Python - List

- **List.** Is a collection of items of same or different data types.
- A list stores elements one after another with comma separator.
- List is index based with index starting with zero.
- Elements are enclosed in square brackets [ ]
- Elements in the list can repeat
- It does not provide fast lookups.
- Finding an element is often slow.
- A search is required.

Python - List

- List is collection of elements of different types:
- >>> listMixed = [1,1.5,'A','KMIT Hyderabad']
- >>> print(listMixed)
- [1, 1.5, 'A', “KMIT Hyderabad”]

Python - List

- List can be collection of elements of same type
- >>> list = ['Ram', 'Shyam', 'Ravi', 'Kishor']
- >>> print(list)
- ['Ram', 'Shyam', 'Ravi', 'Kishor']
- String can be in single or double quotes
- >>> listDoubleQuote = ["one", ‘two’,"three"]
- >>> print(listDoubleQuote)
- ['one', 'two', 'three']

Exercise

- Write a program to create a list “items” and initialize it with 5 elements.
- Let first three elements be strings having the first element of the list repeating in the list and other two elements as integer and float values.
- Print the data type of each element in the list
- print(type(items[1]),type(items[4]),type(items[5]))

Exercise

```
items = [ "book", "computer", ‘book’,1,1.5]
for element in items:
        print(type(element))
```

Output

```
<class 'str'>
<class 'str'>
<class 'str'>
<class 'int'>
<class 'float'>
```

Python - List

- **Append:**This method is called upon the list instance (which must not equal None). It receives the value we are adding.
- **Append**
- list = []
- list.**append**(1)
- list.append(2)
- list.append(6)
- list.append(3)
- print(list)
- **Output**
- [1, 2, 6, 3]

Demo

```
mylist = []
mylist.append(1)
mylist.append(2)
mylist.append(3)
print(mylist[0]) # prints 1
print(mylist[1]) # prints 2
print(mylist[2]) # prints 3
mylist.append("Ram")
mylist.append('A')
mylist.append(15.56)
print(mylist[3]) # prints Ram
print(mylist[4]) # prints A
print(mylist[5]) # prints 15.56
```

Output

```
1
2
3
Ram
A
15.56
```

Demo

```
mylist = []
mylist.append(1)
mylist.append(2)
mylist.append(3)
print(mylist[0]) # prints 1
print(mylist[1]) # prints 2
print(mylist[2]) # prints 3
mylist.append("Ram")
mylist.append('A')
mylist.append(15.56)
print(mylist[3]) # prints Ram
print(mylist[4]) # prints A
print(mylist[5]) # prints 15.56
Print(mylist[6])
```

Output

```
1
2
3
Ram
A
15.56
Traceback (most recent call last):
  File
"D:\UBG\Python\PythonCode\Li
stCode\ListDemo1.py", line 14,
in <module>
    print(mylist[6]) # prints 15.56
IndexError: list index out of range
```

Python - List

- **Insert.** An element can be added anywhere in a list. With insert() we can add to the first part or somewhere in the middle of the list.
- Lists are indexed starting at zero—they are zero-based.
- The index 1 indicates the second element location.
- Example:
- list = ["dot", "perls"]
- # Insert at index 1.
- list.**insert**(1, "net")
- print(list)

Python - List

- **Extend.** A list can be appended to another list with extend().
- So we extend one list to include another list at its end.
- We concatenate (combine) lists.
- **Caution:**If we try to call append() to add a list, the entire new list will be added as a single element of the result list.
- **Tip:** Another option is to use a for-loop, or use the range syntax to concatenate (extend) the list.

Python - List

- **Extend.**
- A list can be appended to another list with extend().
- We extend one list to include another list at its end.
- We concatenate (combine) lists.
- **Caution:**If we try to call append() to add a list, the entire new list will be added as a single element of the result list.
- **Tip:**Another option is to use a for-loop, or use the range syntax to concatenate (extend) the list.

Python - List

- # Two lists.
- a = [1, 2, 3]
- b = [4, 5, 6]
- # Add all elements in list b to list a.
- **a.extend**(b)
- List a now contains six elements.
- print(a)
- **Output** [1, 2, 3, 4, 5, 6]

Python - List

- **Len.** A list contains a certain number of elements. This may be zero if it is empty. With len, a built-in method, we access the element count.
- **Python program that uses len**
- animals = []
- animals.append("cat")
- animals.append("dog")
- count = **len**(animals)
- # Display the list and the
- length. print(animals)
- print(count)
- **Output**
- ['cat', 'dog']
- 2

Python - List

- in keyword. Is an element in a list?
- We use "in" and "not in“ keywords as membership operators to determine:
- Is an element in a list?
- Other approaches are possible but "in" is simplest.
- Here we search a list with "in" and "not in."

Python - List

```
items = [‘computer’,”atlas”,1,1.5,”,marks”]
if "computer" in items:
    print(1)
if "atlas" in items:
    # This is not reached.
    print(2)
else:
    print(3)
if "marker" not in items:
    print(4)
```

Python - List

- **Sort, reverse.** Lists maintain the order of their elements. And they can be reordered.
- With the sort method, we change the order of elements from low to high.
- With reverse, we invert the current order of the elements.
- Sort and reverse can be combined (for a reversed sort).

Python - List

- list = [400, 500, 100, 2000]
- # Reversed.
- list.**reverse**()
- print(list)
- # Sorted.
- list.**sort**()
- print(list)
- # Sorted and reversed.
- list.reverse()
- print(list)
- **Output**
- [2000, 100, 500, 400]
- [100, 400, 500, 2000]
- [2000, 500, 400, 100]

Python - List

- **Remove, del.** Remove acts upon a value.
- It first searches for that value, and then removes it.
- Elements (at an index) can also be removed with the del statement.
- **Remove:**This takes away the first matching element in the list. We can call it multiple times, and it could keep changing the list.
- **Del:**This meanwhile removes elements by index, or a range of indices. Del uses the slice syntax.
- names = ["Tommy", "Bill", "Janet", "Bill", "Stacy"]
- # Remove this value.
- names.**remove**("Bill")
- print(names)
- # Delete all except first two elements.
- **del** names[2:]
- print(names)
- # Delete all except last element.
- **del** names[:1]
- print(names)
- **Output** ['Tommy', 'Janet', 'Bill', 'Stacy']
- ['Tommy', 'Janet']
- ['Janet']

Python - List

- **Count.** This method does not return the number of elements in the list. Instead it counts a specific element. As an argument, pass the value of the element you wish to count.
- **Note:**Internally count() loops through all the elements and keeps track of the count. It then returns this value.

Python - List

- **Python that uses count**
- names = ['a', 'a', 'b', 'c', 'a']
- # Count the letter a.
- value = names.**count**('a')
- print(value)
- **Output**
- 3

Python - List

- **Index.** This searches lists.
- We pass it an argument that matches a value in the list.
- It returns the index where that value is found.
- If no value is found, it throws an error.
- **Tip:**For programs where you need more control, please consider using a for-loop instead of index().
- **Info:**With a for-loop, we can more elegantly handle cases where the value is not found. This avoids the ValueError.

Python - List

- # Input list.
- values = ["uno", "dos", "tres", "cuatro"]
- # Locate string.
- n = values.**index**("dos")
- print(n, values[n])
- # Locate another string.
- n = values.**index**("tres")
- print(n, values[n])
- # Handle nonexistent string.

```
try:
    n = values.index("?") # Not reached.
    print(n)
except: # Value not found.
    print("Not found")
```

- **Output** 1 dos 2 tres Not found

Python - List

- # Input list.
- values = ["uno", "dos", "tres", "cuatro"]
- # Locate string.
- n = values.**index**("dos")
- print(n, values[n])
- # Locate another string.
- n = values.**index**("tres")
- print(n, values[n])
- n = values.**index**("?")
- **Output**
- 1 dos
- 2 tres
- Not found

Python - List

- **For-loop.** In list loops, we often need no index. We require just the elements in order. The for-loop is ideal in this situation. It eliminates the confusion of an index variable.
- **Here:**We encounter each of the four list elements in the for-loop. We use the identifier "element" in the loop body.

Python - List

- # An input list.
- elements = ["spider", "moth", "butterfly", "lizard"]
- # Use simple for-loop.
- **for** ele in elements:
- print(ele)
- **Output**
- spider moth butterfly lizard

Python - List

- **Min, max.** We do not need to search for the smallest or largest element in a list. Instead we use max and min. Internally this searches.
- **Here:**Max returns the value 1000, which is larger than all other elements in the list. And min returns negative -100.

Python - List

- values = [-100, 1, 10, 1000]
- # Find the max and min elements.
  print(**max**(values))
- print(**min**(values))

Python – List
Exercise -1

- Create a list having names of any number of students and another list having marks of any number of subjects.
- Prompt the user to enter name of a student and then his marks.
- Display average marks of above student.
- Repeat above for all students.
- Finally display the highest average marks and the name of student

Python - List

```
marks=[]
names=[]
avg1=[]
num=int(input("Enter the number of students:"))
num1=int(input("Enter the number of subjects:"))
for i in range(1,num+1):
    name=str(input("Name of student %d: " %i))
    names.append(name)
    marks=[]
    for j in range(1,num1+1):
        mark=int(input("Enter subject %d mark : " %j))
        marks.append(mark)
    tot=sum(int(j) for j in marks)
    print("Student " , i," marks are: ")
    print(marks)

    print("Total marks:  " ,tot)
    avg=tot/num1
    avg1.append(avg)
    print("Average marks obtained:  ",avg)
    for k in range(len(marks)):
        if(marks[k]<40):
            print("Failed")
        else:
            print("Passed")
print("Highest average is: %d\n Name: %s\n"%(max(avg1),name))
```

Python - List