THE UNIVERSITY OF SYDNEY

INFO1110 Week 4 Tutorial

Loops and Lists

Collections in Python

So far you've only seen problems that involve a small number of variables that need to be stored and saved. However, what happens when the number of variables reaches into the hundreds, thousands or even millions? Rather than writing each variable out on an individual line (a task as pointless as it is time consuming) we instead build a single object that contains other objects: a collection.

The simplest of these is a list. A list stores a set of items in a single object, and can access them using an index. The first item in the list has index 0, the second has index 1 and so forth. The syntax for initializing a list is:

```
>>> list_obj = [4, "five", 6]
```

In Python, lists can store any type of variable in any element, other languages are not so forgiving and impose strict uniform typing requirements over all elements of a list. The elements of the list can be accessed or modified using the following syntax:

```
>>> list_obj[0]
4
>>> list_obj[1]
"five"
>>> list_obj[1] = 3
>>> list_obj[1]
3
```

You may notice that sys.argv from the previous tutorial is itself a collection and follows the same syntax as above, and that sys.argv is then either a list or a tuple.

Of course there are some pre-built Python functions that can generate certain collections for us, in particular the range() function which generates a series of numbers between two values with some defined difference between them.

Loops

Of course once we have a collection of variables, we need some easy method of accessing them all, for this we have loops. Loops delineate some section of code that is repeated until some condition is met. The simplest of these is the *while* loop:

```
a = 0
while a < 10:
    print(a)
    a = a + 1
```

In the above code, so long as the value of *a* is less than 10, then the code in the indented block will run. When the end of the indented block is reached, the condition is checked again.

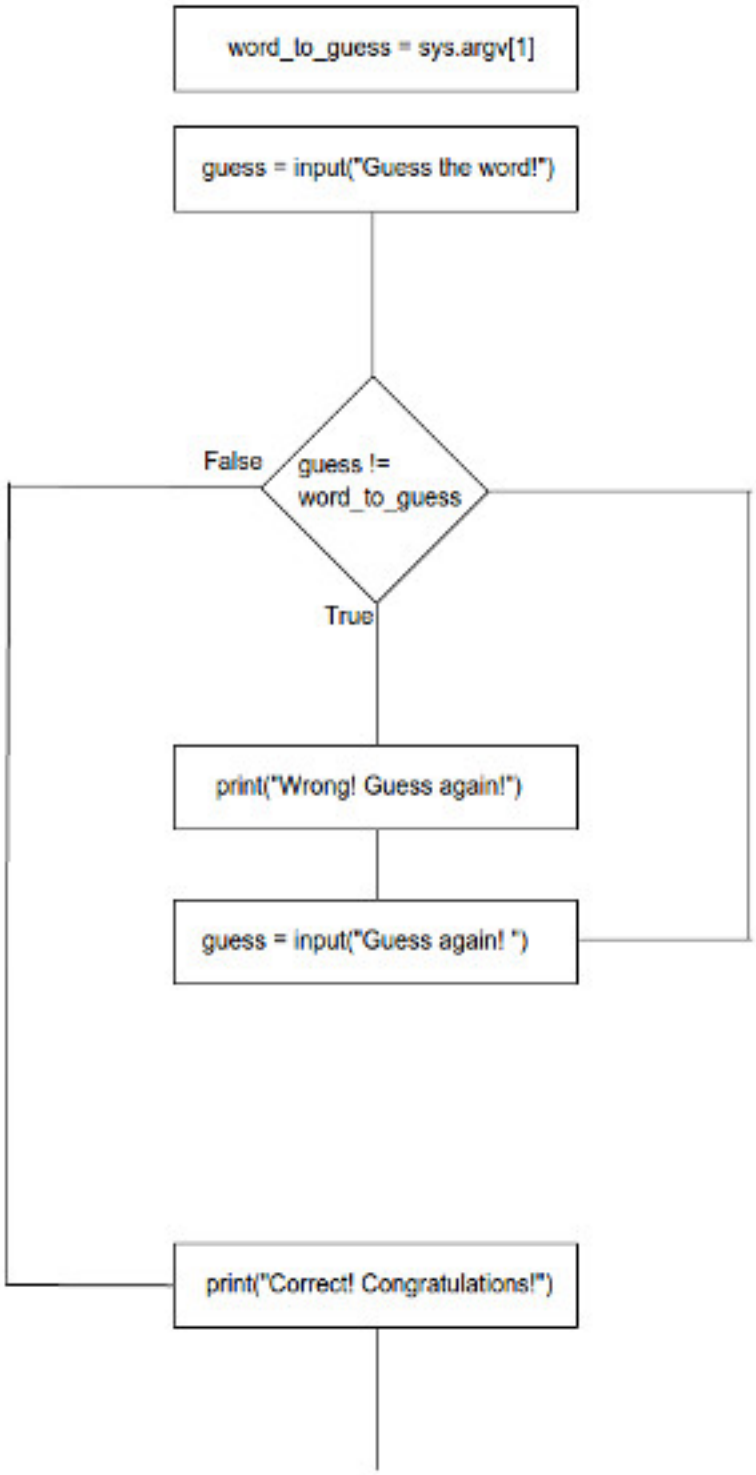
In Python (but not other languages) loops can also be called directly on collections. For this we use the *for* loop.

```
a = ['test', 2 'silly walk']
for word in a:
    print(word)
```

Question 1

*[Convert]

Given the current control flow diagram, translate the diagram to your own python program.



Question 2

*[Entscheidungs] What will the following code do?

```
a = 0
while a < 10:
    print(a)
```

What about this code?

```
import random
a = 0
while a < 10:
    print(a)
    a = a + random.random() - 0.5
```

If you are unsure run it a few times and see what happens. What does this imply about loops with poorly defined conditions? *Ctrl + C* might be helpful here.

Question 3

*[Hovercraft]

Write a Python program called *hovercraft.py* that reads program arguments. Your program should loop through all arguments and check if any of the arguments are the string 'eel' or 'eels'. If all the arguments are eels then your program should print "My hovercraft is full of eels.". If there is at least one eel but your hovercraft is not full of eels then you should print how many eels are in your hovercraft.

If there are no eels, then you should print how disappointed you are at the lack of your slippery friends.

Question 4

*[Reading between the Lines] Using loops, read and print each line in a file using the readline() function. Then treat the file as a collection of lines and try to loop over it. What happens when you try to print the elements you are looping over?

Enumerate

Occasionally cases arise where Python's looping over collections may be somewhat restrictive. In particular when trying to loop over multiple related collections. For this we have the enumerate and zip functions.

Enumerate returns both the index of the current element (indicating its position) along with the value of the element.

```
a = ["cut down the", "spam", "I will not"]
for index, element in enumerate(a):
    print(index, element)
```

So if we have multiple collections that we wish to print over that are related in some manner, we can use the index:

```
list_a = ["cut down the", "spam", "I will not"]
list_b = ["buy this record", "tallest tree ", "spam"]
list_c = ["spam", "it is scratched", "with this herring"]
for index, element in enumerate(a):
    print(list_a[index], list_b[(index + 1)%3, list_c[(index + 2)%3]])
```

Question 5

*[Highscores]

You will need to categorise and sort the highscore for each player. Using enumerate function, you can show the current rank of the player. We want to ensure that the player with the highest score is the first element of the list and the player with the lowest score is the last.

Hint: Sort elements in a list, you can use `.sort` function. You may want to consider how you store the elements (like a Tuple) and their order. You may want to use `reversed`.

Given input:

```
Player1 5000
Player2 9000
Goku 12000

1. Goku 12000
2. Player2 9000
3. Player1 5000
```

Zip

Manipulating the values of the indexes can be somewhat fiddly and in cases where the relationship between the collections is reasonably trivial we can simply join two or more of them using the **zip** function.

```
a = ['knights', 'who', 'say', 'ni']
b = ['ekki', 'ekki', 'pthang', 'floop']
for i in zip(a,b):
   print(i)
```

As you can see, these are zipped together as a tuple. They can also be separated into different variables

```
for i,j in zip(a,b):
    print(i)
    print(j)
```

zip provides an elegant way merging collections together into a singular collection that we can use. When working with multiple related data sources this is a very powerful function to utilise.

Loops, Collections and Generators

Just as Python can loop over a collection, it can also generate a collection from a loop. By encapsulating an expression in square brackets, each iteration of the loop is mapped to an element of the list.

For example, the following code squares each element of the range collection and returns a list containing the squared numbers.

```
a = [i * i for i in range(10)]
```

Question 6

*[One line ROT] ROT is an encoding algorithm that asks for a rotation of text. Given that you are trying to encode messages you are sending between you and your friend, you agree on using ROT3 as a way of sending encoded messages to each other and being able to decode them.

Example Encoding:

```
Hey Friend! #String for input
Khb Iulhqg! #Encoded string in ROT3
```

Write a program that takes three command line arguments. The first specifying whether to 'encode' or 'decode', the second the rotation number and the third message in question. You are to write a program that implements the ROTN algorithm. It is recommend that you start off by writing an algorithm for ROT13 and then adapt it to allow for any rotation number. The rotation number should be provided as input.

```
Please provide a rotation number: 3
Hey Friend!
Khb Iulhqg!
```

- Check that you are reading the 'encode' and 'decode' strings properly, your program should print an error message and exit gracefully if neither of these modes are specified.
- Ensure that you have properly converted the rotation number to an integer format.

- Combine all the strings from the third argument onwards into a single string.
- The rotation code itself simply increments a letter by the number used. So if the letter is 'A' and the number is 1, then it will give 'B', if the letter is 'A' and the number is 10 then the output will be 'K'. The **ord** and **chr** functions will be useful here. If you are stuck, try working with just a single character and manipulate it.
- Consider what happens when you rotate past the end of the alphabet. The modulus function and something to capitalise letters will be helpful here.
- How do you handle spaces?
- Convert and print your message, make sure you can encode and decode to the original message correctly.

Question 7

*[ROT Breaker] Similar to the previous problem, you are given an encoded string and a decoded string this time you're going to need to loop through the rotation numbers until you find the one used with this encoding.

Question 8

*[Supermax] The following code will create a list of length 10, where each element of the list is another list of length 10 containing random numbers. You are to find the largest number in the list of lists.

```
import numpy as np

list_of_lists = [np.random.random(10) for i in range(10)]
```

Using a more Pythonic approach to loops, this is doable in a single line of code. The **max** function may be useful.