LIST

There are four collection data types in the Python programming language:

1. **List** is a collection which is ordered and changeable. Allows duplicate members.
2. **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
3. **Set** is a collection which is unordered and unindexed. No duplicate members.
4. **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

In this section we will be discussing about most important data type - **list** .

1. Why do we use list?

Let us say we have a collection of numbers, and we would like to store it in python.

e.g. 1,2,5,34,54,40

Obviously we don't want to store these numbers randomly. We want to store them in a particular sequence, i.e. one after another. In this case want we need is a list in python.

List helps us to store a collection of items in a sequential manner.

The sequence number which is given to the items in a list is known as <u>***index.***</u>

So with the help of list we can store the collection of numbers as:

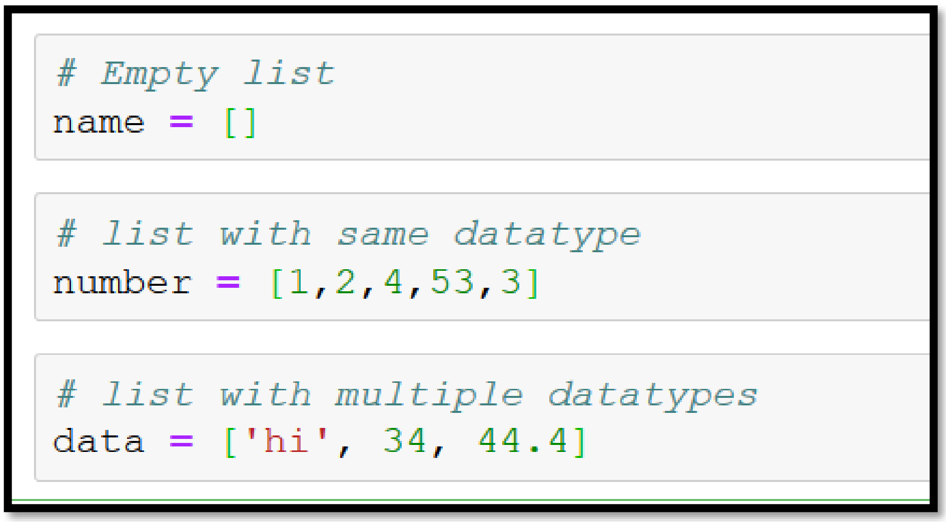
Numbers = [1,2,5,34,54,40]

<u>One of the most important property of list is that it is mutable. i.e. its content can always be changed after creation.</u>

2. Creating a list

2.1 Simple List

- In python, a list can be created by placing sequence of elements within square brackets, separated with ','(comma).
- Values in list can be any datatype, list can contain multiple data types also.
- Values can be repeated in a list also.



```
# Empty list
name = []

# list with same datatype
number = [1,2,4,53,3]

# list with multiple datatypes
data = ['hi', 34, 44.4]
```

2.2 Nested List

In python, a nested listed is a list inside a list.

Using this we can create complex lists.

For e.g.: list = [1, [3, 4, 5] , 34, 78]

3. Indexing in List.

- Say you have the list ['cat', 'bat', 'rat', 'elephant'] stored in a variable named spam.
- The Python code spam[0] would evaluate to 'cat', and spam[1] would evaluate to 'bat', and so on.
- The integer inside the square brackets that follows the list is called an index.
- The first value in the list is at index 0, the second value is at index 1, the third value is at index 2, and so on.
- Figure (1) shows a list value assigned to spam, along with what the index expressions would evaluate to.

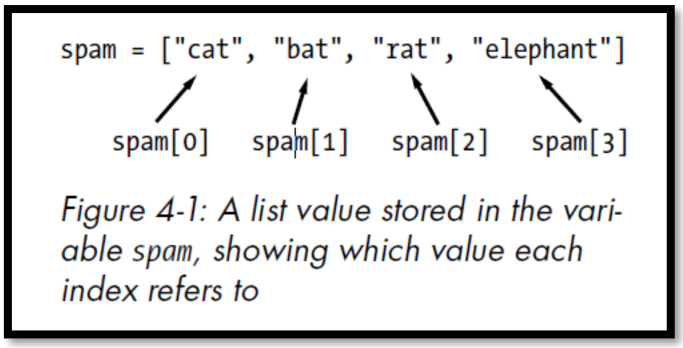

spam = ["cat", "bat", "rat", "elephant"]
spam[0]
spam[1]
spam[2]
spam[3]
Figure 4-1: A list value stored in the variable spam, showing which value each index refers to

Fig. 1: A list value stored in the variable spam, showing which value each index refer to

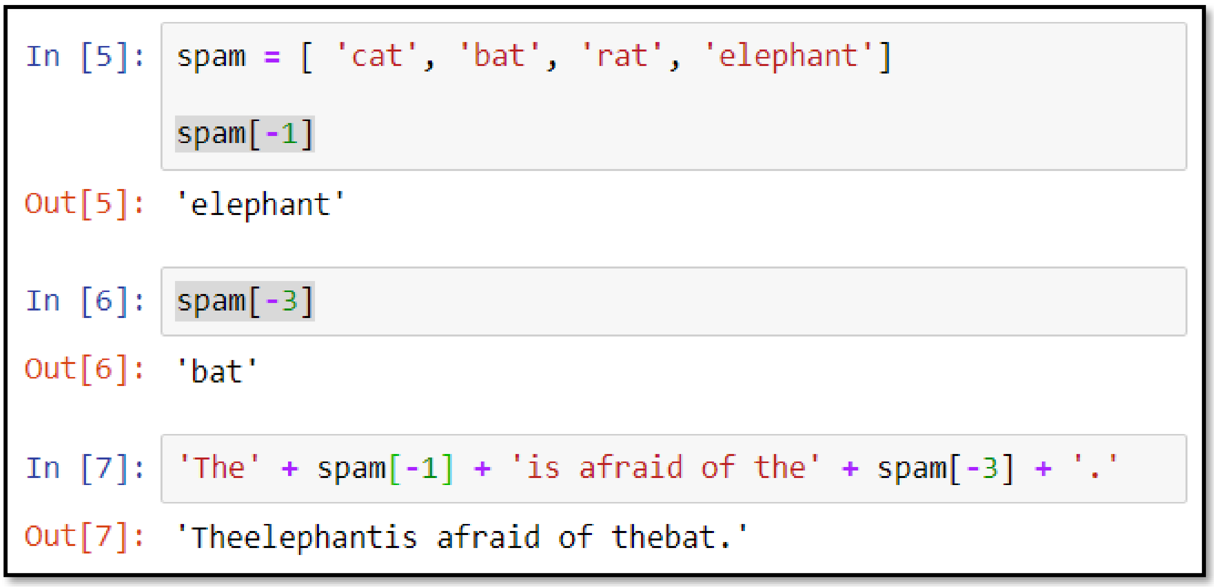
Points to remember about indexing:

1. Python will give you an IndexError error message if you use an index that exceeds the number of values in your list value.
2. Indexes can be only integer values, not floats. Python will give you a TypeError error, if you use float in indexes.

4. Negative indexing.

While indexes start at 0 and go up, you can also use negative integers for the index. The integer value -1 refers to the last index in a list, the value -2 refers to the second-to-last index in a list, and so on.

Fig. 2 shows an example of negative indexing.

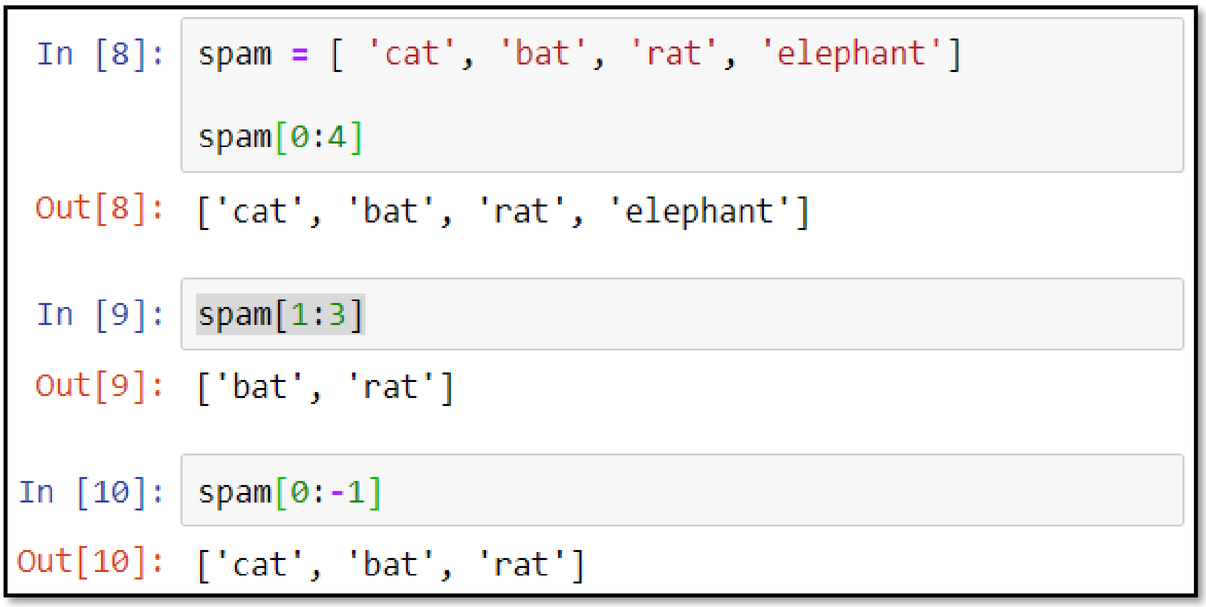


```
In [5]: spam = [ 'cat', 'bat', 'rat', 'elephant']

        spam[-1]
Out[5]: 'elephant'

In [6]: spam[-3]
Out[6]: 'bat'

In [7]: 'The' + spam[-1] + 'is afraid of the' + spam[-3] + '.'
Out[7]: 'Theelephantis afraid of thebat.'
```

Fig. 2

5. Slicing in list.

- Just as an index can get a single value from a list, a slice can get several values from a list, in the form of a new list.
- A slice is typed between square brackets, like an index, but it has two integers separated by a colon.
- Notice the difference between indexes and slices.
  - spam[2] is a list with an index (one integer).
  - spam[1:4] is a list with a slice (two integers).
- In a slice, the first integer is the index where the slice starts.
- The second integer is the index where the slice ends.
- A slice goes up to, but will not include, the value at the second index. A slice evaluates to a new list value.

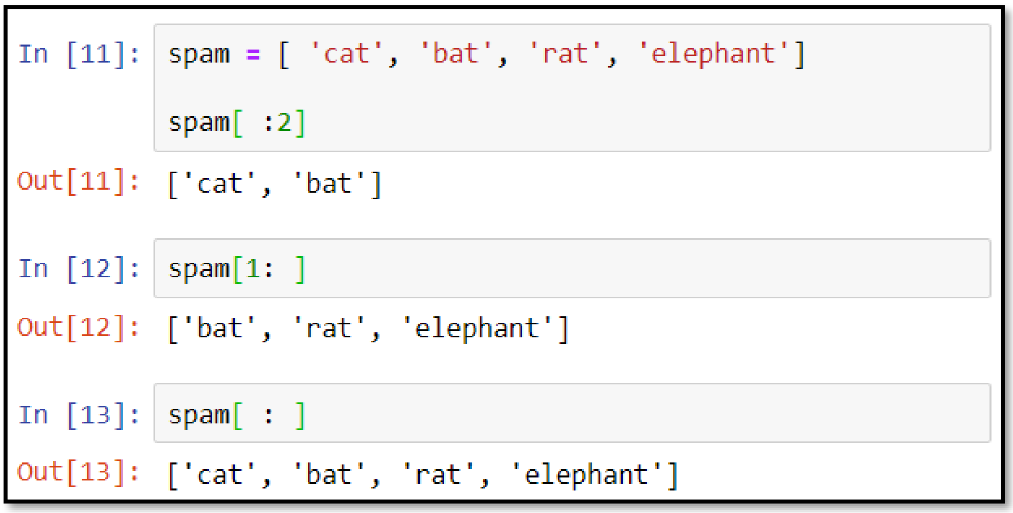


```
In [8]:  spam = [ 'cat', 'bat', 'rat', 'elephant']

          spam[0:4]
Out[8]:  ['cat', 'bat', 'rat', 'elephant']

In [9]:  spam[1:3]
Out[9]:  ['bat', 'rat']

In [10]: spam[0:-1]
Out[10]: ['cat', 'bat', 'rat']
```

Fig. 3 Example of slicing

- As a shortcut, you can leave out one or both of the indexes on either side of the colon in the slice. Leaving out the first index is the same as using 0, or the beginning of the list.
- Leaving out the second index is the same as using the length of the list, which will slice to the end of the list.

Example:



```
In [11]: spam = [ 'cat', 'bat', 'rat', 'elephant']

          spam[ :2]
Out[11]: ['cat', 'bat']

In [12]: spam[1: ]
Out[12]: ['bat', 'rat', 'elephant']

In [13]: spam[ : ]
Out[13]: ['cat', 'bat', 'rat', 'elephant']
```

6. Methods for list

As list in an object, it has some methods attached to it.

Some of the methods are:

1. **append( ) :** The append( ) method adds an item to the end of the list.

   The append( ) method adds a single item to the end of the list.
   Syntax: list.append(item)

2. **insert( ) :**

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

   Syntax: list.insert(index, element)
   index : the index where you want to insert the element and element is the element that you want to insert.

3. **remove( ) :**

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

4. **reverse( ) :**

   The reverse( ) method reverses the elements of the given list.
   Syntax: list.reverse( )

There are many more methods. You can find more about them in the python documentation. We will be discussing them more in the Jupyter notebook.

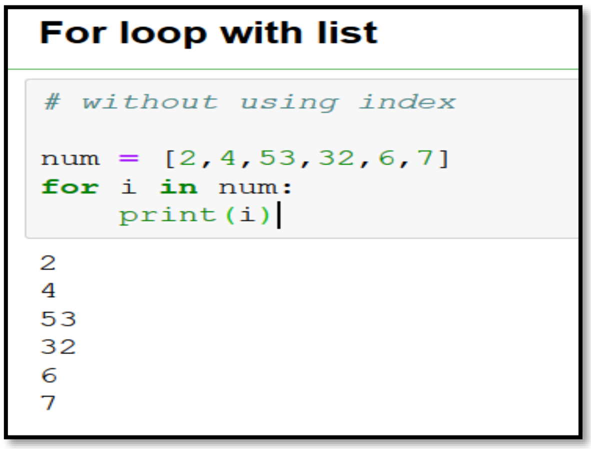
7. Using Loops with list.

You have learned the concept of loops in previous chapter. So I will not be repeating it. But in this section we will learn how to use loops with list.

Loops are usually used to iterate though the list.

We can iterate through the list using loop in 2 ways:

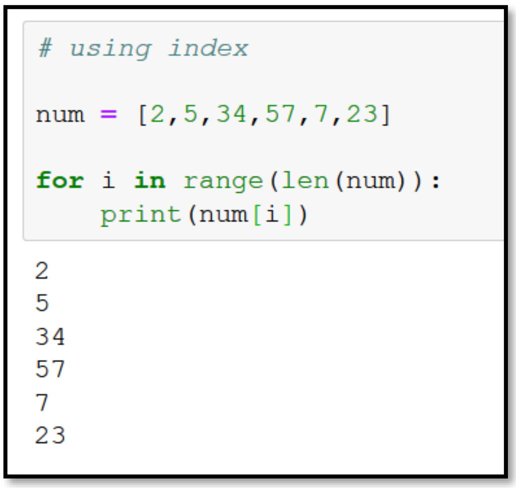
i. Without using index like in the below example


For loop with list

```
# without using index

num = [2,4,53,32,6,7]
for i in num:
    print(i)
```

```
2
4
53
32
6
7
```

ii. Using index



```
# using index

num = [2,5,34,57,7,23]

for i in range(len(num)):
    print(num[i])
```

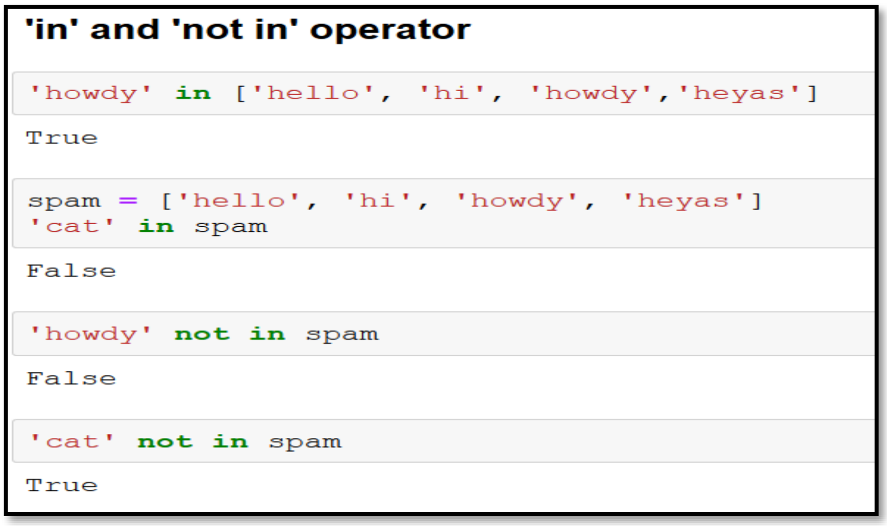
```
2
5
34
57
7
23
```

Here the *len( )* function returns the length of the list. And range function generates a list of numbers from 0 to length-1.

8. The ‘in’ and ‘not in’ operator in list.

- Sometimes we may need to find whether an element is present in the list or not and do something depending on the result.
- For this purpose, we have the ‘in’ and ‘not in’ operators.
- You can determine whether a value is or isn’t in a list with the in and not in operators.
- Like other operators, **in** and **not in** are used in expressions and connect two values: a value to look for in a list and the list where it may be found. These expressions will evaluate to a Boolean value.

Example:



'in' and 'not in' operator

```
'howdy' in ['hello', 'hi', 'howdy','heyas']
```

```
True
```

```
spam = ['hello', 'hi', 'howdy', 'heyas']
'cat' in spam
```

```
False
```

```
'howdy' not in spam
```

```
False
```

```
'cat' not in spam
```

```
True
```

You may watch the below given video to learn more about list.

Video links: https://youtu.be/Eaz5e6M8tL4

References:

1. Automate the boring stuff with python by Al Sweigart (Ebook)
2. https://www.w3schools.com/python/python_lists.asp
3. https://www.programiz.com/python-programming/methods/list