

# Вычисления на основе Python

## Вычислительные методы (1)

Лю Шисян (刘世贤)

Ассистент  
Кафедра теплофизики (Э6)  
МГТУ им. Н.Э. Баумана  
[lyu@bmstu.ru](mailto:lyu@bmstu.ru)



- 1 Основы Python**
- 2 Библиотека Numpy**
- 3 Библиотека SciPy**
- 4 Библиотека Matplotlib**

# Содержание

- 1 Основы Python
- 2 Библиотека Numpy
- 3 Библиотека SciPy
- 4 Библиотека Matplotlib

# Почему стоит изучать Python

- ★ **Простота и читаемость** Синтаксис интуитивно понятен, близок к английскому языку. Легко начать писать код «с нуля».
- ★ **Универсальность** Один и тот же язык — для научных расчётов, анализа данных, автоматизации, веб-разработки, машинного обучения и т.д.
- ★ **Богатая экосистема** NumPy, SciPy, Matplotlib, Pandas, TensorFlow, PyTorch и многие другие библиотеки.
- ★ **Сильное сообщество** Миллионы пользователей, огромное количество туториалов, блогов, документации и примеров.
- ★ **Кроссплатформенность** Работает на Windows, Linux и macOS, легко интегрируется с C/C++, Fortran, MATLAB и т.д.
- ★ **Востребованность на рынке** Один из самых популярных языков в индустрии и науке: анализ данных, финансы, ИИ, инженерные расчёты.

## Популярные IDE для Python

Даже простой текстовый редактор подходит, но для работы удобнее использовать IDE:

### 1. PyCharm

- Community – бесплатная версия, Professional – платная.
- Рекомендуется для средних и крупных проектов.

### 2. VS Code

- Лёгкий редактор с огромным количеством расширений.
- Плагины для Python, Jupyter, Git, удалённой работы по SSH.
- Рекомендуется тем, кто хочет гибкую настроенную среду.

### 3. Spyder

- Похож на MATLAB (консоль, переменные, графики).
- Удобен для численных расчётов и анализа данных.
- Рекомендуется новичкам и тем, кто привык к MATLAB.

### 4. Jupyter Notebook

- Код + текст + формулы + графики в одном файле.
- Рекомендуется для лабораторных, отчётов и визуализаций.

# Инструменты для управления средой Python

Чтобы не «ломать» систему и разные проекты друг другу, мы используем виртуальные среды:

## 1. Anaconda

- Большой готовый набор библиотек + IDE (Jupyter, Spyder).
- Хороший выбор для начала и научных расчётов.

## 2. Miniconda

- «Лёгкая» версия Anaconda: только Python и conda.
- Все пакеты устанавливаются по мере необходимости.
- Подходит тем, кто хочет контролировать размер и состав среды.

## 3. venv

- Стандартный инструмент Python.
- Создаёт изолированные среды без установки сторонних программ.
- Удобен для небольших проектов и серверов без Anaconda.

Руководство по установке доступно: [русский](#) и [китайский](#)

## Типы данных: числа

- ★ Целые числа (**int**): например, 10, -5
- ★ Числа с плавающей точкой (**float**): например, 3.14
- ★ Комплексные числа (**complex**): например, 3+4j

- ▶ Сложение: a + b
- ▶ Вычитание: a - b
- ▶ Умножение: a \* b
- ▶ Деление: a / b
- ▶ Целочисленное деление: a // b
- ▶ Остаток от деления: a % b
- ▶ Возведение в степень: a \*\* b

```

1 a = 5
2 b = 2
3
4 a + b # 7
5 a - b # 3
6 a * b # 10
7 a / b # 2.5
8 a // b # 2
9 a % b # 1
10 a ** b # 25

```

- 📎 None — это специальное значение, обозначающее “ничего” или “нет значения”.

```

1 a = None
2 type(a)      # <class 'NoneType'>
3 a is None    # True

```

## Типы данных: строки (str)

★ **Строки (str):** представляют текстовые данные, например:  
"hello" или 'world'

- Соединение: `s1 + s2`
- Повторение: `s1 * 3`
- Индексация: `s1[0]` возвращает первый символ
- Срез: `s1[0:3]` возвращает подстроку
- Длина строки: `len(s1)`
- Разделение: `s.split()`
- Замена: `s1.replace('h', 'H')`

```

1  s1 = "hello "
2  s2 = 'world'
3  s = s1 + s2
4
5  s
6  # 'hello world'
7  s1 * 3
8  # 'hello hello hello '
9  s1[0]      # 'h'
10 s1[0:3]    # 'hel'
11 len(s1)   # 6
12 s.split() 
13 # ['hello', 'world']
14 s.replace('h', 'H')
15 # 'Hello world'
```

📎 Пустая строка также допустима:

```

1  a = ''
2  type(a)    # <class 'str'>
3  len(a)     # 0
```

## Типы данных: булевы (bool)

★ **Булевы (bool):** имеют только два значения –True и False

- Логическое И (and): `x and y`
- Логическое ИЛИ (or): `x or y`
- Логическое НЕ (not): `not x`
- Сравнение: `==`, `!=`, `>`, `<`, `>=`, `<=`
- Преобразование в bool: `bool(x)`
- False-значения: `0`, `''`, `[]`, `None`

```

1 x = True
2 y = False
3
4 x and y      # False
5 x or y       # True
6 not x        # False
7
8 a = 2
9 b = 3
10 a == b        # False
11 a != b        # True
12 a > b         # False
13 a < b         # True
14 (a > b) or (a == b) # False

```

✎ Булевые значения часто используются в условиях:

```

1 age = 20
2 if age >= 18:
3     print("Взрослый")
4 else:
5     print("Несовершеннолетний")

```

## Структура данных: список (list)

- ★ Упорядоченная коллекция
- ★ Можно добавлять, удалять и изменять элементы
- ★ Можно хранить разные типы данных (числа, строки и т.д.)
- ★ Можно использовать повторяющиеся значения
- ★ Можно обращаться к элементам по индексу и делать срезы
- ★ Записывается в квадратных скобках: [ ]

### Операции со списками:

- `append(x)` – добавить в конец
- `insert(i, x)` – вставить на позицию
- `remove(x)` – удалить первое вхождение
- `pop(i)` – удалить и вернуть элемент по индексу
- `lst[i:j]` – срез от i до j (не включая j)

```

1  lst = [1, 2, 3, 4]
2  lst.append(5)
3  # [1, 2, 3, 4, 5]
4  lst.insert(1, 10)
5  # [1, 10, 2, 3, 4, 5]
6  lst.remove(3)
7  # [1, 10, 2, 4, 5]
8  y = lst.pop(3) # y = 4
9  # lst = [1, 10, 2, 5]
10 print(lst[1:3])
# [10, 2]

```

# Структура данных: Кортеж (Tuple)

- ★ Упорядоченная коллекция, как список, но **нельзя изменять**
- ★ Можно хранить разные типы данных
- ★ Можно использовать повторяющиеся значения
- ★ Можно обращаться к элементам по индексу и делать срезы
- ★ Записывается в круглых скобках: ( )

## Что можно делать:

- ➡ `tpl[i]` – доступ к элементу по индексу
- ➡ `tpl[i:j]` – срез (подкортеж)
- ➡ `len(tpl)` – длина кортежа
- ➡ `in` – проверка наличия элемента

## Что нельзя:

- ✗ `append()`, `remove()`, `pop()` – нельзя добавлять или удалять элементы

```

1  tpl = (1, 2, 3, 4)
2
3  tpl[0]          # 1
4  tpl[1:3]        # (2, 3)
5  len(tpl)        # 4
6  3 in tpl        # True
7
8  tpl.append(5)   # Ошибка!

```

## Структура данных: Множество (Set)

- ★ Неупорядоченная коллекция без повторов
- ★ Не допускает дубликатов
- ★ Изменяемый тип: можно добавлять и удалять элементы
- ★ Элементы должны быть неизменяемыми
- ★ Обозначается фигурными скобками: { }

### Основные операции:

- ➡ `add(x)` — добавить элемент
- ➡ `remove(x)` — удалить элемент
- ➡ `|` — объединение
- ➡ `&` — пересечение
- ➡ `-` — разность

```
1 s = {1, 2, 3, 4}
2 s.add(5)      # {1, 2, 3, 4, 5}
3 s.remove(3)   # {1, 2, 4, 5}
4
5 s1 = {1, 2, 3}
6 s2 = {3, 4, 5}
7
8 s1 | s2       # {1, 2, 3, 4, 5}
9 s1 & s2       # {3}
10 s1 - s2      # {1, 2}
```

# Структура данных: Словарь (Dictionary)

- ★ Структура **ключ: значение**
- ★ Ключи уникальны, значения могут повторяться
- ★ С 3.7 версии Python сохраняет порядок добавления
- ★ Обозначается фигурными скобками: {ключ: значение}

## Основные операции:

- d[key] —получить значение по ключу
- d[key] = val —добавить / изменить значение
- del d[key] —удалить пару
- d.keys() —все ключи
- d.values() —все значения
- d.items() —пары ключ-значение

```

1 dic = {'a': 1, 'b': 2, 'c': 3}
2 dic['d'] = 4
3 dic['a'] = 10
4 del dic['b']
5
6 dic.keys()
7 # dict_keys(['a', 'c', 'd'])
8 dic.values()
9 # dict_values([10, 3, 4])
10 dic.items()
11 # dict_items([('a', 10), ...])

```

# Встроенные функции Python

★ В Python есть множество встроенных функций:

Функция	Описание	Пример
<code>abs(x)</code>	Модуль числа	<code>abs(-5) → 5</code>
<code>round(x)</code>	Округление числа	<code>round(3.1415, 2) → 3.14</code>
<code>divmod(a, b)</code>	Целая часть и остаток	<code>divmod(7, 3) → (2, 1)</code>
<code>sum(iterable)</code>	Сумма элементов	<code>sum([1, 2, 3]) → 6</code>
<code>max(iterable)</code>	Наибольшее значение	<code>max([1, 2, 3]) → 3</code>
<code>min(iterable)</code>	Наименьшее значение	<code>min([1, 2, 3]) → 1</code>
<code>len(iterable)</code>	Кол-во элементов	<code>len("hello") → 5</code>
<code>sorted(iterable)</code>	Сортировка списка	<code>sorted([3, 1, 2]) → [1, 2, 3]</code>
<code>open(file)</code>	Открытие файла	<code>f = open('test.txt')</code>
<code>all(iterable)</code>	Все элементы –True?	<code>all([1, 2, 3]) → True</code>
<code>any(iterable)</code>	Есть хотя бы один True?	<code>any([0, None, 3]) → True</code>
<code>bool(x)</code>	Булево значение	<code>bool(0) → False</code>
<code>enumerate(iterable)</code>	Индекс и значение	<code>list(enumerate(['a', 'b'])) → [(0, 'a'), (1, 'b')]</code>
<code>map(func, iterable)</code>	Применить функцию	<code>list(map(lambda x: x+1, [1,2])) → [2, 3]</code>
<code>filter(func, iterable)</code>	Отбор по условию	<code>list(filter(lambda x: x&gt;0, [-1, 2])) → [2]</code>

# Пользовательские функции и анонимные функции (lambda)

## → Пользовательская функция:

Используется ключевое слово **def** для определения своей функции.

```

1 def average(lst):
2     return sum(lst) / len(lst)
3
4 a = [2, 3, 1, -4, 12, 6]
5 print(average(a)) # 3.3333333333333335

```

✎ В MATLAB блоки кода закрываются словом **end**, а в Python структура определяется **отступами**.

## → Анонимная функция (lambda):

Краткая форма, используется для простых операций.

```

1 square = lambda x: x ** 2
2 print(square(5)) # 25
3
4 nums = [1, 2, 3, 4]
5 squared = list(map(lambda x: x ** 2, nums))
6 print(squared) # [1, 4, 9, 16]

```

# Цикл for в Python

★ **Цикл for:** используется для перебора итерируемых объектов (списки, кортежи, строки, словари, множества)

## Перебор списка:

```

1  nums = [1, 2, 3, 4]
2  for num in nums:
3      print(num)
4  # Вывод:
5  # 1
6  # 2
7  # 3
8  # 4

```

## Перебор строки:

```

1  text = "abcd"
2  for char in text:
3      print(char)
4  # Вывод:
5  # a
6  # b
7  # c
8  # d

```

## Использование range:

```

1  for i in range(4):
2      print(i)
3  # Вывод:
4  # 0
5  # 1
6  # 2
7  # 3

```

## Перебор словаря:

```

1  dic = {'a': 1, 'b': 2, 'c': 3}
2  for key, value in dic.items():
3      print(f"{key}: {value}")
4  # Вывод:
5  # a: 1
6  # b: 2
7  # c: 3

```

# Цикл while в Python

- **Цикл while:** выполняется, пока условие остаётся True.  
Если условие становится False, цикл завершается.

## Счётный цикл:

```
1 count = 0
2 while count < 5:
3     print(count)
4     count += 1
5 # 0
6 # 1
7 # 2
8 # 3
9 # 4
```

## Условный цикл:

```
1 n = 10
2 while n > 0:
3     print(n)
4     n -= 2
5 # 10
6 # 8
7 # 6
8 # 4
9 # 2
```

# Управление циклами в Python

## ★ Операторы управления циклами

- `break` – досрочно прерывает цикл. Используется, когда нужно немедленно выйти из цикла.
- `continue` – перейти к следующей итерации. Используется, когда нужно пропустить часть кода, но цикл продолжить.

### `break`

```
1 for i in range(10):
2     if i == 5:
3         break
4     print(i)
5 # Вывод:
6 # 0
7 # 1
8 # 2
9 # 3
# 4
```

### `continue`

```
1 for i in range(5):
2     if i == 2:
3         continue
4     print(i)
5 # Вывод:
6 # 0
7 # 1
8 # 3
# 4
```

## Задание: Циклы и условия в Python

### ★ Задание 1. Построить таблицу умножения от 1 до 9:

```
1 for i in range(1, 10):
2     for j in range(1, 10):
3         print(f"{j} * {i} = {j*i}", end="\t")
4     print()
```

### ★ Задание 2. Дан список целых чисел. Найдите наибольшее произведение двух разных элементов списка:

```
1 nums = [1, 5, 3, 9, 2]
2
3 max_product = 0
4 for i in range(len(nums)):
5     for j in range(i + 1, len(nums)):
6         product = nums[i] * nums[j]
7         if product > max_product:
8             max_product = product
9
10 print("Максимальное произведение:", max_product)
```

# Содержание

- 1 Основы Python**
- 2 Библиотека Numpy**
- 3 Библиотека SciPy**
- 4 Библиотека Matplotlib**

## Работа со списками (list) в Python

- ★ Списки (list) в Python нельзя напрямую складывать с числами.

```
1 a = [1, 2, 3, 4]          # a + 1 ошибка!
2 [x + 1 for x in a]      # Результат: [2, 3, 4, 5]
```

- ★ Оператор `a + b` соединяет два списка, а не складывает их элементы поэлементно:

```
1 b = [2, 3, 4, 5]
2 a + b                  # Результат: [1, 2, 3, 4, 2, 3, 4, 5]
3 [x + y for (x, y) in zip(a, b)] # Результат: [3, 5, 7, 9]
```

### Выводы:

- ➡ Нельзя напрямую складывать список с числом;
- ➡ `list + list` – это объединение списков, а не поэлементное сложение;
- ➡ Для поэлементных операций нужно использовать `zip()` и генераторы списков.
- ➡ **NumPy** позволяет делать такие операции проще и быстрее по сравнению с обычными списками Python!

## Введение в библиотеку NumPy

- ★ NumPy — это популярная библиотека Python с открытым исходным кодом, созданная для удобных и быстрых вычислений.
- ★ Главное в NumPy — это тип данных `ndarray`, с помощью которого можно легко работать с многомерными массивами (например, векторами и матрицами).
- ★ NumPy работает быстрее, чем обычные списки Python, особенно при больших объёмах данных.
- ★ Благодаря векторным операциям можно писать меньше циклов и быстрее выполнять расчёты.
- ★ NumPy — основа для многих других библиотек, таких как `pandas` и `TensorFlow`.
- ★ Чтобы использовать NumPy, сначала нужно его импортировать:

```
1 import numpy as np
```

# Операции с массивами в NumPy

★ NumPy поддерживает поэлементные операции с массивами:

```
1 a = np.array([1, 2, 3, 4])
2 a + 1          # array([2, 3, 4, 5])
3 a * 2          # array([2, 4, 6, 8])
4
5 b = np.array([2, 3, 4, 5])
6 a + b          # array([3, 5, 7, 9])
7
8 a = np.array([[1, 2, 3], [4, 5, 6]])
9 b = np.array([1, 2, 3])
10 a + b         # array([[2, 4, 6], [5, 7, 9]])
```

## Основные особенности:

- NumPy автоматически распространяет скаляр на все элементы массива;
- Операции применяются к каждому элементу массива без циклов и списковых выражений;
- NumPy поддерживает гибкую систему **broadcasting** – распространения массива по другим массивам с совместимой формой.

## Инициализация массивов в NumPy

★ **Создание массива из списка:** (тип данных массива определяется по типу данных списка)

```
1 a = np.array([1, 2, 3, 4])           # array([1, 2, 3, 4])
2 a = np.array([1.0, 2.0, 3.0, 4.0])   # array([1., 2., 3., 4.])
```

★ **Создание массивов из нулей и единиц:** (по умолчанию используется тип float64, но можно указать другой)

```
1 np.zeros(4)                      # array([0., 0., 0., 0.])
2 np.ones(4)                       # array([1., 1., 1., 1.])
3 np.zeros(4, dtype="int32")        # array([0, 0, 0, 0])
4 np.ones(4, dtype="bool")          # array([True, True, True, True])
```

★ **np.fill():** Заполняет все элементы массива заданным значением.

```
1 a = np.array([1, 2, 3, 4])
2 a.fill(5)                      # array([5, 5, 5, 5])
3 a.fill(2.5)                    # array([2, 2, 2, 2])
4 a = a.astype("float")
5 a.fill(2.5)                    # array([2.5, 2.5, 2.5, 2.5])
```

## Создание массивов: arange и linspace

★ **np.arange()**: создание массива с заданным шагом

- start – начало (включается), по умолчанию 0
- stop – конец (не включается)
- step – шаг (может быть отрицательным)

```

1 np.arange(10)          # [0 1 2 3 4 5 6 7 8 9]
2 np.arange(1, 10)        # [1 2 3 4 5 6 7 8 9]
3 np.arange(1, 10, 2)     # [1 3 5 7 9]
4 np.arange(10, 1, -2)    # [10 8 6 4 2]
```

★ **np.linspace()**: создание массива с заданным числом точек

- start, stop – начало и конец (по умолчанию оба включаются)
- num – число точек (по умолчанию 50)
- retstep=True – возвращает шаг

```

1 np.linspace(1, 10, 10)                  # [1. 2. ... 10.]
2 np.linspace(1, 10, 10, dtype=int)        # [1 2 ... 10]
3 a, step = np.linspace(1, 10, 10, retstep=True) # step: 1.0
```

## Создание массивов: списковые выражения

★ **Списковые выражения (List Comprehension)** позволяют гибко создавать массивы NumPy по любому правилу.

### Пример 1: линейное приращение

```
1 a = np.array([1 + i for i in range(10)])
2 # [1 2 3 4 5 6 7 8 9 10]
```

### Пример 2: квадратичное приращение

```
1 a = np.array([(1 + i)**2 for i in range(10)])
2 # [1 4 9 16 25 36 49 64 81 100]
```

### Пример 3: двумерный массив через вложенные циклы

```
1 a = np.array([[i * j for j in range(4)] for i in range(3)])
2 # [[0 0 0 0]
3 # [0 1 2 3]
4 # [0 2 4 6]]
```

## Генерация случайных чисел

- ★ np.random.rand(n): равномерное распределение на [0, 1]
- ★ np.random.randn(n): нормальное распределение со средним 0 и отклонением 1
- ★ np.random.randint(low, high, size): случайные целые числа
- ★ np.random.normal(loc, scale, size): нормальное распределение с центром loc и отклонением scale
- ★ np.random.choice(a, size, replace=True): выбор случайных элементов из массива a

```
1 a = np.random.rand(10)          # [0.69, 0.20, ..., 0.41]
2 b = np.random.randn(10)        # [-0.86, -0.56, ..., -0.73]
3 c = np.random.randint(1, 20, 10) # [9, 11, 2, ..., 4]
4 d = np.random.normal(0, 1, 10)  # [0.1, -1.2, ..., 0.56]
5 a = np.array([1,2,3,4,5,6,7,8,9])
6 np.random.choice(a, 3)         # [3, 5, 1]
```

## Свойства массива NumPy

- ★ `a.dtype` — тип данных внутри массива
- ★ `a.shape` — форма массива (возвращает кортеж, указывающий размерность по каждой оси)
- ★ `a.size` — общее количество элементов в массиве
- ★ `a.ndim` — количество измерений (размерность массива)

```

1 a = np.array([1, 2, 3, 4, 5])
2
3 print("Тип:", type(a))
4     # <class 'numpy.ndarray'>
5 print("Тип данных:", a.dtype)
6     # int32
7 print("Форма:", a.shape)      # (5, )
8 print("Размер:", a.size)       # 5
9 print("Измерения:", a.ndim)   # 1

```

```

1 a = np.array([[1, 2, 3],[4, 5, 6]])
2
3 print("Тип:", type(a))
4     # <class 'numpy.ndarray'>
5 print("Тип данных:", a.dtype)
6     # int32
7 print("Форма:", a.shape)      # (2, 3)
8 print("Размер:", a.size)       # 6
9 print("Измерения:", a.ndim)   # 2

```

- `a.shape, a.size, a.ndim` — только для NumPy массивов
- `np.shape(a), np.size(a), np.ndim(a)` — применимы к массивам и другим структурам (например, спискам, кортежам)

# Индексация и срезы массивов NumPy

## Одномерный массив:

```

1 a = np.array([1,2,3,4,5,6])
2 a[1:3]    # array([2, 3])
3 a[1:-2]   # array([2, 3, 4])
4 a[-4:5]   # array([3, 4, 5])
5 a[-2:]    # array([5, 6])
6 a[::-2]   # array([1, 3, 5])
7 a[0] = 10
8 a         # array([10,2,3,4,5,6])

```

## Двумерный массив:

```

1 a = np.array([[0,1,2,3],
2                  [10,11,12,13]])
3
4 a[1,3]      # 13
5 a[1]        # array([10,11,12,13])
6 a[:,1]      # array([1,11])
7 a[1,3] = -1
8 a          # [[0,1,2,3],
9                  # [10,11,12,-1]]
10 a[1,2:]    # array([12, -1])
11 a[1,-1]    # -1

```

1	2	3	4	5	6
[0]	[1]	[2]	[3]	[4]	[5]
[-6]	[-5]	[-4]	[-3]	[-2]	[-1]

	[0]	[1]	[2]	[3]
[0]	0	1	2	3
[1]	10	11	12	13

# Выборка элементов и ссылки

## Использование маски (mask):

```

1 a = np.arange(0, 100, 20)
2 mask = np.array([0,2,0,0,1], dtype=bool)
3 a[mask] # array([20, 80])

```

## Советы по срезам в NumPy:

- NumPy использует ссылки при срезе — изменения в подмассиве влияют на исходный.
- Список Python копируется — оригинал не изменяется.
- Используйте `copy()` для создания независимого массива.

```

1 a = np.array([0,1,2,3,4])
2 b = a[2:4]
3 b[0] = 10
4 a # array([0,1,10,3,4])
5
6 a = [1,2,3,4,5]
7 b = a[2:4]
8 b[0] = 10
9 a # [1,2,3,4,5]
10
11 a = np.array([0,1,2,3,4])
12 b = a[2:4].copy()
13 b[0] = 10
14 a # array([0,1,2,3,4])

```

# Соединение и преобразование массивов

## Соединение массивов:

```

1  x = np.array([[0,1,2],
2                  [10,11,12]])
3  y = np.array([[50,51,52],
4                  [60,61,62]])
5
6  z = np.concatenate((x, y))
7  # along axis 0
8  z = np.concatenate((x, y), axis=1)
9  # along axis 1
10 z = np.array((x, y))
11 # along axis 2
12
13 np.vstack((x, y))    # along axis 0
14 np.hstack((x, y))    # along axis 1

```

## Преобразование формы:

```

1  a = np.arange(6)
2  a.shape = (2, 3)
3  # array([[0,1,2], [3,4,5]])
4
5  a = np.arange(6)
6  b = a.reshape((2, 3))
7  # b: array([[0,1,2], [3,4,5]])
8
9  b.T
10 # array([[0,3],[1,4],[2,5]])

```

## Сортировка значений:

```

1  data = np.array([3, 1, 4, 1, 5])
2  sorted_data = np.sort(data)
3  # [1, 1, 3, 4, 5]

```

## Основные функции NumPy

Функция	Только для NumPy	Общая форма
Среднее	a.mean()	np.mean(a)
Максимум	a.max()	np.max(a)
Минимум	a.min()	np.min(a)
Индекс макс.	a.argmax()	np.argmax(a)
Индекс мин.	a.argmin()	np.argmin(a)
Сумма	a.sum()	np.sum(a)
Накопл. сумма	a.cumsum()	np.cumsum(a)
Стандарт. отклон.	a.std()	np.std(a)
Произведение	a.prod()	np.prod(a)
Накопл. произвед.	a.cumprod()	np.cumprod(a)

**Примечание:** Методы a.\*() применимы только к массивам NumPy, а функции np.\*(a) — к любым совместимым объектам (например, спискам).

## Скалярное произведение массивов (dot product)

**Операция `np.dot(A, B)` вычисляет скалярное произведение двух массивов.**

$$\text{np.dot}(\begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\\hline\end{array}, \begin{array}{|c|c|c|}\hline 1 & 100 & 10000 \\\hline\end{array}) = \boxed{30201}$$

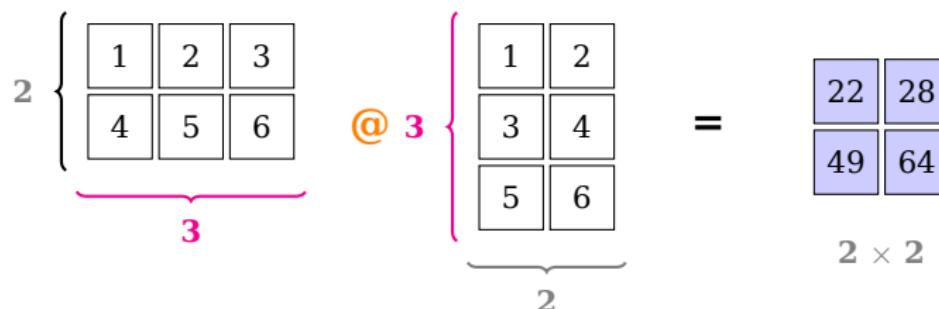
### Пример кода:

```
1 import numpy as np
2
3 A = np.array([1, 2, 3])
4 B = np.array([1, 100, 10000])
5
6 print(np.dot(A, B))
7 # 30201
```

### Пояснение:

- ➡ Скалярное произведение — это сумма произведений соответствующих элементов:
- ➡  $1 \cdot 1 + 2 \cdot 100 + 3 \cdot 10000 = 30201$
- ➡ Результат — одно число (скаляр).

## Матричное произведение массивов

**Операция np.matmul(A, B) или сокращённая форма A @ B****Пример кода:**

```

1 import numpy as np
2 data1 = np.array([[1, 2, 3],
3                  [4, 5, 6]])
4 data2 = np.array([[1, 2],
5                  [3, 4],
6                  [5, 6]])
7
8 print(np.matmul(data1, data2))
# [[22 28]
9 #   [49 64]]
10

```

**Пояснение:**

- Размерности:  
 $2 \times 3 \cdot 3 \times 2 \Rightarrow 2 \times 2$
- Каждый элемент результата — это скалярное произведение строки из `data1` и столбца из `data2`:

$$1 \cdot 1 + 2 \cdot 3 + 3 \cdot 5 = 22$$

# Линейная алгебра с Numpy (Модуль numpy.linalg)

## ★ Определитель матрицы: np.linalg.det()

**Пример:**

```

1 import numpy as np
2
3 A = np.array([[2.0, -1.0, 0.0],
4               [-1.0, 2.0, -1.0],
5               [0.0, -1.0, 2.0]])
6
7 print(np.linalg.det(A))
8 # 4.0

```

**Пояснение:**

- ➡ Определитель (determinant) – это скалярная величина, характеризующая матрицу.
- ➡ Если  $\det(A) \neq 0$ , то существует обратная матрица  $A^{-1}$ .

## ★ Обратная матрица: np.linalg.inv()

**Пример:**

```

1 A_inverse = np.linalg.inv(A)
2 print(A_inverse)
3 # [[0.75 0.5 0.25]
4 #  [0.5 1.0 0.5 ]
5 #  [0.25 0.5 0.75]]

```

**Пояснение:**

- ➡ Функция `inv()` возвращает  $A^{-1}$ , такую, что  $A \cdot A^{-1} = I$ .
- ➡ Работает только для квадратных невырожденных матриц.

# Собственные значения и линейные уравнения

## ★ Функция `np.linalg.eig()`: нахождение собственных значений и векторов

**Пример:**

```

1 import numpy as np
2
3 A = np.array([[2.0, -1.0, 0.0],
4               [-1.0, 2.0, -1.0],
5               [0.0, -1.0, 2.0]])
6
7 lam, v = np.linalg.eig(A)
8 print(lam)
9 print(v)
```

**Результат:**

$$\begin{aligned} \text{lam} &= [3.41 \quad 2.00 \quad 0.59] \\ \text{v} &= \begin{bmatrix} -0.5 & -0.71 & 0.5 \\ 0.71 & 0.00 & 0.71 \\ -0.5 & 0.71 & 0.5 \end{bmatrix} \end{aligned}$$

## ★ Функция `np.linalg.solve(A, b)`: решение $Ax = b$

**Пример:**

```

1 b = np.array([2.0, -1.0, 3.0])
2 x = np.linalg.solve(A, b)
3 print(x)
```

**Результат:**

$$x = [1.75 \quad 1.5 \quad 2.25]$$

# Содержание

- 1 Основы Python
- 2 Библиотека Numpy
- 3 Библиотека SciPy
- 4 Библиотека Matplotlib

## Введение в библиотеку SciPy

**SciPy** – это одна из ключевых библиотек на Python для научных и инженерных расчётов. Она основана на NumPy и предоставляет расширенные инструменты для:

- ★ решения задач оптимизации
- ★ вычисления интегралов и выполнения интерполяции
- ★ работы со специальными математическими функциями
- ★ быстрого преобразования Фурье (FFT)
- ★ обработки сигналов и изображений
- ★ численного решения дифференциальных уравнений

SciPy широко используется в физике, инженерии, биоинформатике, астрономии, климатологии и других научных областях.

## Физические константы: `scipy.constants`

Модуль `constants` из библиотеки **SciPy** предоставляет множество часто используемых физических констант, которые можно вызывать напрямую:

- $\pi$  —число пи: `constants.pi`
- $c$  —скорость света (м/с): `constants.c`
- $h$  —постоянная Планка ( $\text{Дж} \cdot \text{с}$ ): `constants.h`
- $k_B$  —постоянная Больцмана ( $\text{Дж}/\text{К}$ ): `constants.Boltzmann`
- $N_A$  —число Авогадро: `constants.Avogadro`
- $G$  —гравитационная постоянная: `constants.G`

### Пример кода:

```
1 from scipy import constants
2
3 print(constants.c)          # 299792458.0
4 print(constants.Boltzmann)   # 1.380649e-23
5 print(constants.Avogadro)    # 6.02214076e+23
6 print(constants.h)           # 6.62607015e-34
```

# Нахождение корней нелинейного уравнения

Функция `scipy.optimize.root` используется для поиска корней уравнений.

Например, для уравнения:

$$x + \cos(x) = 0$$

**Синтаксис:** `root(fun, x0)`, где:

- `fun` —функция уравнения
- `x0` —начальное приближение

**Пример кода:**

```
1 from scipy.optimize import root
2 import numpy as np
3
4 def eqn(x):
5     return x + np.cos(x)
6
7 myroot = root(eqn, 0)
8
9 print(myroot)
```

**Результат:**

- Корень: -0.73908513
- Статус: `success = True`
- Сообщение: `The solution converged.`
- Кол-во итераций: `nfev = 9`

## Одномерная интерполяция: `scipy.interpolate.interp1d`

Модуль `scipy.interpolate` предоставляет мощные инструменты для выполнения интерполяции.

Для одномерной интерполяции используется функция `interp1d()`, которая принимает массивы значений аргумента (`xs`) и функции (`ys`) и возвращает вызываемую функцию для вычисления промежуточных значений  $y = f(x)$ .

### Пример кода:

```
1 from scipy.interpolate import interp1d
2 import numpy as np
3
4 xs = np.arange(10)
5 ys = 2 * xs + 1
6
7 interp_func = interp1d(xs, ys)
8
9 newarr = interp_func(np.arange(2.1, 3, 0.1))
10
11 print(newarr)
```

### Результат:

```
[5.2 5.4 5.6 5.8 6.0 6.2 6.4 6.6 6.8]
```

# Содержание

- 1 Основы Python
- 2 Библиотека Numpy
- 3 Библиотека SciPy
- 4 Библиотека Matplotlib

## Визуализация данных с помощью matplotlib

Matplotlib – это базовая библиотека для построения графиков в Python.

### Возможности:

- Построение линейных, точечных, гистограмм и 3D-графиков
- Настройка подписей осей, легенд, цвета, толщины и шрифтов
- Поддержка интерактивного режима и сохранения в PDF/PNG
- Интеграция с NumPy, Pandas, SciPy

### Основной модуль: `matplotlib.pyplot`

### Импорт:

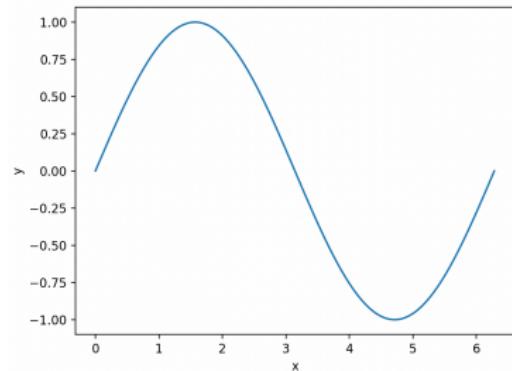
```
1 import matplotlib.pyplot as plt
```

# Пример построения линейного графика

## Код:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.linspace(0, 2 * np.pi, 100)
5 y = np.sin(x)
6
7 plt.plot(x, y)
8 plt.xlabel('x')
9 plt.ylabel('y')
10 plt.show()
```

## График:



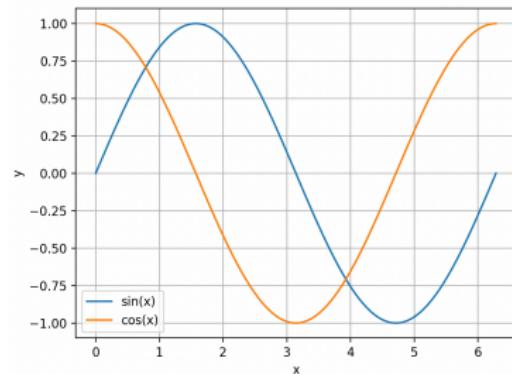
Построен график функции  $\sin(x)$  на интервале от 0 до  $2\pi$ .

# Построение нескольких графиков с легендой

## Код:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.linspace(0, 2 * np.pi, 100)
5 y1 = np.sin(x)
6 y2 = np.cos(x)
7
8 plt.plot(x, y1, label='sin(x)')
9 plt.plot(x, y2, label='cos(x)')
10
11 plt.xlabel('x')
12 plt.ylabel('y')
13 plt.legend()
14 plt.grid()
15 plt.show()
```

## График:



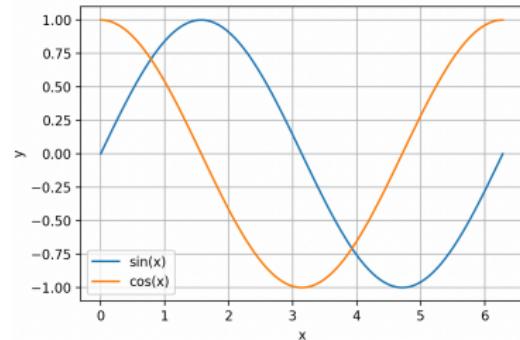
Построены функции синуса и косинуса. Добавлены легенда и сетка для удобства восприятия.

# Добавление настройки размера изображения

## Код:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.linspace(0, 2 * np.pi, 100)
5 y1 = np.sin(x)
6 y2 = np.cos(x)
7
8 plt.figure(figsize=(6, 4), dpi=100,
9             facecolor='white')
10
11 plt.plot(x, y1, label='sin(x)')
12 plt.plot(x, y2, label='cos(x)')
13
14 plt.xlabel('x')
15 plt.ylabel('y')
16 plt.legend()
17 plt.grid()
18 plt.show()
```

## График:



Используется `plt.figure(...)` для задания размеров и фона изображения.

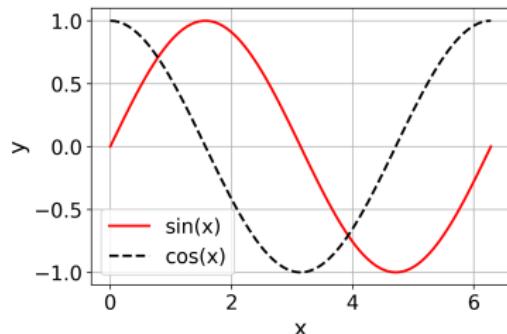
# Построение нескольких графиков с легендой

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.linspace(0, 2 * np.pi, 100)
5 y1 = np.sin(x)
6 y2 = np.cos(x)
7
8 plt.figure(figsize=(6, 4), dpi=100,
9             facecolor='white')
10
11 plt.plot(x, y1, label='sin(x)',
12           color='red', linestyle='--',
13           linewidth=2)
14 plt.plot(x, y2, label='cos(x)',
15           color='black', linestyle='--',
16           linewidth=2)
17
18 plt.xlabel('x', fontsize=18)
19 plt.ylabel('y', fontsize=18)
20 plt.tick_params(labelsize=16)
21 plt.legend(fontsize=16)
22 plt.grid()
23
24 plt.savefig("plot.png", bbox_inches='tight', dpi=300)
25 plt.show()

```

## График:



Цветовая дифференциация и стили линий подчёркивают различие между  $\sin(x)$  и  $\cos(x)$ . Шрифты и толщина линий увеличены для читаемости. График сохранён в PNG с высоким разрешением.

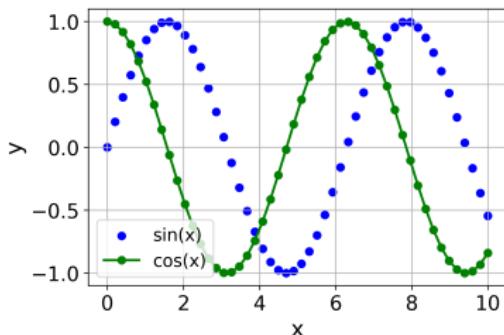
# Scatter-график и линия с маркерами

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.linspace(0, 10, 50)
5 y1 = np.sin(x)
6 y2 = np.cos(x)
7
8 plt.figure(figsize=(6, 4), dpi=100)
9
10 plt.scatter(x, y1, color='blue', label
11             ='sin(x)')
12 plt.plot(x, y2, color='green', marker=
13           'o',
14           linestyle='--', linewidth=2,
15           label='cos(x)')
16
17 plt.xlabel('x', fontsize=18)
18 plt.ylabel('y', fontsize=18)
19 plt.tick_params(labelsize=16)
20 plt.legend(fontsize=14)
21 plt.grid()
22
23 plt.savefig("scatter.png", bbox_inches
24             ='tight', dpi=300)
25 plt.show()

```

## График:



$\sin(x)$  изображён синими точками (scatter-график),  $\cos(x)$  — зелёной линией с круглыми маркерами.

Использована легенда и сетка.

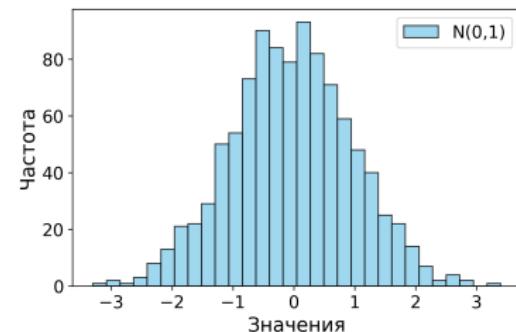
# Построение гистограммы (Histogram)

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 data = np.random.normal(loc=0, scale
5 =1, size=1000)
6
7 plt.figure(figsize=(6, 4), dpi=100)
8
9 plt.hist(data, bins=30,
10 color='skyblue',
11 edgecolor='black',
12 alpha=0.8, label='N(0,1)')
13
14 plt.xlabel('Значения', fontsize=16)
15 plt.ylabel('Частота', fontsize=16)
16 plt.tick_params(labelsize=14)
17 plt.legend(fontsize=14)
18
19 plt.tight_layout()
20 plt.savefig("histogram.png",
21             bbox_inches='tight', dpi=300)
22 plt.show()

```

## График:



Сгенерированы 1000 случайных чисел из нормального распределения  $N(0, 1)$ . Построена гистограмма с 30 интервалами и оформленiem для лучшей читаемости.

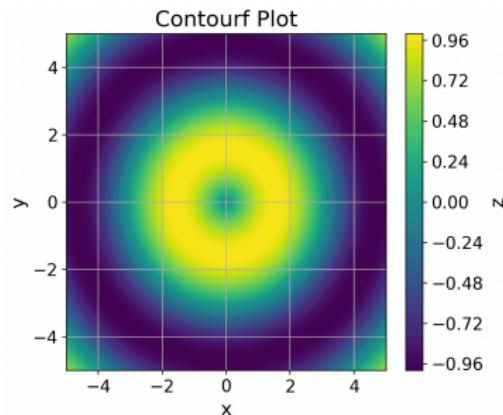
# Построение двумерной карты (contourf)

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 x = np.linspace(-5, 5, 100)
5 y = np.linspace(-5, 5, 100)
6 X, Y = np.meshgrid(x, y)
7
8 Z = np.sin(np.sqrt(X**2 + Y**2))
9
10 plt.figure(figsize=(6, 5))
11 contour = plt.contourf(X, Y, Z, levels
12                         =50, cmap='viridis')
13
14 plt.xlabel('x', fontsize=16)
15 plt.ylabel('y', fontsize=16)
16 plt.title('Contourf Plot', fontsize
17             =18)
18 plt.tick_params(labelsize=14)
19 cbar = plt.colorbar(contour)
20 cbar.set_label('z', fontsize=16)
21 cbar.ax.tick_params(labelsize=14)
22
23 plt.grid(True)
24 plt.tight_layout()
25 plt.show()

```

## График:



Двумерная цветная карта функции

$$z = \sin(\sqrt{x^2 + y^2})$$

Цветовая шкала показывает значения переменной  $z$ . Оси и цветовая шкала снабжены подписями увеличенного шрифта.

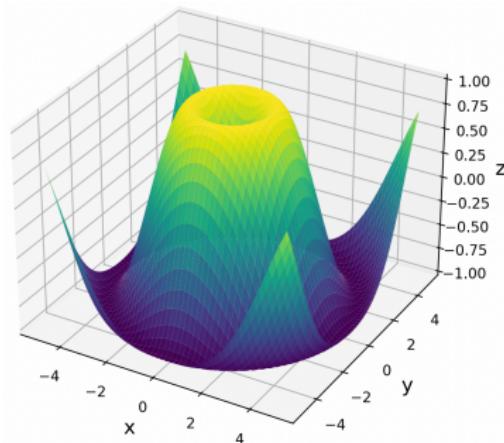
# Построение трёхмерной поверхности (plot\_surface)

```

1 import matplotlib.pyplot as plt
2 from mpl_toolkits.mplot3d import
3 Axes3D
4 import numpy as np
5
6 x = np.linspace(-5, 5, 100)
7 y = np.linspace(-5, 5, 100)
8 X, Y = np.meshgrid(x, y)
9 Z = np.sin(np.sqrt(X**2 + Y**2))
10
11 fig = plt.figure(figsize=(7, 5))
12 ax = fig.add_subplot(111, projection='
13 3d')
14
15 surf = ax.plot_surface(X, Y, Z, cmap='
16 viridis')
17 ax.set_xlabel('x', fontsize=14)
18 ax.set_ylabel('y', fontsize=14)
19 ax.set_zlabel('z', fontsize=14)
20
21 plt.tight_layout()
22 plt.show()

```

**График:**



Объёмная поверхность  
функции

$$z = \sin(\sqrt{x^2 + y^2})$$