Introduction to Python

Definite Iteration: For Loops

Topics

1) For Loops

For Loops

Iteration is a repeating portion of an algorithm. Iteration repeats a specified number of times or until a given condition is met. Iteration loops are frequently referred to as **for** loops because for is the keyword that is used to introduce them in nearly all programming languages, including Python.

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

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

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

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

```
2 3 5 7
```

For Each Loops

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

```
2
3
5
7
```

Iterate through each number in the list.

"for each x in" list.

```
for x in "hello":
    print(x)
```

```
h
e
l
l
o
```

Iterate through each character in the string.

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).

```
for i in range(5):
    print(i, end=" ")
```

Think of range(5) as generating this list: [0, 1, 2, 3, 4].

0 1 2 3 4

range(start, stop)

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

```
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.

```
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.

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

10 9 8 7 6 5 4 3

Definite Iteration

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

```
for i in range(5):
     print("*", end="")
```

```
*****
```

The loop above prints five *'s. We can determine this ahead of time 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

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

We need a variable that accumulate the sum at each iteration of the loop. This variable should be initialized to 0.

```
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 parameter n and returns the sum of integers from 1 to n(including).

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


print(sum(5))   # 1+2+3+4+5=15
a = sum(100)    # a = 5050
print(a)        # 5050 is printed on console
```

Conditional Summing

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

```
sum = 0
for i in range(0, 101, 3):
    sum += i
```

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

```
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.
4) otherwise, just add the number.

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 parameter n and returns the number of factors of n.

```
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

print(count_factors(10))  # 4 (factors of 10 = {1,2,5,10})
print(count_factors(7))   # 2 (factors of 7 = {1,7})
```

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: $1^2+2^2+3^2+4^2+\ldots+19^2+20^2$

Lab 2: Counting Primes

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.

Call the three above functions with different inputs and make sure that your functions work as expected.

References

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