For Loop with Else

Else can be used in while and for loop.

The statements in the else block will be executed after all iterations are completed.

for i in range(5):

    print(i)

else:

    print('Loop completed')

now what if I use break statement on certain condition what will happen ?

for i in range(5):

    print(i)

    if i ==4:

        break

else:

    print('Loop completed')

now else statement is not going to be executed because loop is not completed successfully it got break out of that loop so it didn’t completed all the iterations.

But if I use continue statement, the else block will execute

for i in range(5):

    if i ==4:

        continue

    print(i)

else:

    print('Loop completed')

else statement ke khtm hone ka mtlb hi yahi hai ke loop break nh huwa, loop khtm huwa hai.

Same goes for while loop:

i = 0

while i < 7:

    print(i)

    i += 1

    if i==5:

        break

else:

    print("Done!")

Exception Handling

We use error handling when we don’t want our program to get terminate, that’s why we use error handling.

Try-except code:

a = input("Enter a number: ")

print("Multiplication table of", a)

try:

    for i in range(1, 11):

        print(a, "x", i, "=", int(a)\*i)

except:

    print("Invalid Input.")

print("Some lines of code.")

print("important lines of code.")

print("End of program")

this code demonstrates if user enter some invalid input like “harry” then print Invalid Input and run all below lines of code because that’s what error handling is like we don’t want our other code to stop running.

First the control will move to try if it throws an error then the control will went to except.

We can also handle the specific type of error like a valueerror:

try:

    a = int(input("Enter a number: "))

    print("Multiplication table of", a)

    for i in range(1, 11):

        print(a, "x", i, "=", int(a)\*i)

except ValueError:

    print("Invalid Input.")

we can also use multiple excepts like below:

try:

    a = int(input("Enter a number: "))

    b = [2,3]

    print(b[a])

except ValueError:

    print("Invalid Input. Input data must be a number.")

except IndexError:

    print("Invalid Index.")

print("Some lines of code.")

print("important lines of code.")

print("End of program")

handling nameerror:

try:

    print(a)

except NameError:

    print("Variable a is not defined")

just like this we can handle multiple different errors.

Finally Keyword in python

Finally clause are used for cleanups.

try:

    l = [1, 2, 3]

    i = int(input('Enter an index: '))

    print(l[i])

except:

        print("Some error occurred")

finally:

    print("Finally block always executed")

now you might wanna question ke zain finally block likhne ka mtlb kia huwa , hn smjhgiya ke beshak try or except execute ho ya na ho finally hamesha hoga execute, isko hum simple bhi toh likhskte print statement why should I use finally clause?

The usability of finally clause can be seen when we use it inside a function like below:

def func():

    try:

        l = [1, 2, 3]

        i = int(input('Enter an index: '))

        print(l[i])

        return 1

    except:

        print("Some error occurred")

        return 0

  print("Im always executed.")

x = func()

print(x)

now If I execute this “im always executed is not going to run because above it , we are returning a value from a function that’s why it is not executing so we need to use finally clause here to actually run that “Im always executed”. Like this

def func():

    try:

        l = [1, 2, 3]

        i = int(input('Enter an index: '))

        print(l[i])

        return 1

    except:

        print("Some error occurred")

        return 0

    finally:

        print("Im always executed.")

x = func()

print(x)

now despite the fact that function returns a value, the finally clause will be executed no matter what happened, the function is returning a value but finally doesn’t care he be like “do what you can do, I will execute anyway” now this can be used for closing some file resources, closing a database connection or maybe ending the program/function with a delightful message.

Custom Errors

When you want to raise custom errors on your specific defined actions.

We use this take program ruk jaye on the basis of our custom errors.

a = int(input("Enter any value between 5 and 9: "))

if a < 5 or a>9:

    raise ValueError("Value should be between 5 and 9.")

Just like above program, if a user don’t enter value between 5 and 9, user might see an error.

You want to do this because you sometimes want to stop a program if user enters a wrong instruction/data so you might wanna stop the program right there, so you can raise your custom errors. You wanna stop the user take us ghalti ki wajah se line 12 pr na mistake dikhe agr line 5 pr ghalat instruction user ne dedi toh, agr dedi toh usko wahin rok dia jaye.

Challenge: If user write “quit” exit the program.

a = input("Enter any value between 5 and 9: ")

if a=='quit':

    exit()

try:

    a = int(a)

except ValueError:

    print("Please enter a valid integer.")

    exit()

if a < 5 or a>9:

    raise ValueError("Value should be between 5 and 9.")

Now what are custom exceptions? Phele classes prhkr aao phr dekhna isko!

Short hand if else

a = 2302

b = 230

# one if, else

print(a) if a>b else print(b)

# two if, else

print(a) if a>b else print(b) if a<b else print("Both are equal")

# expression handling if, else

c = 8 if a>b else 9 if a==b else 11

print(c)

Enumerate Functions

Fun fact: linting is a process for identifying bugs and stylistic errors in your code.

Without using enumerate:

marks = [12,56,22,99,12]

index = 0

for mark in marks:

    print(mark)

    if index==3:

        print("Zain is awesome in dld.")

    index+=1

without using enumerate when I wanted to access the index, this is what im doing is writing index out of the for loop and then keeps on updating it or there is another method like below you might wanna question ke bhai simply range use krlo,

marks = [12,56,22,99,12]

for i in range(len(marks)):

    print(marks[i])

    if i==3:

        print("Zain is awesome in dld.")

this is also correct but you have to add square brackets as an extra effort to it to get the index and trust me programmers are way too lazy for that that’s why they introduced enumerate function.

marks = [12,56,22,99,12]

for index,mark in enumerate(marks):

    print(mark)

    if index==3:

        print("zain is awsome in dld.")

see how simple is that by just using the enumerate function you can do the following without updating the index or using square brackets. ☺

if I directly use this below,

marks = [12,56,22,99,12]

for mark in enumerate(marks):

    print(mark)

it will give mark output as tuple in which there are index and element. Like (0,12) and when I used index,marks it unpacked that tuple basically.

Change the start index:

By default index starts from zero, but we can do modify the index point. Like it was starting from 0 to every element it was starting from zero like below:

for index,mark in enumerate(marks):

    print(index,mark)

# 0 12

# 1 56

# 2 22

# 3 99

# 4 12

See how first element got index 0, but lets say I wanna modify that , what can I do the following:

for index,mark in enumerate(marks,start=21):

    print(index,mark)

# 21 12

# 22 56

# 23 22

# 24 99

# 25 12

See how I just modified the index value and now I can use conditions on the basis of that.

Virtual Environments in Python

A tool to isolate specific python environments on a single machine.

Before we needed to install the venv using pip install venv but now its just built in.

# create a virtual environment

Python –m **venv** virtual\_env\_name

# activating environment on windows

.\environment\_name\Scripts\activate

Fun fact: ctrl + alt + 1 will shift to first powershell tab , then 2 will shift to second tab. Hahaha!

# deactivate an environment

deactivate

the requirements.txt file:

pip freeze # list out all the packages in a current environment

pip freeze > requirements.txt # store all packages name and version to a requirements.txt file

pip install –r requirements.txt # it will install all the packages that are available in requirements.txt

now before that I also know how to use conda and in miniconda you can create environments with different python versions but in this I wanted to know how can we create environments with different python versions so yes it is possible with venv but you need to install python version lets say on my computer there is already a python==3.13 but I want to create venv with python==3.11 so I need to install 3.11 from internet then use power shell navigate to that python version and with that I can use my command like for example: **c> /python3.11/python.exe –m venv myvenv**

Imports

Important concepts

# when you write just import math, you have to use math. prefix to use the function

# import math

# print(math.sqrt(16))

# when you write from math import \*, you can use the function directly without math. prefix

# from math import \*

# print(sqrt(16))

# print(f'{pi:.2f}')

# when you write from math import sqrt, you can use only sqrt function

# from math import sqrt

# print(sqrt(16))

Alias in imports “as”

from math import sqrt as s # Importing sqrt function from math module and renaming it as s

print(s(25)) # Calling sqrt function using s

dir method:

import math

print(dir(math)) # we can use dir() to see the list of functions in the module

importing functions from other python file:

**zain.py**

def welcome():

    print("Welcome to Zain's module")

**main.py**

import zain

zain.welcome()

similaryly if I store a variable in zain I can import it zain.variable name.

if name==main concept

main.py

import zain

zain.welcome()

zain.py

def welcome():

    print("Welcome to Zain's module")

if \_\_name\_\_ == "\_\_main\_\_":

    welcome()

if I use welcome() outside the name==main then when I import this zain to any script, the welcome function will execute automatically , so in zain.py im checking first that if name==main then execute welcome, if im importing this zain.py then don’t execute this welcome that is invoked in zain.py file.

45