



डॉ. ए.पी.जे. अब्दुल कलाम प्रौद्योगिकी संस्थान, टनकपुर, उत्तराखंड
(संघटक संस्थान, उत्तराखंड तकनीकी विश्वविद्यालय, उत्तराखंड सरकार)
Dr. A.P.J. Abdul Kalam Institute of Technology, Tanakpur, Uttarakhand
(A Constituent Institute of Uttarakhand Technical University, Govt. of Uttarakhand)
Approved by AICTE, MHRD New Delhi



LAB FILE

Academic Session

(2022-23)

Program : B. Tech in Computer Science and Engineering

Course : Computer Workshop (Python Programming)

No. of credits : 1

Semester : III

Session : July - Dec 2022

Batch : 2021-2025

Faculty : Dr. Hitesh Kumar Sharma

Submitted By :

Name : Rohit Kumar

Roll No. : 211620101045

Submitted to :

Dr. Hitesh Kumar Sharma

Dept. of Computer Science

LIST OF EXPERIMENT

Exp.no	Experiment Name	Date of Experiment
1	How to declare and use variables and operators	
2	Programming using Basic Library (Numpy ,Pandas, SK Learn etc.).	
3	.To write a Python program to print HELLO INDIA.	
4	To write a Python program that takes in command line argument as input and print the number of arguments	
5	To write a Python program find the division of students.	
6	To write a Python program implement Fibonacci Series.	
7	To write a Python program for factorial.	
8	To write a Python program to use of function.	
9	To write a Python program to implement list.	
10	To write a Python program to implement tuples.	
11	To write a Python program Insertion sort.	
12	To write a Python program Merge sort.	
13	To write a Python program first n prime numbers.	
14	Implementation of Data Science concept using Python.	

LAB EXERCISE 1ST

Aim: How to declare and use variables and operators.

Description:

In Python, we can declare a variable by simply assigning a value to it.
For example:

```
x = 10
```

```
y = "Hello, World!"
```

Here, x is an integer variable and y is a string variable.

We can also use various operators in Python to perform operations on variables. Some common operators include:

+: Addition

-: Subtraction

*: Multiplication

/: Division

?: Modulus (remainder after division)

For example:

```
a = 10
```

```
b = 20
```

```
c = a + b # c is now 30
```

```
d = a - b # d is now -10
```

```
e = a * b # e is now 200
```

```
f = b / a # f is now 2.0
```

```
g = b % a # g is now 0
```

We can also use comparison operators to compare the values of two variables. Some common comparison operators include :

== : Equal to

!= : Not equal to

< : Less than

> : Greater than

<= : Less than or equal to

>= : Greater than or equal to

For example:

```
h = 10
```

```
i = 20
```

```
if h < i:
```

```
    print("h is less than i")
```

```
else:
```

```
    print("h is greater than or equal to i")
```

This would print "h is less than i" because the value of h is indeed less than the value of i.

LAB EXERCISE 2ND

Aim: Programming using Basic Libraries (Numpy, Pandas, SK Learn etc)

Description:

Pandas

Pandas is a very popular library for working with data (its goal is to be the most powerful and flexible open-source tool, and in our opinion, it has reached that goal). Data Frames are at the centre of pandas. A Data Frame is structured like a table or spreadsheet. The rows and the columns both have indexes, and you can perform operations on rows or columns separately.

A pandas Data Frame can be easily changed and manipulated. Pandas has helpful functions for handling missing data, performing operations on columns and rows, and transforming data. If that wasn't enough, a lot of SQL functions have counterparts in pandas, such as join, merge, filter by, and group by. With all of these powerful tools, it should come as no surprise that pandas is very popular among data scientists.

Command to install : `pip install pandas`

NumPy

NumPy is an open-source Python library that facilitates efficient numerical operations on large quantities of data. There are a few functions that exist in NumPy that we use on pandas Data Frames. For us, the most important part about NumPy is that pandas is built on top of it. So, NumPy is a dependence.

Command to install : `pip install numpy`

SK Learn

Scikit-learn is probably the most useful library for machine learning in Python. The SK-learn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

Command to install : `pip install sk learn`.

CODE :

```
labexercise2aa.py > ...  
1  import numpy  
2  
3  arr = numpy.array([1, 2, 3, 4, 5])  
4  
5  print(arr)
```

OUTPUT :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  COMMENTS  
  
PS R:\hiteshsir_practical> python -u "r:\hiteshsir_practical\labexercise2aa.py"  
[1 2 3 4 5]  
PS R:\hiteshsir_practical>  
  
⊗ 0 △ 0  </> Cloud Code  ☁ Connect to Google Cloud  🏠 tabnine starter  UTF-8  CRLF  🐍 Python
```

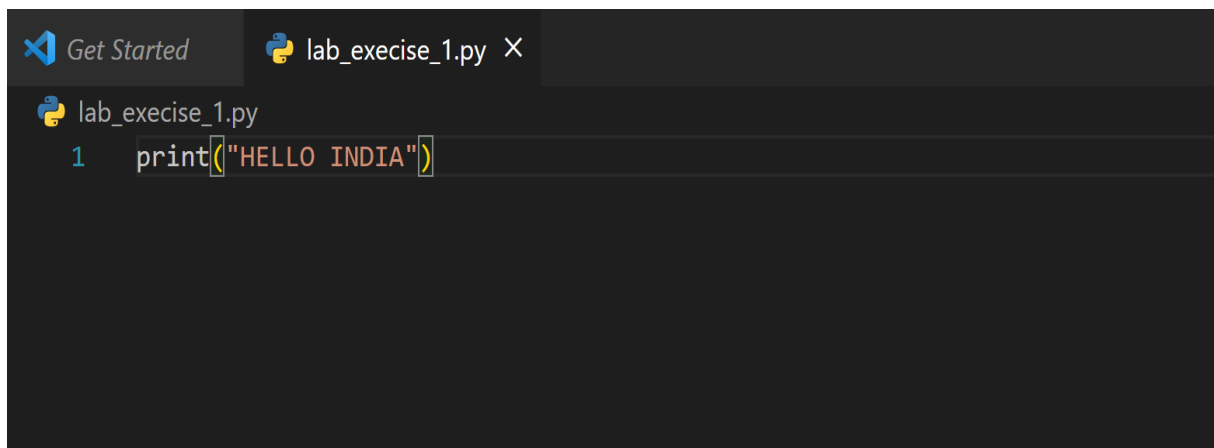
LAB EXERCISE 3rd

Aim :- TO write a python program to print HELLO INDA.

DESCRIPTION :-

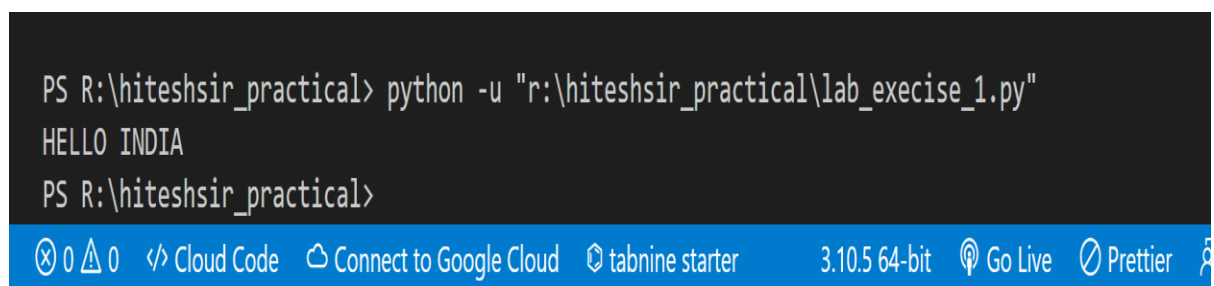
Print function is used in this program.

CODE :-

A screenshot of a code editor with a dark theme. The top bar shows a 'Get Started' button and a tab for 'lab_excise_1.py'. The editor area shows a single line of Python code: `1 print("HELLO INDIA")`.

```
lab_excise_1.py
1 print("HELLO INDIA")
```

OUTPUT :-

A screenshot of a terminal window with a dark background. It shows the command `python -u "r:\hiteshsir_practical\lab_excise_1.py"` being executed, followed by the output `HELLO INDIA`. The terminal prompt is `PS R:\hiteshsir_practical>`. The bottom status bar is blue and contains icons for Cloud Code, Google Cloud, tabnine starter, 3.10.5 64-bit, Go Live, and Prettier.

```
PS R:\hiteshsir_practical> python -u "r:\hiteshsir_practical\lab_excise_1.py"
HELLO INDIA
PS R:\hiteshsir_practical>
```

LAB EXERCISE 4th

Aim :- To write a python program that make in command line arguments as input and print the number of arguments.

DESCRIPTION:-

- The argument that are given after the name of the program in the command line shell of the operating system are known as Command Line Arguments.
- Python provides various ways of dealing with these types of arguments. The three most common are :-
 - a. using sys.argv
 - b. using getopt module
 - c. using argparse module

1. USING SYS.ARGV

CODE :-


```

lab_exercise2a.py > ...
1  import sys
2
3  n = len(sys.argv)
4  print("Total arguments passed:", n)
5
6  print("\nName of Python script:", sys.argv[0])
7
8  print("\nArguments passed:", end = " ")
9  for i in range(1, n):
10     print(sys.argv[i], end = " ")
11
12  Sum = 0
13  for i in range(1, n):
14     Sum += int(sys.argv[i])
15
16  print("\n\nResult:", Sum)
17

```

Output :-

```

PS R:\hiteshsir_practical> python -u "r:\hiteshsir_practical\lab_exercise2a.py"
Total arguments passed: 1

Name of Python script: r:\hiteshsir_practical\lab_exercise2a.py

Arguments passed:

Result: 0
PS R:\hiteshsir_practical> 

```

0 0 0

Cloud Code

Connect to Google Cloud

tabnine starter

Screen Reader Optimized

Ln 17, Col 1

Spaces: 4

2. USING GETOPT MODULE

CODE :-

lab_exercise_2b.py > ...

```
1  import getopt, sys
2
3  argumentList = sys.argv[1:]
4
5
6  options = "hmo:"
7
8  long_options = ["Help", "My_file", "Output="]
9
10 try:
11     arguments, values = getopt.getopt(argumentList, options, long_options)
12     for currentArgument, currentValue in arguments:
13
14         if currentArgument in ("-h", "--Help"):
15             print ("Displaying Help")
16
17         elif currentArgument in ("-m", "--My_file"):
18             print ("Displaying file_name:", sys.argv[0])
19
20         elif currentArgument in ("-o", "--Output"):
21             print (("Enabling special output mode (% s)" % (currentValue))
22
23 except getopt.error as err:
24     print (str(err))
25
```

3. USING ARGPARSE MODULE

CODE :-

```
lab_exercise_2c.py > ...  
1 import argparse  
2  
3 # Initialize parser  
4 parser = argparse.ArgumentParser()  
5 parser.parse_args()  
6
```

OUTPUT :- Command Line Argument verified.

LAB EXERCISE 5th

Aim : To write a Python program find the division of student.

DESCREPTION :-

The division of student are determine by the obtaining the total percentage the then set a criteria for first , second ,third and fail division .

Suppose we have five subject which are as follows :-

1. English (eng)
2. Maths (maths)
3. Science (sci)
4. Hindi (hindi)
5. Sanskrit (Sanskrit)

Code :-

```
labexercise3.py > ...
1 eng = float(input("Enter the eng marks : "))
2 maths = float(input("Enter the maths marks : "))
3 sci = float(input("Enter the sci marks : "))
4 hindi = float(input("Enter the hindi marks : "))
5 sanskrit = float(input("Enter the sanskrit marks : "))
6 total = eng + maths + sci + hindi + sanskrit
7 per = (total/500)*100
8
9 if per>=60 :
10     print("1st division ")
11 elif per >45 and per<=59:
12     print("2nd division")
13 elif per>=33 and per<=45:
14     print("2nd division")
15 elif per<33:
16     print("fail")
```

OUTPUT :-

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  COMMENTS

PS R:\hiteshsir_practical> python -u "r:\hiteshsir_practical\labexercise3.py"
Enter the eng marks : 88
Enter the maths marks : 99
Enter the sci marks : 77
Enter the hindi marks : 99
Enter the sanskrit marks : 99
1st division
PS R:\hiteshsir_practical> 
```

⊗ 0 △ 0 ↗ Cloud Code Blackbox ☁ Connect to Google Cloud 🚪 tabnine starter Screen Reader Optimized Ln 16, Col 23 Spaces: 4 UTF-8 CRLF 🐍 Python 3.10.5 64-bit 🔍

LAB EXERCISE 6th

Aim :- To write a program implement Fibonacci series.

DESCRIPTION :-

The Fibonacci numbers are the numbers in the following integer sequence.
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,

In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

with seed values

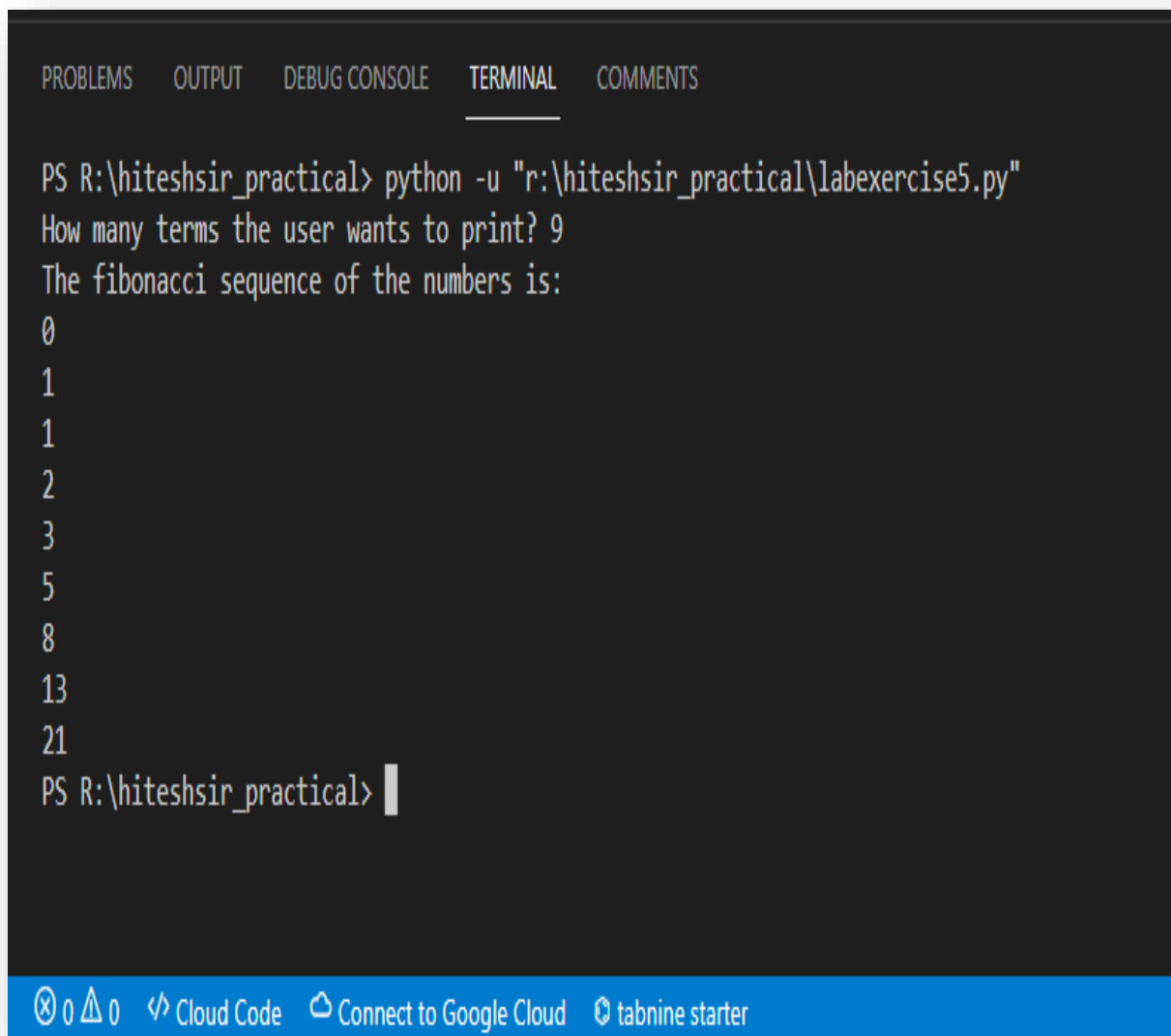
$$F_0 = 0 \text{ and } F_1 = 1.$$

SOLUTION :-

CODE :-

```
labexercise5.py > ...
1  n_terms = int(input ("How many terms the user wants to print? "))
2
3  n_1 = 0
4  n_2 = 1
5  count = 0
6
7  if n_terms <= 0:
8      print ("Please enter a positive integer, the given number is not valid")
9
10 elif n_terms == 1:
11     print ("The Fibonacci sequence of the numbers up to", n_terms, ": ")
12     print(n_1)
13 else:
14     print ("The fibonacci sequence of the numbers is:")
15     while count < n_terms:
16         print(n_1)
17         nth = n_1 + n_2
18         n_1 = n_2
19         n_2 = nth
20         count += 1
```

Output :-



The screenshot shows a terminal window with a dark background. At the top, there are five tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is active and underlined), and COMMENTS. The terminal content shows a command prompt where a Python script is run. The script asks for the number of terms to print, and the user enters 9. The script then prints the first 9 terms of the Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, and 21. The prompt returns to the command line.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL COMMENTS

PS R:\hiteshsir_practical> python -u "r:\hiteshsir_practical\labexercise5.py"
How many terms the user wants to print? 9
The fibonacci sequence of the numbers is:
0
1
1
2
3
5
8
13
21
PS R:\hiteshsir_practical> |
```

At the bottom of the terminal window, there is a blue status bar with icons for a close button, a warning icon, a debug icon, and links to 'Cloud Code', 'Connect to Google Cloud', and 'tabnine starter'.

LAB EXERCISE 7th

Aim :- To write a python program for factorial.

DESCRIPTION :-

What is factorial ?

Factorial is a non negative integer. It is the product of all positive integer less than or equal to that number you ask for factorial. It is denoted by an exclamation sign (!).

EXAMPLE :-

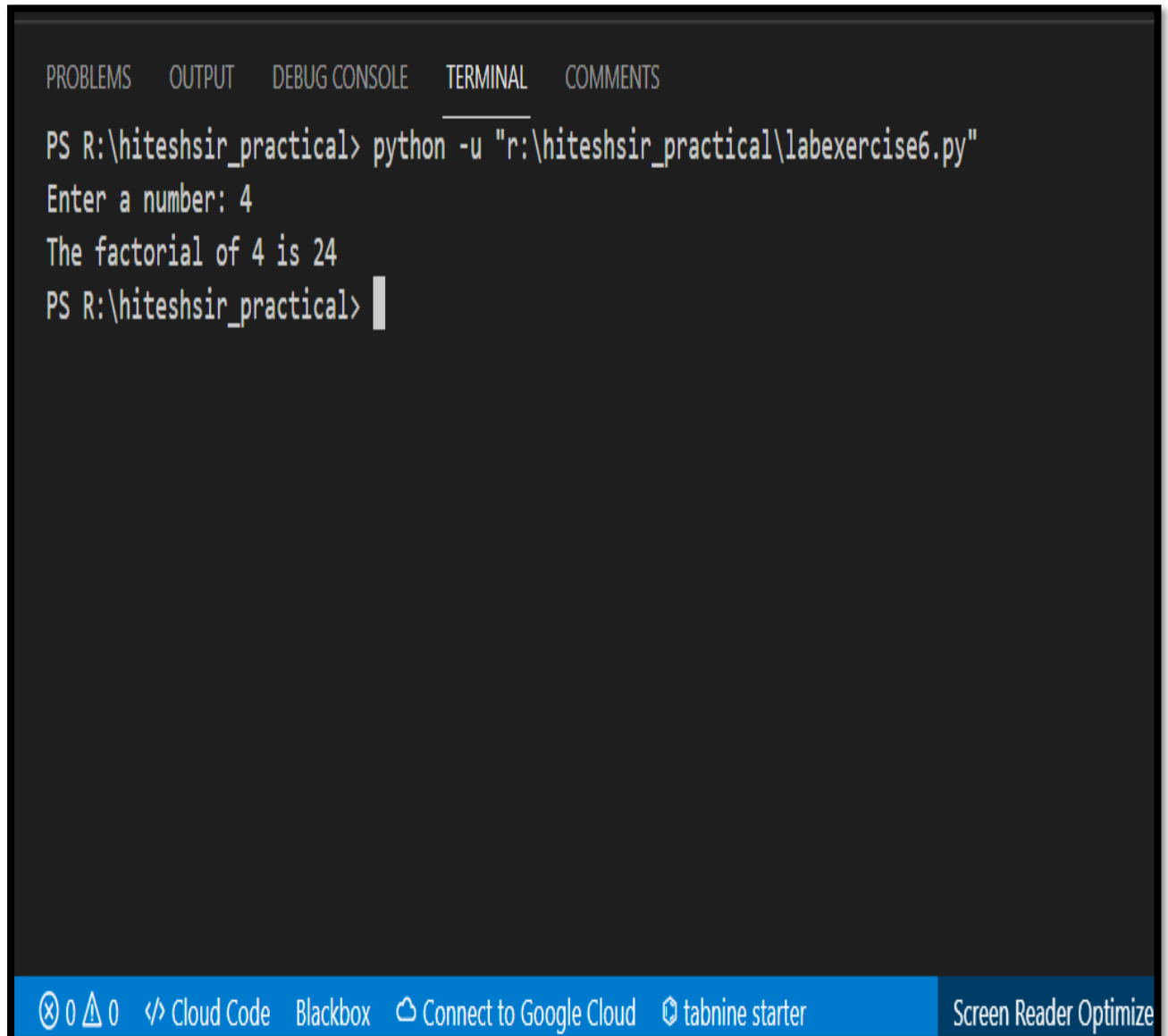
$$n! = n*(n-1)*(n-2)*.....1$$

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

CODE :-

```
labexercise6.py > ...
1  num = int(input("Enter a number: "))
2  factorial = 1
3  if num < 0:
4      print(" Factorial does not exist for negative numbers")
5  elif num == 0:
6      print("The factorial of 0 is 1")
7  else:
8      for i in range(1,num + 1):
9          factorial = factorial*i
10     print("The factorial of",num,"is",factorial)
11
```


OUTPUT :-



The screenshot shows a terminal window with a dark background. At the top, there are five tabs: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'COMMENTS'. The terminal content shows a PowerShell prompt 'PS R:\hiteshsir_practical>' followed by the command 'python -u "r:\hiteshsir_practical\labexercise6.py"'. The program then prompts 'Enter a number: 4', and the user has entered '4'. The program outputs 'The factorial of 4 is 24'. The terminal ends with the PowerShell prompt 'PS R:\hiteshsir_practical>' and a cursor. At the bottom of the terminal window is a blue status bar containing icons for error and warning counts (0 each), 'Cloud Code', 'Blackbox', 'Connect to Google Cloud', 'tabnine starter', and a 'Screen Reader Optimize' button.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  COMMENTS

PS R:\hiteshsir_practical> python -u "r:\hiteshsir_practical\labexercise6.py"
Enter a number: 4
The factorial of 4 is 24
PS R:\hiteshsir_practical> |
```

⊗ 0 △ 0 </> Cloud Code Blackbox ☁ Connect to Google Cloud 🔍 tabnine starter Screen Reader Optimize

LAB EXERCISE 8th

Aim :- To write a python program to use of functions.

DESCRIPTION :

A function is a block of code which only runs when it is called.
You can pass data, known as parameter, into a function.
A function can return data as a result.

EXMPLES :

1.Creating a Function :

In python a function is define using the **def** keyword.

Code :

```
def my_function():  
    print("Hello from a function")
```

2.Calling a function :

To call a function, use the function name followed by pranthesis:

Code :

```
def my_funtion() :  
    print("Hello from a function")  
  
my_function()
```

SOLUTION :-

CODE :-

```
labexercise7a.py > ...  
1  def my_function(country = "Norway"):  
2      print("I am from " + country)  
3  
4  my_function("Sweden")  
5  my_function("India")  
6  my_function()  
7  my_function("Brazil")
```

OUTPUT:-

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL COMMENTS

```
PS R:\hiteshsir_practical> python -u "r:\hiteshsir_practical\labexercise7a.py"  
I am from Sweden  
I am from India  
I am from Norway  
I am from Brazil  
PS R:\hiteshsir_practical> █
```

⊗ 0 △ 0 </> Cloud Code Blackbox ☁ Connect to Google Cloud 🐞 tabnine starter Ln 7, Col 22 Spaces: 4 UTF

LAB EXERCISE 9th

Aim :- To write a python to implement list

DESCRIPTION: -

A list in python is used to store the sequence of various of data. Python lists are mutable type its mean we can modify its element after it created. However, Python consist of data types that are capable to store the sequence but the most common and reliable type is the list.

A list can be defined as a collection of values or items of different types. The item in the list are separated with the comma(,) and enclosed with the square brackets[].

Examples :

```
L1 = ["Rohit",102," India"]
```

```
L2 = [2,1,6,0,4,5]
```

```
print (type(L1))
```

```
print (type(L2))
```

OUTPUT :-

```
<class 'list'>
```

```
<class 'list'>
```

SOLUTION :-

CODE :-

```
labexercise8.py > ...
1  emp = ["Rohit", 102, "INDIA"]
2  Dep1 = ["CS",10]
3  Dep2 = ["IT",11]
4  HOD_CS = [10,"Mr.Alap Mahar"]
5  HOD_IT = [11, "Mr. Hitesh Kumar Sharma"]
6  print("printing employee data...")
7  print("Name : %s, ID: %d, Country: %s"%(emp[0],emp[1],emp[2]))
8  print("printing departments...")
9  print("Department 1:\nName: %s, ID: %d\nDepartment 2:\nName: %s, ID: %s"%(Dep1[0],Dep1[1],Dep2[0],Dep2[1]))
10 print("HOD Details ....")
11 print("CS HOD Name: %s, Id: %d"%(HOD_CS[1],HOD_CS[0]))
12 print("IT HOD Name: %s, Id: %d"%(HOD_IT[1],HOD_IT[0]))
13 print(type(emp),type(Dep1),type(Dep2),type(HOD_CS),type(HOD_IT))
```

OUTPUT :-

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  COMMENTS
PS R:\hiteshsir_practical> python -u "r:\hiteshsir_practical\labexercise8.py"
printing employee data...
Name : Rohit, ID: 102, Country: INDIA
printing departments...
Department 1:
Name: CS, ID: 11
Department 2:
Name: IT, ID: 11
HOD Details ....
CS HOD Name: Mr.Alap Mahar, Id: 10
IT HOD Name: Mr. Hitesh Kumar Sharma, Id: 11
<class 'list'> <class 'list'> <class 'list'> <class 'list'> <class 'list'>
PS R:\hiteshsir_practical> █
```

⊗ 0 △ 0

↔ Cloud Code

Blackbox

☁ Connect to Google Cloud

🔌 tabnine starter

Screen Reader Op

LAB EXERCISE 10th

Aim :- To write a python program to implement tuples

DESCRIPTION :-

A collection of ordered and immutable objects is known as a tuples. Tuples and lists are similar as they both are sequences. Through tuples and lists are different because we cannot modify tuples although we can modify lists after creating them and also because we use parentheses to create tuples whiles we use square brackets to create lists.

SOLUTION :-

CODE :-

```
Labexercise9.py > ...
1  empty_tuple = ()
2  print("Empty tuple: ", empty_tuple)
3  int_tuple = (4, 6, 8, 10, 12, 14)
4  print("Tuple with integers: ", int_tuple)
5  nested_tuple = ("Python", {4: 5, 6: 2, 8:2}, (5, 3, 5, 6))
6  print("A nested tuple: ", nested_tuple)
```

OUTPUT :-

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  COMMENTS

PS R:\hiteshsir_practical> python -u "r:\hiteshsir_practical\Labexercise9.py"
Empty tuple: ()
Tuple with integers: (4, 6, 8, 10, 12, 14)
A nested tuple: ('Python', {4: 5, 6: 2, 8: 2}, (5, 3, 5, 6))
PS R:\hiteshsir_practical>
```

⊗ 0 ▲ 0 </> Cloud Code ☁ Connect to Google Cloud 🏠 tabnine starter Ln 2, Col 38 Space

LAB EXERCISE 11th

Aim : To write a python program Insertion sort.

DESCRIPTION :-

The Insertion sort algorithm concept is based on the deck of card where er sort the playing card according to particular card.

The insertion sort implementation is easy and simple because it's generally taught in the beginning programming lesson. Its in **in-place** and **stable-algorithm** that is more beneficial for nearly -sorted or fewer element.

1. In place : It is the algorithm require additional space without caring for the input size of the collection. After performing the sorting it rewrite the original memory location of the element in the collection.

2. stable : The stable is a term that manages the relative order of equal object from the initial array.

SOLUTION :-

CODE :-


```
labexercise10.py > ...
1  def insertion_sort(list1):
2
3      for i in range(1, len(list1)):
4
5          value = list1[i]
6
7          j = i - 1
8          while j >= 0 and value < list1[j]:
9              list1[j + 1] = list1[j]
10             j -= 1
11             list1[j + 1] = value
12     return list1
13
14
15 list1 = [10, 5, 13, 8, 2]
16 print("The unsorted list is:", list1)
17
18 print("The sorted list1 is:", insertion_sort(list1))
19
```

OUTPUT :-

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL COMMENTS

```
PS R:\hiteshsir_practical> python -u "r:\hiteshsir_practical\labexercise10.py"
The unsorted list is: [10, 5, 13, 8, 2]
The sorted list1 is: [2, 5, 8, 10, 13]
PS R:\hiteshsir_practical>
```

LAB EXERCISE 12th

Aim :- To write a Python program merge sort.

DESCRIPTION :-

Merge sort is similar to the quick sort algorithms as works on the concept of divide and conquer.

It divides the given list in the two halves calls itself for the halves and then merge the two sorted halves. We defines the **merge()** function used to merging two halves.

The sub list are divided again and again into halves until we get the only one element each. Then we combine the pair of one element list into two element lists sorting them in the process. The sorted two element pairs is merged into the four element list and so on until we get sorted list.

SOLUTION :-

CODE :-

```
labexercise11.py > merge
1 def merge_sort(list1, left_index, right_index):
2     if left_index >= right_index:
3         return
4
5     middle = (left_index + right_index)//2
6     merge_sort(list1, left_index, middle)
7     merge_sort(list1, middle + 1, right_index)
8     merge(list1, left_index, right_index, middle)
9
10
11 def merge(list1, left_index, right_index, middle):
12
13     left_sublist = list1[left_index:middle + 1]
14     right_sublist = list1[middle+1:right_index+1]
15
16     left_sublist_index = 0
17     right_sublist_index = 0
18     sorted_index = left_index
19
20     while left_sublist_index < len(left_sublist) and right_sublist_index < len(right_sublist):
21
22         if left_sublist[left_sublist_index] <= right_sublist[right_sublist_index]:
23             list1[sorted_index] = left_sublist[left_sublist_index]
24             left_sublist_index = left_sublist_index + 1
25         else:
26             list1[sorted_index] = right_sublist[right_sublist_index]
27             right_sublist_index = right_sublist_index + 1
28         sorted_index = sorted_index + 1
29     while left_sublist_index < len(left_sublist):
30         list1[sorted_index] = left_sublist[left_sublist_index]
31         left_sublist_index = left_sublist_index + 1
32         sorted_index = sorted_index + 1
33     while right_sublist_index < len(right_sublist):
34         list1[sorted_index] = right_sublist[right_sublist_index]
35         right_sublist_index = right_sublist_index + 1
36         sorted_index = sorted_index + 1
37
38
39 list1 = [44, 65, 2, 3, 58, 14, 57, 23, 10, 1, 7, 74, 48]
40 merge_sort(list1, 0, len(list1) - 1)
41 print(list1)
```

OUTPUT :-

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL COMMENTS

```
PS R:\hiteshsir_practical> python -u "r:\hiteshsir_practical\labexercise11.py"
[1, 2, 3, 7, 10, 14, 23, 44, 48, 57, 58, 65, 74]
PS R:\hiteshsir_practical>
```

LAB EXERCISE 13th

Aim :- To write a Python program first n prime numbers.

DESCRIPTION :-

- First we will take number of limit n as input.
- Then we will use for loop to iterate the number from 1 to n.
- Then we will check each number to be a prime number. If it is a prime number then we will simply use print function to print it.

CODE :-

```
labexercise12.py > ...
1  prime=0
2  def primenum(x):
3      if x>=2:
4          for y in range(2,x):
5              if not(x%y):
6                  return False
7      else:
8          return False
9      return True
10 for i in range(int(input("How many numbers you wish to check: "))):
11     if primenum(i):
12         prime+=1
13         print(i)
14 print("We found "+str(prime)+ " prime numbers.")
```

OUTPUT :-

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL COMMENTS

```
PS R:\hiteshsir_practical> python -u "r:\hiteshsir_practical\labexercise12.py"
How many numbers you wish to check: 20
2
3
5
7
11
13
17
19
We found 8 prime numbers.
PS R:\hiteshsir_practical> █
```

LAB EXERCISE 14TH

Aim: Implementation of Data Science concepts using Python

Description:

This Data Science with Python program provides learners with a complete understanding of data analytics tools & techniques. Getting started with Python can help you gain knowledge on data analysis, visualization, NumPy, SciPy, web scraping, and natural language processing.

Python offers a good number of libraries used in data science such as Pandas, Numpy, and Scikit-learn. Learning those libraries right away and skipping the basics isn't good though.

If you would like to learn Python for data science, you should master Python core concepts first. Having a solid foundation in Python will help you avoid common mistakes and bad practices. As a result, learning Python libraries used in data science will be much easier.

In this guide, we'll see some must-know Python concepts every data scientist should know. At the end of this article, you will find a Python for Data Science Cheat Sheet in PDF version (section 9 in the table of contents below

1. Python Attributes vs Methods

I can't tell how long I used the words "attribute" and "method" interchangeably when I was a beginner in Python. and methods, so it's good to know what's the difference between them.

- Attribute: An attribute is a variable stored in a class. That is, a value
- Method: A method is a function that is defined inside a class body.

When you learn libraries like Pandas, you'll frequently call attributes and methods, so it's good to know what the difference between them.

- Attributes : An attributes is a variables stored in a class. That is a value.
- Method : A method is a function that is defined inside a class body.