Основные функции и операторы Python

Функция print

Формат вызова:

```
print(value, …, sep=' ', end='\n', file=sys.stdout, flush=False)
```

выводит в файл file значение value, добавляя в конце вывода строку end элементы value разделены строкой sep. Если flush=True, тогда после выполнения команды посылается команда очистки буферов ввода/вывода.

value может быть любым объектом python

чаще всего эта функция используется для вывода строковых сообщений.

форматрирование строк для того, чтобы вывести форматированную строку на экран, нужно использовать строку с символами форматирования:

%s — подстановка строки

%d — подстановка целого числа

%f — подстановка числа с плавающей точкой

Подстановочные аргументы передаются в строку форматирования с помощью оператора %, за которым следует кортеж с постановочными аргументами.

```
str_tmp = "2+3=%4d"%(5)
print("2+3=%d"%(5))

>>2+3=5


print(str_tmp)
>> 2+3=   5
```

Функция input

Формат вызова:

```
input(prompt=None, /)
```

Читает строку со стандартного ввода. Символ перевода строки опускается.

Если prompt указан, то он выводится в стандартный вывод без символа перевода строки.

Если пользователь послал сигнал EOF (*nix: Ctrl-D, Windows: Ctrl-Z-Return), вызывает исключение EOFError. На *nix системах используется библиотека readline, если таковая установлена.

Оператор присваивания

Оператор присваивания в Python, как и во многих других языках программирования это =. Поскольку все в Python объекты, операция присваивания копирует ссылку на объект. Это так в случае изменяемых объектов (array, bytearray, list, dict, set), однако для неизменяемых, таких как int, float, complex, str, bytes, tuple, frozenset, bool, происходит создание нового объекта.

Циклы

В питоне выделяют два циклических выражения: for и while.

While loop Выражение while или цикл «пока» имеет следующий вид:

```
while "logical expression":
    suite
else:
    suite
```

Цикл выполняется, пока logical expression истинно, если условие нарушается, выполняется блок else и осуществляется выход из цикла

Пример:

```
i=0
while i<10:
    print(i)
    i+=2
else:
    print("I>10")
```

```
# Вывод следующий
0
2
4
6
8
I>10
```

For loop В питоне цикл for используется для прохода всех элементов в последовательности (строка, список, кортеж) или другого итерируемого объекта.

```
for "target_list" in "expression_list":
    suite
```

```
else:
    suite
```

expression_list вычисляется один раз; оно должно вернуть итерируемый объект. Suite выполняется каждый раз для каждого элемента из итератора. Каждый элемент итератора в свою очередь присваивается target_list и затем выполняется suite.

Когда элементы итератора исчерпываются (когда последовательность заканчивается или итератор вызывает StopException исключение), выполняется suite из ветки else и цикл завершается.

Если в теле цикла вызывается break, она завершает цикл, без выполнения ветки else. continue в теле цикла пропускает оставшуюся часть кода до новой итерации или до ветки else, если новой итерации нет.

Цикл for присваивает значения переменным из target_list. Это действие переписывает все предыдущие присваивания переменным, включае те, что были сделаны в теле цикла.

```
for i in range(10):
    print(i)
    i = 5             # не нарушает ход выполнения цикла
                      # так как  i  будет переписана на следующей итерации цикла
```

имена из target_list не удаляются по завершении цикла, но если итерируемая последовательность пуста, они не будут инициализированы.

функция range() возвращает итератор, с помощью которого можно с эмулировать работу цикла for в паскале. list(range(5))=[1,2,3,4].

Если мы итерируем по mutable объекту и нам нужно удалять или вставлять туда элементы, то цикл вида:

```
for i in x:
    if i<0:
        x.remove(i)
```

будет выполняться неверно, поскольку при удалении из списка его размер уменьшится, и в позиции, куда указывает итератор, будет стоять следующий элемент. На следующем шаге позиция итератора снова сдвинется, приведя к тому, что один элемент будет пропущен.

То же касается и вставки.

Выход из решения — создать временную копию списка, например с помощью сечения.

```
for i in x[:]:
    if i<0:
        x.remove(i)
```

Здесь мы итерировать будем копию списка, а удалять элементы из оригинала.