#5: FOR Loop, Range, Tuples

Reading: Chapter 4
Contents

• Range function
• Tuples
• FOR loop
• Examples Break and Continue in FOR loop
Example

```python
while True:
    s = input('Enter something : ')
    if s == 'quit':
        break
```

Notes:

1) `while True:`
   # short for `while True == True:
   # becomes `while True == True`

Also a similar statement is `while 1:`

2) In Python, we have two **Boolean** variables (predefined) - `True` and `False`. But Python will also regard as `False` - the number zero (0), an empty string (""), or the reserved word `None`. All other values are interpreted as `True`.

3) Watch for a `if` with a `break` statement inside such `while` statement. If such statement does not exist, you have an infinite loop!
Range function

Syntax:

range(fromNumber, toNumber, step)

or

range(fromNumber, toNumber) # step = 1

or

range(toNumber) # fromNumber = 0, step = 1

The function returns a sequence of numbers starting from “fromNumber” to “toNumber” counting them with the given “step”.
Range function

range(5, 15, 3)
5, 8, 11, 14
range(1, 5) # step = 1
1, 2, 3, 4
range(5) # fromNumber = 0, step = 1
0, 1, 2, 3, 4 # note that it starts from 0, has 5 numbers, but does not include 5
range(25, 5, -6) # countdown
25, 19, 13, 7

The range function auto-builds a list of integers for you. Typically it's used to create indexes for a for statement but you can use it anywhere.
Tuples

- A tuple is a sequence of immutable (cannot be changed) Python objects.

- Creating tuples
  
  ```python
tupleName = (object1, object2, object3)
  ```

  Examples:

  ```python
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5)
tup3 = "a", "b", "c", "d"
  ```

- The empty tuple: `tup1 = ()`

- Tuple containing a single value `tup2 = (11,)`
Tuples

Accessing values in a tuple:

tupleName[index]

tup1 = ('physics', 'chemistry', 1997, 2000)
print tup1[0]
print tup1[1]
print tup1[2]
print tup1[3]
physics
chemistry
1997
2000
Tuples - index

tup1 = ('physics', 'chemistry', 1997, 2000)

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negative</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

Positive index always starts from the first element with index = 0

Negative index starts from the last element with index = -1
Tuples operations

Concatenation:

tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')
tup3 = tup1 + tup2
print tup3
(12, 34.56, 'abc', 'xyz')

Length: len(tuple)  # len(tup2) = 2, len(tup3) = 4
Gives the total length of the tuple.

max(tuple)  # max(tup2) = ‘xyz’, max(tup3) = ‘xyz’
Returns item from the tuple with max value.

min(tuple)  # min(tup2) = 12
Returns item from the tuple with min value.
Tuples operations

In operator:

```python
tup = (1,2,3,4,5)
3 in tup
True
```

Slicing tuples: `tup[firstPosition:lastPosition]`
- Results in a new tuple containing every element between these two positions

```python
tup[1:3]
(2,3)
```
FOR loop

A loop construct in Python allows you to repeat a block of code several times.

- **For** loop - when you have a piece of code which you want to repeat a certain (known) number of times

- **For** repeats a block of statements for each element of a sequence in order; while repeats a block of statements based on a condition
FOR loop

Syntax:

```python
for iteratingVariable in sequence:
    loop_body
```

Examples:

```python
for letter in 'Python':  # First Example
    print ('Current Letter :', letter)
```

Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
FOR loop

fruits = ('banana', 'apple', 'mango')
for fruit in fruits:  # Second Example
    print ('Current fruit :', fruit)
print ("Good bye!")

Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
FOR loop

fruits = ('banana', 'apple', 'mango')
for index in range(len(fruits)):  # Third Example
    print ('Current fruit : ', fruits[index])
print ('Good bye!')

Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
FOR loop

Example of counter up:

```py
for counter in range(10):
    print(counter)
print("Final counter after loop: ", counter)
```

0 1 2 3 4 5 6 7 8 9

Final counter after loop: 9
FOR loop

Counting down:

```python
for counter in range(10, 1, -1):
    print(counter)

print("And now the loop has ended.")
```

10 9 8 7 6 5 4 3 2

And now the loop has ended
FOR loop

Counting down with step of 4:
for counter in range(20,1,-4):
    print(counter)
print("And now the loop has ended.")
20
16
12
8
4
And now the loop has ended
Break Statement

**break** statement in Python terminates the current loop and resumes execution at the next statement after the loop.

Can be used inside WHILE and FOR loops.

It is included in the body of an if statement, when you want to skip the execution of `loop_body` from a certain case/condition on.
Break Statement

for letter in 'Python':
    if letter == 'h':
        break
    print ('Current Letter :', letter)

Current Letter : P
Current Letter : y
Current Letter : t
Continue Statement

The `continue` statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

The `continue` statement can be used in both `while` and `for` loops.

It is included in the body of an if statement, when you want to skip the execution of `loop_body` for a certain case/condition.
Continue Statement

for letter in 'Python':
    if letter == 'h':
        continue
    print ('Current Letter :', letter)

Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Next lecture ...

- Strings
- Lists