EXPERIMENT NO: 05

Title: Implement python program to perform following operations on the List: Create, Access, Print, Delete, Convert.

Prior Concepts: Variables and basic data types, decision making and looping in python

Objective: To understand and implement basic operations on lists like creating, accessing, printing, deleting, and converting lists.

Theory: A list in Python is an ordered, mutable sequence of elements. It allows for various operations such as creation, modification, access, and deletion of elements. Lists can hold items of different data types, including integers, strings, and other lists.

Basic Concepts:

1. **Indexing:** Each element in a list can be accessed via its index. Indexing starts from 0 for the first element and goes up to n-1 for the last element (where n is the length of the list). Python also supports negative indexing, where -1 refers to the last element.

```
my_list = [10, 20, 30]
print(my_list[0])   #Output:10
print(my_list[-1])  #Output: 30
```

2. **Mutable Data Structure:** Lists are mutable, meaning they can be changed after creation. You can add, remove, or modify elements in a list.

```
my_list = [10, 20, 30]
my_list[1] = 25 # Changing the second element
print(my_list) # Output: [10, 25, 30]
```

3. **Built-in Functions:** Python provides several built-in functions to manipulate lists:
   - len(): Returns the number of elements in the list.
   - del: Deletes elements from the list or the entire list.
   - list(): Converts other data types into lists.

```
my_list = [1, 2, 3]
print(len(my_list)) # Output: 3
del my_list[1] # Deletes the second element
print(my_list) # Output: [1, 3]
tuple_data = (4, 5, 6)
list_from_tuple = list(tuple_data)
# Converts tuple to list
print(list_from_tuple)
# Output: [4, 5, 6]
```

Procedure

Step 1: Setting Up the Environment

1. Open your Python IDE or text editor.
2. Create a new Python file (.py) to write the code for list operations.

Step 2: Writing the Python Code

1. **List Creation:** Lists can be created by placing elements inside square brackets [], separated by commas. You can also use the list() constructor to create an empty list or convert other data types to lists.

```
# Creating a list
my_list = [1, 2, 3, 4, 5]

# Creating an empty list
empty_list = []

# Using the list() constructor
constructed_list = list((6, 7, 8)) # Converting a tuple to a list
```

2. **Accessing List Elements:** Elements in a list can be accessed using indexing. The first element has index 0, and the last element can be accessed using index -1 (negative indexing).

```
my_list = [10, 20, 30, 40, 50]
# Accessing the first
element print(my_list[0])
# Output: 10


# Accessing the last
element print(my_list[-1])
# Output: 50
```

3. **Printing Lists:** Lists can be printed using the print() function. You can print the entire list or iterate through the elements to print each one individually.

```
my_list = [10, 20, 30]
# Printing the entire list
print(my_list) # Output: [10, 20, 30]

# Printing each element
for item in my_list:
     print(item)
```

4. **Updating a List:** List in python can be updated using various ways. Some of them are :

4.1.**Changing Elements:** You can change individual elements in a list by accessing them using their index.

```
# Original list
my_list = [10, 20, 30, 40]

# Update the element at index 1
my_list[1] = 25

# Resulting list
print(my_list) # Output: [10, 25, 30, 40]
```

4.2. Adding Elements: You can add elements to a list using various methods:

- append(): Adds an element to the end of the list.
- extend(): Adds all elements from an iterable (e.g., another list) to the end of the list.
- insert(): Inserts an element at a specified position in the list.

```
# Original list
my_list = [1, 2, 3]

# Adding a single element to the end
my_list.append(4)
print(my_list) # Output: [1, 2, 3, 4]

# Adding multiple elements to the end
my_list.extend([5, 6])
print(my_list) # Output: [1, 2, 3, 4, 5, 6]

# Inserting an element at a specific position
my_list.insert(2, 'a') # Inserts 'a' at index 2
print(my_list) # Output: [1, 2, 'a', 3, 4, 5, 6]
```

4.3. Slicing and Reassigning: You can use slicing to modify parts of a list. For example, you can replace a section of a list with new elements.

```
# Original list
my_list = [1, 2, 3, 4, 5]

# Replace elements at index 1 through 3 with new elements
my_list[1:4] = ['a', 'b', 'c']
print(my_list) # Output: [1, 'a', 'b', 'c', 5]
```

4.4. Using List Comprehensions: List comprehensions provide a concise way to create or update lists based on existing lists.

```
# Original list
my_list = [1, 2, 3, 4, 5]

# Update the list to include squares of the original elements
squared_list = [x**2 for x in my_list]
print(squared_list)  # Output: [1, 4, 9, 16, 25]
```

5. **Deleting Elements from a List:** You can delete elements from a list using various methods:

- remove(): Removes the first occurrence of a specified value.
- pop(): Removes and returns an element at a specified index. If no index is provided, it removes and returns the last element.
- del: Deletes an element at a specified index or the entire list.

```
my_list = [10, 20, 30, 40] #Deleting an element by index
del my_list[1]  #Removes the element at index 1
print(my_list)  #Output: [10, 30, 40]

# Using pop() to remove the last element
my_list.pop() # Removes 40 print(my_list) # Output: [10, 30]

# Removing an element by value
my_list.remove(30) # Removes the first occurrence of 30
print(my_list) # Output: [10]
# Deleting an element by index
del my_list[0]
print(my_list)  # Output: [40]

# Deleting the entire list
del my_list
```

6. **Converting Other Data Types to Lists:** Other data types such as tuples, strings, or sets can be converted into lists using the list() constructor.

```
# Converting a tuple to a list
t1 = (1, 2, 3)
l_to_t = list(t1)
print(l_to_t)  # Output: [1, 2, 3]

# Converting a string to a list
s1 = "hello"
l_to_s = list(s1)
print(l_to_s)  # Output: ['h', 'e', 'l', 'l', 'o']
```

Step 3: Running the Python Script

- Save your Python file.
- Run the script in your IDE or from the command line: python your_script_name.py
- Observe the outputs generated from each operation.

Conclusion: Thus, we have successfully performed various operations on lists in Python.

Questions:

1. Write a Python program to reverse a list using different methods.
2. Write a program that finds the maximum and minimum elements in a list.
3. Write a program that counts the frequency of each element in the list with and without using inbuilt count () function.

EXPERIMENT NO: 06

Title: Implement python program to perform following operations on the Tuple: Create, Access, Print, Delete, Convert.

Prior Concepts: Lists and its operations in python.

Objective: To understand and implement basic operations on tuples like creating, accessing, printing, deleting, and converting lists.

Theory: A tuple is an immutable, ordered sequence of elements. Once created, the elements of a tuple cannot be modified, making it a static data structure. Tuples can hold a variety of data types, including other tuples or lists. They are defined using parentheses () or by using the tuple() constructor.

Basic Concepts:

1. **Indexing in Tuples:** Just like lists, tuples support indexing. The first element has index 0, and negative indexing is also supported, where -1 refers to the last element.

```
my_tuple = (10, 20, 30)
print(my_tuple[0]) # Output: 10

print(my_tuple[-1])  # Output: 30
```

2. **Immutable Data Structure:** Unlike lists, tuples are immutable. This means once a tuple is created, you cannot modify its elements (no adding, removing, or changing elements).

```
my_tuple = (1, 2, 3)
# Attempting to modify a tuple will raise an error
# my_tuple[0] = 10 # Raises TypeError
```

3. Built-in Functions for Tuples: Python provides several functions to manipulate tuples:
   - len(): Returns the number of elements in a tuple.
   - min() and max(): Return the smallest and largest elements of the tuple, respectively.
   - tuple(): Converts other data types (like lists or strings) into tuples.

Procedure:

Step 1: Setting Up the Environment

1. Open your Python IDE or text editor.
2. Create a new Python file (tuple_operations.py) to write the code for tuple operations.

Step 2: Writing the Python Code

1. **Tuple Creation:** Tuples can be created by placing elements inside parentheses (). You can also use the tuple() constructor to create an empty tuple or convert other data types to tuples.

```
# Creating a tuple
my_tuple = (1,2,3, 4, 5)

# Creating an empty tuple
empty_tuple = ()

# Using the tuple() constructor
constructed_tuple = tuple([6, 7, 8])  # Converting a list to a tuple

print("Created Tuples:")
print("my_tuple:", my_tuple)
print("empty_tuple:", empty_tuple)
print("constructed_tuple:", constructed_tuple)
```

2. **Accessing Tuple Elements:** Elements in a tuple can be accessed using indexing. The first element has index 0, and the last element can be accessed using index -1.

```
my_tuple = (10, 20, 30, 40,50)
# Accessing the first element
print(my_tuple[0]) # Output:10

# Accessing the last element
print(my_tuple[-1]) # Output:50
```

3. **Printing Tuples:** Tuples can be printed using the print() function. You can print the entire tuple or iterate through the elements to print each one individually.

```
my_tuple = (10, 20, 30)
# Printing the entire tuple print(my_tuple) # Output: (10, 20, 30)
# Printing each element
for item in my_tuple:
     print(item)
```

4. **Converting Other Data Types to Tuples:** Other data types, such as lists or strings, can be converted into tuples using the tuple() constructor.

```
# Converting a list to a tuple
list_data = [1, 2, 3]
tuple_from_list = tuple(list_data)
print(tuple_from_list)  # Output: (1, 2, 3)

# Converting a string to a tuple
string_data = "hello"
tuple_from_string = tuple(string_data)
print(tuple_from_string)  # Output: ('h', 'e', 'l', 'l', 'o')
```

5. **Updating a tuple:** Tuples in Python are immutable, meaning that once a tuple is created, its elements cannot be directly modified (you cannot add, remove, or change elements). However, there are several techniques to indirectly "update" a tuple by converting it into a mutable data structure (like a list), modifying that structure, and then converting it back into a tuple.

Techniques to "Update" Tuples:

5.1.Updating a Tuple by Conversion to List: Since lists are mutable, you can convert the tuple to a list, update the list, and then convert it back to a tuple.

```
# Original tuple
```

```
my_tuple = (10, 20, 30)

# Step 1: Convert the tuple into a list
temp_list = list(my_tuple)

# Step 2: Modify the list (add, update, or delete elements)
temp_list[1] = 25 # Updating the second element
temp_list.append(40) # Adding a new element

# Step 3: Convert the list back to a tuple
updated_tuple = tuple(temp_list)

# Final updated tuple
print(updated_tuple)  # Output: (10, 25, 30, 40)
```

5.2.Concatenating Tuples (Adding Elements): You can add new elements to a tuple by concatenating it with another tuple.

```
# Original tuple
my_tuple = (1, 2, 3)

# Adding a new element by concatenation
new_tuple = my_tuple + (4,)

# Resulting tuple after concatenation
print(new_tuple) # Output: (1, 2, 3, 4)
```

5.3.Removing Elements from a Tuple (By Rebuilding): Although you can't remove elements from a tuple directly, you can rebuild the tuple by removing unwanted elements.

```
# Original tuple
```

```
my_tuple = (1, 2, 3, 4, 5)

# Rebuild the tuple by excluding the element you want to remove
updated_tuple = my_tuple[:2] + my_tuple[3:] # Removes element at index 2

# Final tuple after removing an element
print(updated_tuple) # Output: (1, 2, 4, 5)
```

5.4.Reassigning a new tuple: You can reassign the entire tuple with new values if you want to completely replace the contents.

```
# Original tuple
my_tuple = (1, 2, 3)

# Reassign a new tuple
my_tuple = (4, 5, 6)

# Resulting tuple
print(my_tuple)  # Output: (4, 5, 6)
```

6. **Deleting Tuples:** Tuples themselves are immutable, but you can delete entire tuples using the del keyword.

```
print(my_tuple)  # Output: (10, 20, 30)

# Deleting the entire tuple
del my_tuple

# Trying to print a deleted tuple will raise an error
# print(my_tuple)  # Raises NameError: name 'my_tuple' is not defined
```

Step 3: Running the Python Script

- Save your Python file.
- Run the script in your IDE or from the command line.
- Observe the outputs generated from each operation.

Conclusion: Thus, we successfully implemented basic tuple operations, including creation, access, printing, updating and conversion from other data types.

Questions:

1. Create a tuple that contains other tuples. Perform operations to access elements from both the outer and inner tuples.
2. Concatenate two tuples and after concatenation, sort the concatenated tuple in both ascending and descending order.
3. Create a tuple of three elements and demonstrate tuple unpacking to assign each element to a separate variable. Then, swap the values of these variables using tuple unpacking.