Python Crash Course

Generating Random Numbers:

By using the python random libray you can generate random numbers, the ability to generate random numbers is very important in many programs including most games.
Generate random integers between range 5-15

In [10]:
```
import random
random.randint(5,15)
```

Out[10]:
```
6
```

Note: there are a lot of useful things you can do with the random , but we are covering that right now

Loops

For Loops

The for loop is a very important loop design and its used to iterate over the items of any sequence including the Python list, string, tuple etc.

In [12]:
```
seq = ['cat','dog','boy','girl','mango']
```

In [13]:
```
for item in seq:
    print(item)
```

```
cat
dog
boy
girl
mango
```

The Loop iterates by lenght of sequence times

In [14]:
```
print(len(seq))
```

```
5
```

While Loops

The while loop runs as long as the expression (condition) evaluates to True and execute the program block. The condition is checked every time at the beginning of the loop and the first time when the expression evaluates to False, the loop stops without executing any remaining statement(s).

In [15]:
```
i = 0

while i < 5:
    print('i is: {}'.format(i))
    i = i+1

print('looping Completed')
```

```
i is: 0
i is: 1
i is: 2
i is: 3
i is: 4
looping Completed
```

Infinite Loops , Don't create infinite while loops , these are ones which will always stay true and never be false, an easy example of creating an infinite while loop is this: `while True:` or `while 1` or `while i < 5:` in which you never increase the value of i so i stays below 5.

break: You can use `break` keyword to break/exit out of an infinte loop, so infinite while loops are acceptable if you plan to use `break`

See below we break out of the loop when i is > 3

In [16]:
```
i=0
while True:
    print('i is: {}'.format(i))
    i = i+1
    if i > 3:
        break

print('Breaked out of the Infinite Loop')
```

```
i is: 0
i is: 1
i is: 2
i is: 3
Breaked out of the Infinite Loop
```

Looping With range()

The range() function returns a list of consecutive integers. The function has one, two or three parameters where last two parameters are optional. It is widely used for loops.

This range will allow you to loop n times, n is the param passed inside the range

In [17]:
```
for i in range(5):  #note the upper limit is not included
    print(i)
```

```
0
1
2
3
4
```

See the range starts from 0 and does not include the upper limit

You can also set define a range between numbers to loop

In [18]:
```
for i in range(5,10):
    print(i)
```

```
5
6
7
8
9
```

You can also define a step size which tells you how much to move by for the next step or next iteration of the loop

In [21]:
```
for i in range(1,10,2):  # using step size of 2
    print(i)
```

```
1
3
5
7
9
```

Range can also be used to create a list like this:

In [22]:
```
list(range(10,18))
```

Out[22]:
```
[10, 11, 12, 13, 14, 15, 16, 17]
```

Enumerate

It allows us to loop over something and have an automatic counter
If you want access to the index of each element within the body of a loop, use the built-in enumerate function

In [26]:
```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print ( '#{}: {}'.format(idx + 1, animal))
```

```
#1: cat
#2: dog
#3: monkey
```

Doing the Same thing with For loop

In [25]:
```
i = 0
for animal in animals:
    i += 1
    print('#{} {}'.format(i,animal))
```

```
#1 cat
#2 dog
#3 monkey
```

Now doing the above with a Dictionary

In [27]:
```
d = {'cat': 1, 'dog': 2, 'monkey': 3}
for animal in d:
    index = d[animal]
    print('#{} {}'.format(index,animal))
```

```
#1 cat
#2 dog
#3 monkey
```

The proper way to do it with a dictionary

In [28]:
```
for key,value in d.items():
    print('#{} {}'.format(value,key))
```

```
#1 cat
#2 dog
#3 monkey
```

Assignment 4: Random Quizz Generator

Create a random Quizz generator that asks user 7 or 12 random math questions like multiplications, addtions , subtractions with random numbers. And at the end gives the user their percentage of score with a Grade.

List comprehension

Elegant way to define and create lists based on existing lists.
A list comprehension consists of the following parts:

1. An Input Sequence
2. A Variable representing members of the input sequence.
3. An Optional Predicate expression.
4. An Output Expression producing elements of the output list from members of the Input Sequence that satisfy the predicate.

So lets suppose you want to take a list of 4 elements like below and modify it such that it becomes a list of its squares.
First lets do that with normal python code

In [30]:
```
x = [1,2,3,4]
```

In [31]:
```
#using for loop
out = []
for item in x:
    out.append(item**2)
print(out)
```

```
[1, 4, 9, 16]
```

So it took us 4 lines and with list comprehension its just 1 line of code

In [32]:
```
#using list comprehension
[item**2 for item in x]
# part of functional programming
```

Out[32]:
```
[1, 4, 9, 16]
```

Similarly list comprehension can become more complex and include conditions too, now here we are adding 2 to the original list elements if there are greater than 3

In [33]:
```
[ item +2 for item in x if item >= 3 ]
```

Out[33]:
```
[5, 6]
```

Squaring Elements if they are > 3

In [34]:
```
a_list = [1,3,5,7,8,2,3]

squared_ints = [ e**2 for e in a_list if e >= 3 ]
print (squared_ints)

# Output expression is          "e**2"
# Variable is                    "e"
# Input sequence is              "a_list"
# Optional Predicate is      "if e >= 3"
```

```
[9, 25, 49, 64, 9]
```