IOTA ACADEMY

Become Job Ready...

Lists

Data Structure:

Data Structures are a way of organizing data so that it can be accessed more efficiently depending upon the situation.

A data structure is a collection of data elements (such as numbers or characters—or even other data structures) that is structured in some way, for example, by numbering the elements. The most basic data structure in Python is the "sequence".

-> List is one of the Sequence Data structure

-> Lists are collection of items (Strings, integers or even other lists)

-> Lists are enclosed in [ ]

-> Each item in the list has an assigned index value.

-> Each item in a list is separated by a comma

-> Lists are mutable, which means they can be changed.

List Creation

- To create a list in Python, we use square brackets `[]`

In [1]:
```python
emptyList = []

lst = ['one', 'two', 'three', 'four'] # list of strings

lst2 = [1, 2, 3, 4]     # list of integers

lst3 = [[1, 2,3], [3, 4]]    # list of lists

lst4 = [1, 'ramu', 24, 1.24] # list of different datatypes

print(lst)
print(lst2)
print(lst3)
print(lst4)
```
```
['one', 'two', 'three', 'four']
[1, 2, 3, 4]
[[1, 2, 3], [3, 4]]
[1, 'ramu', 24, 1.24]
```

List Length

we can use the built-in `len()` function to find the length of a list.

The `len()` function accepts a *sequence or a collection* as an argument and returns the number of elements present in the sequence or collection

In [2]:
```python
lst = ['one', 'two', 'three', 'four']
lst2 = [1,2,3,4,5,6]

# find length of a list

print(len(lst))
print(len(lst2))
```
```
4
6
```

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

`append()`

The append() method adds an item to the end of the list

- **Syntax:** list.append(object)

In [3]:
```python
# Example
lst = ['one', 'two', 'three', 'four']
lst.append("iota")
print(lst)
```
```
['one', 'two', 'three', 'four', 'iota']
```

In [4]:
```python
# Example
lst = ['one', 'two', 'three', 'four']
lst.append(2)
lst.append(4)

print(lst)
```
```
['one', 'two', 'three', 'four', 2, 4]
```

In [5]:
```python
lst = ['one', 'two', 'three', 'four']

lst.append(['six','seven']) # In this case, append will add a list as a single element.
print(lst)
```
```
['one', 'two', 'three', 'four', ['six', 'seven']]
```

Note: The append() method modifies the original list. It doesn't return any value.

In [6]:
```python
# Example
lst = [11,22,33]
print(lst.append(44))
```
```
None
```

`extend()`

The extend() method adds all the elements of an iterable (list, tuple, string etc.) to the end of the list

- **Syntax:** list.extend(iterable)

In [10]:
```python
lst = ['one', 'two', 'three', 'four']
lst.extend(['six','seven']) # here each element of an iterable object added one by one in the list
print(lst)
```
```
['one', 'two', 'three', 'four', 'six', 'seven']
```

append() & extend()

In [10]:
```python
lst = ['one', 'two', 'three', 'four']

lst2 = ['five', 'six']

# append
lst.append(lst2)

print(lst)
len(lst)
```
```
['one', 'two', 'three', 'four', ['five', 'six']]
```
Out[10]:
```
5
```

In [7]:
```python
lst = ['one', 'two', 'three', 'four']

lst2 = ['five', 'six']

# extend will join the lst with lst2

lst.extend(lst2)

print(lst)
print(len(lst))
```
```
['one', 'two', 'three', 'four', 'five', 'six']
6
```

`insert()`

The insert() method inserts an element to the list at the specified index.

- **Syntax:** list.insert(index, object)

```
index - the index where the object needs to be inserted
object - this is the object to be inserted in the list
```

In [8]:
```python
# Example
lst = ['one', 'two', 'four']

lst.insert(2, "three") # 'three' is inserted at index 2 (3rd position)

print(lst)
```
```
['one', 'two', 'three', 'four']
```

`remove()`

The remove() method removes the first matching element (which is passed as an argument) from the list.

Raises ValueError if the value is not present.

- **Syntax:** list.remove(value)

In [9]:
```python
lst = ['one', 'two', 'three', 'four', 'two']

lst.remove('two') # it will remove first occurence of 'two' in a given list

# change will happen in original list
print(lst)
```
```
['one', 'three', 'four', 'two']
```

In [10]:
```python
# raises ValueError if item is not present in the list
lst = ['one', 'two', 'three', 'four', 'two']
lst.remove("five")
```
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[10], line 3
      1 # raises ValueError if item is not present in the list
      2 lst = ['one', 'two', 'three', 'four', 'two']
----> 3 lst.remove("five")

ValueError: list.remove(x): x not in list
```

In [11]:
```python
# Example

lst = ['one', 'two', 'three', 'four', 'two']
lst.remove("one")
print(lst)
lst.remove("one")
print(lst)
```
```
['two', 'three', 'four', 'two']
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[11], line 6
      4 lst.remove("one")
      5 print(lst)
----> 6 lst.remove("one")
      7 print(lst)

ValueError: list.remove(x): x not in list
```

`pop()`

The pop() method removes the item at the given index from the list and returns the removed item.

Raises IndexError if list is empty or index is out of range.

- **Syntax:** list.pop(index)

```
If index is not passed, it will remove the last element of the list because the default index is -1
```

In [12]:
```python
lst = ['one', 'two', 'three', 'four', 'two']

popped_item = lst.pop(2)

print(popped_item)

print(lst)
```
```
three
['one', 'two', 'four', 'two']
```

In [16]:
```python
lst = ['one', 'two', 'three', 'four', 'two']

popped_item = lst.pop()

print(popped_item)

print(lst)
```
```
two
['one', 'two', 'three', 'four']
```

`clear()`

The clear() method removes all items from the list.

- **Syntax:** list.clear()

In [13]:
```python
lst = ['one', 'two', 'three', 'four', 'five']
print(lst)
lst.clear()
print(lst)
```
```
['one', 'two', 'three', 'four', 'five']
[]
```

`del`

The del keyword removes items from the specified index as well as having the ability to delete the entire list although it is not a list method.

- **Syntax:**

```
del list[index] (it will delete an item from specific index)

del list      (it will delete entire list)
```

In [14]:
```python
# del to remove item based on index position

lst = ['one', 'two', 'three', 'four', 'five']

del lst[1]
print(lst)
```
```
['one', 'three', 'four', 'five']
```

In [15]:
```python
lst = ['one', 'two', 'three', 'four', 'five']
print(lst)
del lst   # it will delete entire list
print(lst) # it will give an error because the list has been deleted
```
```
['one', 'two', 'three', 'four', 'five']
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[15], line 4
      2 print(lst)
      3 del lst   # it will delete entire list
----> 4 print(lst)

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

`reverse()`

The reverse() method reverses the elements of the list.

- **Syntax:** list.reverse()

```
Reverse *IN PLACE*.
```

In [16]:
```python
lst = ['one', 'two', 'three', 'four']

lst.reverse()

print(lst)
```
```
['four', 'three', 'two', 'one']
```

`sort()`

the sort() method sorts the items of a list in ascending or descending order.

- **Syntax:** list.sort(*, key=None, reverse=False)

```
*reverse* - If True, the sorted list is reversed (or sorted in Descending order)

The sort is in-place (i.e. the list itself is modified) and stable (i.e. the order of two equal elements is maintained).
```

In [17]:
```python
st = [1, 20, 5, 5, 4.2]

# sort the list and stored in itself
st.sort()

print("Sorted list: ", st)
```
```
Sorted list:  [1, 4.2, 5, 5, 20]
```

In [18]:
```python
# sort a list in descending order
st = [1, 20, 5, 5, 4.2]

# set the reverse parametere to True.
st.sort(reverse=True)

print("Sorted list in descending order: ", st)
```
```
Sorted list in descending order:  [20, 5, 5, 4.2, 1]
```

`sorted()`

The easiest way to sort a List is with the sorted(list) function.

That takes a list and returns a new list with those elements in sorted order.

The original list is not changed.

The sorted() optional argument reverse=True, e.g. sorted(list, reverse=True), makes it sort backwards.

In [19]:
```python
# create a list with numbers
numbers = [3, 1, 6, 2, 8]

sorted_lst = sorted(numbers,reverse=False)

# original list remain unchanged
print("Original list: ", numbers)

print("Sorted list :", sorted_lst)
```
```
Original list:  [3, 1, 6, 2, 8]
Sorted list : [1, 2, 3, 6, 8]
```

In [20]:
```python
# print a list in reverse sorted order
print("Reverse sorted list :", sorted(numbers, reverse=True))

# orginal list remain unchanged
print("Original list :",  numbers)
```
```
Reverse sorted list : [8, 6, 3, 2, 1]
Original list : [3, 1, 6, 2, 8]
```

`Count()`

The count() method returns the number of times the specified element appears in the list.

- **Syntax:** list.count(value)

In [21]:
```python
numbers = [1, 2, 3, 1, 3, 4, 2, 5,'iota']

# frequency of 1 in a list
print(numbers.count(1))

# frequency of 3 in a list
print(numbers.count(3))
```
```
2
2
```

List realted keywords in Python

In [22]:
```python
#keyword 'in' is used to test if an item is in a list

lst = ['one', 'two', 'three', 'four']

if 'two' in lst:
    print('AI')

#keyword 'not' can combined with 'in'
if 'six' not in lst:
    print('ML')
```
```
AI
ML
```

In [23]:
```python
"Two" not in lst
```
Out[23]:
```
True
```

List Having Multiple References

In [25]:
```python
# observe changes in lists: lst and abc

lst = [1, 2, 3, 4, 5]
abc = lst
abc.append(6)


# print original list
print("Original list: ", lst)
print("Updated list: ",abc)
```
```
Original list:  [1, 2, 3, 4, 5, 6]
Updated list:  [1, 2, 3, 4, 5, 6]
```

In [22]:
```python
lst = [1, 2, 3, 4, 5]
abc = lst.copy()  # this will create a new list at new location
abc.append(6)

# print original list
print("Original list: ", lst)
print("Updated List: ",abc)
```
```
Original list:  [1, 2, 3, 4, 5]
Updated List:  [1, 2, 3, 4, 5, 6]
```

List Indexing

Each item in the list has an assigned index value starting from 0.

Accessing elements in a list is called indexing.

syntax: variable_name[index]

In [26]:
```python
lst = [1, 2, 3, 4]
print(lst[1]) # print second element

# print last element using negative index
print(lst[-2])
```
```
2
3
```

List Slicing

Accessing parts of segments is called slicing.

The key point to remember is that the :end value represents the first value that is not in the selected slice.

syntax: variable_name[start:end]

In [28]:
```python
numbers = [10, 20, 30, 40, 50,60,70,80,['IOTA','Academy']]

# print all numbers
print(numbers[:])

# print from index 0 to index 3
print(numbers[0:4])    #[start:end] ----> actual_end is end-1
```
```
[10, 20, 30, 40, 50, 60, 70, 80, ['IOTA', 'Academy']]
[10, 20, 30, 40]
```

In [29]:
```python
# Example
numbers[::-1]   # print the list in reverse order
```
Out[29]:
```
[['IOTA', 'Academy'], 80, 70, 60, 50, 40, 30, 20, 10]
```

In [30]:
```python
# Example
numbers[::2]   # here stepsize is 2
```
Out[30]:
```
[10, 30, 50, 70, ['IOTA', 'Academy']]
```

In [32]:
```python
# Example
numbers[:5]   # print from index 0 to index 4
```
Out[32]:
```
[10, 20, 30, 40, 50]
```

In [33]:
```python
# Example
numbers[-1].append('for Python')
numbers
```
Out[33]:
```
[10, 20, 30, 40, 50, 60, 70, 80, ['IOTA', 'Academy', 'for Python']]
```

In [34]:
```python
numbers[-1][0]
```
Out[34]:
```
'IOTA'
```

In [35]:
```python
print(numbers[-1])
type(numbers[-1])
```
```
['IOTA', 'Academy', 'for Python']
```
Out[35]:
```
list
```

In [36]:
```python
print (numbers)
#print alternate elements in a list
print(numbers[::2])    #[start:end:step_size]


#print elemnts start from 0 through rest of the list
print(numbers[1::2])
```
```
[10, 20, 30, 40, 50, 60, 70, 80, ['IOTA', 'Academy', 'for Python']]
[10, 30, 50, 70, ['IOTA', 'Academy', 'for Python']]
[20, 40, 60, 80]
```

In [37]:
```python
numbers[6:1:-2]
```
Out[37]:
```
[70, 50, 30]
```

In [38]:
```python
numbers[-3:1:-2]
```
Out[38]:
```
[70, 50, 30]
```

List extend using "+"

In [39]:
```python
lst1 = [1, 2, 3, 4]
lst2 = ['varma', 'naveen', 'murali', 'brahma']
new_lst = lst1 + lst2

print(new_lst)
```
```
[1, 2, 3, 4, 'varma', 'naveen', 'murali', 'brahma']
```

List Looping

In [40]:
```python
# loop through a list

lst = ['one', 'two', 'three', 'four']

for ele in lst:
    print(ele)
```
```
one
two
three
four
```

In [41]:
```python
for i in range(len(lst)):
    print(i)
    print(f"I am {lst[i]}")
```
```
0
I am one
1
I am two
2
I am three
3
I am four
```

List Comprehensions (will cover later)

List comprehensions provide a concise way to create lists.

Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.

In [42]:
```python
# without list comprehension
squares = []
for i in range(10):
    if i%2==0:
        squares.append(i**2)   #list append
print(squares)
```
```
[0, 4, 16, 36, 64]
```

In [43]:
```python
#using list comprehension
squares = [i**2 for i in range(10)]    #whole thing in just one line of code
print(squares)
```
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [44]:
```python
#example

lst = [-10, -20, 10, 20, 50]

#create a new list with values doubled
new_lst = [i*2 for i in lst]
print(new_lst)

#filter the list to exclude negative numbers
new_lst = [i for i in lst if i >= 0]
print(new_lst)


#create a list of tuples like (number, square_of_number)
new_lst = [(i, i**2) for i in range(10)]
print(new_lst)
```
```
[-20, -40, 20, 40, 100]
[10, 20, 50]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81)]
```

Nested List Comprehensions (will cover later)

In [45]:
```python
#let's suppose we have a matrix

matrix = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12]
]

#transpose of a matrix without list comprehension
transposed = []
for i in range(4):
    lst = []
    for row in matrix:
        lst.append(row[i])
    transposed.append(lst)

print(transposed)
```
```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

In [46]:
```python
matrix = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12]
]

matrix
```
Out[46]:
```
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

In [47]:
```python
matrix[1][2]
```
Out[47]:
```
7
```

In [48]:
```python
#with list comprehension
transposed = [[row[i] for row in matrix] for i in range(4)]    #drawback: decrease readibility
print(transposed)
```
```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

That's Great