**FSDS MAY BATCH 2022(Python Assignment -12)**

**Submitted by: Shubham Tiwari**

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

Ans: Yes, assigning a value to a string's indexed character violates Python's string immutability. In Python, strings are immutable, meaning that their values cannot be changed after they are created. Attempting to change a string by assigning a value to an indexed character will result in a TypeError.

However, it's possible to create a new string by using slicing and concatenation, for example:

**string = "Hello World"**

**string = string[:5] + "Python" + string[-5:]**

This will create a new string with the value "**Hello Python World".**

Alternatively, we can use a list of characters and then join them to form a string, for example:

**string\_list = list("Hello World")**

**string\_list[5] = "P"**

**string = "".join(string\_list)**

This will also create a new string with the value "Hello Python World".

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

Ans: Using the += operator to concatenate strings does not violate Python's string immutability because the += operator creates a new string object in memory, rather than modifying the original string.

In python, strings are immutable, which means once a string object is created, it cannot be modified. When you use the += operator to concatenate two strings, it creates a new string object in memory that contains the concatenated value, and assigns it to the variable. The original strings are not modified, they are still the same as they were before the operation.

For example, the following code creates two strings, "hello" and "world", and concatenates them using the += operator:

**a = "hello"**

**b = "world"**

**a += b**

After this operation, the variable "a" will contain the concatenated string "helloworld", and the original strings "hello" and "world" will remain unchanged.

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

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

1. Using the square bracket notation, where the index of the desired character is placed within the brackets.
2. Using the string method **str.\_\_getitem\_\_()**, which also takes the index of the desired character as an argument.

Q4. What is the relationship between indexing and slicing?

Ans: **In Python, indexing and slicing are two ways to access a subset of a sequence (such as a string, list, or tuple).**

Indexing refers to accessing a single element of a sequence by its position, using the square bracket notation. For example, if the variable "x" contains the string "Hello", x[1] would return "e", the second character in the string.

Slicing, on the other hand, refers to accessing a range of elements of a sequence by specifying a start and end position. The start and end positions are separated by a colon, and the slice is inclusive of the start position and exclusive of the end position. For example, x[1:4] would return "ell", the second, third and fourth characters in the string.

So indexing is a way of getting one element of the sequence while slicing is a way of getting a sub sequence from a start index to end index.

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

Ans: In Python, when indexing a character in a string, the returned value is of the data type **str**, which represents a string of characters. For example, if the variable "x" contains the string "Hello", x[1] would return "e" which is of the data type **str**.

On the other hand, when slicing a string to generate a substring, the returned value is also of the data type **str**. For example, if the variable "x" contains the string "Hello", x[1:4] would return "ell" which is also of the data type **str**.

It's worth noting that when slicing or indexing any sequence types like list or tuple, the returned value is of the same type as the original sequence, for example if the variable "x" contains the list [1,2,3,4,5], x[1:4] would return [2,3,4] which is a list of int.

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

Ans: In Python, a string is a sequence of characters. A character is a single letter, number, punctuation mark, or other symbol that is used to represent text.

A string can contain any number of characters, including zero characters (an empty string). Each character in a string is stored as an individual element in the string's memory representation. Python provides several ways to access and manipulate the individual characters in a string.

For example, we can use square brackets [] to access individual characters in a string. The following code will print the first character of the string "hello":

**s = "hello"**

**print(s[0]) # prints "h"**

We can also use the **string[start:stop:step]** slicing syntax to access a sub-sequence of characters within a string. The following code will print the substring "ell" from the string "hello":

**s = "hello"**

**print(s[1:4]) # prints "ell"**

In addition to the above, python provide many built-in string methods to manipulate the string like **find()**, **replace()**, **upper()**, **lower()** and many more.

In summary, a string in Python is a sequence of characters, and a character is a single element within a string. Python provides a variety of ways to access and manipulate the individual characters within a string, including indexing and slicing.

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

Ans: The "+" operator allows you to combine two or more strings by concatenation. For example, "Hello" + " " + "world" would result in the string "Hello world".

1. The "join()" method allows you to combine a list or tuple of strings into a single string, using a specified delimiter. For example, " ".join(["Hello", "world"]) would result in the string "Hello world".
2. The "\*" operator allows to repeat a string n times. For example, "Hello" \* 3 would result in the string "**HelloHelloHello".**

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?

Ans: Checking if a substring is present in a target string using the "in" or "not in" operator before using the "index()" method can be beneficial because it allows you to avoid potential ValueError exceptions. The "index()" method raises a ValueError exception if the substring is not found in the target string.

By checking for the substring's presence in the target string using the "in" or "not in" operator, you can determine whether or not the substring is present before attempting to use the "index()" method. If the substring is not present, you can handle the case accordingly, such as by returning a default value or performing a different operation. This can make the code more efficient and easy to understand.

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

Ans: The "in" and "not in" operators are used to check if a substring is present in a target string and return a Boolean value (True if the substring is found, False if it is not). For example, "world" in "Hello world" would return True, while "foo" not in "Hello world" would return True.

1. The "==" and "!=" operators are used to check if two strings are equal or not equal, respectively, and return a Boolean value. For example, "Hello" == "world" would return False.
2. The "isalnum()" method returns True if all characters in the string are alphanumeric (letters and digits), and False otherwise.
3. The "isalpha()" method returns True if all characters in the string are alphabetic (letters) and False otherwise.
4. The "isdigit()" method returns True if all characters in the string are digits and False otherwise.
5. The "isspace()" method returns True if all characters in the string are whitespace characters and False otherwise.
6. The "istitle()" method returns True if the string is in title case, meaning that the first character of each word is uppercase and the remaining characters are lowercase, and False otherwise.