#Infytq Programming Fundamentals Using Python PART-2

Topics: -

1. Functions & Arguments: -

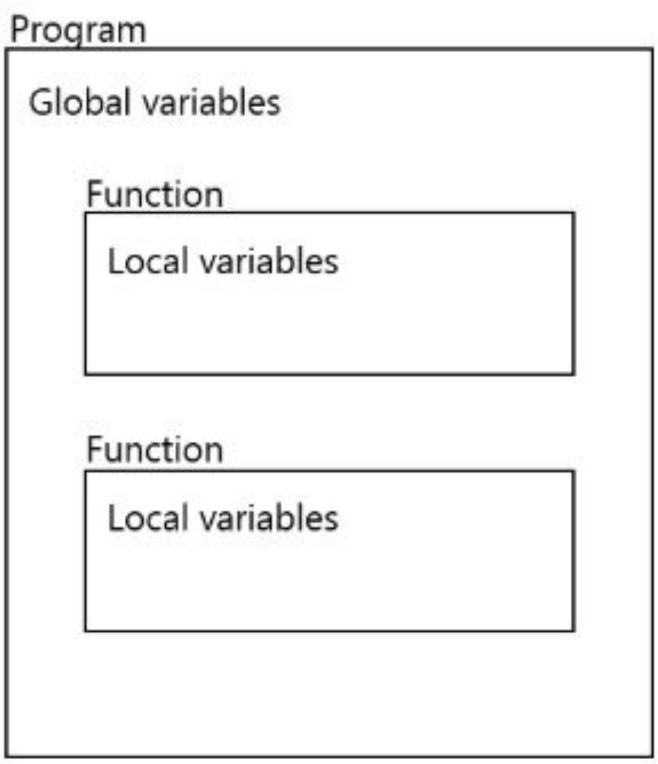
(Check from Previous Notes)

2. Variables & Scope: -

There are two types of variables which we can use in a program - local variables and global variables.

Local variables are the ones which are created inside a function. They are created when the owning function starts execution and remains in memory till owning function finishes execution. They can be accessed only inside that function.

Global variables are the ones which are created outside the functions. They are created when the program execution starts and remains in memory till the program terminates. They can be read anywhere in the program - within a function or outside



Variables & It's Scope: -

A variable is only available from inside the region it is created. This is called **scope**.

Local Scope:-

A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.

A variable created inside function is available or accessible inside that function

Ex: -

```
def myfunc():
  x = 300
  print(x)

myfunc()
```

Global Scope: -

A variable created in the main body of the Python code is a global variable and belongs to the global scope.

Global variables are available from within any scope, global and local.

A variable created outside of function is global and can be used by anyone.

```
x=300

def myfunc():
  print(x)

myfunc()

print(x)
```

: - Global Keyword: - With the help of global keyword we make global variable inside the scope also.

3. Unit Testing & Logic Coverage & Code Review: -

We have written couple of programs by now. How do you know that your programs are correct?

One way is to ask your peer or an expert to go through the code and identify if there are any mistakes. This way of reviewing your code for correctness is known as **Code Review**. Though there are guidelines for code review, it is possible that the person who is reviewing may miss some things.

Another way of ensuring that your code is correct is to test it following the below steps:

1. Identify values for input variables and its corresponding expected outputs based on business requirements
2. Run/Execute your program with the values identified in Step 1
3. Check whether the output is matching the expected output for the given input
4. If there is a match, it means that your code is working as expected for the given input
5. If not, it means that something has gone wrong in your code and you have to fix your code

Repeat this cycle with different values of input.

If the code you are testing is written inside a function, invoke the function with different input values and check whether the return value from the function is matching the expected output.

This way of testing your program by yourself is known as **unit testing.**

In this course we are going to discuss two techniques of unit testing:

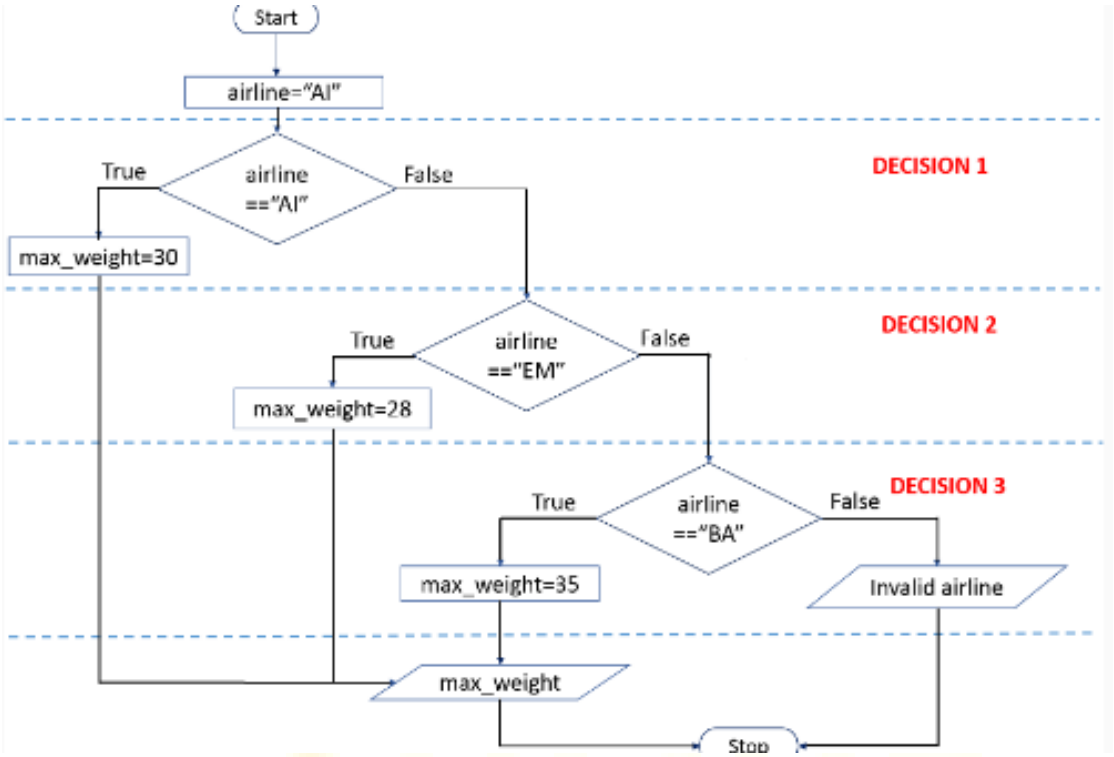
1. Path coverage
2. Boundary value analysis

The information desk folks receive a lot of queries daily from passengers regarding the maximum weight they can carry. They use a program which accepts the airline name and returns the maximum allowed weight by that airline.

The maximum luggage weight allowed by the airlines that operate from this airport are as follows:

Airline	Luggage limit (kg)
AI	30
EM	28
BA	35
Any other value	Invalid airline

Logic Coverage: -



Boundary-Value Analysis: -

What if the programmer has written the condition like:
luggage_wt >= 1 and luggage_wt < 30?

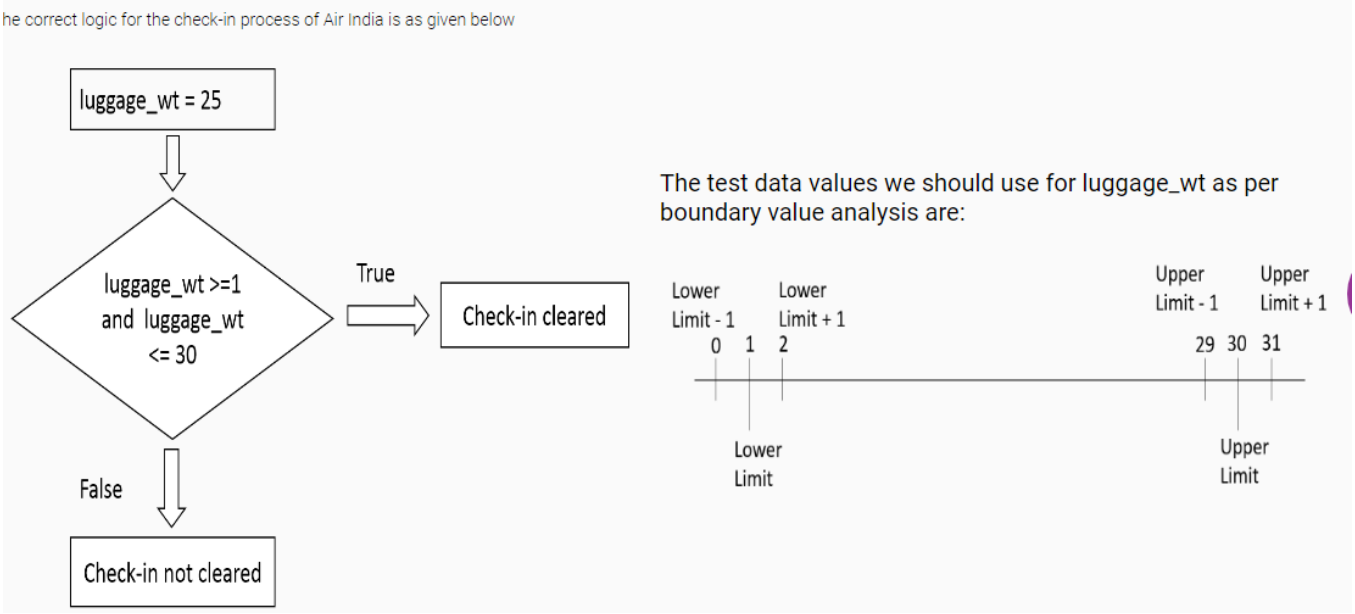
Will it work as per the requirement for luggage weight of 30kg?

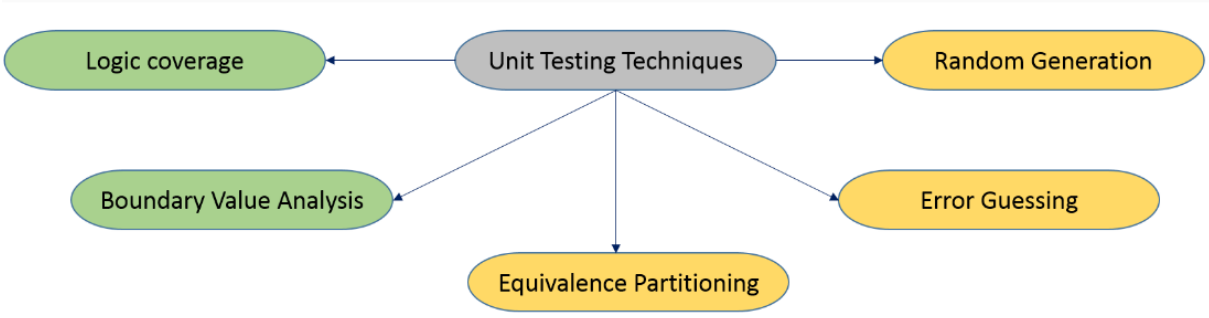
Whenever ranges (say >0 and <=30, >=30, >=1 and <=30 etc.) are being written in conditions, we need to exhaustively test as most of the programmers make errors while writing these conditions. Hence it may not be sufficient to check with just two values that take control to the true and false paths alone.

In that case we should test the boundaries by considering values

- on the boundary,
- one above the boundary and
- one below the boundary

This technique of identifying test data on the boundaries is known as **boundary value analysis.**





#Exception Handling In Python: -

The `try` block lets you test a block of code for errors.

The `except` block lets you handle the error.

The `finally` block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the `try` statement:

```
try:
  print(x)
except:
  print("An exception occurred")
```

Since the try block raises an error, the except block will be executed.

Without the try block, the program will crash and raise an error

Finally: -

The `finally` block, if specified, will be executed regardless if the try block raises an error or not.

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the `raise` keyword.

Python has many kinds of errors predefined as part of the language. Here are some of the common types.

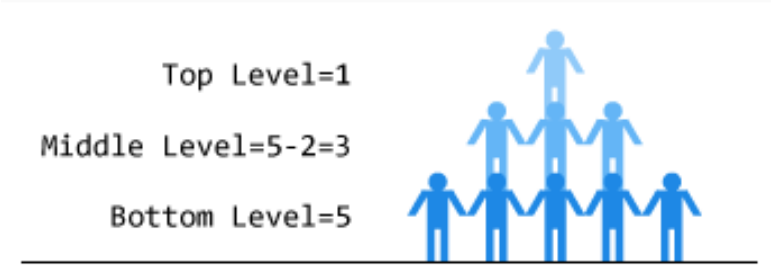
Built-in Exception	When it will be raised	Example
ZeroDivisionError	When a value is divided by zero	num_list=[] total=10 avg=total/len(num_list)
TypeError	When we try to do an operation with incompatible data types	total=10 total+="20"
NameError	When we try to access a variable which is not defined	avg=total/10 where total is not defined
IndexError	When we try to access an index value which is out of range	num_list=[1,2,3,4] value=num_list[4]

ValueError	When we use a valid data type for an argument of a built-in function but passes an invalid value for it	#string is a valid data type for int() but the value "A" is invalid, as "A" can't be converted into int.
		value="A"
		num=int(value)

#Recursion: -

Function Call by itself is called recursion.

Human tower is created by men standing on shoulders of the men standing at the lower level. It can have any number of levels. Each level will have 2 men less than the previous level. However, there will always be one person at the top of the tower, that means there will always be odd number of men standing at the bottom most/base level.

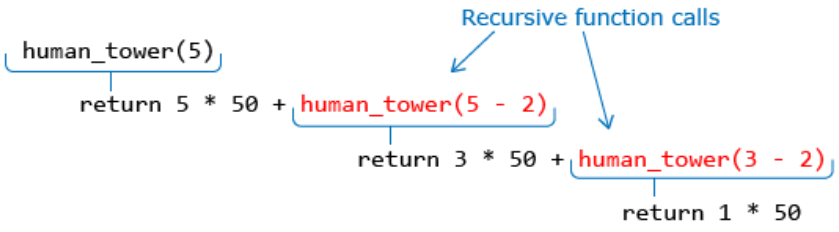


Suppose we want to find the total weight of a human tower which has 5 people standing at the bottom level. Assume that each person weighs 50 kg and there will always be odd number of men at the base level. We can solve this problem recursively. Have a look.

```
def human_tower(no_of_people):
    if(no_of_people == 1):
        return 1*(50)
    else:
        return no_of_people*(50) + human_tower(no_of_people - 2)

print "Total weight of human tower: ", human_tower(5)
```

Recursive function call



In recursion, two things are important – termination condition and recursive call. Recursive call invokes itself with a smaller argument and termination condition helps to terminate the recursive calls.

```
def human_tower(no_of_people):
    if(no_of_people == 1):
        return 1*(50)
    else:
        return no_of_people*(50) + human_tower(no_of_people - 2)

print "Total weight of human tower: ", human_tower(5)
```

Termination condition or base condition

Recursive function call

Ex:- Tower Of Hanoi Problem

Tower of Hanoi is another problem which can be used to learn recursion. This problem was discovered by the French mathematician Edouard Lucas in 1883.

Rules :

- Move only one disk at a time. Rings must be in decreasing size
- No move should result in a larger disk on top of a smaller disk
- For temporarily holding a disk, the third tower can be used

File Handling: -

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

ython allows to create files, read from files, write content to file and append content to existing content through inbuilt functions!

Method	Description
pen(file_path,operation)	This method is used to open the file for the specified operation. The operation can either be r,w,a for read, write and append.
lose()	This method is used to close a file which is already open.
rite()	This method is used to write a string to a file, if file is present. If not, it creates the file and writes the string into it.
ead()	This method is used to read all the contents of a file into a string.

The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists