LISTS

A List is a data structure in Python that holds an organized collection of items. The items, or values, that are contained within the list can be various data types themselves. In fact, you can even have lists that exist within lists.

Note that lists are created by using comma separated values between square brackets. The pattern is **list_name = [item_1, item_2, item_N]**.

To create an empty list use: **list_name = []**.

Items within a list can always be accessed by index. List indices are always zero based, which means that the first item in the list will have an index of 0, the second item an index of 1, and so on. To access any item in a list using an index, simply enclose the index in square brackets directly following the list name. The pattern is **list_name[index]**.

 # creating a List

```
>>> fruits = ['Apple', 'Mango', 'Orange']

>>> fruits

['Apple', 'Mango', 'Orange']
```

Indexing a List:

```
>>> fruits[0]

'Apple'

>>> fruits[1]

'Mango'

>>> fruits[2]

'Orange'

>>>

>>> fruits[2] = 'Water Melon' # replacing an element with index position

>>> fruits

['Apple', 'Mango', 'Water Melon']
```

As you can see in the above code, the element/item ‘Orange’ has been replaced with ‘Water Melon’

Adding elements to a list

Generally, there are three ways in which you can add elements to a list.

append(): add (only one) element in a list
extend(): adds multiple elements to a list
insert(): adds an element at a particular position using an index

```
>>> fruits.append("Grapes")

>>> fruits

['Apple', 'Mango', 'Water Melon', 'Grapes']

>>>

>>> fruits.extend(['Orange', 'kiwi', 'pears'])

>>> fruits

['Apple', 'Mango', 'Water Melon', 'Grapes', 'Orange', 'kiwi', 'pears']

>>>

>>> fruits[0]

'Apple'

>>> fruits.insert(0, 'Banana')

>>> fruits

['Banana', 'Apple', 'Mango', 'Water Melon', 'Grapes', 'Orange', 'kiwi', 'pears']
```

Note: you can't pass multiple arguments using insert method:

```
>>>  fruits.insert(3, 'Guava', 7, 'Avacoado')
Traceback (most recent call last):

  File "<pyshell#60>", line 1, in <module>

    fruits.insert(3, 'Guava', 7, 'Avacoado')

TypeError: insert() takes exactly 2 arguments (4 given)

>>> fruits.insert(3, 'Guava')
```

finding the index of a list element

```
>>> fruits
```

['Banana', 'Apple', 'Mango', 'Guava', 'Water Melon', 'Grapes', 'Orange', 'kiwi', 'pears']

```
>>> fruits_index = fruits.index('Mango')

>>> fruits_index

2
```

Iterating a list

Iterating a list is nothing but spanning each and every element of a list. Generally we iterate a list using a for loop.

```
>>> for x in fruits:
        print(x)
```

Banana

Apple

Mango

Guava

Water Melon

Grapes

Orange

kiwi

pears

```
>>>

>>> sorted_fruits = sorted(fruits)    # Sorting a List

>>> sorted_fruits
```

['Apple', 'Banana', 'Grapes', 'Guava', 'Mango', 'Orange', 'Water Melon', 'kiwi', 'pears']

```
>>>

>>> some_fruits = ['Strawberry', 'pomegranate']
```

```
>>>
>>> # list concatenation
>>> Total_fruits = fruits + some_fruits
>>> Total_fruits
['Banana', 'Apple', 'Mango', 'Guava', 'Water Melon', 'Grapes', 'Orange',
'kiwi', 'pears', 'Strawberry', 'pomegranate']
>>>
# Range in Python
# range with single argument
>>> for x in range(10):
        print(x)



0
1
2
3
4
5
6
7
8
9
>>>
# range with start and end values
>>> for x in range(1,10):
        print(x)
```

1

2

```
3
4

5

6

7

8

9

>>>
```

 # range with start, end and step values

```
>>> for x in range(0,20,2):
        print(x)



0

2

4

6

8

10

12

14

16

18

>>>
```

 # convert a string to a list

```
>>> x = "My name is vamshi"

>>> print(x.split())

['My', 'name', 'is', 'vamshi']
>>>
```

 # convert a list to a string

```
>>> a = ['My', 'name', 'is', 'vamshi']

>>> print(' '.join(a))

My name is vamshi

>>>
```

POP() function:
the pop() function removes (last) element from a list.

```
>>> fruits

['Banana', 'Apple', 'Mango', 'Guava', 'Water Melon', 'Grapes', 'Orange',
'kiwi', 'pears']

>>> fruits.pop()

'pears'

>>> fruits

['Banana', 'Apple', 'Mango', 'Guava', 'Water Melon', 'Grapes', 'Orange', 'kiwi']

>>>

>>>

>>>

>>>
```

List Functions

Split(), Index(), Find() with example scenarios

Scenario:

Given a following string:
Game = 'Xbox 360 | 1000 | New'
Extract and display the string field before the first pipe symbol as **"**Product**"**. The field between the two pipe symbols is **"**Price**"** of the product, and the last field is the **"**Condition**"** of the product.

As a Python programmer/developer your job is to extract the portions of the string field as follows:

Product	Price	Condition
Xbox 360	1000	New

Example using Split()

```
>>> Game = 'Xbox 360 | 1000 | New'
>>>
>>> Game_elements = Game.split('|')
>>> Game_elements
['Xbox 360 ', ' 1000 ', ' New']
>>>
>>> Product = Game_elements[0]
>>> Product
'Xbox 360 '
>>>
>>> Price = Game_elements[1]
>>> Price
' 1000 '
>>> Condition = Game_elements[2]
>>> Condition
' New'
>>>
```

Example using index()

```
>>> Game = 'Xbox 360 | 1000 | New'
>>> Game_index = Game.index('Xbox 360')
# let's find out the index position of first and last elements:
>>> Game_index
0
>>> Game_last = Game.index('New')
```

```
>>> Game_last
18
# let's extract the Product field from the string field using indexing concept:
>>> Game = 'Xbox 360 | 1000 | New'
>>> Product = Game[ :Game.index('|')]  # the indexing starts from 0th position
>>> Product
'Xbox 360 '
# Let's now find out the index position of the first pipe (|) symbol:
>>> Game.index('|')
9
# now let's extract the Price field:
>>> Price = Game[11:15]
>>> Price
'1000'
# extract the Condition field:
>>> Condition = Game[18: ]  #index starts from 18th character to end of string
>>> Condition
'New'
>>>
```

Note: the disadvantage of index() is that it doesn't give you the index position of second instance of a string or a delimiter.

Find():

Find() function will display the index position of any element, delimiter or a string.

Solving the example using find()

```
>>> Game = 'Xbox 360 | 1000 | New'
# first let's find the index position of first pipe (|) symbol:
>>> Game.find('|')
```

9

Note: Inorder to know the position of second pipe symbol, you need to issue a number which is one more than the position of first pipe symbol:

```
>>> Game.find('|', 11)

16
```

As we know the positions of the two pipe symbols we can easily extract our desired fields.

```
>>> Product = Game[0:9]

>>> Product

'Xbox 360 '

>>> Price = Game[Game.find('|')+1 : Game.find('|',11)]

>>> Price

' 1000 '

>>> Condition = Game[Game.find('|',11)+1: ]

>>> Condition

' New'

>>>
```

List Comprehensions

List comprehension is an elegant way to define and create list in Python. These lists have often the qualities of sets, but are not in all cases sets.

```
>>> squares_list = []

>>> for x in range(1,10):

        squares_list.append(x**2)



>>> print(squares_list)

[1, 4, 9, 16, 25, 36, 49, 64, 81]

>>>
```

```
>>> squares_listcomp = [x**2 for x in range(1,10)]
>>> print(squares_list)
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

>>> [1, 4, 9, 16, 25, 36, 49, 64, 81]

[1, 4, 9, 16, 25, 36, 49, 64, 81]

>>>

>>>

Assignment:

>>> paragraph = ["I'm a Data Scientist"]

o/p:

["I'm", 'a', 'Data', 'Scientist']

Solve using normal way and list comprehension way