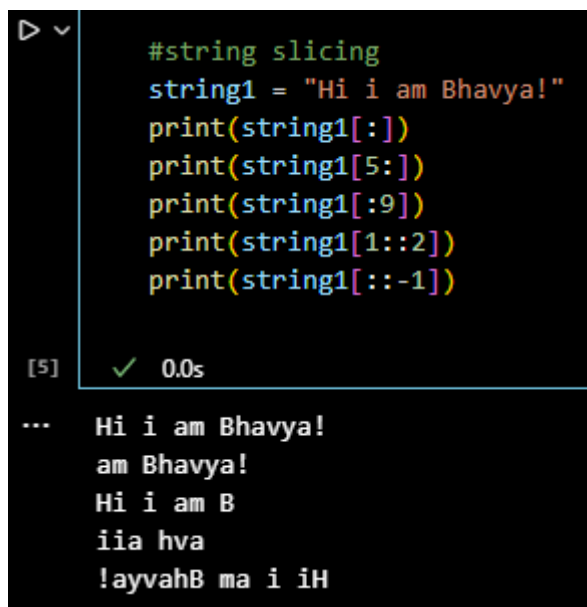


# PYTHON ASSIGNMENT 2

## DATA STRUCTURES

GITHUB : <https://github.com/bha411/PW-Skills>

1. String slicing is a technique used in Python in for extracting a portion of a string. It basically separates a string into parts based on the index specified by range. This can be done by using a colon and square brackets, which can 3 inputs namely the start, stop and the step index.



```
#string slicing
string1 = "Hi i am Bhavya!"
print(string1[:])
print(string1[5:])
print(string1[:9])
print(string1[1::2])
print(string1[::-1])
```

[5] ✓ 0.0s

```
... Hi i am Bhavya!
    am Bhavya!
    Hi i am B
    iia hva
    !ayvahB ma i iH
```

2. Lists in Python are a collection of ordered elements which can accept any kinds of datatypes be it integer/string/float/complex. It can even store different data structures like nested lists / dictionary / sets / tuples etc. One of the main features of a list is that they are mutable, meaning they can be altered after their creation. We can add/subtract or replace the elements in a list unlike the tuples or strings. There are a lot of methods or functions which are exclusive to lists. Some of the common functions include are append to add, pop or remove to delete the elements and replace to replace them. We can iterate though the elements of a list and can even run loops to access them. Each element in a list if assigned with indices.

### 3. ACCESSING:

- We can access the elements of list with their indices.
- For example:

```
▷ ▾  
    List1 = ["Bhavya","Gupta","PWSkills"]  
    print(List1[0])  
    print(List1[1])  
    print(List1[-1])  
    List1  
[10] ✓ 0.0s  
... Bhavya  
    Gupta  
    PWSkills  
... ['Bhavya', 'Gupta', 'PWSkills']
```

### MODIFYING:

- We can change the elements of a list by simply assigning the new values to the index.
- For example:

```
▷ ▾  
    List1[2] = "Alakh Pandey Sir"  
    List1[1] = "Data Science Course"  
    List1  
[11] ✓ 0.0s  
... ['Bhavya', 'Data Science Course', 'Alakh Pandey Sir']
```

- We can even add new elements and there are multiple ways to do so.
- We can use **append** method which takes the element as the input
- **Insert** method is used to insert an element at a specified index.
- The **extend** method can be used to merge 2 lists and add the elements of the previous list to the new list.

```
▷ ▾  
    List2 = ["Machine Learning","Artificial Intelligence"]  
    print(List1)  
    List1.append("Physics Wallah")  
    List1.insert(1,"Gupta")  
    List1.extend(List2)  
    print(List1)  
[16] ✓ 0.0s Python  
... ['Bhavya', 'Data Science Course', 'Alakh Pandey Sir']  
    ['Bhavya', 'Gupta', 'Data Science Course', 'Alakh Pandey Sir', 'Physics  
    Wallah', 'Machine Learning', 'Artificial Intelligence']
```

## DELETING:

- We can delete the elements of a list in several ways.
- One of the ways is to use the **del** method to remove the element at a particular index.
- Then comes the **pop** method, which by default deletes the last element unless an index is specified.
- We can also use the **remove** function which can delete the first occurrence of a specific element.
- The **clear** method directly eliminates all the elements in a list.

```
▶ ▾  
    del List1[-1]  
    List1.pop()  
    List1.pop(1)  
    List1.remove("Data Science Course")  
    List2.clear()  
    print(List1)  
    print(List2)  
[17] ✓ 0.0s  
... ['Bhavya', 'Alakh Pandey Sir', 'Physics Wallah']  
    []
```

## 4. LIST vs TUPLE

LIST	TUPLE
<ul style="list-style-type: none"><li>• Mutable (can be changed)</li></ul>	<ul style="list-style-type: none"><li>• Immutable (cannot be changed once created).</li></ul>
<ul style="list-style-type: none"><li>• Defined using square brackets []</li></ul>	<ul style="list-style-type: none"><li>• Defined using parentheses ().</li></ul>
<ul style="list-style-type: none"><li>• Has more built-in methods like append(), extend(), insert(), remove(), pop(), clear(), sort(), reverse().</li></ul>	<ul style="list-style-type: none"><li>• Has only two built-in methods: count() and index().</li></ul>
<ul style="list-style-type: none"><li>• Consumes more memory because it is mutable and has additional methods.</li></ul>	<ul style="list-style-type: none"><li>• Generally consumes less memory than a list of the same size due to its immutability.</li></ul>
<ul style="list-style-type: none"><li>• Collection of ordered elements.</li></ul>	<ul style="list-style-type: none"><li>• Sequence of elements separated by commas.</li></ul>

5. A set in Python is an unordered collection of elements, which does not allow storage of duplicate values . It can accept duplicate values but will store only the unique elements. It works on the same principles as in the mathematical terms. We can perform various functions like union, intersection and difference. This is also a mutable data structure. They are especially useful when working with data that requires uniqueness and when performing operations like union, intersection, and difference.

- Unordered collection of mixed elements.

```
Set1 = {1,2,3,4,"Bhavya","PWSkills"}
Set1
[18] ✓ 0.0s
... {1, 2, 3, 4, 'Bhavya', 'PWSkills'}
```

- Mutable with only unique elements.

```
Set1.add("Bhavya")
Set1.add(5)
Set1.add(1)
Set1
[19] ✓ 0.0s
... {1, 2, 3, 4, 5, 'Bhavya', 'PWSkills'}
```

- Membership testing.

```
print(1 in Set1)
print(2 in Set1)
print(10 in Set1)
[24] ✓ 0.0s
... True
    True
    False
```

This way they can be used to check the membership of a particular element in the set, or only record the unique items or, to perform mathematical operations like & and |.

6. Tuples and sets both are important data structures and serve multiple purposes. They can be used in the following ways.

- Since tuples are immutable, they can be used to store the data that shouldn't change. This way we can store integral data which cannot be modified once stored. The main advantage of this quality is that they can be used to store the coordinates of a specific system.

```
PointA = (12,1.3,0)
PointB = (3,3,3)
PointA,PointB
[26] ✓ 0.0s
... ((12, 1.3, 0), (3, 3, 3))
```

- Secondly we can use them to store heterogeneous data, meaning that we can store any kind of datatype be it string or integers or lists or any data struct.

```
Details_X = ("Bhavya Gupta",100,22,170.5,[1,2,3,4])
Details_X
[27] ✓ 0.0s
... ('Bhavya Gupta', 100, 22, 170.5, [1, 2, 3, 4])
```

- They can also be used as multiple keys for a value for a dictionary, like:

```
Student_Details = {("Bhavya","x","y"): "Pass", ("a","b","c"): "Fail"}
print(Student_Details[("Bhavya","x","y")])
[40] ✓ 0.0s
... Pass
```

- Sets on the other hand can however take duplicate values as input, but will only store unique ones. It automatically removes the duplicate values. It can be used in the cases when there are multiple records of the same element but we need to perform analysis and check only the unique items.

```
Bhavya_tech_stack = {"Python","Machine learning","Deep learning","Python","AI"}
Bhavya_tech_stack
[41] ✓ 0.0s
... {'AI', 'Deep learning', 'Machine learning', 'Python'}
```

- Secondly it can be used to check the membership of a particular element in the set. This use case is important as when it comes to list, they can take a lot of time finding the element as they search thorough the list linearly, which will make it slow for large amount of elements.

```
▶ ▾  
    if "Python" in Bhavya_tech_stack:  
        print("Eligible to apply")  
  
    if "C" not in Bhavya_tech_stack:  
        print("Sorry, you are ineligible")  
[42] ✓ 0.0s  
... Eligible to apply  
    Sorry, you are ineligible
```

- Another Important use case is that it can be used to find the common and different items belonging to 2 different sets. This way we can find the intersection and the difference between 2 sets.

```
▶ ▾  
    Student_tech_stack = {"C", "Python", "AI"}  
    print(Bhavya_tech_stack & Student_tech_stack)  
    print(Bhavya_tech_stack - Student_tech_stack)  
[45] ✓ 0.0s  
... {'AI', 'Python'}  
    {'Machine learning', 'Deep learning'}
```

- We can also check if the duplicate elemnts were ever added to the sets.

```
▶ ▾  
    data = [1, 2, 3, 4, 4, 5]  
    if len(data) != len(set(data)):  
        print("Duplicates found")  
    else:  
        print("NO duplicates found")  
[52] ✓ 0.0s  
... Duplicates found
```

7. Since dictionaries are mutable in python, we can add, modify and delete the elements. Inorder to add an element, we need to provide the key and assign the value to it. Similarly to modify an elemt, we just need to provide the key and then we can assign the new value to it . When it comes to deleting the elements, we have several ways to do so, Either we can remove the key or the value or the key value pair directly else we can delete the whole dict.

```
My_Dict = {"Name": "Bhavya", "Age": 22, "Language": "Python"}
My_Dict
[53] ✓ 0.0s
... {'Name': 'Bhavya', 'Age': 22, 'Language': 'Python'}
```

```
My_Dict["College"] = "Manipal University Jaipur"
My_Dict["CGPA"] = 8.5
My_Dict
[54] ✓ 0.0s
... {'Name': 'Bhavya',
    'Age': 22,
    'Language': 'Python',
    'College': 'Manipal University Jaipur',
    'CGPA': 8.5}
```

```
My_Dict["Name"] = "Bhavya Gupta"
My_Dict
[55] ✓ 0.0s
... {'Name': 'Bhavya Gupta',
    'Age': 22,
    'Language': 'Python',
    'College': 'Manipal University Jaipur',
    'CGPA': 8.5}
```

```
del My_Dict["College"]
print(My_Dict.pop("CGPA"))
My_Dict
[56] ✓ 0.0s
... 8.5
... {'Name': 'Bhavya Gupta', 'Age': 22, 'Language': 'Python'}
```

8. In python, the need for immutable Keys for a dictionary is very crucial, this is because there is no other way to maintain the integrity of the data. This makes the dictionaries more efficient and more reliable. If the keys were to be changed, the dictionary will throw error if the older key is used and this way keeping a track will be very cumbersome. This quality has a huge benefit to the lookup method of the dictionaries and thus make them fast. There is no way to make mutable data structures a key for the dictionaries as they will throw an “Unhashable” Error. Thus we can only use tuples when it comes to multiple data for the keys.

```
Student_Details = {("Bhavya","x","y"): "Pass", ("a","b","c"): "Fail"}
print(Student_Details[("Bhavya","x","y")])
```

[40] ✓ 0.0s

... Pass

```
invalid_dict = {[1, 2, 3]: "This is a list"}
```

[57] ✗ 0.0s

... -----

TypeError Traceback (most recent call last)

Cell In[57], line 1

----> 1 invalid\_dict = {[1, 2, 3]: "This is a list"}

TypeError: unhashable type: 'list'

```
invalid_dict = {"a": "string": "string"}
```

[58] ✗ 0.0s Python

... -----

TypeError Traceback (most recent call last)

Cell In[58], line 1

----> 1 invalid\_dict = {"a": "string": "string"}

TypeError: unhashable type: 'dict'