All You Need to Know About Python Lists

BEGINNER PROGRAMMING PYTHON

Embark on an exciting journey to master Python Lists, one of the most versatile and powerful data structures in the programming world! Whether you're a beginner or looking to level up your Python skills, this guide will provide you with everything you need to create, manipulate, and conquer the dynamic world of Python Lists or list operations in python. Let's dive in and start exploring!

This article was published as a part of the Data Science Blogathon!

Table of contents

What is a Python List?

A Python List is a built-in, ordered, mutable collection of elements or items. Each item in a list can be of any data type, including numbers, strings, or other objects like tuples, dictionaries, or even other lists. Lists are versatile and widely used in Python for tasks like data manipulation, organizing, and storage. They are created using square brackets [] with items separated by commas, like this:

```
my_list = [1, 'apple', 3.14, [1, 2, 3]]
```

Characteristics of a Python List

- **Ordered:** Lists maintain the order of elements, allowing you to access items by their index.
- **Mutable**: Lists can be modified after creation, enabling you to add, remove, or change elements.

- **Dynamic:** Lists can grow or shrink in size as needed, adapting to the number of elements.
- **Heterogeneous:** Lists can store elements of different data types, including other lists or objects.
- **Indexable**: Elements in a list can be accessed, updated, or removed using their index position.
- **Iterable**: Lists can be looped over or iterated through, making them ideal for processing collections of items.

How to Create a Python List?

Creating a list operations in python is simple and straightforward. A list is a built-in data structure in Python that holds an ordered collection of items. These items can be of different data types, including numbers, strings, and even other lists.

Here's how you can create a Python list:

Step 1: Use square brackets `[]` and separate items with commas `,`.

```
pythonCopy codemy_list = [1, 2, 3, 4, 5]
```

In this example, we've created a list called `my_list` containing integers from 1 to 5.

Step 2: You can also create a list with different data types:

```
pythonCopy codemixed_list = [1, 'apple', 3.14, True]
```

Here, `mixed_list` contains an integer, a string, a float, and a boolean value.

Step 3: To create an empty list, simply use an empty set of square brackets:

```
pythonCopy codeempty_list = []
```

Step 4: Alternatively, you can use the `list()` constructor to create a list:

```
pythonCopy codeanother_list = list([1, 2, 3, 4, 5])
```

In this example, we created a list with the same integer values as the first example, but using the `list()` constructor. Remember that lists in Python are mutable, meaning you can modify their content by assigning new values to specific indices, adding items, or removing items.

Accessing Values/Elements in List in Python

To access values or elements in a list operations in python, you can use indexing or slicing. Indexing starts from 0, which means the first element has an index of 0, the second element has an index of 1, and so on. You can also use negative indexing, which starts from the last element (-1), the second last element (-2), and so on.

Here's how to access values in a list operations in python using indexing:

```
pythonCopy codemy_list = [1, 2, 3, 4, 5] # Accessing the first element (index 0) first_element = my_list[0] print(first_element) # Output: 1 # Accessing the third element (index 2) third_element = my_list[2] print(third_element) # Output: 3 # Accessing the last element using negative indexing last_element =
```

```
my_list[-1] print(last_element) # Output: 5 # Accessing the second last element using negative indexing second_last_element = my_list[-2] print(second_last_element) #
```

Output:

```
4
```

To access a range of elements or a subsequence from the list, you can use slicing. Slicing uses a colon `:` to separate the start and end indices. The start index is included, but the end index is excluded.

```
pythonCopy codemy_list = [1, 2, 3, 4, 5] # Slicing from the second element (index 1) to the fourth element (index 3) sub_list = my_list[1:4] print(sub_list) # Output: [2, 3, 4] # Slicing from the beginning to the third element (index 2) first_three_elements = my_list[:3] print(first_three_elements) # Output: [1, 2, 3] # Slicing from the third element (index 2) to the end of the list from_third_to_end = my_list[2:] print(from_third_to_end) # Output: [3, 4, 5] # Slicing the entire list (shallow copy) entire_list = my_list[:] print(entire_list) #
```

Output:

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

Updating List in Python

Remember to use valid indices when accessing list elements. Accessing an out-of-range index will result in an `IndexError`.

In Python, lists are mutable, which means you can update their elements directly. To update a list, you can modify individual elements, add new elements, or remove existing elements.

Here's how you can perform different types of updates on a Python list:

Modifying an Existing Element in Python List

You can update a specific element in a list by using its index:

```
pythonCopy codemy_list = [1, 2, 3, 4, 5] # Updating the second element (index 1) my_list[1] = 20 print(my_list) #
```

Output:

```
[1, 20, 3, 4, 5]
```

Adding New Elements in List Operations in Python

To add new elements to a list, you can use the `append()` method or the `extend()` method, or the `insert()` method:

Using `append()`

The `append()` method adds an element to the end of the list:

```
pythonCopy codemy_list = [1, 2, 3, 4, 5] # Adding a new element to the end of the list my_list.append(6) print(my_list) #
```

Output:

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

Using `extend()`

The `extend()` method adds multiple elements to the end of the list. You can pass any iterable (e.g., list, tuple, string) as an argument:

```
pythonCopy codemy_list = [1, 2, 3, 4, 5] # Adding multiple elements to the end of the list my_list.extend([6, 7, 8]) print(my_list) #
```

Output:

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

Using `insert()`

The `insert()` method adds an element at a specific index in the list. It takes two arguments: the index where the new element will be inserted, and the new element itself:

```
pythonCopy codemy_list = [1, 2, 3, 4, 5] # Adding a new element at index 2 (third position) my_list.insert(2, 2.5) print(my_list) #
```

Output:

```
[1, 2, 2.5, 3, 4, 5]
```

Also Read: 90+ Python Interview Questions

Removing Elements in Python List

To remove elements from a list, you can use the `remove()` method, the `pop()` method, or the `del` statement:

Using `remove()`

The `remove()` method removes the first occurrence of a specified value in the list:

```
pythonCopy codemy_list = [1, 2, 3, 4, 5, 4] # Removing the first occurrence of the value 4 my_list.remove(4) print(my_list) #
```

Output:

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

Using pop( )

The `pop()` method removes the element at a specified index and returns it. If no index is specified, it removes the last element:

```
pythonCopy codemy_list = [1, 2, 3, 4, 5] # Removing the last element removed_element = my_list.pop() print(my_list) # Output: [1, 2, 3, 4] print(removed_element) # Output: 5 # Removing the element at index 1 (second element) removed_element = my_list.pop(1) print(my_list) # Output: [1, 3, 4] print(removed_element) #
```

Output:

```
2
```

List Operations in Python

1. Joining Lists

Joining lists is pretty easy. We can use the concatenation operator '**+**' to join two lists.

```
>>> l1 =[1,2,3,4,5] >>> l2 = [6,7,8,9,10] >>> l1+l2
```

Output:

```
[1,2,3,4,5,6,7,8,9,10]
```

2. Repeating or Replicating Lists

Like strings, we can use the multiplication operator '*' to replicate a list specified number of times.

```
>>>l =[1,2,3] >>> l*2
```

Output:

```
[1,2,3,1,2,3]
```

3. Slicing the Lists

We can use the indexes to extract the list slices. **(Remember the ending element will be the one preceding the element, whose index is given as the second parameter)**.

```
L= [1,2,3,4,5,6,7,8,9,10] print(L[2:7])
```

Output will be :

```
3,4,5,6,7
```

Here, although the specified ending parameter was 7, traversing stopped at 6

Quick trick: To reverse a given list, use the code :

```
#assuming the list variable is L L[: : -1] #
```

Output:

```
this would return the reversed list
```

Working with Python Lists

Appending elements to list: To append an element at the end of a list, we use append() function.

```
L=[1,2,3] L.append(4) print(L)
```

Output:

```
1,2,3,4
```

Updating elements in a list : Syntax is L[index] =

```
L = [1,2,3,4,5,7,7] L[5] =6 print(L)
```

Output:

```
1,2,3,4,5,7,7
```

Deleting elements from a list: The del method deletes the first occurrence of the element from the list.

Syntax :

```
del list_name[]
```

If we write del list_name, the entire list would be deleted.

We can also use the **pop() function**, to delete an element from a given index. If no index is specified, it removes the last element.

Syntax :

```
list_name.pop()
```

List Methods in Python

Method	Code	Description
append()	`list.append(item)`	Adds an item to the end of a list.
extend()	`list.extend(iterable)`	Adds all items in an iterable (e.g. another list) to the end of a list.
insert()	`list.insert(index, item)`	Inserts an item at a specific position in a list.
remove()	`list.remove(item)`	Removes the first occurrence of an item from a list. Raises an error if the item is not found.
pop()	`list.pop([index])`	Removes and returns the item at a specific position in a list. If no index is specified, removes and returns the last item in the list.
index()	`list.index(item[, start[, end]])`	Returns the index of the first occurrence of an item in a list. Raises an error if the item is not found.
count()	`list.count(item)`	Returns the number of times an item appears in a list.
sort()	`list.sort(key=None, reverse=False)`	Sorts the items in a list in ascending order.
reverse()	`list.reverse()`	Reverses the order of the items in a list.
copy()	`list.copy()`	Returns a copy of a list.

Built-in List Methods/Functions in Python

Here's a table of some common built-in list methods and functions in Python along with their descriptions:

Method/Function	Description
len()	Returns the number of items in a list.
max()	Returns the largest item in a list.
min()	Returns the smallest item in a list.
sum()	Returns the sum of all items in a list.
sorted()	Returns a new sorted list from the items in a list.
all()	Returns True if all items in a list are True, otherwise returns False.
any()	Returns True if at least one item in a list is True, otherwise returns False.
enumerate()	Returns an iterator that yields tuples containing the index and value of each item in a list.
filter()	Returns a new list containing only the items in a list for which a function returns True.
map()	Returns a new list containing the result of applying a function to each item in a list.
zip()	Returns an iterator that combines multiple lists into tuples. The iterator stops when the shortest input iterable is exhausted.

Learn More with our Full Python Course

Source: Analytics Vidhya

Want to Become Pro at Python?

Python lists are a powerful and versatile data structure that allow you to store and manipulate collections of items. With their extensive range of built-in methods and functions, you can easily add, remove, sort, search, and iterate over list items to perform complex data processing tasks.

Whether you are a beginner or an experienced Python programmer, mastering lists is an essential skill for data analysis, machine learning, and other applications in the field of data science. If you're looking to enhance your Python skills and become a Full Stack Data Scientist. Our program is designed to provide comprehensive training and hands-on experience with the latest tools and techniques used in data science, including Python programming, machine learning, deep learning, and more. By enrolling in this program, you can gain the knowledge and skills needed to advance your career in data science and become a highly sought-after professional in this fast-growing field.

Conclusion

Python Lists are a fundamental and highly versatile data structure. They allow you to store ordered collections of elements which can be of mixed data types. Lists provide a rich set of built-in methods and operations for adding, removing, modifying, sorting, searching, and manipulating elements. With their dynamic and mutable nature, lists are invaluable for organizing and processing data in Python. Whether you are a beginner or an experienced programmer, mastering lists is essential for unleashing the full power of Python across domains like data science, web development, automation scripting, and more. With this comprehensive guide, you now have the knowledge to confidently work with lists and take your Python skills to new heights.

Frequently Asked Questions?

Q1. What is a Python list?

A. A Python list is a data structure that stores a collection of items, such as numbers, strings, or other objects, in a single variable. It is a mutable, ordered sequence of elements that can be indexed and sliced.

Q2. What is list list [- 1 in Python?

A. In Python, list[-1] returns the last element of the list. It is a shorthand for list[len(list)-1], which can also be used to access the last element of the list.

Q3. How to create list in Python?

A. To create a list in Python, you can use square brackets [] and separate the elements with commas. For example, my_list = [1, 2, "hello", 3.5] creates a list with four elements of different types.

Q4. What are the 15 list function in Python?

A. Here are 15 common list functions in Python: append(), extend(), insert(), remove(), pop(), index(), count(), sort(), reverse(), copy(), len(), max(), min(), sum(), and sorted(). These functions allow you to add, remove, sort, search, and manipulate the elements in a list.

Article Url - https://www.analyticsvidhya.com/blog/2021/07/all-the-basics-to-begin-with-python-lists/

Pinak Datta

2nd year CSE Undergrad who loves to write clean code 😀