Lists in Python

- List can be written as a list of comma-separated values (items) between square brackets.
- A list is that items in a list need not be of the same type.
- Ex:

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5 ]
list3 = ["a", "b", "c", "d"]
```

Accessing Values in Lists

- Similar to string indices, list indices start at 0.
- Ex:

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5, 6, 7 ]
print("list1[0]: ", list1[0])
print("list2[1:5]: ", list2[1:5])
```

Output:

```
list1[0]:  physics
list2[1:5]:  [2, 3, 4, 5]
```

Deleting List Elements

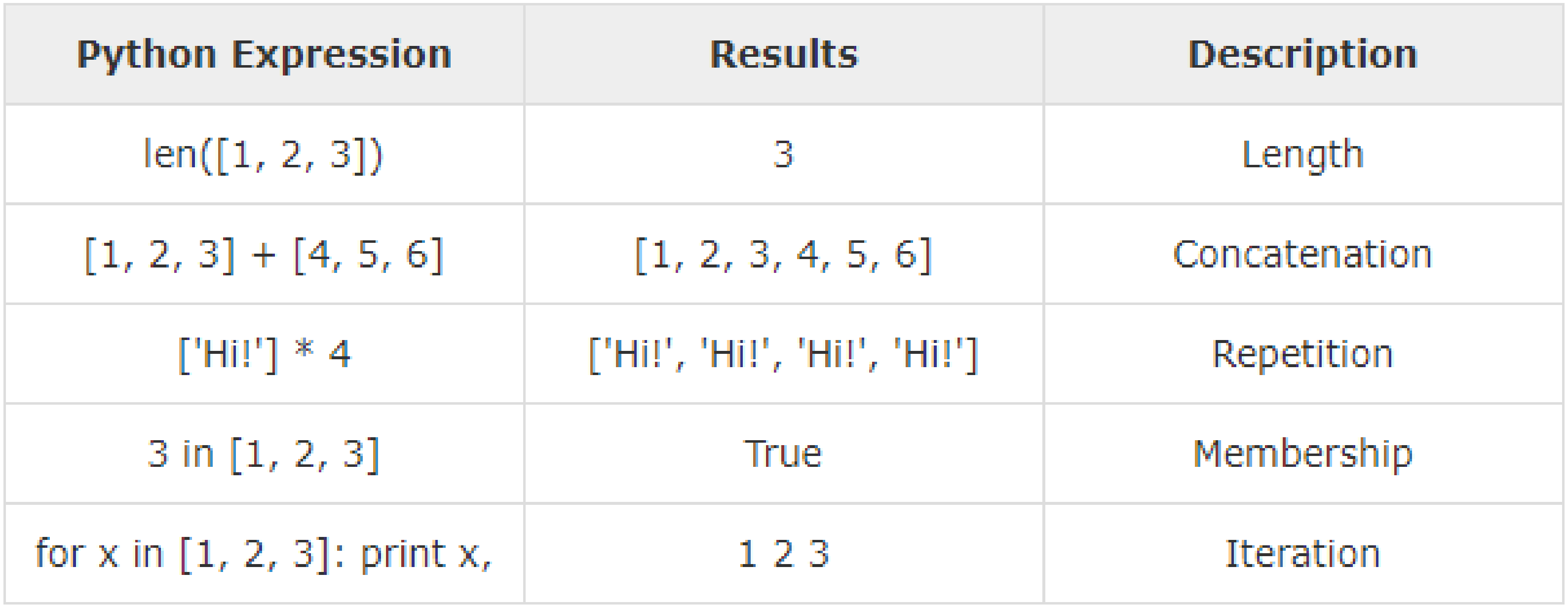
- Ex:

```
list1 = ['physics', 'chemistry', 1997, 2000]
print(list1)
del list1[2]
print("After deleting value at index 2 : ")
print(list1)
```

Output:

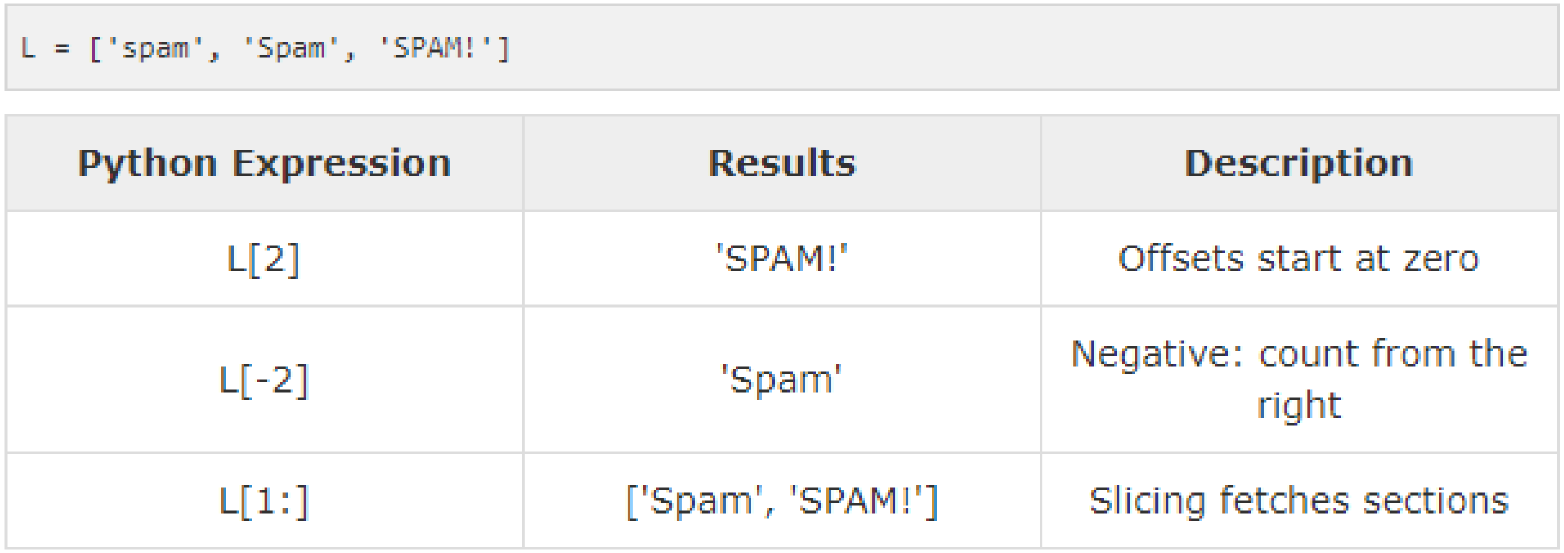
```
['physics', 'chemistry', 1997, 2000]
After deleting value at index 2 : ['physics',
'chemistry', 2000]
```

Basic List Operations



Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Indexing, Slicing, and Matrixes



```
L = ['spam', 'Spam', 'SPAM!']
```

Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

- For lists, a single position returns a values, a slice returns a list.
- Ex:

```
l=[1,2,5,10]
print(l[0])
print(l[1:2])
```

Output:

```
1
[2]
```

Nested Lists

- Ex:

```
nested = [[2,[37]],4,["hello"]]

print(nested[0])          # [2, [37]]
print(nested[1])          # 4
print(nested[2])          # ['hello']
print(nested[0][0])       # 2
print(nested[0][1])       #[37]
print(nested[0][1][0])    #37
print(nested[2][0])       #hello
print(nested[2][0][0])    #h
print(nested[0][1:2])     #[[37]]
```

Updating Lists

- Ex:

```
list = ['physics', 'chemistry', 1997, 2000]
print("Value available at index 2 : ", list[2])
list[2] = 2001
print("New value at index 2 : ", list[2])
```

Output:

```
Value available at index 2 : 1997
New value available at index 2 : 2001
```

- Ex:

```
nested = [[2,[37]],4,["hello"]]
nested[1]=7
print(nested)
nested[0][1][0]=19
print(nested)
```

Output:

```
[[2, [37]], 7, ['hello']]
[[2, [19]], 7, ['hello']]
```

Mutable vs. Immutable

- Values of type *int, float, bool, str* are immutable.
- In immutable data types, updating one value does not affect the copy. Assignment **makes a fresh copy** of a value.
- Ex:

```
x=5
y=x
print(y)        # 5
x=7
print(y)        # 5
```

- For mutable values, assignment **does not make** a fresh copy.
- List is mutable.

- Ex:

```
l1=[1,3,5,7]
l2=l1
l1[2]=4
print(l1[2])
print(l2[2])
```

- Output:

```
4
4
```

- l1 and l2 are two names for the **same** list.
- What if we don't want this?

- We can use slice, because it creates list from old one.
- Recall that

  l[:k] is l[0:k]

  l[k:] is l[k:len(l)]

- Omitting both ends gives **a full slice.**
- To make copy of a list, use a full slice.

  l2 = l1 [:]

- Ex:

```
l1=[1,3,5,7]
l2=l1[:]
l1[2]=4
print(l1)
print(l2)
```

- Output:

```
[1, 3, 4, 7]
[1, 3, 5, 7]
```

Digression on equality

- Consider the following lists:

```
l1=[1,3,5,7]
l2=[1,3,5,7]
l3=l2
```

- All three lists are equal, but there is a difference.
  - l1 and l2 are lists with same value.
  - l2 and l3 are two names for same list.

- x==y checks is x and y have same value.
- x is y checks if x and y refer to same object.

- l1 **==** l2 is True.
- l2 **==** l3 is True.

- l2 **is** l3 is True.
- l1 **is** l2 is False.

Concatenation

- Lists can be glued together using +.
- Note that + always produces a new list.
- Ex:

```
l1=[1,3,5,7]
l2=l1
l1=l1+[9]
print(l1)       #[1, 3, 5, 7, 9]
print(l2)       #[1, 3, 5, 7]
```

Built-in List Functions

- **len(list)**

Ex:

```
list1, list2 = [123, 'xyz', 'zara'], [456, 'abc']
print("First list length : ", len(list1))
print("Second list length : ", len(list2))
```

Output:

```
First list length : 3
Second list length : 2
```

- **max(list)**

Ex:

```
        list1, list2 = [123, 'xyz', 'zara', 'abc'], [456, 
700, 200]
        print("Max value element : ", max(list1))
        print("Max value element : ", max(list2))
```

Output:

```
        TypeError
        700
```

- **min(list)**

Ex:

```
        list1, list2 = [123, 'xyz', 'zara', 'abc'], [456, 700, 
200]
        print("Max value element : ", max(list1))
        print("Max value element : ", max(list2))
```

Output:

```
        TypeError
        200
```

- **list(seq)**

  Converts tuple into list.

  Ex:

```
aTuple = (123, 'xyz', 'zara', 'abc')
aList = list(aTuple)
print("List elements : ", aList)
```

Output:

```
List elements : [123, 'xyz', 'zara', 'abc']
```

Methods

- **list.append()**

Ex:

```
aList = [123, 'xyz', 'zara', 'abc']
aList.append(2009);
print("Updated List : ", aList)
```

Output:

```
        Updated List : [123, 'xyz', 'zara', 'abc',
2009]
```

- **list.extend()**

Ex:

```
aList = [123, 'xyz', 'zara', 'abc', 123]
bList = [2009, 'manni']
aList.extend(bList)
print("Extended List : ", aList)
```

Output:

```
Extended List : [123, 'xyz', 'zara', 'abc',
123, 2009, 'manni']
```

- Ex:

```
l1=[1,3,5,7]
l1.append(9)
print(l1)
#l1.extend(10) #not allowed
#l1.extend([10]) #allowed
l1.extend([6,8])
l1.append([1,2])
print(l1)
```

- Output:

```
[1, 3, 5, 7, 9]
[1, 3, 5, 7, 9, 6, 8, [1, 2]]
```

- **list.count()**

Ex:

```
aList = [123, 'xyz', 'zara', 'abc', 123]
print("Count for 123 : ", aList.count(123))
print("Count for zara : ", aList.count('zara'))
```

Output:

```
Count for 123 : 2
Count for zara : 1
```

- **list.copy()**

Ex:

```
list1 = ['cat', 0, 6.7]
new_list = list1.copy()
#new_list = list1[:]
new_list.append('dog')
print('Old List: ', list1)
print('New List: ', new_list)
```

Output:

```
Old List: ['cat', 0, 6.7]
New List: ['cat', 0, 6.7, 'dog']
```

- **list.index()**

Ex:

```
        aList = [123, 'xyz', 'zara', 'abc']
        print("Index for xyz : ", aList.index( 'xyz' ))
        print("Index for zara : ", aList.index( 'zara'
))
```

Output:

```
        Index for xyz : 1
        Index for zara : 2
```

- **list.insert()**

Ex:

```
aList = [123, 'xyz', 'zara', 'abc']
aList.insert( 3, 2009)
print("Final List : ", aList)
```

Output:

```
Final List : [123, 'xyz', 'zara', 2009, 'abc']
```

#try aList.insert(6, [2009,2008])

- **list.pop()**

Ex:

```
aList = [123, 'xyz', 'zara', 'abc']
print("A List : ", aList.pop())
print(aList)
print("B List : ", aList.pop(2))
print(aList)
```

Output:

```
A List : abc
[123, 'xyz', 'zara']
B List : zara
[123, 'xyz']
```

- **list.remove()**

Ex:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
aList.remove('xyz');
print("List : ", aList)
aList.remove('abc');
print("List : ", aList)
```

Output:

```
List : [123, 'zara', 'abc', 'xyz']
List : [123, 'zara', 'xyz']
```

- **list.reverse()**

Ex:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
aList.reverse();
print("List : ", aList)
```

Output:

```
List : ['xyz', 'abc', 'zara', 'xyz', 123]
```

- **list.sort()**

Ex:

```
        aList = ['xyz', 'zara', 'abc', 'xyz'];
        aList.sort();
        print("List : ", aList)
```

Output:

```
        List :  ['abc', 'xyz', 'xyz', 'zara']
```

```
#Try aList.sort(reverse=True)
```

- Ex:

```
l=[1,2,3,4]
l[1:4]=[6,7]
print(l)
l[1:1]=[8,9]
print(l)
l[1]=[8,9]
print(l)
```

Output:

```
[1, 6, 7]
[1, 8, 9, 6, 7]
[1, [8, 9], 9, 6, 7]
```

Problem:

- Define a function which can generate and print a list where the values are square of numbers between 1 and 20 (both included).

```
def printList():
    li=list()
    for i in range(1,21):
        li.append(i**2)
    print(li)
printList()
```

Output:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144,
169, 196, 225, 256, 289, 324, 361, 400]
```

```
#Try with print(li[:5]), print(li[-5:]), print(li[5:]),
print(li[1:5])
```

Problem:

- Write a Python program to check a list is empty or not.

```
l = []
if not l:
        print("List is empty")
```

# **Python not keyword** is a logical operator which is usually used for figured out the negation or opposite boolean value of the operand. The Keyword '*not*' is a unary type operator which means that it takes only one operand for the logical operation and returns the complementary of the boolean value of the operand. For example, if we will give false as an operand to the not keyword we get true as the value of returns.

Predict output:

-
  ```
  l=[[]] * 4
  print(l)
  ```

-
  ```
  l=[['Hi'] * 4]
  print(l)
  ```

-
  ```
  l=[[] * 4]
  print(l)
  ```

-
  ```
  l=[['Hi' * 4]]
  print(l)
  ```

-
  ```
  l=[['Hi']] * 4
  print(l)
  ```

Predict output:

-
  ```
  l=[[]] * 4
  print(l)      #output: [[], [], [], []]
  ```

-
  ```
  l=[['Hi'] * 4]
  print(l)        #output: [['Hi', 'Hi', 'Hi', 'Hi']]
  ```

-
  ```
  l=[[] * 4]
  print(l)        #output: [[]]
  ```

-
  ```
  l=[['Hi' * 4]]
  print(l)        #output: [['HiHiHiHi']]
  ```

-
  ```
  l=[['Hi']] * 4
  print(l)        #output: [['Hi'], ['Hi'], ['Hi'], ['Hi']]
  ```

Problem:

Write a Python program to multiply all the items in a list.

```
def multiply_list(items):
        tot = 1
        for x in items:
                tot *= x
        return tot
print(multiply_list([1,2,-8]))

'''
#summation of list elements
x=0
for i in [1,2,3,4]:
  x=x+i
print(x)
'''
```

Problem:

Write a Python program to concatenate elements of a list.(with and without join method)

Hint: Convert elements in to string first.

```
#Without using built-in function join
def lists(list1):
    answer=''
    for number in list1:
        answer+=str(number)
    print(int(answer))


lists([12,4,15,11])
```

```
color = ['red', 'green', 'orange']
print('-'.join(color))
print(''.join(color))
```

#Python join list means concatenating a list of strings with a specified delimiter to form a string. Sometimes it's useful when you have to convert list to string. For example, convert a list of alphabets to a comma-separated string to save in a file.

List Comprehension

```
x = [i for i in range(10)]
print(x)
# This will give the output:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Syntax:

```
[ expression for item in list if condition ]
```

```
# You can either use loops:
squares = []
for x in range(10):
   squares.append(x**2)
print(squares)
```

Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

```
# Or you can use list comprehensions to get
   the same result:
squares = [x**2 for x in range(10)]
print(squares)
```

Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Problem:

- Multiply every part of a list by three and assign it to a new list.

Problem:

- Multiply every part of a list by three and assign it to a new list.

```
list1 = [3,4,5]
multiplied = [i*3 for i in list1]
Print(multiplied)
```

Output: [9,12,15]

Problem:

- Use a list comprehension to square each odd number in a list. The list is input by a sequence of comma-separated numbers.
- Suppose the following input is supplied to the program:

```
[1,2,3,4,5,6,7,8,9]
```

Then, the output should be:

```
[1,3,5,7,9]
[1,9,25,49,81]
```

```
Way 1:
values = [1,2,3,4,5,6,7,8,9,10]
numbers = [x*x for x in values if int(x)%2
!=0]
print(numbers)

Way 2:
list1=[]
for i in list:
 if i%2!=0:
  k=i*i
  list1.append(k)
print(list1)
```

Problem

- Print first letter of each word.

Problem

- Print first letter of each word.

```
l = ["this","is","a","list","of","words"]
l1 = [i[0] for i in l]
print(l1)
```

Output: ['t', 'i', 'a', 'l', 'o', 'w']

Problem

- Write a program to print the list after removing even numbers in it.

```
li = [5,6,77,45,22,12,24]
li = [x for x in li if x%2!=0]
print(li)
```

Problem

- Write a program to print the list after removing numbers which are divisible by 5 and 7.

Ex:

```
li = [12,24,35,70,88,120,155]
li = [x for x in li if x%5!=0 and x%7!=0]
print(li)
```

Output:

```
[12, 24, 88]
```

Problem

- write a program to print the list after removing the value 24 in [12,24,35,24,88,120,155].

```
li = [12,24,35,24,88,120,155]
li = [x for x in li if x!=24]
print(li)
```

Problem

- write a program using list comprehension to print the Fibonacci Sequence.
- If the following n is given as input to the program:

    7

    Then, the output of the program should be:

    [0,1,1,2,3,5,8,13]

```
def f(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return f(n-1)+f(n-2)

n=int(input())
values = [f(x) for x in range(0, n+1)]
print(values)
```

Problem

- Write a program generate a 3*5*8 3D array whose each element is 0.

- array = [0 for col in range(8)]
  print(array)

#output: [0, 0, 0, 0, 0, 0, 0, 0]

- array = [[0 for col in range(8)]]
  print(array)

#output: [[0, 0, 0, 0, 0, 0, 0, 0]]

- array = [[0 for col in range(8)] for i in range(5)]
  print(array)

#output: [[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0]]

- array = [[[0 for col in range(8)] for col in range(5)] for row in range(3)]

```
print(array)
#output:
[[[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0]], [[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0]], [[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0]]]
```