What is a Data Structure?

Organizing, managing and storing data is important as it enables easier access and efficient modifications. Data Structures allows you to organize your data in such a way that enables you to store collections of data, relate them and perform operations on them accordingly.

Types of Data Structures in Python

1. List

2. Dictionary

3. Tuple

4. Set

1.List

In [11]:

```
#List
list1 = []
print(list1)
list1 = [1, 2, 3, 'wooo', 3.50]
print(list1)
```

```
[]
[1, 2, 3, 'wooo', 3.5]
```

In [12]:

```
# Adding Elements
list2 = [1, 2, 3]
print(list2)
list2.append([443, 10]) #add as a single element
print(list2)
list2.extend([234, 'other_example']) #add as different elements
print(list2)
list2.insert(1, 'other_example') #add element i
print(list2)
```

```
[1, 2, 3]
[1, 2, 3, [443, 10]]
[1, 2, 3, [443, 10], 234, 'other_example']
[1, 'other_example', 2, 3, [443, 10], 234, 'other_example']
```

In [15]:

```
# Deleting Elements

list2 = [1, 2, 3, 'example', 2.34, 12, 23]
del list2[4] #delete element at index 4
print(list2)
list2.remove('example') #remove element with value
print(list2)
result = list2.pop(2) #pop element from list
print('Popped Element: ', result, ' List remaining: ', list2)
list2.clear() #empty the list
print(list2)
```

```
[1, 2, 3, 'example', 12, 23]
[1, 2, 3, 12, 23]
Popped Element:  3  List remaining:  [1, 2, 12, 23]
[]
```

In [14]:

```
#Accessing Elements

my_list = [1, 2, 3, 'example', 3.132, 10, 30]
for element in my_list: #access elements one by one
    print(element)
print(my_list) #access all elements
print(my_list[3]) #access index 3 element
print(my_list[0:2]) #access elements from 0 to 1 and exclude 2
print(my_list[::-1]) #access elements in reverse
```

```
1
2
3
example
3.132
10
30
[1, 2, 3, 'example', 3.132, 10, 30]
example
[1, 2]
[30, 10, 3.132, 'example', 3, 2, 1]
```

In [17]:

```
#Other Functions

list3 = [1, 2, 3, 4, 5, 6]
print(len(list3)) #find length of list
print(list3.index(3)) #find index of element that occurs first
print(list3.count(3)) #find count of the element
print(sorted(list3)) #print sorted list but not change original
list3.sort(reverse=True) #sort original list
print(list3)
```

```
6
2
1
[1, 2, 3, 4, 5, 6]
[6, 5, 4, 3, 2, 1]
```

2. Dictionary

In [19]:

```
# Dictionary
dict1 = {} #empty dictionary
print(dict1)
dict1 = {1: 'Python', 2: 'c++'} #dictionary with elements
print(dict1)
```

```
{}
{1: 'Python', 2: 'c++'}
```

In [21]:

```
#Changing and Adding key, value pairs

dict2 = {'First': 'Python', 'Second': 'c++'}
print(dict3)
dict3['Second'] = 'java' #changing element
print(dict2)
dict2['Third'] = 'Ruby' #adding key-value pair
print(dict2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-2c771198f700> in <module>
      2 
      3 dict2 = {'First': 'Python', 'Second': 'c++'}
----> 4 print(dict3)
      5 dict3['Second'] = 'java' #changing element
      6 print(dict2)

NameError: name 'dict3' is not defined
```

In [8]:

```
#Deleting key, value pairs

my_dict = {'First': 'Python', 'Second': 'Java', 'Third': 'Ruby'}
a = my_dict.pop('Third') #pop element
print('Value:', a)
print('Dictionary:', my_dict)
b = my_dict.popitem() #pop the key-value pair
print('Key, value pair:', b)
print('Dictionary', my_dict)
my_dict.clear() #empty dictionary
print('n', my_dict)
```

```
Value: Ruby
Dictionary: {'First': 'Python', 'Second': 'Java'}
Key, value pair: ('Second', 'Java')
Dictionary {'First': 'Python'}
n {}
```

In [22]:

```
# Accessing Elements

dict3 = {'First': 'Python', 'Second': 'c++', 'Third': 'java'}
print(dict3.keys())
print(dict3.values())
print(dict3.items())
print(dict3.get('First'))
```

```
dict_keys(['First', 'Second', 'Third'])
dict_values(['Python', 'c++', 'java'])
dict_items([('First', 'Python'), ('Second', 'c++'), ('Third', 'java')])
Python
```

In [23]:

```
#Other Functions

my_dict = {'First': 'Python', 'Second': 'Java', 'Third': 'Ruby'}
print(my_dict.keys()) #get keys
print(my_dict.values()) #get values
print(my_dict.items()) #get key-value pairs
print(my_dict.get('First'))
```

```
dict_keys(['First', 'Second', 'Third'])
dict_values(['Python', 'Java', 'Ruby'])
dict_items([('First', 'Python'), ('Second', 'Java'), ('Third', 'Ruby')])
Python
```

4.Tuple

In [24]:

```
tuple1 = (1, 2, 3) #create tuple
print(tuple1)
```

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

In [26]:

```
# Accessing Elements
tuple2 = (1, 2, 3, 'hello') #access elements
for x in tuple2:
    print(x)
print(tuple2)
print(tuple2[0])
print(tuple2[:])
print(tuple2[3][2])
```

```
1
2
3
hello
(1, 2, 3, 'hello')
1
(1, 2, 3, 'hello')
l
```

In [28]:

```
# Appending Elements
tuple1 = (1, 2, 3)
tuple1 = tuple1 + (4, 5, 6) #add elements
print(tuple1)
```

```
(1, 2, 3, 4, 5, 6)
```

In [ ]:

4. Sets

In [31]:

```
set1 = {1, 2, 3, 4, 6, 7, 8} #create set
print(set1)
```

```
{1, 2, 3, 4, 6, 7, 8}
```

In [16]:

```
# Adding elements

my_set = {1, 2, 3}
my_set.add(4) #add element to set
print(my_set)
```

```
{1, 2, 3, 4}
```

In [33]:

```
# Operations in sets

set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
print(set1.union(set2), '-', set1 | set2)# Union of two sets

print(set1.intersection(set2), '-', set1 & set2)# Intersection of two sets

print(set1.difference(set2), '-', set1 - set2)# Difference of two sets

print(set1.symmetric_difference(set2), '-', set1 ^ set2)# Symmetric_difference of
set1.clear()
print(set1)
```

```
{1, 2, 3, 4, 5, 6} - {1, 2, 3, 4, 5, 6}
{3, 4} - {3, 4}
{1, 2} - {1, 2}
{1, 2, 5, 6} - {1, 2, 5, 6}
set()
```

In [ ]: