Return Statement

`return` Statement

`return None`

Some functions return nothing (for example the `print()` function). To achieve this you can either return `None`, leave the return value blank after `return`, or put no `return` statement at all.

[11]:
```
def none1():
    return

def none2():
    return None

def none3():
    x = 2 #Needs code to work
```

[13]:
```
type(none1())
```

[13]:
```
NoneType
```

[14]:
```
type(none2())
```

[14]:
```
NoneType
```

[15]:
```
type(none3())
```

[15]:
```
NoneType
```

`return` Breaks Out of the Function

It was stated above that the `return` statement breaks out of the function. This means that anything that comes directly after a `return` inside the function body will not execute. Consider the following example to illustrate this:

[16]:
```
def message():
    print('This code will execute')
    return
    print('This code will not execute')
```

[17]:
```
message()
```

```
This code will execute
```

It can be useful to use this feature of `return` to break out of a loop, or even to ignore the `else` or `elif` parts of an `if` statement.

For example, consider the function that checks if it's argument is even or odd:

[1]:
```
def is_even(value):
    if value%2 == 0:
        return True
    else:
        return False
```

[2]:
```
is_even(3)
```

[2]:
```
False
```

[3]:
```
is_even(6)
```

[3]:
```
True
```

The else part of the function is unnecessary:

[1]:
```
def is_even(value):
    if value%2 == 0:
        return True
    return False
```

[18]:
```
is_even(3)
```

[18]:
```
False
```