Python Tasks

**1 .) Describe the difference between mutable and immutable types in Python. Provide examples.**

**Ans.** Mutable types are those whose values can be hanged after they are created

1. List

Lists are mutable, meaning elements in a list can be modified.

Ex. my\_list = [1,2,3,4,5]

my\_list[1] = 20

print(my\_list) # output : [1,20,3,4,5]

1. Dictionary

Dictionaries are mutable, meaning the keys or values can be changed.

Ex. my\_dict = {“name” : “vikush”, “age” : “22”}

my\_dict[“age”] = 25

print(my\_dict)

1. Set

Sets are mutable, meaning elements can be added or removed

Ex. my\_set = {10,20,30}

my\_set.add(4)

my\_set.remove(2)

print(my\_set)

\* Immutable types are those whose values cannot be changed after they are created.

1. String

**String**: Strings cannot be modified after they are created.

Ex . my\_string = “hello”

my\_string = “Hello” # a new string is created instrad

print(my\_string) # output : Hello

2. Tuple

**Tuple**: Tuples cannot have their elements changed.

Ex. my\_tuple = (1,2,3)

print(my\_tuple[2]) # output : 3

Ex 2 . tuple1 = (1, 2)

tuple2 = (3, 4)

new\_tuple = tuple1+tuple2

print(new\_tuple) # output : (1.2.3.4)

**2.) Write a Python program to check if a number is a palindrome.**

def is\_palindrome(number):

num\_str = str(number)

return num\_str == num\_str[::-1]

number = int(input("Enter a number: "))

if is\_palindrome(number):

print(f"{number} is a palindrome.")

else:

print(f"{number} is not a palindrome.")

**3. What are Python decorators? Provide an example of how to use one. Answer: A decorator is a function that takes another function as input and extends or modifies its behaviour without changing the original function.**

Ans. A decorator is a function that takes another function as input and returns a new function that usually extends the behaviour of the original function

# Define the decorator function

def simple\_decorator(func):

def wrapper():

print("Something before the function.")

func() # Call the original function

print("Something after the function.")

return wrapper

# Define functions with different names

@simple\_decorator

def hello():

print("Hello!")

@simple\_decorator

def goodbye():

print("Goodbye!")

# Call the decorated functions

hello()

goodbye()

**4. Explain the purpose of \_\_init\_\_ in Python classes.**Ans.

The primary purpose of \_\_init\_\_ is to initialize the object's state by assigning values to its attributes when the object is created.

Ex. class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def greet(self):

print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# Create an object of the Person class

person1 = Person("Alice", 30)

# Call the greet method

person1.greet()

# output: Hello, my name is Alice and I am 30 years old.

**5. What is the difference between a list and a tuple? When would you use each?**Ans.

**List**: Defined with square brackets [], like [1, 2, 3].

**Tuple**: Defined with parentheses (), like (1, 2, 3).

**List**: A list is **mutable**, meaning you can change its elements (add, remove, or modify items).

**Tuple**: A tuple is **immutable**, meaning once you create a tuple, you cannot change its elements.

**List**: Lists have many built-in methods like append(), remove(), insert(), and sort(), to modify the list.

**Tuple**: Tuples have fewer methods and cannot be changed. They only support methods like count() and index().

**List**: Lists are generally slower than tuples because they are mutable (you can modify them).

**Tuple**: Tuples are faster and use less memory since they are immutable.

**When to Use Each**:

* **List**: Use lists when you need to change the collection later (e.g., adding or removing items).
* **Tuple**: Use tuples when the data should not change (e.g., for fixed data, coordinates, or data that shouldn't be altered).

Ex. List

my\_list = [1, 2, 3]

my\_list.append(4) # You can add new items

my\_list[0] = 10 # You can modify existing items

print(my\_list) # Output: [10, 2, 3, 4]

Ex. Tuple

my\_tuple = (1, 2, 3)

# my\_tuple[0] = 10 # This will give an error because tuples are immutable

print(my\_tuple) # Output: (1, 2, 3)