

## Конспект

### Знакомство с переменными и базовыми функциями

#### Переменные.

Слово «переменные» говорит само за себя – их значение может меняться, а значит, вы можете хранить в переменной всё, что угодно. Переменные – это

просто области памяти компьютера, в которых вы храните некоторую информацию.

В отличие от констант, к такой информации нужно каким-то образом получать доступ, поэтому переменным даются имена.

Переменные (variable) это своего рода подписанные коробочки, в которых хранится информация. Вы можете изменять эту информацию, но только не название коробочки

(Переменные полезны, когда надо отслеживать изменение данных, например, счета в игре. )

Переменные невероятно важны, так как позволяют хранить информацию и использовать её в дальнейшем. Вначале может быть не совсем понятно зачем вообще что-то записывать в переменную, если можно просто оперировать значениями без них. Понимание переменных придет немного позже, когда мы начнем создавать более сложные программы и нам потребуется хранить информацию в каком-либо месте.

Типы переменных в языке Python не объявляются очевидно, тем не менее они присутствуют. Интерпретатор понимает что записывается в переменную и на основании этого добавляет тип к этой переменной.

В одной строке можно создать сразу несколько переменных:

```
first = sec = third = 1 # Всем трём переменным будет присвоено значение 1
first, sec, third = "Hi", 75, 23.1 # Поочередное присвоение значений
```

Во время выполнения программы есть возможность перезаписывать переменные.

Пример:

```
a = 8
```

```
print(a)
```

```
a = 9
```

```
print(a)
```

### Типы переменных/данных.

Всего в Python есть 4 базовых типа переменных:

some = 1 Integer - целые числа;

some = 1.12 Float - числа с плавающей точкой;

some = "Привет" String - строки;

some = True Boolean - тип данных принимающий либо False, либо True.

Есть и другие типы, но мы будем их разбирать в последующих уроках.

То есть, если мы к числу 1 попробуем прибавить 2, затем еще 2 и вычесть 3 то понятно что произойдет - все считается.

Но если из строки “привет” мы попытаемся вычесть 1, то явно не понятно что должно произойти.

При объединение нескольких переменных с разными типами данных программа провоцирует ошибку.

Пример:

```
first_num = "IloveYou"
second_num = 13
res = first_num + second_num # Скрипт выдаст ошибку
```

На этом этапе нам пока достаточно знаний о типах, более детально мы их рассмотрим позже.

## **Code style.**

Хочу упомянуть на счет такой штуки как code style, то есть стиль вашего кода, то как он выглядит и читается.

К примеру есть почерк многих врачей который разобрать без помощи аналитиков и криптографов довольно таки сложно.

А есть газета в которой все разобрано по полочкам и в которой легко найти нужную информацию.

Так вот во время написания кода лучше опираться на второй вариант, более подробно мы будем разбираться с этим на протяжении всех вебинаров, но пока я просто резюмирую это.

---

## **Арифметические операции с целыми числами.**

### **Объекты.**

Помните, Python рассматривает всё, что есть в программе, как объекты. Имеется в виду, у, в самом общем смысле. Вместо того, чтобы говорить «нечто», мы говорим «объект».

### **Операторы.**

Символы, такие как «+» и «-», называются в среде Python математическими операторами. Вот самые распространенные математические операторы:

- + — сложение
- — вычитание
- \* — умножение
- / — деление

В целом это не сильно отличается от обычной математики. Т.е.

Нам достаточно просто сложить два числа.

$a = 5 + 6$

$b = 6 - 5$

$c = 4 * 5$

И деление, с ним важно уточнить что существует два типа деления, целочисленное и с остатком.

Например если мы 7 поделим на 5, то целая 5ка в 7 содержится 1 раз, то есть результат деления 7 на 5 будет равен 1.

Если мы захотим взять остаток от деления то мы получим 2.

$d = 7 // 5$  - целочисленное деление где мы получим 1

$e = 7 \% 5$  - это деление с остатком.

Теперь давайте выведем их все по очереди.

Также числа можно выводить в степень

$a = 2 ** 2$

Окей давайте рассмотрим еще один приятный момент который есть в python.

Есть переменная которая равна допустим 5.

Мы хотим её увеличить, как это можно сделать:

```
1  a = 5
2  a = a + 10
3  |
4  print(a)
```

Но с такими операциями можно делать не большое сокращение.

Как это будет выглядеть:

```
a = 5
a += 10
print(a)
```

То есть вместо того чтобы писать переменную мы сразу объявляем что мы прибавляем 10 и присваиваем новое значение.

Аналогично можно сделать и с другими арифметическими выражениями:

```
a = 5
a *= 10
print(a)
```

Так мы базово разобрались как работают арифметические операции.

Теперь попробуем повзаимодействовать с вводом.

Давайте попробуем ввести число с клавиатуры.

```
a = input()
print(a * 5)
```

Давайте попробуем это запустить и получим неожиданный результат (55555)

Это работает так , потому что input это пока что строка чтобы привезти её к целому числу нужно прописать в начале int.

Что делает int, он пытается взять значение и привести его к целому числу, то есть 5 будет равно 5, но если ввести абракадабру то получим ошибку.

```
a = int(input())
print(a * 5)
```

Если введем число получим результат, если введем слово получим ошибку.

А что если наша задача ввести несколько чисел?

допустим мы хотим ввести 2 и 3.

```
a = int(input())  
b = int(input())  
print(a * b)
```

Но если ввести числа на одной строке через пробел то получим ошибку. Потому что, инпут вводит всю строку целиком, а наша строка это число пробел число и совершенно не очевидно для программы какое число должно получиться.

Что в таком случае можно сделать.  
Изменим немного строку.

Есть такая операция как `split`, что она делает, берет и разбивает строку по какому то разделителю. Пока это необходимо просто понять и запомнить, далее будем разбираться глубже.

например:

```
a = input().split()  
print(a)
```

То есть как это работает, сплит разделяет на строки в тех местах где у нас пробел

3 пробел 2 пробел 4 - это три строки

Попробуем вывести это.

Как видите числа в кавычках, то есть это еще не полноценные числа.

Чтобы привести это к обычным числам, можно написать переменные через запятую.

```
a, b = input().split()  
print(a)
```

но так у нас тоже получаются не совсем числа, программа еще не понимает что это числа, то есть если ввести:

```
a, b = input().split()
print(a * 2)
```

То получим а два раза.

Познакомимся еще с командой которая нам поможет map, далее мы так же все разберем подробнее.

Сейчас будет сложновато, но я все объясню по порядку.

```
a, b = map(int, input().split())
print(a * 2)
```

Input - ввод

Split - разделитель строки

Int - тип данных

map - применяет int к каждому значению из input split.

```
a, b, c = map(int, input().split())
print(a * 2)
```

Если Вы сейчас ничего не поняли, то это нормально и мы разберемся с этим подробнее, просто пока я показываю способ который есть.

Теперь более длинный, но более легкий способ:

```
a, b, c = input().split()
a = int(a)
print(a * 2)
```

**Показать пример для дз.**

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

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

```
print (f'ОТВЕТ: {a * b}')
```

## Приоритет операций:

```
a, b = map(int, input().split())
c = (a + b) * 4
print(c)
```

Приоритет вычислений такой же как в математике.

Можно еще попробовать такой способ:

```
a, b = map(int, input().split())
d = a + b
c = d * 4
print(c)
```

Также питон может считать и числа огромных размеров:

```
d = 10000000000000000000000000000000000000000000000000000000 * 81247390821734098712038470198273408912734098  
print(d)
```

### Вещественные числа:

```
1 a = 10
2 b = 3
3 print(a // b)
4
```

Как же сделать так чтобы мы считали и число после точки:

У нас существует три типа деления :

/ - вещественное

// - целочисленное

% - остаток.

Также питон округляет числа:



```
pers.py > ...
1 a = 10
2 b = 3
3 print (a / b)
4
5

ПРОБЛЕМЫ Выходные данные КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ JUPYTER

2
PS C:\Users\home\Desktop\Web2> python pers.py
2
3
Ответ: 6
PS C:\Users\home\Desktop\Web2> python pers.py
3.3333333333333335
```

но если умножим на 3 то получим 10.0

```
print(10 / 3 * 3)
```

С большими числами мы можем получить погрешность и получать уже другое число.

Если мы хотим поделить на ноль то питон будет ругаться.

## FLOAT

Для того чтобы сделать число вещественным, есть операция float.

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

Для неё так же применяются все операции умножения деления и тд.

Также мы можем менять тип данных как с переменными.

Если вначале переменная была с типом float, то потом её можно преобразовать в другой тип, к примеру, в string.

```
first_num = 23.2 # Тип данных float
first_num = "1" # Тип данных string
```

---

## Логические и условные операторы.

Мы познакомились с тремя типами данных – целыми и вещественными числами, а также строками. Сейчас мы введем четвертый – логический тип данных (тип bool). Его также называют булевым. У этого типа всего два возможных значения: True (правда) и False (ложь).

Как это выглядит на примере

Например, если вам скажут, что сумма чисел 3 и 5 больше 7, вы согласитесь, скажете: «Да, это правда». Если же кто-то будет утверждать, что сумма трех и пяти меньше семи, то вы расцените такое утверждение как ложное.

Подобные фразы предполагают только два возможных ответа – либо "да", когда выражение оценивается как правда, истина, либо "нет", когда утверждение оценивается как ошибочное, ложное. В программировании и математике если результатом вычисления выражения может быть лишь истина или ложь, то такое выражение называется логическим.

Например, выражение  $4 > 5$  является логическим, так как его результатом является либо правда, либо ложь. Выражение  $4 + 5$  не является логическим, так как результатом его выполнения является число.

В программировании False обычно приравнивают к нулю, а True – к единице. Чтобы в этом убедиться, можно преобразовать булево значение к целочисленному типу:

```
1 a = int(True)
2 print(a)
```

Возможно и обратное. Можно преобразовать какое-либо значение к булевому типу:

```
>>> bool(3.4)
True
>>> bool(-150)
True
>>> bool(0)
False
>>> bool(' ')
True
>>> bool('')
False
```

И здесь работает правило: всё, что не 0 и не пустота, является правдой.

Говоря на естественном языке (например, русском) мы обозначаем сравнения словами "равно", "больше", "меньше". В языках программирования используются специальные знаки, подобные тем, которые используются в математике:

> (больше),

< (меньше),

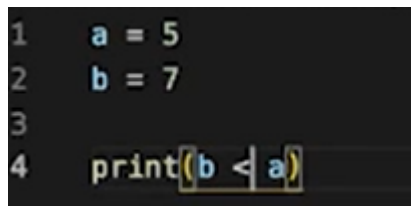
>= (больше или равно),

<= (меньше или равно),

== (равно),

!= (не равно).

Пример:



```
1 a = 5
2 b = 7
3
4 print(b < a)
```

Не путайте операцию присваивания значения переменной, обозначаемую в языке Python одиночным знаком "равно", и операцию сравнения (два знака "равно"). Присваивание и сравнение – разные операции.

```
>>> a = 10
>>> b = 5
>>> a + b > 14
True
>>> a < 14 - b
False
>>> a <= b + 5
True
>>> a != b
True
>>> a == b
False
```

```
1 a = bool(10 + 5 > 30)
2 print(a)
3
```

---

## IF

Сделаем проверку меньше ли а чем б.

Мы пишем if, затем пишем какое-то условие результатом которого будет либо True либо False.

Некорректное условие:

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

if a < "lakshdf;l;hasfd" |
```

Теперь появляется развилка, то есть либо выполняется либо нет.

Пишем двоеточие т.е. “что будет дальше если условие выполнилось”.

Теперь важное уточнение, то есть все что мы вводим сейчас будет выводиться только если выполнено условие.

```
3 / Desktop / Synergy
a = int(input())
b = int(input())

if a < b:
    print("YES")
```

### **Зачем нужны отступы.**

В Python пробелы важны. Точнее, пробелы в начале строки важны. Это называется отступами. Передние отступы (пробелы и табуляции) в начале логической строки используются для определения уровня отступа логической строки, который, в свою очередь, используется для группировки предложений.

Это означает, что предложения, идущие вместе, должны иметь одинаковый отступ. Каждый такой набор предложений называется блоком. В дальнейших главах мы увидим примеры того, насколько важны блоки. Вы должны запомнить, что неправильные отступы могут приводить к возникновению ошибок.

Все что мы напишем дальше без отступа будет выполняться независимо от условия if

```
1  a = int(input())
2  b = int(input())
3
4  if a < b:
5      print("YES")
6
7  print("Genadiy")
```

еще пример

```

1  a = int(input())
2  b = int(input())
3
4  if a < b:
5      print("YES")
6      print("Genadiy")

```

Теперь обработаем второй случай если наше условие не выполнилось.

Будет использовать слово “иначе” else.

Заметьте что каждый раз как ставим двоеточие следующий блок идет с отступом. финиш

```

1  a = int(input())
2  b = int(input())
3
4  if a < b:
5      print("YES")
6  else:
7      print("NO")

```

Простой пример

```

1  a = int(input())
2  b = int(input())
3
4  if a < b:
5      print("less")
6  else:
7      if a == b:
8          print("equal")
9      else:
10         print("greater")

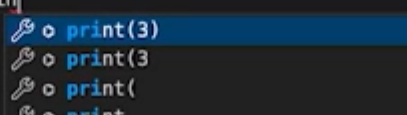
```

Слишком много повторений

```

2
3  if a <= 2:
4      print(1)
5  else:
6      if a <= 4:
7          print(2)
8      else:
9          if a <= 6:
10             prin

```



Лучший вариант.

```

1  a = int(input())
2
3  if a <= 2:
4      print(1)
5  elif a <= 4:
6      print(2)
7  elif a <= 5:
8      print(4)
9  else:
10     print(3)

```