<u>Experiment 01</u>

1.1) Write a python program to understand basic data types, operators, expressions, input/output statements with format function.

1.2) Write a python program to understand control flow statements: conditional statements (if, if...else, nested if).

1.3) Looping in python (while loop, for loop, nested loop)

Roll No.	61
Name	V Krishnasubramaniam
Class	D10-A
Subject	Python Lab
LO Mapped	LO1: Understand the structure, syntax, and semantics of the Python language LO2: Interpret advanced data types and functions in python

<u>Aim</u>:

1.1) Write a python program to understand basic data types, operators, expressions, input/output statements with format function.

1.2) Write a python program to understand control flow statements: conditional statements (if, if...else, nested if).

1.3) Looping in python (while loop, for loop, nested loop)

<u>Introduction</u>:

Data Types in Python

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are several basic data types like Numeric, String, and Boolean types that are built into Python.

<u>Numeric</u>:

In Python, numeric data type represents the data which has numeric value. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.

1. **Integers**
   This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be. Python interprets a decimal digit without any prefix to be a decimal number.

   <u>Examples:</u>
   ```
   a = 10
   print(a)                #Output: 10
   print(type(a))          #Output: <class 'int'>
   ```

   In Python, integers can be prepended with the following strings to indicate a base other than 10:

Prefix	Interpreted as	Example
0b (zero + lowercase letter 'b') 0B (zero + uppercase letter 'B')	Binary Base: 2	print(0o10) #Output: 8 print(type(0o10)) #Output:<class 'int'>
0o (zero + lowercase letter 'o') 0O (zero + uppercase letter 'O')	Octal Base: 8	print(0x10) #Output: 16 print(type(0x10)) #Output:<class 'int'>
0x (zero + lowercase letter 'x') 0X (zero + uppercase letter 'X')	Hexadecimal Base: 16	print(0b10) #Output: 2 print(type(0b10)) #Output:<class 'int'>

2. **Float**

This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.

Examples:

```
print(type(4.2))        #Output: <class 'float'>
print(4.)               #Output: 4.0
print(.2)               #Output: 0.2
print(.4e7)             #Output: 4000000.0
print(4.2e-4)           #Output: 0.00042
```

The maximum value a floating-point number can have is approximately 1.8 ✕ 10308. Python will indicate a number greater than that by the string inf. The closest a nonzero number can be to zero is approximately 5.0 ✕ 10-324. Anything closer to zero than that is effectively zero.

Examples:

```
print(1.79e308)         #Output: 1.79e+308
print(1.8e308)          #Output: inf
print(5e-324)           #Output: 5e-324
print(1e-325)           #Output: 0.0
```

3. **Complex Numbers**

Complex number is represented by complex class. Python complex number can be created either using direct assignment statement or by using complex () function.

Example:

```
c = 2 + 4j
print(c)                #Output: (2+4j)
print(type(c))          #Output: <class 'complex'>
c = complex(1,3)
print(c)                #Output: (1+3j)
print(type(c))          #Output: <class 'complex'>
```

Each complex number consist of one real part and one imaginary part, which can be access using the real and imaginary data members, respectively. conjugate() function can be used to get the conjugate of the complex number.

Example:

```
c = 3 + 6j
print(c.real)           #Output: 3.0
print(c.imag)           #Output: 6.0
print(c.conjugate())    #Output: (3-6j)
```

We can do simple mathematical calculations on complex numbers.

Example:

```
    c1 = 3 + 6j
    c2 = 6 + 15j
    print(c1+c2)              #Output: (9+21j)
    print(c1-c2)              #Output: (-3-9j)
    print(c1*c2)              #Output: (-72+81j)
    print(c1/c2)              #Output: (0.4137931034482759-0.034482758620689664j)
```

Strings:

In Python, Strings are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class. Strings in Python can be created using single quotes or double quotes or even triple quotes (for multi-line string).

Example:

```
s= 'Hello World'
print(s)                      #Output: Hello World
s= "Hello World"
print(s)                      #Output: Hello World
s= '''Geeks
    For
    Life'''
print(s)
#Output:
Geeks
    For
    Life
```

In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String. While accessing an index out of the range will cause an IndexError. Only Integers are allowed to be passed as an index, float or other types will cause a TypeError. To access a range of characters in the String, method of slicing is used. Slicing in a String is done by using a Slicing operator (colon).

Example:

```
s= 'Hello World'
print(s[0])                   #Output: H
print(s[-1])                  #Output: d
print(s[2:7])                 #Output: llo W
```

In Python, updation or deletion of characters from a String is not allowed. This is because Strings are immutable, hence elements of a String cannot be changed once it has been assigned. Only new strings can be reassigned to the same name. Deletion of entire string is possible with the use of del keyword.

Boolean:

Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). But non-Boolean objects can be evaluated in Boolean context as well and determined to be true or false. It is denoted by the class bool.

Example:

```
s = True
print(type(s))                    #Output: <class 'bool'>
```

Generally, booleans values are returned as a result of some sort of comparison.

Example:

```
a = 10
b = 20
print(a==b)                       #Output: False
print(a<b)                        #Output: True
```

Numbers can be used as bool values by using Python’s built-in bool() method. Any integer, floating-point number, or complex number having zero as a value is considered as False, while if they are having value as any positive or negative number then it is considered as True.

Example:

```
a = 0
b = 10
c = -9.6
print(bool(a))                    #Output: False
print(bool(b))                    #Output: True
print(bool(c))                    #Output: True
```

Boolean Operations are simple arithmetic of True and False values. These values can be manipulated by the use of boolean operators which include AND, Or, and NOT.

Example:

```
a = True
b = False
print(a or b)                     #Output: True
print(a and b)                    #Output: False
print(not a)                      #Output: False
```

Operators in Python

Operators are the constructs which can manipulate the value of operands. Operators in general are used to perform operations on values and variables in Python. These are standard symbols used for the purpose of logical and arithmetic operations.

Types of Operators: Python language supports the following types of operators.

1. **Arithmetic Operators**
   Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division.

Operator	Description	Syntax	Example	
+	Addition	x + y	print(9 + 4)	#Output: 13
-	Subtraction	x - y	print(9 - 4)	#Output: 5
*	Multiplication	x * y	print(9 + 4)	#Output: 36
/	Division (float)	x / y	print(9/4)	#Output: 2.25
//	Division (floor)	x // y	print(9//4)	#Output: 2
%	Modulus (remainder)	x % y	print(9%4)	#Output: 1
**	Power	x ** y	print(2**3)	#Output: 8

2. **Relational Operators**
   Relational operators compares the values. It either returns True or False according to the condition.

Operator	Description	Syntax	Example	
>	Greater than	x > y	print(9 > 4)	#Output: True
<	Less than	x < y	print(9 < 4)	#Output: False
==	Equal to	x == y	print(9 == 4)	#Output: False
!=	Not equal to	x != y	print(9 != 4)	#Output: True

>=	Greater than or equal to	x >= y	print(9 >= 4)
			print(4 >= 4)
				#Output: True
				#Output: True

<=	Less than or equal to	x <= y	print(9 <= 4)
			print(9 >= 9)
				#Output: False
				#Output: True

3. **Logical Operators**
   Logical operators perform Logical AND, Logical OR and Logical NOT operations.

Operator	Description	Syntax	Example	
and	Logical AND	x > y	print(True and False)	#Output: False
or	Logical OR	x < y	print(True or False)	#Output: True
not	Logical NOT	x == y	print(not True)	#Output: False

4. **Bitwise Operators**
   Bitwise operator works on bits and performs bit by bit operation.
   <u>Example:</u>

```
a = 60
b = 13
c = 0
print(c)                    #Output: 0
c = a&b
print(c)                    #Output: 12
c = a|b
```

```
print(c)                    #Output: 61
c = a^b
print(c)                    #Output: 49
c = ~a
print(c)                    #Output: -61
c = a<<2
print(c)                    #Output: 240
c = a>>2
print(c)                    #Output: 15
```

Operator	Description	Syntax
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b)
| Binary OR	It copies a bit if it exists in either operand.	(a | b)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a)
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2

5. Assignment Operators

Assignment operators are used to assign values to the variables.

Example:

```
a = 10
b = a
c=1
c+=b
print(a," ",b," ",c)        #Output: 10 10 11
```

Operator	Description	Syntax
=	Assign value of right side of expression to left side operand	x = y+z
+=	Add AND: Add right side operand with left side operand and then assign to left operand	a+=b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b
%=	Modulus AND: Takes modulus using left and right operands and assign result to left operand	a%=b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b

**=	Exponent AND: Calculate exponent (raise power) value using operands and assign value to left operand	a**=b
&=	Performs Bitwise AND on operands and assign value to left operand	a&=b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b
^=	Performs Bitwise xOR on operands and assign value to left operand	a^=b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a>>=b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a <<= b

6. **Membership Operators**
   Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators:

Operator	Description	Explanation
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not find a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Example:

```
a = 10
b = 20
list = [1, 2, 3, 4, 5]
print(a in list)          #Output: False
print(b not in list)      #Output: True
```

7. **Identity Operators**
   Identity operators compare the memory locations of two objects. There are two Identity operators:

Operator	Description	Explanation
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

Example:

```
a = 20
b = 20
print(a is b)             #Output: True
print(a is not b)         #Output: False
```

format() function in Python

With Python 3.0, the format() method has been introduced for handling complex string formatting more efficiently. This method of the built-in string class provides functionality for complex variable substitutions and value formatting. This new formatting technique is regarded as more elegant. The general syntax of format() method is:

string.format(var1, var2,…)

Formatters work by putting in one or more replacement fields and placeholders defined by a pair of curly braces { } into a string and calling the str.format(). The value we wish to put into the placeholders and concatenate with the string passed as parameters into the format function.

Syntax: {}.format(value)

Examples:

```
print ("{} world".format("Hello"))              #Output: Hello world
str = "This text is written in {}"
print (str.format("Python"))                    #Output: This text is written in Python
print ("I am {} years old !".format(19))        #Output: I am  19 years old!
```

Multiple pairs of curly braces can be used while formatting the string. Let's say if another variable substitution is needed in the sentence, can be done by adding a second pair of curly braces and passing a second value into the method. Python will replace the placeholders with values in order. IndexError occurs when string has an extra placeholder, and we didn't pass any value for it in the format() method.

Syntax: {}{}.format(value1, value2)

Examples:

```
str = " This {} is written in {} "
print(str.format("text", "Python"))             #Output: This text is written in Python
print ("I am {} years {}! ".format(19,"old"))   #Output: I am  19 years old!
```

You can use two or more specially designated characters within a string to format a string or perform a command. These characters are called escape sequences. An Escape sequence in Python starts with a backslash (\). For example, \n is an escape sequence in which the common meaning of the letter n is literally escaped and given an alternative meaning – a new line.

Escape sequence	**Description**	**Example**
\n	Breaks the string into a new line	print('I designed this rhyme to explain\nAll I know')
\t	Adds a horizontal tab	print('Time is a \tvaluable thing')
\\	Prints a backslash	print('Watch it fly by\\as the pendulum swings')
\'	Prints a single quote	print('It doesn\'t even matter how hard you try')
\"	Prints a double quote	print('It is so\"unreal\"')

When placeholders {} are empty, Python will replace the values passed through str.format() in order. The values that exist within the str.format() method are essentially tuple data types and each individual value contained in the tuple can be called by its index number, which starts with the index number 0. These index numbers can be passed into the curly braces that serve as the placeholders in the original string.

Examples:

```
print ("I am {0} years {1}! ".format(19,"old"))          #Output: I am  19 years old!
print ("I am {1} years {0}! ".format(19,"old"))          #Output: I am old years 19!
```

More parameters can be included within the curly braces of our syntax. Use the format code syntax {field_name:conversion}, where field_name specifies the index number of the argument to the str.format() method, and conversion refers to the conversion code of the data type.

Examples:

```
print(“%20s” % (‘hello’, ))                              #Output:       hello
print("%.5s" % ('HelloWorld',))                          #Output: Hello
print("That costs %d rupees"%(1200))                     #Output: That costs 1200 rupees
print ("The average{0} was {1:.0f}%".format("rate", 78.2))#Output: The avarage rate was 78%
```

By default, strings are left-justified within the field, and numbers are right-justified. We can modify this by placing an alignment code just following the colon.

< : left-align text in the field
^ : center text in the field
> : right-align text in the field

Examples:

```
print("{0:^6} was launched in {1:<4}!".format("WWW", 1989))
#Output:  WWW   was launched in 1989!
print("%.5s" % ('HelloWorld',))                          #Output: Hello
print("{:*^20s}".format("Hello"))                        #Output: *******Hello********
```

Conditional/Decision-Making Statements in Python

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions. Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome.

Python programming language provides the following types of decision-making statements:

1. **if statement:**
   if statement is the simplest decision-making statement. The if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison. It is used to decide whether a certain statement or block of statements will be executed or not i.e. if a certain condition is true then a block of statement is executed otherwise not.

   Syntax:

   ```
   if expression:
           statement(s)
   ```

Example:

```
i = 10
if (i > 15):
        print ("10 is less than 15")
print ("Outside if block")
```

Output:

```
Outside if block
```

2. **if...else statement:**

The if statement alone tells us that if a condition is true, it will execute a block of statements and if the condition is false, it won't. But what if we want to do something else if the condition is false. Here comes the else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

Syntax:

```
if (condition):
      # Executes this block if condition is true
else:
      # Executes this block ifcondition is false
```

Example:

```
i = 20;
if (i < 15):
   print ("i is smaller than 15")
   print ("i'm in if Block")
else:
   print ("i is greater than 15")
   print ("i'm in else Block")
print ("i'm not in if and not in else Block")
```

Output:

```
i is greater than 15
i'm in else Block
i'm not in if and not in else Block
```

3. **if...elif...else statement:**

An else statement can be combined with an if statement. An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value. The else statement is an optional statement and there could be at most only one else statement following if.

The elif statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE. Similar to the else, the elif statement is optional. However, unlike else, for which there can be at most one statement, there can be an arbitrary number of elif statements following an if.

Syntax:

```
if expression1:
        statement(s)
```

```
elif expression2:
        statement(s)
elif expression3:
        statement(s)
else:
        statement(s)
```

Example:

```
i = 20
if (i == 10):
   print ("i is 10")
elif (i == 15):
   print ("i is 15")
elif (i == 20):
   print ("i is 20")
else:
   print ("i is not present")
```

Output:

```
i is 20
```

4. **Shorthand if-else statement**

   This can be used to write the if-else statements in a single line where there is only one statement to be executed in both if and else block.

   Syntax:

   ```
   statement_when_True if condition else statement_when_False
   ```

   Example:

   ```
   i = 10
   print(True) if i < 15 else print(False)
   ```

   Output:

   ```
   True
   ```

Looping Statements in Python

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times. Python provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times.

Python programming language provides the following types of loops to handle looping requirements:

1. **while loop**

   A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

   Syntax:

```
while expression:
     statement(s)
```

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.

Example:

```
count = 0
while (count < 3):
    count = count + 1
    print("Hello", count)
```

Output:

```
Hello 1
Hello 2
Hello 3
```

Python supports to have an else statement associated with a loop statement. If the else statement is used with a for loop, the else statement is executed when the loop has exhausted iterating the list. If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

Example:

```
count = 0
while count < 5:
    print(count, " is less than 5")
    count = count + 1
else:
    print(count, " is not less than 5")
```

Output:

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
```

2. **for loop**

For loops are used for sequential traversal. For example: traversing a list or string or array etc.

Syntax:

```
for iterating_var in sequence:
    statements(s)
```

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating_var. Next, the statements block is executed. Each item in the list is assigned to iterating_var, and the statement(s) block is executed until the entire sequence is exhausted.

Example:

```
for letter in 'Python':
   print('Current Letter :', letter)
fruits = ['banana', 'apple', 'mango']
for fruit in fruits:
   print('Current fruit :', fruit)
print("Good bye!")
```

Output:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

3. **nested loops**

Python programming language allows to use one loop inside another loop.

Syntax:

```
for iterator_var in sequence:
   for iterator_var in sequence:
      statements(s)
   statements(s)
```

A final note on loop nesting is that we can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

Example:

```
for i in range(1, 5):
   for j in range(i):
       print(i, end=' ')
   print()
```

Output:

```
1
2 2
3 3 3
4 4 4 4
```

Loop Control Statements in Python

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Python supports the following control statements:

1. **break statement**

   It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C. The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both while and for loops. If you are using nested loops, the break statement stops the execution of the innermost loop and start executing the next line of code after the block.

   Example:

   ```
   for letter in 'HelloWorld':
      #break the loop as soon it sees 'e' or 'l'
      if letter == 'e' or letter == 'l':
           break
   print ('Current Letter :', letter)
   ```

   Output:

   ```
   Current Letter : e
   ```

2. **continue statement**

   It returns the control to the beginning of the while loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop. The continue statement can be used in both while and for loops

   Example:

   ```
   for letter in 'HelloWorld':
      if letter == 'e' or letter == 'l':
           continue
      print ('Current Letter :', letter)
   ```

   Output:

   ```
   Current Letter : H
   Current Letter : o
   Current Letter : W
   Current Letter : o
   Current Letter : r
   Current Letter : d
   ```

3. **pass statement**

   It is used when a statement is required syntactically but you do not want any command or code to execute. The pass statement is a null operation; nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written yet.

   Example:

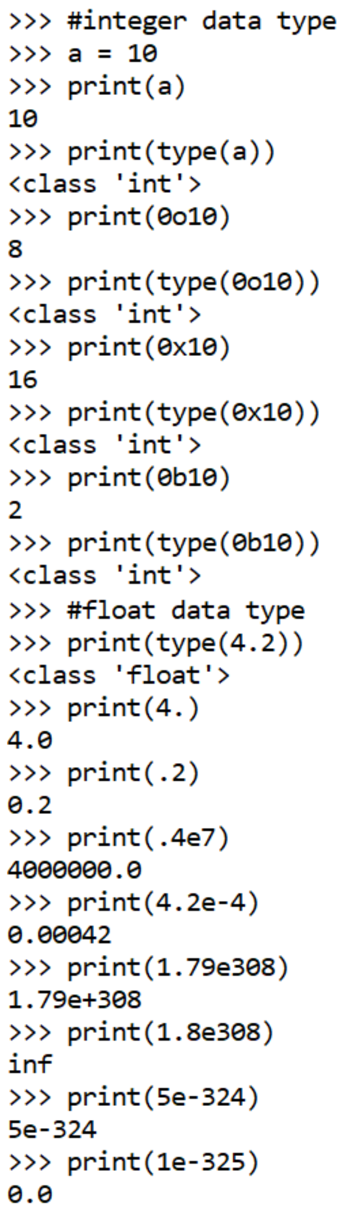
   ```
   for letter in 'HelloWorld':
      pass
   print ('Last Letter :', letter)
   ```

   Output:

   ```
   Last Letter : d
   ```

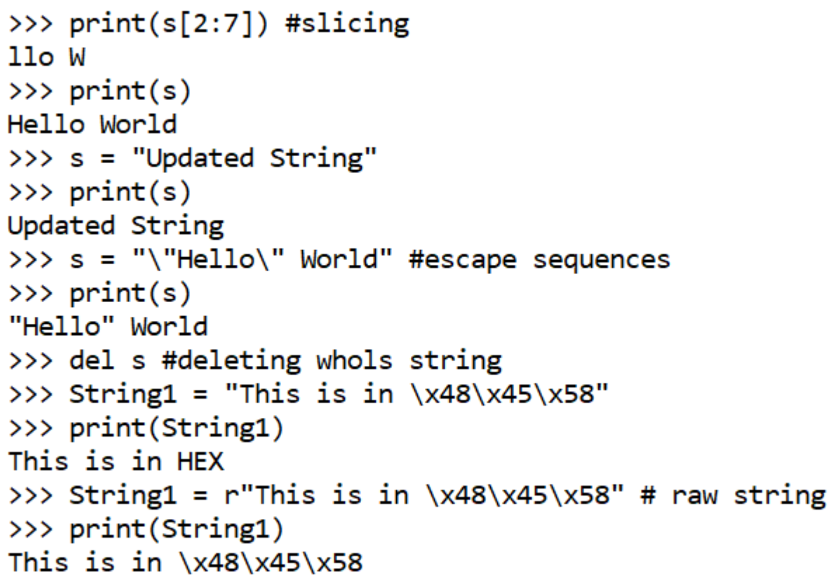
<u>Results</u>:

<u>1.1) Write a python program to understand basic data types, operators, expressions, input/output statements with format function.</u>

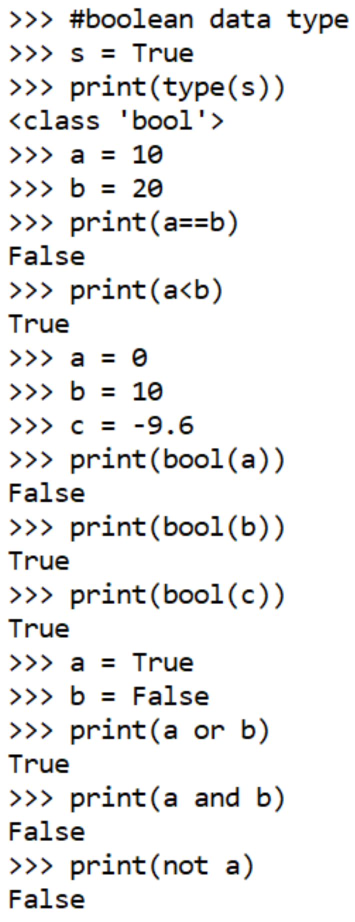


```
>>> #integer data type
>>> a = 10
>>> print(a)
10
>>> print(type(a))
<class 'int'>
>>> print(0o10)
8
>>> print(type(0o10))
<class 'int'>
>>> print(0x10)
16
>>> print(type(0x10))
<class 'int'>
>>> print(0b10)
2
>>> print(type(0b10))
<class 'int'>
>>> #float data type
>>> print(type(4.2))
<class 'float'>
>>> print(4.)
4.0
>>> print(.2)
0.2
>>> print(.4e7)
4000000.0
>>> print(4.2e-4)
0.00042
>>> print(1.79e308)
1.79e+308
>>> print(1.8e308)
inf
>>> print(5e-324)
5e-324
>>> print(1e-325)
0.0
```

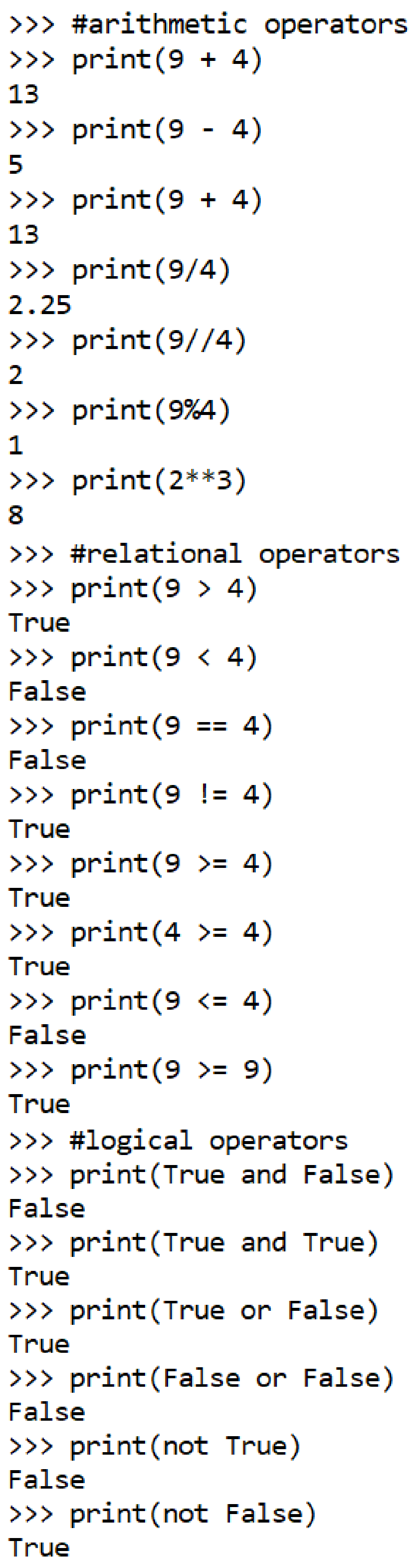
```
>>> #complex numbers data type
>>> c = 2 + 4j
>>> print(c)
(2+4j)
>>> print(type(c))
<class 'complex'>
>>> c = complex(1,3)
>>> print(c)
(1+3j)
>>> print(type(c))
<class 'complex'>
>>> c = 3 + 6j
>>> print(c.real)
3.0
>>> print(c.imag)
6.0
>>> print(c.conjugate())
(3-6j)
>>> c1 = 3 + 6j
>>> c2 = 6 + 15j
>>> print(c1+c2)
(9+21j)
>>> print(c1-c2)
(-3-9j)
>>> print(c1*c2)
(-72+81j)
>>> print(c1/c2)
(0.4137931034482759-0.03448275862068964j)
>>> #string data type
>>> s= 'Hello World'
>>> print(s)
Hello World
>>> print(type(s))
<class 'str'>
>>> s= "Hello World"
>>> print(s)
Hello World
>>> s= '''Geeks
...         For
...         Life'''
>>> s= 'Hello World'
>>> print(s[0])
H
>>> print(s[-1])
d
```



```
>>> print(s[2:7]) #slicing
llo W
>>> print(s)
Hello World
>>> s = "Updated String"
>>> print(s)
Updated String
>>> s = "\"Hello\" World" #escape sequences
>>> print(s)
"Hello" World
>>> del s #deleting whols string
>>> String1 = "This is in \x48\x45\x58"
>>> print(String1)
This is in HEX
>>> String1 = r"This is in \x48\x45\x58" # raw string
>>> print(String1)
This is in \x48\x45\x58
```



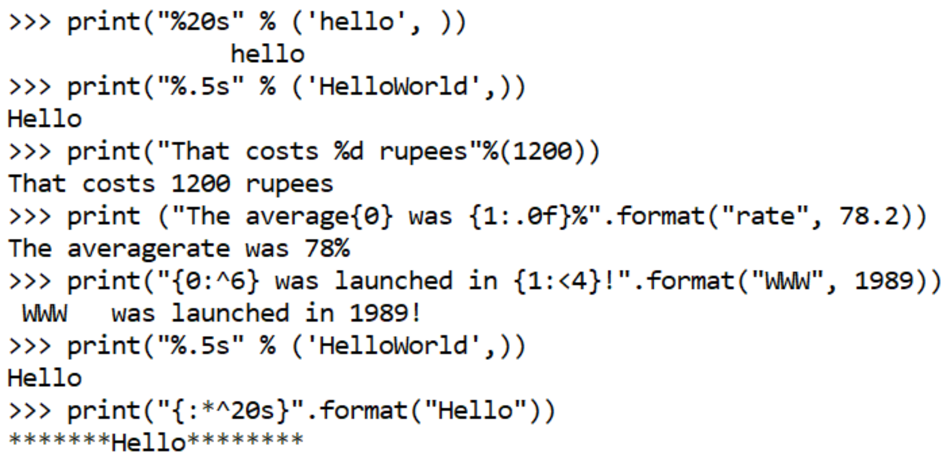
```
>>> #boolean data type
>>> s = True
>>> print(type(s))
<class 'bool'>
>>> a = 10
>>> b = 20
>>> print(a==b)
False
>>> print(a<b)
True
>>> a = 0
>>> b = 10
>>> c = -9.6
>>> print(bool(a))
False
>>> print(bool(b))
True
>>> print(bool(c))
True
>>> a = True
>>> b = False
>>> print(a or b)
True
>>> print(a and b)
False
>>> print(not a)
False
```



```
>>> #arithmetic operators
>>> print(9 + 4)
13
>>> print(9 - 4)
5
>>> print(9 + 4)
13
>>> print(9/4)
2.25
>>> print(9//4)
2
>>> print(9%4)
1
>>> print(2**3)
8
>>> #relational operators
>>> print(9 > 4)
True
>>> print(9 < 4)
False
>>> print(9 == 4)
False
>>> print(9 != 4)
True
>>> print(9 >= 4)
True
>>> print(4 >= 4)
True
>>> print(9 <= 4)
False
>>> print(9 >= 9)
True
>>> #logical operators
>>> print(True and False)
False
>>> print(True and True)
True
>>> print(True or False)
True
>>> print(False or False)
False
>>> print(not True)
False
>>> print(not False)
True
```

```
>>> #assignment operators
>>> a=10
>>> b=a
>>> print(a)
10
>>> print(b)
10
>>> c=1
>>> print(c)
1
>>> c+=100
>>> print(c)
101
>>> c-=5
>>> print(c)
96
>>> c*=6
>>> print(c)
576
>>> c/=2
>>> print(c)
288.0
>>> c%=7
>>> print(c)
1.0
>>> c**=2
>>> print(c)
1.0
>>> #bitwise operators
>>> a = 10
>>> b = 4
>>> print(a&b)
0
>>> print(a|b)
14
>>> print(~a)
-11
>>> print(a^b)
14
>>> print(a>>2)
2
>>> print(a<<2)
40
```

```
>>> #membership operator
>>> a = 10
>>> b = 20
>>> list = [1, 2, 3, 4, 5]
>>> print(a in list)
False
>>> print(a not in list)
True
>>> print(b in list)
False
>>> print(b not in list)
True
>>> #identity operators
>>> a = 20
>>> b = 20
>>> c = 30
>>> print(a is b)
True
>>> print(a is c)
False
>>> print(a is not c)
True
>>> print(a is not b)
False
>>> #input and output statements
>>> name = input('Enter your name: ')
Enter your name: Krishna
>>> print(name)
Krishna
>>> num = int(input('Enter your age: '))
Enter your age: 19
>>> print(num)
19
>>> #format function
>>> print ("{} world".format("Hello"))
Hello world
>>> str = "This text is written in {}"
>>> print (str.format("Python"))
This text is written in Python
>>> print ("I am {} years old !".format(19))
I am 19 years old !
>>> str = " This {} is written in {} "
>>> print(str.format("text", "Python"))
 This text is written in Python
>>> print ("I am {} years {}! ".format(19,"old"))
I am 19 years old!
>>> print ("I am {0} years {1}! ".format(19,"old"))
I am 19 years old!
>>> print ("I am {1} years {0}! ".format(19,"old"))
I am old years 19!
```



```
>>> print("%20s" % ('hello', ))
               hello
>>> print("%.5s" % ('HelloWorld',))
Hello
>>> print("That costs %d rupees"%(1200))
That costs 1200 rupees
>>> print ("The average{0} was {1:.0f}%".format("rate", 78.2))
The averagerate was 78%
>>> print("{0:^6} was launched in {1:<4}!".format("WWW", 1989))
 WWW   was launched in 1989!
>>> print("%.5s" % ('HelloWorld',))
Hello
>>> print("{:*^20s}".format("Hello"))
*******Hello********
```

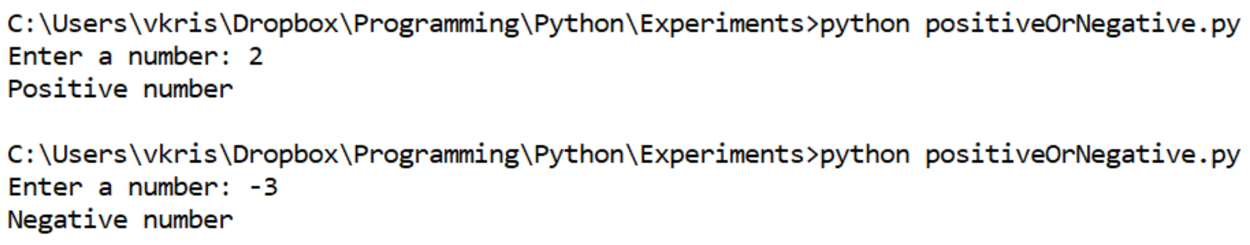
<u>1.2) Write python programs to understand control flow statements: conditional statements (if, if...else, nested if)</u>

1. Python Program to Check if a Number is Positive, Negative or 0

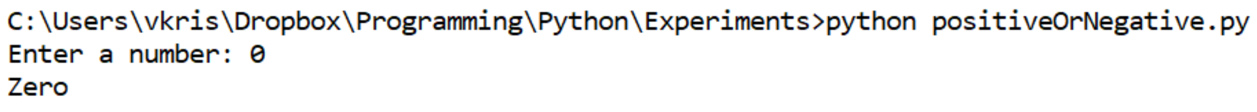
<u>Program in positiveOrNegative.py:</u>

```
num = float(input("Enter a number: "))
if num > 0:
  print("Positive number")
elif num == 0:
  print("Zero")
else:
  print("Negative number")
```

<u>Output:</u>



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python positiveOrNegative.py
Enter a number: 2
Positive number

C:\Users\vkris\Dropbox\Programming\Python\Experiments>python positiveOrNegative.py
Enter a number: -3
Negative number
```



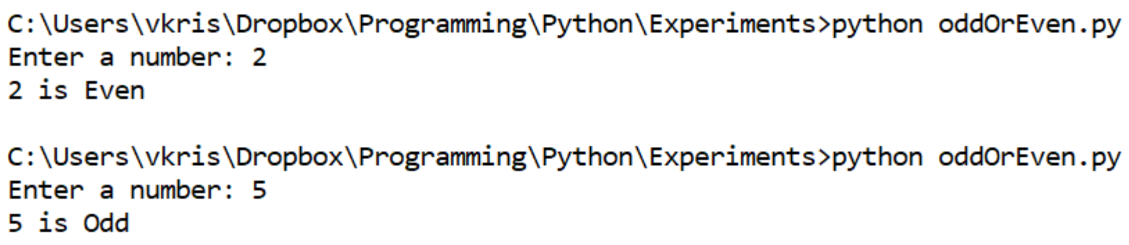
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python positiveOrNegative.py
Enter a number: 0
Zero
```

2. Python Program to Check if a Number is Odd or Even

<u>Program in oddOrEven.py:</u>

```
num = int(input("Enter a number: "))
if (num % 2) == 0:
  print("{0} is Even".format(num))
else:
  print("{0} is Odd".format(num))
```

Output:



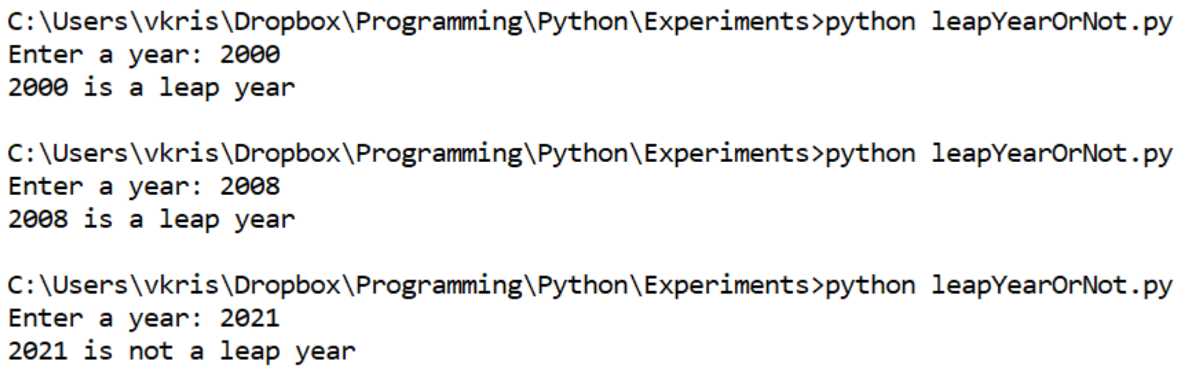
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python oddOrEven.py
Enter a number: 2
2 is Even

C:\Users\vkris\Dropbox\Programming\Python\Experiments>python oddOrEven.py
Enter a number: 5
5 is Odd
```

3. Python Program to Check Leap Year

Program in leapYearOrNot.py:

```
year = int(input("Enter a year: "))
if (year % 4) == 0:
   if (year % 100) == 0:
       if (year % 400) == 0:
           print("{0} is a leap year".format(year))
       else:
           print("{0} is not a leap year".format(year))
   else:
       print("{0} is a leap year".format(year))
else:
   print("{0} is not a leap year".format(year))
```

Output:



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python leapYearOrNot.py
Enter a year: 2000
2000 is a leap year

C:\Users\vkris\Dropbox\Programming\Python\Experiments>python leapYearOrNot.py
Enter a year: 2008
2008 is a leap year

C:\Users\vkris\Dropbox\Programming\Python\Experiments>python leapYearOrNot.py
Enter a year: 2021
2021 is not a leap year
```

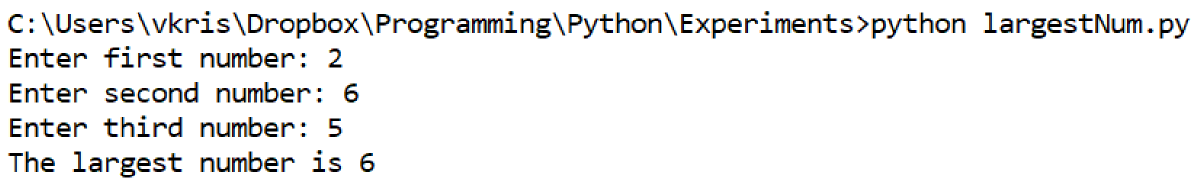
4. Python Program to Find the Largest Among Three Numbers

Program in largestNum.py:

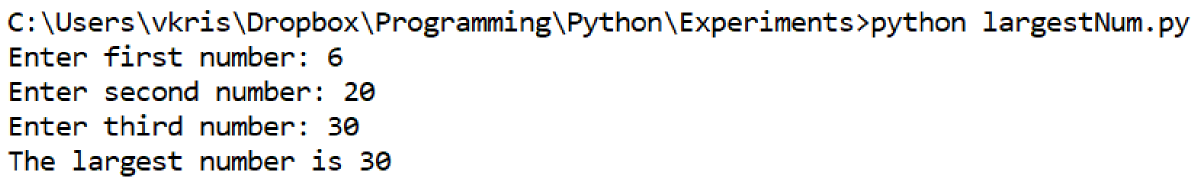
```
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
num3 = int(input("Enter third number: "))
if (num1 >= num2) and (num1 >= num3):
   largest = num1
```

```
elif (num2 >= num1) and (num2 >= num3):
   largest = num2
else:
   largest = num3
print("The largest number is", largest)
```

<u>Output:</u>



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python largestNum.py
Enter first number: 2
Enter second number: 6
Enter third number: 5
The largest number is 6
```



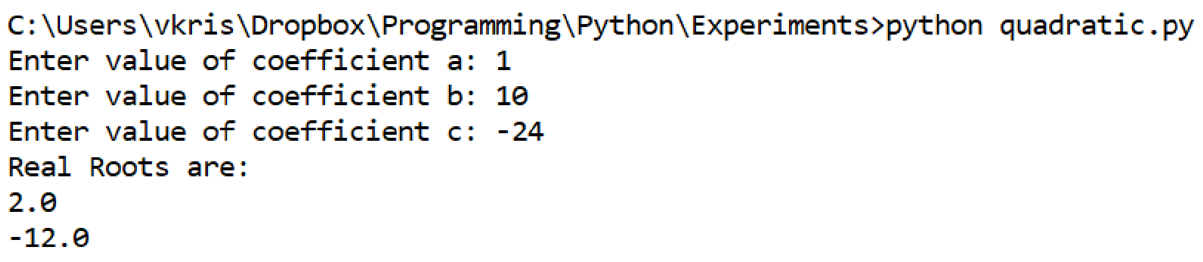
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python largestNum.py
Enter first number: 6
Enter second number: 20
Enter third number: 30
The largest number is 30
```

5. Python program that prints all real solutions to the quadratic equation ax2+bx+c = 0. Read in a, b, c and use the quadratic formula. If the discriminate b2-4ac is negative, display a message stating that there are no real solutions.

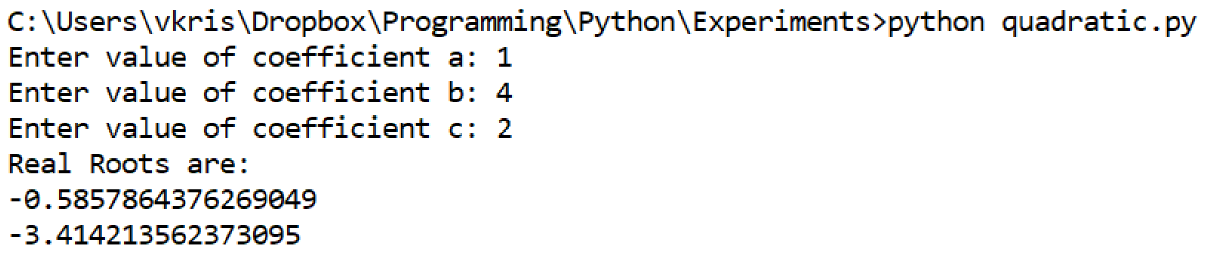
<u>Program in quadratic.py:</u>

```
import math

a = int(input("Enter value of coefficient a: "))
b = int(input("Enter value of coefficient b: "))
c = int(input("Enter value of coefficient c: "))
if a==0:
    print("Invalid Quadratic Equation!")
else:
    dis = b * b - 4 * a * c
    sqrt_val = math.sqrt(abs(dis))
    if dis >= 0:
        print("Real Roots are:")
        print((-b + sqrt_val)/(2 * a))
        print((-b - sqrt_val)/(2 * a))
    else:
        print("There are no real roots!")
```

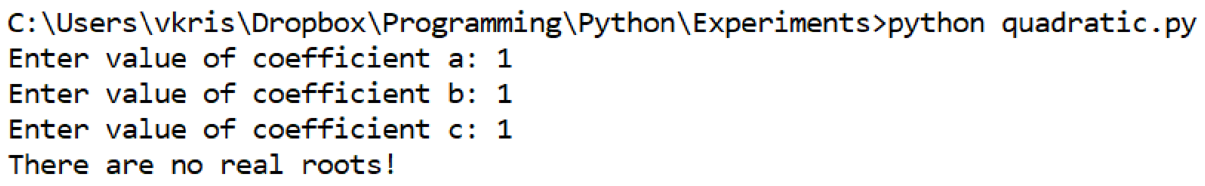
<u>Output:</u>



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python quadratic.py
Enter value of coefficient a: 1
Enter value of coefficient b: 10
Enter value of coefficient c: -24
Real Roots are:
2.0
-12.0
```



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python quadratic.py
Enter value of coefficient a: 1
Enter value of coefficient b: 4
Enter value of coefficient c: 2
Real Roots are:
-0.5857864376269049
-3.414213562373095
```



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python quadratic.py
Enter value of coefficient a: 1
Enter value of coefficient b: 1
Enter value of coefficient c: 1
There are no real roots!
```

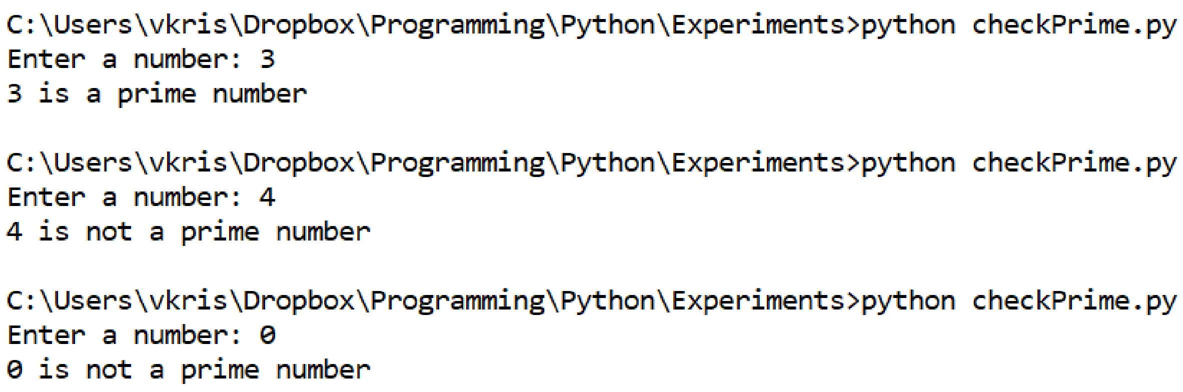
1.3) Write python programs to understand Looping in python(while loop, for loop, nested loop)

1. Python Program to Check Prime Number

Program in checkPrime.py:

```
num = int(input("Enter a number: "))
if num > 1:
   for i in range(2,num):
       if (num % i) == 0:
           print(num,"is not a prime number")
           break
   else:
       print(num,"is a prime number")
else:
   print(num,"is not a prime number")
```

Output:



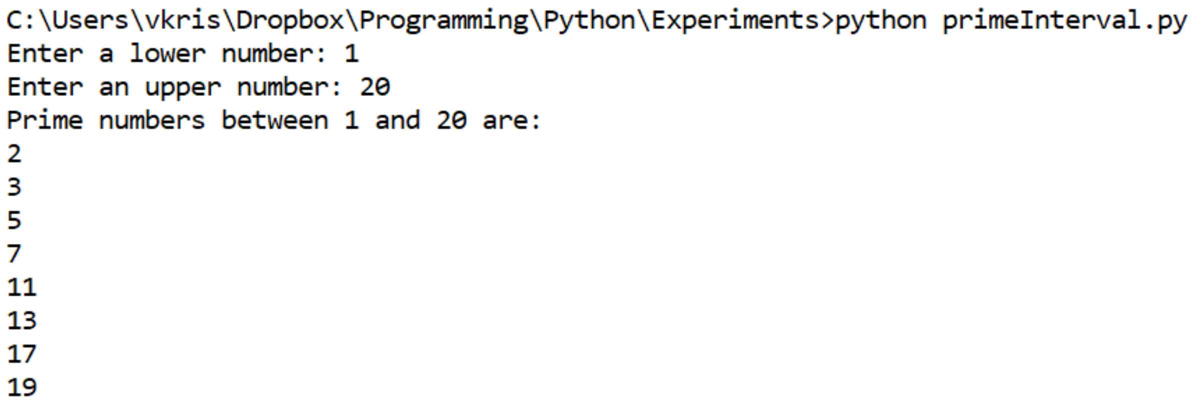
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python checkPrime.py
Enter a number: 3
3 is a prime number

C:\Users\vkris\Dropbox\Programming\Python\Experiments>python checkPrime.py
Enter a number: 4
4 is not a prime number

C:\Users\vkris\Dropbox\Programming\Python\Experiments>python checkPrime.py
Enter a number: 0
0 is not a prime number
```

2. Python Program to Print all Prime Numbers in an Interval

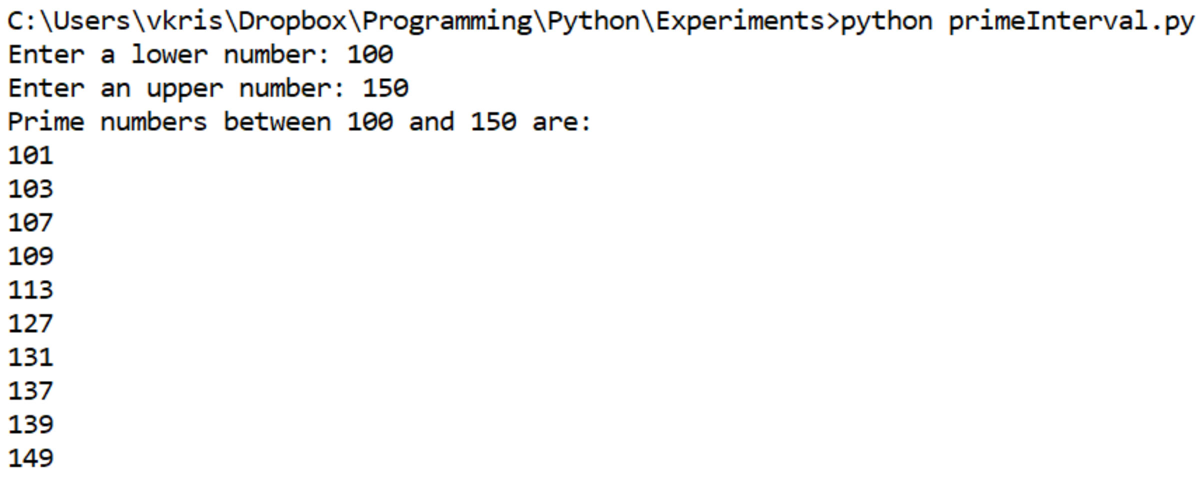
Program in primeInterval.py:

```
lower = int(input("Enter a lower number: "))
upper = int(input("Enter an upper number: "))
print("Prime numbers between", lower, "and", upper, "are:")

for num in range(lower, upper + 1):
   if num > 1:
       for i in range(2, num):
           if (num % i) == 0:
               break
       else:
           print(num)
```

Output:



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python primeInterval.py
Enter a lower number: 1
Enter an upper number: 20
Prime numbers between 1 and 20 are:
2
3
5
7
11
13
17
19
```



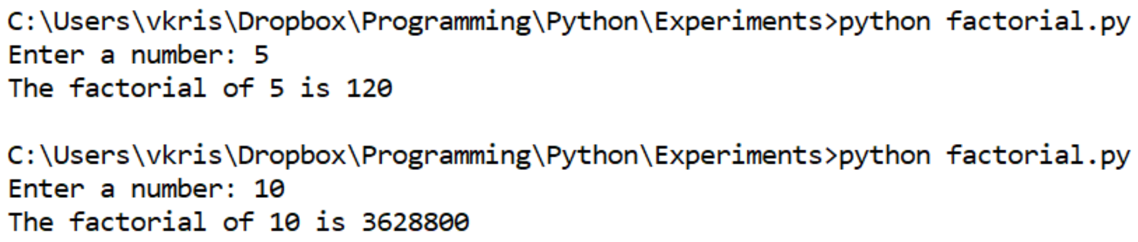
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python primeInterval.py
Enter a lower number: 100
Enter an upper number: 150
Prime numbers between 100 and 150 are:
101
103
107
109
113
127
131
137
139
149
```

3. Python Program to Find the Factorial of a Number

Program in factorial.py:

```
num = int(input("Enter a number: "))
factorial = 1

if num < 0:
   print("Factorial does not exist for negative numbers")
elif num == 0:
   print("The factorial of 0 is 1")
else:
   for i in range(1,num + 1):
       factorial = factorial*i
   print("The factorial of",num,"is",factorial)
```

Output:



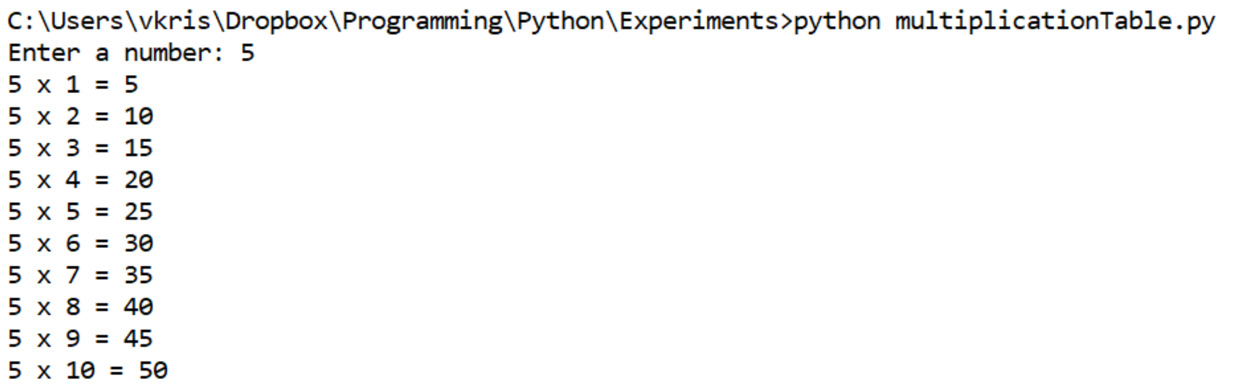
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python factorial.py
Enter a number: 5
The factorial of 5 is 120

C:\Users\vkris\Dropbox\Programming\Python\Experiments>python factorial.py
Enter a number: 10
The factorial of 10 is 3628800
```

4. Python Program to Display the multiplication Table

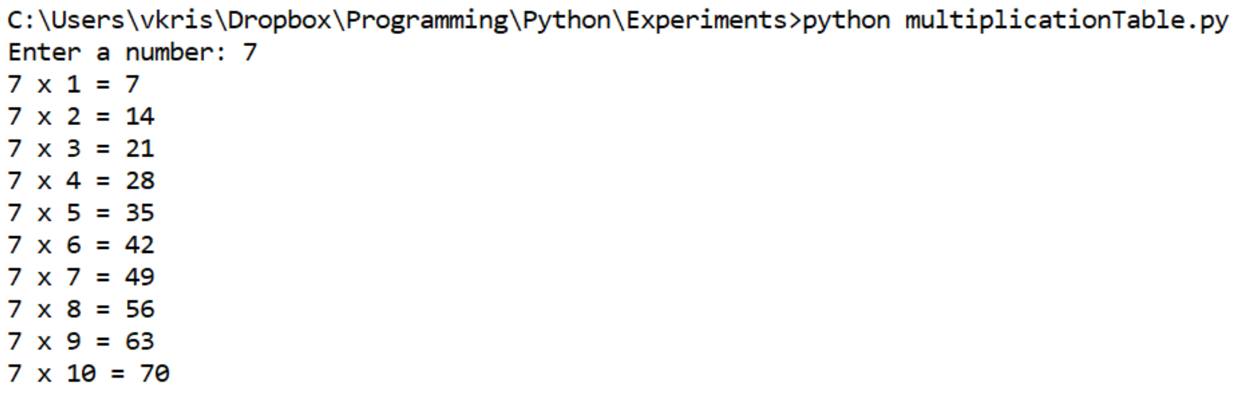
Program in multiplicationTable.py:

```
num = int(input("Enter a number: "))

for i in range(1, 11):
   print(num, 'x', i, '=', num*i)
```

Output:



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python multiplicationTable.py
Enter a number: 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```



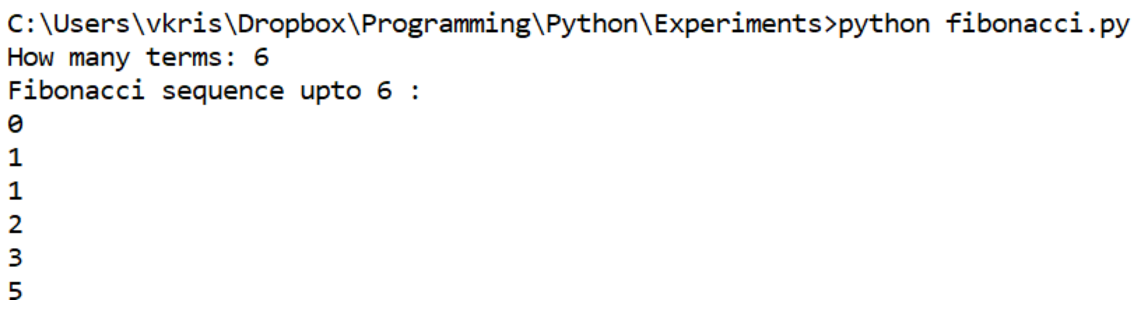
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python multiplicationTable.py
Enter a number: 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

5. Python Program to Print the Fibonacci sequence

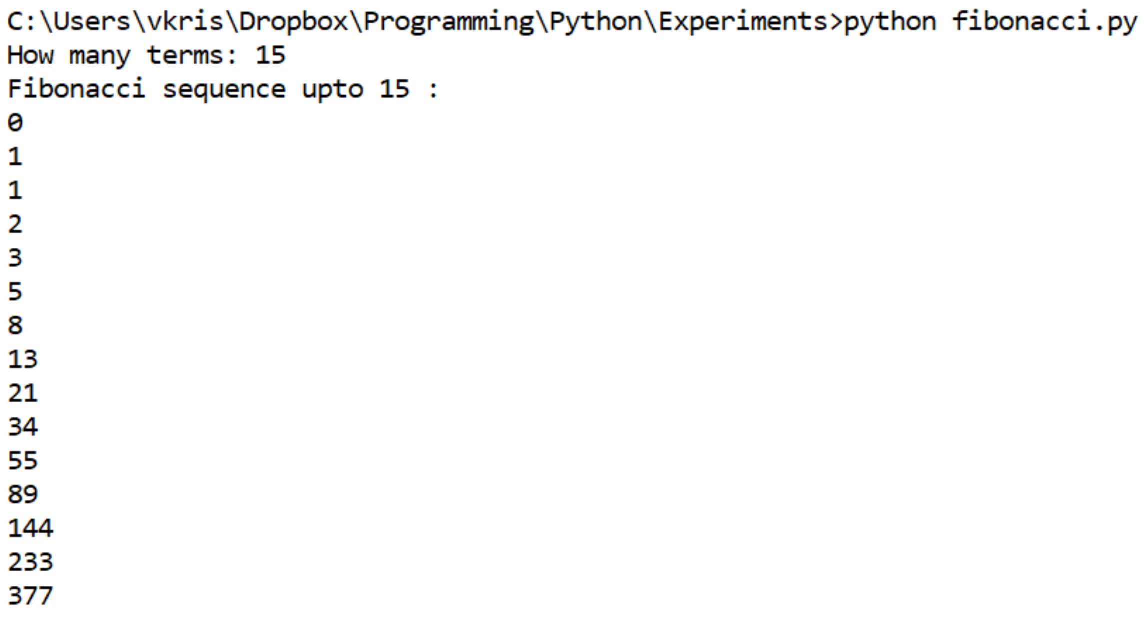
Program in fibonacci.py:

```
nterms = int(input("How many terms: "))

n1, n2 = 0, 1
count = 0

if nterms <= 0:
   print("Please enter a positive integer")
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
else:
   print("Fibonacci sequence upto",nterms,":")
   while count < nterms:
       print(n1)
       nth = n1 + n2
       n1 = n2
       n2 = nth
       count += 1
```

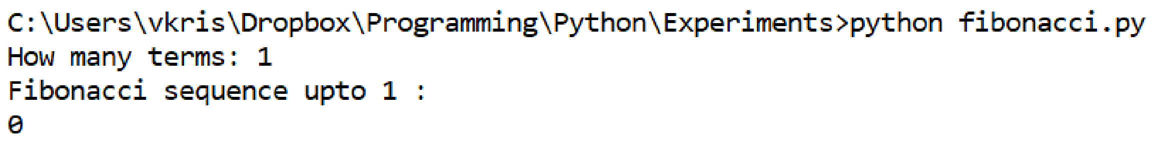
Output:



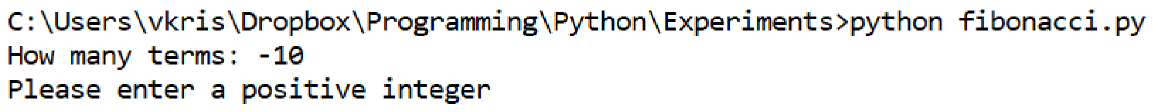
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python fibonacci.py
How many terms: 6
Fibonacci sequence upto 6 :
0
1
1
2
3
5
```



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python fibonacci.py
How many terms: 15
Fibonacci sequence upto 15 :
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
```



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python fibonacci.py
How many terms: 1
Fibonacci sequence upto 1 :
0
```



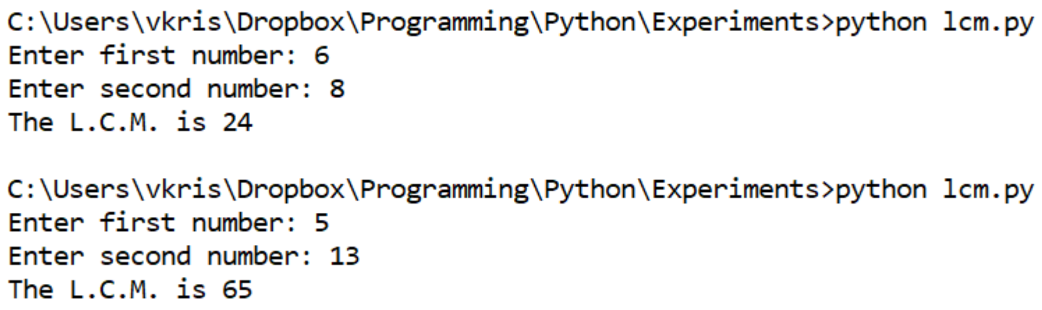
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python fibonacci.py
How many terms: -10
Please enter a positive integer
```

6. Python Program to L.C.M. of 2 input Number

Program in lcm.py:

```
x = int(input("Enter first number: "))
y = int(input("Enter second number: "))

if x > y:
    greater = x
else:
    greater = y

while(True):
    if((greater % x == 0) and (greater % y == 0)):
        lcm = greater
        break
    greater += 1

print("The L.C.M. is",lcm)
```

Output:



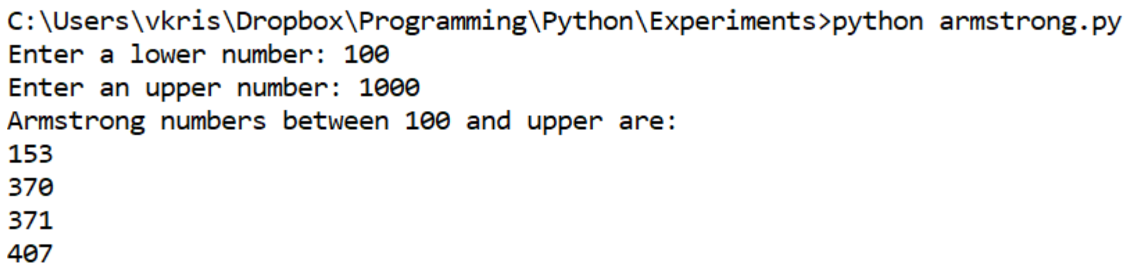
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python lcm.py
Enter first number: 6
Enter second number: 8
The L.C.M. is 24

C:\Users\vkris\Dropbox\Programming\Python\Experiments>python lcm.py
Enter first number: 5
Enter second number: 13
The L.C.M. is 65
```

7. Python Program to Find Armstrong Number in an Interval

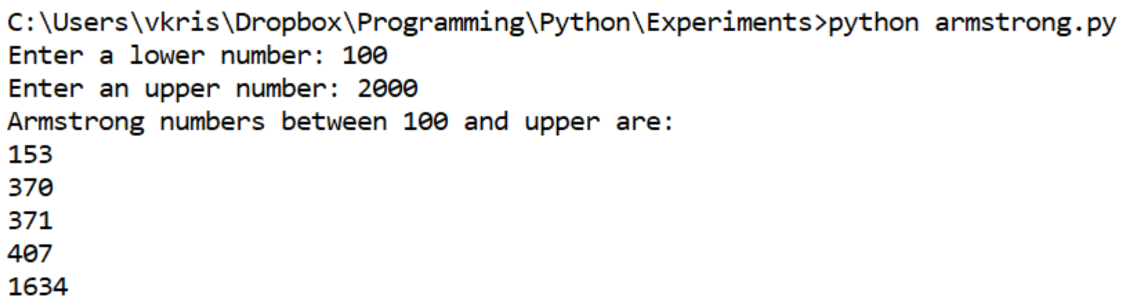
Program in armstrong.py:

```
lower = int(input("Enter a lower number: "))
upper = int(input("Enter an upper number: "))

print("Armstrong numbers between",lower,"and","upper","are:")
for num in range(lower, upper + 1):
   order = len(str(num))
   sum = 0
   temp = num
   while temp > 0:
       digit = temp % 10
       sum += digit ** order
       temp //= 10
   if num == sum:
       print(num)
```

Output:



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python armstrong.py
Enter a lower number: 100
Enter an upper number: 1000
Armstrong numbers between 100 and upper are:
153
370
371
407
```



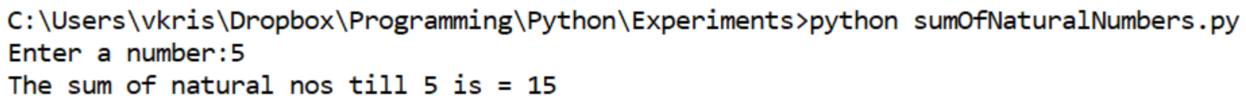
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python armstrong.py
Enter a lower number: 100
Enter an upper number: 2000
Armstrong numbers between 100 and upper are:
153
370
371
407
1634
```

8. Python Program to Find the Sum of Natural Numbers

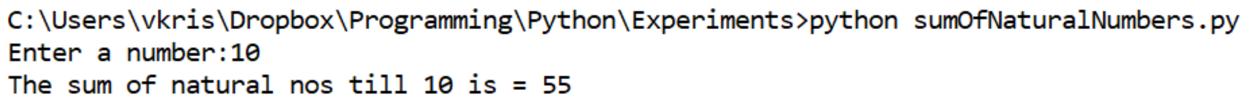
Program in sumOfNaturalNumbers.py:

```
num = int(input("Enter a number:"))
if num < 0:
   print("Enter a positive number")
else:
   temp = num
   sum = 0
   while(num > 0):
       sum += num
       num -= 1
   print("The sum of natural nos till",temp,"is =", sum)
```

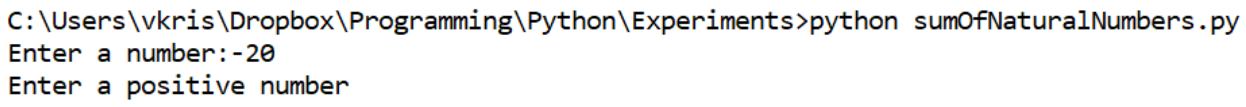
Output:



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python sumOfNaturalNumbers.py
Enter a number:5
The sum of natural nos till 5 is = 15
```



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python sumOfNaturalNumbers.py
Enter a number:10
The sum of natural nos till 10 is = 55
```



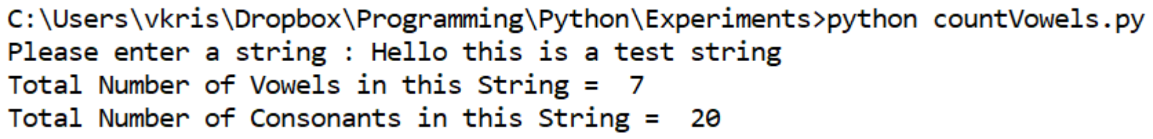
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python sumOfNaturalNumbers.py
Enter a number:-20
Enter a positive number
```

9. WAP to count number of vowels and consonants from the given strings

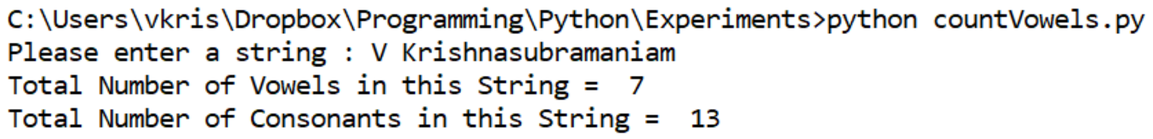
Program in countVowels.py:

```
str1 = input("Please enter a string : ")
vowels = 0
consonants = 0

for i in str1:
    if(i == 'a' or i == 'e' or i == 'i' or i == 'o' or i == 'u'
       or i == 'A' or i == 'E' or i == 'I' or i == 'O' or i == 'U'):
        vowels = vowels + 1
    else:
        consonants = consonants + 1

print("Total Number of Vowels in this String = ", vowels)
print("Total Number of Consonants in this String = ", consonants)
```

Output:



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python countVowels.py
Please enter a string : Hello this is a test string
Total Number of Vowels in this String =  7
Total Number of Consonants in this String =  20
```



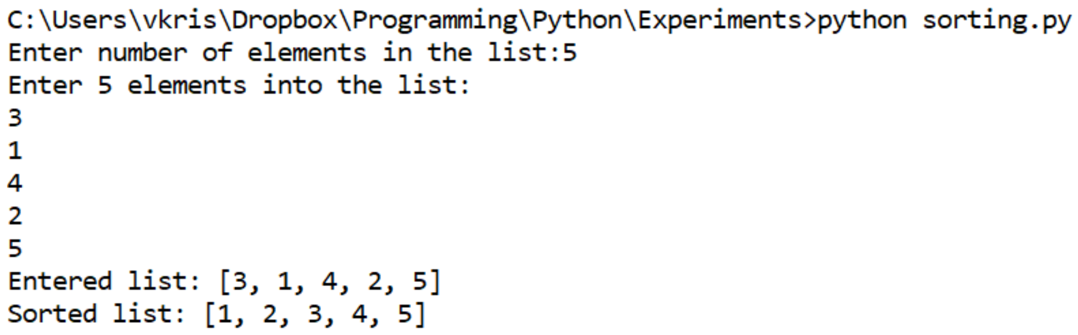
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python countVowels.py
Please enter a string : V Krishnasubramaniam
Total Number of Vowels in this String =  7
Total Number of Consonants in this String =  13
```

10. Python program to demonstrate sorting algorithms using List

Program in sorting.py:

```
list = []
n = int(input("Enter number of elements in the list:"))
print("Enter",n,"elements into the list:")
for i in range(n):
    a = int(input())
    list.append(a)
print("Entered list:",list)
for iter_num in range(len(list)-1,0,-1):
    for idx in range(iter_num):
        if list[idx]>list[idx+1]:
            temp = list[idx]
            list[idx] = list[idx+1]
            list[idx+1] = temp
print("Sorted list:",list)
```

Output:



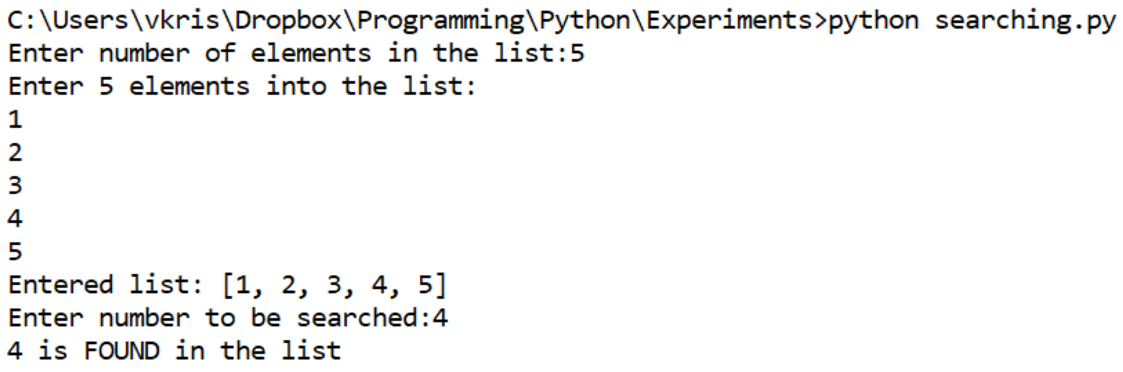
```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python sorting.py
Enter number of elements in the list:5
Enter 5 elements into the list:
3
1
4
2
5
Entered list: [3, 1, 4, 2, 5]
Sorted list: [1, 2, 3, 4, 5]
```

11. Python program to demonstrate searching algorithms using List or tuples

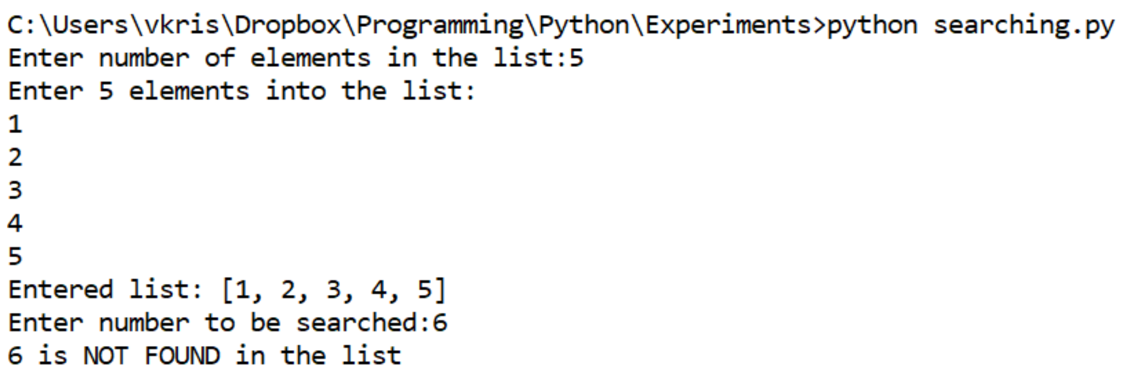
Program in searching.py:

```
list = []
n = int(input("Enter number of elements in the list:"))
print("Enter",n,"elements into the list:")
for i in range(n):
   a = int(input())
   list.append(a)
print("Entered list:",list)
search_for = int(input("Enter number to be searched:"))
search_at = 0
search_res = False
while search_at < len(list) and search_res is False:
   if list[search_at] == search_for:
      search_res = True
   else:
      search_at = search_at + 1
if search_res:
   print(search_for,"is FOUND in the list")
else:
   print(search_for,"is NOT FOUND in the list")
```

Output:



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python searching.py
Enter number of elements in the list:5
Enter 5 elements into the list:
1
2
3
4
5
Entered list: [1, 2, 3, 4, 5]
Enter number to be searched:4
4 is FOUND in the list
```



```
C:\Users\vkris\Dropbox\Programming\Python\Experiments>python searching.py
Enter number of elements in the list:5
Enter 5 elements into the list:
1
2
3
4
5
Entered list: [1, 2, 3, 4, 5]
Enter number to be searched:6
6 is NOT FOUND in the list
```

Conclusion:

Thus, we have understood basic data types, operators, expressions, input output statements with format function. We have also learnt and performed hands-on programs using controls flow statements like decision-making and looping statements in Python.