Advanced Programming-Python

Presented by

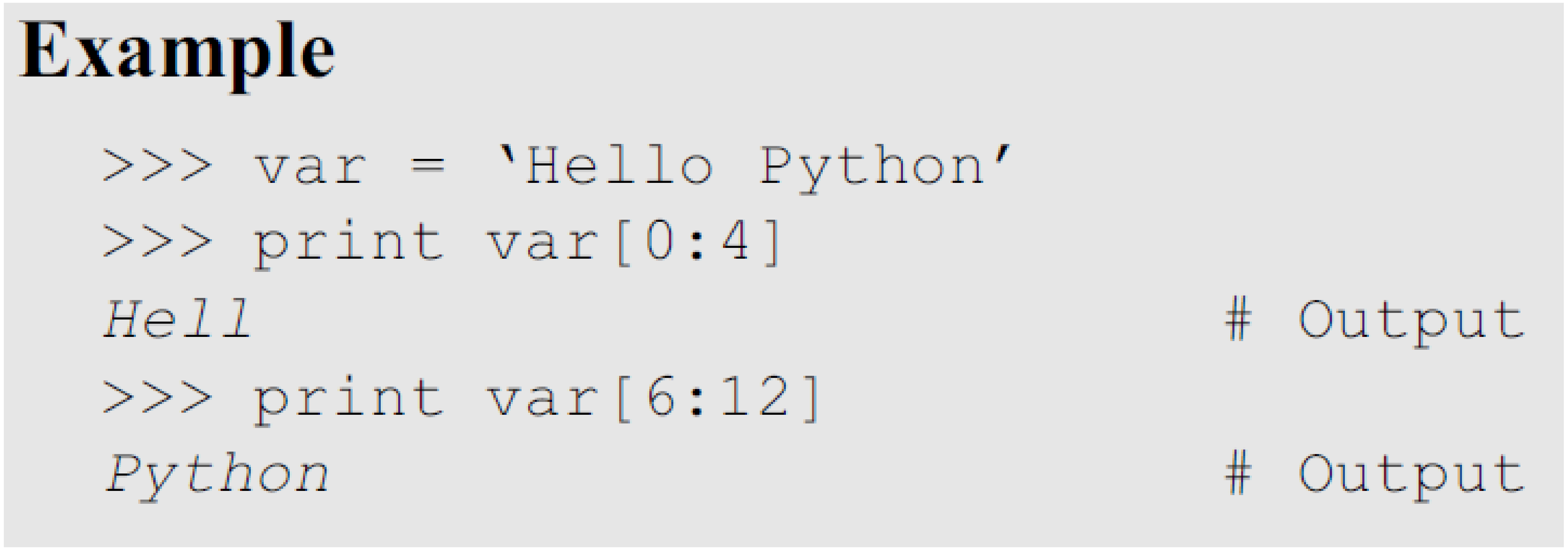
Siddhanta Borah

Contents of Todays (Day-7) Discussion

- Operations of String
  - String Slices
  - Len function
  - String traversal
  - Strings are immutable
- Operations of List
  - Accessing elements from list
  - Lists are mutable
  - Traversing a list
  - Deleting elements from list
  - Built-in list operators

STRING SLICES

- A piece or subset of a string is known as slice. Slice operator is applied to a string with the use of square braces ([]). Operator *[n:m*] will give a substring which consists of letters between n and m indices, including letter at index n but excluding that at m, i.e. letter from nth index to *(m-1)th* index.

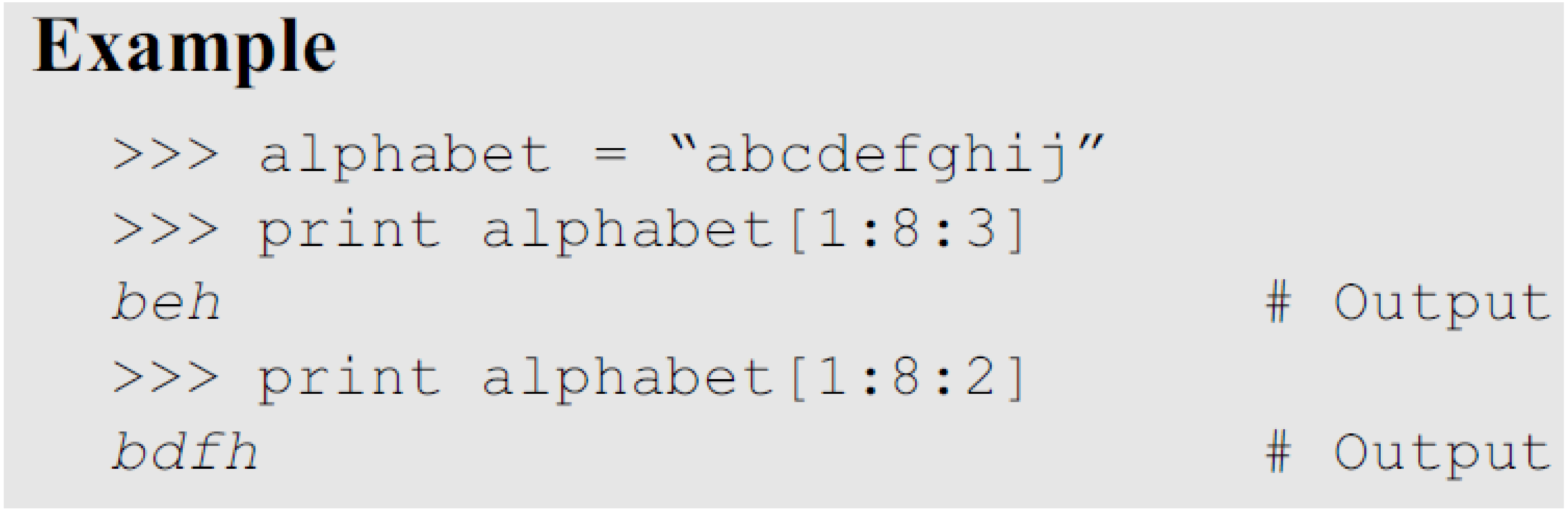


Example

```
>>> var = 'Hello Python'
>>> print var[0:4]
Hell                                  # Output
>>> print var[6:12]
Python                                # Output
```

STRING SLICES

- Similarly, operator *[n:m:s]* will give a substring which consists of letters from nth index to (m-1)th index, where s is called the step value, i.e. after letter at n, that at n+s will be included, then n+2s, n+3s, etc…

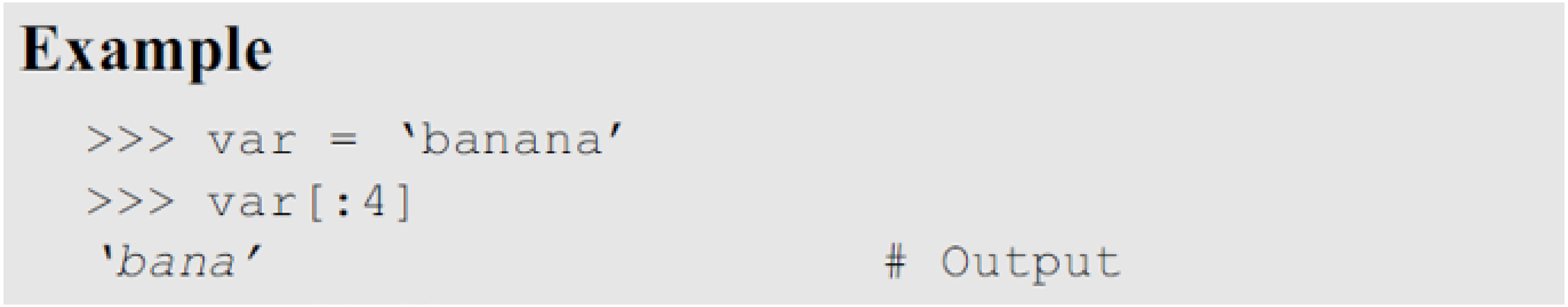


Example

```
>>> alphabet = "abcdefghij"
>>> print alphabet[1:8:3]
beh                                    # Output
>>> print alphabet[1:8:2]
bdfh                                   # Output
```

SOME MORE STRING SLICE OPERATION

Now, if we do not give any value for the index before the colon, i.e., n, then the slice will start from the first element of the string. Similarly, if we do not give any value for the index, i.e., m after the colon, the slice will extend to the end of the string.



Example

```
>>> var = 'banana'
>>> var[:4]
 'bana'                          # Output
```

SOME MORE STRING SLICE OPERATION

Similarly, if we don't give any value at both the sides of the colon, i.e., values for `n` and `m` are not given then it will print the whole string.

```
>>> var[ : ]
'banana'                        # Output
```

Now, if the second index, i.e., `m`, is smaller than the first index, i.e., `n`, then output will be an empty string represented by two single quotes:

```
>>> var = 'banana'
>>> var[4:3]
''                              # Output
```

Now, if we give the value of step as -1 and no value for `n` and `m`, then it will print the string in reverse order. For example,

```
>>> var = 'banana'
>>> var[ : : -1]
'ananab'                        # Output
```

LEN FUNCTION

`len` is a built-in function in Python. When used with a string, `len` returns the length or the number of characters in the string.

Example

```
>>> var = "Hello Python!"
>>> len(var)
13                              # Output
```

Here, we took a string `'Hello Python!'` and used the `len` function with it. The `len` function returned the value `13` because not only characters, but also special characters and blank spaces are considered in the string. Therefore, the *blank space* and *exclamation mark* in our string will also be counted as elements.

LEN FUNCTION

Example

Access the last letter of our string with the help of length.

```
>>>length = len(var)
>>> last = [length - 1]
>>> print last
!                              # Output
```

In this example, we store the length of the string in a variable `length` to access the last element of our string. To access the last element, we need the index of last element which is `12` and not 13 because the index of the string starts with 0 not 1, as we discussed earlier. This is the reason we subtracted 1 from the length of the string in order to get the last element index.

Alternatively, we can also use negative indices for accessing the string from last. So, the expression `var [-1]` yields the last letter of the string and `var [-2]` yields the second last letter and so on.

LEN FUNCTION

Example

```
>>> var = "Hello world"
>>> last = var[-1]
>>> second_last = var[-2]
>>> print last
d                              # Output
>>> print second_last
l                              # Output
```

STRING TRAVERSAL

Traversal is a process in which we access all the elements of the string one by one using some conditional statements such as for loop, while loop, etc. String traversal is an important pattern since there will be many situations in different programs where we need to visit each element of the string and do some operations, continuing till the end of the string.

STRING TRAVERSAL

Example

Let us try the traversal of string using `while` loop

```
>>> i=0
>>> while i < len(var):    # string was assigned
... letter = var[i]
... print letter
... i = i + 1
```

Output:

```
h
e
l
l
o

p
y
t
h
o
n
```

IMMUTABLE STRING

Strings are immutable which means that we cannot change any element of a string. If we want to change an element of a string, we have to create a new string.

Example

```
>>> var = 'hello python'
>>> var[0] = 'p'
```

Output:

```
TypeError: 'str' object does not support item assignment
```

Here, we try to change the 0th index of the string to a character `p`, but the Python interpreter generates an error.

OPERATIONS OF LIST-ACCESSING ELEMENTS FROM LIST

Like strings, lists are also a series of values in Python. In a string, all the values are of character type but in a list, values can be of any type. The values in a list are called elements or items.

A list is a collection of items or elements; the sequence of data in a list is ordered. The elements or items in a list can be accessed by their positions, i.e., indices. We have already studied the index in the strings section.

Like all other variables, lists are also defined before they are used. There are several ways of defining or creating a list. The most convenient way is using square brackets (`[]`).

Example

```
>>> list1 = [2,-1,0,-2,8]
>>> list2 = ['crunchy chocolate', 'hello', 'python programming']
```

OPERATIONS OF LIST-ACCESSING ELEMENTS FROM LIST

The first line list contains only the numerals that are integers. The second line list contains strings. However, it is not necessary that the lists have homogenous data type elements.

There can be elements of different data types in the list:

```
>>> list3 = ['python', 5.5, 8]
```

The list given above contains three different types of elements: string, float and integer. A list-type data item can also be defined inside a list:

```
>>> list4 = ['python', 5.6, [20,40]]
```

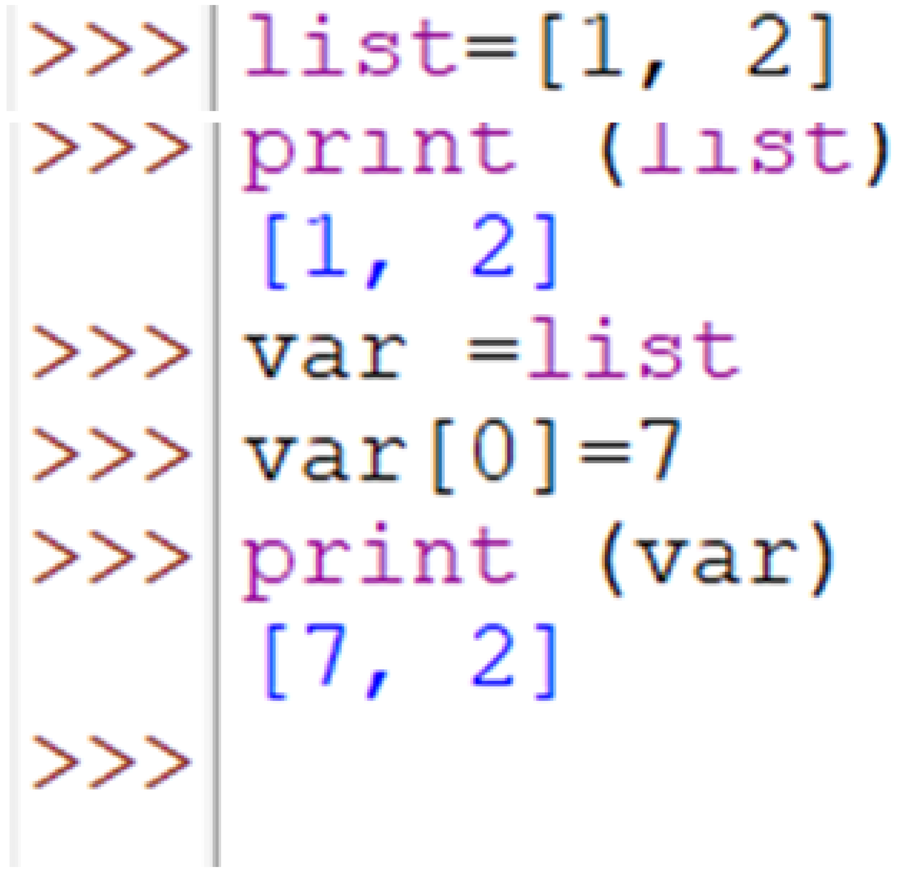
Here, a list is contained in another list. Alternatively, we can say that a list is **nested** within another list.

OPERATIONS OF LIST- MUTABLE LIST

Lists are mutable. The value of any element inside the list can be changed at any point of time. The elements of the list are accessible with their index value. The index always starts with `0` and ends with `n-1`, if the list contains `n` elements. The syntax for accessing the elements of a list is the same as in the case of a string. We use square brackets around the variable and index number.

Example

```
>>> list = [10,20,30,40]
>>> list[1]
20            # Output
```



```
>>> list=[1, 2]
>>> print (list)
[1, 2]
>>> var =list
>>> var[0]=7
>>> print (var)
[7, 2]
>>>
```

OPERATIONS OF LIST- TRAVERSING LIST

Traversing a list means accessing all the elements or items of the list. Traversing can be done by using any conditional statement of Python, but it is preferable to use `for` loop. Traversing in list is done in the same way as in string.

Example

```
>>> list = ['a','b','c','d']
>>> for x in list:
...    print x
```

Output:

```
a
b
c
d
```

OPERATIONS OF LIST- DELETING ELEMENT FROM LIST

Python provides many ways in which the elements in a list can be deleted.

1. pop Operator If we know the index of the element that we want to delete, then we can use the `pop` operator.

```
>>> list = [10,20,30,40]
>>> a = list.pop(2)
>>> print list
[10,20,40]                    # Output
>>> print a
30                            # Output
```

2. del Operator The `del` operator deletes the value on the provided index, but it does not store the value for further use.

```
>>> list = ['w','x','y','z']
>>> del list(1)
>>> print list
['w', 'y', 'z']               # Output
```

OPERATIONS OF LIST- DELETING ELEMENT FROM LIST

3. `remove` Operator We use the `remove` operator if we know the item that we want to remove or delete from the list (but not the index).

```
>>> list = [10,20,30,40]
>>> list.remove(10)
>>> print list
[20,30,40]                    # Output
```

Note *In order to delete more than one value from a list,* `del` *operator with* `slicing` *is used.*

```
>>> list = [1,2,3,4,5,6,7,8]
>>> del list[1:3]
>>> print list
[1,4,5,6,7,8]                 # Output
```

OPERATIONS OF LIST- BUILD IN LIST OPERATORS

1. Concatenation The concatenation operator works in lists in the same way it does in a string. This operator concatenates two strings. This is done by the + operator in Python.

Example

```
>>> list1 = [10,20,30,40]
>>> list2 = [50,60,70]
>>> list3 = list1 + list2
>>> print list3
[10,20,30,40,50,60,70]          # Output
```

In the given example, there are two lists, `list1` and `list2`. Here, `list1` and `list2` are concatenated using + operator between them and the resulting list is stored in the variable `list3`. Now, when we print `list3`, it gives the concatenation of `list1` and `list2`.

OPERATIONS OF LIST- BUILD IN LIST OPERATORS

2. Repetition The repetition operator works as suggested by its name; it repeats the list for a given number of times. Repetition is performed by the `*` operator.

Example

```
>>> list1 = [1,2,3]
>>> list1 * 4
[1,2,3,1,2,3,1,2,3,1,2,3]          # Output
>>> [2] * 6
[2,2,2,2,2,2]                      # Output
```

In the given example, the list `[1,2,3]` was repeated 4 times and the list `[2]` was repeated 6 times.

OPERATIONS OF LIST- BUILD IN LIST OPERATORS

3. `In` Operator The `In` operator tells the user whether the given string exists in the list or not. It gives a Boolean output, i.e., `True` or `False`. If the given input exists in the string, it gives `True` as output, otherwise, `False`.

Example 1

```
>>> list = ['Hello', 'Python', 'Program']
>>> 'Hello' in list
True          # Output
>>> 'World' in list
False                  # Output
```

Example 2

```
>>> list = [10,20,30,40]
>>> 10 in list
True# Output
>>> 50 in list
False                  # Output
```

OPERATIONS OF LIST- BUILD IN LIST OPERATORS

Sr. No.	Method	Description
1.	`cmp(list1,list2)`	It compares the elements of both the lists, `list1` and `list2`.
2.	`len(list)`	It returns the length of the string, i.e., the distance from starting element to last element.
3.	`max(list)`	It returns the item that has the maximum value in a list.
4.	`min(list)`	It returns the item that has the minimum value in a list.
5.	`list(seq)`	It converts a tuple into a list.
6.	`list.append(item)`	It adds the item to the end of the list.
7.	`list.count(item)`	It returns number of times the item occurs in the list.
8.	`list.extend(seq)`	It adds the elements of the sequence at the end of the list.
9.	`list.index(item)`	It returns the index number of the item. If item appears more than one time, it returns the lowest index number.
10.	`list.insert(index,item)`	It inserts the given item onto the given index number while the elements in the list take one right shift.
11.	`list.pop(item=list[-1])`	It deletes and returns the last element of the list.
12.	`list.remove(item)`	It deletes the given item from the list.
13.	`list.reverse()`	It reverses the position (index number) of the items in the list.
14.	`list.sort([func])`	It sorts the elements inside the list and uses compare function if provided.

OPERATIONS OF LIST- BUILD IN LIST OPERATORS

Example

```
>>> list = [1,2,3,4,]
>>> list.append(0)
>>> print list
[1,2,3,4,0]
```

Example

```
>>> list1 = ['x','y','z']
>>> list2 = [1,2,3]
>>> list1.extend(list2)
>>> print list1
['x', 'y', 'z', 1, 2, 3]
```

Example

```
>>> list=[4,2,5,8,1,9]
>>> list.sort()
>>> print list
[1, 2, 4, 5, 8, 9]                # Output
```

PROBLEM FROM LIST AND STRING

- Write a python program to add two matrices.
- Write a python program to multiply two matrices.
- Write a python program to find the transpose of a matrix.
- Write a python program to check whether a string is palindrome or not.
- Write a python program to remove punctuations from a string.

Thank You