Course Description

- Prerequisite:
  - Object oriented Concepts, Basic programming constructs.

- Course Objectives
  - • To understand and use the basic of python.
  - • To understand advance concepts of python and able to apply it for solving the complex problems.
  - • To understand the reading and writing data through file handling.
  - • To understand basic database concepts in python.
  - • To develop the critical thinking and analytical approach by using python libraries.

Pseudo Code

No_Of_Takeoffs, No_Of_Landings, Initial_No_Of_Flights, Current_No_Of_Flights are variables used in this pseudo-code.

```
input No_Of_Takeoffs, No_Of_Landings
Initial_No_Of_Flights = 100
Current_No_Of_Flights = Initial_No_Of_Flights + No_Of_Landings - No_Of_Takeoffs
display "Current number of flights", Current_No_Of_Flights
```

To assign a value to the variable, we can use the = symbol. It is also called as the assignment operator. It's called an operator because it operates on the data.

Variables are like containers for data(i.e. they hold the data) and the value of the variable can vary in the pseudo-code.

Operators

- Like assignment operator, there are other operators also which can be used to perform various operations.
  Arithmetic operators: Used for performing arithmetic operations

Sr. No.	Operators	Description
1	+	Addition
2	-	Subtraction
3	*	Multiplication
4	/	Division
5	%	Modulus

- **Relational operators:** Also known as comparison operators, are used to compare values. Result of a relational expression is always either true or false.

Sr. No.	Operators	Description
1	==	Equal to
2	<	Less than
3	>	Greater than
4	<=	Less than or equal to
5	>=	Greater than or equal to
6	!=	Not equal to

Logical operators are used to combine one or more relational expressions.

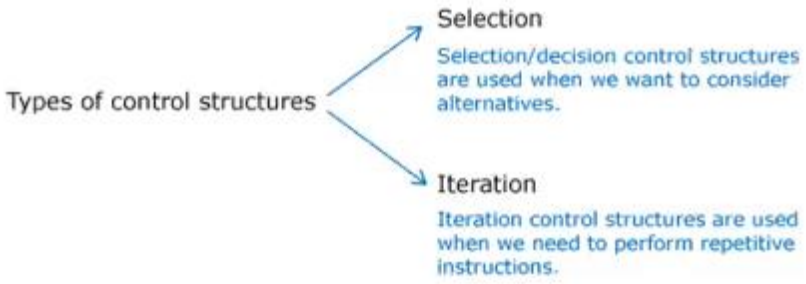
Operators	Description
AND	Result will be true, if both the expressions are true. If any one or both the expressions are false, the result will be false
OR	Result will be true, even if one of the expression is true. If both the expressions are false, the result will be false
NOT	If the expression is true, result will be false and vice versa

- If A and B are two relational expressions, say A = (Num1>2000), B= (Num2>100), the result of combining A and B using logical operator is based on the result of A and B as shown below:

A	B	A AND B
True	True	True
True	False	False
False	True	False
False	False	False

A	NOT A
True	False
False	True

A	B	A OR B
True	True	True
True	False	True
False	True	True
False	False	False



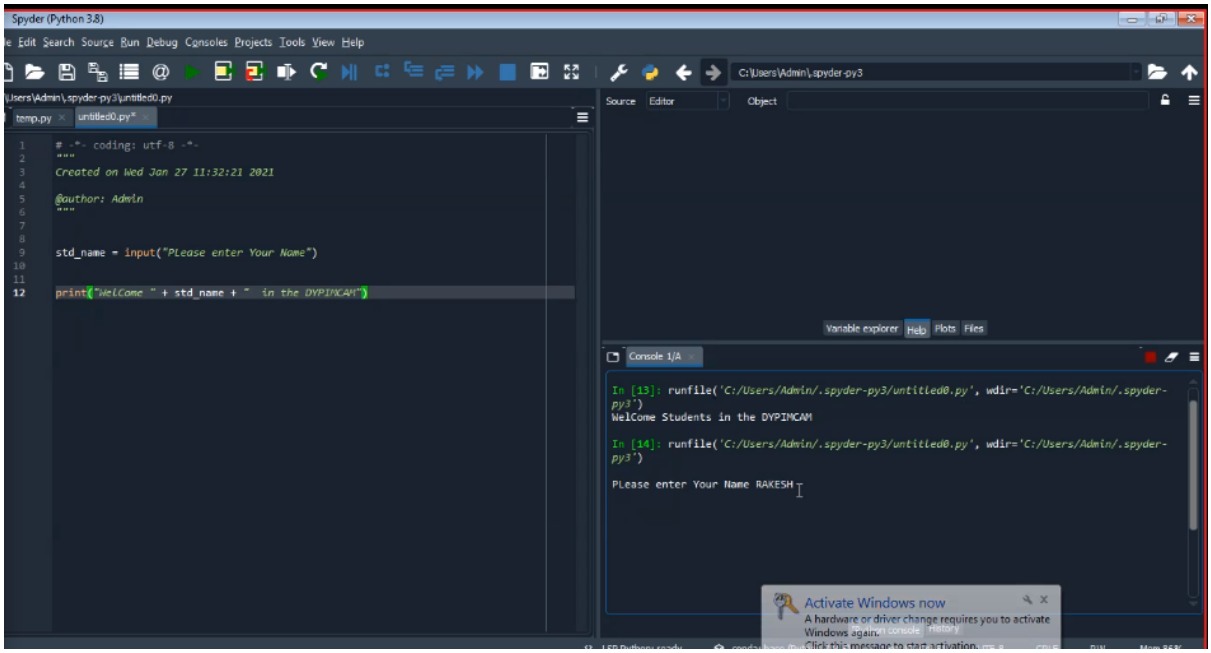
Selection using if statement

- ATC takes lot of decisions as part of its air traffic control operations.
- For example, if a flight is approaching the runway, ATC has to check if the runway is free. If the runway is not free, then the flight should not land immediately. It should circle in the air and wait for further instructions from the ATC.
- Such decision making process can be conveniently represented in a pseudo-code using an if statement.

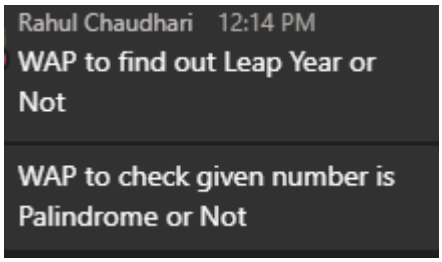
```
if (Runway == "free") then
    display "Land"
else
    display "Circle in the air"
end-if
```

if block — If result of Runway == " free" is true, if block will be executed

else block — If result of Runway == " free" is false, else block will be executed



```python
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""


mileage=15
amount_per_litre=82
distance_one_way=100
per_head_cost=0
divisible_by_five=False


per_head_cost = (((distance_one_way*2)/mileage)*amount_per_litre)/4

if (per_head_cost%5 ==0):
    divisible_by_five = True
else:
    divisible_by_five = False

#Do not modify the below print statements for verification to work
print(per_head_cost)
print(divisible_by_five)
```



PYTHON List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed,
the first item has index [0],
the second item has index [1] etc.

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
If you add new items to a list, the new items will be placed at the end of the list.

PYTHON List Example

Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

PYTHON DICTIONARY

Definition : *A dictionary is mutable and container type can store any number of python objects.*

- *Python dictionaries are also known as associative arrays or hash (#) tables.*

- *Dictionaries consists of pairs (items) of keys and their corresponding values.*

- *The general syntax of a dictionary is as follows:*

  dict= {'A': '215', 'B': '412', 'C': '125'}

Important point of dictionary

- *Each key is separated from its value by a colon (:).*
- *The items are separated by commas, and the whole thing is enclosed in curly braces { }.*
- *An empty dictionary without any items is written with just two curly braces, like this { }.*
- *The values of a dictionary can be of any type, but the keys must be an mutable data type such as strings, numbers, or tuples.*

Accessing values in dictionary:

- *To access dictionary elements, we use square brackets [ ] along with the key to obtain its value.*

  for example,

dict= {'Name': 'Sagar', 'Age': '21 'Class': 'MCA'}

Print dict

Print(dict["Name"])

Deletion in dictionary:

- *We can either remove or delete individual dictionary elements or clear the entire contents of a dictionary.*
- *To delete or remove in dictionary; we use the del statement.*

for example,

```
dict= {'Name': 'Sagar', 'Age': '18', 'Class': 'BCA'}
del dict[Name];          # remove entry with key 'Name'
dict.clear0;             # remove all entries in dict
del dict;                # delete entire dictionary
```

This is a Python Program to find the largest number in a list.

- **Problem Description**
- The program takes a list and prints the largest number in the list.
- **Problem Solution**
- 1. Take in the number of elements and store it in a variable.
  2. Take in the elements of the list one by one.
  3. Sort the list in ascending order.
  4. Print the last element of the list.
  5. Exit.

Program/Source Code

```
a=[]
n=int(input("Enter number of elements:"))
for i in range(1,n+1):
    b=int(input("Enter element:"))
    a.append(b)
a.sort()
print("Largest element is:",a[n-1])
```

- **Program Explanation**
- 1. User must enter the number of elements and store it in a variable.
  2. User must then enter the elements of the list one by one using a for loop and store it in a list.
  3. The list should then be sorted.
  4. Then the last element of the list is printed which is also the largest element of the list.

Python Program to Put Even and Odd elements in a List into Two Different Lists

```
a=[]
n=int(input("Enter number of elements:"))
for i in range(1,n+1):
    b=int(input("Enter element:"))
    a.append(b)
even=[]
odd=[]
for j in a:
    if(j%2==0):
        even.append(j)
    else:
        odd.append(j)
print("The even list",even)
print("The odd list",odd)
```

Tuple in Python

- A tuple is a collection of objects which ordered and immutable.
- Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

- tup1 = ('physics', 'chemistry', 1997, 2000);
- tup2 = (1, 2, 3, 4, 5);
- tup3 = "a", "b", "c", "d";

Repetition in Tuples

- # Code to create a tuple with repetition

- tuple3 = ('python',)*3
- print(tuple3)

- Output
- ('python', 'python', 'python')

Immutable Tuples

- #code to test that tuples are immutable

- tuple1 = (0, 1, 2, 3)
- tuple1[0] = 4
- print(tuple1)

- Output
- Traceback (most recent call last): File "e0eaddff843a8695575daec34506f126.py", line 3, in tuple1[0]=4 TypeError: 'tuple' object does not support item assignment

Slicing in Tuples

- # code to test slicing

- tuple1 = (0 ,1, 2, 3)
- print(tuple1[1:])
- print(tuple1[::-1])
- print(tuple1[2:4])

- Output
- (1, 2, 3)
- (3, 2, 1, 0)
- (2, 3)

Finding Length of a Tuple

- # Code for printing the length of a tuple

- tuple2 = ('python', ‘DYPIMCAM')
- print(len(tuple2))

Converting list to a Tuple

- # Code for converting a list and a string into a tuple

- list1 = [0, 1, 2]
- print(tuple(list1))
- print(tuple('python')) # string 'python‘

- Output
- (0, 1, 2)
- ('p', 'y', 't', 'h', 'o', 'n')

set in Python

What is a set in Python?

- A set is an unordered collection of items.
- Every element is unique (no duplicates) and must be immutable (which cannot be changed).
- However, the set itself is mutable. We can add or remove items from it.
- Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

How to create a set?

- A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().
- It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like list, set or dictionary, as its element.

Example

- # set of integers
- my_set = {1, 2, 3}
- print(my_set)

- # set of mixed data types
- my_set = {1.0, "Hello", (1, 2, 3)}
- print(my_set)

How to change a set in Python?

- We can add single element using the add() method and multiple elements using the update() method.
- The update() method can take tuples, lists, strings or other sets as its argument.
- In all cases, duplicates are avoided.

Example

- my_set = {1,3}
- print(my_set)

- my_set.add(2)
- print(my_set)

- my_set.update([2,3,4])
- print(my_set)

- my_set.update([4,5], {1,6,8})
- print(my_set)

- Similarly, we can remove and return an item using the pop() method.
- Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary.
- We can also remove all items from a set using clear().

Python Set Operations

- Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference.
- We can do this with operators or methods.

Set Union

- Union of A and B is a set of all elements from both sets.
- A = {1, 2, 3, 4, 5}B = {4, 5, 6, 7, 8}
- # use | operator
- # Output: {1, 2, 3, 4, 5, 6, 7, 8}
- print(A | B)

Set Intersection

- Intersection of A and B is a set of elements that are common in both sets.

- # initialize A and B
- A = {1, 2, 3, 4, 5}
- B = {4, 5, 6, 7, 8}
- # use & operator
- # Output: {4, 5}
- print(A & B)

- # initialize A and B
- A = {1, 2, 3, 4, 5}
- B = {4, 5, 6, 7, 8}
- # use - operator on A
- # Output: {1, 2, 3}
- print(A - B)

Set Difference

- Difference of A and B (A - B) is a set of elements that are only in A but not in B.
- Similarly, B - A is a set of element in B but not in A.
- Difference is performed using - operator.
- Same can be accomplished using the method difference().

Iterating Through a Set

- Using a for loop, we can iterate though each item in a set.
- for letter in set("apple"):

```
    print(letter)
```

o/p

a p e l

Python Frozenset

- Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned.
- While tuples are immutable lists, frozensets are immutable sets.
- Frozensets can be created using the function frozenset().

Example

- # initialize A and B
- A = frozenset([1, 2, 3, 4])
- B = frozenset([3, 4, 5, 6])

- >>> A.isdisjoint(B)
- False
- >>> A.difference(B)
- frozenset({1, 2})
- >>> A | B
- frozenset({1, 2, 3, 4, 5, 6})
- >>> A.add(3)
- ... AttributeError: 'frozenset' object has no attribute 'add'

Python Program to Merge Two Lists and Sort it

```
a=[]
c=[]
n1=int(input("Enter number of elements:"))
for i in range(1,n1+1):
    b=int(input("Enter element:"))
    a.append(b)
n2=int(input("Enter number of elements:"))
for i in range(1,n2+1):
    d=int(input("Enter element:"))
    c.append(d)
new=a+c
new.sort()
print("Sorted list is:",new)
```

Python Program to Find the Second Largest Number in a List Using Bubble Sort

```
a=[]
n=int(input("Enter number of elements:"))
for i in range(1,n+1):
    b=int(input("Enter element:"))
    a.append(b)
for i in range(0,len(a)):
    for j in range(0,len(a)-i-1):
        if(a[j]>a[j+1]):
            temp=a[j]
            a[j]=a[j+1]
            a[j+1]=temp
print('Second largest number is:',a[n-2])
```

Program/Source Code

```
a=[]
n=int(input("Enter number of elements:"))
for i in range(1,n+1):
    b=int(input("Enter element:"))
    a.append(b)
a.sort()
print("Largest element is:",a[n-1])
```

Python Program to Generate Random Numbers from 1 to 20 and Append Them to the List

```
import random
a=[]
n=int(input("Enter number of elements:"))
for j in range(n):
    a.append(random.randint(1,20))
print('Randomised list is: ',a)
```

Python Program to Add Two Matrices

```
X = [[12,7,3],
     [4 ,5,6],
     [7 ,8,9]]

Y = [[5,8,1],
     [6,7,3],
     [4,5,9]]

result = [[0,0,0],
          [0,0,0],
          [0,0,0]]

# iterate through rows
for i in range(len(X)):
   # iterate through columns
   for j in range(len(X[0])):
       result[i][j] = X[i][j] + Y[i][j]

for r in result:
   print(r)
```

Python Programs On Dictionary

Python Program to Add a Key-Value Pair to the Dictionary

```
key=int(input("Enter the key (int) to be added:"))
value=int(input("Enter the value for the key to be added:"))
d={} d.update({key:value})
print("Updated dictionary is:")
print(d)
```

Python Program to Generate a Dictionary that Contains Numbers (between 1 and n) in the Form (x,x*x).

```
n=int(input("Enter a number:"))
d={x:x*x for x in range(1,n+1)}
print(d)
```

```
d = {}
for x in range(1,n+1):
    d[x] = x*x
```

Python Program to Sum All the Items in a Dictionary

```
d={'A':100,'B':540,'C':239}
print("Total sum of values in the dictionary:")
print(sum(d.values()))
```

Python Program to Multiply All the Items in a Dictionary

```
d={'A':10,'B':10,'C':239}
tot=1
for i in d:
    tot=tot*d[i]
print(tot)
```

Python Program to Count the Frequency of Words Appearing in a String Using a Dictionary

```
test_string=input("Enter string:")
l=[]
l=test_string.split()
wordfreq=[l.count(p) for p in l]
print(dict(zip(l,wordfreq)))
```

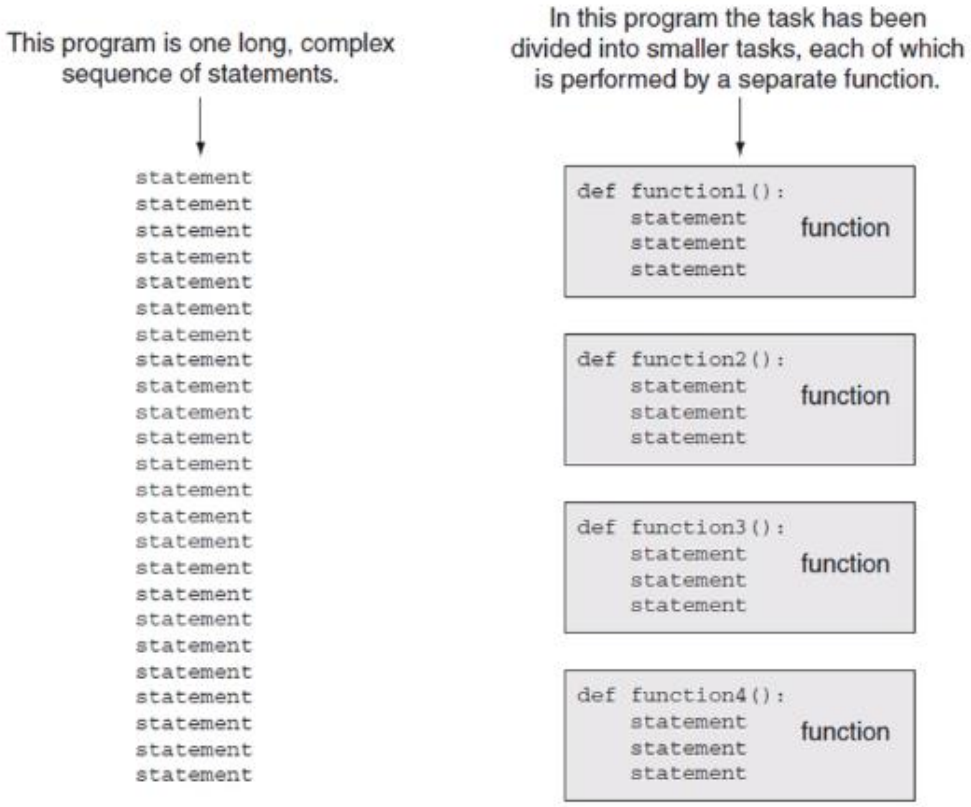
Python Program to Create a Dictionary with Key as First Character and Value as Words Starting with that Character

```
test_string=input("Enter string:")
l=test_string.split()
d={}
for word in l:
    if(word[0] not in d.keys()):
        d[word[0]]=[]
        d[word[0]].append(word)
    else:
        if(word not in d[word[0]]):
          d[word[0]].append(word)
for k,v in d.items():
        print(k,":",v)
```

Introduction to Functions

- Function: group of statements within a program that perform as specific task
  - Usually one task of a large program
    - Functions can be executed in order to perform overall program task
  - Known as *divide and conquer* approach
- Modularized program: program where in each task within the program is in its own function

Figure 5-1 Using functions to divide and conquer a large task



Benefits of Modularizing a Program with Functions

- The benefits of using functions include:
  - Simpler code
  - Code reuse
    - write the code once and call it multiple times
  - Better testing and debugging
    - Can test and debug each function individually
  - Faster development
  - Easier facilitation of teamwork
    - Different team members can write different functions

Defining and Calling a Function

- Functions are given names
  - Function naming rules:
    - Cannot use key words as a function name
    - Cannot contain spaces
    - First character must be a letter or underscore
    - All other characters must be a letter, number or underscore
    - Uppercase and lowercase characters are distinct

Defining A function

Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax

- def functionname(parameters):
  "function_docstring" function_suite return [expression]

Calling a Function

- # Function definition is here
- def printme(str):
- "This prints a passed string into this function"
- print str
- return;

- # Now you can call printme function
- printme("I'm first call to user defined function!")
- printme("Again second call to the same function")

Pass by reference

- All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example –

```
# Function definition is here
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist.append([1,2,3,4]);
   print "Values inside the function: ", mylist
   return

# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

Function Arguments

You can call a function by using the following types of formal arguments –

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Keyword arguments

- Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

```
# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print str
   return;

# Now you can call printme function
printme( str = "My string")
```

Default arguments

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

```
# Function definition is here
def printinfo( name, age = 35 ):
   "This prints a passed info into this function"
   print "Name: ", name
   print "Age ", age
   return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

Variable-length arguments

- You may need to process a function for more arguments than you specified while defining the function. These arguments are called *variable-length* arguments and are not named in the function definition, unlike required and default arguments.

Syntax for a function with non-keyword variable arguments is this –

```
def functionname([formal_args,] *var_args_tuple ):
   "function_docstring"
   function_suite
   return [expression]
```

- An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call.

```
Function definition is here
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print "Output is: "
   print arg1
   for var in vartuple:
      print var
   return;

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

- The flight ticket rates for a round-trip (Mumbai->Dubai) were as follows:
  Rate per Adult: Rs. 37550.0
  Rate per Child: 1/3rd of the rate per adult
  Service Tax: 7% of the ticket amount (including all passengers)
  As it was a holiday season, the airline also offered 10% discount on the final ticket cost (after inclusion of the service tax).
  Find and display the total ticket cost for a group which had adults and children.

  Test the program with different input values for number of adults and children.

```
ticket = 37750;
def ticket_cal(adults,children):

    cost = ticket*adults+(ticket/3)*children
    with_tax_cost = cost + (cost*0.07)
    final_cost = with_tax_cost - (with_tax_cost*0.1)
    return final_cost
a = int(input("Enter the number of Adults "))
c = int(input("Enter the number of childrens"))

x = ticket_cal(a,c)
print("Total Ticket Cost:",x)
```

- Write a python program to find and display the product of three positive integer values based on the rule mentioned below:
- It should display the product of the three values except when one of the integer value is 7. In that case, 7 should not be included in the product and the values to its left also should not be included.
- If there is only one value to be considered, display that value itself. If no values can be included in the product, display -1
- Note: Assume that if 7 is one of the positive integer values, then it will occur only once. Refer the sample I/O given below.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 17 22:52:24 2021
```

```
@author: aparna
"""

def product(x,y,z):
    if(x==7):
        temp = y*z
    elif(y==7):
        temp = z
    elif(z == 7):
        temp = -1
    else:
        temp = x*y*z

        return temp
a = int(input("Enter the 1st number  "))
b = int(input("Enter the 2nd number "))
c = int(input("Enter the 3rd number "))

p = product(a, b, c)
print("product =" , p);
```

- You have x no. of 5 rupee coins and y no. of 1 rupee coins. You want to purchase an item for amount z. The shopkeeper wants you to provide exact change. You want to pay using minimum number of coins. How many 5 rupee coins and 1 rupee coins will you use? If exact change is not possible then display -1.

Sample Input			Expected Output	
Available Rs. 1 coins	Available Rs. 5 notes	Amount to be made	Rs. 1 coins needed	Rs. 5 notes needed
2	4	21	1	4
11	2	11	1	2
3	3	19	-1	

- An organization has decided to provide salary hike to its employees based on their job level. Employees can be in job levels 3 , 4 or 5. Hike percentage based on job levels are given below:

Job level	Hike Percentage (applicable on current salary)
3	15
4	7
5	5

- In case of invalid job level, consider hike percentage to be 0.
- Given the current salary and job level, write a python program to find and display the new salary of an employee.

```
def find_new_salary(current_salary,job_level):
    # write your logic here

    return new_salary

# provide different values for current_salary and job_level and test yor program
new_salary=find_new_salary(10000,3)
print(new_salary)
```

- A traveler on a visit to India is in need of some Indian Rupees (INR) but he has money belonging to another currency. He wants to know how much money he should provide in the currency he has, to get the specified amount in INR.
- Write a python program to implement a currency calculator which accepts the amount needed in INR and the name of the currency which the traveler has. The program should identify and display the amount the traveler should provide in the currency he has, to get the specified amount in INR.
- Note: Use the forex information provided in the table below for the calculation. Consider that only the currency names mentioned in the table are valid. For any invalid currency name, display -1.

Currency	Equivalent of 1.00 INR
Euro	0.01417
British Pound	0.0100
Australian Dollar	0.02140
Canadian Dollar	0.02027

```
def convert_currency(amount_needed_inr,current_currency_name):
    current_currency_amount=0
    #write your logic here
    return current_currency_amount


#Provide different values for amount_needed_inr,current_currency_name and test your program
currency_needed=convert_currency(2000,"Euro")
if(currency_needed!= -1):
    print(currency_needed )
else:
    print("Invalid currency name")
```

Python abs()

returns absolute value of a number

Python all()

returns true when all elements in iterable is true

Python any()

Checks if any Element of an Iterable is True

Python ascii()

Returns String Containing Printable Representation

Python bin()

converts integer to binary string

Python bool()

Converts a Value to Boolean

Python bytearray()

Python bytearray()

returns array of given byte size

Python bytes()

returns immutable bytes object

Python callable()

Checks if the Object is Callable

Python chr()

Returns a Character (a string) from an Integer

Python classmethod()

returns class method for given function

Python compile()

Returns a Python code object

Python classmethod()

returns class method for given function

Python compile()

Returns a Python code object

Python complex()

Creates a Complex Number

Python delattr()

Deletes Attribute From the Object

Python dict()

Creates a Dictionary

Python dir()

Tries to Return Attributes of Object

Python divmod()

Returns a Tuple of Quotient and Remainder

Python divmod()

Returns a Tuple of Quotient and Remainder

Python enumerate()

Returns an Enumerate Object

Python eval()

Runs Python Code Within Program

Python exec()

Executes Dynamically Created Program

Python filter()

constructs iterator from elements which are true

constructs iterator from elements which are true

Python float()

returns floating point number from number, string

Python format()

returns formatted representation of a value

Python frozenset()

returns immutable frozenset object

Python getattr()

returns value of named attribute of an object

Python globals()

returns dictionary of current global symbol table

Python hasattr()

returns whether object has named attribute

Python hash()

returns hash value of an object

Python help()

Invokes the built-in Help System

Python hex()

Converts to Integer to Hexadecimal

Python id()

Returns Identity of an Object

Python input()

reads and returns a line of string

Python int()

returns integer from a number or string

Python isinstance()

Python issubclass()

Checks if a Class is Subclass of another Class

Python iter()

returns an iterator

Python len()

Returns Length of an Object

Python list()

creates a list in Python

Python locals()

Returns dictionary of a current local symbol table

Python map()

Applies Function and Returns a List

Python max()

returns the largest item

Python memoryview()

returns memory view of an argument

Python min()

Python oct()

returns the octal representation of an integer

Python open()

Returns a file object

Python ord()

returns an integer of the Unicode character

Python pow()

returns the power of a number

Python print()

Prints the Given Object

Python property()

returns the property attribute

Python range()

return sequence of integers between start and stop

Python repr()

returns a printable representation of the object

Python reversed()

returns the reversed iterator of a sequence

Python round()

rounds a number to specified decimals

Python set()

constructs and returns a set

Python setattr()

sets the value of an attribute of an object

Python slice()

returns a slice object

Python sorted()

returns a sorted list from the given iterable

Python staticmethod()

transforms a method into a static method

Python str()

returns the string version of the object

Python sum()

Adds items of an Iterable

Python super()

Returns a proxy object of the base class

Python tuple()

Returns a tuple

Python type()

Returns the type of the object

Python vars()

Returns the __dict__ attribute

Python zip()

Returns an iterator of tuples

Python __import__()

Function called by the import statement

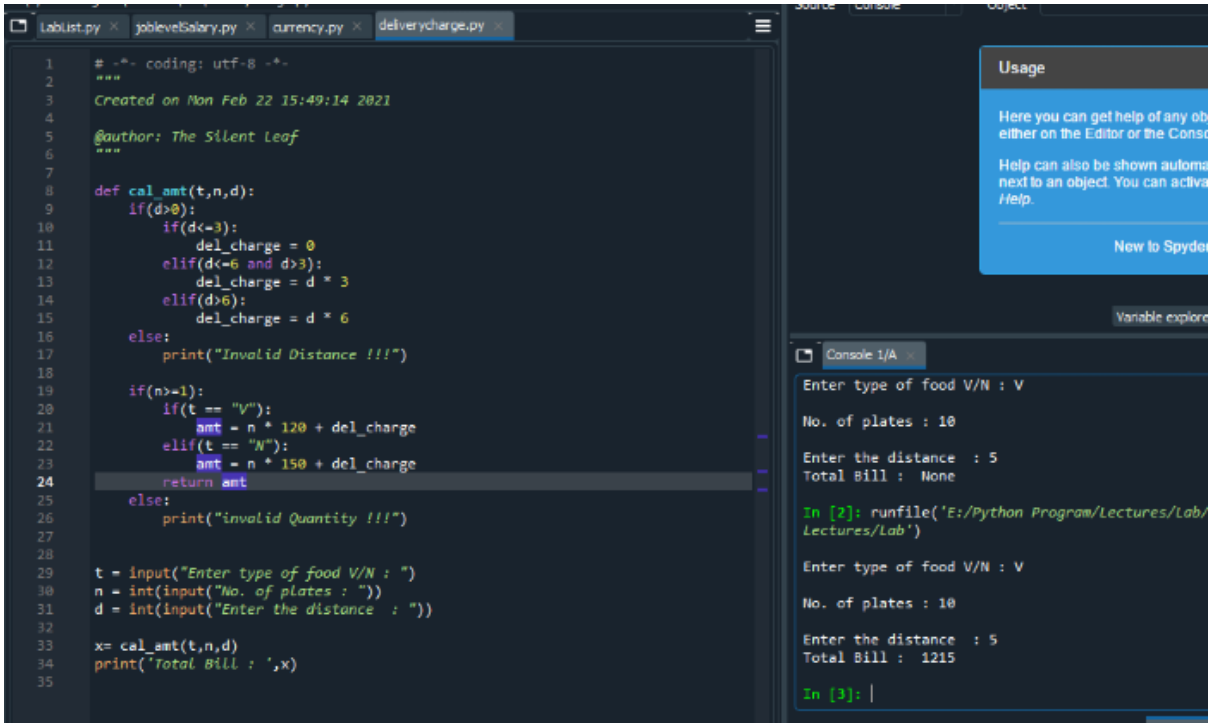
- FoodCorner home delivers vegetarian and non-vegetarian combos to its customer based on order.

  A vegetarian combo costs Rs.120 per plate and a non-vegetarian combo costs Rs.150 per plate. Their non-veg combo is really famous that they get more orders for their non-vegetarian combo than the vegetarian combo.

  Apart from the cost per plate of food, customers are also charged for home delivery based on the distance in kms from the restaurant to the delivery point. The delivery charges are as mentioned below:

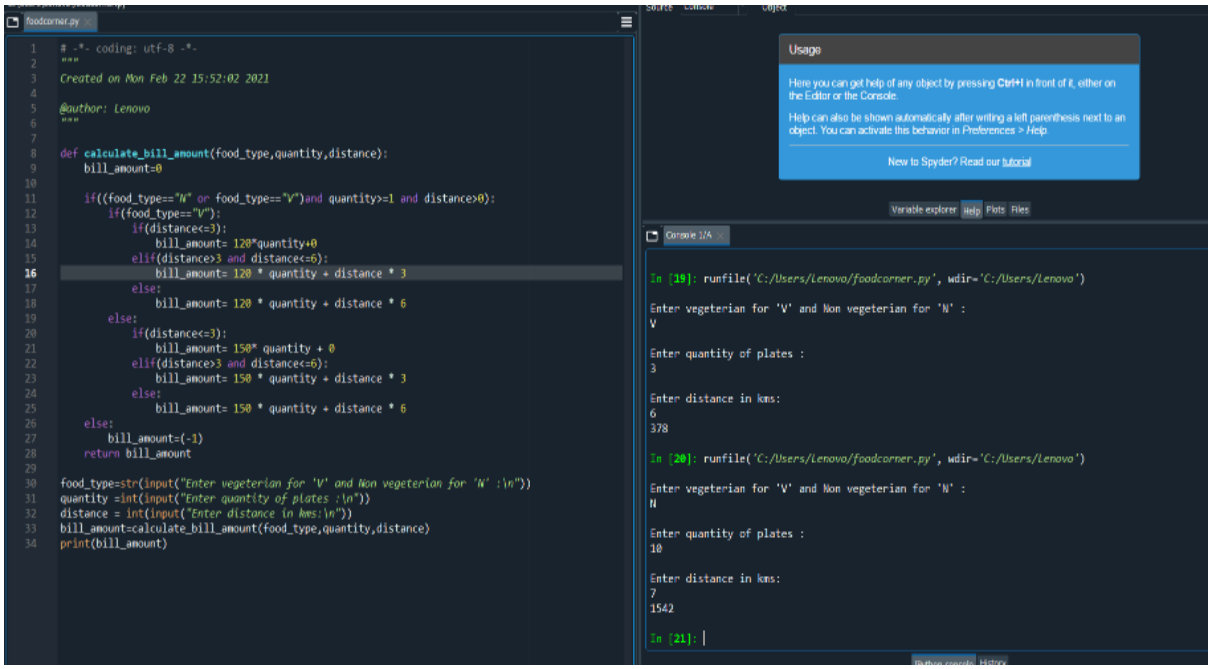
Distance in kms	**Delivery charge in Rs per km**
For first 3kms	0
For next 3kms	3
For the remaining	6

```
def calculate_bill_amount(food_type,quantity_ordered,distance_in_kms):
bill_amount=0
#write your logic here
return bill_amount

#Provide different values for food_type,quantity_ordered,distance_in_kms and test your program
bill_amount=calculate_bill_amount("N",2,7)
print(bill_amount)
```



```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 22 15:49:14 2021

@author: The Silent Leaf
"""

def cal_amt(t,n,d):
    if(d>0):
        if(d<=3):
            del_charge = 0
        elif(d<=6 and d>3):
            del_charge = d * 3
        elif(d>6):
            del_charge = d * 6
    else:
        print("Invalid Distance !!!")

    if(n>=1):
        if(t == "V"):
            amt = n * 120 + del_charge
        elif(t == "N"):
            amt = n * 150 + del_charge
        return amt
    else:
        print("invalid Quantity !!!")


t = input("Enter type of food V/N : ")
n = int(input("No. of plates : "))
d = int(input("Enter the distance  : "))

x= cal_amt(t,n,d)
print('Total Bill : ',x)
```

```
Enter type of food V/N : V

No. of plates : 10

Enter the distance  : 5
Total Bill :  None

In [2]: runfile('E:/Python Program/Lectures/Lab/
Lectures/Lab')

Enter type of food V/N : V

No. of plates : 10

Enter the distance  : 5
Total Bill :  1215

In [3]:
```



```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 22 15:52:02 2021

@author: Lenovo
"""

def calculate_bill_amount(food_type,quantity,distance):
    bill_amount=0

    if((food_type=="N" or food_type=="V")and quantity>=1 and distance>0):
        if(food_type=="V"):
            if(distance<=3):
                bill_amount= 120*quantity+0
            elif(distance>3 and distance<=6):
                bill_amount= 120 * quantity + distance * 3
            else:
                bill_amount= 120 * quantity + distance * 6
        else:
            if(distance<=3):
                bill_amount= 150* quantity + 0
            elif(distance>3 and distance<=6):
                bill_amount= 150 * quantity + distance * 3
            else:
                bill_amount= 150 * quantity + distance * 6
    else:
        bill_amount=(-1)
    return bill_amount

food_type=str(input("Enter vegeterian for 'V' and Non vegeterian for 'N' :\n"))
quantity =int(input("Enter quantity of plates :\n"))
distance = int(input("Enter distance in kms:\n"))
bill_amount=calculate_bill_amount(food_type,quantity,distance)
print(bill_amount)
```

```
In [19]: runfile('C:/Users/Lenovo/foodcorner.py', wdir='C:/Users/Lenovo')

Enter vegeterian for 'V' and Non vegeterian for 'N' :
V

Enter quantity of plates :
3

Enter distance in kms:
6
378

In [20]: runfile('C:/Users/Lenovo/foodcorner.py', wdir='C:/Users/Lenovo')

Enter vegeterian for 'V' and Non vegeterian for 'N' :
N

Enter quantity of plates :
10

Enter distance in kms:
7
1542

In [21]:
```

- The Metro Bank provides various types of loans such as car loans, business loans and house loans to its account holders. Write a python program to implement the following requirements:
- Initialize the following variables with appropriate input values:account_number, account_balance, salary, loan_type, loan_amount_expected and customer_emi_expected.
- The account number should be of 4 digits and its first digit should be 1.
- The customer should have a minimum balance of Rupees 1 Lakh in the account.
- If the above rules are valid, determine the eligible loan amount and the EMI that the bank can provide to its customers based on their salary and the loan type they expect to avail.
- The bank would provide the loan, only if the loan amount and the number of EMI's requested by the customer is less than or equal to the loan amount and the number of EMI's decided by the bank respectively.
- Display appropriate error messages for all invalid data. If all the business rules are satisfied ,then display account number, eligible and requested loan amount and EMI's.
  Test your code by providing different values for the input variables.

Salary	Loan type	Eligible loan amount	No. of EMI's
> 25000	Car	500000	36
>50000	House	6000000	60
> 75000	Business	7500000	84

```
def
calculate_loan(account_number,salary,account_balance,loan_type,loan_amount_expected,customer_emi_expected):
eligible_loan_amount=0
bank_emi_expected=0
eligible_loan_amount=0
#Start writing your code here
#Populate the variables: eligible_loan_amount and bank_emi_expected

#Use the below given print statements to display the output, in case of success
#print("Account number:", account_number)
#print("The customer can avail the amount of Rs.", eligible_loan_amount)
#print("Eligible EMIs :", bank_emi_expected)
#print("Requested loan amount:", loan_amount_expected)
#print("Requested EMI's:",customer_emi_expected)

#Use the below given print statements to display the output, in case of invalid data.
#print("Insufficient account balance")
#print("The customer is not eligible for the loan")
#print("Invalid account number")
#print("Invalid loan type or salary")
```

Also, do not modify the above print statements for verification to work

#Test your code for different values and observe the results
calculate_loan(1001,40000,250000,"Car",300000,30)

Passing function as an Argument

```
def add(x):
    return x+1

def operate(func,  y):
    result = func(y)
    return result;

Print(operate(add,4))
```

Writing function inside the function

```
def print_msg(message):
    greeting="Hello"

    def printer():
        print(greeting.message)

    printer()


print_msg("Python is awesome")
```

Decorators in Python

- Decorators are very powerful and useful tool in Python since it allows programmers to modify the behavior of function or class.
- Decorators allow us to wrap another function in order to extend the behavior of wrapped function, without permanently modifying it.
- In Decorators, functions are taken as the argument into another function and then called inside the wrapper function.

Syntax for Decorator:

```
@gfg_decorator
def hello_decorator():
        print("Gfg")

'''Above code is equivalent to -

def hello_decorator():
        print("Gfg")

hello_decorator = gfg_decorator(hello_decorator)'''
```

- In the above code, gfg_decorator is a callable function, will add some code on the top of some another callable function, hello_decorator function and return the wrapper function.

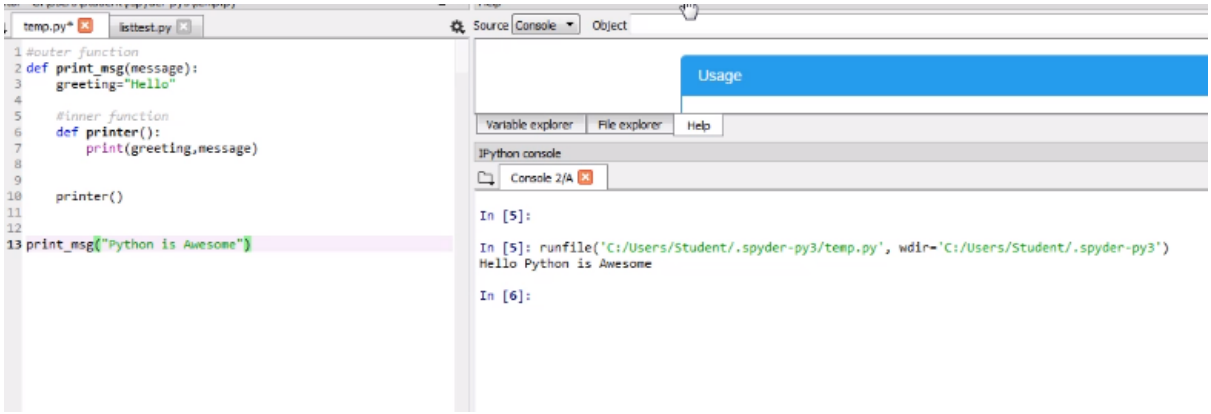
Decorator can modify the behavior:

```
# defining a decorator
def hello_decorator(func):

        # inner1 is a Wrapper function in
        # which the argument is called

        # inner function can access the outer local
        # functions like in this case "func"
        def inner1():
                print("Hello, this is before function execution")

                # calling the actual function now
                # inside the wrapper function.
                func()

                print("This is after function execution")

        return inner1


# defining a function, to be called inside wrapper
def function_to_be_used():
        print("This is inside the function !!")


# passing 'function_to_be_used' inside the
# decorator to control its behavior
function_to_be_used = hello_decorator(function_to_be_use


# calling the function
function_to_be_used()
```

- Hello, this is before function execution
- This is inside the function !!
- This is after function execution

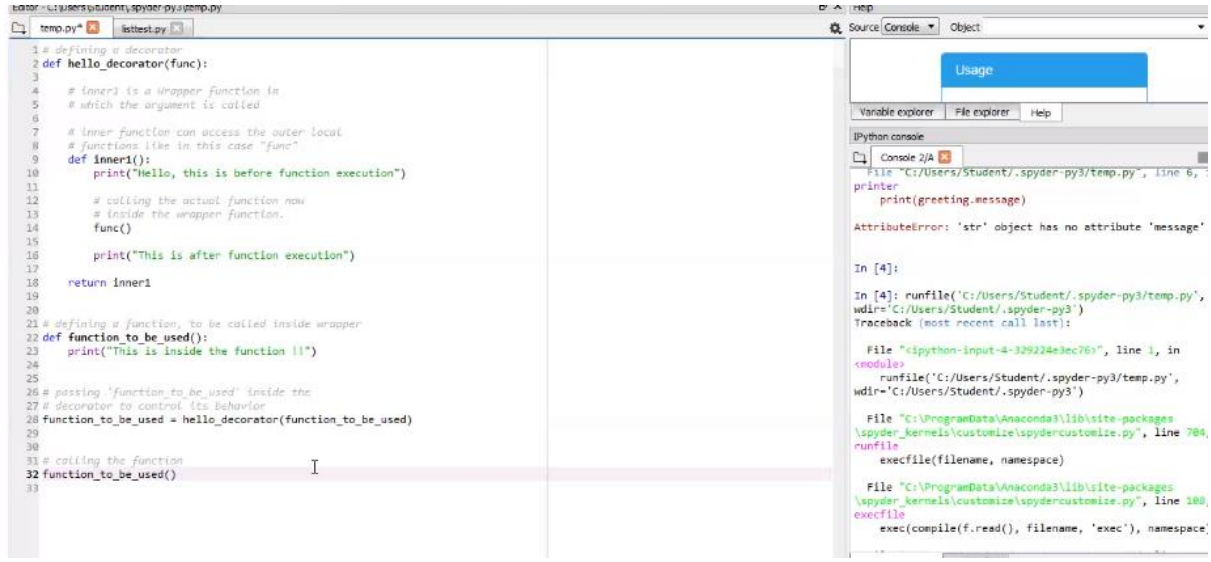
```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 24 22:37:09 2021

@author: aparna
"""
#define a decorator
def  hello_decorator(func):
   #inner1 is a Wrapper Function in
   #inner function can access the outer local
   #function like in this case "func "
   def inner1():
      print("Hello , this is before function execution ")
      #calling the actual function now
      #inside the wrapper function
      func()
      print("this is after fuction execution")
   return inner1
# defining a function to be called inside wrapper
def function_to_be_used():
   print("I am new Functionality ")
   print("this is inside the function")

#passing 'funtion_to_be_used' inside the decorater to control its behavior
function_to_be_used =  hello_decorator(function_to_be_used)
#calling the function
function_to_be_used()
```

Output :-
Hello , this is before function execution
I am new Functionality
this is inside the function
this is after fuction execution



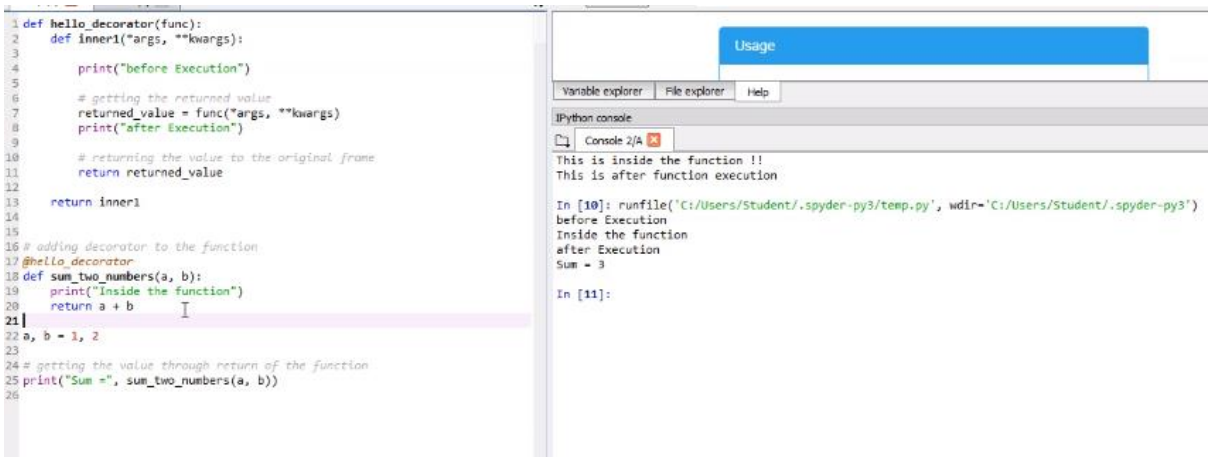
```
#outer function
def print_msg(message):
    greeting="Hello"

    #inner function
    def printer():
        print(greeting,message)


    printer()


print_msg("Python is Awesome")
```

```
In [5]:

In [5]: runfile('C:/Users/Student/.spyder-py3/temp.py', wdir='C:/Users/Student/.spyder-py3')
Hello Python is Awesome

In [6]:
```



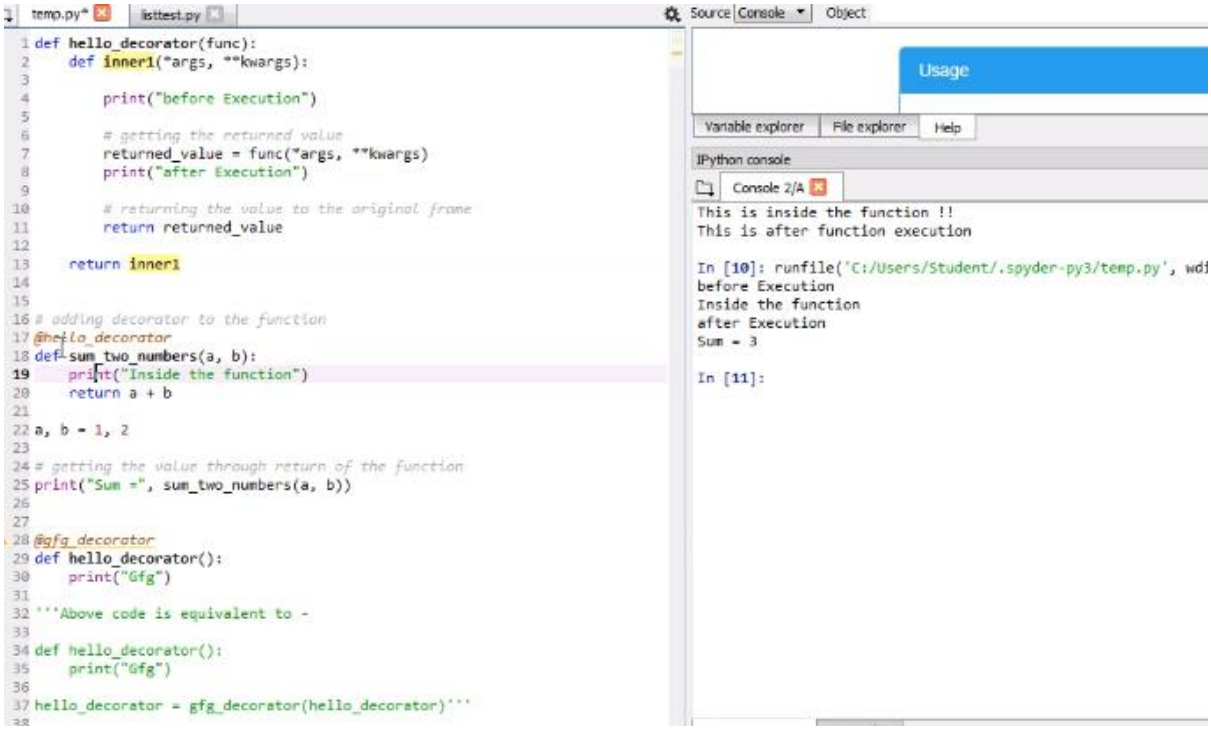
```
# defining a decorator
def hello_decorator(func):

    # inner1 is a Wrapper function in
    # which the argument is called

    # inner function can access the outer local
    # functions like in this case "func"
    def inner1():
        print("Hello, this is before function execution")

        # calling the actual function now
        # inside the wrapper function.
        func()

        print("This is after function execution")

    return inner1


# defining a function, to be called inside wrapper
def function_to_be_used():
    print("This is inside the function !!")


# passing 'function_to_be_used' inside the
# decorator to control its behavior
function_to_be_used = hello_decorator(function_to_be_used)


# calling the function
function_to_be_used()
```

```
  File "C:/Users/Student/.spyder-py3/temp.py", line 6,
printer
    print(greeting.message)

AttributeError: 'str' object has no attribute 'message'


In [4]:

In [4]: runfile('C:/Users/Student/.spyder-py3/temp.py',
wdir='C:/Users/Student/.spyder-py3')
Traceback (most recent call last):

  File "<ipython-input-4-329224e3ec76>", line 1, in
<module>
    runfile('C:/Users/Student/.spyder-py3/temp.py',
wdir='C:/Users/Student/.spyder-py3')

  File "C:\ProgramData\Anaconda3\lib\site-packages
\spyder_kernels\customize\spydercustomize.py", line 704,
runfile
    execfile(filename, namespace)

  File "C:\ProgramData\Anaconda3\lib\site-packages
\spyder_kernels\customize\spydercustomize.py", line 108,
execfile
    exec(compile(f.read(), filename, 'exec'), namespace)
```



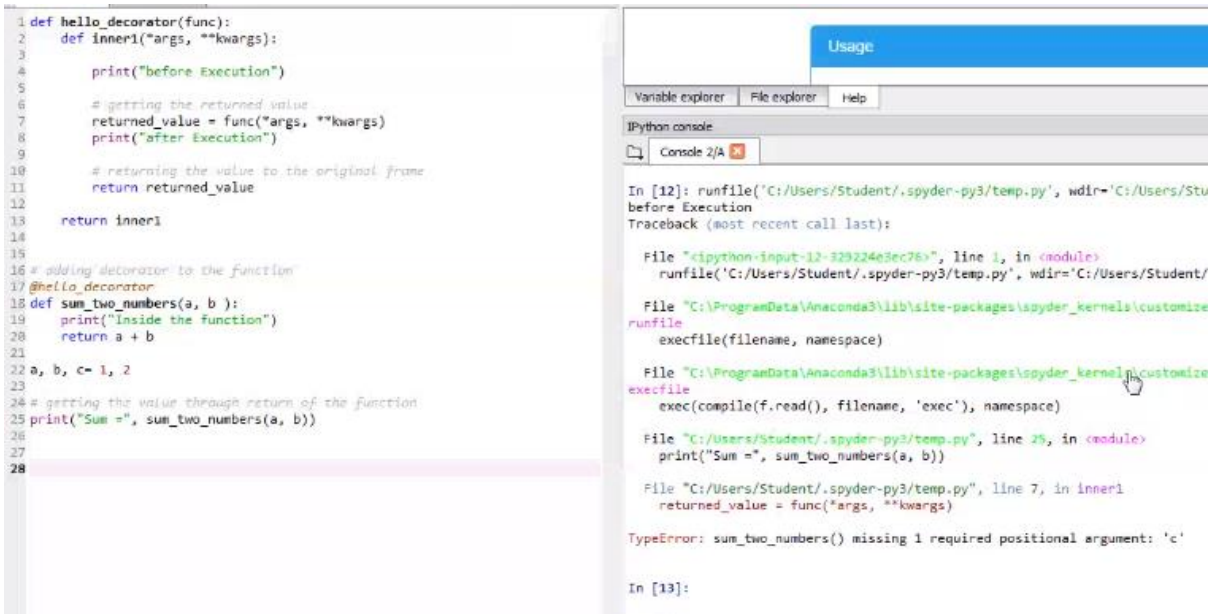
```
def hello_decorator(func):
    def inner1(*args, **kwargs):

        print("before Execution")

        # getting the returned value
        returned_value = func(*args, **kwargs)
        print("after Execution")

        # returning the value to the original frame
        return returned_value

    return inner1


# adding decorator to the function
@hello_decorator
def sum_two_numbers(a, b):
    print("Inside the function")
    return a + b

a, b = 1, 2

# getting the value through return of the function
print("Sum =", sum_two_numbers(a, b))
```

```
This is inside the function !!
This is after function execution

In [10]: runfile('C:/Users/Student/.spyder-py3/temp.py', wdir='C:/Users/Student/.spyder-py3')
before Execution
Inside the function
after Execution
Sum = 3

In [11]:
```



```
def hello_decorator(func):
    def inner1(*args, **kwargs):

        print("before Execution")

        # getting the returned value
        returned_value = func(*args, **kwargs)
        print("after Execution")

        # returning the value to the original frame
        return returned_value

    return inner1


# adding decorator to the function
@hello_decorator
def sum_two_numbers(a, b):
    print("Inside the function")
    return a + b

a, b = 1, 2

# getting the value through return of the function
print("Sum =", sum_two_numbers(a, b))


@gfg_decorator
def hello_decorator():
    print("Gfg")

'''Above code is equivalent to -

def hello_decorator():
    print("Gfg")

hello_decorator = gfg_decorator(hello_decorator)'''
```

```
This is inside the function !!
This is after function execution

In [10]: runfile('C:/Users/Student/.spyder-py3/temp.py', wdi
before Execution
Inside the function
after Execution
Sum = 3

In [11]:
```



```
def hello_decorator(func):
    def inner1(*args, **kwargs):

        print("before Execution")

        # getting the returned value
        returned_value = func(*args, **kwargs)
        print("after Execution")

        # returning the value to the original frame
        return returned_value

    return inner1


# adding decorator to the function
@hello_decorator
def sum_two_numbers(a, b ):
    print("Inside the function")
    return a + b

a, b, c= 1, 2

# getting the value through return of the function
print("Sum =", sum_two_numbers(a, b))
```

```
In [12]: runfile('C:/Users/Student/.spyder-py3/temp.py', wdir='C:/Users/Stu
before Execution
Traceback (most recent call last):

  File "<ipython-input-12-329224e3ec76>", line 1, in <module>
    runfile('C:/Users/Student/.spyder-py3/temp.py', wdir='C:/Users/Student/

  File "C:\ProgramData\Anaconda3\lib\site-packages\spyder_kernels\customize
runfile
    execfile(filename, namespace)

  File "C:\ProgramData\Anaconda3\lib\site-packages\spyder_kernels\customize
execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

  File "C:/Users/Student/.spyder-py3/temp.py", line 25, in <module>
    print("Sum =", sum_two_numbers(a, b))

  File "C:/Users/Student/.spyder-py3/temp.py", line 7, in inner1
    returned_value = func(*args, **kwargs)

TypeError: sum_two_numbers() missing 1 required positional argument: 'c'


In [13]:
```

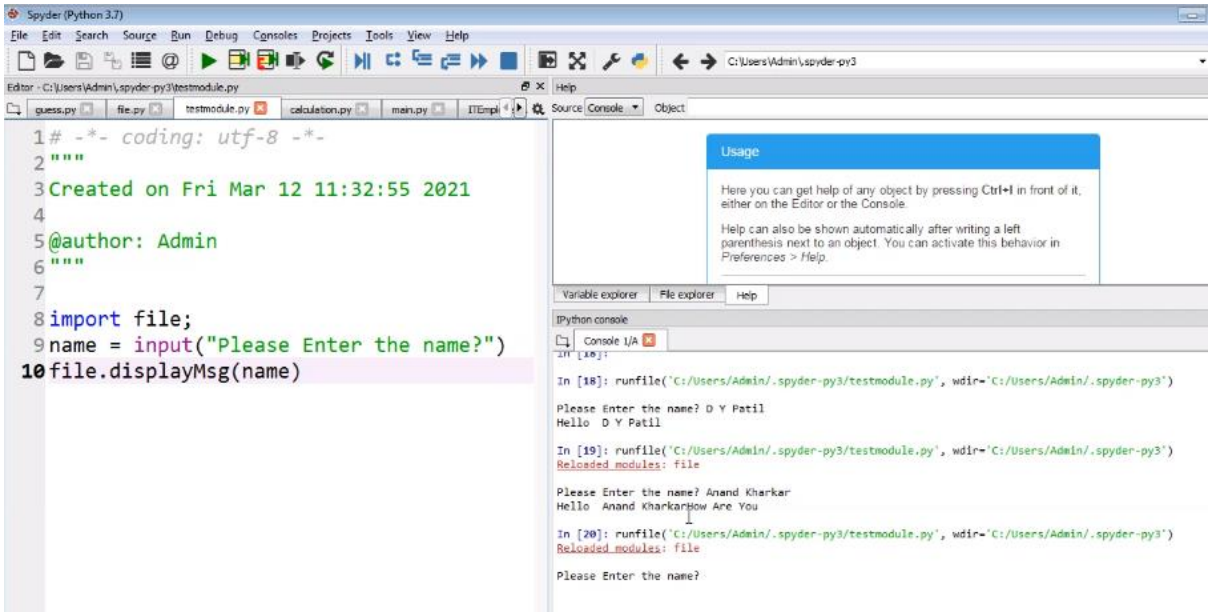
Python Module

- A python module can be defined as a python program file which contains a python code including python functions, class, or variables. In other words, we can say that our python code file saved with the extension (.py) is treated as the module.
- Modules in Python provides us the flexibility to organize the code in a logical way.
- To use the functionality of one module into another, we must have to import the specific module.

The import statement

- The import statement is used to import all the functionality of one module into another. Here, we must notice that we can use the functionality of any python source file by importing that file as the module into another python source file.
- We can import multiple modules with a single import statement, but a module is loaded once regardless of the number of times, it has been imported into our file.
- The syntax to use the import statement is given below.
- **import** module1,module2,..... module n

- **import** file;
- name = input("Enter the name?")
- file.displayMsg(name)


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Admin\.spyder-py3
Editor - C:\Users\Admin\.spyder-py3\testmodule.py
guess.py
file.py
testmodule.py
calculation.py
main.py
-*- coding: utf-8 -*-
"""
Created on Fri Mar 12 11:32:55 2021
@author: Admin
"""
import file;
name = input("Please Enter the name?")
file.displayMsg(name)
Help
Source Console Object
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.
Variable explorer
File explorer
Help
IPython console
Console 1/A
In [18]: runfile('C:/Users/Admin/.spyder-py3/testmodule.py', wdir='C:/Users/Admin/.spyder-py3')
Please Enter the name? D Y Patil
Hello D Y Patil
In [19]: runfile('C:/Users/Admin/.spyder-py3/testmodule.py', wdir='C:/Users/Admin/.spyder-py3')
Reloaded modules: file
Please Enter the name? Anand Kharkar
Hello Anand KharkarHow Are You
In [20]: runfile('C:/Users/Admin/.spyder-py3/testmodule.py', wdir='C:/Users/Admin/.spyder-py3')
Reloaded modules: file
Please Enter the name?