Data Science

Introduction to Python

CS202 09.12.22

While Loops

Python has two types of loop commands: one of which is called the while loop. With the while loop we can execute a set of statements as long as a condition is true.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

The while loop requires relevant variables to be ready, in the above example we had to define an indexing variable “i” which we set to 1.

We can include If and Else statements in while loops:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    print(“hello”)
  i += 1
```

While Loops

Break and Continue

The break statement:

The break statement stops the loop even if the while condition is true.

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

In this example when our variable “i” is 3 our loop will break regardless of being below 6.

While Loops

Break and continue

The continue statement:

The continue statement will stop the current iteration and continue with the next.

```python
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

In the above example we will not see ‘3’ since we said stop when “i” is 3 and then continue. We can try this during demo.

While Loops

With If and Else statements

We’ve seen examples where we can incorporate if statements inside our loop. We can incorporate else as well. With the else statement we can run a block of code once when the condition is no longer true.

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6”)
```

We will see that in this example we will see the print statement after 6.

For Loops

A for loop is used for iterating over a sequence, remember that there is indexing in lists, tuples, dictionaries, set or a string. This means we can use the for loop for these data types.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

In the above example the for loop iterates through each item in the list called “fruits” and it prints the item.

For Loops

Break Statement

With the break statement we can stop the loop before it has looped through all the items.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

This will break when we hit banana (inclusive)

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

This will break when we hit banana (exclusive)

For Loops

Continue statement

With the continue statement we can stop the current iteration of the loop, and continue with the next.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

This will stop the loop at banana and then continue. So we will see that this prints “apple” and “cherry” but “banana” will be missing.

For Loops

range( )

To loop through a set of code a specified number of times, we can use the range( ) function,

The range( ) function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
for x in range(6):
  print(x)
```

This for loop will run through the numbers 0 through 5.

For Loops

range( )

We can have a start parameter and we can have increment parameter.

```
for x in range(2, 6):
  print(x)
```

The above is a for loop that iterates from 2 to 5 (the end is non-inclusive)

```
for x in range(2, 30, 3):
  print(x)
```

The above is a for loop that iterates from 2 to 30 (not including 30) but it will increment by 3. So you will see 2,5,8… etc.,

For Loops

If and Else

The else statement in a for loop specifies a block of code to be executed when the loop is finished.

```
for x in range(6):
  print(x)
else:
  print("Finally finished!”)
```

The above for loop will print all numbers from 0 to 5 and print a message when the loop is finished.

The else would not be executed if it blocked by a break statement.

```
for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
```

The above for loop will break at x=3 and you will therefore not be able to get to the else statement.

For Loops

Nested loops

A nested loop is a loop inside a loop. The inner loop will be executed for each iteration of the outer loop. Be careful, often times this can get confusing and you may be running things too many times.

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

The inner for loop here is “for y in fruits” we are executing this for each adjective in “adj”. So what will we get? Lets iterate through the outer loop first:

The first item in “adj” is “red”, so for “red” we iterate for every item inside “fruits” so we will get “red apple”, “red banana” and “red cherry”. Then we go to the second item on the “adj” list and repeat.

For Loops

Pass statement

A for loop cannot be empty. If you have an empty for loop you can use the pass statement to avoid getting an error.

For example, if you have a for loop in mind but maybe you want to pass it onto a coworker or a collaborator to make that for loop you can write a pass statement so that the rest of your code wont break and you can ask another person to fill in the loop.

```
for x in [0, 1, 2]:
  pass
```

The above loop will not return anything.