Язык Python

Начало

Переменные в Python

Определение

Переменная — это простейшая именованная структура данных, в которой может быть сохранён промежуточный или конечный результат работы программы.

Переменную в Python создать очень просто — нужно присвоить некоторому идентификатору значение при помощи оператора присваивания «=».

Пример

```
a = 10
b = 3.1415926
c = "Hello"
d = [1, 2, 3]
```

В этом примере используются четыре переменные:

- переменная **a** хранит значение типа **int** (целое число),
- переменная **b** — типа **float** (действительное число),
- переменная **c** — типа **str** (строка),
- переменная **d** — типа **list** (список, в данном случае из трех целых чисел).

Никакого специального объявления переменных не требуется, первое присваивание переменной значения и является ее объявлением. Идентификатор в Python является "ссылкой" на хранимые в памяти данные.

Python — язык с динамической типизацией: каждая переменная в каждый момент времени имеет определенный тип, но этот тип может меняться по ходу выполнения программы, достаточно просто присвоить ей новое значение другого типа.

Идентификатор обязательно есть у каждой переменной, функции, объекта и т.п. Идентификаторы в Питоне не ограничены по длине и чувствительны к регистру. То есть A и a — это разные имена переменных или функций.

В идентификаторах допустимы только символы от "A" до "Z" в верхнем и нижнем регистре, подчеркивание "_" и, кроме первого символа идентификатора, цифры от "0" до "9".

Например, **__aAb12_as111_1_4_5** — корректный идентификатор, а **1z** — некорректный, т.к. начинается с цифры.

Для идентификаторов на Питоне версии 3 можно использовать не только ASCII символы, но и Unicode. На практике это означает, что переменные можно называть по-русски. Но делать так не стоит, ведь ваш код вполне может попасть в руки разработчиков из других стран: будет ли вам удобно, если в библиотеке, которой вы захотите воспользоваться, найдутся функции с именами, записанными китайскими иероглифами?

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

1) Конец строки является концом инструкции (точка с запятой не требуется).

Пример

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

2) Вложенные инструкции объединяются в блоки по величине отступов. Отступ может быть любым, главное, чтобы в пределах одного вложенного блока отступ был одинаков.

Пример

```
if a == 5:
        print('yes')
        a += 1
```

3) Вложенные инструкции в Python записываются в соответствии с одним и тем же шаблоном, когда основная инструкция завершается двоеточием, вслед за которым располагается вложенный блок кода, обычно с отступом под строкой основной инструкции.

Основные операции с целыми числами

- A + B — сумма;
- A - B — разность;
- A * B — произведение;
- A / B — частное, *(результатом этого действия является вещественное число, даже если A нацело делится на B);*
- *ОСНОВНЫЕ ОПЕРАЦИИ С ЦЕЛЫМИ ЧИСЛАМИ*
- *A + B — сумма;*
- *A - B — разность;*
- A % B — взятие остатка от деления A на B;
- A // B — взятие целой части от деления A на B
- A ** B — возведение в степень.

Приоритеты операций

Приоритеты операций в Pythonсовпадают с приоритетом операций в математике, а именно:

1) Выполняются возведения в степень справа налево, то есть 3 ** 3 ** 3 это 3 ** (3 ** 3).
2) Выполняются унарные минусы (отрицания).
3) Выполняются умножения и деления слева направо. Операции умножения и деления имеют одинаковый приоритет.
4) Выполняются сложения и вычитания слева направо. Операции сложения и вычитания имеют одинаковый приоритет.

Вещественное число в Python имеет тип float. Оно записывается как последовательность цифр, перед которой также может стоять знак минус. В качестве разделителя целой и дробной части используется точка.

Основные операции с вещественными числами

- A + B — сумма;
- A - B — разность;
- A * B — произведение;
- A / B — частное, *(результатом этого действия является вещественное число, даже если A нацело делится на B);*
- A % B — взятие остатка от деления A на B, *(подразумевается, что неполное частное является целым числом);*
- A // B — взятие целой части от деления A на B, *(подразумевается, что неполное частное является целым числом);*
- A ** B — возведение в степень.

Приоритеты операций

Приоритеты операций совпадают с приоритетами операций с целыми числами.

Строки в Python имеют тип str. Строкой называется последовательность символов: букв, цифр, знакоы препинания и т.д.

Основные операции со строками

- A + B — конкатенация (строка B приписывается к строке A);
- A * n — повторение n раз, значение n должно быть целого типа.

В Python есть особый и при этом универсальный способ обмена переменных значениями.

```
(a,b) = (b,a)
```

Этот способ используется очень часто из-за своей прозрачности.

Он работает всегда, даже если переменные разных типов (в этом случае они обмениваются не только значениями, но и типами). Круглые скобки в этой записи можно опустить:

```
a, b = b, a
```

Ввод

Для считывания строки со стандартного ввода используется функция **input()**, которая считывает строку с клавиатуры и возвращает значение считанной строки, которое сразу же можно присвоить переменным:

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

Правда, функция input возвращает текстовую строку. Если нужно сделать так, чтобы переменные имели целочисленные значения, то сразу же после считывания выполним преобразование типов при помощи функции int, и запишем новые значения в переменные a и b:

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

Лучше объединить считывание строк и преобразование типов, если вызывать функцию int для того значения, которое вернет функция input():

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

Сложнее считать значения переменных, если они записаны в отдельной строке. Здесь нужно применить к считанной строке метод **split()**, который разделяет строку на части по одному или нескольким пробелам (а также табуляциям и др. пробельным символам). Затем результат выполнения этой функции присвоим двум или нескольким переменным. Например, если в строке вводятся два числа через пробел, то считать их можно так:

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

Аналогично, три переменные можно считать, записав слева от оператора присваивания кортеж из трех переменных:

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

Можно также сразу же преобразовать считанные значения в числовой тип (например, int), если воспользоваться функцией map, которая применяет к каждому элементу списка заданную функцию (для преобразования к типу int нужно, соответственно, задать функцию int для применения к каждому элементу). Для начала можно просто запомнить эту конструкцию:

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

Для вывода данных используется функция **print**, которая может выводить не только значения переменных, но и значения любых выражений. Например, допустима запись **print(2 + 2 * 2)**. Также при помощи функции print можно выводить значение не одного, а нескольких выражений, для этого нужно перечислить их через запятую:

```
a = 1
b = 2
print(a, '+', b, '=', a + b)
```

В данном случае будет напечатан текст 1 + 2 = 3: сначала выводится значение переменной a, затем строка из знака “+”, затем значение переменной b, затем строка из знака “=”, наконец, значение суммы a + b.

Обратите внимание: выводимые значение разделяются одним пробелом. Но такое поведение можно изменить: можно разделять выводимые значения двумя пробелами, любым другим символом, любой другой строкой, выводить их в отдельных строках или не разделять никак. Для этого нужно функции print передать специальный именованный параметр, называемый **sep** (от англ. separator — разделитель), равный строке, используемый в качестве разделителя. По умолчанию параметр sep равен строке из одного пробела и между значениями выводится пробел. Чтобы использовать в качестве разделителя, например, символ двоеточия, нужно передать параметр sep, равный строке ':':

```
print(a, b, c, sep = ':')
```

Аналогично, для того, чтобы совсем убрать разделитель при выводе нужно передать параметр sep, равный пустой строке:

```
print(a, '+', b, '=', a + b, sep = '')
```

Для того, чтобы значения выводились с новой строке, нужно в качестве параметра sep передать строку, состоящую из специального символа новой строки, которая задается так:

Для того, чтобы значения выводились с новой строке, нужно в качестве параметра sep передать строку, состоящую из специального символа новой строки, которая задается так:

```
print(a, b, sep = '\n')
```

Символ обратного слэша в текстовых строках является указанием на обозначение специального символа, в зависимости от того, какой символ записан после него. Наиболее часто употребляется символ новой строки **'\n'**. А для того, чтобы вставить в строку сам символ обратного слэша, нужно повторить его два раза: '\\'.

Вторым полезным именованным параметром функции print является параметр **end**, который указывает на то, что выводится после вывода всех значений, перечисленных в функции print. По умолчанию параметр end равен '\n', то есть следующий вывод будет происходить с новой строки. Этот параметр также можно исправить, например, для того, чтобы убрать все дополнительные выводимые символы можно вызывать функцию print так:

```
print(a, b, c, sep = '', end = '')
print(d)
```

Значения переменных a, b, c, d будут напечатаны без пробелов в одну строку.

Иногда бывает полезно целое число записать как строку. И, наоборот, если строка состоит из цифр, то полезно эту строку представить в виде числа, чтобы дальше можно было выполнять арифметические операции с ней. Для этого используются функции, название которых совпадает с именем типа, то есть int, float, str. Например, int('123') вернет целое число 123, str(123) вернет строку '123', а следующая инструкция:

```
print(str(2 + 2) * int('2' + '2'))
```

выведет символ "4", повторенный 22 раза.

Функция int также поможет превратить дробное число в целое, отбросив дробную часть: int(12.3) = 12, int(-12.3) = 12.

Еще один полезный пример использования — преобразование строки в список букв:

```
list('abc') = ['a', 'b', 'c']
```

Также преобразование типов активно используется с функцией map и генераторами, например,

```
numbers = list(map(int, input().split()))
```