Programming with Python 3

Loops in Python

- Harsh Nandwana

Python Loops

- Python has two primitive loop commands:
  - `while` loops
  - `for` loops

The while Loop

- With the `while` loop we can execute a set of statements as long as a condition is true.
- **Example**
  - Print i as long as i is less than 6:

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

The break Statement (while)

- With the `break` statement we can stop the loop even if the while condition is true:
- Example
- Exit the loop when i is 3:

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

The continue Statement (while)

- With the `continue` statement we can stop the current iteration, and continue with the next:

- **Example**
  - Continue to the next iteration if i is 3:

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

The else Statement (while)

- With the `else` statement we can run a block of code once when the condition no longer is true:

- **Example**
  - Print a message once the condition is false:

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

Python For Loops

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

- This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

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

Looping Through a String

- Even strings are iterable objects, they contain a sequence of characters:
- Loop through the letters in the word "banana":

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

The break Statement (for)

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

- Example
  - Exit the loop when x is "banana":

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

The Continue Statement (for)

- With the `continue` statement we can stop the current iteration of the loop, and continue with the next:

- Example
  - Do not print banana:

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

The range() Function

- 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.

- **Example**
  - Using the range() function:

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

Else in For Loop

- The `else` keyword in a `for` loop specifies a block of code to be executed when the loop is finished:
- The `else` block will NOT be executed if the loop is stopped by a `break` statement.

- Example
  - Print all numbers from 0 to 5, and print a message when the loop has ended:

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

Nested Loops

- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":

- **Example**
  - Print each adjective for every fruit:

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

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

The pass Statement (for)

- `for` loops cannot be empty, but if you for some reason have a `for` loop with no content, put in the `pass` statement to avoid getting an error.

- **Example**

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

Thank you