Python Collections (Arrays)

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

List is a collection which is ordered and changeable. Allows duplicate members.

Tuple is a collection which is ordered and unchangeable. Allows duplicate members.

Set is a collection which is unordered and unindexed. No duplicate members.

Dictionary is a collection which is unordered, changeable and indexed. No duplicate members.

- The most basic data structure in Python is the **sequence**. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.
- There are certain things you can do with all sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership. In addition, Python has built-in functions for finding the length of a sequence and for finding its largest and smallest elements.

Python Lists

- A list is a collection which is ordered and changeable. In Python lists are written with square brackets.
- In Python programming, a list is created by placing all the items (elements) inside square brackets `[]`, separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.).
- Important thing about a list is that items in a list need not be of the same type.

Example:

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

```
# Creating a List
List = []
print("Blank List: ")
print(List)
   # Creating a List of numbers
   List = [10, 20, 14]
   print("\nList of numbers: ")
   print(List)
 # Creating a List of strings and accessing using index
List = ["Geeks", "For", "Geeks"]
print("\nList Items: ")
print(List[0])
print(List[2])
```

```
# Creating a Multi-Dimensional List  (By Nesting a list inside a List)
List = [['Geeks', 'For'] , ['Geeks']]
print("\nMulti-Dimensional List: ")
print(List)
```

Output:

```
Blank List:
[]

List of numbers:
[10, 20, 14]

List Items
Geeks
Geeks

Multi-Dimensional List:
[['Geeks', 'For'], ['Geeks']]
```

Accessing the items of a list

Syntax to access the list items:

list_name[index]

```
# a list of numbers
numbers = [11, 22, 33, 100, 200, 300]
# prints 11
print(numbers[0])
# prints 300
print(numbers[5])
# prints 22
print(numbers[1])
```

Output

```
11
300
22
```

Points to Note:

1. The index cannot be a float number.

For example:

```
# a list of numbers
numbers = [11, 22, 33, 100, 200, 300]


# error
print(numbers[1.0])
```

Output:

```
TypeError: list indices must be integers or slices, not float
```

The index must be in range to avoid IndexError. The range of the index of a list having 10 elements is 0 to 9, if we go beyond 9 then we will get IndexError. However if we go below 0 then it would not cause issue in certain cases

For example:

```
# a list of numbers
numbers = [11, 22, 33, 100, 200, 300]

# error
print(numbers[6])
```

Output:

```
IndexError: list index out of range
```

Negative Index to access the list items from the end

- Unlike other programming languages where negative index may cause issue, Python allows you to use negative indexes.
- The idea behind this to allow you to access the list elements starting from the end. For example an index of -1 would access the last element of the list, -2 second last, -3 third last and so on.

```
# a list of strings
my_list = ["hello", "world", "hi", "bye"]

# prints "bye"
print(my_list[-1])

# prints "world"
print(my_list[-3])

# prints "hello"
print(my_list[-4])
```

Output

bye
World
hello

Creating a list with multiple distinct or duplicate elements

- A list may contain duplicate values with their distinct positions and hence, multiple distinct or duplicate values can be passed as a sequence at the time of list creation.

```
List = [1, 2, 4, 4, 3, 3, 3, 6, 5]
print("\nList with the use of Numbers: ")
print(List)

List = [1, 2, 'Geeks', 4, 'For', 6, 'Geeks']
print("\nList with the use of Mixed Values: ")
print(List)
```

Output:

List with the use of Numbers:

[1, 2, 4, 4, 3, 3, 3, 6, 5]

List with the use of Mixed Values:

[1, 2, 'Geeks', 4, 'For', 6, 'Geeks']

Knowing the size of List

```
# Creating a List
List1 = []
print(len(List1))

# Creating a List of numbers
List2 = [10, 20, 14]
print(len(List2))
```

Output:

```
0
3
```

How to get a sublist in Python using slicing

- We can get a sublist from a list in Python using slicing operation. Lets say we have a list n_list having 10 elements, then we can slice this list using colon : operator.

```
# list of numbers
n_list = [1, 2, 3, 4, 5, 6, 7]

# list items from 2nd to 3rd
print(n_list[1:3])

# list items from beginning to 3rd
print(n_list[:3])

# list items from 4th to end of list
print(n_list[3:])

# Whole list
print(n_list[:])
```

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

```
list1 = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(list1[-4:-1])
```

Output:

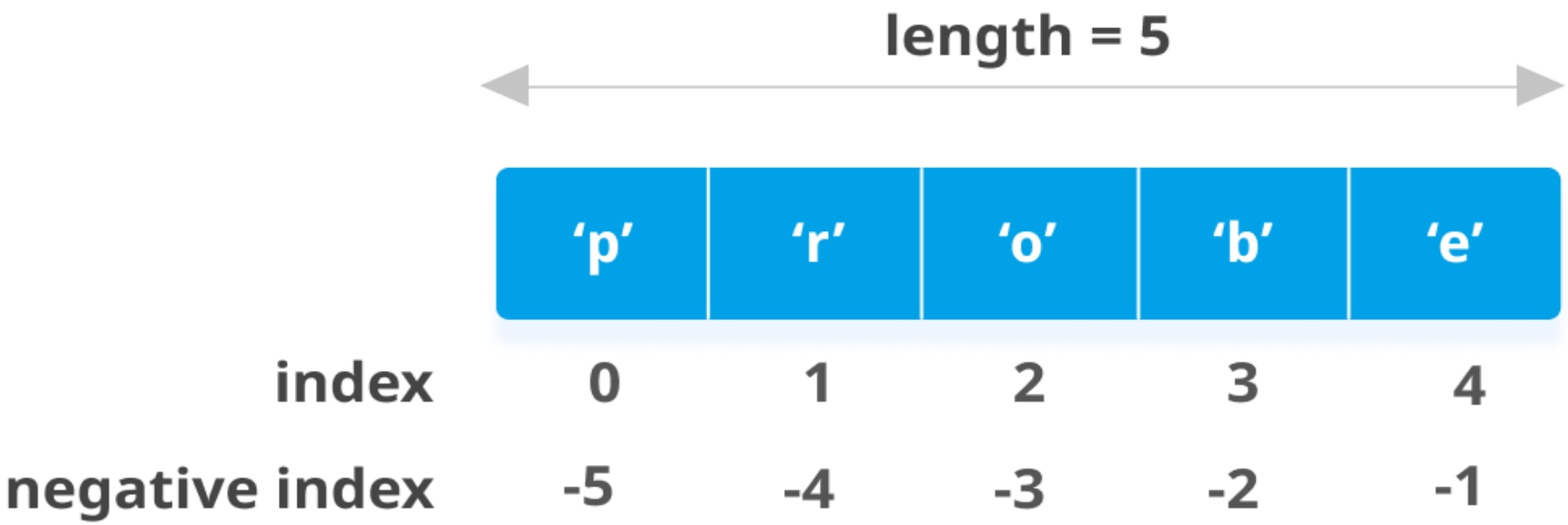
['orange', 'kiwi', 'melon']

EX:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Output:

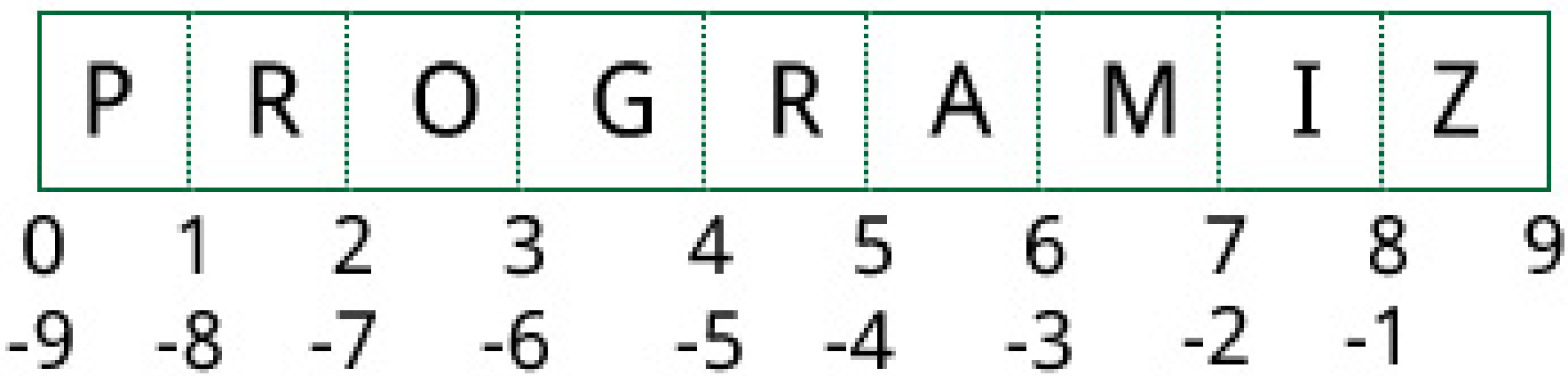
['apple', 'blackcurrant', 'cherry']


length = 5
'p'
'r'
'o'
'b'
'e'
index
0
1
2
3
4
negative index
-5
-4
-3
-2
-1

```
# List slicing in Python

my_list = ['p','r','o','g','r','a','m','i','z']

# elements 3rd to 5th
print(my_list[2:5])

# elements beginning to 4th
print(my_list[:-5])

# elements 6th to end
print(my_list[5:])

# elements beginning to end
print(my_list[:])
```

Output

```
['o', 'g', 'r']
['p', 'r', 'o', 'g']
['a', 'm', 'i', 'z']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```


P R O G R A M I Z
0 1 2 3 4 5 6 7 8 9
-9 -8 -7 -6 -5 -4 -3 -2 -1

How to change or add elements to a list?

- Lists are mutable, meaning their elements can be changed unlike string or tuple.
- We can use the assignment operator (=) to change an item or a range of items.

```
# Correcting mistake values in a list
 odd = [2, 4, 6, 8]
 # change the 1st item
 odd[0] = 1
 print(odd)
 # change 2nd to 4th items
 odd[1:4] = [3, 5, 7]
print(odd)
```

Output

[1, 4, 6, 8]
[1, 3, 5, 7]

- We can add one item to a list using the **append()** method

```
odd = [1, 3, 5]
odd.append(7)
 print(odd)
```

Output

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

- add several items using extend() method.

```
odd = [1, 3, 5]
odd.extend([9, 11, 13])
 print(odd)
```

Output

[1, 3, 5, 7,9,11,13]

- We can also use + operator to combine two lists. This is also called concatenation.

```
odd = [1, 3, 5]

print(odd + [9, 7, 5])
```

Output

[1, 3, 5, 9,7,5]

- The * operator repeats a list for the given number of times.

```
print(["re"] * 3)
```

Output

```
['re', 're', 're']
```

- we can insert one item at a desired location by using the method insert()

```
odd = [1, 9]
 odd.insert(1,3)
 print(odd)
```

Output

[1,3,9]

[1,3,5,7,9]

- insert multiple items by squeezing it into an empty slice of a list.

```
odd[2:2] = [5, 7]
 print(odd)
```

How to delete or remove elements from a list?

- We can delete one or more items from a list using the keyword del. It can even delete the list entirely.

```
# Deleting list items
 my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']
# delete one item
 del my_list[2]
 print(my_list)
 # delete multiple items
del my_list[1:5]
 print(my_list)
 # delete entire list
 del my_list
# Error: List not defined
 print(my_list)
```

Output

```
['p', 'r', 'b', 'l', 'e', 'm']
 ['p', 'm']
Traceback (most recent call last): File "
<string>", line 18, in <module>
 NameError: name 'my_list' is not defined
```

- We can use remove() method to remove the given item

```
my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')
print(my_list)
print(my_list.pop(1))
print(my_list)
print(my_list.pop())
print(my_list)
 my_list.clear()
 print(my_list)
```

Output

```
['r', 'o', 'b', 'l', 'e', 'm']
o
['r', 'b', 'l', 'e', 'm']
 m
['r', 'b', 'l', 'e'] []
```

- we can also delete items in a list by assigning an empty list to a slice of elements.

```
>>> my_list = ['p','r','o','b','l','e','m']
 >>> my_list[2:3] = []
 >>> my_list
       ['p', 'r', 'b', 'l', 'e', 'm']
 >>> my_list[2:5] = []
 >>> my_list
       ['p', 'r', 'm']
```

Python List Methods

- Methods that are available with list objects in Python programming are tabulated below.
- They are accessed as **list.method**
- **append() -** Add an element to the end of the list
- **extend()** - Add all elements of a list to the another list
- **insert()** - Insert an item at the defined index
- **remove()** - Removes an item from the list
- **pop()** - Removes and returns an element at the given index
- **clear()** - Removes all items from the list

- **index()** - Returns the index of the first matched item
- **count()** - Returns the count of the number of items passed as an argument
- **sort()** - Sort items in a list in ascending order
- **reverse()** - Reverse the order of items in the list
- **copy()** - Returns a shallow copy of the list

```
# Python list methods
my_list = [3, 8, 1, 6, 0, 8, 4]
 # Output: 1
 print(my_list.index(8))
# Output: 2
print(my_list.count(8))
 my_list.sort()
 # Output: [0, 1, 3, 4, 6, 8, 8]
 print(my_list)
my_list.reverse()
 # Output: [8, 8, 6, 4, 3, 1, 0]
 print(my_list)
```

- **List Membership Test**
- We can test if an item exists in a list or not, using the keyword in.

```
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']
  print('p' in my_list)
 print('a' in my_list)
print('c' not in my_list)
```

Output

True

False

True

- Using a for loop we can iterate through each item in a list.

```
for fruit in ['apple','banana','mango']:
 print("I like",fruit)
```

Output

```
I like apple
 I like banana
 I like mango
```

Built-in functions with List

reduce()apply a particular function passed in its argument to all of the list elements stores the intermediate result and only returns the final summation value

sum()Sums up the numbers in the list

ord()Returns an integer representing the Unicode code point of the given Unicode character

cmp()This function returns 1, if first list is “greater” than second listmax()return maximum element of given list

- max()return maximum element of given list
- min()return minimum element of given list
- all()Returns true if all element are true or if list is empty
- any()return true if any element of the list is true. if list is empty, return false
- len()Returns length of the list or size of the list
- enumerate()Returns enumerate object of list
- accumulate()apply a particular function passed in its argument to all of the list elements returns a list containing the intermediate results
- filter()tests if each element of a list true or not
- map()returns a list of the results after applying the given function to each item of a given iterable
- lambda()This function can have any number of arguments but only one expression, which is evaluated and returned.

- # Python code to demonstrate the working of sum()

```
numbers = [1,2,3,4,5,1,4,5]
  # start parameter is not provided
Sum = sum(numbers)
print(Sum)
  # start = 10
Sum = sum(numbers, 10)
print(Sum)
```

Output:

25

35

```
numbers = [1,2,3,4,5,1,4,5]
# start = 10
Sum = sum(numbers)
average= Sum/len(numbers)
print average
```

Output:

3

- >>> a ['foo', 'bye', 'baz', 'qux', 'quux', 'corge']
- >>> len(a)
- 6
- >>> min(a)
- 'baz‘
- >>> max(a)
- 'qux'