

In Python, lists are used to store multiple data at once. For example,

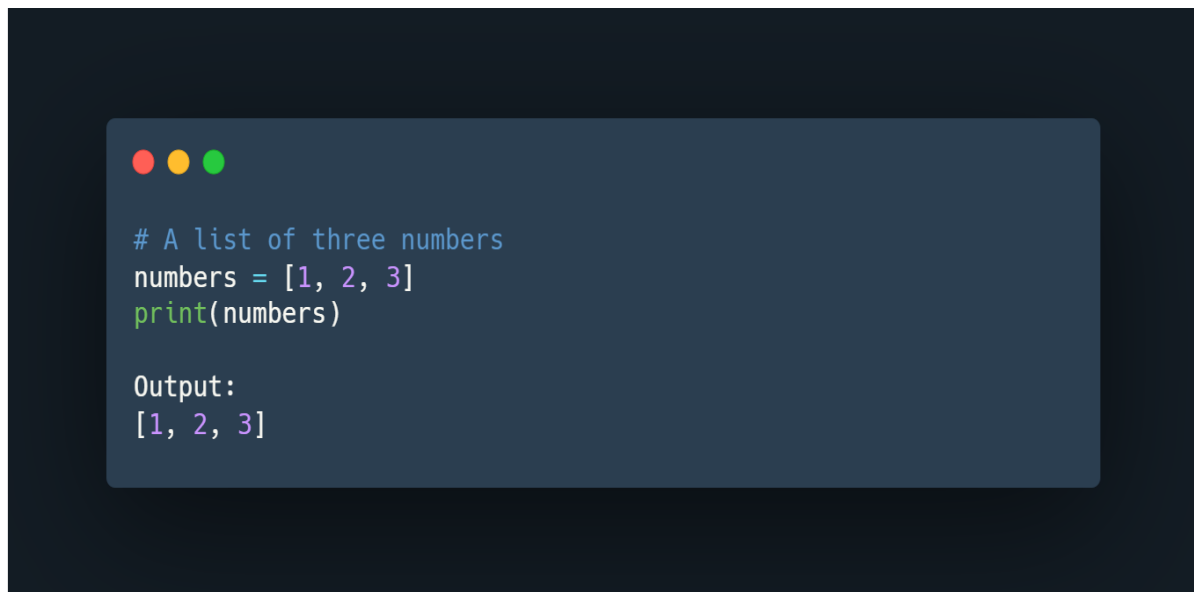
Suppose we need to record the ages of **5** students. Instead of creating **5** separate variables, we can simply create a list:



Lists Elements

## Create a Python List

A list is created in Python by placing items inside [], separated by commas. For example,



```
# A list of three numbers
numbers = [1, 2, 3]
print(numbers)

Output:
[1, 2, 3]
```

Here, we have created a list named numbers with **3** integer items.

A list can have any number of items and they may be of different types (integer, float, string, etc.). For example,

```
# empty list

my_list = []

# list with mixed data types

my_list = [1, "Hello", 3.4] # int, str, float
```

## Access Python List Elements

In Python, each item in a list is associated with a number. The number is known as a list index.

We can access elements of an array using the index number (**0, 1, 2 ...**). For example,

```
languages = ["Python", "Swift", "C++"]

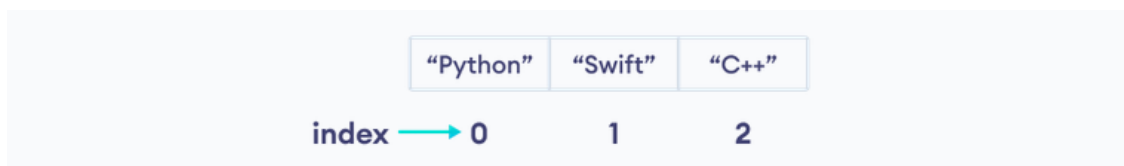
# access item at index 0

print(languages[0]) # Python

# access item at index 2

print(languages[2]) # C++
```

In the above example, we have created a list named languages.



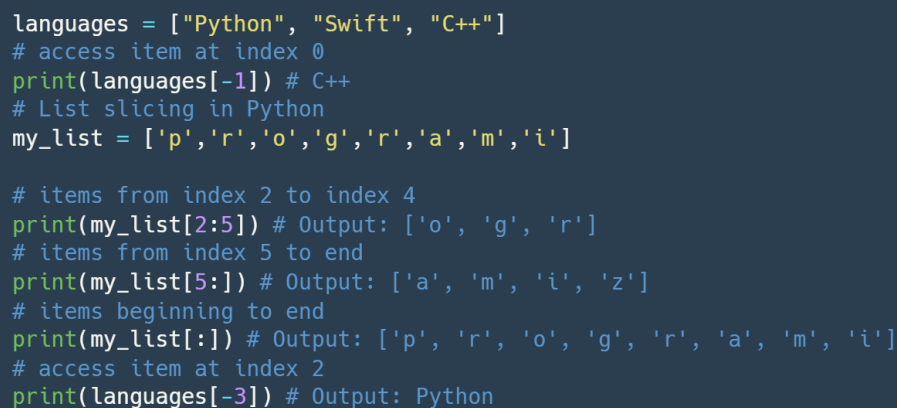
List Indexing in Python

Here, we can see each list item is associated with the index number. And, we have used the index number to access the items.

**Note:** The list index always starts with **0**. Hence, the first element of a list is present at index **0**, not **1**.

## Slicing of a Python List

In Python, it is possible to access a section of items from the list using the slicing operator :, not just a single item. For example,

A screenshot of a code editor with a dark background and light-colored text. The code defines a list 'languages' with elements 'Python', 'Swift', and 'C++'. It then performs several slicing operations on another list 'my\_list' and on 'languages'. Comments explain the purpose of each line and the expected output.

```
languages = ["Python", "Swift", "C++"]
# access item at index 0
print(languages[-1]) # C++
# List slicing in Python
my_list = ['p','r','o','g','r','a','m','i']

# items from index 2 to index 4
print(my_list[2:5]) # Output: ['o', 'g', 'r']
# items from index 5 to end
print(my_list[5:]) # Output: ['a', 'm', 'i', 'z']
# items beginning to end
print(my_list[:]) # Output: ['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i']
# access item at index 2
print(languages[-3]) # Output: Python
```

Here,

- `my_list[2:5]` returns a list with items from index **2** to index **4**.
- `my_list[5:]` returns a list with items from index **1** to the end.
- `my_list[:]` returns all list items

**Note:** When we slice lists, the start index is inclusive but the end index is exclusive.

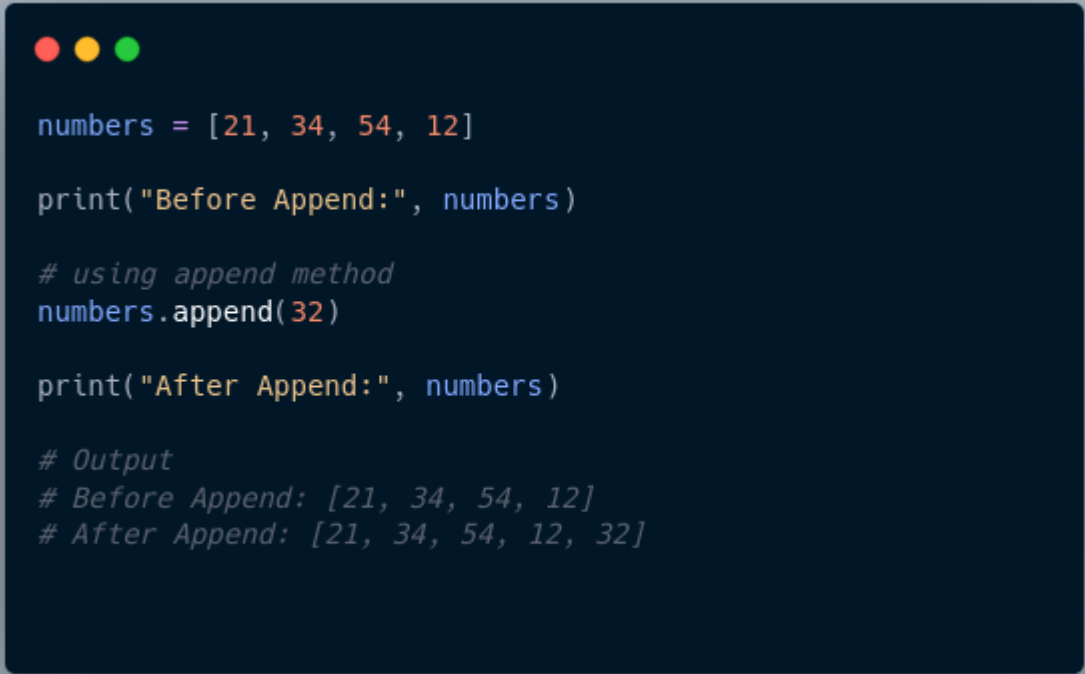
# Add Elements to a Python List

Python List provides different methods to add items to a list.

## 1. Using `append()`

The `append()` method adds an item at the end of the list.

For example,



```
numbers = [21, 34, 54, 12]

print("Before Append:", numbers)

# using append method
numbers.append(32)

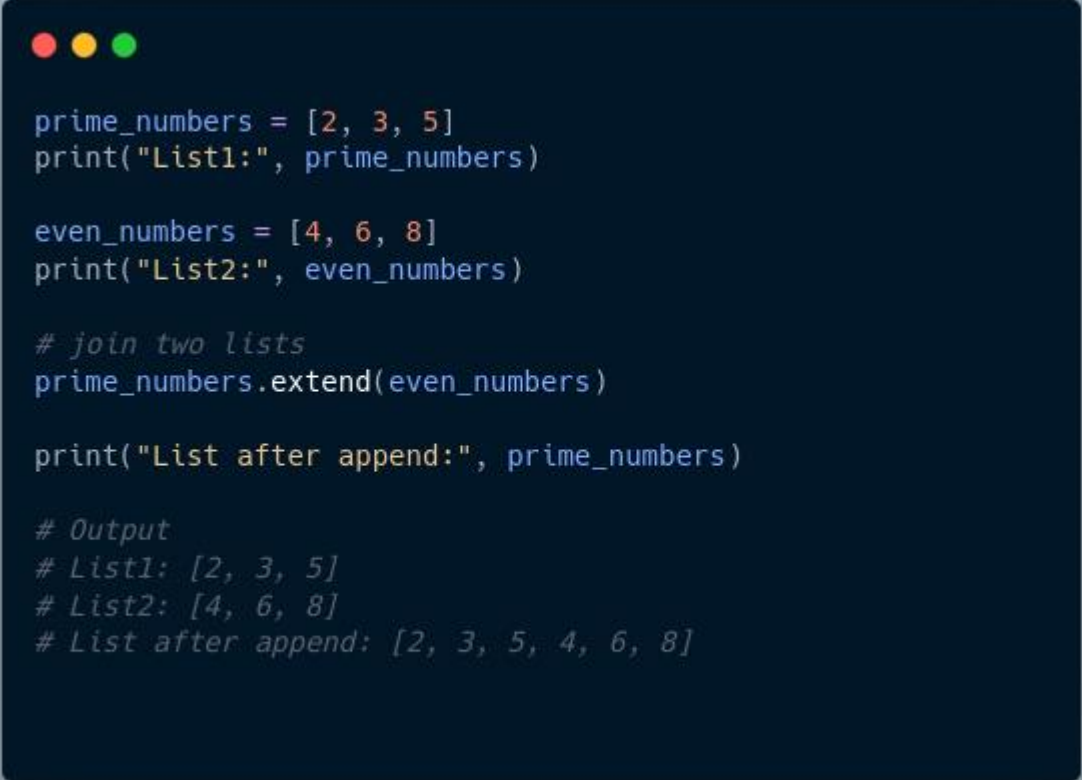
print("After Append:", numbers)

# Output
# Before Append: [21, 34, 54, 12]
# After Append: [21, 34, 54, 12, 32]
```

Here, `append()` adds **32** at the end of the array.

## 2. Using `extend()`

We use the `extend()` method to add all items of one list to another. For example,



```
prime_numbers = [2, 3, 5]
print("List1:", prime_numbers)

even_numbers = [4, 6, 8]
print("List2:", even_numbers)

# join two lists
prime_numbers.extend(even_numbers)

print("List after append:", prime_numbers)

# Output
# List1: [2, 3, 5]
# List2: [4, 6, 8]
# List after append: [2, 3, 5, 4, 6, 8]
```

In the above example, we have two lists named `prime_numbers` and `even_numbers`. Notice the statement,

## Change List Items

Python lists are mutable. Meaning lists are changeable. And, we can change items of a list by assigning new values using `=` operator. For example,



```
languages = ['Python', 'Swift', 'C++']  
  
# changing the third item to 'C'  
  
languages[2] = 'C'  
  
print(languages) # ['Python', 'Swift', 'C']
```

## Remove an Item From a List

### 1. Using del()

In Python, we can use the del statement to remove one or more items from a list. For example,



```
languages = ['Python', 'Swift', 'C++', 'C', 'Java', 'Rust',  
'R']
```

```
# deleting the second item
```

```
del languages[1]
```

```
print(languages) # ['Python', 'C++', 'C', 'Java', 'Rust',  
'R']
```

```
# deleting the last item
```

```
del languages[-1]
```

```
print(languages) # ['Python', 'C++', 'C', 'Java', 'Rust']
```

```
# delete first two items
```


```
del languages[0 : 2] # ['C', 'Java', 'Rust']
```

```
print(languages)
```

## 2. Using remove()

We can also use the remove() method to delete a list item.

For example:



```
languages = ['Python', 'Swift', 'C++', 'C', 'Java', 'Rust',  
            'R']  
  
# remove 'Python' from the list  
languages.remove('Python')  
  
print(languages) # ['Swift', 'C++', 'C', 'Java', 'Rust',  
                  'R']
```

Here, `languages.remove('Python')` removes 'Python' from the languages list.

## Python List Methods


Python has many useful list methods that make it really easy to work with lists.



Method	Description
<code>append()</code>	add an item to the end of the list
<code>extend()</code>	add items of lists and other iterables to the end of the list
<code>insert()</code>	inserts an item at the specified index
<code>remove()</code>	removes item present at the given index
<code>pop()</code>	returns and removes item present at the given index
<code>clear()</code>	removes all items from the list
<code>index()</code>	returns the index of the first matched item
<code>count()</code>	returns the count of the specified item in the list
<code>sort()</code>	sort the list in ascending/descending order
<code>reverse()</code>	reverses the item of the list
<code>copy()</code>	returns the shallow copy of the list

## Iterating through a List

We can use the for loop to iterate over the elements of a list. For example,



```
languages = ['Python', 'Swift', 'C++']

# iterating through the list
for language in languages:
    print(language)


# Output
# Python
# Swift
# C++
```

## Python List Comprehension

List comprehension is a concise and elegant way to create lists.

A list comprehension consists of an expression followed by the for statement inside square brackets.

Here is an example to make a list with each item being increasing by power of 2.

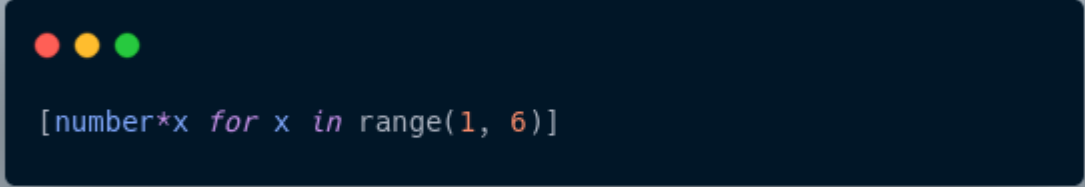


```
numbers = [number*number for number in range(1, 6)]

print(numbers)

# Output: [1, 4, 9, 16, 25]
```

In the above example, we have used the list comprehension to make a list with each item being increased by power of **2**. Notice the code,



```
[number*x for x in range(1, 6)]
```

## Tuples

A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

### Creating a Tuple

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. The parentheses are optional, however, it is a good practice to use them.

A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.).

### Create a Python Tuple With one Element

In Python, creating a tuple with one element is a bit tricky. Having one element within parentheses is not enough.

We can use the `type()` function to know which class a variable or a value belongs to.



```
var1 = ("hello")
print(type(var1)) # <class 'str'>

# Creating a tuple having one element
var2 = ("hello",)
print(type(var2)) # <class 'tuple'>

# Parentheses is optional
var3 = "hello",
print(type(var3)) # <class 'tuple'>
```

Here,

- ("hello") is a string so type() returns str as class of var1 i.e. <class 'str'>
- ("hello",) and "hello", both are tuples so type() returns tuple as class of var1 i.e. <class 'tuple'>
- 

## Access Python Tuple Elements

Like a list, each element of a tuple is represented by index numbers (**0, 1, ...**) where the first element is at index **0**.

We use the index number to access tuple elements. For example,

### 1. Indexing

We can use the index operator `[]` to access an item in a tuple, where the index starts from 0.

So, a tuple having **6** elements will have indices from **0** to **5**. Trying to access an index outside of the tuple index range( **6,7,...** in this example) will raise an `IndexError`.

The index must be an integer, so we cannot use float or other types. This will result in `TypeError`.

Likewise, nested tuples are accessed using nested indexing, as shown in the example below.



```
# accessing tuple elements using indexing
letters = ("p", "r", "o", "g", "r", "a", "m", "i", "z")

print(letters[0])    # prints "p"
print(letters[5])    # prints "a"
```

In the above example,

- letters[0] - accesses the first element
- letters[5] - accesses the sixth element

## 2. Negative Indexing

Python allows negative indexing for its sequences.

The index of **-1** refers to the last item, **-2** to the second last item and so on. For example,



```
# accessing tuple elements using negative indexing
letters = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

print(letters[-1])   # prints 'z'
print(letters[-3])   # prints 'm'
```

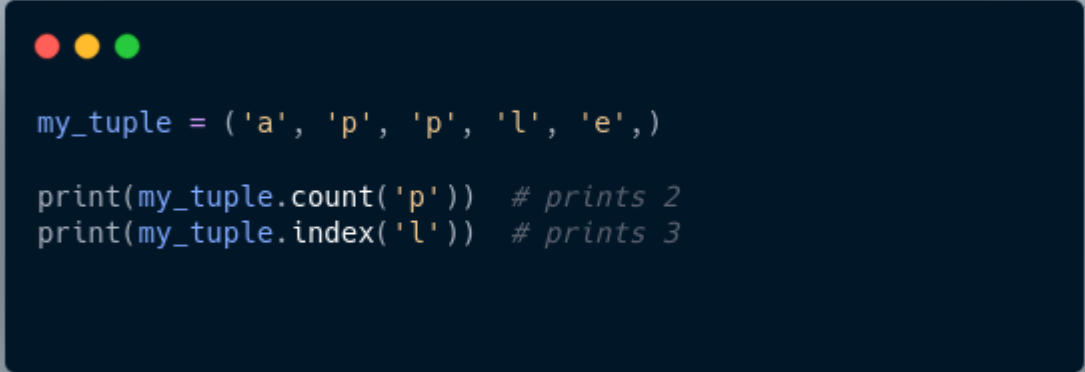
In the above example,

- letters[-1] - access last element
- letters[-3] - access third last element

# Python Tuple Methods

In Python, methods that add items or remove items are not available with tuples. Only the following two methods are available.

Some examples of Python tuple methods:



```
my_tuple = ('a', 'p', 'p', 'l', 'e',)

print(my_tuple.count('p')) # prints 2
print(my_tuple.index('l')) # prints 3
```

Here,

- `my_tuple.count('p')` - counts total number of 'p' in `my_tuple`
- `my_tuple.index('l')` - returns the first occurrence of 'l' in `my_tuple`

## More Resources:

1. <https://www.knowledgehut.com/tutorials/python-tutorial/python-lists-tuples>
2. <https://www.freecodecamp.org/news/python-tuple-vs-list-what-is-the-difference/>
3. <https://builtin.com/software-engineering-perspectives/python-tuples-vs-lists>
4. <https://realpython.com/python-lists-tuples/>