LBD Loops

Write a code where the user will enter a number, and the program will print the multiplication table for that number.

- # Get user input for the number

  user_input = int(input('Enter a number: '))

  # Print the multiplication table

  print(f'Table of {user_input} is as follows:')

  for i in range(1, 11):

  result = user_input * i

  print(f'{user_input} x {i} = {result}')

Here, the program asks the user to enter a number. The entered value is then stored in a variable called user_input. The for loop starts from 1 but does not include 11. Therefore, i will take on values 1, 2, 3, ..., 10 in each iteration of the loop.

Now let's say the user enters 5, the program would print:

Table of 5 is as follows:

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

5 x 10 = 50

Now let’s say we want to print a multiplication table of the number that the user enters for the odd multiples from 1 to 10 i.e. print the table of the number multiplied by odd numbers (like 1, 3, 5, 7, 9) up to 10.

we will be using the same previous ex and modifying it:

- # Get user input for the number

  user_input = int(input('Enter a number: '))

  # Print the multiplication table for odd multiples from 1 to 10

- print(f'Table of {user_input} for odd multiples is as follows:')
- for i in range(1, 11, 2):
- result = user_input * i
- print(f'{user_input} x {i} = {result}')

This code is similar to the one provided earlier. The program focuses explicitly on odd multiples, which are the results obtained by multiplying the given number by odd numbers (such as 1, 3, 5, 7, 9) up to 10. This is achieved by using the range(start, stop, step) function.

Therefore, the for loop will iterate through numbers starting from 1, up to (but not including) 11, in steps of 2.

To break it down:

It starts at 1: The loop initiates from 1.

It ends at 11: The loop stops just before reaching 11.

It steps by 2: The loop increments by 2 in each iteration.

In essence, it effectively goes through odd numbers from 1 to 10 (1, 3, 5, 7, 9).

Let's consider the user enters the number 2. The output of the program would be:

Enter a number: 2

Table of 2 for odd multiples is as follows:

2 x 1 = 2

2 x 3 = 6

2 x 5 = 10

2 x 7 = 14

2 x 9 = 18

So, for the input 2, the program gives you the multiplication table for 2, but only for the odd multiples (1, 3, 5, 7, 9) up to 10.

Now let’s say we want to print the multiplication table in reverse.

- # Get user input for the number

  user_input = int(input('Enter a number: '))

  print(f'Reverse table of {user_input} is as follows:')

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

  result = user_input * i

  print(f'{user_input} x {i} = {result}')

In this case the loop is with the range(10, 0, -1), which

Start at 10: It means we begin counting from the number 10.

Stop at 0: We stop just before reaching 0.

Step by -1: We move backward by 1 in each step.

So if we iterate through this range:

First iteration: 10

Second iteration: 9

Third iteration: 8

... and so on, until it stops just before 0.

Let’s say the user enters 10, the output for the provided code would be:

Enter a number: 10

The reverse table of 10 is as follows:

10 x 10 = 100

10 x 9 = 90

10 x 8 = 80

10 x 7 = 70

10 x 6 = 60

10 x 5 = 50

10 x 4 = 40

10 x 3 = 30

10 x 2 = 20

10 x 1 = 10

let’s calculate the sum of numbers from 0 to 10. Let’s write the code to do the same

- a=0

  for i in range(0,11):

  a+=i

  print(a)

In this code, initially the variable ‘a’ is set to 0. The for loop is initiated then that goes through values of i starting from 0 up to, but not including, 11.

a += i means "add the current value of i to the current value of a."

So, in each iteration of the loop, the value of i is added to the current total in a. After the loop completes, the code prints the final value of a using print(a).

The output of the code is :

55

Which is the sum of all the numbers from 0 to 10.

Now let’s modify the code and see the sum of numbers from 0 to 5 and see how the values of i and ‘a’ change at each iteration:

- a = 0

  prev_a = 0

  for i in range(0, 6):

  prev_a = a # Store the previous value of a

  a += i

  print(f'current value of i: {i} and current value of a: {a} which is {prev_a}+{i}')

  print(f’Sum first 5 numbers is: {a}’)

Here, a new variable, prev_a, is introduced, and both the variables a and prev_a are initialized to 0.

The for loop runs for values of i from 0 to 5.

prev_a = a stores the current value of a before it is updated in the next line.

The next line a += i updates the value of a by adding the current value of i to it.

The print statement outputs the current values of i and a along with the calculation that led to the new value of a.

The loop repeats this process for each iteration, showing how both i and ‘a’ change at each step.

By storing the previous value of ‘a’ in prev_a before updating a, we ensure that prev_a retains the value of a from the previous iteration and does not reflect the immediate update. It allows us to keep track of how ‘a’ changes from iteration to iteration, showing both the current and previous values in the print statements.

After the loop completes, the final value of a is printed as follows:

current value of i: 0 and current value of a: 0, which is 0+0

current value of i: 1 and current value of a: 1, which is 0+1

current value of i: 2 and current value of a: 3, which is 1+2

current value of i: 3 and current value of a: 6, which is 3+3

current value of i: 4 and current value of a: 10, which is 6+4

current value of i: 5 and current value of a: 15, which is 10+5

Sum first 5 numbers is: 15

Now let’s write the code that prompts the user to enter a value and then calculates the sum of numbers up to that value:

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

  sum_result = 0

  # Calculate the sum of numbers up to the user-entered value

  for i in range(0, user_input + 1):

  sum_result += i

  print(f'Sum of numbers from 0 to {user_input} is: {sum_result}')

This code is similar to the codes discussed previously, the only difference is that instead of initializing the value of the number the user is asked to enter a number.

range(0, user_input + 1) is used in the code to generate a sequence of numbers starting from 0 up to and including the user_input. + 1 is important to ensure that the loop includes the number entered by the user (user_input).

Continuing on let’s see a simple example in Python that demonstrates how to print every element of a list:

- my_list = ['Apple','Mango','Kiwi','Watermelon','Cherry']

  # Using a for loop to print each element

  for i in my_list:

```
print(i)
```

In the code, a list named my_list is created, containing strings representing different fruits like 'Apple', 'Mango', 'Kiwi', 'Watermelon', and 'Cherry'.

The for loop is then used to go through each element in my_list. The loop variable i takes on the value of each fruit during each iteration. Inside the loop, the print(i) statement is used to display each fruit on a new line.

The output of the code will be:

Apple

Mango

Kiwi

Watermelon

Cherry

Now let’s write a program to print key-value pairs of a dictionary.

- ```
  student_marks = {'Deepti': 90, 'Arpita': 85, 'Mansi': 78, 'Rishabh': 92}

  for i in student_marks:

  print("Student:", i, "Marks:", student_marks[i])
  ```

Here student_marks is a dictionary where each key-value pair represents a student's name as key and their corresponding marks as value.

Next is the for loop that iterates over the keys of the student_marks dictionary. In each iteration, ‘i’ takes on the value of a student's name, which is the key.

The print statement prints the current student's name (i) where i represents the current key (student's name) in the iteration and their marks (student_marks[i]).
  ```

student_marks[i] retrieves the corresponding value (marks) associated with the current student's name.

After running the code the output will be:

Student: Deepti Marks: 90

Student: Arpita Marks: 85

Student: Mansi Marks: 78

Student: Rishabh Marks: 92

Now as discussed in the previous chapters in dictionaries, you can access the value associated with a specific key using square brackets []. Here's an example:

- deepti_marks = student_marks['Deepti']

  print("Deepti's Marks:", deepti_marks)

In this example, student_marks['Deepti'] retrieves the value associated with the key 'Deepti' in the dictionary student_marks. The value (marks) is then assigned to the variable deepti_marks and printed.

Moving on let’s print a sequence of negative integers starting from -1 and counting down to -5.

- for i in range(-1,-6,-1):

  print(i)

  range(-1, -6, -1) creates a sequence of numbers starting from -1, decreasing by 1 each time, and stopping just before reaching -6.

Now let’s learn how to reverse the list. But before we dive into the code, let's take a moment to understand a handy tool we'll be using: the len() function.

len() function in Python is used to find the length or the number of elements in a sequence, such as a string, list, or tuple. It simply returns the count of items in the given sequence.

For example let’s say we want to find how many character are their in the string=’Hello’

So this is how we’ll find that:

- my_string = "Hello"

  length_of_string = len(my_string)

  print(length_of_string)

  So the output will be 5, which is true, as there are 5 characters in the string "Hello".

Now let's get back onto how we can reverse a list.

- my_list = ['Apple', 'Mango', 'Kiwi', 'Watermelon', 'Cherry']

  # Using a for loop to print the list in reverse order

  for i in range(len(my_list) - 1, -1, -1):

  print(my_list[i])

my_list: is the list of fruits containing 'Apple', 'Mango', 'Kiwi', 'Watermelon', and 'Cherry'. Each fruit is an element in the list.

After this we're using a for loop to go through the list. len(my_list) tells us the number of fruits in the basket. Subtracting 1 (len(my_list) - 1) helps us start from the last fruit because in programming, we often begin counting from 0. So, if there are 5 fruits, the indices are 0, 1, 2, 3, 4. The loop then goes backward (-1 step) until reaching -1.

print(my_list[i]) line prints each fruit in reverse order. my_list[i] refers to the fruit at the current position of the loop.

So the output is :

Cherry

Watermelon

Kiwi

Mango

Apple

Let’s achieve the same result but with a while loop.

```
my_list = ['Apple', 'Mango', 'Kiwi', 'Watermelon', 'Cherry']

i = len(my_list) - 1 # Start from the last index

while i >= 0:

print(my_list[i])

i -= 1
```

We start by initializing the variable i with the value of the last index of the list (len(my_list) - 1). In Python, indices start from 0, so the last index is one less than the length of the list.

The while loop continues to execute as long as the value of i is greater than or equal to 0. This ensures that we iterate through the list from the last element to the first. Inside the loop, we print the element at the current index i. Since i is initially set to the last index, it starts by printing the last element of the list. i -= 1 ensures that In each iteration of the loop, the value of i is decreased by 1. This ensures that we move backward through the list, printing each element in reverse order.

Output:

```
Cherry
Watermelon
Kiwi
Mango
Apple
```

Moving on to a different scenario instead of printing the entire list suppose we want the user to input a number, which represents the count of fruits they want to print from the list.

Let’s see the code on how to do that:

- ```
  my_list = ['Apple','Mango','Kiwi','Watermelon','Cherry']

  n=int(input('Enter the number of fruits to print from the list: '))

  i=0
  ```
  ```

while i<= n-1:

print(my_list[i])

i+=1

In this code, the user is prompted to input a number, which represents the count of fruits they want to print from the list and then the value is stored in the variable n.

Initially, i is initialized to 0. The while loop then continues to execute as long as the value of i is less than or equal to n-1. This ensures that we print the first ‘n’ fruits from the list.

Element at the current index ‘i’ in the list is then printed. At last, the value of i is incremented by 1.

Now, let's imagine a user enters the value 3 when prompted:

The code will then print the first 3 fruits from the list:

```
Enter the number of fruits to print from the list: 3
Apple
Mango
Kiwi
```

Let’s consider another example where the program first checks if a user-entered number is present in a list and then determines whether it's even or not:

- number_list = [2, 5, 8, 12, 15, 20, 25]

  # Ask the user to enter a number

  user_number = int(input("Enter a number to check: "))

  # Check if the entered number is present in the list

  if user_number in number_list:

  print(f"{user_number} is present in the list.")

  # Check if the entered number is even or odd

  if user_number % 2 == 0:

print(f"{user_number} is an even number.")

else:

print(f"{user_number} is an odd number.")

else:

print(f"{user_number} is not present in the list.")

The outer if block checks whether the user_number is present in the number_list. If this condition is true (i.e., the user's number is in the list), it enters the block of code under this if statement.

The second if block which is if user_number % 2 == 0: checks whether the user_number is even by using the modulus operator (%). If the remainder is 0, it means the number is even.

So If the user's number is even, it enters the block under this inner if statement and prints a message saying that the number is an even number.

If the user's number is not even (i.e., it's odd), it goes to the else block under this inner if statement and prints a message saying that the number is an odd number.

If the user's number is not in the list (i.e., the condition of the outer if statement is false), it goes to the outer else block and prints a message saying that the number is not in the list.

Let’s say we enter 15 then the output will be:

```
Enter a number to check: 15
15 is present in the list.
15 is an odd number.
```

Let’s see another code that iterates through a list of colors and prints each color along with its individual characters.

- Colors = ["Red", "Blue", "Green", "Yellow", "Orange"]

  for color in Colors:

  print(color)

```
    for char in color:

    print(char)
```

Understand the code:

We have a list called Colors containing five strings: "Red", "Blue", "Green", "Yellow", and "Orange".

The outer loop iterates over each color in the list Colors. For each iteration, the current color is assigned to the variable color.

Inside the outer loop, the code prints the name of the current color using print(color).

After printing the color name, there is an inner loop that iterates over each character in the current color's name (stored in the variable color).

For example, if the current color is "Red," the inner loop will iterate over the characters "R", "e", "d".

Inside the inner loop, the code prints each character (char) of the current color's name.

The outer loop then moves on to the next color in the list, and the process is repeated.

Here's the flow in a nutshell:

Iteration 1:

color is assigned "Red".

Prints "Red".

Inner loop prints "R", "e", "d".

Iteration 2:

color is assigned "Blue".

Prints "Blue".

Inner loop prints "B", "l", "u", "e".

Iteration 3:

color is assigned "Green".

Prints "Green".

Inner loop prints "G", "r", "e", "e", "n".

Iteration 4:

color is assigned "Yellow".

Prints "Yellow".

Inner loop prints "Y", "e", "l", "l", "o", "w".

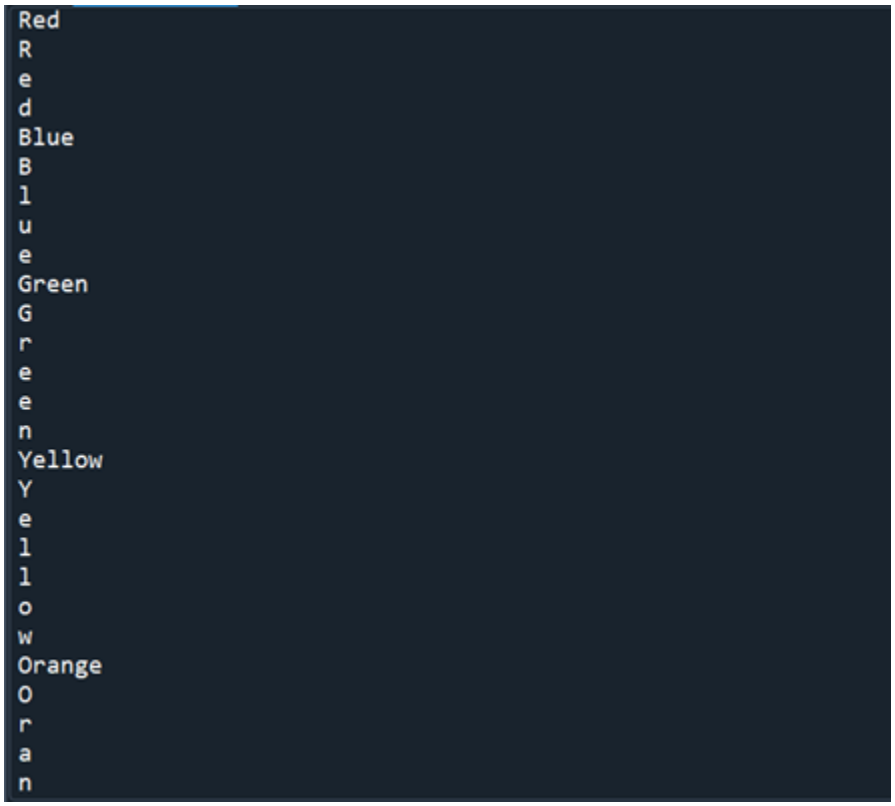
Iteration 5:

color is assigned "Orange".

Prints "Orange".

Inner loop prints "O", "r", "a", "n", "g", "e".

Now let’s see the output of this code:



Let’s consider another example suppose we want to check the performance of students in a class and assign grades based on their scores: Let’s write the code for it.

- # List of student scores

```
student_scores = [85, 92, 78, 65, 95, 88, 75, 60, 72, 98]

# For loop to iterate through each student's score

for score in student_scores:

print(f"Student score: {score}")

# Grading based on the score

if score >= 90:

print("Grade: A")

elif 80 <= score < 90:

print("Grade: B")

elif 70 <= score < 80:

print("Grade: C")

elif 60 <= score < 70:

print("Grade: D")

else:

print("Grade: F")

print("----------") # Separator for better readability
```

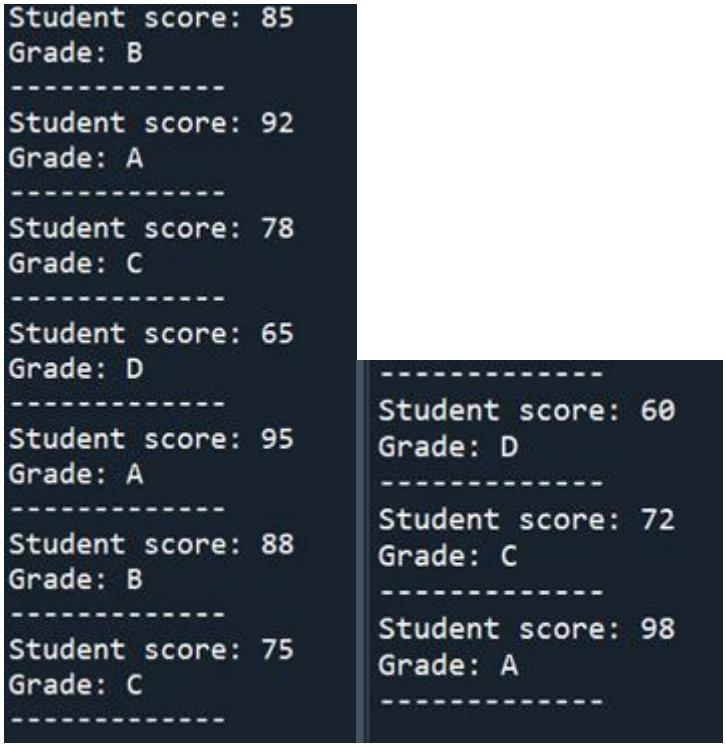
In this example:

The for loop iterates through each student's score in the student_scores list.

Inside the loop, there's an if-elif-else statement block to determine the grade based on the student's score.

Depending on the score, it prints the student's score and the corresponding grade.

Let’s see the output:



Now, let's write a program that demonstrates a simple login system where the user is prompted to enter a username and password. The correct username is set as "user123" and the correct password is set as "pass123". The user has a maximum of three attempts to enter the correct credentials. If successful, a "Login successful!" message is displayed, and if the maximum attempts are reached, the program informs the user that the account is locked. Let’s get started

correct_username = "user123"

correct_password = "pass123"

attempts = 0

max_attempts = 3

while attempts < max_attempts:

username = input("Enter your username: ")

password = input("Enter your password: ")

if username == correct_username and password == correct_password:

print("Login successful!")

break

else:

print("Invalid username or password. Please try again.")

attempts += 1

if attempts == max_attempts:

print("Sorry, you've reached the maximum number of login attempts. Account locked.")

The correct username is set as "user123", and the correct password is set as "pass123".

attempts is initialized to 0, representing the number of login attempts made by the user.

max_attempts is set to 3, indicating the maximum allowed login attempts.

While Loop (while attempts < max_attempts):

The program enters a while loop, which continues as long as the number of attempts is less than the maximum allowed attempts.

User Input:

Inside the loop, the user is prompted to enter a username and password using the input function.

In (if username == correct_username and password == correct_password):

The program checks if the entered username and password match the correct values.

If the login is successful, the program prints "Login successful!" and breaks out of the loop using break.

If Invalid Credentials are entered, it enters the (else block):

The program prints "Invalid username or password. Please try again."

The number of attempts is then incremented by 1 after each login attempt (attempts += 1).

The loop continues until either the correct credentials are entered or the maximum allowed attempts are reached.

Maximum Attempts Check (if attempts == max_attempts):

If the user can't log in within the allowed attempts, the program prints "Sorry, you've reached the maximum number of login attempts. Account locked."

Output:

Both credentials should be right :

```
Enter your username: 123user
Enter your password: pass123
Invalid username or password. Please try again.
```

The code snippet is case sensitive:

```
Enter your username: User123
Enter your password: pass123
Invalid username or password. Please try again.
```

The Right output:

```
Enter your username: user123
Enter your password: pass123
Login successful!
```

Try it yourself, you can add print(attempts) after attempts += 1 line to check the change in the value of the attempt whenever the wrong value is entered.

Moving on let’s print the smallest number in the list. Let’s see the code:

- my_list = [5, 3, 8,0]

  # Initialize the variable to store the smallest value

  smallest = my_list[0]

  # Loop through the list

  for num in my_list:

  # Check if the current number is smaller than the current smallest

```
    if num < smallest:

    smallest = num

    # Print the smallest value

    print("The smallest value in the list is:", smallest)
```

Here, a list named my_list is initialized with four integer values: 5, 3, 8, and 0.

The variable smallest is initialized with the first element of the list (my_list[0]). In this case, the smallest is initially set to 5.

This line starts a for loop that iterates through each element (num) in the my_list.

Within the loop, this if statement checks if the current value (num) is smaller than the current smallest value (smallest).

If the condition is true, meaning that the current num is smaller than the current smallest, the smallest variable is updated to the value of num.

Let's go through the loop iterations with the given list [5, 3, 8, 0]:

Iteration 1: num=5, smallest=5 (no change)

Iteration 2: num=3, smallest=3 (update smallest to 3)

Iteration 3: num=8, smallest=3 (no change)

Iteration 4: num=0, smallest=0 (update smallest to 0)

Finally, the code prints the smallest value found after iterating through the entire list. So the output will be: "The smallest value in the list is: 0".

```
The smallest value in the list is: 0
```

Modify the code and try to print the greatest number in a list.