Python Classes Week 6

1 Functions

If we want to do the same thing in multiple places in our code then we can use *functions* rather than *copy pasting*

1.1 Declaration

We create functions using the **def** keyword - This process is called "declaration"

```
In [12]: # "Declare" a function
         def my_function():
             print("I'm nOT DysFUnctIonaL!")

         # "Call" the function twice
         my_function()
         my_function()
```

I'm nOT DysFUnctIonaL!
I'm nOT DysFUnctIonaL!

1.2 Returning

Most of the time, we use functions to carry out an operation and collect the result. **return** allows us to get the function to give us an output.

```
In [1]: # Return lower-case user input from the function
        def get_lowercase_input():
            user_input = input("All good in the hood? ")
            user_input = user_input.lower() # Convert to lowercase
            return user_input # <---

In [3]: lowered_input = get_lowercase_input()
        print(lowered_input)
```

All good in the hood? damn right it is
damn right it is

1.3 "Passing parameters"

If we want our function to manipulate some data, we can add **"parameters"** into the definition of the function. Then any time we try and call it, we will need to provide something.

We've already seen examples of this:

```
In [4]: # Some in-built python functions

        input("We can pass a string to the input function")

        len(['need', 'to', 'pass', 'something', 'to', 'get', 'its', 'length'])
```

If they can do it, so can we!

```
In [13]: # Function prints the first and last character in string
         def first_and_last(some_string):
             print(some_string[0] + some_string[-1])

         first_and_last("abcde")
```

ae

If the function takes a parameter, then we have to provide one

```
In [9]: # But, we HAVE to pass something compatible!
        first_and_last()     # ERROR - Need to pass something!
        first_and_last(1321) # ERROR - We can't use variable[index] on integers
```

```
In [10]: # Need to think about these things!
         def first_and_last_v2(mebbe_string):
             defo_string = str(mebbe_string) # Now we can be sure!
             print(defo_string)
```

2 File I/O

Python has a convenient set of functions which we can use to manipulate files.

2.1 Open and Close

Before we can manipulate files, we need to open / create them. To open a file we use the **open()** function and store a "link" to the file in a variable.

- If we want to write to the file: **open("file_name", "w")**
- If we want to read fom the file: **open("file_name", "r")**
- There are more options - see the documentation at https://docs.python.org/3/library/functions.html

Once we're done with the file, we call **.close()** on the file link to close it

```
In [9]: # Create a file called "my_file.txt"
        new_file = open("my_file.txt", "w")
```

```
In [ ]: # Close the file
        new_file.close()
```

2.2 Write

The simplest way to add to our file is by using the **.write()** function *on* the variable containing the file:

```
In [3]: new_file = open("my_second_file.txt", "w")
```

```
In [ ]: # We can write as many times as we want

        new_file.write("Stories outlive their authors")
        new_file.write("\n") # Start a new line
        new_file.write("So too will these python class materials")
```

```
In [ ]: new_file.close()
```

2.3 Read

Sometimes we'd rather read the profound works of others than create our own. To this end, we can use

- **.read(n)** to read the first **n** characters * **.readline()** to read the first line
- calling .readline() a second time reads the second line, etc.

```
In [1]: wisdom = open("profound.txt", "r") # read in

        print(wisdom.read(7))

        wisdom.close()
```

My mama

```
In [35]: wisdom = open("profound.txt", "r") # read in

         print(wisdom.readline())
         print(wisdom.readline())

         wisdom.close()
```

My mama don't like you and she likes everyone
And I never like to admit that I was wrong

3 Extensions

3.1 Recursion

Recursion is a fun and occasionally useful concept. It refers to the ability to have functions **call themselves**

```
In [6]: # A recursive function calls itself

        # This function will count down to zero
        def count_down(number):
            print(number)
            if (number > 0): # Stop recursing once we've reached 0
                count_down(number - 1) # "Recursive" call!

        count_down(3) # Will print 3 2 1 0
```

3.2 Dictionaries

Along with lists, dictionaries are amongst the most useful and commonly used datastructures. Dictionaries allow us to store *Key : Value* pairs.

Whereas lists are created using [], we create dictionaries using {}

```
In [11]: # Create a dictionary
         my_dict = {"name" : "alex", "age" : 21, 1 : "one"}

         print(my_dict["age"])

         my_dict["age"] = 17 # Change my name

         print(my_dict["age"])
```

21
17

Naturally, we can also use for loops to iterate through our dictionary

```
In [12]: # Would have been useful for our username-password checking:
         users = {"alex" : "password123", "frank" : "L0LPr0"}

         for user in users:
             print("Username is " + user)
             print("Password is " + users[user])
```

Username is alex
Password is password123
Username is frank
Password is L0LPr0