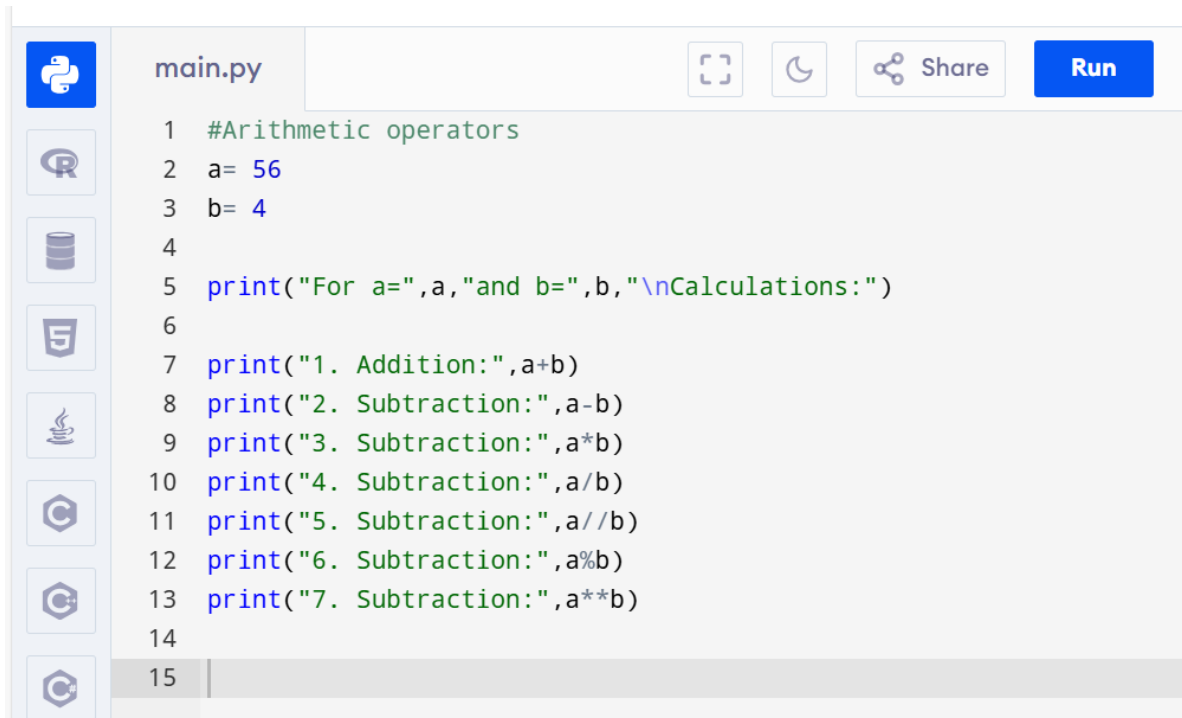


1. Arithmetic Operators:



```
main.py

1 #Arithmetic operators
2 a= 56
3 b= 4
4
5 print("For a=",a,"and b=",b,"\nCalculations:")
6
7 print("1. Addition:",a+b)
8 print("2. Subtraction:",a-b)
9 print("3. Subtraction:",a*b)
10 print("4. Subtraction:",a/b)
11 print("5. Subtraction:",a//b)
12 print("6. Subtraction:",a%b)
13 print("7. Subtraction:",a**b)
14
15
```

Output

Clear

For a= 56 and b= 4
Calculations:
1. Addition: 60
2. Subtraction: 52
3. Subtraction: 224
4. Subtraction: 14.0
5. Subtraction: 14
6. Subtraction: 0
7. Subtraction: 9834496

=== Code Execution Successful ===

2. Comparison operators:

```
main.py  [Full Screen] [Dark Mode] [Share] [Run]

1 #Comparison operators
2 a = 56
3 b = 4
4
5 print("For a =", a, "and b =", b, "\nCheck the following:")
6
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)
13
14
```




```
Output [Clear]

For a = 56 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 ===
```

3. Assignment operators:

main.py

 Share

Run

```
1 #Assignment operators
2 a = 56
3 b = 4
4
5 # printing the different results
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)
12 print('a //= b:', a // b)
```

Output

Clear

```
a += b: 60
a -= b: 52
a *= b: 224
a /= b: 14.0
a %= b: 0
a **= b: 9834496
a //= b: 14

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

4. Bitwise operators:

```
main.py    Share 
```

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

Output




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

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

5. Logical operators:

main.py




Output



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

main.py





Output



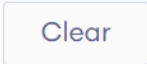
```
For a = 7, checking whether the following conditions are True or False:
"a > 5 and a < 7" => False
"a > 5 or a < 7" => True
"not (a > 5 and a < 7)" => True

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

6. Membership operators:

main.py    Share 

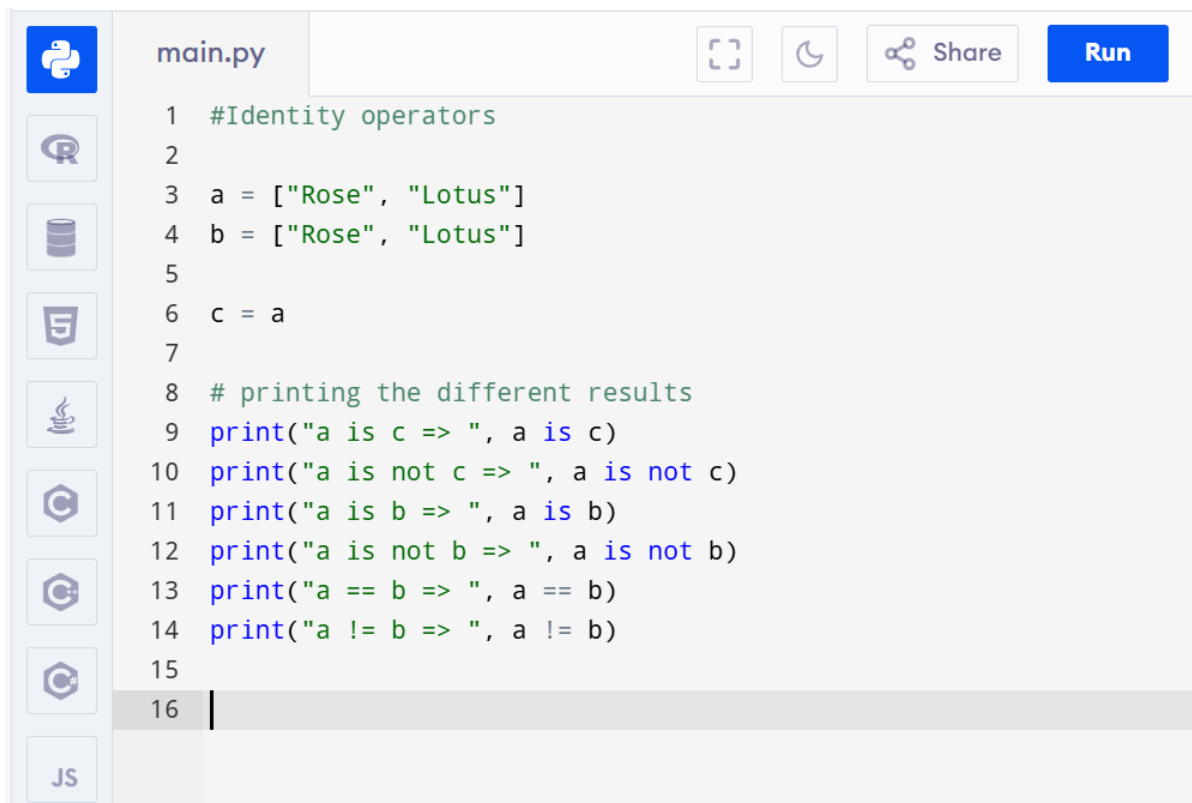
```
1 #Membership operators
2 myList = [12, 22, 28, 35, 42, 49, 54, 65, 92, 103, 245, 874]
3
4 x = 31
5 y = 28
6
7 print("Given List:", myList)
8
9 # checking if x is present in the list or not
10 if (x not in myList):
11     print("x =", x,"is NOT present in the given list.")
12 else:
13     print("x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

Output 

```
Given List: [12, 22, 28, 35, 42, 49, 54, 65, 92, 103, 245, 874]
x = 31 is NOT present in the given list.
y = 28 is present in the given list.

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

7. Identity operators:



```
main.py
1  #Identity operators
2
3  a = ["Rose", "Lotus"]
4  b = ["Rose", "Lotus"]
5
6  c = a
7
8  # printing the different results
9  print("a is c => ", a is c)
10 print("a is not c => ", a is not c)
11 print("a is b => ", a is b)
12 print("a is not b => ", a is not b)
13 print("a == b => ", a == b)
14 print("a != b => ", a != b)
15
16
```

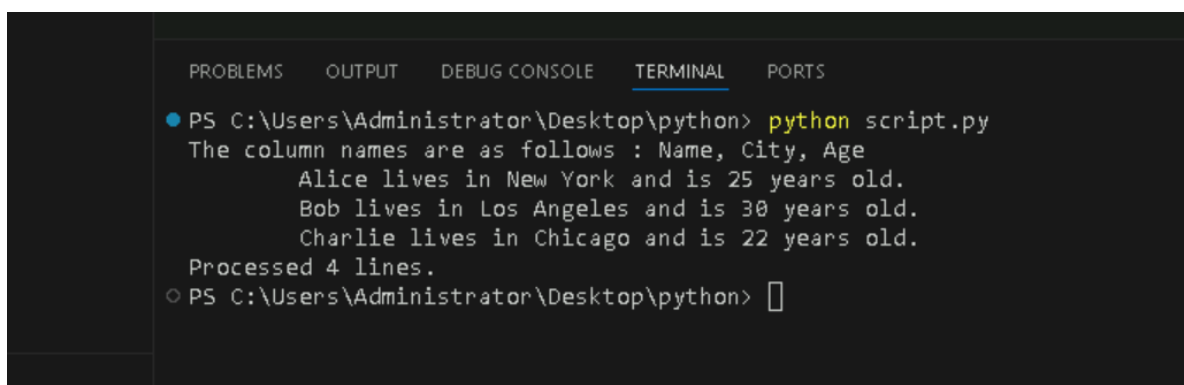
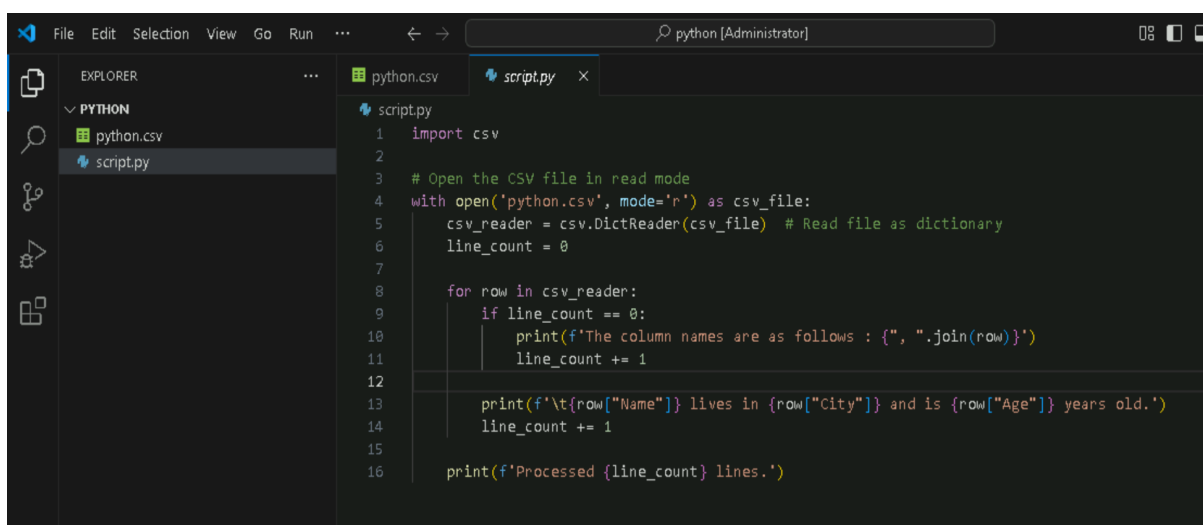
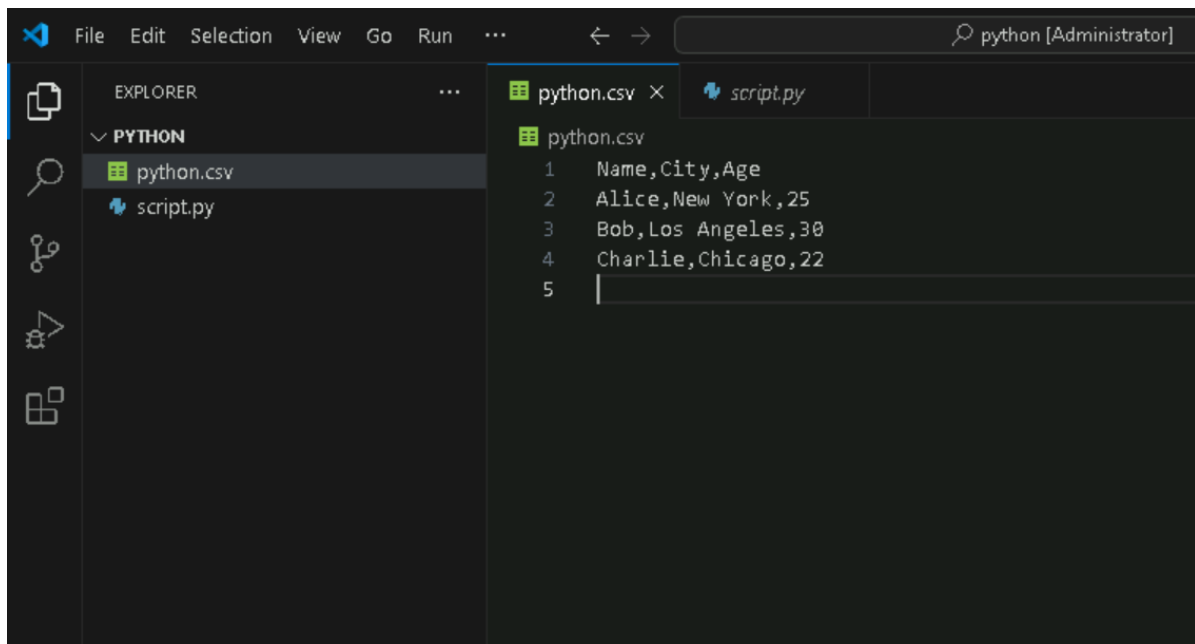


Output Clear

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

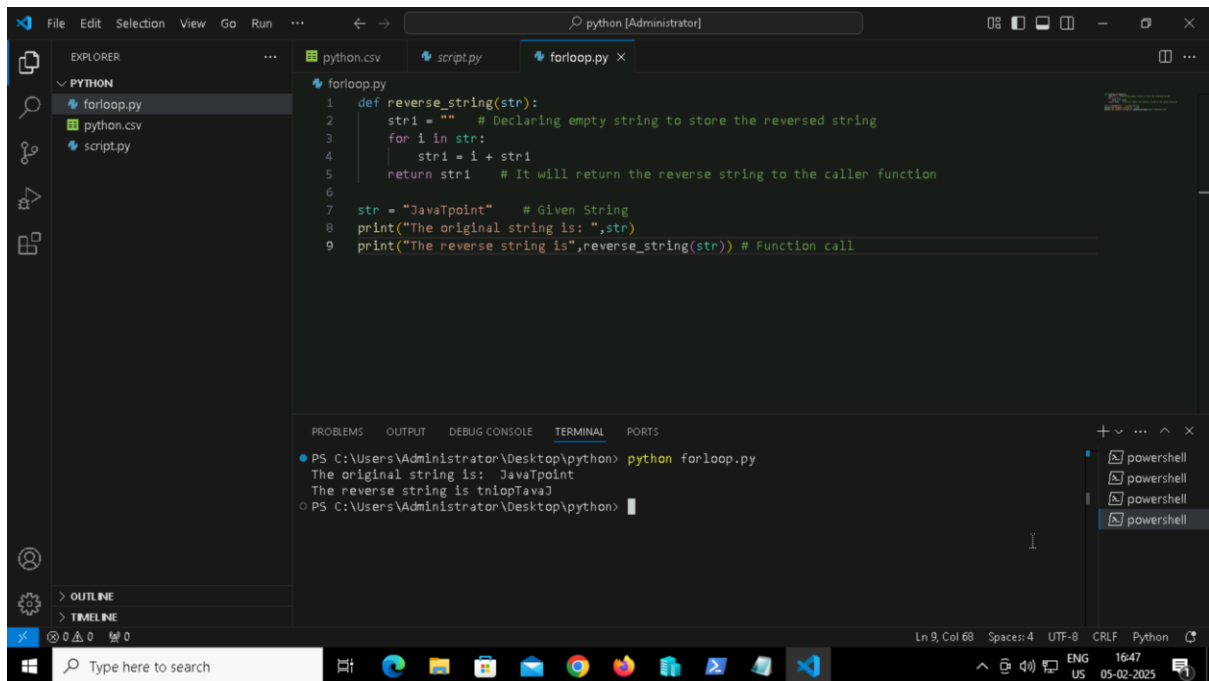
=== Code Execution Successful ===
```

How to read CSV file in Python?



REVERSE OF A STRING

Using For loop:



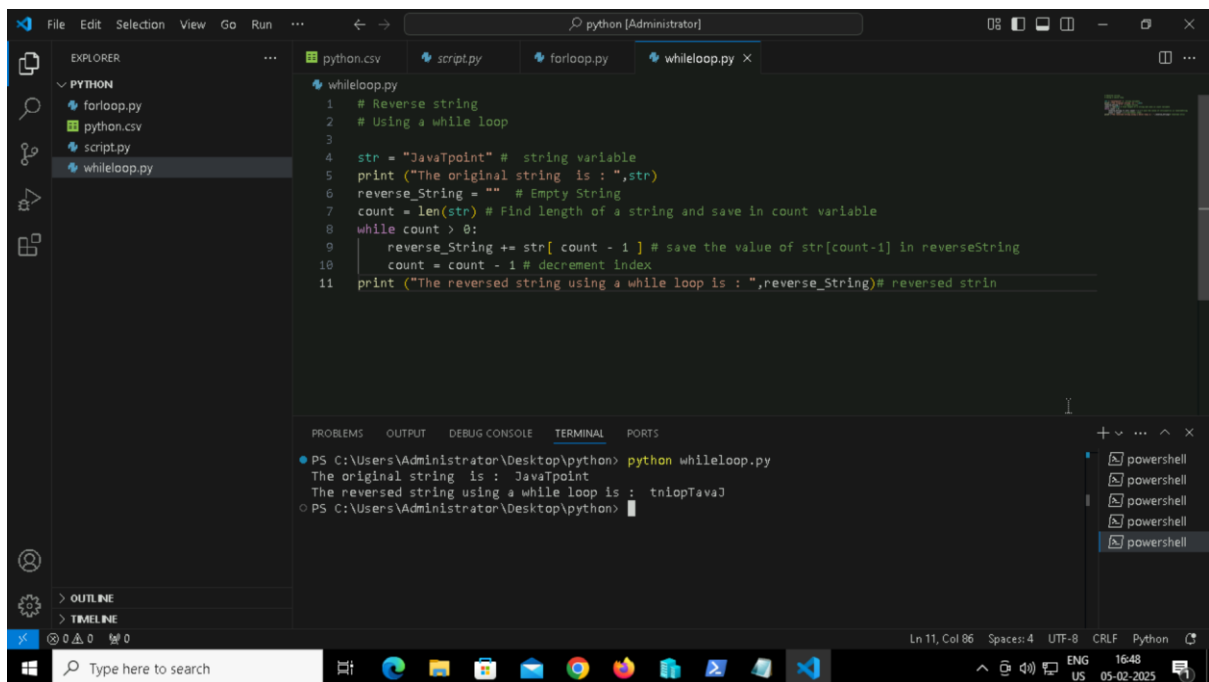
The screenshot shows the Visual Studio Code editor with a file named `forloop.py` open. The code defines a function `reverse_string` that takes a string `str` and returns its reverse using a `for` loop. The string `str` is set to `"JavaTpoint"`, and the function is called to print the original and reversed strings.

```
1 def reverse_string(str):
2     str1 = "" # Declaring empty string to store the reversed string
3     for i in str:
4         str1 = i + str1
5     return str1 # It will return the reverse string to the caller function
6
7 str = "JavaTpoint" # Given String
8 print("The original string is: ",str)
9 print("The reverse string is",reverse_string(str)) # Function call
```

The terminal output shows the execution of the script:

```
PS C:\Users\Administrator\Desktop\python> python forloop.py
The original string is: JavaTpoint
The reverse string is: tnioPavaJ
```

Using while loop:



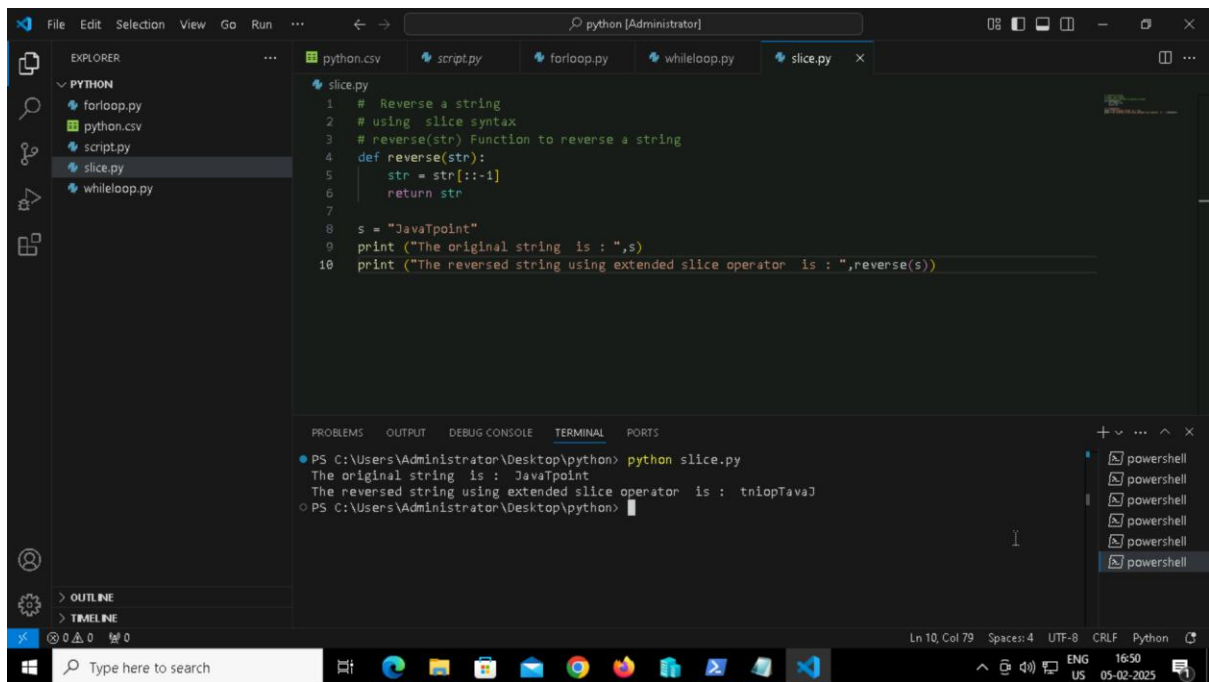
The screenshot shows the Visual Studio Code editor with a file named `whileloop.py` open. The code defines a function `reverse_string` that takes a string `str` and returns its reverse using a `while` loop. The string `str` is set to `"JavaTpoint"`, and the function is called to print the original and reversed strings.

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

The terminal output shows the execution of the script:

```
PS C:\Users\Administrator\Desktop\python> python whileloop.py
The original string is : JavaTpoint
The reversed string using a while loop is : tnioPavaJ
```

Using slice operator



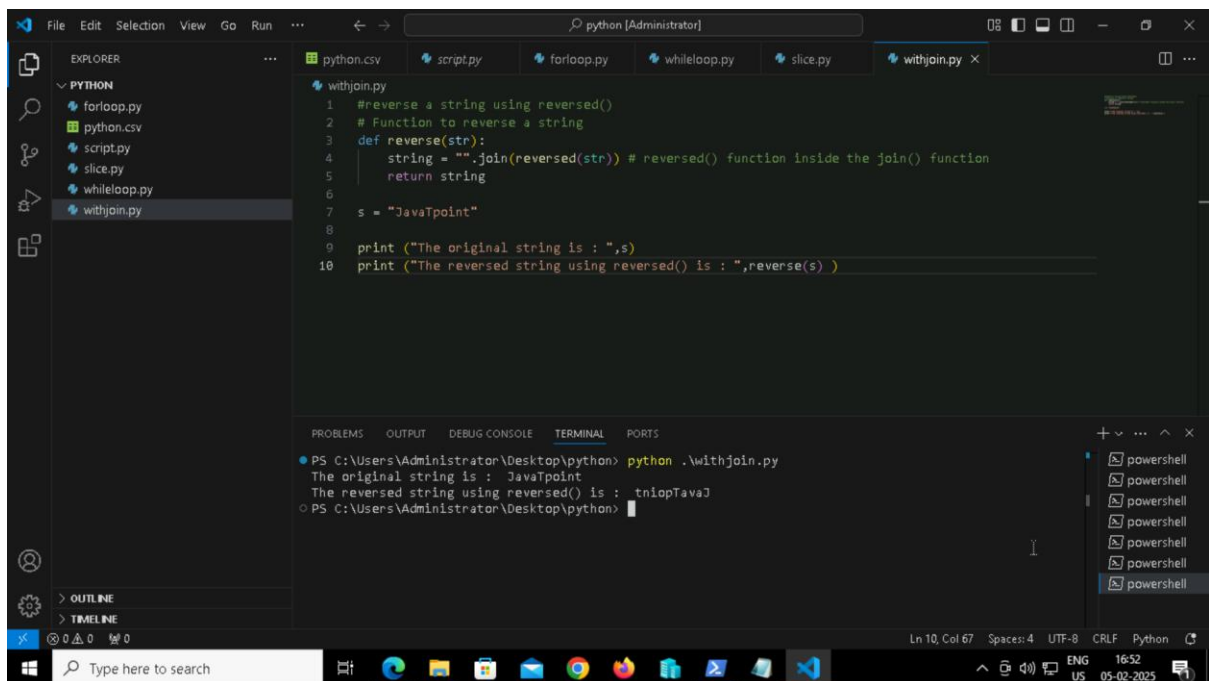
The screenshot shows the Visual Studio Code editor with a Python file named `slice.py` open. The code defines a function `reverse(str)` that uses slicing to reverse a string. The string `s = "JavaTpoint"` is printed, and then the reversed string is printed using the `reverse(s)` function.

```
1 # Reverse a string
2 # using slice syntax
3 # reverse(str) Function to reverse a string
4 def reverse(str):
5     str = str[::-1]
6     return str
7
8 s = "JavaTpoint"
9 print ("The original string is :",s)
10 print ("The reversed string using extended slice operator is :",reverse(s))
```

The terminal output shows the execution of the script:

```
PS C:\Users\Administrator\Desktop\python> python slice.py
The original string is : JavaTpoint
The reversed string using extended slice operator is : tniopTavaJ
PS C:\Users\Administrator\Desktop\python>
```

Using reverse function with join



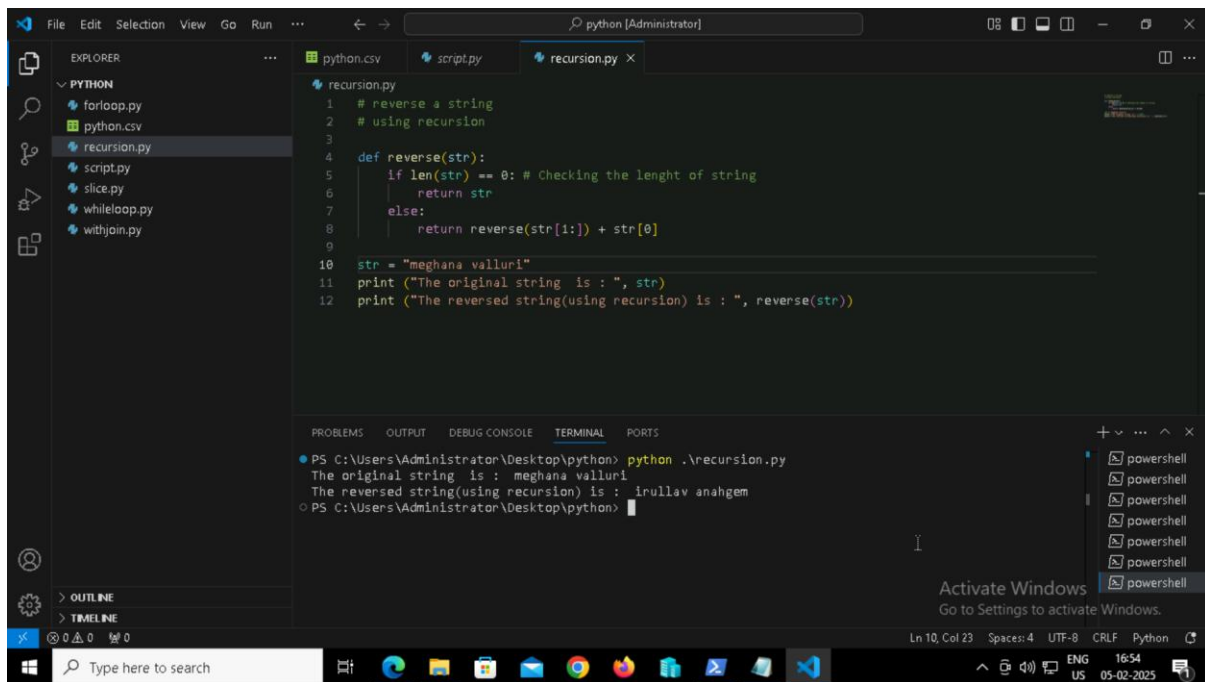
The screenshot shows the Visual Studio Code editor with a Python file named `withjoin.py` open. The code defines a function `reverse(str)` that uses the `reversed()` function inside the `join()` function to reverse a string. The string `s = "JavaTpoint"` is printed, and then the reversed string is printed using the `reverse(s)` function.

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

The terminal output shows the execution of the script:

```
PS C:\Users\Administrator\Desktop\python> python .\withjoin.py
The original string is : JavaTpoint
The reversed string using reversed() is : tniopTavaJ
PS C:\Users\Administrator\Desktop\python>
```

Using recursion



The screenshot shows a Python IDE with a file explorer on the left containing files like `forloop.py`, `python.csv`, `recursion.py`, `script.py`, `slice.py`, `whileloop.py`, and `withjoin.py`. The main editor displays `recursion.py` with the following code:

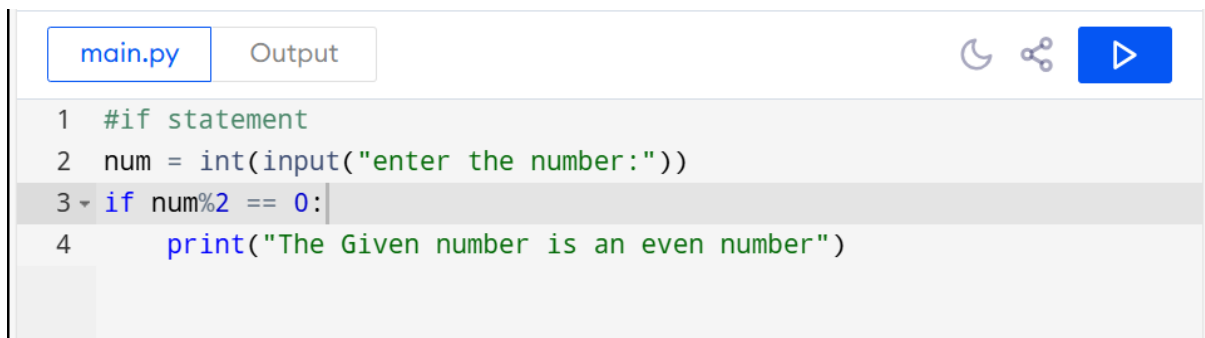
```
1 # reverse a string
2 # using recursion
3
4 def reverse(str):
5     if len(str) == 0: # Checking the length of string
6         return str
7     else:
8         return reverse(str[1:]) + str[0]
9
10 str = "meghana valluri"
11 print("The original string is : ", str)
12 print("The reversed string(using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

The terminal at the bottom shows the execution of `python .\recursion.py`, resulting in the output:

```
PS C:\Users\Administrator\Desktop\python> python .\recursion.py
The original string is : meghana valluri
The reversed string(using recursion) is : irullav anahgem
PS C:\Users\Administrator\Desktop\python>
```

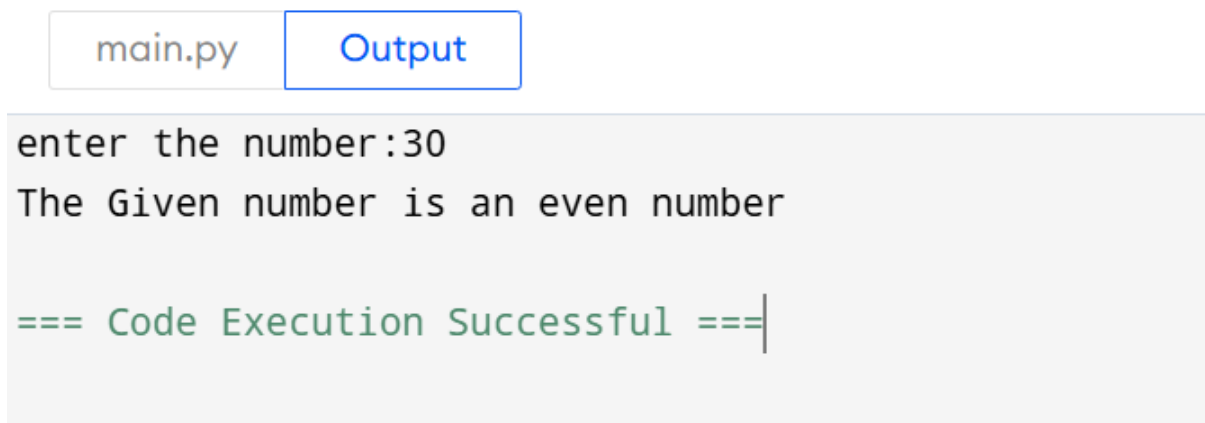
If statement:

Example 1:



The screenshot shows a code editor with a file explorer on the left containing `main.py` and `Output`. The main editor displays `main.py` with the following code:

```
1 #if statement
2 num = int(input("enter the number:"))
3 if num%2 == 0:
4     print("The Given number is an even number")
```






The screenshot shows a code editor with a file explorer on the left containing `main.py` and `Output`. The main editor displays `main.py` with the following code:

```
enter the number:30
The Given number is an even number
=== Code Execution Successful ===
```

Example 2: Program to print the largest of the three numbers.

main.py

Output



```
1 a = int (input("Enter a: "));
2 b = int (input("Enter b: "));
3 c = int (input("Enter c: "));
4 ▾ if a>b and a>c:
5     # Here, we are checking the condition. If the condition is true, we
      will enter the block
6     print ("From the above three numbers given a is largest");
7 ▾ if b>a and b>c:
8     # Here, we are checking the condition. If the condition is true, we
      will enter the block
9     print ("From the above three numbers given b is largest");
10 ▾ if c>a and c>b:
11     # Here, we are checking the condition. If the condition is true, we
      will enter the block
12     print ("From the above three numbers given c is largest");|
```

main.py

Output

```
Enter a: 3
Enter b: 887
Enter c: 567
From the above three numbers given b is largest




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

The if-else statement

Example 1:

main.py

Output



```
1 age = int (input("Enter your age: "))
2 ▾ if age>=18:|
3     print("You are eligible to vote !!");
4 ▾ else:
5     print("Sorry! you have to wait !!");
```

main.py

Output

```
Enter your age: 22  
You are eligible to vote !!
```

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

Example 2: Program to check whether a number is even or not.

main.py

Output

```
1 num = int(input("enter the number:"))  
2 if num%2 == 0:  
3     print("The Given number is an even number")  
4 else:  
5     print("The Given Number is an odd number")
```

main.py

Output

```
enter the number:45  
The Given Number is an odd number
```




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

The elif statement

Example 1:

main.py




Output



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

main.py

Output






Enter the number:50
The given number is equal to 50

=== Code Execution Successful ===

Example 2

main.py

Output



```
1 marks = int(input("Enter the marks: "))
2 ▾ if marks > 85 and marks <= 100:
3     print("Congrats ! you scored grade A ...")
4 ▾ elif marks > 60 and marks <= 85:
5     print("You scored grade B + ...")
6 ▾ elif marks > 40 and marks <= 60:
7     print("You scored grade B ...")
8 ▾ elif (marks > 30 and marks <= 40):
9     print("You scored grade C ...")
10 ▾ else:
11     print("Sorry you are fail ?")
```

main.py

Output

Enter the marks: 89
Congrats ! you scored grade A ...

=== Code Execution Successful ===

Python Loops

The for Loop:

main.py

Output



```
1 numbers = [4, 2, 6, 7, 3, 5, 8, 10, 6, 1, 9, 2]
2
3 square = 0
4
5 squares = []
6
7 for value in numbers:
8     square = value ** 2
9     squares.append(square)
10 print("The list of squares is", squares)
```

main.py

Output






The list of squares is [16, 4, 36, 49, 9, 25, 64, 100, 36, 1, 81, 4]

=== Code Execution Successful ===

Using else Statement with for Loop

Example 1:

```
main.py Output   
```

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




```
main.py Output   
```

```
P
y
t
h
If block
n

L
If block
If block
p

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




Example 2:

```
main.py Output   
```

```
1 tuple_ = (3, 4, 6, 8, 9, 2, 3, 8, 9, 7)
2
3 # Initiating the loop
4 for value in tuple_:
5     if value % 2 != 0:
6         print(value)
7     # giving an else statement
8     else:
9         print("These are the odd numbers present in the tuple")
```


main.py

Output






```
3
9
3
9
7
These are the odd numbers present in the tuple

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

The range() Function

main.py




Output



```
1 print(range(15))
2
3 print(list(range(15)))
4
5 print(list(range(4, 9)))
6
7 print(list(range(5, 25, 4)))
```

main.py

Output






```
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 ===
```

To iterate over a sequence with the help of indexing

main.py




Output



```
1 tuple_ = ("Python", "Loops", "Sequence", "Condition", "Range")
2
3 # iterating over tuple_ using range() function
4 for iterator in range(len(tuple_)):
5     print(tuple_[iterator].upper())
```

main.py

Output



```
PYTHON
LOOPS
SEQUENCE
CONDITION
RANGE




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

While Loop

Example 1:

main.py




Output



```
1 counter = 0
2
3 while counter < 10: # giving the condition
4     counter = counter + 3
5     print("Python Loops")
```

main.py

Output






```
Python Loops
Python Loops
Python Loops
Python Loops

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

Using else Statement with while Loops

main.py




Output



```
1 counter = 0
2
3 while (counter < 10):
4     counter = counter + 3
5     print("Python Loops")
6 else:
7     print("Code block inside the else statement")
```

main.py

Output



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

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

to write a single statement while loop

main.py




Output



```
1 count = 0
2 while (count < 3):
3     print("Python Loops")
```

main.py

Output






```
Python Loops
Python Loops
Python Loops
Python Loops
Python Loops
Python Loops
Python Loops
Python Loops
Python Loops
Python Loops
```

Loop Control Statements

Continue Statement

main.py




Output



```
1 for string in "Python Loops":
2     if string == "o" or string == "p" or string == "t":
3         continue
4     print('Current Letter:', string)
```

main.py

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

main.py




Output



```
1 for string in "Python Loops":
2     if string == 'L':
3         break
4     print('Current Letter: ', string)
```

main.py

Output






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

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

Pass Statement

main.py




Output



```
1 for string in "Python Loops":
2     pass
3     print( 'Last Letter:', string)
```

main.py

Output



Last Letter: s




=== Code Execution Successful ===

Python String

Creating String in Python

main.py




Output



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

main.py

Output




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

Strings indexing and splitting

Example 1:

main.py




Output



```
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])
```

main.py

Output





```
H
E
L
L
O
```

Example 2:

main.py



Output



```
1 # Given String
2 str = "JAVATPOINT"
3 # Start 0th index to end
4 print(str[0:])
5 # Starts 1th index to 4th index
6 print(str[1:5])
7 # Starts 2nd index to 3rd index
8 print(str[2:4])
9 # Starts 0th to 2nd index
10 print(str[:3])
11 #Starts 4th to 6th index
12 print(str[4:7])
```

main.py

Output



```
JAVATPOINT
AVAT
VA
JAV
TPO

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

Example 3:

main.py




Output



```
1 str = 'JAVATPOINT'
2 print(str[-1])
3 print(str[-3])
4 print(str[-2:])
5 print(str[-4:-1])
6 print(str[-7:-2])
7 # Reversing the given string
8 print(str[::-1])
9 print(str[-12:])
```

main.py

Output






```
T
I
NT
OIN
ATPOI
TNIPTAVAJ
.....
```

Reassigning Strings

main.py

Output



```
1 str = "HELLO"
2 print(str)
3 str = "hello"
4 print(str) |
```

```
HELLO
hello

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

The format() method

main.py




Output



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

main.py

Output






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

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

Python String Formatting Using % Operator

main.py




Output



```
1 Integer = 10;
2 Float = 1.290
3 String = "Devansh"
4 print("Hi I am Integer ... My value is %d\nHi I am float ... My value
    is %f\nHi I am string ... My value is %s"%(Integer,Float,String))
```

main.py

Output



```
Hi I am Integer ... My value is 10
Hi I am float ... My value is 1.290000
Hi I am string ... My value is Devansh




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

Python List

List Declaration

main.py




Output



```
1 # a simple list
2 list1 = [1, 2, "Python", "Program", 15.9]
3 list2 = ["Amy", "Ryan", "Henry", "Emma"]
4
5 # printing the list
6 print(list1)
7 print(list2)
8
9 # printing the type of list
10 print(type(list1))
11 print(type(list2))
```


main.py

Output



```
[1, 2, 'Python', 'Program', 15.9]
['Amy', 'Ryan', 'Henry', 'Emma']
<class 'list'>
<class 'list'>




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

Ordered List Checking

Example 1:

main.py

Output



```
1 # example
2 a = [ 1, 2, "Ram", 3.50, "Rahul", 5, 6 ]
3 b = [ 1, 2, 5, "Ram", 3.50, "Rahul", 6 ]
4 print(a == b)
```

main.py

Output






```
False

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

Example 2:

main.py




Output



```
1 # example
2 a = [ 1, 2, "Ram", 3.50, "Rahul", 5, 6 ]
3 b = [ 1, 2, "Ram", 3.50, "Rahul", 5, 6 ]
4 print(a == b)
```

main.py

Output






```
True

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

List Indexing and Splitting

main.py




Output



```
1 list = [1,2,3,4,5,6,7]
2 print(list[0])
3 print(list[1])
4 print(list[2])
5 print(list[3])
6 # Slicing the elements
7 print(list[0:6])
8 # By default, the index value is 0 so its starts from the 0th
  element and go for index -1.
9 print(list[:])
10 print(list[2:5])
11 print(list[1:6:2])
```

main.py

Output






```
1
2
3
4
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6, 7]
[3, 4, 5]
[2, 4, 6]

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

Updating List Values

main.py




Output



```
1 # updating list values
2 list = [1, 2, 3, 4, 5, 6]
3 print(list)
4 # It will assign value to the value to the second index
5 list[2] = 10
6 print(list)
7 # Adding multiple-element
8 list[1:3] = [89, 78]
9 print(list)
10 # It will add value at the end of the list
11 list[-1] = 25
12 print(list)
```

main.py

Output






```
[1, 2, 3, 4, 5, 6]
[1, 2, 10, 4, 5, 6]
[1, 89, 78, 4, 5, 6]
[1, 89, 78, 4, 5, 25]

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

List and Tuple Syntax Differences

main.py



Output



```
1 list_ = [4, 5, 7, 1, 7]
2 tuple_ = (4, 1, 8, 3, 9)
3
4 print("List is: ", list_)
5 print("Tuple is: ", tuple_)
```

main.py

Output






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

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

Mutable List vs. Immutable Tuple

main.py




Output



```
1 list_ = ["Python", "Lists", "Tuples", "Differences"]
2 tuple_ = ("Python", "Lists", "Tuples", "Differences")
3
4 # modifying the last string in both data structures
5 list_[3] = "Mutable"
6 print( list_ )
7 try:
8     tuple_[3] = "Immutable"
9     print( tuple_ )
10 except TypeError:
11     print( "Tuples cannot be modified because they are immutable" )
```

main.py

Output



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

Size Difference

main.py




Output



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

main.py

Output






```
Size of tuple: 56  
Size of list: 72  
  
=== Code Execution Successful ===
```

Python Functions:

Illustration of a User-Defined Function

main.py

Output



```
1 def square( num ):  
2     """  
3     This function computes the square of the number.  
4     """  
5     return num**2  
6 object_ = square(6)  
7 print( "The square of the given number is: ", object_ )
```

main.py




Output

```
The square of the given number is: 36  
  
=== Code Execution Successful ===
```

Calling a Function

main.py

Output



```
1 def a_function( string ):  
2     "This prints the value of length of string"  
3     return len(string)  
4  
5 # Calling the function we defined  
6 print( "Length of the string Functions is: ", a_function( "Functions"  
7       ) )  
7 print( "Length of the string Python is: ", a_function( "Python" ) )
```

main.py




Output

```
Length of the string Functions is:  9  
Length of the string Python is:  6
```

Pass by Reference vs. Pass by Value

main.py

Output



```
1 def square( item_list ):  
2     '''This function will find the square of items in the  
3     list'''  
4     squares = [ ]  
5     for l in item_list:  
6         squares.append( l**2 )  
7     return squares  
8  
9 # calling the defined function  
10 my_list = [17, 52, 8];  
11 my_result = square( my_list )  
11 print( "Squares of the list are: ", my_result )
```

main.py

Output




```
Squares of the list are:  [289, 2704, 64]
```

Function Arguments

1. Default arguments

main.py

Output



```
1 def function( n1, n2 = 20 ):
2     print("number 1 is: ", n1)
3     print("number 2 is: ", n2)
4
5
6 # Calling the function and passing only one argument
7 print( "Passing only one argument" )
8 function(30)
9
10 # Now giving two arguments to the function
11 print( "Passing two arguments" )
12 function(50,30)
```

main.py




Output

```
Passing only one argument
number 1 is: 30
number 2 is: 20
Passing two arguments
number 1 is: 50
number 2 is: 30
```

2. Keyword arguments

main.py

Output



```
1 def function( n1, n2 ):
2     print("number 1 is: ", n1)
3     print("number 2 is: ", n2)
4
5 # Calling function and passing arguments without using keyword
6 print( "Without using keyword" )
7 function( 50, 30)
8
9 # Calling function and passing arguments using keyword
10 print( "With using keyword" )
11 function( n2 = 50, n1 = 30)
```

main.py

Output

```
Without using keyword
number 1 is: 50
number 2 is: 30
With using keyword
number 1 is: 30
number 2 is: 50
```

3. Required arguments

```
main.py Output
1 def function( n1, n2 ):
2     print("number 1 is: ", n1)
3     print("number 2 is: ", n2)
4
5     # Calling function and passing two arguments out of order, we need
      num1 to be 20 and num2 to be 30
6     print( "Passing out of order arguments" )
7     function( 30, 20 )
8
9     # Calling function and passing only one argument
10    print( "Passing only one argument" )
11    try:
12        function( 30 )
13    except:
14        print( "Function needs two positional arguments" )
```

```
main.py Output
Passing out of order arguments
number 1 is: 30
number 2 is: 20
Passing only one argument
Function needs two positional arguments
```

4. Variable-length arguments

```
main.py Output
1 def function( *args_list ):
2     ans = []
3     for l in args_list:
4         ans.append( l.upper() )
5     return ans
6 # Passing args arguments
7 object = function('Python', 'Functions', 'tutorial')
8 print( object )
9
10 # defining a function
11 def function( **kargs_list ):
12     ans = []
13     for key, value in kargs_list.items():
14         ans.append([key, value])
15     return ans
16 # Passing kwargs arguments
17 object = function(First = "Python", Second = "Functions", Third =
      "Tutorial")
18 print(object)
```

```
main.py Output
['PYTHON', 'FUNCTIONS', 'TUTORIAL']
[['First', 'Python'], ['Second', 'Functions'], ['Third', 'Tutorial']]
```

return Statement

```
main.py Output
1- def square( num ):
2     return num**2
3
4 # Calling function and passing arguments.
5 print( "With return statement" )
6 print( square( 52 ) )
7
8 # Defining a function without return statement
9- def square( num ):
10     num**2
11
12 # Calling function and passing arguments.
13 print( "Without return statement" )
14 print( square( 52 ) )
```

```
main.py Output
With return statement
2704
Without return statement
None
```

The Anonymous Functions

```
main.py Output
1 lambda_ = lambda argument1, argument2: argument1 + argument2;
2
3 # Calling the function and passing values
4 print( "Value of the function is : ", lambda_( 20, 30 ) )
5 print( "Value of the function is : ", lambda_( 40, 50 ) )
```

```
main.py Output
Value of the function is : 50
Value of the function is : 90
```

Scope and Lifetime of Variables

```
main.py Output
1- def number( ):
2     num = 50
3     print( "Value of num inside the function: ", num)
4 num = 10
5 number()
6 print( "Value of num outside the function:", num)
```


main.py

Output

Value of num inside the function: 50
Value of num outside the function: 10

Python Capability inside Another Capability

main.py

Output

```
1 def word():  
2     string = 'Python functions tutorial'  
3     x = 5  
4 def number():  
5     print( string )  
6     print( x )  
7     number()  
8 word()
```

main.py

Output

Python functions tutorial
5