Q1. Does assigning a value to a string's indexed character violate Python's string immutability?

Sol:-

Yes, assigning a value to a string's indexed character directly violates Python's string immutability. In Python, strings are immutable objects, which means that once a string is created, its contents cannot be changed.

my\_string = "Hello, World!"

my\_string[0] = 'h' # Raises a TypeError: 'str' object does not support item assignment

To modify a string, you need to create a new string with the desired changes. String concatenation or other string manipulation methods can be used to achieve this.

my\_string = "Hello, World!"

modified\_string = 'h' + my\_string[1:]

print(modified\_string) # Output: hello, World!

Q2. Does using the += operator to concatenate strings violate Python's string immutability? Why or why not?

Sol:-

No, using the += operator to concatenate strings does not violate Python's string immutability. This may seem contradictory at first, but it's because the += operator behaves differently when used with strings.

Python internally creates a new string that combines the original string and the concatenated part.

my\_string = "Hello, "

my\_string += "World!"

Q3. In Python, how many different ways are there to index a character?

Sol:-

Positive Indexing: This is the most common and straightforward way to index a character. It uses positive integers starting from 0 to access characters from the beginning of the string. For example, string[0] retrieves the first character, string[1] retrieves the second character, and so on.

Negative Indexing: Negative indexing allows you to access characters from the end of the string. The index -1 refers to the last character, -2 refers to the second-to-last character, and so on. For example, string[-1] retrieves the last character, string[-2] retrieves the second-to-last character, and so on.

Slicing: Slicing allows you to extract a portion of a string by specifying a range of indices. The syntax is string[start:end], where the start index is inclusive and the end index is exclusive. For example, string[2:5] retrieves a substring consisting of characters from index 2 to index 4.

Stride Slicing: Stride slicing is an extension of slicing that allows you to specify a step value, indicating the increment between characters. The syntax is string[start:end:step]. For example, string[0:10:2] retrieves characters at even indices from 0 to 9.

Iteration: You can iterate over each character in a string using a loop or iterator constructs such as for loops or iter() function. This allows you to access each character individually without explicitly using indexing.

Q4. What is the relationship between indexing and slicing?

Sol:-

Indexing refers to the process of accessing a single element within a sequence by specifying its position using an index. In Python, indexing starts from 0 for the first element, and you can use positive or negative integers to access elements from the beginning or end of the sequence, respectively. For example, string[0] retrieves the first character of a string, while list[-1] retrieves the last element of a list.

Slicing, on the other hand, allows you to extract a portion of a sequence by specifying a range of indices. It involves specifying a starting index and an ending index (exclusive), separated by a colon (:). The result is a new sequence containing the elements within that range. For example, string[2:5] retrieves a substring consisting of characters from index 2 to index 4, while list[1:4] retrieves a new list containing elements from index 1 to index 3.

Q5. What is an indexed character's exact data type? What is the data form of a slicing-generated substring?

Sol:-

For a string, an indexed character is of type str. It represents a single character within the string.

If the original sequence is a string, the slicing-generated substring will be a new string.

Q6. What is the relationship between string and character "types" in Python?

Sol:-

In Python, a string is a sequence of characters. Each character within a string is considered a separate element of the string sequence. Therefore, the relationship between strings and characters in Python is that a string is composed of one or more individual characters.

Q7. Identify at least two operators and one method that allow you to combine one or more smaller strings to create a larger string.

Sol:-

String Concatenation Operator (+): The + operator allows you to concatenate multiple strings together. When used with strings, it combines the strings in left-to-right order to create a new larger string.

String Formatting Operator (%): The % operator, also known as the string formatting operator, allows you to create a larger string by inserting values into a format string. This operator is used with the % character followed by a tuple or dictionary of values.

String Join Method (join()): The join() method is used to concatenate a sequence of strings with a specified delimiter. It takes an iterable as an argument and returns a new string by concatenating the strings in the iterable, separated by the specified delimiter.

Q8. What is the benefit of first checking the target string with in or not in before using the index method to find a substring?

Sol:-

The benefit of first checking the target string with the in or not in operator before using the index() method to find a substring is to avoid raising an exception when the substring is not found.

If the substring is not found, the index() method raises a ValueError.

Q9. Which operators and built-in string methods produce simple Boolean (true/false) results?

Sol:-

Operators:

Equality Operator (==): Compares two strings and returns True if they are equal, False otherwise.

Inequality Operator (!=): Compares two strings and returns True if they are not equal, False otherwise.

Less Than Operator (<): Compares two strings lexicographically and returns True if the left string is less than the right string, False otherwise.

Greater Than Operator (>): Compares two strings lexicographically and returns True if the left string is greater than the right string, False otherwise.

Built-in String Methods:

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

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

isalpha(): Checks if a string contains only alphabetic characters and returns True if it does, False otherwise.

isdigit(): Checks if a string contains only numeric digits and returns True if it does, False otherwise.

islower(): Checks if all characters in a string are lowercase and returns True if they are, False otherwise.

isupper(): Checks if all characters in a string are uppercase and returns True if they are, False otherwise.

isspace(): Checks if a string contains only whitespace characters and returns True if it does, False otherwise.