Lists []

- ✔ Creating Lists,
- ✔ Basic List Operations,
- ✔ Indexing and Slicing in Lists,
- ✔ Built-In Functions Used on Lists,
- ✔ List Methods,
- ✔ The del Statement.

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.

Lists-Introduction

- Lists are one of the most flexible data storage formats in Python because they can have values added, removed, and changed.
- You can think of the list as a container that holds a number of items.
- Each element or value that is inside a list is called an item. All the items in a list are assigned to a single variable.
- Lists avoid having a separate variable to store each item which is less efficient and more error prone when you have to perform some operations on these items.
- Lists can be simple or nested lists with varying types of values.

Creating Lists

Lists are constructed using square brackets [ ] wherein you can include a list of items separated by commas.

You can create an empty list without any items. The syntax is,

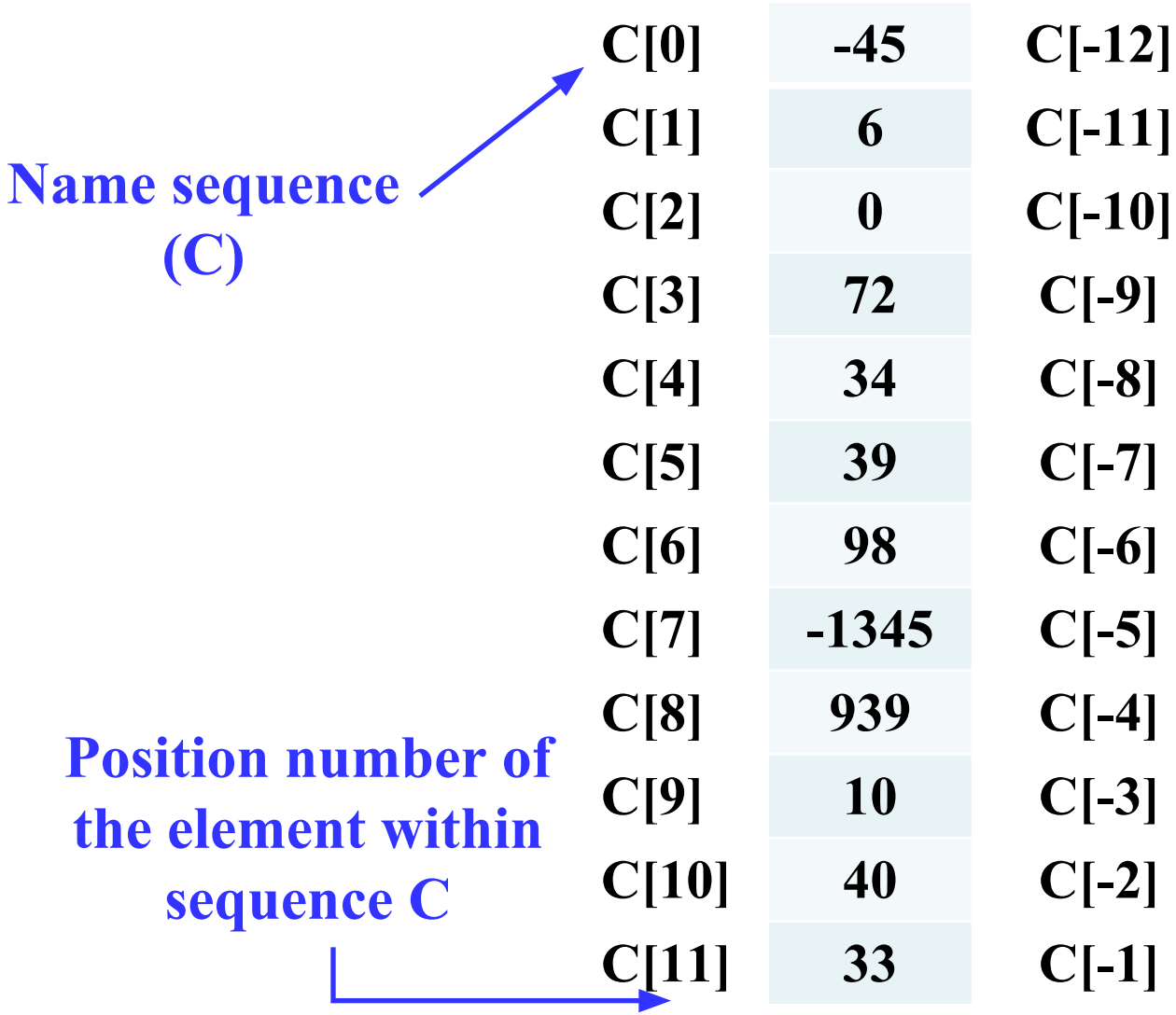
list_name = [ ]

Creating Lists

- number_list = [4, 4, 6, 7, 2, 9, 10, 15]
- vals=[12,'tiger',10.5, True]
- type(vals)
- empty_list = []

You can store any item in a list like string, number, object, another variable and even another list. You can have a mix of different item types and these item types need not have to be homogeneous. For example, you can have a list which is a mix of type numbers, strings and another list itself.

Example of Sequences



Basic List Operations

```
>>> list_1 = [1, 3, 5, 7]
>>> list_2 = [2, 4, 6, 8]
>>> list_1 + list_2
>>> list_1 * 3
>>> list_1 == list_2
>>> 5 in list_1
>>> 10 in list_1
```

The *list()* Function

- The built-in *list()* function is used to create a list. The syntax for *list()* function is,

list([sequence])

where the sequence can be a string, tuple or list itself. If the optional sequence is not specified then an empty list is created.

```
>>> quote = "How you doing?"
>>> string_to_list = list(quote)
>>> string_to_list
>>> friends = ["j", "o", "e", "y"]
>>> friends + quote
>>> friends + list(quote)
```

Indexing and Slicing in Lists

- As an ordered sequence of elements, each item in a list can be called individually, through indexing. The expression inside the bracket is called the index. Lists use square brackets [ ] to access individual items, with the first item at index 0, the second item at index 1 and so on. The index provided within the square brackets indicates the value being accessed.
- The syntax for accessing an item in a list is,

list_name[index]

- where index should always be an integer value and indicates the item to be selected

Indexing and Slicing in Lists

- . For the list *superstore*, the index breakdown is shown below.

Indexing and Slicing in Lists

- superstore = ["metro", "tesco", "walmart", "kmart", "carrefour"]
- >>> superstore[0]
- >>>superstore[9]

- superstore[-3]

Modifying Items in Lists

- Lists are mutable in nature as the list items can be modified after you have created a list. You can modify a list by replacing the older item with a newer item in its place and without assigning the list to a completely new variable.
- >>> fauna = ["pronghorn", "alligator", "bison"]
- >>> fauna[0] = "groundhog"
- >>> fauna
- >>> fauna[-1] = "beaver"
- >>> fauna

Modifying Items in Lists

- When you assign an existing list variable to a new variable, an assignment (=) on lists does not make a new copy. Instead, assignment makes both the variable names point to the same list in memory.

```
>>> zoo = ["Lion", "Tiger", "Zebra"]
>>> forest = zoo
>>> id(forest)
>>> id(zoo)
>>>zoo[0] = "Fox“
>>> zoo
>>> forest
>>> type(zoo)
```

Slicing in Lists

- Slicing of lists is allowed in Python wherein a part of the list can be extracted by specifying index range along with the colon (:) operator which itself is a list.
- The syntax for list slicing is,

- where both start and stop are integer values (positive or negative values).
- List slicing returns a part of the list from the start index value to stop index value which includes the start index value but excludes the stop index value.
- Step specifies the increment value to slice by and it is optional.

Slicing in Lists

- >>> fruits = ["grapefruit", "pineapple", "blueberries", "mango", "banana"]
- >>> fruits[1:3]
- >>> fruits[:3]
- >>> fruits[2:]
- >>> fruits[1:4:2]
- >>> fruits[:]
- >>> fruits[::2]
- >>> fruits[-3:-1]

Built-In Functions Used on Lists

Built-In Functions Used on Lists

- >>> len(lakes)
- >>> numbers = [1, 2, 3, 4, 5]
- >>> sum(numbers)
- >>> max(numbers)
- >>> min(numbers)
- >>> any([1, 1, 0, 0, 1, 0])
- >>> any([0, 0, 0, 0])
- >>> any([0, 1, 0, 0])
- >>> all([1, 1, 1, 1])
- >>> lakes_sorted_new = sorted(lakes)

List Methods

- The list size changes dynamically whenever you add or remove the items and there is no need for you to manage it yourself.
- dir(list)

Populating Lists with Items

```
>>> cities = ["oslo", "delhi", "washington", "london", "seattle", "paris", "washington"]
>>> cities.count('seattle')
>>> cities.count('washington')
>>> cities.index('london')
>>> cities.reverse()
>>> cities
>>> cities.append('brussels')
>>> cities
>>> cities.sort()
>>> cities.pop()
>>> cities.pop(2)
>>> more_cities = ["brussels", "copenhagen"]
>>> cities.extend(more_cities)
>>> cities
>>> cities.remove("brussels")
>>> cities
```

Nested Lists

- The syntax for nested lists is,

- Each list inside another list is separated by a comma. For example,

Nested Lists

```
>>> vals=[12,'tiger',10.5, True ]
>>> numbers = [1, 2, 3, 4, 5]
>>> newlist=[vals,numbers]
asia = [["India", "Japan", "Korea"], ["Srilanka", "Myanmar", "Thailand"],
["Cambodia", "Vietnam", "Israel"]]
>>> asia[0]
>>> asia[0][1]
>>> asia[1][2] = "Philippines"
```

Two-Dimensional Lists

- Two-dimensional list: a list that contains other lists as its elements
  - Also known as nested list
  - Common to think of two-dimensional lists as having rows and columns
  - Useful for working with multiple sets of data
- To process data in a two-dimensional list need to use two indexes
- Typically use nested loops to process

The del Statement

- You can remove an item from a list based on its index rather than its value.
- The difference between del statement and pop() function is that the del statement does not return any value while the pop() function returns a value.
- The del statement can also be used to remove slices from a list or clear the entire list

The del Statement

```
>>> a = [5, -8, 99.99, 432, 108, 213]
>>> del a[0]
>>> a
[-8, 99.99, 432, 108, 213]
>>> del a[2:4]
>>> a
[-8, 99.99, 213]
>>> del a[:]
>>> a
[]
```

Summary

• Lists are a basic and useful data structure built into the Python language.
• Built-in functions include *len()*, which returns the length of the list; *max()*, which returns the maximum element in the list; *min()*, which returns the minimum element in the list and *sum()*, which returns the sum of all the elements in the list.
• An individual elements in the list can be accessed using the index operator [].
• Lists are mutable sequences which can be used to add, delete, sort and even reverse list elements.
• The *sort()* method is used to sort items in the list.
• The *split()* method can be used to split a string into a list.
• Nested list means a list within another list.

```
#Program to Dynamically Build User Input as a List
list_items = input("Enter list items separated by a space ").split()
print(f"List items are {list_items}")
items_of_list = []
total_items = int(input("Enter the number of items "))
for i in range(total_items):
    item = input("Enter list item: ")
    items_of_list.append(item)
print(f"List items are {items_of_list}")
```

Traversing of Lists

```
#Program to Illustrate Traversing of Lists Using the for loop
fast_food = ["waffles", "sandwich", "burger", "fries"]
for each_food_item in fast_food:
    print(f"I like to eat {each_food_item}")
for each_food_item in ["waffles", "sandwich", "burger", "fries"]:
    print(f"I like to eat {each_food_item}")
```

```
#Program to Display the Index Values of Items in List
silicon_valley = ["google", "amd", "yahoo", "cisco", "oracle"]
for index_value in range(len(silicon_valley)):
    print(f"The index value of '{silicon_valley[index_value]}' is {index_value}")
```

Write Python Program to Sort Numbers in a List in Ascending Order Using Bubble Sort by Passing the List as an Argument to the Function Call

```
def bubble_sort(list_items):
    for i in range(len(list_items)):
        for j in range(len(list_items)-i-1):
            if list_items[j] > list_items[j+1]:
                temp = list_items[j]
                list_items[j] = list_items[j+1]
                list_items[j+1] = temp
    print(f"The sorted list using Bubble Sort is {list_items}")
items_to_sort = [5, 4, 3, 2, 1]
bubble_sort(items_to_sort)
```

Find Mean, Variance and Standard Deviation of List Numbers

```
import math
def statistics(list_items):
    mean = sum(list_items)/len(list_items)
    print(f"Mean is {mean}")
    variance = 0
    for item in list_items:
        variance += (item-mean)**2
    variance /= len(list_items)
    print(f"Variance is {variance}")
    standard_deviation = math.sqrt(variance)
    print(f"Standard Deviation is {standard_deviation}")

statistics([1, 2, 3, 4])
```

```
#Input Five Integers (+ve and −ve). Find the Sum of Negative Numbers,
#Positive Numbers and Print Them. Also, Find the Average of All the Numbers
#and Numbers Above Average

def find_sum(list_items):
    positive_sum = 0
    negative_sum =0
    for item in list_items:
        if item > 0:positive_sum = positive_sum + item
        else:negative_sum = negative_sum + item
    average = (positive_sum + negative_sum) / len(list_items)
    print(f"Sum of Positive numbers in list is {positive_sum}", )
    print(f"Sum of Negative numbers in list is {negative_sum}")
    print(f"Average of item numbers in list is {average}")
    print("Items above average are")
    for item in list_items:
        if item > average:print(item)

find_sum([-1, -2, -3, 4.2, 5, 6, -3, 0.5])
```

```
#Write a Program to Find the Transpose of a Matrix
matrix = [[10, 20], [30, 40],[50, 60]]
matrix_transpose = [[0, 0, 0],[0, 0, 0]]
def matirxtrans():
    for rows in range(len(matrix)):
        for columns in range(len(matrix[0])):
            matrix_transpose[columns][rows] = matrix[rows][columns]
    print("Transposed Matrix is")
    for items in matrix_transpose:
        print(items)
matirxtrans()
```

```
# Write Python Program to Add Two Matrices
matrix_1 = [[1, 2, 3], [4, 5, 6],[7, 8, 9]]
matrix_2 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matrix_result = [[0, 0, 0], [0, 0, 0],[0, 0, 0]]
for rows in range(len(matrix_1)):
    for columns in range(len(matrix_2[0])):
        matrix_result[rows][columns] = matrix_1[rows][columns] +
matrix_2[rows][columns]
print("Addition of two matrices is")
for items in matrix_result:
    print(items)
```

```
#Write Python Program to Perform Queue Operations
from collections import deque
def queue_operations():
    queue = deque(["Eric", "John", "Michael"])
    print(f"Queue items are {queue}")
    print("Adding few items to Queue")
    queue.append("Terry")
    queue.append("Graham")
    print(f"Queue items are {queue}")
    print(f"Removed item from Queue is {queue.pop()}")
    print(f"Removed item from Queue is {queue.popleft()}")
    print(f"Queue items are {queue}")

queue_operations()
```