# Python assignment 17th Sept FSDS Pro

September 21, 2023

[1]:
```
#Map

#1. Explain the purpose of the map() function in Python and provide an example␣
 ↪of how it can be used to apply a function to each element of an iterable.

#The map() function in Python is used to apply a specified function to each␣
 ↪item in an iterable (e.g., a list, tuple, or other iterable) and returns an␣
 ↪iterable containing the results. It is a built-in function that simplifies␣
 ↪the process of applying a function to all elements of an iterable without␣
 ↪the need for explicit loops.

#Example of using map() to apply a function to a list of numbers:

def square(x):
    return x ** 2

numbers = [1, 2, 3, 4, 5]
squared_numbers = map(square, numbers)

# Convert the result to a list to see the values
squared_numbers = list(squared_numbers)
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]

#2. Write a Python program that uses the map() function to square each element␣
 ↪of a list of numbers.

def square(x):
    return x ** 2

numbers = [1, 2, 3, 4, 5]
squared_numbers = map(square, numbers)

# Convert the result to a list to see the squared values
squared_numbers = list(squared_numbers)
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]
```

```
#3. How does the map() function differ from a list comprehension in Python, and␣
 ↪when would you choose one over the other?

#map() is a built-in function that applies a given function to each element of␣
 ↪an iterable and returns an iterable. It's useful when you want to transform␣
 ↪every item in an iterable using a function.
#List comprehensions are a concise way to create lists by applying an␣
 ↪expression to each item in an iterable and filtering items based on a␣
 ↪condition. They are more versatile as they can filter and transform elements␣
 ↪simultaneously.
#You might choose one over the other based on readability and specific␣
 ↪requirements. List comprehensions are often preferred for their simplicity␣
 ↪when you need to create a new list. map() is a better choice when you want␣
 ↪to apply an existing function to an iterable without creating a new list.

#4. Create a Python program that uses the map() function to convert a list of␣
 ↪names to uppercase.


names = ["Alice", "Bob", "Charlie"]
upper_names = map(str.upper, names)

# Convert the result to a list to see the uppercase names
upper_names = list(upper_names)
print(upper_names)  # Output: ['ALICE', 'BOB', 'CHARLIE']

#5. Write a Python program that uses the map() function to calculate the length␣
 ↪of each word in a list of strings.


words = ["apple", "banana", "cherry"]
word_lengths = map(len, words)

# Convert the result to a list to see the word lengths
word_lengths = list(word_lengths)
print(word_lengths)  # Output: [5, 6, 6]

#6. How can you use the map() function to apply a custom function to elements␣
 ↪of multiple lists simultaneously in Python?

#To apply a custom function to elements of multiple lists simultaneously using␣
 ↪map(), you can use the zip() function to combine the lists and then apply␣
 ↪the function to each combination. Here's an example that calculates the sum␣
 ↪of corresponding elements from two lists:
```

```python
list1 = [1, 2, 3]
list2 = [4, 5, 6]

def custom_function(x, y):
    return x + y

result = map(custom_function, list1, list2)

# Convert the result to a list to see the sum of corresponding elements
result = list(result)
print(result)  # Output: [5, 7, 9]

#7. Create a Python program that uses map() to convert a list of temperatures␣
 ↪from Celsius to Fahrenheit.


def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32

temperatures_celsius = [0, 20, 37]
temperatures_fahrenheit = map(celsius_to_fahrenheit, temperatures_celsius)

# Convert the result to a list to see the temperatures in Fahrenheit
temperatures_fahrenheit = list(temperatures_fahrenheit)
print(temperatures_fahrenheit)  # Output: [32.0, 68.0, 98.6]

#8. Write a Python program that uses the map() function to round each element␣
 ↪of a list of floating-point numbers to the nearest integer.


def round_to_integer(x):
    return round(x)

numbers = [1.2, 2.7, 3.5, 4.9]
rounded_numbers = map(round_to_integer, numbers)

# Convert the result to a list to see the rounded numbers
rounded_numbers = list(rounded_numbers)
print(rounded_numbers)  # Output: [1, 3, 4, 5]
```

```
[1, 4, 9, 16, 25]
[1, 4, 9, 16, 25]
['ALICE', 'BOB', 'CHARLIE']
[5, 6, 6]
[5, 7, 9]
[32.0, 68.0, 98.6]
[1, 3, 4, 5]
```

[3]:
```
# Reduce

#1. What is the reduce() function in Python, and what module should you import␣
 ↪to use it? Provide an example of its basic usage.

#The reduce() function is part of the functools module in Python. It is used to␣
 ↪successively apply a given function to the elements of an iterable,␣
 ↪accumulating a single result. The function should take two arguments and␣
 ↪return a single value. The reduce() function reduces the iterable to a␣
 ↪single value by applying the function cumulatively.

#Example of basic usage of reduce():

from functools import reduce

# Define a function to add two numbers
def add(x, y):
    return x + y

numbers = [1, 2, 3, 4, 5]
result = reduce(add, numbers)
print(result)  # Output: 15 (1 + 2 + 3 + 4 + 5)

#2. Write a Python program that uses the reduce() function to find the product␣
 ↪of all elements in a list.

from functools import reduce

# Define a function to multiply two numbers
def multiply(x, y):
    return x * y

numbers = [1, 2, 3, 4, 5]
product = reduce(multiply, numbers)
print(product)  # Output: 120 (1 * 2 * 3 * 4 * 5)

#3. Create a Python program that uses reduce() to find the maximum element in a␣
 ↪list of numbers.


from functools import reduce

# Define a function to find the maximum of two numbers
def find_max(x, y):
    return x if x > y else y

numbers = [12, 45, 6, 89, 34, 72]
```

```
max_number = reduce(find_max, numbers)
print(max_number)  # Output: 89 (the maximum element)

#4. How can you use the reduce() function to concatenate a list of strings into␣
 ↪a single string?

#To concatenate a list of strings into a single string using reduce(), you can␣
 ↪define a function that combines two strings and apply it successively to the␣
 ↪elements of the list.

from functools import reduce

# Define a function to concatenate two strings
def concatenate_strings(x, y):
    return x + y

strings = ["Hello, ", "world!", " This is a test."]
concatenated_string = reduce(concatenate_strings, strings)
print(concatenated_string)  # Output: "Hello, world! This is a test."

# 5. Write a Python program that calculates the factorial of a number using the␣
 ↪reduce() function.

from functools import reduce

# Define a function to calculate the factorial of a number
def factorial(x, y):
    return x * y

n = 5
factorial_result = reduce(factorial, range(1, n + 1))
print(factorial_result)  # Output: 120 (5!)

#6. Create a Python program that uses reduce() to find the GCD (Greatest Common␣
 ↪Divisor) of a list of numbers.


from functools import reduce
import math

# Define a function to find the GCD of two numbers
def find_gcd(x, y):
    return math.gcd(x, y)

numbers = [12, 18, 24, 36]
gcd = reduce(find_gcd, numbers)
print(gcd)  # Output: 6 (GCD of the list)
```

```
#7. Write a Python program that uses the reduce() function to find the sum of the digits of a given number.


from functools import reduce

# Define a function to add the digits of a number
def sum_digits(x, y):
    return x + str(y)

number = 12345
digits_sum = reduce(sum_digits, str(number))
print(digits_sum)  # Output: 15 (1 + 2 + 3 + 4 + 5)
```

```
15
120
89
Hello, world! This is a test.
120
6
12345
```

[4]:
```
# Filter

#1. Explain the purpose of the filter() function in Python and provide an example of how it can be used to filter elements from an iterable.

#The filter() function in Python is used to filter elements from an iterable (e.g., a list, tuple, or other iterable) based on a specified condition or function. It creates a new iterable containing only the elements that satisfy the given condition.

#Example of using filter() to filter even numbers from a list:


# Define a function to check if a number is even
def is_even(x):
    return x % 2 == 0

numbers = [1, 2, 3, 4, 5, 6]
filtered_numbers = filter(is_even, numbers)

# Convert the result to a list to see the filtered values
filtered_numbers = list(filtered_numbers)
print(filtered_numbers)  # Output: [2, 4, 6]
```

```python
#2. Write a Python program that uses the filter() function to select even numbers from a list of integers.


# Define a function to check if a number is even
def is_even(x):
    return x % 2 == 0

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = filter(is_even, numbers)

# Convert the result to a list to see the even numbers
even_numbers = list(even_numbers)
print(even_numbers)  # Output: [2, 4, 6, 8, 10]

#3. Create a Python program that uses the filter() function to select names that start with a specific letter from a list of strings.


# Define a function to check if a name starts with a specific letter
def starts_with_letter(letter, name):
    return name.startswith(letter)

names = ["Alice", "Bob", "Charlie", "David", "Eve"]
letter_to_filter = "D"
filtered_names = filter(lambda name: starts_with_letter(letter_to_filter, name), names)

# Convert the result to a list to see the filtered names
filtered_names = list(filtered_names)
print(filtered_names)  # Output: ["David"]

#4. Write a Python program that uses the filter() function to select prime numbers from a list of integers.


# Define a function to check if a number is prime
def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
```

```
            return False
        i += 6
    return True

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
prime_numbers = filter(is_prime, numbers)

# Convert the result to a list to see the prime numbers
prime_numbers = list(prime_numbers)
print(prime_numbers)  # Output: [2, 3, 5, 7, 11]

#5. How can you use the filter() function to remove None values from a list in␣
 ↪Python?

#You can use the filter() function to remove None values from a list by␣
 ↪specifying a condition that filters out elements that are None.

#Example:

values = [1, None, 2, 3, None, 4, 5, None]
filtered_values = filter(lambda x: x is not None, values)

# Convert the result to a list to see the filtered values
filtered_values = list(filtered_values)
print(filtered_values)  # Output: [1, 2, 3, 4, 5]

#6. Create a Python program that uses filter() to select words longer than a␣
 ↪certain length from a list of strings.


# Define a function to check if a word has more than a certain length
def is_long_word(word, length):
    return len(word) > length

words = ["apple", "banana", "cherry", "date", "elderberry"]
min_length = 6
long_words = filter(lambda word: is_long_word(word, min_length), words)

# Convert the result to a list to see the long words
long_words = list(long_words)
print(long_words)  # Output: ["banana", "cherry", "elderberry"]

#7. Write a Python program that uses the filter() function to select elements␣
 ↪greater than a specified threshold from a list of values.

# Define a function to check if a value is greater than a threshold
def is_greater_than_threshold(value, threshold):
```

```
    return value > threshold

values = [10, 15, 20, 25, 30, 35, 40]
threshold = 25
filtered_values = filter(lambda value: is_greater_than_threshold(value,␣
 ↪threshold), values)

# Convert the result to a list to see the filtered values
filtered_values = list(filtered_values)
print(filtered_values)  # Output: [30, 35, 40]
```

```
[2, 4, 6]
[2, 4, 6, 8, 10]
['David']
[2, 3, 5, 7, 11]
[1, 2, 3, 4, 5]
['elderberry']
[30, 35, 40]
```

[5]:
```
#Recursion


#1. Explain the concept of recursion in Python. How does it differ from␣
 ↪iteration?

#Recursion in Python is a programming technique where a function calls itself␣
 ↪to solve a problem. It is based on the idea of breaking down a problem into␣
 ↪smaller, similar subproblems and solving each subproblem. Recursion involves␣
 ↪two main components:

#Base Case: A condition that determines when the recursion should stop. It␣
 ↪provides the termination condition for the recursive calls.
#Recursive Case: The part of the function where it calls itself with a modified␣
 ↪version of the problem, moving towards the base case.
#Recursion differs from iteration in that iteration uses loops (e.g., for or␣
 ↪while) to repeatedly execute a block of code, whereas recursion solves a␣
 ↪problem by dividing it into smaller instances of the same problem, calling␣
 ↪the function itself to handle each instance.

#2. Write a Python program to calculate the factorial of a number using␣
 ↪recursion.


def factorial(n):
    if n == 0:
        return 1  # Base case: factorial of 0 is 1
    else:
```

```
        return n * factorial(n - 1)  # Recursive case

n = 5
result = factorial(n)
print(f"The factorial of {n} is {result}")  # Output: The factorial of 5 is 120

#3. Create a recursive Python function to find the nth Fibonacci number.

def fibonacci(n):
    if n <= 1:
        return n  # Base cases: Fibonacci of 0 is 0, and of 1 is 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)  # Recursive case

n = 6
result = fibonacci(n)
print(f"The {n}th Fibonacci number is {result}")  # Output: The 6th Fibonacci␣
 ↪number is 8

#4. Write a recursive Python function to calculate the sum of all elements in a␣
 ↪list.


def list_sum(arr):
    if not arr:
        return 0  # Base case: an empty list has a sum of 0
    else:
        return arr[0] + list_sum(arr[1:])  # Recursive case

my_list = [1, 2, 3, 4, 5]
result = list_sum(my_list)
print(f"The sum of the list is {result}")  # Output: The sum of the list is 15

#5. How can you prevent a recursive function from running indefinitely, causing␣
 ↪a stack overflow error?

#To prevent a recursive function from running indefinitely and causing a stack␣
 ↪overflow error, you should ensure that your recursive calls progress towards␣
 ↪a base case. Here are key practices to prevent infinite recursion:

#Define clear base cases that specify when the recursion should stop.
#Ensure that the input to the recursive function changes with each recursive␣
 ↪call, moving closer to the base case.
#Always validate input parameters to avoid unintended infinite recursion.
#Test your recursive function with various inputs to verify that it terminates␣
 ↪correctly.
```

```python
#6. Create a recursive Python function to find the greatest common divisor␣
 ↪(GCD) of two numbers using the Euclidean algorithm.


def gcd(a, b):
    if b == 0:
        return a  # Base case: GCD of a and 0 is a
    else:
        return gcd(b, a % b)  # Recursive case

num1 = 48
num2 = 18
result = gcd(num1, num2)
print(f"The GCD of {num1} and {num2} is {result}")  # Output: The GCD of 48 and␣
 ↪18 is 6

#7. Write a recursive Python function to reverse a string.


def reverse_string(s):
    if len(s) == 0:
        return s  # Base case: an empty string remains empty
    else:
        return reverse_string(s[1:]) + s[0]  # Recursive case

original_str = "hello"
reversed_str = reverse_string(original_str)
print(f"The reversed string is: {reversed_str}")  # Output: The reversed string␣
 ↪is: olleh

#8. Create a recursive Python function to calculate the power of a number (x^n).

def power(x, n):
    if n == 0:
        return 1  # Base case: x^0 is 1
    else:
        return x * power(x, n - 1)  # Recursive case

x = 2
n = 3
result = power(x, n)
print(f"{x} raised to the power {n} is {result}")  # Output: 2 raised to the␣
 ↪power 3 is 8

#9. Write a recursive Python function to find all permutations of a given␣
 ↪string.
```

```python
from itertools import permutations

def string_permutations(s):
    if len(s) <= 1:
        return set([s])
    else:
        perms = set()
        for i, char in enumerate(s):
            remaining_chars = s[:i] + s[i + 1:]
            for perm in string_permutations(remaining_chars):
                perms.add(char + perm)
        return perms

input_str = "abc"
permutations_set = string_permutations(input_str)
print(f"All permutations of '{input_str}': {permutations_set}")

#10. Write a recursive Python function to check if a string is a palindrome.


def is_palindrome(s):
    s = s.lower()  # Convert to lowercase for case-insensitive comparison
    if len(s) <= 1:
        return True  # Base case: empty or single-character string is a␣
 ↪palindrome
    elif s[0] == s[-1]:
        return is_palindrome(s[1:-1])  # Recursive case
    else:
        return False

test_str = "racecar"
result = is_palindrome(test_str)
print(f"'{test_str}' is a palindrome: {result}")  # Output: 'racecar' is a␣
 ↪palindrome: True

#11. Create a recursive Python function to generate all possible combinations␣
 ↪of a list of elements.


from itertools import combinations

def all_combinations(arr):
    combs = []
    for r in range(1, len(arr) + 1):
        combs.extend(combinations(arr, r))
```

```
    return combs

elements = [1, 2, 3]
combinations_list = all_combinations(elements)
print(f"All combinations of {elements}: {combinations_list}")
```

```
The factorial of 5 is 120
The 6th Fibonacci number is 8
The sum of the list is 15
The GCD of 48 and 18 is 6
The reversed string is: olleh
2 raised to the power 3 is 8
All permutations of 'abc': {'abc', 'bca', 'cab', 'bac', 'cba', 'acb'}
'racecar' is a palindrome: True
All combinations of [1, 2, 3]: [(1,), (2,), (3,), (1, 2), (1, 3), (2, 3), (1, 2,
3)]
```

[6]:
```
#Basics of Functions


#1. What is a function in Python, and why is it used?

#In Python, a function is a reusable block of code that performs a specific␣
 ↪task or a set of tasks. Functions are used to organize and encapsulate code,␣
 ↪making it more modular, readable, and maintainable. They allow you to break␣
 ↪down complex programs into smaller, manageable parts, and you can call a␣
 ↪function whenever you need to perform the associated task without rewriting␣
 ↪the code.

#2. How do you define a function in Python? Provide an example.

#You can define a function in Python using the def keyword, followed by the␣
 ↪function name, a pair of parentheses (), and a colon :. The function body is␣
 ↪indented and contains the code to be executed when the function is called.␣
 ↪Here's an example:

def greet(name):
    """This function greets the person passed in as a parameter."""
    print(f"Hello, {name}!")

# Calling the function
greet("Alice")

#3. Explain the difference between a function definition and a function call.
```

```
#Function Definition: A function definition is the code block that defines what␣
 ↪the function does. It includes the def keyword, the function name,␣
 ↪parameters (if any), and the code that gets executed when the function is␣
 ↪called. It defines the function's behavior but does not execute it.

#Function Call: A function call is when you invoke or execute the function you␣
 ↪defined. It involves using the function name followed by parentheses (). The␣
 ↪arguments or values you pass inside the parentheses are provided to the␣
 ↪function, and the code inside the function definition is executed with those␣
 ↪values.

#4. Write a Python program that defines a function to calculate the sum of two␣
 ↪numbers and then calls the function.


def add_numbers(a, b):
    """This function calculates the sum of two numbers."""
    result = a + b
    return result

# Calling the function
num1 = 5
num2 = 7
sum_result = add_numbers(num1, num2)
print(f"The sum of {num1} and {num2} is {sum_result}")

#5. What is a function signature, and what information does it typically␣
 ↪include?

#A function signature is a part of a function's definition that includes the␣
 ↪following information:

#Function Name: The name that identifies the function.
#Parameters: The input values (if any) that the function expects, enclosed in␣
 ↪parentheses. Parameters are placeholders for the actual values that will be␣
 ↪passed when the function is called.
#Return Type: The data type of the value that the function will return. If the␣
 ↪function doesn't return anything, the return type is typically None.
#For example, in the function signature def add(a, b):, "add" is the function␣
 ↪name, and "a" and "b" are parameters. If this function returns an integer,␣
 ↪the complete signature could be def add(a, b) -> int:.

#6. Create a Python function that takes two arguments and returns their product.


def multiply(a, b):
```

```
    """This function calculates the product of two numbers."""
    result = a * b
    return result

# Calling the function
num1 = 3
num2 = 4
product_result = multiply(num1, num2)
print(f"The product of {num1} and {num2} is {product_result}")
```

```
Hello, Alice!
The sum of 5 and 7 is 12
The product of 3 and 4 is 12
```

[7]:
```
#Function Parameters and Arguments:

#1. Explain the concepts of formal parameters and actual arguments in Python␣
 ↪functions.

#Formal Parameters: Formal parameters are placeholders or variable names used␣
 ↪in the function definition to specify the values that the function expects␣
 ↪as input. These parameters are defined inside the parentheses of the␣
 ↪function definition and act as local variables within the function.

#Actual Arguments: Actual arguments, also known as arguments or parameters, are␣
 ↪the values or expressions passed to a function when it is called. These␣
 ↪values are assigned to the corresponding formal parameters in the function␣
 ↪definition, allowing the function to operate on the provided data.

#2. Write a Python program that defines a function with default argument values.


def greet(name="Guest"):
    """This function greets a person with a default name of 'Guest'."""
    print(f"Hello, {name}!")

# Calling the function with and without an argument
greet()  # Output: Hello, Guest!
greet("Alice")  # Output: Hello, Alice!

#In the example above, the name parameter has a default value of "Guest," so if␣
 ↪no argument is provided when calling greet(), it uses the default value.

#3. How do you use keyword arguments in Python function calls? Provide an␣
 ↪example.
```

```python
#Keyword arguments are used in function calls to specify which argument␣
 ↪corresponds to which parameter by using the parameter names as keywords.␣
 ↪This allows you to provide arguments in a different order or skip some␣
 ↪arguments if the function has default values.


def person_info(name, age, city):
    """This function prints person information."""
    print(f"Name: {name}")
    print(f"Age: {age}")
    print(f"City: {city}")

# Using keyword arguments
person_info(name="Alice", age=30, city="New York")
#Using keyword arguments makes it clear which value corresponds to which␣
 ↪parameter, even if the order is different from the function definition.

#4. Create a Python function that accepts a variable number of arguments and␣
 ↪calculates their sum.


def calculate_sum(*args):
    """This function calculates the sum of a variable number of arguments."""
    total = sum(args)
    return total

# Calling the function with different numbers of arguments
result1 = calculate_sum(1, 2, 3)
result2 = calculate_sum(10, 20, 30, 40, 50)
print(f"Result 1: {result1}")  # Output: Result 1: 6
print(f"Result 2: {result2}")  # Output: Result 2: 150
#In this example, *args allows the function to accept any number of arguments,␣
 ↪which are treated as a tuple within the function. The sum() function is used␣
 ↪to calculate their sum.

#5. What is the purpose of the *args and **kwargs syntax in function parameter␣
 ↪lists?

#*args: The *args syntax in a function parameter list allows the function to␣
 ↪accept a variable number of positional arguments. These arguments are␣
 ↪collected into a tuple, and the function can process them as needed. It is␣
 ↪used when you want to pass an arbitrary number of positional arguments to a␣
 ↪function.
```

```
#**kwargs: The **kwargs syntax in a function parameter list allows the function␣
 ↪to accept a variable number of keyword arguments (keyword-value pairs).␣
 ↪These arguments are collected into a dictionary, and the function can access␣
 ↪their values by the provided keywords. It is used when you want to pass an␣
 ↪arbitrary number of keyword arguments to a function.

#Both *args and **kwargs provide flexibility when defining functions that need␣
 ↪to handle different numbers of arguments without explicitly specifying them␣
 ↪in the function signature.
```

```
Hello, Guest!
Hello, Alice!
Name: Alice
Age: 30
City: New York
Result 1: 6
Result 2: 150
```

[8]:
```
#Return Values and Scoping:



#1. Describe the role of the return statement in Python functions and provide␣
 ↪examples.

#The return statement in Python functions is used to specify what value the␣
 ↪function should return when it is called. It allows a function to compute a␣
 ↪result and provide that result to the caller. If a function doesn't contain␣
 ↪a return statement, it defaults to returning None.

#Examples of return statement usage:

def add(a, b):
    result = a + b
    return result  # This function returns the sum of 'a' and 'b'

result = add(3, 5)
print(result)  # Output: 8

def greet(name):
    return f"Hello, {name}!"  # This function returns a greeting message

message = greet("Alice")
print(message)  # Output: Hello, Alice!

#2. Explain the concept of variable scope in Python, including local and global␣
 ↪variables.
```

```
#Local Variables: Variables defined within a function are called local␣
 ↪variables. They have a local scope, which means they can only be accessed␣
 ↪and used within that specific function. Local variables are temporary and␣
 ↪exist only during the execution of the function.

#Global Variables: Variables defined outside of any function, at the top level␣
 ↪of a script or module, are called global variables. They have a global scope␣
 ↪and can be accessed and modified from anywhere within the script or module.␣
 ↪Global variables persist throughout the program's execution.

#3. Write a Python program that demonstrates the use of global variables within␣
 ↪functions.


# Global variable
global_var = 10

def modify_global():
    # Access and modify the global variable
    global global_var
    global_var += 5

modify_global()
print(global_var)  # Output: 15 (global_var was modified within the function)
#In the example above, we use the global keyword inside the function␣
 ↪modify_global() to indicate that we want to work with the global variable␣
 ↪global_var. This allows us to modify its value within the function.

#4. Create a Python function that calculates the factorial of a number and␣
 ↪returns it.


def factorial(n):
    if n == 0:
        return 1  # Base case: factorial of 0 is 1
    else:
        result = 1
        for i in range(1, n + 1):
            result *= i
        return result

num = 5
result = factorial(num)
print(f"The factorial of {num} is {result}")  # Output: The factorial of 5 is␣
 ↪120
```

```
#The factorial function calculates the factorial of a number using a loop and␣
 ↪returns the result.

#5. How can you access variables defined outside a function from within the␣
 ↪function?

#To access variables defined outside a function from within the function, you␣
 ↪can use the global keyword to indicate that you want to work with the global␣
 ↪variable. Here's an example:


global_var = 20  # Global variable

def access_global():
    # Access the global variable within the function
    global global_var
    return global_var

result = access_global()
print(result)  # Output: 20 (accessed the global variable from within the␣
 ↪function)
```

```
8
Hello, Alice!
15
The factorial of 5 is 120
20
```

[9]:
```
#Lambda Functions and Higher-Order Functions:


#1. What are lambda functions in Python, and when are they typically used?

#Lambda functions, also known as anonymous functions, are small, unnamed␣
 ↪functions defined using the lambda keyword. They are typically used when you␣
 ↪need a simple, one-line function for a short period and don't want to define␣
 ↪a formal function using def. Lambda functions are often used in functional␣
 ↪programming constructs like map(), filter(), and sorted().

#2. Write a Python program that uses lambda functions to sort a list of tuples␣
 ↪based on the second element.


# List of tuples
grades = [("Alice", 90), ("Bob", 85), ("Charlie", 95), ("David", 88)]

# Sort the list based on the second element of each tuple (grades)
```

```python
sorted_grades = sorted(grades, key=lambda x: x[1])

# Print the sorted list
print(sorted_grades)
#In the above example, the key parameter of the sorted() function is used to␣
 ↪specify the sorting criterion. The lambda function lambda x: x[1] extracts␣
 ↪the second element (grades) from each tuple, and the list is sorted based on␣
 ↪these grades.

#3. Explain the concept of higher-order functions in Python, and provide an␣
 ↪example.

#Higher-order functions are functions that can take other functions as␣
 ↪arguments and/or return functions as results. In Python, functions are␣
 ↪first-class citizens, which means they can be treated like any other data␣
 ↪type, such as integers or strings. Higher-order functions are a fundamental␣
 ↪concept in functional programming and allow for more modular and flexible␣
 ↪code.

#Example of a higher-order function in Python:

# Higher-order function that applies a given function to each element of a list
def apply_function_to_list(func, lst):
    result = []
    for item in lst:
        result.append(func(item))
    return result

# Example usage:
def square(x):
    return x * x

numbers = [1, 2, 3, 4, 5]
squared_numbers = apply_function_to_list(square, numbers)
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]


#4. Create a Python function that takes a list of numbers and a function as␣
 ↪arguments, applying the function to each element in the list.

def apply_function_to_list(func, num_list):

    #    Apply the given function to each element in the list.

    #Args:
        #func (function): The function to apply to each element.
        #num_list (list): The list of numbers.
```

```
    #Returns:
        #list: A new list containing the results of applying the function.

    result = []
    for num in num_list:
        result.append(func(num))
    return result

# Define a function to square a number
def square(x):
    return x * x

# List of numbers
numbers = [1, 2, 3, 4, 5]

# Apply the square function to each number in the list
squared_numbers = apply_function_to_list(square, numbers)

print(squared_numbers)  # Output: [1, 4, 9, 16, 25]
```

```
[('Bob', 85), ('David', 88), ('Alice', 90), ('Charlie', 95)]
[1, 4, 9, 16, 25]
[1, 4, 9, 16, 25]
```

[10]:
```
#Built-in functions

#1. Describe the role of built-in functions like len(), max(), and min() in␣
 ↪Python.

#len(): The len() function is used to calculate the length or the number of␣
 ↪items in a sequence (e.g., a string, list, tuple, or dictionary). It returns␣
 ↪an integer representing the length of the sequence.

#max(): The max() function is used to find the maximum element from a sequence,␣
 ↪whether it's a list of numbers or a collection of objects. It returns the␣
 ↪maximum value from the given input.

#min(): The min() function is used to find the minimum element from a sequence,␣
 ↪similar to max(). It returns the minimum value from the given input.

#2. Write a Python program that uses the map() function to apply a function to␣
 ↪each element of a list.



# Define a function to square a number
```

```python
def square(x):
    return x * x

# List of numbers
numbers = [1, 2, 3, 4, 5]

# Use the map() function to apply the square function to each element
squared_numbers = list(map(square, numbers))

print(squared_numbers)  # Output: [1, 4, 9, 16, 25]
#In this example, map(square, numbers) applies the square function to each␣
 ↪element in the numbers list, and the result is converted to a list.

#3. How does the filter() function work in Python, and when would you use it?

#The filter() function is used to filter elements from an iterable (e.g., a␣
 ↪list) based on a given function or condition. It takes two arguments:

#A function (or None for filtering with a truthy test) that specifies the␣
 ↪filtering condition.
#An iterable containing elements to be filtered.
#The filter() function returns an iterator containing the elements from the␣
 ↪input iterable that satisfy the filtering condition.

#Use cases for filter() include:

#Filtering a list to retain only elements that meet specific criteria.
#Removing unwanted elements from a collection.
#Selecting elements from a dataset based on a specific condition.

#4. Create a Python program that uses the reduce() function to find the product␣
 ↪of all elements in a list.

#Here's an example that uses the reduce() function from the functools module to␣
 ↪find the product of all elements in a list:


from functools import reduce

# Define a function to calculate the product of two numbers
def multiply(x, y):
    return x * y

# List of numbers
numbers = [1, 2, 3, 4, 5]
```

```
# Use the reduce() function to find the product of all elements
product = reduce(multiply, numbers)

print(product)  # Output: 120
```

```
[1, 4, 9, 16, 25]
120
```

[1]:
```
#Function Documentation and Best Practices:

#1. Explain the purpose of docstrings in Python functions and how to write them.

#Docstrings in Python are used to provide documentation or descriptive␣
 ↪information about functions, modules, classes, or methods. They serve as␣
 ↪human-readable documentation and help users understand the purpose, usage,␣
 ↪and parameters of the code. Docstrings are enclosed in triple quotes (single␣
 ↪or double) and should be placed immediately after the function, class, or␣
 ↪module definition.

#Here's an example of how to write a docstring for a function:


def calculate_sum(a, b):
    """
    Calculate the sum of two numbers.

    Args:
        a (int): The first number.
        b (int): The second number.

    Returns:
        int: The sum of 'a' and 'b'.
    """
    result = a + b
    return result
#In the above example:

#The docstring is enclosed in triple double quotes.
#It provides a brief description of what the function does.
#It documents the function's parameters (arguments) with their types and␣
 ↪descriptions.
#It documents the return value, including its type and what it represents.
#Properly documenting your code with docstrings makes it more readable and␣
 ↪helps other developers understand how to use your functions or modules.

#2. Describe some best practices for naming functions and variables in Python,␣
 ↪including naming conventions and guidelines.
```

```
#Function and Variable Names: Follow these naming conventions and guidelines␣
 ↪for functions and variables:

#Use lowercase letters for function and variable names.
#Separate words in names with underscores (e.g., calculate_sum, my_variable).
#Choose descriptive and meaningful names that indicate the purpose or content␣
 ↪of the function or variable.
#Avoid using single-letter variable names (except in cases like loop counters).
#Use lowercase for variable names, and capitalize the first letter of each word␣
 ↪in function names (e.g., calculate_sum, process_data).
#Constants: Use uppercase letters for constants, and separate words with␣
 ↪underscores. For example, MAX_VALUE, PI.

#Module Names: Module names should be lowercase, and if the name consists of␣
 ↪multiple words, use underscores (e.g., my_module, my_utilities).

#Avoid Reserved Words: Do not use Python reserved words (e.g., if, for, while)␣
 ↪as function or variable names.

#Be Consistent: Maintain consistency in your naming conventions throughout your␣
 ↪codebase. If you adopt a specific naming style, stick to it.

#Use Descriptive Names: Make your function and variable names self-explanatory,␣
 ↪so that someone reading your code can easily understand their purpose␣
 ↪without needing extensive comments.

#Consider PEP 8: PEP 8 is the Python Enhancement Proposal that provides style␣
 ↪guidelines for Python code. Following PEP 8 conventions for naming and␣
 ↪formatting is a widely accepted practice in the Python community.

#Here's an example of following these best practices:


# Good naming practices
def calculate_sum(a, b):
    #Calculate the sum of two numbers."""
    result = a + b
    return result

my_variable = 42
MAX_VALUE = 100
```

[2]:
```
#For loop
```

```python
#1. Write a Python program to print numbers from 1 to 10 using a for loop.


for i in range(1, 11):
    print(i)

#2. Explain the difference between a for loop and a while loop in Python.

#In Python, a for loop is used for iterating over a sequence (e.g., a list, tuple, string, or range), and it runs a predefined number of times. A while loop, on the other hand, runs as long as a specified condition is True. for loops are generally used when you know the number of iterations in advance, while while loops are used when you want to repeat a block of code until a condition is met.

#3. Write a Python program to calculate the sum of all numbers from 1 to 100 using a for loop.


sum = 0
for i in range(1, 101):
    sum += i
print("Sum:", sum)

#4. How do you iterate through a list using a for loop in Python?

#You can iterate through a list using a for loop by specifying the list as the sequence to loop through. For example:

my_list = [1, 2, 3, 4, 5]
for item in my_list:
    print(item)

#5. Write a Python program to find the product of all elements in a list using a for loop.

my_list = [1, 2, 3, 4, 5]
product = 1
for num in my_list:
    product *= num
print("Product:", product)

#6. Create a Python program that prints all even numbers from 1 to 20 using a for loop.


for i in range(2, 21, 2):
```

```
    print(i)

#7. Write a Python program that calculates the factorial of a number using a␣
 ↪for loop.


def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

num = 5  # Example number
print("Factorial of", num, "is", factorial(num))

#8. How can you iterate through the characters of a string using a for loop in␣
 ↪Python?

#You can iterate through the characters of a string using a for loop by␣
 ↪treating the string as a sequence. For example:


my_string = "Hello"
for char in my_string:
    print(char)

#9. Write a Python program to find the largest number in a list using a for␣
 ↪loop.

my_list = [12, 45, 67, 23, 9, 56]
max_num = my_list[0]
for num in my_list:
    if num > max_num:
        max_num = num
print("Largest number:", max_num)

#10. Create a Python program that prints the Fibonacci sequence up to a␣
 ↪specified limit using a for loop.


def fibonacci(limit):
    a, b = 0, 1
    while a < limit:
        print(a)
        a, b = b, a + b

fibonacci(50)  # Example limit
```

```python
#11. Write a Python program to count the number of vowels in a given string␣
 ↪using a for loop.


def count_vowels(string):
    vowels = "AEIOUaeiou"
    count = 0
    for char in string:
        if char in vowels:
            count += 1
    return count

input_string = "Hello, World!"  # Example input
print("Number of vowels:", count_vowels(input_string))

#12. Create a Python program that generates a multiplication table for a given␣
 ↪number using a for loop.


def multiplication_table(number):
    for i in range(1, 11):
        print(f"{number} x {i} = {number * i}")

multiplication_table(5)  # Example number

#13. Write a Python program to reverse a list using a for loop.


my_list = [1, 2, 3, 4, 5]
reversed_list = []
for i in range(len(my_list) - 1, -1, -1):
    reversed_list.append(my_list[i])
print("Reversed list:", reversed_list)

#14. Write a Python program to find the common elements between two lists using␣
 ↪a for loop.


list1 = [1, 2, 3, 4, 5]
list2 = [3, 4, 5, 6, 7]
common_elements = []

for item in list1:
    if item in list2:
        common_elements.append(item)
```

```python
print("Common elements:", common_elements)

#15. Explain how to use a for loop to iterate through the keys and values of a␣
 ↪dictionary in Python.

#You can iterate through the keys and values of a dictionary in Python using a␣
 ↪for loop along with the items() method of the dictionary. Here's an example:

my_dict = {'a': 1, 'b': 2, 'c': 3}

for key, value in my_dict.items():
    print(f"Key: {key}, Value: {value}")

#16. Write a Python program to find the GCD (Greatest Common Divisor) of two␣
 ↪numbers using a for loop.

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

num1 = 24  # Example numbers
num2 = 18
print("GCD:", gcd(num1, num2))

#17. Create a Python program that checks if a string is a palindrome using a␣
 ↪for loop.


def is_palindrome(string):
    string = string.lower().replace(" ", "")  # Convert to lowercase and remove␣
 ↪spaces
    for i in range(len(string) // 2):
        if string[i] != string[-i - 1]:
            return False
    return True

input_string = "A man a plan a canal Panama"  # Example input
print("Is palindrome:", is_palindrome(input_string))

#18. Write a Python program to remove duplicates from a list using a for loop.


my_list = [1, 2, 2, 3, 4, 4, 5, 5]
unique_list = []
```

```python
for item in my_list:
    if item not in unique_list:
        unique_list.append(item)

print("List with duplicates removed:", unique_list)

#19. Create a Python program that counts the number of words in a sentence␣
 ↪using a for loop.

sentence = "This is a sample sentence with several words."
word_count = 0

words = sentence.split()
for word in words:
    word_count += 1

print("Number of words:", word_count)

#20. Write a Python program to find the sum of all odd numbers from 1 to 50␣
 ↪using a for loop.


sum_of_odds = 0
for num in range(1, 51):
    if num % 2 != 0:
        sum_of_odds += num

print("Sum of odd numbers:", sum_of_odds)

#21. Write a Python program that checks if a given year is a leap year using a␣
 ↪for loop.


def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False

year_to_check = 2024  # Example year
if is_leap_year(year_to_check):
    print(f"{year_to_check} is a leap year.")
else:
    print(f"{year_to_check} is not a leap year.")

#22. Create a Python program that calculates the square root of a number using␣
 ↪a for loop.
```

```python
def square_root(x, epsilon=0.00001):
    guess = x / 2.0
    while abs(guess * guess - x) >= epsilon:
        guess = (guess + x / guess) / 2.0
    return guess

number = 25.0  # Example number
print("Square root:", square_root(number))

#23. Write a Python program to find the LCM (Least Common Multiple) of two␣
 ↪numbers using a for loop.

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def lcm(a, b):
    return (a * b) // gcd(a, b)

num1 = 12  # Example numbers
num2 = 18
print("LCM:", lcm(num1, num2))
```

```
1
2
3
4
5
6
7
8
9
10
Sum: 5050
1
2
3
4
5
Product: 120
2
4
6
8
```

```
10
12
14
16
18
20
Factorial of 5 is 120
H
e
l
l
o
Largest number: 67
0
1
1
2
3
5
8
13
21
34
Number of vowels: 3
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
Reversed list: [5, 4, 3, 2, 1]
Common elements: [3, 4, 5]
Key: a, Value: 1
Key: b, Value: 2
Key: c, Value: 3
GCD: 6
Is palindrome: True
List with duplicates removed: [1, 2, 3, 4, 5]
Number of words: 8
Sum of odd numbers: 625
2024 is a leap year.
Square root: 5.000000000016778
LCM: 36
```

[4]:
```
#If else

#1. Write a Python program to check if a number is positive, negative, or zero␣
 ↪using an if-else statement.


num = float(input("Enter a number: "))

if num > 0:
    print("Positive number")
elif num < 0:
    print("Negative number")
else:
    print("Zero")

#2. Create a Python program that checks if a given number is even or odd using␣
 ↪an if-else statement.


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

if num % 2 == 0:
    print("Even number")
else:
    print("Odd number")

#3. How can you use nested if-else statements in Python, and provide an example?

#Nested if-else statements are used when you have multiple conditions to check␣
 ↪within another condition. Here's an example:


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

if num > 0:
    if num % 2 == 0:
        print("Positive even number")
    else:
        print("Positive odd number")
elif num < 0:
    print("Negative number")
else:
    print("Zero")

#4. Write a Python program to determine the largest of three numbers using␣
 ↪if-else.
```

```python
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
num3 = float(input("Enter third number: "))

if num1 >= num2 and num1 >= num3:
    print("Largest number:", num1)
elif num2 >= num1 and num2 >= num3:
    print("Largest number:", num2)
else:
    print("Largest number:", num3)

#5. Write a Python program that calculates the absolute value of a number using␣
 ↪if-else.

num = float(input("Enter a number: "))

if num >= 0:
    print("Absolute value:", num)
else:
    print("Absolute value:", -num)

#6. Create a Python program that checks if a given character is a vowel or␣
 ↪consonant using if-else.


char = input("Enter a character: ").lower()

if char in 'aeiou':
    print("Vowel")
else:
    print("Consonant")

#7. Write a Python program to determine if a user is eligible to vote based on␣
 ↪their age using if-else.


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

if age >= 18:
    print("You are eligible to vote")
else:
    print("You are not eligible to vote")

#8. Create a Python program that calculates the discount amount based on the␣
 ↪purchase amount using if-else.
```

```
purchase_amount = float(input("Enter the purchase amount: "))

if purchase_amount > 1000:
    discount = 0.1 * purchase_amount
else:
    discount = 0

print("Discount amount:", discount)
#9. Write a Python program to check if a number is within a specified range␣
 ↪using if-else.

num = int(input("Enter a number: "))
lower_limit = 10
upper_limit = 50

if lower_limit <= num <= upper_limit:
    print("Number is within the range")
else:
    print("Number is outside the range")

#10. Create a Python program that determines the grade of a student based on␣
 ↪their score using if-else.


score = float(input("Enter your score: "))

if 90 <= score <= 100:
    grade = 'A'
elif 80 <= score < 90:
    grade = 'B'
elif 70 <= score < 80:
    grade = 'C'
elif 60 <= score < 70:
    grade = 'D'
else:
    grade = 'F'

print("Your grade:", grade)


#11. Write a Python program to check if a string is empty or not using if-else.


string = input("Enter a string: ")

if string:
```

```
    print("The string is not empty")
else:
    print("The string is empty")

#12. Create a Python program that identifies the type of a triangle (e.g.,␣
 ↪equilateral, isosceles, or scalene) based on input values using if-else.


side1 = float(input("Enter the length of side 1: "))
side2 = float(input("Enter the length of side 2: "))
side3 = float(input("Enter the length of side 3: "))

if side1 == side2 == side3:
    print("Equilateral triangle")
elif side1 == side2 or side1 == side3 or side2 == side3:
    print("Isosceles triangle")
else:
    print("Scalene triangle")

#13. Write a Python program to determine the day of the week based on a␣
 ↪user-provided number using if-else.


day_number = int(input("Enter a number (1-7) for the day of the week: "))

if day_number == 1:
    day_name = "Monday"
elif day_number == 2:
    day_name = "Tuesday"
elif day_number == 3:
    day_name = "Wednesday"
elif day_number == 4:
    day_name = "Thursday"
elif day_number == 5:
    day_name = "Friday"
elif day_number == 6:
    day_name = "Saturday"
elif day_number == 7:
    day_name = "Sunday"
else:
    day_name = "Invalid input"

print("Day of the week:", day_name)

#14. Create a Python program that checks if a given year is a leap year using␣
 ↪both if-else and a function.
```

```python
def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False

year_to_check = int(input("Enter a year: "))

if is_leap_year(year_to_check):
    print(f"{year_to_check} is a leap year.")
else:
    print(f"{year_to_check} is not a leap year.")

#15. How do you use the "assert" statement in Python to add debugging checks␣
 ↪within if-else blocks?

#The "assert" statement is used to check whether a given condition is True or␣
 ↪False. If the condition is True, the program continues to execute normally;␣
 ↪if the condition is False, an AssertionError is raised. Here's an example:


x = 10
assert x > 0, "x should be greater than 0"
print("x is positive")
#In this example, if x is not greater than 0, the program will raise an␣
 ↪AssertionError with the specified message.

#16. Create a Python program that determines the eligibility of a person for a␣
 ↪senior citizen discount based on age using if-else.


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

if age >= 60:
    print("You are eligible for a senior citizen discount.")
else:
    print("You are not eligible for a senior citizen discount.")

#17. Write a Python program to categorize a given character as uppercase,␣
 ↪lowercase, or neither using if-else.

char = input("Enter a character: ")

if char.isupper():
    print("Uppercase letter")
elif char.islower():
    print("Lowercase letter")
```

```
else:
    print("Neither uppercase nor lowercase")

#18. Write a Python program to determine the roots of a quadratic equation␣
 ↪using if-else.


import math

a = float(input("Enter coefficient a: "))
b = float(input("Enter coefficient b: "))
c = float(input("Enter coefficient c: "))

discriminant = b**2 - 4*a*c

if discriminant > 0:
    root1 = (-b + math.sqrt(discriminant)) / (2*a)
    root2 = (-b - math.sqrt(discriminant)) / (2*a)
    print(f"Roots are real and different: {root1}, {root2}")
elif discriminant == 0:
    root1 = -b / (2*a)
    print(f"Roots are real and equal: {root1}")
else:
    real_part = -b / (2*a)
    imaginary_part = math.sqrt(abs(discriminant)) / (2*a)
    print(f"Roots are complex: {real_part} + {imaginary_part}i, {real_part} -␣
 ↪{imaginary_part}i")

#19. Create a Python program that checks if a given year is a century year or␣
 ↪not using if-else.


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

if year % 100 == 0:
    print("Century year")
else:
    print("Not a century year")

#20. Write a Python program to determine if a given number is a perfect square␣
 ↪using if-else.


import math

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

```
if math.isqrt(num)**2 == num:
    print("Perfect square")
else:
    print("Not a perfect square")

#21. Explain the purpose of the "continue" and "break" statements within␣
 ↪if-else loops.

#The "continue" statement is used to skip the current iteration of a loop and␣
 ↪continue with the next iteration.
#The "break" statement is used to exit the loop prematurely, even if the loop␣
 ↪condition is still True.
#These statements are often used to control the flow of a loop based on certain␣
 ↪conditions.

#22. Create a Python program that calculates the BMI (Body Mass Index) of a␣
 ↪person based on their weight and height using if-else.


weight = float(input("Enter your weight (kg): "))
height = float(input("Enter your height (m): "))

bmi = weight / (height ** 2)

if bmi < 18.5:
    category = "Underweight"
elif 18.5 <= bmi < 24.9:
    category = "Normal weight"
elif 24.9 <= bmi < 29.9:
    category = "Overweight"
else:
    category = "Obese"

print(f"Your BMI is {bmi:.2f}, which is classified as {category}.")

#23. How can you use the "filter()" function with if-else statements to filter␣
 ↪elements from a list?

#You can use the filter() function to filter elements from a list based on a␣
 ↪given condition using a lambda function. Here's an example:


my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
filtered_list = list(filter(lambda x: x % 2 == 0, my_list))  # Filters even␣
 ↪numbers
```

```
print("Filtered list of even numbers:", filtered_list)
#The filter() function applies the condition specified in the lambda function␣
 ↪to each element of the list and returns a filtered list containing elements␣
 ↪that satisfy the condition.

#24. Write a Python program to determine if a given number is prime or not␣
 ↪using if-else.


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

if num <= 1:
    is_prime = False
else:
    is_prime = True
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            is_prime = False
            break

if is_prime:
    print(f"{num} is a prime number.")
else:
    print(f"{num} is not a prime number.")
```

```
Enter a number:  100
Positive number
Enter a number:  32
Even number
Enter a number:  33
Positive odd number
Enter first number:  22
Enter second number:  11
Enter third number:  22
Largest number: 22.0
Enter a number:  11
Absolute value: 11.0
Enter a character:  11
Consonant
Enter your age:  45
You are eligible to vote
```

```
Enter the purchase amount:  1000

Discount amount: 0

Enter a number:  222

Number is outside the range

Enter your score:  2

Your grade: F

Enter a string:  I am a human being

The string is not empty

Enter the length of side 1:  12
Enter the length of side 2:  13
Enter the length of side 3:  13

Isosceles triangle

Enter a number (1-7) for the day of the week:  4

Day of the week: Thursday

Enter a year:  2023

2023 is not a leap year.
x is positive

Enter your age:  45

You are not eligible for a senior citizen discount.

Enter a character:  5

Neither uppercase nor lowercase

Enter coefficient a:  3
Enter coefficient b:  3
Enter coefficient c:  3

Roots are complex: -0.5 + 0.8660254037844387i, -0.5 - 0.8660254037844387i

Enter a year:  2022

Not a century year

Enter a number:  11

Not a perfect square

Enter your weight (kg):  80
Enter your height (m):  180

Your BMI is 0.00, which is classified as Underweight.
Filtered list of even numbers: [2, 4, 6, 8, 10]

Enter a number:  22
```

```
22 is not a prime number.
```

[ ]: