**Аннотация**

*В этом уроке мы познакомимся с типами данных, научимся работать с числами и узнаем о простейших функциях.*

**Повторение**

На прошлом уроке мы рассмотрели условный оператор, который позволяет выполнять различные ветки кода в зависимости от заданных условий. Научились составлять сложные условия при помощи операций not, and и or.

**Типы данных**

Пока единственным типом данных, с которым мы работали, были строки. Теперь нам предстоит рассмотреть целые и вещественные числа. У каждого элемента данных, который встречается в программе, есть свой тип. (В случае с Python более правильный термин — «класс объекта», но об этом мы будем говорить гораздо позже).

Например, «привет» — это строка, а вот 15.3 — это число (дробное). Даже если данные не записаны прямо в программе, а получаются откуда-то еще, у них есть совершенно определенный тип. Например, на место input() всегда подставляется строка, а 2 + 2 даст именно число 4, а не строку "4".

Пользователь может ввести с клавиатуры какие-то цифры, но в результате input() вернет строку, состоящую из этих цифр. Если мы попытаемся, например, прибавить к этой строке 1, получим ошибку.

Давайте попробуем это сделать:

a = input()

print(a + 1)

Сохраните и запустите программу. Введите любое число и посмотрите, что получится.

Ошибка возникнет потому, что в переменную а у нас попадает строка, а в функции print мы пытаемся сложить эту строку из переменной а и число 1. Исправьте программу так, чтобы она работала.

А если нам надо работать с числами? Мы пока будем рассматривать целые и вещественные числа.

Когда речь идет о числовых данных, они записываются **без кавычек**.

А для вещественных чисел, чтобы отделить дробную часть от целой, используют **точку**.

На прошлом занятии мы складывали две строки:

print('10' + '20')

И получали результат — строку "1020".

Давайте попробуем в этом примере убрать кавычки. В таком случае речь пойдет уже не о строках, а о двух целых числах.

И результатом функции print(10 + 20) будет целое число 30.

А если мы попробуем сложить два вещественных числа print(10.0 + 20.0), результатом будет вещественное число 30.0.

Попробуйте предположить, что будет, если сложить вещественное число и целое число print(10.0 + 20). Почему?

**Операции над числами**

Мы выполняли сложение двух чисел внутри функции print, но мы можем переменным давать нужные значения и выполнять действия над переменными.

Давайте напишем программу, которая задаст нужные значения двум переменным (10 и 20), потом вычислит их сумму, положит это значение в третью переменную и выведет на экран полученный результат. Допишите начальные строки, чтобы программа решала поставленную задачу:

...

print(summ)

Обратите внимание: если в качестве имени переменной для суммы взять sum, оно выделяется цветом. Это означает, что такое имя знакомо среде и принадлежит какой-то функции, в качестве имени переменной его лучше не использовать.

Как складывать два числа, мы научились. Еще числа можно вычитать, умножать, делить, возводить в степень, получать целую часть от деления и остаток от деления нацело. Давайте разберем эти операции на примерах.

print(30 - 10)

print(30.0 - 10)

print(3 \* 3)

С вычитанием и умножением все понятно, они аналогичны сложению.

Возведение в степень обозначается двумя звездочками \*\*, которые должны записываться без разделителей.

print(9 \*\* 2)

Обратите внимание: результат деления всегда вещественный, даже если мы делим два целых числа, которые делятся нацело.

print(10 / 2)

Попробуйте поделить на 0. Посмотрите, как будет выглядеть ошибка деления на 0.

**Целочисленное деление**

Для реализации целочисленного деления существуют два действия: деление нацело и остаток от деления нацело. Получение целой части от деления обозначается как удвоенный знак деления, а остатка от деления нацело — %.

Давайте подробнее разберем эти операции. Что будет выведено в результате этих действий?

print(10 // 3, 10 % 3)

print(10 // 5, 10 % 5)

print(10 // 11, 10 % 11)

Допустим, вам известны результаты a // b, a % b и число b, напишите формулу, как найти число a?

Давайте проверим вашу формулу:

a = 10

b = 3

print(…А сюда напишем формулу…)

Обратите внимание на порядок выполнения действий в вашей формуле. Целочисленное деление имеет тот же приоритет, что и обычное деление, значит, будет выполнятся раньше, чем вычитание и сложение. Для изменения приоритета выполнения операций используются скобки, все так же, как в математике.

А теперь, немного разобравшись с этими операциями, попробуйте предположить, что выведется на экран после выполнения следующего куска кода:

print(10 // 3, 10 % 3)

print(-10 // 3, -10 % 3)

Определите, что будет выведено на экран?

a = 4

b = 15

c = b / 5 \* 3 - a

print(c)

**Приоритет операций**

Мы уже изучили несколько типов операций в языке Python:

* Операции присваивания (=, +=, -=, \*= и т. д.)
* Операции сравнения (==, !=, >, <, >=, <=)
* Арифметические операции (+, -, \*, /, %, //)
* Логические операции (and, or, not)

Есть и другие, с которыми познакомимся позднее. Все эти операции могут использоваться совместно в довольно сложных конструкциях, поэтому нужно помнить о приоритете операций и в случае необходимости менять его при помощи скобок.

Итак, приоритет выполнения операций в Python от высшего (выполняется первой) до низшего:

1. Возведение в степень (\*\*).
2. Унарный минус (-). Используется для получения, например, противоположного числа.
3. Умножение, деление (\* / % //).
4. Сложение и вычитание (+ -).
5. Операции сравнения (<= < > >=).
6. Операции равенства (== !=).
7. Логические операции (not or and).
8. Операции присваивания (=).

**PEP 8**

По обе стороны бинарной операции нужно обязательно ставить по одному пробелу. Но операции, состоящие из двойных символов (<= >= == != -= +=), пробелами не разделяются.

**Простейшие функции**

С действиями над числами определились, осталось разобраться, как получать числа из ввода. Здесь нам поможет важное новое понятие — **функция**. В математике функция из одного числа (или даже нескольких) делает другое.

**Функция**

В программировании (и в Python, в частности) функция — это сущность, которая из одного (или даже нескольких) значений делает другое. При этом она может еще и выполнять какие-то действия.

Например, есть функция модуля у = |x|, аналогично в Python есть функция y = abs(x).

Но функции в Python необязательно принимают только числа.

Для того чтобы вводить числа с клавиатуры и потом работать с ними, необходимо найти функцию, которая из строки делает число. И такие функции есть!

Тип данных целых чисел в Python называется **int**, дробных чисел — **float**.

Одноименные функции принимают в качестве аргумента строку и возвращают число, если в этой строке было записано число (иначе выдают ошибку):

a = input()

b = int(a)

print(b + 1)

Или можно написать даже так:

a = int(input())

что будет означать — «получи строку из ввода, сделай из нее целое число и результат помести в переменную а».

И тогда предыдущая программа может быть записана в виде:

a = int(input())

print(a + 1)

Но можно сократить код еще, написав вот так:

print(int(input()) + 1)

Функция int может быть применена и для получения целого числа из вещественного, в таком случае дробная часть будет отброшена (без округления).

Например, print(int(20.5 + 34.1)) выдаст на экран число 54, хотя, если сложить эти числа и не отправлять их в функцию int, результат будет 54.6.

В Python cуществует огромное количество различных функций, мы будем знакомиться с ними постепенно. Так, например, для строки можно определить еще и ее длину.

**Длина строки**

Длина строки — это количество символов в строке.

Для определения длины строки используется стандартная функция Python **len()**.

На примере функции len разберемся с основными понятиями, связанными с использованием функций. Изучите код:

word = input()

length = len(word)

print('Вы ввели слово длиной', length, 'букв.')

Использование в программе функции называется «вызов функции». Он устроен так: пишем имя функции — len, а затем в скобках те данные, которые мы передаем этой функции, чтобы она что-то с ними сделала. Такие данные называются **аргументами**.

В нашем примере данные в скобках должны быть строкой. Мы выбрали в качестве данных значение переменной word, которое пользователь до этого ввел с клавиатуры. То есть значение переменной word выступает здесь в роли аргумента. А функция len выдает длину этой строки. Если пользователь ввел, например, «привет», word оказывается равно строке «привет», и на место len(word) подставляется длина строки «привет», то есть 6.

Обратите внимание: каждый раз, когда мы пишем имя переменной (кроме самого первого раза — в операции присваивания слева от знака равно), вместо этого имени интерпретатор подставляет значение переменной.

Точно так же на место вызова функции (то есть имени функции и ее аргументов в скобках) подставляется результат ее работы, это называется **возвращаемое значение функции**.

Таким образом, функция len возвращает длину своего аргумента. input — тоже функция (отсюда скобки), она может работать, не принимая никаких аргументов, а может в качестве аргумента принимать сообщение, которое надо вывести перед ожиданием пользовательского ввода. Но всегда считывает строку с клавиатуры и возвращает ее.

print — тоже функция, она не возвращает никакого осмысленного значения, зато выводит свои аргументы на экран. Эта функция может принимать не один аргумент, а сколько угодно. Несколько аргументов одной функции следует разделять запятыми.

На самом деле функция сама по себе — это фактически небольшая программа, но об этом позже.

Функции безразлично происхождение значений, которые ей передали в качестве аргумента. Это может быть значение переменной, результат работы другой функции или записанное прямо в коде значение:

print('Это слово длиной', len('абракадабра'), 'букв.')

Обратите внимание: в предыдущем примере значение переменной word вообще никак не изменилось от вызова функции len. C другой стороны, вызов функции может стоять где угодно, необязательно сразу класть возвращаемое значение в переменную.

Как существует функция int, которая пытается сделать из того, что ей передали, целое число, так же существует и функция str, которая возвращает строку из тех данных, что в нее передали.

print(str(10) + str(20)) # выведет '1020'

print(int('10') + int('20')) # выведет 30

Каждый раз, когда вы пишете программу, важно понимать, какой тип имеет каждое значение и каждая переменная.

**Обмен значениями переменных**

Мы изучили операции с различными типами данных.

Давайте попробуем написать программу, которая поменяет местами содержимое переменных а и b. Пусть есть такой код:

a = 3

b = 5

...

...

print(a)

print(b)

Что надо вписать в пропущенные места, чтобы в а лежало 5, а в b лежало 3? При этом числами 3 и 5 пользоваться нельзя.

Как один из вариантов можно использовать дополнительную переменную:

a = 3

b = 5

с = a

a = b

b = c

print(a)

print(b)

А теперь попробуйте написать вариант без дополнительной переменной, через сумму двух чисел.

Но нам повезло, что мы изучаем язык Python, потому что он и поддерживает более простой вариант записи:

a = 3

b = 5

a, b = b, a

print(a)

print(b)

Значения переменных, которые расположены справа от знака «присвоить», в указанном порядке помещаются в переменные слева, в порядке их указания.

Так, используя множественное присваивание, можно задавать нескольким переменным одно значение:

a = b = c = 5