Operations on String, List and Tuple

1. **Python Strings:** The string can be defined as the sequence of characters. String handling in Python is a straightforward task since Python provides built-in functions and operators to perform operations in the string.

   In the case of string handling, the operator + is used to concatenate two strings as the operation "hello"+" python" returns "hello python".

   The operator *is known as a repetition operator as the operation "Python"* 2 returns 'Python Python'.

In [1]:

```
d='String in double or single quotes'
d
```

Out[1]:

```
'String in double or single quotes'
```

In [2]:

```
s="""A multiple lines
    string using triple
    quotes."""
print(s)
```

```
A multiple lines
    string using triple
    quotes.
```

In [3]:

```
a=5
a
b=6
b
```

Out[3]:

```
6
```

In [4]:

```
str1 = 'string str1'
str2 = 'how you doing?' #string str2

#printing first two character using slice operator
print(str1[0:2])
#Get the characters from position 1 to position 4 in str1 (called slicing of string)
print(str1[1:5])

#printing 5th character of the string
print(str1[5])

#printing the string twice
print(str1*2)

#printing the concatenation of str1 and str2
str1+str2
```

```
st
trin
g
string str1string str1
```

Out[4]:

```
'string str1how you doing?'
```

In [5]:

```
# Negative Indexing
str2[-1]
```

Out[5]:

```
'?'
```

In [6]:

```
#The len() function returns the length of a string:
print(len(str2))
```

```
14
```

In [7]:

```
#The strip() method removes any whitespace from the beginning or the end:
c= "      Hi     Everyone    "
print(c.strip())
```

```
Hi     Everyone
```

In [8]:

```
c.lower()          #The lower() method returns the string in lower case
c.upper()          #The upper() method returns the string in upper case
```

Out[8]:

```
'      HI     EVERYONE    '
```

In [9]:

```
c.replace("H","J")
c.split("E")    #replace and split
```

Out[9]:

```
['      Hi     ', 'veryone    ']
```

Check String: To check if a certain phrase or character is present in a string, we can use the keywords **in** or **not in**.

In [10]:

```
txt = "The rain in Spain stays mainly in the plain"
x = "abc" in txt
print(x)
```

```
False
```

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

In [11]:

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item 567 for 49.95 dollars.
```

In [12]:

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

```
I want to pay 49.95 dollars for 3 pieces of item 567.
```

1. **List** : Python Lists are similar to arrays in C. However, the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

   We can use slice [:] operators to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

In [13]:

```
list2 = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

In [14]:

```
list2[2:5]
```

Out[14]:

```
['cherry', 'orange', 'kiwi']
```

In [15]:

```
list2[2:]
```

Out[15]:

```
['cherry', 'orange', 'kiwi', 'melon', 'mango']
```

In [16]:

```
list2[:4]
```

Out[16]:

```
['apple', 'banana', 'cherry', 'orange']
```

In [17]:

```
#This example returns the items from index -4 (included) to index -1 (excluded)
#Remember that the last item has the index -1

print(list2[-4:-1])
```

```
['orange', 'kiwi', 'melon']
```

In [18]:

```
#change a particular value in list
list2[3]="tomatoes"
list2
```

Out[18]:

```
['apple', 'banana', 'cherry', 'tomatoes', 'kiwi', 'melon', 'mango']
```

In [19]:

```
# List Concatenation using + operator
list1=[1,2,3]
print (list1 + list2)
```

```
[1, 2, 3, 'apple', 'banana', 'cherry', 'tomatoes', 'kiwi', 'melon', 'mango']
```

In [20]:

```
# List repetation using * operator
print (list1 * 3)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

1. **Tuple**: A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

   A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

In [21]:

```
tup  = ("hi", "Python", 2)

# Tuple slicing
print (tup[1:])
print (tup[0:1])

# Tuple concatenation using + operator
print (tup + tup)

# Tuple repatation using * operator
print (tup * 3)

# Adding value to tup. It will throw an error.
#tup[2] = "hi"
```

```
('Python', 2)
('hi',)
('hi', 'Python', 2, 'hi', 'Python', 2)
('hi', 'Python', 2, 'hi', 'Python', 2, 'hi', 'Python', 2)
```

In [ ]:

```

```