

Урок 6.

Математические функции, хранение чисел. Модули

Встроенные математические функции	1
Задание для закрепления 1	7
Модули	9
Оператор import	10
Стандартная библиотека Python	13
Модуль math	14
Задания для закрепления 2	17
Биты и байты	18
Хранение чисел в памяти	19
Точность операций с вещественными числами	20
Задания для закрепления 3	23
Модуль Decimal	24
Задания для закрепления 4	25
Ответы на задания	26
Практические задания	27

Встроенные математические функции

Python предоставляет ряд встроенных математических функций, которые упрощают выполнение базовых вычислений и работы с числами. Эти функции не требуют дополнительного импорта и доступны по умолчанию. Вот некоторые из наиболее часто используемых встроенных математических функций:

Функция	Описание	Пример кода	Результат
<code>abs()</code>	Возвращает абсолютное значение числа	<code>abs(-7)</code>	7
<code>max()</code>	Возвращает максимальное значение	<code>max(1, 5, 3)</code>	5
<code>min()</code>	Возвращает минимальное значение	<code>min(2, 8, 4, 1)</code>	1
<code>pow()</code>	Возводит число в степень	<code>pow(2, 3)</code>	8
<code>sum()</code>	Возвращает сумму элементов	<code>sum([1, 2, 3, 4])</code>	10
<code>round()</code>	Округляет число до ближайшего целого или заданного количества знаков	<code>round(4.567, 2)</code>	4.57

`abs()`

Функция `abs()` возвращает абсолютное значение числа (модуль), то есть его значение без знака.



Пример

Python

```
print(abs(-7)) # Результат: 7
```

```
print(abs(3.5)) # Результат: 3.5
```

max()

Функция `max()` возвращает наибольшее из переданных чисел или элементов коллекции.



Пример

Python

```
a = 1
```

```
b = 5
```

```
c = 3
```

```
print(max(a, b, c))           # Передаем несколько значений
```

```
print(max([2, 8, 4, 1]))     # Передаем в виде коллекции
```

min()

Функция `min()` возвращает наименьшее из переданных чисел или элементов списка.



Пример

Python

```
print(min(1, 5, 3))          # Передаем несколько значений
```

```
print(min([2, 8, 4, 1]))     # Передаем в виде коллекции
```

pow()

Функция `pow()` принимает два аргумента: число и степень, в которую нужно возвести число.



Пример

Python

```
print(pow(2, 3)) # Передаем сначала число, а после степень  
  
print(pow(5, 2))
```

sum()

Функция `sum()` возвращает сумму всех элементов коллекции.



Пример

Python

```
numbers = [1, 2, 3, 4]  
print(sum(numbers)) # Передаем только в виде коллекции (элементы в  
квадратных скобках)
```

round()

Функция `round()` округляет число до ближайшего целого или до указанного количества знаков после запятой.



Пример

Python

```
print(round(4.56)) # Округление до целого  
  
print(round(4.567, 2)) # Округление до указанного количества знаков)
```

Банковское округление

По умолчанию, `round()` округляет число до ближайшего целого. Если десятичная часть числа меньше 0.5, округление идёт в меньшую сторону, если больше 0.5 — в большую.



Пример

Python

```
print(round(3.4)) # Округление в меньшую сторону
```

```
print(round(3.6)) # Округление в большую сторону
```

Для случаев, когда десятичная часть точно равна 0.5 Python использует банковское округление. Это правило заключается в том, что числа округляются в ближайшее чётное целое. Это позволяет уменьшить накопление ошибок в финансовых расчётах.



Пример

Python

```
print(round(1.5)) # Результат: 2 (округление к ближайшему чётному)
```

```
print(round(2.5)) # Результат: 2 (округление к ближайшему чётному)
```

```
print(round(3.5)) # Результат: 4 (округление к ближайшему чётному)
```

```
print(round(4.5)) # Результат: 4 (округление к ближайшему чётному)
```

Функция `round()` работает аналогично с отрицательными числами. Правила те же: числа округляются к ближайшему целому или к ближайшему чётному числу в случае 0.5.



Пример

Python

```
print(round(-1.5)) # Результат: -2 (округление к ближайшему чётному)
```

```
print(round(-2.5)) # Результат: -2 (округление к ближайшему чётному)
```

Задание для закрепления 1

1. Какой результат выведет следующий код?

```
Python  
print(abs(-10))
```

- a. -10
- b. 10
- c. Ошибка
- d. Программа ничего не выведет

[Посмотреть ответ](#)

2. Какой результат выведет следующий код?

```
Python  
print(sum(3, 9, 5))
```

- a. 3
- b. 17
- c. 9
- d. Ошибка

[Посмотреть ответ](#)

3. Какой результат выведет следующий код?

```
Python  
print(round(4.678, 2))
```

- a. 4.67
- b. 4.68
- c. 4.7
- d. Ошибка

[Посмотреть ответ](#)

4. Какой результат выведет следующий код?

```
Python  
print(round(2.5))
```

- a. 3
- b. 4
- c. 2
- d. Ошибка

[Посмотреть ответ](#)

Модули



Модуль — это файл с расширением `.py`, который содержит код на языке Python: функции, классы, переменные, и даже исполняемые инструкции.

Модули позволяют организовать код, разделяя его на логически независимые части, и повторно использовать его в разных программах.

Зачем нужны модули?

- **Организация кода:** Модули помогают разбить большие программы на более мелкие, логически связанные части, что упрощает поддержку и понимание кода.
- **Повторное использование:** Модуль можно импортировать в несколько разных программ, не переписывая код каждый раз заново.
- **Изоляция кода:** Модули помогают изолировать код, что позволяет избежать конфликтов имён функций, классов и переменных.

Использование модулей

Модули в Python можно импортировать в программу, используя ключевое слово `import`. Это позволяет использовать необходимый функционал в других частях программы.

Оператор import



import — это ключевое слово в Python, которое используется для загрузки модулей (или пакетов) в программу.

С помощью `import` вы можете использовать код, функции и классы, написанные в других модулях, без необходимости переписывать их заново.

Способы использования import

Способ использования	Пример кода	Описание
Импорт всего модуля	<code>import math</code>	Импортирует весь модуль
Импорт конкретных функций	<code>from math import sqrt, pi</code>	Импортирует только конкретные функции или переменные из модуля
Импорт с псевдонимом	<code>import math as m</code>	Импортирует весь модуль с указанием псевдонима
Импорт всего содержимого	<code>from math import *</code>	Импортирует всё содержимое модуля

1. Импорт всего модуля:

Самый простой способ использования оператора `import` — это импортировать весь модуль.



Пример

Python

```
import math
```

```
print(math.sqrt(16)) # Используем функцию sqrt() из модуля math
```

Для использования функций нужно обращаться к ним через имя модуля, например, `math.sqrt()`

2. Импорт конкретных функций или классов:

Можно импортировать конкретные функции или классы из модуля с помощью ключевого слова `from`.



Пример

Python

```
from math import sqrt, pi

print(sqrt(16)) # Используем функцию sqrt() без указания имени модуля

print(pi)      # Используем константу pi
```

Нужно использовать их напрямую, например, `sqrt()` или `pi`.

3. Импорт модуля с псевдонимом:

Если название модуля длинное или если вы хотите использовать более короткое имя, можно задать псевдоним для модуля с помощью ключевого слова `as`.



Пример

Python

```
import math as m
```

```
print(m.sqrt(16)) # Используем функцию sqrt() из модуля math, но с псевдонимом "m"
```

Для использования функций нужно обращаться к модулю через псевдоним, например, `m.sqrt()`

4. Импорт всего содержимого модуля:

Можно импортировать всё содержимое модуля в текущую область видимости с помощью оператора `*`. Однако этот метод не рекомендуется, так как может привести к конфликтам имён и затруднить чтение и сопровождение кода.



Пример

Python

```
from math import *

print(sqrt(16)) # Используем функцию sqrt() напрямую

print(pi)      # Используем константу pi напрямую
```

Функции и переменные можно использовать напрямую без указания имени модуля (не рекомендуется).

Стандартная библиотека Python

Стандартная библиотека Python - это набор модулей и пакетов, которые включены в Python по умолчанию и предоставляют широкий спектр функциональности для выполнения различных задач. Она позволяет разработчикам использовать уже готовые решения без необходимости установки дополнительных пакетов.

Примеры модулей из стандартной библиотеки:

1. **os** — взаимодействие с операционной системой.
2. **sys** — доступ к переменным и функциям, связанным с интерпретатором Python.
3. **math** — математические функции.
4. **datetime** — работа с датами и временем.
5. **json** — работа с JSON-форматом.
6. **re** — работа с регулярными выражениями.
7. **random** — генерация случайных чисел.

Модуль math



Модуль `math` — это стандартный модуль, предоставляющий функции для выполнения различных математических операций, таких как вычисление тригонометрических функций, логарифмов, возведение в степень, и многое другое.

Модуль `math` очень полезен при выполнении базовых математических расчётов и поддерживает как целые числа, так и числа с плавающей запятой.

Как использовать модуль `math`?

Чтобы использовать функции из модуля `math`, необходимо сначала импортировать его в свой код:

```
Python
import math
```

Основные функции модуля `math`

Функция/ переменная	Описание
<code>math.sqrt(x)</code>	Возвращает квадратный корень числа <code>x</code>
<code>math.pow(x, y)</code>	Возводит число <code>x</code> в степень <code>y</code>
<code>math.factorial(x)</code>	Возвращает факториал числа <code>x</code> (произведение всех чисел от 1 до <code>x</code>)
<code>math.ceil(x)</code>	Округляет число <code>x</code> в большую сторону (до ближайшего целого числа)
<code>math.floor(x)</code>	Округляет число <code>x</code> в меньшую сторону (до ближайшего целого числа)
<code>math.pi</code>	Константа, представляющая число Пи ($\pi \approx 3.14159$)
<code>math.e</code>	Константа, представляющая основание натурального логарифма ($e \approx 2.71828$)

<code>math.sin(x)</code>	Возвращает синус угла x (где x в радианах)
<code>math.cos(x)</code>	Возвращает косинус угла x (где x в радианах)
<code>math.tan(x)</code>	Возвращает тангенс угла x (где x в радианах)
<code>math.degrees(x)</code>	Преобразует угол из радианов в градусы
<code>math.radians(x)</code>	Преобразует угол из градусов в радианы



Примеры использования модуля `math`

Вычисление квадратного корня

```
Python
import math

x = 16

result = math.sqrt(x)

print("Квадратный корень из", x, "равен", result)
```

Округление числа вверх и вниз

```
Python
import math

print(math.ceil(4.3)) # Округление вверх: 5

print(math.floor(4.8)) # Округление вниз: 4
```

Вычисление факториала

```
Python
import math

n = 5
factorial_result = math.factorial(n)
print("Факториал числа", n, ":", factorial_result)
```

Константы в модуле math

- `math.pi`: Число Пи ($\pi \approx 3.14159$)
- `math.e`: Основание натурального логарифма ($e \approx 2.71828$)

Эти константы часто используются в вычислениях, связанных с кругами, углами и логарифмами.

Задания для закрепления 2

1. Какой результат выведет следующий код?

```
Python
import math

print(math.sqrt(25))
```

- a. 25
- b. 5
- c. Ошибка
- d. Программа ничего не выведет

[Посмотреть ответ](#)

2. Какой результат выведет следующий код?

```
Python
import math

print(pi)
```

- a. 3.14
- b. 3.141592653589793
- c. Ошибка
- d. Программа ничего не выведет

[Посмотреть ответ](#)

Биты и байты

Компьютеры работают с данными в виде битов и байтов. Эти единицы являются основой для представления информации в памяти.

- **Бит** — это минимальная единица информации, которая может иметь одно из двух значений: 0 или 1. Биты представляют данные в двоичной системе счисления, которая лежит в основе всех вычислительных процессов.
- **Байт** — это набор из 8 битов. Он используется как базовая единица для хранения данных в памяти компьютера. Например, 1 байт может хранить одно целое число в пределах от 0 до 255 (или от -128 до 127 в случае знаковых чисел).



Пример

- 1 бит — это 0 или 1.
- 1 байт — это 8 битов, например: 01001101.

Хранение чисел в памяти

Числа хранятся в памяти в двоичной системе счисления.

Целые числа (int):

- 32-битные целые числа:
 - Занимают 4 байта памяти (32 бита).
 - Могут хранить значения от -2^{31} до $2^{31} - 1$ (от -2 147 483 648 до 2 147 483 647).
- 64-битные целые числа:
 - Занимают 8 байт памяти (64 бита).
 - Могут хранить значения от -2^{63} до $2^{63} - 1$ (от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807).



Python — это высокоуровневый язык программирования, и одна из его особенностей заключается в том, что целые числа в Python не имеют фиксированного размера.

Это означает, что Python автоматически выбирает, сколько памяти нужно для хранения числа.

Вещественные числа (float):

- В Python вещественные числа обычно представлены в формате 64-битного числа с плавающей запятой.
 - Занимают 8 байт памяти (64 бита).
 - В таком формате числа хранятся как комбинация мантииссы и экспоненты, что позволяет представлять как очень большие, так и очень маленькие числа.
- Пример диапазона для 64-битных чисел с плавающей запятой:
 - Примерно от $\pm 10^{-308}$ до $\pm 10^{308}$.

Пример представления целого числа:

Число 42 в двоичной системе записывается как:

00101010 (в 8-битном формате)

Точность операций с вещественными числами



Числа с плавающей запятой (вещественные числа) в Python хранятся по стандарту IEEE 754, который используется большинством современных систем.

Чтобы понять, как это работает, давайте разберёмся, что такое мантисса и экспонента и почему точность чисел ограничена.

В рамках этого стандарта вещественные числа хранятся в следующем формате:

- **1 бит** — знак числа (положительное или отрицательное).
- **11 битов** — экспонента, которая указывает на степень числа.
- **52 бита** — мантисса, представляющая саму дробную часть числа.

Этот формат позволяет представлять как очень маленькие, так и очень большие числа, но из-за ограниченного количества битов возникают погрешности при работе с дробными числами.



Пример

$$0.0001011 = 1.011 * 10^{-4}$$

Здесь:

- Мантисса: 1.011 (сохраняет значимые цифры числа, избавившись от нулей)
- Экспонента: -4 (указывает на степень 10, то есть количество потерянных нулей)

Ограниченная точность

Компьютер может хранить только определённое количество цифр в мантиссе. В Python это обычно 52 бита для мантиссы (в 64-битной системе).

Погрешность в вычислениях

Когда мы складываем такие числа, могут возникнуть неточности.



Пример

Python

```
print(0.1 + 0.2) # Ожидаем 0.3
```

Результат:

0.30000000000000004

Почему возникают такие ошибки?

Компьютеры используют двоичную систему, а многие десятичные дроби, такие как 0.1 или 0.2, не могут быть точно представлены в двоичной системе. В результате, когда компьютер пытается сохранить их, он округляет число до ближайшего возможного значения, которое может быть сохранено в памяти. Это приводит к небольшим погрешностям, которые накапливаются в вычислениях.



Пример: представление числа в двоичной системе

Число 0.1 в двоичной системе выглядит так:

0.000110011001100110011... (бесконечная последовательность)

Компьютер вынужден обрезать эту последовательность, потому что у него есть ограниченное количество битов для хранения числа. В итоге, точное значение не может быть сохранено, и при вычислениях возникают ошибки.

Сравнение вещественных чисел:

1. **Операции сравнения:** Из-за этих погрешностей сравнение вещественных чисел может работать не так, как ожидается. Например, сравнение $0.1 + 0.2 == 0.3$ вернет False из-за малой ошибки в представлении.



Пример

Python

```
print(0.1 + 0.2 == 0.3) # Вывод: False
```

- 2. Округление результата:** Для устранения небольших ошибок при выводе или сравнении результатов можно использовать функцию `round()`, которая округляет вещественные числа до указанного количества знаков после запятой.



Пример

Python

```
a = 0.1 + 0.2

print(round(a, 2))          # Вывод: 0.3

print(round(a, 2) == 0.3)  # Вывод: True
```

- 3. Использование модуля `decimal`:** Если требуется высокая точность при работе с вещественными числами, в Python можно использовать модуль `decimal`, который позволяет задавать точность операций и избегать ошибок округления.



Задания для закрепления 3

1. Сколько битов в одном байте?

- a. 4
- b. 8
- c. 16
- d. 32

[Посмотреть ответ](#)

2. Какое значение может хранить 1 байт?

- a. от 0 до 255
- b. от -255 до 255
- c. от -128 до 127
- d. от 0 до 1024

[Посмотреть ответ](#)

Модуль Decimal



Модуль `decimal` позволяет представлять числа с плавающей запятой в десятичной системе без потери точности.

Он сохраняет точные значения дробных чисел и предотвращает ошибки округления, что делает его полезным в таких областях, как финансы, наука, бухгалтерия, где каждая единица важна.



Пример

Python

```
# Импортируем класс Decimal из модуля decimal

from decimal import Decimal

# Создаём два объекта класса Decimal с помощью строк '0.1' и '0.2'

# Это гарантирует, что числа будут точно представлены в десятичном формате

a = Decimal('0.1')

b = Decimal('0.2')

# Сложение без ошибки округления

c = a + b

print(c) # Вывод: 0.3
```

Задания для закрепления 4

Какой результат выведет следующий код?

```
Python
from decimal import Decimal

a = Decimal('1.1')

b = Decimal('2.2')

print(a + b)
```

- a. 3.300000000000000003
- b. 3.3
- c. Ошибка
- d. Программа ничего не выведет

[Посмотреть ответ](#)



Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
1. Результат выполнения кода	Ответ: b
2. Результат выполнения кода	Ответ: d
3. Результат выполнения кода	Ответ: b
4. Результат выполнения кода	Ответ: c
Задания на закрепление 2	Вернуться к заданиям
1. Результат выполнения кода	Ответ: b
2. Результат выполнения кода	Ответ: c
Задания на закрепление 3	Вернуться к заданиям
1. Количество бит	Ответ: b
2. Значения в байте	Ответ: a, c
Задания на закрепление 4	Вернуться к заданиям
Результат выполнения кода	Ответ: b

Практические задания

1. Скидки для оптовых покупателей

Напишите программу, которая получает от пользователя цену на товар и количество товара. Программа должна рассчитать итоговую сумму заказа со скидкой, исходя из количества товара. Цену нужно округлить до сотых.

Условия скидки:

- Если количество товара от 10 до 19 — скидка 5%.
- Если количество товара от 20 до 49 — скидка 10%.
- Если количество товара от 50 до 99 — скидка 15%.
- Если количество товара 100 и более — скидка 20%.

Пример вывода:

Python

Введите цену товара: 100

Введите количество товара: 25

Сумма заказа с учётом скидки: 2250.00

Решение:

Python

```
price = float(input("Введите цену товара: "))
quantity = int(input("Введите количество товара: "))

if 10 <= quantity < 20:
    discount = 0.05
elif 20 <= quantity < 50:
    discount = 0.10
elif 50 <= quantity < 100:
    discount = 0.15
elif quantity >= 100:
    discount = 0.20
else:
    discount = 0.0

# Рассчитываем общую стоимость с учетом скидки
total = price * quantity * (1 - discount)
# Округляем итоговую сумму до двух знаков после запятой
total_rounded = round(total, 2)
print("Сумма заказа с учётом скидки:", total_rounded)
```

2. Площадь и длина круга

Напишите программу, которая определяет площадь круга и длину окружности по радиусу, полученному от пользователя, и выводит их на экран, округлив до сотых.

Формулы:

- Площадь круга: $S = \pi * r^2$
- Длина окружности: $C = 2 * \pi * r$

Пример вывода:

Python

Введите радиус: 5

Площадь круга: 78.54

Длина окружности: 31.42

Решение:

Python

```
import math
```

```
radius = float(input("Введите радиус круга: "))
```

```
area = math.pi * radius ** 2
```

```
circumference = 2 * math.pi * radius
```

```
print("Площадь круга:", round(area, 2))
```

```
print("Длина окружности:", round(circumference, 2))
```