Day 4 Tutorial : Data Collection Programming.

Task 1: Knowledge and understanding (Tuple and Dictionaries)

1. Compare and contrast tuple and dictionaries in Python, and how does it differ from a list?

Tuples and dictionaries are both data structures in Python that can store collections of data. However, there are some key differences between the two.

Tuples are immutable, meaning that their contents cannot be changed once they are created. Dictionaries, on the other hand, are mutable, meaning that their contents can be changed after they are created.

Tuples are also unordered, meaning that the order of the elements in a tuple does not matter. Dictionaries, on the other hand, are ordered, meaning that the order of the elements in a dictionary matters.

2. Can you modify the elements of a tuple after it's created? Explain.

- No, you cannot modify the elements of a tuple after it is created. Tuples are immutable, meaning that their contents cannot be changed once they are created. This is because tuples are stored in memory as a single contiguous block of data.

If you try to modify an element of a tuple, Python will throw a TypeError exception.

3. How do you access elements within a tuple using indexing and slicing?

To access elements within a tuple using indexing, you use square brackets ([]) and the index of the element you want to access. The index of the first element in a tuple is 0, and the index of the last element is the length of the tuple minus 1.

For example, the following code accesses the first and third elements of the tuple my_tuple:

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

first_element = my_tuple[0]

third_element = my_tuple[2]

print(first_element)

print(third_element)
```

4. What is the purpose of the `count()` and `index()` methods in tuples? Provide examples.

- The count() method returns the number of times a specified value appears in the tuple.

For example:

```
my_tuple = (1, 2, 3, 1, 4, 5)
# Count the number of times the value 1 appears in the tuple
count = my_tuple.count(1)
print(count)
```

- The index() method returns the index of the first occurrence of a specified value in the tuple. If the value does not appear in the tuple, the index() method raises a ValueError exception.

For example:

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

# Get the index of the first occurrence of the value 1 in the tuple
index = my_tuple.index(1)
print(index)
```

5. Explain how key-value pairs work in dictionaries, and provide an example.

- Key-value pairs are the fundamental building block of dictionaries in Python. A key-value pair is a simple data structure that consists of two parts: a key and a value. The key is a unique identifier for the value, and the value is the data that is associated with the key.

Example:

```
key = "name"
value = "Alice"
```

6. What happens if you try to access a key that doesn't exist in a dictionary?

If you try to access a key that doesn't exist in a dictionary in Python, you will get a KeyError exception.

Example:

```
my_dictionary = {"name": "Alice"}
# Try to access the key "age"
print(my_dictionary["age"])
```

Task 2: Practical (Tuple and dictionary)

1. Write a function that takes a list of numbers and returns a tuple containing the sum and average of the numbers.

```
def sum_and_average(numbers):
    if not numbers:
        return(0,0)

    total=sum(numbers)
    average=total/len(numbers)
    return(total, average)

numbers=(1,2,2,3,4,5)
result=sum_and_average(numbers)
print("sum:",result[0])
print("average:",result[1])
```

2.Create a dictionary of student names and a corresponding tuple of their exam scores. Write a program to find and display the name of the student with the highest score.

```
# Create a dictionary of student names and a corresponding tuple of their
exam
# scores. Write a program to find and display the name of the student with
the highest
# score.
#
# Create a dictionary of student names and their exam scores

student_scores = {
    "Alice": (90, 85, 88),
    "Bob": (78, 92, 95),
    "Charlie": (85, 88, 92),
    "David": (92, 78, 86),
}

# Function to find the student with the highest score
def find_student_with_highest_score(scores_dict):
    if not scores_dict:
        return None

    highest_score = -1
    highest_scoring_student = None

    for student, scores in scores_dict.items():
        average_score = sum(scores) / len(scores)
        if average_score > highest_score:
            highest_score = average_score
            highest_scoring_student = student

    return highest_scoring_student

# Find and display the name of the student with the highest score
highest_scorer = find_student_with_highest_score(student_scores)

if highest_scorer:
    print(f"The student with the highest score is {highest_scorer}.")
```

```
else:
    print("No students in the dictionary.")
```

3. Write a program that takes a list of dictionaries, where each dictionary represents a person with keys 'name' and 'age'. Calculate and display the average age of all the people in the list.

```
# Write a program that takes a list of dictionaries, where each dictionary
represents a
# person with keys &amp;#39;name&amp;#39; and &amp;#39;age&amp;#39;. Calculate and display
the average age of all the
# people in the list.

person=[
    {"name":"raghul","age":20},
    {"name":"dinesh potta","age":19},
]

total_age=sum(people["age"] for people in person)
average_age=total_age/len(person)
print(f"the average age is {average_age:.2f} years")
```

4. Given a list of tuples, each containing a student's name and their scores in three subjects, calculate and display the average score for each student. Store the results in a dictionary with student names as keys and average scores as values.

```
# List of tuples containing student names and their scores in three
subjects
student_scores = [
    ("Alice", (90, 85, 88)),
    ("Bob", (78, 92, 95)),
    ("Charlie", (85, 88, 92)),
    ("David", (92, 78, 86)),
]

# Initialize an empty dictionary to store the average scores
average_scores = {}

# Calculate the average score for each student and store in the dictionary
for student, scores in student_scores:
    average_score = sum(scores) / len(scores)
    average_scores[student] = average_score

# Display the average scores for each student
for student, average_score in average_scores.items():
    print(f"{student}: Average Score = {average_score:.2f}")
```