

Lists

In this lesson, we define lists, how they are used in Python and how they are different from arrays!

We'll cover the following



- Initializing a list
- Important list functions
 - The append() function
 - The insert() function
 - The remove() function
 - The pop() function
 - The reverse() function
- What is Slicing?
 - Slice Notation Examples
 - Example 1: Indexing elements of a list
 - Example 2: Stepped Indexing
 - Example 3: Initializing list elements
 - Example 4: Deleting elements from a list
 - Example 5: Negative Indexing
 - Example 6: Slicing in Strings

In Python, a list is an ordered sequence of *heterogeneous* elements. In other words, a list can hold elements with different data types. For example,

```
list = ['a', 'apple', 23, 3.14]
```

Initializing a list

```
1 example_list = [3.14159, 'apple', 23] # Create a list of elements
2 empty_list = [] # Create an empty list
3 sequence_list = list(range(10)) # Create a list of first 10 whole numbers
4 print(example_list)
5 print(empty_list)
6 print(sequence_list)
```



Output

0.164s

```
[3.14159, 'apple', 23]
[]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```



So lists can hold integers, strings, characters, functions, and pretty much any other data type including *other lists* simultaneously! Look at the following example. `another_list` contains two lists, a string, and a function! The elements can be accessed or ‘indexed’ using square brackets. The first element in a list is accessed using index 0 (as on **line 7**), the second element is accessed using 1, and so on. So list indices start at 0.

```
1 a_list = [2, 'Educative', 'A']
2
3
4 def foo():
5     print('Hello from foo()!')
6
7
8 another_list = [a_list, 'Python', foo, ['yet another list']]
9
10 print(another_list[0]) # Elements of 'aList'
11 print(another_list[0][1]) # Second element of 'aList'
12 print(another_list[1]) # 'Python'
13 print(another_list[3]) # 'yet another list'
14
15 # You can also invoke the functions inside a list!
16 another_list[2]() # 'Hello from foo()!'
```



Output

0.236s

```
[2, 'Educative', 'A']
Educative
Python
['yet another list']
Hello from foo()!
```

Important list functions

Let’s have a look at some useful built-in Python list functions. The time complexity of each of these operations is the asymptotic average case taken from the Python Wiki (<https://wiki.python.org/moin/TimeComplexity>) page. A word of caution though: don’t use these to replace your interview answers. For example, if you are asked to sort an array/list, don’t simply use the list `sort()` function to answer that question!

The `append()` function

Use this function to add a single element to the end of a list. This function works in $O(1)$, constant time.

```
1 list = [1, 3, 5, 'seven']
2 print(list)
3 list.append(9)
4 print(list)
```



educative

▶

📄

↶

⚙️

📋

×

Output
0.160s

```
[1, 3, 5, 'seven']
[1, 3, 5, 'seven', 9]
```

The `insert()` function

Inserts element to the list. Use it like `list.insert(index, value)`. It works in $O(n)$ time. Try it out yourself!

The following use of `list.insert(0,2)` inserts the element 2 at index 0.

```
1 list = [1, 3, 5, 'seven']
2 list.insert(0, 2)
3 print(list)
```

▶

📄

↶

⌕

×

Output
0.297s

```
[2, 1, 3, 5, 'seven']
```

The `remove()` function

Removes the given element at a given index. Use it like `list.remove(element)`. It works in $O(n)$ time. If the element does not exist, you will get a runtime error as in the following example.

```
1 list = [1, 3, 5, 'seven']
2 print(list)
3 list.remove('seven')
4 print(list)
5 list.remove(0)
6 print(list)
```

▶

📄

↶

⌕

×

Output
0.234s


```
[1, 3, 5, 'seven']
[1, 3, 5]
```

Traceback (most recent call last):
 File "main.py", line 5, in <module>
 list.remove(0)
ValueError: list.remove(x): x not in list

The `pop()` function

Removes the element at given index. If no index is given, then it removes the last element. So `list.pop()` would remove the last element. This works in $O(1)$. `list.pop(2)` would remove the element with index 2, i.e., 5 in this case. Also, popping the k th intermediate element takes $O(k)$ time where $k < n$.

```
1 list = [1, 3, 5, 'seven']
2 print(list)
3 list.pop(2)
4 print(list)
```

Output 0.233s

```
[1, 3, 5, 'seven']
[1, 3, 'seven']
```

The `reverse()` function

This function reverses the list. It can be used as `list.reverse()` and takes $O(n)$ time

```
1 list = [1, 3, 5, 'seven']
2 print(list)
3 list.reverse()
4 print(list)
```

Output 0.254s

```
[1, 3, 5, 'seven']
['seven', 5, 3, 1]
```

What is Slicing?

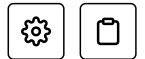
Accessing and modifying several elements from objects such as lists/tuples/strings requires using a for loop in most languages. However, in Python, you can use square brackets and a colon to define a range of elements within a list that you want to access or ‘slice’.

```
list[start:end]
```

Here start and end indicate the starting and ending index of a list that is desired to be accessed.

You can print these values, reinitialize them, execute mathematical functions on them, and a plethora of other operations. Let's look at a few examples.

Slice Notation Examples



The following examples use slicing to perform various operations on lists.

Example 1: Indexing elements of a list

List elements can be indexed and printed as in the following code example:

```
1 list = list(range(10))
2 print(list) # 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
3 print(list[0:4]) # 0, 1, 2, 3
```

Output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3]
```

Also, note that it is not necessary to specify the last or the first index explicitly, you can simply leave the `end` or `start` index blank respectively.

```
list[start:] means all numbers greater than start upto the range
list[:end] means all numbers less than end upto the range
list[:] means all numbers within the range
```

Study the following example for more details.

```
1 list = list(range(10))
2 print(list[3:]) # 3, 4, 5, 6, 7, 8, 9
3 print(list[:3]) # 0 1 2
4 print(list[:]) # 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

Output

```
[3, 4, 5, 6, 7, 8, 9]
[0, 1, 2]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Example 2: Stepped Indexing

You can also index elements in steps like in a for loop. So a C++ for loop like the following:

```
for(int i = 0; i < 10; i+=2)
{
    cout << arr[i] << endl; // prints the array with index i
}
```

The variable `i` is incremented by two at every iteration and the array `arr` is indexed accordingly to be displayed. You can do the same in Python very concisely with the notation:

```
list[start:stop:step]
```

Here `step` specifies the increment in the list indices and can also be negative. For example,

```
1 list = list(range(10)) # define a range of values 0
2 print(list[0:9:2]) # 0, 2, 4, 6, 8
3 print(list[9:0:-2]) # 9, 7, 5, 3, 1
```



Output

0.216s

```
[0, 2, 4, 6, 8]
[9, 7, 5, 3, 1]
```

line 2 prints every second value of the list starting from the beginning

line 3 prints every second value of the list starting from the end

Example 3: Initializing list elements #

You can add/modify the contents of a list by specifying a range of elements that you want to update and setting it to the new value:

```
arr[start:end] = [list, of, New, values]
```

```
1 x = list(range(5))
2 print(x) # 0, 1, 2, 3, 4
3 x[1:4] = [45, 21, 83]
4 print(x) # 0, 45, 21, 83, 4
```



Output

0.275s

```
[0, 1, 2, 3, 4]
[0, 45, 21, 83, 4]
```

The `1:4` in the square brackets means that the elements at positions 1, 2 and 3, up to but not including position 4, would be set to new values, i.e., `[45, 21, 83]`.



Note that in Python, range counts up to the second index given but never hits the index itself. So in this case, the 4th index, i.e., the element **4** is not replaced.

Example 4: Deleting elements from a list

The `del` keyword is used to delete elements from a list. In the following example, all the elements at even-numbered indices are deleted.

```
1 list = list(range(10))
2 print(list) # 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
3 del list[::2]
4 print(list) # 1, 3, 5, 7, 9
```



Output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 3, 5, 7, 9]
```

0.322s

Line 3 uses the `del` function. Here, the empty start and end indices refer to 0 and length of the list by default, whereas 2 is the step size.

Example 5: Negative Indexing

We can use negative numbers to begin indexing the list elements from the end. For example, to access the fifth-last element of a list, we use:

```
list[-5]
```

```
1 list = list(range(10))
2 print(list)
3 print(list[4:-1]) # 4, 5, 6, 7, 8
```



Output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[4, 5, 6, 7, 8]
```

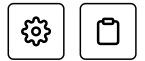
0.411s

The example above displays `list` from the 4th index to the second last index



educative

Example 6: Slicing in Strings #



We can also use slicing techniques on strings since strings *are* lists of characters! (well, technically, they're *arrays* of characters, but we'll get to that in a bit!) For example, the string "somehow" can be broken down into two strings like:

```
1 my_string = "somehow"
2 string1 = my_string[:4]
3 string2 = my_string[4:]
4 print(string1, string2)
```



Output

some how

0.283s

Now that we have studied the basics of list manipulation, let's move on to Python arrays in the next lesson!

Back

Next

Complexity Quiz: Test your understand...

Arrays

Mark as Completed



Report an
Issue



Ask a Question

(https://discuss.educative.io/tag/lists__introduction-to-lists__data-structures-for-coding-interviews-in-python)