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

In python, the string data types are immutable. Which means a string value cannot be updated. We can verify this by trying to update a part of the string which will led us to an error.

# Can not reassign

t= "Tutorialspoint"

print type(t)

t[0] = "M"

When we run the above program, we get the following output −

t[0] = "M"

TypeError: 'str' object does not support item assignment

We can further verify this by checking the memory location address of the position of the letters of the string.

.

x = 'banana'

for idx in range (0,5):

print x[idx], "=", id(x[idx])

When we run the above program we get the following output. As you can see above a and a point to same location. Also N and N also point to the same location.

b = 91909376

a = 91836864

n = 91259888

a = 91836864

n = 91259888

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

Concatenation of strings create a new string and allocate new memory for every concatenation! So using the += operator to concatenate strings violate Python's string immutability.

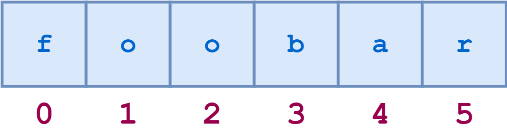
Q3. In Python, how many different ways are there to index a character?

Often in programming languages, individual items in an ordered set of data can be accessed directly using a numeric index or key value. This process is referred to as indexing.

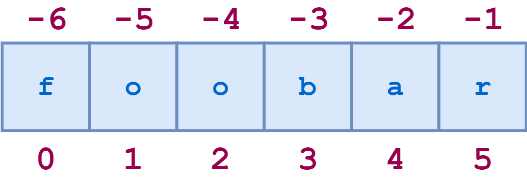
In Python, strings are ordered sequences of character data, and thus can be indexed in this way. Individual characters in a string can be accessed by specifying the string name followed by a number in square brackets ([]).

String indexing in Python is zero-based: the first character in the string has index 0, the next has index 1, and so on. The index of the last character will be the length of the string minus one.

For example, a schematic diagram of the indices of the string 'foobar' would look like this:



String indices can also be specified with negative numbers, in which case indexing occurs from the end of the string backward: -1 refers to the last character, -2 the second-to-last character, and so on. Here is the same diagram showing both the positive and negative indices into the string 'foobar':



Q4. What is the relationship between indexing and slicing?

Indexing: Indexing is used to obtain individual elements.

Slicing: Slicing is used to obtain a sequence of elements.

Indexing and Slicing can be be done in Python Sequences types like list, string, tuple, range objects. The order in which elements are inserted is preserved in sequence type, allowing us to access its elements via indexing and slicing.

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

indexing returns a string — Python has no special type for a single character. It is just a string of length 1. A slicing-generated substring is also a string data form.

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

In Python, strings are ordered sequences of character data, and thus can be indexed in this way. Individual characters in a string can be accessed by specifying the string name followed by a number in square brackets ( [] ).

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

The most common among them is using the plus (“+”) operator. You can combine both string variables and string literals using the “+” operator.

However, there’s another method that allows an easy way of concatenating multiple strings. It uses the in-place (+=) operator. The in-place operator concatenates the sequence with the right operand, and the result gets assigned to that sequence.

Code:

a = "Artificial "

b = "Intelligence"

a += b

print(a)

Another way of concatenating multiple strings in Python is just by writing string literals consecutively:

Code:

a = "Red""Green""Blue"

print(a)

You can concatenate a list of strings into one string using the join() method. It takes a character as a delimiter string. If you use an empty string as the delimiter, the list of strings will be simply concatenated without any separator.

Let us see an example of concatenating a list of strings using the join() function:

Code:

a = ["Apple", "Orange", “Banana”, “Mango”]

print(“\n”.join(a))

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?

If substring is not present in the string and we try to find the substring using the index method it will give a value error: substring not found.

Code:

text = 'Python is fun'

# find the index of the

result = text.index('the')

print(result)

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

Method True if

str.isalnum() String consists of only alphanumeric characters (no symbols)

str.isalpha() String consists of only alphabetic characters (no symbols)

str.islower() String’s alphabetic characters are all lower case

str.isnumeric() String consists of only numeric characters

str.isspace() String consists of only whitespace characters

str.istitle() String is in title case

str.isupper() String’s alphabetic characters are all upper case