Python Iterative Statements (Loops)

S1 MCA

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.
- Syntax:

```
for variable in iterable:
    # code block
```

- **The range() Function**

If you do need to iterate over a sequence of numbers, the built-in function range() comes in handy. It generates arithmetic progressions

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)
```

→Output :

0

2

.

5

- The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter:
- range(2, 6), which means values from 2 to 6 (but not including 6):

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

→Output: 2 3 4 5

- The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter:

range(2, 30, **3**):

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

→ Output : 2,5,8,…,29

- **Else in For Loop**

The else keyword 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!")
```

break and continue Statements

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

break Statement Used to exit the loop immediately when a condition is met.

```
>>>for i in range(10):
        if i == 5:
                break
        print(i)
```

continue Statement

Used to **skip the current iteration** and move to the next one.

```
>>>for i in range(5):
        if i == 2:
                continue
        print(i)
```

- **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":

Syntax:

```
for outer in outer_iterable:
    for inner in inner_iterable:
        # code block
>>>for i in range(1, 4):      # Outer loop (1 to 3)
          for j in range(1, 4):  # Inner loop (1 to 3)
           print(i, j)
```

- **print(item, end=" ")**

In Python, the print() function normally ends with a newline (\n), so each print appears on a new line.

print(value, end="something")

The end=" " part replaces the default newline with a space (" "), or any string you choose. So instead of printing each item on a new line, it prints on the same line with a space between.

```
>>>for i in range(3):
        print(i, end=" ")
```

end value	Output style
" "	0 1 2
"-"	0-1-2
""	012
"\n"	default new line

- The pass Statement

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.

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

- # Iterating over a list

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

- # Iterating using range()

```
for i in range(5):  # Iterates from 0 to 4
        print(i)
```

Python While Loops

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

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

increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

- With the break statement we can stop the loop even if the while condition is true:

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

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

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

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

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