ASSIGNMENT 12

# Q1

Assigning a value to a string's indexed character does violate Python's string immutability. Strings in Python are immutable, which means that once a string is created, its contents cannot be changed. If we try to assign a new value to a specific indexed character of a string, it will result in a TypeError.

# Q2

Using the += operator to concatenate strings does not violate Python's string immutability. While strings are immutable, the += operator does not directly modify the original string. Instead, it creates a new string object that is the result of concatenating the original string with the new string. Therefore, the original string remains unchanged, and a new string object is created.

# Q3

In Python, there are two main ways to index a character in a string:

Using positive indexing: Characters in a string can be accessed by their position using positive integers. The indexing starts from 0, where the first character is at index 0, the second character at index 1, and so on.

Using negative indexing: Characters in a string can also be accessed using negative integers. Negative indexing starts from -1, where the last character is at index -1, the second-to-last character at index -2, and so on.

# Q4

Indexing refers to accessing a specific character in a string by its position. It is done using square brackets [] and the index of the desired character.

Slicing refers to extracting a substring from a string by specifying a range of indices. It is done using the colon : operator inside the square brackets []. The syntax for slicing is string[start:end:step], where start is the starting index, end is the ending index (exclusive), and step is the step size.

# Q5

An indexed character in Python is of type string. It is a string of length 1 that represents a single character. The data form of a slicing-generated substring is also a string.

# Q6

In Python, strings are sequences of characters. Each character in a string is itself a string of length 1. Therefore, the "type" of a string is also a string. Python does not have a separate character type.

# Q7

The concatenation operator +: It concatenates two strings together to create a new string that contains the contents of both strings.

The repetition operator \*: It repeats a string a specified number of times to create a new string with multiple copies of the original string.

# Q8

The benefit of first checking the target string with in or not in before using the index method to find a substring is to avoid a ValueError if the substring is not found. If we use the in or not in operator to check for the presence of the substring before calling the index() method, we can handle the case where the substring is not found gracefully without raising an exception.

# Q9

The == operator: It checks if two strings are equal and returns True if they are, False otherwise.

The != operator: It checks if two strings are not equal and returns True if they are not, False otherwise.

The in operator: It checks if a substring is present in a string and returns True if it is, False otherwise.

The not in operator: It checks if a substring is not present in a string and returns True if it is not, False otherwise.

The startswith() method: It checks if a string starts with a specified prefix and returns True if it does, False otherwise.

The endswith() method: It checks if a string ends with a specified suffix and returns True if it does, False otherwise.