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 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
```

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
```

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
```

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)

Indefinite Loop

A **for** loop, we discussed earlier is an example of a definite loop, the number of iterations can be specified ahead of time by the programmer.

In some cases, however, the number of iterations can be unknown. For example, a user is asked to enter a set of inputs. The number of inputs the user enter is not known in advance.

The program's input loop accepts these values until the user enters a special value or **sentinel** that terminates the input.

In this section, we explore the use of the **while** loop to describe conditional iteration: iteration that repeats as long as a condition is true.

While Loop

The **while loop** has the syntax:

```
while <condition>:
      block
```

The loop **r**epeatedly executes its block of code as long as the condition is true.

At least one statement in the block of the loop must update a variable that affects the value of the condition. Otherwise, the loop will continue forever, an error known as an **infinite loop**.

While vs For Loops 1

The following two segments of code are equivalent.

```
# Counting down with a for loop from 10 to 1.
for count in range(10, 0, -1):
      print(count, end = " ")


# Counting down with a while loop from 10 to 1.
count = 10
while count >= 1:
      print(count, end = " ")
      count -= 1
```

While vs For Loops 2

The following two segments of code are equivalent.

```
# Summation with a for loop
theSum = 0
for count in range(1, 101):
      theSum += count
print(theSum)
# Summation with a while loop
theSum = 0
count = 1
while count <= 100:
      theSum += count
      count += 1
print(theSum)
```

While Loop

```
while True:
        number = int(input("Enter the numeric grade: "))
        if number >= 0 and number <= 100:
                break
        else:
                print("Error: grade must be between 100 and 0")
print(number) # Just echo the valid input
```

```
Enter the numeric grade: 101
Error: grade must be between 100 and 0
Enter the numeric grade: –1
Error: grade must be between 100 and 0
Enter the numeric grade: 45
45
```

While Loop

What does the following program do?

```
s = 0.0
data = input("Enter a number or just enter to quit: ")
while data != "":
    number = float(data)
    s += number
    data = input("Enter a number or just enter to quit: ")
print("The sum is", s)
```

While Loop

```
s = 0.0
data = input("Enter a number or just enter to quit: ")
while data != "":
    number = float(data)
    s += number
    data = input("Enter a number or just enter to quit: ")
print("The sum is", s)
```

Sample Run:

Enter a number or just enter to quit: 3

Enter a number or just enter to quit: 4

Enter a number or just enter to quit: 5

Enter a number or just enter to quit:

The sum is 12.0

While Loop

Alternatively, the previous program can be simplified so that only one input is used. The program uses the **break** statement to exit the loop if the input is an empty string.

```
s = 0.0
while True:
    data = input("Enter a number or just enter to quit: ")
    if data == "":
        break
    number = float(data)
    s += number
print("The sum is", s)
```

Random Numbers

In some situation, we like to be able to simulate randomness. For example, we might toss a coin or roll a die.

The Python's random module contains many functions to do this. The function **randrange()** is easy to use since it is similar to the **range()** function we used in for loops.

randrange(start, stop, step) generates a random integer beginning with start(including) and ending with stop(not including) with step.

```
for i in range(10):
    num = random.randrange(1, 5)
    print(num, end=" ")
```

Output:

3 3 3 2 1 3 2 2 3 4

Guessing Game

Let's write a simple guessing game.

The computer randomly generate a number in some given range. On each pass through the loop, the user enters a number to attempt to guess the number selected by the computer.

The program responds by saying "You've got it," "Too large, try again," or "Too small, try again." When the user finally guesses the correct number, the program congratulates him and tells him the total number of guesses.

Guessing Game

```
import random
smaller = int(input("Enter the smaller number: "))
larger = int(input("Enter the larger number: "))
myNumber = random.randint(smaller, larger)
count = 0
while True:
	count += 1
	userNumber = int(input("Enter your guess: "))
	if userNumber < myNumber:
		print("Too small!")
	elif userNumber > myNumber:
		print("Too large!")
	else:
		print("Congratulations! You've got it in",
			count, "tries!")
		break
```

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 and a conditional)
3) Print 1 4 9 16 … 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

*

**

***

****

*****

References

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