THE UNIVERSITY OF SYDNEY

INFO1110 / COMP9001 Week 3 Tutorial

Switches and File I/O

Switches

It is quite common for the execution of a program to depend on its input. For example a function that takes square roots might exempt negative numbers, or a ticket booking program might give discounts only if the correct code is entered. Either way, altering the flow of the program contingent on some condition is known as a switch or an 'if statement'.

The syntax of the statement in python is given as:

```
if <condition>:
        <do something>
elif <condition>:
        <do a different thing>
else:
        <Something different again>
```

Where elif is a contraction of 'else if'.

Note that the positions of the colons and the indentation on subsequent lines is incredibly important syntactically, and forgetting either will throw an error. Indentations should be four spaces, not a tab, mixing tabs and spaces will cause an error when you try to run the code.

```
a = 1
if a > 0:
        print("a is positive!")
elif 0 == a:
        print("a is zero!")
else:
        print("a is negative!")
print("Done.")
```

Note that when comparing the equality of two numbers the double equals operator is required. Using a single equals might result in the variable being assigned the value rather than being compared to it.

Question 1: Positivity

Write a Python program that accepts integer values as command line arguments. You may find the **int()** function useful here.

- If there is only one argument, this program should double it and print it
- If there are two arguments, the program should print the larger of the two
- If there are three or more arguments the program should inform the user that this problem is too hard.

Command line arguments can be used within your program as follows:

```
import sys
# Check that there are more than three command line arguments
if len(sys.argv) > 3:
        print(sys.argv[1]) # Print the first
        print(sys.argv[2]) # The second
        print(sys.argv[3]) # And the third
```

And can be passed to your program simply by entering them after the Python call to run it:

```
python my_program.py 1 2 teapot
```

Multiple Condition Switches

Occasionally you may wish to include more that one condition in a switch, such that the overall condition is true if any or all of the conditions are met. To do this we use the **and**, **or** and **not** operations. For example, in checking whether a number is divisible by 6, we can check whether it is divisible by 2 and divisible by 3.

```
        if a % 2 == 0 and a % 3 == 0:
                print("a is divisible by 6!")
```

Another handy Pythonicism is the **in** keyword that can be used to compare an element to some collection of elements. For instance whether a certain letter or word is in a string.

```
        poem = '''
        There was movement at the station, for the word had passed around
        That the colt from old Regret had got away.
        '''
        if "colt" in poem:
                print("The Man from Snowy River.")
        else:
                print("Any other poem")
```

Question 2: Flags

Write a Python program that accepts flags as command line arguments. The first argument will be some single word.

- If the '-f=<text>' flag is passed it should append whatever the specified text is to the front of the word.
- If the '-e=<text>' flag is passed it should append whatever the specified text is to the end of the word.
- If the '-caps' flag is passed then the output should be all uppercase.

For example if the input is:

```
python flags.py quokka -f=cute -e=srunning -caps
```

Then the output should be:

```
CUTEQUOKKASRUNNING
```

Format

In the previous tutorial you should have seen how adding strings can concatenate two strings together. However we may wish to manipulate strings in a more general manner.

Python contains a format function that can

```
my_string = """
For all sad words of tongue and pen,
The saddest are these,
'{words}'.
"""
formatted_string = my_sting.format(words="It might have been")
```

Formatting can also be done on a string without specifying a name to format on, instead relying on the position of the braces within the string.

```
my_string = "We are the {} who say {}".format("knights", "Ni!")
```

Question 3: Flags Returned

Change your solution to the 'flags' problem to use the format method.

Question 4: Serpentine

Write a program that takes a single character as a command line argument and uses it to print a text based snake. The format command will be helpful here.

```
python serpentine.py s
s          ssss        ss0s0s
 s        ss sss      ssssssss
  s      ss   sss    ssssss \
   s    ss      sss ssss     |\
    sss          sssss
```

```
python serpentine.py q
q          qqqq        qq0q0q
 q        qq qqq      qqqqqqqq
  q      qq   qqq    qqqqqq \
   q    qq      qqq qqqq     |\
    qqq          qqqqq
```

Question 5: Double Format

Describe what is happening with the format function on the following lines of Python code.

```
"{}".format("dog")
"{{}}".format("dog")
"{{{}}}".format("dog")
"{{{}}}".format("dog").format(dog="cat")
```

File I/O and Python Objects

Files in Python are their own type, and can be opened and accessed using the **open** function. When opening a file a mode needs to be set, typically *r*ead,*w*rite or *a*ppend represented by 'r', 'w' and 'a' respectively.

```
my_file = open('filename','r')
```

It is essential to close open files when you have finished using them. In order to prevent conflicts with two programs modifying the same file at the same time, a file is temporarily 'owned' by the process that opened it, potentially blocking other programs.

Files can be closed using the close method.

```
my_file.close()
```

Question 6: Under the Hood

- Use the **dir** function on a file that you have opened. By accessing the *__class__* attribute, what is the type of a file?
- Access and read the *__doc__* attribute. How would you allow simultaneous reading and writing to a file?
- Open a plaintext file in read mode (create one if you don't have one handy, or open up another Python file you have written).
- What is contained within the *name* attribute for the file? What about the *mode* attribute?
- What methods listed in **dir** will allow you to read from a file? Try to read the contents of the file.
- You should have found four different methods that read from the file, what differences are there between these methods? You may need to close and re-open the file once you reach the end in order to start reading from the beginning. The **help** function may assist you here.
- How do the two write methods differ? Again the **help** function may be useful.
- What does the truncate function do?

With

In larger programs, the points at which a particular file is opened or close can be unclear. The **with** keyword can be used using a very similar syntax to the **if** statement to delineate the scope over which the file is open.

So long as the indented block that starts with the **with** keyword continues, the file is open, when the indented block ends the file is closed automatically.

```
with open("my_file", "r") as my_file:
        contents = my_file.readlines()
print(contents)
```

Question 7: Filing

Write a Python program that does the following:

- Using the with keyword open a file and read its contents. The file should only consist of a single positive integer.
- If the integer is even, divide it by two and write it back to the file. If the integer is odd multiply it by 3 and add one and then write it back to the file.
- If the integer in the file is one, print 'Finished!'.

Write an initial number to the file that you are reading and keep running your program until it reaches the end condition. Does it always reach this end condition? Is there any helpful programming technique that allows us to run the same snippet of code a large number of times?

Question 8: Counting out Time

Sometimes your program needs to know and use the time. For this you should explore the properties of the datetime object:

```
import datetime
datetime_object = datetime.datetime.now()
```

- Print the current date in a human readable format.
- If the day of the month is divisible by three, print a string describing your surprise at this revelation.
- If the day of the month is divisible by two, print a string describing how predictable this is.
- Extend your program to print the day of the week as a string.
- If it's the same day as your tutorial, your program should remind you to attend your tutorial. Otherwise it should tell you to practice programming.
- If the day starts with a 'T', print it backwards.
- In addition to the above, append each of the statements you have printed to a file called 'down_by_numbers.txt'.