Python, Day 4.5: Loops

Andrew Bydlon

January 10, 2019

How loops work

Often times running a command iteratively is useful. Instead of writing a command over and over, we use **loops**.

The idea of a loop is fairly simple. While a condition is satisfied (True), it performs a given action.

There are two types of loops:

- While loops.
- For loops.

Continuing with while loop logic

Syntax (While loops)

```
while BooleanStatement:
    command1
    command2
    ...
     commandN
     SomeCommandToAlterBooleanStatement
commandNew   # command executing outside the loop
```

Note that when the program reaches a while loop, it proceeds as follows:

1. If BooleanStatement == True, continue. Otherwise, exit loop.
2. Execute command1, command2, . . . , commandN.
3. Execute SomeCommandToAlterBooleanStatement.
4. Return to step 1.

Example 1: A sum of values

Example

```
i,sum1,sum2 = 1,0,0

N = int(input("Enter the integer you want to sum up to"))

while i<=N:
    sum1 += i
    sum2 += i**2
    i += 1 # This is extremely important.
print("The sum of the first", N, "integers is", sum1)
print("The sum of the square of the first", N, "integers is", sum2)
```

Without the i += 1 command, the loop would execute until the heat death of the universe (or until terminated)!

Example 2: Computing the factorial

Example

```
i,Factorial = 1,1

N = int(input("Enter a positive integer. "))

while i<=N:
    Factorial *= i
    i += 1
print(N, "! = ", Factorial, ".", sep="")
```

Example 3: recursively running a program

Sometimes the user may want to run the same code over and over again. Here is a simple way to do that:

Example

```
UserString = "yes"

while UserString.lower() == "yes":

    YourCodeHere

    UserString = input("Would you like to rerun the code? [yes/no]")

print("Thanks for playing.")
```

The for loop

There are times where you want to be more specific with the range for your loop. In this case a for loop is often ideal.

Syntax (For loops)

```
for MyIndex in MySequence:
    command1   # Runs during the loop.
    command2   # Runs during the loop.
    command3   # Runs during the loop.
command 4   # Runs after the completion of the loop.
```

A brief statement on ‘Sequences’

A **sequence** is a generic statement used to represent a collection of smaller pieces of data.

Some examples are the following:

1. Sets
2. Strings
3. Tuples
4. Lists

Syntax

MySequence = [a,b,c,d,e,f]

Example

Example

```
»> for i in [2,3,5,7,11,13,17,19]:
...     print i

2
3
5
7
11
13
17
19
```

Note that *i* is literally going through the sequence entry by entry.

Similar example

Example

```
»> for i in "Andrew":
...     print i

A
n
d
r
e
w
```

The range() function

The range() function makes it easier to emulate the while loop with a for loop.
Here are some of the applications:

1. One entry yields all natural numbers **less than** the entry:

   range(n) = [0,1,2,. . . , n-1]

2. Two entries yield all numbers between the first entry and the second:

   range(n,m) = [n,n+1,n+2,. . . , m-1]

3. Finally, 3 entries yields the previous step, but with **incrementation**:

   range(n,m,i) = [n,n+i,n+2i,. . . , n+ji]

   Here $m - i \leq n + j \cdot i < m$. For example,

   range(11,40,3) = [11,14,17,20,23,26,29,32,35,38]

Examples

Example (Range)

```
>>> for i in range(2,19,2):
...         print i
2
4
6
8
10
12
14
16
18
```

```
>>> for i in range(100,55,-5):
...         print i
100
95
90
85
80
75
70
65
60
```

Assignment 7

Write a program which takes input as input 3 integers, *N, div1, div2*. Then print all of the integers from 0 to *N*, with the following exceptions:

- For numbers divisible by *div1*, but not *div2*, print "Multiple of first divisor.".
- For numbers divisible by *div2*, but not *div1*, print "Multiple of second divisor."
- For numbers divisible by both *div1* and *div2*, print ":)"

Upload your .py file when you finish.