Lab 1-A

Installing Python:

www.python.org

For Windows (32 bit): https://www.python.org/ftp/python/3.4.3/python-3.4.3.msi

For Windows (64 bit): https://www.python.org/ftp/python/3.4.3/python-3.4.3.amd64.msi

Opening IDLE

Go to the start menu, find Python, and run the program labeled 'IDLE'

(Stands for Integrated DeveLopment Environment)

Code Example 1 - Hello, World!

```
>>> print ("Hello, World!" )
```

Learning python for a C++/C# programmer

Let us try to quickly compare the syntax of python with that of C++/C#:

	C++/C#	**Python**
Comment begins with	//	#
Statement ends with	;	No semi-colon needed
Blocks of code	Defined by {}	Defined by indentation (usually four spaces)
Indentation of code and use of white space	Is irrelevant	Must be same for same block of code (for example for a set of statements to be executed after a particular if statement)
Conditional statement	if-else if-else	if – elif – else:

Parentheses for loop execution condition	Required	Not required but loop condition followed by a colon :
		
		while a < n:
		
		    print(a)

Lab 1-B

Math in Python

Calculations are simple with Python, and expression syntax is straightforward: the operators +, -, * and / work as expected; parentheses () can be used for grouping.

```
# Python 3: Simple arithmetic

>>> 1 / 2

0.5

>>> 2 ** 3      #Exponent operator

8

>>> 17 / 3  # classic division returns a float

5.666666666666667

>>> 17 // 3  # floor division

5

>>> 23%3       #Modulus operator

2
```

Python Operators

Command	Name	Example	Output
+	Addition	4+5	9
-	Subtraction	8-5	3
*	Multiplication	4*5	20
/	Classic Division	19/3	6.3333
%	Modulus	19%3	5
**	Exponent	2**4	16
//	Floor Division	19/3	6

Comments in Python:

```
#I am a comment. I can say whatever I want!
```

Variables:

```
print ("This program is a demo of variables")

v = 1

print ("The value of v is now", v)

v = v + 1

print ("v now equals itself plus one, making it worth", v)

print ("To make v five times bigger, you would have to type v = v * 5")

v = v * 5

print ("There you go, now v equals", v, "and not", v / 5 )
```

Strings:

```
word1 = "Good"

word2 = "Morning"

word3 = "to you too!"

print (word1, word2)

sentence = word1 + " " + word2 + " " +word3

print (sentence)
```

Relational operators:

Expression	Function
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
!=	not equal to
==	is equal to

Boolean Logic:

Boolean logic is used to make more complicated conditions for **if** statements that rely on more than one condition. Python’s Boolean operators are **and**, **or**, and **not**. The **and** operator takes two arguments, and evaluates as **True** if, and only if, both of its arguments are True. Otherwise it evaluates to **False**.

The **or** operator also takes two arguments. It evaluates if either (or both) of its arguments are **False**.

Unlike the other operators we've seen so far, **not** only takes one argument and inverts it. The result of **not True** is **False**, and **not False** is **True**.

Operator Precedence:

Operator	Description
()	Parentheses
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus
* / % //	Multiply, divide, modulo, and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive 'OR' and regular 'OR'
<= < > >=	Comparison Operators
== !=	Equality Operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Conditional Statements:

'if' - Statement

```
y = 1

if y == 1:

    print ("y still equals 1, I was just checking")
```

'if - else' - Statement

```
a = 1

if a > 5:

    print ("This shouldn't happen.")

else:

    print ("This should happen.")
```

‘elif’ - Statement

```
z = 4

if z > 70:

    print ("Something is very wrong")

elif z < 7:

    print ("This is normal")
```

LAB TASK:

1. Open IDLE and run the following program. Try different integer values for separate runs of the program. Play around with the indentation of the program lines of code and run it again. See what happens. Make a note of what changes you made and how it made the program behave. Also note any errors, as well as the changes you need to make to remove the errors.

```
x = input("Please enter an integer: ")

if x < 0:

    x = 0

    print('Negative changed to zero')

elif x == 0:

    print('Zero')

elif x == 1:

    print('Single')

else:

    print('More')
```

Awnser

This code gives an error of comparison between a string and integer. You can correct this by taking the input as

```
x = int(input("Please enter an integer: "))
```

Now the code looks like:

```
x = int(input("Please enter an integer: "))
if x < 0:
    x = 0
    print('Negative changed to zero')
elif x == 0:
    print('Zero')
elif x == 1:
    print('Single')
else:
    print('More')
```

After correction, if you input a string value, it will raise a TypeError.

If you enter values according to their condition, they give the output as intended hence there are no Logical Errors.

Lab 1-C

Input from user:

The **input()** function prompts for input and returns a string.

```
a = input (“Enter Value for variable a:  ”)
```

```
print (a)
```

Indexes of String:

Characters in a string are numbered with *indexes* starting at 0:

Example:

name = "J. Smith“

Index	0	1	2	3	4	5	6	7
Character	J	.		S	m	i	t	h

Accessing an individual character of a string:

variableName [**index**]

Example:

```
print (name, " starts with", name[0])
```

Output:

J. Smith starts with J

input:

input: Reads a string of text from user input.

Example:

```
name = input("What's your name? ")
```

```
print (name, "... what a nice name!")
```

Output:

What's your name? Ali

Ali... what a nice name!

String Properties:

len(*string*)	- number of characters in a string (including spaces)
str.lower(*string*)	- lowercase version of a string
str.upper(*string*)	- uppercase version of a string

Example:

```
name = "Linkin Park"

length = len(name)

big_name = str.upper(name)

print (big_name, "has", length, "characters")
```

Output:

```
LINKIN PARK has 11 characters
```

Strings and numbers:

ord(*text*) - converts a string into a number.

Example: ord('a') is 97, ord("b") is 98, ...

Characters map to numbers using standardized mappings such as *ASCII* and *Unicode*.

chr (*number*) - converts a number into a string.

Example: chr(99) is "c"

Loops in Python:

The 'while' loop

```
a = 0

while a < 10:

    a = a + 1

    print (a )
```

The 'for' loop

```
for i in range(1, 5):

      print (i )
```

```
for i in range(1, 5):
      print (i)
else:
print ('The for loop is over')
```

Functions:

How to call a function?

function_name(parameters)

Code Example - Using a function

```
def multiplybytwo(x):
   return x*2
a = multiplybytwo(70)
```

The computer would actually see this:

a=140

Define a Function?

def function_name(parameter_1,parameter_2):

{this is the code in the function}

return {value (e.g. text or number) to return to the main program}

range() **Function:**

If you need to iterate over a sequence of numbers, the built-in function range() comes in handy. It generates iterator containing arithmetic progressions:

```
>>> range(10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

It is possible to let the range start at another number, or to specify a different increment (even negative; sometimes this is called the ‘step’):

```
>>> list(range(5, 10) )

[5, 6, 7, 8, 9]

>>> list(range(0, 10, 3) )

[0, 3, 6, 9]

>>> list(range(-10, -100, -30) )

[-10, -40, -70]
```

The range() function is especially useful in loops.

Lab 1-D

Classes & Inheritance:

The word 'class' can be used when describing the code where the class is defined.
A variable inside a class is known as an *Attribute*
A function inside a class is known as a *method*

- A class is like a
  - Prototype
  - Blue-print
  - An object creator
- A class defines potential objects
  - What their structure will be
  - What they will be able to do
- Objects are instances of a class
  - An object is a container of data: attributes
  - An object has associated functions: methods

Syntax:
# Defining a class
class class_name:
[statement 1]
[statement 2]
[statement 3] [etc]

Inheritance Syntax:
```
   class child_class(parent_class):
         def __init__(self,x):
               # it will modify the _init_ function from parent class
         # additional methods can be defined here
```

Example1:
```
class MyClass:
   i = 12345
   def f(self):
               return 'hello world'
x = MyClass()
print (x.i)
print (x.f() )
```

Example2:
```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart
x = Complex(3.0, -4.5)
print (x.r,"      ",x.i )
```
Example3:
```
class Shape:
    def __init__(self,x,y):      #The __init__ function always runs 
first
        self.x = x
```

```
            self.y = y
        description = "This shape has not been described yet"
        author = "Nobody has claimed to make this shape yet"

        def area(self):
            return self.x * self.y
        def perimeter(self):
            return 2 * self.x + 2 * self.y
        def describe(self,text):
            self.description = text
        def authorName(self,text):
            self.author = text
        def scaleSize(self,scale):
            self.x = self.x * scale
            self.y = self.y * scale

    a=Shape(3,4)
    print (a.area())
```

Inheritance Example:

```
class Square(Shape):
   def __init__(self,x):
         self.x = x
         self.y = x

class DoubleSquare(Square):
   def __init__(self,y):
         self.x = 2 * y
         self.y = y
   def perimeter(self):
         return 2 * self.x + 2 * self.y
```

Module:

A module is a python file that (generally) has only definitions of variables, functions, and classes.

Example: Module name mymodule.py

```
# Define some variables:
ageofqueen = 78

# define some functions
def printhello():
    print ("hello")
# define a class
class Piano:
    def __init__(self):
        self.type = input("What type of piano?: ")
        self.height = input("What height (in feet)?: ")
        self.price = input("How much did it cost?: ")
        self.age = input("How old is it (in years)?: ")

    def printdetails(self):
        print ("This piano is a/an " + self.height + " foot")
        print (self.type, "piano, " + self.age, "years old and costing " +
self.price + " dollars.")
```

Importing module in main program:

```
### mainprogam.py ##
# IMPORTS ANOTHER MODULE
```

```
import mymodule
print (mymodule.ageofqueen )
cfcpiano = mymodule.Piano()
cfcpiano.printdetails()
```

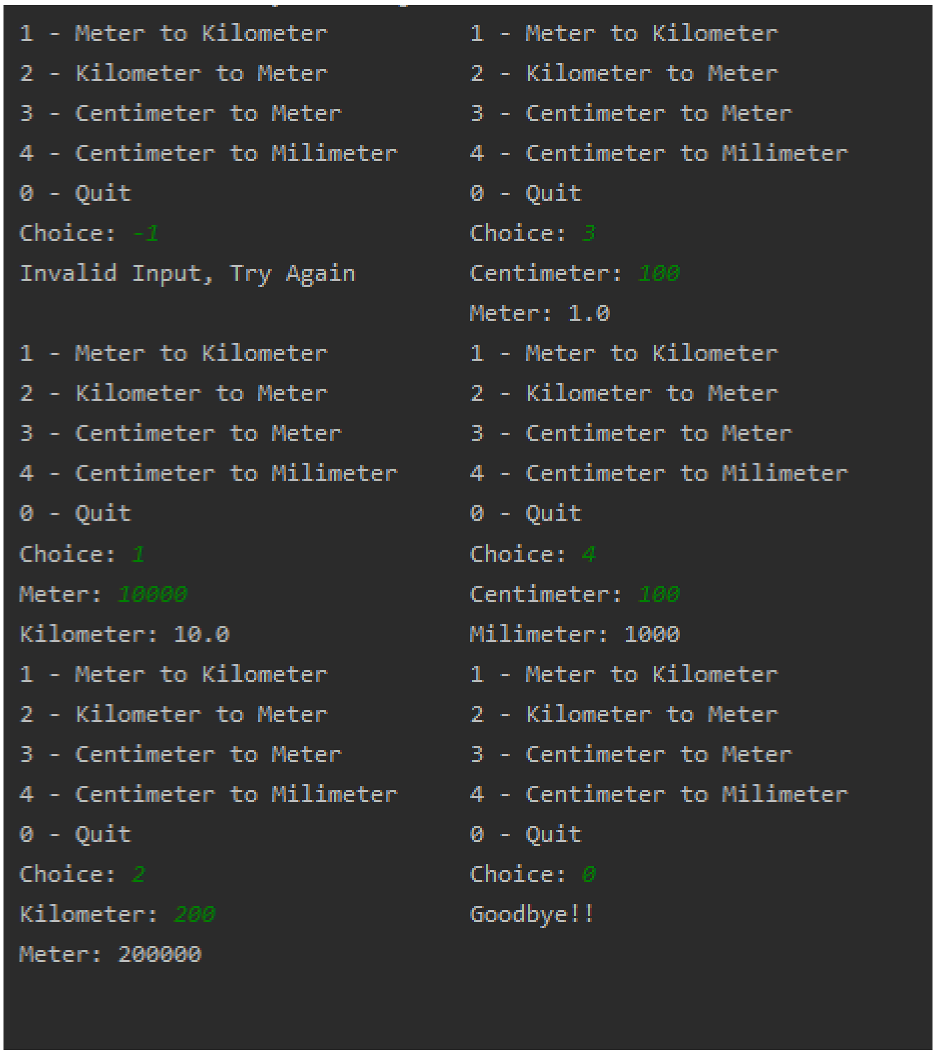
Another way of importing the module is:

```
from mymodule import Piano, ageofqueen
print (ageofqueen)
cfcpiano = Piano()
cfcpiano.printdetails()
```

LAB TASK:

2. Write a simple unit calculator program. Follow the steps below:
   a. Declare and define a function named Menu which displays a list of choices for user such as meter to kilometer, kilometer to meter, centimeter to meter, & centimeter to millimeter. It takes the choice from user as an input and return.
   b. Define and declare a separate function for each choice.
   c. In the main body of the program call respective function depending on user's choice.
   d. Program should not terminate till user chooses option to "Quit".

```python
def menu():
    print("1 - Meter to Kilometer")
    print("2 - Kilometer to Meter")
    print("3 - Centimeter to Meter")
    print("4 - Centimeter to Milimeter")
    print("0 - Quit")
    choice = int(input("Choice: "))
    return choice

def meter_to_kilometer(meter):
    return meter/1000

def kilometer_to_meter(kilometer):
    return kilometer*1000

def centimeter_to_meter(centimeter):
    return centimeter/100

def centimeter_to_milimeter(centimeter):
    return centimeter*10


choice = 1
while(choice!=0):
    choice = menu()
    if(choice == 1):
        meter = int(input("Meter: "))
        print(f"Kilometer: {meter_to_kilometer(meter)}")
    elif(choice == 2):
        kilometer = int(input("Kilometer: "))
        print(f"Meter: {kilometer_to_meter(kilometer)}")
    elif(choice == 3):
        centimeter = int(input("Centimeter: "))
        print(f"Meter: {centimeter_to_meter(centimeter)}")
    elif(choice == 4):
        centimeter = int(input("Centimeter: "))
        print(f"Milimeter: {centimeter_to_milimeter(centimeter)}")
    elif(choice == 0):
        print("Goodbye!!")
    else:
        print("Invalid Input, Try Again\n")
```

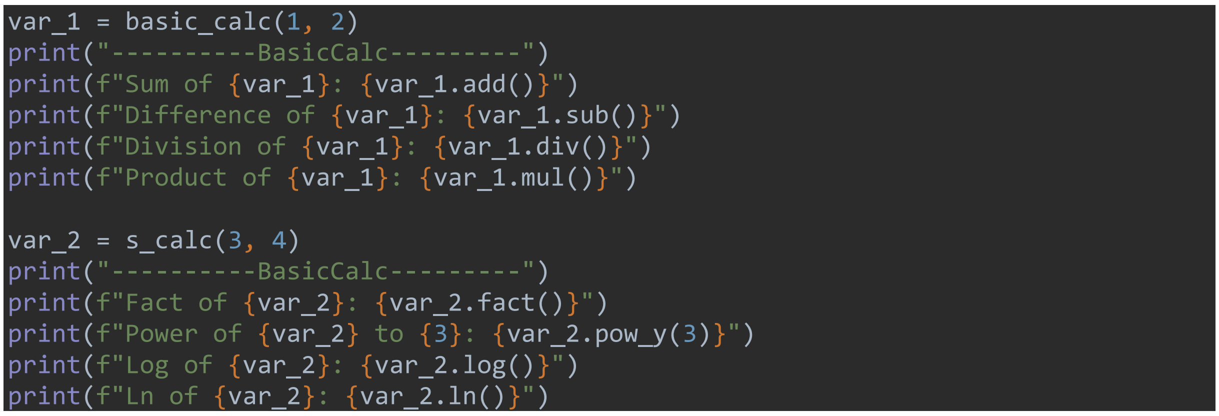


```
1 - Meter to Kilometer
2 - Kilometer to Meter
3 - Centimeter to Meter
4 - Centimeter to Milimeter
0 - Quit
Choice: -1
Invalid Input, Try Again

1 - Meter to Kilometer
2 - Kilometer to Meter
3 - Centimeter to Meter
4 - Centimeter to Milimeter
0 - Quit
Choice: 1
Meter: 10000
Kilometer: 10.0
1 - Meter to Kilometer
2 - Kilometer to Meter
3 - Centimeter to Meter
4 - Centimeter to Milimeter
0 - Quit
Choice: 2
Kilometer: 200
Meter: 200000
```

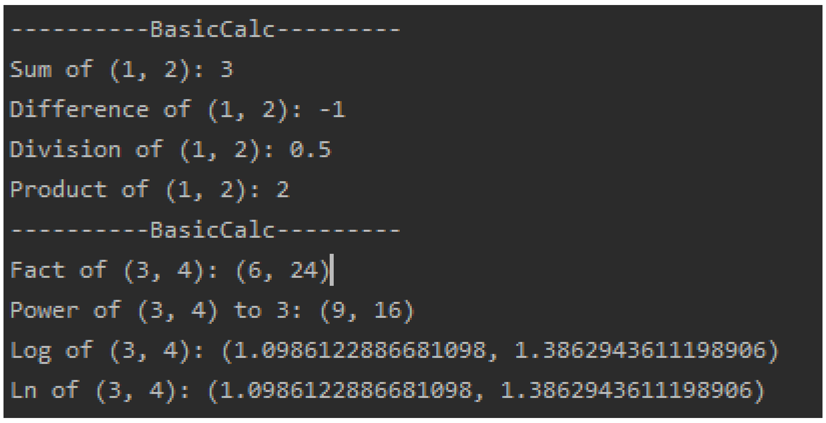
```
1 - Meter to Kilometer
2 - Kilometer to Meter
3 - Centimeter to Meter
4 - Centimeter to Milimeter
0 - Quit
Choice: 3
Centimeter: 100
Meter: 1.0
1 - Meter to Kilometer
2 - Kilometer to Meter
3 - Centimeter to Meter
4 - Centimeter to Milimeter
0 - Quit
Choice: 4
Centimeter: 100
Milimeter: 1000
1 - Meter to Kilometer
2 - Kilometer to Meter
3 - Centimeter to Meter
4 - Centimeter to Milimeter
0 - Quit
Choice: 0
Goodbye!!
```

3. Create a class name basic_calc with following attributes and methods;
   Two integers (values are passed with instance creation)
   Different methods such as addition, subtraction, division, multiplication
   Create another class inherited from basic_calc named s_calc which should have the following additional methods;
   Factorial, x_power_y,log, ln etc

```python
import math

class basic_calc:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return (f"({self.a}, {self.b})")

    def add(self):
        return self.a + self.b

    def div(self):
        return self.a / self.b

    def mul(self):
        return self.a * self.b

    def sub(self):
        return self.a - self.b


class s_calc(basic_calc):
    def fact(self):
        a = self.a
        b = self.b
        fact_a = 1
        while a:
            fact_a *= a
            a -= 1
        fact_b = 1
        while b:
            fact_b *= b
            b -= 1
        return s_calc(fact_a, fact_b)

    def pow_y(self, y):
        power_a=1
        power_b=1
        for i in range(y):
            power_a = self.a*self.a
            power_b = self.b*self.b
        return power_a, power_b

    def log(self):
        return math.log(self.a), math.log(self.b)

    def ln(self):
        return self.log()
```



```python
var_1 = basic_calc(1, 2)
print("----------BasicCalc---------")
print(f"Sum of {var_1}: {var_1.add()}")
print(f"Difference of {var_1}: {var_1.sub()}")
print(f"Division of {var_1}: {var_1.div()}")
print(f"Product of {var_1}: {var_1.mul()}")

var_2 = s_calc(3, 4)
print("----------BasicCalc---------")
print(f"Fact of {var_2}: {var_2.fact()}")
print(f"Power of {var_2} to {3}: {var_2.pow_y(3)}")
print(f"Log of {var_2}: {var_2.log()}")
print(f"Ln of {var_2}: {var_2.ln()}")
```



```
----------BasicCalc---------
Sum of (1, 2): 3
Difference of (1, 2): -1
Division of (1, 2): 0.5
Product of (1, 2): 2
----------BasicCalc---------
Fact of (3, 4): (6, 24)
Power of (3, 4) to 3: (9, 16)
Log of (3, 4): (1.0986122886681098, 1.3862943611198906)
Ln of (3, 4): (1.0986122886681098, 1.3862943611198906)
```

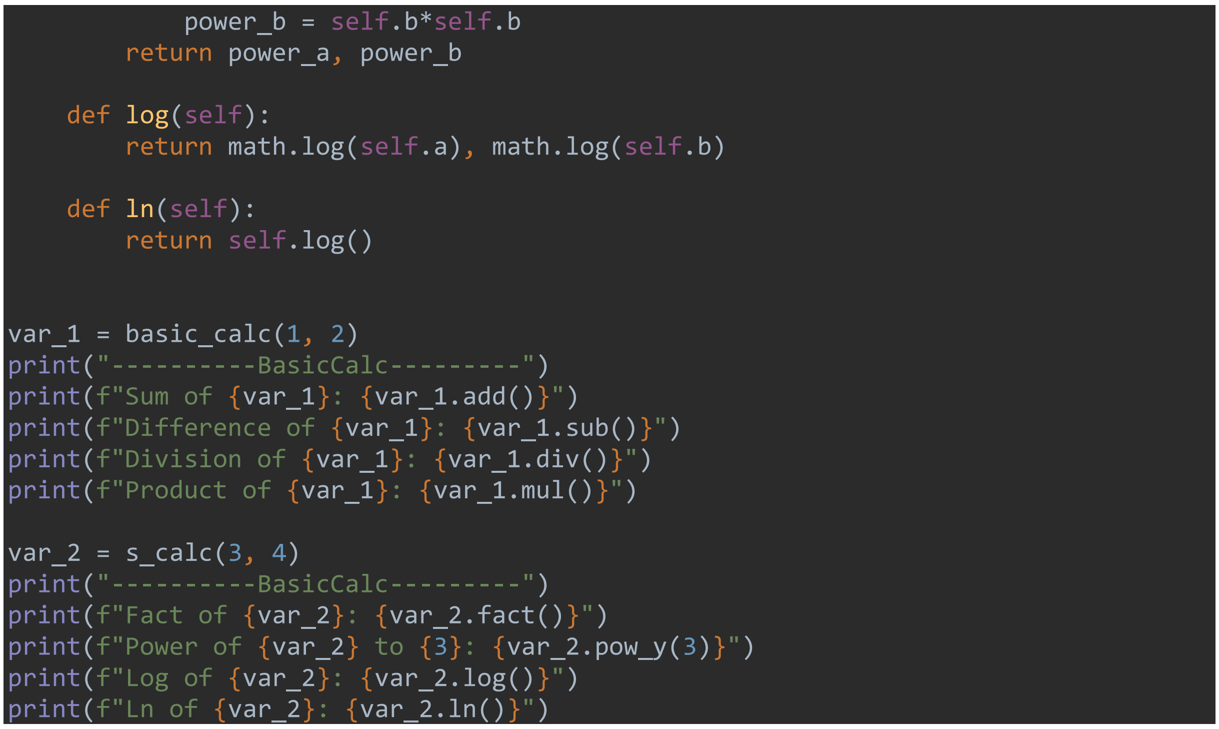
4. Modify the classes created in the above task under as follows:
   Create a module name basic.py having the class name basic_calc with all the attributes and methods defined before.
   Now import the basic.py module in your program and do the inheritance step defined before i.e.
   Create another class inherited from basic_calc named s_calc which should have the following additional methods;
   Factorial, x_power_y, log, ln etc

Basic.py

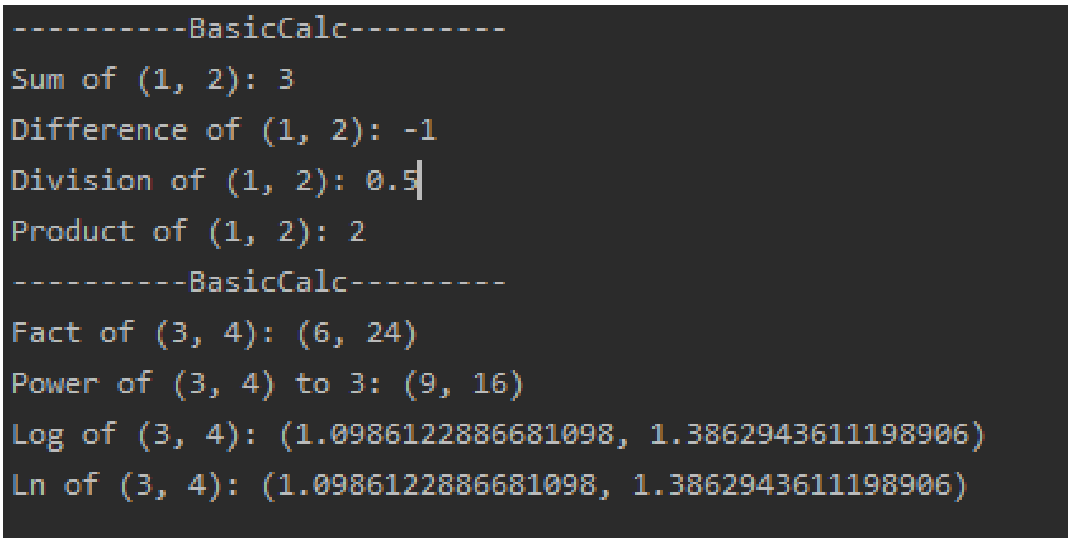
```
class basic_calc:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return (f"({self.a}, {self.b})")

    def add(self):
        return self.a + self.b

    def div(self):
        return self.a / self.b

    def mul(self):
        return self.a * self.b

    def sub(self):
        return self.a - self.b
```

app.py

```
import math
from basic import *


class s_calc(basic_calc):
    def fact(self):
        a = self.a
        b = self.b
        fact_a = 1
        while a:
            fact_a *= a
            a -= 1
        fact_b = 1
        while b:
            fact_b *= b
            b -= 1
        return s_calc(fact_a, fact_b)

    def pow_y(self, y):
        power_a=1
        power_b=1
        for i in range(y):
            power_a = self.a*self.a
```



```python
            power_b = self.b*self.b
        return power_a, power_b

    def log(self):
        return math.log(self.a), math.log(self.b)

    def ln(self):
        return self.log()


var_1 = basic_calc(1, 2)
print("----------BasicCalc---------")
print(f"Sum of {var_1}: {var_1.add()}")
print(f"Difference of {var_1}: {var_1.sub()}")
print(f"Division of {var_1}: {var_1.div()}")
print(f"Product of {var_1}: {var_1.mul()}")

var_2 = s_calc(3, 4)
print("----------BasicCalc---------")
print(f"Fact of {var_2}: {var_2.fact()}")
print(f"Power of {var_2} to {3}: {var_2.pow_y(3)}")
print(f"Log of {var_2}: {var_2.log()}")
print(f"Ln of {var_2}: {var_2.ln()}")
```



```
----------BasicCalc---------
Sum of (1, 2): 3
Difference of (1, 2): -1
Division of (1, 2): 0.5
Product of (1, 2): 2
----------BasicCalc---------
Fact of (3, 4): (6, 24)
Power of (3, 4) to 3: (9, 16)
Log of (3, 4): (1.0986122886681098, 1.3862943611198906)
Ln of (3, 4): (1.0986122886681098, 1.3862943611198906)
```