04 Practical

Nested Loops

1. Create this structure (list of lists)

```
matrix = [ [ 2, 3 ], [5, 2] ]
```

2. Write a for loop that will print out each number
3. Write a while loop that will print out each number

Functions in Loops

4. Write a function called *average* that will return the average of a list

   Definition
   ```
   def average(nums):
       # do stuff
   ```

   Usage examples:
   ```
   >>> average([59, 1, 5, 42])
   26.75
   >>> average([])
   None
   >>> average([1])
   1.0
   ```

5. Use this function to find the average of each list inside the matrix defined above
6. Use this function to then find the average of all of the lists

Turn over!

Functions with More Complex Data Structures

7. (Advanced) Define the following structure (a list of dictionaries, with lists inside them)

```
classes = [{
    'name': 'Finance',
    'students': ['Bob', 'Jack', 'Lauren']
}, {
    'name': 'Astro',
    'students': ['Foo', 'Lauren']
}]
```

Usage examples:

```
>>> classes[0]
{'name': 'Finance', 'students': ['Bob', 'Jack', 'Lauren']}
>>> classes[0]['students']
['Bob', 'Jack', 'Lauren']
>>> classes[1]['name']
'Astro'
```

8. Write a function that will take a name, and output which classes they are in, returning them in a list. Remembering that there could be more than two classes.

   Definition

   ```
   def what_classes(classes, name):
       # do stuff
   ```

   Usage example:

   ```
   >>> what_classes(classes, 'Bob')
   ['Finance']

   >>> what_classes(classes, 'Lauren')
   ['Finance', Astro]
   ```

9. Write a function that will find students who are in both classes, returning them in a list.
   What if there are more than two classes?

   Definition

   ```
   def both_classes(classes):
       # do stuff
   ```

   Usage example:

   ```
   >>> both_classes(classes)
   ['Lauren']
   ```