1. Write a Python function named calculate\_area that takes the radius of a circle as a parameter and returns the area of the circle. Call the function with a radius of 5 and print the result.

Answer:

import math

def calculate\_area(radius):

return math.pi \* radius \*\* 2

# Calling the function with radius 5

area = calculate\_area(5)

print("Area of circle with radius 5:", area)

2. Differentiate between built-in functions and user-defined functions in Python. Provide examples of each.

Answer:

* Built-in Functions: These are functions provided by Python itself.

Example: len(), print(), max()

print(len("Hello")) # Built-in function

* User-defined Functions: These are functions created by users to perform specific tasks.

Example:

def greet():

print("Hello, User!")

greet() # User-defined function

3. Create a function called add\_numbers that takes variable arguments and returns their sum. Call the function with different numbers of arguments (e.g., 2, 3, 5) and print the results.

def add\_numbers(\*args):

return sum(args)

print(add\_numbers(2, 3))

print(add\_numbers(5, 10, 15))

print(add\_numbers(1, 2, 3, 4, 5))

4. Define an anonymous function (lambda function) that squares a given number. Use the lambda function to square the numbers 1 to 5 and print the results.

square = lambda x: x \*\* 2

for i in range(1, 6):

print(f"Square of {i} is {square(i)}")

5. Explain the concepts of global and local variables in Python. Write a function that uses both global and local variables, demonstrating their scopes.

Answer:

* Global Variable: Declared outside all functions, accessible throughout the program.
* Local Variable: Declared inside a function, accessible only within that function.

x = 10 # Global variable

def my\_function():

y = 5 # Local variable

print("Inside function: x =", x)

print("Inside function: y =", y)

my\_function()

print("Outside function: x =", x)

6. Why is exception handling important? Explain with a suitable example.

Answer:

Exception handling is important to catch errors during program execution and handle them gracefully without crashing the program.

try:

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

print("You entered:", number)

except ValueError:

print("Invalid input! Please enter a valid number.")

7. Write a Python script that attempts to open a non-existent file. Use the try and except clauses to handle the FileNotFoundError and print a custom error message.

try:

file = open("non\_existent\_file.txt", "r")

except FileNotFoundError:

print("Error: The file you are trying to open does not exist.")

8. Explain the purpose of the finally clause in exception handling. Provide an example scenario where the finally clause is useful.

Answer:

The finally clause is used to execute code that must run regardless of whether an exception was raised or not. Useful for cleanup tasks like closing files.

try:

file = open("example.txt", "r")

data = file.read()

except FileNotFoundError:

print("File not found.")

finally:

print("This block always runs.")

9. Create a Python program that defines a custom exception called NegativeValueError. Write a function that accepts a number as a parameter and raises this exception if the number is negative.

class NegativeValueError(Exception):

pass

def check\_positive(number):

if number < 0:

raise NegativeValueError("Negative value is not allowed.")

else:

print("Number is positive:", number)

# Example usage

try:

check\_positive(-5)

except NegativeValueError as e:

print(e)

10. Design a Python function that takes two parameters (numerator and denominator) and calculates the quotient. Handle potential division by zero exceptions using a custom exception. Test the function with different inputs.

class DivisionByZeroError(Exception):

pass

def divide(numerator, denominator):

if denominator == 0:

raise DivisionByZeroError("Denominator cannot be zero.")

return numerator / denominator

# Test cases

try:

print(divide(10, 2))

print(divide(5, 0))

except DivisionByZeroError as e:

print(e)

11. Write a recursive function to generate the first n terms of the Fibonacci sequence.

def fibonacci(n):

if n <= 1:

return n

return fibonacci(n-1) + fibonacci(n-2)

n\_terms = 10

print("Fibonacci sequence:")

for i in range(n\_terms):

print(fibonacci(i), end=" ")

12. Define a function named greet\_user that takes two parameters, name and greeting. If greeting is not provided, the function should default to “Hello.” The function should then print the greeting followed by the name.

def greet\_user(name, greeting="Hello"):

print(f"{greeting}, {name}!")

greet\_user("Alice")

greet\_user("Bob", "Good morning")