**Question 1**

def classify\_number(number):

if number > 0:

return "positive"

elf number < 0:

return "negative"

Else:

return "zero"

while True:

try:

user\_input = input ("Enter an integer: ")

number = int(user\_input)

result = classify\_number(number)

print(f"The number is {result}.")

break

except ValueError:

print("Invalid input. Please enter a valid integer.")

**Question 2.**

def calculate\_average(\*args):

"""

Calculate the average of a variable number of numbers.

Args:

\*args: A variable number of numeric arguments.

Returns:

The average of the input numbers.

Raises:

TypeError: If any of the input arguments are not numbers.

ZeroDivisionError: If no arguments are provided.

raise ZeroDivisionError("Cannot calculate average of zero numbers")

# Calculate the average

average = sum(args) / len(args)

return average

**Question 3.**

def calculate\_average(\*args):

"""

Calculate the average of a variable number of numbers.

Args:

\*args: A variable number of numeric arguments.

Returns:

float: The average of the input numbers.

Raises:

TypeError: If any of the input arguments are not numbers.

ZeroDivisionError: If no arguments are provided.

if not all(isinstance(arg, (int, float)) for arg in args):

raise TypeError("All arguments must be numbers")

if len(args) == 0:

raise ZeroDivisionError("Cannot calculate average of zero numbers")

return sum(args) / len(args)

**Question 4.**

def get\_user\_input():

while True:

try:

# Attempt to convert the user's input to a float

num = float(input("Please enter a number: "))

return num

except ValueError:

# If the input cannot be converted to a float, print an error message

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

def main():

num = get\_user\_input()

print(f"You entered: {num}")

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

main()

```

**Question 5.**

def write\_names\_to\_file(names, filename):

try:

# Use the with statement to open the file in write mode

with open(filename, 'w') as file:

# Write each name to the file on a new line

for name in names:

file.write(name + '\n')

print(f"Names written to {filename} successfully.")

except Exception as e:

print(f"An error occurred: {e}")

def read\_names\_from\_file(filename):

try:

# Use the with statement to open the file in read mode

with open(filename, 'r') as file:

# Read each line from the file and print it

for line in file:

print(line.strip())

except Exception as e:

print(f"An error occurred: {e}")

def main():

names = ["John", "Alice", "Bob", "Eve"]

filename = "names.txt"

write\_names\_to\_file(names, filename)

print("\nNames in the file:")

read\_names\_from\_file(filename)

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

main()

**Question 6**

names = ["John", "Alice", "Bob", "Eve"]

# Write names to the file

with open("names.txt", "w") as file:

for name in names:

file.write(name + "\n")

# Read names from the file and print them

print("Names in the file:")

with open("names.txt", "r") as file:

for line in file:

print(line.strip())

```

**Question 7**

celsius\_temps = [0, 10, 20, 30, 40]

# Lambda function to convert Celsius to Fahrenheit

convert\_to\_fahrenheit = lambda c: c \* 9/5 + 32

# Use map function to apply the lambda function to each temperature

fahrenheit\_temps = list(map(convert\_to\_fahrenheit, celsius\_temps))

# Print the converted list

print("Celsius Temperatures:", celsius\_temps)

print("Fahrenheit Temperatures:", fahrenheit\_temps)

```

# Explanation

- We define a sample list of Celsius temperatures `celsius\_temps`.

Celsius Temperatures: [0, 10, 20, 30, 40]

Fahrenheit Temperatures: [32.0, 50.0, 68.0, 86.0, 104.0]

```

# Benefits of Using Lambda Functions and Map

- Lambda functions provide a concise way to define small, one-time-use functions.

- The `map` function applies a function to each element of an iterable, making it easy to perform transformations on lists.

- This approach is often more readable and efficient than using loops or list comprehensions. ‎<This message was edited>

Here's a Python function that handles division by zero and invalid input types:

**Question 8**

def divide\_numbers(numerator, denominator):

try:

# Attempt to perform division

result = numerator / denominator

return result

except ZeroDivisionError:

# Handle division by zero

print("Error: Division by zero is not allowed.")

except TypeError:

# Handle invalid input types

print("Error: Both numerator and denominator must be numbers.")

def check\_positive(num):

if num < 0:

raise NegativeNumberError("Number cannot be negative")

else:

print("Number is positive")

try:

num = -5

check\_positive(num)

except NegativeNumberError as e:

print(f"Error: {e}")

```

**Question 9**

import random

def generate\_random\_numbers(n, min\_val, max\_val):

"""Generate a list of n random integers between min\_val and max\_val."""

return [random.randint(min\_val, max\_val) for \_ in range(n)]

def calculate\_average(numbers):

"""Calculate the average of a list of numbers."""

return sum(numbers) / len(numbers)

def main():

# Generate a list of 10 random integers between 1 and 100

numbers = generate\_random\_numbers(10, 1, 100)

print("Generated Numbers:", numbers)

# Calculate and print the average

average = calculate\_average(numbers)

print("Average:", average)

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

main()

**Question 10**

import re

def extract\_emails(text):

"""Extract all email addresses from a given text."""

pattern = r'\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

return re.findall(pattern, text)

def validate\_date(date\_str):

"""Validate a date in the format 'YYYY-MM-DD'."""

pattern = r'^\d{4}-\d{2}-\d{2}$'

return bool(re.match(pattern, date\_str))

def replace\_word(text, old\_word, new\_word):

"""Replace all occurrences of a word with another word in a string."""

return re.sub(old\_word, new\_word, text)

def split\_string(text):

def main():

text = "Contact me at john.doe@example.com or jane.smith@example.com. The meeting is on 2024-03-16."

print("Extracted Emails:", extract\_emails(text))

date\_str = "2024-03-16"

print("Is Date Valid?", validate\_date(date\_str))

old\_word = "old"

new\_word = "new"

text = "This is an old car."

print("Replaced Text:", replace\_word(text, old\_word, new\_word))

text = "Hello, World! How's it going?"

print("Split Text:", split\_string(text))

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

main()

```