Introduction

The lines starting with >>> are the commands

```
>>> 2+3
5
>>> 3>2
True
>>> print ('Hello World')
Hello World
```

Single line comment : #This is a comment

Multiline comment : ''' This is a comment
This is also a comment '''

Variables are names given to data that we need to store and manipulate in our programs.

can only contain letters, numbers or underscores, case sensitive

Assignment sign : assign value

```
x = 5
y = 10
x = y
print ("x = ", x)
print ("y = ", y)

Now run the program. You should get this output:

x = 10
y = 10
```

Basic Operator :

```
Suppose x = 5, y = 2

Addition:
x + y = 7

Subtraction:
x - y = 3

Multiplication:
x*y = 10

Division:
x/y = 2.5

Floor Division:
x//y = 2 (rounds down the answer to the nearest whole number)

Modulus:
x%y = 1 (gives the remainder when 5 is divided by 2)

Exponent:
x**y = 25 (5 to the power of 2)
```

More Assignment Operator :

x=x+2 : x+=2

x=x-2 : x-=2

Data Types in Python

1. Integers : numbers with no decimal parts, such as -5, 4, variableName = initial value, Eg :Age = 20
2. Float : numbers that have decimal parts, Eg : userHeight = 1.82,
3. String (text) : To declare a string, you can either use variableName = 'initial value' (single quotes) or variableName = "initial value" (double quotes)
4. List : collection of data which are normally related. Eg : listName = [initial values] , userAge = [21, 22, 23, 24, 25]

```
#declaring the list, list elements can be of different data types
myList = [1, 2, 3, 4, 5, “Hello”]

#print the entire list.
print(myList)
#You’ll get [1, 2, 3, 4, 5, “Hello”]

#print the third item (recall: Index starts from zero).
print(myList[2])
#You’ll get 3

#print the last item.
print(myList[-1])
#You’ll get “Hello”

#assign myList (from index 1 to 4) to myList2 and print myList2
myList2 = myList[1:5]
print (myList2)
#You’ll get [2, 3, 4, 5]
```

```
#modify the second item in myList and print the updated list
myList[1] = 20
print(myList)
#You’ll get [1, 20, 3, 4, 5, 'Hello']

#append a new item to myList and print the updated list
myList.append(“How are you”)
print(myList)
#You’ll get [1, 20, 3, 4, 5, 'Hello', 'How are you']

#remove the sixth item from myList and print the updated list
del myList[5]
print(myList)
#You’ll get [1, 20, 3, 4, 5, 'How are you']
```

5. Tuple : cannot modify their values. The initial values are the values that will stay for the rest of the program, tupleName = (initial values).

```
Example:
monthsOfYear = (“Jan”, “Feb”, “Mar”, “Apr”, “May”, “Jun”, “Jul”,
“Aug”, “Sep”, “Oct”, “Nov”, “Dec”)

You access the individual values of a tuple using their indexes, just like with a list.
Hence, monthsOfYear[0] = “Jan”, monthsOfYear[-1] = “Dec”.
```

6. Dictionary : collection of related data PAIRS, dictionaryName = {dictionary key : data}
   userNameAndAge = {“Peter”:38, “John”:51, “Alex”:13, “Alvin”:“Not Available”}
   userNameAndAge = dict(Peter = 38, John = 51, Alex = 13, Alvin = “Not Available”)

```
#declaring the dictionary, dictionary keys and data can be of
different data types
myDict = {“One”:1.35, 2.5:”Two Point Five”, 3:”+”, 7.9:2}

#print the entire dictionary
print(myDict)
#You’ll get {2.5: 'Two Point Five', 3: '+', 'One': 1.35, 7.9: 2}
#Note that items in a dictionary are not stored in the same order as
the way you declare them.

#print the item with key = “One”.
print(myDict[“One”])
#You’ll get 1.35

#print the item with key = 7.9.
print(myDict[7.9])
#You’ll get 2

#modify the item with key = 2.5 and print the updated dictionary
myDict[2.5] = “Two and a Half”
print(myDict)
#You’ll get {2.5: 'Two and a Half', 3: '+', 'One': 1.35, 7.9: 2}

#add a new item and print the updated dictionary
myDict[“New item”] = “I’m new”
print(myDict)
#You’ll get {'New item': 'I’m new', 2.5: 'Two and a Half', 3: '+',
'One': 1.35, 7.9: 2}

#remove the item with key = “One” and print the updated dictionary
```

```
del myDict[“One”]
print(myDict)
#You’ll get {'New item': 'I’m new', 2.5: 'Two and a Half', 3: '+',
7.9: 2}
```

TypeCasting : to convert from one data type to another

Making Your Program Interactive :input() and print().

```
myName = input("Please enter your name: ")
myAge = input("What about your age: ")

print ("Hello World, my name is", myName, "and I am", myAge, "years
old.")

The program should prompt you for your name.

Please enter your name:

Supposed you entered James. Now press Enter and it’ll prompt you for your age.

What about your age:

Say you keyed in 20. Now press Enter again. You should get the following statement:

Hello World, my name is James and I am 20 years old.
```

Triple Quotes : If you need to display a long message, print (‘’’Hello World. My name is James and I am 20 years old.’’’)

Hello World. My name is James and I am 20 years old.

Escape Characters

\n (Prints a newline)

```
>>> print ('Hello\nWorld')
Hello
World
```

\\ (Prints the backslash character itself)

```
>>> print ('\\')
\
```

\" (Prints double quote, so that the double quote does not signal the end of the string)

```
>>> print ("I am 5'9\" tall")
I am 5'9" tall
```

\' (Print single quote, so that the single quote does not signal the end of the string)

```
>>> print ('I am 5\'9" tall')
I am 5'9" tall
```

Condition Statements : give output as true or false

Not equals:
5 != 2

Greater than:
5>2

Smaller than:
2<5

Greater than or equals to:
5>=2
5>=5

Smaller than or equals to:
2 <= 5
2 <= 2

1. If statement : allows the program to evaluate if a certain condition is met, and to perform the appropriate action based on the result of the evaluation

```
if condition 1 is met:
    do A
elif condition 2 is met:
    do B
elif condition 3 is met:
    do C
elif condition 4 is met:
    do D
else:
    do E
```

```
userInput = input('Enter 1 or 2: ')

if userInput == "1":
    print ("Hello World")
    print ("How are you?")
elif userInput == "2":
    print ("Python Rocks!")
    print ("I love Python")
else:
    print ("You did not enter a valid number")
```

```
Enter 1 or 2: 1
Hello World
How are you?

Enter 1 or 2: 2
Python Rocks!
I love Python

Enter 1 or 2: 3
You did not enter a valid number
```

2. Incline If : simpler form of an if statement and is more convenient if you only need to perform a simple task, Syntax : do Task A if condition is true else do Task B
   Eg : print ("This is task A" if myInt == 10 else "This is task B")
   This statement prints "This is task A" (Task A) if myInt equals to 10. Else it prints "This is task B" (Task B).
3. For statement : an iterable refers to anything that can be looped over,
   Looping through an iterable
   Syntax : for a in iterable:
   print (a)

```
pets = ['cats', 'dogs', 'rabbits', 'hamsters']

for myPets in pets:
    print (myPets)
```

```
cats
dogs
rabbits
hamsters
```

Looping through a string

```
message = 'Hello'

for i in message:
    print (i)
```

The output is

```
H
e
l
l
o
```

Looping through sequence of numbers : The range() function generates a list of numbers and has the syntax range (start, end, step)

For instance,
`range(5)` will generate the list [0, 1, 2, 3, 4]
`range(3, 10)` will generate [3, 4, 5, 6, 7, 8, 9]
`range(4, 10, 2)` will generate [4, 6, 8]

To see how the `range()` function works in a `for` statement, try running the following code:

```
for i in range(5):
    print (i)
```

You should get

```
0
1
2
3
4
```

4. While : repeatedly executes instructions inside the loop while a certain condition remains valid. while condition is true: do A

```
counter = 5

while counter > 0:
    print ("Counter = ", counter)
    counter = counter - 1
```

If you run the program, you'll get the following output

```
Counter = 5
Counter = 4
Counter = 3
Counter = 2
Counter = 1
```

5. Break : When working with loops, sometimes you may want to exit the entire loop when a certain condition is met

```
j = 0
for i in range(5):
    j = j + 2
    print ('i = ', i, ', j = ', j)
    if j == 6:
          break
```

You should get the following output.

```
i = 0 , j = 2
i = 1 , j = 4
i = 2 , j = 6
```

6. Continue : rest of the loop after the keyword is skipped for that iteration.

```
j = 0
for i in range(5):
    j = j + 2
    print ('\ni = ', i, ', j = ', j)
    if j == 6:
          continue
    print ('I will be skipped over if j=6')
```

You will get the following output:

```
i = 0 , j = 2
I will be skipped over if j=6

i = 1 , j = 4
I will be skipped over if j=6

i = 2 , j = 6

i = 3 , j = 8
I will be skipped over if j=6

i = 4 , j = 10
I will be skipped over if j=6
```

When j = 6, the line after the `continue` keyword is not printed. Other than that, everything runs as per normal.

7. Try, Except : controls how the program proceeds when an error occurs

```
try:
    do something
except:
    do something else when an error occurs
```

For instance, try running the program below

```
try:
    answer =12/0
    print (answer)
except:
    print ("An error occurred")
```

When you run the program, you'll get the message `"An error occurred"`.

Errors Displayed

IOError:
Raised when an I/O operation (such as the built-in `open( )` function) fails for an I/O-related reason, e.g., "file not found".

ImportError:
Raised when an import statement fails to find the module definition

IndexError:
Raised when a sequence (e.g. string, list, tuple) index is out of range.

KeyError:
Raised when a dictionary key is not found.

NameError:
Raised when a local or global name is not found.

TypeError:
Raised when an operation or function is applied to an object of inappropriate type.

Functions : simply pre-written codes that perform a certain task.

2 key words here - def and return.

def tells the program that the indented code from the next line onwards is part of the function. return is the keyword that we use to return an answer from the function.

Binary files refer to any file that contains non-text, such as image or video files. To work with binary files, we simply use the ‘rb’ or ‘wb’ mode.

```
inputFile = open (‘myfile.txt’, 'r')
outputFile = open (‘myoutputfile.txt’, 'w')

to

inputFile = open (‘myimage.jpg’, 'rb')
outputFile = open (‘myoutputimage.jpg’, 'wb')
```

The remove() function deletes a file. The syntax is remove(filename). For instance, to delete myfile.txt, we write remove(‘myfile.txt’).
The rename() function renames a file. The syntax is rename (old name, new name). To rename oldfile.txt to newfile.txt, we write rename(‘oldfile.txt’ , ‘newfile.txt’)

isalnum() : Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise.
isalpha() : Return true if all characters in the string are alphabetic and there is at least one character, false otherwise.
isdigit() : Return true if all characters in the string are digits and there is at least one character, false otherwise
islower() :Return true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise.
isspace() : Return true if there are only whitespace characters in the string and there is at least one character, false otherwise.
istitle() Return true if the string is a titlecased string and there is at least one character
isupper() Return true if all cased characters in the string are uppercase and there is at least one cased character, false otherwise.
join() Return a string in which the parameter provided is joined by a separator
lower() Return a copy of the string converted to lowercase.
replace(old, new[, count]) Return a copy of the string with all occurrences of substring old replaced by new. count is optional.
split([sep [,maxsplit]]) Return a list of the words in the string, using sep as the delimiter string.
splitlines ([keepends]) Return a list of the lines in the string, breaking at line boundaries.
startswith (prefix[, start[, end]]) Return True if string starts with the prefix, otherwise return False.
strip ([chars]) Return a copy of the string with the leading and trailing characters char removed.
upper() Return a copy of the string converted to uppercase.

append( )

Add item to the end of a list

[Example]

```
myList = [‘a’, ‘b’, ‘c’, ‘d’]
myList.append(‘e’)
print (myList)
=> ['a', 'b', 'c', 'd', 'e']
```

del

Remove items from a list

[Example]

```
myList = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’, ‘k’, ‘l’]

#delete the third item (index = 2)
del myList[2]
print (myList)
=> ['a', 'b', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l']
```

extend( )

Combine two lists

[Example]

```
myList = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
myList2 = [1, 2, 3, 4]
myList.extend(myList2)
print (myList)
=> ['a', 'b', 'c', 'd', 'e', 1, 2, 3, 4]
```

In

Check if an item is in a list

[Example]

```
myList = [‘a’, ‘b’, ‘c’, ‘d’]
‘c’ in myList
=> True

‘e’ in myList
=> False
```

insert( )

Add item to a list at a particular position

[Example]

```
myList = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
myList.insert(1, ‘Hi’)
print (myList)
=> ['a', 'Hi', 'b', 'c', 'd', 'e']
```

len( )

Find the number of items in a list

[Example]

```
myList = [‘a’, ‘b’, ‘c’, ‘d’]
print (len(myList))
=> 4
```

pop( )

Get the value of an item and remove it from the list
Requires index of item as the parameter

[Example]

```
myList = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]

#remove the third item
member = myList.pop(2)
print (member)
=> c
```

remove( )

Remove an item from a list. Requires the value of the item as the parameter.

[Example]

```
myList = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]

#remove the item ‘c’
myList.remove(‘c’)
print (myList)
=> [‘a’, ‘b’, ‘d’, ‘e’]
```

reverse()

Reverse the items in a list

[Example]

```
myList = [1, 2, 3, 4]
myList.reverse()
print (myList)
=> [4, 3, 2, 1]
```

sort()

Sort a list alphabetically or numerically

[Example]

```
myList = [3, 0, -1, 4, 6]
myList.sort()
print(myList)
=> [-1, 0, 3, 4, 6]
```

sorted()

Return a new sorted list without sorting the original list. Requires a list as the parameter

[Example]

```
myList = [3, 0, -1, 4, 6]
myList2 = sorted(myList)

#Original list is not sorted
print (myList)
=> [3, 0, -1, 4, 6]

#New list is sorted
print (myList2)
=> [-1, 0, 3, 4, 6]
```

Addition Operator: +

Concatenate List

[Example]

```
myList = [‘a’, ‘b’, ‘c’, ‘d’]
print (myList + [‘e’, ‘f’])
=> [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’]

print (myList)
=> [‘a’, ‘b’, ‘c’, ‘d’]
```

Multiplication Operator: *

Duplicate a list and concatenate it to the end of the list

[Example]

```
myList = [‘a’, ‘b’, ‘c’, ‘d’]
print (myList*3)
=> [‘a’, ‘b’, ‘c’, ‘d’, ‘a’, ‘b’, ‘c’, ‘d’, ‘a’, ‘b’, ‘c’, ‘d’]

print (myList)
=> [‘a’, ‘b’, ‘c’, ‘d’]
```

del

Delete the entire tuple

[Example]

```
myTuple = ('a', 'b', 'c', 'd')
del myTuple
print (myTuple)
=> NameError: name 'myTuple' is not defined
```

in

Check if an item is in a tuple

[Example]

```
myTuple = ('a', 'b', 'c', 'd')
'c' in myTuple
=> True

'e' in myTuple
=> False
```

len( )

Find the number of items in a tuple

[Example]

```
myTuple = ('a', 'b', 'c', 'd')
print (len(myTuple))
=> 4
```

Project : Our program will randomly set an arithmetic question for us to answer. If we get the answer wrong, the program will display the correct answer and ask if we want to try a new question. If we get it correct, the program will compliment us and ask if we want a new question. In addition, the program will keep track of our scores and save the scores in an external text file. After each question, we can key "-1" to terminate the program

Exercise 1

```
from random import randint
from os import remove, rename
```

Exercise 2

```
def getUserScore(userName):
    try:
        input = open('userScores.txt', 'r')
        for line in input:
            content = line.split(',')
            if content[0] == userName:
                input.close()
                return content[1]
        input.close()
        return "-1"
    except IOError:
        print ("\nFile userScores.txt not found. A new file will be
    created.")
        input = open('userScores.txt', 'w')
        input.close()
        return "-1"
```

Exercise 3

```
def updateUserPoints(newUser, userName, score):
    if newUser:
        input = open('userScores.txt', 'a')
        input.write('\n' + userName + ', ' + score)
        input.close()
    else:
        input = open('userScores.txt', 'r')
        output = open('userScores.tmp', 'w')
    for line in input:
        content = line.split(',')
        if content[0] == userName:
            content[1] = score
            line = content[0] + ', ' + content[1] + '\n'

        output.write(line)
    input.close()
    output.close()

    remove('userScores.txt')
```

```
    rename('userScores.tmp', 'userScores.txt')
```

Exercise 4

```
def generateQuestion():
    operandList = [0, 0, 0, 0, 0]
    operatorList = ['', '', '', '']
    operatorDict = {1:' + ', 2:' - ', 3:'*', 4:'**'}

    for index in range(0, 5):
        operandList[index] = randint(1, 9)

    for index in range(0, 4):
        if index > 0 and operatorList[index-1] != '**':
                operator = operatorDict[randint(1, 4)]
          else:
                operator = operatorDict[randint(1, 3)]

          operatorList[index] = operator

    questionString = str(operandList[0])

    for index in range(1, 5):
          questionString = questionString + operatorList[index-1] +
str(operandList[index])

    result = eval(questionString)

    questionString = questionString.replace("**", "^")

    print ('\n' + questionString)

    userResult = input('Answer: ')

    while True:
          try:
                if int(userResult) == result:
                      print ("So Smart")
                      return 1
                else:
                      print ("Sorry, wrong answer. The correct answer
is", result)
                      return 0
```

```
          except Exception as e:
                print ("You did not enter a number. Please try
again.")
                userResult = input('Answer: ')
```

[Explanation for Exercise 4.2]

Starting from the second item (i.e. index = 1) in `operatorList`, the line `if index > 0 and operatorList[index-1] != '**'`: checks if the previous item in `operatorList` is the '**' symbol..
If it is not, the statement `operator = operatorDict[randint(1, 4)]` will execute. Since the range given to the `randint` function is 1 to 4, the numbers 1, 2, 3 or 4 will be generated. Hence, the symbols '+', '-', '*' or '**' will be assigned to the variable `operator`.

However, if the previous symbol is '**', the else statement (`operator = operatorDict[randint(1, 3)]`) will execute. In this case, the range given to the `randint` function is from 1 to 3. Hence, the '**' symbol, which has a key of 4 in `operatorDict` will NOT be assigned to the `operator` variable.

Exercise 5

```
try:

    import myPythonFunctions as m

    userName = input('''Please enter your user name or
create a new one if this is the first time
you are running the program: ''')

    userScore = int(m.getUserScore(userName))

    if userScore == -1:
          newUser = True
          userScore = 0
    else:
          newUser = False

    userChoice = 0

    while userChoice != '-1':

          userScore += m.generateQuestion()
          print ("Current Score = ", userScore)
          userChoice = input("Press Enter To Continue or -1 to Exit:
")

    m.updateUserPoints(newUser, userName, str(userScore))

except Exception as e:
    print ("An unexpected error occurred. Program will be exited.")
```