

OOPS-Assignment---2

1) Class and Object

Class:

- A class is a user-defined blueprint or prototype from which objects are created.
- Classes provide a means of bundling data and functionality together.
- Creating a new class creates a new type of object, allowing new instances of that type to be made.
- Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state.

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and a tab for 'Polymorphism.py - Class and obj.py'. The left sidebar has a 'Project' view showing a folder 'DEPython' containing numerous Python files like 'Arbitrary args.py', 'Break.py', etc., with 'Class and obj.py' selected. The main code editor window displays the following Python code:

```
# Creating a class and obj
class My_Class:
    def __init__(self):
        self.x=0
        self.y=0
    def setData(self):
        self.x=1000
        self.y=2000
        self.z=3000
    def getData(self):
        print(f'Value of x {self.x} \nValue of y {self.y} \nValue of z {self.z}')
obj=My_Class()
obj.setData()
obj.getData()
```

The 'Run' tool window at the bottom shows the output of running the script:

```
C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/Class and obj.py"
Value of x 1000
Value of y 2000
Value of z 3000
Process finished with exit code 0
```

The status bar at the bottom right indicates the time as 12:15 and Python version as 3.9.

Object:

- An Object is an instance of a Class.
- An object consists of:
 1. **State:** It is represented by the attributes of an object. It also reflects the properties of an object.
 2. **Behavior:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
 3. **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

2) Constructor

- `__init__` method is a constructor in python
- It is a special member of class.
- It is used to initialize data member of class.
- It is automatically called when an object is created.
- It contains instructions that are executed at the time of object creation.
- There are two types of constructors.

Default constructor

Parameterized constructor

i) Default Constructor:

Default Constructor definition has only one argument which is reference to instance being constructed.

ii) Parameterized Constructor:

It is a constructor with parameters. The first argument is reference to the instance being constructed known as self and the remaining are provided by programmer.

The screenshot shows the PyCharm IDE interface with the following details:

- File Menu:** File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help.
- Toolbar:** Standard toolbar with icons for Open, Save, Run, Stop, etc.
- Project Explorer:** Shows a project named "DEPython" containing files like "Arbitrary args.py", "break.py", "Calculator.py", "Class and obj.py", "Constructor.py", "continue.py", "Datatypes.py", "Date and time function.py", "Default arguments.py", "Dictionary and dict methods.py", "Example-1.csv", "extra.py", "First program.py", "for loop.py", "functions.py", "Identifiers.py", "if else if else.py", "if else statement.py", "if statement.py", "Inheritance.py", and "Input and output.py".
- Code Editor:** The active file is "Constructor.py" with the following code:

```
# Default constructor
class My_Class:
    def __init__(self):
        print("Default constructor")
        print('Hello')
obj1=My_Class()

# Parameterized Constructor
class My_Class:
    def __init__(self,a,b):
        print("Parameterized Constructor")
        self.a=a
        self.b=b
        print(self.a)
        print(self.b)
obj=My_Class(100,200)
```
- Run Tab:** Shows the command: C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/Constructor.py"
- Output Tab:** Displays the run results:

```
Default constructor
Hello
Parameterized Constructor
100
200

Process finished with exit code 0
```
- Bottom Bar:** Shows the Python version (Python 3.9), system status (ENG IN), and date/time (29-01-2024 23:52).

3) Access specifiers

- Access specifiers are used to restrict the access of the class member variable and methods from outside the class.
- Encapsulation is an OOPs principle which protects the internal data of the class using Access modifiers like Public, Private and Protected.

- Python supports three types of access modifiers which are public, private and protected.
- These access modifiers provide restrictions on the access of member variables and methods of the class from any object outside the class.

i) Public:

By default the member variables and methods are public which means they can be accessed from anywhere outside or inside the class.

ii) Private:

- Class properties and methods with private access modifier can only be accessed within the class where they are defined and cannot be accessed outside the class.
- In Python private properties and methods are declared by adding a prefix with two underscores('__') before their declaration.

iii) Protected:

- Class properties and methods with protected access modifier can be accessed within the class and from the class that inherits the protected class.
- Protected members and methods are declared using single underscore('_') as prefix before their names.

```

#Public access modifier
class Student:
    def __init__(self, name, age):
        self.name=name
        self.age=age
    def display(self):
        print(f'Name:{self.name}')
        print(f'Age:{self.age}')
s=Student('Ashutosh',23)
s.display()

```

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and Polymorphism.py - Access Specifiers.py. The Project tool window on the left lists various Python files in the DEPython project. The main code editor window contains the following code:

```
# Private Access modifier
class BankAccount:
    def __init__(self, accno, balance):
        self.__accno = accno
        self.__balance = balance
    def __display(self):
        print("Account Number: " + self.__accno)
        print("Balance: " + str(self.__balance))

b = BankAccount(123345666, 560000.90)
b.__display()
```

The Run tool window at the bottom shows the output of running the script:

```
C:\Users\DELL\OneDrive\Desktop\Data Engineering\DEPython> python Access Specifiers.py
Traceback (most recent call last):
  File "C:\Users\DELL\OneDrive\Desktop\Data Engineering\DEPython\Access Specifiers.py", line 25, in <module>
    b.__display()
AttributeError: 'BankAccount' object has no attribute '__display'
```

The status bar at the bottom right indicates the system is in English (ENG), connected to the internet (IN), and the date is 30-01-2024.

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and Polymorphism.py - Access Specifiers.py. The Project tool window on the left lists various Python files in the DEPython project. The main code editor window contains the following code:

```
# Protected Access Specifier
# Parent class
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def display(self):
        print("Name: " + self.name)
        print("Age: " + str(self.age))

# Child class
class Student(Person):
    def __init__(self, name, age, rollno):
        super().__init__(name, age)
        self._rollno = rollno
    def display(self):
        self._display()
        print("Rollno: " + str(self._rollno))

s = Student('Sona', 22, 78)
s.display()
```

The Run tool window at the bottom shows the output of running the script:

```
Name: Sona
Age: 22
Name: Sona
Age: 22
Rollno: 78

Process finished with exit code 0
```

The status bar at the bottom right indicates the system is in English (ENG), connected to the internet (IN), and the date is 30-01-2024.

4) Polymorphism

- Polymorphism refers to having multiple forms.
- Polymorphism is a programming term that refers to the use of the same function name, but with different signatures, for multiple types.

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** DEPython
- File:** Polymorphism.py
- Code Content:**

```
#Polymorphism
class Person:# parent class
    def __init__(self):
        self.height=157
        self.weight=60
        self.comp='fair'
    def display(self):
        print(self.height,self.weight,self.comp)
class Student(Person):# child class
    def __init__(self):
        self.name='Mona'
        self.course='BTech'
    def display(self):
        print(self.name,self.course)
obj=Student()
obj.display() # overriding takes place and displays child class details
Student().__init__()
```

- Run Output:** C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/Polymorphism.py"
- Output:** Mona BTech
Mona BTech
157 60 fair
- Message Bar:** PyCharm 2021.1.3 available Update...
- System Tray:** 20°, Search, Taskbar icons, Network, Battery, ENG IN, 1039, 30-01-2024

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** DEPython
- File:** Polymorphism.py
- Code Content:**

```
# To print parent class details use super/calls the base class constructor
class Person:# parent class
    def __init__(self):
        self.height=157
        self.weight=60
        self.comp='fair'
    def display(self):
        print(self.height,self.weight,self.comp)
class Student(Person):# child class
    def __init__(self):
        self.name='Mona'
        self.course='BTech'
        super().__init__()
    def display(self):
        print(self.name,self.course)
        super().display()
obj=Student()
obj.display()
```

- Run Output:** C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/Polymorphism.py"
- Output:** Mona BTech
Mona BTech
157 60 fair
- Message Bar:** PyCharm 2021.1.3 available Update...
- System Tray:** 20°, Search, Taskbar icons, Network, Battery, ENG IN, 1010, 30-01-2024

5) Method Overriding

For implementing it you have to use inheritance.

In method overriding you can reimplement the methods of parent class inside child class wherein the methods of child class and parent class have the similarity in the following features:

same name, parameters or signatures and same return type.

The screenshot shows the PyCharm Professional Edition interface. The project navigation bar at the top lists several Python files. The main code editor window displays a file named 'Method overriding.py' with the following content:

```
#Parent table
class Bird:
    def __init__(self, name):
        self.name = name

    def print_info(self):
        print(f'This bird is:{self.name}')

    def fly(self):
        print('The bird can fly')

#child table
Shalik : __init__
```

The 'Run' tab in the bottom left shows the command: C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data_Engineering_Python/DEPython/Method overriding.py". The output pane shows the execution results:

```
This bird is:Shalik
Color of bird:black
Character of bird:good
It can fly high

Process finished with exit code 0
```

A status bar at the bottom right indicates: 8:1 CRLF - UTF-8 4 spaces Python 3.9 ENG IN 12:32 30-01-2024.

The screenshot shows the PyCharm Professional Edition interface. The project navigation bar at the top lists several Python files. The main code editor window displays a file named 'Method overriding.py' with the following content:

```
#child table
class Shalik(Bird):
    def __init__(self, name, color, character):
        #call the parent class method
        super().__init__(name)
        self.color = color
        self.character = character

    # override method
    def print_info(self):
        #call the method of parent class
        super().print_info()
        print(f'Color of bird:{self.color}')
        print(f'Character of bird:{self.character}')

    # override method
    def fly(self):
        print('It can fly high')

obj_Shalik=Shalik('Shalik', 'black', 'good')
obj_Shalik.print_info()
obj_Shalik.fly()

Shalik : __init__
```

The 'Run' tab in the bottom left shows the command: C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data_Engineering_Python/DEPython/Method overriding.py". The output pane shows the execution results:

```
This bird is:Shalik
Color of bird:black
Character of bird:good
It can fly high

Process finished with exit code 0
```

A status bar at the bottom right indicates: 8:1 CRLF - UTF-8 4 spaces Python 3.9 ENG IN 12:33 30-01-2024.

6) Inheritance

- Inheritance is the capability of one class to derive or inherit the properties from another class.
- The class that derives properties is called the **derived class or child class** and the class from which the properties are being derived is called the **base class or parent class**.

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.
- **Hybrid Inheritance:** It is a combination of one or more inheritances.

1) Single Inheritance

The screenshot shows the PyCharm Professional Edition interface. The project navigation bar at the top lists several Python files. The main editor window displays the code for 'Inheritance.py'. The code defines a 'Father' class with a 'quality' method and a 'Son' class that inherits from 'Father' and overrides the 'quality' method with its own implementation. A comment at the bottom of the code indicates the creation of a 'son' object and its call to the 'quality' method. The 'Run' tool window at the bottom shows the execution of the script and its output: 'Inside father class', 'father has intelligence and deep thinking power.', 'Inside Son class', and 'Son wants to be Doctor'. A status bar at the bottom right provides system information like date and time.

```
# 1) Single Inheritance
# parent class
class Father:
    def quality(self):
        print('Inside father class')
        print('father has intelligence and deep thinking power.')
        print('\n')

# child class
class Son(Father):
    def aim(self):
        print('Inside Son class')
        print('Son wants to be Doctor')
        print('\n')

#creating object for son class
s=Son()
s.quality()
s.aim()
```

2) Multi level Inheritance

The screenshot shows the PyCharm Professional Edition interface with the file 'Inheritance.py' open. The code defines three classes: Grandfather, Father (which inherits from Grandfather), and Son (which inherits from Father). Each class has a method named 'quality' that prints specific messages. A variable 's1' is created for the Son class, and its 'quality' method is called, which in turn calls the 'quality' methods of the父类 (parent classes).

```
# 2) Multi level Inheritance
class Grandfather:
    def gf_quality(self):
        print('Inside Grand father class')
        print('Grand father is honest person')
class Father(Grandfather):
    def quality(self):
        print('Inside father class')
        print('father has intelligence and deep thinking power.')
class Son(Father):
    def aim(self):
        print('Inside Son class')
        print('Son wants to be Doctor')
# creating object for Son class
s1=Son()
s1.gf_quality()
s1.quality()
s1.aim()
```

3) Multiple Inheritance

The screenshot shows the PyCharm Professional Edition interface with the file 'Inheritance.py' open. The code defines four classes: Father, Mother, and Son, along with a child class Son(Father, Mother). The Father class has methods 'father_quality' and 'father_nature'. The Mother class has methods 'mother_quality' and 'mother_nature'. The Son class has a method 'son_quality'. When an object 's2' of the Son class is created and its 'son_quality' method is called, it also calls the 'father_quality' and 'mother_quality' methods of its parent classes.

```
Multiple Inheritance
class Father:
    def father_quality(self):
        print('Inside father class')
        print('father has intelligence and deep thinking power.')
    def father_nature(self):
        print('Inside father class')
        print('Father is very strict')
# parent class-2
class Mother:
    def mother_quality(self):
        print('Inside mother class')
        print('mother cooks good.')
    def mother_nature(self):
        print('Inside mother class')
        print('Mother is Kind hearted')
#child class
class Son(Father, Mother):
    def son_quality(self):
        print('Inside Son class')
        print('Son is very naughty')
s2=Son()
s2.son_quality()
s2.father_quality()
s2.father_nature()
s2.mother_quality()
s2.mother_nature()
s2.son_quality()
```

The screenshot shows a PyCharm Professional Edition interface with the following details:

- File Menu:** file edit View Navigate Code Refactor Run tools VCS Window Help
- Toolbar:** Polymorphism.py Inheritance.py
- Project Tree:** DEPython Inheritance Project Examples.py first program.py for loops.py functions.py lists & tuples.py if elif else.py if else statement.py imports.py Inheritance.py Input and output.py Key word args.py Lambda function.py List methods and slicing.py Map & Map methods.py Map methods function.py
- Code Editor:** Class and object x Constructors x Polymorphism.py x Method overriding.py x Inheritance.py x simple pandas pgm.py x Read data from CSV file.py
- Code Content:** Python code demonstrating inheritance and method overriding. It includes classes Father, Mother, and Son, each with their own methods and nested classes s2Son and s2Mother.
- Run Tab:** Shows the execution results:
 - Mother class
 - Inside father class
 - Father has intelligence and deep thinking power.
 - Inside father class
 - Father is very strict
 - Inside mother class
 - Mother cooks good
 - Inside mother class
 - Mother is kind hearted
 - Inside Son class
 - Son is very naughty
- Status Bar:** You are using Jupyter notebook PyCharm Professional Edition has special support for it.
- System Tray:** Shows icons for battery, signal strength, volume, and system status.

4) Hierarchical Inheritance

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help Polymorphism.py - Inheritance.py
DEPython Inheritance.py
Project Example_1.csv Constructor.py Polymorphism.py Method overriding.py Inheritance.py simple pandas pgm.py Read data from CSV file.py
extra.py
First program.py
For loop.py
Function.py
Identifiers.py
if elif else.py
if else statement.py
if statement.py
Inheritance.py
Input and output.py
Key word args.py
Keywords.py
Lambda function.py
List methods & slicing.py
Map & Map methods.py
Mapping functions.py
Method overriding.py
Module Calculator.py
Nested loop.py
Number function.py
Operations.py
Pass.py
Polymorphism.py
Read data from file.py
Set & Set methods.py
Simple pandas pgm.py
String parameters.py
String Function.py
Topic.py
Topics.py
Variables.py
While loop.py

68     son_quality()
69
70     # Hierarchical Inheritance
71     parent_class
72     print('Hierarchical Inheritance')
73     class Father:
74         def father_quality(self):
75             print('Inside father class')
76             print('father has intelligence and deep thinking power.')
77     #child class-1
78     class Son1(Father):
79         def son1_quality(self):
80             print('Inside Son 1 class')
81             print('Son is very naughty')
82
83     son1=Son1()
84     son1.father_quality()
85     son1.son1_quality()
86
87     #child class-2
88     class Son2(Father):
89         def son2_quality(self):
90             print('Inside Son 2 class')
91             print('Son is very disciplined')
92
93     son2=Son2()
94     son2.father_quality()
95     son2.son2_quality()
96
97 Father.father_quality()
```

Output:

The screenshot shows the PyCharm Professional Edition interface with the following details:

- File Menu:** File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help.
- Project Bar:** DEPython, Inheritance.py
- Toolbars:** Standard, Project, Editor, Run, Structure, Find, Help.
- Editor:** The code editor displays a Python script named `Inheritance.py`. It contains two classes, `Father` and `Son1`, and `Son2`. The `Father` class has a method `father_quality()` which prints "Inside father class" and "father has intelligence and deep thinking power". The `Son1` class inherits from `Father` and adds its own method `son1_quality()` which prints "Inside Son 1 class" and "Son is very naughty". The `Son2` class also inherits from `Father` and adds its own method `son2_quality()` which prints "Inside Son 2 class" and "Son is very disciplined". A variable `son1=Son1()` is created and its `father_quality()` method is called, followed by `son1.son1_quality()`. Another call to `father_quality()` is shown at the bottom.
- Run Tab:** Shows the output of the script:

```
Inside father class
father has intelligence and deep thinking power.
Inside Son 1 class
Son is very naughty
Inside father class
father has intelligence and deep thinking power.
Inside Son 2 class
Son is very disciplined
```
- Bottom Status Bar:** PEP 8: E301 expected 1 blank line, found 0, 38:1, CR LF, UTF-8, 4 spaces, Python 3.9.
- System Tray:** Shows battery level, network, and date/time (30-01-2024).

5) Hybrid Inheritance

The screenshot shows the PyCharm Professional Edition interface with the following details:

- Project:** DEPython
- File:** Inheritance.py
- Code Content:**

```
# Hybrid Inheritance
print('Hybrid Inheritance')
99 class Grandfather:
100     def gf_quality(self):
101         print('inside Grandfather class')
102         print('Grandfather was a honest person')
103
104     def gf_nature(self):
105         print('inside Grandfather class')
106         print('Grandfather is a soft mind person')
107
108     def gf_hobby(self):
109         print('inside Grandfather class')
110         print('Grandfather s hobby is farming')
111
112 class Father(Grandfather):
113     def father_quality(self):
114         print('inside Father class')
115         print('Father has intelligence and deep thinking power')
116
117     def father_nature(self):
118         print('inside Father class')
119         print('Father is strict in principle')
120
121 class Son1(Father):
122     def son1_quality(self):
123         print('inside Son1 class')
124         print('son1 has deep knowledge about Islamic law')
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
```

- Toolbars and Status Bar:** The status bar at the bottom right shows "51:1 Python 3.9", "14:20", and the date "30-01-2024".

The screenshot shows the PyCharm Professional Edition interface with the following details:

- Project:** DEPython
- File:** Inheritance.py
- Code Content:**

```
def father_nature(self):
    print('inside Father class')
    print('Father is strict in principle')

class Son1(Father):
    def son1_quality(self):
        print('inside Son1 class')
        print('son1 has deep knowledge about Islamic law')

k1=Son1()
k1.gf_hobby()
k1.father_nature()
k1.son1_quality()

class Son2(Father):
    def son2_quality(self):
        print('inside Son2 class')
        print('son2 want to become software engineer')

k2=Son2()
k2.gf_quality()
k2.father_quality()
k2.son2_quality()
```

- Toolbars and Status Bar:** The status bar at the bottom right shows "51:1 Python 3.9", "14:20", and the date "30-01-2024".

Output:

The screenshot shows the PyCharm Professional Edition interface. The project navigation bar at the top lists several Python files. The main code editor window displays the `Inheritance.py` file. The code defines a `Father` class with methods `father_nature` and `son1_quality`, and a `Son1` class that inherits from `Father`. A variable `k1` is assigned to an instance of `Son1`, and then `k1`'s methods are called. The output window below the editor shows the execution results, which include the printed statements from the code. The status bar at the bottom right indicates the current date and time.

```
def father_nature(self):
    print('inside Father class')
    print('Father is strict in principle')

class Son1(Father):
    def son1_quality(self):
        print('Inside Son1 class')
        print('son1 has deep knowledge about Islamic law')

k1=Son1()
k1.gf_hobby()
k1.father_nature()
k1.son1.quality()

Process finished with exit code 0
```

Son is very disciplined
Hybrid Inheritance
inside Grandfather class
Grandfather's hobby is farming
inside Father class
Father is strict in principle
inside Son1 class
son1 has deep knowledge about Islamic law
inside Grandfather class
Grandfather was a honest person
inside Father class
Father has intelligence and deep thinking power!
inside Son2 class
son2 want to become software engineer

Process finished with exit code 0

7) Exception Handling

Exception is an event which occurs during the execution of program that disrupts the normal flow of program

Exception handling can be done using try, except, else and finally statements.

The screenshot shows the PyCharm Professional Edition interface. The project navigation bar at the top lists several Python files. The main code editor window displays the `Exception.py` file. The code contains a `try` block with a division operation and an `except` block for `IndexError`. The output window below the editor shows the execution results, which include the printed error message. The status bar at the bottom right indicates the current date and time.

```
# Example-1
l=[1,2,3,4]
a=2
try:
    x=10/2
    print(l[2])
    print(z)
except IndexError as e:
    print(e)
except ZeroDivisionError as e:
    print(e)
except NameError as e:
    print(e)
```

C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/Exception.py"
3
name 'z' is not defined

Process finished with exit code 0

User Defined Exception

The screenshot shows the PyCharm Professional Edition interface. The project navigation bar at the top includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and Polymorphism.py - Exception.py. The main window displays a code editor with Exception.py open, containing Python code for a user-defined exception named Employee. The code checks if a name is valid (alpha-numeric) and if a salary is between 30000 and 100000. A run configuration titled 'Exception' is active, showing the output of the program running. The output window shows the user inputting 'nn' as a name and '76' as a salary, resulting in an 'Invalid Salary' error message. The status bar at the bottom right indicates the current date and time as 30-01-2024.

```
# User defined Exception
class Employee(Exception):
    def __init__(self, msg):
        super().__init__(msg)
name=input('Enter name:')
if name.replace(" ", "").isalpha():
    print('Name:', name)
else:
    try:
        raise Employee('invalid name')
    except Employee as e:
        print(e)
salary=int(input("Enter Salary:"))
if salary<30000 or salary>100000:
    try:
        raise Employee('Invalid Salary')
    except Employee as e:
        print(e)
else:
    print('Salary:', salary)
```

8) File Handling

Python allows users to handle files i.e., to read and write files along with other operations.

Creating a file in Python:

1) Create a file using `open()`

2) Two parameters: filename, access mode to create a file

`seek()` : change or move cursor to place where data must be read or written in the file

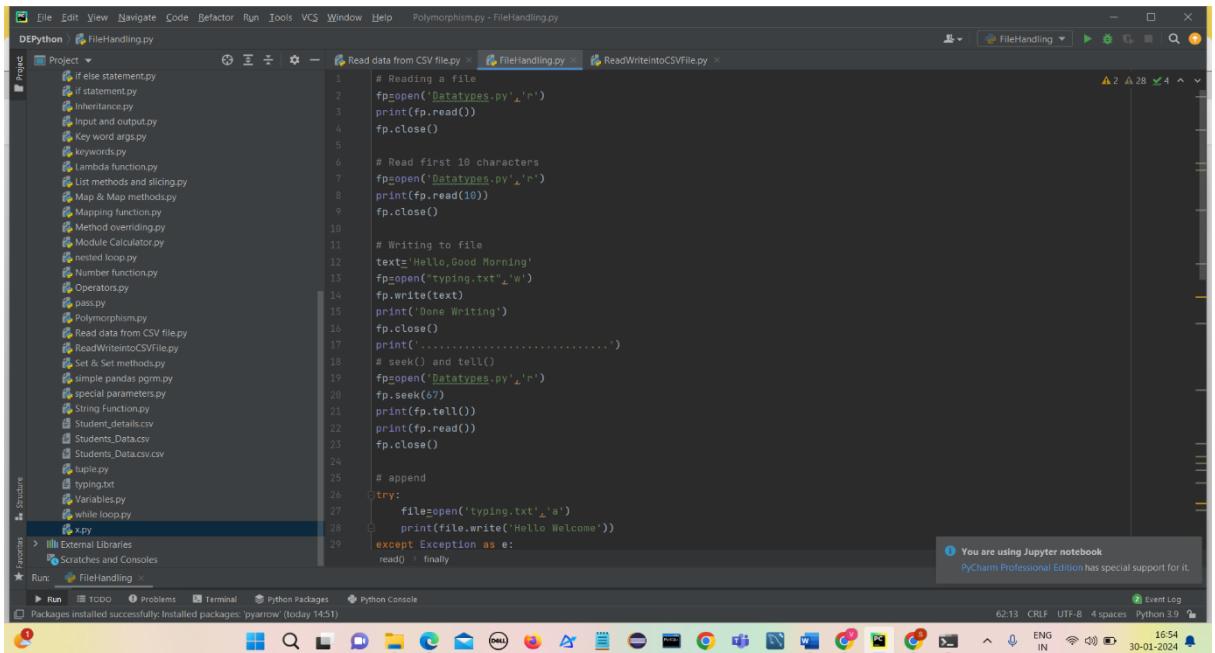
`tell()`: return current position of file pointer from beginning of file

`read()`: returns file content in the form of string

`readLine()`: Reads single line

`readLines()`: read file into list

`writelines()`: write list of strings to file



```
# Reading a file
fp=open('Datatypes.py','r')
print(fp.read())
fp.close()

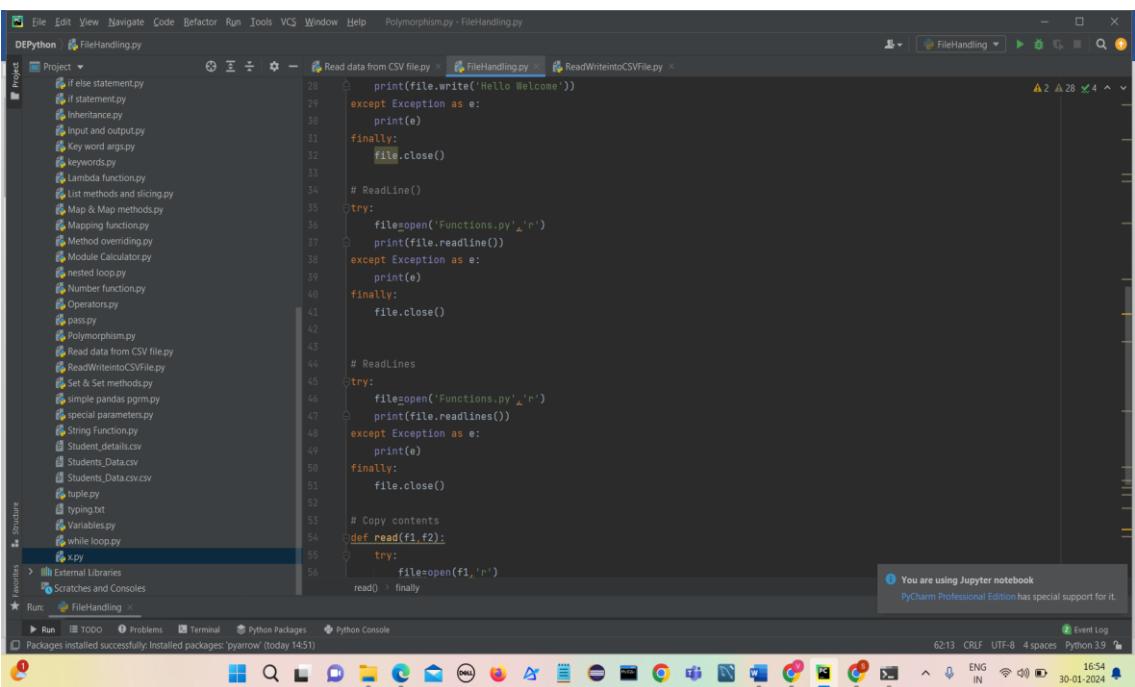
# Read first 10 characters
fp=open('Datatypes.py','r')
print(fp.read(10))
fp.close()

# Writing to File
text='Hello, Good Morning'
fp=open('typing.txt','w')
fp.write(text)
print('Done Writing')
fp.close()

print('.....')

# seek() and tell()
fp=open('Datatypes.py','r')
fp.seek(67)
print(fp.tell())
print(fp.read())
fp.close()

# append
try:
    file=open('typing.txt','a')
    print(file.write('Hello Welcome'))
except Exception as e:
    read() finally
```



```
print(file.write('Hello Welcome'))

except Exception as e:
    print(e)
finally:
    file.close()

# ReadLine()
try:
    file=open('Functions.py','r')
    print(file.readline())
except Exception as e:
    print(e)
finally:
    file.close()

# ReadLines
try:
    file=open('Functions.py','r')
    print(file.readlines())
except Exception as e:
    print(e)
finally:
    file.close()

# Copy contents
def read(f1,f2):
    try:
        file=open(f1,'r')
        read() finally
```

```
file.close()

# Readlines
try:
    file=open('Functions.py','r')
    print(file.readlines())
except Exception as e:
    print(e)
finally:
    file.close()

# Copy contents
def read(f1,f2):
    try:
        file=open(f1,'r')
        txt=file.read()
        file=open(f2,'w')
        file.write(txt)
    except Exception as e:
        print(e)
    finally:
        file.close()
read('Datatypes.py','abc.txt')
```

You are using Jupyter notebook
PyCharm Professional Edition has special support for it.

The screenshot shows the PyCharm Professional Edition interface with a dark theme. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and Polymorphism.py - FileHandling.py. The Project tool window on the left lists files like if else statement.py, if statement.py, Inheritance.py, and FileHandling.py. The Run tool window shows a successful run of FileHandling.py with the command C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/FileHandling.py". The main editor window displays the following code:

```
C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/FileHandling.py"
FileHandling.py
print(e)
read() > finally
Run: FileHandling
C:\Users\DELL\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/DELL/OneDrive/Desktop/Data Engineering Python/DEPython/FileHandling.py"
#Datatypes
print("Datatypes")
#int
print(type(12))
#float
print(type(18.9))
#complex
print(type(2+4j))
#List
l=[12,11,90.89,'apple']
print(type(l))
#Tuple
t=(12,90,34.33,'tuple')
print(type(t))
#Set
s={1,2,3,4,5}
print(type(s))
#Dictionary
d={'a':1,'b':2,'c':3}
print(type(d))
# String
str='Hello'
print(type(str))
#Boolean
b=True
print(type(b))
#DataFrames

```

A message box in the center says "You are using Jupyter notebook" and "PyCharm Professional Edition has special support for it." The bottom status bar shows "59:1 CRLF UTF-8 4 spaces Python 3.9" and the date "30-01-2024".

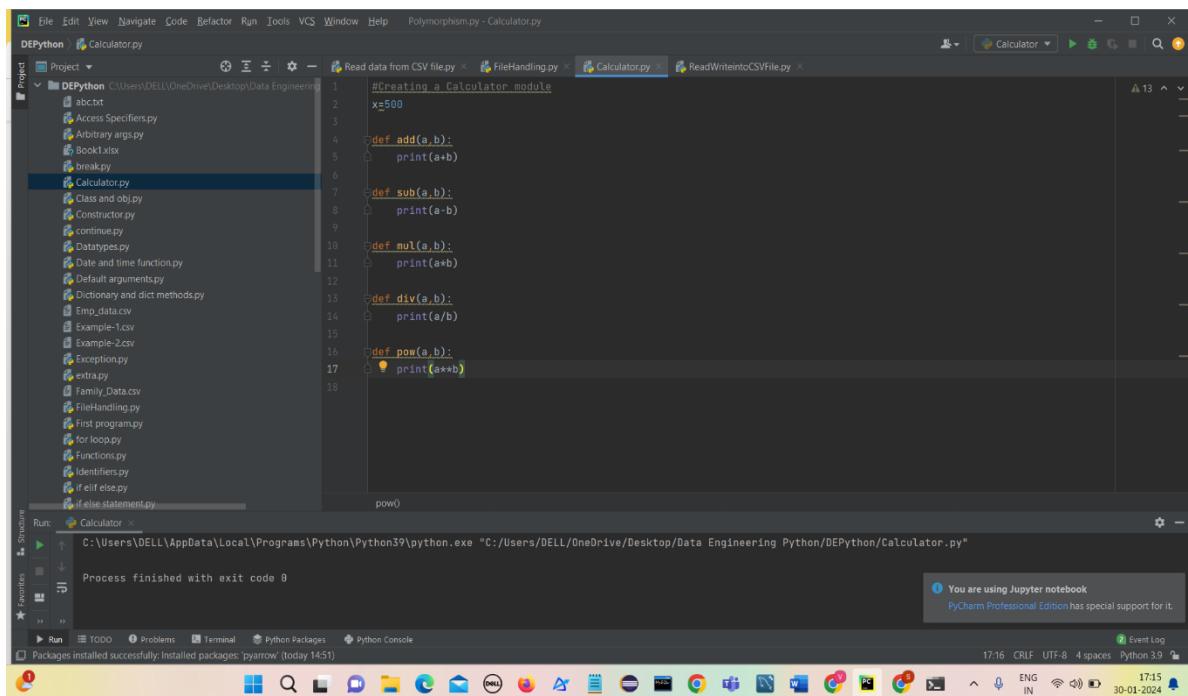
This screenshot is nearly identical to the one above, showing the same PyCharm interface and code execution results. The main difference is in the code editor, where the last few lines of the script have been modified to include a function definition and a print statement:

```
    # Functions
    ['# Functions\n', 'def greetings():\n', '    print("Welcome to Python")\n', 'greetings()\n', '\n', 'def sum():\n', '    print("Enter two numbers")\n', '    a, b = map(int, input())\nProcess finished with exit code 0
|
```

The message box "You are using Jupyter notebook" and "PyCharm Professional Edition has special support for it." is still present, along with the status bar information.

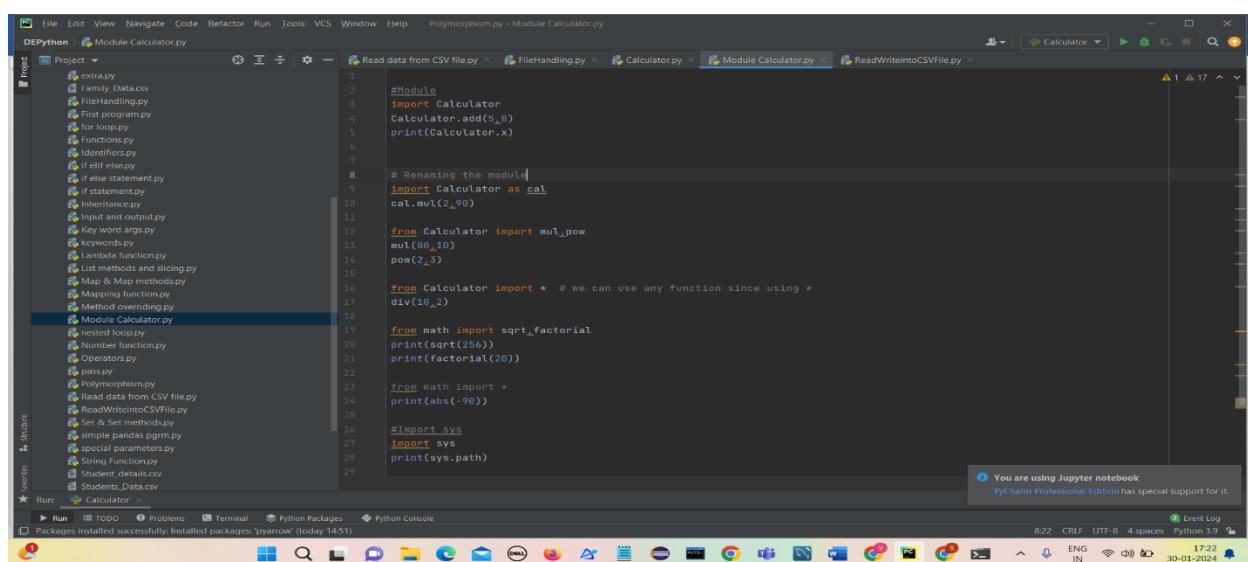
9) Modules

- Module is a file containing Python code.
- The code within a module can define functions, classes, and variables that can be utilized in other Python scripts or modules.
- Modules help organize code into logical units, making it easier to manage and reuse.
- Modules can be shared, reused and collaborated with others.



```
#Creating a Calculator module
x=500
def add(a,b):
    print(a+b)
def sub(a,b):
    print(a-b)
def mul(a,b):
    print(a*b)
def div(a,b):
    print(a/b)
def pow(a,b):
    print(a**b)
```

Calculator.py is a module containing all python functions to add,subtract,multiply,divide and calculate power of two numbers.



```
#Module
import Calculator
Calculator.add(5,8)
print(Calculator.x)

# Renaming the module
import Calculator as cal
cal.mul(2,99)

from Calculator import mul,pow
mul(80,10)
pow(2,3)

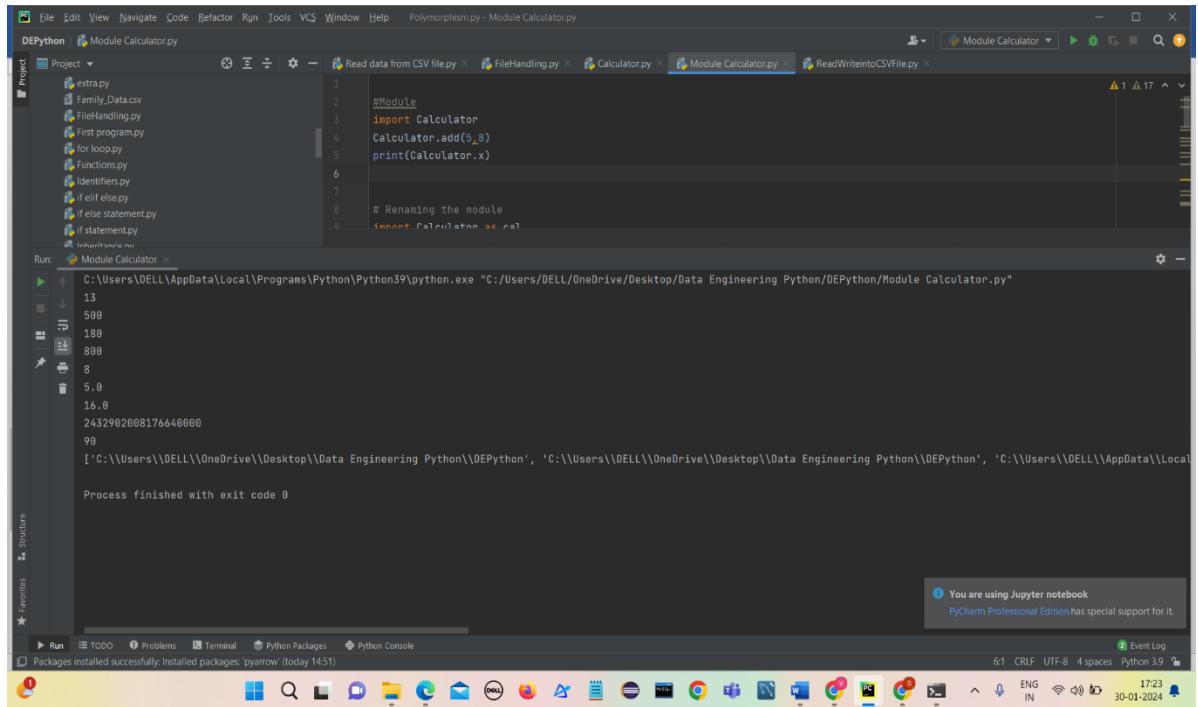
from Calculator import * # we can use any function since using *
div(10,2)

from math import sqrt,factorial
print(sqrt(256))
print(factorial(20))

from math import *
print(fabs(-98))

#import sys
#import sys
#print(sys.path)
```

The methods in Calculator module can be used by importing the required functions.



```
#Module
import Calculator
Calculator.add(5,8)
print(Calculator.x)

# Renaming the module
import Calculator as cal
```

Process finished with exit code 0

You are using Jupyter notebook
PyCharm Professional Edition has special support for it.

Different ways to import:

- 1) import module_name
- 2) Renaming a module----- import module_name as xyz
- 3) From module_name import fun1
- 4) To import all functions----- from module_name import *