Python DataStructures or Collections (Arrays)

- There are four collection data types in the Python programming language:
- List is a collection which is ordered and changeable. Allows duplicate members.[,]
- Tuple is a collection which is ordered and unchangeable(immutable). Allows duplicate members.(,)
- Set is a collection which is unordered and unindexed. No duplicate members.{,}
- Dictionary is a collection which is unordered, changeable and indexed. No duplicate members. {name:',Govardhan',age:23}

In [3]:
```
mylist = ["Green","Green", "Red", "Yello",1,1,2,3,4,True,False]
print(mylist)
type(mylist)
```

```
['Green', 'Green', 'Red', 'Yello', 1, 1, 2, 3, 4, True, False]
```
Out[3]:
```
list
```

In [4]:
```
mylist[3]
```

Out[4]:
```
'Yello'
```

In [5]:
```
a_list = ["test1","test2",1,2,3,4,2,3,4,5,6,6,7,8,8,9,10]
print(a_list[-10 : -5])
```

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

In [6]:
```
a_list = [1,2,3,4,5,6,7,8,9,10]
print(a_list[-6:-2])
```

```
[5, 6, 7, 8]
```

In [7]:
```
print(a_list[-1])
```

```
10
```

In [8]:
```
mylist = ["Green", "Red", "Yello"]
print('Minus one (-1) Index : ',mylist[-1])
print('Minus two (-2) Index : ',mylist[-2])
print('Minus three (-3) Index : ',mylist[-3])
```

```
Minus one (-1) Index :  Yello
Minus two (-2) Index :  Red
Minus three (-3) Index :  Green
```

In [9]:
```
print(mylist[0])
```

```
Green
```

Range of Indexes

- lists can be indexed and sliced
- You can specify a range of indexes by specifying where to start and where to end the range.
- When specifying a range, the return value will be a new list with the specified items.

In [10]:
```
mylist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
mylist[0:3]
```

Out[10]:
```
['apple', 'banana', 'cherry']
```

In [11]:
```
mylist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
mylist[:6]
```

Out[11]:
```
['apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon']
```

In [12]:
```
mylist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
mylist[2:]
```

Out[12]:
```
['cherry', 'orange', 'kiwi', 'melon', 'mango']
```

Updating existing item using `indexes`

In [13]:
```
mylist = ["Jan", "Feb", "Mar","Apr"]
mylist[3]="Nov"
mylist
```

Out[13]:
```
['Jan', 'Feb', 'Mar', 'Nov']
```

In [14]:
```
mylist = ["Jan", "Feb", "Mar","Apr"]
print("Before Updating : ",mylist[0:])
mylist[3] = "Dec"
print("After Updating : ",mylist[0:])
```

```
Before Updating :  ['Jan', 'Feb', 'Mar', 'Apr']
After Updating :  ['Jan', 'Feb', 'Mar', 'Dec']
```

In [15]:
```
help(mylist)
```

```
Help on list object:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __repr__(self, /)
```

```
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the list.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(self, /)
|      Return the size of the list in memory, in bytes.
|
|  append(self, object, /)
|      Append object to the end of the list.
|
|  clear(self, /)
|      Remove all items from list.
|
|  copy(self, /)
|      Return a shallow copy of the list.
|
|  count(self, value, /)
|      Return number of occurrences of value.
|
|  extend(self, iterable, /)
|      Extend list by appending elements from the iterable.
|
|  index(self, value, start=0, stop=9223372036854775807, /)
|      Return first index of value.
|
|      Raises ValueError if the value is not present.
|
|  insert(self, index, object, /)
|      Insert object before index.
|
|  pop(self, index=-1, /)
|      Remove and return item at index (default last).
|
|      Raises IndexError if list is empty or index is out of range.
|
|  remove(self, value, /)
|      Remove first occurrence of value.
|
|      Raises ValueError if the value is not present.
|
|  reverse(self, /)
|      Reverse *IN PLACE*.
|
|  sort(self, /, *, key=None, reverse=False)
|      Sort the list in ascending order and return None.
|
|      The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
|      order of two equal elements is maintained).
|
|      If a key function is given, apply it once to each list item and sort them,
|      ascending or descending, according to their function values.
|
|      The reverse flag can be set to sort in descending order.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
```

```
|  __class_getitem__(...) from builtins.type
|      See PEP 585
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```

In [16]:
```
list=[1,2,2]
help(list)
```

```
Help on list object:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __repr__(self, /)
```

```
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the list.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(self, /)
|      Return the size of the list in memory, in bytes.
|
|  append(self, object, /)
|      Append object to the end of the list.
|
|  clear(self, /)
|      Remove all items from list.
|
|  copy(self, /)
|      Return a shallow copy of the list.
|
|  count(self, value, /)
|      Return number of occurrences of value.
|
|  extend(self, iterable, /)
|      Extend list by appending elements from the iterable.
|
|  index(self, value, start=0, stop=9223372036854775807, /)
|      Return first index of value.
|
|      Raises ValueError if the value is not present.
|
|  insert(self, index, object, /)
|      Insert object before index.
|
|  pop(self, index=-1, /)
|      Remove and return item at index (default last).
|
|      Raises IndexError if list is empty or index is out of range.
|
|  remove(self, value, /)
|      Remove first occurrence of value.
|
|      Raises ValueError if the value is not present.
|
|  reverse(self, /)
|      Reverse *IN PLACE*.
|
|  sort(self, /, *, key=None, reverse=False)
|      Sort the list in ascending order and return None.
|
|      The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
|      order of two equal elements is maintained).
|
|      If a key function is given, apply it once to each list item and sort them,
|      ascending or descending, according to their function values.
|
|      The reverse flag can be set to sort in descending order.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
```

```
|  __class_getitem__(...) from builtins.type
|      See PEP 585
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```

- Inserting or appending new item or value into LIST using `append`

In [17]:
```
mylist = ["Jan", "Feb", "Mar","Apr"]
print('before adding new value :',mylist)
mylist.append(["May","June","Jul"])
print('After adding new value : ',mylist)
```

```
before adding new value : ['Jan', 'Feb', 'Mar', 'Apr']
After adding new value :  ['Jan', 'Feb', 'Mar', 'Apr', ['May', 'June', 'Jul']]
```

- if we want add any value inbetween we can go with `insert` specifying index value

In [18]:
```
mylist = ["Jan", "Mar", "Apr"]
print('Before inserting :',mylist)
mylist.insert(1, "Feb")
print('After inserting : ',mylist)
```

```
Before inserting : ['Jan', 'Mar', 'Apr']
After inserting :  ['Jan', 'Feb', 'Mar', 'Apr']
```

In [19]:
```
list_b =['val1','val1','va2','val3']
list_b.index('val3')
```

Out[19]:
```
3
```

In [20]:
```
help(mylist)
```

```
Help on list object:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __repr__(self, /)
```

```
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the list.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(self, /)
|      Return the size of the list in memory, in bytes.
|
|  append(self, object, /)
|      Append object to the end of the list.
|
|  clear(self, /)
|      Remove all items from list.
|
|  copy(self, /)
|      Return a shallow copy of the list.
|
|  count(self, value, /)
|      Return number of occurrences of value.
|
|  extend(self, iterable, /)
|      Extend list by appending elements from the iterable.
|
|  index(self, value, start=0, stop=9223372036854775807, /)
|      Return first index of value.
|
|      Raises ValueError if the value is not present.
|
|  insert(self, index, object, /)
|      Insert object before index.
|
|  pop(self, index=-1, /)
|      Remove and return item at index (default last).
|
|      Raises IndexError if list is empty or index is out of range.
|
|  remove(self, value, /)
|      Remove first occurrence of value.
|
|      Raises ValueError if the value is not present.
|
|  reverse(self, /)
|      Reverse *IN PLACE*.
|
|  sort(self, /, *, key=None, reverse=False)
|      Sort the list in ascending order and return None.
|
|      The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
|      order of two equal elements is maintained).
|
|      If a key function is given, apply it once to each list item and sort them,
|      ascending or descending, according to their function values.
|
|      The reverse flag can be set to sort in descending order.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
```

```
|  __class_getitem__(...) from builtins.type
|      See PEP 585
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```

extend

extend takes a list as an argument and appends all of the elements

In [21]:
```
mylist = ["Jan", "Feb", "Mar","Apr"]
monthlist = ["May","June","Jul"]
mylist.extend(monthlist)
mylist
```

Out[21]:
```
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'June', 'Jul']
```

List Length

- To determine how many items a list has, use the len() function:

In [23]:
```
lenlist = [1,2,3,4,5,6,7,8,9,10]
len(lenlist)
```

Out[23]:
```
10
```

SORT

- sort arranges the elements of the list from low to high

In [24]:
```
unsortlist = ['a','d','e','c','f','h','g','b']
unsortlist.sort()
unsortlist
```

Out[24]:
```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

In [25]:
```
unsortlist = ['a','d','e','c','f','h','g','b']
unsortlist.sort(reverse=True)
unsortlist
```

Out[25]:
```
['h', 'g', 'f', 'e', 'd', 'c', 'b', 'a']
```

In [26]:
```
# just reversing entire list items.
list =[1,2,3,3,1,2,8,9,10,4,5]
list.reverse()
list
```

Out[26]:
```
[5, 4, 10, 9, 8, 2, 1, 3, 3, 2, 1]
```

- Removing individual items from LISt we ca use `remove` method with value

In [27]:
```
mylist = ["Jan", "Feb", "Mar","Apr"]
print('Before Removing :',mylist)
mylist.remove("Feb")
print('After removing :',mylist)
```

```
Before Removing : ['Jan', 'Feb', 'Mar', 'Apr']
After removing : ['Jan', 'Mar', 'Apr']
```

- Remove last item from list is `pop()` it will be removed last item from list

In [28]:
```
mylist = ["jan", "feb", "mar","apr"]
mylist.append("dec")
removed_var=mylist.pop()
print(mylist)
print(removed_var)
```

```
['jan', 'feb', 'mar', 'apr']
dec
```

In [29]:
```
mylist = ["jan", "feb", "mar","apr"]
removed_var=mylist.pop(1)
print(mylist)
print(removed_var)
```

```
['jan', 'mar', 'apr']
feb
```

In [30]:
```
# len()
# del
var=55
print('before deleting ' ,var)
del var
print('after deleting ',var)
```

```
before deleting  55
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [30], in <cell line: 6>()
      4 print('before deleting ' ,var)
      5 del var
----> 6 print('after deleting ',var)

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

In [31]:
```
mylist = ["jan", "feb", "mar"]
del mylist[0]
print(mylist)
```

```
['feb', 'mar']
```

In [32]:
```
mylist = ["jan", "jan", "jan"]
mylist.clear()
print(mylist)
```

```
[]
```

In [33]:
```
mylist = ["jan", "jan", "jan"]
del mylist
print(mylist)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [33], in <cell line: 3>()
      1 mylist = ["jan", "jan", "jan"]
      2 del mylist
----> 3 print(mylist)

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

In [34]:
```
a_list = [1,2,3,4,5,6,7,8,9,9,10]
```

In [35]:
```
sum(a_list)
```

Out[35]:
```
64
```

In [36]:
```
mylist = ["jan", "feb", "mar"]
mylist.clear()
print(mylist)
```

```
[]
```

In [39]:
```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
list1.extend(list2)
print(list1)
print(type(list1))
```

```
['a', 'b', 'c', 1, 2, 3]
<class 'list'>
```

COPY

- You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.
- There are ways to make a copy, one way is to use the built-in List method copy().

In [40]:
```
thislist = ["apple", "banana", "cherry"]
copylist=thislist
thislist.append("kiwi")
print('thislist values : ',thislist)
print('copylist values : ',copylist)
```

```
thislist values :  ['apple', 'banana', 'cherry', 'kiwi']
copylist values :  ['apple', 'banana', 'cherry', 'kiwi']
```

In [41]:
```
print(list2)
```

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

In [42]:
```
thislist = ["Krishna", "Naveen", "Ram","Govardhan","Ram"]
mylist = thislist.copy()
print(mylist)
type(mylist)
```

```
['Krishna', 'Naveen', 'Ram', 'Govardhan', 'Ram']
```

Out[42]:
```
list
```

In [45]:
```
a=[1,2,3,4,5]
```

In [46]:
```
sum(a)
```

Out[46]:
```
15
```

In [47]:
```
# list Count
list_count = ['a', 'e', 'i', 'o', 'i', 'u','I','i']

print(list_count.count('i'))
```

```
3
```

In [48]:
```
list1 = ['apple', 'banana', 'cherry']
list1.sort()
print(list1)
```

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

In [49]:
```
a.sort(reverse = True)
a
```

Out[49]:
```
[5, 4, 3, 2, 1]
```

In [50]:
```
cars = [1,44,2,34,100,36]

cars.sort()
print(cars)
```

```
[1, 2, 34, 36, 44, 100]
```

In [51]:
```
 list.clear()
```

In [52]:
```
# sorted will support single datatype
list_num = [4,3,1,2,5,6]
sorted(list_num)
```

Out[52]:
```
[1, 2, 3, 4, 5, 6]
```