ASSIGNMENT - 02 (List, Tuple)

Solution/Ans by - Pranav Rode(29)

1. What is a List?

In [ ]:
```
In Python, a list is a built-in data structure that represents an ordered collection of elements.
It is one of the most commonly used data structures and provides flexibility for storing and manipulating data.

Here are some key characteristics of lists in Python:

1. Order: Lists maintain the order of elements, meaning the elements are stored in a specific sequence.
    The order in which elements are added is preserved.

2. Mutable: Lists are mutable, which means you can change, add, or remove elements after the list is created.
    This allows for dynamic updates to the list.

3. Heterogeneous Elements: Lists can contain elements of different data types, such as
    integers, floats, strings, or even other lists.
    This flexibility allows you to store diverse types of data in a single list.

4. Indexed Access: Elements in a list are accessed using an index, starting from 0 for the first element.
    You can retrieve or modify specific elements by referencing their index.

5. Length: Lists can vary in size, and you can determine the number of elements
    in a list using the built-in `len()` function.

Lists in Python are created using square brackets `[]`, and elements within the list are separated by commas.

Example:

fruits = ["apple", "banana", "cherry"]

In the example above, `fruits` is a list containing three elements: "apple", "banana", and "cherry".
    The order of the elements is preserved, and each element can be accessed by its index.

Lists provide a versatile way to store and manipulate collections of data in Python,
making them useful for a wide range of programming tasks.
```

2. What is a Tuple?

In [ ]:
```
A tuple is an ordered collection of elements in Python. It is similar to a list, but with one key difference:
tuples are immutable, meaning their elements cannot be changed once they are created.
Tuples are created using parentheses `()` or the `tuple()` function, although
the parentheses are often omitted if the tuple has a single element.

Tuples have the following characteristics:
1. Order: Like lists, tuples maintain the order of elements,
    preserving the sequence in which they were added.
2. Immutable: Once a tuple is created, its elements cannot be modified, added, or removed.
    However, you can create a new tuple by concatenating or slicing existing tuples.
3. Heterogeneous Elements: Tuples can contain elements of different data types,
    such as integers, floats, strings, or even other tuples.
4. Indexed Access: Elements in a tuple can be accessed using an index,
    starting from 0 for the first element.
5. Length: The number of elements in a tuple can be determined using the built-in `len()` function.

Tuples are commonly used in scenarios where the data needs to be protected from accidental modification.
For example, you might use a tuple to represent a point in a 2D space, where the coordinates should remain constant.
Additionally, tuples are often used to return multiple values from a function.

It's important to note that due to their immutability, tuples are more memory-efficient and
offer better performance compared to lists.
If you need a collection that can be modified, you should use a list instead.
```

3. What is the difference between List and Tuple?

	List	Tuple
Mutability	Mutable	Immutable
Syntax	Square brackets `[]`	Parentheses `()`
Example	`[1, 2, 3]`	`(1, 2, 3)`
Modifiability	Elements can be added, removed, or modified.	Elements cannot be added, removed, or modified.
Usage	Suitable for storing dynamic data or when elements need to be changed.	Suitable for representing fixed data or when immutability is desired.
Performance	Slightly slower due to potential resizing and copying operations when modifying.	Slightly faster and more memory-efficient.
Common Operations	`append()`, `extend()`, `insert()`, `remove()`, `pop()`, `index()`, `count()`, `sort()`, `reverse()`, etc.	`index()`, `count()`, slicing, iteration.
Use Cases	When flexibility and dynamic changes to the data are required.	When data needs to remain constant and protected from accidental modifications.
Syntax Flexibility	Can hold zero or multiple elements.	Can hold zero or more elements, but a trailing comma is needed for a single-element tuple.

4. Python Program to find the largest element in the list

In [1]:
```
numbers = [12, 45, 67, 23, 9, 56]

largest_number = numbers[0]  # Assume the first element is the largest

for num in numbers:
    if num > largest_number:
        largest_number = num

print("The largest number is:", largest_number)
```

```
The largest number is: 67
```

5. Python program to interchange first and last elements in a list.

In [10]:
```
numbers = [12, 45, 67, 23, 9, 56]
print('List before interchange =',numbers)
numbers[0], numbers[-1] = numbers[-1], numbers[0]
print('List after interchange =',numbers)
```

```
List before interchange = [12, 45, 67, 23, 9, 56]
List after interchange = [56, 45, 67, 23, 9, 12]
```

6. Python program to swap two elements in a list

In [12]:
```
lst = [12, 45, 67, 23, 9, 54]
index1 = int(input("Enter position = "))
index2 = int(input("Enter position = "))
print("Unswapped List:", lst)

lst[index1], lst[index2] = lst[index2], lst[index1]
swapped_list = lst
print("Swapped List:", swapped_list)
```

```
Enter position = 2
Enter position = 5
Unswapped List: [12, 45, 67, 23, 9, 54]
Swapped List: [12, 45, 54, 23, 9, 67]
```

7. Python program to Reverse a List

In [1]:
```
numbers = [1, 2, 3, 4, 5, 6]
print('Reversed list :',numbers[::-1])
```

```
Reversed list : [6, 5, 4, 3, 2, 1]
```

In [3]:
```
def reverse_list(lst):
    start_index = 0
    end_index = len(lst) - 1

    while start_index < end_index:
        lst[start_index], lst[end_index] = lst[end_index], lst[start_index]
        start_index += 1
        end_index -= 1

    return lst
```

```
# Test the function
numbers = [1, 2, 3, 4, 5, 6]
reversed_list = reverse_list(numbers)
print("Reversed List:", reversed_list)
```

```
Reversed List: [6, 5, 4, 3, 2, 1]
```

8. Python program to count occurrences of an element in a list

In [20]:
```
lst = [1, 2,'a', 3, 4, 4, 5, 4, 5, 7,'a','b', 'a', 'a']
element = eval(input('Enter element from given list = '))
count = 0
for item in lst:
    if item == element:
        count += 1
print(f'Count of {element} is = ', count)
```

```
Enter element from given list = 'a'
Count of a is =  4
```

9. Python program to find the sum of elements in a list

In [26]:
```
lst = [12, 45, 67, 23, 9, 50]
total = 0
for i in lst:
    total = total + i
print('Sum is =', total)
print('--------------')
# Other way using sum()
print('Sum is =', sum(lst))
```

```
Sum is = 206
--------------
Sum is = 206
```

10. Python program to Multiply all numbers in the list

In [27]:
```
lst = [2, 3, 4, 5, 6]
mul = 1
for i in lst:
    mul = mul * i
print('Multiplication is =', mul)
```

```
Multiplication is = 720
```

11. What are the ways to find the length of a list

In [ ]:
```
There are several ways to find the length of a list in Python. Here are some common approaches:
```

In [3]:
```
# Method 1: Using the len() function
my_list = [1, 2, 3, 4, 5]
length = len(my_list)
print(length)
```

```
5
```

In [6]:
```
# Method 2: Using a loop to count the elements
my_list = [1, 2, 3, 4, 5]
count = 0
for _ in my_list:
    count += 1
print(count)
```

```
5
```

In [7]:
```
# Method 3: Using list comprehension and the sum() function

my_list = [1, 2, 3, 4, 5]
length = sum(1 for _ in my_list)
print(length)
```

```
5
```

In [1]:
```
# Method 4: Using enumerate function
list1 = [1, 4, 5, 7, 8]
s = 0
for i, a in enumerate(list1):
    s += 1
print(s)
```

```
5
```

12. Python program to find the smallest and largest number in a list (Without min-max function)

In [2]:
```python
numbers = [5, 2, 9, 1, 7, 3]

# Finding the smallest number without using min()
smallest = numbers[0]
for num in numbers:
    if num < smallest:
        smallest = num

# Finding the largest number without using max()
largest = numbers[0]
for num in numbers:
    if num > largest:
        largest = num

print(f"The smallest number is: {smallest}")
print(f"The largest number is: {largest}")
```

```
The smallest number is: 1
The largest number is: 9
```

13. Python Program to find the area of a circle

In [3]:
```python
import math

def circle_area(radius):
    area = math.pi * radius**2
    return area

# Example usage:
radius = 5
area_of_circle = circle_area(radius)
print(f"The area of the circle with radius {radius} is: {area_of_circle}")
```

```
The area of the circle with radius 5 is: 78.53981633974483
```

14. Take inputs from the user to make a list. Again take one input from the user and search it in the list and delete that element, if found. Iterate over a list using for loop.

In [4]:
```python
# Take inputs from the user to create a list
user_list = input("Enter elements of the list separated by spaces: ").split()

# Take input from the user to search and delete
element_to_delete = input("Enter the element to search and delete: ")

# Initialize a flag to track if the element was found and deleted
element_found = False

# Iterate over the list using a for loop
for item in user_list:
    if item == element_to_delete:
        user_list.remove(item)
        element_found = True
        break  # Exit the loop after deleting the first occurrence

if element_found:
    print(f"'{element_to_delete}' found and deleted.")
    print("Updated list:", user_list)
else:
    print(f"'{element_to_delete}' not found in the list.")
```

```
Enter elements of the list separated by spaces: 8 4 5 6 9 8
Enter the element to search and delete: 3
'3' not found in the list.
```

15. You are given a list of integer elements. Make a new list that will store a square of elements of the previous list. (With and without list comprehension)

i. Input_list = [2,5,6,12]

ii. Output_list = [4,25,36,144]

In [11]:
```python
# Using list comprehension
Input_list = [2,5,6,12]

Output_list_comp = [i**2 for i in Input_list ]
print('Using list comprehension:',Output_list_comp)
print('--------')

Output_list = []
for i in Input_list:
    Output_list.append(i**2)
print('Using For Loop:',Output_list)
```

```
Using list comprehension: [4, 25, 36, 144]
--------
Using For Loop: [4, 25, 36, 144]
```

16. WAP to create two lists, one containing all even numbers and the other containing all odd numbers between 0 to 151

In [18]:
```python
even_numbers = []
odd_numbers = []

for number in range(0, 152):
    if number % 2 == 0:
        even_numbers.append(number)
    else:
        odd_numbers.append(number)

print("Even numbers:", even_numbers)
print("----------------------------------------------------------------" * 2)
print("Odd numbers:", odd_numbers)
```

```
Even numbers: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 5
8, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116,
118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150]
----------------------------------------------------------------------------------------------------------------------------
Odd numbers: [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59,
61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103, 105, 107, 109, 111, 113, 115, 117, 11
9, 121, 123, 125, 127, 129, 131, 133, 135, 137, 139, 141, 143, 145, 147, 149, 151]
```

17. Python program to Count Even and Odd numbers in a List

In [34]:
```python
import random

l = [random.randint(1, 50) for i in range(10)]
print('List:',l)
print("----------------------------------------------------------------" * 2)

even_numbers = []
odd_numbers = []

even_count = 0
odd_count = 0

for number in l:
    if number % 2 == 0:
        even_count += 1
        even_numbers.append(number)
    else:
        odd_count += 1
        odd_numbers.append(number)

print("Even numbers:", even_numbers)
print('Even Count:', even_count)
print("----------------------------------------------------------------" * 2)
print("Odd numbers:", odd_numbers)
print('Odd Count:', odd_count)
```

```
List: [26, 8, 1, 23, 17, 20, 30, 38, 13, 48]
----------------------------------------------------------------------------------------------------------------------------
Even numbers: [26, 8, 20, 30, 38, 48]
Even Count: 6
----------------------------------------------------------------------------------------------------------------------------
Odd numbers: [1, 23, 17, 13]
Odd Count: 4
```

18. WAP to make new lists, containing only numbers which are divisible by 4, 6, 8, 10, 3, 5, 7, and 9 in separate lists for range(0,151)

In [51]:
```python
nums = [4, 6, 8, 10, 3, 5, 7, 9]
lists_dict = {}
for i in nums:

    list_name = f"Divisible_by_{i}"
    lists_dict[list_name] = []
    for number in range(0, 151):
        if number % i == 0:
            lists_dict[list_name].append(number)
#print(lists_dict)

for key, value in lists_dict.items():
    print(f"{key}: {value}")
    print()
```

```
Divisible_by_4: [0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92, 96, 100, 104, 108,
112, 116, 120, 124, 128, 132, 136, 140, 144, 148]

Divisible_by_6: [0, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90, 96, 102, 108, 114, 120, 126, 132, 138, 144, 150]

Divisible_by_8: [0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, 128, 136, 144]

Divisible_by_10: [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]

Divisible_by_3: [0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84,
87, 90, 93, 96, 99, 102, 105, 108, 111, 114, 117, 120, 123, 126, 129, 132, 135, 138, 141, 144, 147, 150]

Divisible_by_5: [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 13
0, 135, 140, 145, 150]

Divisible_by_7: [0, 7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98, 105, 112, 119, 126, 133, 140, 147]

Divisible_by_9: [0, 9, 18, 27, 36, 45, 54, 63, 72, 81, 90, 99, 108, 117, 126, 135, 144]
```

19. From a list containing ints, strings, and floats, make three lists to store them separately.

In [52]:
```python
mixed_list = [1, 'hello', 3.14, 'world', 42, 2.71, 'python']

int_list = []
float_list = []
str_list = []

for item in mixed_list:
    if isinstance(item, int):
        int_list.append(item)
    elif isinstance(item, float):
        float_list.append(item)
    elif isinstance(item, str):
        str_list.append(item)

print("Integer list:", int_list)
print("Float list:", float_list)
print("String list:", str_list)
```

```
Integer list: [1, 42]
Float list: [3.14, 2.71]
String list: ['hello', 'world', 'python']
```

20. What's The Difference Between The Python append() and extend() Methods?

In [ ]:
```
Both append() and extend() are list methods in Python,
but they are used for slightly different purposes when
it comes to adding elements to a list.

append():
The append() method is used to add a single element to the end of a list.
It takes one argument, which is the element you want to add to the list.
After using append(), the list will have one more element than before.
```

In [53]:
```python
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)  # Output: [1, 2, 3, 4]
```

```
[1, 2, 3, 4]
```

In [ ]:
```
extend():
The extend() method is used to add elements from an iterable (like a list, tuple, or string)
to the end of the list. It essentially concatenates the elements of the iterable to the list.
It takes one argument, which is the iterable you want to add. After using extend(), the
list will be extended with the elements from the iterable.
In contrast, using extend() with an iterable will add the individual elements of
the iterable to the list, not the entire iterable itself.
```

In [54]:
```python
my_list = [1, 2, 3]
my_list.extend([4, 5, 6])
print(my_list)  # Output: [1, 2, 3, 4, 5, 6]
```

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

In [ ]:
```
In summary:

append() adds a single element to the end of the list.
extend() adds the elements of an iterable to the end of the list.
```

21. Write a Python program to append a list to the second list

In [4]:
```python
def append_list(list1, list2):
    """Appends the second list to the first list."""
    list1.extend(list2)
    return list1
```

```python
if __name__ == "__main__":
    list1 = ["a", "b", "c"]
    list2 = ["d", "e", "f"]
    print("The original lists are:")
    print(list1)
    print(list2)

# Append list2 to list1
list1 = append_list(list1, list2)
print("The new list is:")
print(list1)
```

```
The original lists are:
['a', 'b', 'c']
['d', 'e', 'f']
The new list is:
['a', 'b', 'c', 'd', 'e', 'f']
```

In [5]:
```python
list1 = [1, 2, 3]
list2 = [4, 5, 6]

list2.extend(list1)  # Append elements of list1 to list2

print("List 1:", list1)
print("List 2:", list2)
```

```
List 1: [1, 2, 3]
List 2: [4, 5, 6, 1, 2, 3]
```

22. Write a Python program to find the third-largest number in a list

In [39]:
```python
# Creating Example list
import random
l = [random.randint(1, 30) for i in range(5)]
print("Created List:",l)
print("------------------------------")

#---------------------------------------------------------------
# Using 3 varaibles
def third_largest(numbers):
    if len(numbers) < 3:
        return "List should contain at least three numbers"

    first_largest = second_largest = third_largest = float('-inf')

    for num in numbers:
        if num > first_largest:
            third_largest = second_largest
            second_largest = first_largest
            first_largest = num
        elif second_largest < num < first_largest:
            third_largest = second_largest
            second_largest = num
        elif third_largest < num < second_largest:
            third_largest = num

    return third_largest


#---------------------------------------------------------------
third_largest_num = third_largest(l)
print("Third-largest number:", third_largest_num)
```

```
Created List: [16, 11, 27, 7, 5]
------------------------------
Third-largest number: 11
```

In [40]:
```python
# Using Bubble Sort
def bubble_sort(numbers):
    n = len(numbers)
    for i in range(n):
        swapped = False
        for j in range(0, n-i-1):
            if numbers[j] > numbers[j+1]:
                numbers[j], numbers[j+1] = numbers[j+1], numbers[j]
                swapped = True
        if not swapped:
            break

bubble_sort(l)
print("Using Bubble Sort :")
print("Sorted list:", l)
print("Third-largest number:", l[-3])
```

```
Using Bubble Sort :
Sorted list: [5, 7, 11, 16, 27]
Third-largest number: 11
```

23. Write a Python program to get the frequency of the elements in a list.

In [41]:
```
print('Using Dictionary')
# Using Dictionary---------------------------------
def get_element_frequency(lst):
    frequency = {}

    for element in lst:
        if element in frequency:
            frequency[element] += 1
        else:
            frequency[element] = 1

    return frequency

# Example list
my_list = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]

element_frequency = get_element_frequency(my_list)
print("Dict with Counts:",element_frequency)
for element, freq in element_frequency.items():
    print(f"Element {element} appears {freq} times")
```

```
Using Dictionary
Dict with Counts: {1: 1, 2: 2, 3: 3, 4: 4}
Element 1 appears 1 times
Element 2 appears 2 times
Element 3 appears 3 times
Element 4 appears 4 times
```

In [42]:
```
# Without Using Dictionary---------------------------------
print('Without Using Dictionary')
def get_element_frequency(lst):
    unique_elements = set(lst)

    for element in unique_elements:
        count = 0
        for num in lst:
            if num == element:
                count += 1
        print(f"Element {element} appears {count} times")

# Example list
my_list = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]

get_element_frequency(my_list)
```

```
Without Using Dictionary
Element 1 appears 1 times
Element 2 appears 2 times
Element 3 appears 3 times
Element 4 appears 4 times
```

24. Write a Python program to check whether a list contains a sublist

In [44]:
```
def contains_sublist(main_list, sublist):
    n = len(sublist)

    for i in range(len(main_list) - n + 1):
        if main_list[i:i + n] == sublist:
            return True

    return False

# Example lists
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
sub_list1 = [3, 4, 5]

list2 = [10, 20, 30, 40, 50]
sub_list2 = [30, 50]

print("List 1 contains sublist 1:", contains_sublist(list1, sub_list1))
print("List 2 contains sublist 2:", contains_sublist(list2, sub_list2))
```

```
List 1 contains sublist 1: True
List 2 contains sublist 2: False
```

25. Write a Python program to generate all sublists of a list

In [48]:
```
def generate_sublists(input_list):
    sublists = []
    n = len(input_list)

    for length in range(1, n + 1):
        for start in range(n - length + 1):
            sublist = input_list[start:start + length]
            sublists.append(sublist)
```

```
    return sublists

# Example list
my_list = [1, 2, 3]

all_sublists = generate_sublists(my_list)
print("All sublists:", all_sublists)
```

```
All sublists: [[1], [2], [3], [1, 2], [2, 3], [1, 2, 3]]
```

In [51]:
```
def generate_sublists(input_list, index=0, current=[]):
    if index == len(input_list):
        return [current]

    sublists = []
    sublists.extend(generate_sublists(input_list, index + 1, current))
    sublists.extend(generate_sublists(input_list, index + 1, current + [input_list[index]]))

    return sublists

# Example list
my_list = [1, 2, 3]

all_sublists = generate_sublists(my_list)
print("All sublists:", all_sublists)
```

```
All sublists: [[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]
```

26. Write a Python program to find common items from two lists

In [30]:
```
# Creating two lists using random library
import random
l1 = [random.randint(1,40) for i in range(10)]
l2 = [random.randint(1,40) for i in range(10)]
print("l1:",l1)
print("l2:",l2)
```

```
l1: [15, 37, 16, 21, 22, 1, 33, 30, 11, 16]
l2: [8, 15, 34, 1, 35, 25, 27, 21, 19, 32]
```

In [31]:
```
# Using For Loop & if statement
common = []
for i in l1:
    for j in l2:
        if i == j:
            common.append(i)
print("Common Items:",list(set(common) )  )
```

```
Common Items: [1, 21, 15]
```

In [32]:
```
# Using List Comprehension

print("l1:",l1)
print("l2:",l2)

common = [element for element in l2 if element in l1]

# Print the common elements
print("Common Items:", list(set(common)))
```

```
l1: [15, 37, 16, 21, 22, 1, 33, 30, 11, 16]
l2: [8, 15, 34, 1, 35, 25, 27, 21, 19, 32]
Common Items: [1, 21, 15]
```

27. How to flatten a list in python?

In [42]:
```
# Using List Comprehension
nested_list = [[1, 2, 3], [4, 5], [6, 7, 8]]
flat_list = [i for j in nested_list for i in j]
print(flat_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

In [43]:
```
# Using For Loop
nested_list = [[1, 2, 3], [4, 5], [6, 7, 8]]
flat_list = []
for j in nested_list:
    for i in j:
        flat_list.append(i)
print(flat_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

28. How to sort a list in ascending and descending order without using the sort function?

In [67]:
```
# Generate a list of unique random integers between 1 and 20

import random
```

```
my_list = random.sample(range(1, 20), 6)
print("Unsorted list:",my_list)
```

```
Unsorted list: [5, 12, 11, 15, 7, 17]
```

In [68]:
```
# For Ascending Order

def sort_asc(arr):
    n = len(arr)

    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j] # Swapping

# Driver Code:
sort_asc(my_list)
print("Ascending Sorted List:", my_list)
```

```
Ascending Sorted List: [5, 7, 11, 12, 15, 17]
```

In [69]:
```
# For Descending Order

def sort_asc(arr):
    n = len(arr)

    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] < arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j] # Swapping

# Driver Code:
sort_asc(my_list)
print("Descending Sorted List:", my_list)
```

```
Descending Sorted List: [17, 15, 12, 11, 7, 5]
```

29. How to sort a tuple?

In [70]:
```
# Create a tuple
my_tuple = (4, 2, 7, 1, 9, 5)

# Convert the tuple to a list
my_list = list(my_tuple)

# Sort the list
my_list.sort()

# Convert the sorted list back to a tuple
sorted_tuple = tuple(my_list)

print("Original Tuple:", my_tuple)
print("Sorted Tuple:", sorted_tuple)
```

```
Original Tuple: (4, 2, 7, 1, 9, 5)
Sorted Tuple: (1, 2, 4, 5, 7, 9)
```

30. Write a Python program to convert a list of multiple integers into a single integer

a. [11, 33, 50] >>> 113350

In [15]:
```
a=[11, 33, 50]
b=''
for i in a:
    b =  b + str(i)
print(int(b))
```

```
113350
```

31. Difference between del and clear?

Feature	`del`	`clear()`
Purpose	Deletes a variable or element from a list/dictionary.	Clears all elements from a list or removes all key-value pairs from a dictionary.
Usage	`del variable`	`list.clear()` or `dict.clear()`
Applicability	Works with variables, lists, dictionaries, and other data structures.	Primarily used with lists and dictionaries.
Effect on Variables	Deletes a variable entirely.	N/A (Not applicable to variables).
Effect on Lists	Removes all elements from a list.	Removes all elements from a list, making it empty.
Effect on Dictionaries	Removes all key-value pairs from a dictionary.	Removes all key-value pairs from a dictionary, making it empty.
Return Value	No return value.	No return value (returns `None`).
Error Handling	Raises an error if the variable or element doesn't exist.	No error if the list or dictionary is already empty.

Feature	`del`	`clear()`
Syntax	`del object_name`	`object_name.clear()`
Description	Deletes the object_name completely, including the reference to it.	Removes all the elements from the object_name, but the object_name itself remains.
Availability	Available in all data types	Available in lists, dictionaries, sets, and strings

In [ ]:
```
Example:
The clear() function removes all the key:value pairs from the dictionary and
makes it empty dictionary while del <dict> statement removes the complete dictionary as an object.
```

In [15]:
```
#Example:
d = {1: 'a' , 2 : 'b'}
d.clear()
print(d)
del d
print(d)
```

```
{}
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[15], line 6
      4 print(d)
      5 del d
----> 6 print(d)

NameError: name 'd' is not defined
```

32. Difference between remove and pop?

Feature	`remove()` Method	`pop()` Method
Purpose	Removes the first occurrence of a specified element from the list.	Removes and returns an element at a specified index from the list.
Usage	`list.remove(element)`	`list.pop(index)`
Element Requirement	Requires the element to be in the list; raises a `ValueError` if not found.	Requires an index within the valid range; raises an `IndexError` if the index is out of range.
Return Value	No return value (returns `None`).	Returns the removed element.
Original List Modified	Modifies the list in-place by removing the element.	Modifies the list in-place by removing and returning the element.

In [18]:
```
# Example
list1 = [1, 2, 3, 4, 5]
```

```
# Remove the first occurrence of the number 3
list1.remove(3)
print(list1)

# Remove the element at index 2
list1.pop(2)
print(list1)
```

```
[1, 2, 4, 5]
[1, 2, 5]
```

33. Difference between indexing and Slicing?

Feature	Indexing	Slicing
Purpose	Access a single element of a sequence.	Extract a portion (subsequence) of a sequence.
Syntax	`` `sequence[index]` ``	`` `sequence[start:end:step]` ``
Return Value	Individual element at the specified index.	New sequence containing a portion of the original sequence.
Example	`` `element = sequence[index]` ``	`` `subsequence = sequence[start:end:step]` ``
Applicability	Retrieve specific elements within a sequence.	Extract or manipulate a range of elements within a sequence.
Number of Parameters	Requires a single index.	Requires start, end (exclusive), and optional step.
Range Control	Accesses a single element at the specified position.	Defines a range (start to end) for subsequence extraction.
Use of Step Parameter	Not applicable.	Optional step parameter controls the increment between elements.

In [22]:
```
# Examples:
my_list = [10, 20, 30, 40, 50]
# Accessing a single element by index
element = my_list[2]  # Accesses the element at index 2 (value 30)
print(element)


my_string = "Hello, World!"
# Accessing a string by slicing with start and end.
character = my_string[3:8]  # Accesses the character at index 7
print(character)
```

```
30
lo, W
```

34. Difference between sort and sorted?

Feature	`` `sort()` Method ``	`` `sorted()` Function ``
Purpose	Sorts a list in-place, i.e., it modifies the original list.	Returns a new sorted list without modifying the original list.
Usage	`` `list.sort(key=None, reverse=False)` ``	`` `sorted(iterable, key=None, reverse=False)` ``
Input	Operates on a list.	Works with any iterable (e.g., list, tuple, string).
Return Value	None (returns `` `None` ``).	Returns a new sorted list.
Modification of Input	Modifies the original list.	Leaves the original iterable unchanged.

In [36]:
```
list1 = [1, 5, 3, 2, 4]

# Sort the list in place
list1.sort()
print(list1)

# Sort the list and return a new list
sorted_list = sorted(list1)
print(sorted_list)
```

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

35. Difference between reverse and reversed?

Feature	`reverse()` Method	`reversed()` Function
Purpose	Reverses a list in-place, i.e., it modifies the original list.	Returns a reverse iterator that allows you to iterate over the elements in reverse order without modifying the original iterable.
Usage	`list.reverse()`	`reversed(iterable)`
Input	Operates on a list.	Works with any iterable (e.g., list, tuple, string).
Return Value	None (returns `None`).	Returns a reverse iterator.
Modification of Input	Modifies the original list.	Leaves the original iterable unchanged.

In [37]:
```
list1 = [1, 2, 3, 4, 5]

# Reverse the list in place
list1.reverse()
print(list1)

# Get the reversed list
reversed_list = reversed(list1)
print(list(reversed_list))
```

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

36. Difference between copy and deep copy?

Feature	`copy()`	`deepcopy()`
Purpose	Creates a shallow copy of an object, i.e., it copies the object itself and references to its elements (if it's a compound object like a list or dictionary), but not the elements themselves.	Creates a deep copy of an object, i.e., it creates a new object and recursively copies all elements within the object, including nested elements.
Usage	`copy.copy(object)`	`copy.deepcopy(object)`
Input	Works with any object.	Works with any object.
Copies Nested Objects	Copies references to nested objects (shallow copy).	Recursively copies all nested objects (deep copy).
Effect on Nested Objects	Changes to nested objects are reflected in both the original and copied objects (shallow copy).	Changes to nested objects do not affect the original object, and vice versa (deep copy).

In [56]:
```
# Shallow copy
import copy

original_list = [1, [2, 3]]
print("Original List:",original_list)
print('-----------')

shallow_copied_list = copy.copy(original_list)
shallow_copied_list[1][0] = 4

print("Original List:",original_list)
print("Shallow Copy List:",shallow_copied_list)
```

```
Original List: [1, [2, 3]]
-----------
Original List: [1, [4, 3]]
Shallow Copy List: [1, [4, 3]]
```

In [57]:
```
# Deep copy
import copy

original_list = [1, [2, 3]]
print("Original List:",original_list)
print('-----------')

deep_copied_list = copy.deepcopy(original_list)
deep_copied_list[1][0] = 4

print("Original List:",original_list)
print("Deep Copy List:",deep_copied_list)
```

```
Original List: [1, [2, 3]]
-----------
Original List: [1, [2, 3]]
Deep Copy List: [1, [4, 3]]
```

37. How to check whether the list is empty or not?

In [ ]:
```
Using if statement: You can use the if statement to check if the list has any elements.
```

In [58]:
```
my_list = []

if not my_list:
    print("The list is empty.")
else:
    print("The list is not empty.")
```

```
The list is empty.
```

In [ ]:
```
Using len(): You can use the len() function to check the length of the list.
```

In [59]:
```
my_list = []

if len(my_list) == 0:
    print("The list is empty.")
else:
    print("The list is not empty.")
```

```
The list is empty.
```

In [ ]:
```
Using Boolean Conversion: You can directly use the boolean value of the list in a conditional statement.
```

In [62]:
```
my_list = []

if my_list:
    print("The list is not empty.")
else:
    print("The list is empty.")
```

```
The list is empty.
```

38. How to concatenate two lists?

In [22]:
```
# Using '+' operator
list1 = [1, 2, 3 ]
list2 = [2, 4, 2, 5]
l3 = list1 + list2
l3
```

Out[22]:
```
[1, 2, 3, 2, 4, 2, 5]
```

In [23]:
```
# Using append
list1 = [1, 2, 3 ]
list2 = [2, 4, 2, 5]
app = []
for i in list1:
    list2.append(i)
print(list2)
```

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

In [63]:
```
# Using extend
list1 = [1, 2, 3 ]
list2 = [2, 4, 2, 5]

list1.extend(list2)

print(list1)
```

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

In [64]:
```
# Using List Comprehension
list1 = [1, 2, 3]
list2 = [4, 5, 6]

concatenated_list = [x for x in list1] + [x for x in list2]

print(concatenated_list)
```

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

39. How to find the occurrences of an element in the python list?

In [17]:
```
my_list = [1, 2, 3, 2, 4, 2, 5]
element_to_find = 2

count = 0
for i in my_list:
    if i==element_to_find:
```

```
        count = count+1
print(f'{element_to_find} occurred {count} times ')
```

```
2 occurred 3 times
```

40. How to flatten a list in python?

In [68]:
```
# Using List Comprehension:

l = [[1, 2, 3], [4, 5], [6, 7, 8]]
l1 = [j for i in l for j in i]
print(l1)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

In [69]:
```
nested_list = [[1, 2, 3], [4, 5], [6, 7, 8]]
flat_list = []
for sublist in nested_list:
    for item in sublist:
        flat_list.append(item)

print(flat_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

In [ ]:
```

```