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

String’s indexed character cannot to be assigned a New value , as Strings are **immutable.**

x='string'

x[2]='t'

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

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

Input **In [2]**, in <cell line: 2>**()**

1 x='string'

**----> 2** x[2]='t'

**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?

**+=** operator is used to concatenate strings, it does not violate Python’s string immutability Property. Because doing so creates a new association with data and variable. E.g. str\_1="a" and str\_1+="b. effect of this statements to create string ab and reassign it to variable str\_1, any string data is not actually modified.

str\_1 = 'a'

print(id(str\_1))

str\_1 += 'b'

print(id(str\_1))

2372207768304

2372288515312

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

2 ways are there

a. Positive indexing

b. negative indexing

Q4. What is the relationship between indexing and slicing?

We can access elements of sequence datatypes by using slicing and indexing. Indexing is used to obtaining individual element while slicing for sequence of elements.

**Indexing**

s='saturday'

s[2]

't'

**Slicing:**

**s[1:4]**

'atu'

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

 Indexed characters and sliced substrings have datatype **String**.

Eg: s='saturday'

type(s[2])

str

**type(s[1:4])**

str

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

Object that contains sequence of character datatypes are called 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.

**+**, **+=** and **\*** allow to combine one or more smaller strings to create a larger string. **<string>.join(<sep>)** method joins element of iterable type like list and tuple to get a combined string.

string = 'iNeuron '

string += 'Full Stack Data Science'

print(string + ' FSDS')

print('FSDS '\*3)

print(" ".join(['I','N','E','U','R','O','N']))

print(" ".join(('I','N','E','U','R','O','N')).lower())

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?

Checking the target string with **in** or **not** operators before using the index method to find a substring just helps confirming availability of substring and thus avoid raising of **ValueError.**

string = "ineuron"

string.index('x')

string.index('u')

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

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

Input **In [16]**, in <cell line: 2>**()**

1 string = "ineuron"

**----> 2** string.index('x')

3 string.index('u')

**ValueError**: substring not found

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

* **in**
* **not**
* **<string>.isalpha()**
* **<string>.isalnum()**
* **<string>.isdecimal()**
* **<string>.isdigit()**
* **<string>.islower()**
* **<string>.isnumeric()**
* **<string>.isprintable()**
* **<string>.isspace()**
* **<string>.istitle()**