Day 3:

**Exception handling:**

Syntax errors are the most common error that we encounter most of the time. The syntax errors are shown by a small arrow where error has occurred. But these kind of errors are mostly take care during the development stage.

Sometimes the entire program might be syntactically correct even then we may encounter errors . It can be of any type of error. We can get to know the error in the error messages.

The problem above is that we aren't considering a way to handle these kind of errors. If you're building a application you don't want the user to see the error message shown in our program, so we need to come up a solution these situations -(hence the name error handling).

The error handling is mostly done with the "try" and "except" statements.

>>>while True:

>>> try:

>>> st = int(input("Please enter a number"))

>>> break

>>> except:

>>> print("Invalid input")

This program executes until a valid integer/number is given. Here you can see the try and except blocks, the try block works fine until some error occurs inside it. Suppose you provide a string in input (which for sure causes error) then it goes inside the except block and executes those instructions.

Make sure to be simple and concise in the except block because you don't want errors to happen there. If there is try block then there should be a except or finally block.

Here in the except block we didn't specify any type of error. We can also do that like this >>> except ValueError: ->this except only handles value errors not other errors

There are many built in exceptions you can refer them here: <https://docs.python.org/3/library/exceptions.html#bltin-exceptions>

A try statement can have multiple except blocks for handling different exceptions

Eg:

>>>try:

>>> number = int(input("Enter userId: "))

>>> name = input("Enter name: ")

>>> generatedName = name[1] + str(number)

>>> print(generatedName)

>>>except ValueError:

>>> print("User id incorrect format")

>>>except IndexError:

>>> print("Error name should be at least 2 characters long")

You can also have a common except with tuple of built-in exceptions

>>>except (ValueError,IndexError) :

Suppose you want to stop the execution of the program when it goes to except then you can give >>>sys.exit() for this you may also need >>>import sys

Since you're seeing import for the first time, it means python gets access to another files or folder similar to what we see in other programming languages as header files.

>>>import numpy as np #here the as keyword is used to alias the module name as np

>>>x = np.array(arr.shape) #using np we can call numpy functions

You can also have a else part after except block , this else will execute if try does not cause any errors.

**raise keyword:**

The raise statement forces the exception to occur, and quits the program execution.

>>>try:

>>> file = open("file.txt")

>>>except:

>>> print("Cannot open file")

>>> raise IOError

Here the raise can also be given without parameters and python is intelligent enough to find the type of error.

finally statement:

If you want some logic to happen always whether or not the error occurs then you can you use the finally statement.

>>>try:

>>> name = input("Enter your first name")

>>> age = int(input("Enter your age"))

>>>except ValueError:

>>> print("Input format not proper")

>>>finally:

>>> print("Thanks for your response") #statement is executed irrespective of errors

**File IO:**

Like every other programming, python also supports file read and write functions

The open function is the one we normally use for file reading and writing. The open function open(filename,mode) has these parameters where you can mode to be 'w' or 'r' or 'a' each with a specific role.

open('filename.txt','w') - for writing into the file, it generally erases the contents already present in the file and writes the contents

open('filename.txt','r') - for reading from the files

open('filename.txt','a') - append to already existing files

If mode parameter is not specified then it considers 'r' by default. Files are normally opened in text mode. If you have a file that is not text then you can open your file in binary mode open('filename','rb') -> rb specifies read byte.

Normally the open command reads till the end of file, suppose you want to read one line then you can use fileobject.readline() .

Another important thing , whenever we open a file it's always advised to close it at the end. Otherwise the file may not be the same after writing or it won't be completely saved in disk.

>>>f = open("filename.txt","w")

>>>f.close() #closing the file for proper saving

There is also another way to achieve this with the use of the "with" command

>>>with open("filename") as f:

>>> fileData = f.read()

The with keyword is considered as a good practice when dealing with file objects because even if a exception occurs when reading/writing the file , the file is properly closed. You also don't need to specify the close() command as the with keyword also takes care of that

If you want to loop through each line in a file

>>>with open("filename") as f:

>>> for line in f: #line corresponds to each line in file occurs until EOF

>>> print(line)

**Notable point:**

* Whenever you're writing inside a file make sure it's a string object or byte object
* You can also serialize and deserialize python data structures using the json module (JSON is used in modern applications for data exchange).Learn more about it : <https://docs.python.org/3/library/json.html>

**Classes and functions:**

Before jumping into classes, we need to know about namespaces and scopes. Namespace is a mapping from names to objects. The built-in functions like abs() are all present inside the global namespace. Namespaces are implemented as python dictionaries.

Consider having two namespaces but each one has the same variable/function method but that does not affect the contents inside another namespace. Suppose you need to use variables or functions from another file(indicating it's in different namespace) you can use import function as discussed above.

Namespaces are created at different times and have a different lifetimes. All the built-in variables/methods(abs,max,sum etc.) are present in a global namespace which is created when python interpreter starts and is never deleted.

For understanding the scope part I think it's better to a see an example

>>>def testing\_scope():

>>> def func():

>>> a = "locally changed" #unable to change a since a assignment here is local to function func()

>>> def func1():

>>> nonlocal a #nonlocal keyword changes the value of a

>>> a = "non locally changed"

>>> def func2():

>>> global a #global keyword changes a globally

>>> a = "globally changed"

>>> a = "unchanged"

>>> func()

>>> print(a)

>>> func1()

>>> print(a)

>>> func2()

>>> print(a)

>>>a = "before Change"

>>>print(a)

>>>testing\_scope()

>>>print(a) #if the function definitions are not understandable that's fine we'll cover it

The above will output:

before Change

unchanged

non locally changed

non locally changed

globally changed

Classes provide the functionality of bundling data and functions. After creating a class with class members and functions we can create object to use them. We can create many objects which are different instances of the same class (promoting reusability) . A object is nothing but a instance of class with unique address(like pointer to a class) . You can use the object to access the class properties (variables/methods)

Enough said we can go to code:

**class syntax:**

>>>class carClass: #class keyword for declaring a class

>>> car\_type = "four wheeler" #this variable is considered global inside carClass

>>> def findCarType(self): #member functions inside the class

>>> print("output from function")

>>>obj = carClass() #creates a object to carClass

>>>print(obj.car\_type)

>>>print(obj.findCarType())

output of above code:

four wheeler

output from function

Here we can encounter many new terms like self and def. Let's see what they are

**def:** This keyword denotes a function followed by function name. Also notice that there is no return type for a function so it can be anything like string,char,int,void etc.

Python functions can have a return statement also. return is a statement used to exit a function. Another unique functionalities of python functions is that you can have a function inside another function (*refer the scope code for example)*

Simple function to two numbers

>>>def add(x,y)

>>> return x + y

>>>print(add(5,10))

output:

15

The functions can also have two parameters but you can send only one like this

>>>def add(x,y=None):

>>> if y is not None:

>>> return x+y

>>> return x

>>>print(add(5)) #prints 5

>>>print(add(5,10)) #prints 10

You can pass list/tuples as parameters to functions like \*args in function similarly for dictionary we use \*\*kwargs

>>>def func(\*args):

>>> for i in args:

>>> print(i)

>>>def func1(\*\*kwargs):

>>> for k,v in kwargs.items()

>>> print(k,v)

You can assign a variable to a function and even use is else where

eg:

>>>def fun():

>>> print("OK")

>>>x = func

>>>x() #call func using x

**self:** self parameter refers to the object itself. Whenever we're calling a function like *obj.findCarType()* this gets converted internally in python to

*carClass. findCarType(obj)* so we are declaring a self parameter in a function so that we can access the function using the object.

Suppose a method does not have a self parameter then we can't access that function because we won't be passing the object parameter.

**\_\_init\_\_ method:** This is considered as the constructor in python. Whenever an object is created for a class the \_\_init\_\_ method is called. There can also be parameters for this function/method for more flexibility.

>>>class carClass:

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

>>> self.const\_var = 10

>>> def subtract(self,x,y):

>>> return x + y - self.const\_var

>>>obj = carClass() #const\_var is set when creation of object

>>>print(obj.subtract(20,30)) #20 + 30 - 10

**pass** is a keyword used to return nothing from a method and pass can be used to create empty class

>>>class myClass:

>>> pass

**Inheritance:**

Situations may arise where we need one class properties in another class, then we go for a concept called inheritance. Inheritance is mainly preferred where code reusability is needed. If a class B inherits class A, then all methods of class B can access class A member functions and member variables.

eg:

>>>class baseClass:

>>> def add(self,x,y):

>>> return x+y

>>> def subtract(self,x,y):

>>> return x-y

>>>class derivedClass(baseClass): #the parentheses used for inheritance

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

>>> self.promotion\_rate = 5

>>> def promoted(self,x):

>>> return baseClass .add(self,self.promotion\_rate,x) #calling baseClass method from derived class

>>> def demotion(self,x):

>>> return baseClass .subtract(self,self.promotion\_rate,x)

You can also have multi-level inheritance like specifying different class names separated by commas

>>>class derivedClass(baseClass,baseClass1):

**Assignment for Day 3:**

1)Given a text file write a program to read each line in the file and add "--" between each line and write it to another text file.

refer/use this text: <https://github.com/DeepakVelmurugan/pythonBeginnerCourse/blob/main/Day3Assignmenttxt>

2)Write a python program which gets the name and password from the user and checks if the password is greater than six characters else prompt user to try again. Implement error handling where ever needed.

3)Create a class Appraisal which holds a constructor , bonus method and newCTC method. The employees are subject to 10 & 20% gain in oldCTC depending on performance.

* If employees have n>3 they have 20% gain in oldCTC + bonus which is constant value of 10,000 where n is the rating of the employee.

newCTC = 20% of oldCTC + oldCTC + 10000

* If n<=3 then they have 10% gain in oldCTC

newCTC = 10% of oldCTC + oldCTC

Now if n is given for a employee calculate his/her newCTC

*where ctc denotes cost to company*