Loops

January 12, 2021

1 Module 0 - Python Loops, Conditionals

A PDF version of this notebook is available at Module 0 - Python Loops, Conditionals

1.1 While Loop

The **while** loop is a way to repeat a block of code as long as a specified condition is True.

```
while <exp is true>:
    code block
```

The loop will execute while `<exp>` is `True` or a "truthy" statement.

[1]:
```
i = 0
while i < 5:
    print(i)
    i += 1
```

```
0
1
2
3
4
```

We need to be careful of infinite loops as in this example:

```
while True:
    print("Hello")
```

Or loops that never get evaluated as:

```
while 0:
    print("Hello")
```

Here 0 is `False`. The same with the statement:

```
while None:
    print("Hello")
```

In some cases we can create an infinite loop and test the condition inside the loop and break out of the loop when the condition becomes false using the `break` statement:

[2]:
```
i = 1

while True:
    print(i)
    i += 1
    if i >= 8:
        break
```

```
1
2
3
4
5
6
7
```

The indentation below the `while` statement represents the code to be looped. E.g., `code block 2` is not part of the loop below:

```
while <exp is true>:
    code block 1
code block 2
```

1.2 Conditionals

A conditional is a construct that allows you to branch your code based on conditions being met (or not). This is achieved using **if**, **elif** and **else** or the **ternary operator** (aka conditional expression). The `if` statement must follow the format:

```
if <exp is true>:
    code block 1
```

[3]:
```
a = 2
if a < 3:
    print('a < 3')
else:
    print('a >= 3')
```

```
a < 3
```

[4]:
```
a = 15
if a < 5:
    print('a < 5')
elif a < 10:
    print('5 <= a < 10')
else:
    print('a >= 10')
```

```
a >= 10
```

1.2.1 *Conditional Expressions*

[5]:
```
a = 15
res = 'a < 10' if a < 10 else 'a >= 10'
print(res)
```

```
a >= 10
```

1.3 For Loop

The `for` keyword can be used to iterate an iterable object. To use the **for** loop in Python, we **require** an iterable object to work with. A simple iterable object is generated via the `range()` function

[6]:
```
## Note that the range iterable function starts at 0 and does not include the␣
 ↪number inside the iterable function
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

[7]:
```
for x in [1, 2, 3]:
    print(x)
```

```
1
2
3
```

[8]:
```
for x in 'hello':
    print(x)
```

```
h
e
l
l
o
```

[9]:
```
for x in ('a', 'b', 'c'):
    print(x)
```

```
a
b
c
```

[10]:
```
for x in [(1, 2), (3, 4), (5, 6)]:
    print(x)
```

```
(1, 2)
(3, 4)
(5, 6)
```

We can use more than 1 index

[11]:
```
for i, j in [(1, 2), (3, 4), (5, 6)]:
    print(i, j)
```

```
1 2
3 4
5 6
```