1. Tuples

Like list, Tuple is a Linear Data Structure and follows zero-based indexing.

Unlike list, Tuple is Immutable, that is the value of tuple elements cannot be altered.

Tuples are denoted by comma-separated list of elements within parentheses.

```
num = (1,2,3)
print(num)
```

```
(1, 2, 3)
```

```
type(num)
```

```
tuple
```

```
('abc' , 'def' , 'hij')
```

```
('abc', 'def', 'hij')
```

Like list, Tuple can contain elements with dissimilar data type.

```
('abs' , 40 , True)
```

```
('abs', 40, True)
```

Unlike List, Tuple of one element must include a comma after the element.

```
a= (1,) # data type of variable b will be tuple
a
```

```
(1,)
```

```
a1=[2]
```

```
b= (1) # data type of variable b will be int
b
```

```
1
```

Empty Tuple is denoted by empty pairs of parenthesis without space.

```
c=()
```

Like List, Tuple can be accessed by using an index value within square brackets.

```
a= (2,5,6,8)
print('value at index 2: ',a[2])
```

```
value at index 2:  6
```

```
a[2]=3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-64084f7c3ad6> in <module>()
```

Since tuple is immutable, therefore operations that try to modify the tuple are not allowed that is following operations are invalid:

delete, update, insert, and append.

SEARCH STACK OVERFLOW

2. Strings

String is a Linear Data Structure and follows zero-based indexing.

Like Tuple, String is Immutable

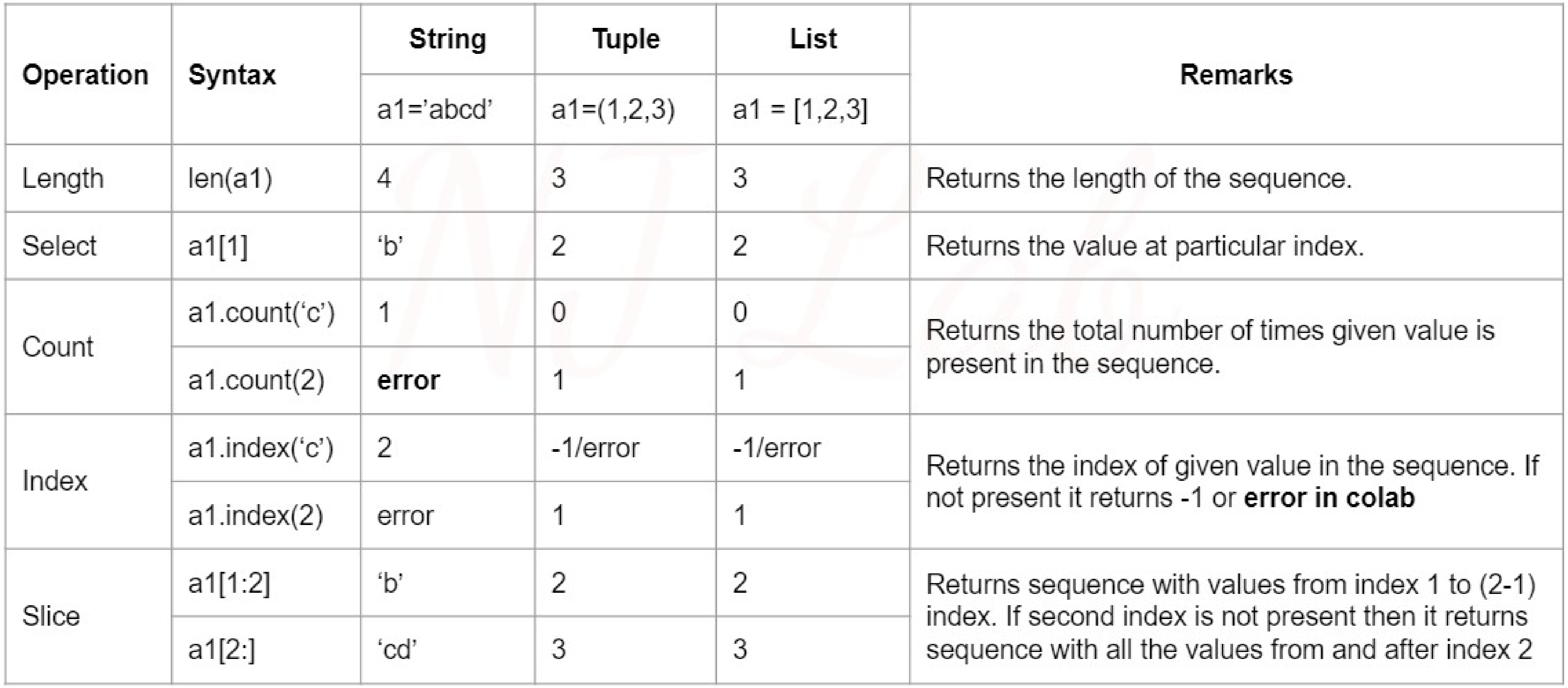
Since string is immutable, therefore operations that try to modify the string are not allowed that is following operations are invalid: delete, update, insert, and append.

```
str = 'Hello World'
str
```

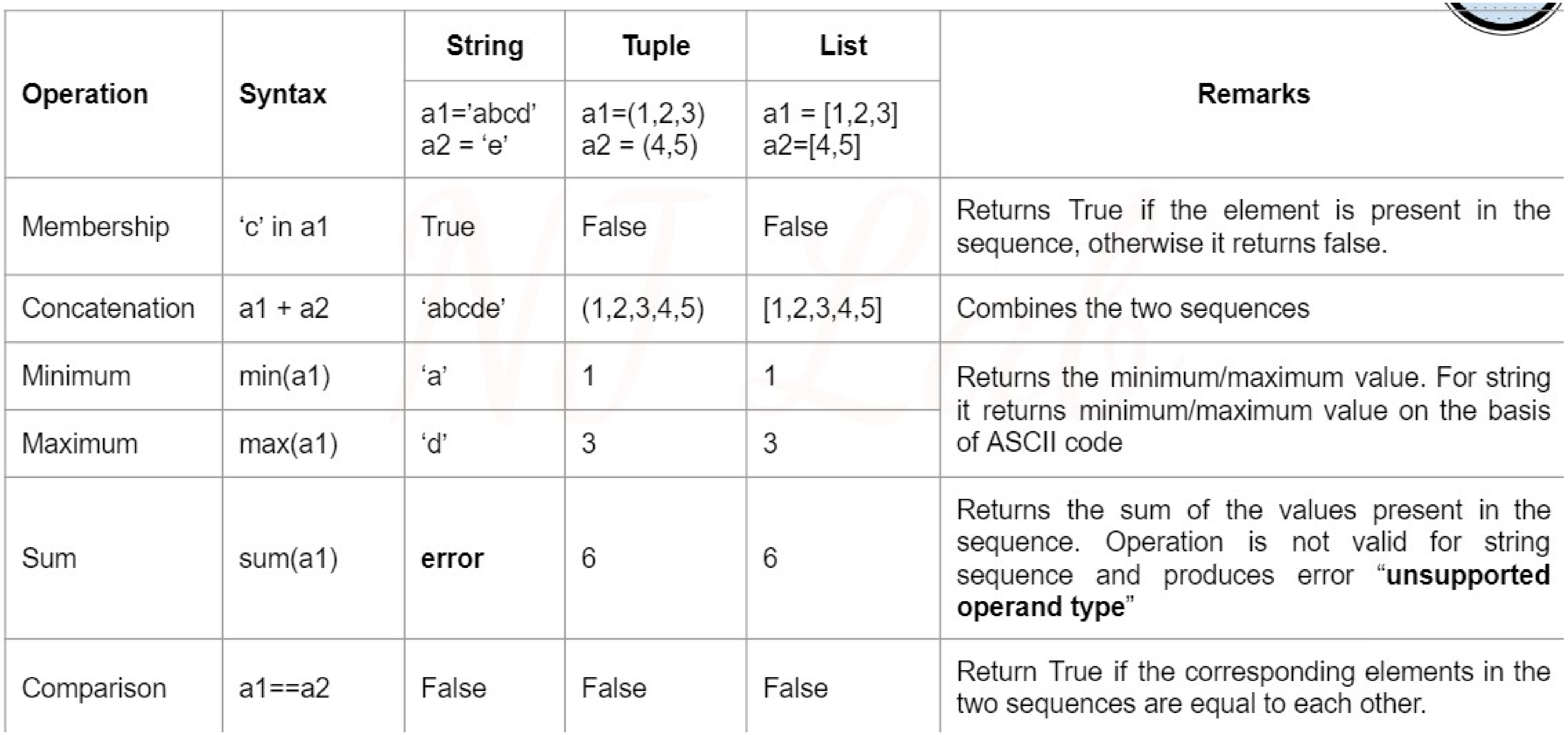
```
'Hello World'
```

```
s = 'a'
s
```

```
'a'
```



Operation	Syntax	**String**	**Tuple**	**List**	**Remarks**
		a1='abcd'	a1=(1,2,3)	a1 = [1,2,3]	
Length	len(a1)	4	3	3	Returns the length of the sequence.
Select	a1[1]	'b'	2	2	Returns the value at particular index.
Count	a1.count('c')	1	0	0	Returns the total number of times given value is present in the sequence.
	a1.count(2)	**error**	1	1	
Index	a1.index('c')	2	-1/error	-1/error	Returns the index of given value in the sequence. If not present it returns -1 or **error in colab**
	a1.index(2)	error	1	1	
Slice	a1[1:2]	'b'	2	2	Returns sequence with values from index 1 to (2-1) index. If second index is not present then it returns sequence with all the values from and after index 2
	a1[2:]	'cd'	3	3	



Operation	**Syntax**	**String**	**Tuple**	**List**	**Remarks**
		a1='abcd' a2 = 'e'	a1=(1,2,3) a2 = (4,5)	a1 = [1,2,3] a2=[4,5]	
Membership	'c' in a1	True	False	False	Returns True if the element is present in the sequence, otherwise it returns false.
Concatenation	a1 + a2	'abcde'	(1,2,3,4,5)	[1,2,3,4,5]	Combines the two sequences
Minimum	min(a1)	'a'	1	1	Returns the minimum/maximum value. For string it returns minimum/maximum value on the basis of ASCII code
Maximum	max(a1)	'd'	3	3	
Sum	sum(a1)	**error**	6	6	Returns the sum of the values present in the sequence. Operation is not valid for string sequence and produces error "**unsupported operand type**"
Comparison	a1==a2	False	False	False	Return True if the corresponding elements in the two sequences are equal to each other.