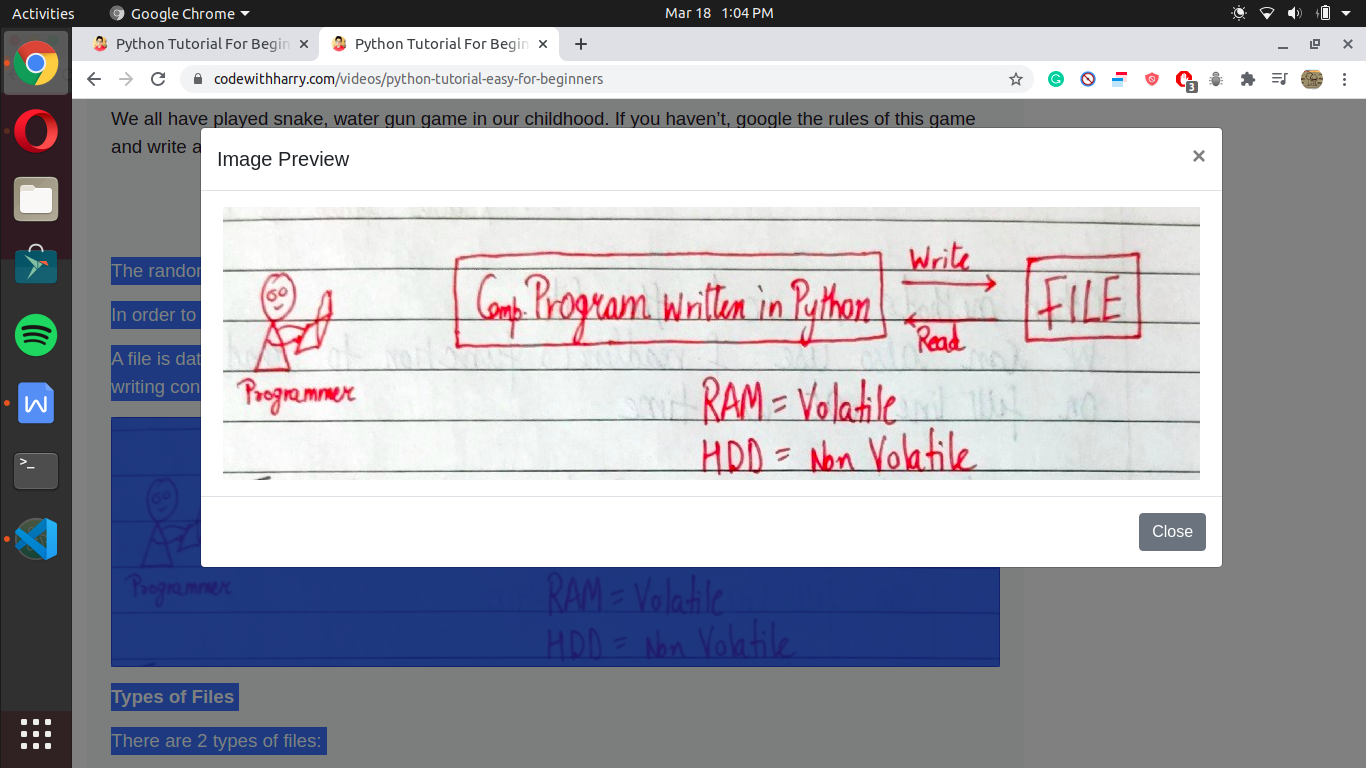
python\_CodeWithHarry\_2

#### Chapter 9 – File I/O

The random access memory is volatile and all its contents are lost once a program terminates.

In order to persist the data forever, we use files.

A file is data stored in a storage device. A python program can talk to the file by reading content from it and writing content to it.

********

****Types of Files****

There are 2 types of files:

1. Text files (.txt, .c, etc)
2. Binary files (.jpg, .dat, etc)

Python has a lot of functions for reading, updating, and deleting files.

****Opening a file****

Python has an open() function for opening files. It takes 2 parameters: filename and mode.

open(“this.txt”, “r”)

Here, “this” is the file name and “r” is the mode of opening (read mode)

****Reading a file in Python****

f = open(“this.txt”, “r”)     #Opens the file in r mode

text = f.read()          #Read its content

print(text)     #Print its contents

f.close()         #Close the fie

We can also specify the number of characters in read () function:

f.read(2)       #Reads first 2 characters

****Other methods to read the file****

We can also use f.readline() function to read on full line at a time.

f.readline()               #Reads one line from the file

****Modes of opening a file****

r – open for reading

w – open for writing

a – open for appending

+ -> open for updating

‘rb’ will open for read in binary mode

‘rt’ will open for read in text mode

****Writing Files in Python****

In order to write to a file, we first open it in write or append mode after which, we use the python’s f.write() method to write to the file!

f = open(“this.txt”, “w”)

f.write(“This is nice”)        #Can be called multiple times

f.close()

****With statement****

The best way to open and close the file automatically is the “with” statement.

with open(“this.txt”) as f:

            f.read()

#There is no need to write f.close() as it is done automatically

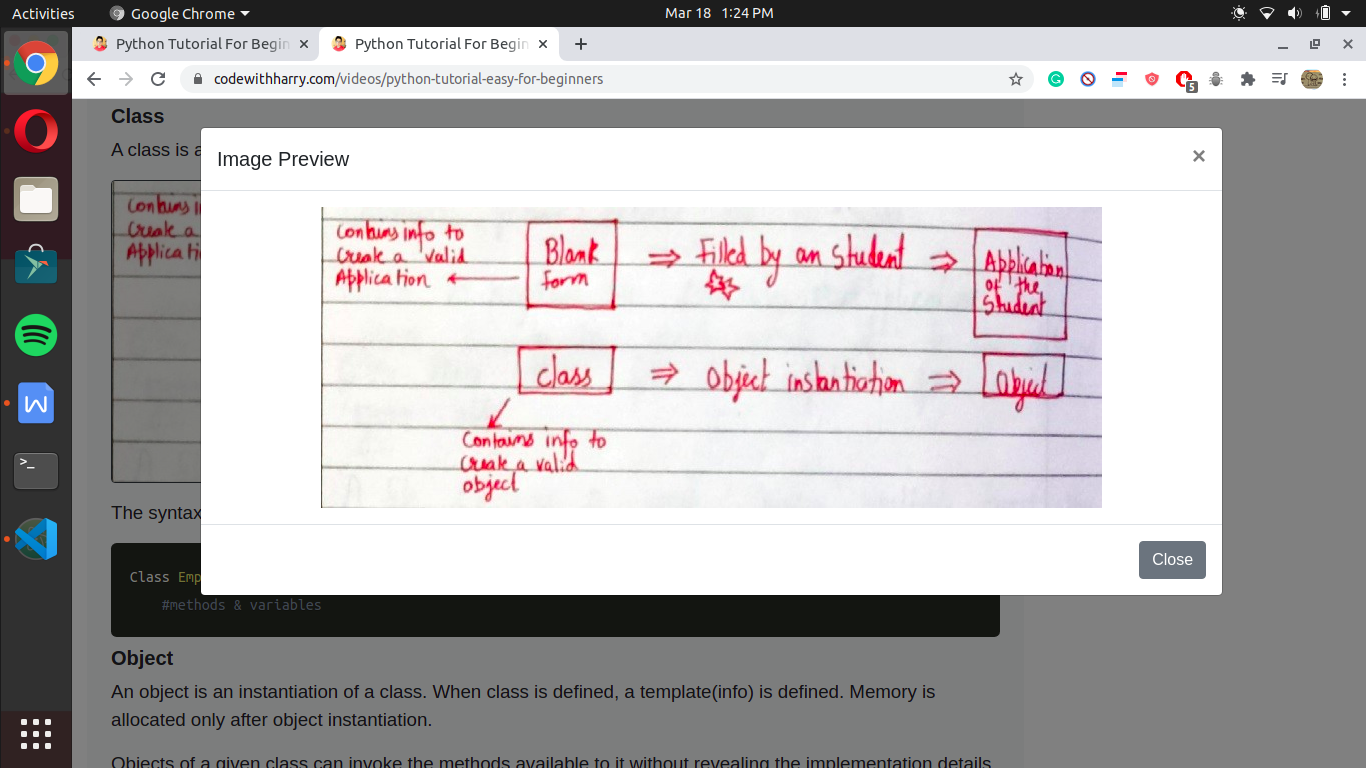
#### Chapter 10 – Object-Oriented Programming

Solving a problem by creating objects is one of the most popular approaches in programming. This is called object-oriented programming.

This concept focuses on using reusable code. (Implements DRY principle)

##### ****Class****

A class is a blueprint for creating objects.



The syntax of a class looks like this:

Class Employee: [classname is written in PascalCase]

#methods & variables

##### ****Object****

An object is an instantiation of a class. When class is defined, a template(info) is defined. Memory is allocated only after object instantiation.

Objects of a given class can invoke the methods available to it without revealing the implementation details to the user.     #Abstraction & Encapsulation!

****Modelling a problem in OOPs****

We identify the following in our problem

Noun -> Class -> Employee

Adjective -> Attributes -> name,age,salary

Verbs -> Methods -> getSalary(), increment()

****Class Attributes****

An attribute that belongs to the class rather than a particular object.

Example:

Class Employee:

company = “Google” #Specific to each class

harry = Employee() #Object instantiation

harry.company

Employee.company = “YouTube” #changing class attribute

****Instance Attributes****

An attribute that belongs to the Instance (object)

Assuming the class from the previous example:

harry.name = “Harry”

harry.salary = “30K” #Adding instance attributes

Note: Instance attributes take preference over class attributes during assignment and retrieval.

harry.attribute1  :

1. Is attribute1 present in object?
2. Is attribute1 present in class?

****‘self’ parameter****

self refers to the instance of the class.

It is automatically passed with a function call from an object.

harry.getSalary()

here, self is harry and above line of code is equivalent to Employee.getSalary(harry)

This function getsalary is defined as:

class Employee:

company = “Google”

def getSalary(self):

print(“Salary is not there”)

****Static method****

Sometimes we need a function that doesn’t use the self-parameter. We can define a static method like this:

@staticmethod #decorator to mark greet as a static methoddef greet():

print(“Hello user”)

****\_\_init\_\_() constructor****

\_\_init\_\_() is a special method which runs as soon as the object is created.

\_\_init\_\_() method is also known as constructor.

It takes self-argument and can also take further arguments.

For Example:

class Employee:

def \_\_init\_\_(self,name):

self.name = name

def getSalary(self):

#Some code…

harry = Employee(“Harry”) #Object can be instantiated using constructor like this!

class Employee:

company ="Google"

def getsalary(self):

print("salary is 100K")

print(f"salary is {self.salary}")

#self.salary ka mtlb jis pe run kr rahe ho us ki salary

shashank= Employee()

# shashank.getsalary() = Employee.getsalary(shashank)

# this above written line get converted to # Employee.getsalary(shashank)

#which passes a argument and self take the address of the parameter we pass when we calls the function

shashank.salary= 100000

shashank.getsalary()

#### Chapter 11 – Inheritance & more on OOPs

Inheritance is a way of creating a new class from an existing class.

****Syntax:****

class Emoloyee: #Base Class

#Codeclass Programmer(Employee): #Derived or child class

#Code

Copy

We can use the methods and attributes of Employee in Programmer object.

Also, we can overwrite or add new attributes and methods in the Programmer class.

##### ****Type of Inheritance****

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance

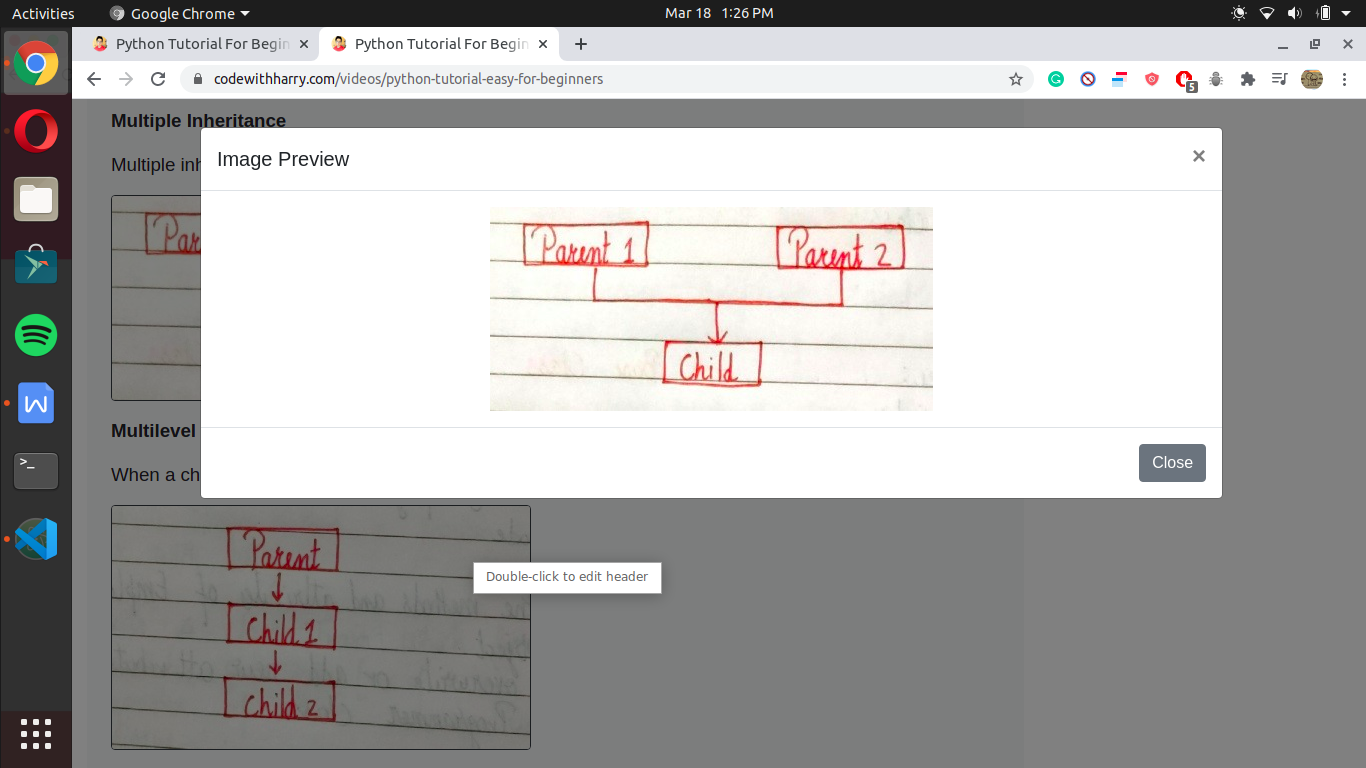
****Single Inheritance****

Single inheritance occurs when child class inherits only a single parent class.

Base -> Derived

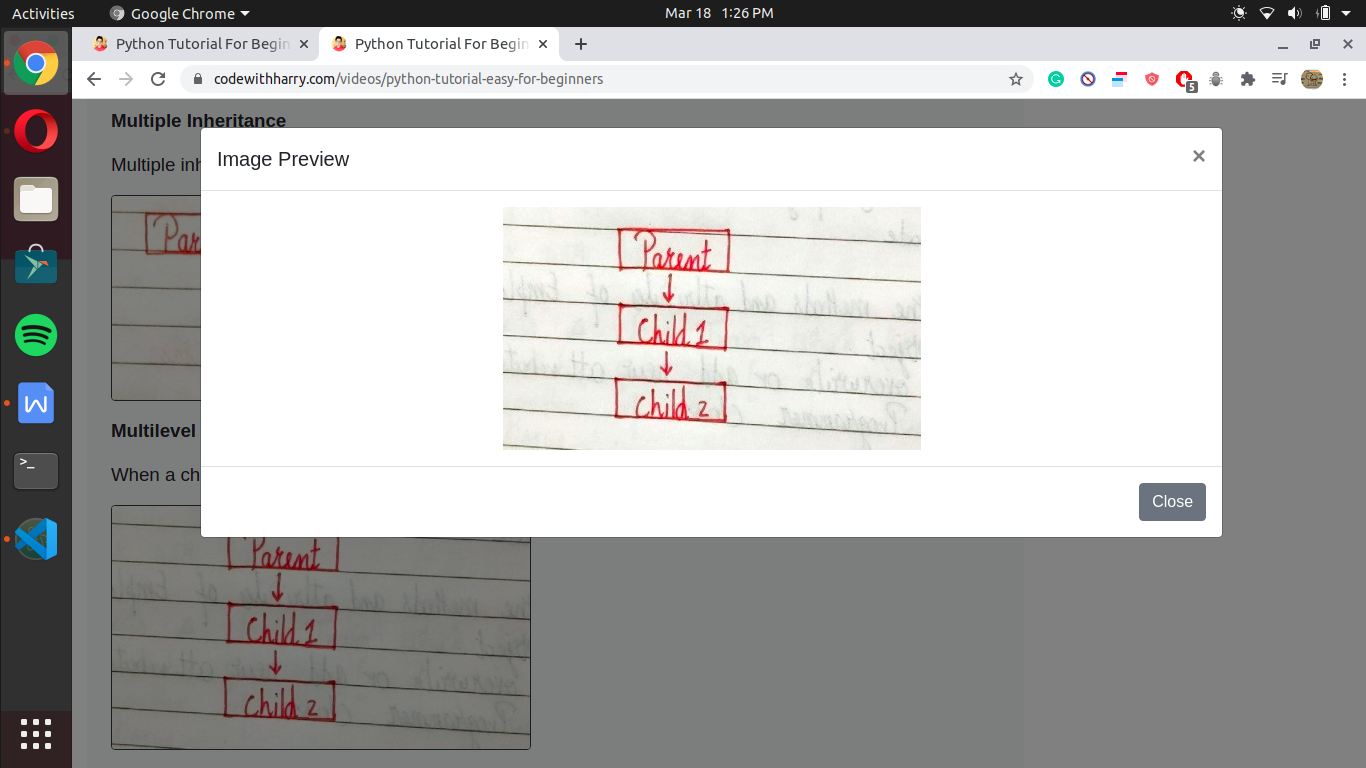
****Multiple Inheritance****

Multiple inheritance occurs when the child class inherits from more than one parent class.

********

****Multilevel Inheritance****

When a child class becomes a parent for another child class.

********

****Super() method****

Super method is used to access the methods of a super class in the derived class.

super().\_\_init\_\_()  #Calls constructor of the base class

****Class methods****

A class method is a method which is bound to the class and not the object of the class.

@classmethod decorator is used to create a class method.

Syntax to create a class method:

@classmethoddef (cls, p1, p2):

#code

****@property decorators****

Consider the following class

class Employee:

@property

def name(self):

return self.ename

if e = Employee() is an object of class employee, we can print (e.name) top print the ename/call name() function.

****@.getters and @.setters****

The method name with @property decorator is called getter method.

We can define a function + @name.setter decorator like below:

@name.setterdef name(self, value):

self.ename = value

****Operator overloading in Python****

Operators in python can be overloaded using dunder methods.

These methods are called when a given operator is used on the objects.

Operators in python can be overloaded using the following methods:

p1 + p2 -> p1.\_\_add\_\_(p2)

p1 – p2 -> p1.\_\_sub\_\_(p2)

p1 \* p2 -> p1.\_\_mul\_\_(p2)

p1 / p2 -> p1.\_\_truediv\_\_(p2)

p1 // p2 -> p1.\_\_floordiv\_\_(p2)

****Other dunder/magic methods in Python****

\_\_str\_\_() -> used  to set what gets displayed upon calling str(obj)

\_\_len\_\_() -> used to set what gets displayed upon calling .\_\_len\_\_() or len(obj)

class Employee:

company ="Google"

salary =100

location ="delhi"

# def changeSalary(self,sal):

# self.salary=sal

#self.\_\_class\_\_.salary=sal #by this we can change class attribute

#another form to change the class attribute is

@classmethod #classmethod ke andar hum jp kuch bhi karege wo class me change hoga

def changeSalary(cls,sal): #here cls is a class reference

cls.salary =sal

e= Employee()

print(e.salary)

e.changeSalary(455) #its added a instance attribute but the class attribute is same it doesn't change

print(e.salary)

print(Employee.salary)