Aim: Write a program to find the largest element among three Numbers.

https://onlinegdb.com/s_S888s90

Source Code: ***find_largest.py***

```python
# Input three numbers from the user
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))
num3 = int(input("Enter the third number: "))
# Initialize the largest number
if num1 >= num2 and num1 >= num3:
    largest = num1
elif num2 >= num1 and num2 >= num3:
    largest = num2
else:
    largest = num3
print("The largest number is:", largest)
```

Result: Successfully executed the largest element among three numbers.

Run: `D:\> python find_largest.py`

Output:

```
Enter the first number: 5
Enter the second number: 12
Enter the third number: 7
The largest number is: 12
```

Screenshot:

```python
# Input three numbers from the user
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))
num3 = int(input("Enter the third number: "))
# Initialize the largest number
if num1 >= num2 and num1 >= num3:
    largest = num1
elif num2 >= num1 and num2 >= num3:
    largest = num2
else:
    largest = num3
print("The largest number is:", largest)
```

input

```
Enter the first number: 5
Enter the second number: 12
Enter the third number: 7
The largest number is: 12
```

Aim: Write a Program to display all prime numbers within an interval

https://onlinegdb.com/ySc YK7p2V

Source Code: prime_numbers_interval.py

```python
start = int(input("Enter the start of the interval: "));
end = int(input("Enter the end of the interval: "));
if(start<=end):
    print(f"Prime numbers between {start} and {end} are:");
    while(start<=end):
        i=2;
        flag=True;
        if(start==1):
            start+=1;
            continue;
        else:
            while(i<start):
                if(start%i==0):
                    flag=False;
                    break;
                i+=1;
        if(flag == True):
            print(f"{start} is Prime Number");
        start+=1;
else:
    print("The start of the interval must be less than or equal to the end.");
```

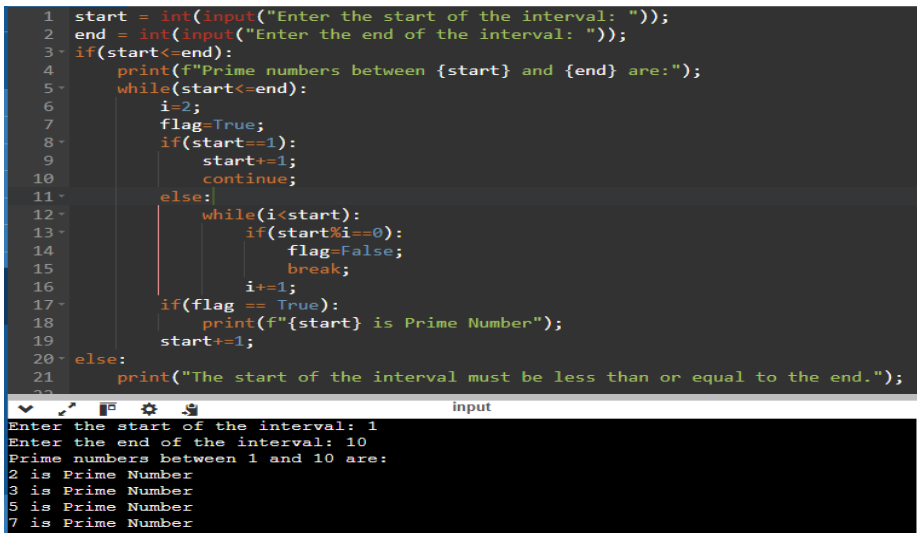
Result: Successfully executed the all prime numbers within an interval.

Run: D:\> python prime_numbers_interval.py

Output:-

```
Enter the start of the interval: 1
Enter the end of the interval: 10
Prime numbers between 1 and 10 are:
2 is Prime Number
3 is Prime Number
5 is Prime Number
7 is Prime Number
```

Screenshot:



Aim: Write a program to swap two numbers without using a temporary variable.

https://onlinegdb.com/LjNWDyxp3

Source Code: swap_without_temp.py

```python
# Taking A and B values for swap
a = int(input("Enter A value:"));
b = int(input("Enter B value:"));

# Before Swap
print(f"Before Swap:A={a}\tB={b}")

# Swaping without temporary variable
a, b = b, a
# After Swap
print(f"After Swap: A={a}\tB={b}")

```

Result: Successfully executed the swap two numbers without using a temporary variable.

Run:

D:\> python swap_without_temp.py

Output:

```
Enter A value:1
Enter B value:2
Before Swap:A=1 B=2
After Swap: A=2 B=1
```

Screenshot:

```python
# Taking A and B values for swap
a = int(input("Enter A value:"));
b = int(input("Enter B value:"));

# Before Swap
print(f"Before Swap:A={a}\tB={b}")

# Swaping without temporary variable
a, b = b, a
# After Swap
print(f"After Swap: A={a}\tB={b}")

```

```
Enter A value:1
Enter B value:2
Before Swap:A=1 B=2
After Swap: A=2 B=1
```

Aim: Demonstrate the following Operators in Python with suitable examples.

i) Arithmetic Operators
ii) Relational Operators
iii) Assignment Operators
iv) Logical Operators
v) Bit wise Operators
vi) Ternary Operator
vii) Membership Operators
viii) Identity Operators

https://onlinegdb.com/VK2ogEjv4

Source Code: Operators.py

```
# Arithmetic Operators

a = 10

b = 3

print(f"A={a}\nB={b}")

print("i) Arithmetic Operators:\n")

print("\tA+B:", a + b)

print("\tA-B:", a - b)

print("\tA*B:", a * b)

print("\tA Exponentiation B:", a ** b)

print("\t:A/B:", a / b)

print("\tA//B:", a // b)

print("\t:A%b:", a % b)



print("\nii) Relational Operators:\n")

a = 10

b = 3


print("\tA < B :", a < b)

print("\tA <= B :", a <= b)

print("\tA > B :", a > b)
```

```
print("\tA >= B :", a >= b)
print("\tA == B :", a >= b)
print("\tA != B :", a <= b)

# Assignment Operators
print("\niii) Assignment Operators\n")

# iii) a. Simple Assignment Operator
i=100
print(f"\ta. Simple Assignment Operator: i={i}")
# iii) b. Compound Assignment Operator
i+=1
print("\n\tb. Compound Assignment Operator:\n")
print("\t\ti += 1 :",i)
i-=1
print("\t\ti -= 1 :",i)
i*=2
print("\t\ti *= 2 :",i)
i**=2
print("\t\ti **= 2 :",i)
i/=2
print("\t\ti /= 2 :",i)
i//=2
print("\t\ti //= 2 :",i)
i%=2
print("\t\ti %= 2 :",i)

# parllel Assignment Operators
print("\tc. parallel Assignment Operators:-\n");
p=100
```

```
q=200
print(f"\t\tBefore : P={p},Q={q}")
p,q=q,p
print(f"\t\tAfter : P={p},Q={q}")
# Logical Operators
print("\niv) Logical Operators:-\n")
print("\t(1<2) and (1<2) :", (1<2) and (1<2))
print("\t(1<2) or (1>2) :", (1<2) or (1>2))
print("\tnot(1<2):", not(1<2))

# Bitwise Operators
print("\nv) Bitwise Operators:-\n")
print("\t7 & 2:", 7 & 2)
print("\t7 | 2 :", 7 | 2)
print("\t7 ^ 3 :", 7 ^ 2)
print("\t~7:", ~7)
print("\t7<<1:", 7 << 1)
print("\t7>>1:", 7 >> 1)

# Ternary Operator
print("\nvi) Ternary Operator:-\n")
a = 10
b = 5

result = "a is greater" if a > b else "b is greater or equal"
print("\t",result)

# Membership Operators
print("\nvii) Membership Operators:-\n")
list1 = [1, 2, 3, 4, 5]
```

```
print("\tMy List:",list1)

print("\t3 in list1:",3 in list1)

print("\t6 not in list1:",6 not in list1)


# viii) Identity Operators

print("\nviii) Identity Operators:-\n")

a = [1, 2, 3]

b = [1, 2, 3]


print("\tA:",a)

print("\tB:",b)


print("\ta is b : ",a is b)

print("\ta is not b :",a is not b)


c = a

print("\ta is c : ",a is c)

print("\ta is not c : ",a is not c)
```

Result: Successfully executed types of operator examples.

Run: D:\> python Operators.py

```
A=10
B=3
i) Arithmetic Operators:

        A+B: 13
        A-B: 7
        A*B: 30
        A Exponentiation B: 1000
        :A/B: 3.3333333333333335
        A//B: 3
        :A%b: 1

ii) Relational Operators:

        A < B : False
        A <= B : False
        A > B : True
        A >= B : True
        A == B : True
        A != B : False

iii) Assignment Operators

        a. Simple Assignment Operator: i=100

        b. Compound Assignment Operator:

                i += 1 : 101
                i -= 1 : 100
                i *= 2 : 200
                i **= 2 : 40000
                i /= 2 : 20000.0
                i //= 2 : 10000.0
                i %= 2 : 0.0
```

```
        c. parallel Assignment Operators:-

                Before : P=100,Q=200
                After : P=200,Q=100

iv) Logical Operators:-

        (1<2) and (1<2) : True
        (1<2) or (1>2) : True
        not(1<2): False

v) Bitwise Operators:-

        7 & 2: 2
        7 | 2 : 7
        7 ^ 3 : 5
        ~7: -8
        7<<1: 14
        7>>1: 3

vi) Ternary Operator:-

         a is greater

vii) Membership Operators:-

        My List: [1, 2, 3, 4, 5]
        3 in list1: True
        6 not in list1: True
```

```
viii) Identity Operators:-

        A: [1, 2, 3]
        B: [1, 2, 3]
        a is b :  False
        a is not b : True
        a is c :  True
        a is not c :  False
```

Output:

Aim: Write a program to add and multiply complex numbers

Source Code: complex_operations.py

```python
# Input complex numbers from user
real1 = float(input("Enter the real part of the first complex number: "))
imag1 = float(input("Enter the imaginary part of the first complex number: "))
real2 = float(input("Enter the real part of the second complex number: "))
imag2 = float(input("Enter the imaginary part of the second complex number: "))

# Create complex numbers
complex1 = complex(real1, imag1)
complex2 = complex(real2, imag2)

# Perform addition and multiplication
sum_result = complex1 + complex2
product_result = complex1 * complex2

# Print results
print(f"The sum of {complex1} and {complex2} is: {sum_result}")
print(f"The product of {complex1} and {complex2} is: {product_result}")
```

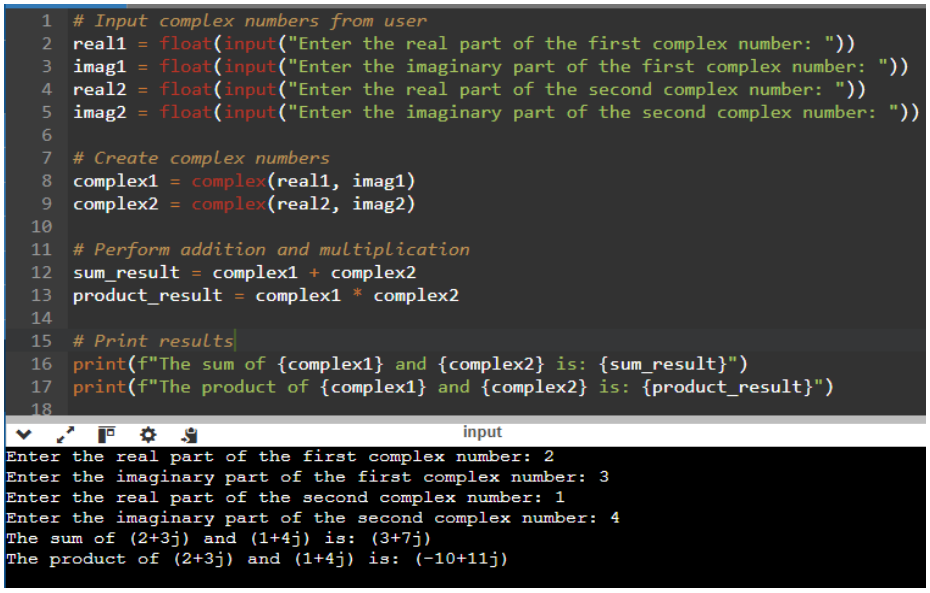
Result: Successfully executed the add and multiply complex numbers.

Run: D:\> python complex_operations.py

Output:

```
Enter the real part of the first complex number: 2
Enter the imaginary part of the first complex number: 3
Enter the real part of the second complex number: 1
Enter the imaginary part of the second complex number: 4
The sum of (2+3j) and (1+4j) is: (3+7j)
The product of (2+3j) and (1+4j) is: (-10+11j)
```

Screenshot:



6. Write a program to print multiplication table of a given number.

https://onlinegdb.com/7l93s5Ry_

Source Code: multiplication_table.py

```python
# Multiplication table of a given number
number = int(input("Enter a number to print its multiplication table: "))
# Print the multiplication table from 1 to 10
print(f"Multiplication table for {number}:")
for i in range(1, 11):
    result = number * i
    print(f"{number} x {i} = {result}")

```

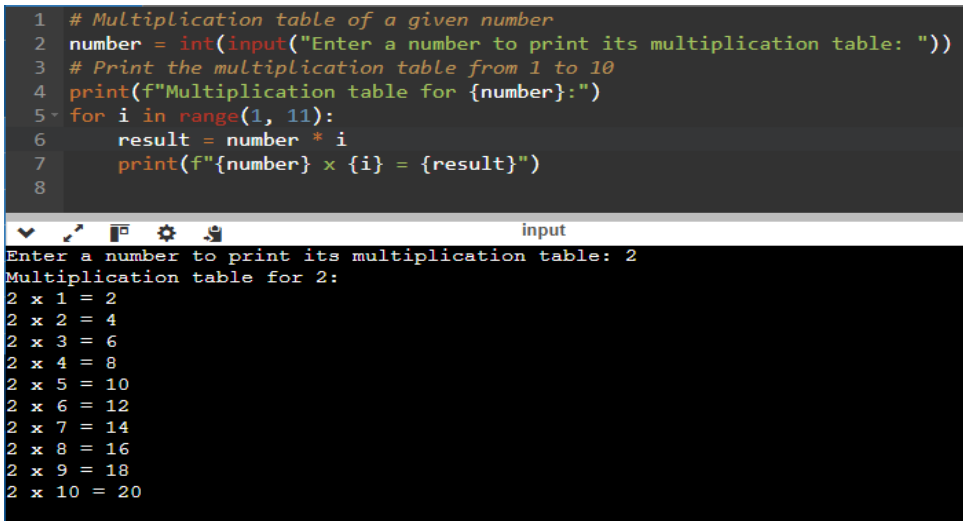
Result: Successfully executed the multiplication table of a given number

Run: D:\> python multiplication_table.py

Output:

```
Enter a number to print its multiplication table: 2
Multiplication table for 2:
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```

Screenshot:



7. Write a program to define a function with multiple return values.

https://onlinegdb.com/z4xe6pQkQ

Source Code: multiple_return_values_fun.py

```python
def calculate_sum_and_product(a, b):
    """Calculate the sum and product of two numbers."""
    total = a + b
    product = a * b
    return total, product

# Input numbers from the user
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

# Call the function and unpack the returned values
sum_result, product_result = calculate_sum_and_product(num1, num2)

# Print the results
print(f"The sum of {num1} and {num2} is: {sum_result}")
print(f"The product of {num1} and {num2} is: {product_result}")

```

input

```
Enter the first number: 2
Enter the second number: 3
The sum of 2 and 3 is: 5
The product of 2 and 3 is: 6
```

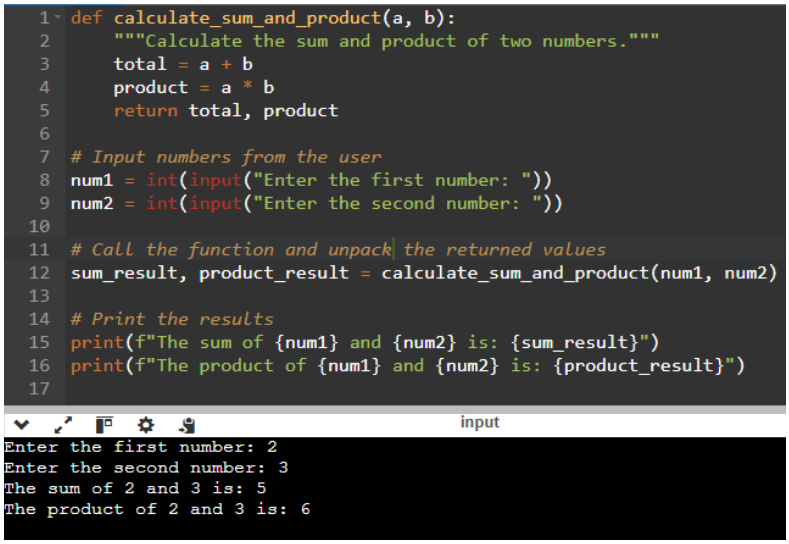
Result: Successfully executed the define a function with multiple return values

Run: D:\>python multiple_return_values_fun.py

Output:

```
Enter the first number: 2
Enter the second number: 3
The sum of 2 and 3 is: 5
The product of 2 and 3 is: 6
```

Screenshot:



Aim: Write a program to define a function using default arguments.

https://onlinegdb.com/r8S-D5ONmX

Source Code: default_args_function.py

```python
def greet(name="PEC", message="Welcome"):
    """
    Prints a greeting message.

    Parameters:
    - name (str): The name of the person to greet. Default is "PEC".
    - message (str): The greeting message. Default is "Welcome".
    """
    print(f"Hello, {name}! {message}")

# Calling the function without arguments
greet()

# Calling the function with one argument
greet("Alluri")

# Calling the function with both arguments
greet("Anil", "Good to see you!")

# Calling the function with default message but a custom name
greet(name="kumar")
```

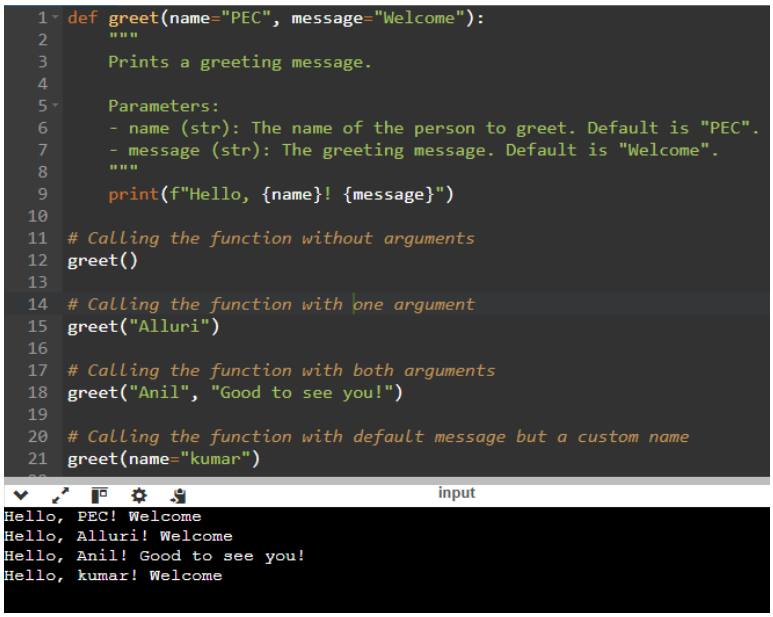
Result: Successfully executed the defining a function using default arguments.

Run: D:\> python Default_Args_Function.py

Output:

```
Hello, PEC! Welcome
Hello, Alluri! Welcome
Hello, Anil! Good to see you!
Hello, kumar! Welcome
```

Screenshot:



Aim: Write a program to find the length of the string without using any library functions

https://onlinegdb.com/-HNY_2mcqh

Source Code: string_length.py

```python
def string_length(my_name):
    """
    Returns the length of the input string without using library functions.

    Parameters:
    - s (str): The input string whose length is to be determined.

    Returns:
    - int: The length of the string.
    """
    count = 0
    for char in my_name:
        count += 1
    return count

# Reading Data from keyboard
my_name = input("Enter a your Name:")
my_name_length = string_length(my_name)
print(f"The length of the string is: {my_name_length}")
```

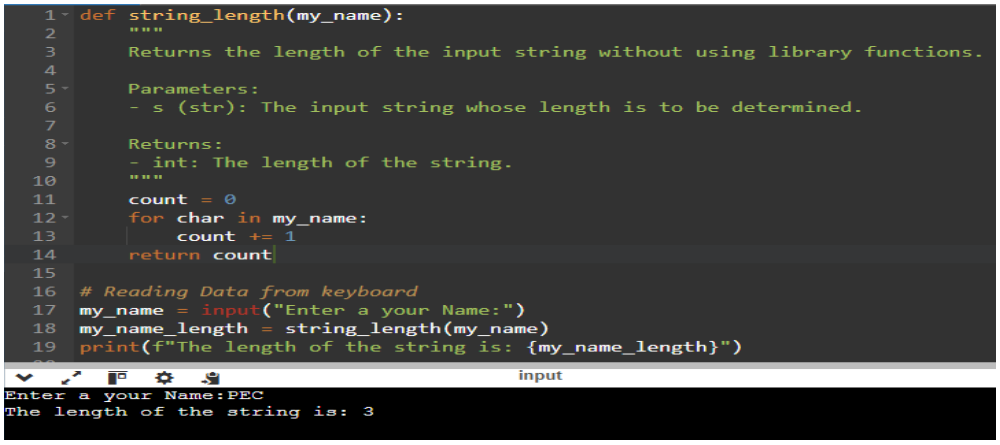
Result: Successfully executed the length of the string without using any library functions.

Run: D:\> python string_length.py

Output:

```
Enter a your Name:PEC
The length of the string is: 3
```

Screenshot:



Aim: Write a program to check if the substring is present in a given string or not.

https://onlinegdb.com/6pq06Wfgwx

Source Code: check_substring.py

```python
def isSubString(main_string, substring):
    """
    Check if `substring` is present in `main_string`.

    Args:
    - main_string (str): The string to be searched.
    - substring (str): The string to search for.

    Returns:
    - bool: True if `substring` is found in `main_string`, False otherwise.
    """
    return substring.lower() in main_string.lower()

# Input from the user
main_string = input("Enter the main string: ")
substring = input("Enter the substring to check: ")

# Check if substring is present in the main string
if isSubString(main_string, substring):
    print(f"The substring '{substring}' is present in the main string.")
else:
    print(f"The substring '{substring}' is not present in the main string.")

```

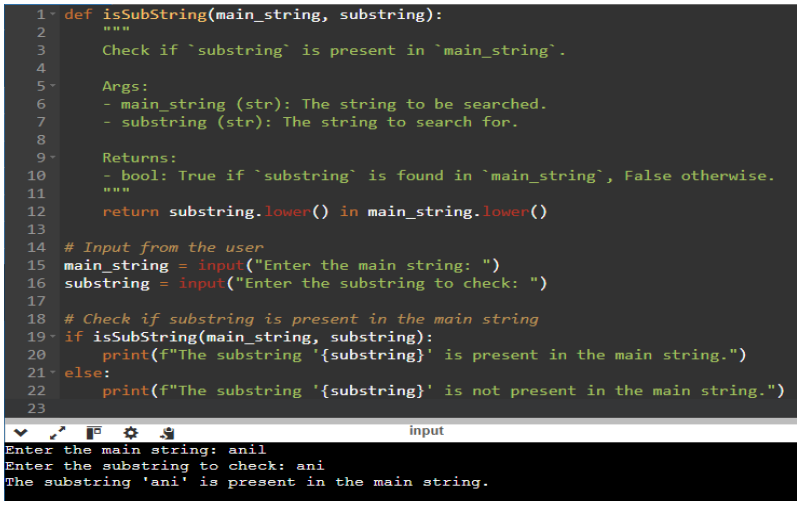
Result: Successfully executed the program if the substring is present in a given string or not.

Run: python check_substring.py

Output:

```
Enter the main string: Anil
Enter the substring to check: ani
The substring 'ani' is present in the main string.
```

Screenshot:



Aim: Write a program to perform the given operations on a list: (i) addition (ii) Insertion (iii) slicing

https://onlinegdb.com/ka2re0A8nA

Source Code: list_operations.py

```python
# Initialize a list with some elements
my_list = [1, 2, 3, 4, 5]
print("Initial List:", my_list)

# Addition of elements (appending to the end of the list)
def add_elements(elements):
    my_list.extend(elements)
    print("After Addition:", my_list)

# Insertion of elements at a specific index
def insert_element(index, element):
    if 0 <= index <= len(my_list):
        my_list.insert(index, element)
        print(f"After Insertion of {element} at index {index}:", my_list)
    else:
        print("Index out of bounds")

# Slicing of the list
def slice_list(start, end):
    sliced = my_list[start:end]
    print(f"Sliced List from index {start} to {end}:", sliced)

# Addition a list
add_elements([6, 7, 8])          # Adding elements [6, 7, 8]
insert_element(2, 'new')        # Inserting 'new' at index 2
slice_list(1, 4)                # Slicing from index 1 to 4
```

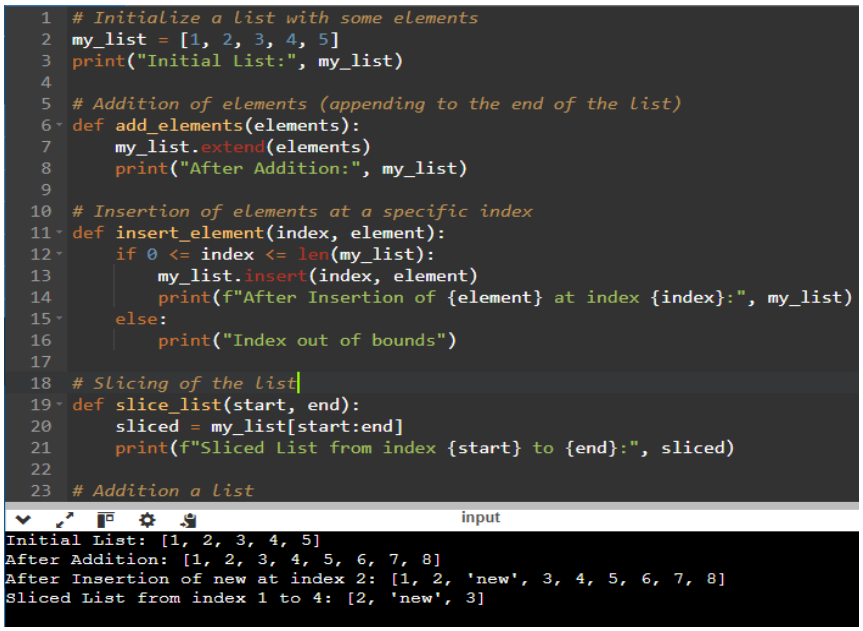
Result: Successfully executed the list operation on addition, insertion and slicing.

Run: D:\> python list_operations.py

Output:

```
Initial List: [1, 2, 3, 4, 5]
After Addition: [1, 2, 3, 4, 5, 6, 7, 8]
After Insertion of new at index 2: [1, 2, 'new', 3, 4, 5, 6, 7, 8]
Sliced List from index 1 to 4: [2, 'new', 3]
```

Screenshot:



Aim: Write a program to perform any 5 built-in functions by taking any list

https://onlinegdb.com/Lwx8ENgVRn

Source Code: list_5built_in_functions.py

```python
# Initialize a list with some elements
my_list = [1,2,3,4,5]
print("Initial List:", my_list)

# 1. append() - Add an element to the end of the list
my_list.append(6)
print("After append(6):", my_list)

# 2. extend() - Extend the list by appending elements from another iterable
my_list.extend([7, 8, 9, 10])
print("After extend([7, 8]):", my_list)

# 3. insert() - Insert an element at a specified position
my_list.insert(2, 'inserted')
print("After insert(2, 'inserted'):", my_list)

# 4. remove() - Remove the first occurrence of a specified value
my_list.remove('inserted')
print("After remove('inserted'):", my_list)

# 5. pop() - Remove and return an element at a specified position (default is the last element)
popped_element = my_list.pop()
print("After pop():", my_list)
print("Popped Element:", popped_element)
```

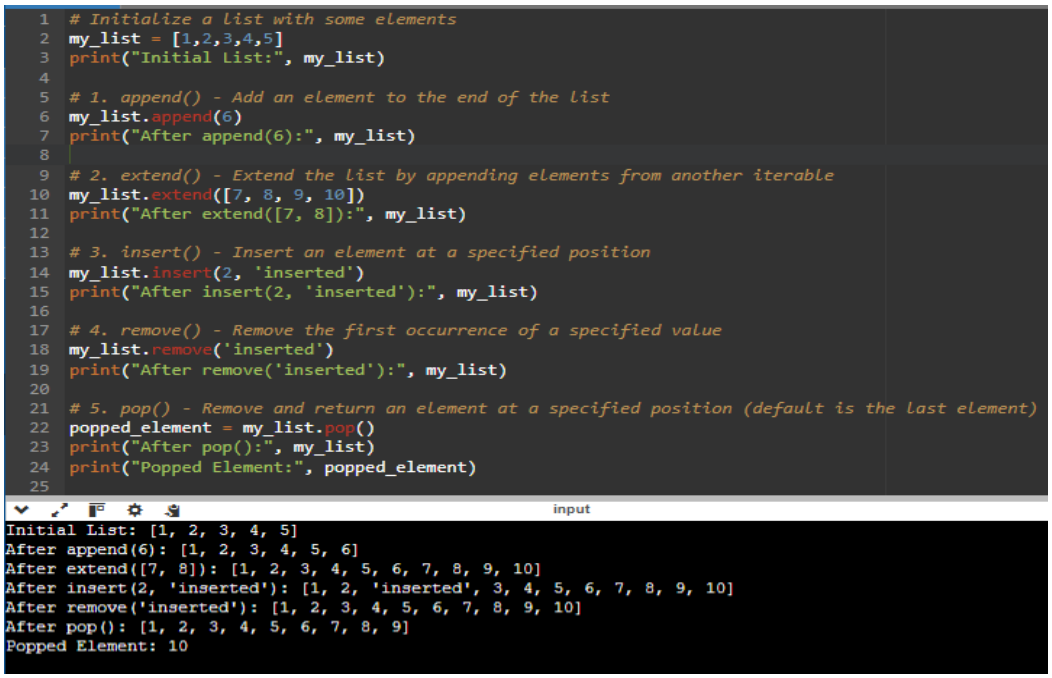
Run or Execute: D:\> python list_5built_in_functions.py

Result: Successfully executed the five built-in functions are append(v), extend(list), insert(i,v), remove(v), pop() of list.

Output:

```
Initial List: [1, 2, 3, 4, 5]
After append(6): [1, 2, 3, 4, 5, 6]
After extend([7, 8]): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
After insert(2, 'inserted'): [1, 2, 'inserted', 3, 4, 5, 6, 7, 8, 9, 10]
After remove('inserted'): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
After pop(): [1, 2, 3, 4, 5, 6, 7, 8, 9]
Popped Element: 10
```

Screenshot:



Aim: Write a program to create tuples (name, age, address, college) for at least two members and concatenate the tuples and print the concatenated tuples.

https://onlinegdb.com/b34Vma8zN

Source Code: concatenated_tuples.py

```python
# Define tuples for each member
member1 = ("Alice",26 , "123 Maple St", "Harvard University")
member2 = ("Bob", 30, "456 Oak Ave", "Stanford University")

# Concatenate the tuples
combined_tuples = member1 + member2

# Print the concatenated tuples
print("Concatenated Tuples:", combined_tuples)
```

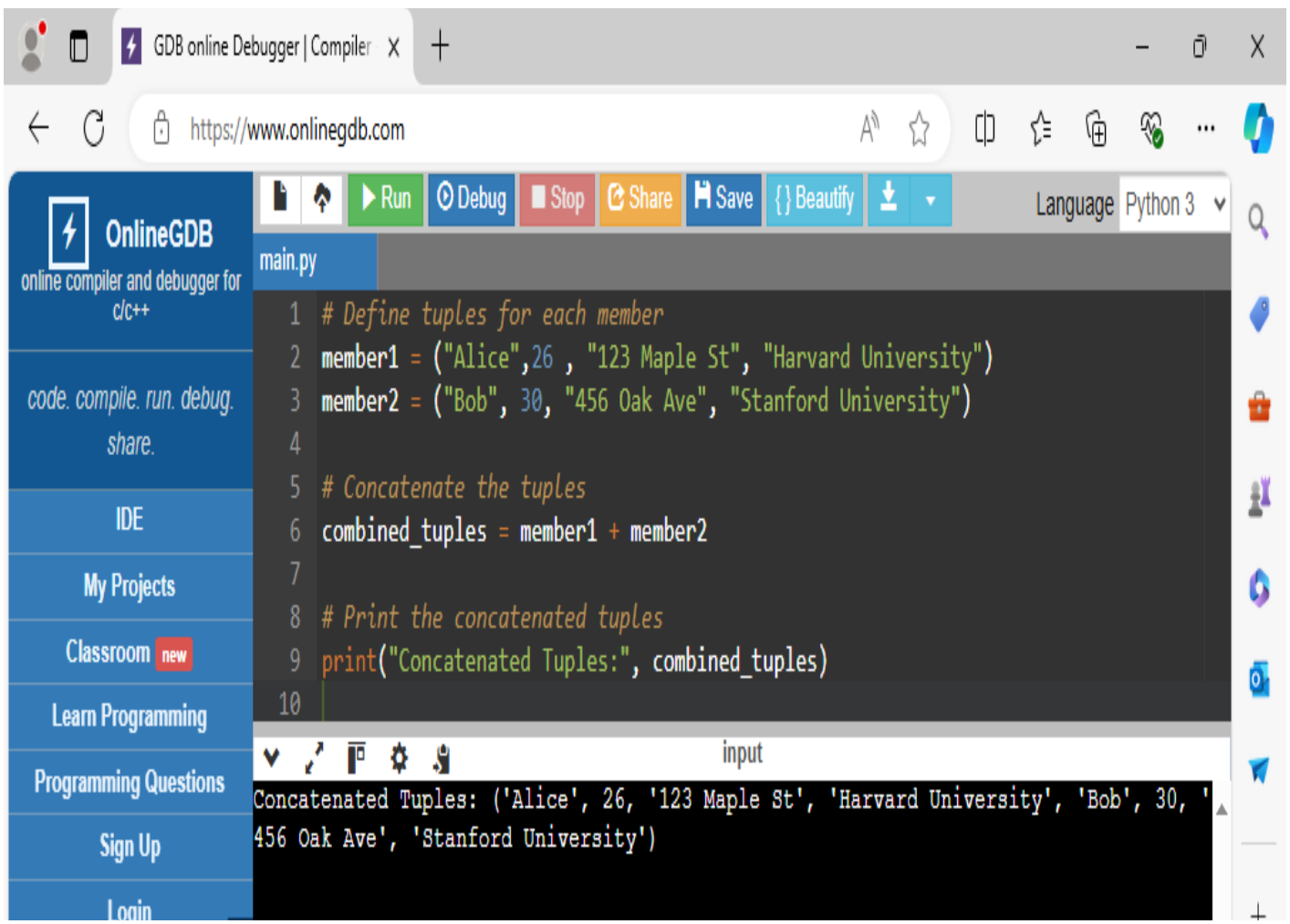
Run or Execute: D:\> python concatenated_tuples.py

Result: Successfully executed the concatenated two tuples.

Output:

```
Concatenated Tuples: ('Alice', 26, '123 Maple St', 'Harvard University', 'Bob', 30, '
456 Oak Ave', 'Stanford University')
```

Screenshot:



Aim: Write a program to count the number of vowels in a string (No control flow allowed).

https://onlinegdb.com/LG8VWrt_H

Source Code: number_of_vowels.py

```python
# Define a function to count vowels using list
# comprehensions and the `sum` function
def count_vowels(s):
    # Define a set of vowels
    vowels = 'aeiouAEIOU'
    # Use a list comprehension to filter out the vowels
    # and then count them using `len`
    return sum(1 for char in s if char in vowels)

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

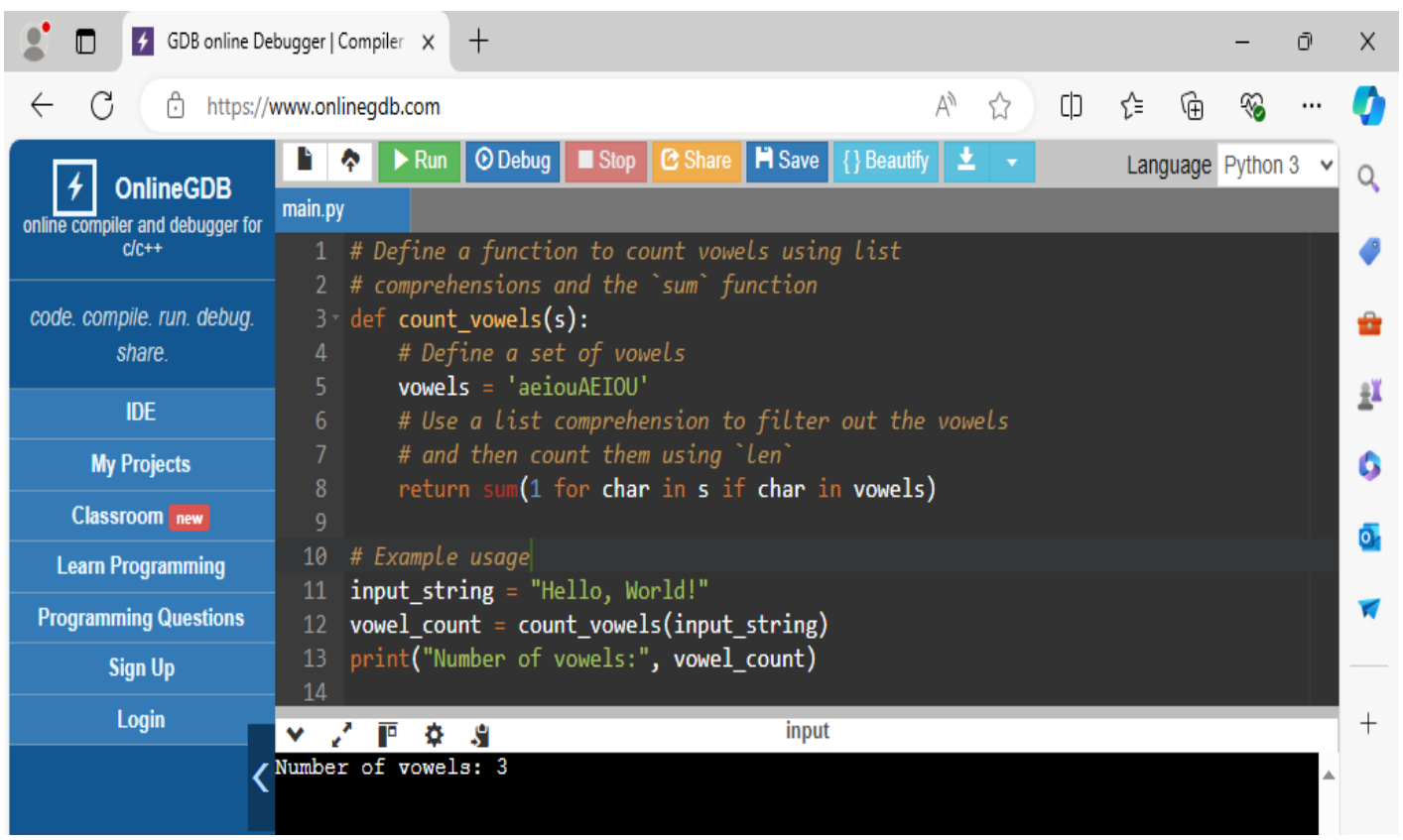
Run or Execute: D:\> python number_of_vowels.py

Result: Successfully counting the number of vowels in a string.

Output:

```
Number of vowels: 3
```

Screenshot:



Aim: Write a program to check if a given key exists in a dictionary or not

https://onlinegdb.com/clRgU_NJ-2

Source Code: key_exists_in_dictionary.py

```python
# Define a dictionary
my_dict = {
    'name': 'Alice',
    'age': 30,
    'address': '123 Maple St',
    'college': 'Harvard University'
}

# Function to check if a key exists in the dictionary
def key_exists(dictionary, key):
    return key in dictionary

# Example usage
key_to_check = 'age'
exists = key_exists(my_dict, key_to_check)

print(f"Does the key '{key_to_check}' exist in the dictionary? {exists}")

# Check for a key that does not exist
key_to_check = 'phone'
exists = key_exists(my_dict, key_to_check)

print(f"Does the key '{key_to_check}' exist in the dictionary? {exists}")
```

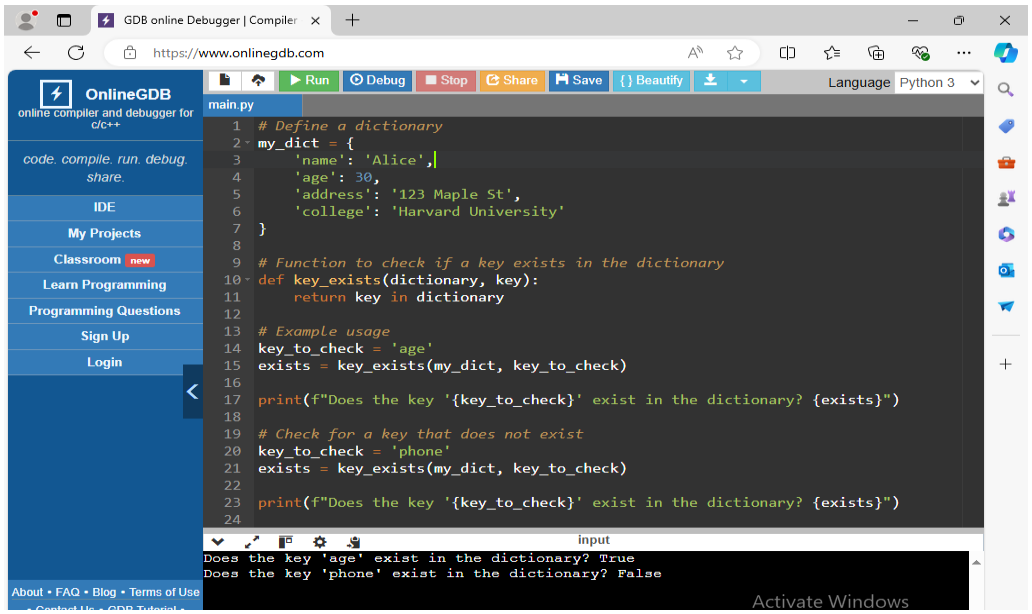
Run or Execute: D:\> python key_exists_in_dictionary.py

Result: Successfully executed to checked if a key exists in a dictionary or not.

Output:

```
Does the key 'age' exist in the dictionary? True
Does the key 'phone' exist in the dictionary? False
```

Screenshot:



Aim: Write a program to add a new key-value pair to an existing dictionary.

https://onlinegdb.com/OGpAm0I_c

Source Code: add_new_pair_in_dictionary.py

```python
# Define an existing dictionary
my_dict = {
    'name': 'Alice',
    'age': 30,
    'address': '123 Maple St',
    'college': 'Harvard University'
}

# Function to add a new key-value pair to the dictionary
def add_key_value(dictionary, key, value):
    dictionary[key] = value

# Example usage
new_key = 'phone'
new_value = '555-1234'

# Adding the new key-value pair to the dictionary
add_key_value(my_dict, new_key, new_value)

# Print the updated dictionary
print("Updated dictionary:", my_dict)
```

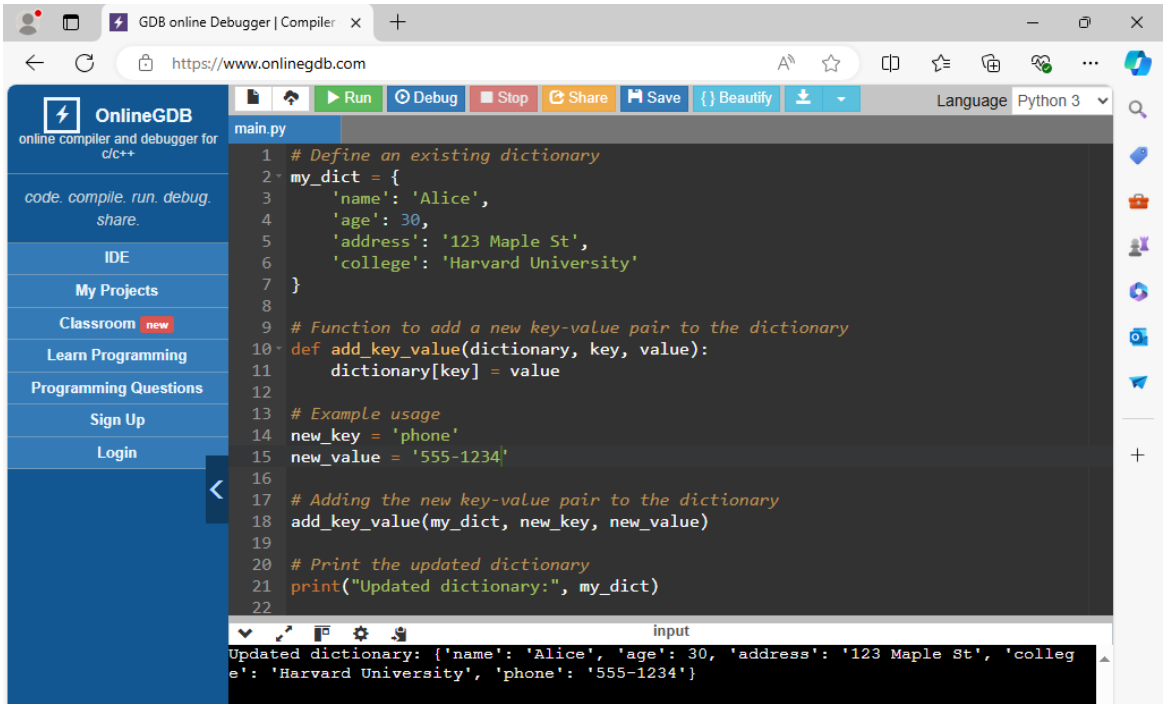
Run or Execute: D:\> python add_new_pair_in_dictionary.py

Result: Successfully executed to add a new pair into an existing dictionary.

Output:

```
Updated dictionary: {'name': 'Alice', 'age': 30, 'address': '123 Maple St', 'colleg
e': 'Harvard University', 'phone': '555-1234'}
```

Screenshot:



Aim: Write a program to sum all the items in a given dictionary

https://onlinegdb.com/hbs-5PvxoX

Source Code: sum_all_items_in_dictionary.py

```python
# Define a dictionary with numeric values
my_dict = {
    'item1': 10,
    'item2': 20,
    'item3': 30,
    'item4': 40,
    'item5': 50
}

# Function to sum all the values in the dictionary
def sum_dictionary_values(dictionary):
    # Use the sum function to add up all the values
    return sum(dictionary.values())

# Calculate the sum of all values
total_sum = sum_dictionary_values(my_dict)

# Print the result
print("The sum of all values in the dictionary is:",
```

Run or Execute: D:\> python sum_all_items_in_dictionary.py

Result: Successfully executed the sum of all item values in a dictionary.

Output:

```
The sum of all values in the dictionary is: 150
```

Screenshot: