MASTERING PYTHON

100+

solved and commented

exercises

to accelerate your learning

RUHAN CONCEIÇÃO

MASTERING PYTHON:

100+ Solved and Commented Exercises to Accelerate your Learning

Ruhan Avila da Conceição

Preface

Welcome to this book where solved and commented Python exercises are presented. In this book, you will find a collection of over 100 exercises designed to help you improve your programming skills in this powerful language.

Learning to program involves not only understanding theoretical concepts but also applying those concepts in real-life situations. That's exactly what you will find in this book: a wide variety of problems ranging from basic fundamentals to more complex challenges.

Each exercise is accompanied by a complete and detailed solution, which not only presents the source code but also explains the reasoning behind the approach taken. These comments discuss important concepts, provide valuable tips, and help understand how programming logic can be efficiently applied in problem-solving.

As you progress through the exercises, you will be challenged with tasks involving mathematical formula manipulation, strings, conditionals, loops, vector manipulation, matrices, and much more.

The main goal of this book is to provide a practical and comprehensive resource for programmers seeking improvement. Whether you are a beginner in Python looking to solidify your knowledge or an experienced programmer wishing to deepen your expertise, these exercises will serve as an excellent study guide and reference. This book is also suitable for teachers who would like to have a rich collection of solved Programming Logic exercises to create exercises and questions for their students.

In several exercises, multiple solutions are presented for the same proposed problem, involving different strategies and techniques.

Enjoy this learning journey and dive into the solved and commented Python exercises. Prepare yourself for stimulating challenges, creative solutions, and a unique opportunity to enhance your programming skills.

This book was written using artificial intelligence tools in content creation, but all materials have been reviewed and edited by the author to deliver a final high-quality product.

Happy reading, happy studying, and have fun exploring the fascinating world of Python programming

Ruhan Avila da Conceição.

Summary

Introduction

If you have acquired this book, you want to start programming and be logically challenged as soon as possible, without wanting to read a sermon on the mount. But it is important to highlight a few things before we begin.

Even though many exercises may be considered easy, if you are new to this programming journey, it is important for you to first try to solve the problem on your own before looking at the solution. There is more than one possible solution to the same problem, and you need to think and develop your own solution. Then, you can compare it with the proposed one in the book, identify the strengths of each, and try to learn a little more.

If the exercise is too difficult and you can't solve it, move on to the next one and try again the next day. Don't immediately jump to the answer, even if you can't solve it, and definitely don't look at the answer without even attempting to solve it.

Learning programming logic is not about getting the answer; it's about the journey you take to arrive at the answer.

With that being said, the remaining chapters of this book are divided according to the programming topics covered in the proposed exercises

- Mathematical Formulas (15 exercises)
- Conditionals (20 exercises)
- Loop Structures (25 exercises)
- Arrays (10 exercises)
- Strings (10 exercises)
- Matrices (10 exercises)
- Recursive Functions (10 exercises)
- + Extra Exercises at the End

You can check out the complete list of exercises at the end of the book.

From now on, it's all up to you!

Mathematical Formulas

Before starting with the exercises, and their respective commented solutions, let's review some important commands and functions to solve the exercises in this chapter.

The ***print*** function

In Python, the ***print()*** function is used to display output on the console or terminal. It allows you to output text, variables, or any other data you want to visualize while running your program. The ***print()*** function has the following syntax:

*print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)*

Let's break down the different components of the ***print()*** function:

****objects***: This is a positional argument that represents one or more objects or values to be displayed. You can pass multiple objects separated by commas, and they will be concatenated and printed. The print() function automatically converts the objects into strings before printing.

sep=' ': This is an optional keyword argument that specifies the separator between the objects being printed. By default, it is set to a space (' '). You can change it to any other string value as per your requirement.

end='\n': This is an optional keyword argument that determines what character(s) should be printed at the end. By default, it is set to a newline character (***'\n'***), which means each call to ***print()*** will end with a newline. You can change it to an empty string (***''***) or any other string if you want to alter the ending character(s).

file=sys.stdout: This is an optional keyword argument that specifies the file-like object where the output will be written. By default, it is set to ***sys.stdout***, which represents the standard output (console/terminal). You can pass other file objects if you want to redirect the output to a file or a different destination.

flush=False: This is an optional keyword argument that determines whether the output should be flushed immediately. By default, it is set to ***False***, which means the output is not immediately flushed. Setting it to ***True*** will ensure the output is written immediately.

Here are a few examples of using the ***print()*** function in Python:

```
print("Hello, World!")  # Output: Hello, World!

name = "Alice"
age = 25
print("Name:", name, "Age:", age)  # Output: Name: Alice Age: 25

print("1", "2", "3", sep="-")  # Output: 1-2-3

print("Hello", end="")
print("World!")  # Output: HelloWorld!

file = open("output.txt", "w")
print("Redirected output", file=file)
file.close()  # The output is written to 'output.txt'
```

The ***input*** function

In Python, the ***input()*** function is used to read input from the user via the console or terminal. It allows your program to prompt the user for information, and the user can enter a value, which is then returned as a string. The ***input()*** function has the following syntax:

input(prompt)

Let's break down the components of the ***input()*** function:

prompt: This is an optional parameter that represents the string to be displayed as a prompt to the user. It is displayed on the console before waiting for the user's input. If you omit the ***prompt*** parameter, the function will not display anything as a prompt.

The ***input()*** function waits for the user to enter input and press the Enter key. Once the user provides input, the function returns the entered value as a string. You can assign the returned value to a variable for further processing.

Here's an example of using the ***input()*** function in Python:

```
name = input("Enter your name: ")
print("Hello, " + name + "!")

age = input("Enter your age: ")
age = int(age)  # Convert the string to an integer for numerical operations
print("You will be " + str(age + 1) + " years old next year.")
```

In this example, the first ***input()*** function prompts the user to enter their name. The entered name is then stored in the ***name*** variable and displayed as part of the greeting message.

The second ***input()*** function prompts the user to enter their age. The entered value is stored as a string in the ***age*** variable. To perform numerical operations, we convert the ***age*** string to an integer using the ***int()*** function. Then we add 1 to the age and convert it back to a string using ***str()*** before displaying it in the message.

Note: The ***input()*** function always returns a string, even if the user enters a number. If you need to use the input as a number, you'll need to convert it to the appropriate data type using functions like ***int()*** or ***float()***.

Math operations

Python provides several built-in operators and functions for performing mathematical operations. Here are some commonly used math operations in Python:

Addition (+): Adds two numbers together.

```
a = 5
b = 3
result = a + b  # 8
```

Subtraction (-): Subtracts one number from another.

```
a = 5
b = 3
result = a - b  # 2
```

Multiplication (*): Multiplies two numbers.

```
a = 5
b = 3
result = a * b  # 15
```

Division (*/*): Divides one number by another, returning a floating-point result.

```
a = 5
b = 3
result = a / b  # 1.6666666666666667
```

Floor Division (*//*): Divides one number by another and rounds down to the nearest whole number.

```
a = 5
b = 3
result = a // b  # 1
```

Modulo (%): Returns the remainder after division.

```
a = 5
b = 3
result = a % b  # 2
```

Exponentiation (**): Raises a number to the power of another number.

```
a = 2
b = 3
result = a ** b  # 8
```

Absolute Value (***abs()***): Returns the absolute (positive) value of a number.

```
x = -5
result = abs(x)  # 5
```

Round (***round()***): Rounds a number to the nearest integer or a specified number of decimal places.

```
x = 3.14159
result = round(x)  # 3

y = 3.14159
result = round(y, 2)  # 3.14
```

Square Root (***math.sqrt()***): Calculates the square root of a number. This requires importing the ***math*** module.

```
import math

x = 16
result = math.sqrt(x)  # 4.0
```

These are just a few examples of the math operations available in Python. Python also provides additional mathematical functions in the ***math*** module, such as trigonometric functions (***sin()***, ***cos()***, ***tan()***) and logarithmic functions (***log()***, ***log10()***, etc.).

Now, let's get to the exercises.

1. Write a program that prompts the user for two numbers and displays the addition, subtraction, multiplication, and division between them.

```
# Prompt the user for two numbers
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

# Perform arithmetic operations
addition = num1 + num2
subtraction = num1 - num2
multiplication = num1 * num2
division = num1 / num2

# Display the results
print("Addition:", addition)
print("Subtraction:", subtraction)
print("Multiplication:", multiplication)
print("Division:", division)
```

We start by prompting the user to enter two numbers. The ***input*** function is used to receive user input, and ***float*** is used to convert the input into a floating-point number (decimal number).

Next, we perform the arithmetic operations using the given numbers:

Addition: The + operator adds the two numbers together.

Subtraction: The - operator subtracts the second number from the first number.

Multiplication: The * operator multiplies the two numbers.

Division: The / operator divides the first number by the second number.

The results of the arithmetic operations are stored in separate variables (***addition***, ***subtraction***, ***multiplication***, ***division***).

Finally, we use the ***print*** function to display the results to the user.

Note that the program assumes the user will input valid numbers. If the user enters non-numeric values, the program will raise a ***ValueError***. You can add error handling to handle such cases if needed.

2. Write a program that calculates the arithmetic mean of two numbers.

```
# Prompt the user for two numbers
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

# Calculate the arithmetic mean
mean = (num1 + num2) / 2

# Display the result
print("Arithmetic mean:", mean)
```

We start by prompting the user to enter two numbers using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the two numbers, we calculate the arithmetic mean by adding the two numbers together (***num1*** + ***num2***) and dividing the sum by 2. This is done using the / operator.

The result of the arithmetic mean calculation is stored in the variable ***mean***.

Finally, we use the ***print*** function to display the arithmetic mean to the user.

Note that the program assumes the user will input valid numbers. If the user enters non-numeric values, the program will raise a ***ValueError***. You can add error handling to handle such cases if needed.

3. Create a program that calculates and displays the arithmetic mean of three grades entered by the user.

```
# Prompt the user for three grades
grade1 = float(input("Enter the first grade: "))
grade2 = float(input("Enter the second grade: "))
grade3 = float(input("Enter the third grade: "))

# Calculate the arithmetic mean
mean = (grade1 + grade2 + grade3) / 3

# Display the result
print("Arithmetic mean:", mean)
```

We prompt the user to enter three grades using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the three grades, we calculate the arithmetic mean by adding the three grades together (***grade1*** + ***grade2*** + ***grade3***) and dividing the sum by 3. This is done using the / operator.

The result of the arithmetic mean calculation is stored in the variable ***mean***.

Finally, we use the ***print*** function to display the arithmetic mean to the user.

Note that the program assumes the user will input valid numeric grades. If the user enters non-numeric values, the program will raise a ***ValueError***. You can add error handling to handle such cases if needed.

Formatting the Output

To limit the output of the arithmetic mean to two decimal places, you can use string formatting or the built-in ***round*** function. Here's an updated version of the program that displays the arithmetic mean with two decimal places:

```
# Prompt the user for three grades
grade1 = float(input("Enter the first grade: "))
grade2 = float(input("Enter the second grade: "))
grade3 = float(input("Enter the third grade: "))

# Calculate the arithmetic mean
mean = (grade1 + grade2 + grade3) / 3

# Round the mean to two decimal places
rounded_mean = round(mean, 2)

# Display the result with two decimal places
print("Arithmetic mean:", "{:.2f}".format(rounded_mean))
```

We use the ***round*** function to round the arithmetic mean (***mean***) to two decimal places. The round function takes two arguments: the number to be rounded and the number of decimal places.

We then use string formatting to display the rounded mean (***rounded_mean***) with exactly two decimal places. The ***"{:.2f}".format(rounded_mean)*** format specifier formats the number with two decimal places.

By using these additional steps, the program will display the arithmetic mean with a maximum of two decimal places.

4. Write a program that calculates the geometric mean of three numbers entered by the user

```python
import math

# Prompt the user for three numbers
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
num3 = float(input("Enter the third number: "))

# Calculate the geometric mean
product = num1 * num2 * num3
geometric_mean = math.pow(product, 1/3)

# Display the result
print("Geometric mean:", geometric_mean)
```

Now let's break down the solution step by step:

We start by importing the ***math*** module, which provides mathematical functions and constants for mathematical operations. We need it to use the ***pow*** function to calculate the cubic root later.

We prompt the user to enter three numbers using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the three numbers, we calculate the product of the numbers by multiplying them together (***num1 * num2 * num3***).

Next, we calculate the geometric mean by taking the cubic root of the product using the ***math.pow*** function. We raise the product to the power of 1/3 to obtain the cubic root.

The result of the geometric mean calculation is stored in the variable ***geometric_mean***.

Finally, we use the print function to display the geometric mean to the user.

5. Write a program that calculates the BMI of an individual, using the formula BMI = weight / height^2

```
# Prompt the user for weight and height
weight = float(input("Enter your weight in kilograms: "))
height = float(input("Enter your height in meters: "))

# Calculate the BMI
bmi = weight / (height ** 2)

# Display the result
print("Your BMI is:", bmi)
```

Now let's go through the solution step by step:

We prompt the user to enter their weight in kilograms using the ***input*** function. The ***float*** function is used to convert the input into a floating-point number.

Similarly, we prompt the user to enter their height in meters.

After obtaining the weight and height, we calculate the BMI by dividing the ***weight*** by the square of the ***height***. We use the ****** operator to calculate the square of the ***height***.

The result of the BMI calculation is stored in the variable ***bmi***.

Finally, we use the ***print*** function to display the calculated BMI to the user.

6. Create a program that calculates and displays the perimeter of a circle, prompting the user for the radius.

```
import math

# Prompt the user for the radius
radius = float(input("Enter the radius of the circle: "))

# Calculate the perimeter
perimeter = 2 * math.pi * radius

# Display the result
print("The perimeter of the circle is:", perimeter)
```

We start by importing the ***math*** module, which provides mathematical functions and constants for mathematical operations. We need it to access the value of pi (***math.pi***).

We prompt the user to enter the radius of the circle using the ***input*** function. The ***float*** function is used to convert the input into a floating-point number.

After obtaining the radius, we calculate the perimeter of the circle using the formula ***2 * pi * radius***, where ***pi*** is the mathematical constant representing the ratio of the circumference of a circle to its diameter.

The result of the perimeter calculation is stored in the variable ***perimeter***.

Finally, we use the ***print*** function to display the calculated perimeter of the circle to the user.

7. Write a program that calculates the area of a circle from the radius, using the formula $A = \pi r^2$

```
import math

# Prompt the user for the radius
radius = float(input("Enter the radius of the circle: "))

# Calculate the area
area = math.pi * radius ** 2

# Display the result
print("The area of the circle is:", area)
```

We start by importing the ***math*** module, which provides mathematical functions and constants for mathematical operations. We need it to access the value of pi (***math.pi***).

We prompt the user to enter the radius of the circle using the ***input*** function. The ***float*** function is used to convert the input into a floating-point number.

After obtaining the radius, we calculate the area of the circle using the formula ***pi * radius ** 2***, where ***pi*** is the mathematical constant

representing the ratio of the circumference of a circle to its diameter.

The result of the area calculation is stored in the variable ***area***.

Finally, we use the ***print*** function to display the calculated area of the circle to the user.

8. Write a program that calculates the delta of a quadratic equation ($\Delta = b^2 - 4ac$).

```
# Prompt the user for the coefficients of the quadratic equation
a = float(input("Enter the coefficient a: "))
b = float(input("Enter the coefficient b: "))
c = float(input("Enter the coefficient c: "))

# Calculate the delta
delta = b**2 - 4*a*c

# Display the result
print("The delta (Δ) of the quadratic equation is:", delta)
```

We prompt the user to enter the coefficients of the quadratic equation (***a***, ***b***, and ***c***) using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the coefficients, we calculate the delta using the formula ***b**2 - 4*a*c***, where ** is the exponentiation operator.

The result of the delta calculation is stored in the variable ***delta***.

Finally, we use the ***print*** function to display the calculated delta of the quadratic equation to the user.

9. Write a program that calculates the perimeter and area of a rectangle, using the formulas P = 2(w + l) and A = wl, where w is the width and l is the length

```
# Prompt the user for the width and length of the rectangle
width = float(input("Enter the width of the rectangle: "))
length = float(input("Enter the length of the rectangle: "))

# Calculate the perimeter
perimeter = 2 * (width + length)

# Calculate the area
area = width * length

# Display the results
print("Perimeter of the rectangle:", perimeter)
print("Area of the rectangle:", area)
```

We prompt the user to enter the width and length of the rectangle using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the width and length, we calculate the perimeter using the formula ***2 * (width + length)***, which adds the two sides and doubles the sum.

Similarly, we calculate the area using the formula ***width * length***, which multiplies the width and length of the rectangle.

The results of the perimeter and area calculations are stored in the variables ***perimeter*** and ***area***, respectively.

Finally, we use the ***print*** function to display the calculated perimeter and area of the rectangle to the user.

10. Write a program that calculates the perimeter and area of a triangle, using the formulas P = a + b + c and A = (b * h) / 2, where a, b and c are the sides of the triangle and h is the height relative to the side B.

```
# Prompt the user for the lengths of the sides and height of the triangle
side_a = float(input("Enter the length of side a: "))
side_b = float(input("Enter the length of side b: "))
side_c = float(input("Enter the length of side c: "))
height = float(input("Enter the height relative to side b: "))

# Calculate the perimeter
perimeter = side_a + side_b + side_c

# Calculate the area
area = (side_b * height) / 2

# Display the results
print("Perimeter of the triangle:", perimeter)
print("Area of the triangle:", area)
```

We prompt the user to enter the lengths of side a, side b, side c, and the height relative to side b of the triangle using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the side lengths and height, we calculate the perimeter of the triangle using the formula ***side_a + side_b + side_c***, which adds the lengths of all three sides.

Similarly, we calculate the area of the triangle using the formula ***(side_b * height) / 2***, which multiplies the length of side b by the height and divides the result by 2.

The results of the perimeter and area calculations are stored in the variables ***perimeter*** and ***area***, respectively.

Finally, we use the ***print*** function to display the calculated perimeter and area of the triangle to the user.

Note that the program assumes the user will input valid numeric values for the side lengths and height. If the user enters non-numeric values, the program will raise a ***ValueError***. You can add error handling to handle such cases if needed.

11. Write a program that calculates the average velocity of an object, using the formula v = Δs/Δt, where v is the average velocity, Δs is the space variation, and Δt is the time variation

```
# Prompt the user for space variation and time variation
space_variation = float(input("Enter the space variation (Δs): "))
time_variation = float(input("Enter the time variation (Δt): "))

# Calculate the average velocity
average_velocity = space_variation / time_variation

# Display the result
print("The average velocity is:", average_velocity)
```

We prompt the user to enter the space variation (***Δs***) and time variation (***Δt***) using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the space variation and time variation, we calculate the average velocity using the formula ***space_variation / time_variation***, which divides the space variation by the time variation.

The result of the average velocity calculation is stored in the variable ***average_velocity***.

Finally, we use the print function to display the calculated average velocity of the object to the user.

12. Write a program that calculates the kinetic energy of a moving object, using the formula E = (mv²) / 2, where E is the kinetic energy, m is the mass of the object, and v is the velocity.

```
# Prompt the user for the mass and velocity of the object
mass = float(input("Enter the mass of the object: "))
velocity = float(input("Enter the velocity of the object: "))

# Calculate the kinetic energy
kinetic_energy = (mass * velocity ** 2) / 2

# Display the result
print("The kinetic energy of the object is:", kinetic_energy)
```

We prompt the user to enter the mass and velocity of the object using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the mass and velocity, we calculate the kinetic energy using the formula ***(mass * velocity ** 2) / 2***. Here, ******** is the exponentiation operator used to calculate the square of the velocity.

The result of the kinetic energy calculation is stored in the variable ***kinetic_energy***.

Finally, we use the ***print*** function to display the calculated kinetic energy of the object to the user.

13. Write a program that calculates the work done by a force acting on an object, using the formula T = F * d, where T is the work, F is the applied force, and d is the distance traveled by the object.

```
# Prompt the user for the applied force and distance traveled
applied_force = float(input("Enter the applied force: "))
distance = float(input("Enter the distance traveled: "))

# Calculate the work done
work_done = applied_force * distance

# Display the result
print("The work done by the force is:", work_done)
```

We prompt the user to enter the applied force and distance traveled using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the applied force and distance, we calculate the work done using the formula ***applied_force * distance***. This multiplies the applied force by the distance traveled.

The result of the work done calculation is stored in the variable ***work_done***.

Finally, we use the print function to display the calculated work done by the force to the user.

14. Write a program that reads the x and y position of two points in the Cartesian plane, and calculates the distance between them.

```
import math

# Prompt the user for the coordinates of point 1
x1 = float(input("Enter the x-coordinate of point 1: "))
y1 = float(input("Enter the y-coordinate of point 1: "))

# Prompt the user for the coordinates of point 2
x2 = float(input("Enter the x-coordinate of point 2: "))
y2 = float(input("Enter the y-coordinate of point 2: "))

# Calculate the distance between the two points
distance = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

# Display the result
print("The distance between the two points is:", distance)
```

We start by importing the ***math*** module, which provides mathematical functions and constants for mathematical operations. We need it to access the square root function (***math.sqrt)***.

We prompt the user to enter the x and y coordinates of point 1 using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

Similarly, we prompt the user to enter the x and y coordinates of point 2.

After obtaining the coordinates of both points, we calculate the distance between them using the distance formula. The formula is: ***sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)***. This formula calculates the square root of the sum of the squares of the differences in the x-coordinates and y-coordinates of the two points.

The result of the distance calculation is stored in the variable ***distance***.

Finally, we use the ***print*** function to display the calculated distance between the two points to the user.

15. Create a program that prompts the user for the radius of a sphere and calculates and displays its volume.

```
import math

# Prompt the user for the radius of the sphere
radius = float(input("Enter the radius of the sphere: "))

# Calculate the volume of the sphere
volume = (4/3) * math.pi * radius**3

# Display the result to the user
print("The volume of the sphere is:", volume)
```

The program will start by prompting the user to enter the radius of the sphere. The radius is the distance from the center of the sphere to any point on its surface.

Once the user enters the radius, the program will calculate the volume of the sphere using the formula $V = (4/3) * \pi * r^3$, where V is the volume and r is the radius of the sphere.

To perform the calculation, the program will utilize the mathematical constant π (pi), which is available in Python's ***math*** module. The math

module provides various mathematical functions and constants, including π. We'll use ***math.pi*** to access the value of π in our program.

After calculating the volume of the sphere, the program will display the result to the user.

Conditionals

Before moving on to the exercises and their respective commented solutions, let's make an introduction to some important content for solving the activities.

Comparison Operators

In Python, logical comparators are used to compare values and evaluate conditions. They return Boolean values (***True*** or ***False***) based on the comparison result. Here are the commonly used logical comparators in Python:

Equal to (==): Checks if two values are equal.

```
x = 5
y = 3
result = x == y  # False
```

Not equal to (!=): Checks if two values are not equal.

```
x = 5
y = 3
result = x != y  # True
```

Greater than (>): Checks if the left operand is greater than the right operand.

```
x = 5
y = 3
result = x > y  # True
```

Less than (<): Checks if the left operand is less than the right operand.

```
x = 5
y = 3
result = x < y  # False
```

Greater than or equal to (>=): Checks if the left operand is greater than or equal to the right operand.

```
x = 5
y = 3
result = x >= y  # True
```

Less than or equal to (<=): Checks if the left operand is less than or equal to the right operand.

```
x = 5
y = 3
result = x <= y  # False
```

Logical Operators

In Python, there are three main logical operators: ***and***, ***or*** and ***not***. These operators are used to combine logical expressions and evaluate complex conditions. Here is a detailed explanation of each operator:

and operator:

The ***and*** operator is used to combine two or more logical expressions. It returns ***True*** only if all expressions are true. Otherwise, it returns ***False***. The truth table for the and operator is as follows:

x	**y**	**x *and* y**
True	*True*	*True*
True	*False*	*False*
False	*True*	*False*
False	*False*	*False*

```
x = True
y = False
result = x and y  # False
```

or operator:

The ***or*** operator is used to combine two or more logical expressions. It returns ***True*** if at least one of the expressions is true. Returns ***False*** only if all expressions are false. The truth table for the or operator is as follows:

x	**y**	**x *or* y**
True	*True*	*True*
True	*False*	*True*
False	*True*	*True*
False	*False*	*False*

```
x = True
y = False
result = x or y  # True
```

not operator:

The ***not*** operator is used to negate a logical expression. It reverses the value of the expression. If the expression is ***True***, the ***not*** operator returns ***False***. If the expression is ***False***, the not operator returns ***True***.

```
x = True
result = not x  # False
```

Logical operators can be used to combine multiple conditions or Boolean values to create more complex expressions. Here are a few examples:

```
x = 5
y = 3
z = 7

# Using logical AND
result = x > y and y < z  # True

# Using logical OR
result = x > y or y > z  # True

# Combining multiple conditions
result = (x > y) and (y < z) or (x == z)  # True
```

Logical operators are often used in conditional statements (***if***, ***elif***, ***else***) and loops (***while***, ***for***) to control the flow of the program based on certain conditions.

Conditionals in Python

Conditionals in Python are used to control the flow of a program based on certain conditions. They allow you to execute different blocks of code depending on whether a condition is ***True*** or ***False***. Python provides the ***if***, ***elif*** (short for "***else if***"), and ***else*** statements for creating conditionals.

Here's the basic syntax for conditionals in Python:

```
if condition1:
    print('Code to be executed if condition1 is True')
elif condition2:
    print('Code to be executed if condition1 is False and condition2 is True')
else:
    print('Code to be executed if both condition1 and condition2 are False')
```

Let's go through each part of the conditional syntax:

if: The ***if*** statement is the starting point of a conditional block. It checks whether a condition is ***True***. If the condition is ***True***, the code inside the corresponding block is executed. If the condition is ***False***, the program moves on to the next part of the conditional.

elif (optional): The ***elif*** statement allows you to check additional conditions if the previous conditions were ***False***. It can be used multiple times in a conditional block. If an ***elif*** condition is ***True***, the code inside the corresponding block is executed. If the condition is ***False***, the program moves to the next ***elif*** or ***else*** block.

else (optional): The ***else*** statement is used at the end of a conditional block and does not have a condition to check. It executes its block of code when all the previous conditions are ***False***. An ***else*** block is optional, and you can have only one else block in a conditional.

Here's an example that demonstrates the use of conditionals in Python:

```
x = 5

if x > 10:
    print("x is greater than 10")
elif x > 5:
    print("x is greater than 5 but not greater than 1
else:
    print("x is 5 or less")

# Output: x is 5 or less
```

In this example, the program checks the value of ***x*** using conditionals. Since ***x*** is 5, the first condition ***x > 10*** is ***False***. The second condition ***x > 5*** is also ***False***. Therefore, the program executes the code inside the else block and prints "*x is 5 or less*".

You can have more complex conditions by combining logical operators (***and***, ***or***, ***not***) and comparison operators (==, ***!***=, >, <, >=, <=) inside the condition of the if, elif, and else statements.

Conditionals are powerful constructs that allow your program to make decisions based on different scenarios and conditions. They are fundamental to control the flow and behavior of your code.

The importance of indentation

In Python, indentation is not just a matter of style but is crucial to the structure and correct execution of the code. Python uses indentation to define blocks of code, such as those within conditionals, loops, and function definitions. The importance of indentation in Python can be summarized as follows:

Block Structure: Indentation is used to define the structure and hierarchy of blocks in Python. It helps determine which lines of code are part of a specific block and which are outside of it. Blocks are typically denoted by indentation levels of consistent spaces or tabs.

Readability: Indentation improves the readability of the code by visually separating different blocks of code. It makes the code easier to understand and follow, both for the programmer writing the code and for others who may read or maintain it in the future.

Code Execution: Python interprets the indentation to determine the scope and execution of code blocks. Incorrect indentation can lead to syntax errors or cause the code to behave unexpectedly. Proper indentation ensures that the code is executed as intended.

Here's an example to illustrate the importance of indentation in Python:

```
if x > 10:
    print("x is greater than 10")
    print("This line is part of the if block")

print("This line is outside of the if block")
```

In this example, the two ***print*** statements are indented with four spaces, indicating that they are part of the if block. The last ***print*** statement is not indented and is therefore outside of the if block. If the indentation is incorrect, the code may not execute as expected, leading to errors or logical issues.

It's important to note that Python does not use braces ***{}*** like some other programming languages to define blocks. Instead, it relies solely on indentation. Therefore, consistent and proper indentation is critical in Python to avoid syntax errors and ensure the code is structured correctly.

In most Python editors and IDEs, indentation is automatically handled, making it easier for developers to follow indentation conventions. It is generally recommended to use consistent indentation throughout the codebase, typically with four spaces or one tab, to ensure clarity and conformity.

Overall, indentation plays a fundamental role in Python, contributing to both the visual structure and the correct execution of the code.

Now, let's get to the exercises.

16. Make a program that asks for a person's age and displays whether they are of legal age or not.

```
# Prompt the user for their age
age = int(input("Enter your age: "))

# Check if the person is of legal age
if age >= 18:
    print("You are of legal age.")
else:
    print("You are not of legal age.")
```

We start by prompting the user to enter their age using the ***input*** function. The ***int*** function is used to convert the input into an integer.

After obtaining the age, we use an ***if*** statement to check if the age is greater than or equal to 18, which is the legal age in many countries. If the condition is ***true***, we print the message "*You are of legal age.*" Otherwise, we print the message "*You are not of legal age.*"

17. Write a program that reads two numbers and tells you which one is bigger.

```
# Prompt the user for the first number
number1 = float(input("Enter the first number: "))

# Prompt the user for the second number
number2 = float(input("Enter the second number: "))

# Compare the numbers and determine the bigger one
if number1 > number2:
    print("The first number is bigger.")
elif number1 < number2:
    print("The second number is bigger.")
else:
    print("Both numbers are equal.")
```

We start by prompting the user to enter the first number using the ***input*** function. The float function is used to convert the input into a floating-point number.

Similarly, we prompt the user to enter the second number.

After obtaining both numbers, we use an ***if-elif-else*** statement to compare the two numbers and determine which one is bigger.

In the first ***if*** condition, we check if ***number1*** is greater than ***number2***. If true, we print the message "*The first number is bigger.*"

In the ***elif*** condition, we check if ***number1*** is less than ***number2***. If true, we print the message "*The second number is bigger.*"

If none of the above conditions are true, it means the numbers are equal, and we print the message "*Both numbers are equal*" in the else block.

The program will display the appropriate message based on the comparison of the two numbers.

18. Write a program that asks the user for three numbers and displays the largest one.

```
# Prompt the user for three numbers
number1 = float(input("Enter the first number: "))
number2 = float(input("Enter the second number: "))
number3 = float(input("Enter the third number: "))

# Determine the largest number using conditional statements
if number1 >= number2 and number1 >= number3:
    largest_number = number1
elif number2 >= number1 and number2 >= number3:
    largest_number = number2
else:
    largest_number = number3

# Display the largest number to the user
print("The largest number is:", largest_number)
```

We start by prompting the user to enter three numbers using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the three numbers, we use conditional statements (***if***, ***elif***, ***else***) to determine the largest number.

In the first ***if*** condition, we check if ***number1*** is greater than or equal to ***number2*** and ***number3***. If true, we assign number1 to the variable ***largest_number***.

In the ***elif*** condition, we check if ***number2*** is greater than or equal to ***number1*** and ***number3***. If true, we assign ***number2*** to ***largest_number***.

If none of the above conditions are true, it means that ***number3*** must be the largest. Therefore, we assign ***number3*** to ***largest_number*** in the ***else*** block.

Finally, we use the ***print*** function to display the largest number to the user.

19. Write a program that reads a number and reports whether it is odd or even.

```
# Prompt the user to enter a number
number = int(input("Enter a number: "))

# Check if the number is odd or even
if number % 2 == 0:
    print("The number is even.")
else:
    print("The number is odd.")
```

We start by prompting the user to enter a number using the ***input*** function. The ***int*** function is used to convert the input into an integer.

After obtaining the number, we use an ***if*** statement to check if the number is divisible by 2. We do this by checking if the remainder (***%***) of the number divided by 2 is equal to 0. If the condition is true, it means the number is even.

If the condition in the ***if*** statement is true, we print the message "*The number is even.*" This indicates that the number is divisible by 2 and therefore even.

If the condition in the ***if*** statement is false, it means the number is not divisible by 2, indicating that it is odd.

In the ***else*** block, we print the message "*The number is odd.*" This confirms that the number is not even and hence must be odd.

The program will display whether the entered number is odd or even based on the condition.

20. Write a program that reads a number and reports whether it is positive, negative or zero.

```
# Prompt the user to enter a number
number = float(input("Enter a number: "))

# Check if the number is positive, negative, or zero
if number > 0:
    print("The number is positive.")
elif number < 0:
    print("The number is negative.")
else:
    print("The number is zero.")
```

We start by prompting the user to enter a number using the ***input*** function. The ***float*** function is used to convert the input into a floating-point number.

After obtaining the number, we use conditional statements (***if, elif, else***) to determine whether the number is positive, negative, or zero.

In the first ***if*** condition, we check if the number is greater than 0. If true, we print the message "*The number is positive.*"

In the ***elif*** condition, we check if the number is less than 0. If true, we print the message "*The number is negative.*"

If none of the above conditions are true, it means the number must be equal to 0. Therefore, we print the message "*The number is zero*" in the ***else*** block.

Finally, the program will display whether the entered number is positive, negative, or zero based on the condition.

21. Make a program that reads the scores of two tests and reports whether the student passed (score greater than or equal to 6) or failed (score less than 6) in each of the tests.

```
# Prompt the user to enter the score of the first test
score1 = float(input("Enter the score of the first test: "))

# Prompt the user to enter the score of the second test
score2 = float(input("Enter the score of the second test: "))

# Check if the student passed or failed in each test
if score1 >= 6:
    print("The student passed the first test.")
else:
    print("The student failed the first test.")

if score2 >= 6:
    print("The student passed the second test.")
else:
    print("The student failed the second test.")
```

We start by prompting the user to enter the score of the first test using the ***input*** function. The ***float*** function is used to convert the input into a floating-point number.

Similarly, we prompt the user to enter the score of the second test.

After obtaining the scores, we use ***if-else*** statements to check if the student passed or failed in each test.

In the first ***if*** condition, we check if ***score1*** is greater than or equal to 6. If true, we print the message "*The student passed the first test*." Otherwise, in the ***else*** block, we print the message "*The student failed the first test.*"

Similarly, in the second ***if*** condition, we check if ***score2*** is greater than or equal to 6. If true, we print the message "*The student passed the second test.*" Otherwise, in the else block,

we print the message "*The student failed the second test*."

The program will display whether the student passed or failed in each test based on the conditions.

22. Make a program that reads the grades of two tests, calculates the simple arithmetic mean, and informs whether the student passed (average greater than or equal to 6) or failed (average less than 6).

```
# Prompt the user to enter the grade of the first test
grade1 = float(input("Enter the grade of the first test: "))

# Prompt the user to enter the grade of the second test
grade2 = float(input("Enter the grade of the second test: "))

# Calculate the average of the grades
average = (grade1 + grade2) / 2

# Check if the student passed or failed based on the average
if average >= 6:
    print("The student passed with an average of", average)
else:
    print("The student failed with an average of", average)
```

We start by prompting the user to enter the grade of the first test using the ***input*** function. The ***float*** function is used to convert the input into a floating-point number.

Similarly, we prompt the user to enter the grade of the second test.

After obtaining the grades, we calculate the average by summing the two grades (***grade1*** and ***grade2***) and dividing the result by 2.

We then use an if statement to check if the average is greater than or equal to 6. If true, we print the message "*The student passed with an average of*" followed by the average value.

If the condition in the ***if*** statement is false, it means the average is less than 6. In this case, we execute the ***else*** block and print the message "*The student failed with an average of*" followed by the average value.

The program will display whether the student passed or failed based on the average grade.

23. Make a program that reads three numbers, and informs if their sum is divisible by 5 or not.

```
# Prompt the user to enter three numbers
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))
number3 = int(input("Enter the third number: "))

# Calculate the sum of the three numbers
sum_of_numbers = number1 + number2 + number3

# Check if the sum is divisible by 5
if sum_of_numbers % 5 == 0:
    print("The sum is divisible by 5.")
else:
    print("The sum is not divisible by 5.")
```

We start by prompting the user to enter three numbers using the ***input*** function. The ***int*** function is used to convert the input into integers.

After obtaining the numbers, we calculate their sum by adding ***number1***, ***number2***, and ***number3*** together.

We then use an if statement to check if the sum of the numbers is divisible by 5. We do this by checking if the remainder (***%***) of the sum divided by 5 is equal to 0.

If the condition in the ***if*** statement is true, it means the sum is divisible by 5. In this case, we print the message "*The sum is divisible by 5.*"

If the condition in the ***if*** statement is false, it means the sum is not divisible by 5. In the ***else*** block, we print the message "*The sum is not divisible by 5.*"

The program will inform whether the sum of the three numbers is divisible by 5 or not.

24. Create a program that reads three numbers and checks if their sum is positive, negative or equal to zero

```
# Prompt the user to enter three numbers
number1 = float(input("Enter the first number: "))
number2 = float(input("Enter the second number: "))
number3 = float(input("Enter the third number: "))

# Calculate the sum of the three numbers
sum_of_numbers = number1 + number2 + number3

# Check if the sum is positive, negative, or zero
if sum_of_numbers > 0:
    print("The sum is positive.")
elif sum_of_numbers < 0:
    print("The sum is negative.")
else:
    print("The sum is zero.")
```

We start by prompting the user to enter three numbers using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the numbers, we calculate their sum by adding ***number1***, ***number2***, and ***number3*** together.

We then use conditional statements (***if***, ***elif***, ***else***) to check if the sum is positive, negative, or zero.

In the first ***if*** condition, we check if ***sum_of_numbers*** is greater than 0. If true, we print the message "*The sum is positive.*"

In the ***elif*** condition, we check if ***sum_of_numbers*** is less than 0. If true, we print the message "*The sum is negative.*"

If none of the above conditions are true, it means the sum must be equal to 0. Therefore, we print the message "*The sum is zero*" in the else block.

The program will inform whether the sum of the three numbers is positive, negative, or zero.

25. Make a program that reads three numbers, and displays them on the screen in ascending order.

```
# Prompt the user to enter three numbers
number1 = float(input("Enter the first number: "))
number2 = float(input("Enter the second number: "))
number3 = float(input("Enter the third number: "))

# Check the order of the numbers using if statements
if number1 <= number2 and number1 <= number3:
    if number2 <= number3:
        print("The numbers in ascending order are:", number1, number2, number3)
    else:
        print("The numbers in ascending order are:", number1, number3, number2)
elif number2 <= number1 and number2 <= number3:
    if number1 <= number3:
        print("The numbers in ascending order are:", number2, number1, number3)
    else:
        print("The numbers in ascending order are:", number2, number3, number1)
else:
    if number1 <= number2:
        print("The numbers in ascending order are:", number3, number1, number2)
    else:
        print("The numbers in ascending order are:", number3, number2, number1)
```

We start by prompting the user to enter three numbers using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the numbers, we use ***if*** statements to compare the numbers and determine their order.

We first compare ***number1*** with ***number2*** and ***number3***. If ***number1*** is less than or equal to both ***number2*** and ***number3***, it means ***number1*** is the smallest number. In this case, we further compare ***number2*** and ***number3*** to determine their order.

Similarly, we check if ***number2*** is less than or equal to both ***number1*** and ***number3***. If true, it means ***number2*** is the smallest number. In this case, we further compare ***number1*** and ***number3*** to determine their order.

If none of the above conditions are true, it means ***number3*** is the smallest number. In this case, we compare ***number1*** and ***number2*** to determine their order.

Based on the comparisons, we use nested ***if*** statements to print the numbers in ascending order accordingly.

The program will display the numbers entered by the user in ascending order.

Without using conditionals (using ***list*** and ***sort***)

```
# Prompt the user to enter three numbers
number1 = float(input("Enter the first number: "))
number2 = float(input("Enter the second number: "))
number3 = float(input("Enter the third number: "))

# Create a list to store the numbers
numbers = [number1, number2, number3]

# Sort the list in ascending order
numbers.sort()

# Display the numbers in ascending order
print("The numbers in ascending order are:", numbers)
```

We start by prompting the user to enter three numbers using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the numbers, we create a list called ***numbers*** and store the three numbers in it.

We use the ***sort()*** method to sort the elements in the numbers list in ascending order.

Finally, we display the numbers in ascending order using the ***print*** function. We concatenate the string "*The numbers in ascending order are:*" with the ***numbers*** list, which will automatically be converted to a string representation.

The program will display the numbers entered by the user in ascending order.

26. Make a program that reads the age of three people and how many of them are of legal age (age 18 or older).

```
# Prompt the user to enter the age of three people
age1 = int(input("Enter the age of the first person: "))
age2 = int(input("Enter the age of the second person: "))
age3 = int(input("Enter the age of the third person: "))

# Initialize a variable to keep track of the count of legal age individuals
legal_age_count = 0

# Check if each person is of legal age
if age1 >= 18:
    legal_age_count += 1
if age2 >= 18:
    legal_age_count += 1
if age3 >= 18:
    legal_age_count += 1

# Display the count of legal age individuals
print("Out of the three people,", legal_age_count, "person(s) is/are of legal age.")
```

We start by prompting the user to enter the age of three people using the ***input*** function. The ***int*** function is used to convert the input into integers.

After obtaining the ages, we initialize a variable called ***legal_age_count*** to keep track of the count of legal age individuals. We set it to 0 initially.

We use if statements to check if each person's age is 18 or older. If true, we increment the ***legal_age_count*** variable by 1.

After checking the ages of all three people, we display the count of legal age individuals using the ***print*** function.

The program will inform how many out of the three people are of legal age.

27. Write a program that reads three numbers and tells you if they can be the sides of a triangle (the sum of two sides must always be greater than the third side).

```
# Prompt the user to enter three numbers
side1 = float(input("Enter the length of the first side: "))
side2 = float(input("Enter the length of the second side: "))
side3 = float(input("Enter the length of the third side: "))

# Check if the three sides can form a triangle
if side1 + side2 > side3 and side1 + side3 > side2 and side2 + side3 > side1:
    print("The three sides can form a triangle.")
else:
    print("The three sides cannot form a triangle.")
```

We start by prompting the user to enter the lengths of three sides of a triangle using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the side lengths, we use ***if*** statements to check if the sum of any two sides is greater than the third side. According to the triangle inequality theorem, this condition must be true for the sides to form a triangle.

If the condition is true for all three pairs of sides, we print the message "*The three sides can form a triangle.*"

If the condition is not true for any of the pairs of sides, we print the message "*The three sides cannot form a triangle.*"

The program will inform whether the three given side lengths can form a triangle or not.

28. Make a program that reads the year of birth of a person and informs if he is able to vote (age greater than or equal to 16 years old).

```python
# Prompt the user to enter the year of birth
year_of_birth = int(input("Enter the year of birth: "))

# Calculate the current year
import datetime
current_year = datetime.datetime.now().year

# Calculate the age of the person
age = current_year - year_of_birth

# Check if the person is eligible to vote
if age >= 16:
    print("You are eligible to vote!")
else:
    print("You are not eligible to vote yet.")
```

We start by prompting the user to enter the year of birth using the ***input*** function. The ***int*** function is used to convert the input into an integer.

We import the ***datetime*** module to get the current year using the ***now().year*** method. This allows us to calculate the age accurately.

We calculate the age of the person by subtracting the year of birth from the current year.

We use an if statement to check if the calculated age is greater than or equal to 16. If true, we print the message "*You are eligible to vote!*" indicating that the person is eligible to vote.

If the condition is not true, we print the message "*You are not eligible to vote yet.*"

The program will inform whether the person is eligible to vote based on their year of birth.

29. Make a program that reads a person's age and informs if he is not able to vote (age less than 16 years old), if he is able to vote but is not obligated (16, 17 years old, or age equal to or greater than 70 years), or if it is obligatory (18 to 69 years old).

*These conditions are in accordance with Brazilian legislation.

```python
# Prompt the user to enter their age
age = int(input("Enter your age: "))

# Check the age range and inform the voting eligibility
if age < 16:
    print("You are not able to vote.")
elif age >= 18 and age <= 69:
    print("Voting is obligatory for you.")
else:
    print("You are able to vote, but it is not obligatory.")
```

We start by prompting the user to enter their age using the ***input*** function. The ***int*** function is used to convert the input into an integer.

We use ***if*** and ***elif*** statements to check the age range and inform the voting eligibility accordingly.

If the age is less than 16, we print the message "*You are not able to vote*" indicating that the person is not eligible to vote.

If the age is between 18 and 69 (inclusive), we print the message "*Voting is obligatory for you*" indicating that the person is required to vote.

For all other age values (16, 17, and age greater than or equal to 70), we print the message "*You are able to vote, but it is not obligatory*" indicating that the person is eligible to vote but not obligated to do so.

The program will inform the voting eligibility based on the entered age.

30. Make a program that reads three grades from a student and reports whether he passed (final grade greater than or equal to 7), failed (final grade less than 4) or was in recovery (final grade between 4 and 7).

```
# Prompt the user to enter three grades
grade1 = float(input("Enter the first grade: "))
grade2 = float(input("Enter the second grade: "))
grade3 = float(input("Enter the third grade: "))

# Calculate the average grade
average_grade = (grade1 + grade2 + grade3) / 3

# Check the final result based on the average grade
if average_grade >= 7:
    result = "Pass"
elif average_grade < 4:
    result = "Fail"
else:
    result = "Recovery"

# Display the final result
print("The student's final result is:", result)
```

We start by prompting the user to enter three grades using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the grades, we calculate the average grade by summing up the three grades and dividing by 3.

We use ***if***, ***elif***, and ***else*** statements to check the final result based on the average grade.

If the average grade is greater than or equal to 7, we set the result as "*Pass*".

If the average grade is less than 4, we set the result as "*Fail*".

For all other average grade values (between 4 and 7), we set the result as "*Recovery*".

We use the *print* function to display the final result to the user.

The program will inform the final result of the student as pass, fail, or in recovery based on the average grade.

31. Write a program that asks for the name of a day of the week and displays whether it is a weekday (Monday to Friday) or a weekend day (Saturday and Sunday).

```
# Prompt the user to enter the name of a day
day = input("Enter the name of a day: ")

# Convert the input to lowercase for case-insensitive comparison
day = day.lower()

# Check if the day is a weekday or a weekend day
if day == "saturday" or day == "sunday":
    result = "Weekend day"
else:
    result = "Weekday"

# Display the result
print(f"{day.capitalize()} is a {result}.")
```

We start by prompting the user to enter the name of a day using the ***input*** function.

We convert the input to lowercase using the ***lower()*** method to perform a case-insensitive comparison.

We use an ***if*** statement to check if the input day is equal to "*saturday*" or "*sunday*". If true, we set the result as "*Weekend day*".

If the input day is not "*saturday*" or "*sunday*", the ***else*** block is executed, and we set the result as "*Weekday*".

We use the ***print*** function to display the result to the user. The ***capitalize()*** method is used to capitalize the first letter of the input day for better presentation.

The program will inform whether the input day is a weekday or a weekend day based on the comparison.

Note that the program assumes the user will input valid day names. If the user enters invalid day names or misspelled inputs, the program will treat them as weekdays. You can add additional input validation or error handling to handle such cases if needed.

32. Write a program that asks for a person's height and weight and calculates their body mass index (BMI), displaying the corresponding category (underweight, normal weight, overweight, obese, severely obese).

```
# Prompt the user to enter their height and weight
height = float(input("Enter your height in meters: "))
weight = float(input("Enter your weight in kilograms: "))

# Calculate the BMI using the formula BMI = weight / (height^2)
bmi = weight / (height ** 2)

# Determine the corresponding BMI category
if bmi < 18.5:
    category = "Underweight"
elif bmi < 25:
    category = "Normal weight"
elif bmi < 30:
    category = "Overweight"
elif bmi < 35:
    category = "Obese"
else:
    category = "Severely obese"

# Display the BMI and category
print("Your BMI is:", bmi)
print("Category:", category)
```

We start by prompting the user to enter their height and weight using the ***input*** function. The ***float*** function is used to convert the input into floating-point numbers.

After obtaining the height and weight, we calculate the BMI using the formula ***BMI = weight / (height^2)***, where the height is squared using the **** exponentiation operator.

We use ***if***, ***elif***, and ***else*** statements to determine the corresponding BMI category based on the calculated BMI value.

If the BMI is less than 18.5, we set the category as "*Underweight*".

If the BMI is between 18.5 and less than 25, we set the category as "*Normal weight*".

If the BMI is between 25 and less than 30, we set the category as "*Overweight*".

If the BMI is between 30 and less than 35, we set the category as "*Obese*".

For all other BMI values equal to or greater than 35, we set the category as "*Severely obese*".

We use the ***print*** function to display the calculated BMI and the corresponding category to the user.

The program will calculate the BMI and inform the user about their BMI category.

33. Write a program that asks for an integer and checks if it is divisible by 3 and 5 at the same time.

```
# Prompt the user to enter an integer
number = int(input("Enter an integer: "))

# Check if the number is divisible by 3 and 5
if number % 3 == 0 and number % 5 == 0:
    result = "divisible by 3 and 5"
else:
    result = "not divisible by 3 and 5"

# Display the result
print(f"The number {number} is {result}.")
```

We start by prompting the user to enter an integer using the ***input*** function. The ***int*** function is used to convert the input into an integer.

After obtaining the number, we use the ***%*** modulus operator to check if the number is divisible by 3 and 5 at the same time. If the remainder of the division by both 3 and 5 is 0, then the number is divisible by both.

We use an ***if*** statement to check if the condition ***number % 3 == 0 and number % 5 == 0*** is true. If true, we set the ***result*** as "*divisible by 3 and 5*".

If the condition is not true, the ***else*** block is executed, and we set the ***result*** as "*not divisible by 3 and 5*".

We use the ***print*** function to display the result to the user. The ***f-string*** is used to format the output string with the entered ***number*** and the ***result***.

The program will inform the user whether the entered number is divisible by both 3 and 5 or not.

34. Create a program that asks for a person's age and displays whether they are a child (0-12 years old), teenager (13-17 years old), adult (18-59 years old), or elderly (60 years old or older).

```
# Prompt the user to enter their age
age = int(input("Enter your age: "))

# Check the age range and assign the corresponding category
if age <= 12:
    category = "Child"
elif age <= 17:
    category = "Teenager"
elif age <= 59:
    category = "Adult"
else:
    category = "Elderly"

# Display the category
print("You are a", category + ".")
```

We start by prompting the user to enter their age using the ***input*** function. The ***int*** function is used to convert the input into an integer.

After obtaining the age, we use ***if***, ***elif***, and ***else*** statements to check the age range and assign the corresponding category.

If the age is less than or equal to 12, we set the category as "*Child*".

If the age is greater than 12 and less than or equal to 17, we set the category as "*Teenager*".

If the age is greater than 17 and less than or equal to 59, we set the category as "*Adult*".

For all other age values greater than 59, we set the category as "*Elderly*".

We use the ***print*** function to display the category to the user.

The program will determine the category based on the entered age.

35. Make a program that asks for two numbers and displays if the first is divisible by the second

```
# Prompt the user to enter the first number
num1 = int(input("Enter the first number: "))

# Prompt the user to enter the second number
num2 = int(input("Enter the second number: "))

# Check if the first number is divisible by the second number
if num1 % num2 == 0:
    result = "divisible"
else:
    result = "not divisible"

# Display the result
print(f"The first number is {result} by the second number.")
```

We start by prompting the user to enter the first number using the ***input*** function. The ***int*** function is used to convert the input into an integer.

After obtaining the first number, we prompt the user to enter the second number using the ***input*** function. Again, we convert the input into an integer using the ***int*** function.

We use the ***%*** modulus operator to check if the first number is divisible by the second number. If the remainder of the division is 0, then the first number is divisible by the second number.

We use an ***if*** statement to check if the condition ***num1 % num2 == 0*** is true. If true, we set the result as "*divisible*".

If the condition is not true, the ***else*** block is executed, and we set the result as "*not divisible*".

We use the ***print*** function to display the result to the user. The ***f-string*** is used to format the output string with the ***result***.

The program will inform the user whether the first number is divisible by the second number or not.

Repeat Loops

In Python, you can use repeat loops to execute a block of code multiple times. There are two main types of repeat loops: the ***while*** loop and the ***for*** loop.

while loop

The ***while*** loop repeatedly executes a block of code as long as a specified condition is ***True***. The general syntax of a ***while*** loop is as follows:

```
condition = True

while condition:
    # Code to be executed
```

The condition is evaluated at the beginning of each iteration. If the condition is ***True***, the code block is executed. Afterward, the condition is checked again, and the loop continues until the condition becomes ***False***.

Here's an example of a ***while*** loop:

```
count = 0
while count < 5:
    print("Count:", count)
    count += 1
```

In this example, the ***while*** loop executes the code block as long as ***count*** is less than 5. The loop prints the current value of count and increments it by 1. The loop continues until ***count*** reaches 5, at which point the condition becomes ***False***, and the loop terminates.

for loop

The ***for*** loop iterates over a sequence (such as a list, tuple, or string) or any iterable object. It executes a block of code for each item in the sequence. The general syntax of a ***for*** loop is as follows:

```
for item in sequence:
    # Code to be executed
```

Here's an example of a ***for*** loop:

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

In this example, the ***for*** loop iterates over each item in the fruits ***list***. For each iteration, the current item is assigned to the variable ***fruit***, and the code block is executed. The loop prints each fruit on a separate line.

The number of loop iterations in a ***for*** loop is determined by the length of the sequence. Once all the items have been processed, the loop terminates.

You can also combine ***for*** loops with the ***range()*** function to iterate a specific number of times. For example:

```
for i in range(5):
    print(i)
```

This ***for*** loop will iterate five times, printing the values 0, 1, 2, 3, and 4.

Repeat loops provide a way to automate repetitive tasks and process data in a controlled manner. Understanding how to use ***while*** and for ***loops*** effectively is an essential skill in Python programming.

Função *range*

In Python, the ***range()*** function is used to generate a sequence of numbers. It is commonly used in combination with loops, particularly ***for*** loops, to iterate over a specific range of values. The ***range()*** function returns an iterable object that represents the sequence of numbers.

The syntax of the ***range()*** function is as follows:

range(start, stop, step)

The ***range()*** function can take up to three arguments:

start (optional): Specifies the starting value of the sequence (inclusive). If not provided, the default value is 0.

stop (required): Specifies the stopping value of the sequence (exclusive). The sequence will go up to, but not include, this value.

step (optional): Specifies the step or increment between each number in the sequence. If not provided, the default value is 1.

Here are a few examples of using the ***range()*** function:

Generating a sequence of numbers from 0 to 4 (exclusive):

```
for i in range(5):
    print(i)
```

Generating a sequence of even numbers from 0 to 10 (exclusive):

```
for i in range(0, 10, 2):
    print(i)
```

Generating a sequence of numbers in reverse order from 10 down to 1 (exclusive):

```
for i in range(10, 0, -1):
    print(i)
```

Note that the ***range()*** function does not create a list of numbers immediately. Instead, it generates numbers on the fly as you iterate over them. If you need to create a list, you can pass the ***range()*** function to the ***list()*** constructor:

```
numbers = list(range(5))
print(numbers)  # [0, 1, 2, 3, 4]
```

The ***range()*** function is a powerful tool for creating sequences of numbers, especially for iterating over a specific range in loops. It provides flexibility with the ability to specify the starting value, stopping value, and step size of the sequence.

Now, let's get to the exercises.

36. Write a program that displays the numbers 1 through 10 using a loop.

Using ***while***

```
# Initialize a variable to start at 1
num = 1

# Use a while loop to iterate until the number reaches 11
while num <= 10:
    print(num)
    num += 1
```

We start by initializing a variable ***num*** with a value of 1. This variable will keep track of the current number being printed.

We use a ***while*** loop to iterate as long as the condition ***num*** <= ***10*** is true. This condition ensures that the loop will execute until the number reaches 11.

Inside the loop, we print the value of num using the ***print*** function.

After printing the number, we increment the value of ***num*** by 1 using the ***num*** += ***1*** statement. This ensures that the loop progresses to the next number.

The loop continues executing until the value of num reaches 11, at which point the condition ***num*** <= ***10*** becomes false and the loop terminates.

By using the ***while*** loop and incrementing the value of num inside the loop, we can repeatedly print the numbers from 1 to 10 until the desired condition is met.

Using *for*

```
# Use a for loop to iterate through the numbers 1 to 10
for num in range(1, 11):
    print(num)
```

We use a ***for*** loop to iterate through the numbers 1 to 10. The ***range*** function is used to generate a sequence of numbers from 1 to 10 (inclusive). The ***range*** function is called with two arguments: the ***start*** value (1 in this case) and the ***end*** value (11 in this case, exclusive).

Inside the loop, the variable ***num*** takes on each value in the range from 1 to 10 in each iteration.

The ***print*** function is used to display the value of ***num*** on the console.

The loop continues for each number in the range, displaying the numbers 1 through 10.

Once the loop finishes iterating through all the numbers, the program ends.

By using the ***for*** loop and the ***range*** function, we can easily iterate through a sequence of numbers and perform a specific action for each number. In this case, we are simply printing each number, but you can modify the code inside the loop to perform any desired operations on the numbers.

37. Write a program that displays all numbers from 1 to 100

```
# Use a for loop to iterate through the numbers 1 to 100
for num in range(1, 101):
    print(num)
```

We use a ***for*** loop to iterate through the numbers 1 to 100. The ***range*** function is used to generate a sequence of numbers from 1 to 100 (inclusive). The range function is called with two arguments: the ***start*** value (1 in this case) and the ***end*** value (101 in this case, exclusive).

Inside the loop, the variable ***num*** takes on each value in the range from 1 to 100 in each iteration.

The ***print*** function is used to display the value of ***num*** on the console.

The loop continues for each number in the range, displaying the numbers from 1 to 100.

Once the loop finishes iterating through all the numbers, the program ends.

38. Write a program that prints all even numbers from 1 to 100.

Solution 1

```
# Initialize a variable to start at 1
num = 1

# Use a while loop to iterate until the number reaches 100
while num <= 100:
    if num % 2 == 0:
        print(num)
    num += 1
```

We start by initializing a variable ***num*** with a value of 1. This variable will keep track of the current number being checked.

We use a ***while*** loop to iterate as long as the condition ***num <= 100*** is true. This ensures that the loop will execute until the number reaches 100.

Inside the loop, we use an ***if*** statement to check if the number ***num*** is even. We do this by checking if the remainder of dividing ***num*** by 2 is 0 using the expression ***num % 2 == 0***.

If the condition is true (meaning ***num*** is even), we use the ***print*** function to display the value of ***num*** on the console.

After checking and possibly printing the number, we increment the value of num by 1 using the ***num += 1*** statement. This ensures that the loop progresses to the next number.

The loop continues executing until the value of num reaches 100, at which point the condition ***num <= 100*** becomes false and the loop terminates.

Solution 2

```
# Initialize a variable to start at 2
num = 2

# Use a while loop to iterate until the number reaches 100
while num <= 100:
    print(num)
    num += 2
```

We start by initializing a variable ***num*** with a value of 2. This variable will keep track of the current even number being printed.

We use a while loop to iterate as long as the condition ***num*** <= ***100*** is true. This ensures that the loop will execute until the number reaches 100.

Inside the loop, we use the ***print*** function to display the value of ***num*** on the console.

After printing the number, we increment the value of ***num*** by 2 using the ***num += 2*** statement. This ensures that the loop progresses to the next even number by skipping odd numbers.

The loop continues executing until the value of ***num*** reaches 100, at which point the condition ***num*** <= ***100*** becomes false and the loop terminates.

By incrementing ***num*** by 2 in each iteration, we are directly skipping the odd numbers and printing only the even numbers from 1 to 100.

Solution 3

```
# Use a for loop to iterate through the even numbers from 2 to 100
for num in range(2, 101, 2):
    print(num)
```

We use a ***for*** loop to iterate through the even numbers from 2 to 100. The ***range*** function is used with three arguments: the ***start*** value (2 in this case), the ***end*** value (101 in this case, exclusive), and the ***step*** value (2 in this case).

Inside the loop, the variable ***num*** takes on each even number in the range from 2 to 100 in each iteration.

The ***print*** function is used to display the value of ***num*** on the console.

The loop continues for each even number in the range, printing the even numbers from 2 to 100.

Once the loop finishes iterating through all the even numbers, the program ends.

By using the ***for*** loop and the ***range*** function with a step value of 2, we can easily iterate through the even numbers within the desired range and perform a specific action for each even number. In this case, we are simply printing each even number, but you can modify the code inside the loop to perform any desired operations on the even numbers.

39. Write a program that displays even numbers 1 to 50 and odd numbers 51 to 100 using a repeating loop.

```
# Initialize a variable to start at 1
num = 1

# Use a while loop to iterate until the number reaches 100
while num <= 100:
    if num <= 50:
        if num % 2 == 0:
            print("Even:", num)
    else:
        if num % 2 != 0:
            print("Odd:", num)
    num += 1
```

We start by initializing a variable ***num*** with a value of 1. This variable will keep track of the current number being checked.

We use a ***while*** loop to iterate as long as the condition ***num*** <= ***100*** is true. This ensures that the loop will execute until the number reaches 100.

Inside the loop, we have nested if statements to check the value of ***num***.

The first ***if*** statement checks if ***num*** is less than or equal to 50. If it is, we proceed to the inner ***if*** statement.

The inner ***if*** statement checks if num is even by using the expression ***num % 2 == 0***.

If the condition is true (meaning ***num*** is even and less than or equal to 50), we print the message "*Even:*" followed by the value of ***num***.

If the value of num is greater than 50, we proceed to the ***else*** block.

The ***else*** block is executed when ***num*** is greater than 50. Here, we check if num is odd by using the expression ***num % 2 != 0***.

If the condition is true (meaning ***num*** is odd and greater than 50), we print the message "*Odd:*" followed by the value of ***num***.

After checking and possibly printing the number, we increment the value of ***num*** by 1 using the ***num += 1*** statement. This ensures that the loop progresses to the next number.

The loop continues executing until the value of ***num*** reaches 100, at which point the condition ***num*** <= ***100*** becomes false and the loop terminates.

40. Create a program that prompts the user for a number and displays the table of that number using a loop.

```
# Prompt the user for a number
number = int(input("Enter a number: "))

# Use a for loop to iterate through the range 1 to 11
for i in range(1, 11):
    # Calculate the product of the number and the current iteration value
    product = number * i

    # Display the multiplication table entry
    print(number, "x", i, "=", product)
```

We start by prompting the user to enter a number using the ***input*** function. The ***int*** function is used to convert the user input from a string to an integer. The entered number is then stored in the ***number*** variable.

We use a ***for*** loop to iterate through the ***range*** from 1 to 11. This range includes numbers from 1 to 10, as the upper limit (11) is exclusive.

Inside the loop, we calculate the product of the number and the current iteration value (***i***) and store it in the ***product*** variable.

We use the ***print*** function to display the multiplication table entry in the format: "***number*** *x* ***i*** = ***product***".

The loop continues for each iteration, displaying the multiplication table entry for each value of ***i***.

Once the loop finishes iterating through all values from 1 to 10, the program ends.

Using ***while***

```
# Prompt the user for a number
number = int(input("Enter a number: "))

# Initialize a counter variable
i = 1

# Use a while loop to iterate until the counter reaches 11
while i <= 10:
    # Calculate the product of the number and the current iteration value
    product = number * i

    # Display the multiplication table entry
    print(number, "x", i, "=", product)

    # Increment the counter by 1
    i += 1
```

We start by prompting the user to enter a number using the ***input*** function. The ***int*** function is used to convert the user input from a string to an integer. The entered number is then stored in the ***number*** variable.

We initialize a counter variable ***i*** with a value of 1. This variable will keep track of the current iteration.

We use a ***while*** loop with the condition ***i*** <= ***10*** to iterate until the counter reaches 11. This ensures that the loop will execute 10 times, corresponding to the range 1 to 10.

Inside the loop, we calculate the product of the number and the current iteration value (***i***) and store it in the ***product*** variable.

We use the ***print*** function to display the multiplication table entry in the format: "***number*** *x* ***i*** = ***product***".

After printing the entry, we increment the value of the counter ***i*** by 1 using the ***i*** += ***1*** statement. This ensures that the loop progresses to the next iteration.

The loop continues executing until the counter reaches 11, at which point the condition ***i*** <= ***10*** becomes false and the loop terminates.

41. Create a program that prompts the user for a number and displays the table of that number using a loop.

```
# Use a nested for loop to iterate through the range 1 to 11
  #for both multiplicands
for i in range(1, 11):
    for j in range(1, 11):
        # Calculate the product of the two numbers
        product = i * j

        # Display the multiplication table entry
        print(i, "x", j, "=", product)

    # Print a separator between each multiplication table
    print("-" * 20)
```

We use a nested ***for*** loop to iterate through the ***range*** from 1 to 11 for both multiplicands (***i*** and ***j***). This range includes numbers from 1 to 10, as the upper limit (11) is exclusive.

Inside the nested loop, we calculate the product of the two numbers (***i*** and ***j***) and store it in the ***product*** variable.

We use the ***print*** function to display the multiplication table entry in the format: "***i x j = product***".

After printing each entry, the inner loop continues iterating until all values of ***j*** from 1 to 10 have been processed.

Once the inner loop finishes iterating for a particular ***i*** value, we print a separator line (***"-" * 20***) to visually separate each multiplication table.

The outer loop continues iterating for each value of ***i***, repeating the inner loop process and printing the corresponding multiplication table.

Once the outer loop finishes iterating for all values of ***i*** from 1 to 10, the program ends.

By using a nested ***for*** loop and iterating through the range from 1 to 11 for both multiplicands, we can easily calculate and display the multiplication table for numbers from 1 to 10. The inner loop calculates the product of the two numbers for each combination, and the corresponding entry is printed on the console. The outer loop ensures that this process is repeated for each value of ***i***, generating the complete multiplication table.

42. Write a program that asks the user for a number N and displays the sum of all numbers from 1 to N.

```
# Prompt the user for a number
N = int(input("Enter a number: "))

# Initialize a variable to hold the sum
sum_of_numbers = 0

# Use a for loop to iterate from 1 to N (inclusive)
for num in range(1, N + 1):
    # Add the current number to the sum
    sum_of_numbers += num

# Display the sum of the numbers
print("The sum of numbers from 1 to", N, "is:", sum_of_numbers)
```

We start by prompting the user to enter a number using the ***input*** function. The ***int*** function is used to convert the user input from a string to an integer. The entered number is then stored in the variable ***N***.

We initialize a variable ***sum_of_numbers*** to hold the sum of all numbers from 1 to ***N***. We set its initial value to 0.

We use a ***for*** loop with the range function to iterate from 1 to ***N*** (inclusive). The range function generates a sequence of numbers starting from 1 up to ***N*** (not including ***N+1***).

Inside the loop, for each iteration, the current number (***num***) is added to the ***sum_of_numbers*** using the += operator. This accumulates the sum of all numbers.

Once the loop finishes iterating over all numbers from 1 to ***N***, we have the total sum stored in the ***sum_of_numbers*** variable.

Finally, we use the print function to display the message "*The sum of numbers from 1 to* ***N*** *is:* ***sum_of_numbers***", where ***N*** is the user-entered number and ***sum_of_numbers*** is the calculated sum.

By using a for loop and the range function, we iterate through all numbers from 1 to ***N***, accumulating their sum. The program then displays the final sum on the console.

Using ***while***

```
# Prompt the user for a number
N = int(input("Enter a number: "))

# Initialize variables
sum_of_numbers = 0
num = 1

# Use a while loop to iterate until num reaches N
while num <= N:
    # Add the current number to the sum
    sum_of_numbers += num

    # Increment the number
    num += 1

# Display the sum of the numbers
print("The sum of numbers from 1 to", N, "is:", sum_of_numbers)
```

Similar to the previous solution, we prompt the user to enter a number ***N*** using the ***input*** function and convert it to an integer.

We initialize the variables ***sum_of_numbers*** and ***num***. ***sum_of_numbers*** holds the sum of all numbers, and num starts with the value 1.

We use a ***while*** loop with the condition ***num*** <= ***N***. This loop will continue until the value of num exceeds ***N***.

Inside the loop, we add the current number (***num***) to the ***sum_of_numbers*** using the += operator, just like in the previous solution.

After adding the number to the ***sum***, we increment ***num*** by 1 using ***num*** += ***1***. This ensures that the loop progresses to the next number.

The loop continues until num reaches ***N***, and then it terminates.

Finally, we use the ***print*** function to display the message "*The sum of numbers from 1 to* ***N*** *is:* ***sum_of_numbers***", where ***N*** is the user-entered number and ***sum_of_numbers*** is the calculated ***sum***.

By using a ***while*** loop and incrementing the number (***num***) inside the loop, we iterate through all numbers from 1 to ***N*** and accumulate their sum.

The program then displays the final sum on the console.

43. Write a program that calculates and displays the sum of even numbers from 1 to 100 using a repeating loop.

```
# Initialize variables
sum_of_evens = 0
num = 2

# Use a while loop to iterate until num reaches 100
while num <= 100:
    # Add the current number to the sum if it is even
    sum_of_evens += num

    # Increment the number by 2 to get the next even number
    num += 2

# Display the sum of even numbers
print("The sum of even numbers from 1 to 100 is:", sum_of_evens)
```

We initialize the variables ***sum_of_evens*** and ***num***. ***sum_of_evens*** will hold the sum of even numbers, and num starts with the value 2, which is the first even number.

We use a ***while*** loop with the condition ***num*** <= ***100***. This loop will continue until the value of num exceeds 100.

Inside the loop, we add the current number (***num***) to the ***sum_of_evens*** using the += operator. Since we want to calculate the sum of even numbers, we only add even numbers to the sum.

After adding the number to the sum, we increment ***num*** by 2 using ***num*** += ***2***. This ensures that we get the next even number in each iteration.

The loop continues until ***num*** reaches 100, and then it terminates.

Finally, we use the ***print*** function to display the message "*The sum of even numbers from 1 to 100 is:* ***sum_of_evens***", where ***sum_of_evens*** is the calculated sum.

By using a ***while*** loop and incrementing the number (***num***) by 2 in each iteration, we iterate through even numbers from 2 to 100 and accumulate their sum. The program then displays the final sum on the console.

Using ***for in range*** with ***step = 2***

```
# Initialize variable
sum_of_evens = 0

# Use a for loop with range and step 2 to iterate through
  #even numbers from 2 to 100
for num in range(2, 101, 2):
    # Add the current number to the sum
    sum_of_evens += num

# Display the sum of even numbers
print("The sum of even numbers from 1 to 100 is:", sum_of_evens)
```

We initialize the variable ***sum_of_evens*** to hold the sum of even numbers.

We use a ***for*** loop with the ***range*** function and specify the starting point as 2, ending point as 101 (exclusive), and step as 2. This ensures that the loop iterates through even numbers from 2 to 100.

Inside the loop, the ***num*** variable takes the values of each even number in each iteration.

We add the current number (***num***) to the ***sum_of_evens*** using the += operator.

The loop continues until it iterates through all even numbers from 2 to 100.

Finally, we use the ***print*** function to display the message "*The sum of even numbers from 1 to 100 is:* ***sum_of_evens***", where ***sum_of_evens*** is the calculated sum.

By using a ***for*** loop with the ***range*** function and a step of 2, we iterate through even numbers from 2 to 100 and accumulate their sum. The program then displays the final sum on the console.

44. Write a program that calculates and displays the value of the power of a number entered by the user raised to an exponent also entered by the user, using repetition loops.

```
# Prompt the user for the base number and the exponent
base = int(input("Enter the base number: "))
exponent = int(input("Enter the exponent: "))

# Initialize the result variable to hold the calculated value
result = 1

# Use a for loop to iterate from 1 to the exponent
for _ in range(1, exponent + 1):
    # Multiply the result by the base number in each iteration
    result *= base

# Display the calculated result
print("The result of", base, "raised to the power of", exponent, "is:", result)
```

We use the ***input*** function to prompt the user to enter the ***base*** number and the ***exponent***. The ***int*** function is used to convert the input values to integers.

We initialize the ***result*** variable to 1. This variable will hold the calculated value of the base raised to the exponent.

We use a ***for*** loop with the range function to iterate from 1 to the ***exponent*** (inclusive). This loop will execute the specified number of times.

Inside the loop, we multiply the ***result*** by the ***base*** in each iteration using the ****=*** operator. This effectively calculates the power of the base.

The loop continues until it completes the specified number of iterations (equal to the ***exponent***).

Finally, we use the ***print*** function to display the message "*The result of* ***base*** *raised to the power of* ***exponent*** *is:* ***result***", where ***base*** is the base number entered by the user, ***exponent*** is the exponent entered by the user, and ***result*** is the calculated value.

45. Write a program that asks the user for a number N and says whether it is prime or not.

```
# Prompt the user for a number
number = int(input("Enter a number: "))

# Check if the number is less than 2 (not prime)
if number < 2:
    is_prime = False
else:
    is_prime = True

    # Check if the number is divisible by any integer
    # from 2 to the square root of the number
    for i in range(2, int(number ** 0.5) + 1):
        if number % i == 0:
            is_prime = False
            break

# Display the result
if is_prime:
    print(number, "is a prime number.")
else:
    print(number, "is not a prime number.")
```

We use the ***input*** function to prompt the user to enter a number. The ***int*** function is used to convert the input value to an integer.

We initialize the variable ***is_prime*** to ***False*** if the number is less than 2, as numbers less than 2 are not prime.

If the number is greater than or equal to 2, we set ***is_prime*** to ***True*** initially.

We use a ***for*** loop with the ***range*** function to iterate from 2 to the square root of the number (inclusive). This loop checks if the number is divisible by any integer in that range.

Inside the loop, we use the modulo operator ***%*** to check if the number is divisible by the current value of ***i***. If it is divisible, we set ***is_prime*** to ***False*** and break out of the loop.

After the loop, we have determined whether the number is prime or not based on the value of ***is_prime***.

Finally, we use the ***print*** function to display the appropriate message based on the value of ***is_prime***.

46. Write a program that prompts the user for a number N and displays all prime numbers less than N.

```
# Prompt the user for a number
n = int(input("Enter a number: "))

# Iterate through numbers from 2 to n
for num in range(2, n):
    is_prime = True

    # Check if the current number is divisible by any integer from 2
    # to the square root of the number
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            is_prime = False
            break

    # Display the prime number
    if is_prime:
        print(num, end=" ")

# Add a new line for better output formatting
print()
```

We use the ***input*** function to prompt the user to enter a number. The ***int*** function is used to convert the input value to an integer.

We use a ***for*** loop to iterate through numbers from 2 to ***n*** (exclusive), as 1 is not a prime number.

Inside the loop, we initialize the variable ***is_prime*** to ***True*** for each number.

We use a nested ***for*** loop to check if the current number is divisible by any integer from 2 to the square root of the number (inclusive).

Inside the nested loop, we use the modulo operator ***%*** to check if the number is divisible by the current value of ***i***. If it is divisible, we set ***is_prime*** to ***False*** and break out of the loop.

After the nested loop, if ***is_prime*** is still ***True***, it means the current number is prime. We use the ***print*** function to display the prime number, separated by a space. We use the ***end=" "*** parameter to ensure the numbers are printed on the same line.

The outer loop continues iterating through the remaining numbers.

After the loop, we use the ***print*** function without any arguments to add a new line, which improves the output formatting.

The solution checks for prime numbers by iterating through numbers from 2 to ***n*** and checking for divisibility. If a number is not divisible by any integer from 2 to the square root of the number, it is considered prime. The program then displays all the prime numbers less than ***n*** on the console.

47. Create a program that displays the first N prime numbers, where N is informed by the user, using a loop.

```
# Prompt the user for a number
N = int(input("Enter the value of N: "))

count = 0  # Initialize a counter to keep track of the number of prime numbers found
num = 2    # Start checking prime numbers from 2

while count < N:
    is_prime = True

    # Check if the current number is divisible by any integer
    # from 2 to the square root of the number
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            is_prime = False
            break

    # If the number is prime, display it
    if is_prime:
        print(num, end=" ")
        count += 1

    num += 1  # Move to the next number for checking

# Add a new line for better output formatting
print()
```

We use the ***input*** function to prompt the user to enter a value for ***N***. The ***int*** function is used to convert the input value to an integer.

We initialize a variable ***count*** to keep track of the number of prime numbers found. We start with 0.

We initialize a variable ***num*** to 2, as the first prime number is 2.

We use a ***while*** loop to continue finding prime numbers until we have found ***N*** prime numbers.

Inside the loop, we initialize a variable ***is_prime*** to ***True*** for each number.

We use a ***for*** loop to check if the current number is divisible by any integer from 2 to the square root of the number (inclusive).

Inside the nested loop, we use the modulo operator ***%*** to check if the number is divisible by the current value of ***i***. If it is divisible, we set ***is_prime*** to ***False*** and break out of the loop.

After the nested loop, if ***is_prime*** is still ***True***, it means the current number is prime. We use the ***print*** function to display the prime number, separated by a space. We use the ***end=" "*** parameter to ensure the numbers are printed on the same line.

We increment the ***count*** by 1 to keep track of the number of prime numbers found.

We increment ***num*** by 1 to move to the next number for checking.

The loop continues until count reaches ***N***, i.e., we have found ***N*** prime numbers.

After the loop, we use the print function without any arguments to add a new line, which improves the output formatting.

48. Create a program that displays the first N first perfect squares, where N is informed by the user, using a loop.

```
# Prompt the user for a number
N = int(input("Enter the value of N: "))

count = 0  # Initialize a counter to keep track of the number of perfect squares found
num = 1    # Start with the first positive integer

while count < N:
    square = num ** 2  # Calculate the square of the current number

    # Display the perfect square
    print(square, end=" ")
    count += 1

    num += 1  # Move to the next number

# Add a new line for better output formatting
print()
```

We use the ***input*** function to prompt the user to enter a value for ***N***. The ***int*** function is used to convert the input value to an integer.

We initialize a variable ***count*** to keep track of the number of perfect squares found. We start with 0.

We initialize a variable ***num*** to 1, as we want to start with the first positive integer.

We use a ***while*** loop to continue finding perfect squares until we have found ***N*** perfect squares.

Inside the loop, we calculate the square of the current number using the ** operator.

We use the ***print*** function to display the perfect square, separated by a space. We use the ***end=" "*** parameter to ensure the numbers are printed on the same line.

We increment the ***count*** by 1 to keep track of the number of perfect squares found.

We increment ***num*** by 1 to move to the next number.

The loop continues until ***count*** reaches ***N***, i.e., we have found ***N*** perfect squares.

After the loop, we use the print function without any arguments to add a new line, which improves the output formatting.

49. Write a program that prompts the user for two numbers A and B and displays all numbers between A and B.

Solution 1

```
# Prompt the user for the two numbers
A = int(input("Enter the value of A: "))
B = int(input("Enter the value of B: "))

# Determine the starting and ending values for the loop
start = min(A, B)
end = max(A, B)

# Use a for loop to iterate over the
# numbers between A and B (inclusive)
for num in range(start, end + 1):
    print(num, end=" ")

# Add a new line for better output formatting
print()
```

We use the ***input*** function to prompt the user to enter two numbers ***A*** and ***B***. The ***int*** function is used to convert the input values to integers.

We determine the starting value (***start***) and ending value (***end***) for the loop by comparing ***A*** and ***B***. We use the ***min*** and ***max*** functions to ensure that start represents the smaller value and end represents the larger value.

We use a for loop and the range function to iterate over the numbers between ***start*** and ***end***, inclusive. The ***range*** function generates a sequence of numbers from ***start*** to ***end***, and the ***for*** loop iterates over each number in that sequence.

Inside the loop, we use the ***print*** function to display each number, separated by a space. We use the ***end=" "*** parameter to ensure the numbers are printed on the same line.

After the loop, we use the ***print*** function without any arguments to add a new line, which improves the output formatting.

Solution 2

```
# Prompt the user for the two numbers
A = int(input("Enter the value of A: "))
B = int(input("Enter the value of B: "))

# Determine the starting and ending values for the loop
if A <= B:
    start = A
    end = B
else:
    start = B
    end = A

# Use a for loop to iterate over the numbers between A and B (inclusive)
for num in range(start, end + 1):
    print(num, end=" ")

# Add a new line for better output formatting
print()
```

In this modified version, we use an ***if*** statement to determine the starting value (***start***) and ending value (***end***) for the loop. We compare ***A*** and ***B*** directly in the condition of the ***if*** statement.

If ***A*** is less than or equal to ***B***, we assign ***A*** to ***start*** and ***B*** to ***end***. Otherwise, if ***B*** is less than ***A***, we assign ***B*** to ***start*** and ***A*** to ***end***. This ensures that start represents the smaller value and end represents the larger value.

The rest of the program remains the same. We use a ***for*** loop and the ***range*** function to iterate over the numbers between ***start*** and ***end***, inclusive, and ***print*** each number on the console.

50. Write a program that reads numbers from the user until a negative number is entered, and prints the sum of the positive numbers.

```
# Initialize variables
sum_positive = 0

# Read numbers from the user
while True:
    number = int(input("Enter a number (negative number to exit): "))

    # Check if the number is positive
    if number >= 0:
        sum_positive += number
    else:
        break

# Print the sum of positive numbers
print("Sum of positive numbers:", sum_positive)
```

In this program, we start by initializing the variable ***sum_positive*** to 0. This variable will hold the sum of all positive numbers entered by the user.

We then enter a while loop that runs indefinitely (***while True***). Inside the loop, we prompt the user to enter a number using ***input*** and convert it to an integer using ***int***.

We check if the number is positive by comparing it to 0 using ***number >= 0***. If the number is positive, we add it to the ***sum_positive*** variable. If the number is negative, we break out of the loop using the ***break*** statement.

Once the loop is exited, we print the sum of the positive numbers using ***print(****"Sum of positive numbers:"****, sum_positive).***

The program will keep asking the user for numbers until a negative number is entered. It will then calculate and display the sum of the positive numbers entered.

Without using ***break*** statement

```
# Initialize variables
sum_positive = 0
number = 0

# Read numbers from the user until a negative number is entered
while number >= 0:
    number = int(input("Enter a number (negative number to exit): "))

    # Check if the number is positive
    if number >= 0:
        sum_positive += number

# Print the sum of positive numbers
print("Sum of positive numbers:", sum_positive)
```

In this solution, we still use a ***while*** loop to continuously read numbers from the user until a negative number is entered. However, instead of using the ***break*** statement, we use the condition while ***number >= 0*** as the loop termination condition.

Inside the loop, we first prompt the user to enter a number and convert it to an integer. We then check if the number is positive. If it is, we add it to the ***sum_positive*** variable. Otherwise, the loop continues until a negative number is entered.

Once the loop is exited, we print the sum of the positive numbers using ***print(****"Sum of positive numbers:"****, sum_positive).***

By using the condition of the ***while*** loop as the exit condition, we avoid the need for the ***break*** statement. The loop will naturally terminate when a negative number is entered.

51. Write a program that prompts the user for a number and displays the Fibonacci sequence up to the given number using a repeating loop.

```
# Prompt the user for a number
number = int(input("Enter a number: "))

# Initialize variables
previous_number = 0
current_number = 1

# Display the Fibonacci sequence up to the given number
print("Fibonacci sequence up to", number, ":")
print(previous_number, end=" ")

while current_number <= number:
    print(current_number, end=" ")

    # Calculate the next Fibonacci number
    next_number = previous_number + current_number

    # Update previous_number and current_number
    previous_number = current_number
    current_number = next_number
```

It prompts the user to enter a number using ***input*** and converts it to an integer using ***int***.

The program initializes two variables: ***previous_number*** with the initial Fibonacci number 0, and ***current_number*** with the initial Fibonacci number 1.

The program uses a ***while*** loop to generate and print the Fibonacci sequence up to the given number.

Inside the loop, the program prints the ***current_number*** (which is the next Fibonacci number) using ***print***.

The program calculates the next Fibonacci number by adding the ***previous_number*** and ***current_number*** and assigns it to the ***next_number*** variable.

Then, the program updates the ***previous_number*** with the value of ***current_number***, and ***current_number*** with the value of ***next_number***.

The loop continues until the ***current_number*** exceeds the given number.

By using only two variables and updating them in each iteration, the program generates and prints the Fibonacci sequence up to the given number without using a list. The print function is used to display each Fibonacci number as it is calculated.

52. Write a program that reads numbers from the user until zero is entered, and displays the average of the numbers entered.

```
# Initialize variables
total = 0
count = 0

# Read numbers from the user until zero is entered
while True:
    number = int(input("Enter a number (enter 0 to stop): "))

    # Check if the number is zero
    if number == 0:
        break

    # Add the number to the total and increment the count
    total += number
    count += 1

# Calculate the average
if count > 0:
    average = total / count
    print("Average:", average)
else:
    print("No numbers were entered.")
```

The program initializes two variables, ***total*** and ***count***, to keep track of the sum of the numbers entered and the number of numbers entered, respectively.

The program enters a ***while*** loop with the condition ***True***, which means the loop will continue indefinitely until a ***break*** statement is encountered.

Inside the loop, the program prompts the user to enter a number using ***input***, converts it to an integer using ***int***, and assigns it to the ***number*** variable.

The program checks if the entered number is zero using an ***if*** statement. If it is zero, the ***break*** statement is executed, which terminates the loop and continues to the next part of the program.

If the entered number is not zero, the program adds the number to the ***total*** variable and increments the ***count*** variable.

After the loop terminates, the program checks if any numbers were entered by comparing the ***count*** variable to zero. If there were numbers entered, it calculates the average by dividing the ***total*** by the ***count*** and assigns it to the ***average*** variable.

Finally, the program prints the calculated average using the ***print*** function.

By repeatedly reading numbers from the user, summing them up, and counting the number of inputs, the program calculates and displays the average of the numbers entered. When the user enters zero, the loop is terminated, and the average is calculated and printed if there were any numbers entered. Otherwise, if no numbers were entered, a corresponding message is displayed.

53. Write a program that prompts the user for a list of numbers, until the user types the number zero, and displays the largest and smallest numbers in the list.

```
# Initialize variables
largest = float('-inf')  # Initialize largest to negative infinity
smallest = float('inf')  # Initialize smallest to positive infinity

# Read numbers from the user until zero is entered
while True:
    number = float(input("Enter a number (enter 0 to stop): "))

    # Check if the number is zero
    if number == 0:
        break

    # Update largest and smallest numbers
    if number > largest:
        largest = number
    if number < smallest:
        smallest = number

# Display the largest and smallest numbers
if largest != float('-inf') and smallest != float('inf'):
    print("Largest number:", largest)
    print("Smallest number:", smallest)
else:
    print("No numbers were entered.")
```

The program initializes two variables, ***largest*** and ***smallest***, to keep track of the largest and smallest numbers entered, respectively. The initial values are set to negative infinity and positive infinity, respectively, to ensure that any number entered by the user will be larger than the initial largest value and smaller than the initial smallest value.

The program enters a ***while*** loop with the condition ***True***, which means the loop will continue indefinitely until a ***break*** statement is encountered.

Inside the loop, the program prompts the user to enter a number using ***input***, converts it to a float using ***float***, and assigns it to the ***number*** variable.

The program checks if the entered number is zero using an ***if*** statement. If it is zero, the ***break*** statement is executed, which terminates the loop and continues to the next part of the program.

If the entered number is not zero, the program compares it with the current largest and smallest numbers using ***if*** statements. If the number is larger than the current largest, it updates the value of ***largest*** to the entered number. If the number is smaller than the current smallest, it updates the value of ***smallest*** to the entered number.

After the loop terminates, the program checks if any numbers were entered by comparing the values of ***largest*** and ***smallest*** with the initial values. If numbers were entered and updated, it prints the values of largest and smallest using the ***print*** function.

Finally, if no numbers were entered and the values of ***largest*** and ***smallest*** were not updated, the program prints a corresponding message.

54. Write a program that prompts the user for a sentence and displays the number of vowels in the sentence.

```
# Prompt the user for a sentence
sentence = input("Enter a sentence: ")

# Initialize the vowel count to 0
vowel_count = 0

# Iterate over each character in the sentence
for char in sentence:
    # Convert the character to lowercase for case-insensitive matching
    char_lower = char.lower()

    # Check if the character is a vowel
    if char_lower in 'aeiou':
        # Increment the vowel count
        vowel_count += 1

# Display the number of vowels in the sentence
print("Number of vowels:", vowel_count)
```

The program prompts the user to enter a sentence using the ***input*** function and assigns it to the ***sentence*** variable.

The program initializes the vowel count to 0 using the variable ***vowel_count***.

The program iterates over each character in the sentence using a ***for*** loop.

Inside the loop, the program converts each character to lowercase using the lower method to ensure case-insensitive matching.

The program checks if the lowercase character is a vowel by comparing it to the string *'aeiou'*, which contains all lowercase vowels.

If the character is a vowel, the program increments the ***vowel_count*** by 1.

After iterating over all characters in the sentence, the program displays the number of vowels in the sentence using the ***print*** function.

55. Write a program that prompts the user for a number and displays its divisors.

```
# Prompt the user for a number
number = int(input("Enter a number: "))

# Display the divisors
print("Divisors of", number, ":")
for i in range(1, number+1):
    if number % i == 0:
        print(i)
```

The program prompts the user to enter a number using the ***input*** function and converts it to an integer using the ***int*** function. The number is then assigned to the ***number*** variable.

The program uses a ***for*** loop to iterate from 1 to the given number (inclusive).

Inside the loop, the program checks if the number is divisible by the current iteration value (i.e., if ***number % i == 0***). If it is, it means that ***i*** is a divisor of the number, so it is directly printed using the ***print*** function.

After iterating over all possible divisors, the program displays the divisors on separate lines.

56. Write a program that determines the lowest common multiple (LCM) between two numbers entered by the user.

```
# Prompt the user for two numbers
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

# Find the maximum of the two numbers
maximum = max(num1, num2)

# Calculate the LCM
while True:
    if maximum % num1 == 0 and maximum % num2 == 0:
        lcm = maximum
        break
    maximum += 1

# Display the LCM
print("(LCM) of", num1, "and", num2, "is:", lcm)
```

The program prompts the user to enter two numbers using the ***input*** function and converts them to integers using the ***int*** function. The numbers are then assigned to the variables ***num1*** and ***num2***.

The program determines the maximum of the two numbers using the ***max*** function and assigns it to the variable ***maximum***. This ensures that we start searching for the LCM from the larger number.

The program uses a ***while*** loop to find the LCM. Inside the loop, it checks if ***maximum*** is divisible by both ***num1*** and ***num2*** without any remainder. If it is, it means that ***maximum*** is the LCM, so it is assigned to the variable ***lcm*** and the loop is terminated using the ***break*** statement.

If ***maximum*** is not the LCM, the program increments ***maximum*** by 1 and continues the loop until the LCM is found.

Once the LCM is determined, the program displays it using the ***print*** function.

57. Write a program that determines the greatest common divisor (GCD) between two numbers entered by the user.

```
# Prompt the user for two numbers
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

# Find the smaller of the two numbers
smaller = min(num1, num2)

# Initialize the GCD variable
gcd = 1

# Calculate the GCD
for i in range(1, smaller + 1):
    if num1 % i == 0 and num2 % i == 0:
        gcd = i

# Display the GCD
print("GCD of", num1, "and", num2, "is:", gcd)
```

The program prompts the user to enter two numbers using the ***input*** function and converts them to integers using the ***int*** function. The numbers are then assigned to the variables ***num1*** and ***num2***.

The program determines the smaller of the two numbers using the ***min*** function and assigns it to the variable ***smaller***. This ensures that we only iterate up to the smaller number when finding the GCD.

The program initializes the GCD variable ***gcd*** to 1.

The program uses a ***for*** loop to find the GCD. It iterates from 1 to ***smaller*** (inclusive) using the ***range*** function. For each iteration, it checks if both ***num1*** and ***num2*** are divisible by ***i*** without any remainder. If they are, it means that i is a common divisor, so it updates the gcd variable to ***i***.

After the loop completes, the gcd variable will hold the GCD of ***num1*** and ***num2***.

The program displays the GCD using the ***print*** function.

58. Write a program that calculates the series below up to the tenth element:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

```
import math

# Prompt the user for the value of x
x = float(input("Enter the value of x: "))

# Initialize the sum and the term
sum = 1
term = 1

# Calculate the series up to the tenth element
for n in range(1, 11):
    term *= x / n
    sum += term

# Display the result
result = math.exp(x)
print("Approximation of e^x:", sum)
print("Actual value of e^x:", result)
```

The program prompts the user to enter the value of *x* using the ***input*** function and converts it to a float using the ***float*** function. The value is then assigned to the variable ***x***.

The program initializes the sum variable ***sum*** to 1 and the term variable ***term*** to 1. The ***sum*** represents the running total of the series, and the ***term*** represents the current term in the series.

The program uses a ***for*** loop to calculate the series up to the tenth element. The loop iterates from 1 to 10 using the ***range*** function. For each iteration, it updates the ***term*** by multiplying it with ***x*** divided by ***n***, where ***n*** is the current iteration. It then adds the ***term*** to the ***sum***.

After the loop completes, the ***sum*** variable will hold the approximation of e^x up to the tenth element.

The program uses the ***math.exp*** function to calculate the actual value of e^x and assigns it to the variable ***result***.

The program displays the approximation of e^x calculated using the series and the actual value of e^x using the ***print*** function.

Explaining better the calculation of the term

In the given code, the line ***term *= x / n*** is used to update the ***term*** variable, which represents each term of the series e^x. Let's break down how this line works:

1 - ***term*** is initially set to 1 before the loop starts.

2 - In each iteration of the loop, the line ***term *= x / n*** is executed. This line performs the following calculation:

x / n calculates the ratio between the value of ***x*** and the current iteration number ***n***. This represents the contribution of ***x*** and the factorial component in the formula of each term.

term *= x / n multiplies the current value of ***term*** by the calculated ratio. This updates ***term*** to include the new term's value.

By continuously multiplying ***term*** by the ratio ***x / n*** in each iteration, the value of ***term*** accumulates the contribution of each term in the series.

At the end of the loop, the variable term will represent the value of the current term in the series. It is then added to the running total ***sum*** using the line ***sum += term***. This process is repeated for each iteration of the loop, gradually summing up the terms of the series.

It's important to note that the term ***x / n*** is calculated using the value of ***x*** and the factorial component of the formula (***1 / n!***). The division by ***n*** accounts for the factorial, which is the product of all positive integers from 1 to ***n***. This ensures that each ***term*** in the series follows the pattern of the formula for e^x.

By continuously updating and summing the terms, the code accurately approximates the value of e^x up to the tenth element of the series.

59. Rewrite the previous exercise code until the difference between the terms is less than 0.001.

```python
import math

# Prompt the user for the value of x
x = float(input("Enter the value of x: "))

term = 1.0
sum = 1.0
n = 1

while abs(term) >= 0.001:
    term *= x / n
    sum += term
    n += 1

# Calculate e^x using the math library's exponential function
math_result = math.exp(x)

# Display the calculated result and the math library's result
print("Result (Custom Calculation):", sum)
print("Result (math.exp):", math_result)
```

In this code, we initialize the variables ***term***, ***sum***, and ***n*** to keep track of the current term, the sum of terms, and the denominator value, respectively.

Inside the ***while*** loop, we update the ***term*** by multiplying it with ***x / n***, where ***x*** is the user-provided value and ***n*** is the current denominator value. This calculation follows the series definition for e^x. We then add the updated term to the sum and increment the value of ***n*** by 1.

The loop continues until the absolute value of the current term (***abs(term)***) is greater than or equal to 0.001, ensuring that the difference between consecutive terms is less than the desired threshold.

Finally, we calculate the e^x value using the ***math.exp*** function from the ***math*** library for comparison, and display both the calculated result using the loop and the result obtained using the ***math.exp*** function.

60. Make a program that calculates the value of sine using the Taylor series according to the equation below until the difference between the terms is less than 0.0001.

Solution 1

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

```python
import math

# Prompt the user for the angle in radians
angle = float(input("Enter the angle in radians: "))

term = angle
sum = angle
n = 3

while abs(term) >= 0.0001:
    term *= - (angle ** 2) / ((n - 1) * n)
    sum += term
    n += 2

# Calculate the sine using the math library's sine function
math_result = math.sin(angle)

# Display the calculated result and the math library's result
print("Result (Custom Calculation):", sum)
print("Result (math.sin):", math_result)
```

In this code, we initialize the variables ***term*** and ***sum*** with the value of the angle provided by the user. The ***term*** represents each term of the Taylor series, and the ***sum*** keeps track of the running sum. We also initialize ***n*** as 3, which represents the starting value for the iteration in the Taylor series.

We use a ***while*** loop that continues as long as the absolute value of ***term*** is greater than or equal to 0.0001. Inside the loop, we update the term by multiplying it with ***-(angle ** 2) / ((n - 1) * n)***. This calculation follows the formula for the terms in the Taylor series for sine. We then add the updated term to the sum and increment n by 2 to move to the next odd value.

The loop continues until the difference between consecutive terms becomes smaller than the specified threshold of 0.0001, indicating convergence up to the tenth term in the Taylor series.

Finally, we calculate the sine of the given angle using the ***math.sin*** function from the math library for comparison, and display both the calculated result using the Taylor series and the result obtained using the ***math.sin*** function.

Explaining the Term Calculation

Let's break down the expression ***term *= - (angle ** 2) / ((n - 1) * n)*** step by step:

1 - ***(angle ** 2)***: This calculates the square of the ***angle*** variable. It represents the power of the angle term in the Taylor series.

2 - ***- (angle ** 2)***: The negative sign is applied to the squared angle term. This alternates the sign of the term in each iteration of the loop, as specified by the Taylor series formula.

3 - ***((n - 1) * n)***: This calculates the denominator of the term in the Taylor series. It is the product of two consecutive odd numbers, ***(n - 1)*** and ***n***. The odd numbers are used to ensure the proper alternation of signs in the series.

4 - ***- (angle ** 2) / ((n - 1) * n)***: This expression divides the negative squared angle term by the denominator calculated in step 3. It represents a single term in the Taylor series.

5 - ***term *= - (angle ** 2) / ((n - 1) * n)***: The ***=** operator is a shorthand notation for multiplying and assigning the result back to the term variable. It updates the ***term*** variable by multiplying it with the value calculated in step 4. This step ensures that each iteration of the loop modifies the term by adding the next term in the Taylor series.

In summary, the expression ***term *= - (angle ** 2) / ((n - 1) * n)*** calculates and updates the term variable by multiplying it with the next term in the Taylor series formula, while taking care of the proper alternation of signs.

Arrays

In Python, arrays are a collection of elements of the same type that are stored in contiguous memory locations. Unlike some other programming languages, Python does not have a built-in data type called "array." Instead, Python provides a powerful data structure called a "list" that can be used to store collections of elements. Lists in Python are flexible and can contain elements of different types.

Here's how you can create a list in Python:

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

In this example, ***my_list*** is a list that contains five elements, each represented by an integer.

Lists in Python have several useful features:

Dynamic Size: Lists in Python can dynamically grow or shrink as elements are added or removed. There is no fixed size limit, and you can change the length of a list as needed.

Mutable: Lists are mutable, which means you can modify individual elements or the entire list after it is created. You can assign new values to specific elements, append or remove elements, or even change the order of elements in a list.

Heterogeneous Elements: Lists can contain elements of different data types. For example, a single list can hold integers, strings, booleans, or even other lists.

Here are some common operations you can perform on lists in Python:

Accessing Elements: You can access individual elements in a list using their indices. Indices start at 0 for the first element. For example, ***my_list[0]*** would give you the first element of the list.

Slicing: You can extract a sublist from a list using slicing. Slicing allows you to specify a range of indices to extract a portion of the list. For example, ***my_list[1:4]*** would give you a sublist containing elements at indices 1, 2, and 3.

Modifying Elements: You can modify elements in a list by assigning new values to specific indices. For example, ***my_list[2] = 10*** would change the third element of the list to 10.

Appending and Removing Elements: You can add elements to the end of a list using the ***append()*** method or insert elements at specific positions using the ***insert()*** method. Conversely, you can remove elements from a list using the ***remove()*** method or delete elements at specific indices using the del statement.

List Length: You can find the length of a list (the number of elements in the list) using the ***len()*** function.

Iterating over a List: You can iterate over the elements of a list using a for loop or other looping constructs.

Python also provides various built-in functions and methods for working with lists, such as sorting, searching, and manipulating list elements.

Although Python does not have a separate "array" data type, the flexible and powerful list structure serves as a practical alternative for most use cases.

Now, let's go to the exercises.

61. Create a program that reads an array of integers and displays the sum of all the elements..

```
# Prompt the user for the number of elements in the array
n = int(input("Enter the number of elements: "))

# Initialize an empty list to store the elements
array = []

# Read the elements from the user
for i in range(n):
    element = int(input("Enter element {}: ".format(i + 1)))
    array.append(element)

# Calculate the sum of all the elements
sum = 0
for element in array:
    sum += element

# Display the sum of all the elements
print("Sum of all the elements:", sum)
```

The program prompts the user to enter the number of elements in the array.

An empty list ***array*** is initialized to store the elements.

The program then reads each element from the user and appends it to the ***array***.

After reading all the elements, the program initializes a variable ***sum*** to 0 to store the sum of the elements.

Using a ***for*** loop, the program iterates over each element in the array and adds it to the sum.

Finally, the program displays the sum of all the elements on the screen.

Using a ***while*** loop

```
# Prompt the user for the number of elements in the array
n = int(input("Enter the number of elements: "))

# Initialize an empty list to store the elements
array = []

# Read the elements from the user
i = 0
while i < n:
    element = int(input("Enter element {}: ".format(i + 1)))
    array.append(element)
    i += 1

# Calculate the sum of all the elements
sum = 0
i = 0
while i < len(array):
    sum += array[i]
    i += 1

# Display the sum of all the elements
print("Sum of all the elements:", sum)
```

The program prompts the user to enter the number of elements in the array.

An empty list ***array*** is initialized to store the elements.

The program uses a ***while*** loop to read each element from the user and appends it to the array. The loop continues until ***i*** reaches the value of ***n***.

After reading all the elements, the program initializes a variable ***sum*** to 0 to store the sum of the elements.

Using another ***while*** loop, the program iterates over each element in the ***array*** and adds it to the ***sum***. The loop continues until ***i*** reaches the length of the ***array***.

Finally, the program displays the sum of all the elements on the screen.

62. Write a program that reads an array of integers and displays the largest element in the array.

```
# Prompt the user for the number of elements in the array
n = int(input("Enter the number of elements: "))

# Initialize an empty list to store the elements
array = []

# Read the elements from the user
for i in range(n):
    element = int(input("Enter element {}: ".format(i + 1)))
    array.append(element)

# Initialize a variable to store the largest element
largest = array[0]

# Iterate over the elements in the array
for i in range(1, n):
    if array[i] > largest:
        largest = array[i]

# Display the largest element
print("Largest element:", largest)
```

The program prompts the user to enter the number of elements in the array.

An empty list ***array*** is initialized to store the elements.

Using a ***for*** loop, the program reads each element from the user and appends it to the array. The loop iterates ***n*** times, where ***n*** is the number of elements specified by the user.

The program initializes a variable largest to store the largest element. We initialize it with the first element of the array (***array[0]***) assuming it is the largest for now.

Using another ***for*** loop, the program iterates over the elements in the array starting from the second element (***range(1, n)***). It compares each element with the current largest value, and if a larger element is found, it updates the ***largest*** variable.

Finally, the program displays the largest element on the screen.

63. Write a program that reads an array of integers and displays the average of the elements.

```
# Prompt the user for the number of elements in the array
n = int(input("Enter the number of elements: "))

# Initialize an empty list to store the elements
array = []

# Read the elements from the user
for i in range(n):
    element = int(input("Enter element {}: ".format(i + 1)))
    array.append(element)

# Calculate the sum of the elements
sum = 0
for element in array:
    sum += element

# Calculate the average
average = sum / n

# Display the average
print("Average:", average)
```

The program prompts the user to enter the number of elements in the array.

An empty list ***array*** is initialized to store the elements.

Using a ***for*** loop, the program reads each element from the user and appends it to the array. The loop iterates ***n*** times, where ***n*** is the number of elements specified by the user.

The program initializes a variable ***sum*** to store the sum of the elements, and sets it to 0.

Using another ***for*** loop, the program iterates over each element in the ***array***. It adds each element to the ***sum*** variable.

After the loop, the program calculates the average by dividing the ***sum*** by the number of elements (***n***).

Finally, the program displays the average on the screen.

64. Create a program that reads two vectors of integers of the same size and displays a new vector with the sum of the corresponding elements of the two vectors.

```python
# Prompt the user for the size of the vectors
n = int(input("Enter the size of the vectors: "))

# Initialize empty lists to store the vectors
vector1 = []
vector2 = []

# Read the elements of the first vector from the user
print("Enter elements for vector 1:")
for i in range(n):
    element = int(input("Enter element {}: ".format(i + 1)))
    vector1.append(element)

# Read the elements of the second vector from the user
print("Enter elements for vector 2:")
for i in range(n):
    element = int(input("Enter element {}: ".format(i + 1)))
    vector2.append(element)

# Initialize an empty list to store the sum of corresponding elements
result_vector = []

# Calculate the sum of corresponding elements and store it in the result vector
for i in range(n):
    sum_element = vector1[i] + vector2[i]
    result_vector.append(sum_element)

# Display the resulting vector
print("Resulting vector (sum of corresponding elements):", result_vector)
```

The program prompts the user to enter the size of the vectors.

Empty lists ***vector1*** and ***vector2*** are initialized to store the elements of the first and second vectors, respectively.

Using a ***for*** loop, the program reads the elements of the first vector from the user and appends them to ***vector1***. The loop iterates ***n*** times, where ***n*** is the size of the vectors specified by the user.

Similarly, another ***for*** loop is used to read the elements of the second vector from the user and append them to ***vector2***.

An empty list ***result_vector*** is initialized to store the sum of the corresponding elements of the two vectors.

Using another ***for*** loop, the program iterates over each index ***i*** from 0 to ***n-1***. It calculates the sum of the elements at index ***i*** of ***vector1*** and ***vector2***, and appends the result to ***result_vector***.

Finally, the program displays the resulting vector, which contains the sum of the corresponding elements of the two input vectors.

65. Write a program that reads an array of integers and checks if they are in ascending order.

```
# Prompt the user for the size of the array
n = int(input("Enter the size of the array: "))

# Initialize an empty list to store the array elements
array = []

# Read the elements of the array from the user
print("Enter elements for the array:")
for i in range(n):
    element = int(input("Enter element {}: ".format(i + 1)))
    array.append(element)

# Assume the array is in ascending order initially
is_ascending = True

# Check if the array is in ascending order
for i in range(1, n):
    if array[i] < array[i - 1]:
        is_ascending = False
        break

# Display the result
if is_ascending:
    print("The array is in ascending order.")
else:
    print("The array is not in ascending order.")
```

The program prompts the user to enter the size of the array.

An empty list ***array*** is initialized to store the elements of the array.

Using a ***for*** loop, the program reads the elements of the array from the user and appends them to the ***array*** list. The loop iterates ***n*** times, where ***n*** is the size of the array specified by the user.

A boolean variable ***is_ascending*** is initialized as ***True***, assuming the array is in ascending order initially.

Another ***for*** loop is used to iterate over the elements of the array, starting from the second element (***i = 1***) up to the last element. For each element, it compares it with the previous element (***array[i-1]***). If any element is found to be less than its previous element, it means the array is not in ascending order, and the ***is_ascending*** variable is set to ***False***.

If the ***is_ascending*** variable remains ***True*** after the loop completes, it means the array is in ascending order. Otherwise, it is not.

Finally, the program displays the result, indicating whether the ***array*** is in ascending order or not.

66. Write a program that reads an array of integers and displays the elements in reverse order.

```
# Prompt the user for the size of the array
n = int(input("Enter the size of the array: "))

# Initialize an empty list to store the array elements
array = []

# Read the elements of the array from the user
print("Enter elements for the array:")
for i in range(n):
    element = int(input("Enter element {}: ".format(i + 1)))
    array.append(element)

# Invert the array using a loop
start = 0
end = n - 1
while start < end:
    # Swap the elements at the start and end positions
    array[start], array[end] = array[end], array[start]
    start += 1
    end -= 1

# Display the elements in reverse order
print("Elements in reverse order:")
for element in array:
    print(element)
```

The program prompts the user to enter the size of the ***array***.

An empty list ***array*** is initialized to store the elements of the array.

Using a ***for*** loop, the program reads the elements of the ***array*** from the user and appends them to the ***array*** list. The loop iterates ***n*** times, where ***n*** is the size of the ***array*** specified by the user.

To reverse the array, a ***while*** loop is used. The loop continues until the ***start*** index is less than the ***end*** index.

Within the loop, the program swaps the elements at the ***start*** and ***end*** positions using simultaneous assignment. This effectively reverses the order of the elements.

After each iteration, the ***start*** index is incremented by 1 and the ***end*** index is decremented by 1 to move towards the center of the array.

Once the array is inverted, a ***for*** loop is used to iterate over each element in the array and display them one by one.

The program prints the elements in reverse order, displaying them on separate lines.

Solution 2 - creating another array using list slicing

```
# Prompt the user for the size of the array
n = int(input("Enter the size of the array: "))

# Initialize an empty list to store the array elements
array = []

# Read the elements of the array from the user
print("Enter elements for the array:")
for i in range(n):
    element = int(input("Enter element {}: ".format(i + 1)))
    array.append(element)

# Reverse the array using list slicing
reversed_array = array[::-1]

# Display the elements in reverse order
print("Elements in reverse order:")
for element in reversed_array:
    print(element)
```

Solution 3 - without modifying the array itself or creating another one

```
# Prompt the user for the size of the array
n = int(input("Enter the size of the array: "))

# Initialize an empty list to store the array elements
array = []

# Read the elements of the array from the user
print("Enter elements for the array:")
for i in range(n):
    element = int(input("Enter element {}: ".format(i + 1)))
    array.append(element)

# Display the elements in reverse order
print("Elements in reverse order:")
for i in range(n - 1, -1, -1):
    print(array[i])
```

The program prompts the user to enter the size of the array. An empty list ***array*** is initialized to store the elements of the ***array***.

Using a ***for*** loop, the program reads the elements of the ***array*** from the user and appends them to the ***array*** list. The loop iterates ***n*** times, where ***n*** is the size of the array specified by the user.

After reading the elements, the program uses another ***for*** loop to iterate over the indices of the ***array*** in reverse order. The loop starts from ***n - 1*** (the last index) and goes down to 0, decrementing by 1 at each iteration.

Within the loop, the program accesses the element at index ***i*** in the array and prints it. Since the loop iterates in reverse order, the elements are displayed in reverse order.

The program prints the elements in reverse order, displaying them on separate lines.

By iterating over the array indices in reverse order, the solution effectively displays the elements of the array in reverse order without modifying the array itself.

67. Create a program that reads an array of integers and finds the second largest element in the array.

```
# Prompt the user for the size of the array
n = int(input("Enter the size of the array: "))

# Initialize an empty list to store the array elements
array = []

# Read the elements of the array from the user
print("Enter elements for the array:")
for i in range(n):
    element = int(input("Enter element {}: ".format(i + 1)))
    array.append(element)

# Initialize variables to store the largest and second largest elements
largest = float('-inf')
second_largest = float('-inf')

# Find the largest and second largest elements
for element in array:
    if element > largest:
        second_largest = largest
        largest = element
    elif element > second_largest and element != largest:
        second_largest = element

# Display the second largest element
print("Second largest element:", second_largest)
```

The program prompts the user to enter the size of the ***array***.

An empty list ***array*** is initialized to store the elements of the ***array***.

Using a ***for*** loop, the program reads the elements of the ***array*** from the user and appends them to the ***array*** list. The loop iterates ***n*** times, where ***n*** is the size of the ***array*** specified by the user.

After reading the elements, the program initializes two variables ***largest*** and ***second_largest*** to store the largest and second largest elements. Both variables are initialized with a value of negative infinity (***float('-inf')***) to ensure that any element in the array will be larger than them initially.

The program then iterates over each element in the ***array***. For each element, it compares it with the current largest element. If the element is larger than ***largest***, it updates both ***largest*** and ***second_largest*** accordingly.

If the element is not larger than ***largest*** but larger than ***second_largest*** and not equal to ***largest***, it updates only ***second_largest***.

After iterating over all the elements, the program displays the value of ***second_largest***, which represents the second largest element in the array.

68. Write a program that reads an array of integers and displays how many times a specific number appears in the array.

```
# Prompt the user for the size of the array
n = int(input("Enter the size of the array: "))

# Initialize an empty list to store the array elements
array = []

# Read the elements of the array from the user
print("Enter elements for the array:")
for i in range(n):
    element = int(input("Enter element {}: ".format(i + 1)))
    array.append(element)

# Prompt the user for the number to search for
target = int(input("Enter the number to search for: "))

# Count the number of occurrences of the target number in the array
count = 0
for element in array:
    if element == target:
        count += 1

# Display the result
print("The number", target, "appears", count, "time(s) in the array.")
```

The program prompts the user to enter the size of the array.

An empty list ***array*** is initialized to store the elements of the array.

Using a ***for*** loop, the program reads the elements of the array from the user and appends them to the ***array*** list. The loop iterates ***n*** times, where ***n*** is the size of the array specified by the user.

After reading the elements, the program prompts the user to enter the number to search for.

The program initializes a variable ***count*** to keep track of the number of occurrences of the ***target*** number in the array.

The program iterates over each element in the array. For each element, it checks if it is equal to the ***target*** number. If it is, the ***count*** variable is incremented by 1.

After iterating over all the elements, the program displays the result by printing the ***target*** number and the ***count*** of its occurrences in the array.

69. Write a program that reads two arrays of integers with the same size and displays a new array with the elements resulting from the multiplication of the corresponding elements of the two arrays.

```
# Prompt the user for the size of the arrays
size = int(input("Enter the size of the arrays: "))

# Initialize empty lists to store the elements of the arrays
array1 = []
array2 = []

# Read the elements of the first array from the user
print("Enter elements for array 1:")
for i in range(size):
    element = int(input("Enter element {}: ".format(i + 1)))
    array1.append(element)

# Read the elements of the second array from the user
print("Enter elements for array 2:")
for i in range(size):
    element = int(input("Enter element {}: ".format(i + 1)))
    array2.append(element)

# Initialize an empty list to store the resulting array
result_array = []

# Calculate the multiplication of the corresponding elements
for i in range(size):
    result = array1[i] * array2[i]
    result_array.append(result)

# Display the resulting array
print("Resulting array:", result_array)
```

The program prompts the user to enter the size of the arrays.

Empty lists ***array1*** and ***array2*** are initialized to store the elements of the two arrays.

Using a ***for*** loop, the program reads the elements of the first array from the user and appends them to the ***array1*** list. The loop iterates size times, where ***size*** is the size of the arrays specified by the user.

Similarly, the program reads the elements of the second array from the user and appends them to the ***array2*** list.

An empty list ***result_array*** is initialized to store the resulting array.

The program calculates the multiplication of the corresponding elements of ***array1*** and ***array2*** using another for loop. For each index ***i***, it multiplies ***array1[i]*** with ***array2[i]*** and appends the result to the ***result_array*** list.

Finally, the program displays the resulting array by printing the ***result_array*** list.

70. Create a program that reads an array of integers and checks that all elements are even.

```
# Prompt the user for the size of the array
size = int(input("Enter the size of the array: "))

# Initialize an empty list to store the elements of the array
array = []

# Read the elements of the array from the user
print("Enter elements for the array:")
for i in range(size):
    element = int(input("Enter element {}: ".format(i + 1)))
    array.append(element)

# Flag to track if all elements are even
all_even = True

# Check if all elements are even
for element in array:
    if element % 2 != 0:  # If the element is not even
        all_even = False
        break

# Display the result
if all_even:
    print("All elements are even.")
else:
    print("Not all elements are even.")
```

The program prompts the user to enter the size of the array.

An empty list ***array*** is initialized to store the elements of the array.

Using a ***for*** loop, the program reads the elements of the array from the user and appends them to the ***array*** list. The loop iterates ***size*** times, where ***size*** is the size of the array specified by the user.

A flag ***all_even*** is initialized as ***True*** to track whether all elements are even.

The program then checks each element in the array using a ***for*** loop. If any element is not divisible by 2 (i.e., not even), the ***all_even*** flag is set to ***False***, and the loop is terminated using the ***break*** statement.

Finally, the program displays the result by checking the value of the ***all_even*** flag. If it is ***True***, it prints that all elements are even; otherwise, it prints that not all elements are even.

Strings

In Python, a string is a sequence of characters enclosed in quotation marks. It can be represented using single quotes (') or double quotes ("). Strings in Python are immutable, which means they cannot be changed once they are created. However, you can perform various operations on strings to manipulate and extract information from them.

Here's an example of creating a string in Python:

```
my_string = "Hello, World!"
```

In this example, my_string is a string that contains the text "*Hello, World!*".

Strings in Python have several useful features and operations:

Accessing Characters: You can access individual characters in a string using indices. Indices start at 0 for the first character. For example, ***my_string[0]*** would give you the first character of the string.

Slicing: You can extract a substring from a string using slicing. Slicing allows you to specify a range of indices to extract a portion of the string. For example, ***my_string[7:12]*** would give you the substring "*World*".

Concatenation: You can concatenate (combine) strings using the + operator. This allows you to join multiple strings together. For example:

```
greeting = "Hello"
name = "Alice"
message = greeting + ", " + name
print(message)  # Output: Hello, Alice
```

String Length: You can find the length of a string (the number of characters) using the ***len()*** function. For example:

```
my_string = "Hello, World!"
length = len(my_string)
print(length)  # Output: 13
```

String Methods: Python provides numerous built-in methods for working with strings. These methods allow you to perform operations like converting the case of a string (***lower()***, ***upper()***), finding substrings (***find()***,

index()), replacing characters or substrings (***replace()***), splitting strings into lists (***split()***), and more.

String Formatting: Python provides several ways to format strings. One popular method is to use the ***.format()*** method or f-strings (formatted string literals) to insert variables or values into a string. For example:

```
name = "Alice"
age = 25
message = "My name is {} and I am {} years old.".format(name, age)
print(message)  # Output: My name is Alice and I am 25 years old.
```

Escape Characters: You can include special characters in a string using escape sequences. For example, to include a newline character, you can use ***\n***. Other common escape sequences include ***\t*** for tab, \" for double quotes, and \' for single quotes.

These are just some of the basic operations and features of strings in Python. Strings are widely used for representing and manipulating textual data in Python programs. They are versatile and can be combined with other data types and structures to solve a wide range of programming problems.

Now, let's go to the exercises.

Dictionaries

In Python, a dictionary is a built-in data structure that allows you to store and retrieve data in key-value pairs. It is also known as an associative array or a hash map in other programming languages. Dictionaries are unordered, mutable, and can contain elements of different data types. They are enclosed in curly braces ***{}*** and consist of comma-separated key-value pairs.

Here's an example of creating a dictionary in Python:

```
my_dict = {"name": "John", "age": 30, "city": "New York"}
```

In this example, ***my_dict*** is a dictionary that contains three key-value pairs. The keys are "*name*", "*age*", and "*city*", and the corresponding values are "*John*", *30*, and "*New York*".

Dictionaries in Python have several useful features and operations:

Accessing Values: You can access the value associated with a specific key in a dictionary using square brackets ***[]*** and providing the key. For

example, ***my_dict["name"]*** would give you the value "*John*".

Modifying Values: Dictionaries are mutable, so you can change the value associated with a key by assigning a new value to it. For example, ***my_dict["age"] = 35*** would change the value of the "*age*" key to *35*.

Adding and Removing Elements: You can add new key-value pairs to a dictionary by assigning a value to a new key. For example, ***my_dict["gender"] = "Male"*** would add a new key-value pair to the dictionary. Conversely, you can remove key-value pairs using the del statement or the ***pop()*** method.

Dictionary Methods: Python provides various built-in methods for working with dictionaries. Some commonly used methods include ***keys()*** to get a list of all keys, ***values()*** to get a list of all values, ***items()*** to get a list of all key-value pairs, ***get()*** to retrieve a value with a default fallback if the key is not found, and more.

Iterating over a Dictionary: You can iterate over the keys, values, or key-value pairs of a dictionary using a for loop or other looping constructs.

Dictionary Length: You can find the number of key-value pairs in a dictionary using the ***len()*** function.

Dictionaries provide a flexible and efficient way to store and retrieve data based on meaningful keys. They are commonly used to represent structured data, configuration settings, and mappings between related values. Dictionaries are particularly useful when you need to access values quickly based on their associated keys, as the lookup time is typically constant regardless of the dictionary's size.

Split

In Python, the ***split()*** method is used to split a string into a list of substrings based on a specified delimiter. It returns a list of substrings generated by breaking the original string at each occurrence of the delimiter. The syntax for using the ***split()*** method is as follows:

string.split(delimiter, maxsplit)

string is the original string that you want to split.

delimiter is the character or substring at which the original string will be split. It is an optional parameter, and if not specified, the split() method will

split the string at whitespace characters (spaces, tabs, and newlines) by default.

maxsplit is an optional parameter that specifies the maximum number of splits to be performed. If provided, the string will be split at most maxsplit - 1 times.

Here are a few examples to illustrate how the ***split()*** method works:

Splitting a string based on whitespace:

```
sentence = "Hello, world! How are you today?"
words = sentence.split()
print(words) #['Hello,', 'world!', 'How', 'are', 'you', 'today?']
```

In this example, the ***split()*** method is called without specifying a delimiter, so it splits the ***sentence*** string at whitespace characters. The resulting list contains each word as a separate element.

Splitting a string based on a specific delimiter:

```
data = "apple,banana,cherry,orange"
fruits = data.split(",")
print(fruits) #['apple', 'banana', 'cherry', 'orange']
```

In this example, the ***split()*** method is called with a comma (",") as the delimiter. The original string ***data*** is split at each occurrence of the comma, resulting in a list of individual fruits.

Limiting the number of splits:

```
sentence = "I like to code in Python, it's fun and Pythonic!"
words = sentence.split("Python", 1)
print(words) #['I like to code in ', ", it's fun and Pythonic!"]
```

In this example, the ***split()*** method is called with the delimiter as "*Python*" and the ***maxsplit*** parameter set to 1. It splits the ***sentence*** string at the first occurrence of "*Python*" and returns a list with two elements.

The ***split()*** method is useful when you need to split a string into substrings based on a specific delimiter. It allows you to extract meaningful components from a string and process them individually. The resulting substrings are stored as elements in a list, providing easy access and further manipulation.

Now, let's go to the exercises.

71. Create a program that reads two words and concatenates them, displaying the resulting word.

```
word1 = input("Enter the first word: ")
word2 = input("Enter the second word: ")

result = word1 + word2

print("Concatenated word:", result)
```

The program starts by prompting the user to enter the first word using the ***input()*** function. The text "*Enter the first word:* " is displayed as a prompt, and the user can type their response. The entered word is stored in the variable ***word1***.

Next, the program prompts the user to enter the second word using the ***input()*** function again. The text "*Enter the second word:* " is displayed as a prompt, and the user can provide their input. The entered word is stored in the variable ***word2***.

The program then concatenates the two words together using the + operator. In Python, when the + operator is used with strings, it performs concatenation, i.e., it joins the two strings together.

The concatenated word is stored in the variable ***result***.

Finally, the program uses the ***print()*** function to display the concatenated word. It prints the text "*Concatenated word:*" followed by the value of the result variable.

72. Write a program that takes a word and displays each letter separately.

```
word = input("Enter a word: ")

for letter in word:
    print(letter)
```

The program starts by prompting the user to enter a word using the ***input()*** function. The text "*Enter a word: *" is displayed as a prompt, and the user can provide their input. The entered word is stored in the variable ***word***.

The program uses a ***for*** loop to iterate over each character (***letter***) in the word. The loop variable ***letter*** takes on the value of each ***character*** in the ***word***, one at a time.

Inside the loop, the program uses the ***print()*** function to display each letter on a separate line. The ***print(letter)*** statement prints the value of the letter variable, which represents the current letter in the iteration.

The loop continues to iterate over each letter in the word until all letters have been processed.

Using a ***while*** loop

```
word = input("Enter a word: ")

index = 0
while index < len(word):
    letter = word[index]
    print(letter)
    index += 1
```

The program prompts the user to enter a word using the ***input()*** function, and the entered word is stored in the variable ***word***.

We initialize the variable ***index*** to 0, which will be used to keep track of the current position in the word.

The ***while*** loop is used to iterate over the characters of the ***word***. The loop continues as long as the index is less than the length of the ***word***, which is obtained using the ***len()*** function.

Inside the loop, the program retrieves the character at the current index position using ***word[index]*** and assigns it to the variable ***letter***.

The program then prints the current ***letter*** using the ***print(letter)*** statement.

After printing the letter, the ***index*** is incremented by 1 using ***index += 1***, to move to the next position in the word.

The loop continues until all characters in the word have been processed

73. Create a program that takes a sentence and replaces all the letters "a" with "e".

Using *Replace*

```
sentence = input("Enter a sentence: ")

new_sentence = sentence.replace('a', 'e')

print("Modified sentence:", new_sentence)
```

The program prompts the user to enter a sentence using the ***input()*** function, and the entered sentence is stored in the variable ***sentence***.

The program uses the ***replace()*** method of strings to replace all occurrences of the letter "*a*" in the sentence with the letter "*e*". The method takes two arguments: the character to be replaced ('***a***') and the replacement character ('***e***'). The resulting modified sentence is stored in the variable ***new_sentence***.

Finally, the program uses the ***print()*** function to display the modified sentence. It prints the text "*Modified sentence:*" followed by the value of the ***new_sentence*** variable.

Using ***for*** loop

```
sentence = input("Enter a sentence: ")

new_sentence = ""
for letter in sentence:
    if letter == 'a':
        new_sentence += 'e'
    else:
        new_sentence += letter

print("Modified sentence:", new_sentence)
```

The program prompts the user to enter a sentence using the ***input()*** function, and the entered sentence is stored in the variable ***sentence***.

We initialize an empty string ***new_sentence***, which will be used to store the modified sentence.

The program uses a for loop to iterate over each character (***letter***) in the sentence.

Inside the loop, an if statement checks if the current character (***letter***) is equal to the letter '***a***'.

If the condition is true, meaning we have found an '***a***', the program appends the letter '***e***' to the ***new_sentence*** using the += operator.

If the condition is false, indicating any character other than '***a***', the program appends the current letter as it is to the ***new_sentence***.

The loop continues to iterate over each character in the sentence until all characters have been processed.

Finally, the program uses the ***print()*** function to display the modified sentence. It prints the text "*Modified sentence:*" followed by the value of the ***new_sentence*** variable.

74. Write a program that receives a name and checks that it starts with the letter "A".

```
name = input("Enter a name: ")

if name.startswith('A'):
    print("The name starts with 'A'.")
else:
    print("The name does not start with 'A'.")
```

The program prompts the user to enter a name using the ***input()*** function, and the entered name is stored in the variable ***name***.

The program uses the ***startswith()*** method of strings to check if the name starts with the letter '*A*'. The method takes a single argument, which is the prefix to be checked ('*A*' in this case).

Inside the ***if*** statement, if the condition is true (the name starts with '*A*'), it executes the code block that prints the message "*The name starts with 'A'*".

If the condition is false (the name does not start with '*A*'), it executes the code block inside the ***else*** statement that prints the message "*The name does not start with 'A'*".

75. Write a program that reads a word and checks if it is a palindrome (if it can be read backwards the same way).

```
word = input("Enter a word: ")

reversed_word = word[::-1]  # Reverse the word using slicing

if word == reversed_word:
    print("The word is a palindrome.")
else:
    print("The word is not a palindrome.")
```

The program prompts the user to enter a word using the ***input()*** function, and the entered word is stored in the variable ***word***.

The program uses slicing ***word[::-1]*** to reverse the order of characters in the word. The ***[::-1]*** slice notation starts from the last character and iterates backward with a step of -1, effectively reversing the word.

The program then checks if the original word is equal to the reversed word using the == operator.

If the condition is true (the word is equal to its reverse), it executes the code block inside the ***if*** statement that prints the message "*The word is a palindrome.*"

If the condition is false (the word is not equal to its reverse), it executes the code block inside the ***else*** statement that prints the message "*The word is not a palindrome.*"

Using ***for*** loop

```
word = input("Enter a word: ")

is_palindrome = True

length = len(word)
for i in range(length // 2):
    if word[i] != word[length - 1 - i]:
        is_palindrome = False
        break

if is_palindrome:
    print("The word is a palindrome.")
else:
    print("The word is not a palindrome.")
```

The program prompts the user to enter a word using the ***input()*** function, and the entered word is stored in the variable ***word***.

We initialize a boolean variable ***is_palindrome*** to ***True***. This variable will be used to keep track of whether the word is a palindrome or not.

We get the length of the word using the ***len()*** function and store it in the variable ***length***.

The program uses a ***for*** loop to iterate over the first half of the word (up to ***length // 2***). This avoids unnecessary comparisons in the second half of the word, as they would be redundant for checking palindromes.

Inside the loop, the program checks if the character at index ***i*** is equal to the character at the corresponding index from the end of the word (***length - 1 - i***). If they are not equal, the word is not a palindrome, and we set ***is_palindrome*** to ***False*** and break out of the loop.

After the loop, the program checks the value of ***is_palindrome***. If it is still ***True***, it means that the loop completed without finding any non-matching characters, and thus the word is a palindrome.

If ***is_palindrome*** is ***True***, the program prints the message "*The word is a palindrome.*" If it is ***False***, the program prints the message "*The word is not a palindrome.*"

76. Create a program that reads two words and checks if the second word is an anagram of the first.

```
word1 = input("Enter the first word: ")
word2 = input("Enter the second word: ")

# Convert both words to lowercase and remove whitespace
word1 = word1.lower().replace(" ", "")
word2 = word2.lower().replace(" ", "")

# Check if the lengths of the words are equal
if len(word1) != len(word2):
    print("The second word is not an anagram of the first.")
else:
    # Sort the characters of both words
    sorted_word1 = sorted(word1)
    sorted_word2 = sorted(word2)

    # Check if the sorted words are equal
    if sorted_word1 == sorted_word2:
        print("The second word is an anagram of the first.")
    else:
        print("The second word is not an anagram of the first.")
```

The program prompts the user to enter the first word using the ***input()*** function, and the entered word is stored in the variable ***word1***.

Similarly, the program prompts the user to enter the second word, which is stored in the variable ***word2***.

To ensure case-insensitive comparison and remove any whitespace, both ***word1*** and ***word2*** are converted to lowercase using the ***lower()*** method and whitespace is removed using the ***replace()*** method.

The program checks if the lengths of ***word1*** and ***word2*** are equal. If they are not, it means that the second word cannot be an anagram of the first, and

the program prints the corresponding message.

If the lengths of the words are equal, the program proceeds to sort the characters of both words using the ***sorted()*** function. This will rearrange the characters in ascending order.

The sorted words, ***sorted_word1*** and ***sorted_word2***, are then compared using the == operator to check if they are equal. If they are, it means that the second word is an anagram of the first, and the program prints the corresponding message.

If the sorted words are not equal, it means that the second word is not an anagram of the first, and the program prints the corresponding message.

Solution 2 (more complex)

```
word1 = input("Enter the first word: ")
word2 = input("Enter the second word: ")

# Convert both words to lowercase and remove whitespace
word1 = word1.lower().replace(" ", "")
word2 = word2.lower().replace(" ", "")

# Check if the lengths of the words are equal
if len(word1) != len(word2):
    print("The second word is not an anagram of the first.")
else:
    # Create dictionaries to count the occurrences of each character in the words
    char_count1 = {}
    char_count2 = {}

    # Count the occurrences of each character in word1
    for char in word1:
        char_count1[char] = char_count1.get(char, 0) + 1

    # Count the occurrences of each character in word2
    for char in word2:
        char_count2[char] = char_count2.get(char, 0) + 1

    # Check if the character counts are equal
    if char_count1 == char_count2:
        print("The second word is an anagram of the first.")
    else:
        print("The second word is not an anagram of the first.")
```

The program prompts the user to enter the first word using the ***input()*** function, and the entered word is stored in the variable ***word1***.

Similarly, the program prompts the user to enter the second word, which is stored in the variable ***word2***.

To ensure case-insensitive comparison and remove any whitespace, both ***word1*** and ***word2*** are converted to lowercase using the ***lower()*** method and whitespace is removed using the ***replace()*** method.

The program checks if the lengths of ***word1*** and ***word2*** are equal. If they are not, it means that the second word cannot be an anagram of the first, and the program prints the corresponding message.

If the lengths of the words are equal, the program proceeds to create dictionaries ***char_count1*** and ***char_count2*** to count the occurrences of each character in ***word1*** and ***word2***, respectively.

The program uses a loop to iterate through each character in ***word1*** and ***word2***. For each character, it updates the corresponding count in the dictionaries using the ***get()*** method to retrieve the current count and increment it by 1.

After counting the occurrences of each character in both words, the program compares the dictionaries ***char_count1*** and ***char_count2*** using the == operator to check if they are equal. If they are, it means that the second word is an anagram of the first, and the program prints the corresponding message.

If the character counts are not equal, it means that the second word is not an anagram of the first, and the program prints the corresponding message.

77. Write a program that takes a full name and displays only the first name.

```
full_name = input("Enter your full name: ")

# Split the full name into a list of names using whitespace
  # as the delimiter
names = full_name.split()

# Get the first name from the list
first_name = names[0]

print("First name:", first_name)
```

The program prompts the user to enter their full name using the ***input()*** function, and the entered full name is stored in the variable ***full_name***.

The ***split()*** method is used on the ***full_name*** string to split it into a list of names based on whitespace as the delimiter. This assumes that the first name is the first element in the list.

The first name is obtained by accessing the first element of the names list using indexing (***names[0]***), and it is stored in the variable ***first_name***.

Finally, the program prints the first name using the ***print()*** function, along with the message "*First name:*", to indicate the output.

78. Make a program that receives a sentence and displays the amount of blank spaces present in it.

Using ***for*** loop

```
sentence = input("Enter a sentence: ")

# Initialize a variable to count the number of blank spaces
count = 0

# Iterate over each character in the sentence
for char in sentence:
    # Check if the character is a blank space
    if char == " ":
        count += 1

print("Number of blank spaces:", count)
```

The program prompts the user to enter a sentence using the ***input()*** function, and the entered sentence is stored in the variable ***sentence***.

The variable ***count*** is initialized to 0. This variable will keep track of the number of blank spaces in the sentence.

The program uses a ***for*** loop to iterate over each character in the sentence. For each character, it performs the following steps:

- It checks if the character is a blank space by comparing it to the string " ".

- If the character is a blank space, the ***count*** variable is incremented by 1.

After iterating over all the characters in the sentence, the program prints the number of blank spaces using the ***print()*** function, along with the message "*Number of blank spaces:*", to indicate the output.

Using ***count*** function

```
sentence = input("Enter a sentence: ")

# Use the count() method to count the number of blank spaces
count = sentence.count(" ")

print("Number of blank spaces:", count)
```

The program prompts the user to enter a sentence using the ***input()*** function, and the entered sentence is stored in the variable ***sentence***.

The ***count()*** method is used on the sentence string to count the occurrences of blank spaces. We pass *" "* as the argument to count the number of blank spaces.

The result of the ***count()*** method is stored in the variable count, representing the number of blank spaces in the sentence.

Finally, the program prints the number of blank spaces using the ***print()*** function, along with the message "*Number of blank spaces:*", to indicate the output.

79. Create a program that reads a word and displays the number of vowels present in it.

```
word = input("Enter a word: ")

# Define a list of vowels
vowels = ['a', 'e', 'i', 'o', 'u']

# Initialize a variable to count the number of vowels
count = 0

# Iterate over each character in the word
for char in word:
    # Check if the character is a vowel
    if char.lower() in vowels:
        count += 1

print("Number of vowels:", count)
```

The program prompts the user to enter a word using the ***input()*** function, and the entered word is stored in the variable ***word***.

The program defines a list of vowels containing the lowercase vowels ***['a', 'e', 'i', 'o', 'u']***.

The variable ***count*** is initialized to 0. This variable will keep track of the number of vowels in the word.

The program uses a ***for*** loop to iterate over each character in the word. For each character, it performs the following steps:

- It checks if the lowercase version of the character (***char.lower()***) is present in the list of vowels using the in operator.

- If the character is a vowel, the ***count*** variable is incremented by 1.

After iterating over all the characters in the word, the program prints the number of vowels using the ***print()*** function, along with the message "*Number of vowels:*", to indicate the output.

80. Write a program that takes a full name and displays the last name (last name) first.

```
full_name = input("Enter your full name: ")

# Split the full name into a list of names using whitespace
# as the delimiter
names = full_name.split()

# Get the last name from the list
last_name = names[-1]

print("Last name first:", last_name)
```

The program prompts the user to enter their full name using the ***input()*** function, and the entered full name is stored in the variable ***full_name***.

The ***split()*** method is used on the ***full_name*** string to split it into a list of names based on whitespace as the delimiter. This assumes that the last name is the last element in the list.

The last name is obtained by accessing the last element of the names list using negative indexing (***names[-1]***), and it is stored in the variable ***last_name***.

Finally, the program prints the last name first using the ***print()*** function, along with the message "*Last name first:*", to indicate the output.

Matrices

In Python, matrices can be represented using nested lists or using specialized libraries such as ***NumPy***.

Using nested lists, you can create a matrix as a list of lists, where each inner list represents a row of the matrix. Here's an example of creating a 3x3 matrix using nested lists:

```
matrix = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]
```

In this example, ***matrix*** represents a 3x3 matrix where the first row is ***[1, 2, 3]***, the second row is ***[4, 5, 6]***, and the third row is ***[7, 8, 9]***.

You can access individual elements in the matrix using indices. For example, ***matrix[0][0]*** would give you the element in the first row and first column, which is 1.

Tuples

In Python, a tuple is an ordered, immutable collection of elements enclosed in parentheses ***()*** or without any enclosing brackets. Tuples are similar to lists, but they cannot be modified once created. Each element in a tuple is separated by a comma.

Here's an example of creating a tuple in Python:

```
my_tuple = (1, 2, 3)
```

In this example, ***my_tuple*** is a tuple that contains three elements: 1, 2, and 3.

Tuples in Python have several useful features and operations:

Accessing Elements: You can access individual elements in a tuple using indices. Indices start at 0 for the first element. For example, ***my_tuple[0]*** would give you the first element of the tuple.

Immutable Nature: Tuples are immutable, meaning you cannot modify, add, or remove elements from a tuple after it is created. This property ensures that the data stored in the tuple remains unchanged.

Tuple Packing and Unpacking: You can assign multiple values to a tuple in a single line, which is called tuple packing. Similarly, you can assign the

elements of a tuple to multiple variables, which is known as tuple unpacking. For example:

```
my_tuple = 1, 2, 3  # Tuple packing
a, b, c = my_tuple  # Tuple unpacking
```

Tuple Methods: Python provides a few built-in methods for tuples, such as ***count()*** to count the number of occurrences of a specific element and ***index()*** to find the index of the first occurrence of a specific element.

Tuple Operations: Tuples support operations like concatenation (**+** operator) and repetition (***** operator). For example:

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
concatenated_tuple = tuple1 + tuple2
repeated_tuple = tuple1 * 3
```

Iterating over a Tuple: You can iterate over the elements of a tuple using a ***for*** loop or other looping constructs.

Tuples are commonly used when you want to group related values together, especially when those values should not be modified. They are useful for representing fixed collections of items, function return values, and dictionary keys, among other use cases.

Although tuples are immutable, they can contain mutable objects, such as lists, allowing you to have a mix of mutable and immutable elements within a tuple.

Now, let's go to the exercises.

81. Write a program that fills a 3x3 matrix with values entered by the user and displays the sum of the main diagonal values.

```
# Initialize a 3x3 matrix
matrix = [[0, 0, 0],
          [0, 0, 0],
          [0, 0, 0]]

# Prompt the user to enter the values for the matrix
print("Enter values for the 3x3 matrix:")

for i in range(3):
    for j in range(3):
        value = int(input("({}, {}): ".format(i, j)))
        matrix[i][j] = value

# Calculate the sum of the main diagonal values
sum_diagonal = 0

for i in range(3):
    sum_diagonal += matrix[i][i]

# Display the matrix
print("Matrix:")
for row in matrix:
    print(row)

# Display the sum of the main diagonal values
print("Sum of main diagonal values:", sum_diagonal)
```

The program initializes a 3x3 matrix ***matrix*** with all elements initially set to 0.

It prompts the user to enter the values for the matrix.

Using nested ***for*** loops, the program iterates over each position of the matrix and prompts the user to enter the corresponding value. The values entered by the user are then assigned to the respective positions in the matrix.

After filling the matrix, the program calculates the sum of the main diagonal values. It initializes the variable ***sum_diagonal*** to 0 and uses a single for loop to iterate over the main diagonal elements (positions where the row index equals the column index), adding each element to the ***sum_diagonal*** variable.

The program displays the matrix by iterating over each row and printing it.

Finally, it displays the sum of the main diagonal values by printing the value stored in the ***sum_diagonal*** variable.

82. Write a program that fills a 4x4 matrix with random values and displays the transposed matrix.

```
import random

# Initialize a 4x4 matrix
matrix = [[0, 0, 0, 0],
          [0, 0, 0, 0],
          [0, 0, 0, 0],
          [0, 0, 0, 0]]

# Fill the matrix with random values
for i in range(4):
    for j in range(4):
        matrix[i][j] = random.randint(1, 100)

# Display the original matrix
print("Original Matrix:")
for row in matrix:
    print(row)
```

```
# Display the original matrix
print("Original Matrix:")
for row in matrix:
    print(row)

# Calculate the transpose of the matrix
transposed_matrix = [[0, 0, 0, 0],
                     [0, 0, 0, 0],
                     [0, 0, 0, 0],
                     [0, 0, 0, 0]]

for i in range(4):
    for j in range(4):
        transposed_matrix[i][j] = matrix[j][i]

# Display the transposed matrix
print("\nTransposed Matrix:")
for row in transposed_matrix:
    print(row)
```

The program imports the ***random*** module to generate random values for the matrix.

It initializes a 4x4 matrix ***matrix*** with all elements initially set to 0.

Using nested ***for*** loops, the program iterates over each position of the matrix and assigns a random value (generated using ***random.randint(1, 100)***) to that position.

After filling the matrix, the program displays the original matrix by iterating over each row and printing it.

The program then initializes a new 4x4 matrix ***transposed_matrix*** with all elements initially set to 0 to store the transposed matrix.

Using nested ***for*** loops, the program iterates over each position of the original matrix and assigns the corresponding element to the transposed matrix by swapping the row and column indices.

Finally, the program displays the transposed matrix by iterating over each row and printing it.

83. Write a program that reads two 2x2 matrices and displays the sum of the two matrices.

```
# Initialize two 2x2 matrices
matrix1 = [[0, 0], [0, 0]]
matrix2 = [[0, 0], [0, 0]]
sum_matrix = [[0, 0], [0, 0]]

# Read the values for the first matrix
print("Enter the elements of the first matrix:")
for i in range(2):
    for j in range(2):
        matrix1[i][j] = int(input(f"Enter element at position ({i + 1}, {j + 1}): "))

# Read the values for the second matrix
print("\nEnter the elements of the second matrix:")
for i in range(2):
    for j in range(2):
        matrix2[i][j] = int(input(f"Enter element at position ({i + 1}, {j + 1}): "))

# Calculate the sum of the two matrices
for i in range(2):
    for j in range(2):
        sum_matrix[i][j] = matrix1[i][j] + matrix2[i][j]

# Display the sum of the matrices
print("\nSum of the two matrices:")
for row in sum_matrix:
    print(row)
```

The program initializes three 2x2 matrices: ***matrix1***, ***matrix2***, and ***sum_matrix*** to store the input matrices and their sum, respectively.

The program prompts the user to enter the elements of the first matrix using nested ***for*** loops. The user is asked to enter each element individually and the values are stored in ***matrix1***.

Similarly, the program prompts the user to enter the elements of the second matrix using nested ***for*** loops. The user is asked to enter each element individually and the values are stored in ***matrix2***.

Using nested ***for*** loops, the program calculates the sum of the two matrices by adding the corresponding elements from ***matrix1*** and ***matrix2***, and stores the results in ***sum_matrix***.

Finally, the program displays the sum of the matrices by iterating over each row of ***sum_matrix*** and printing it.

84. Write a program that fills a 5x5 matrix with integers and displays the largest value in the matrix and its position.

```
# Initialize a 5x5 matrix
matrix = [[0] * 5 for _ in range(5)]

# Read the values for the matrix
print("Enter the elements of the matrix:")
for i in range(5):
    for j in range(5):
        matrix[i][j] = int(input(f"Enter element at position ({i + 1}, {j + 1}): "))

# Find the largest value and its position in the matrix
max_value = matrix[0][0]
max_position = (0, 0)
```

```
# Find the largest value and its position in the matrix
max_value = matrix[0][0]
max_position = (0, 0)

for i in range(5):
    for j in range(5):
        if matrix[i][j] > max_value:
            max_value = matrix[i][j]
            max_position = (i, j)

# Display the largest value and its position
print(f"\nLargest value: {max_value}")
print(f"Position: ({max_position[0] + 1}, {max_position[1] + 1})")
```

The program initializes a 5x5 matrix named ***matrix*** with all elements initially set to 0.

[0] * 5: This creates a list ***[0, 0, 0, 0, 0]*** with 5 zeros.

for _ in range(5): This loop runs 5 times, representing the number of rows in the matrix.

[[0] * 5 for _ in range(5)]: This creates a 2D list where each row is a copy of the list ***[0, 0, 0, 0, 0]***. The loop creates a new row for each iteration, resulting in a 5x5 matrix where all elements are initially set to 0.

The program prompts the user to enter the elements of the matrix using nested ***for*** loops. The user is asked to enter each element individually, and the values are stored in the corresponding positions of the matrix.

A variable named ***max_value*** is initialized to the value at position (0, 0) in the matrix, and a tuple named ***max_position*** is initialized with the coordinates (0, 0).

Using nested ***for*** loops, the program iterates through each element of the matrix. If an element is found to be greater than the current ***max_value***, it updates ***max_value*** to the new maximum value and updates ***max_position*** to the position of that element.

Finally, the program displays the largest value in the matrix and its position by accessing ***max_value*** and ***max_position*** variables, respectively. The position is displayed by adding 1 to the row and column indices to account for the 0-based indexing.

85. Write a program that reads a 3x3 matrix and calculates the average of the values present in the even positions (sum of the even indices) of the matrix.

```
# Initialize a 3x3 matrix
matrix = []

# Read the matrix from the user
print("Enter the values of the 3x3 matrix:")
for _ in range(3):
    row = []
    for _ in range(3):
        value = int(input("Enter a value: "))
        row.append(value)
    matrix.append(row)

# Calculate the sum and count of values at even positions
sum_even = 0
count_even = 0
```

```
for i in range(3):
    for j in range(3):
        if (i + j) % 2 == 0:  # Check if the sum of indices is even
            sum_even += matrix[i][j]
            count_even += 1

# Calculate the average of values at even positions
average_even = sum_even / count_even

# Display the result
print("Average of values at even positions:", average_even)
```

We start by initializing an empty matrix using an empty list: ***matrix = []***. This matrix will store the values entered by the user.

We use a nested loop to read the values of the 3x3 matrix from the user and store them in the matrix variable. The outer loop iterates over the rows, and the inner loop iterates over the columns.

- In the outer loop (***for _ in range(3)***), we create a new empty list called ***row*** for each row in the matrix.
- In the inner loop (***for _ in range(3)***), we prompt the user to enter a value and convert it to an integer using ***int(input("Enter a value: "))***. We then append the value to the row list.
- After each row is filled, we append the ***row*** list to the ***matrix*** list, effectively adding the row to the matrix.

Next, we initialize two variables: ***sum_even*** to store the sum of values at even positions, and ***count_even*** to keep track of the number of values at even positions.

We iterate over each element of the matrix using two nested loops, one for rows and one for columns.

Inside the loop, we check if the sum of the row index ***i*** and column index ***j*** is even using the expression ***(i + j) % 2 == 0***.

- If the sum is even, it means the element is at an even position in the matrix. We add the corresponding value ***matrix[i][j]*** to the ***sum_even*** variable and increment the ***count_even*** variable.

Finally, we calculate the average of the values at even positions by dividing the ***sum_even*** by the ***count_even***.

The program then displays the calculated average to the user using the ***print()*** function.

86. Write a program that fills a 4x4 matrix with random numbers and displays the sum of the values present in each row and in each column.

```
import random

# Create a 4x4 matrix filled with random numbers
matrix = [[random.randint(1, 10) for _ in range(4)] for _ in range(4)]

# Initialize variables to store the sums of rows and columns
row_sums = [0] * 4
column_sums = [0] * 4

# Calculate the sum of values in each row and column
for i in range(4):
    for j in range(4):
        # Update the sum of the i-th row
        row_sums[i] += matrix[i][j]

        # Update the sum of the j-th column
        column_sums[j] += matrix[i][j]

# Display the sums of rows
print("Sum of values in each row:")
for i in range(4):
    print("Row", i + 1, ":", row_sums[i])

# Display the sums of columns
print("Sum of values in each column:")
for j in range(4):
    print("Column", j + 1, ":", column_sums[j])
```

We start by importing the ***random*** module to generate random numbers.

We create a 4x4 matrix filled with random numbers using a nested list comprehension. Each element in the matrix is generated using

random.randint(1, 10), which generates a random integer between 1 and 10.

We initialize two lists, ***row_sums*** and ***column_sums***, with zeros to store the sums of values in each row and column, respectively.

We use nested loops to iterate over each element of the matrix. Inside the loop, we update the sum of the ***i***-th row by adding the value ***matrix[i][j]*** to ***row_sums[i]***, and update the sum of the ***j***-th column by adding the same value to ***column_sums[j]***.

After calculating the sums, we display the sum of values in each row using a loop and the ***print()*** function. Similarly, we display the sum of values in each column using a loop and the ***print()*** function.

Explaining better the creation of the matrix

- ***random.randint(1, 10)*** is a function from the random module that generates a random integer between the specified range, in this case, between 1 and 10 (inclusive).
- The outer ***for*** loop ***for _ in range(4)*** is used to iterate 4 times, creating 4 rows in the matrix. The _ is a convention often used as a throwaway variable name when the variable itself is not needed.
- The inner ***for*** loop ***for _ in range(4)*** is used to iterate 4 times within each row, creating 4 elements in each row of the matrix.
- The expression ***random.randint(1, 10)*** is used as the value for each element in the row. This creates a new random integer between 1 and 10 for each element in the row.
- The inner list comprehension ***[random.randint(1, 10) for _ in range(4)]*** generates a list of 4 random integers between 1 and 10. This represents a single row in the matrix.
- The outer list comprehension ***[[random.randint(1, 10) for _ in range(4)] for _ in range(4)]*** uses the inner list comprehension to create 4 rows in the matrix. Each row is generated independently, resulting in a 4x4 matrix.

87. Write a program that reads a 3x3 matrix and calculates the determinant of the matrix.

```
# Read the elements of the 3x3 matrix from the user
matrix = []
for _ in range(3):
    row = []
    for _ in range(3):
        element = int(input("Enter an element: "))
        row.append(element)
    matrix.append(row)

# Calculate the determinant using the formula
det = (
    matrix[0][0] * (matrix[1][1] * matrix[2][2] - matrix[1][2] * matrix[2][1])
    - matrix[0][1] * (matrix[1][0] * matrix[2][2] - matrix[1][2] * matrix[2][0])
    + matrix[0][2] * (matrix[1][0] * matrix[2][1] - matrix[1][1] * matrix[2][0])
)

# Display the determinant
print("Determinant:", det)
```

In this program, we first read the elements of the 3x3 matrix from the user and store them in the ***matrix*** list. Then, we use the determinant formula to calculate the determinant value and store it in the ***det*** variable. Finally, we display the determinant using the ***print*** statement.

We use the expansion formula to calculate the determinant:

$$\det\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix}$$

$$= 1 \cdot \det\begin{pmatrix}5 & 6\\ 8 & 9\end{pmatrix} - 2 \cdot \det\begin{pmatrix}4 & 6\\ 7 & 9\end{pmatrix} + 3 \cdot \det\begin{pmatrix}4 & 5\\ 7 & 8\end{pmatrix}$$

Using Sarrus rule to calculate determinant

```
# Read the elements of the 3x3 matrix from the user
matrix = []
for _ in range(3):
    row = []
    for _ in range(3):
        element = int(input("Enter an element: "))
        row.append(element)
    matrix.append(row)

# Calculate the determinant using the Sarrus rule
det = (
    matrix[0][0] * matrix[1][1] * matrix[2][2]
    + matrix[0][1] * matrix[1][2] * matrix[2][0]
    + matrix[0][2] * matrix[1][0] * matrix[2][1]
    - matrix[2][0] * matrix[1][1] * matrix[0][2]
    - matrix[2][1] * matrix[1][2] * matrix[0][0]
    - matrix[2][2] * matrix[1][0] * matrix[0][1]
)

# Display the determinant
print("Determinant:", det)
```

In this program, we first read the elements of the 3x3 matrix from the user and store them in the ***matrix*** list. Then, we calculate the determinant using the Sarrus rule, which involves multiplying certain elements of the matrix and adding or subtracting them based on their positions. Finally, we display the calculated determinant.

88. Write a program that reads two matrices and returns the multiplication between them as an answer. The program should observe whether or not it is possible to perform the multiplication between the two matrices.

```python
# Read the dimensions of matrix A
rows_a = int(input("Enter the number of rows of matrix A: "))
cols_a = int(input("Enter the number of columns of matrix A: "))

# Read the dimensions of matrix B
rows_b = int(input("Enter the number of rows of matrix B: "))
cols_b = int(input("Enter the number of columns of matrix B: "))

# Check if multiplication is possible
if cols_a != rows_b:
    print("Matrix multiplication is not possible.")
else:
    # Read matrix A
    print("Enter the elements of matrix A:")
    matrix_a = []
    for _ in range(rows_a):
        row = []
        for _ in range(cols_a):
            element = int(input())
            row.append(element)
        matrix_a.append(row)
```

```
# Read matrix B
print("Enter the elements of matrix B:")
matrix_b = []
for _ in range(rows_b):
    row = []
    for _ in range(cols_b):
        element = int(input())
        row.append(element)
    matrix_b.append(row)

# Multiply the matrices
result = [[0] * cols_b for _ in range(rows_a)]

for i in range(rows_a):
    for j in range(cols_b):
        for k in range(cols_a):
            result[i][j] += matrix_a[i][k] * matrix_b[k][j]

# Display the result
print("The result of the multiplication of matrices A and B is:")
for row in result:
    print(row)
```

We start by reading the dimensions of the two matrices: ***rows_a***, ***cols_a*** for matrix A and ***rows_b***, ***cols_b*** for matrix B. This allows the user to specify the number of rows and columns for each matrix.

Next, we check if matrix multiplication is possible by comparing the number of columns of matrix A (***cols_a***) with the number of rows of matrix B (***rows_b***). If these values are not equal, it means the matrices cannot be multiplied, so we display a message indicating that matrix multiplication is not possible.

If matrix multiplication is possible, we proceed to read the elements of matrix A. We use nested loops to iterate over each row and column of matrix A and prompt the user to enter the corresponding element. We store these elements in the ***matrix_a*** list.

Similarly, we read the elements of matrix B using nested loops and store them in the ***matrix_b*** list.

Now that we have both matrices A and B, we create a result matrix (***result***) to store the result of the multiplication. We initialize it as a matrix of zeros with dimensions ***rows_a*** x ***cols_b***.

To perform the matrix multiplication, we use three nested loops. The outer loop iterates over the rows of matrix A, the middle loop iterates over the columns of matrix B, and the inner loop iterates over the columns of matrix A (or rows of matrix B). This allows us to multiply the corresponding elements and accumulate the sum in the corresponding position of the result matrix.

Finally, we display the result of the matrix multiplication by iterating over each row of the result matrix and printing the elements. This gives us the final matrix that represents the multiplication of matrices A and B.

89. Write a program that reads a 4x4 matrix and checks if it is a diagonal matrix, that is, if all elements outside the main diagonal are equal to zero.

```
# Read the elements of the matrix
matrix = []
for i in range(4):
    row = []
    for j in range(4):
        element = int(input(f"Element at pos. ({i}, {j}): "))
        row.append(element)
    matrix.append(row)

# Check if it is a diagonal matrix
is_diagonal = True
for i in range(4):
    for j in range(4):
        if i != j and matrix[i][j] != 0:
            is_diagonal = False
            break

# Display the result
if is_diagonal:
    print("The matrix is a diagonal matrix.")
else:
    print("The matrix is not a diagonal matrix.")
```

We start by creating an empty list ***matrix*** to store the elements of the matrix.

Using nested loops, we iterate over each row and column of the matrix (4x4 in this case) and prompt the user to enter the corresponding element. We store these elements in the ***matrix*** list.

Next, we initialize a boolean variable ***is_diagonal*** as ***True*** to assume that the matrix is a diagonal matrix. This variable will be used to track if any element outside the main diagonal is non-zero.

To check if the matrix is a diagonal matrix, we iterate over each element of the matrix using nested loops. For each element at position ***(i, j)***, if ***i*** is not equal to ***j*** (indicating an element outside the main diagonal) and the element is non-zero, we set ***is_diagonal*** to ***False*** and break out of the loop.

Finally, we display the result by checking the value of ***is_diagonal***. If it is ***True***, we print that the matrix is a diagonal matrix. Otherwise, we print that it is not a diagonal matrix.

90. Write a program that reads an m x n matrix, indicating the location where there are mines in a Minesweeper game (being 0 for a neutral field, and 1 for locations where there would be mines), and the program should return a matrix indicating, for each position, the number of mines in neighboring houses.

```
# Read the dimensions of the matrix
m = int(input("Enter the number of rows: "))
n = int(input("Enter the number of columns: "))

# Read the minefield matrix
minefield = []
for i in range(m):
    row = []
    for j in range(n):
        element = int(input(f"Element at position ({i}, {j}): "))
        row.append(element)
    minefield.append(row)

# Calculate the number of neighboring mines
for i in range(m):
    for j in range(n):
        # Check the eight neighboring positions
        for dx in [-1, 0, 1]:
            for dy in [-1, 0, 1]:
                new_i = i + dx
                new_j = j + dy
                # Check if the neighboring position is within bounds
                if 0 <= new_i < m and 0 <= new_j < n:
                    # Increment the count if there is a mine
                    neighboring_mines[i][j] += minefield[new_i][new_j]
```

We start by prompting the user to enter the dimensions of the matrix, ***m*** (number of rows) and ***n*** (number of columns).

Using nested loops, we iterate over each row and column of the minefield matrix. We prompt the user to enter the element at each position and store these elements in the ***minefield*** matrix.

Next, we initialize the ***neighboring_mines*** matrix with zeros. This matrix will store the number of neighboring mines for each position.

We calculate the number of neighboring mines for each position in the ***minefield*** matrix. We iterate over each row and column of the ***minefield*** matrix using nested loops.

For each position ***(i, j)*** in the ***minefield*** matrix, we check the eight neighboring positions by using nested loops with offsets ***dx*** and ***dy*** ranging from -1 to 1. We calculate the indices of the neighboring positions (***new_i*** and ***new_j***) by adding the offsets to the current position indices.

Inside the innermost loop, we check if the neighboring position ***(new_i, new_j)*** is within the bounds of the matrix (***0*** <= ***new_i*** < ***m*** and ***0*** <= ***new_j*** < ***n***). If it is within bounds, we increment the ***neighboring_mines[i][j]*** count if there is a mine (***minefield[new_i][new_j]*** is 1).

Finally, we display the ***neighboring_mines*** matrix, which represents the number of neighboring mines for each position in the minefield.

Recursive Functions

In Python, a function is a block of reusable code that performs a specific task. Functions help in organizing code, improving code reusability, and reducing code duplication. They allow you to break down a complex problem into smaller, manageable parts.

Here's the basic syntax for defining a function in Python:

```
def function_name(parameter1, parameter2):
    # Function body
    # Code statements
    # Return statement (optional)
```

Let's break down the components of a function:

def: It is a keyword used to define a function.

function_name: It is the name of the function. Choose a meaningful name that describes the purpose of the function.

(parameter1, parameter2): Parameters are optional placeholders that receive values when the function is called. They are enclosed in parentheses and separated by commas. You can have zero or more parameters.

Function body: It consists of the code statements that define the functionality of the function. Indentation is important in Python, and the function body should be indented by four spaces (or a tab) to indicate it is part of the function.

return statement (optional): It is used to specify the value that the function should return when it is called. If no return statement is provided, the function returns None by default.

Here's an example of a simple function that adds two numbers and returns the result:

```
def add_numbers(a, b):
    sum = a + b
    return sum

result = add_numbers(3, 4)
print(result)  # Output: 7
```

In this example, the ***add_numbers*** function takes two parameters ***a*** and ***b***. It calculates the sum of the two numbers and returns the result using the

return statement. The function is called with arguments 3 and 4, and the returned value is stored in the variable ***result***, which is then printed.

Functions can have multiple parameters, perform complex calculations, include conditional statements, use loops, and call other functions. They can be defined anywhere in the program, but it is a good practice to define them before they are called.

Functions help in organizing code into reusable blocks, making the code easier to read, understand, and maintain. They enable modular programming and promote code reusability, allowing you to use the same function in different parts of your program or in other programs.

Default values

In Python, you can provide default values for function parameters. Default values allow you to specify a value that a parameter should take if no argument is provided when the function is called. This provides flexibility by making certain parameters optional.

When a function is called, if an argument is provided for a parameter, the provided value is used. However, if no argument is provided, the default value specified in the function definition is used instead.

Here's an example that demonstrates the use of default values in a function:

```
def greet(name, message="Hello"):
    print(message, name)

greet("Alice")  # Output: Hello Alice
greet("Bob", "Hi")  # Output: Hi Bob
```

In this example, the ***greet*** function has two parameters: ***name*** and ***message***. The ***message*** parameter has a default value of "*Hello*". When the function is called with only one argument (***greet("Alice")***), the default value is used for the ***message*** parameter, resulting in the output "*Hello Alice*". When the function is called with two arguments (***greet("Bob", "Hi")***), the provided value "*Hi*" is used for the ***message*** parameter, resulting in the output "*Hi Bob*".

It's important to note that parameters with default values should be placed at the end of the parameter list. This is because when calling a function, arguments are assigned to parameters based on their position. If

you have a mix of parameters with default values and without default values, the parameters without default values should come before the ones with default values.

Default values in function parameters provide a convenient way to make certain parameters optional, allowing you to provide a sensible default behavior while still offering flexibility to override the default values when needed.

Now, let's go to the exercises.

91. Write a recursive function to calculate the factorial of a number.

```
def factorial(n):
    # Base case: factorial of 0 or 1 is 1
    if n == 0 or n == 1:
        return 1
    # Recursive case: multiply n by the factorial of (n-1)
    else:
        return n * factorial(n - 1)
```

The ***factorial*** function takes an input ***n***, which represents the number for which we want to calculate the factorial.

The base case is defined when ***n*** is equal to 0 or 1. In these cases, the factorial is defined as 1.

In the recursive case, the function multiplies ***n*** by the factorial of (***n-1***). This step recursively calls the ***factorial*** function with a smaller value of ***n*** until it reaches the base case.

The function returns the calculated factorial value.

92. Implement a recursive function to calculate the Fibonacci sequence up to a given number.

```
def fibonacci(n):
    # Base case: Fibonacci of 0 is 0, and Fibonacci of 1 is 1
    if n == 0:
        return 0
    elif n == 1:
        return 1
    # Recursive case: Fibonacci of n is the sum of
    # the previous two Fibonacci numbers
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

The fibonacci function takes an input ***n***, which represents the position in the Fibonacci sequence for which we want to calculate the value.

The base case is defined when ***n*** is equal to 0 or 1. In these cases, the Fibonacci value is defined as 0 and 1, respectively.

In the recursive case, the function calculates the Fibonacci value of ***n*** by recursively calling the ***fibonacci*** function with (***n-1***) and (***n-2***), and adding the results together.

The function returns the calculated Fibonacci value.

93. Create a recursive function to check if a number is prime.

```
def is_prime(n, divisor=2):
    # Base cases
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % divisor == 0:
        return False
    if divisor * divisor > n:
        return True
    # Recursive case
    return is_prime(n, divisor + 1)
```

The ***is_prime*** function takes an input ***n*** and an optional parameter ***divisor*** to check if ***n*** is divisible by ***divisor***.

The base cases are defined as follows:

- If ***n*** is less than or equal to 1, it is not prime, so we return ***False***.
- If ***n*** is equal to 2, it is the only even prime number, so we return ***True***.
- If ***n*** is divisible by the current ***divisor***, it is not prime, so we return ***False***.
- If the square of the ***divisor*** is greater than ***n***, we have checked all possible divisors, and ***n*** is prime, so we return ***True***.

In the recursive case, the function calls itself with the same ***n*** and increments ***divisor*** by 1.

The function returns the result of the recursive call.

94. Develop a recursive function to calculate the sum of the digits of an integer.

```
def sum_of_digits(n):
    # Base case
    if n == 0:
        return 0
    # Recursive case
    return (n % 10) + sum_of_digits(n // 10)
```

The ***sum_of_digits*** function takes an input ***n*** representing the integer for which we want to calculate the sum of digits.

The base case is when ***n*** becomes 0. In this case, there are no more digits to process, so we return 0.

In the recursive case, the function extracts the last digit of ***n*** using the modulus operator ***%*** and adds it to the sum of the remaining digits.

The function calls itself with the updated value of ***n*** obtained by integer division ***n // 10***, which removes the last digit.

The recursive calls continue until the base case is reached.

The function returns the sum of the current digit and the result of the recursive call.

95. Write a recursive function to calculate the power of an integer raised to an exponent.

```
def power(base, exponent):
    # Base case: exponent is 0, return 1
    if exponent == 0:
        return 1
    # Recursive case: multiply base with
    # power of (base, exponent-1)
    return base * power(base, exponent - 1)
```

The ***power*** function takes two arguments: ***base*** and ***exponent***, representing the number and the power to which it will be raised.

The base case is when ***exponent*** becomes 0. In this case, any number raised to the power of 0 is 1, so we return 1.

In the recursive case, the function multiplies ***base*** with the result of the recursive call to ***power***, where the ***exponent*** is decreased by 1.

The recursive calls continue until the base case is reached.

The function returns the result of multiplying ***base*** with the power of ***(base, exponent-1)***.

96. Implement a recursive function to find the greatest common divisor (GCD) of two numbers.

```
def gcd(a, b):
    # Base case: if b is 0, a is the GCD
    if b == 0:
        return a
    # Recursive case: compute GCD of b and the
    # remainder of a divided by b
    return gcd(b, a % b)
```

The ***gcd*** function takes two arguments: ***a*** and ***b***, representing the two numbers for which we want to find the GCD.

The base case is when ***b*** becomes 0. In this case, we have found the GCD, which is ***a***.

In the recursive case, the function calculates the GCD of ***b*** and the remainder of a divided by ***b*** using the Euclidean algorithm.

The recursive calls continue until the base case is reached.

97. Create a recursive function to reverse a string.

```
def reverse_string(string):
    # Base case: if the string is empty or
    # has only one character, return the string itself
    if len(string) <= 1:
        return string
    # Recursive case: reverse the substring starting
    # from the second character and append the first character
    return reverse_string(string[1:]) + string[0]
```

The ***reverse_string*** function takes a string ***string*** as an argument.

The base case is when the length of the string is 0 or 1. In this case, there's no need to reverse the string, so it is returned as is.

In the recursive case, the function recursively calls itself, passing the substring starting from the second character (***string[1:]***), and then appends the first character of the original string (***string[0]***).

The recursive calls continue until the base case is reached, and the reversed substrings are concatenated to reverse the entire string.

The function returns the reversed string.

98. Develop a recursive function to find the smallest value in an array.

```
def find_smallest(arr):
    # Base case: if the array has only one element,
    # return that element
    if len(arr) == 1:
        return arr[0]
    # Recursive case: compare the first element
    # with the smallest element in the rest of the array
    return min(arr[0], find_smallest(arr[1:]))
```

The ***find_smallest*** function takes an array ***arr*** as an argument.

The base case is when the array has only one element. In this case, there's no need to compare, so the single element is returned as the smallest value.

In the recursive case, the function compares the first element of the array (***arr[0]***) with the smallest element in the rest of the array (***find_smallest(arr[1:])***).

The recursive calls continue until the base case is reached, and the smallest values are compared and propagated back up.

99. Write a recursive function to determine whether a word is a palindrome.

```
def is_palindrome(word):
    # Base case: if the word has 0 or 1 character, it is a palindrome
    if len(word) <= 1:
        return True
    # Recursive case: check if the first and last characters are equal,
    # and recursively check if the remaining substring is a palindrome
    if word[0] == word[-1]:
        return is_palindrome(word[1:-1])
    else:
        return False
```

The ***is_palindrome*** function takes a word as an argument.

The base case is when the word has 0 or 1 character. In these cases, the word is considered a palindrome, so ***True*** is returned.

In the recursive case, the function checks if the first and last characters of the word are equal.

If they are equal, the function recursively calls itself with the remaining substring (***word[1:-1]***) to check if the substring is also a palindrome.

The recursive calls continue until the base case is reached, and the results are propagated back up.

If at any point the first and last characters are not equal, the function returns False as the word is not a palindrome.

100. Implement a recursive function to calculate the sum of elements of an array.

```
def vector_sum(vector):
    if len(vector) == 0:
        return 0
    else:
        return vector[0] + vector_sum(vector[1:])
```

The ***vector_sum*** function takes a vector (list) as an argument.

The base case is when the vector is empty. In this case, the sum is 0, so we return 0.

In the recursive case, the function adds the first element of the vector to the sum of the remaining elements.

To calculate the sum of the remaining elements, we make a recursive call to vector_sum with the slice ***vector[1:]***, which represents the vector without the first element.

The recursive calls continue until the base case is reached, and the results are propagated back up.

Extra Exercises

101. Write a program that prompts the user for a sentence and displays the sentence backwards

```
sentence = input("Enter a sentence: ")
reversed_sentence = ' '.join(reversed(sentence.split()))
print("Reversed sentence:", reversed_sentence)
```

The program starts by prompting the user to enter a sentence using the ***input()*** function. The entered sentence is stored in the ***sentence*** variable.

The ***split()*** method is then applied to the ***sentence*** variable, which splits the sentence into a list of words. By default, ***split()*** splits the sentence based on whitespace.

The ***reversed()*** function is used to reverse the order of the words in the list generated by ***split()***. The ***reversed()*** function returns an iterator that allows us to iterate over the words in reverse order.

The ***reversed()*** function returns the iterator, so we convert it back to a list using the ***list()*** function. However, instead of converting it to a list, we directly pass it to the ***join()*** method.

The ***join()*** method is used to join the reversed words using a space as the separator. It concatenates the words in the list into a single string, separated by the specified separator.

Finally, the reversed sentence is printed using the ***print()*** function.

102. Make the BubbleSort Algorithm

The Bubble Sort algorithm is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The algorithm continues to pass through the list until the entire list is sorted. Here's the implementation of the Bubble Sort algorithm in Python:

```
def bubble_sort(arr):
    n = len(arr)

    # Traverse through all array elements
    for i in range(n):

        # Last i elements are already in place
        for j in range(0, n-i-1):

            # Swap if the element found is greater than the next ele
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

# Test the bubble_sort function
arr = [64, 34, 25, 12, 22, 11, 90]
print("Original array:", arr)

bubble_sort(arr)

print("Sorted array:", arr)
```

The ***bubble_sort*** function takes an array ***arr*** as an input.

The length of the array is stored in the variable ***n***.

The outer loop iterates ***n*** times, representing the number of passes needed to sort the array.

Inside the outer loop, the inner loop iterates from the first element to the second-to-last element in the unsorted part of the array.

Within the inner loop, we compare adjacent elements ***arr[j]*** and ***arr[j+1]***. If ***arr[j]*** is greater than ***arr[j+1]***, we swap the elements using a simultaneous assignment.

After the inner loop completes for a pass, the largest element in the unsorted part of the array "bubbles up" to the end of the array.

The process repeats until the outer loop completes all the passes.

Finally, we test the ***bubble_sort*** function by initializing an array ***arr*** and printing the original and sorted arrays.

103. Make an algorithm that solves the Tower of Hanoi

The Tower of Hanoi is a classic mathematical puzzle that consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a stack on one rod in ascending order of size, with the smallest disk at the top. The objective is to move the entire stack to another rod, following these rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
- No disk may be placed on top of a smaller disk.

Here's a recursive algorithm to solve the Tower of Hanoi puzzle in Python:

```
def tower_of_hanoi(n, source, destination, auxiliary):
    if n > 0:
        # Move n-1 disks from source to auxiliary rod
        tower_of_hanoi(n-1, source, auxiliary, destination)

        # Move the nth disk from source to destination rod
        print(f"Move disk {n} from {source} to {destination}")

        # Move the n-1 disks from auxiliary to destination rod
        tower_of_hanoi(n-1, auxiliary, destination, source)

# Test the tower_of_hanoi function
n = 3  # Number of disks
source = "A"  # Source rod
destination = "C"  # Destination rod
auxiliary = "B"  # Auxiliary rod

print(f"Solving Tower of Hanoi with {n} disks:")
tower_of_hanoi(n, source, destination, auxiliary)
```

Let's break down the algorithm step by step:

The ***tower_of_hanoi*** function takes four parameters: ***n*** (the number of disks), ***source*** (the rod from which to move the disks), ***destination*** (the rod where the disks should be moved), and ***auxiliary*** (the spare rod to facilitate the moves).

The base case for the recursive function is when ***n*** is 0 (or less), in which case there are no disks to move, so the function returns.

In the recursive case, the algorithm follows three steps:

- Move ***n-1*** disks from the ***source*** rod to the ***auxiliary*** rod, using the ***destination*** rod as a spare rod. This is done by recursively calling the ***tower_of_hanoi*** function with the parameters appropriately swapped.
- Move the ***n***th disk from the ***source*** rod to the ***destination*** rod.
- Move the ***n***-1 disks from the ***auxiliary*** rod to the ***destination*** rod, using the source rod as a spare rod. Again, this is done by recursively calling the ***tower_of_hanoi*** function with the parameters swapped.

Finally, we test the ***tower_of_hanoi*** function by setting the number of disks ***n*** to 3 and specifying the source rod as "A", the destination rod as "C", and the auxiliary rod as "B". The function is called with these parameters, and the moves are printed out to solve the Tower of Hanoi puzzle.

104. Make a function that receives a 3x3 matrix representing the game of tic-tac-toe, and check if there is a winner, if there is a tie, or if the game is not over yet.

```
def check_tic_tac_toe(matrix):
    # Check rows
    for row in matrix:
        if row[0] == row[1] == row[2] and row[0] != ' ':
            return row[0]

    # Check columns
    for col in range(3):
        if matrix[0][col] == matrix[1][col] == matrix[2][col] and matrix[0][col] != ' ':
            return matrix[0][col]

    # Check diagonals
    if matrix[0][0] == matrix[1][1] == matrix[2][2] and matrix[0][0] != ' ':
        return matrix[0][0]

    if matrix[0][2] == matrix[1][1] == matrix[2][0] and matrix[0][2] != ' ':
        return matrix[0][2]

    # Check if game is not over yet
    for row in matrix:
        if ' ' in row:
            return "Game not over yet"

    # If no empty spaces and no winner, it's a tie
    return "Tie"
```

The ***check_tic_tac_toe*** function takes a 3x3 matrix ***matrix*** as an input.

It first checks for a winning condition in the rows. It iterates over each row of the matrix and checks if all three elements in a row are the same and not empty (' '). If a winning condition is found, the corresponding symbol ('X' or 'O') is returned.

Next, it checks for a winning condition in the columns. It iterates over each column of the matrix and checks if all three elements in a column are the same and not empty. If a winning condition is found, the corresponding symbol is returned.

The function then checks the diagonals. It checks if the elements in the top-left to bottom-right diagonal and the top-right to bottom-left diagonal are the same and not empty. If a winning condition is found, the corresponding symbol is returned.

If no winning condition is found, the function checks if the game is not over yet. It iterates over each row and checks if there are any empty spaces (' '). If an empty space is found, it returns "*Game not over yet*".

If there are no empty spaces and no winner, the function returns "*Tie*" to indicate a tie game.

Now you can call the ***check_tic_tac_toe*** function with your tic-tac-toe matrix to determine the outcome of the game.

105. Rawwords: Write an algorithm that checks whether a word is a "rawword". A word is considered a "prime word" if the sum of the letter values (where 'a' = 1, 'b' = 2, etc.) is a prime number.

To check whether a word is a "rawword" (a prime word), we can write an algorithm that calculates the sum of the letter values in the word and then checks if the sum is a prime number. Here's a step-by-step algorithm to accomplish this:

Define a function, let's call it ***is_rawword***, that takes a word as input.

Create a dictionary, let's call it ***letter_values***, to store the letter values. Map each letter of the alphabet to its corresponding value. For example, 'a' maps to 1, 'b' maps to 2, and so on. You can create this dictionary manually or programmatically.

Initialize a variable, let's call it ***word_sum***, to keep track of the sum of letter values in the word. Set it to 0 initially.

Iterate through each character in the word.

- Convert the character to lowercase to handle both uppercase and lowercase letters consistently.
- Check if the character is a letter (using **isalpha()** method). If not, continue to the next character.
- Get the letter value from the ***letter_values*** dictionary using the character as the key and add it to ***word_sum***.

Once the iteration is complete, ***word_sum*** will contain the sum of the letter values in the word.

Implement a separate helper function, let's call it ***is_prime***, that takes a number as input and checks whether it is a prime number. There are several approaches to checking for prime numbers, such as trial division or the Sieve of Eratosthenes. You can choose an approach that suits your requirements.

Finally, in the ***is_rawword*** function, call the ***is_prime*** function with ***word_sum*** as the input. If ***word_sum*** is a prime number, return ***True***, indicating that the word is a rawword. Otherwise, return ***False***.

Here's a Python implementation of the algorithm:

```
def is_prime(number):
    if number < 2:
        return False
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return False
    return True

def is_rawword(word):
    letter_values = {
        'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9,
        'j': 10, 'k': 11, 'l': 12, 'm': 13, 'n': 14, 'o': 15, 'p': 16, 'q': 17,
        'r': 18, 's': 19, 't': 20, 'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 25, 'z': 26
    }
    word_sum = 0
    for char in word.lower():
        if char.isalpha():
            word_sum += letter_values[char]
    return is_prime(word_sum)
```

You can now use the ***is_rawword*** function to check whether a word is a rawword. For example:

```
word = "hello"
if is_rawword(word):
    print(f"{word} is a rawword.")
else:
    print(f"{word} is not a rawword.")
```

This will output "*hello is not a rawword.*" since the sum of letter values for "*hello*" is not a prime number.

106. Implement an algorithm that takes an integer and generates the Collatz sequence for that number. The Collatz sequence is generated by applying the following rules: if the number is even, divide it by 2; if the number is odd, multiply it by 3 and add 1. Repeat this process until you reach the number 1.

The Collatz sequence algorithm can be implemented using a simple while loop. Here's the step-by-step algorithm:

Define a function, let's call it ***collatz_sequence***, that takes an integer number as input.

Create an empty list, let's call it ***sequence***, to store the Collatz sequence.

Append the initial number to the sequence list.

Start a ***while*** loop with the condition ***number != 1***. This loop will continue until the number becomes 1.

Inside the loop, check if the number is even. You can do this by checking if ***number % 2 == 0***.

If the number is even, divide it by 2 and update the value of ***number*** accordingly.

If the number is odd, multiply it by 3 and add 1, and update the value of ***number***.

After updating the value of ***number***, append it to the sequence list.

Finally, return the sequence list.

Here's a Python implementation of the algorithm:

```
def collatz_sequence(number):
    sequence = [number]
    while number != 1:
        if number % 2 == 0:
            number = number // 2
        else:
            number = number * 3 + 1
        sequence.append(number)
    return sequence
```

Complete List of Exercises

1. Write a program that prompts the user for two numbers and displays the addition, subtraction, multiplication, and division between them.
2. Write a program that calculates the arithmetic mean of two numbers.
3. Create a program that calculates and displays the arithmetic mean of three grades entered by the user.
4. Write a program that calculates the geometric mean of three numbers entered by the user
5. Write a program that calculates the BMI of an individual, using the formula BMI = weight / height²
6. Create a program that calculates and displays the perimeter of a circle, prompting the user for the radius.
7. Write a program that calculates the area of a circle from the radius, using the formula $A = \pi r^2$
8. Write a program that calculates the delta of a quadratic equation ($\Delta = b^2 - 4ac$).
9. Write a program that calculates the perimeter and area of a rectangle, using the formulas $P = 2(w + l)$ and $A = wl$, where w is the width and l is the length
10. Write a program that calculates the perimeter and area of a triangle, using the formulas $P = a + b + c$ and $A = (b * h) / 2$, where a, b and c are the sides of the triangle and h is the height relative to the side B.
11. Write a program that calculates the average velocity of an object, using the formula $v = \Delta s/\Delta t$, where v is the average velocity, Δs is the space variation, and Δt is the time variation
12. Write a program that calculates the kinetic energy of a moving object, using the formula $E = (mv^2) / 2$, where E is the kinetic energy, m is the mass of the object, and v is the velocity.
13. Write a program that calculates the work done by a force acting on an object, using the formula T = F * d, where T is the work, F is the applied force, and d is the distance traveled by the object.
14. Write a program that reads the x and y position of two points in the

Cartesian plane, and calculates the distance between them.

15. Create a program that prompts the user for the radius of a sphere and calculates and displays its volume.

16. Make a program that asks for a person's age and displays whether they are of legal age or not.

17. Write a program that reads two numbers and tells you which one is bigger.

18. Write a program that asks the user for three numbers and displays the largest one.

19. Write a program that reads a number and reports whether it is odd or even.

20. Write a program that reads a number and reports whether it is positive, negative or zero.

21. Make a program that reads the scores of two tests and reports whether the student passed (score greater than or equal to 6) or failed (score less than 6) in each of the tests.

22. Make a program that reads the grades of two tests, calculates the simple arithmetic mean, and informs whether the student passed (average greater than or equal to 6) or failed (average less than 6).

23. Make a program that reads three numbers, and informs if their sum is divisible by 5 or not.

24. Create a program that reads three numbers and checks if their sum is positive, negative or equal to zero

25. Make a program that reads three numbers, and displays them on the screen in ascending order.

26. Make a program that reads the age of three people and how many of them are of legal age (age 18 or older).

27. Write a program that reads three numbers and tells you if they can be the sides of a triangle (the sum of two sides must always be greater than the third side).

28. Make a program that reads the year of birth of a person and informs if he is able to vote (age greater than or equal to 16 years old).

29. Make a program that reads a person's age and informs if he is not able to vote (age less than 16 years old), if he is able to vote but is not obligated

(16, 17 years old, or age equal to or greater than 70 years), or if it is obligatory (18 to 69 years old).

30. Make a program that reads three grades from a student and reports whether he passed (final grade greater than or equal to 7), failed (final grade less than 4) or was in recovery (final grade between 4 and 7).

31. Write a program that asks for the name of a day of the week and displays whether it is a weekday (Monday to Friday) or a weekend day (Saturday and Sunday).

32. Write a program that asks for a person's height and weight and calculates their body mass index (BMI), displaying the corresponding category (underweight, normal weight, overweight, obese, severely obese).

33. Write a program that asks for an integer and checks if it is divisible by 3 and 5 at the same time.

34. Create a program that asks for a person's age and displays whether they are a child (0-12 years old), teenager (13-17 years old), adult (18-59 years old), or elderly (60 years old or older).

35. Make a program that asks for two numbers and displays if the first is divisible by the second

36. Write a program that displays the numbers 1 through 10 using a loop.

37. Write a program that displays all numbers from 1 to 100

38. Write a program that prints all even numbers from 1 to 100.

39. Write a program that displays even numbers 1 to 50 and odd numbers 51 to 100 using a repeating loop.

40. Create a program that prompts the user for a number and displays the table of that number using a loop.

41. Create a program that prompts the user for a number and displays the table of that number using a loop.

42. Write a program that asks the user for a number N and displays the sum of all numbers from 1 to N.

43. Write a program that calculates and displays the sum of even numbers from 1 to 100 using a repeating loop.

44. Write a program that calculates and displays the value of the power of a number entered by the user raised to an exponent also entered by the user, using repetition loops.

45. Write a program that asks the user for a number N and says whether it is prime or not.

46. Write a program that prompts the user for a number N and displays all prime numbers less than N.

47. Create a program that displays the first N prime numbers, where N is informed by the user, using a loop.

48. Create a program that displays the first N first perfect squares, where N is informed by the user, using a loop.

49. Write a program that prompts the user for two numbers A and B and displays all numbers between A and B.

50. Write a program that reads numbers from the user until a negative number is entered, and prints the sum of the positive numbers.

51. Write a program that prompts the user for a number and displays the Fibonacci sequence up to the given number using a repeating loop.

52. Write a program that reads numbers from the user until zero is entered, and displays the average of the numbers entered.

53. Write a program that prompts the user for a list of numbers, until the user types the number zero, and displays the largest and smallest numbers in the list.

54. Write a program that prompts the user for a sentence and displays the number of vowels in the sentence.

55. Write a program that prompts the user for a number and displays its divisors.

56. Write a program that determines the lowest common multiple (LCM) between two numbers entered by the user.

57. Write a program that determines the greatest common divisor (GCD) between two numbers entered by the user.

58. Write a program that calculates the series below up to the tenth element:

59. Rewrite the previous exercise code until the difference between the terms is less than 0.001.

60. Make a program that calculates the value of sine using the Taylor series according to the equation below until the difference between the terms is less than 0.0001.

61. Create a program that reads an array of integers and displays the sum of

all the elements..

62. Write a program that reads an array of integers and displays the largest element in the array.

63. Write a program that reads an array of integers and displays the average of the elements.

64. Create a program that reads two vectors of integers of the same size and displays a new vector with the sum of the corresponding elements of the two vectors.

65. Write a program that reads an array of integers and checks if they are in ascending order.

66. Write a program that reads an array of integers and displays the elements in reverse order.

67. Create a program that reads an array of integers and finds the second largest element in the array.

68. Write a program that reads an array of integers and displays how many times a specific number appears in the array.

69. Write a program that reads two arrays of integers with the same size and displays a new array with the elements resulting from the multiplication of the corresponding elements of the two arrays.

70. Create a program that reads an array of integers and checks that all elements are even.

71. Create a program that reads two words and concatenates them, displaying the resulting word.

72. Write a program that takes a word and displays each letter separately.

73. Create a program that takes a sentence and replaces all the letters "a" with "e".

74. Write a program that receives a name and checks that it starts with the letter "A".

75. Write a program that reads a word and checks if it is a palindrome (if it can be read backwards the same way).

76. Create a program that reads two words and checks if the second word is an anagram of the first.

77. Write a program that takes a full name and displays only the first name.

78. Make a program that receives a sentence and displays the amount of

blank spaces present in it.

79. Create a program that reads a word and displays the number of vowels present in it.

80. Write a program that takes a full name and displays the last name (last name) first.

81. Write a program that fills a 3x3 matrix with values entered by the user and displays the sum of the main diagonal values.

82. Write a program that fills a 4x4 matrix with random values and displays the transposed matrix.

83. Write a program that reads two 2x2 matrices and displays the sum of the two matrices.

84. Write a program that fills a 5x5 matrix with integers and displays the largest value in the matrix and its position.

85. Write a program that reads a 3x3 matrix and calculates the average of the values present in the even positions (sum of the even indices) of the matrix.

86. Write a program that fills a 4x4 matrix with random numbers and displays the sum of the values present in each row and in each column.

87. Write a program that reads a 3x3 matrix and calculates the determinant of the matrix.

88. Write a program that reads two matrices and returns the multiplication between them as an answer. The program should observe whether or not it is possible to perform the multiplication between the two matrices.

89. Write a program that reads a 4x4 matrix and checks if it is a diagonal matrix, that is, if all elements outside the main diagonal are equal to zero.

90. Write a program that reads an m x n matrix, indicating the location where there are mines in a Minesweeper game (being 0 for a neutral field, and 1 for locations where there would be mines), and the program should return a matrix indicating, for each position, the number of mines in neighboring houses.

91. Write a recursive function to calculate the factorial of a number.

92. Implement a recursive function to calculate the Fibonacci sequence up to a given number.

93. Create a recursive function to check if a number is prime.

94. Develop a recursive function to calculate the sum of the digits of an integer.
95. Write a recursive function to calculate the power of an integer raised to an exponent.
96. Implement a recursive function to find the greatest common divisor (GCD) of two numbers.
97. Create a recursive function to reverse a string.
98. Develop a recursive function to find the smallest value in an array.
99. Write a recursive function to determine whether a word is a palindrome.
100. Implement a recursive function to calculate the sum of elements of an array.
101. Write a program that prompts the user for a sentence and displays the sentence backwards
102. Make the BubbleSort Algorithm
103. Make an algorithm that solves the Tower of Hanoi
104. Make a function that receives a 3x3 matrix representing the game of tic-tac-toe, and check if there is a winner, if there is a tie, or if the game is not over yet.
105. Rawwords: Write an algorithm that checks whether a word is a "rawword". A word is considered a "prime word" if the sum of the letter values (where 'a' = 1, 'b' = 2, etc.) is a prime number.
106. Implement an algorithm that takes an integer and generates the Collatz sequence for that number. The Collatz sequence is generated by applying the following rules: if the number is even, divide it by 2; if the number is odd, multiply it by 3 and add 1. Repeat this process until you reach the number 1.

Additional Content

In case you want to access the code of all the exercises, you can get it from the link below:

https://forms.gle/ye5t9NduXL8y2VvK6

Each file is named with the exercise number, with the extension .py

About the Author

Ruhan Avila da Conceição (@ruhanconceicao, on social media) holds a degree in Computer Engineering (2015) from the Federal University of Pelotas and a Master's in Computing (2016) also from the same university. Since 2018 he has been a professor at the Federal Institute of Education, Science and Technology in the field of Informatics, where he teaches, among others, the disciplines of Object Oriented Programming, Visual Programming and Mobile Device Programming.

In 2014, even as an undergraduate student, Ruhan received the title of Researcher from Rio Grande do Sul in the Young Innovator category due to his academic and scientific career during his years at the university. His research topic has always been related to the algorithmic development of efficient solutions for encoding videos in real time. Still with regard to scientific research, Ruhan accumulates dozens of works published in national and international congresses, as well as articles in scientific journals of great relevance, and two computer programs registered at the National Institute of Intellectual Property.

Initiated in programming in the C language, Ruhan Conceição has extensive knowledge in JavaScript languages, as well as their libraries and frameworks ReactJS, React Native, NextJS; and Java. In addition, the author has also developed projects in Python, C#, Matlab, GoogleScript and C++.