Introduction to Repetition Structures: while and for loops

Beste Filiz Yuksel

Repetition Structures

A repetition structure results in a statement or set of statements to execute repeatedly

- more commonly known as a *loop*.

Two Kinds of Loops in Python

Condition-Controlled Loops - while loops

Repeat while a specified condition is true

Count-Controlled Loops - for loops

Repeat a *pre*-specified number of times

Condition-controlled loops - `while` loops

Statement(s) repeated *while condition is true*

```
while boolean condition:
    statement
    statement
    etc
```

If the boolean condition is false, program exits the loop

Condition-controlled loops - `while` loops

Let’s trace through together

```
n=0
while n<3:
   print(n)
   n+=1
```

If the boolean condition is false, program exits the loop

Beware of the infinite loop!

In most cases, loops must find a way to terminate.

Infinite loops occur when the programmer forgets to write code to make the boolean condition false.

```
n=1
while n>0:
    print(n)
    n+=1
```

Try this in a script. Now try it with the print statement outside of the loop.

While loop example

Suppose you want to make a cup of Oolong Tea where the perfect water temperature is 88℃. You want to simulate the water cooling down and print each temperature until it reaches the perfect Oolong Tea temperature.

```
temperature =100
while temperature > 88:
   print(temperature)
   temperature-=1
print('Your water is ready for your Oolong Tea!')
```

Is temperature equal to 88 or 89 when the statement is printed? Why? How can we check it? What would happen if `temperature >= 88`

Count-controlled or `for` loop

A count-controlled loop iterates a specific number of times.

for *variable* in *[value1, value2, value3]*:

    statement

    statement

    etc.

The *variable* (also called the target variable) is assigned each item in the list [*value1, value2, value3*]

for loop example with list

```
print(‘Print the numbers 1 through 5’)
for num in [1, 2, 3, 4, 5]:
   print(num)
```

iteration #	num
1	
2	
3	
4	
5	

for loop example with list

```
print(‘Print the odd numbers 1 through 9’)
for num in [1, 3, 5, 7, 9]:
   print(num)
```

iteration #	num
1	
2	
3	
4	
5	

for loop with String

for *variable* in “hello”:

    statement

```
print("The alphabet")
for letter in "abcedfghijklmnopqrstuvwxyz":
  print(letter)
```

for loop with range() function

The `range()` function creates a type of object, iterable, that contains a sequence of values that can be iterated over with a loop.

for *variable* in range(***end+1***)

    statement

```
print('Print the numbers 0 through 4')
for num in range(5):
    print(num)
```

iteration #	num
1	
2	
3	
4	
5	

for loop with range() function

It’s also useful if you want to repeat something n times

e.g.

```
print(‘Print this message 5 times’)
for x in range(5):
   print(‘Hello world!’)
```

iteration #	x	print
1		
2		
3		
4		
5		

for loop with range() function

for *variable* in range(***start***, *end+1*)

    statement

```
print('Print the numbers 2 through 6')
for num in range(2,7):
    print(num)
```

iteration #	num	print
1		
2		
3		
4		
5		

for loop with range() function

for *variable* in range(*start, end+1,* ***increment value***)

    statement

```
print('Print odd numbers between 1 and 10')
for num in range(1, 10, 2):
   print(num)
```

for loop with range() function

for *variable* in range(*start, end+1,* ***decrement value***)

    statement

```
print('Count down from 10 to 1 by decrements of 1')
for num in range(10, 0, -1):
   print(num)
```

for loop with range() function

for *variable* in range(*start, end+1, increment value*)

    statement

start defaults to 0 if omitted

increment value defaults to 1 if omitted

for Loop with range() function

Good for generating tables, e.g.:

```
n=11
print("n", "\t", "x**2")     # print column headers
print("---", "\t", "-----") # print header border
for x in range(1, n):        # generate values for columns
   print(x, "\t", x**2)
```

Accumulators - or calculating a running total

A running total is a sum of numbers that accumulates with each iteration of a loop. e.g.

```
print('Here is the sum of numbers from 1 through 5')
total=0 #accumulator
for n in range(1, 6):
   total+=n
print('Total is: ', total)
```

Why is the variable total initialized to 0?

```
print('Here is the sum of numbers from 1 through 5')
total=0
for n in range(1, 6):
    total+=n
print('Total is: ', total)
```

Iteration #	n	total
initialization		
1		
2		
3		
4		
5		

More examples of back end:
http://interactivepython.org/runestone/static/thinkcspy/MoreAboutIteration/Theforlooprevisited.html

And let us revisit the while loops and look at the backend:
http://interactivepython.org/runestone/static/thinkcspy/MoreAboutIteration/ThewhileStatement.html