DATE : 7 june 2024

DAY : Friday

TOPICS : List & list methods , Tuple and Tuple methods.

LISTS IN PYTHON

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

```
#Example
#Create a List:
FRUITS = ["apple", "banana", "cherry"]
print(FRUITS)
```

```
['apple', 'banana', 'cherry']
```

```
Students = []
Students
```

```
[]
```

Some of the more relevant characteristics of list objects include being:

Ordered: They contain elements or items that are sequentially arranged according to their specific insertion order.

Zero-based: They allow you to access their elements by indices that start from zero.

Mutable: They support in-place mutations or changes to their contained elements.

Heterogeneous: They can store objects of different types.

Growable and dynamic: They can grow or shrink dynamically, which means that they support the addition, insertion, and removal of elements.

Nestable: They can contain other lists, so you can have lists of lists.

Iterable: They support iteration, so you can traverse them using a loop or comprehension while you perform operations on each of their elements.

Sliceable: They support slicing operations, meaning that you can extract a series of elements from them.

Combinable: They support concatenation operations, so you can combine two or more lists using the concatenation operators.

Copyable: They allow you to make copies of their content using various techniques.

Lists are sequences of objects. They’re commonly called containers or collections because a single list can contain or collect an arbitrary number of other objects.

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc, the third item has index [2] etc.

ORDERED

When we say that lists are ordered, it means that the items have a defined order, and that order will not change. If you add new items to a list, the new items will be placed at the end of the list.

Note: There are some list methods that will change the order, but in general: the order of the items will not change.

Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

```
#Example
#Lists allow duplicate values:

my_list = ["apple", "banana", "cherry", "apple", "cherry"]
print(my_list)
```

```
['apple', 'banana', 'cherry', 'apple', 'cherry']
```

List Length: To determine how many items a list has, use the len() function:

```
#Example
#Print the number of items in the list:

UNI = ["CTU", "KTU", "DTU"]
print(len(UNI))
```

```
3
```

List Items - Data Types List items can be of any data type:

It is also possible to use the list() constructor when creating a new list.

```
#Example
#Using the list() constructor to make a List:
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

LIST METHODS :

1. APPEND :

```
#APPEND: Used to new elements to the end of an existing list.
#syntax = list_name.append("item")

UNI = ["CTU", "KTU", "DTU"]
UNI.append("CU")
print(UNI)
```

```
['CTU', 'KTU', 'DTU', 'CU']
```

```
#appending another list at the end of one list
# we can also append another list in an already existing list in order to make nested lists.
#syntax = list1_name.append(list2_name)
UNI = ["CTU", "KTU", "DTU"]
Courses = ["BCA", "MCA"]
UNI.append(Courses)
print(UNI)
```

```
['CTU', 'KTU', 'DTU', ['BCA', 'MCA']]
```

2. CLEAR :

```
#clear() method: It removes all the elements from the list, meaning makes the list empty.
#syntax = list_name.clear()
UNI = ["CTU", "KTU", "DTU"]
UNI.clear()
print(UNI)
```

```
[]
```

3. COPY :

```
#copy() method: It creates another copy of existing list.
UNI = ["CTU", "KTU", "DTU"]
print( UNI.copy())

colleges = UNI.copy()
print(colleges)
```

```
['CTU', 'KTU', 'DTU']
['CTU', 'KTU', 'DTU']
```

4. COUNT :

```
#count(): it counts the occurrence of particular element in a list
#syntax = list_name.count("element")
UNI = ["CTU", "KTU", "DTU", "CTU"]
UNI.count("CTU")
```

```
2
```

5. EXTEND :

```
#extend() method: It is used to concatenate 2 existing lists().
UNI = ["CTU", "KTU", "DTU", "CTU"]
UNI2 = ["CU", "PU"]
UNI2.extend(UNI)
print(UNI2)
```

```
['CU', 'PU', 'CTU', 'KTU', 'DTU', 'CTU']
```

6. INDEX :

```
#index() method : It returns the index no. of first occurrence of a particular element from the list.

UNI2.index("KTU")
```

```
3
```

7. INSERT :

```
#insert() method: This method is used to add new element at a particular index to an existing list.
#syntax = list_name.insert(index, "item")

UNI2.insert(1,"PUJNABI UNIVERSITY")
UNI2
```

```
['CU', 'PUJNABI UNIVERSITY', 'PU', 'CTU', 'KTU', 'DTU', 'CTU']
```

8. POP :

```
#pop() method: by default used to remove last element from the list.
# but can also be used to remove a particular element by providing its index.
FUR = ["TABLE", "CHAIR", "STOOL", "BED"]
FUR.pop()
FUR
FUR.pop(2)
print(FUR)
```

```
['TABLE', 'CHAIR']
```

9. REMOVE :

```
#remove() : It is also used to remove an element from an existing list like pop() method. But the difference is that it takes element itself
FUR = ["TABLE", "CHAIR", "STOOL", "BED"]
FUR.remove("CHAIR")
FUR
```

```
['TABLE', 'STOOL', 'BED']
```

10. REVERSE :

```
#reverse method
FUR = ["TABLE", "CHAIR", "STOOL", "BED"]
FUR.reverse()
FUR
```

```
['BED', 'STOOL', 'CHAIR', 'TABLE']
```

11. SORT :

```
#sorting a list

# by using sort() method
FUR.sort()
print(FUR)

#by using sorted() function
FUR = ["TABLE", "CHAIR", "STOOL", "BED"]
print(sorted(FUR))
```

```
['BED', 'CHAIR', 'STOOL', 'TABLE']
['BED', 'CHAIR', 'STOOL', 'TABLE']
```

In general, you should use lists when you need to:

Keep your data ordered: Lists maintain the order of insertion of their items.

Store a sequence of values: Lists are a great choice when you need to store a sequence of related values.

Mutate your data: Lists are mutable data types that support multiple mutations.

Access random values by index: Lists allow quick and easy access to elements based on their index.

In contrast, avoid using lists when you need to:

Store immutable data: In this case, you should use a tuple. They're immutable and more memory efficient.

Represent database records: In this case, consider using a tuple or a data class.

Store unique and unordered values: In this scenario, consider using a set or dictionary. Sets don't allow duplicated values, and dictionaries can't hold duplicated keys.

LIST SLICING:

List slicing in Python allows you to extract a portion of a list by specifying a range of indices. The general syntax for list slicing is:

```
list[start:stop:step]
```

start: The index where the slice begins (inclusive). If omitted, it defaults to the beginning of the list.

stop: The index where the slice ends (exclusive). If omitted, it defaults to the end of the list.

step: The step size, which determines how many items to skip between indices. If omitted, it defaults to 1.

```
#example 1
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Extracting a slice from index 2 to 5 (exclusive)
slice1 = numbers[2:6]
print(slice1)
```

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

```
#Example 2: Omitting start and stop
# Omitting start, defaults to the beginning of the list
slice2 = numbers[:4]
print(slice2)

# Omitting stop, defaults to the end of the list
```

```
slice3 = numbers[5:]
print(slice3)
```

```
[1, 2, 3, 4]
[6, 7, 8, 9]
```

```
#Example 3: Using a Negative Index
# Negative index, counts from the end of the list
slice4 = numbers[-5:]
print(slice4)

slice5 = numbers[:-3]
print(slice5)
```

```
[5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6]
```

```
#Example 4: Step Parameter
# Using step to get every second item
slice6 = numbers[::2]
print(slice6)
# Using step to get every third item
slice7 = numbers[1::3]
print(slice7)
```

```
[1, 3, 5, 7, 9]
[2, 5, 8]
```

```
#Example 5: Negative Step

# Using negative step to reverse the list
slice8 = numbers[::-1]
print(slice8)

# Using negative step to get every second item in reverse
slice9 = numbers[8:2:-2]
print(slice9)
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1]
[9, 7, 5]
```

```
#Example 6: Advanced Slicing
# Extracting a middle portion with a step
slice10 = numbers[2:8:2]
print(slice10)
```

```
[3, 5, 7]
```

```
#Example 7: Slicing with an Empty Result
# When the start index is greater than or equal to the stop index
slice11 = numbers[5:2]
print(slice11)

# When the step is negative but start is less than stop
slice12 = numbers[2:5:-1]
print(slice12)
```

```
[]
[]
```

Questions (list)

Q.1 colors = ['red', 'blue', 'green', 'yellow'] Using the colors list defined above, print the: First element, Second element, Last element, Second-to-last element, Second and third elements, Element at index 4.

```
colors = ["red","blue","green", "yellow"]
print(colors[0])
print(colors[1])
print(colors[3])
print(colors[2])
print(colors[1:3])
```

```
try:
  print(colors[4])
except IndexError:
  print("Index out of range")
```

```
red
blue
yellow
green
['blue', 'green']
Index out of range
```

Q.2 Below is a list with seven integer values representing the daily water level (in cm) in an imaginary lake. However, there is a mistake in the data. The third day’s water level should be 693. Correct the mistake and print the changed list. water_level = [730, 709, 682, 712, 733, 751, 740]

```
water_level = [730,709,682,712,733,751,740]
water_level.insert(3,693)
print(water_level)
```

```
[730, 709, 682, 693, 712, 733, 751, 740]
```

Q.3 Add the data for the eighth day to the list from above. The water level was 772 cm on that day. Print the list contents afterwards.

```
water_level = [730,709,682,712,733,751,740]
water_level.append(772)
print(water_level)
```

```
[730, 709, 682, 712, 733, 751, 740, 772]
```

Q.4 Still using the same list, add three consecutive days using a single instruction. The water levels on the 9th through 11th days were 772 cm, 770 cm, and 745 cm. Add these values and then print the whole list.

```
water_level.append(772)
water_level.append(770)
water_level.append(745)
print(water_level)
```

```
[730, 709, 682, 712, 733, 751, 740, 772, 772, 770, 745]
```

Q.5 There are two ways to delete data from a list: by using the index or by using the value. Start with the original water_level list we defined in the second exercise and delete the first element using its index. Then define the list again and delete the first element using its value.

```
water_level =[730,709,682,712,733,751,740]
water_level.pop(0)
print(water_level)
```

```
[709, 682, 712, 733, 751, 740]
```

```
water_level =[730,709,682,712,733,751,740]
water_level.remove(730)
print(water_level)
```

```
[709, 682, 712, 733, 751, 740]
```

Practice Questions(List) :

You are managing the inventory for a small bookstore. Create a list of book titles available in the store. Add new titles to the list as they arrive. If a book is sold out, remove it from the list. Write a function to check if a specific book is in stock.

```
book_titles = ["This is how you heal","Pride and Prejudice"]
book_titles.append("Power of subconcious mind ")
#book_titles.append("1984")
print(book_titles)
book_titles.remove("Pride and Prejudice")
```

```
print(book_titles)
if "1984" in book_titles:
  print("Book is present")
else:
  print("Book is sold out")
```

```
['This is how you heal', 'Pride and Prejudice', 'Power of subconcious mind ']
['This is how you heal', 'Power of subconcious mind ']
Book is sold out
```

You have a list of grades for a class of students. Write a function to add a new grade to the list. Another function should remove the lowest grade. Write a third function to calculate the average grade.

```
grades = [98,70,85,90,77,63]
grades.append(89)
print(grades)
grades.remove(min(grades))
print(grades)

grades_average = sum(grades)/len(grades)
print(grades_average)
```

```
[98, 70, 85, 90, 77, 63, 89]
[98, 70, 85, 90, 77, 89]
84.83333333333333
```

Implement a simple to-do list application. Create a list to store tasks. Write functions to add a task, remove a task by its name, and display all tasks. Ensure that the tasks are displayed in the order they were added.

```
to_do = ["do laundary", "oven repairing"]
to_do.append("Buy groceries")
to_do.append("do assignments")
print(to_do)
to_do.remove("oven repairing")
print(to_do)
```

```
['do laundary', 'oven repairing', 'Buy groceries', 'do assignments']
['do laundary', 'Buy groceries', 'do assignments']
```

Create a list to store items you need to buy from the grocery store. Write functions to add items, remove items, and check if a specific item is already on the list. Ensure that duplicate items are not added.

```
groceries = ["milk","eggs"]
groceries.append("flour")
groceries.append("chocolate")
groceries.append("bread")
print(groceries)
if "bread" in groceries:
  print(" bread is in the list")
else:
  print("bread is  not in the list")
```

```
['milk', 'eggs', 'flour', 'chocolate', 'bread']
 bread is in the list
```

You are tracking daily temperatures for a month. Create a list to store these temperatures. Write functions to find the highest and lowest temperatures, and to calculate the average temperature for the month

```
temperatures = [43,52,50,47]
highest_temp = max(temperatures)
print( f"Highest temperature :  {highest_temp}")
lowest_temp = min(temperatures)
print( f"Lowest temprature : {lowest_temp} ")
average_temp = sum(temperatures)/len(temperatures)
print(f"Average temperature : {average_temp} ")
```

```
Highest temperature :  52
Lowest temprature : 43
Average temperature : 48.0
```

You are responsible for assigning seats to students in a classroom. Create a list to represent the seating arrangement. Write functions to assign a seat to a new student, find a student's seat by name, and swap seats between two students.

```
students = ["Arsh", "Manpreet", "Inder", "Karan"]
students.append("Jashan")
student1_name = 'Arsh'
student1_seat= students.index(student1_name)
student2_name = 'Manpreet'
student2_seat = students.index(student2_name)
student3_name= 'Inder'
student3_seat = students.index(student3_name)
students[student1_seat], students[student2_seat] = students[student2_seat],students[student1_seat]
print(students)
```

```
['Manpreet', 'Arsh', 'Inder', 'Karan', 'Jashan']
```

You are organizing an event and need to manage the guest list. Create a list to store the names of the guests. Write functions to add a guest, remove a guest by name, and check if a person is on the guest list.

```
#guest list
guests = [ "Carol","Dave","Ele", "Steve"]
guests.append("Haili")
guests.remove("Dave")
print(guests)
if "Maden" in guests:
 print(" Maden is present in the guest list ")
else:
  print(" Maden is not  present in the guest list")
```

```
['Carol', 'Ele', 'Steve', 'Haili']
 Maden is not  present in the guest list
```

Create a playlist for a party. Use a list to store song titles. Write functions to add a song, remove a song by title, and shuffle the playlist. Ensure no duplicate songs are added to the playlist.

```
#party playlist
song_titles = ["Perfect", "Let me down "]
song_titles.append("Friend like me ")
print(song_titles)
song_titles.remove("Perfect")
print(song_titles)

import random
random.shuffle(song_titles)
print(song_titles)
```

```
['Perfect', 'Let me down ', 'Friend like me ']
['Let me down ', 'Friend like me ']
['Let me down ', 'Friend like me ']
```

You are managing course enrollments for a university. Create a list of students enrolled in a course. Write functions to add a student, remove a student by name, and find the total number of students enrolled.

```
enrolled_students = ["Carol", "Dave"," Ele"]
enrolled_students.append("Steve")
enrolled_students.append("Haili")
print(enrolled_students)
enrolled_students.remove("Carol")
print(enrolled_students)

total_enrolled = len(enrolled_students)
```

```
print(total_enrolled)
```

```
['Carol', 'Dave', ' Ele', 'Steve', 'Haili']
['Dave', ' Ele', 'Steve', 'Haili']
4
```

You are creating a recipe management system. Create a list of ingredients required for a recipe. Write functions to add an ingredient, remove an ingredient by name, and check if a specific ingredient is in the list. Sort the list of ingredients alphabetically.

```
ingredient_list = ["Salt", "Flour", "Sugar"]
ingredient_list.append("Pepper")
ingredient_list.append("Water")
print(ingredient_list)
ingredient_list.remove("Sugar")
print(ingredient_list)
if "Eggs" in ingredient_list:
  print("Eggs are present ")
else:
  print("eggs not present")
ingredient_list.sort()
print(ingredient_list)
```

```
['Salt', 'Flour', 'Sugar', 'Pepper', 'Water']
['Salt', 'Flour', 'Pepper', 'Water']
eggs not present
['Flour', 'Pepper', 'Salt', 'Water']
```

You are analyzing responses from a survey. Create a list of responses. Write functions to add a response, remove a response by its index, and calculate the frequency of each unique response.

```
responses = ["agree", "strongly agree"]
responses.append("satisfied")
responses.append("Disagree")
responses.pop(0)
print(responses)
```

```
['strongly agree', 'satisfied', 'Disagree']
```

Implement a flight booking system. Create a list to store the names of passengers on a flight. Write functions to add a passenger, remove a passenger by name, and find the seat number of a passenger (assuming the index represents the seat number).

```
passengers = ["John", "Dave"]
passengers.append("Haili")
print(passengers)
passengers.remove("Dave")
 #print(passengers)
passenger_name = 'Haili'
if "passenger_name" in passengers:
  seat_number= passengers.index(passenger_name)
  print(f" {passenger_name}'s seat is : {seat_number +1} ")
else:
  print(f"{passenger_name} is not in flight")
```

```
['John', 'Dave', 'Haili']
Haili is not in flight
```

Develop an online shopping cart system. Create a list to store the items in the cart. Write functions to add an item, remove an item by name, and calculate the total price of all items in the cart.

```
cart_items = ["Mango","Orange"]
cart_items.append("Banana")

cart_items.remove("Mango")
print(cart_items)
total_items = len(cart_items)
```

```
print(f"Total number of items in cart : {total_items}")
price_item = 50
total_price = total_items*50
print(f" Total amount to be paid is ${total_price}")
```

```
['Orange', 'Banana']
Total number of items in cart : 2
 Total amount to be paid is $100
```

You are managing a library system. Create a list of books currently borrowed by patrons. Write functions to add a borrowed book, return a book (remove it from the list), and check if a specific book is currently borrowed.

```
borrowed_books = ["This is how you heal", "Pride and Prejudice"]
borrowed_books.append("1984")
print(borrowed_books)
return_book = '1984'

if return_book in borrowed_books:
  borrowed_books.remove(return_book)
  print(borrowed_books)
specific_book = 'Pride and Prejudice'
if specific_book in borrowed_books:
  print(f"{specific_book} is currently borrowed")
else:
  print(f"{specific_book} is not currently borrowed")
```

```
['This is how you heal', 'Pride and Prejudice', '1984']
['This is how you heal', 'Pride and Prejudice']
Pride and Prejudice is currently borrowed
```

TUPLE

In Python, a tuple is an ordered, immutable collection of elements. Tuples are similar to lists, but the key difference lies in their immutability. Once a tuple is created, its elements cannot be changed or modified. This makes tuples suitable for situations where data should remain constant throughout the program.

CREATING TUPLES

Tuples can be created using parentheses () and separating elements with commas ,. Even a single element should be followed by a comma to distinguish it as a tuple.

```
# Creating an empty tuple
empty_tuple = ()
# Creating a tuple with elements
fruits = ('apple', 'orange', 'banana')
# Single element tuple
single_element_tuple = ("42",)
type(single_element_tuple)
```

```
tuple
```

The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

```
#Example
#Using the tuple() method to make a tuple:
mytuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
print(mytuple)
type(mytuple)
```

```
('apple', 'banana', 'cherry')
tuple
```

Accessing Elements

Accessing elements in a tuple is similar to lists, using indexing. Indexing starts from 0 for the first element.

```
fruits = ('apple', 'orange', 'banana')
fruits[1]
```

```
'orange'
```

Check if Item Exists

To determine if a specified item is present in a tuple use the "in" keyword:

```
#Example
#Check if "Ludhiana" is present in the tuple:
City = ("DELHI", "MUMBAI", "KOTA", "LUDHIANA")
if "LUDHIANA" in City:
  print("Yes, 'LUDHIANA' is in the city tuple")
else:
  print("No, 'Ludhiana' is not in city tuple")
```

```
Yes, 'LUDHIANA' is in the city tuple
```

Immutable Nature

Once a tuple is created, its elements cannot be modified or changed.

```
# Attempting to modify a tuple (will result in an error)
Countries = ("INDIA", "SRI LANKA", "AMERICA")
Countries[0] = 'PAKISTAN' # TypeError: 'tuple' object does not support item assignment
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-29-c92a7a5cd179> in <cell line: 3>()
      1 # Attempting to modify a tuple (will result in an error)
      2 Countries = ("INDIA", "SRI LANKA", "AMERICA")
----> 3 Countries[0] = 'PAKISTAN' # TypeError: 'tuple' object does not support item assignment

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

But there are some workarounds.

We can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
#Convert the tuple into a list to be able to change it:
Country = ("INDIA", "SRI LANKA", "AMERICA")
y = list(Country)
y[1] = "BANGLADESH"
X = tuple(y)
print(X)
type(X)
```

```
('INDIA', 'BANGLADESH', 'AMERICA')
tuple
```

Similary we can apply other list operations by converting tuple into list.

Count() Method

The count() method of Tuple returns the number of times the given element appears in the tuple. Syntax: tuple.count(element)

```
# Creating tuples
Tuple1 = (0, 1, 2, 3, 2, 3, 1, 3, 2)
```

```
Tuple2 = ('python', 'geek', 'python',
'for', 'java', 'python')
# count the appearance of 3
res = Tuple1.count(3)
print('Count of 3 in Tuple1 is:', res)
# count the appearance of python
res = Tuple2.count('python')
print('Count of Python in Tuple2 is:', res)
```

```
Count of 3 in Tuple1 is: 3
Count of Python in Tuple2 is: 3
```

```
# Creating tuples
Tuple = (0, 1, (2, 3), (2, 3), 1,
[3, 2], 'geeks', (0,))
# count the appearance of (2, 3)
res = Tuple.count((2, 3))
print('Count of (2, 3) in Tuple is:', res)
# count the appearance of [3, 2]
res = Tuple.count([3, 2])
print('Count of [3, 2] in Tuple is:', res)
```

```
Count of (2, 3) in Tuple is: 2
Count of [3, 2] in Tuple is: 1
```

Index() Method

The Index() method returns the first occurrence of the given element from the tuple. Syntax: tuple.index(element, start, end) Parameters: element: The element to be searched. start (Optional): The starting index from where the searching is started end (Optional): The ending index till where the searching is done Note: This method raises a ValueError if the element is not found in the tuple.

```
# Creating tuples
Tuple = (0, 1, 2, 3, 2, 3, 1, 3, 2)
# getting the index of 3
res = Tuple.index(3)
print('First occurrence of 3 is', res)
# getting the index of 3 after 4th
# index
res = Tuple.index(3, 4, 6)
print('First occurrence of 3 after 4th index is:', res)
```

```
First occurrence of 3 is 3
First occurrence of 3 after 4th index is: 5
```

Reversing a tuple

```
numbers = (1, 2, 3, 4, 5)
numbers[::-1]
```

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

Advantages of Tuples:

Immutability: Tuples provide data integrity by ensuring that the data remains constant.

Performance: Tuples are generally faster than lists for certain operations due to their immutability.

Valid Dictionary Key: Tuples can be used as keys in dictionaries, unlike lists.

Practice Questions (Tuple)

Tuple Creation and Access: Create a tuple named colors with the elements 'red', 'green', and 'blue'.

Access the second element of the tuple and print it.

```
colors = ( "red","green", "blue")
print(colors[1])
```

```
green
```

. **Immutable Nature:** Explain in your own words why tuples are considered immutable. Attempt to modify an element in an existing tuple and observe the resulting error.

```
colors = ("red","green", "blue")
colors.append("yellow")
print(colors)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-2-31af28482ca6> in <cell line: 2>()
      1 colors = ("red","green", "blue")
----> 2 colors.append("yellow")
      3 print(colors)

AttributeError: 'tuple' object has no attribute 'append'
```

Tuple Slicing: Given the tuple numbers = (1, 2, 3, 4, 5) , use slicing to extract the elements from index 1 to 3 (inclusive). What would be the output of numbers[::-1] ?

```
numbers = (1,2,3,4,5)
slice = numbers[1:4]
print(slice)
slice1= numbers[::-1]
print(slice1)
```

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

Tuple Concatenation and Repetition: Create two tuples, fruits with elements 'apple', 'banana', and berries with elements 'strawberry', 'blueberry'. Concatenate the two tuples and store the result in a new tuple named combined_fruits . Repeat the combined_fruits tuple three times and print the result.

```
fruits = ("apple", "banana")
berries = ("strawberry", "blueberry")
x = list(fruits)
y = list(berries)
x.extend(y)
combined_fruits = tuple(x)
print(combined_fruits)
repeated_fruits = (combined_fruits)*3
print(repeated_fruits)
```

```
('apple', 'banana', 'strawberry', 'blueberry')
('apple', 'banana', 'strawberry', 'blueberry', 'apple', 'banana', 'strawberry', 'blueberry', 'apple', 'banana', 'strawberry', 'blueberry
```

Built-in Tuple Methods: Create a tuple named grades with the elements 90, 85, 92, 88, 95. Use the count() method to find how many times the grade 88 appears in the tuple. Use the index() method to find the index of the grade 92.

```
grades = (90,85,92,88,95)
print(grades.count(88))
print(grades.index(92))
```

```
1
2
```

Multiple Data Types in a Tuple: Create a tuple named mixed_types with elements 'apple', 42, and 3.14. Access and print the second element of the tuple.

```
mixed_types = ("apple", 42, 3.14)
print(mixed_types[1])
```

```
42
```

Conversion: Convert the list ['cat', 'dog', 'rabbit'] into a tuple named animals . Print the tuple to verify the conversion.

```
list1 = ["cat","dog","rabbit"]
animals = tuple(list1)
print(type(animals))
```

```
<class 'tuple'>
```

Nested Tuples: Create a tuple outer_tuple with two elements: 'apple' and another tuple ('red', 'green', 'yellow'). Access the second element of the inner tuple and print it.

```
outer_tuple = ("apple", ("red","green","yellow"))
print(outer_tuple[1])
inner_tuple = outer_tuple[1]
second_element = inner_tuple[1]
print(second_element)
```

```
('red', 'green', 'yellow')
green
```