Приоритет операций. Простейшие функции

Аннотация

В этом уроке мы узнаем, в каком порядке выполняются арифметические и логические операции, встречающиеся в одном выражении, познакомимся с простейшими встроенными функциями и научимся обменивать значениями переменные.

1.Повторение

На прошлом уроке мы научились использовать целочисленное деление. Вспомните, как определить четность числа?

2.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:

Возведение в степень (**).
Унарный минус (-). Используется для получения, например, противоположного числа.
Умножение, деление (* / % //).

Сложение и вычитание (`+ -`).
Операции сравнения (`<= < > >=`).
Операции равенства (`== !=`).
Операции присваивания (`=`).
Логические операции (`not or and`).

Если используются операции с разными приоритетами, попробуйте добавить пробелы вокруг операций с самым низким приоритетом. Руководствуйтесь своими собственными суждениями, но никогда не используйте более одного пробела и всегда используйте одинаковое количество пробелов по обе стороны бинарной операции.

3.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 y = |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 a ».

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 фунцкию `int`, результат будет 54.6.

Попробуйте решить задачу «Одно число».

В Python существует 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`. С 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.

Функции min() и max()

Для определения соответственно минимального или максимального значения в последовательности однотипных данных используются функции `min()` и `max()`. Аргументов у этих функций может быть любое количество, главное, чтобы они все были одного типа.

Например, при передаче последовательности целых чисел будет выведено:

```
a = max(3, 8, -3, 12, 9)
b = min(3, 8, -3, 12, 9)
print(a, b)  # Выведет -> 12 -3
```

Строки тоже можно сравнивать:

```
a = max('a', 'd', 'ee', 'A', 'aa')
b = min('a', 'd', 'ee', 'A', 'aa')
print(a, b)  # Выведет -> ee A
```

4. Обмен значениями переменных

Мы изучили операции с различными типами данных.

Давайте попробуем написать программу, которая поменяет местами

содержимое переменных a и b. Пусть есть такой код:

```
a = 3
b = 5
...
...
print(a)
print(b)
```

Что надо вписать в пропущенные места, чтобы в a 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
 c  = 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
```