Unit – 5

Inheritance and File Handling

Introduction

Inheritance is a powerful feature in object oriented **programming**. It refers to defining a new class with little or no modification to an existing class. The new class is called derived (or child) class and the one from which it **inherits** is called the base (or parent) class.

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

Inheritance is the capability of one class to derive or inherit the properties from another class. The benefits of inheritance are:

1. It represents real-world relationships well.
2. It provides **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.
3. It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Ex: # A Python program to demonstrate inheritance

```
# Base or Super class. Note object in bracket.
# (Generally, object is made ancestor of all classes)
# In Python 3.x "class Person" is
# equivalent to "class Person(object)"
class Person(object):

    # Constructor
    def __init__(self, name):
        self.name = name

    # To get name
    def getName(self):
        return self.name

    # To check if this person is an employee
    def isEmployee(self):
        return False


# Inherited or Subclass (Note Person in bracket)
class Employee(Person):
    # Here we return true
    def isEmployee(self):
        return True

# Driver code
emp = Person("Geek1")  # An Object of Person
print(emp.getName(), emp.isEmployee())

emp = Employee("Geek2") # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

Output: Greek1 False

Greek2 True

Inheriting Classes in python:
Like Java Object class, in Python (from version 3.x), object is root of all classes.
In Python 3.x, "class Test(object)" and "class Test" are same.
In Python 2.x, "class Test(object)" creates a class with object as parent (called new style class) and "class Test" creates old style class (without object parent). Refer this for more details.
Subclassing (Calling constructor of parent class)
A child class needs to identify which class is its parent class. This can be done by mentioning the parent class name in the definition of the child class.
Eg: class **subclass_name (superclass_name)**:

```
_ _ _
_ _ _
# Python code to demonstrate how parent constructors
# are called.

# parent class
class Person( object ):

        # __init__ is known as the constructor
        def __init__(self, name, idnumber):
                self.name = name
                self.idnumber = idnumber
        def display(self):
                print(self.name)
                print(self.idnumber)

# child class
class Employee( Person ):
        def __init__(self, name, idnumber, salary, post):
                self.salary = salary
                self.post = post

                # invoking the __init__ of the parent class
                Person.__init__(self, name, idnumber)
```

```
# creation of an object variable or an instance
a = Employee('Rahul', 886012)

# calling a function of the class Person using its instance
a.display()
```

Output:

```
Rahul

886012
```

‘a’ is the instance created for the class Person. It invokes the __init__() of the referred class. You can see ‘object’ written in the declaration of the class Person. In Python, every class inherits from a built-in basic class called ‘object’. The constructor i.e. the ‘__init__’ function of a class is invoked when we create an object variable or an instance of the class.

The variables defined within __init__() are called as the instance variables or objects. Hence, ‘name’ and ‘idnumber’ are the objects of the class Person. Similarly, ‘salary’ and ‘post’ are the objects of the class Employee. Since the class Employee inherits from class Person, ‘name’ and ‘idnumber’ are also the objects of class Employee.

If you forget to invoke the __init__() of the parent class then its instance variables would not be available to the child class.

Different forms of Inheritance:

Types of Inheritance depends upon the number of child and parent classes involved. There are four types of inheritance in Python:

1. **Single Inheritance:** Single inheritance enables a derived class to inherit properties from a single parent class, thus enabling code reusability and addition of new features to existing code.

1. Single inheritance: When a child class inherits from only one parent class, it is called single inheritance.

Ex: Python program to demonstrate

```
# single inheritance


# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")

# Derived class
class Child(Parent):
    def func2(self):
        print("This function is in child class.")

# Driver's code
object = Child()
object.func1()
object.func2()
```

```
Output:     This function is in parent class.

            This function is in child class.
```

2. Multiple inheritance: When a child class inherits from multiple parent classes, it is called multiple inheritance.
Unlike Java and like C++, Python supports multiple inheritance. We specify all parent classes as a comma-separated list in the bracket.

When a class can be derived from more than one base classes this type of inheritance is called multiple inheritance. In multiple inheritance, all the features of the base classes are inherited into the derived class.

```
# Python program to demonstrate
# multiple inheritance


# Base class1
class Mother:
    mothername = ""
    def mother(self):
        print(self.mothername)

# Base class2
class Father:
    fathername = ""
    def father(self):
        print(self.fathername)

# Derived class
class Son(Mother, Father):
    def parents(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)

# Driver's code
s1 = Son()
s1.fathername = "RAM"
s1.mothername = "SITA"
s1.parents()
```

```
Output:    Father : RAM

           Mother : SITA
```

3.Multilevel Inheritance

In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class. This is similar to a relationship representing a child and grandfather.

```
# Python program to demonstrate
# multilevel inheritance


# Base class
class Grandfather:
    grandfathername =""
    def grandfather(self):
        print(self.grandfathername)

# Intermediate class
class Father(Grandfather):
    fathername = ""
    def father(self):
        print(self.fathername)

# Derived class
class Son(Father):
    def parent(self):
        print("GrandFather :", self.grandfathername)
        print("Father :", self.fathername)

# Driver's code
s1 = Son()
s1.grandfathername = "Srinivas"
s1.fathername = "Ankush"
s1.parent()
```

Output:

```
GrandFather : Srinivas

Father : Ankush
```

4. **Hierarchical Inheritance:** When more than one derived classes are created from a single base this type of inheritence is called hierarchical inheritance. In this program, we have a parent (base) class and two child (derived) classes.

Ex: # Python program to demonstrate

```
# Hierarchical inheritance


# Base class
class Parent:
      def func1(self):
          print("This function is in parent class.")
```

```
# Derived class1
class Child1(Parent):
     def func2(self):
         print("This function is in child 1.")

# Derivied class2
class Child2(Parent):
     def func3(self):
         print("This function is in child 2.")

# Driver's code
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
             object2.func3()
```

Output:

```
 This function is in parent class.
This function is in child 1.
This function is in parent class.
This function is in child 2.
```

5. **Hybrid Inheritance:** Inheritence consisting of multiple types of inheritence is called hybrid inheritence.

1. Ex:

```
# Python program to demonstrate
# hybrid inheritance


class School:
     def func1(self):
         print("This function is in school.")

class Student1(School):
     def func2(self):
         print("This function is in student 1. ")

class Student2(School):
     def func3(self):
         print("This function is in student 2.")

class Student3(Student1, School):
    def func4(self):
        print("This function is in student 3.")

# Driver's code
object = Student3()
object.func1()
object.func2()
```

Output:

```
This function is in school.

This function is in student 1.
```

FILES

Introduction: **Python** has a built-in open() function to open a **file**. This function returns a **file** object, also called a handle, as it is used to read or modify the **file** accordingly. We can specify the mode while opening a **file**. In mode, we specify whether we want to read r , write w or append a to the **file**.

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but alike other concepts of Python, this concept here is also easy and short. Python treats file differently as text or binary and this is important. Each line of code includes a sequence of characters and they form text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun. Let’s start with Reading and Writing files.

Working of open() function

We use **open ()** function in Python to open a file in read or write mode. As explained above, open ( ) will return a file object. To return a file object we use **open()** function along with two arguments, that accepts file name and the mode, whether to read or write. So, the syntax being: **open(filename, mode)**. There are three kinds of mode, that Python provides and how files can be opened:

- **r** “, for reading.
- “ **w** “, for writing.
- “ **a** “, for appending.
- “ **r+** “, for both reading and writing

One must keep in mind that the mode argument is not mandatory. If not passed, then Python will assume it to be “ **r** ” by default. Let’s look at this program and try to analyze how the read mode works:

Ex: # a file named "geek", will be opened with the reading mode.

```
file = open('geek.txt', 'r')
# This will print every line one by one in the file
for each in file:
    print (each)
```

The open command will open the file in the read mode and the for loop will print each line present in the file.

Working of read() mode

There is more than one way to read a file in Python. If you need to extract a string that contains all characters in the file then we can use **file.read()**. The full code would work like this:

Ex: # Python code to illustrate read() mode

```
file = open("file.text", "r")
print file.read()
```

Another way to read a file is to call a certain number of characters like in the following code the interpreter will read the first five characters of stored data and return it as a string:

Ex: # Python code to illustrate read() mode character wise

```
file = open("file.txt", "r")
print file.read(5)
```

Creating a file using write() mode

Let’s see how to create a file and how write mode works:
To manipulate the file, write the following in your Python environment:

Ex: # Python code to create a file

```
file = open('geek.txt','w')
file.write("This is the write command")
file.write("It allows us to write in a particular file")
file.close()
```

The close() command terminates all the resources in use and frees the system of this particular program.

Working of append() mode

Let’s see how the append mode works:

Ex: # Python code to illustrate append() mode

```
file = open('geek.txt','a')
file.write("This will add this line")
file.close()
```

There are also various other commands in file handling that is used to handle various tasks like:

rstrip(): This function strips each line of a file off spaces from the right-hand side.

lstrip(): This function strips each line of a file off spaces from the left-hand side.

It is designed to provide much cleaner syntax and exceptions handling when you are working with code. That explains why it’s good practice to use them with a statement where applicable. This is helpful because using this method any files opened will be closed automatically after one is done, so autocleanup.

Example:

```
# Python code to illustrate with()
with open("file.txt") as file:
    data = file.read()
# do something with data
```

Using write along with with() function

We can also use write function along with with() function:

Ex: # Python code to illustrate with() alongwith write()

```
with open("file.txt", "w") as f:
    f.write("Hello World!!!")
```

split() using file handling

We can also split lines using file handling in Python. This splits the variable when space is encountered. You can also split using any characters as we wish. Here is the code:

Ex: # Python code to illustrate split() function

```
with open("file.text", "r") as file:
    data = file.readlines()
    for line in data:
        word = line.split()
        print word
```