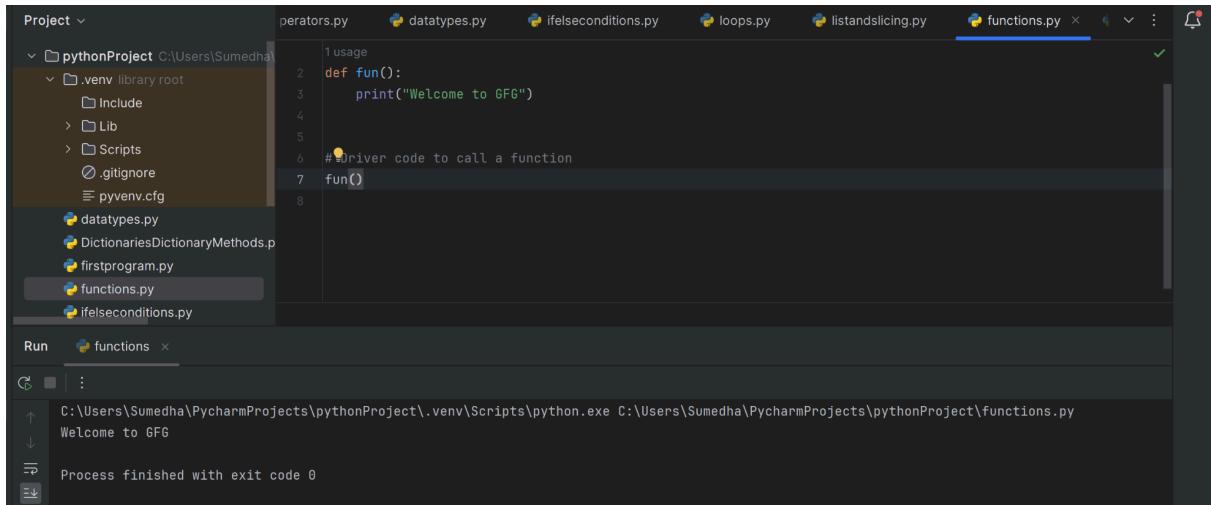


NAME: AKULA SHARATH CHANDRA
BATCH: DATA ENGINEERING
TOPICS: FUNCTIONS ,OOPS CONCEPTS

1)FUNCTIONS : Python Functions is a block of statements that return the specific task.
→ Defining and calling a function:

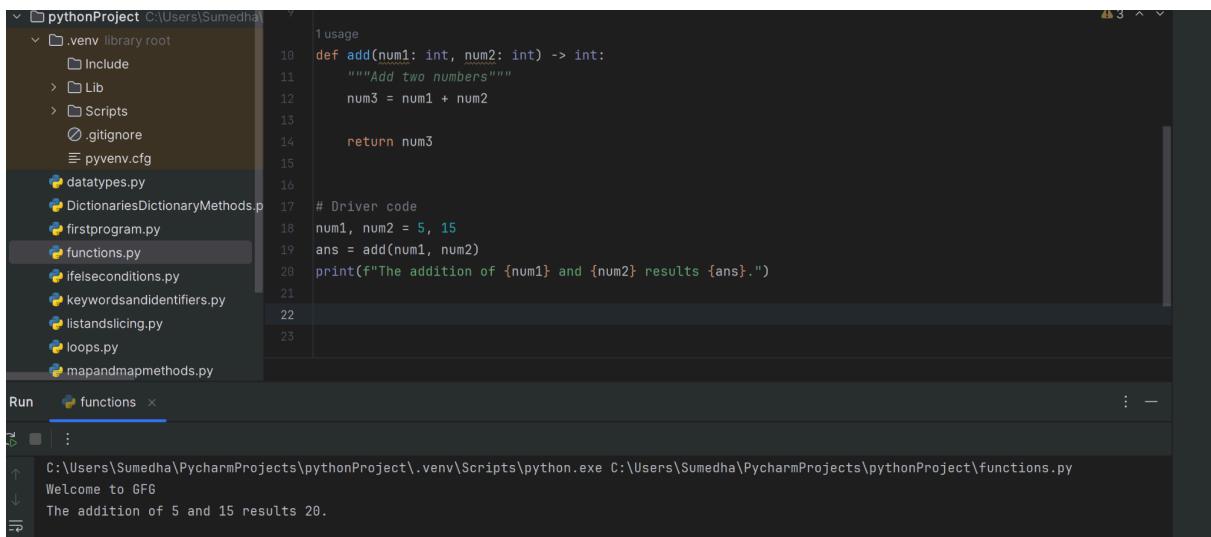


The screenshot shows a PyCharm interface with a project named 'pythonProject'. The 'functions.py' file is open in the editor. It contains the following code:

```
1 usage
2 def fun():
3     print("Welcome to GFG")
4
5
6 # Driver code to call a function
7 fun()
```

The 'Run' tab shows a successful execution of the script, outputting "Welcome to GFG".

→Defining and calling a function with parameters:



The screenshot shows a PyCharm interface with a project named 'pythonProject'. The 'functions.py' file is open in the editor. It contains the following code:

```
1 usage
2 def add(num1: int, num2: int) -> int:
3     """Add two numbers"""
4     num3 = num1 + num2
5
6     return num3
7
8 # Driver code
9 num1, num2 = 5, 15
10 ans = add(num1, num2)
11 print(f"The addition of {num1} and {num2} results {ans}.")
```

The 'Run' tab shows a successful execution of the script, outputting "The addition of 5 and 15 results 20."

→ **Python Function Arguments:** Arguments are the values passed inside the parentheses of the function. A function can have any number of arguments separated by a comma.

File Structure:

- Include
- Lib
- Scripts
- .gitignore
- pyenv.cfg
- datatype.py
- DictionariesDictionaryMethods.p
- firstprogram.py
- functions.py
- ifelseconditions.py
- keywordsandidentifiers.py
- listsandslicing.py
- loops.py
- mapandmapmethods.py

Code Editor (functions.py):

```
22 # A simple Python function to check
23 # whether x is even or odd
24 2 usages
25 def evenOdd(x):
26     if (x % 2 == 0):
27         print("even")
28     else:
29         print("odd")
30
31
32 # Driver code to call the function
33 evenOdd(2)
34 evenOdd(3)
```

Run Configuration: functions

Terminal Output:

```
C:\Users\Sumedha\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\pythonProject\functions.py
Welcome to GFG
The addition of 5 and 15 results 20.
even
odd

Process finished with exit code 0
```

→Types of Python Function Arguments

- **Default argument**
 - **Keyword arguments (named arguments)**
 - **Positional arguments**
 - **Arbitrary arguments** (variable-length arguments *args and **kwargs)

i) **DEFAULT ARGUMENT:** A Default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument.

The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure for 'pythonProject' containing files like .venv, arguments.py, and loops.py. The 'arguments.py' file is currently selected and open in the editor, showing Python code for a function 'myFun'. The right panel shows the run output for 'arguments.py', displaying the console output: 'x: 10' and 'y: 50', indicating the function was called with x=10 and y=50.

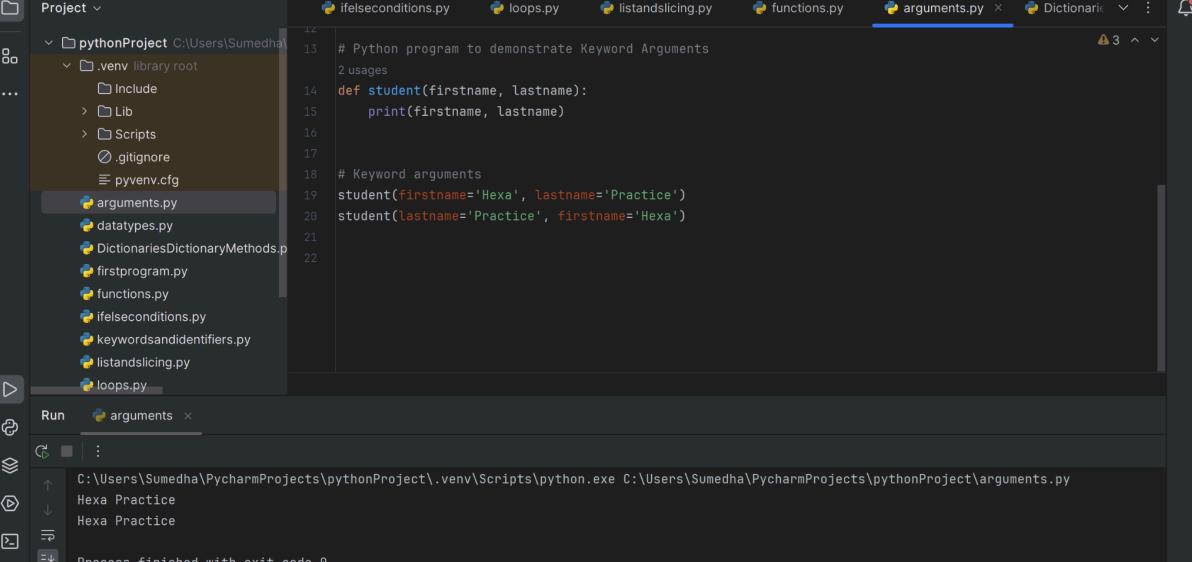
```
2 # default arguments
3 usage
4 def myFun(x, y=50):
5     print("x: ", x)
6     print("y: ", y)
7
8 # Driver code (We call myFun() with only
9 # argument)
10 myFun(10)
11
12
```

Run arguments x

```
C:\Users\Sumedha\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\pythonProject\arguments.py
x: 10
y: 50

Process finished with exit code 0
```

ii)Keyword arguments (named arguments):In Python, keyword arguments allow passing values to a function by explicitly associating them with parameter names. This provides clarity, flexibility, and allows for non-sequential argument passing.



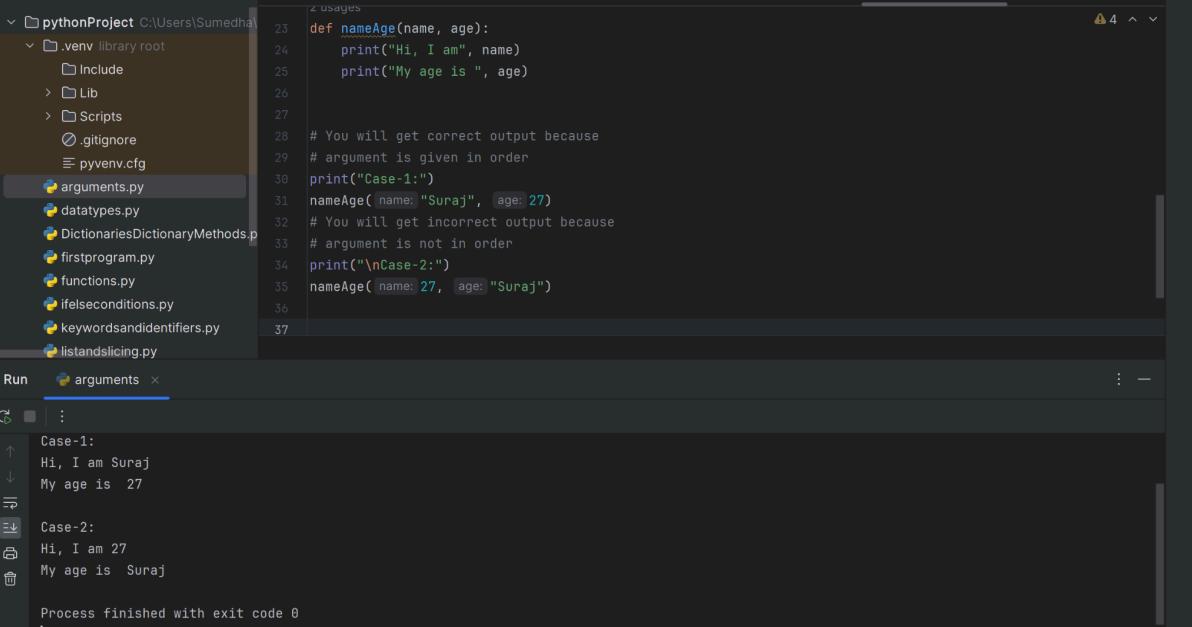
The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure for 'pythonProject' with files like ifelseconditions.py, loops.py, listandslicing.py, functions.py, DictionariesDictionaryMethods.py, firstprogram.py, datatypes.py, .gitignore, pyvenv.cfg, and arguments.py. The 'arguments.py' file is selected and open in the main editor. The code defines a 'student' function with keyword arguments:

```
# Python program to demonstrate Keyword Arguments
# 2 usages
def student(firstname, lastname):
    print(firstname, lastname)

# Keyword arguments
student(firstname='Hexa', lastname='Practice')
student(lastname='Practice', firstname='Hexa')
```

The 'Run' tab at the bottom shows the command: C:\Users\Sumedha\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\pythonProject\arguments.py. The run output shows two executions of the function with different argument orders, both resulting in 'Hexa Practice'. The status bar at the bottom right indicates 'Process finished with exit code 0'.

iii)Positional arguments:Positional arguments in Python are passed based on their order in the function signature. Values are assigned to parameters in the order they appear, making the argument order crucial. This is the traditional method of passing arguments.



The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure for 'pythonProject' with files like ifelseconditions.py, loops.py, listandslicing.py, functions.py, .gitignore, pyvenv.cfg, and arguments.py. The 'arguments.py' file is selected and open in the main editor. The code defines a 'nameAge' function:

```
2 usages
def nameAge(name, age):
    print("Hi, I am", name)
    print("My age is ", age)

# You will get correct output because
# argument is given in order
print("\nCase-1:")
nameAge( name: "Suraj", age: 27)
# You will get incorrect output because
# argument is not in order
print("\nCase-2:")
nameAge( name: 27, age: "Suraj")
```

The 'Run' tab at the bottom shows the command: C:\Users\Sumedha\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\pythonProject\arguments.py. The run output shows two cases: 'Case-1' where the arguments are in the correct order ('Hi, I am Suraj' and 'My age is 27'), and 'Case-2' where they are swapped ('Hi, I am 27' and 'My age is Suraj'). The status bar at the bottom right indicates 'Process finished with exit code 0'.

iv) Arbitrary Keyword Arguments: In Python Arbitrary Keyword Arguments , *args and **kwargs can pass a variable number of arguments to a function using special symbols. There are two special symbols:

- *args in Python (Non-Keyword Arguments)
- **kwargs in Python (Keyword Arguments)

The screenshot shows two code snippets in PyCharm's code editor and run terminal.

Top Snippet (Using *args):

```

36
37
38 # Python program to illustrate
39 # *args for variable number of arguments
40
41 def myFun(*argv):
42     for arg in argv:
43         print(arg)
44
45 myFun( "Hello", "Welcome", 'to', 'HexaforHexa')
46
47

```

Bottom Snippet (Using **kwargs):

```

51
52 usage
53 def myFun(**kwargs):
54     for key, value in kwargs.items():
55         print("%s == %s" % (key, value))
56
57 # Driver code
58 myFun(first='Hexa', mid='for', last='Hexa')
59
60
61

```

The run terminal shows the output for both snippets:

Output for Top Snippet:

```

Hello
Welcome
to
HexaforHexa
Process finished with exit code 0

```

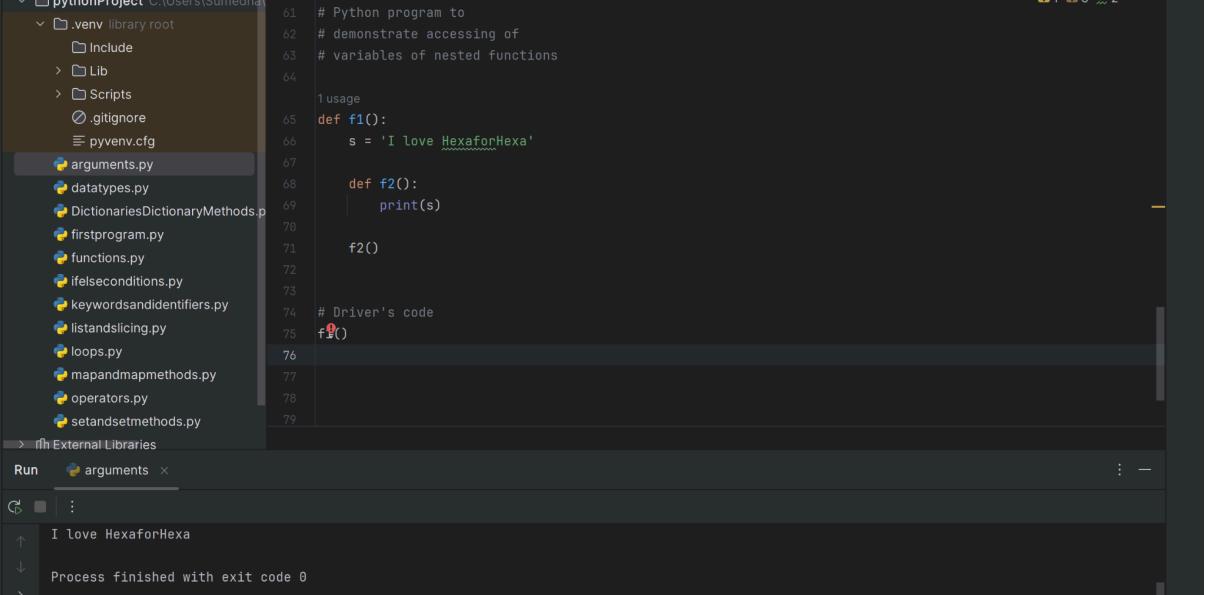
Output for Bottom Snippet:

```

first == Hexa
mid == for
last == Hexa
Process finished with exit code 0

```

→ **Function within Functions:** A function that is defined inside another function is known as the inner function or nested function.

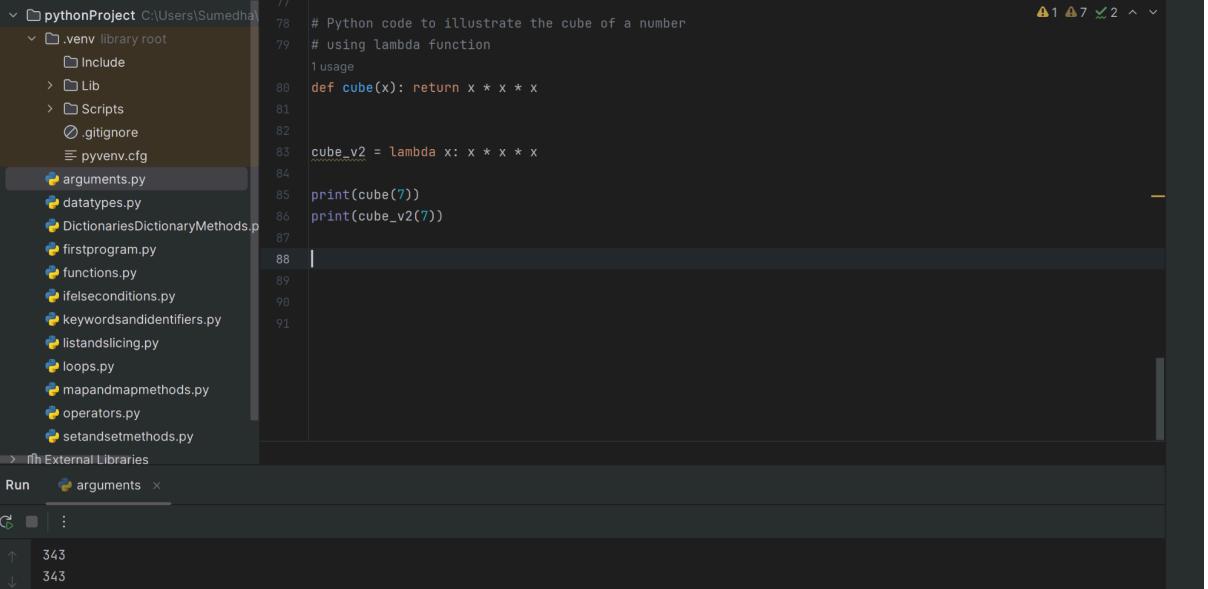


The screenshot shows a Python project structure in the left sidebar with files like .venv, arguments.py, datatypes.py, etc. The code editor displays the following nested function example:

```
61 # Python program to
62 # demonstrate accessing of
63 # variables of nested functions
64
65 usage:
66 def f1():
67     s = 'I love HexaforHexa'
68
69     def f2():
70         print(s)
71
72     f2()
73
74 # Driver's code
75 f1()
76
77
78
79
```

The output window at the bottom shows the result of running the code: "I love HexaforHexa".

→ **Anonymous Functions in Python:** A function without a name is called an anonymous function.



The screenshot shows a Python project structure in the left sidebar with files like .venv, arguments.py, datatypes.py, etc. The code editor displays the following anonymous function example:

```
77 # Python code to illustrate the cube of a number
78 # using lambda function
79
80 usage:
81
82
83 cube_v2 = lambda x: x * x * x
84
85 print(cube(7))
86 print(cube_v2(7))
87
88
89
90
91
```

The output window at the bottom shows the result of running the code: 343.

→ **Return Statement in Python Function:** The function return statement is used to exit from a function and go back to the function caller and return the specified value or data item to the caller.

```

Project: pythonProject C:\Users\Sumedha\...
pythonProject
  .venv library root
    Include
      Lib
      Scripts
    .gitignore
    pyenv.cfg
  arguments.py
  datatypes.py
  DictionariesDictionaryMethods.p
  firstprogram.py
  functions.py
  ifelseconditions.py
  keywordsandidentifiers.py
  listslicing.py
  loops.py
  mapandmapmethods.py
  operators.py
  setandsetmethods.py
  External Libraries
Run arguments
  4
  16

```

```

# Python code to illustrate the cube of a number
# using lambda function
usage
def cube(x): return x * x * x

cube_v2 = lambda x: x * x * x

print(cube(7))
print(cube_v2(7))

2 usages
def square_value(num):
    """This function returns the square
    value of the entered number"""
    return num ** 2

print(square_value(2))
print(square_value(-4))

```

→ **Pass by Reference and Pass by Value:** In Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created.

```

Project: pythonProject C:\Users\Sumedha\...
pythonProject
  .venv library root
    Include
      Lib
      Scripts
    .gitignore
    pyenv.cfg
  arguments.py
  datatypes.py
  DictionariesDictionaryMethods.p
  firstprogram.py
  functions.py
  ifelseconditions.py
  keywordsandidentifiers.py
  listslicing.py
  loops.py
  mapandmapmethods.py
  operators.py
  setandsetmethods.py
  External Libraries
Run arguments
  4
  16

```

```

# Here x is a new reference to same list lst
usage
def myFun(x):
    x[0] = 20

# Driver Code (Note that lst is modified
# after function call.
lst = [10, 11, 12, 13, 14, 15]
myFun(lst)
print(lst)

```

```

[20, 11, 12, 13, 14, 15]
Process finished with exit code 0

```

→ **Python map() Function:** map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)
Syntax: map(fun, iter)

```
# Python program to demonstrate working
# of map.

# Return double of n
usage
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(addition, numbers)
print(list(result))

Process finished with exit code 0
```

→map() with Lambda Expressions:

```
# Python program to demonstrate working
# of map.

# Return double of n
usage
def addition(n):
    return n + n

# We double all numbers using map()
numbers = (1, 2, 3, 4)
result = map(lambda x: x + x, numbers)
print(list(result))

# Double all numbers using map and lambda
numbers = (1, 2, 3, 4)
result = map(lambda x: x + x, numbers)
print(list(result))

Process finished with exit code 0
```

→String Functions in Python:

i) **Capitalize**: The capitalize() is used in Python where the first letter of the string is converted into UPPERCASE and the rest of the characters remain the same. On the other hand, if all the characters are in UPPERCASE then the string will return the same value (except the first character).

The screenshot shows a Python code editor interface. The left sidebar lists several Python files: arguments.py, datatypes.py, DictionariesDictionaryMethods.py, firstprogram.py, functions.py, ifelseconditions.py, keywordsandidentifiers.py, listslicing.py, loops.py, mapandmapmethods.py, operators.py, setandsetmethods.py, and stringfunctions.py. The file 'stringfunctions.py' is currently selected. The main code editor area contains the following code:

```
# input from users
sentence_1 = "mY name is SHARATH"
sentence_2 = "MY name is SRIJAN"

# Convert case using capitalize()
capitalized_string = sentence_1.capitalize()
print("Sentence 1 output -> ", capitalized_string)
capitalized_string = sentence_2.capitalize()
print("Sentence 2 output -> ", capitalized_string)
```

The run console at the bottom shows the output of the script:

```
Sentence 1 output -> My name is sharath
Sentence 2 output -> My name is srujan
Process finished with exit code 0
```

ii) **Count**: The count() throws the numeric value that provides the detail of an actual count of a given string.

The screenshot shows a Python code editor interface. The left sidebar lists several Python files: arguments.py, datatypes.py, DictionariesDictionaryMethods.py, firstprogram.py, functions.py, ifelseconditions.py, keywordsandidentifiers.py, listslicing.py, loops.py, mapandmapmethods.py, operators.py, setandsetmethods.py, and stringfunctions.py. The file 'stringfunctions.py' is currently selected. The main code editor area contains the following code:

```
message = 'PYTHON PROGRAMMING IS FUN'

# number of occurrence of 'O'
print('Number of occurrence of O:', message.count('O'))
```

The run console at the bottom shows the output of the script:

```
Number of occurrence of O: 2
Process finished with exit code 0
```

iii) Find: The `find()` is used in Python to return the lowest index value from the first occurrence of a string (only in case it's found): else the value would be -1.

The screenshot shows a PyCharm interface. On the left is a file tree with files like `ifelseconditions.py`, `keywordsandidentifiers.py`, etc., and a `stringfunctions.py` file which is selected. The main window contains the following Python code:

```
19
20 message = 'SHARATH is my name'
21
22 # check the index of 'is'
23 print(message.find('is'))
```

Below the code, the output window shows the result of the execution:

```
8
Process finished with exit code 0
```

iv) Lower: The `lower()` is used in Python programming to ensure that all the UPPERCASE characters in the string are converted into lowercase and fetched with a new lowercase string and the original copy of the string remains intact.

The screenshot shows a PyCharm interface. On the left is a file tree with files like `loops.py`, `mapandmapmethods.py`, etc., and a `stringfunctions.py` file which is selected. The main window contains the following Python code:

```
25 message = 'HEXAFORHEXA IS A COMPUTER SCIENCE PORTAL'
26
27 # convert message to lowercase
28 print(message.lower())
29
30
```

Below the code, the output window shows the result of the execution:

```
hexaforhexa is a computer science portal
Process finished with exit code 0
```

v) Upper: The `upper()` is used in Python programming to ensure that all the lowercase characters in the string are converted into UPPERCASE and fetched with a new string whereas the original copy of the string remains intact.

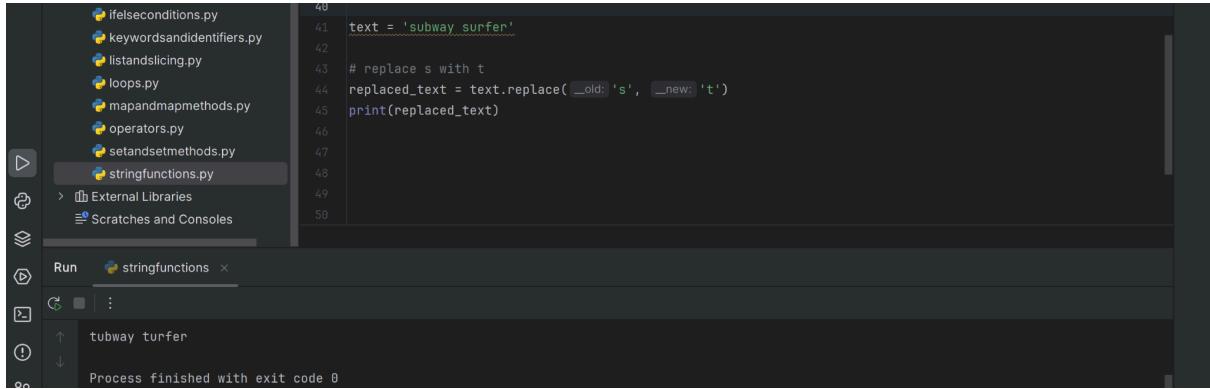
The screenshot shows a PyCharm interface. On the left is a file tree with files like `arguments.py`, `datatype.py`, etc., and a `stringfunctions.py` file which is selected. The main window contains the following Python code:

```
35
36
37 message = 'HexaforHexa is a computer science portal'
38
39 # convert message to uppercase
40 print(message.upper())
41
42
43
44
```

Below the code, the output window shows the result of the execution:

```
HEXAFORHEXA IS A COMPUTER SCIENCE PORTAL
Process finished with exit code 0
```

vi) Replace: The replace() is used in Python to replace any unwanted character or text and replace it with the new desired output within the string.



```
ifelseconditions.py
keywordsandidentifiers.py
listslicing.py
loops.py
mapandmapmethods.py
operators.py
setandsetmethods.py
stringfunctions.py
> External Libraries
Scatches and Consoles

Run stringfunctions x

tubway turfer
Process finished with exit code 0
```

The code in the editor is as follows:

```
40
41 text = 'subway surfer'
42
43 # replace s with t
44 replaced_text = text.replace( _old: 's', _new: 't')
45 print(replaced_text)
46
47
48
49
50
```

vii) JOIN(): join() returns a concatenated string and it will throw a TypeError exception if the iterable contains any non-string element within it.



```
loops.py
mapandmapmethods.py
operators.py
setandsetmethods.py
stringfunctions.py
> External Libraries
Scatches and Consoles

Run stringfunctions x

sharath is my only friend
Process finished with exit code 0
```

The code in the editor is as follows:

```
47 text = ['sharath', 'is', 'my', 'only', 'friend']
48
49 # join elements of text with space
50 print(' '.join(text))
51
52
53
```

2) OOPs Concepts in Python:

- Class
- Objects
- Polymorphism
- Encapsulation
- Inheritance
- Data Abstraction

i) Class: A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

Class Definition Syntax:

```
class ClassName:
```

```
    # Statement-1
```

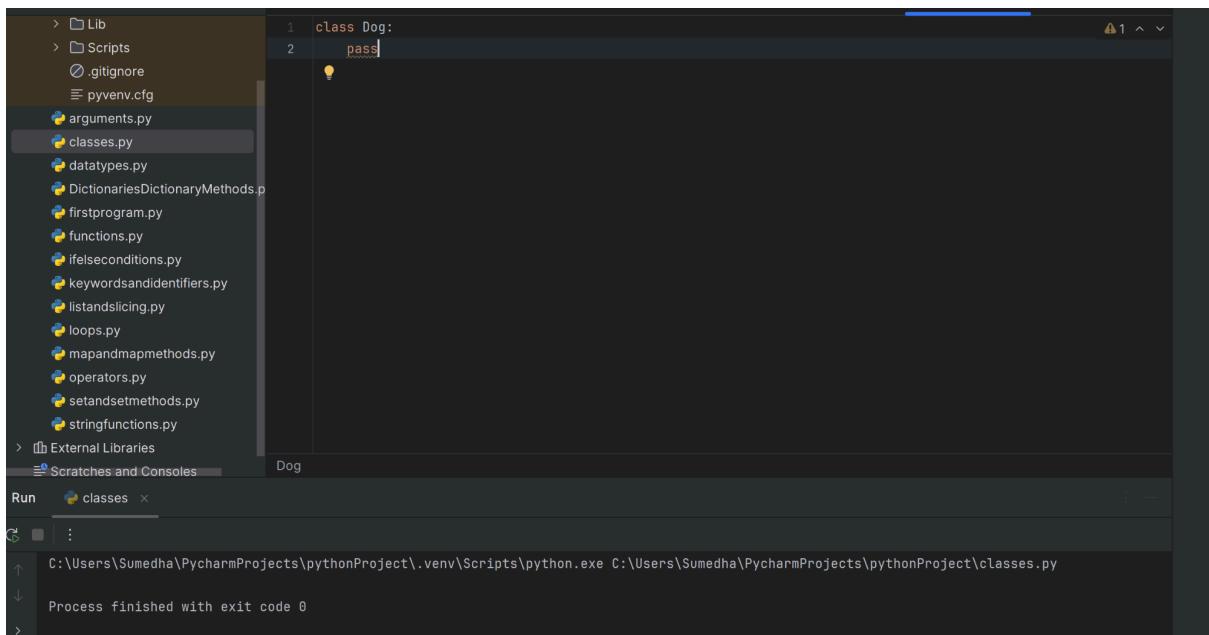
```
.
```

```
.
```

```
.
```

```
    # Statement-N
```

Creating an Empty Class in Python:



The screenshot shows the PyCharm IDE interface. On the left is the Project tool window displaying a file tree with various Python files like arguments.py, classes.py, datatypes.py, etc. In the center is the Code Editor with the following code:

```
1 class Dog:  
2     pass
```

On the right is the Terminal tool window showing the output of a run command:

```
C:\Users\Sumedha\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\pythonProject\classes.py  
Process finished with exit code 0
```

ii) Objects: The object is an entity that has a state and behaviour associated with it.

Creating a class and object with class and instance attributes:

The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure with files like Lib, Scripts, .gitignore, pyenv.cfg, arguments.py, classes.py, datatypes.py, DictionariesDictionaryMethods.p, firstprogram.py, functions.py, ifelseconditions.py, keywordsandidentifiers.py, listslicing.py, loops.py, mapandmapmethods.py, operators.py, setandsetmethods.py, and stringfunctions.py. The 'Run' tab is selected, showing the run configuration 'classes'. The main editor window contains Python code defining a 'Dog' class with a class attribute 'attr1' and an instance attribute 'name'. It includes driver code to create instances 'Rodger' and 'Tommy', and print statements to access their attributes. The terminal below shows the execution of the script and its output.

```
2 usages
1 class Dog:
2     # class attribute
3     attr1 = "mammal"
4
5     # Instance attribute
6     def __init__(self, name):
7         self.name = name
8
9     # Driver code
10    Rodger = Dog("Rodger")
11    Tommy = Dog("Tommy")
12
13    # Accessing class attributes
14    print("Rodger is a {}".format(Rodger.__class__.attr1))
15    print("Tommy is also a {}".format(Tommy.__class__.attr1))
16
17    # Accessing instance attributes
18    print("My name is {}".format(Rodger.name))
19    print("My name is {}".format(Tommy.name))
```

C:\Users\Sumedha\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\pythonProject\classes.py

Rodger is a mammal
Tommy is also a mammal
My name is Rodger
My name is Tommy

Creating Classes and objects with methods:

The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure with files like Lib, Scripts, .gitignore, pyenv.cfg, arguments.py, classes.py, datatypes.py, DictionariesDictionaryMethods.p, firstprogram.py, functions.py, ifelseconditions.py, keywordsandidentifiers.py, listslicing.py, loops.py, mapandmapmethods.py, operators.py, setandsetmethods.py, and stringfunctions.py. The 'Run' tab is selected, showing the run configuration 'classes'. The main editor window contains Python code defining a 'Dog' class with a class attribute 'attr1', an instance attribute 'name', and a method 'speak'. It includes driver code to create instances 'Rodger' and 'Tommy', and print statements to access their attributes. The terminal below shows the execution of the script and its output.

```
2 usages
19 class Dog:
20     # class attribute
21     attr1 = "mammal"
22
23     # Instance attribute
24     def __init__(self, name):
25         self.name = name
26
27     # Driver code
28     def speak(self):
29         print("My name is {}".format(self.name))
30
31 # Object instantiation
32 Rodger = Dog("Rodger")
33 Tommy = Dog("Tommy")
```

C:\Users\Sumedha\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\pythonProject\classes.py

My name is Rodger
My name is Tommy

iii) Inheritance: Inheritance allows a class (subclass/derived class) to inherit attributes and methods from another class (superclass/base class). It provides the reusability of a code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.

The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure with files like functions.py, arguments.py, stringfunctions.py, classes.py, and inheritance.py. The inheritance.py file is open in the main editor window, containing the following Python code:

```

# sample example of inheritance
1 usage
2 class Parent():
3     1 usage
4         def first(self):
5             print('first function')
6 usage
7 class Child(Parent):
8     1 usage
9         def second(self):
10            print('second function')
11 ob = Child()
12 ob.first()
13 ob.second()

```

The bottom right corner shows a warning icon with the number 6. Below the editor is a 'Run' tab with 'inheritance' selected. The run output window shows the following text:

```

C:\Users\Sumedha\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\pythonProject\inheritance.py
first function
second function
Process finished with exit code 0

```

Types of Inheritance:

- **Single Inheritance:** Single-level inheritance enables a derived class to inherit characteristics from a single-parent class.
- **Multilevel Inheritance:** Multi-level inheritance enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent class.
- **Hierarchical Inheritance:** Hierarchical-level inheritance enables more than one derived class to inherit properties from a parent class.
- **Multiple Inheritance:** Multiple-level inheritance enables one derived class to inherit properties from more than one base class.

i) single inheritance :When a child class inherits only a single parent class

The screenshot shows a PyCharm interface with a file tree on the left containing files like pyenv.cfg, arguments.py, classes.py, datatypes.py, etc. A file named inheritance.py is selected. The main editor window displays the following Python code:

```
#single inheritance
1 usage
2 @ class Parent:
3     1 usage
4         def func1(self):
5             print("this is function one")
6     1 usage
7 @ class Child(Parent):
8     1 usage
9         def func2(self):
10            print(" this is function 2 ")
11
12 ob = Child()
13 ob.func1()
14 ob.func2()
```

The status bar at the bottom indicates "Child > func2()". The run tab shows the output: "this is function one" and "this is function 2". The message "Process finished with exit code 0" is also present.

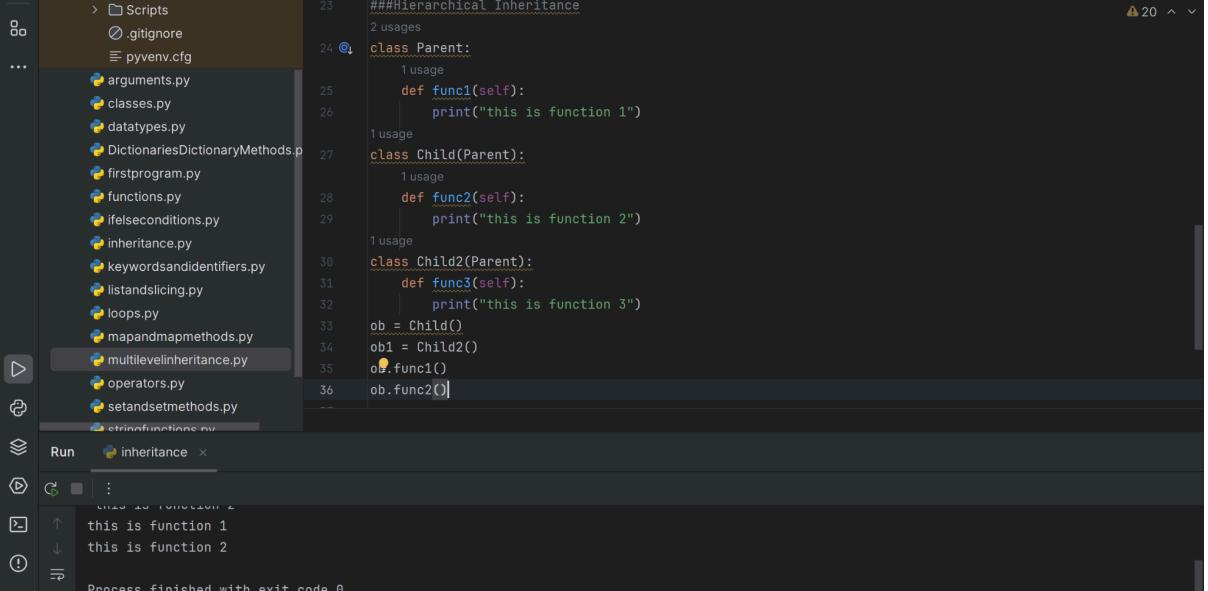
ii) Multilevel Inheritance: When a child class becomes a parent class for another child class.

The screenshot shows a PyCharm interface with a file tree on the left containing Scripts, .gitignore, and pyenv.cfg. A file named multilevelinheritance.py is selected. The main editor window displays the following Python code:

```
class Parent:
    1 usage
    2     def func1(self):
        3         print("this is function 1")
    4 usage
@ class Child(Parent):
    5 usage
    6     def func2(self):
        7         print("this is function 2")
    8 usage
@ class Child2(Child):
    9 usage
    10    def func3(self):
        11        print("this is function 3")
    12
    13 ob = Child2()
    14 ob.func1()
    15 ob.func2()
    16 ob.func3()
```

The status bar at the bottom indicates "Child2 > func3()". The run tab shows the output: "this is function 1", "this is function 2", and "this is function 3". The message "C:\Users\Sumedha\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\pythonProject\multilevelinheritance.py" is also present.

iii) Hierarchical Inheritance: Hierarchical inheritance involves multiple inheritance from the same base or parent class.

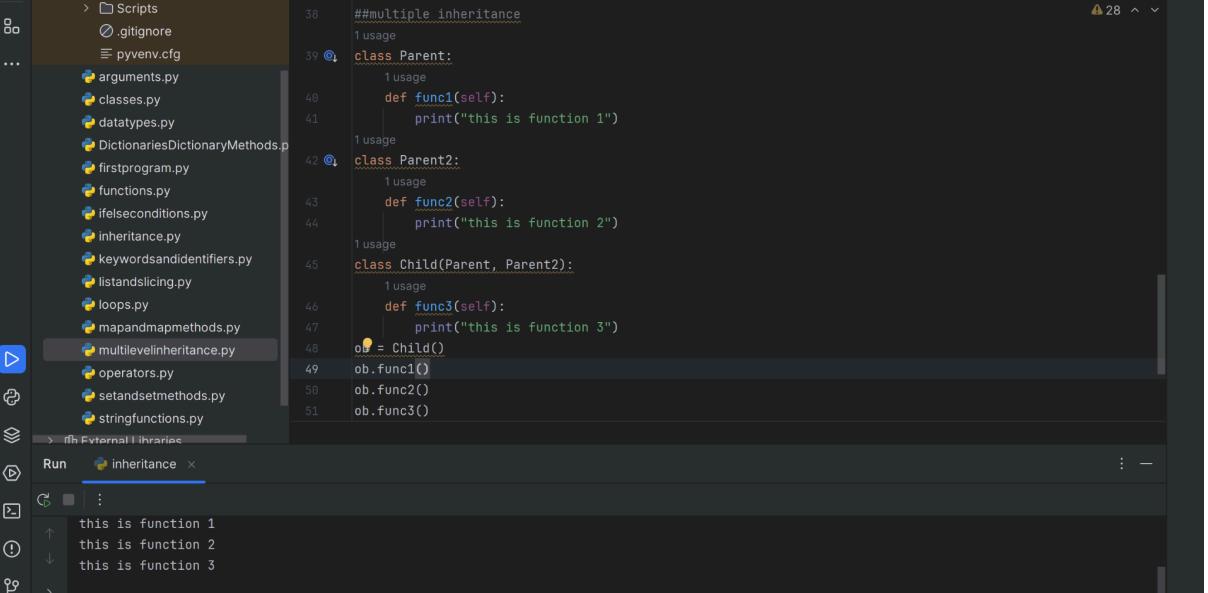


The screenshot shows a PyCharm interface with a file tree on the left containing Python scripts like arguments.py, classes.py, datatypes.py, etc. The current script is 'multilevelinheritance.py'. The code defines a Parent class with func1, a Child class that inherits from Parent and adds func2, and a Child2 class that inherits from Parent and adds func3. An object ob is created and calls func1, func2, and func3. The run output shows the expected results: 'this is function 1', 'this is function 2', and 'this is function 3' respectively.

```
23 #####Hierarchical Inheritance
24 @usage
25 class Parent:
26     1 usage
27         def func1(self):
28             print("this is function 1")
29
30 class Child(Parent):
31     1 usage
32         def func2(self):
33             print("this is function 2")
34
35 class Child2(Parent):
36     1 usage
37         def func3(self):
38             print("this is function 3")
39 ob = Child()
40 ob1 = Child2()
41 ob.func1()
42 ob.func2()
43 ob.func3()
```

Process finished with exit code 0

iv) Multiple Inheritance: When a child class inherits from more than one parent class.



The screenshot shows a PyCharm interface with a file tree on the left containing Python scripts like arguments.py, classes.py, datatypes.py, etc. The current script is 'multilevelinheritance.py'. The code defines a Parent class with func1, a Parent2 class with func2, and a Child class that inherits from both Parent and Parent2, adding func3. An object ob is created and calls func1, func2, and func3. The run output shows the expected results: 'this is function 1', 'this is function 2', and 'this is function 3' respectively.

```
38 #####multiple inheritance
39 @usage
40 class Parent:
41     1 usage
42         def func1(self):
43             print("this is function 1")
44
45 class Parent2:
46     1 usage
47         def func2(self):
48             print("this is function 2")
49
50 class Child(Parent, Parent2):
51     1 usage
52         def func3(self):
53             print("this is function 3")
54 ob = Child()
55 ob.func1()
56 ob.func2()
57 ob.func3()
```

Process finished with exit code 0

iv) Polymorphism: Polymorphism means taking different forms. In programming, it enables operators, functions, and methods to act differently when subjected to different conditions.

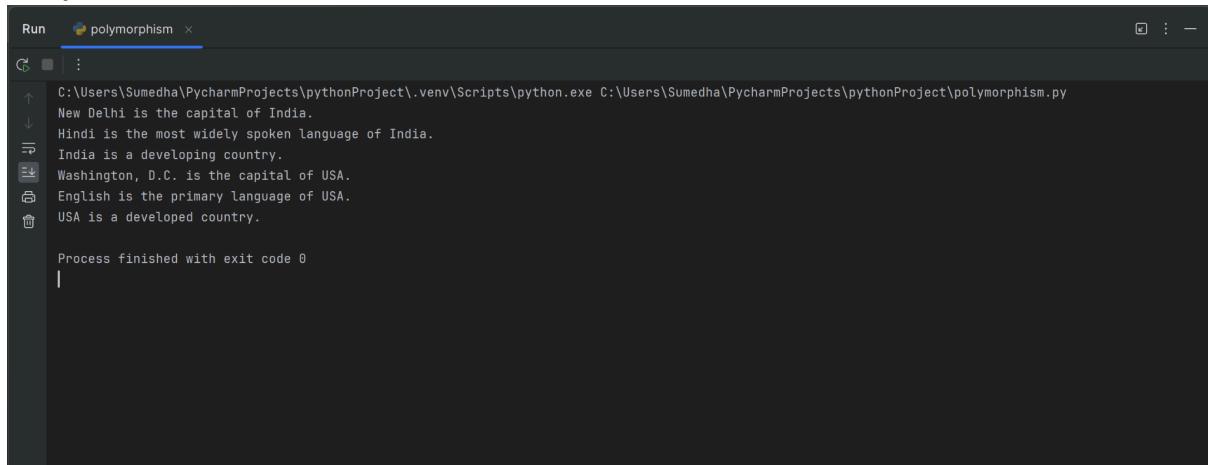
The screenshot shows a code editor interface with a dark theme. On the left is a project sidebar titled "pythonProject" containing files like "functions.py", "arguments.py", "stringfunctions.py", "classes.py", "inheritance.py", and "polymorphism.py". The main pane displays the following Python code:

```
1 class India():
2     1 usage (1 dynamic)
3         def capital(self):
4             print("New Delhi is the capital of India.")
5             1 usage (1 dynamic)
6             def language(self):
7                 print("Hindi is the most widely spoken language of India.")
8                 1 usage (1 dynamic)
9                     def type(self):
10                         print("India is a developing country.")
11
12 class USA():
13     1 usage (1 dynamic)
14         def capital(self):
15             print("Washington, D.C. is the capital of USA.")
16             1 usage (1 dynamic)
17             def language(self):
18                 print("English is the primary language of USA.")
19                 1 usage (1 dynamic)
20                     def type(self):
21                         print("USA is a developed country.")
22
23 def func(obj):
24     obj.capital()
25     obj.language()
26     obj.type()
27
28 obj_ind = India()
29 obj_usa = USA()
30
31 func(obj_ind)
32 func(obj_usa)
```

The status bar at the bottom indicates "Python 3.10 (pythonProject)".

This screenshot is identical to the one above, showing the same Python code for demonstrating polymorphism. The code defines two classes, India and USA, each with a capital, language, and type method. It then defines a func function that calls these methods on objects of both classes. The status bar at the bottom indicates "Python 3.10 (pythonProject)".

Output:

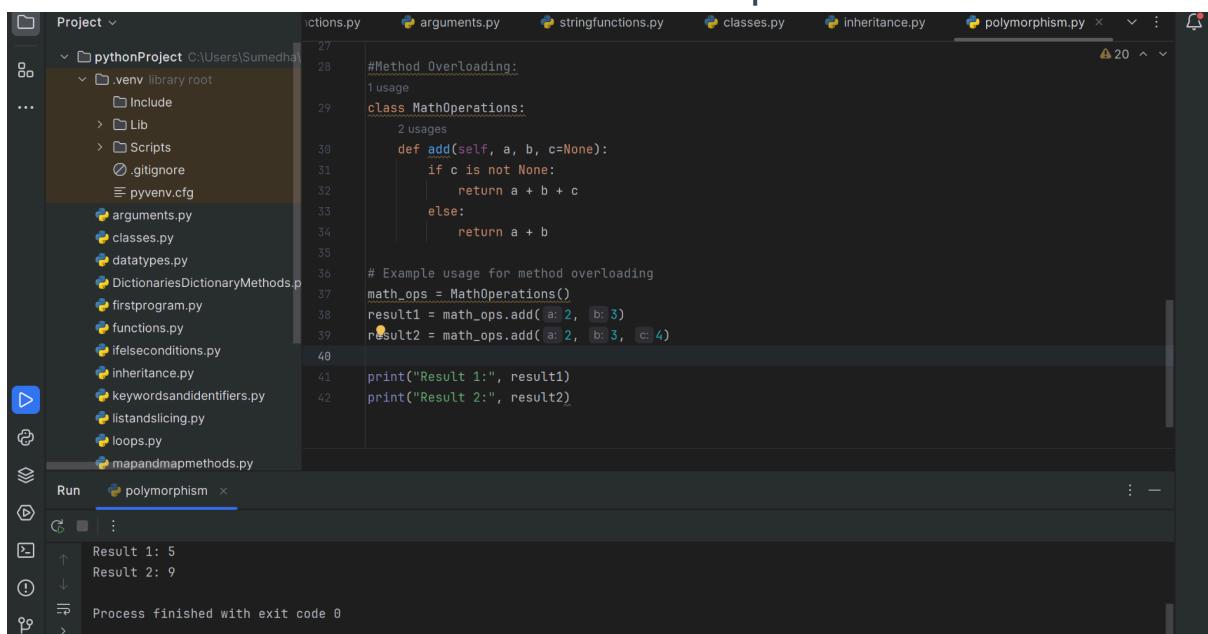


```
C:\Users\Sumedha\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\pythonProject\polymorphism.py
↑
↓
New Delhi is the capital of India.
Hindi is the most widely spoken language of India.
India is a developing country.
Washington, D.C. is the capital of USA.
English is the primary language of USA.
USA is a developed country.

Process finished with exit code 0
```

→METHOD OVERLOADING AND METHOD OVERRIDING:

i)METHOD OVERLOADING: Method overloading in Python refers to the ability to define multiple methods in the same class with the same name but different parameters.



```
#Method OverLoading:
1 usage
class MathOperations:
    2 usages
        def add(self, a, b, c=None):
            if c is not None:
                return a + b + c
            else:
                return a + b
# Example usage for method overloading
math_ops = MathOperations()
result1 = math_ops.add( a: 2, b: 3)
result2 = math_ops.add( a: 2, b: 3, c: 4)
print("Result 1:", result1)
print("Result 2:", result2)
```

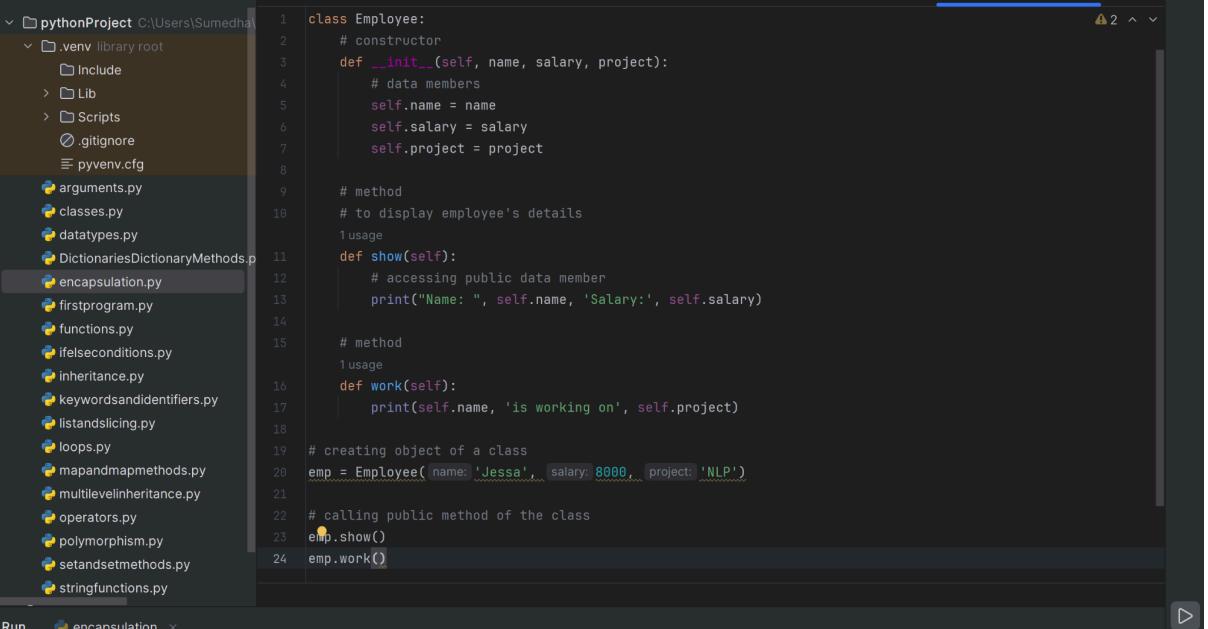
ii) METHOD OVERRIDING: Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass. This allows a subclass to customise or extend the behaviour of the inherited method.

```
45 # Method Overriding:
46
47     class Animal:
48         def speak(self):
49             raise NotImplementedError("Subclass must implement this method")
50
51     class Dog(Animal):
52         def speak(self):
53             return "Woof!"
54
55     class Cat(Animal):
56         def speak(self):
57             return "Meow!"
58
59     # Create a list of Animal objects
60     animals = [Dog(), Cat()]
61
62     for animal in animals:
63         print(animal.speak())
```

Run polymorphism

Woof!
Meow!

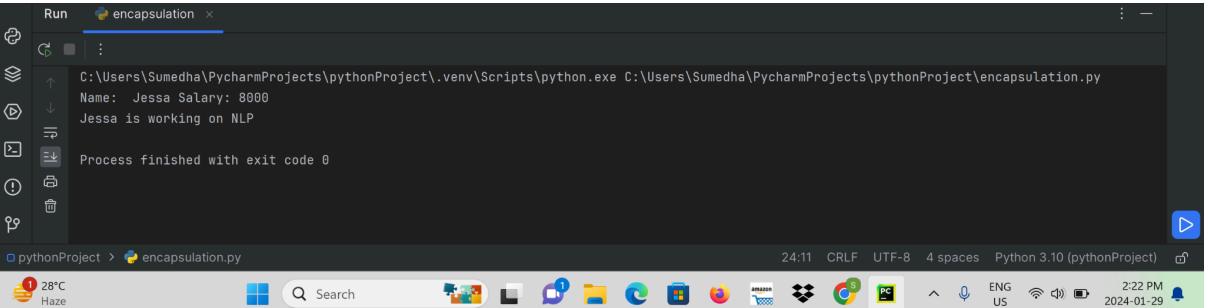
V) ENCAPSULATION: It describes the idea of wrapping data and the methods that work on data within one unit.



The screenshot shows the PyCharm IDE interface. On the left, the project structure is displayed under 'pythonProject C:\Users\Sumedha'. It includes a '.venv' library root folder containing 'Include', 'Lib', 'Scripts', '.gitignore', and 'pyvenv.cfg'. The 'encapsulation.py' file is selected in the list of files, which also includes 'arguments.py', 'classes.py', 'datatypes.py', 'DictionariesDictionaryMethods.py', 'firstprogram.py', 'functions.py', 'ifelseconditions.py', 'inheritance.py', 'keywordsandidentifiers.py', 'listslicing.py', 'loops.py', 'mapandmapmethods.py', 'multilevelinheritance.py', 'operators.py', 'polymorphism.py', 'setandsetmethods.py', and 'stringfunctions.py'. The main editor window contains the following Python code:

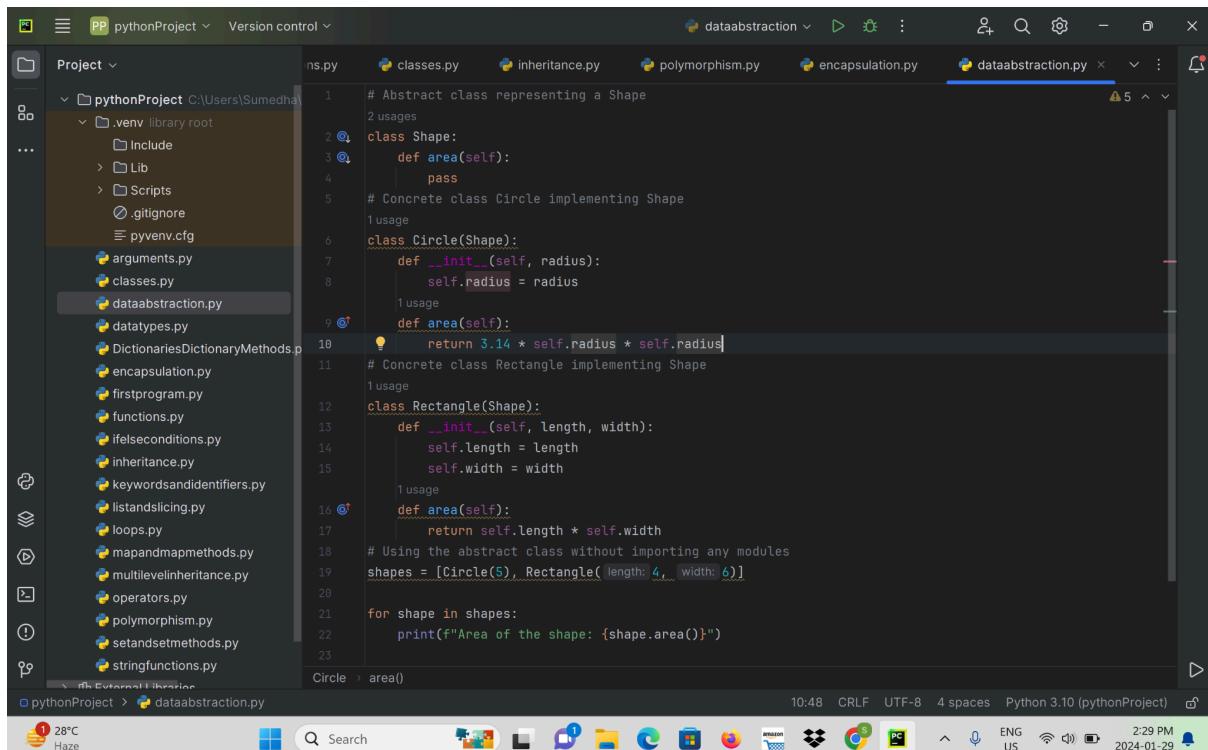
```
1 class Employee:
2     # constructor
3     def __init__(self, name, salary, project):
4         # data members
5         self.name = name
6         self.salary = salary
7         self.project = project
8
9     # method
10    # to display employee's details
11    1 usage
12    def show(self):
13        # accessing public data member
14        print("Name: ", self.name, 'Salary:', self.salary)
15
16    # method
17    1 usage
18    def work(self):
19        print(self.name, 'is working on', self.project)
20
21 # creating object of a class
22 emp = Employee( name='Jessa', salary=8000, project='NLP')
23
24 # calling public method of the class
25 emp.show()
26 emp.work()
```

OUTPUT:



The screenshot shows the PyCharm 'Run' tab. The 'encapsulation' run configuration is selected. The output pane displays the following text:
C:\Users\Sumedha\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\pythonProject\encapsulation.py
Name: Jessa Salary: 8000
Jessa is working on NLP
Process finished with exit code 0

vi) DATA ABSTRACTION: It focuses on hiding the complex implementation details and showing only the necessary features of an object. In Python, data abstraction is often achieved through the use of abstract classes and abstract methods.



The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure for 'pythonProject' containing files like 'classes.py', 'dataabstraction.py', and 'encapsulation.py'. The main editor window shows the following Python code:

```
# Abstract class representing a Shape
class Shape:
    def area(self):
        pass

# Concrete class Circle implementing Shape
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

# Concrete class Rectangle implementing Shape
class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

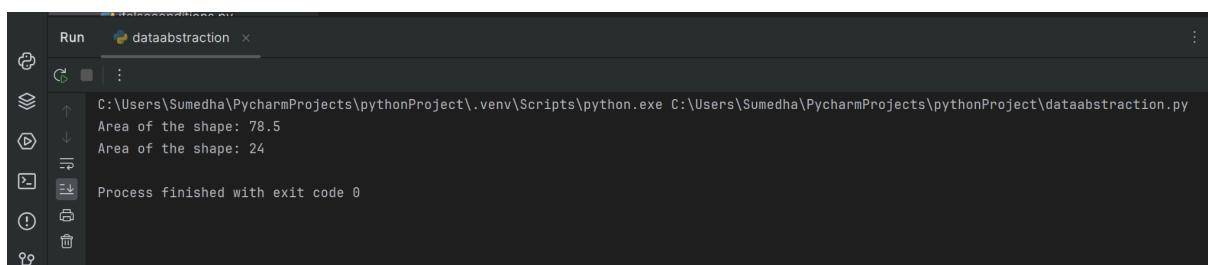
    def area(self):
        return self.length * self.width

# Using the abstract class without importing any modules
shapes = [Circle(5), Rectangle( length: 4, width: 6)]

for shape in shapes:
    print(f"Area of the shape: {shape.area()}")
```

The status bar at the bottom indicates the file is 'dataabstraction.py', the time is '10:48', and the Python version is 'Python 3.10 (pythonProject)'.

OUTPUT:



The screenshot shows the PyCharm 'Run' tab. The run configuration is set to 'dataabstraction'. The output window displays the following text:

```
C:\Users\Sumedha\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\pythonProject\dataabstraction.py
Area of the shape: 78.5
Area of the shape: 24
Process finished with exit code 0
```

