Data structures in Python- Lists

By

Prof. Muhammad Iqbal Bhat
Department of Higher Education
Government Degree College Beerwah

Topics

Basic Data Structures in Python

Lists in Python

Basic Data Structures in Python:

- Data structures are used in programming to organize and store data in a way that enables efficient access and manipulation.
- Python provides several built-in data structures, including:
  - Lists: ordered, mutable collections of elements
  - Tuples: ordered, immutable collections of elements
  - Sets: unordered, mutable collections of unique elements
  - Dictionaries: unordered, mutable collections of key-value pairs

Lists in Python:

- Lists are a fundamental data structure in Python used to store and manipulate collections of data.
- Lists are ordered, mutable collections of elements.
- They can be used to store different types of data such as numbers, strings, and other objects.
- List elements can be accessed and modified using index notation.

Creating Lists:

A list is created using square brackets [ ] and elements separated by commas.

Example: `my_list = [1, 2, 3, 4, 5]`

Lists are mutable, meaning they can be changed after they are created.

Example: `my_list[0] = 6`

Lists are ordered, meaning the elements are stored in a specific order and can be accessed using an index.

Example: `my_list[2] will return 3`

Lists can contain heterogeneous elements, meaning they can store elements of different types.

Example: `my_list = [1, "two", 3.0, [4, 5]]`

Accessing List elements:

List elements can be accessed using index notation.

The index starts at 0 for the first element and increases by 1 for each subsequent element.

Negative indices can be used to access elements from the end of the list.

Example: `my_list = [1, 2, 3, 4, 5]`

my_list[0] will return 1

my_list[-1] will return 5

List Methods:

Python provides several built-in methods for manipulating lists.

Some commonly used list methods include:

- append(): adds an element to the end of the list
- insert(): inserts an element at a specific index
- remove(): removes the first occurrence of a specified element
- pop(): removes and returns the element at a specific index
- sort(): sorts the elements in ascending order
- reverse(): reverses the order of the elements in the list

Examples of usage of list:

Example: my_list = [1, 2, 3, 4, 5]

my_list.append(6) will add 6 to the end of the list

my_list.insert(0, 0) will insert 0 at the beginning of the list

my_list.remove(3) will remove the first occurrence of 3

my_list.pop(1) will remove and return the element at index 1 (i.e., 2)

my_list.sort() will sort the list in ascending order

my_list.reverse() will reverse the order of the elements in the list

Program examples:

- Sorting a list of numbers in ascending order:

```
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
numbers.sort()
print(numbers)
Output: [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
```

- Removing duplicates from a list:

```
fruits = ["apple", "banana", "orange", "apple", "pear",
"orange"]
unique_fruits = list(set(fruits))
print(unique_fruits)
```

- Finding the index of a specific element in a list

```
students = ["Alice", "Bob", "Charlie", "David", "Eve"]
index = students.index("Charlie")
print(index)
```

Program examples:

- Counting the number of occurrences of an element in a list:

```
grades = [80, 90, 75, 85, 90, 95, 80, 85, 90]
count = grades.count(90)
print(count)
```

- `Creating a new list by iterating over an existing list:`

```
numbers = [1, 2, 3, 4, 5]
squares = [x ** 2 for x in numbers]
print(squares)
```

- Combining two lists into a single list:

```
fruits = ["apple", "banana", "orange"]
vegetables = ["carrot", "potato", "onion"]
groceries = fruits + vegetables
print(groceries)
```

Program examples:

- Reversing a list:

```
numbers = [1, 2, 3, 4, 5]
numbers.reverse()
print(numbers)
```

- Slicing a list to get a subset of elements:

```
numbers = [1, 2, 3, 4, 5]
subset = numbers[1:4]
print(subset)
```

- Inserting an element at a specific position in a list:

```
fruits = ["apple", "banana", "orange"]
fruits.insert(1, "pear")
print(fruits)
```

Program examples:

- Removing an element from a list by value:

```
numbers = [1, 2, 3, 4, 5]
numbers.remove(3)
print(numbers)
```

- Checking if an element exists in a list::

```
fruits = ["apple", "banana", "orange"]
exists = "banana" in fruits
print(exists)
```

- Finding the length of a list:

```
numbers = [1, 2, 3, 4, 5]
squares = []
for x in numbers:
    squares.append(x ** 2)
print(squares)
```

Conclusion:

- Lists are a versatile and powerful data structure in Python.
- Python provides several built-in methods for manipulating lists that can help simplify common list operations.
- Familiarizing yourself with list methods can greatly improve your productivity and effectiveness when working with lists in Python.