Assignment 12 Solutions

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

Ans: In Python, strings are made immutable so that programmers cannot alter the contents of the object (even by mistake). This avoids unnecessary bugs. we can assign a value to a string's indexed character it doesn't violate Python's string immutability because for ding that we first have to split the string word into characters which leaves us with simple character string we can replace it easily with some other character but still it will be string even if the replaced value will be an interger or floting number or anything.

In [2]:

```python
#Example:
str = "as"
print(str.replace("a","1"))
print(type(str.replace("a","1")))
```

```
1s
<class 'str'>
```

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

Ans: `+=` 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 statement is to create string `ab` and reassign it to variable `str_1` . Any string data is not actually modified.

In [3]:

```python
str_1 = 'a'
print(id(str_1))
str_1 += 'b'
print(id(str_1)) # Does not Modify existing string, Creates a New String Object
```

```
2037839480560
2037914319600
```

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

Ans: Each of a string's characters corresponds to an index number and each character can be accessed using their index number. We can access characters in a String in Two ways :

- Accessing Characters by Positive Index Number.
- Accessing Characters by Negative Index Number.

In [4]:

```python
in_string = "iNeuron Full Stack Data Science"
print(in_string[9],in_string[10],in_string[2]) # Positive Indexing
print(in_string[-1],in_string[-5],in_string[-2]) # Negative Indexing
```

```
u l e
e i c
```

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

In [5]:

```python
in_string = "iNeuron Full Stack Data Science"
print(in_string[1],in_string[3],in_string[5]) # Indexing
print(in_string[1:15]) # Slicing
```

```
N u o
Neuron Full St
```

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

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

In [6]:

```
in_string = "iNeuron Full Stack Data Science"
print(type(in_string[3])) # Indexing -> str
print(type(in_string[1:10])) # Indexing -> str
```

```
<class 'str'>
<class 'str'>
```

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

Ans: In 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 & one method that allow you to combine one or more smaller strings to create a larger string ?

Ans: `+` , `+=` 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.

In [7]:

```
in_string = 'iNeuron '
in_string += 'Full Stack Data Science'
print(in_string + ' FSDS')
print('FSDS '*3)
print(" ".join(['I','N','E','U','R','O','N'])) # List Iterable
print(" ".join(('I','N','E','U','R','O','N')).lower()) # Tuple Iterable
```

```
iNeuron Full Stack Data Science FSDS
FSDS FSDS FSDS
I N E U R O N
i n e u r o n
```

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 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.`**

```
Example:
in_string = "ineuron"
in_string.index('x') # Raises ValueError
in_string.index('u') # 3
```

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

Ans: The String Operators and built-in methods to Produce Simple Boolean (True/False) Results are:

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