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 violates the python string's immutability.

e.g.

a = 'fig'

print(a[2])

a[2] = 'd'

**---------------------------------------------------------------------------**

**TypeError** Traceback (most recent call last)

**~\AppData\Local\Temp/ipykernel\_6516/2286811595.py** in <module>

**----> 1** a**[2]** **=** **'d'**

**TypeError**: 'str' object does not support item assignment

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

Sol. += operator, doesnt violate the python's string immutability, because we are concatenating another string to the end of the first, not trying to replace any of the characters in the strings in between.

Eg:

a="hello"

b='world'

a+=b

print(a)

>>> helloworld

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

Sol. We can access characters in a String in Two ways :

1)Accessing Characters by Positive Index Number.

2)Accessing Characters by Negative Index Number.

Eg:

a ='hello world'

print(a[5])

print(a[-7])

>> w

l

Q4. What is the relationship between indexing and slicing?

Sol. Using indexes, the slicing operation is done

e.g string[startindex:stopindex:increment]

a = "Hello World"

print(type(a[3]))

print(type(a[1:5]))

>>> <class 'str'>

<class 'str'>

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

Sol. The data type of the indexed character is a string. The data form of slicing-generated substring is also a string.

a = "Hello World"

print(type(a[3]))

print(type(a[1:5]))

>>> <class 'str'>

<class 'str'>

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

Sol. In python there is no character data type, a character is a string of length one. Strings are sequences of character data.

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. Two operators : +, +=

Method : join()

print('Hello'+'World')

a='Hello'

b='World'

a+=b

print(a)

print(''.join([a,b]))

>>> HelloWorld

HelloWorld

HelloWorldWorld

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 is that it returns True or False by using the 'in' or 'not in’

Eg:

*print('Hello' in 'Hello World')*

*print('Eh' in 'Hello World')*

*print('Eh' not in 'Hello World')*

*print('Hello' not in 'Hello World')*

*>>>* True

False

True

False

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

Sol. Operators : >,<,>=,<=,==

built-in string methods : isalpha, isdecimal, isdigit, islower, isupper, isspace, isprintable etc