What is a Python List

A Python list is a data structure which can contain more than one value at-a-time.

A value in a list is generally referred to as an **item** or an **element**. Let's suppose that you want to store the names of the countries who won the FIFA world cup from `1982` till `2018`.

Year	Winner
1982	Italy
1986	Argentina
1990	Germany
1994	Brazil
1998	France
2002	Brazil
2006	Italy
2010	Spain
2014	Germany
2018	France

In this, you would require a variable for every winner. But using a Python list, you can store all the winners in one go.

A Python list is created using the square brackets `[]`. Each item in a Python list is separated by a comma.

```
# Create a Python list containing all the winners of the FIFA world cup from 1982 to 2018. $
fifa=["Italy" ,"Argentina" ,"Germany",  "Brazil", "France", "Brazil", "Italy", "Spain", "Ger
```

```
#Verify whether the 'fifa_wc_winners' is a list or not using the 'type()' function.
type(fifa)
```

```
list
```

```
fifa
```

```
['Italy',
 'Argentina',
 'Germany',
 'Brazil',
 'France',
 'Brazil',
 'Italy',
 'Spain',
 'Germany',
 'France']
```

```
print(fifa)
```

```
['Italy', 'Argentina', 'Germany', 'Brazil', 'France', 'Brazil', 'Italy', 'Spain', 'Germany', 'France']
```

The `in` And `not in` Keywords

```
# Check whether 'Spain' exists in the 'fifa_wc_winners' list or not.
"Spain" in fifa
```

```
True
```

```
'spain' in fifa
```

```
False
```

```
#Check whether 'Spain' exists in the 'fifa_wc_winners' list or not.
"India" not in fifa
```

```
True
```

The count() Function

Syntax: `list_name.count(item)`

```
# Count the number of times 'France' has won the world cup between the years 1982 and 2018.
fifa.count('Brazil')
```

```
2
```

```
fifa
```

```
['Italy',
 'Argentina',
 'Germany',
 'Brazil',
 'France',
 'Brazil',
 'Italy',
 'Spain',
 'Germany',
 'France']
```

The `append()` Function^^

Syntax: `list_name.append(item)`

```
#Append India in the list
fifa.append("India")
```

```
fifa
```

```
['Italy',
 'Argentina',
 'Germany',
 'Brazil',
 'France',
 'Brazil',
 'Italy',
 'S. Korea',
 'Spain',
 'Germany',
 'France',
 'India']
```

```
#Insert a country at a specific location using insert function.
fifa.insert(7,'S. Korea')
```

```
len(fifa)
```

```
12
```

```
#Find the length of list
len(fifa)
```

```
fifa.insert(7,10)
```

```
fifa
```

```
['Italy',
 'Argentina',
 'Germany',
 'Brazil',
 'France',
 'Brazil',
 'Italy',
 10,
 'S. Korea',
 'Spain',
 'Germany',
```

```
 'France',
 'India']
```

Multiple Data Types

A Python list can store values of different types. Recall that in the trial class we created 4 different variables to store the attributes of a planet.

	Mercury
Diameter (km)	4879
Gravity (m/s^2)	3.7
Ring	No

```
# Student Action: Create a list which contains "Mercury", its diameter, gravity and whether
planet=['Mercury',4879, 3.7, False ]
```

```
planet[0]
```

```
'Mercury'
```

```
planet[-3]
```

```
4879
```

```
planet
```

```
['Mercury', 4879, 3.7, False]
```

List Length To see how many items are stored in a list, you can use a function called `len()`. *italicized text*

```
#Find the number of items stored in the 'planet' list.
len(planet)
```

```
4
```

List Indexing **Syntax:** `list_name[index_number]`

```
#  Print the items at each index in the 'planet' list.
planet[40]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipython-input-924464374.py in <cell line: 0>()
      1 #  Print the items at each index in the 'planet' list.
----> 2 planet[40]

IndexError: list index out of range
```

```
print("value present at 0 location is", planet[0])
print("value present at 1 location is", planet[1])
print("value present at 2 location is", planet[2])
print("value present at 3 location is",planet[3])
```

```
value present at 0 location is Mercury
value present at 1 location is 4879
value present at 2 location is 3.7
value present at 3 location is False
```

```
n=len(planet)
```

```
n
```

```
4
```

```
for i in range(0,10):
  print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```
for i in range(0,n):
  print(planet[i])
```

```
Mercury
4879
3.7
False
```

```
#  Using 'for' loop, print each element of the 'planet' list.
for i in range(0,len(planet)):
  print(planet[i])
```

```
Mercury
4879
3.7
False
```

```
for element in planet:
  print(element)
```

```
Mercury
4879
3.7
False
```

```
for i in range(len(planet)-1, -1, -1):
  print(planet[i])
```

```
False
3.7
4879
Mercury
```

```
# Print every item of the 'planet' in the reverse order, using the 'for' loop.
for i in range(len(planet)-1,-1,-1):
  print(planet[i])
```

```
planet[-1]
```

```
False
```

```
l=len(planet)
```

```
print(planet[-4])
print(planet[-3])
print(planet[-2])
print(planet[-1])
```

```
print(planet[-1])
print(planet[-2])
print(planet[-3])
print(planet[-4])
```

```
#  Use the negative indices to retrieve all the items from the 'planet' list.
for i in range(1,l+1,1):
  print(planet[-i])
```

```
my_cars = [['Chrysler', 'chy', [1,2,'ll', (1,2)]], 'Lamborghini', 'Bugatti', 'Porsche', 'Fc
```

```
my_cars[0][2][3]
```

```
(1, 2)
```

```
my_cars[0]
```

```
['Chrysler', 'chy']
```

```
my_cars[0][1]
```

```
'chy'
```

List Slicing

Syntax: list_name[start_index:end_index]

my_cars = ['Chrysler', 'Lamborghini', 'Bugatti', 'Porsche', 'Ford', 'Rolls Royce', 'Suzuki', 'Bentley', 'Lexus', 'Tesla']*italicized text*

```
my_cars
```

```
['Chrysler',
 'Lamborghini',
 'Bugatti',
 'Porsche',
 'Ford',
 'Rolls Royce',
 'Suzuki',
 'Bentley',
 'Lexus',
 'Tesla']
```

```
my_cars[ 2:  5]
```

```
['Bugatti', 'Porsche', 'Ford']
```

```
my_cars[ 7:  8]
```

```
['Bentley']
```

```
len(my_cars)
```

```
10
```

```
my_cars
```

```
['Chrysler',
 'Lamborghini',
 'Bugatti',
 'Porsche',
 'Ford',
 'Rolls Royce',
 'Suzuki',
 'Bentley',
 'Lexus',
 'Tesla']
```

```
my_cars[1:7 : 3]
```

```
['Lamborghini', 'Ford']
```

```
#Retrieve the items from a list using the slicing method by mentioning both the starting an
my_cars[0 :len(my_cars) ]
```

```
#Retrieve the first 5 items from the 'my_cars' Python list without mentioning the starting
my_cars[:5]
```

```
# Write a code to see all the cars in the list starting from 'Lamborghini' to 'Bentley'.
my_cars[1:8]
```

```
# Retrieve all the alternate items from the 'my_cars' list.
my_cars[ : :3]
```

```
# Retrieve all the items from the 'my_cars' list in the reverse order.
my_cars[ : :-1 ]
```

```
['Tesla',
 'Lexus',
 'Bentley',
 'Suzuki',
 'Rolls Royce',
 'Ford',
 'Porsche',
 'Bugatti',
 'Lamborghini',
 'Chrysler']
```

Index Of An Item

Syntax: list_name.index(item)

```
my_cars
```

```
#Find the index of 'Ford'.
my_cars.index('Ford')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipython-input-1770741715.py in <cell line: 0>()
      1 #Find the index of 'Ford'.
----> 2 my_cars.index('ford')

ValueError: 'ford' is not in list
```

Removing An Item

Syntax: list_name.remove(item)

```
# Remove 'Bentley' from the 'my_cars' list.
my_cars.remove('Bentley')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipython-input-330822157.py in <cell line: 0>()
      1 # Remove 'Bentley' from the 'my_cars' list.
----> 2 my_cars.remove('Bentley')

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

```
my_cars
```

```
['Chrysler',
 'Lamborghini',
 'Bugatti',
 'Porsche',
 'Ford',
 'Rolls Royce',
 'Suzuki',
 'Lexus',
 'Tesla']
```

```
len(my_cars)
```

```
9
```

pop() Function^ You can also remove an item from a list using another function called pop(). It has two behaviours.

Behaviour I: If you do not specify any input, it will always remove only the last item from a Python list.

Syntax: list_name.pop(

```
# Remove the last item from the 'my_cars' list using the 'pop()' function.
my_cars.pop()
```

```
'Tesla'
```

```
my_cars
```

```
['Chrysler',
 'Lamborghini',
 'Bugatti',
 'Porsche',
 'Ford',
 'Rolls Royce',
 'Suzuki',
 'Lexus']
```

Behaviour II: If you want to remove an item from a list located at a specific index, then pass the index of that item as an input to the pop() function.

Syntax: list_name.pop(item_index)

```
#Remove the item at 'index = 3' from the 'my_cars' list using the 'pop()' function.
my_cars.pop(3)
```

```
'Porsche'
```

```
my_cars[5]='Mc Laren'
```

Item Replacement^^ You can replace an existing item in a list with a new item. Suppose I exchanged my Suzuki car with a McLaren car. So, I need to update my_cars list accordingly.

To replace an item, first get the index of that item. Then using the list indexing method, replace the existing item with the new one.

Syntax: list_name[list_name.index('current_item')] = new_item

```
my_cars[my_cars.index('Suzuki')]='McLaren'
```

```
#Replace 'Suzuki' with 'McLaren' in the 'my_cars' list.
a= my_cars.index('Suzuki')
```

```
a
```

```
my_cars[5]='McLaren'
```

```
my_cars
```

Nested Python Lists

A two-dimensional (or a tabular) data can be represented using a Python list.

Consider the example shown below. Here we have a two-dimensional (or a tabular) data of 8 planets and their corresponding diameters (in km).

#	Planet	Diameter (km)
1	Mercury	4879
2	Venus	12104
3	Earth	12756
4	Mars	6972
5	Jupiter	142984
6	Saturn	120536
7	Uranus	51118
8	Neptune	49528

```
p= [

    ["Mercury"  , 4879 ],
    [   "Venus",    12104 ],
    [ ],
    [ ],
    [ ],
    [ ],
    [ ],
    [ ]
   ]
```

```
list_planet=[
             [ "Mercury",  4879, 5427  , 3.7],
             [ "Venus",    12104 ],
             [ ],
             [ ],
             [ ],
             [ ],
             [ ],
             [ ],
             [ ]

            ]
```

```
# Represent a two-dimensional data in a nested list.
p1=[
      [ "Mercury"   ,4879, 5427 ,3.7 , [1,2]],
      ["Venus", 12104],
      ["Earth", 12756],
      ["Mars",  6972],
      ["Jupiter",   142984],
      ["Saturn",120536],
      ["Uranus" ,51118],
      [ "Neptune",  49528]

      ]
```

```
p1
```

The row and column indices begin with `0`. Now, refer to the table below.

#	Planet	Diameter (km)	Density $\left(\frac{kg}{m^3}\right)$	Gravity $\left(\frac{m}{s^2}\right)$
1	Mercury	4879	5427	3.7
2	Venus	12104	5243	8.9
3	Earth	12756	5514	9.8
4	Mars	6972	3933	3.7
5	Jupiter	142984	1326	23.1
6	Saturn	120536	687	9.0
7	Uranus	51118	1271	8.7
8	Neptune	49528	1638	11.0

```
p1
```

```
[['Mercury', 4879],
 ['Venus', 12104],
 ['Earth', 12756],
 ['Mars', 6972],
 ['Jupiter', 142984],
 ['Saturn', 120536],
 ['Uranus', 51118],
 ['Neptune', 49528]]
```

```
# Create a nested list to represent the two-dimensional data shown in the table above.
```

Indexing A Nested List^

Syntax: `list_name[row_index][col_index]`

```
# Retrieve the diameter of 'Jupiter' from the 'planet_data' list.

p1[4][1]
```

```
p1[7][1]
```

```
# Retrieve all the details of 'Mars' using the list indexing method.
p1[3]
```

Length Of Nested Python Lists

The length of a nested list is the number of sublists it contains. In the `planet_data` list, there are 9 sublists. Hence, it's length is 9. You can use the `len()` function to compute the length of a nested list.

```
#  Find the number of sublists in the 'planet_data' list using the 'len()' function.
len(p1)
```

```
8
```

```
for i in p1:
  for j in i:
    print(j)
  print()
```

```
Mercury
4879

Venus
12104

Earth
12756

Mars
6972
```

```
Jupiter
142984

Saturn
120536

Uranus
51118

Neptune
49528
```

```
for i in range(0,len(p1)):
```

```
p1
```

```
#Find the total number of items in a nested list.
count=0
for i in p1:
  count=count+ len(i)
print(count)
```

```
16
```

```
print(count)
```

Data Representation In Three-Dimensions Using Python Lists^^^

Let's learn about three-dimensional lists in Python.

For all practical purposes, a three-dimensional list is a collection of equal-sized two-dimensional lists where the **size** of a two-dimensional list is the number of rows and columns contained in a two-dimensional list. For e.g., if there are 2 two-dimensional lists having 3 rows and 2 columns, then you can create a three-dimensional list using them.

Hence, the equal-sized two-dimensional lists will always have the same number of rows and columns.

Note: In a three-dimensional list, we can refer a two-dimensional list as a **block** for the ease of understanding.

Now, let's create a three-dimensional list which contains three equal-sized two-dimensional lists. Let the size of each two-dimensional list be 3 rows and 3 columns. Let each item in the three-dimensional list indicate the position it acquires in the list.

So each item will be of the form ijk, where ijk indicates that the item is present in the i^{th} block, j^{th} row and k^{th} column.

Hence,

- if $i = 1, j = 1, k = 1$, then it means the item exists in the first block, first row and first column.
- if $i = 1, j = 1, k = 2$, then it means the item exists in the first block, first row and second column.
- if $i = 1, j = 1, k = 3$, then it means the item exists in the first block, first row and third column.
    -
    -
    -
- if $i = 3, j = 3, k = 1$, then it means the item exists in the third block, third row and first column.
- if $i = 3, j = 3, k = 2$, then it means the item exists in the third block, third row and second column.
- if $i = 3, j = 3, k = 3$, then it means the item exists in the third block, third row and third column.

```
# Create a three-dimensional list in Python.

list_3d=[
          [
```

```
            [111 ,112 , 113],
            [ 121,122 ,123 ],
            [ 131,132 ,133 ]
        ],

        [
            [211 ,212 ,213 ],
            [ 221,222, 223 ],
            [ 231,232 ,233 ]
        ],

        [
            [ 311,312 ,313 ],
            [ 321,322 ,323 ],
            [331 ,332 ,333 ]
        ]

      ]
```

```
list_3d
```

```
#Using the 'len()' function, find the number of blocks contained in the 'three_dim_list'.
len(list_3d)
```

Syntax: `list_name[block_index][row_index][col_index]`

```
# Retrieve 121 from 'three_dim_list' using the list indexing method.
list_3d[0][1][0]
```

```
list_3d[0]
```

```
#Retrieve 232 from 'three_dim_list' using the list indexing method.
```

```
#  Retrieve items of a multi-dimensional list using nested 'for' loop.
for i in range(0,3):
  for j in range(0,3):
    for k in range(0,3):
      print(list_3d[i][j][k], end=" ")
    print()
  print()
```

```
#  Iterate through each item in the 'three_dim_list' using the 'for' loop without using the
for i in list_3d:
  for j in i:
    for k in j:
      print(k)
```

```
#  Using the 'for', find the number of items in the 'three_dim_list'.
count=0
for i in list_3d:
  for j in i:
    count=count+ len(j)

print(count)
```