Ex No.03	List Slicing, Tuples, and Dictionaries in Python	Reg.No. URK24CS9068
24.01.25		

1. List Slicing

1a. Demonstrate extracting even-indexed elements from a list using slicing.

Aim: To develop a Python program that extracts and displays even-indexed elements from a given list using slicing.

Objective: To understand and implement list slicing in Python to retrieve elements at even indices efficiently.

Algorithm:

Step 1: Start

Step 2: Define a list with some elements.

Step 3: Use slicing with a step of 2 (list[::2]) to extract even-indexed elements.

Step 4: Print the extracted elements.

Step 5: End

Program:

```
#Demonstrate extracting even-indexed elements from a list using slicing.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
L=[0,1,2,3,4,5,6,7,8,9,10]
print(L[::2])
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
[0, 2, 4, 6, 8, 10]
```

Result: The program successfully extracts and displays elements at even indices from the given list using slicing.

1b: Implement a program to reverse a list using slicing.

Aim: To develop a Python program that reverses a given list using slicing.

Objective: To understand and implement list slicing in Python to reverse the order of elements efficiently.

Algorithm:

Step 1: Start

Step 2: Define a list with some elements.

Step 3: Use slicing with a step of -1 (list[::-1]) to reverse the list.

Step 4: Print the reversed list.

Step 5: End

Program:

```
#2. Implement a program to reverse a list using slicing.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
L=[0,1,2,3,4,5,6,7,8,9,10]
print(L[::-1])
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Result: The program successfully reverses the given list using slicing and displays the output correctly.

1c. Modify a given list by replacing elements in a specific range using slicing.

Aim: To develop a Python program that modifies a given list by replacing elements in a specific range using slicing.

Objective: To understand and implement list slicing in Python to replace elements within a specified index range.

Algorithm:

Step 1: Start

Step 2: Define a list with some elements.

Step 3: Specify the range of indices to be replaced.

Step 4: Assign new values to the specified range using slicing (list[start:end] = new_values).

Step 5: Print the modified list.

Step 6: End

Program:

```
# Modify a given list by replacing elements in a specific range using slicing.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
L=[0,1,2,3,4,5,6,7,8,9,10]
L[0:2]=[4,2]
print(L)
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
[4, 2, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Result: The program successfully replaces elements in the specified range using slicing and displays the modified list.

1d. Construct a sublist containing elements from index 2 to 8, skipping every second element.

Aim: To develop a Python program that extracts a sublist from a given list, containing elements from index 2 to 8 while skipping every second element using slicing.

Objective: To understand and apply list slicing in Python to extract elements within a specified range with a step size.

Algorithm:

Step 1: Start

Step 2: Define a list with some elements.

Step 3: Use slicing (list[2:9:2]) to extract elements from index 2 to 8, skipping every second element.

Step 4: Print the resulting sublist.

Step 5: End

Program:

```
#Construct a sublist containing elements from index 2 to 8, skipping every second element.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
L=[0,1,2,3,4,5,6,7,8,9,10]
A=L[2:9:2]
print(A)
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
[2, 4, 6, 8]
```

Result: The program successfully extracts a sublist containing elements from index 2 to 8 while skipping every second element using slicing.

1e: Create a new list by removing the last n elements of a given list using slicing.

Aim: To develop a Python program that removes the last n elements from a given list using slicing.

Objective: To understand and implement list slicing in Python to create a new list by excluding the last n elements.

Algorithm:

Step 1: Start

Step 2: Define a list with some elements.

Step 3: Input the number of elements (n) to be removed from the end.

Step 4: Use slicing (list[:-n]) to create a new list without the last n elements.

Step 5: Print the modified list.

Step 6: End

Program:

```
#5. Create a new list by removing the last n elements of a given list using slicing.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
L=[0,1,2,3,4,5,6,7,8,9,10]
print("This is our list:",L)
n=int(input("Enter no. of elements to remove"))
L=L[:-n]
print(L)
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
This is our list: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Enter no. of elements to remove 4
[0, 1, 2, 3, 4, 5, 6]
```

Result: The program successfully removes the last n elements from the given list using slicing and displays the modified list.

2a. Define a program to find the maximum and minimum values in a tuple of integers.

Aim: To find the maximum and minimum values from a given tuple of integers using Python.

Objective: To understand how to access and analyze data in tuples using built-in functions in Python.

Algorithm:

Step 1: Start
Step 2: Define a tuple with integers
Step 3: Initialize max and min with the first element of the tuple
Step 4: Loop through each element in the tuple
Step 5: If the element is greater than max, update max
Step 6: If the element is less than min, update min
Step 7: Print the max and min
Step 8: End

Program:

```
#Design a program to find the maximum and minimum values in a tuple of integers.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
T=(12,45,68,23,87,12,67,13,2,54,8,42,45)
max,min=0,T[0]
for i in range(0,len(T),1):
    if(max<T[i]):
        max=T[i]
    if(min>T[i]):
        min=T[i]
print("Maximun value in Tuple is :",max)
print("Minimun value in Tuple is :",min)
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
Maximun value in Tuple is : 87
Minimun value in Tuple is : 2
```

Result: The program successfully removes the last n elements from the given list using slicing and displays the modified list.

2b. Demonstrate tuple concatenation and slicing to create a new tuple

Aim: To develop a Python program that demonstrates tuple concatenation and slicing to create a new tuple.

Objective: To understand and apply tuple operations such as concatenation and slicing to manipulate tuples effectively.

Algorithm:

Step 1: Start

Step 2: Define two tuples with some elements.

Step 3: Concatenate the two tuples using the + operator.

Step 4: Use slicing to extract a subtuple from the concatenated tuple.

Step 5: Print the concatenated and sliced tuples.

Step 6: End

Program:

```
# Demonstrate tuple concatenation and slicing to create a new tuple.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
T=(12,45,68,23,87,12,67,13,2,54,8,42,45)
T1=(23,34,45,56,67,87,98,9,43)
T2=T+T1
T3=T2[0:11]
print(T2)
print(T3)
 0.0s
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
(12, 45, 68, 23, 87, 12, 67, 13, 2, 54, 8, 42, 45, 23, 34, 45, 56, 67, 87, 98, 9, 43)
(12, 45, 68, 23, 87, 12, 67, 13, 2, 54, 8)
```

Result: The program successfully concatenates two tuples and extracts a subtuple using slicing, demonstrating tuple operations effectively.

2c. Write a program to count the occurrences of each element in a tuple.

Aim: To develop a Python program that counts the occurrences of each element in a tuple.

Objective: To understand and implement the use of tuples and dictionaries in Python for counting element frequencies.

Algorithm:

Step 1: Start

Step 2: Define a tuple with some elements.

Step 3: Create an empty dictionary to store element counts.

Step 4: Iterate through the tuple:

- If the element is already in the dictionary, increment its count.
- Otherwise, add the element to the dictionary with a count of 1.

Step 5: Print the dictionary containing element frequencies.

Step 6: End

Program:

```
# Construct a program to count the frequency of words in a given string using a dictionary.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
D={}
d=0
s="Apple Mango Orange Mango Guava Guava Mango"
L=s.split()
for i in range(0,len(L)):
    if L[i] not in D:
        for j in range(i,len(L),1):
            if L[i]==L[j]:
                d=d+1
        D[L[i]]=d
print(D)
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
{'Apple': 1, 'Mango': 4, 'Orange': 5, 'Guava': 7}
```

Result: The program successfully counts and displays the occurrences of each element in the given tuple using a dictionary.

2d. Implement tuple unpacking to assign its elements to multiple variables.

Aim: To develop a Python program that demonstrates tuple unpacking by assigning its elements to multiple variables.

Objective: To understand and implement tuple unpacking in Python, which allows assigning tuple elements to separate variables efficiently.

Algorithm:

Step 1: Start

Step 2: Define a tuple with multiple elements.

Step 3: Assign tuple elements to individual variables using unpacking.

Step 4: Print the assigned variables to verify unpacking.

Step 5: End

Program:

```
# Implement tuple unpacking to assign its elements to multiple variables.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
T=(1,2,3,4,5,6)
a,s,d,f,g,h=T
print(a,s,d,f,g,h)
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
1 2 3 4 5 6
```

Result: The program successfully assigns tuple elements to multiple variables using unpacking and displays their values.

2e. Evaluate the immutability of tuples by attempting to modify an element and observe the result.

Aim: To evaluate the immutability of tuples by attempting to modify an element and observing the result.

Objective: To understand that tuples in Python are immutable, meaning their elements cannot be changed once assigned.

Algorithm:

Step 1: Start

Step 2: Define a tuple with some elements.

Step 3: Attempt to modify an element of the tuple.

Step 4: Observe and handle the error that occurs.

Step 5: End

Program:

```
#Evaluate the immutability of tuples by attempting to modify an element and observe the result.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
T=(1,2,3,4,5,6)
T[3]=34
print(T)
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------


---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[1], line 6
      4 print("------------------")
      5 T=(1,2,3,4,5,6)
----> 6 T[3]=34
      7 print(T)

TypeError: 'tuple' object does not support item assignment
```

Result: The program raises a TypeError, confirming that tuples are immutable and their elements cannot be modified. The original tuple remains unchanged.

3a. Construct a program to count the frequency of words in a given string using a dictionary.

Aim: To develop a Python program that counts the frequency of words in a given string using a dictionary.

Objective: To understand and implement dictionary operations in Python to store and count word occurrences efficiently.

Algorithm:

Step 1: Start

Step 2: Take a string input from the user.

Step 3: Convert the string to lowercase and split it into words.

Step 4: Create an empty dictionary to store word frequencies.

Step 5: Iterate through the words:

- If the word exists in the dictionary, increment its count.
- Otherwise, add the word with a count of 1.

Step 6: Print the dictionary with word frequencies.

Step 7: End

Program:

```
# Construct a program to count the frequency of words in a given string using a dictionary.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
D={}
d=0
s="Apple Mango Orange Mango Guava Guava Mango"
L=s.split()
for i in range(0,len(L)):
    if L[i] not in D:
        for j in range(i,len(L),1):
            if L[i]==L[j]:
                d=d+1
        D[L[i]]=d
print(D)
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
{'Apple': 1, 'Mango': 4, 'Orange': 5, 'Guava': 7}
```

Result: The program successfully counts and displays the frequency of each word in the given string using a dictionary.

3b. Write a program to merge two dictionaries and handle duplicate keys.

Aim: To develop a Python program that merges two dictionaries and handles duplicate keys by adding their values.

Objective: To understand and implement dictionary merging in Python while resolving key conflicts by summing the values of duplicate keys.

Algorithm:

Step 1: Start

Step 2: Define two dictionaries with some key-value pairs.

Step 3: Create an empty dictionary to store the merged result.

Step 4: Iterate through both dictionaries:

- If a key exists in both, sum their values.
- Otherwise, add the key-value pair to the merged dictionary.

Step 5: Print the merged dictionary.

Step 6: End

Program:

```
#Write a program to merge two dictionaries and handle duplicate keys.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
dict1 = {'a': 10, 'b': 20, 'c': 30}
dict2 = {'b': 5, 'c': 15, 'd': 40}
merged_dict = dict1.copy()

for key, value in dict2.items():
    if key in merged_dict:
        merged_dict[key] += value
    else:
        merged_dict[key] = value
print("Merged Dictionary:", merged_dict)
```

0.0s

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
Merged Dictionary: {'a': 10, 'b': 25, 'c': 45, 'd': 40}
```

Result: The program successfully merges two dictionaries, handling duplicate keys by summing their values. The output correctly reflects the combined data.

3c. Demonstrate accessing, updating, and deleting key-value pairs in a dictionary.

Aim: To develop a Python program that demonstrates accessing, updating, and deleting key-value pairs in a dictionary.

Objective: To understand how to work with dictionaries in Python by retrieving, modifying, and removing key-value pairs.

Algorithm:

Step 1: Start

Step 2: Define a dictionary with some key-value pairs.

Step 3: Access a value using its key.

Step 4: Update an existing key's value.

Step 5: Add a new key-value pair.

Step 6: Delete a key-value pair.

Step 7: Print the dictionary after each operation.

Step 8: End

Program:

```
#Demonstrate accessing, updating, and deleting key-value pairs in a dictionary.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
student = {'name': 'Alice', 'age': 20, 'grade': 'A'}
print("Original dict: ",student)
print("Student Name:", student['name'])
student['age'] = 21
print("Updated Dictionary:", student)
student['city'] = 'New York'
print("After Adding City:", student)
del student['grade']
print("After Deleting Grade:", student)
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
Original dict:  {'name': 'Alice', 'age': 20, 'grade': 'A'}
Student Name: Alice
Updated Dictionary: {'name': 'Alice', 'age': 21, 'grade': 'A'}
After Adding City: {'name': 'Alice', 'age': 21, 'grade': 'A', 'city': 'New York'}
After Deleting Grade: {'name': 'Alice', 'age': 21, 'city': 'New York'}
```

Result: The program successfully demonstrates accessing, updating, and deleting key-value pairs in a dictionary, modifying its contents dynamically.

3d. Create a dictionary using comprehension to store squares of numbers from 1 to 10.

Aim: To develop a Python program that creates a dictionary using comprehension to store squares of numbers from 1 to 10.

Objective: To understand and apply dictionary comprehension in Python for efficient dictionary creation.

Algorithm:

Step 1: Start

Step 2: Use dictionary comprehension to generate a dictionary where keys are numbers from 1 to 10 and values are their squares.

Step 3: Print the dictionary.

Step 4: End

Program:

```
# Create a dictionary using comprehension to store squares of numbers from 1 to 10.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
squares_dict = {num: num**2 for num in range(1, 11)}
print("Squares Dictionary:", squares_dict)
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
Squares Dictionary: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

Result: The program successfully creates a dictionary where each number from 1 to 10 is mapped to its square using dictionary comprehension.

3e. Implement a program to find and display the key(s) associated with the maximum value in a dictionary.

Aim: To develop a Python program that finds and displays the key(s) associated with the maximum value in a dictionary.

Objective: To understand how to iterate through a dictionary and extract key(s) corresponding to the highest value.

Algorithm:

Step 1: Start

Step 2: Define a dictionary with key-value pairs.

Step 3: Find the maximum value using the max() function.

Step 4: Iterate through the dictionary to find key(s) with this maximum value.

Step 5: Print the key(s) with the highest value.

Step 6: End

Program:

```
#Implement a program to find and display the key(s) associated with the maximum value in a dictionary.
print("D. Brian Gabriel")
print("URK24CS9068")
print("------------------")
d = {'A': 85, 'B': 92, 'C': 78, 'D': 92, 'E': 88}
max_value = max(d.values())
max_keys = [key for key, value in d.items() if value == max_value]
print("Key(s) with the maximum value:", max_keys)
```

Output:

```
D. Brian Gabriel
URK24CS9068
------------------
Key(s) with the maximum value: ['B', 'D']
```

Result: The program successfully finds and displays the key(s) associated with the highest value in the given dictionary.