Topics covered in the lecture:

1. Tuples are immutable but its elements are mutable
2. Indexing
3. Singleton tuples
4. in or not in, concatenate (+), replicate(*), t.count(<item>), t.index(<item>), sorted()
5. Looping through tuples

Practice/Tutorial Questions:

Some output/error finding/mcq type questions for practice:

1. What is the output of the following code:

```
my_tuple = (1, 2, [3, 4])
my_tuple[2][0] = 5
print(my_tuple)
```

2. Will the following code lead to an error? If yes, then why. If no, then what would be the output?

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

3. What would be the output of the following code?

```
tuple1 = (1, 2, 3)
tuple2 = tuple1
tuple2 += (4,)
print(tuple1)
```

4. Will the following code lead to an error? If yes, then why. If no, then what would be the output?

```
my_tuple = (1, (2, 3))
my_tuple[1][0] = 5
print(my_tuple)
```

Tuple Manipulation Programming Questions:

1. Write a function that takes two lists of tuples and concatenates corresponding tuples from the two lists.
   Input:
   list1 = [(1, 2), (3, 4,10)]
   list2 = [(5, 6, 9), (7, 8)]
   Output: [(1, 2, 5, 6, 9), (3, 4, 10, 7, 8)]

2. Given a list of tuples, write a function that finds the maximum and minimum values for each position across all tuples and returns a new tuple.
   Input: [(1, 5), (3, 2), (8, 6), (10,5), (2,11),(4,5)]
   Output:
   Max tuple: (10, 11)
   Min tuple: (1, 2)
3. Given a list of tuples where each tuple contains two integers, write a function to find all pairs of tuples whose element-wise sum equals a specified target tuple.
   Input:
   input_data = [(2, 6), (4, 8), (8, 6), (9, 8), (1,4), (6,4)]
   target_sum = (10, 12)
   Output:
   [((2, 6), (8, 6)), ((4, 8), (6, 4)), ((9, 8), (1, 4))]
4. Write a function that takes a list of tuples, along with a list of indices, and reorders the elements of each tuple based on the list of indices.
   Inputs:
   input_data = [(1, 2, 3), (4, 5, 6), (7, 8, 9), (9,4,2)]
   indices = [2, 0, 1]
   Output: [(3, 1, 2), (6, 4, 5), (9, 7, 8), (2, 9, 4)]

Some scenario-based questions:

1. You are working on a project with a collection of rectangles. The dimensions of each rectangle are fixed and will not change. To represent these unchangeable dimensions effectively, you decide to use tuples.
   Requirements:

- Store dimensions in a nested tuple: **`((l1,w1),(l2,w2),...)`**
- Implement a function **`calculate_area(dimensions)`** that takes the dimensions of a rectangle as a parameter and returns the area of that rectangle.
- Store the areas in a nested tuple: **`(((l1,w1),area1),((l2,w2),area2)...)`**

2. You are developing a program to track a collection of cities and their geographical coordinates (latitude and longitude). Each city's coordinates should remain constant after they are defined. You decide to use tuples to represent the coordinates because they are immutable, ensuring that the data cannot be accidentally modified.

   Requirements:

   I. Store the cities and their coordinates in a tuple of tuples formatted as **`(("City1",(lat1, lon1)),("City2", (lat2, lon2)), ...)`**.

   II. Implement a function **`calculate_distance(city1, city2)`** that takes two city coordinate tuples and calculates the Euclidean distance between them using the formula:

   distance $=\sqrt{(lat2 - lat1) ** 2 + (lon2 - lon1) ** 2}$

   III. a nested tuple that includes each city and its distance from a reference city (e.g., "**`CityA`**").

.