Day 2 of Training at Ansh Info Tech

Topics Covered

Lists in Python

- **Lists Creation**: Methods to create lists, including using the list constructor.
- **List Slicing**: Techniques for accessing subsets of list elements.
- **List Concatenation**: Combining lists.
- **List Functions**:
    - `len(),sum(), sorted()`
- **List Methods**:
    - .append(), .clear(), .copy(), .count(), .extend(), .index(), .insert(), .pop(), .remove(), .reverse(), .sort()

List Indexing in Python

- Accessing elements using positive and negative indices.

Tuples

- **Creation**: Using parentheses `()` and the `tuple()` constructor.
- **Concatenation**: Combining tuples.
- **Access**: Accessing elements using indices.
- **Immutability**: Tuples cannot be changed after creation. To modify, convert to a list, make changes, and convert back to a tuple.
- **Tuple Methods**:
    - `count(),index()`

Practical Exercises on Lists and Tuples

- **List Exercises**:
    1. Using a list of colors to demonstrate indexing.
    2. Correcting and updating a list of water levels.
    3. Adding new elements to a list.
    4. Removing elements by index and value.
- **Tuple Exercises**:
    1. Creating and accessing elements in a tuple.

2. Demonstrating tuple immutability and attempting to modify elements.
3. Tuple slicing and reversing.
4. Tuple concatenation and repetition.
5. Using tuple methods `count()` and `index()`.
6. Advantages of tuples over lists.
7. Creating tuples with multiple data types.
8. Converting a list to a tuple.
9. Working with nested tuples.

Applied Programming Scenarios

1. **Bookstore Inventory Management**: Using lists to manage book titles, including adding, removing, and checking stock.
2. **Mapping Application**: Storing geographical coordinates in tuples, adding new coordinates, finding specific coordinates, and calculating distances.
3. **Classroom Grades**: Adding grades to a list, removing the lowest grade, and calculating the average grade.
4. **Employee Records**: Managing employee records using tuples, adding new employees, finding employees by ID, and updating departments.
5. **To-Do List Application**: Adding, removing, and displaying tasks in a list.
6. **Grocery Store List**: Managing a grocery list, ensuring no duplicate items, and checking for specific items.
7. **Daily Temperatures**: Finding the highest and lowest temperatures in a list and calculating the average temperature.
8. **Classroom Seating Assignment**: Assigning seats, finding seats by student name, and swapping seats between students.
9. **Event Guest List**: Managing a guest list, including adding, removing, and checking for specific guests.
10. **Party Playlist**: Managing a playlist, adding and removing songs, and shuffling the playlist.
11. **University Course Enrollment**: Managing student enrollments, including adding, removing, and counting students.
12. **Recipe Management System**: Managing a list of ingredients, including adding, removing, and sorting ingredients.
13. **Survey Response Analysis**: Adding responses to a list, removing responses by index, and calculating the frequency of unique responses.
14. **Flight Booking System**: Managing passenger names on a flight, including adding, removing, and finding seat numbers.

The training session provided extensive hands-on practice with lists and tuples, reinforcing the theoretical concepts through practical application scenarios.

```
# Lists, Tuples, Set, Dictionaries
# Lists -> Heterogeneous, Nestable, Mutable
# len()-> Length Function
# list() constructor to create a list
# Example: thislist = list(('a','b'))
# indexing: 0 to n-1, -n to -1
# Methods -> .append(-), .clear(), .copy(), .count(-), .extend(), .index(), .insert(), .pop(
l1 = ['a','b']
l2 = ['c', 1]
l1.append(2)
print(l1)
l1.append(l2)
print(l1)
thislist = list(("Mehak", "Jacky", "Jacky", "Oggy"))
print(thislist)
```

```
['a', 'b', 2]
['a', 'b', 2, ['c', 1]]
['Mehak', 'Jacky', 'Jacky', 'Oggy']
```

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[-1])
print(colors[1:])
print(colors[1:3])
print(colors[4])
```

```
red
blue
yellow
['blue', 'green', 'yellow']
['blue', 'green']
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-8-00f0854c09e3> in <cell line: 7>()
      5 print(colors[1:])
      6 print(colors[1:3])
----> 7 print(colors[4])

IndexError: list index out of range
```

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]

ques 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.

ques 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 = [730, 709, 682, 712, 733, 751, 740]
water_level[2] = 693
print(water_level)

water_level.append(772)
print(water_level)

water_level.extend([772, 770, 745])
print(water_level)
```

```
[730, 709, 693, 712, 733, 751, 740]
[730, 709, 693, 712, 733, 751, 740, 772]
[730, 709, 693, 712, 733, 751, 740, 772, 772, 770, 745]
```

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

Tuples -> (), Mutable, tuple() constructor, single element-> (1,), access, immutable, for changing-> convert to list and then reconvert to tuple Methods -> count(), index()

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])
```

. 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.

```
#Tuples are immutable, because if we try and modify an element of the tuple, we get an error
colors = ('red', 'green', 'blue')
colors[1] = 'yellow'
print(colors)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-798c8f4a4d6e> in <cell line: 3>()
      1 #Tuples are immutable, because if we try and modify an element of the tuple, we
get an error
      2 colors = ('red', 'green', 'blue')
----> 3 colors[1] = 'yellow'
      4 print(colors)

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

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)
print(numbers[1:4])
#Output of numbers[::-1] is the tuple in reverse order
print(numbers[::-1])
```

```
(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')
berries = ('strawberry', 'blueberry')
combined_fruits = fruits + berries
print(combined_fruits)
print(combined_fruits*3)
```

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
```

Advantages of Tuples: Explain one advantage of using tuples over lists in Python. Describe a scenario where the immutability of tuples could be beneficial in a program.

```
# Advantage of using tuples over lists:
# Memory efficiency: Tuples are more memory efficient than lists because they are immutable.
# This means that the memory allocated to a tuple cannot be changed, which makes it more com
# efficient in terms of memory usage.

# Scenario where immutability of tuples is beneficial:
# Security: In situations where data integrity is critical, such as in financial transactior
# secure communication protocols, using tuples can provide an additional layer of security.
# tuples are immutable, once data is stored in a tuple, it cannot be accidentally or intenti
# modified, ensuring the integrity and consistency of the data.
```

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])
```

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

```
l1 = ['cat','dog','rabbit']
animals = tuple(l1)
print(animals)
```

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][1])
```

```
green
```

- 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. Check if a specific book is in stock without using function

```
book_titles = ['Book1', 'Book2', 'Book3']
book_titles.append('Book4')
print(book_titles)
book_titles.remove('Book3')
print(book_titles)
if 'Book2' in book_titles:
    print('Book2 is in stock')
else:
    print('Book2 is not in Stock')
```

```
['Book1', 'Book2', 'Book3', 'Book4']
['Book1', 'Book2', 'Book4']
Book2 is in stock
```

You are working on a mapping application that stores geographical coordinates (latitude and longitude) as tuples. Create a list of these coordinate tuples. Write functions to add a new coordinate, find a specific coordinate, and calculate the distance between two coordinates

```
c1 = (15.12,25.12)
c2 = (15, 23.35)
li = [c1, c2]
li.append((42,24.15))
li.index((15, 23.35))
dist = (li[0][0]-li[1][0], li[0][1]-li[1][1])
print(dist)
```

```
(0.11999999999999922, 1.7699999999999996)
```

2.

- 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.

```
li = [5, 7, 4, 9]
li.append(10)
li.remove(min(li))
print(avg(li))
```

1.

- You are managing employee records for a company. Each employee's record is stored as a tuple containing their name, ID, and department. Create a list of tuples to store these records. Write a function to add a new employee, another to find an employee by their ID, and a function to update the department of a specific employee.

```
t1 = ('rahul', 1234, 'cse')
t2 = ('b', 1235, 'abc')
t3 = ('c', 1352, 'adf')
l1 = [t1, t2, t3]
print(l1)
l1.append(['d', 12345, 'sdf'])
print(l1)
for i in range(0, len(l1)):
  if(1234 == l1[i][1]):
      print("Employee index is ", i);
      print(l1[i]);
temp = list(l1[2])
temp[2]='bcd'
l1[2]=tuple(temp)
print(l1)
```

```
[('rahul', 1234, 'cse'), ('b', 1235, 'abc'), ('c', 1352, 'adf')]
[('rahul', 1234, 'cse'), ('b', 1235, 'abc'), ('c', 1352, 'adf'), ['d', 12345, 'sdf']]
Employee index is  0
('rahul', 1234, 'cse')
[('rahul', 1234, 'cse'), ('b', 1235, 'abc'), ('c', 1352, 'bcd'), ['d', 12345, 'sdf']]
```

3.

- 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.

```
li = ['bathing', 'singing','dancing', 'running']
li.append('eating')
li.remove('dancing')
print(li)
```

4.

- 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.

```
li = ['a','b','c']
if 'd' not in li:
  li.append('d')
else:
  print('d is already present in list')
print(li)
li.remove('b')
print(li)
```

5.

- 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.

```
li = [25.6, 22, 32.1]
print(max(li))
print(min(li))
print(float(sum(li)/len(li)))
```

6.

- 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.

```
seating_arrangement = [('r', 1), ('j', 2), ('p',3)]

seating_arrangement.append(('student_name', 1234))

student_name = 'j' #Student whose seat is to be found
for student, seat in seating_arrangement:
    if student == student_name:
        return seat

#Swap Seats
seat1 = find_seat(student1_name)
seat2 = find_seat(student2_name)
seating_arrangement.remove((student1_name, seat1))
seating_arrangement.remove((student2_name, seat2))
seating_arrangement.append((student1_name, seat2))
seating_arrangement.append((student2_name, seat1))

print(seating_arrangement)
```

7.

- 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.

```
li = ['a','b','c']
if 'd' not in li:
  li.append('d')
else:
  print('d is already present in list')
print(li)
li.remove('b')
print(li)
```

8.

- 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.

```
li = ['a','b','c']
if 'd' not in li:
  li.append('d')
else:
  print('d is already present in list')
print(li)
li.remove('b')
print(li)

#shuffle the playlist
import random
random.shuffle(li)
print(li)
```

9.

- 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.

```
li = ['a','b','c']
#add a Student
li.append('d')
print(li)
#remove a student by name
li.remove('b')
print(li)
#find the total number of students enrolled
print(len(li))
```

10. - 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.

```
li = ['a','b','c']
#add an ingredient to list
li.append('d')
print(li)
#remove an ingredient by name
li.remove('b')
print(li)
#check if a specific ingredient is in the list
if 'd' in li:
  print('d is present in list')
else:
  print('d is not present in list')
#sort the list of ingredients alphabetically
li.sort()
print(li)
```

11. - 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.

```
li = ['a','b','c']
#add a response
li.append('d')
print(li)
#remove a response by its index
li.pop(1)
print(li)
#calculate the frequency of each unique response
print(li.count('a'))
print(li.count('b'))
print(li.count('c'))
print(li.count('d'))
```

12. - 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).

```
li = ['a','b','c']
#add a passenger
li.append('d')
print(li)
#remove a passenger by name
li.remove('b')
print(li)
#find the seat number of a passenger (assuming the index represents the seat number)
print('seat number of passenger d is ', li.index('d'))
```

```
['a', 'b', 'c', 'd']
['a', 'c', 'd']
seat number of passenger d is  2
```

13. - 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.

```
li = [('a', 50), ('b', 100), ('c', 500)]
#add an item
li.append(('d', 1000))
print(li)
#remove an item by name
li.remove(('b', 100))
print(li)
#calculate the total price of all items in the cart
total = 0
for item, price in li:
    total += price
print(total)
```

14. - 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.

```
li = ['a','b','c']
#borrow a book
li.append('d')
print(li)
#return a book (remove it from the list)
li.remove('b')
print(li)
#check if a specific book is currently borrowed
if 'd' in li:
  print('d is present in borrowed list')
else:
  print('d is not present in borrowed list')
```

15. - Create a weekly meal planner. Use a list to store meals for each day of the week. Write functions to add a meal for a specific day, remove a meal by day, and display the meal plan for the entire week.

```
li=[['a','b'],['c','d'],['e','f'],['g','h'],['i','j'],['k','l'],['m','n']]
# add a meal for a specific day like day 5 i=4
li[4].append('o')
print(li)
#remove a meal by day like day 5 i=4
li[4].remove('j')
print(li)
#display the meal plan for the entire week
print(li[0])
print(li[1])
print(li[2])
print(li[3])
print(li[4])
print(li[5])
print(li[6])
```

3.

- You have a list of students, where each student is represented as a tuple containing their name, age, and grade. Write a function to add a new student, another function to remove a student by name, and a function to find the student with the highest grade.

```
l1 = [('a',10,'A'), ('b',11,'B'), ('c',12,'C')]
l1.append(('d',13,'D'))
print(l1)
for name, age, grade in l1:
  if name == 'b':
    l1.remove(('b',11,'B'))
print(l1)
#find student with highest grade
l1.sort(key=lambda x: x[2])
print(l1[-1])

l2 = sorted(l1)
print(l2[-1])
```

```
[('a', 10, 'A'), ('b', 11, 'B'), ('c', 12, 'C'), ('d', 13, 'D')]
[('a', 10, 'A'), ('c', 12, 'C'), ('d', 13, 'D')]
('d', 13, 'D')
('d', 13, 'D')
```

4. on system for a software application where configurations are stored as tuples. Each configuration contains a setting name and its value. Write a function to add a new configuration, check if a specific setting exists, and retrieve the value of a specific setting.

```
con = [('a', 10), ('b', 20), ('c', 30)]
con.append(('d', 50))
for setting, value in con:
    if 'b' == setting:
        print(value)
```

5.

- Manage an inventory system where each product is represented as a tuple containing the product name, price, and quantity. Create a list of these product tuples. Write functions to add a new product, update the price of an existing product, and find the total inventory value.

```
pro = [('a', 10, 10), ('b', 20, 20), ('c', 30, 30)]
pro.append(('d', 50, 50))
print(pro)
pro[0][1] = 20 #update price of an existing product
print(pro)
total = 0
for name, price, quantity in pro:
    total += price * quantity
print(total)
```

6.

- Develop a contact list application where each contact is stored as a tuple containing their name, phone number, and email address. Write functions to add a new contact, search for a contact by name, and update the phone number of a specific contact.

```
contacts = [('a', 10, 'a@gmail.com'), ('b', 20, 'b@gmail.com'), ('c', 30, 'c@gmail.com')]
contacts.append(('d', 50, 'd@gmail.com'))
for name, phone, email in contacts:
    if 'b' == name:
        print(phone)
        phone = 1234554321 #Update Phone No. of a specific contact
print(contacts)
```

7.

- You are managing course schedules for a university. Each course is represented as a tuple containing the course name, instructor, and meeting time. Create a list of these course tuples. Write functions to add a new course, find all courses taught by a specific instructor, and update the meeting time of a specific course.

```
course = [('a', 'b', 'c'), ('d', 'e', 'f'), ('g', 'h', 'i')]
course.append(('j', 'k', 'l'))
for name, instructor, meeting in course:
    if 'b' == instructor:
        print(meeting)
        meeting = 'm' #Update Meeting Time of a specific course
print(course)
```

8.

- Represent points in 3D space as tuples containing x, y, and z coordinates. Create a list of these point tuples. Write functions to add a new point, calculate the distance between two

points, and find the point closest to the origin.

```
li = [(1,2,3), (4,5,6), (7,8,9)]
li.append((10,11,12))
print(li)
dist = (li[0][0]-li[1][0], li[0][1]-li[1][1], li[0][2]-li[1][2])
print(dist)
#find the point closest to the origin
min_distance = float('inf')
min_point = None
for point in li:
    distance = (point[0]**2 + point[1]**2 + point[2]**2)**0.5
    if(distance<min_distance):
        min_distance = distance
        min_point = point
print(min_point)
print(min_distance)
```

9.

- Track stock prices where each stock is represented as a tuple containing the stock symbol, the price, and the date of the price. Write functions to add a new stock price, find the latest price for a specific stock, and calculate the average price of a specific stock over a given period.

```
stocks = [('a', 10, '2023-01-01'), ('b', 20, '2023-01-02'), ('c', 30, '2023-01-03')]
stocks.append(('d', 50, '2023-01-04'))
print(stocks)
latest_price = 0
stocks.sort(key=lambda x:x[2])
print(stocks)
for symbol, price, date in stocks:
    if 'b' == symbol:
        latest_price = price
print(latest_price)
#calculate the average price of a specific stock over a given period
sum = 0
count = 0
for symbol, price, date in stocks:
    if 'b' == symbol:
        sum += price
        count += 1
print(sum/count)
```

10. - Manage a library system where each borrowed book is represented as a tuple containing the book title, borrower's name, and due date. Create a list of these book tuples. Write functions to add a borrowed book, check if a specific book is borrowed, and update the due date of a borrowed book.

```
li = [('title', 'name', 'date'), ('title1', 'name1', 'date1'), ('title2', 'name2', 'date2')]
li.append(('title3', 'name3', 'date3'))
print(li)
for title, name, date in li:
    if 'title2' == title:
        print(date)
        date = 'date4' #Update Due Date of a borrowed book
print(li)
```

11. - Create a restaurant menu where each item is represented as a tuple containing the item name, description, and price. Write functions to add a new menu item, update the description of an existing item, and find all items below a certain price.

```
li = [('a', 'b', 10), ('c', 'd', 20), ('e', 'f', 30)]
li.append(('g', 'h', 40))
print(li)
for name, description, price in li:
    if 'b' == name:
        description = 'i' #Update Description of an existing item
print(li)
for name, description, price in li:
    if price < 25:
        print(name)
```

12. - Develop an immutable shopping cart system where each cart item is represented as a tuple containing the item name, quantity, and price. Write functions to add a new item to the cart, update the quantity of an existing item, and calculate the total cost of the cart.

```
li = [('a', 1, 2), ('c', 3, 4), ('e', 5, 6)]
li.append(('h', 7, 8))
```

13. - Manage flight information where each flight is represented as a tuple containing the flight number, destination, and departure time. Write functions to add a new flight, update the departure time of a flight, and find all flights to a specific destination.

```
total = 0
```

```
info = [(123, 'a', '10:00'), (456, 'b', '11:00'), (789, 'c', '12:00')]
info.append((101, 'd', '13:00'))
for num, dest, time in info:
    if 123 == num:
```