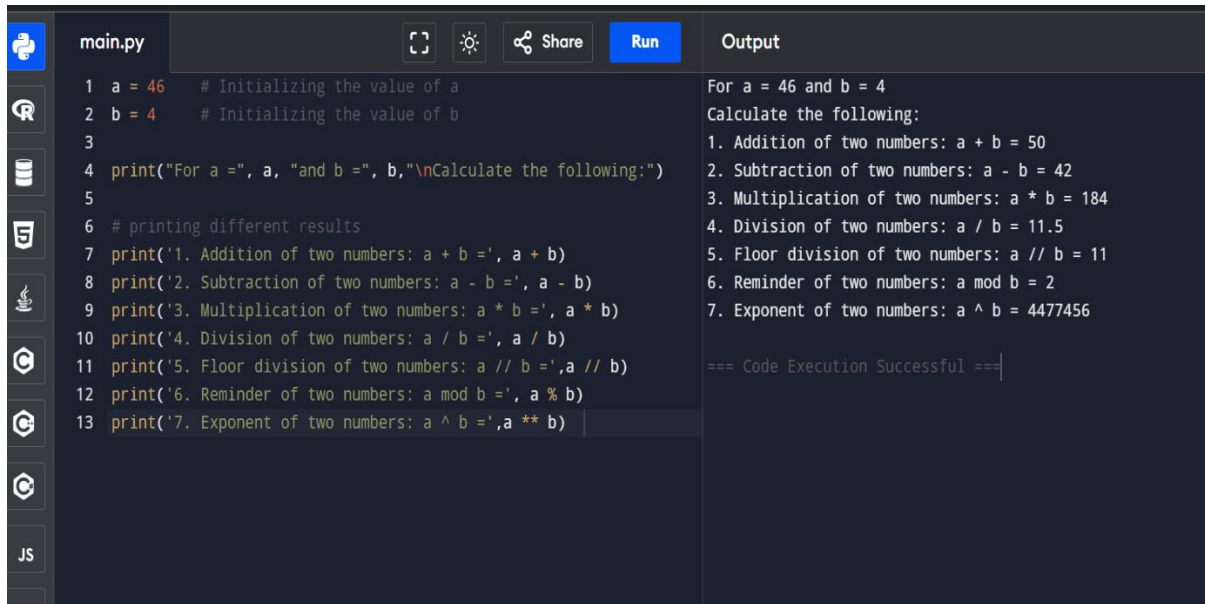


PYTHON PROGRAMS ASSIGNMENT 1

ARITHMETIC OPERATORS



The screenshot shows a Python IDE with a file named `main.py`. The code initializes `a = 46` and `b = 4`, then prints the results of seven arithmetic operations. The output window displays the results of these operations.

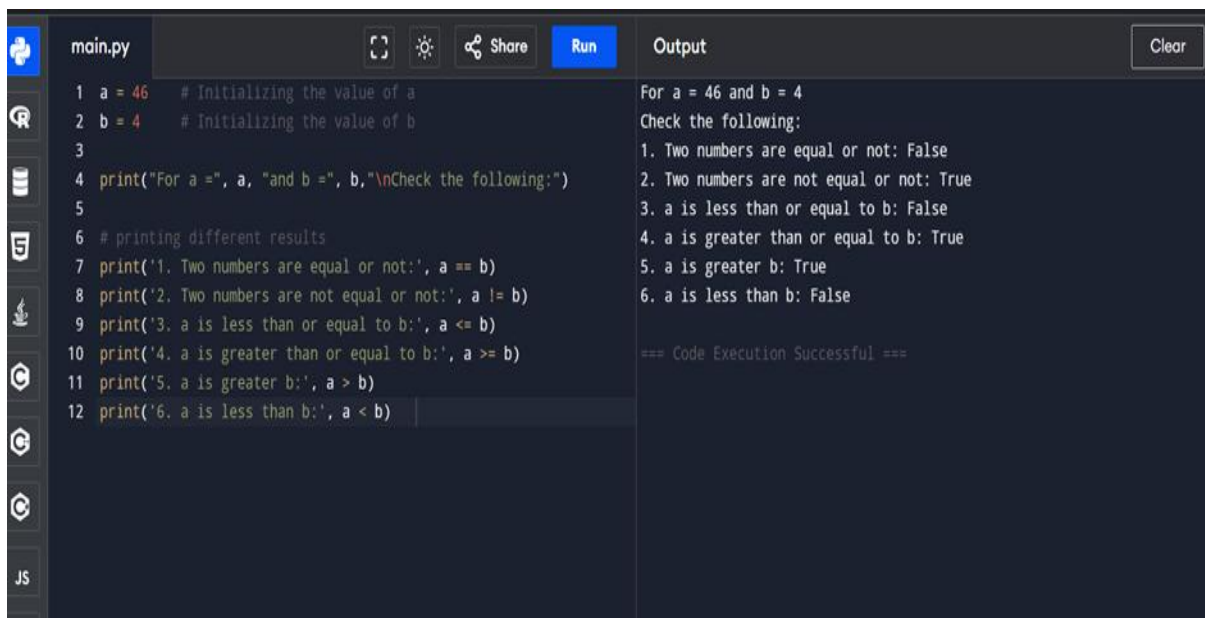
```
1 a = 46 # Initializing the value of a
2 b = 4  # Initializing the value of b
3
4 print("For a =", a, "and b =", b, "\nCalculate the following:")
5
6 # printing different results
7 print('1. Addition of two numbers: a + b =', a + b)
8 print('2. Subtraction of two numbers: a - b =', a - b)
9 print('3. Multiplication of two numbers: a * b =', a * b)
10 print('4. Division of two numbers: a / b =', a / b)
11 print('5. Floor division of two numbers: a // b =', a // b)
12 print('6. Reminder of two numbers: a mod b =', a % b)
13 print('7. Exponent of two numbers: a ^ b =', a ** b)
```

Output:

```
For a = 46 and b = 4
Calculate the following:
1. Addition of two numbers: a + b = 50
2. Subtraction of two numbers: a - b = 42
3. Multiplication of two numbers: a * b = 184
4. Division of two numbers: a / b = 11.5
5. Floor division of two numbers: a // b = 11
6. Reminder of two numbers: a mod b = 2
7. Exponent of two numbers: a ^ b = 4477456

=== Code Execution Successful ===
```

COMPARISON OPERATORS



The screenshot shows a Python IDE with a file named `main.py`. The code initializes `a = 46` and `b = 4`, then prints the results of six comparison operations. The output window displays the results of these operations.

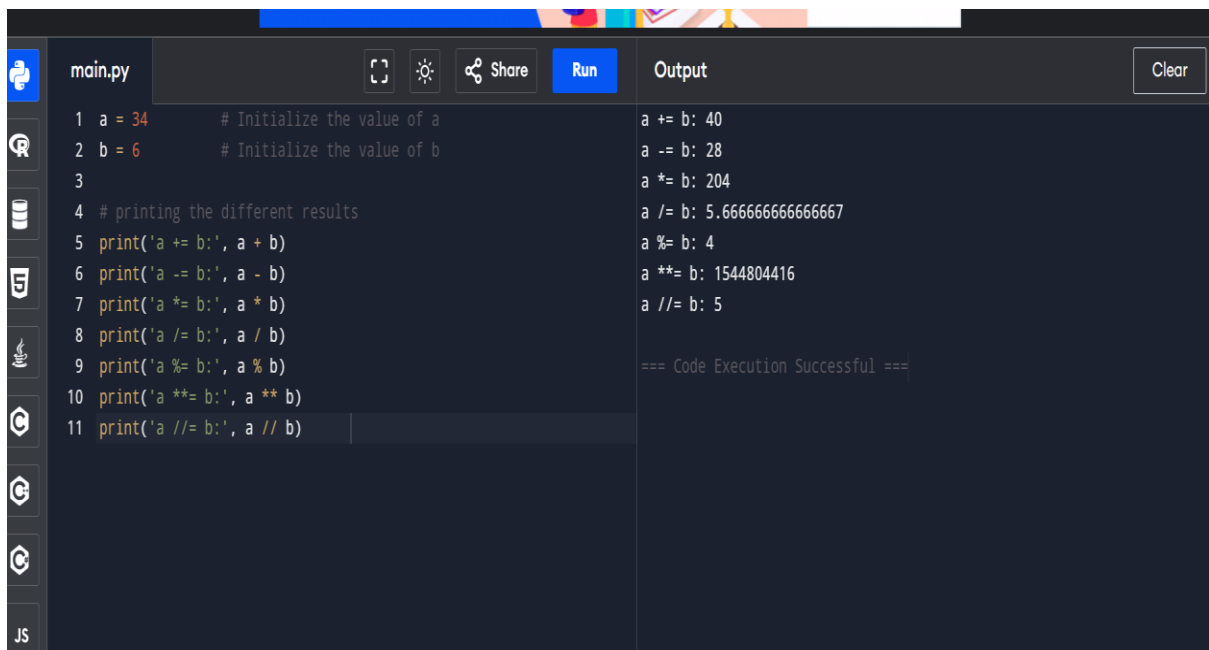
```
1 a = 46 # Initializing the value of a
2 b = 4  # Initializing the value of b
3
4 print("For a =", a, "and b =", b, "\nCheck the following:")
5
6 # printing different results
7 print('1. Two numbers are equal or not:', a == b)
8 print('2. Two numbers are not equal or not:', a != b)
9 print('3. a is less than or equal to b:', a <= b)
10 print('4. a is greater than or equal to b:', a >= b)
11 print('5. a is greater b:', a > b)
12 print('6. a is less than b:', a < b)
```

Output:

```
For a = 46 and b = 4
Check the following:
1. Two numbers are equal or not: False
2. Two numbers are not equal or not: True
3. a is less than or equal to b: False
4. a is greater than or equal to b: True
5. a is greater b: True
6. a is less than b: False

=== Code Execution Successful ===
```

ASSIGNMENT OPERATORS



The screenshot shows a Python IDE with a file named 'main.py'. The code in the editor is as follows:

```
1 a = 34      # Initialize the value of a
2 b = 6       # Initialize the value of b
3
4 # printing the different results
5 print('a += b:', a + b)
6 print('a -= b:', a - b)
7 print('a *= b:', a * b)
8 print('a /= b:', a / b)
9 print('a %= b:', a % b)
10 print('a **= b:', a ** b)
11 print('a //= b:', a // b)
```

The output window on the right displays the results of these operations:

```
a += b: 40
a -= b: 28
a *= b: 204
a /= b: 5.666666666666667
a %= b: 4
a **= b: 1544804416
a //= b: 5

=== Code Execution Successful ===
```

BITWISE OPERATORS



The screenshot shows a Python IDE with a file named 'main.py'. The code in the editor is as follows:

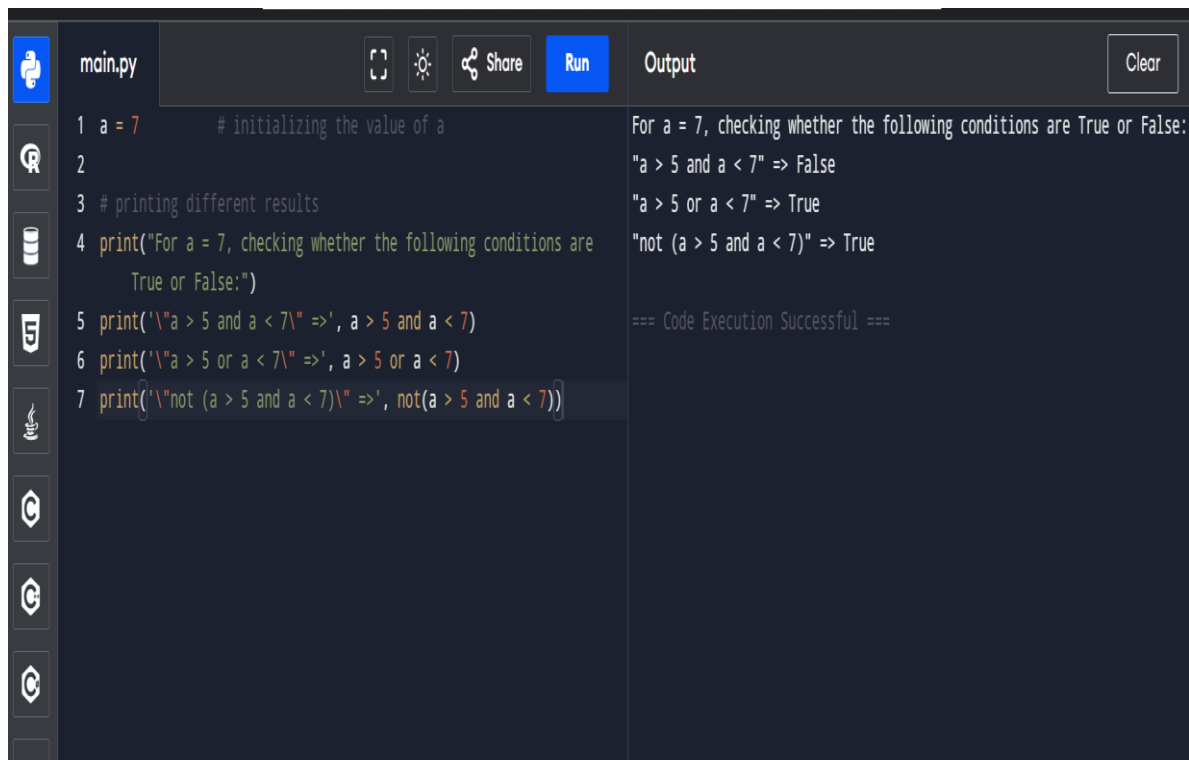
```
1 a = 7       # initializing the value of a
2 b = 8       # initializing the value of b
3
4 # printing different results
5 print('a & b:', a & b)
6 print('a | b:', a | b)
7 print('a ^ b:', a ^ b)
8 print('~a:', ~a)
9 print('a << b:', a << b)
10 print('a >> b:', a >> b)
```

The output window on the right displays the results of these bitwise operations:

```
a & b : 0
a | b : 15
a ^ b : 15
~a : -8
a << b : 1792
a >> b : 0

=== Code Execution Successful ===
```

Logical Operators



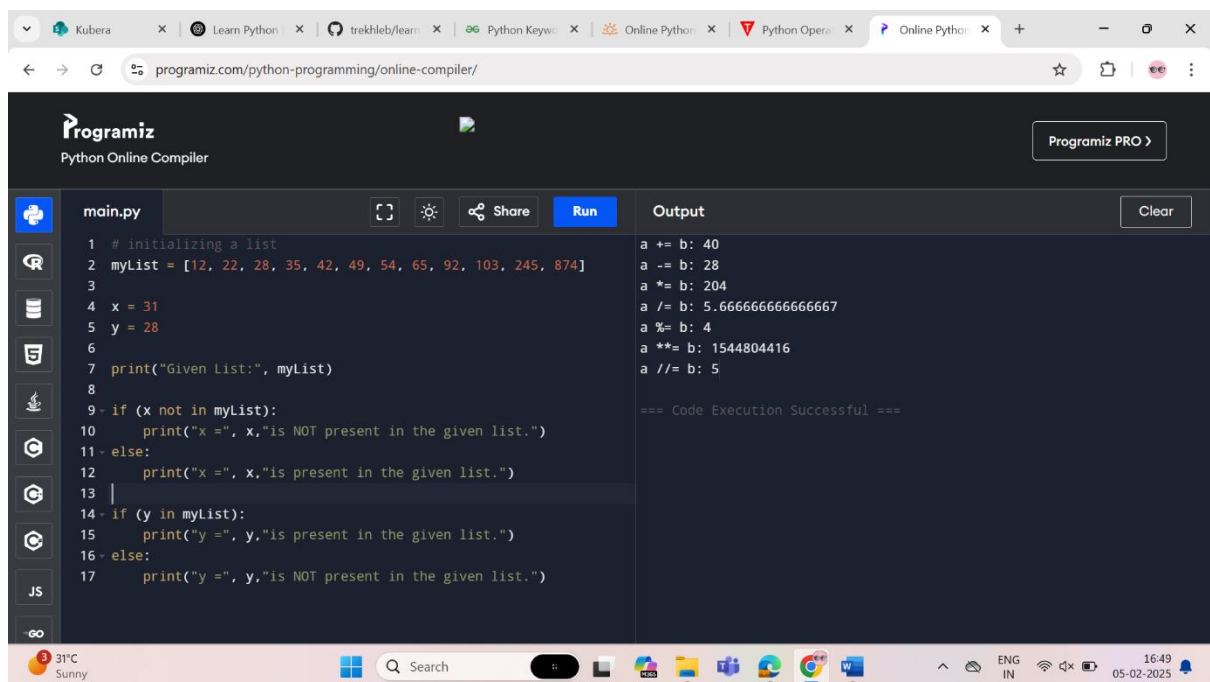
```
main.py
1 a = 7      # initializing the value of a
2
3 # printing different results
4 print("For a = 7, checking whether the following conditions are
    True or False:")
5 print('\na > 5 and a < 7" =>', a > 5 and a < 7)
6 print('\na > 5 or a < 7" =>', a > 5 or a < 7)
7 print('\not (a > 5 and a < 7)" =>', not(a > 5 and a < 7))
```

Output

```
For a = 7, checking whether the following conditions are True or False:
"a > 5 and a < 7" => False
"a > 5 or a < 7" => True
"not (a > 5 and a < 7)" => True

=== Code Execution Successful ===
```

MEMBERSHIP OPERATORS



```
Programiz
Python Online Compiler
Programiz PRO >

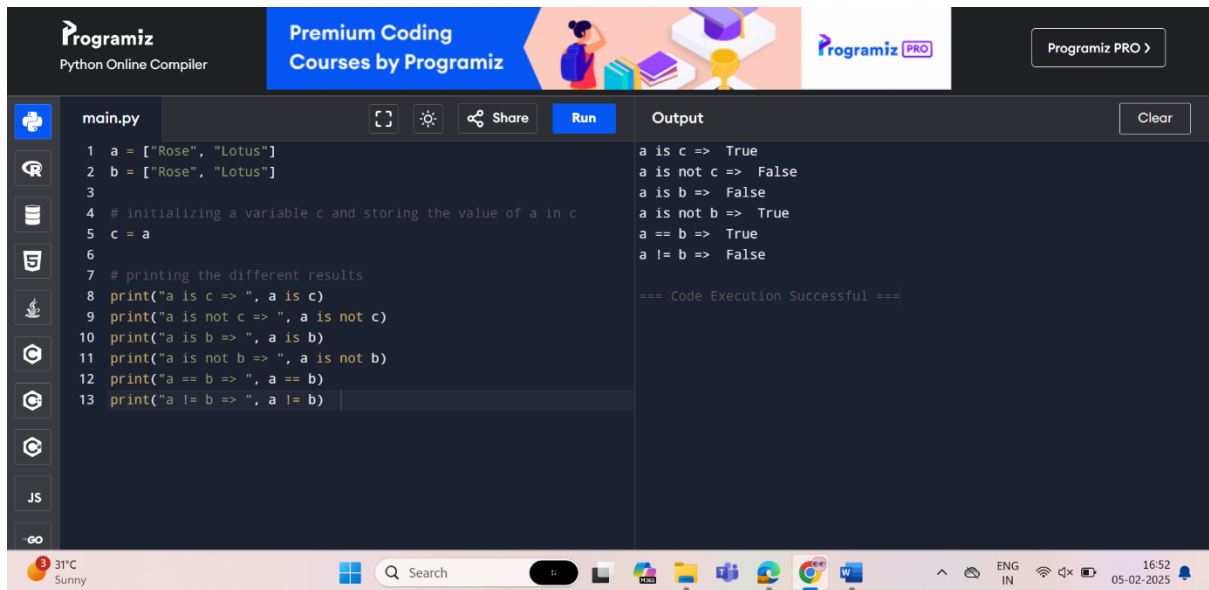
main.py
1 # initializing a list
2 myList = [12, 22, 28, 35, 42, 49, 54, 65, 92, 103, 245, 874]
3
4 x = 31
5 y = 28
6
7 print("Given List:", myList)
8
9 if (x not in myList):
10     print("x =", x, "is NOT present in the given list.")
11 else:
12     print("x =", x, "is present in the given list.")
13
14 if (y in myList):
15     print("y =", y, "is present in the given list.")
16 else:
17     print("y =", y, "is NOT present in the given list.")
```

Output

```
a += b: 40
a -= b: 28
a *= b: 204
a /= b: 5.666666666666667
a %= b: 4
a **= b: 1544804416
a //= b: 5

=== Code Execution Successful ===
```

IDENTITY OPERATOR



The screenshot shows the Programiz Python Online Compiler interface. The editor contains a Python script named `main.py` that demonstrates the identity operator (`is`) and the non-identity operator (`is not`). The script initializes two lists, `a` and `b`, both containing `["Rose", "Lotus"]`. It then assigns `c = a`. The script prints the results of several identity and non-identity comparisons. The output panel shows the results of these comparisons, confirming that `a is c` is `True`, `a is not c` is `False`, `a is b` is `False`, `a is not b` is `True`, `a == b` is `True`, and `a != b` is `False`. The output also indicates that the code execution was successful.

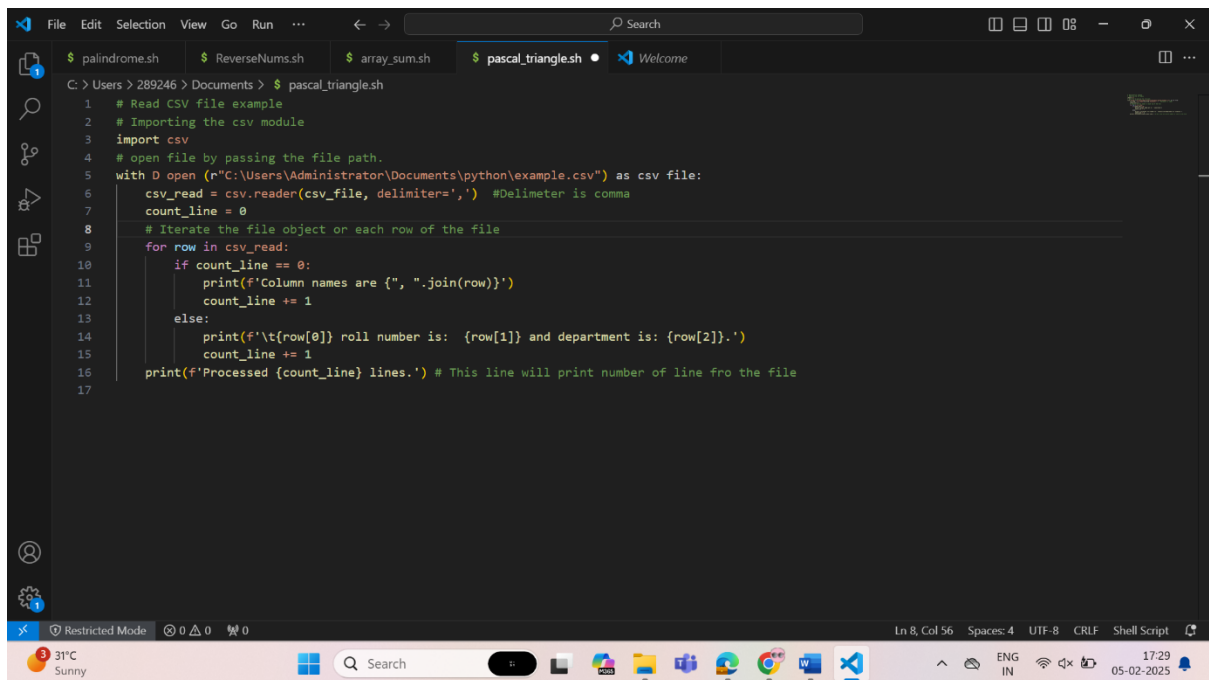
```
1 a = ["Rose", "Lotus"]
2 b = ["Rose", "Lotus"]
3
4 # initializing a variable c and storing the value of a in c
5 c = a
6
7 # printing the different results
8 print("a is c => ", a is c)
9 print("a is not c => ", a is not c)
10 print("a is b => ", a is b)
11 print("a is not b => ", a is not b)
12 print("a == b => ", a == b)
13 print("a != b => ", a != b)
```

Output:

```
a is c => True
a is not c => False
a is b => False
a is not b => True
a == b => True
a != b => False

=== Code Execution Successful ===
```

How to read CSV file in Python



The screenshot shows a Windows terminal window with a Python script for reading a CSV file. The script is named `pascal_triangle.sh` and is located in the `C:\Users\289246\Documents` directory. The script uses the `csv` module to read a CSV file named `example.csv`. It iterates over each row of the file and prints the column names and the roll number and department for each row. The script also prints the total number of lines processed.

```
1 # Read CSV file example
2 # Importing the csv module
3 import csv
4 # open file by passing the file path.
5 with open(r"C:\Users\Administrator\Documents\python\example.csv") as csv_file:
6     csv_read = csv.reader(csv_file, delimiter=',') #Delimiter is comma
7     count_line = 0
8     # Iterate the file object or each row of the file
9     for row in csv_read:
10         if count_line == 0:
11             print(f'Column names are {", ".join(row)}')
12             count_line += 1
13         else:
14             print(f'\t{row[0]} roll number is: {row[1]} and department is: {row[2]}')
15             count_line += 1
16     print(f'Processed {count_line} lines.') # This line will print number of line fro the file
17
```

OUTPUT:-

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Files\Python313\python.exe' 'c:\Users\Administrator\.vscode\extensions\ms-python.debugpy-2024.14.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher' '59728' '--' 'c:\Users\Administrator\recipewebsite\import_csv_module.py'
Column names are Name, Roll Number, Department
  Alice roll number is: 101 and department is: Computer Science.
  Bob roll number is: 102 and department is: Mechanical.
  Charlie roll number is: 103 and department is: Electrical.
  David roll number is: 104 and department is: Civil.
  Emma roll number is: 105 and department is: Electronics.
Processed 6 lines.
PS C:\Users\Administrator\recipewebsite>
```

REVERSE A STRING PYTHON

USING FOR LOOP


```
main.py  [Icons]  Run  Output  Clear

1 def reverse_string(str):
2     str1 = ""
3     for i in str:
4         str1 = i + str1
5     return str1
6
7 str = "JavaTpoint"
8 print("The original string is: ",str)
9 print("The reverse string is",reverse_string(str))




The original string is:  JavaTpoint
The reverse string is tniopTavaJ

=== Code Execution Successful ===
```

USING WHILE LOOP



main.py

 Share

Run

Output

Clear

```
1 str = "JavaTpoint" # string variable
2 print ("The original string is :",str)
3 reverse_String = "" # Empty String
4 count = len(str) # Find length of a string and save in count
   variable
5 - while count > 0:
6     reverse_String += str[ count - 1 ] # save the value of
       str[count-1] in reverseString
7     count = count - 1 # decrement index
8 print ("The reversed string using a while loop is : "
       ,reverse_String)# reversed string
```



The original string is : JavaTpoint
The reversed string using a while loop is : tniopTavaJ

=== Code Execution Successful ===

USING SLICE OPERATOR



main.py

 Share

Run

Output

Clear

```
1- def reverse(str):
2     str = str[::-1]
3     return str
4
5 s = "JavaTpoint"
6 print ("The original string is :",s)
7 print ("The reversed string using extended slice operator is : "
        ,reverse(s))
```

The original string is : JavaTpoint
The reversed string using extended slice operator is : tniopTavaJ

=== Code Execution Successful ===

Using reverse function with join

main.py

Share

Run

Output

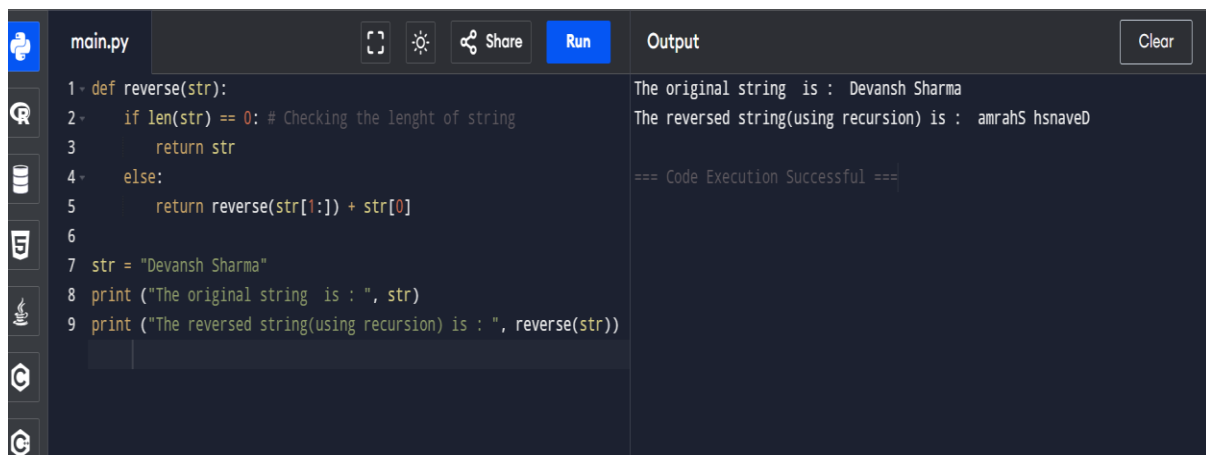
Clear

```
1- def reverse(str):
2     string = "".join(reversed(str)) # reversed() function inside
        the join() function
3     return string
4
5 s = "JavaTpoint"
6
7 print ("The original string is :",s)
8 print ("The reversed string using reversed() is :",reverse(s) )
```

The original string is : JavaTpoint
The reversed string using reversed() is : tniopTavaJ

=== Code Execution Successful ===

Using recursion()



The screenshot shows a Python IDE with a file named 'main.py'. The code defines a recursive function 'reverse(str)' that checks if the string length is 0. If so, it returns the string. Otherwise, it returns the string reversed by concatenating the last character with the reverse of the substring from the first character to the second-to-last. The string 'Devansh Sharma' is assigned to 'str', and two print statements are used to display the original and reversed strings. The output pane shows the execution results.

```
1 def reverse(str):
2     if len(str) == 0: # Checking the length of string
3         return str
4     else:
5         return reverse(str[1:]) + str[0]
6
7 str = "Devansh Sharma"
8 print ("The original string is : ", str)
9 print ("The reversed string(using recursion) is : ", reverse(str))
```

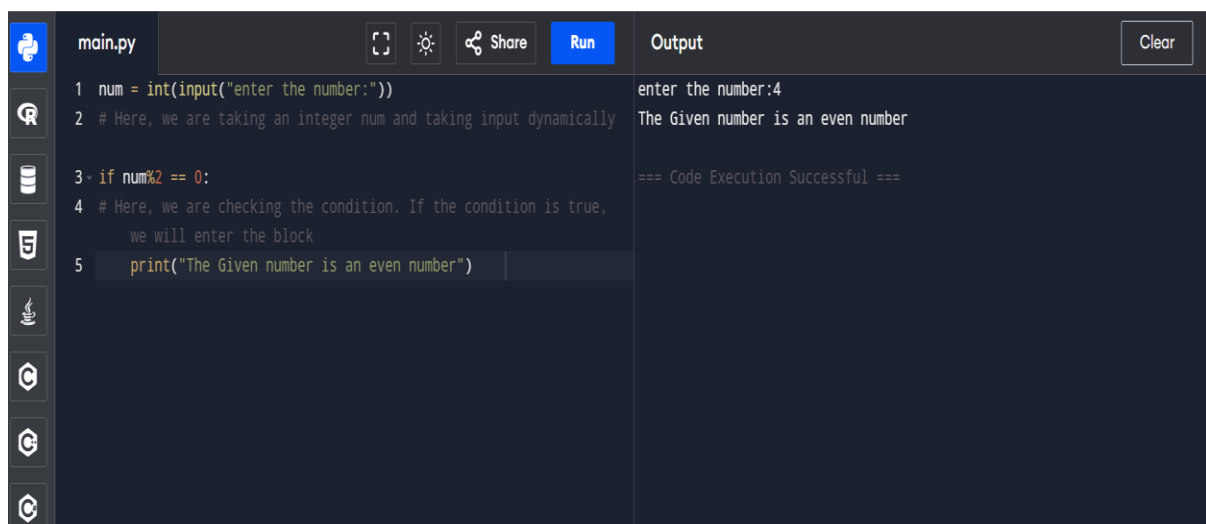
Output

```
The original string is : Devansh Sharma
The reversed string(using recursion) is : amrahS hsnaveD

=== Code Execution Successful ===
```

PYTHON IF ELSE STATEMENTS

Simple Python program to understand the if statement



The screenshot shows a Python IDE with a file named 'main.py'. The code takes an integer input from the user, checks if it is even (num % 2 == 0), and prints a message if the condition is true. The output pane shows the execution results.

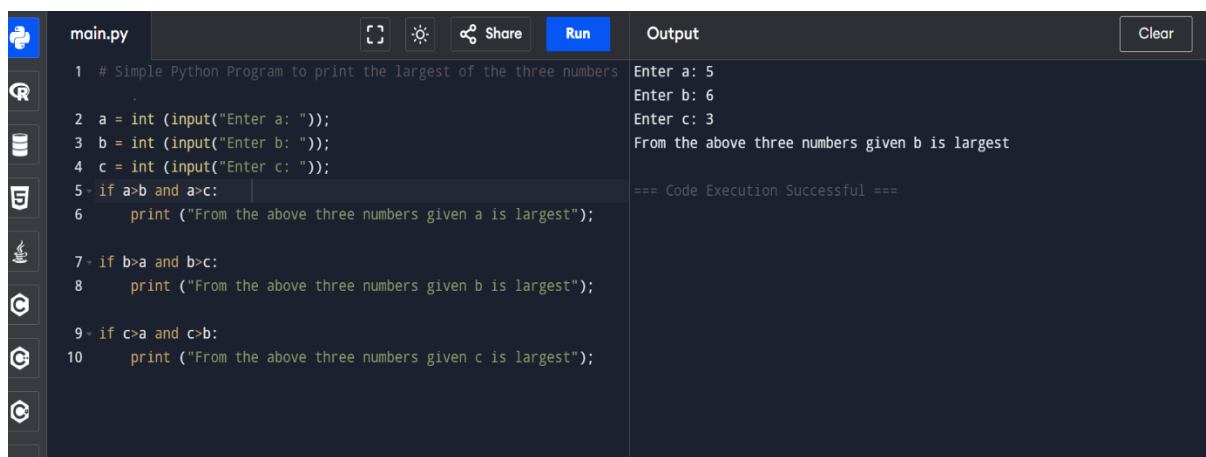
```
1 num = int(input("enter the number:"))
2 # Here, we are taking an integer num and taking input dynamically
3
4 if num%2 == 0:
5     # Here, we are checking the condition. If the condition is true,
6     # we will enter the block
7     print("The Given number is an even number")
```

Output

```
enter the number:4
The Given number is an even number

=== Code Execution Successful ===
```

Program to print the largest of the three numbers.



The screenshot shows a Python IDE with a file named 'main.py'. The code takes three integer inputs (a, b, c) and uses if-else statements to determine and print the largest number. The output pane shows the execution results.

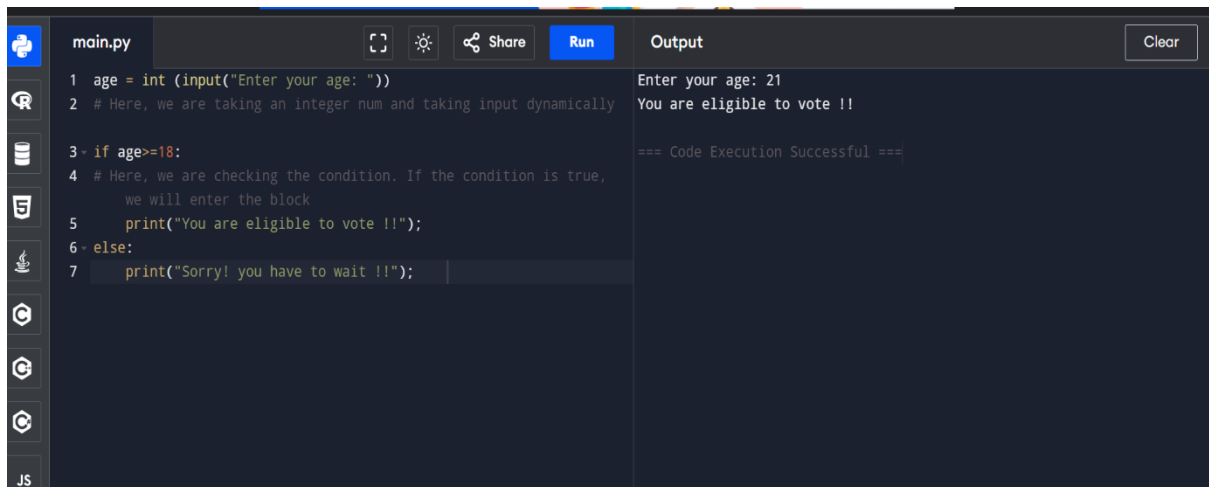
```
1 # Simple Python Program to print the largest of the three numbers
2
3 a = int (input("Enter a: "));
4 b = int (input("Enter b: "));
5 c = int (input("Enter c: "));
6
7 if a>b and a>c:
8     print ("From the above three numbers given a is largest");
9
10 if b>a and b>c:
11     print ("From the above three numbers given b is largest");
12
13 if c>a and c>b:
14     print ("From the above three numbers given c is largest");
```

Output

```
Enter a: 5
Enter b: 6
Enter c: 3
From the above three numbers given b is largest

=== Code Execution Successful ===
```

Program to check whether a person is eligible to vote or not.



The screenshot shows a Python IDE with a file named 'main.py'. The code is as follows:

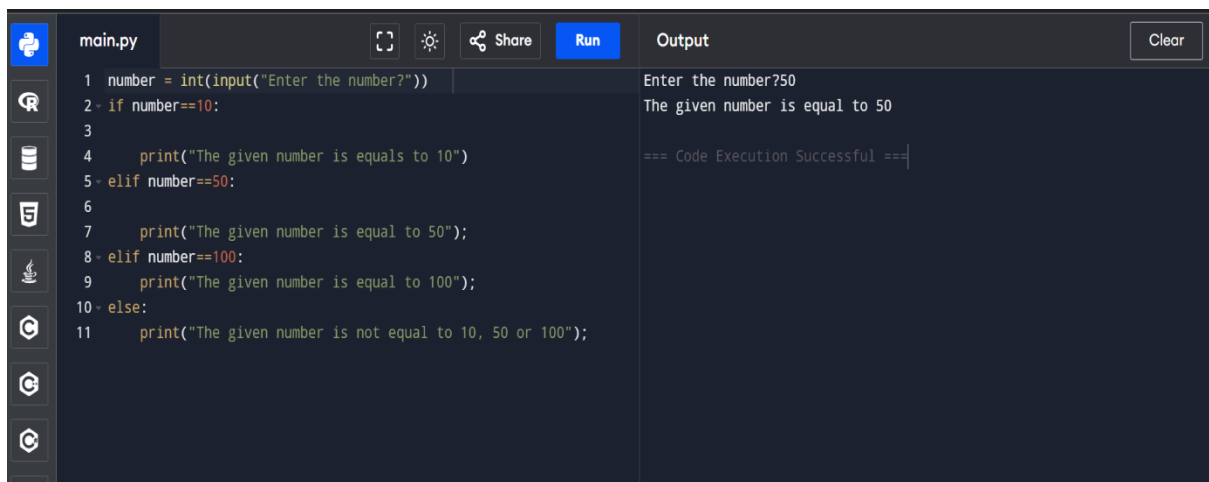
```
1 age = int (input("Enter your age: "))
2 # Here, we are taking an integer num and taking input dynamically
3
4 if age>=18:
5     # Here, we are checking the condition. If the condition is true,
6     # we will enter the block
7     print("You are eligible to vote !!");
8 else:
9     print("Sorry! you have to wait !!");
```

The output window on the right shows the following text:

```
Enter your age: 21
You are eligible to vote !!

=== Code Execution Successful ===
```

Program to check whether a number is even or not



The screenshot shows a Python IDE with a file named 'main.py'. The code is as follows:


```
1 number = int(input("Enter the number?"))
2 if number==10:
3
4     print("The given number is equals to 10")
5 elif number==50:
6
7     print("The given number is equal to 50");
8 elif number==100:
9     print("The given number is equal to 100");
10 else:
11     print("The given number is not equal to 10, 50 or 100");
```

The output window on the right shows the following text:

```
Enter the number?50
The given number is equal to 50

=== Code Execution Successful ===
```

Simple Python program to understand elif statement



The screenshot shows a Python IDE with a file named 'main.py'. The code is as follows:

```
1 number = int(input("Enter the number?"))
2 if number==10:
3
4     print("The given number is equals to 10")
5 elif number==50:
6
7     print("The given number is equal to 50");
8 elif number==100:
9     print("The given number is equal to 100");
10 else:
11     print("The given number is not equal to 10, 50 or 100");
```

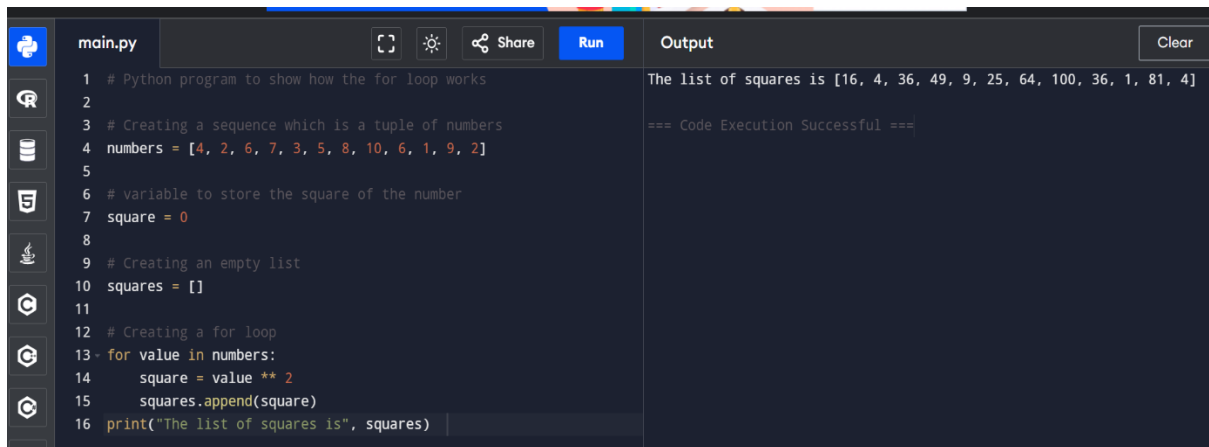
The output window on the right shows the following text:

```
Enter the number?50
The given number is equal to 50

=== Code Execution Successful ===
```


PYTHON FOR LOOP

Python program to show how the for loop works



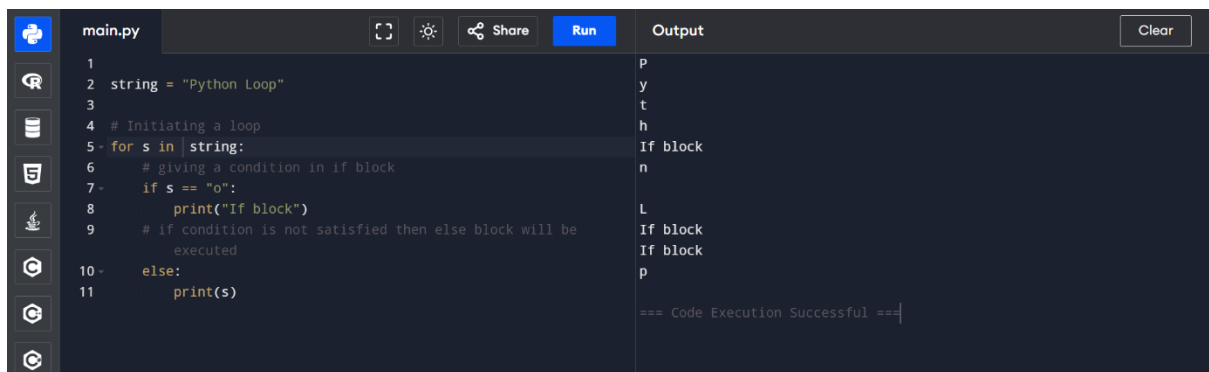
The screenshot shows a Python IDE with a file named 'main.py'. The code in the editor is as follows:

```
1 # Python program to show how the for loop works
2
3 # Creating a sequence which is a tuple of numbers
4 numbers = [4, 2, 6, 7, 3, 5, 8, 10, 6, 1, 9, 2]
5
6 # variable to store the square of the number
7 square = 0
8
9 # Creating an empty list
10 squares = []
11
12 # Creating a for loop
13 for value in numbers:
14     square = value ** 2
15     squares.append(square)
16 print("The list of squares is", squares)
```

The output window on the right shows the result of the execution:

```
The list of squares is [16, 4, 36, 49, 9, 25, 64, 100, 36, 1, 81, 4]
=== Code Execution Successful ===
```

Python program to show how if-else statements work



The screenshot shows a Python IDE with a file named 'main.py'. The code in the editor is as follows:

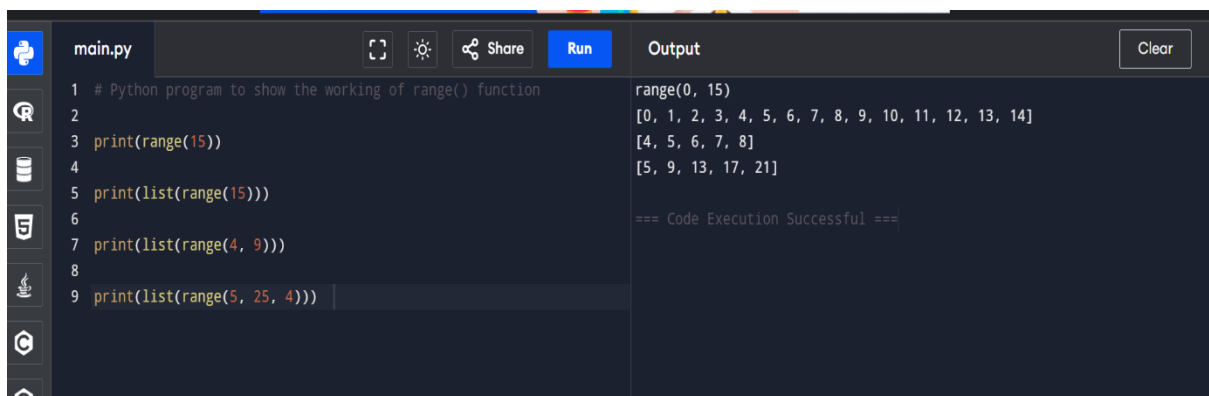
```
1
2 string = "Python Loop"
3
4 # Initiating a loop
5 for s in string:
6     # giving a condition in if block
7     if s == "o":
8         print("If block")
9     # if condition is not satisfied then else block will be
    executed
10 else:
11     print(s)
```

The output window on the right shows the result of the execution:

```
P
y
t
h
o
n

L
If block
If block
p
=== Code Execution Successful ===
```

Python program to show the working of range() function



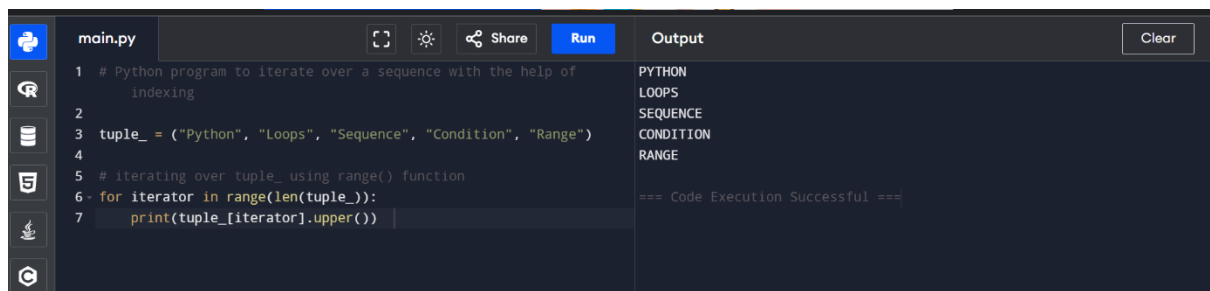
The screenshot shows a Python IDE with a file named 'main.py'. The code in the editor is as follows:

```
1 # Python program to show the working of range() function
2
3 print(range(15))
4
5 print(list(range(15)))
6
7 print(list(range(4, 9)))
8
9 print(list(range(5, 25, 4)))
```

The output window on the right shows the result of the execution:

```
range(0, 15)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[4, 5, 6, 7, 8]
[5, 9, 13, 17, 21]
=== Code Execution Successful ===
```

Python program to iterate over a sequence with the help of indexing



```
main.py
1 # Python program to iterate over a sequence with the help of
  indexing
2
3 tuple_ = ("Python", "Loops", "Sequence", "Condition", "Range")
4
5 # iterating over tuple_ using range() function
6 for iterator in range(len(tuple_)):
7     print(tuple_[iterator].upper())
```

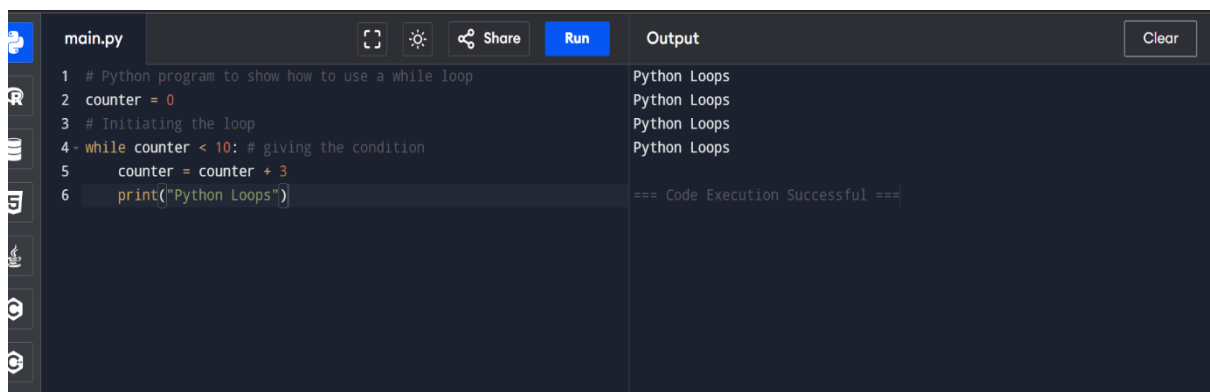
Output

```
PYTHON
LOOPS
SEQUENCE
CONDITION
RANGE

=== Code Execution Successful ===
```

PYTHON WHILE LOOP

Python program to show how to use a while loop



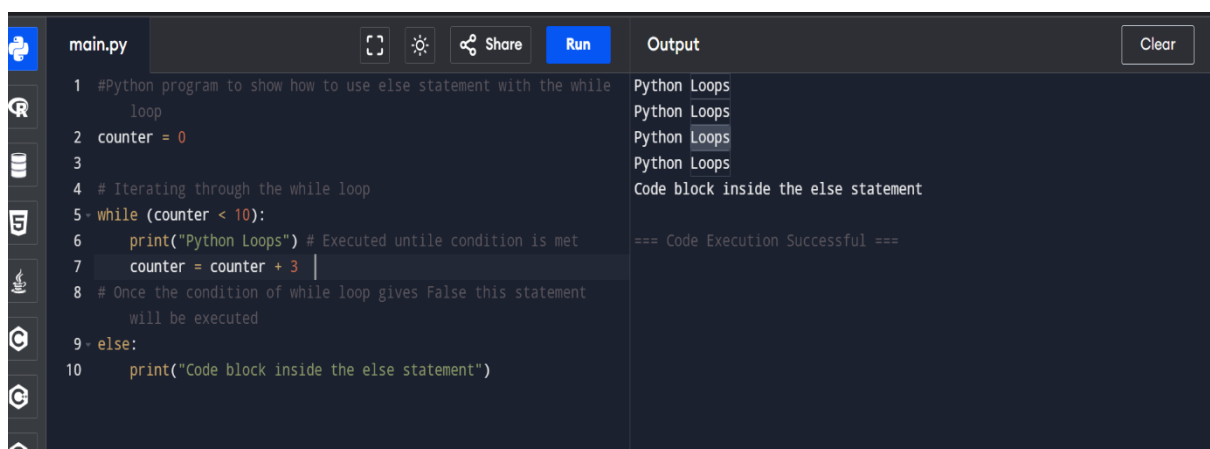
```
main.py
1 # Python program to show how to use a while loop
2 counter = 0
3 # Initiating the loop
4 while counter < 10: # giving the condition
5     counter = counter + 3
6     print("Python Loops")
```

Output

```
Python Loops
Python Loops
Python Loops
Python Loops

=== Code Execution Successful ===
```

Using else Statement with while Loops



```
main.py
1 #Python program to show how to use else statement with the while
  loop
2 counter = 0
3
4 # Iterating through the while loop
5 while (counter < 10):
6     print("Python Loops") # Executed untile condition is met
7     counter = counter + 3
8 # Once the condition of while loop gives False this statement
  will be executed
9 else:
10    print("Code block inside the else statement")
```

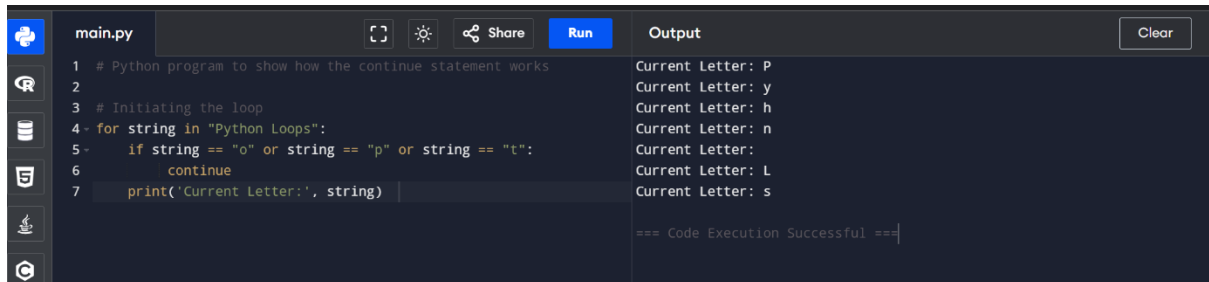
Output

```
Python Loops
Python Loops
Python Loops
Python Loops
Code block inside the else statement

=== Code Execution Successful ===
```

Continue Statement

Python program to show how the continue statement works



```
main.py
1 # Python program to show how the continue statement works
2
3 # Initiating the loop
4 for string in "Python Loops":
5     if string == "o" or string == "p" or string == "t":
6         continue
7     print('Current Letter:', string)
```

Output

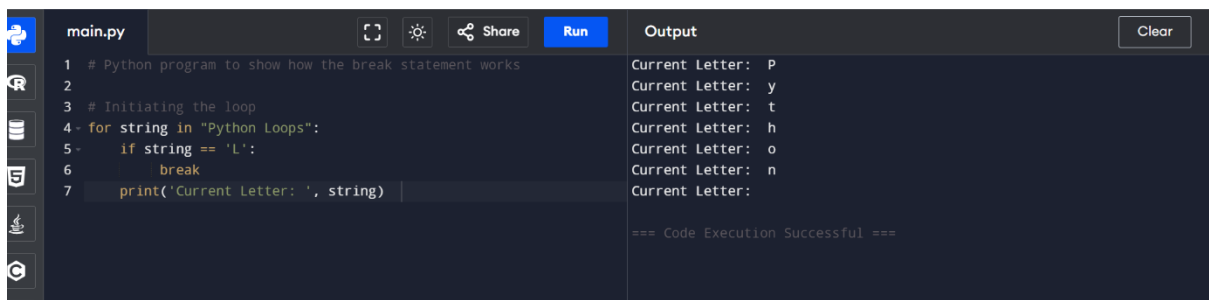
```
Current Letter: P
Current Letter: y
Current Letter: h
Current Letter: n
Current Letter: 
Current Letter: L
Current Letter: s

=== Code Execution Successful ===
```

Break Statement

It stops the execution of the loop when the break statement is reached.

Code:-



```
main.py
1 # Python program to show how the break statement works
2
3 # Initiating the loop
4 for string in "Python Loops":
5     if string == 'L':
6         break
7     print('Current Letter: ', string)
```

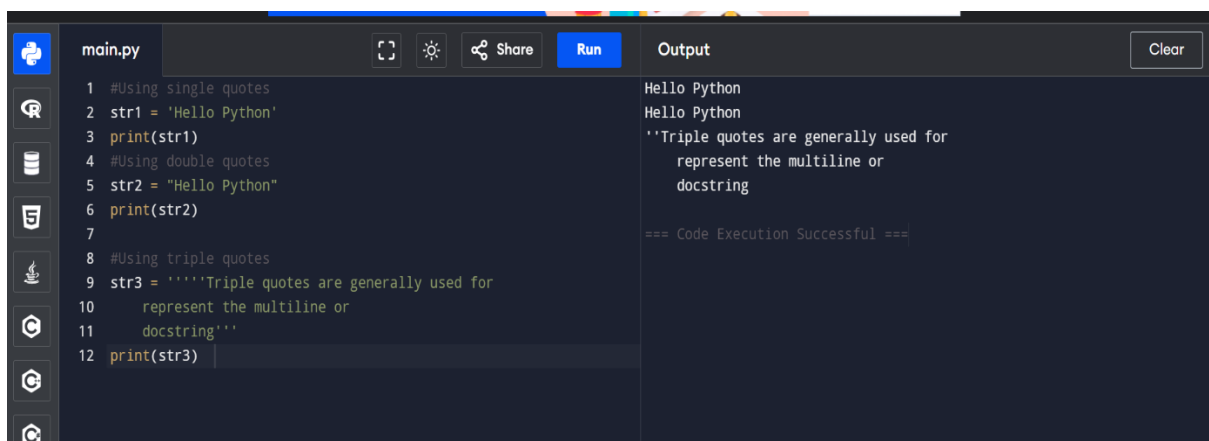
Output

```
Current Letter: P
Current Letter: y
Current Letter: t
Current Letter: h
Current Letter: o
Current Letter: n
Current Letter: 

=== Code Execution Successful ===
```

PYTHON STRINGS

Creating String in Python



```
main.py
1 #Using single quotes
2 str1 = 'Hello Python'
3 print(str1)
4 #Using double quotes
5 str2 = "Hello Python"
6 print(str2)
7
8 #Using triple quotes
9 str3 = '''Triple quotes are generally used for
10     represent the multiline or
11     docstring'''
12 print(str3)
```

Output

```
Hello Python
Hello Python
'''Triple quotes are generally used for
    represent the multiline or
    docstring'''

=== Code Execution Successful ===
```

Strings indexing and splitting

Example 1:

```
main.py  [ ] [ ] [ ] Share Run Output Clear
1 str = "HELLO"
2 print(str[0])
3 print(str[1])
4 print(str[2])
5 print(str[3])
6 print(str[4])
7 # It returns the IndexError because 6th index doesn't exist
8 print(str[6])
```

Output

```
H
E
L
L
O
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 8, in <module>
IndexError: string index out of range

=== Code Exited With Errors ===
```

Example 2

```
main.py  [ ] [ ] [ ] Share Run Output Clear
1 |
2 str = "JAVATPOINT"
3 print(str[0:])
4 # Starts 1th index to 4th index
5 print(str[1:5])
6 # Starts 2nd index to 3rd index
7 print(str[2:4])
8 # Starts 0th to 2nd index
9 print(str[:3])
10 #Starts 4th to 6th index
11 print(str[4:7])
```

Output

```
JAVATPOINT
AVAT
VA
JAV
TPO

=== Code Execution Successful ===
```

Strings Operators

```
main.py  [ ] [ ] [ ] Share Run Output Clear
1 str = "Hello"
2 str1 = " world"
3 print(str*3) # prints HelloHelloHello
4 print(str+str1) # prints Hello world
5 print(str[4]) # prints o
6 print(str[2:4]); # prints ll
7 print('w' in str) # prints false as w is not present in str
8 print('wo' not in str1) # prints false as wo is present in str1.

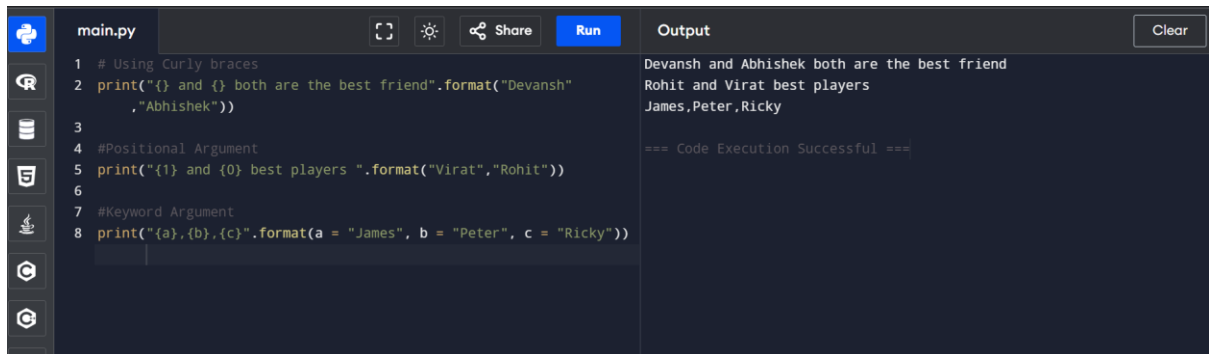
9 print(r'C://python37') # prints C://python37 as it is written
10 print("The string str : %s"%(str)) # prints The string str :
    Hello
```

Output

```
HelloHelloHello
Hello world
o
ll
False
False
C://python37
The string str : Hello

=== Code Execution Successful ===
```

String formatting



The screenshot shows a Python IDE with a file named `main.py`. The code demonstrates three different string formatting methods: curly braces, positional arguments, and keyword arguments. The output window shows the results of these formatting operations.

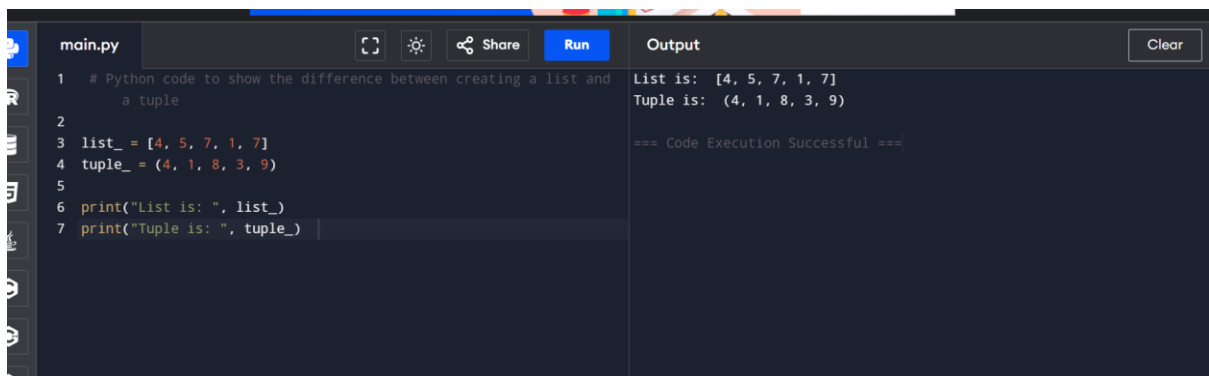
```
1 # Using Curly braces
2 print("{} and {} both are the best friend".format("Devansh",
3         , "Abhishek"))
4
5 #Positional Argument
6 print("{1} and {0} best players ".format("Virat","Rohit"))
7
8 #Keyword Argument
9 print("{a},{b},{c}".format(a = "James", b = "Peter", c = "Ricky"))
```

Output:

```
Devansh and Abhishek both are the best friend
Rohit and Virat best players
James,Peter,Ricky

=== Code Execution Successful ===
```

List and Tuple Syntax Differences



The screenshot shows a Python IDE with a file named `main.py`. The code demonstrates the syntax for creating a list and a tuple, and then printing them. The output window shows the resulting list and tuple.

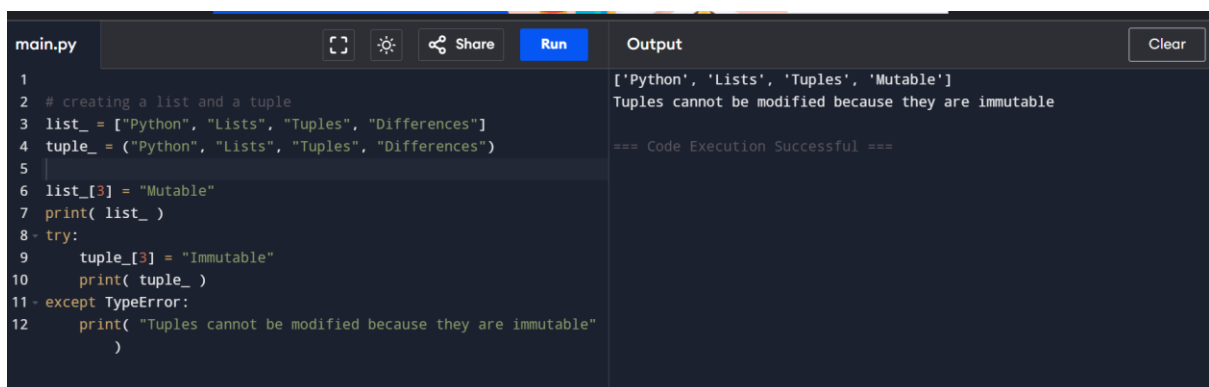
```
1 # Python code to show the difference between creating a list and
  a tuple
2
3 list_ = [4, 5, 7, 1, 7]
4 tuple_ = (4, 1, 8, 3, 9)
5
6 print("List is: ", list_)
7 print("Tuple is: ", tuple_)
```

Output:

```
List is: [4, 5, 7, 1, 7]
Tuple is: (4, 1, 8, 3, 9)

=== Code Execution Successful ===
```

Updating the element of list and tuple at a particular index



The screenshot shows a Python IDE with a file named `main.py`. The code demonstrates attempting to modify a list and a tuple. The list is successfully modified, while the tuple modification raises a `TypeError`. The output window shows the results of these operations.

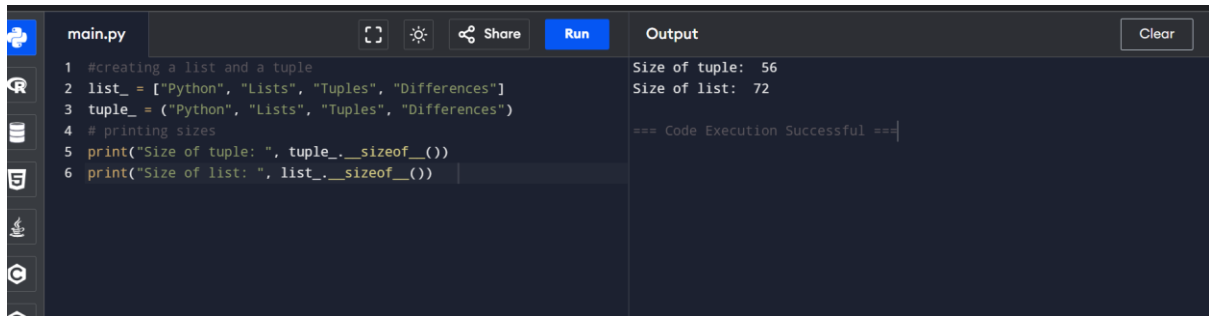
```
1
2 # creating a list and a tuple
3 list_ = ["Python", "Lists", "Tuples", "Differences"]
4 tuple_ = ("Python", "Lists", "Tuples", "Differences")
5
6 list_[3] = "Mutable"
7 print( list_ )
8 - try:
9     tuple_[3] = "Immutable"
10    print( tuple_ )
11 - except TypeError:
12    print( "Tuples cannot be modified because they are immutable"
13          )
```

Output:

```
['Python', 'Lists', 'Tuples', 'Mutable']
Tuples cannot be modified because they are immutable

=== Code Execution Successful ===
```

Code to show the difference in the size of a list and a tuple



The image shows a code editor interface with a dark theme. On the left is a sidebar with icons for file explorer, search, and other tools. The main editor area displays a Python script named `main.py`. The script creates a list and a tuple with the same elements and prints their sizes. The output pane on the right shows the results of the execution.

```
1 #creating a list and a tuple
2 list_ = ["Python", "Lists", "Tuples", "Differences"]
3 tuple_ = ("Python", "Lists", "Tuples", "Differences")
4 # printing sizes
5 print("Size of tuple: ", tuple_.__sizeof__())
6 print("Size of list: ", list_.__sizeof__())
```

Output

```
Size of tuple: 56
Size of list: 72

=== Code Execution Successful ===
```