Q.1 implement a nested function that calculates compound interest using non-local variables.

def compound\_interest(principal, rate, time):

    def calculate():

        nonlocal principal

        principal = principal \* (1 + rate / 100) \*\* time

        return principal

    return calculate()

# Example usage

principal\_amount = 1000

annual\_rate = 5

time\_period = 2

final\_amount = compound\_interest(principal\_amount, annual\_rate, time\_period)

print(f"The final amount after {time\_period} years is: {final\_amount}")

OUTPUT:

|  |
| --- |
|  |

Q.2 write a program using lambda to filter even numbers from a list.

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

even\_numbers = list(filter(lambda x: x % 2 == 0, numbers))

print("Even numbers:", even\_numbers)

OUTPUT:



Q.3 write a progrme using lambda to find area of triangle.

base = 10

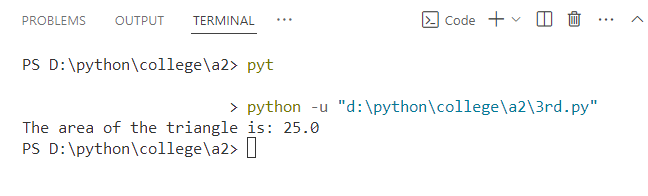
height = 5

area\_of\_t;riangle = lambda b, h: 0.5 \* b \* h

area = area\_of\_triangle(base, height)

print(f"The area of the triangle is: {area}")

OUTPUT:



Q.4 create a generator function to yeild fibonacci numbers upto a certain limit.

def fibonacci(limit):

a, b = 0, 1

while a <= limit:

yield a

a, b = b, a + b

# Example usage

limit = 100 # Set the limit for Fibonacci numbers

for number in fibonacci(limit):

print(number,"\t", end="")

OUTPUT:

A screenshot of a computer code

Description automatically generated

Q.5 develop a custom python module with functions for basic statistical operations (mean,median,mode)

# statistics\_module.py

def mean(numbers):

return sum(numbers) / len(numbers)

def median(numbers):

sorted\_numbers = sorted(numbers)

n = len(numbers)

mid = n // 2

if n % 2 == 0:

return (sorted\_numbers[mid - 1] + sorted\_numbers[mid]) / 2

else:

return sorted\_numbers[mid]

def mode(numbers):

frequency = {}

for number in numbers:

frequency[number] = frequency.get(number, 0) + 1

max\_frequency = max(frequency.values())

modes = [key for key, value in frequency.items() if value == max\_frequency]

if len(modes) == 1:

return modes[0]

return modes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# main.py

import statistics\_module as stats

numbers = [1, 2, 2, 3, 4, 4, 4, 5, 6]

mean\_value = stats.mean(numbers)

median\_value = stats.median(numbers)

mode\_value = stats.mode(numbers)

print(f"Mean: {mean\_value}")

print(f"Median: {median\_value}")

print(f"Mode: {mode\_value}")

OUTPUT:

A screen shot of a computer code

Description automatically generated

Q.6 Write a program to handle file reading errors and log them to a separate file.

import logging

logging.basicConfig(filename='error\_log.txt', level=logging.ERROR, format='%(asctime)s - %(levelname)s - %(message)s')

def read\_file(file\_path):

try:

with open(file\_path, 'r') as file:

content = file.read()

return content

except FileNotFoundError:

logging.error(f"File not found: {file\_path}")

return None

except IOError:

logging.error(f"IO error occurred while reading the file: {file\_path}")

return None

# Example usage

file\_path = 'example.txt'

file\_content = read\_file(file\_path)

if file\_content:

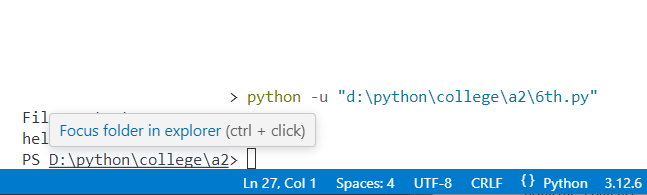
print("File content:")

print(file\_content)

else:

print("An error occurred. Check the error\_log.txt file for details.")

OUTPUT:



Q.7 create a custom exception class AgeException and use it to validate user input.

# Define the custom exception class

class AgeException(Exception):

def \_\_init\_\_(self, message="Invalid age provided"):

self.message = message

super().\_\_init\_\_(self.message)

# Function to validate age

def validate\_age(age):

if age < 0 or age > 120:

raise AgeException(f"Age {age} is not valid. Age must be between 0 and 120.")

# Example usage

try:

user\_age = int(input("Enter your age: "))

validate\_age(user\_age)

print(f"Your age is {user\_age}.")

except AgeException as e:

print(e)

except ValueError:

print("Please enter a valid integer for age.")

OUTPUT:

A screenshot of a computer

Description automatically generated