1 Python Class - Week 4

1.1 More control:

1.1.1 AND & OR

and & **or** allow us to increase the complexity of logic in our programmes, without needing to a bunch of nested if statements. These can be used at declaration, or in our logical conditions, just as with the previously encountered operands (==, != etc.) * **and** is true when both following statements are true * **or** is true when at least one of the following statements is true

```
In [1]: # Use in declaration
        try_and1 = (7 > 6) and ("hi" != "bye") # will be true
        try_and2 = (6 > 7) and ("hi" != "bye") # will be false
        try_or = (6 < 7) or ("hi" != "bye") # will be true!

        # Use in comparison
        is_scary = False
        has_scales = True

        if has_scales and is_scary:
            print("GODZILLA!!! \0/ \0/ \0/")
        elif has_scales or is_scary: # one or the other, as "and" wasn't true
            print("Get it away from me!")
        else:
            print("How boring...")
```

Get it away from me!

1.1.2 in

in allows us to check if a substring or item is contained in a specified object. This will be useful when we discuss lists shortly, but we've already seen how this can be used:

```
In [8]: # For strings
        if ("a" in "alex") or ("b" in "bobby"):
            print("I can spell!")
        else:
            print("Weird alphabet..")
```

I can spell!

1.2 Controlling Flow

1.2.1 While

What if we want something to keep happening until a condition is met? -> **while** statements! * For example, letting users guess their password more than once with *if* statements meant creating ugly nested blocks * While statements will allow us to execute a block of code only *while* a condition is true.

```
In [3]: # This will run forever! :O - "infinite while loop"
        while True:
            print ("Shame...")
        # You can abort with ctrl-c or by pressing the stop button
```

```
In [1]: acceptance = "" # Keep track of condition

        print("I know these python classes are the highlight of your week!")
        while (acceptance != "Alright, FINE! It's true..."):
            acceptance = input("Confess! ")
```

I know these python classes are the highlight of your week!
Confess! no
Confess! no!
Confess! NO!
Confess! ARE YOU DEAF?
Confess! Alright, FINE! It's true...

We can also use while statements to do things a certain number of times:

```
In [5]: # Use while loops to emulate the best muppet
        limbs_remaining = 4

        while limbs_remaining > 0: # Run until the blacknight dies
            print("Come on then!")
            limbs_remaining = limbs_remaining - 1 # Increment the counter

        print("We'll call it a draw")
```

Come on then!
Come on then!
Come on then!
Come on then!
We'll call it a draw

2 Lists

We've already encountered lists, though not explicitly. Lists are the first proper "data structure" that we will cover * Lists allow us to store multiple values in one object * Lists can change size without us needing to re-declare them all the time * Lists can also be used for carrying out vector and matrix arithmetic (*See ayushes course if this kind of thing sounds interesting*)

Lists are created by using square brackets, [], and in python they can contain multiple different types of items:

```
In [11]: # Create a list
         my_first_list = ["Hello", "There", 11]

         # Check if something is in the list
         if (17 in my_first_list):
```

```
        print("My favorite number!")
    else:
        print("Oh well")
```

Oh well

2.1 Accessing list items

In python (and many other programming languages) we **count from 0**. So the first item in a list is always at the **"zeroeth" position**.

If we wish to get items from a specific location in our list, we can do this by using square brackets next to the list's name: e.g. list[index]

```
In [1]: sentence = ["A", "Wizard", "Is","Never","Late","!"]

        first_word = sentence[0]
        second_word = sentence[1]
        fifth_word = sentence[4]

        print(first_word + " " + fifth_word + " " + second_word)
```

A Late Wizard

2.1.1 List length

If we want to find out how many items are in our list, we can use the length function, **len()**:

```
In [13]: my_list = ['Size', 'doesn\'t', 'matter']

         size = len(my_list)
         print(my_list[size - 1]) # Because index starts at 0
```

matter

If we want to know where in our list a certain value is stored, we can use pythons in-built **.index()** function:

```
In [14]: worded_numbers = ["zero", "one", "two", "three", "four"]

         index_of_two = worded_numbers.index("two")
         print("The string 'two' is at position: " + str(index_of_two)) # At what index is the s
```

The string 'two' is at position: 2

2.2 Looping through lists

We can use while loops to go through the items in our lists

```
In [25]: my_list = ['She', 'sells', 'sea', 'shells']
         index = 0

         # We don't need to explicitly use list() unless we want it to print nicely
         while index < len(my_list):
             print(my_list[index])
             index = index + 1
```

She
sells
sea
shells

2.3 Range

The python **range()** function gives us a useful way of creating a list of numbers. **Note:** the function returns integers ranging from the first parameter to the second, but not including the second!

```
In [26]: numbers = list(range(2,14)) # Create a list of integers
         print(numbers)
```

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

3 Notices

3.1 Feedback

Please give us feedback so we can improve these classes! **bit.ly/uompythonclass** ## Reading Week No classes next week! Extra challenges to cement your knowledge - message hacksoc on facebook if you need help with the challenges.

4 Exercises

Using the concepts covered, try and work through as many of the exercises on the exercise sheet as you can

4.1 Extra

4.2 Modulo

The modulu operator provides a useful way of checking whether something a multiple of something else - it essentially returns the remainder after as many steps of division as possible:

```
In [ ]: test1 = 12 % 2 # this will equal 0 as 2 divides 12 exactly
        test2 = 13 % 2 # this will return 1 as 13 = (6 * 2) + 1

        # We can use this to check if something is even or odd!
        number = 10
```

```
        if (number % 2) == 0:
            print("Number divisible by 2. That means it's even!")
        else:
            print("How odd ¯\_([U+30C4])_/¯")
```

4.2.1 List Slicing

Sometimes we want to access sub-sections of our list, without needing to manually get every item contained in that section. To this end we can use colons when accessing arrays. * This works like **my_list[begin:end:step_size]**

Note you don't need to specify all three of these, python will take default values:

- begin - start of list
- end - end of list
- step_size - 1

```
In [29]: sentence = ["What", "A", "Great", ",", "Guy","Never","Said", "Anything","Bad"]

         # Emulate the media
         out_of_context = sentence[4:8:1] # Get part of the list
         print(out_of_context)

         # Strings are effectively lists of characters
         the_dream = "desserts"
         university = the_dream[::-1] # We can go backwards too

         print(university)
```

```
['Guy', 'Never', 'Said', 'Anything']
stressed
```