Practical 1

Jumping Rivers

Practical 1

The aim of this practical is to understand the syntax of functions and loops. In practical 2, we will use this knowledge in a larger example.

Basic functions

Consider the following simple function

```
v = 5
def Fun1():
    v = 0
    return v

Fun1()
```

1. Why does the final line return 0 and not 5.
2. Delete line 3 in the above piece of code. Now change `Fun1()` to allow `v` to be passed as an argument, i.e. we can write `Fun1(5)`. Call this function to make sure it works.

Default arguments:

Consider the two functions defined below:

```
def Fun2(x = 10):
    return(x)

def Fun3(x):
    return(x)
```

1. Why does

```
Fun2()
```

   work, but this raises an error

```
Fun3()
```

2. Change `Fun2` so that it returns `x*x`.

if statements.

Start with the following function definition:

```
def Fun4(x):
    if x == 5:
        y = 0
    else:
        y = 1
    return y
```

1. Change `Fun4` so that it:

- returns 1 if x is positive;
- returns -1 if x is negative;
- returns 0 if x is zero.

for loops.

```
total = 0
for i in range(1,6):
    total = total + i
total
```

The `for` loop above calculates

$$\sum_{i=1}^{5} i = 1 + 2 + 3 + 4 + 5$$

1. What is the final value of `total` in the above piece of code?

2. Change the above loop to calculate the following summations:

$$\text{(i)} \sum_{i=1}^{20} (i + 1)$$

$$\text{(ii)} \sum_{j=-5}^{15} j$$

3. Harder: Rewrite the two for loops as one for loop using the `zip()` function. You will need two separate counters i.e. total1 and total2.

4. Rewrite the two loops using the `sum()` function from the **numpy** library and the `range()` function. For example, the for loop in the first example can be written as `np.sum(range(1,6))`

More `for` loops:

```
a = 2
total = 0
for blob in range(a, 5):
    total = total + blob
total
```

1. In the code above, delete line 1. Now put the above code in a function called `Fun5`, where `a` is passed as an argument, i.e. we can call `Fun5(1)`

2. Alter the code so that the `for` loop goes from `a` to `b`, rather than `a` to 5. Allow `b` to be passed as an argument, i.e. we can call `Fun5(1,6)`.

3. Change `Fun5` so that it has default arguments of `a = 1` and `b = 10`.

4. The `range()` function also has a step argument, so to create the sequence `1, 3, 5` we would write `range(1, 6, 2)`. Alter the code such that `Fun5()` can now go up in steps of `c`. Allow `c` to be passed as an argument.

In the notes, we observed that it was straight forward to loop through a data set and select the maximum values. For instance, the maximum value of each column:

```
import pandas as pd
d = {
"t1": [1,4,7,3,20],
"t2": [10,21,11,8,5],
"t3": [8,9,4,8,4]
}
df = pd.DataFrame(d)

max_cols = []
for i in [0,1,2]:
    max_cols.append(df.iloc[:, i].max())
print(max_cols)
```

- Alter the above the code to calculate the `mean` instead of the maximum value
- Now, calculate the variance (via `var`) as well as the mean.

You should only have a single loop!