***Python for Informatics***
***Автор:*** Dr. Charles R. Severance
***Содержание курса:*** http://pycode.ru/edu/why-python/

# Глава 5. Итерации

## 5.1. Обновление переменной

Общим шаблоном в инструкциях присваивания является инструкция присваивания, которая обновляет переменную, где новое значение переменной зависит от старого:

```
x = x+1
```

Это означает «получить текущее значение *x*, прибавить к нему 1 и затем обновить *x*, присвоив ему новое значение».

Если вы попытаетесь обновить переменную, которая не существует, то получите ошибку, т.к. Python вычисляет правую сторону раньше ее присваивания переменной *x*:

```
>>> x = x+1
NameError: name 'x' is not defined
```

Перед тем как обновить переменную, вам необходимо ее инициализировать (initialize), обычно с помощью простого присваивания:

```
>>> x = 0
>>> x = x+1
```

Обновление переменной, путем прибавления к ней 1, называется инкрементом (increment), вычитание 1, называется декрементом (decrement).

## 5.2. Инструкция while

Компьютеры часто используются для автоматизации повторяющихся задач. Повторение сходных или простых задач без ошибок – это то, что компьютер делает лучше, чем человек. Поскольку итерации распространены, Python предоставляет несколько языковых особенностей для их создания.

Одной из форм итераций в Python является инструкция *while*. Далее представлена простая программа, которая производит обратный отсчет от 5 и затем говорит «*Blastoff!*».

```
n = 5

while n > 0:

     print n

     n = n-1

print 'Blastoff!'
```

Вы также можете читать инструкцию *while*, как если бы это был английский язык. Это означает, «до тех пор, пока *n* больше 0, выводить на экран значение *n* и уменьшать *n* на 1. Когда достигнем 0, покинуть инструкцию *while* и вывести на экран слово *Blastoff*!»

Более формально, поток выполнения для инструкции *while* имеет следующий вид:

1. Вычисление условия, получаем *True* или *False*.
2. Если условие ложно, то выходим из инструкции *while* и продолжаем выполнение со следующей инструкции.
3. Если условие истинно, то выполняем тело и затем возвращаемся на шаг 1.

Этот тип потока называется циклом (loop), т.к. третий шаг цикла возвращается на начало. Выполнение тела цикла называется итерацией (iteration). Для приведенного выше цикла, мы бы сказали, что «прошло 5 итераций», т.е. тело цикла было выполнено 5 раз.

Тело цикла должно изменять одну или более переменных, что в конечном итоге приведет к ложности условия и завершению цикла. Переменные, которые изменяются каждый раз при выполнении цикла и контролируют завершение цикла, называются итерационными переменными (iteration variable). Если итерационная переменная в цикле отсутствует, то такой цикл будет бесконечным (infinite loop).

### 5.3. Бесконечные циклы

Бесконечным источником развлечения для программистов является высказывание на руководстве по шампуню: «Намылить, промыть, повторить». Это бесконечный цикл, поскольку в нем отсутствует итерационная переменная, которая указывает на количество выполнений в цикле.

В случае с обратным отсчетом, мы можем удостовериться, что цикл завершится, т.к. мы знаем, что значение *n* конечно, мы можем увидеть, что значение *n* каждый раз уменьшается и в конечном итоге станет равным нулю.

### 5.4. «Бесконечные циклы» и break

Иногда вы не знаете, когда завершить цикл, пока не достигните некоторого значения в теле. В этом случае, вы можете специально написать бесконечный цикл и затем использовать инструкцию *break*, чтобы из него выйти.

Это явно бесконечный цикл, т.к. логическое выражение в инструкции *while* содержит *True*:

```
n = 10

while True:

     print n,

     n = n - 1

print 'Done!'
```

Если вы совершите ошибку и запустите этот код, то быстро научитесь, как останавливать процесс Python в вашей системе или найдете кнопку выключения питания на вашем компьютере (что делать не желательно). Эта программа работает постоянно или пока не разрядится батарея, т.к. логическое выражение всегда будет истинным.

Мы можем добавить в тело цикла код с *break*, который позволит выйти из цикла при наступлении заданного условия.

Например, предположим, что мы принимаем входные значения с клавиатуры, пока пользователь не наберет «*done*». Можем записать это следующим образом:

```
while True:

     line = raw_input('> ')

     if line == 'done':

          break

     print line

print 'Done!'
```

Условие цикла является *True*, т.е. всегда истинным. Оно будет выполняться, пока на достигнет инструкции прерывания.

### 5.5. Завершение итерации с помощью continue

Иногда вы находитесь в итерации цикла и хотите завершить текущую итерацию и сразу перейти к следующей итерации. В этом случае можно воспользоваться инструкцией *continue*, которая пропускает текущую итерацию и переходит к следующей без завершения тела цикла.

Далее представлен пример цикла, который копирует данные, поступающие на его вход до тех пор, пока не будет введено «*done*». Если строка начинается с символа #, то она не будет выводиться на печать (похоже на комментарии в Python).

```
while True:
     line = raw_input('> ')
     if line[0] == '#' :
          continue
     if line == 'done':
          break
     print line
print 'Done!'
```

Все строки, которые вводятся с клавиатуры, будут выведены на экран, кроме тех, которые начинаются с символа #.

### 5.6. Определение циклов с помощью for

Иногда мы хотим пройти в цикле через множество (set) вещей, таких как список слов, строки в файле или список чисел. Когда у нас есть подобный список, мы можем построить определенный (definite) цикл с использованием инструкции *for*. Мы вызывали инструкцию *while* в неопределенных (indefinite) циклах, т.к. циклы работали до того момента, пока условие не станет *False*. Цикл *for* проходит через известное множество элементов, т.е. столько раз, сколько элементов содержится во множестве.

Синтаксис цикла *for* похож на цикл *while*, здесь есть инструкция *for* и тело цикла:

```
friends = ['Joseph', 'Glenn', 'Sally']
```

```
for friend in friends:
     print 'Happy New Year:', friend
print 'Done!'
```

Переменная *friends* является списком (в следующих главах вы об этом узнаете подробнее) из трех строк, цикл *for* проходит через список и выполняет тело цикла для каждой из трех строк:

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!
```

Посмотрите на цикл *for*, здесь *for* и *in* являются зарезервированными словами, *friend* и *friends* являются переменными.

```
for friend in friends:
     print 'Happy New Year', friend
```

В частности, *friend* является итерационной переменной для цикла *for*. Переменная *friend* изменяется для каждой итерации цикла и контролирует, когда цикл *for* завершится.

### 5.7. Шаблоны цикла

Часто мы используем циклы *for* и *while* для того, чтобы протий в цикле через элементы списка или содержимое файла в поисках наибольшего или наименьшего значения.

Эти циклы обычно строятся по следующей схеме:

- Инициализация одной или более переменных до начала выполнения цикла.
- Выполнение некоторых вычислений для каждого элемента в теле цикла, возможно, изменение переменных в теле цикла.
- Просмотр результирующих переменных, когда цикл завершился.

Воспользуемся списком чисел для демонстрации подходов и создания распространенных шаблонов.

### 5.7.1. Циклы подсчета

Например, для подсчета числа элементов в списке, мы можем написать следующий цикл *for*:

```
count = 0

for itervar in [3, 41, 12, 9, 74, 15]:

     count = count + 1

print 'Count: ', count
```

Мы установили начальное значение для переменной *count* равным нулю перед тем, как начать выполнение цикла. Наша итерационная переменная называется *itervar*.

В теле цикла мы добавляем 1 к текущему значению *count* для каждого значения в списке. К моменту завершения цикла, значение переменной *count* будет содержать количество элементов.

Другой простой цикл подсчитывает сумму набора чисел:

```
total = 0

for itervar in [3, 41, 12, 9, 74, 15]:

     total = total + itervar

print 'Total: ', total
```

Переменная *total* накапливает сумму всех чисел, иногда ее называют сумматором (accumulator).

На практике для подсчета числа элементов используют встроенную функцию *len()*, а для вычисления суммы элементов – *sum()*.

### 5.7.2. Циклы максимума и минимума

Для нахождения наибольшего значения в списке или последовательности, создадим следующий цикл:

```
largest = None

print 'Before:', largest

for itervar in [3, 41, 12, 9, 74, 15]:

     if largest is None or itervar > largest :

          largest = itervar
```

```
     print 'Loop:', itervar, largest
print 'Largest:', largest
```

Результат работы программы будет иметь вид:

```
Before: None
Loop: 3 3
Loop: 41 41
Loop: 12 41
Loop: 9 41
Loop: 74 74
Loop: 15 74
Largest: 74
```

Переменная *largest* содержит наибольшее значение, которое существует на данный момент. Перед запуском цикла мы устанавливаем *largest* в *None*. *None* – это специальная константа, которая может храниться в переменной и маркировать ее как «пустую».

Перед началом цикла, наибольшим значением является *None*. На первой итерации цикла значение 3 больше *None*, мы устанавливаем *largest* равным 3. На последней итерации в *largest* содержится наибольшее значение.

Код для вычисления наименьшего значения:

```
smallest = None
print 'Before:', smallest
for itervar in [3, 41, 12, 9, 74, 15]:
     if smallest is None or itervar < smallest:
          smallest = itervar
     print 'Loop:', itervar, smallest
print 'Smallest:', smallest
```

Снова *smallest* содержит наименьшее значение.

Далее следует простая версия встроенной в Python функции min():

```
def min(values):
```

```
    smallest = None
    for value in values:
        if smallest is None or value < smallest:
            smallest = value
return smallest
```

В этой функции меньше кода, мы удалили все инструкции *print*.

### 5.9. Словарь

*Сумматор (accumulator)*: переменная в цикле, которая используется для накопления суммарного результата.

*Счетчик (counter)*: переменная, используемая в цикле для подсчета количества встречаемости какого-либо события. Мы инициализируем счетчик нулевым значением, затем инкрементируем его при наступлении какого-либо события.

*Декремент (decrement)*: уменьшение значения (обычно на единицу).

*Инициализация (initialize)*: присваивание начального значения переменной, которая в дальнейшем будет инкрементироваться (увеличиваться, часто на единицу).

*Бесконечный цикл (infinite loop)*: цикл, в котором условие завершения никогда не наступят или для которого нет условия завершения.

*Итерация (iteration)*: повторное выполнение множества инструкций, используемое при рекурсивном вызове функции или цикла.

### 5.10. Упражнения

1. Приведите пример выполнения инструкции while.
2. Приведите пример выполнения инструкции for.