**Python- things to Remember**

**1)To connect the two sentences on the same line**

print("Hello how are you"**,** end = ", ")  
print("I am fine thanks")

**--output—**

**Hello how are you, I am fine thanks**

**2) \n gives a new line**

**3)\t gives the space of the tab button**

**4)\’ gives quotes**

***2), 3), 4) are escape sequences, and can be added between strings***

***5) Concatenation: adding 2 strings***

***You cant add a string to an integer or float***

**6) For slicing stringsss**

variable = "Trisha is the best"  
print(variable[**0**:**6**])

**--outout—**

**Trisha**

variable = "Trisha is the best"  
print(variable[**0**:**6**:**2**])

**--output—**

**Tih**

**The 2 in the above code is to skip 1 element**

**Also, if instead of 0,6,2 we leave the value blank, i.e.**

variable = "Trisha is the best"  
print(variable[::])

**Then the default values taken in the first place is 0, second place is- full length of string (18 in this case0, and third value as 1 i.e. skips no digit**

**So –output—**

**Trisha is the best**

**-Putting a negative number in the third slot, reverses the output and then skips the required no. of digits**

**Eg tseb eth si ahsirt**

**7) Lists**

**Lists are written with [] brackets**

**Elements can be changed**

* **List.sort()- sorts it in ascending order**
* **List.reverse()- element order gets reversed i.e. last to first**

**For slicing in general, the second digit is NOT inclusive, whereas the first digit is INCLUSIVE**

**FOR THE THIRD DIGIT DURING SLICING, ALWAYS JUST TAKE -1 TO REVERSE LIST, NUMBERS LIKE -2, -3 ETC DO NOT WORK IF FIRST 2 DIGITS ARE PROVIDED.**

**-2,-3, etc only work if first two digits are left blank.**

* **List.append()- adds it to the END of the list**
* **List.insert(\_\_, \_\_)**

**Takes 2 arguments, first being the index number, and second being what ever you want to add**

* **List.pop()- removes last element of the list**
* **List.remove(), in the beacket you can add what ever you want to remove**

**8) Tuples**

**The digits in it CANNOT BE CHANGED**

**Tuple is written with () bracket**

**If a tuple has 1 element onlt, then write a comma after it, otherwise python won’t recognise it as a tuple**

**9) To swap values of two variables**

a = **1**b = **8**a**,** b = b**,** a  
print(a**,** b)

**-- output –**

**8 1**

**10) Dictionaries**

**Dictionaries are written with {}**

**Key:values**

d2 = {"Harry":"Burger",

"Rohan":"Fish",

"SkillF":"Roti",

"Shubham":{"B":"maggie", "L":"roti", "D":"Chicken"}}

**Given above is the format for dictionaries. The Shubham one is a dictionary in a dictionary**

**You can address it with :**

print(d2["Shubham"])

**del d2[“SkillF”] will delete it from the dictionary along with its value**

**for adding to a list we need to use update.()**

**eg:** d2.update({"Leena": "Toffee"})

**11)Sets**

**The difference between set and list is, that it has only unique values.**

**For eg if you append “Harry” in a list twice, there will be 2 elements in the list, that has harry, whereas in a set there will be just one “Harry”**

**To make a set write :  
set\_you\_want = set()**

**In sets you can add functions like union(), intersection(), etc**

**12) LOOPS**

**(FOR LOOP)**

**Let this be your list**

**It has a lists in a list**

list1 = [ ["Harry", 1], ["Larry", 2],

["Carry", 6], ["Marie", 250]]

**For a normal loop, you would write**

**For item in list1:**

**print(item)**

**--output--**

**['Harry', 1]**

**['Larry', 2]**

**['Carry', 6]**

**['Marie', 250]**

**Wheras if you address the list in list, you write**

for item, lollypop in list1:

print(item, lollypop)

**--output—**

**Harry 1**

**Larry 2**

**Carry 6**

**Marie 250**

**We can even add a string in the print sentence eg:**

for item**,** lollypop in list1:  
 print(item**,** "is"**,** lollypop)

**to get**

**--output—**

**Harry is 1**

**Larry is 2**

**Carry is 6**

**Marie is 250**

**\*\*This does not work for dictionaries\*\***

**For dictionary loops, you can either print the keys(without values) by putting Just items in the print and for statement**

**OR**

**Put** *items()***, at the end of the for statement, to get keys as well as their value**

**(WHILE LOOP)**

**Eg:**

**i = 0**

**While i = 0**

**Print(i)**

**i = I + 1**

**WATCH VIDEO 18- OF WHILE LOOPS IT IS CONFUSING**

**13)Operators**

* **Arithmetic Operators**

**+,-\*,/,**

**//- is for getting the integer value of the division answer**

**\*\*- exponent**

**%- Remainder(modulo)**

* **Assignments operators**

**Basically an operator after a variable that has an integer value**

**+=,/=,-=,%=**

* **Comparison operators**

**==,!=,>=,<=**

* **Logical operators**

**Basically if true and false are assigned to 2 variables, then and, or, is not**

* **Identity Operators**

**is not**

* **Membership Operators**

**If a list is given and you write in, not in, for a particular number/ member in the list then those are membership operators**

* **Bitwise Operators**

**|, & and used for binary**

**14) Functions**

**Syntax is given in the eg of code below:**

def function1(a, b):

print("Hello you are in function 1", a+b)

def function2(a, b):

"""This is a function which will calculate average of two numbers

this function doesnt work for three numbers"""

average = (a+b)/2

# print(average)

return average

# v = function2(5, 7)

# print(v)

**So a function is bacially, having variables, making letssay average, usuing those variables, and getting the average numbers for the numbers that you input while printing.**

**Doc strings are used to remember what the function is when there are MULTIPLE functions that you have defined.**

**After the def like write “””This is your doc string “””**

**It helps you remember what your function does.**

**To print out the docstring you write**

**Print(name\_of\_function.\_\_doc\_\_)**

**15) Try, except, exception handling**

**This comes in handy incase lets say half the program is something user input related, there is a chance that the code might show an error, not because of wrong code, but because the user added in some wrong input. This will show an error, and if the step after this part of the coding is IMPORTANT, then that won’t run. So use, try and except**

print("Enter num 1")

num1 = input()

print("Enter num 2")

num2 = input()

try:

print("The sum of these two numbers is",

int(num1)+int(num2))

except Exception as e:

print(e)

print("This line is very important")

**16) Files IO Basics**

**Ways to open a file**

* **“r” – read mode**
* **“w”- write mode**
* **“x” – It creates if file if the one you have mentioned, doesn’t exit**
* **“a”- append(adds more content to a file)**
* **“t” – text mode**
* **“b” – binary mode**
* **“+”- read and write**

**Read and text mode are default modes.**

**To open a file:**

**f=open(“name\_of\_file.txt”) -This will just open the portal/ access to the file**

**content = f.read()**

**print(content) - after the you can read all the contents of the file we opened.**

**ALWAYS REMEMBER TO CLOSE FILE**

**f.close() – for closing files**

f = open("harry.txt", "rt")

**rt is default mode while opening(read,text)**

content = f.read(**3**)

**That will read only first 3 characters of your file.**

print(content)

# content = f.read(34455)

# print("1", content)

#

# content = f.read(34455)

# print("2", content)

**For first content it will write 1 and then content, but for second incase all characters are covered under 34455 words, then it will only type 2 for second content.**

for line in content:

# print(line, end="")

**This helps iterating the content of the file, and each new sentence comes on a new line**

print(f.readline())

**This will help you print just one line of the file**

print(f.readlines())

**readlines makes each sentence in the file, an item in a list**

**17) File writing**

f = open("harry2.txt", "a")

a = f.write("Harry bhai bahut achhe hain\n")

# print(a)

**this allows you to add stuff to the file over and above what is already in the file.**

f = open("harry.txt", "w")

# a = f.write("Harry bhai bahut achhe hain\n")

# print(a)

# f.close()

**This is normal writing mode, where you write the file, but if there was anything gin the file earlier, it gets erased and only what you have written stays.**

f = open("harry2.txt", "r+")

print(f.read())

f.write("thank you")

**This is write mode along with reading.**

**We can read existing data, and add or append more stuff.**

**Append is only at the end, whereas r+ lets you add it wherever you want to.**

**print(f.readline ())**

**print(f.tell())**

**Tells us the character number at the beginning of the line it is under.**

**Eg if the f.readline is the first one, it will show 0**

**print(f.readline ())**

**f.seek(10)**

**it will read d=from the 10th character, after**

**18) Files using blocks**

**With open(“name\_of\_file”) as f:**

**a=f.readlines()**

**print(a)**

**19)Global Variable**

* **Basically a variable that has a value and is mentioned outside of a function.**
* **The variable value is the same inside as well as outside the function.**
* **UNLESS, the same variable has a different value within a function(local variable), then ONLY WITHIN the function, if you print it, it will show the local variable value.**
* **You cant print a local variable outside the fuction.**
* **You can change the value of a global variable within a function ONLY if you write global within the function, and then write whatever you want to do with the variable.**

**Global l**

**l=l+10**

**within a definition, only then will l change within the function**

* **Although in this case, both before and after, x will remain 20, this is because, even though global is mentioned, it is within a function. And is not recognized as a variable outside. It is still a local variable.**

x = 89

def harry():

x = 20

def rohan():

global x

x = 88

# print("before calling rohan()", x)

rohan()

print("after calling rohan()", x)

harry()

***20) ITERATIVE AND RECURSIVE APPROACH***

***LEFT TO DO***

**21) Lambda Functions/ Anonymous functions**

**It’s a lazy/ one liner method to write functions**

**So**

**minus = lamba(a,b) : a-b**

**print(minus(8,2))**

a =[[1, 14], [5, 6], [8,23]]

a.sort(key=lambda x:x[1])

print(a)

**--output—**

**The list a gets sorted in ascending order.**

**You can use lambda OR a function before the a.sort statement for the same result**

**def a\_first(a):**

**return a[1]**

**a.sort(key=a\_first)**

**print(a)**

**22) Modules**

import random  
blah1 = random.random()  
print(blah1\***100**)  
  
blah2 = random.randint(**2,300**)  
print(blah2)  
  
  
list\_of\_colours = ["Black"**,** "Grey"**,** "Blue"**,** "Green"**,** "Purple"**,** "White"**,** "pink"**,** "Red"**,** "Green"]  
colour = random.choice(list\_of\_colours)  
print(colour)

**It is very simple.**

**These are inbuilt.**

**For downloading your own modules in pycharm, go to terminal, type pipinstall, type in the one you want.**

**Then import the module you have downloaded.**

**23) F strings**

**Given below is one way of writing an f string**

a = "Trisha"  
b = **18**a=f"Hi, My name is {a}, and I am {b} years old"  
  
print(a)

**--output—**

**Hi, My name is Trisha, and I am 18 years old**

**24)Args and kwargs**

**Args is basically used in functions when there are way too many arguments to add, in between the code. It is ridiculous to keep adding arguments while writing a very long code. So args helps you add lists as arguments without adding new arguments while defining a list.**

def func1(\*args):  
 print(args)  
  
list1 = ["Red"**,** "Pink"**,** "Blue"**,** "Green"**,** "Yellow"**,** "White"]  
func1(\*list1)

**--output—**

**('Red', 'Pink', 'Blue', 'Green', 'Yellow', 'White')**

**It gets printed as a tuple (it is a default setting)**

def func1(\*args):  
 print(args[1])  
  
list1 = ["Red"**,** "Pink"**,** "Blue"**,** "Green"**,** "Yellow"**,** "White"]  
func1(\*list1)

**--output—**

**Pink**

* **Now if after defining the function, you write the elements of the function in a for loop, you get each item on a new line.**
* **Moreover, along with the args you can even add normal arguments while defining the fnction.**
* **Normal arguments HAVE to come before the args, while defining as well as while calling**

**FOR KWARGS:-**

def funargs(normal, \*argsrohan, \*\*kwargsbala):

print(normal)

for item in argsrohan:

print(item)

print("\nNow I would Like to introduce some of our heroes")

for key, value in kwargsbala.items():

print(f"{key} is a {value}")

har = ["Harry", "Rohan", "Skillf", "Hammad",

"Shivam", "The programmer"]

normal = "I am a normal Argument and the students are:"

kw = {"Rohan":"Monitor", "Harry":"Fitness Instructor",

"The Programmer": "Coordinator", "Shivam":"Cook"}

funargs(normal, \*har, \*\*kw)

**--output--**

**I am a normal Argument and the students are:**

**Harry**

**Rohan**

**Skillf**

**Hammad**

**Shivam**

**The programmer**

**Now I would Like to introduce some of our heroes**

**Rohan is a Monitor**

**Harry is a Fitness Instructor**

**The Programmer is a Coordinator**

**Shivam is a Cook**

**The only difference between args and kwargs is that args returns it as a tuple, and kwargs has key value format (so basically dictionary)**

**25) Importing time module**

import time

initial = time.time()

k = 0

while(k<45):

print("This is harry bhai")

time.sleep(2)

k+=1

print("While loop ran in", time.time() - initial, "Seconds")

initial2 =time.time()

for i in range(45):

print("This is harry bhai")

print("For loop ran in", time.time() - initial2, "Seconds")

**TIME.SLEEP() is an ADORABLE FUNCTION**

**If in the bracket you put 2, it will print what ever loop/etc you have written the code of after 2 seconds.**

**26) Virtual environment and requirements**

**It is useless for me because I have the latest version but it is used when you have lets say made a code in a particular version of python, but few years down the line someone wants that code. Now python has obviously upgraded, changed a lot of functions etc etc. so it helps in that situation.**

**Use video number 44 for that.**

**27)Enumerate Function**

**Incase you need to skip certain elements in a list, this will help you(not really imp, but if you know it and a q is asked in interview, it will look impressive)**

l1 = ["Bhindi", "Aloo", "chopsticks", "chowmein"]

i = 1

for item in l1:

if i%2 is not 0:

print(f"Jarvis please buy {item}")

i += 1

**instead of this, enumerator lets you do it in the following way:**

for index, item in enumerate(l1):

if index%2==0:

print(f"Jarvis please buy {item}")

**this is to print every alternate one.**

**So –output—(for both)**

**Bhindi**

**Chopsticks**

**28) Main**

**In case you have written a code in a particular file, and want some of those functions to be working in a nother file, you can import your own file in a new one.**

**BUT**

**Issue with this is, the program by default also runs what is there in the original file**

**So to avoid this you type main, right before whatever you want running, in your new file.**

def printhar(string):

return f"Ye string harry ko de de thakur {string}"

def add(num1, num2):

return num1 + num2 + 5

print("aand the name is", \_\_name\_\_)

if \_\_name\_\_ == '\_\_main\_\_':

print(printhar("Harry1"))

o = add(4, 6)

print(o)

**29)**

lis = ["John", "cena", "Randy", "orton",

"Sheamus", "khali", "jinder mahal"]

for item in lis:

print(item, "and", end=" ")

**--output—**

**John and cena and Randy and orton and Sheamus and khali and junder mahal and**

**30) Join function**

lis = ["John", "cena", "Randy", "orton",

"Sheamus", "khali", "jinder mahal"]

a = ", ".join(lis)

print(a, "other wwe superstars")

**--output—**

**John and cena and Randy and orton and Sheamus and khali and junder mahal and other wwe superstars**

**31) Map**

**It is a better way of writing a for loop**

**Map has 2 arguments : map(“What you want to do”,”what do you want to apply it on”)**

**Use map along with lamba(which is a short easy way of writing a function) to write short convenient codes**

**Eg:**

num = [2,3,5,6,76,3,3,2]

square = list(map(lambda x: x\*x, num))

print(square)

**a very cool eg of using lambda and map:**

# func = [square, cube]

# num = [2,3,5,6,76,3,3,2]

# for i in range(5):

# val = list(map(lambda x:x(i), func))

# print(val)

**32) Filter**

**It basically helps you filter out what you need into a list from a bigger list, using function.**

**It takes 2 arguments: function and iterable.**

list\_1 = [**1,2,3,4,5,6,7,8,9**]  
  
def is\_greater\_5(num):  
 return num>**5**gr\_than\_5 = tuple(filter(is\_greater\_5**,** list\_1))  
print(gr\_than\_5)

**33) Reduce**

**It takes in 2 arguments, a function and a list. Usually used for cumulative operators on a list.**

from functools import reduce

list1 = [1,2,3,4,2]

num = reduce(lambda x,y:x\*y, list1)

print(num)

**-----------------------------------------------------------------------------------------------------------------------------**

**\*\*\*In case of map, filter and reduce, we can use lambda instead of a function for convenience, and getting the code done in one line\*\*\***

***EXERCISE 7 IS LEFT TO DO***

**34) Decorators**

* + **We can delete a function by writing**

**Del function\_name**

* + **We can write one function as the output of another, and even if we delete the OG one, the second one will get printed.**

def func\_one():  
 print("Hello everyone!")  
  
func\_two = func\_one  
print(func\_two())

**--output—**

**Hello Everyone!**

* + **Apparently even if you delete func\_one, func\_two gets printed, I don’t know why.**
  + **You can write a function in a variable**

def funcret(num):

# if num==0:

# return print

# if num==1:

# return sum

# a = funcret(1)

# print(a)

# def executor(func):

# func("this")

#

#

# executor(print)

^^^

Just 2 random ways in which functions can be used

\*\*ask trisha decorator explaination, and show her the thing copy pasted below\*\*

def dec1(func1):

def nowexec():

print("Executing now")

func1()

print("Executed")

return nowexec

@dec1

def who\_is\_harry():

print("Harry is a good boy")

# who\_is\_harry = dec1(who\_is\_harry)

**who\_is\_harry()**

**@ dec one is just another way of writing : who\_is\_harry = dec1(who\_is\_harry)**

**35) Incase we don’t want anything to be shown in python, we write pass.**

**36) Class**

* **A class is basically a blueprint or ike an empty pamphlet with a structred way of pre determined into to be filled**
* **And object is a unique product that is produced after the class is filled with actual data.**

class Student:

pass

harry = Student()

larry = Student()

harry.name = "Harry"

harry.std = 12

harry.section = 1

larry.std = 9

larry.subjects = ["hindi", "physics"]

print(harry.section, larry.subjects)

* **This is just an example of class^^**
* **A class isn’t usually empty but this was to explain it and hence is pass is written, because it is empty.**
* **The class should have a capital letter at the beginning**

class Employee:

no\_of\_leaves = 8

pass

harry = Employee()

rohan = Employee()

harry.name = "Harry"

harry.salary = 455

harry.role = "Instructor"

rohan.name = "Rohan"

rohan.salary = 4554

rohan.role = "Student"

print(Employee.no\_of\_leaves)

print(Employee.\_\_dict\_\_)

Employee.no\_of\_leaves = 9

print(Employee.\_\_dict\_\_)

print(Employee.no\_of\_leaves)

* **Very new/ interesting concept^^ While he explained, I understood everything**
* **Rohan cant change value of no. of leaves for everyone, he can only change it for himself. Whereas if you change employee o. of leaves it changes for everyone.**
* **\_\_dict\_\_ is a new function, try it: It gives key and values in dictionary form of attributes such as rohan/ harry / or even employee from the above code.**
* **Know what instance variable is and class variable is.**
* **Instance variable in this case is harry.name, etc**
* **Class variable in this case is no\_of\_leaves**
* **Class variables can change the value of instance variables(if changed) but instance variables cannot change the value of class variables. It ends up making an additional instance variable for That instance f we try to change class variable using instance variable.**

**37) Using self as a function argument under class**

class Employee:  
 no\_of\_leaves = **8** def printdetails(self):  
 return f"The Name is {self.name}. Salary is {self.salary} and role is {self.role}"  
  
  
harry = Employee()  
rohan = Employee()  
  
harry.name = "Harry"  
harry.salary = **455**harry.role = "Instructor"  
  
rohan.name = "Rohan"  
rohan.salary = **4554**rohan.role = "Student"  
  
print(harry.printdetails())

**--output—**

**The Name is Harry. Salary is 455 and role is Instructor**

**38) Using init function under class**

**Init is a constructer**

class Employee:

no\_of\_leaves = 8

def \_\_init\_\_(self, aname, asalary, arole):

self.name = aname

self.salary = asalary

self.role = arole

def printdetails(self):

return f"The Name is {self.name}. Salary is {self.salary} and role is {self.role}"

harry = Employee("Harry", 255, "Instructor")

# rohan = Employee()

# harry.name = "Harry"

# harry.salary = 455

# harry.role = "Instructor"

#

# rohan.name = "Rohan"

# rohan.salary = 4554

# rohan.role = "Student"

print(harry.salary)

**--output—**

**455**

**So init function basically helps you put arguments as the values of instance variables when you put init in the class.**

**39)Class method can also be used instead of init**

**They both do pretty much the same thing**

**Although class method is used when there are no instance variables.**

**It directly talks about values and stuff for the class variables and their values, hence cls comes as an argument, instead of self.**

class Employee:

no\_of\_leaves = 8

def \_\_init\_\_(self, aname, asalary, arole):

self.name = aname

self.salary = asalary

self.role = arole

def printdetails(self):

return f"The Name is {self.name}. Salary is {self.salary} and role is {self.role}"

@classmethod

def change\_leaves(cls, newleaves):

cls.no\_of\_leaves = newleaves

@classmethod

def from\_dash(cls, string):

# params = string.split("-")

# print(params)

# return cls(params[0], params[1], params[2])

return cls(\*string.split("-"))

harry = Employee("Harry", 255, "Instructor")

rohan = Employee("Rohan", 455, "Student")

karan = Employee.from\_dash("Karan-480-Student")

print(karan.no\_of\_leaves)

# rohan.change\_leaves(34)

#

# print(harry.no\_of\_leaves)

**\*\*VIDEO 59 HAS STATIC VARIABLES WHICH ARE FOR NEITHER CLASS NOR INSTANCE\*\***

**You haven’t watched it so if it comes up, it is in this video.**

**40)Abstraction and Encapsulation**

**Abstraction are the various details that make up a ig component**

**Encapsulation is basically hiding all the abstraction layers and showing only the main thing**

**41)Single Inheritance**

**It is basically inheriting properties from one class to another**

**(video 61)- Single Inheritence**

**PRACTICE THIS!! It is tough, interesting, challenging and you took a lot of time to get it**

class Employee:

no\_of\_leaves = 8

def \_\_init\_\_(self, aname, asalary, arole):

self.name = aname

self.salary = asalary

self.role = arole

def printdetails(self):

return f"The Name is {self.name}. Salary is {self.salary} and role is {self.role}"

@classmethod

def change\_leaves(cls, newleaves):

cls.no\_of\_leaves = newleaves

@classmethod

def from\_dash(cls, string):

return cls(\*string.split("-"))

@staticmethod

def printgood(string):

print("This is good " + string)

class Programmer(Employee):

no\_of\_holiday = 56

def \_\_init\_\_(self, aname, asalary, arole, languages):

self.name = aname

self.salary = asalary

self.role = arole

self.languages = languages

def printprog(self):

return f"The Programmer's Name is {self.name}. Salary is {self.salary} and role is {self.role}.The languages are {self.languages}"

harry = Employee("Harry", 255, "Instructor")

rohan = Employee("Rohan", 455, "Student")

shubham = Programmer("Shubham", 555, "Programmer", ["python"])

karan = Programmer("Karan", 777, "Programmer", ["python", "Cpp"])

print(karan.no\_of\_holiday)

**42)Multiple Inheritence**

**Order of inheriting the classes is important. The first one should come first and second one second.**

**43) Hierarchy of inheriting variables of multiple classes.**

**If lets say the name variable is given 2 parent classes as well as the class we are working on, then first priority goes to the class that we are working on, then the FIRST class and then the second, third 4th so on.So for eg if we run the code below:**

class Employee:

no\_of\_leaves = 8

var = 8

def \_\_init\_\_(self, aname, asalary, arole):

self.name = aname

self.salary = asalary

self.role = arole

def printdetails(self):

return f"The Name is {self.name}. Salary is {self.salary} and role is {self.role}"

@classmethod

def change\_leaves(cls, newleaves):

cls.no\_of\_leaves = newleaves

@classmethod

def from\_dash(cls, string):

return cls(\*string.split("-"))

@staticmethod

def printgood(string):

print("This is good " + string)

class Player:

var = 9

no\_of\_games = 4

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

self.name = name

self.game =game

def printdetails(self):

return f"The Name is {self.name}. Game is {self.game}"

class CoolProgramer(Player, Employee):

language = "C++"

def printlanguage(self):

print(self.language)

harry = Employee("Harry", 255, "Instructor")

rohan = Employee("Rohan", 455, "Student")

shubham = Player("Shubham", ["Cricket"])

karan = CoolProgramer("Karan",["Cricket"])

# det = karan.printdetails()

# karan.printlanguage()

# print(det)

print(karan.var)

--output---

10

**(because coolProgrammer has no var value)**

**44) Multilevel Inheritence**

**If we take 3 levels of inheritance, dad, son and grandson. Then the hierarchy of importing is different as compared to that in mumtilevel. It will check the class we are working on(grandson), if there is nothing in that then the class right before that (son) and so on, where the last one will be the first class (dad)**

**42) Public, Protected, private variables**

**Public- Tose that anyone can access/ use, etc basically a normal Variable**

**Protected, You can use, maybe if you share it with fellow coders, they can use, but no one else can use it.**

**\_protected – is how you denote it (basically with a single underscore)**

**Private- only you can access this variable and that to by calling it along with its name, otherwise it shows an error.**

**\_\_private – is how you denote it(with 2underscores)**

**43)Polymorphism**

**Polymorphism is basically the input having variable outputs based on the code written**

**44)Over riding**

**So in this also there is a hierarchy.**

**If lets say we are printing a variable in b, then it looks for an INSTANCE variable in B. If that isn’t there, then it looks for an INSTANCE variable in A. If that isn’t there then it looks for a CLASS variable in B and last a CLASS variable in A.**

**In the code below,For over riding(point 44), ignore the super() part. It will be explained in the next point**

class A:

classvar1 = "I am a class variable in class A"

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

self.var1 = "I am inside class A's constructor"

self.classvar1 = "Instance var in class A"

self.special = "Special"

class B(A):

classvar1 = "I am in class B"

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

self.var1 = "I am inside class B's constructor"

self.classvar1 = "Instance var in class B"

# super().\_\_init\_\_()

# print(super().classvar1)

a = A()

b = B()

print(b.special, b.var1, b.classvar1)

**Once a clas is over ridden, the code will never read it or be able to print its variables again**

**(UNLESS WE USE SUPER, WHICH IS GIVEN BELOW)**

**45) super()**

**If let’s say we have overridden class A. and printed a variable using b.\*randomvariable\*. But this variable doesn’t exist in an override wala thing. Then to make class A(which was over ridden) actually visible, you have to write super()\_\_init\_\_ right under class B.**

**46) Function Over loader**

class Employee:

no\_of\_leaves = 8

def \_\_init\_\_(self, aname, asalary, arole):

self.name = aname

self.salary = asalary

self.role = arole

def printdetails(self):

return f"The Name is {self.name}. Salary is {self.salary} and role is {self.role}"

@classmethod

def change\_leaves(cls, newleaves):

cls.no\_of\_leaves = newleaves

def \_\_add\_\_(self, other):

return self.salary + other.salary

def \_\_truediv\_\_(self, other):

return self.salary / other.salary

emp1 =Employee("Harry", 345, "Programmer")

emp2 =Employee("Rohan", 55, "Cleaner")

print(emp1+emp2)

--output—

400

**47) Abstract Base Class**

**This is basically a method (just like init , etc). The role it plays is that which ever class you derive from this one will have to follow the rules/have certain functions that it sets.**

# from abc import ABCMeta, abstractmethod

from abc import ABC, abstractmethod

class Shape(ABC):

@abstractmethod

def printarea(self):

return 0

class Rectangle(Shape):

type = "Rectangle"

sides = 4

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

self.length = 6

self.breadth = 7

def printarea(self):

return self.length \* self.breadth

rect1 = Rectangle()

print(rect1.printarea())

**In this basically, the abstract class(Shape) is making it compulsory for class Rectangle(because it is derived from Shape) to have printarea as a function. Otherwise it will show an error.**

**48) setters and property decorators**

**Video 70 and 71 has miscellaneous information based on these. You do have an idea of how it works, but you won’t be able to do it on your own, so like watch this video if you ever need it**

**49) Generators**

**They are basically used to generate digits/ or which ever results ON THE SPOT. So basically they store nothing in your file/ laptop. These are used to save your storage. They generate stuff on the spot. All we save on the laptop is the method to generate, and then when we give it an input, the method generates an output on the spot.**

**50) Comprehensions**

**A comprehension is just an easy way to write a for loop along with a condition**

**So instead of writing**

# ls = []

# for i in range(100):

# if i%3==0:

# ls.append(i)

# print(ls)

**We can write:**

# ls = [i for i in range(100) if i%3==0]

#

# print(ls)

**That is the syntax for a comprehension**

**51) Dictionary comprehension**

**Dict1 = { i: f”item{i}” for I in range(100)}**

**Print(dict1)**

**You can add conditions to that^^ after the range, change beginning number, change the gaps between the numbers, etc**

**You can reverse dictionaries**

# dict1 = {i:f"Item {i}" for i in range(5)}

#

# dict2 = {value:key for key,value in dict1.items()}

# print(dict1,"\n", dict2)

**(dict2 being the revered form of dict1)**

**52) Generator comprehensions**

evens = (i for i in range(100) if i%2==0)

# print(evens.\_\_next\_\_())

# for item in evens:

# print(item)

**53) In short, list comprehension uses box bracket, set comprehension requires curly bracket and generator comprehensions require round/parenthesis bracket)**

**54) Using else in the for lppo**

khana = ["roti", "Sabzi", "chawal"]

for item in khana:

if item == "rotiroll":

break

else:

print("Your item was not found")

**try diff conditions in the for loop**

**55) Video 76 has caching**

**I SKIPPED IT**

**Watch it if it is needed**

**56)we have done try and except rarlier**

**Finally is a concept that goes with this only.**

**Try basically runs if it is applicable, except allows us to put an exception so it runs in case of the exception that we have provided, and finally runs irrespective of whether the first two run or only of them them runs or neither of them run.**

**You can write multiple exception statements in case of various types of erros that you want to display.**

**Else can also be a part of all these. It runs only if the except doesn’t run**

**57) Coroutine**

**It is important. It is basically used when the function/ generator requires some time to do the work, so you do sleep and give it a decent amount of time to work.**

**And then it runs**

**For eg in the code given below, you need time to actually search in the book, hence sleep(4) is given**

**Just like in case of files, where you have to close them, you need to close() the coroutine also. After you close your coroutine, it will show an error if you run it.**

def searcher():

import time

# Some 4 seconds time consuming task

book = "This is a book on harry and code with harry and good"

time.sleep(4)

while True:

text = (yield)

if text in book:

print("Your text is in the book")

else:

print("Text is not in the book")

search = searcher()

print("search started")

next(search)

print("Next method run")

search.send("harry")

search.close()

search.send("harry")

# input("press any key")

# search.send("harry and")

# input("press any key")

# search.send("thi si")

# input("press any key")

# search.send("joker")

# input("press any key")

# search.send("like this video")

**58) If after importing a module, you want to see everything you can do with it, then do**

**print(dir(\*\*name of module\*\*))**

**You can change cwd (current working directory.**

**It is the place that python thinks you are in. If you change it, and search for a file in the current location, it will throw an error.**

import os

# print(dir(os))

# print(os.getcwd())

# os.chdir("C://")

# print(os.getcwd())

# f = open("harry.txt")

# print(os.listdir("C://"))

# os.makedirs("This/that")

# os.rename("harry.txt", "codewithharry.txt")

# print(os.environ.get('Path'))

# print(os.path.join("C:/", "/harry.txt"))

# print(os.path.exists("C://Program Files2"))

print(os.path.isfile("C://Program Files"))

**These are a few of the Os module functions. They are easy**

**59) Request Modules**

**60) Java script object notation(json)**

**61) Regular expression Module**

**62) Converting .py to .exe is an important video**