Основы программирования на Python

Вывод на экран

```
print ("Hello World")
```

print — функция вывода на экран, в скобках указываем, то что хотим вывести. Если это простая строка, то указываем в кавычках. Если переменная, то без кавычек. Про переменные позже.

Переменные

Переменная — это такая штука, в которой хранится какое-либо значение. Каждая переменная имеет тип и значение. Значение переменной может изменяться в ходе выполнения программы.

Как могут называться переменные?

- Латинские буквы, как маленькие так и большие
- Переменная может содержать цифры, но не может начинаться с цифры
- знак нижнего подчеркивания _

Математические операции

Деление:

```
a = 24
b = 5
c = a / b
print (c) # c = 4.8
```

Остаток от деления

```
a = 24
b = 5
c = a% b
print (c) # c = 4
```

Возведение в степень

```
a = 2
b = 5
c = a ** b
print (c) # c = 32
```

Целочисленное деление

```
a = 24
b = 5
c = a // b
print (c) # c = 4
```

Умножение

```
a = 24
b = 5
c = a * b
print (c) # c = 120
```

Условные операторы

Знаки сравнения

> Больше
< Меньше
>= Больше или равно
<= Меньше или равно
== Равно
!= Не равно

Операторы условия, такие как if-else, позволяют вам проверять определенные условия и выполнять соответствующие действия

```
# Пример условного оператора if
x = 5
if x > 0:
   print ("x is positive")
```

```
# Пример условного оператора if-else
y = -3
if y > 0:
   print (“y is positive”)
else:
  print (“y is negative”)
```

```
# Пример условного оператора if-elif-else
z = 0
if z > 0:
   print (“z is positive”)
elif z < 0:
   print (“z is negative”)
else:
   print (“z is zero”)
```

Конструкции and, or и not являются логическими операторами в Python и могут использоваться в условных операторах для комбинирования условий.

and (и) — возвращает True, если оба операнда являются истинными (True), иначе — False.

or (или) — возвращает True, если хотя бы один операнд является истинным (True), иначе — False

not (не) — возвращает True, если операнд ложный (False), иначе — False

Пример использования and:

```
x = 5
y = 10
if x > 0 and y > 0:
   print (“Обе переменные положительны”)
else:
   print (“Одна или обе переменные отрицательны”)
```

Пример использования or:

```
x = 5
y = -10
if x > 0 or y > 0:
   print (“Хотя бы одно число положительное”)
```

Пример использования not:

```
x = 5
if not x < 0:
   print (“Число не отрицательное”)
```

Приоритет выполнения операций:

- **Операции сравнения: <, <=, >, >=**
- **not**
- **and**
- **or**

Циклы for и while

В Python есть два основных типа циклов: for и while.

Цикл for используется для итерации по последовательности (список, кортеж, строка и т.д.) и выполняет блок кода для каждого элемента последовательности.

Синтаксис цикла for:

```
for variable in sequence:
# блок кода, который нужно выполнить
```

где **variable** — переменная, которая будет использоваться для хранения текущего элемента последовательности на каждой итерации цикла

sequence — последовательность, по которой будет проходить итерация

Пример:

```
fruits = [“apple”, “banana”, “cherry”]
for x in fruits:
   print (x)
```

Вывод:
apple
banana
cherry

Цикл while выполняет блок кода, пока условие остается истинным.

Синтаксис цикла while:

```
while condition:
# блок кода, который нужно выполнить
```

где **condition** — условие, которое должно быть выполнено для продолжения выполнения цикла.

Пример:

```
i = 1
while i < 6:
   print (i)
i += 1
```

Вывод
1
2
3
4
5

Цикл for с использованием функции range () используется для выполнения определенного блока кода определенное количество раз.

Синтаксис цикла for с функцией range ():

```
for i in range (start, stop, step):
# блок кода, который нужно выполнить
```

где:
start — начальное значение диапазона (по умолчанию 0)
stop — конечное значение диапазона (исключая это значение)
step — шаг изменения значения (по умолчанию 1)

Пример:

```
for i in range (1, 6):
   print (i)
```

Вывод
1
2
3
4
5

В этом примере цикл for проходит от 1 до 5 (не включительно), прибавляя 1 к каждому значению в диапазоне. В теле цикла выполняется команда print (), которая выводит значение i на экран.

Функция range () также может быть использована для создания списков или других последовательностей с определенными значениями.

Например, следующий код создает список, содержащий числа от 1 до 5:

```
my_list = list (range (1, 6))
   print (my_list)
```

Вывод
[1, 2, 3, 4, 5]

Списки

Списки в Python — это упорядоченные коллекции элементов, которые могут быть любого типа данных, включая другие списки. Списки могут быть изменяемыми, то есть вы можете добавлять, удалять и изменять элементы в списке.

Создание списка:

```
my_list = [1, 2, 3, 'apple', 'banana']
```

В этом примере создается список, содержащий пять элементов: три числа и две строки.

Доступ к элементам списка (начинается с 0):

```
print (my_list [0]) # выводит первый элемент списка
print (my_list [-1]) # выводит последний элемент списка
```

Изменение элементов списка:

```
my_list [0] = 4 # заменяет первый элемент списка на число 4
```

Добавление элементов в список:

```
my_list.append ('cherry') # добавляет строку 'cherry' в конец списка
my_list.insert (1, 'orange') # добавляет строку 'orange' на вторую позицию
списка
```

Удаление элементов из списка:

```
my_list.remove (2) # удаляет элемент со значением 2 из списка
del my_list [0] # удаляет первый элемент списка
my_list.pop () # удаляет последний элемент списка и возвращает его значение
```

Длина списка:

```
print (len (my_list)) # выводит количество элементов в списке
```

Срезы списка (позволяют получить подсписок из списка):

```
print (my_list [1:4]) # выводит второй, третий и четвертый элементы списка
print (my_list [:3]) # выводит первые три элемента списка
print (my_list [2:]) # выводит все элементы списка, начиная с третьего
```

Методы списков в Python позволяют работать с ними и изменять их содержимое. Списки являются одним из наиболее универсальных типов данных в Python и используются во многих приложениях, от обработки данных до интерфейсов пользовательского ввода.

Строки

Строки в Python — это упорядоченные коллекции символов, которые могут быть использованы для хранения текстовых данных. Строки в Python могут быть записаны в кавычках одинарных или двойных.

Примеры создания строк:

```
my_string = 'Hello, World!'
my_string2 = "Привет, Мир!"
```

Строки могут быть конкатенированы (соединены) с помощью оператора +:

```
greeting = 'Hello'
name = 'John'
message = greeting + ' ' + name
   print (message)
```

Вывод
Hello John

Доступ к символам в строке осуществляется по индексу (начинается с 0):

```
my_string = ‘Hello, World!’
   print (my_string [0]) # H
   print (my_string [7]) # W
```

Строки в Python являются **неизменяемыми,** что означает, что нельзя изменить символы в строке напрямую. Но можно создать новую строку, используя операции над строками:

```
my_string = ‘Hello, World!’
new_string = my_string [:5] + ‘Python!’
   print (new_string)
```

Вывод
HelloPython!

Методы строк позволяют выполнять множество операций над строками, таких как замена, поиск, разбиение и слияние. Например:

```
my_string = ‘Hello, World!’
   print (my_string.upper ()) # HELLO, WORLD!
   print (my_string.replace (‘Hello’, ‘Hi’)) # Hi, World!
```

Строки в Python используются в различных приложениях, включая обработку текстовых данных, работу с файлами и пользовательский ввод.