f(x)$ и $y(x)$ (предварительно постройте графики функции):

$$y(x) = 2 \cdot x - 6,\ \ f(x) = x^3 - 6x^2 + x + 5.$$

Результат:

```
Точки пересечения графиков k_0 = -1.298, k_1 = 1.449, k_2 = 5.849
Площадь фигуры, ограниченной графиками функции f(x) и y(x): 90.220
```

4.4. Предел функции

Для вычисления предела функции используется функция модуля **`sympy limit()`**. Для вычисления предела:

$$\lim_{x \to x_0} f(x)$$

используется запись:

```
limit(f(x), x, x0)
```

В качестве x_0 можно использовать либо числовое значение, либо бесконечность. В модуле **`sympy`** для обозначения бесконечности служат символы «oo» (две английских прописных о):

```
limit(f(x), x, oo)
```

Для вычисления предела «справа» или «слева» используется запись:

`limit(f(x), x, x0, '+')` и `limit(f(x), x, x0, '-')`

В таблице 26 приведены примеры вычисления пределов, функция предела и константа бесконечности импортированы:

```
from sympy import limit, oo
```

Таблица 26 – Примеры вычисления пределов

Оператор	Описание	Результат
`limit(2*x/(4*x-2), x , oo)`	Вычисляет предел выражения при **x**, стремящемся к бесконечности	`1/2`
`limit(4/(-x ** 2 +2 * x+ 3),x, -1, '-')`	Вычисляет «левый» передел	`-oo`
`limit(4/(-x ** 2 +2 * x+ 3),x, -1, '+')`	Вычисляет «правый» передел	`oo`

4.4.1. Задача, поиск асимптот функции

Задача. Найти уравнения асимптот функции $f(x)$:

$$f(x) = \frac{4 \cdot x}{2 \cdot x + 3}.$$

Пояснение. *Вертикальные* асимптоты следует искать в точках разрыва. Прямая $x = x_b$ является вертикальной асимптотой графика функции $f(x)$, если хотя бы одно из предельных значений:

$$\lim_{x \to x_b^{-0}} f(x) \text{ или } \lim_{x \to x_b^{-0}} f(x)$$

равно $+\infty$ или $-\infty$.

Прямая $y = k \cdot x + b$ является *наклонной* асимптотой графика функции $f(x)$, если

$$\lim_{x \to \infty} |f(x) - k \cdot x - b| = 0.$$

Если наклонная асимптота существует, ее коэффициенты k и b вычисляются по формулам:

$$k = \lim_{x \to \infty} \frac{f(x)}{x}, b = \lim_{x \to \infty} (f(x) - k \cdot x).$$

Горизонтальная асимптота является частным случаем наклонной, если $k = 0$.

Код программы, необходимые пояснения даны в комментариях:

```
import sympy as sp

def f (x):
    z = 4*x/(2*x+3)
    return z

x = sp.Symbol("x")

# точка разрыва
x_b = -3/2

print("Уравнение вертикальной асимптоты:")
lim = sp.limit(f(x), x, x_b)
print ( "предел равен: ",lim, ", уравнение асимптоты x =", x_b)

print("Уравнение наклонной асимптоты:")
k1 = sp.limit(f(x) / x, x, sp.oo)
b1 = sp.limit((f(x) - k1 * x), x, sp.oo)
print(" y = %5.2f *x + %5.2f" % (k1, b1))
```

Результат:

```
Уравнение вертикальной асимптоты:
предел равен:  -oo , уравнение асимптоты x = -1.5
Уравнение наклонной асимптоты:
 y =  0.00 *x +  2.00
```

На рисунке 40 показан график функции и построены асимптоты.

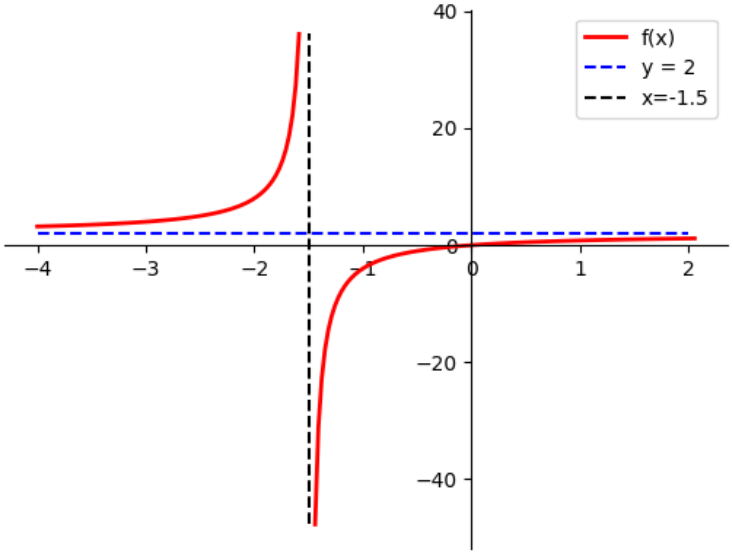


Рисунок 40. График функции и ассимптоты

Задание для самопроверки

Задача. Найти уравнения асимптот функции $f(x)$:

$$f(x) = \sqrt{x^2 + 1}.$$

Результат:

```
Уравнения наклонных асимптот:
 y = -1.00 *x +  0.00
 y =  1.00 *x +  0.00
```

На рисунке 41 показан график функции и асимптоты.

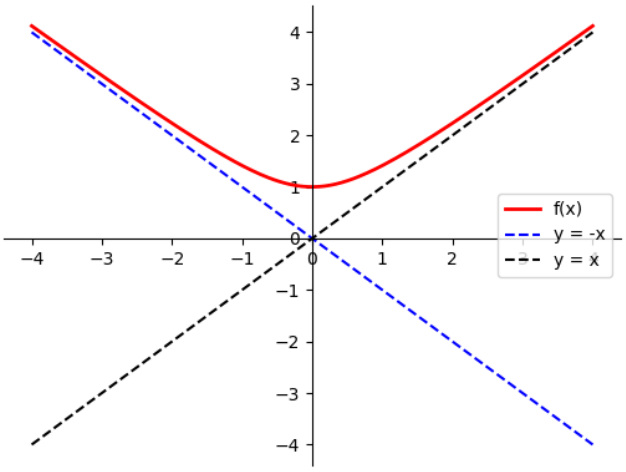


Рисунок 41. График функции и ассимптоты

4.5. Задача: исследование функции и построение ее графика

Задача. Исследовать функцию $f(x)$ и построить ее график.

$$f(x) = x + \frac{1}{x-1}$$

Алгоритм:

1. Найти область определения функции, указать особые точки (точки разрыва):

- выделить знаменатель функции:

$$x - 1;$$

- приравнять знаменатель к нулю, решить уравнение:

$$x - 1 = 0.$$

Результат: функция определена на множестве действительных чисел, кроме $x = 1$, $x_b = 1$ – точка разрыва.

2. Найти точки пересечения с осями координат:

- чтобы найти точки пересечения с осью OX, необходимо решить уравнение:

$$x + \frac{1}{x-1} = 0;$$

- чтобы найти точку пересечения с осью OY, нужно вычислить значение функции в точке $x = 0$:

$$f(0).$$

Результат: точек пересечения с осью OX - нет,
точка пересечения с осью OY – (0, -1).

3. Найти вертикальные, горизонтальные и наклонные асимптоты:

- вертикальные асимптоты нужно искать в точках разрыва, если хотя бы одно из предельных значений:

$$\lim_{x \to x_b - 0} f(x) \text{ или } \lim_{x \to x_b + 0} f(x)$$

равно $+\infty$ или $-\infty$.

- если наклонная асимптота $y = k \cdot x + b$ существует, ее коэффициенты k и b вычисляются по формулам:

$$k = \lim_{x \to \infty} \frac{f(x)}{x}, b = \lim_{x \to \infty} (f(x) - k \cdot x).$$

Результат: вертикальная асимптота: $x = 1$,
наклонная асимптота: $y = 1 \cdot x + 0$.

4. Найти экстремумы функции и интервалы монотонности;

- найти первую производную функции:

$$f'(x);$$

• приравнять производную нулю, решить уравнение, корни уравнения – точки экстремума (x_{ext}):

$$f'(x) = 0;$$

• найти вторую производную функции:

$$f''(x);$$

• вычислить значения второй производной в точках экстремума:

$$f''(x_{ext});$$

• определить тип точки экстремума:

- если $f''(x_{ext}) > 0$, то x_{ext} – минимум,
- если $f''(x_{ext}) < 0$, то x_{ext} – максимум,
- если $f''(x_{ext}) = 0$, то x_{ext} – перегиб;

• найти интервалы возрастания и убывания функции:

- если на промежутке производная функции положительна, то функция возрастает на этом промежутке;

- если на промежутке производная функции отрицательна, то функция убывает.

Результат: точки экстремума $x_{0extr} = 0$, $x_{1extr} = 2$

$x_{0extr} = 0$ – локальный максимум,

$x_{1extr} = 2$ – локальный минимум,

функция возрастает на интервале $(-\infty; 0) \cup (2, +\infty)$,

функция возрастает на интервале $(0; 1) \cup (1; 2)$.

5. Исследовать точки перегиба, найти и интервалы выпуклости/вогнутости:

• исследовать знак второй производной при переходе через точку перегиба:

- если слева (или справа) от точки перегиба значение второй производной больше 0, то на этом интервале функция вогнута;

- если слева (или справа) от точки перегиба значение второй производной меньше 0, то на этом интервале функция выпукла.

Результат: точек перегиба нет.

6. Построить график:

• описать функцию, включить в описание обработку исключений;

• определить количество точек построения n;

• выбрать интервал построения графика $[x_{beg}, x_{end}]$ так, чтобы в него вошли все вычисленные ранее точки (особые точки, экстремумы, точки пересечения с осями) с «запасом»;

• если функция имеет точки разрыва, то разбить интервал на несколько (если одна точка разрыва – то на два, две – на три и т.д):

$$[x_{beg}, x_b - \varepsilon] \cup [x_b + \varepsilon, x_{end}],$$

ε – подобрать произвольно;

• вычислить шаг **h** по формуле:

$$h = \frac{x_{end} - x_{beg}}{n - 1};$$

• вычислить количество точек построения, расположенных до точки разрыва:

$$n_1 = int\left(\frac{x_b - x_{beg}}{h}\right) - 1;$$

• вычислить количество точек построения, расположенных после точки разрыва:

$$n_2 = n - n_1 - 2;$$

• сформировать список (списки) значений изменения абсцисс графика;

• сформировать списки (списки) значений изменения ординат графика;

• построить линию графика для каждой пары списков (абсцисс и ординат).

7. Построить асимптоты:

• для построения вертикальной асимптоты необходимо найти максимальное и минимальное значения ординат, вычисленные по списку (спискам) ординат, тогда для

построения вертикальной асимптоты можно использовать две точки, занесенные в списки абсцисс ($[x_b, x_b]$) и ординат ($[min_f, max_f]$);

- для построения наклонной асимптоты необходимо определить линейную функцию, параметрами которой являются значение x, а также коэффициенты прямой k и b.

8. Обозначить на графике экстремумы функции и точки пересечения с осями.

Код программы, необходимые пояснения приведены в комментариях:

```
import sympy as sp
import matplotlib.pyplot as plt

# линейная функция
def y (x, k, b):
    z = k * x + b
    return z

# функция для исследования
def f(x):
    try:
        z = x + 1/ (x - 1)
    except:
        y = sp.oo
    return z

# символьная переменная, определена для действительных чисел
x = sp.Symbol("x", real=True)

print("Функция:",f(x))

# Область определения:
print("\nОбласть определения:")
fract = sp.fraction(sp.factor(f(x)))
denom_f = fract[1]
print(" знаменатель функции: ", denom_f)
x_solve = sp.solve(denom_f)
x_b = x_solve[0]
print(" функция не определена в точке: ", x_b)

# Точки пересечения с осями
print("\nТочки пересечения с осями:")
result = sp.solve(f(x))
if result ==[]:
    print(" - точек пересечения с осью OX нет")
else:
    print(" - точки пересечения с осью OX: x =", result)
print(" - точка пересечения с осью OY: y =", f(0))

# Уравнения асимптот
print("\nУравнения асимптот:")
lim = sp.limit(f(x), x, x_b, '+')
if abs(lim) ==sp.oo:
    print ( " - вертикальная асимптота: x =", x_b)
else:
    print ( " - x =", x_b, "не является вертикальной асимптотой")

k = sp.limit(f(x) / x, x,  sp.oo)
b = sp.limit((f(x) - k * x), x, sp.oo)
print(" - наклонная асимптота: y = %5.2f *x + %5.2f" % (k, b))

# Точки экстремума и интервалы монотонности
print("\nТочки экстремума и интервалы монотонности:")
# - производная функции f(x):
d_f = sp.diff(f(x), x)
```

```
# - решение уравнения d_f = 0:
x_extr = sp.solve(d_f)
print(" - точки экстремума и интервалы монотонности", x_extr)

# - точки экстремума:
x_0 = x_extr[0]
x_1 = x_extr[1]

# - вторая производная функции:
d2_f = sp.diff(f(x), x, 2)

# - значение второй производной в точках экстремума x_0 и x_1:
f_0_d2 = d2_f.evalf(subs = {x : x_0})
f_1_d2 = d2_f.evalf(subs = {x : x_1})

# - исследование точки x_0
# - локальный минимум, максимум или перегиб
if f_0_d2 > 0:
    print("\t", x_0, "- локальный минимум")
elif f_0_d2 < 0:
    print("\t", x_0, "- локальный максимум")
else:
   print("\t", x_0, "- точка перегиба")

# - исследование точки x_1
# - локальный минимум, максимум или перегиб
if f_1_d2 > 0:
    print("\t", x_1, "- локальный минимум")
elif f_1_d2 < 0:
    print("\t", x_1, "- локальный максимум")
else:
   print("\t", x_1, "- точка перегиба")
# - интервалы монотонности
print(" - интервалы монотонности:")
inc_f = sp.solve(d_f > 0)
dec_f = sp.solve(d_f < 0)
print("\tфункция возрастает на:", inc_f)
print("\tфункция убывает на:", dec_f)
# Построение графика функции
n = 200
x_beg = -2
x_end = 4
h = (x_end - x_beg) / (n - 1)
n_1 = int((x_b - x_beg) / h) - 1
n_2 = n - n_1 - 2

# список со значениями аргумента x до точки разрыва:
x_list_1 = [x_beg + h * i for i in range(n_1)]

# список со значениями аргумента x после точки разрыва:
x_list_2 = [x_b +  2*h + h * i for i in range(n_2)]

# списки со значениями двух частей функций f(x):
f_list_1 = [f(x) for x in x_list_1]
f_list_2 = [f(x) for x in x_list_2]

# линии графиков функций:
line_f_1 = plt.plot(x_list_1, f_list_1, label='f(x)')
line_f_2 = plt.plot(x_list_2, f_list_2)

# стили линий:
plt.setp(line_f_1, color="red", linewidth=2)
plt.setp(line_f_2, color="red", linewidth=2)
```

```
# Построение асимптот
# - наклонная асимптота
#   список со значениями аргумента x на всем интервале:
x_list =  [x_beg + h * i for i in range(n)]

#   список значений линейной функции
f_list_a_o = [y(x, k, b) for x in x_list]

#   линия наклонной асимптоты
plt.plot(x_list, f_list_a_o, "k--", label='асимптота y =' +str(k)+'x +' + str(b))

# - вертикальная асимптота
#   максимальное и минимальное значение функции по двум спискам аргументов
min_f = min(min(f_list_1), min(f_list_2))
max_f = max(max(f_list_1), max(f_list_2))
#   линия вертикальной асимптоты
plt.plot([x_b, x_b], [min_f, max_f], "b--", label='асимптота x='+str(x_b))

# - точки экстремумов
plt.plot([x_0, x_1], [f(x_0), f(x_1)], "go", label='экстремумы')

#Отформатируем оси, выведем легенду и покажем область построения:
plt.gca().spines["left"].set_position("zero")
plt.gca().spines["bottom"].set_position("zero")

plt.gca().spines["top"].set_visible(False)
plt.gca().spines["right"].set_visible(False)

plt.legend()
# loc="upper center"
plt.show()
```

Результат:

```
Функция: x + 1/(x - 1)

Область определения:
 знаменатель функции:  x - 1
 функция не определена в точке:  1

Точки пересечения с осями:
 - точек пересечения с осью OX нет
 - точка пересечения с осью OY: y = -1.0

Уравнения асимптот:
 - вертикальная асимптота: x = 1
 - наклонная асимптота: y =  1.00 *x +  0.00

Точки экстремума и интервалы монотонности:
 - точки экстремума и интервалы монотонности [0, 2]
     0 - локальный максимум
     2 - локальный минимум
 - интервалы монотонности:
    функция возрастает на: (2 < x) | (x < 0)
    функция убывает на: (x > 0) & (x < 2) & Ne(x, 1)
```

Также в результате выполнения программы будет построен график, показанный на рисунке 42.

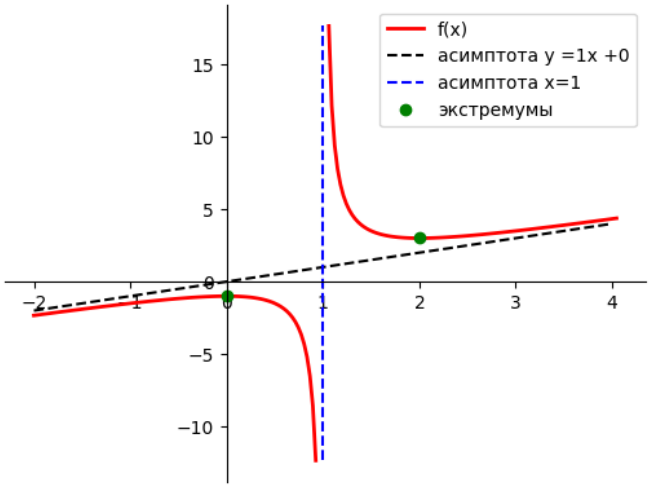


Рисунок 42. Результат выполнения программы

4.6. Варианты заданий для самостоятельного решения

Вариант 1

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{x^2 - 3}{x + 5}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 2

1. Исследовать функцию и построить ее график:

$$f(x) = \left(\frac{x - 1}{x}\right)^2.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 3

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{4 \cdot x}{x^2 - 4}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 4

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{\ln(x + 1)}{x}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 5

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{3 \cdot x^3}{x^2 - 5} + 20.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 6

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{x^2 + 1}{x^2 - 1}$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 7

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{x^3 - 8}{2x + 6}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 8

1. Исследовать функцию и построить ее график:

$$f(x) = \sqrt[3]{\frac{x}{(x-1)^2}}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 9

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{x^2 + 8}{\sqrt{x^2 - 4}}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 10

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{x - 8}{x^2 - 3x + 2}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 11

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{e^x}{x^2 - 25}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 12

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{e^x}{\ln(\frac{1}{x-3})}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 13

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{4 - x}{x^2 - 4}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 14

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{4 \cdot x^3}{x^2 - 8}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 15

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{x - 3}{x^2 - 25}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 16

1. Исследовать функцию и построить ее график:

$$f(x) = \sqrt{\frac{x - 1}{x}}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 17

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{\ln(x^2 + 1)}{x^2 - 9}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

Вариант 18

1. Исследовать функцию и построить ее график:

$$f(x) = \frac{\ln(x^2 + 1)}{x^2}.$$

2. Построить уравнение касательной к той же функции, точку касания выбрать произвольно.

ЗАКЛЮЧЕНИЕ

Python – высокоуровневый язык общего назначения. Это значит, что такой язык эффективен и удобен в работе, а программы, написанные на нем, просты для понимания программистами.

По данным на 2018 год насчитывалось семь миллионов программистов, использующих Python. Такая популярность вызвана универсальностью и простотой изучения языка – он становится главным ориентиром для новичков. Чтобы работать с ним, необязательно быть продвинутым программистом.

Вот пять причин, почему Python актуален.

1. Питон перечеркнул миф о сложности разработки. У языка понятный синтаксис, который базируется на английском языке. На Python легко писать и его легко читать.
2. Большое количество справочной литературы, её много в открытом доступ: книги, сайты, форумы, видеоролики, платные и бесплатные курсы.
3. Множество инструментов. Для Питона создано множество инструментов, фреймворков и сред разработки, которые позволяют упростить решение многих задач.
4. Минимализм. Не нужно писать лишний код. Динамическая типизация и другие функции языка дают возможность меньше заморачиваться над шаблонностью кода и упрощать его.
5. Востребованность специалистов. Количество открытых вакансий на должность Python-разработчика – показатель востребованности специалистов. По данным сайта вакансий **hh.ru,** на июль 2021 года открыто более семи тысяч вакансий на должность Python-разработчиков и инженеров по всей России.

После изучения языка Python легко освоить другие языки программирования.

СПИСОК ЛИТЕРАТУРЫ

1. Бэрри П. Изучаем программирование на Python. М.: Эксмо, 2016. 332 с.
2. Васильев А.Н. Python на примерах. Практический курс по программированию. М.: Наука и техника, 2016. 432 с.
3. Гниденко И.Г., Павлов Ф.Ф., Федоров Д.Ю.Технология разработки программного обеспечения: учеб. пособие для СПО. М.: Юрайт, 2017. 235 с.
4. Гуриков С.Р. Основы алгоритмизации и программирования на Python: учеб. пособие. М.: Инфра-М: Форум, 2018. 707 с.
5. Гэддис Т. Начинаем программировать на Python. 4-е изд.: пер. с англ. СПб.: БХВ-Петербург, 2019. 768 с.
6. Доусон М. Программируем на Python. СПб.: Питер, 2014. 416 с.
7. Златопольский Д.М. Основы программирования на языке Python. М.: ДМК Пресс, 2017. 284 с.
8. Лутц М. Изучаем Python. 4-е изд.: пер. с англ. СПб.: Символ-Плюс, 2011. 1280 с.
9. Рамальо Л. Python. К вершинам мастерства. М.: ДМК Пресс, 2016. 768 с.
10. Любанович Б. Простой Python. Современный стиль программирования. СПб.: Питер, 2016. 480 с.
11. МакГрат М. Программирование на Python для начинающих. М.: Эксмо, 2015. 192 с.
12. Рейтц К., Шлюссер Т. Автостопом по Python. СПб.: Питер, 2017. 336 с.
13. Робертсон Л.А. Программирование — это просто: пошаговый подход / [пер. с 4-го англ. изд. О.С. Журавлевой]; под ред. С.М. Молявко. М.: Бином. Лаб. знаний, 2008. 383 с.
14. Саммерфилд М. Программирование на Python 3. Подробное руководство. М.: Символ, 2016. 608 с.
15. Свейгарт Эл. Автоматизация рутиных задач с помощью Python: практическое руководство для начинающих: пер. с англ. М.: Вильямс, 2016. 592 с.
16. Северенс Ч. Введение в программирование на Python. М.: Интуит, 2016. 231 с.
17. Федоров Д.Ю. Программирование на языке высокого уровня Python: учебное пособие для прикладного бакалавриата. 2-е изд., перераб. и доп. М.: Юрайт, 2019. 161 с. URL: https://urait.ru/bcode/437489 (дата обращения: 13.02.2020).
18. Мэтиз Э. Изучаем Python. Программирование игр, визуализация данных, веб-приложения. СПб.: Питер, 2020. 760 с.

Приложение А. Описание работы в среде IDLE(Python)

Алгоритм

1. Создать папку для хранения своих программ, например **program_Python**.
2. Открыть среду для создания программ на языке Python (**Пуск → Все программы → Python → IDLE**). В результате откроется окно с консолью Python (рисунок А.1).

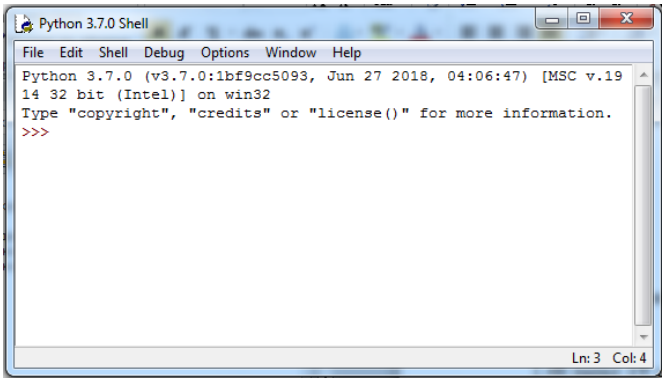


Рисунок А.1. Консоль Python

3. Создать новый документ (**File → NewFile** или нажать клавиши **Ctrl+N**). В результате будет создан новый документ, который отобразится в отдельном окне (рисунок А.2).

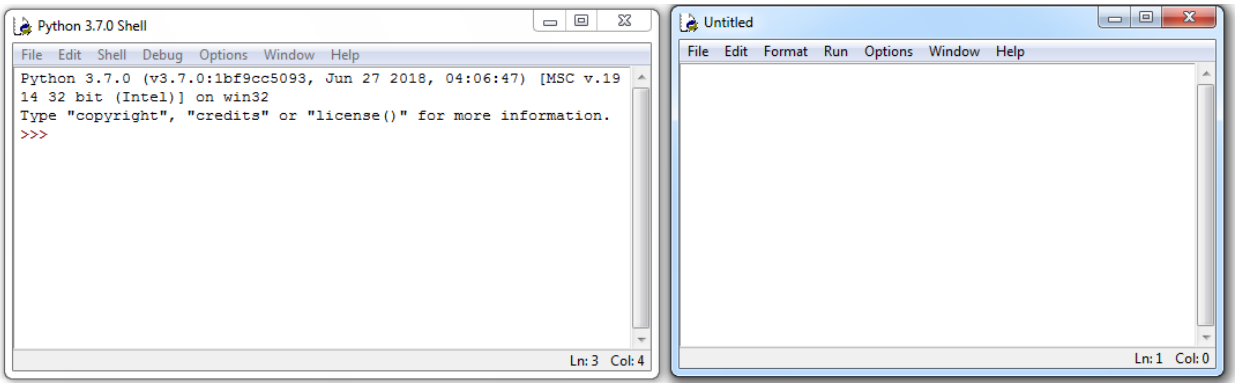
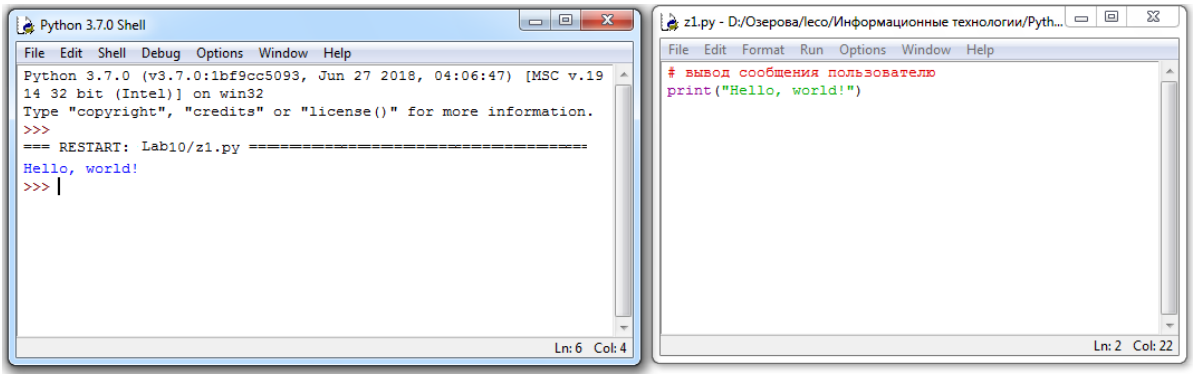


Рисунок А.2. Консоль Python и новый файл

4. Сохранить документ в папке **program_Python**, например, под именем **z1**.

5. Реализовать программу, которая выводит пользователю сообщение «Hello, world!» (рисунок А.3б).

6. Запустить программу на выполнение в окне с документом (**Run → Run Module** или нажать клавишу **F5**), подтвердить запрос о сохранении файла, результат выполнения программы показан на рисунок А.3а.



а) б)

Рисунок А.3. Код программы и результат ее выполнения

Приложение Б. Установка библиотек Python

Установку библиотек Python можно осуществлять с помощью pip. Pip – это специальный менеджер библиотек и модулей Python. Как правило, pip включен в стандартный набор программ, который поставляется вместе с установочным файлом Python.

Алгоритм установки библиотек Python (для Windows)

1. Найти папку, в которой размещена программа **`pip`** или **`pip3`**, например, это путь к папке на моем компьютере

   `C:\Users\OGP\AppData\Local\Programs\Python\Python39\Scripts`

можно скопировать этот путь из командной строки папки `Script` (рисунок Б.1).

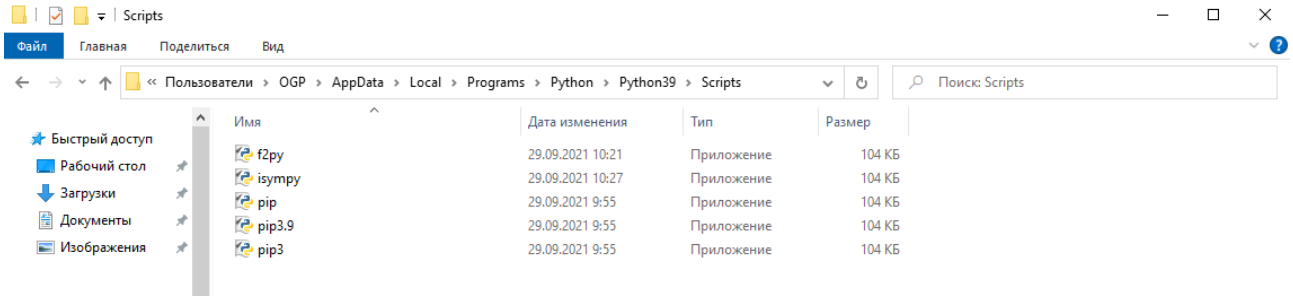


Рисунок Б.1. Папка, где расположена программа pip

2. Открыть приложение «Командная строка»
3. Перейти в папку pip с помощью команды **`cd`** (рисунок Б.2)
4. Инсталировать нужный пакет с помощью команды (рисунок Б.2):

   `pip install имя_пакета`

желательно использовать последнюю версию `pip`, например, `pip3`:

   `pip3 install имя_пакета`

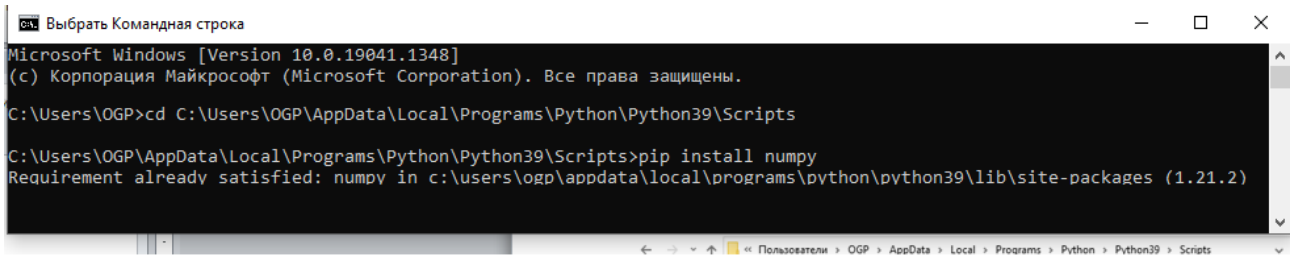


Рисунок Б.2. Использование командной строки для установки библиотек

Приложение В. Стили библиотеки matplotlib.pyplot

Таблица В.1 – Стиль маркеров

Символ	Описание
'.'	точка
'o'	круг
'v'	треугольник, вершина вниз
'^'	треугольник, вершина вверх
'+'	плюс
'x'	крестик
'_'	подчеркивание

Таблица В.2 – Стиль линий

Символ	Описание
'-' или 'solid'	сплошная линия
'--' или 'dashed'	пунктирная линия
'-.' или 'dashdot'	штрих-пунктирная линия
':' или 'dotted'	линия из точек
' 'или 'None'	нет линий

Таблица В.3 – Цвета

Символ	Описание
'b'или blue	синий
'g'или green	зеленый
'r'или red	красный
'c' или cyan	голубой
'm' или magenta	пурпурный
'y' или yellow	желтый
'k'или black	черный
'w' или white	белый