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

Ans:- Yes In Python, **strings are made immutable so that programmers cannot alter the contents of the object** (even by mistake).  **Python** uses zero-based **indexing**, .

**5 Ways to Find the Index of a Substring in Strings in Python**

1. str.find()
2. str.rfind()
3. str.index()
4. str.rindex()
5. re.search()

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

Ans:- **+= operator**  
  
The in-place operator += can also be used. The string on the right is concatenated after the string variable on the left. If you want to add a string to the end of a string variable, use the += operator.

It does not violate python’s string immutability Python's string objects are immutable, so each concatenation generates a new string instead of modifying the existing one in place.

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

Ans:-

**5 Ways to Find the Index of a Substring in Strings in Python**

1. str.find() = str.find(sub,start,end)
2. str.rfind()=str.rfind(sub,start,end)
3. str.index()=str.index(sub)
4. str.rindex()=str.rindex(**"an"**,0,4)
5. re.search()=re.**search**(pattern, string, flags=0)

Q4. What is the relationship between indexing and slicing?

Ans:- “Indexing” means referring to an element of an iterable by its position within the iterable. “Slicing” means **getting a subset of elements from an iterable based on their indices.**  **Indexing**: **Indexing** is used **to** obtain individual elements. **Slicing**: **Slicing** is used **to** obtain a sequence **of** elements.

Slicing in Python is **similar to indexing but returns a sequence of items instead of a single item**. The indices used for slicing are also zero-based. There are two variants of the slicing syntax: sequence[start:stop] and sequence[start:stop:step]. ... That's how you get a shallow copy of a sequence

Indexing in Python is **a way to refer the individual items within an iterable by its position**.you can directly access your elements of choice within an iterable and do various operations depending on your needs. Program slicing can **be used in debugging to locate source of errors more easily**.

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

Ans:-Indexed character’s exact data type is string.

Slicing-generated substring data form is string.

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

Ans:-  Python, Strings are arrays of bytes representing Unicode characters. However, **Python does not have a character** data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

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:- Using the + operator

The simplest and most common method of concatenating a string is using the plus symbol**(“+”).**

### Using the **\*** operator

The asterisk **(\*)** operator is used when you want to concatenate the same string repeatedly.

a = "Python"

print(a \* 3)

**PythonPythonPython**

a = “Python”

b = “is”

c = “cool”

print(a + b + c)

**Pythoniscool**

### Using the **join()** method

The **join()** method is the most flexible way of concatenating strings in Python. If you have many strings and you want to combine them together, use the**join()**method. It is a string method and the most interesting thing about join() is that you can combine strings using a separator. It works on iterators like lists, tuples, string, dictionaries, etc.

a = "Welcome"

b = "to"

c = "Python"

print(“-”.join([a,b,c]))

**Welcome-to-Python**

### Using the **%** operator

The modulus operator **(“%”)** can be used for both string formatting and string concatenation. It is useful for cases in which you need to combine strings and also perform basic formatting.

An example to illustrate concatenation of string using**“%”** operator:

a = "Apple"

b = "Shake"

print(“% s % s” % (a, b))

**Apple Shake**

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:- python String find() method returns the lowest index of the substring if it is found in a given string. If it is not found then it returns -1.

***Syntax:*** *str.find(sub, start, end)*

### Using index() method returns the starting index of the substring passed as a parameter.However, a major con is that it returns a valueError in case the substring does not exist. We can solve this by using a Try Except. **Syntax of index():string.index(value,start,stop)**

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

Ans:- **Logical operators**: A logical operator compares 2 Boolean (logical) expressions and return a Boolean result. Example: the logical operation true AND false returns the Boolean value false.

Boolean values respond to logical operators and / or

>>> True and False  
False

Format Boolean values  
  
The string representation of a Boolean is **either "True" for a true value or "False" for a false value**. The string representation of a Boolean value is defined by the read-only TrueString and FalseString fields. You use the ToString method to convert Boolean values to strings.

my\_string = "Hello World"

my\_string.isalnum()False

my\_string.isalpha()False

my\_string.isdigit()False

my\_string.endswith('d')True

my\_string.startswith('H')True