

Типы данных. Конструкции языка

Цель занятия

После освоения темы вы:

- узнаете базовые типы и конструкции языка Python,
- используете базовые типы и конструкции Python для написания простых программ.

План занятия

1. Числовые типы и операции над ними.
2. Строковый тип данных.
3. Логический тип данных.
4. Условный оператор.
5. Цикл `while`.
6. Цикл `for`.
7. Операторы `continue` и `break`.

Используемые термины

Переменная — именованная область оперативной памяти, в которой хранятся некоторые данные определённого типа.

Тип данных — характеристика, которая определяет возможные значения переменных и действия над ними.

Цикл — конструкция языка программирования, используемая при повторении одних и тех же команд несколько раз.

Конспект занятия

1. Числовые типы и операции над ними

На рисунке 1 показаны основные простые типы данных в Python.

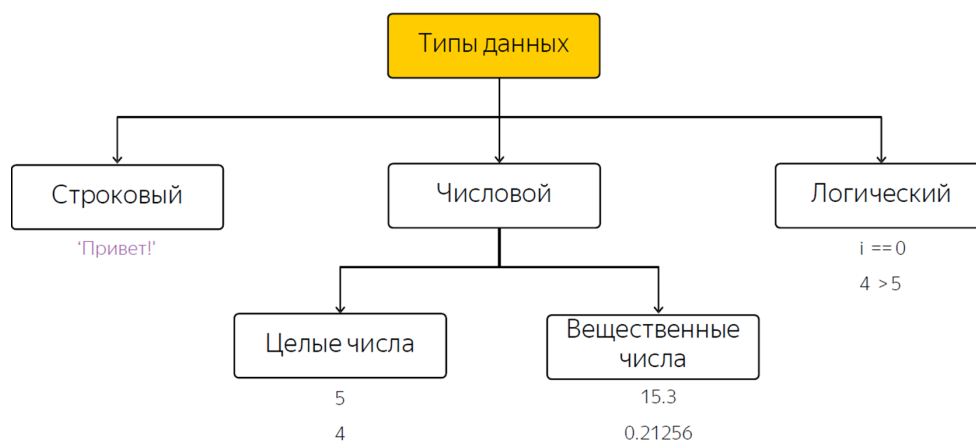


Рисунок 1. Простые типы данных

Целые числа (integer) — положительные и отрицательные целые числа, а также 0 (например: 4, 687, −45, 0). Для преобразование в целое число используется команда `int()`.

Числа с плавающей точкой (float point) — дробные числа (например: 1.45, -3.789654, 0.00453). Разделителем целой и дробной части служит символ точки «.». Для преобразование в целое число используется команда `float()`.

Не всегда при выполнении математических операций тип данных результата совпадает с типом данных операндов. Например, при делении целых чисел результатом будет вещественное число. В таблице 1 показаны основные математические операторы, их описание и примеры использования.

Таблица 1. Математические операторы.

Оператор	Описание	Пример	Результат
+	Сложение	7 + 3	10
−	Вычитание	7 − 3	4
*	Умножение	7 * 3	21
/	Деление (истинное)	7/3	2.3333333333333335
**	Возведение в степень	7 ** 3	343
//	Целочисленное деление	7 // 3	2
%	Остаток от деления	7 % 3	1

Нужно помнить о приоритете выполняемых операций:

1. Возведение в степень(**)
2. Унарный минус (-)
3. Умножение (*), деление(/ % //)
4. Сложение (+), вычитание (-)
5. Операторы сравнения (<= <> >=)
6. Операторы равенства (== !=)
7. Логические операторы (not or and)
8. Оператор присваивания (=)

Если одна из операций в выражении не совпадает со списком выше, нужно поместить ее в скобки.

В Python есть возможность поменять тип данных с помощью функций:

- `int()` — преобразует аргумент в целое число
- `float()` — преобразует аргумент в число с плавающей точкой

Пример кода, преобразующего введенную пользователем строку в число и записывает в переменную `a`

```
a = int(input())
```

Пример программы сложения двух целых чисел: в программе считываются два числа с консоли, преобразуются в целые числа, затем выводится их сумма с помощью функции `print()`.

```
a = int(input())
```

```
b = int(input())
```

```
print(a + b)
```

Пример программы сложения двух чисел с плавающей точкой: отличие — использование команды `float()` для преобразования типов введенных данных.

```
a = float(input())
```

```
b = float(input())
```

```
print(a + b)
```

Программа для нахождения длины окружности, площади круга и объема шара одного и того же радиуса.

Входным параметром является радиус, все остальные параметры рассчитывает программа.

Для ввода значения радиуса используем команду `input()` с подсказкой. Но команда `input()` считывает данные в строковом формате, поэтому используем перевод в вещественное значение — функция `float()`. Значение радиуса считывается в переменную `r`.

```
r = float(input("Введите радиус: "))
```

Точное значение числа π можно считать из специальной библиотеки, но пока мы укажем его в явном виде:

```
pi = 3.14
```

Длину окружности, площади круга и объем шара будем находить по известным математическим формулам:

```
l = 2 * pi * r
```

```
s = pi * r ** 2
```

```
v = (4 / 3) * pi * r ** 3
```

Символ «`**`» означает возведение в степень.

Для вывода в консоль результатов расчетов используем команду `print()`:

```
print("l = ", l)
```

```
print("s = ", s)
```

```
print("v = ", v)
```

Программа, демонстрирующая операции целочисленного деления и нахождения остатка от деления.

```
a = int(input("Введите делимое: "))
```

```
b = int(input("Введите делитель: "))
```

```
d = a // b
```

```
ost = a % b
```

```
print("Результаты целочисленного деления: ", d)
```

```
print("Результат остатка от деления:", ost)
```

Программа ждет ввода от пользователя двух целых чисел. Важно использовать именно целые числа, поскольку операции целочисленного деления (`//`) и нахождения остатка от деления (`%`) выполняются только над целыми числами. С помощью команды `print()` выполняется вывод результатов.

Программа для нахождения периметра и площади треугольника по длинам его сторон.

Программа рассчитывает площадь по [формуле Герона](#), все необходимые пояснения приведены в комментариях (#).

```
a = int(input("Введите a: "))
b = int(input("Введите b: "))
c = int(input("Введите c: "))
p = a + b + c #расчет периметра
pp = p / 2 #расчет полупериметра
s = (pp * (pp - a) * (pp - b) * (pp - c)) ** 0.5 #расчет площади
по формуле Герона
print("Периметр треугольника p = ", p)
print("Площадь треугольника s = ", s)
```

2. Строковый тип данных

Строки (string) — набор символов, заключенных в кавычки (например: "ball", "What is your name?", 'dkfjUUv', '6589'). Символами являются буквы, цифры, специальные знаки и пробел. Для преобразование в целое число используется команда `str()`.

Кавычки в Python могут быть одинарными или двойными.

Строки могут использовать спецсимволы, которые не выводятся, но выполняют определенные действия. Например, таким образом, можно вывести символы, которые используются для написания команд.

- `\n` — конец строки
- `\t` — табуляция
- `\'` — одинарная кавычка
- `\\` — обратный слэш

Пример программы для использования спецсимволов.

```
print("восход\t07:15\nзакат\t22:03")
```

В результате будут выведены значения в столбик, так как используется символ табуляции.

Простейшие операции над строками:

- конкатенация (склеивание), несмотря на то, что в примере переменные содержат числа, программа будет работать с ними как со строками, поскольку значения взяты в кавычки.

```
x = '10'
y = '20'
print(x + y)
```

Результат работы программы: 1020.

- дублирование позволяет несколько раз повторить строку.

```
x = '10'
y = '20'
print(x * 2 + y * 3)
```

Результат работы программы: 1010202020.

Таблица 2. Операции над разными типами данных

Выражение	Результат выполнения, тип данных
34.907 + 320.65	355.55699999999996 число с плавающей запятой
'Hi, ' + 'world :) '	'Hi, world :) ' строка, выполняется конкатенация
'Hi, ' * 10	'Hi, Hi, Hi, Hi, Hi, Hi, Hi, Hi, Hi, Hi, ' строка, выполняется дублирование
'Hi, ' + 15	Ошибка, попытка сложить строку и целое число

3. Логический тип данных

Логический тип данных необходим, когда требуется проверить на истинность некоторое выражение. Чаще выполняется сравнение. Переменные логического типа могут принимать значения: True (истина) либо False (ложь).

Например:

```
2 + 2 == 4 — True
2 + 2 == 5 — False
```

Логический тип данных имеет специальные операции:

- `and` — логическое И, логическое умножение;
- `or` — логическое ИЛИ, логическое сложение;
- `not` — логическое НЕ, логическое отрицание,

- `in` — вхождение левого элемента в то или иное множество, в ту или иную коллекцию.

Таблица 3. Логические операции

Выражение	Результат
<code>True and True</code>	<code>True</code>
<code>True and False</code>	<code>False</code>
<code>False and False</code>	<code>False</code>
<code>True or True</code>	<code>True</code>
<code>True or False</code>	<code>True</code>
<code>False or False</code>	<code>False</code>
<code>not True</code>	<code>False</code>
<code>not False</code>	<code>True</code>

4. Условный оператор

Используется, когда некая часть программы должна быть выполнена, только если верно какое-либо условие, либо должна быть выполнена другая часть программы, когда это условие неверно. Может быть, что никакие действия не выполняются.

Для записи используются ключевые слова `if` и `else` («если», «иначе»), двоеточие, а также отступ в четыре пробела.

Синтаксис конструкции языка.

```
if условие:
    действие, если условие верно
else:
    действие, если условие неверно
```

В условии может быть отношение как логическая переменная, также может быть переменная другого типа. Блоки кода определяются через отступы, чаще всего используют четыре пробела или табуляцию.

Пример программы для проверки пароля.

Программа проверяет введенный пароль, если пароль совпадает со строкой 'qwerty', выводится сообщение «Доступ открыт.», в противном случае – «Ошибка, доступ закрыт!».

```
print('Введите пароль')
password = input()
if password == 'qwerty':
    print('Доступ открыт.')
else:
    print('Ошибка, доступ закрыт!')
```

Если в условном операторе требуется проверка нескольких выражений, можно использовать логические операторы:

- `and` – два условия должны выполняться одновременно,
- `or` – достаточно выполнения одного из двух вариантов (или оба сразу),
- `not` – нужно убрать какой-то вариант.

Приоритет логических операций: `not`, `and`, `or`.

Пример использования логической операции `and`.

В программе условие будет истинным, только при выполнении обоих выражений в операторе `if`.

```
print('Как называются первая и последняя')
print('буквы греческого алфавита?')
greek_letter1 = input()
greek_letter2 = input()
if greek_letter1 == 'альфа' and greek_letter2 == 'омега':
    print('Верно.')
else:
    print('Неверно.')
```

В первых двух строках программы на консоль выводится пояснение, что требуется сделать пользователю. Далее программа считывает строку с помощью команды `input()`. Затем программа выполняет проверку условия с помощью оператора `if`. Если хотя бы одно из выражений `greek_letter1 == 'альфа' and`

`greek_letter2 == 'omega'` не выполняется, осуществляется переход к блоку `else`.

В блоках `if-else` может использоваться не одна команда, как это рассматривалось в предыдущих примерах, а несколько. Блоки кода в Python выделяются с помощью отступов.

Пример программы, использующей блоки кода. Если условие в операторе `if` верно, выполняются две команды:

```
print('Аве, Цезарь!')
print('Слава императору!')
```

Если условие неверно, выполняются также две команды, но которые уже прописаны в блоке `else`:

```
print('Приветик.')
print('Погода сегодня хорошая.')
```

Важно! Количество команд в блоках `if-else` может быть любым, главное – соблюдать отступы.

```
print('Представься, о незнакомец!')
name = input()
if name == 'Цезарь' or name == 'Caesar':
    print('Аве, Цезарь!')
    print('Слава императору!')
else:
    print('Приветик.')
    print('Погода сегодня хорошая.')
print('Засим – заканчиваем.')
```

В Python допускается конструкция `if-elif`. Эта конструкция используется при проверке нескольких условий.

```
if условие1:
    ...
elif условие2:
    ...
elif условие3:
    ...
```

Цепочка может быть заменена цепочкой `else-if`, как это реализовано в других языках программирования. Но использования формы `if-elif` является в Python предпочтительной.

Пример программы «Личностный тест» с использованием условного оператора в конструкции `if-elif` и блоков кода.

С помощью команды `print()` выводим в консоль два вопроса. Ответы, введенные в консоль, считываем командой `input()` и записываем в переменные `answer1` и `answer2`:

```
print('Любите ли вы котиков?')
answer1 = input()
print('Умеете ли вы программировать?')
answer2 = input()
```

Рассмотрим все возможные случаи в предположении того, что ответ должен состоять из «да» или «нет»:

- да-да,
- да-нет,
- нет-да,
- нет-нет.

В каждом случае будем выдавать соответствующее сообщение. В программе предусмотрен случай, когда ни один из вариантов не сработал – блок `else`.

```
if answer1 == 'да' and answer2 == 'да':
    print('Да вы просто идеал!')
elif answer1 == 'да' and answer2 == 'нет':
    print('Вы обладаете редкостной добротой.')
elif answer1 == 'нет' and answer2 == 'да':
    print('Вы обладаете незаурядным умом.')
elif answer1 == 'нет' and answer2 == 'нет':
    print('У вас большие перспективы.')
else:
    print('Ошибка: ожидалось ответы да/нет')
```

В условном операторе может использоваться оператор `in`, который проверяет содержит ли данная строка конкретную подстроку

'хорош' in 'хорошо'	— верно
'хорош' in 'эх, хорошо'	— верно
'хорош' in 'плохо'	— неверно
'хорошо!' in 'хорошо'	— неверно

Пример программы проверки наличия подстроки в строке. Программа проверяет два условия, и в зависимости от значения переменной `text` выводит то или иное сообщение, если ни одно условие неверно, выполняется код в блоке `else`.

```
text = input()
if 'хорош' in text and 'плох' not in text:
    print('Положительная эмоциональная окраска')
elif 'плох' in text and 'хорош' not in text:
    print('Отрицательная эмоциональная окраска')
else:
    print('Нейтральная или смешанная эмоциональная окраска')
```

5. Цикл `while`

Цикл — конструкция языка программирования, используемая при повторении одних и тех же команд несколько раз. То есть цикл будет выполнять блок кода, пока выполнено условие.

В рассматриваемом примере программа считывает действительное число и записывает его в переменную `number`. Пока переменная `number > 0` будет выполняться тело цикла, состоящее из трех строк. Тело цикла отделено отступом слева. Если условие перестанет выполняться, программа выйдет из цикла и перейдет на последнюю строку `print('Вы ввели отрицательное число или ноль. Всё.')`

Условие в цикле `while` очень похоже на условие в конструкции `if`. Но в отличие от условного оператора, команды, прописанные после входа в цикл, будут повторяться до тех пор, пока будет истинным условие.

```
number= float(input())
while number > 0:
    print('Вы ввели положительное число! Вводите дальше.')
    number= float(input())
    print('Так-так, что тут у нас...')
```

```
print('Вы ввели отрицательное число или ноль. Всё.')
```

Цикл `while` сокращает размер кода. В таблице 4 показано сравнение программ сложения действительных чисел, имитирующих цену товара. Числа вводятся с клавиатуры.

Программу из левого столбца легко сократить, используя цикл `while`. В цикле использовано условие: если товары закончились, необходимо ввести 0. Программа стала более универсальной — можно суммировать любое количество чисел.

Таблица 4. Сравнение программ с и без использования оператора цикла.

Без использования цикла <code>while</code>	С использованием цикла <code>while</code>
<pre>total = 0 price = float(input()) total = total + price price = float(input()) total = total + price price = float(input()) total = total + price print('Сумма введенных чисел равна', total)</pre>	<pre>total = 0 print('Вводите цены; для остановки введите 0.') price = float(input()) while price != 0: total = total + price price = float(input()) print('Общая стоимость равна', total)</pre>

Некоторые программы должны выполняться до тех пор, пока пользователь не ввел какое-то значение, или какую-то строку, или не использовал определенную комбинацию клавиш.

Пример программы «Числа до нуля»: пользователь вводит числа, и пока введенные числа не равны нулю, программа работает, как только происходит иное — завершает работу.

```
height = int(input())
while height != 0:
    height = int(input())
```

Составной оператор присваивания используется во многих языках программирования и является стандартом записи изменения переменной. Конструкция позволяет сократить запись кода.

```
number = int(input()) #например, 5
number = number + 1 #number становится равным 6
```

```
print(number)
```

В примере вводится значение переменной `number`, в следующей строке кода ее значение увеличивается на 1.

Таблица 5. Составной оператор присваивания.

<code>x = x + y</code>	<code>x += y</code>
<code>x = x - y</code>	<code>x -= y</code>
<code>x = x * y</code>	<code>x *= y</code>
<code>x = x / y</code>	<code>x /= y</code>
<code>x = x // y</code>	<code>x //= y</code>
<code>x = x % y</code>	<code>x %= y</code>
<code>x = x ** y</code>	<code>x **= y</code>

Во всех случаях (таблица 5) происходит изменение переменной `x` на значение переменной `y`.

Программа проверки ввода пароля до тех пор, пока пароль не был введен верно.

Программа запрашивает у пользователя ввод пароля и записывает его в переменную `password`.

```
print("Введите пароль: ")
password = input()
```

Рассмотрим случай, когда запрашиваемый пароль «12345». Если пароль введен неверно, пользователю выводится соответствующее сообщение и предложение еще раз ввести пароль. Таким образом, цикл будет повторяться до тех пор, пока не будет введен нужный пароль.

```
while password != "12345":
    print("Неверный пароль!")
    print("Введите снова")
    password = input()
```

Если пароль введен верно, выводится сообщение:

```
print("Доступ открыт!")
```

6. Цикл `for`

Цикл `for` используется при известном количестве итераций (повторов) тела цикла.

Рассмотрим пример программы с использованием цикла `for`. С консоли с помощью `input()` вводят количество повторений тела цикла `n`. Конструкция `for` использует функцию `range()`, которая в рассматриваемом примере возвращает последовательность целых чисел, начинающуюся с 0 и оканчивающуюся числом `n-1`.

```
n = int(input())
for i in range(n):
    print(i)
```

Тело цикла выполняется `n` раз. Переменная `i` «пробегаёт» значения от 0 включительно до `n` не включительно:

- сначала `i` равно 0,
- затем `i` равно 1,
- на последней итерации `i` равно `n-1`.

При `n = 10` переменная `i` будет принимать целые значения от 0 до 9.

Тело цикла состоит из одной команды `print(i)`. Переменная `i` как может использоваться, так может и не использоваться в теле цикла.

Пример программы для расчета суммы целых чисел от 0 до `n-1`.

```
n = int(input())
total = 0
for i in range(n):
    print('Рассматриваем число', i)
    total += i
    print('Промежуточная сумма равна', total)
print('Итоговая сумма всех этих чисел равна', total)
```

В программе вводится число `n`, обнуляется счетчик `total`. Далее при каждой итерации рассматривается определенное число, хранящееся в переменной `i`. После прохождения всех итераций выводится итоговая сумма чисел.

Пример программы нахождения суммы четных чисел на отрезке от `m` до `n`.

При решении задачи будем полагать, что `m` и `n` — целые числа, `m < n`.

Сначала вводим значения `m` и `n` с клавиатуры:

```
m = int(input())
```

```
n = int(input())
```

Чтобы посчитать сумму, определяем переменную `s` и задаем ей начальное значение:

```
s = 0
```

Все суммируемые числа, будем проверять на четность внутри цикла. Поскольку функция `range()` исключает последнее число из рассмотрения, мы будем использовать запись `range(m, n + 1)`.

```
for i in range(m, n + 1):
```

```
    if i % 2 == 0:
```

```
        s += i
```

```
print(s)
```

7. Операторы `continue` и `break`

Оператор `break` немедленно прерывает выполнение цикла `for` или `while`. Использование таких операторов может быть необходимо, если на одной из итераций цикла уже был получен ответ.

Пример программы с использованием оператора `break`. Оператор прерывает весь цикл при значении `i = 3`, и происходит переход к концу цикла.

```
for i in range (10):
```

```
    print('Итерация номер', i, 'начинается ...')
```

```
    if i == 3:
```

```
        print('Ха! Внезапный выход из цикла!')
```

```
        break
```

```
    print('Итерация номер', i, 'успешно завершена.')
```

```
print('Цикл завершён')
```

Оператор `break` может использоваться для выхода из бесконечного цикла. Выход из цикла будет выполняться при вводе слова «стоп» в консоль.

```
while True:
```

```
    word = input()
```

```
    if word == 'стоп':
```

```
        break
```

```
    print('Вы ввели:', word)
```

```
print('Конец')
```

Оператор `continue` немедленно прерывает текущую итерацию цикла `for` или `while` и переходит к следующей. Например, это необходимо в случае невозможности выполнения текущей итерации при определенном значении, типичный пример — деление на ноль. Все остальные итерации при этом выполняются.

Пример программы с использованием оператора `continue`. Оператор прерывает итерацию при значении `i = 3`, последующие итерации цикла выполняются.

```
for i in range(10):
    print('Итерация номер', i, 'начинается ...')
    if i == 3:
        print('... но не завершается успешно.')
        continue
    print('Итерация номер', i, 'успешно завершена.')
print('Цикл завершён')
```

Операторы `break` и `continue` имеют особенность использования — они действуют только на внутренний цикл.

В программе используется вложенный цикл, в котором используется оператор `break`. При запуске программы оператор будет прерывать только внутренний цикл, который определяется переменной `j`. Ко внешнему циклу оператор `break` отношение не будет иметь.

```
n = int(input())
for i in range(1, n + 1):
    for j in range(1, n + 1):
        print(i * j, end='\t')
        if j == 7:
            break
    print()
```

Дополнительные материалы для самостоятельного изучения

1. [More Control Flow Tools — Python 3.10.7 documentation.](#)