LAB # 10

TUPLE AND DICTIONARY

OBJECTIVE

Getting familiar with other data storing techniques - Tuple and Dictionary.

THEORY

- A tuple is a sequence of immutable Python objects.
- It is a collection which contain **duplicate items.**
- **Immutable:** items can be modified, replaced, or removed
- **Ordered:** maintains the order in which items are added
- **Index-based:** items are accessed using their position (starting from 0)
- Can store mixed data types (integers, strings, booleans, even other lists)
- The elements in a tuple are separated by commas and are enclosed by a pair of **brackets ().**

Example: Syntax for creating Tuple

```
# Creating Tuples

t1 = ()                              # Empty Tuple

t2 = ('Python', 'Programming')       # Tuple Using Strings

my_list = [5, 10, 15, 20]            # Tuple from a List
t3 = tuple(my_list)

t4 = tuple('Hello')                  # Tuple using Built-in Function

# Printing Tuples

print("Empty Tuple:", t1)
print("Tuple from Strings:", t2)
print("Tuple from List:", t3)
print("Tuple using tuple() function:", t4)
```

Output:

```
>>> %Run Lab_10.py
Empty Tuple: ()
Tuple from Strings: ('Python', 'Programming')
Tuple from List: (5, 10, 15, 20)
Tuple using tuple() function: ('H', 'e', 'l', 'l', 'o')
>>>
```

LAB # 10

Example: Tuple with mixed data, type(), len(), nested tuple, constructor

```
# ========================================
#       Tuple Examples Program
# ========================================

tup = (12, 'Welcome', 45, 'Python')          # Creating a Tuple (mixed data types)

num_tuple = (10, 20, 30, 40)                 # First tuple for nested example
lang_tuple = ('AI', 'ML')                    #Second tuple for nested example
nested_tuple = (num_tuple, lang_tuple)       #Creating a Tuple with nested tuples

repeat_tuple = ('Data',) * 3                 #Creating a Tuple with repetition

single_tuple = ("orange",)                   #Example of type checking a single element tuple

length_tuple = ("mango", "banana", "grapes")     #Example for finding length

constructor_tuple = tuple(("car", "bike", "train"))  #Creating a tuple using tuple() constructor
# ========================================
#            Printing (BOLD TEXT)
# ========================================
print("\033[1mTuple with mixed data types:\033[0m", tup)
print("\033[1mNested Tuple:\033[0m", nested_tuple)
print("\033[1mTuple with repetition:\033[0m", repeat_tuple)
print("\033[1mType of single-element tuple:\033[0m", type(single_tuple))
print("\033[1mLength of length_tuple:\033[0m", len(length_tuple))
print("\033[1mTuple created using constructor:\033[0m" , constructor_tuple)
```

Output:

```
>>> %Run Lab_10.py
Tuple with mixed data types: (12, 'Welcome', 45, 'Python')
Nested Tuple: ((10, 20, 30, 40), ('AI', 'ML'))
Tuple with repetition: ('Data', 'Data', 'Data')
Type of single-element tuple: <class 'tuple'>
Length of length_tuple: 3
Tuple created using constructor: ('car', 'bike', 'train')
```

LAB # 10

Accessing Items from a Tuple

- An element in a tuple can be accessed through the index operator, using the syntax: **myTuple[index].**
- Tuple indexes start from 0 to **len(myTuple) - 1**.
- Negative indexing counts from the end of the tuple:
  - **-1** refers to the last item
  - **-2** refers to the second last item, and so on.
- Tuples, like lists, are ordered, so each element has a unique index.

Example: Accessing Values in Tuples

```
# Accessing Tuple with Indexing

tup1 = tuple("Python")
# Access first element
first_element = tup1[0]

#Accessing a range of elements using slicing

slice1 = tup1[1:4]   # 2nd to 4th elements
slice2 = tup1[:3]    # First 3 elements


# 3. Tuple unpacking

tup2 = ("Data", "Science", "AI")
# Unpacking values into variables
x, y, z = tup2
# Advanced tuple unpacking using *
tup3 = (1, 2, 3, 4, 5)
a, *b, c = tup3

print("First element:", first_element)
print("Slice from index 1 to 3:", slice1)
print("First three elements:", slice2)
print("Tuple unpacking values:")
print(x)
print(y)
print(z)
print("Advanced unpacking:")
print("a:", a)
print("b:", b)
print("c:", c)
```

Output:

```
>>> %Run Lab_10.py
First element: P
Slice from index 1 to 3: ('y', 't', 'h')
First three elements: ('P', 'y', 't')
Tuple unpacking values:
Data
Science
```

LAB # 10

```
AI
Advanced unpacking:
a: 1
b: [2, 3, 4]
c: 5
>>>
```

Example: Iterating over a tuple with a for loop

```
# Iterating over a tuple with a for loo
numbers = (10, 20, 30, 40, 50)

print("Iterating over tuple of numbers:")
for num in numbers:
    print(num * 2)  # Multiply each number by 2 and print
```

Output

```
>>> %Run Lab_10.py
Iterating over tuple of numbers:
20
40
60
80
100
>>>
```

Tuple Functions:

cmp(tuple1, tuple2)	Compares elements of both tuples.
len(tuple)	Gives the total length of the tuple.
max(tuple)	Returns item from the tuple with max value.
min(tuple)	Returns item from the tuple with min value.
tuple(seq)	Converts a list into tuple.

Dictionary:

- A dictionary is a **container object** that stores a collection of key/value pairs.
- It enables **fast retrieval, deletion, and updating** of the value by using the key. A dictionary is also known as a **map**, which maps each key to a value.
- Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be **immutable.**
  - ✓ Keys are **case sensitive** which means same name but different cases of Key will be treated distinctly.

LAB # 10

- ✓ Keys must be **immutable** which means keys can be strings, numbers or tuples **but not lists.**
- ✓ **Duplicate keys** are not allowed and any duplicate key will overwrite the previous value.
- ✓ Internally uses hashing. Hence, operations like search, insert, delete can be performed in Constant Time

Syntax for creating dictionary:

```
# Creating a dictionary using curly braces
d1 = {201: 'Machine Learning', 202: 'Deep Learning', 203: 'AI'}     # Dictionary with integer keys
d2 = {'student': 'Bob', 'subject': 'Python', 'semester': 'Fall'}    # Dictionary with string keys

# Creating a dictionary using dict() constructor
d3 = dict(alpha = 100, beta = 200, gamma = 300)

#Checking the length of a dictionary
length_d1 = len(d1)
length_d2 = len(d2)
length_d3 = len(d3)

# Immutable keys example - Valid keys: strings, numbers, tuples
valid_dict = {('x', 'y'): "Coordinates", 500: "Score", "city": "Paris"}

# Invalid keys (for reference, will cause error if uncommented)
# invalid_dict = {[1, 2]: "List as key"}  # ❌ Not allowed because list is mutable

# Checking the type of dictionaries
type_d1 = type(d1)
type_d2 = type(d2)
type_d3 = type(d3)
type_valid_dict = type(valid_dict)

print("Dictionary 1:", d1)
print("Type of Dictionary 1:", type_d1)
print("Dictionary 2:", d2)
print("Type of Dictionary 2:", type_d2)
print("Dictionary 3 (using dict constructor):", d3)
print("Type of Dictionary 3:", type_d3)
print("Length of Dictionary 1:", length_d1)
print("Length of Dictionary 2:", length_d2)
print("Length of Dictionary 3:", length_d3)
print("Valid dictionary with tuple key:", valid_dict)
print("Type of valid dictionary:", type_valid_dict)
```

LAB # 10

Output:

```
>>> %Run Lab_10.py
Dictionary 1: {201: 'Machine Learning', 202: 'Deep Learning', 203: 'AI'}
Type of Dictionary 1: <class 'dict'>
Dictionary 2: {'student': 'Bob', 'subject': 'Python', 'semester': 'Fall'}
Type of Dictionary 2: <class 'dict'>
Dictionary 3 (using dict constructor): {'alpha': 100, 'beta': 200, 'gamma': 300}
Type of Dictionary 3: <class 'dict'>
Length of Dictionary 1: 3
Length of Dictionary 2: 3
Length of Dictionary 3: 3
Valid dictionary with tuple key: {('x', 'y'): 'Coordinates', 500: 'Score', 'city': 'Paris'}
Type of valid dictionary: <class 'dict'>
>>>
```

Example: Accessing Values in Dictionary

```
#Creating a dictionary
mycar = {
   "brand": "Tesla",
   "model": "Model 3",
   "year": 2022
}

model_value1 = mycar["model"]              # Accessing value using key
model_value2 = mycar.get("model")          # Accessing value using get() method

# Getting all keys
keys_view = mycar.keys()
print("Keys before adding 'color':", keys_view)

# Updating dictionary
mycar["color"] = "Red"
print("Keys after adding 'color':", keys_view)

# Getting all values
values_view = mycar.values()
print("Values before updating 'year':", values_view)

# Updating a value
mycar["year"] = 2025
print("Values after updating 'year':", values_view)

# Getting all items
items_view = mycar.items()
print("All items in dictionary:", items_view)

# Checking if a key exists
if "model" in mycar:
   print("Yes, 'model' is one of the keys in the dictionary")
```

LAB # 10

Output:

```
>>> %Run Lab_10.py
Keys before adding 'color': dict_keys(['brand', 'model', 'year'])
Keys after adding 'color': dict_keys(['brand', 'model', 'year', 'color'])
Values before updating 'year': dict_values(['Tesla', 'Model 3', 2022, 'Red'])
Values after updating 'year': dict_values(['Tesla', 'Model 3', 2025, 'Red'])
All items in dictionary: dict_items([('brand', 'Tesla'), ('model', 'Model 3'), ('year', 2025), ('color', 'Red')])
Yes, 'model' is one of the keys in the dictionary
>>>
```

Dictionary Methods:

Method	Description
clear()	Removes all the elements from the dictionary.
copy()	Returns a shallow copy of the dictionary.
fromkeys()	Returns a dictionary with the specified keys and value.
get()	Returns the value of the specified key.
items()	Returns a view object containing a tuple for each key-value pair.
keys()	Returns a view object containing the dictionary's keys.
pop()	Removes the element with the specified key and returns its value.
popitem()	Removes and returns the last inserted key-value pair.
setdefault()	Returns the value of the specified key. If the key does not exist, inserts it with the specified value.
update()	Updates the dictionary with the specified key-value pairs.
values()	Returns a view object containing all the values in the di

LAB # 10

EXERCISE

A. Point out the errors, if any, in the following Python programs.

1. Code

```
t = (1, 2, 3)
t.append(4)
t.remove(0)
del tup[0]
```

Output

2. Code

```
1user_0=['username':'efermi','first':'enrico','last':'fermi',]
for key, value in 1user_0.items():
    print("\nKey: " ,key)
    print("Value: " ,value)
```

Output:

B. What will be the output of the following programs:

1. Code

```
tuple1 = ("green", "red", "blue")
tuple2 = tuple([7, 1, 2, 23, 4, 5])
tuple3 = tuple1 + tuple2
print(tuple3)
tuple3 = 2 * tuple1
print(tuple3)
print(tuple2[2 : 4])
print(tuple1[-1])
```

Output

LAB # 10

2. Code

```
def main():
    d = {"red":4, "blue":1, "green":14, "yellow":2}
    print(d["red"])
    print(list(d.keys()))
    print(list(d.values()))
    print("blue" in d)
    print("purple" in d)
    d["blue"] += 10
    print(d["blue"])
main() # Call the main function
```

Output

C. Write Python programs for the following:

1. Buffet-style Restaurant Menu

- Create a program for a buffet-style restaurant that offers **five basic foods** stored in a tuple.
- Use **a for loop** to display the menu.
- The chef decides to **change two items** in the menu.
- Display the **updated menu.**
- *(Optional twist: Let the user suggest the new items.)*

2. "Guess the Capitals" Game

Write a program using a dictionary to store **pairs of countries and their capitals**.

- Randomly display the countries and ask the user to **guess the capital**.
- Keep track of the number of **correct** and **incorrect** answers.
- *(Optional twist: Add hints or emojis for correct/incorrect answers.)*

3. Favorite Places Dictionary

Create a dictionary called favorite_places.

- Choose **three people** as keys.
- Store **three favorite places** for each person as a list.
- Loop through the dictionary and print each person's name and their favorite places in a neat format:

```
Alice likes the following places:
- Paris
- Rome
- Barcelona
```