Introduction to Python

Direct Loops: For Loops

Topics

1) For Loops
2) Break vs. Continue
3) Nested Loops

For Loops

In general, a loop allows a sequence of instructions to execute repeatedly until some condition is met.

Python's *for* loop iterates over items of a sequence(e.g. a list, string or tuple) and process them with some code.

```
for x in sequence:
     block
```

This is a list. More on lists in a later lecture.

```
In[1]:  for x in [2,3,5,7]:
            print(x, end=" ")     # print all on same line
```

```
2 3 5 7
```

For Loops

```
In[1]:  for x in [2,3,5,7]:
            print(x)
```

```
2
3
5
7
```

range(stop)

A simple use of a *for* loop runs some code a specified number of times using the *range()* function.

range(stop): returns sequence of numbers from 0 (default) up to but not including stop. Increment by 1 (default).

```
In[1]: for i in range(10):
           print(i, end=" ")
```

0 1 2 3 4 5 6 7 8 9

range(start, stop)

range(start, stop): from start up to but not including stop. Increment by 1 (default).

```
In[2]: for i in range(2, 8):
           print(i, end=' ‘)
```

2 3 4 5 6 7

range(start, stop, step)

range(start, stop, step): from start up to but not including stop, increment by step.

```
In[3]: for i in range(1, 10, 2):
           print(i, end=' ')
```

1 3 5 7 9

If step is negative, a list can be traversed backwards.

```
In[4]: for i in range(10, 2, -1):
           print(i, end=' ')
```

10 9 8 7 6 5 4 3

continue vs. break

The **continue** statement is used to skip the current iteration and move to the next iteration whereas the **break** statement is used to exit a for loop.

```
In [1]:  for n in range(10):
             if n % 2 == 0:
                 continue
             print(n, end=' ')
```

```
1 3 5 7 9
```

continue vs. break

The **continue** statement is used to skip the current iteration and move to the next iteration whereas the **break** statement is used to exit a loop.

```
In [2]:  for n in range(5):
             if n == 3:
                 break
             print(n, end=' ')
```

```
0 1 2
```

Definite Loop

The for loop is an example of a **definite** loop. We can determine ahead of time the number of times the loop repeats. Later, we will talk about **indefinite loop**, a loop where we cannot predict the number of times it repeats.

```
In [1]:  for i in range(5):
             print("*", end="")
```

```
*****
```

The loop above prints five *'s. We can determine this from the for loop statement.

Summing and Counting

There are two common tasks that uses for loops.

1) Summing
2) Counting

Summing Values

Write a segment of code that solve the problem

$$1 + 2 + 3 + \ldots + 98 + 99 + 100.$$

```
In [1]:    sum = 0
           for i in range(1, 101):
               sum += i
```

Writing a function to sum

Now write a function that accepts a non-negative integer input n and returns the sum of integers from 1 to n(including).

```
In [1]:   def sum(n):
              sum = 0
              for i in range(1, n+1):
                  sum += i
              return sum

In [2]: print(sum(5))   # 1+2+3+4+5=15
15
```

Conditional Summing

Write a segment of code that compute the sum of all numbers from 1 to 100 that are multiples of 3.

```
In [1]:   sum = 0
          for i in range(0, 101, 3):
              sum += i
```

Or equivalently, we can use a conditional to select the numbers to add:

```
In [2]:   sum = 0
          for i in range(1, 101):
              if i % 3 == 0:
                  sum += i
```

Better to use if conditional for filtering. In general, using the step size above might not always work.

Conditional Summing Example

Write a segment of code that compute the sum of all numbers from 1 to 100. However:

1) if a number is a multiple of 3, double it before adding,
2) if a number is a multiple of 5, triple it before adding,
3) If a number is a multiple of both, quadruple it before adding.

Conditional Summing Solution?

Is the following a correct solution?

```
sum = 0
for i in range(1, 101):
      if i % 3 == 0:
            sum += 2 * i
      elif i % 5 == 0:
            sum += 3 * i
      elif i % 3 == 0 and i % 5 == 0:
            sum += 4 * i
      else:
            sum += i
```

No! Why not?

Conditional Summing Solution

The following is correct.

```
sum = 0
for i in range(1, 101):
      if i % 3 == 0 and i % 5 == 0:
            sum += 4 * i
      elif i % 3 == 0:
            sum += 2 * i
      elif i % 5 == 0 :
            sum += 3 * i
      else:
            sum += i
```

Counting

Write a function that accepts an integer n and returns the number of factors of n.

```
In [1]:   def count_factors(n):
              count = 0
              for i in range(1, n+1):
                  if n % i == 0:  # i is a factor of n
                      count += 1
              return count
In [2]: print(count_factors(10))
4                             # 4 factors of 10={1,2,5,10}
```

Nested Loops

A *nested loop* is a loop inside of another loop.

```
In[1]: for i in range(1, 4):
           for j in range(1, 5):
               print(i * j, end=‘ ’)
           print()
```

```
1 2 3 4
2 4 6 8
3 6 9 12
```

Nested Loops Example 1

```
In[1]: for i in range(1, 6):
           for j in range(1, i+1):
               print(j, end=‘ ’)
           print()
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Nested Loops Example 2

```
In[2]: for i in range(1, 6):
           for j in range(6, i, -1):
               print(j, end=‘ ’)
           print()
```

```
6 5 4 3 2
6 5 4 3
6 5 4
6 5
6
```

For Loop in Movies and TV-Shows

Movies:

Groundhog Day(1993); Bill Murray.

Looper(2010); Bruce Willis and Joseph Gordon-Levitt, Emily Blunt.

Edge of Tomorrow(2014); Tom Cruise, Emily Blunt.

Happy Death Day(2017).

TV-Show:

Russian Doll(Netflix, Emmy-Nominated)

Lab 1

Create a new repl on repl.it.

Write **a for loop** to do each of the following:

1) Print out "Hello!" 10 times, each on a different line.
2) Alternate between printing "Hello" and "Hi" for a total of 20 times, each on a separate line. Use only one for loop. (Hint: Use a conditional)
3) Print 1 4 9 16 25 … 100
4) Print 10 8 6 4 2 0 -2
5) Compute the sum: $1^2+2^2+3^2+4^2+\ldots+19^2+20^2$

Continue on next page.

Lab 1

Write **a nested for loop** to do each of the following:

1) Print out 10 lines, each line containing 5 "Hello" separated by spaces.

2) ******

******

******

3) Print

*

**

***

****

*****

Lab 2

Create a new repl.

1) Rewrite the function count_factors as explained in a previous slide.

2) A number n is prime if its only factors are 1 and n. Write the function is_prime which accepts an integer n and returns whether it is prime. Note that 1 is not prime. You must call the function count_factors in your implementation of is_prime.

is_prime(13) returns true

is_prime(1245) returns false

3) Write the function num_primes which accepts an integer n and and returns the number of primes up to and including n. You must call the function is_prime in your implementation.

num_prime(11) returns 5 since 2, 3, 5, 7, 11 are the 5 prime numbers less than or equal to 11.

Write the main function and calls the three above functions with different input and make sure that your functions work as expected.

References

1) Vanderplas, Jake, A Whirlwind Tour of Python, O'reilly Media.