Циклы For и While

В этом разделе мы познакомимся с основной парадигмой хода выполнения цикла “for” и введем формальное определение “итерируемого объекта”. Практическую полезность этих понятий трудно переоценить, ведь в дальнейшем вы скорее всего обнаружите, что применяете их едва ли не в каждом фрагменте написанного вами кода на Python!

Примечание:

В тексте встречаются задания для проверки усвоения материала. С их помощью можно попробовать сразу применить прочитанное на практике. Ответы на задания приведены в конце раздела.

Цикл For

Цикл “for” позволяет перебирать элементы коллекции один за другим, каждый раз выполняя блок кода. Например, нижеприведенный код суммирует все положительные элементы кортежа:

```
total = 0
for num in (-22.0, 3.5, 8.1, -10, 0.5):
    if num > 0:
        total = total + num
```

Общая схема синтаксиса цикла “for” выглядит следующим образом:

```
for <var> in <iterable>:
    блок кода
```

Где `<var>` — это идентификатор переменной, а `<iterable>` — любой **итерируемый (перебираемый) объект**. Мы рассмотрим итерируемые объекты подробнее в следующем разделе; пока же достаточно уяснить, что любой объект, представляющий собой какого-либо рода последовательность, является итерируемым. В конце инструкции `for` обязательно ставится двоеточие, а тело цикла for выделяется отступом.

Вот как работает цикл for:

1. Пробуем получить следующий элемент итерируемого объекта.
2. Если итерируемый объект не содержит в себе элементов (пустой), происходит выход из цикла for без выполнения его тела.

3. Если элемент успешно получен, присваиваем его переменной `<var>` (если `<var>` не была задана ранее, она задаётся в этот момент).
4. Выполняется код тела цикла.
5. Возвращаемся к первому шагу.

Рассмотрим на конкретном примере:

```
# демонстрация простейшего цикла for
total = 0
for item in [1, 3, 5]:
    total = total + item

print(total)  # `total` в итоге 1 + 3 + 5 = 9
# `item` на этом этапе все еще существует и хранит значение 5
```

Этот код пошагово выполняет следующие действия:

1. Задает переменную `total` и присваивает ей значение `0`
2. Начиная перебор списка, получает значение `1`, задает переменную `item` и присваивает ей полученное значение первого элемента списка — `1`
3. Присваивает переменной `total` значение `0 + 1`
4. Продолжая перебор списка, получает значение `3` и присваивает переменной `item` теперь его
5. Присваивает переменной `total` значение `1 + 3`
6. Продолжая перебор списка, получает значение `5` и присваивает его переменной `item`
7. Присваивает переменной `total` значение `4 + 5`
8. Продолжает перебор списка, который, как только достигнут конец, подает сигнал `StopIteration` (прекратить перебор), и происходит выход из цикла.
9. Печатает накопленное значение переменной `total` `(9)`

Потенциальная ловушка

Обратите внимание, что переменная `item` продолжает существовать после выхода из цикла for. Она будет содержать указатель на последнее значение, полученное при переборе элементов итерируемого объекта в цикле for (в нашем примере `item` хранит значение `5`). Тем не менее, *не следует писать код, использующий переменную перебора, за пределами цикла for*. Дело в том, что если итерируемое окажется пустым, эта переменная так и не будет задана:

```
for x in []:          # пустое итерируемое – переменная перебора `x` остается незаданной
    print("Hello?")   # эта строка не выполняется
print(x)              # возникает ошибка, так как `x` не была задана
```

При пустом списке сигнал `StopIteration` подается сразу же, еще до начала перебора – и переменная `x` не успевает инициализироваться. Таким образом, до выполнения кода в теле цикла for дело не доходит, а команда `print(x)` приводит к возникновению ошибки `NameError`, так как `x` осталась незаданной!

Контрольное задание: Простейший цикл for

Используя цикл for и условный оператор if, выведите в консоль (print) все гласные из строки `"abcdefghij"` по одной.

Цикл While

Цикл “while” служит для повторного выполнения блока кода до тех пор, пока верно условие:

```
while <condition>:
    block of code
```

Где `<condition>` — это выражение, возвращающее `True` или `False`, или любой объект, к которому применима функция `bool`. “Телом” цикла while является сдвинутый вправо код, идущий вслед за самой инструкцией while.

Цикл while работает следующим образом:

1. Вызывается функция `bool(<condition>)` и, если она возвращает `True`, выполняется сдвинутый вправо блок кода. В противном случае происходит “выход” из цикла без выполнения его тела.
2. Если выполнялся код в теле цикла, по завершении снова начинаем с первого шага.

Рассмотрим конкретный пример:

```
# демонстрация простейшего цикла while
total = 0
while total < 2:
    total += 1  # эквивалент `total = total + 1`

print(total)  # `total` имеет значение 2
```

Этот код пошагово выполняет следующие действия:

1. Задается переменная `total` и ей присваивается значение `0`
2. Проверяется выражение `0 < 2`, возвращающее `True`: переходим к выполнению кода, заключенного в теле цикла
3. Выполняется блок кода: переменной `total` присваивается значение `0 + 1`
4. Проверяется выражение `1 < 2`, возвращающее `True`: переходим к выполнению кода, заключенного в теле цикла
5. Выполняется блок кода: переменной `total` присваивается значение `1 + 1`
6. Проверяется выражение `2 < 2`, возвращающее `False`: код, заключенный в теле цикла, *пропускается*
7. Выводится значение переменной `total` `(2)`

Обратите внимание, что если начать с `total = 3`, условное выражение `3 < 2` сразу же выдаст `False` , и код в теле цикла не выполнится ни разу.

Осторожно!

Цикл while можно написать таким образом, что условное выражение окажется всегда True – выполнение вашего кода в таком случае будет повторяться бесконечно! Если это случится при работе в Jupyter notebook, придется остановить или перезапустить ядро.

Контрольное задание: Простейший цикл while

Дан список положительных чисел `x`. Добавляйте сумму всех чисел из списка в его конец до тех пор, пока последний элемент в списке `x` не окажется равен 100 или больше. Используйте цикл while.

Если начать со списка из одного элемента `x = [1]`, к моменту завершения цикла while `x` должен превратиться в `[1, 1, 2, 4, 8, 16, 32, 64, 128]`.

Команды `break`, `continue` и `else` в циклах[3]

Команды `continue` и `break` могут использоваться в теле как циклов for, так и циклов while. Они обеспечивают дополнительную возможность для “перескока” к следующей итерации или выхода из выполняемого цикла, не дожидаясь завершения перебора, соответственно.

Попадание на `break` в теле выполняемого цикла приводит к немедленному выходу из него:

```
# выход из цикла до завершения перебора с помощью команды break
>>> for item in [1, 2, 3, 4, 5]:
...     if item == 3:
...         print(item, " ...break!")
...         break
...     print(item, " ...следующая итерация")

1  ...следующая итерация
2  ...следующая итерация
3  ...break!
```

В конец любого цикла можно добавить инструкцию `else`. Код в ее теле выполнится *только в том случае, если цикл не был прерван командой ``break``.*

```
# инструкция else в конце цикла
>>> for item in [2, 4, 6]:
...     if item == 3:
...         print(item, " ...break!")
...         break
...     print(item, " ...следующая итерация")
... else:
...     print("Если вы это читаете, значит цикл завершился без 'break'")

2  ...следующая итерация
4  ...следующая итерация
6  ...следующая итерация
Если вы это читаете, значит цикл завершился без 'break'
```

Команда `continue` в теле цикла приводит к немедленному возврату в начало (к следующей итерации).

```
# демонстрация команды `continue` в теле цикла
>>> x = 1
>>> while x < 4:
...     print("x = ", x, ">> входим в тело цикла <<")
...     if x == 2:
...         print("x = ", x, " continue...возврат в начало!")
...         x += 1
...         continue
...     x += 1
...     print("--достигнут конец тела цикла--")

x =  1 >> входим в тело цикла <<
--достигнут конец тела цикла--
x =  2 >> входим в тело цикла <<
x =  2  continue...возврат в начало!
x =  3 >> входим в тело цикла <<
--достигнут конец тела цикла--
```

Контрольное задание: управление ходом выполнения программы внутри цикла

Переберите список целых чисел, суммируя все четные и сохраняя накопленное значение до тех пор, пока оно не превысит 100 или цикл не выполнится более 50 раз. Распечатайте сумму только в том случае, если она больше 100.

Ссылки на официальную документацию

- инструкция ‘for’
- инструкция ‘while’
- команды ‘break’, ‘continue’ и ‘else’
- команда ‘pass’

Решения контрольных заданий:

Простейший цикл for: Решение

```
for letter in "abcdefghij":
    if letter in "aeiou":
        print(letter)
```

Простейший цикл while: Решение

```
while x[-1] < 100:
    x.append(sum(x))
```

Управление ходом выполнения программы в цикле: Решение

```
x = [3, 4, 1, 2, 8, 10, -3, 0]
num_loop = 0
total = 0

while total < 100:
    for item in x:
        # возврат в начало цикла for если
        # в `item` -- нечетное число
        if item % 2 == 1:
            continue
        else:
            total += item
    num_loop += 1

    # выход из цикла while если
    # перебрано более 50 значений
    if 50 < num_loop:
        break
else:
    print(total)
```