VNUHCM - University of Science
fit@hcmus

Fundamentals of Programming for Artificial Intelligence

Session 03 Advanced Function & List

Instructors:
Dr. Lê Thanh Tùng
Dr. Nguyễn Tiến Huy

Content

1. Advanced Function

Functions as parameters

- In Python, you can pass a function as an argument to another function. This is known as a higher-order function
- In other words, a function is called a higher-order function if it contains other functions as parameters or returns functions
- Few examples of higher-order functions in Python are `map()`, `filter()`, `sorted()`, `reduce()`

Functions as parameters

fit@hcmus

```python
def apply_function(func, x):
    return func(x)

def square(x):
    return x ** 2

result = apply_function(square, 3)
print(result)
```

Functions as parameters

- Passing functions as arguments can be useful for writing more flexible and reusable code

```python
def add(x, y): return x + y

def mul(x, y): return x * y

def subtract(x, y): return x - y

a, b = map(int, input().split())
for func in [add, subtract, mul]:
    print(func(a, b))
```

map()

- The map() function executes a given function to each element of an iterable (such as lists, tuples, etc.)

```
# returns the square of a number
def square(number):
  return number * number

# apply square() to each item of the numbers list
numbers = [1,2,3,4]
squared_numbers = map(square, numbers)

print(list(squared_numbers)) # Output: [1,4,9,16]
```

map()

- A lambda function is a small anonymous function
- A lambda function can take any number of arguments, but can only have one expression

Syntax

```
lambda arguments : expression
```

```
x = lambda a, b : a * b
print(x(5, 6))
```

lambda()

- A lambda function is a small anonymous function
- A lambda function can take any number of arguments, but can only have one expression

Syntax

lambda *arguments* : *expression*

```
x = lambda a, b : a * b
print(x(5, 6))
```

filter()

- The filter() function returns an iterator where the items are filtered through a function to test if the item is accepted or not

Syntax

```
filter(function, iterable)
```

```
def myFunc(x): return x < 18

adults = filter(myFunc, [5, 12, 17, 18, 24, 32])

print(list(adults)) # [5, 12, 17]
```

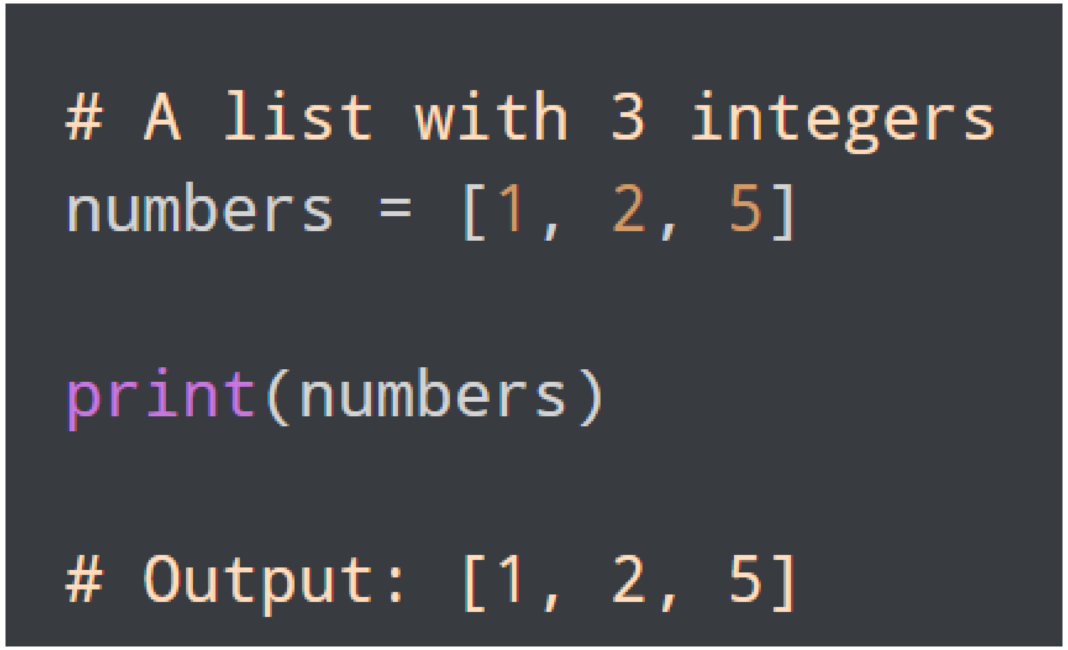
2. List

List

- List:
  - is used to store multiple items in a single variable
  - is just like dynamically sized arrays
  - is a collection of things
  - List items are ordered, changeable/mutable, and allow duplicate values

List

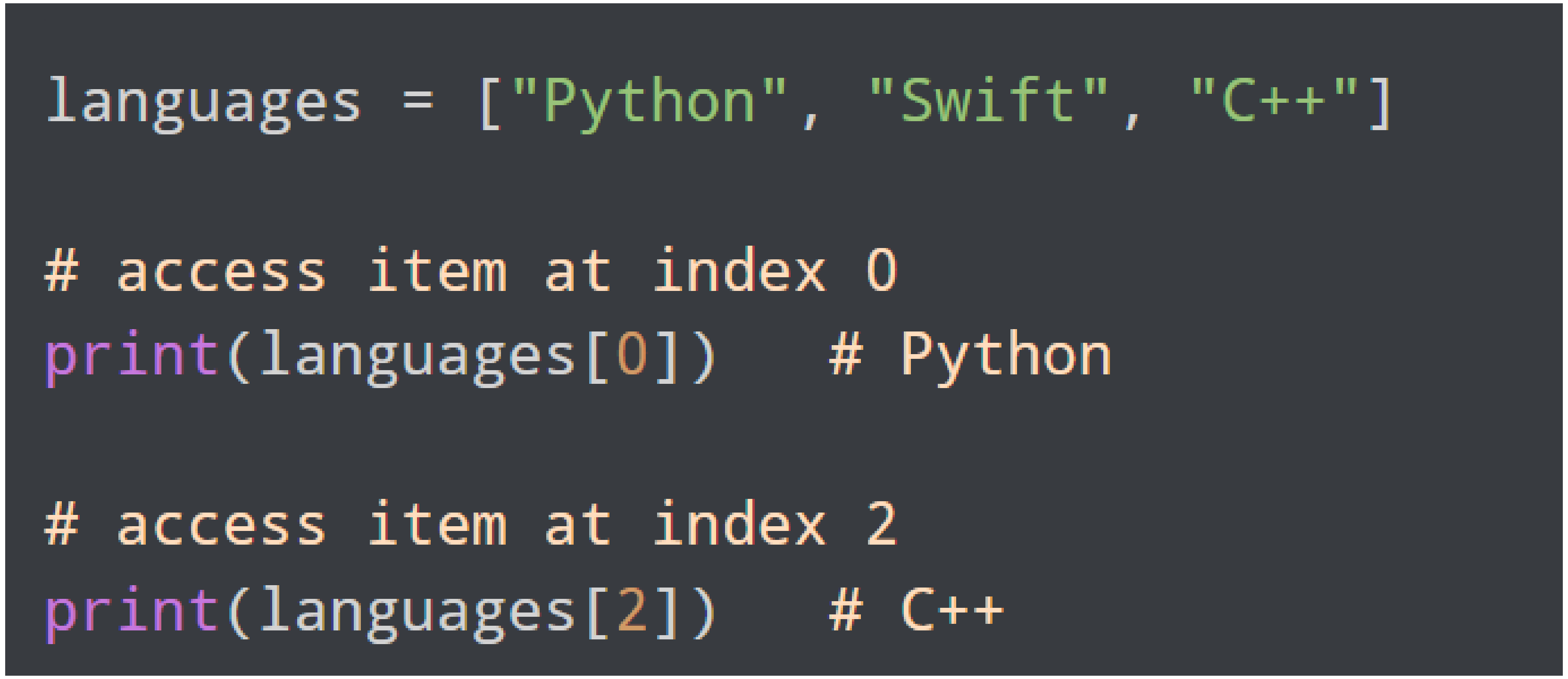
- Declare a list: A list is created in Python by placing items inside [], separated by commas



```
# A list with 3 integers
numbers = [1, 2, 5]

print(numbers)

# Output: [1, 2, 5]
```

List

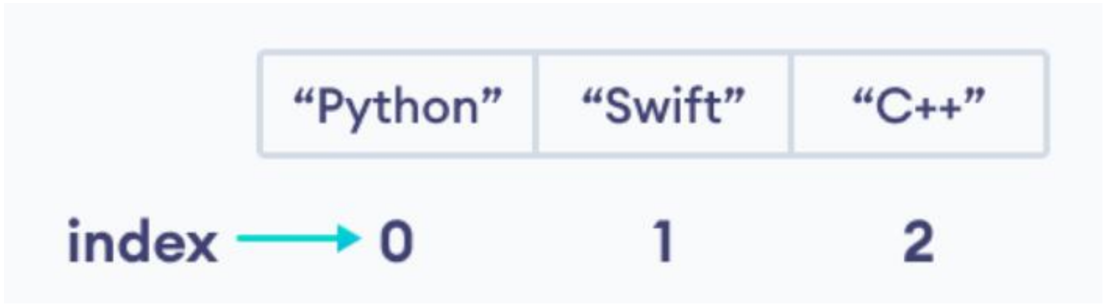
- Access an item in list
  - Via item index

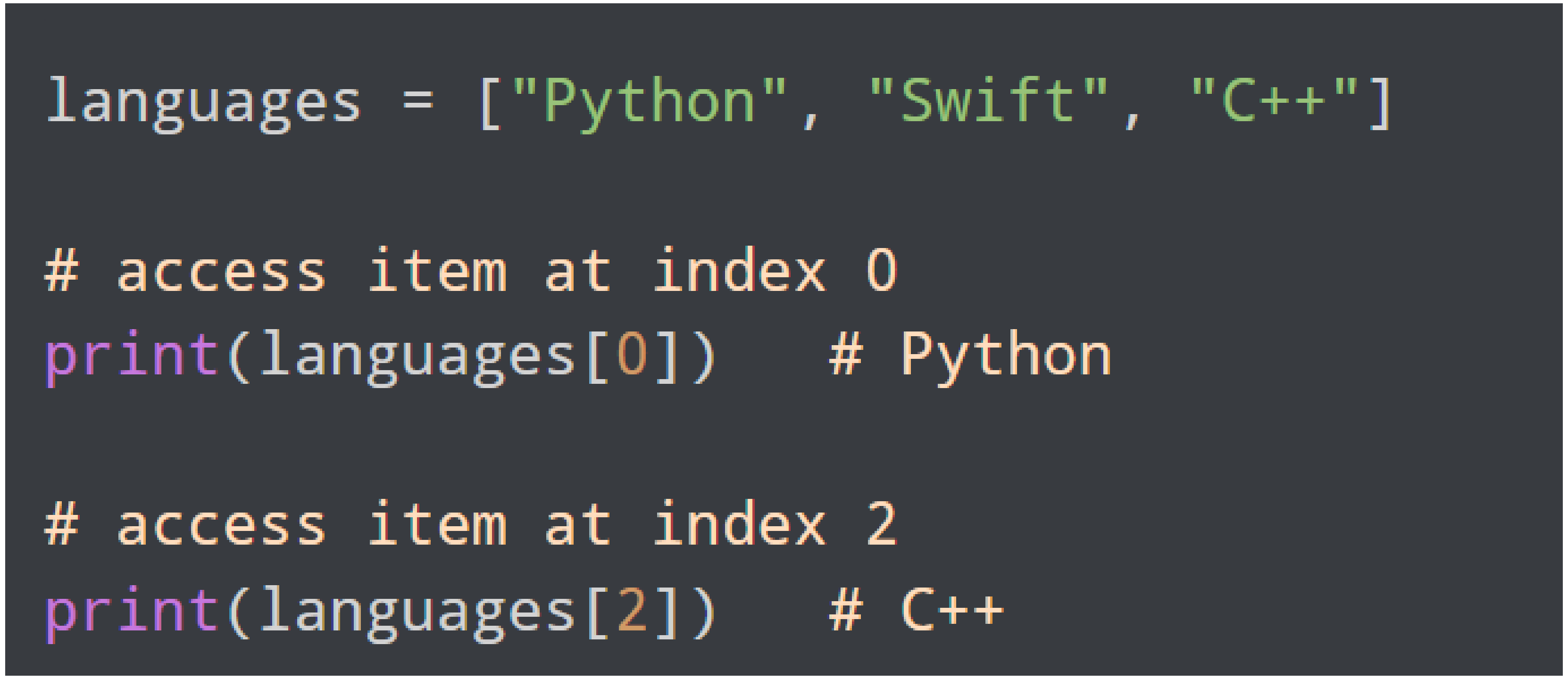


```
languages = ["Python", "Swift", "C++"]

# access item at index 0
print(languages[0])   # Python

# access item at index 2
print(languages[2])   # C++
```

List

- Access an item in list
  - Via item index

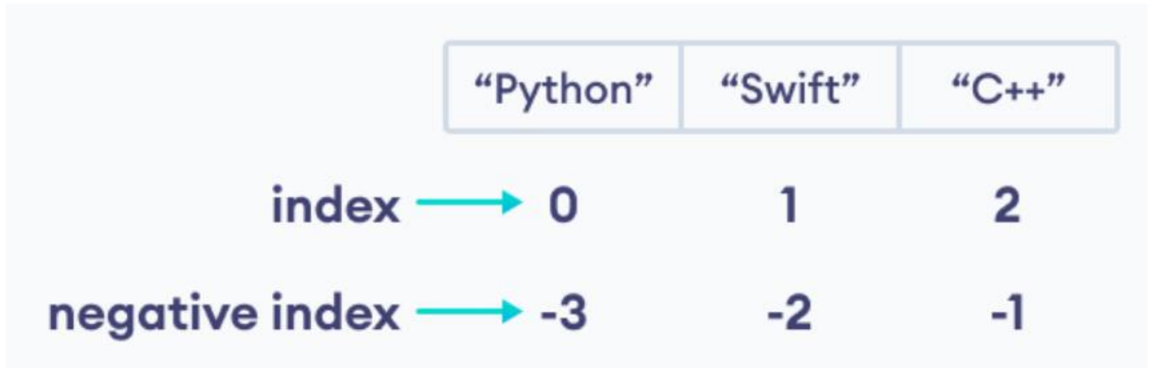


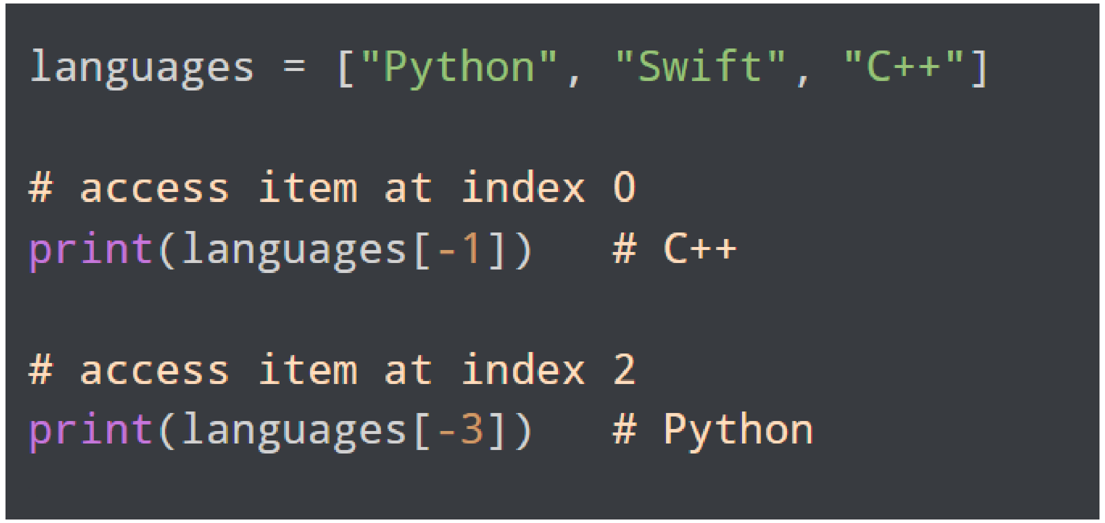


```
languages = ["Python", "Swift", "C++"]

# access item at index 0
print(languages[0])   # Python

# access item at index 2
print(languages[2])   # C++
```

Negative Indexing (Python)

- Python allows negative indexing for its sequences.
- The negative index is in reverse way

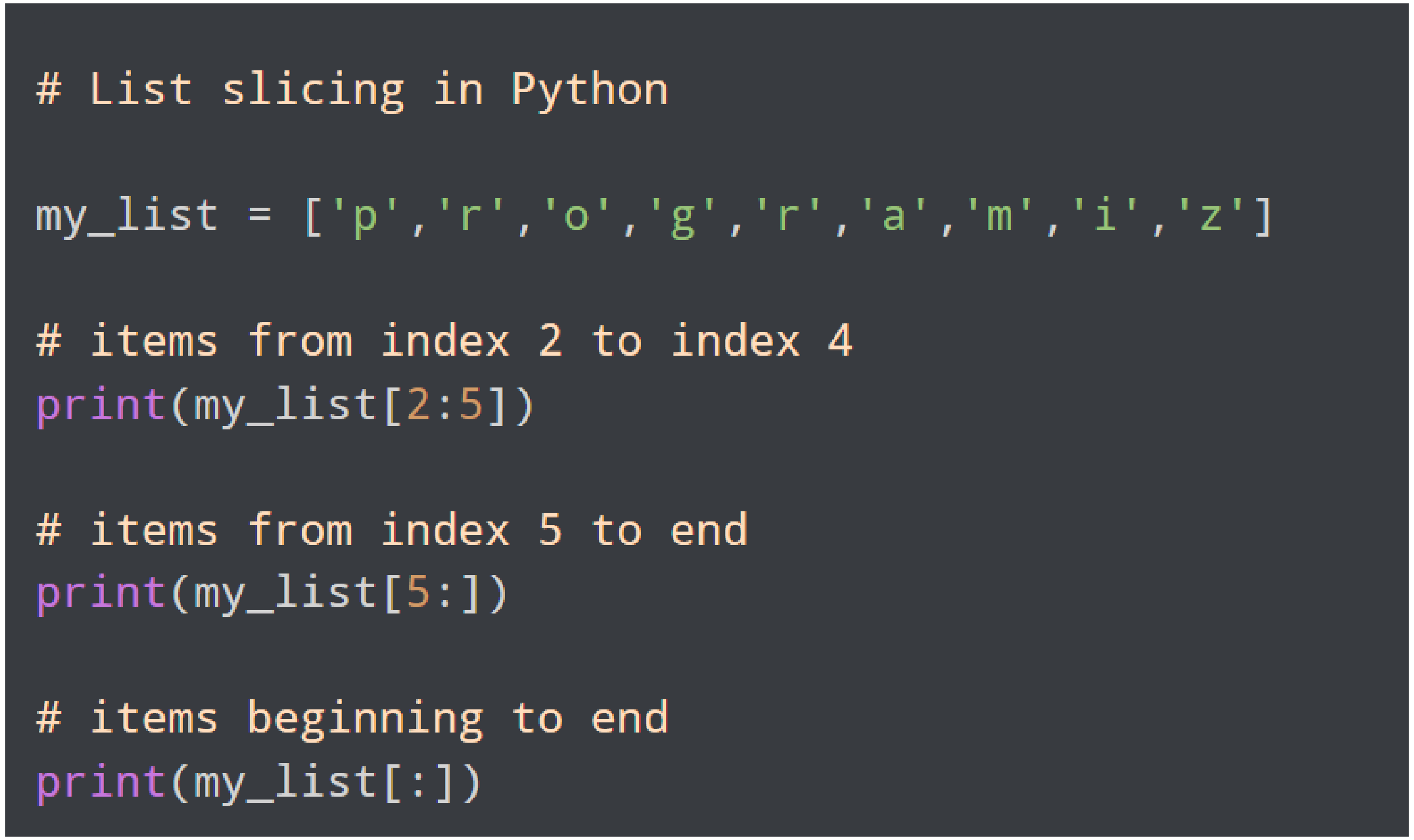




```python
languages = ["Python", "Swift", "C++"]

# access item at index 0
print(languages[-1])   # C++

# access item at index 2
print(languages[-3])   # Python
```

Slicing of a Python List

- A way to access a section of items from the list
- Slicing operator: `:`



```python
# List slicing in Python

my_list = ['p','r','o','g','r','a','m','i','z']

# items from index 2 to index 4
print(my_list[2:5])

# items from index 5 to end
print(my_list[5:])

# items beginning to end
print(my_list[:])
```

Slicing of a Python List

- General syntax:

```
Lst[ Initial : End : IndexJump ]
```

  - returns the portion of the list from index *Initial* to index *End*, at a step size *IndexJump*
  - Default:
    - *IndexJump = 1*
    - *Initial = 0*
    - *End = len(Lst)*

```
# Initialize list
Lst = [50, 70, 30, 20, 90, 10, 50]

# Display list
print(Lst[::])
```

```
[50, 70, 30, 20, 90, 10, 50]
```

Slicing of a Python List

- What is the output of the following code:

```python
# Initialize list
Lst = [50, 70, 30, 20, 90, 10, 50]

# Display list
print(Lst[::-1])
```

Slicing of a Python List

fit@hcmus

- What is the output of the following code:

```python
# Initialize list
Lst = [50, 70, 30, 20, 90, 10, 50]

# Display list
print(Lst[::-1])
```

```
[50, 10, 90, 20, 30, 70, 50]
```

- If some slicing expressions are made that do not make sense or are incomputable, then **empty lists** are generated

Slicing of a Python List

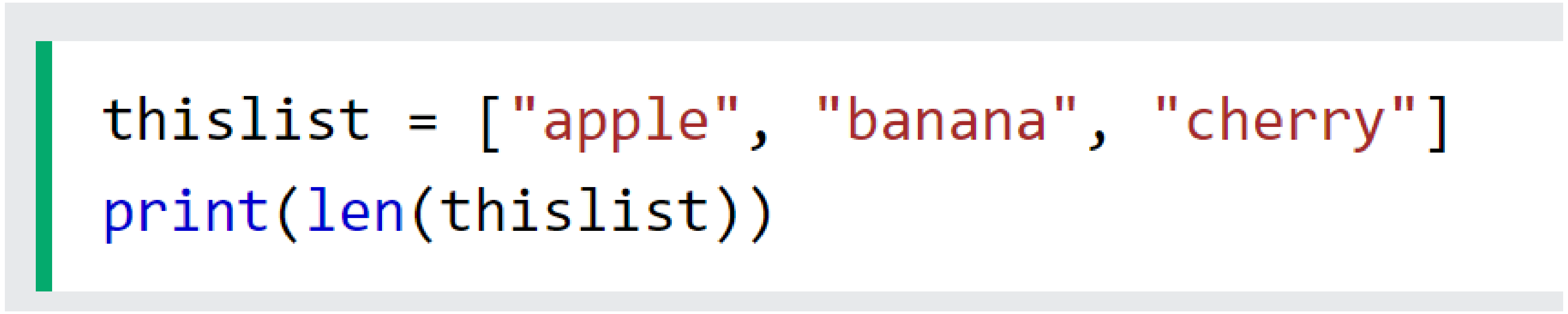
- What is the output of the following code:

```python
# Initialize list
Lst = [50, 70, 30, 20, 90, 10, 50]

# Display list
print(Lst[::-1])
```

```
[50, 10, 90, 20, 30, 70, 50]
```

Length of List

- len()



```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

Add element to List

- Use the pre-defined functions:
  - Append
  - Insert
  - Extend
  - List concatenation: by operator +

Append

- Add **element** in the end of list

Syntax

```
list.append(elmnt)
```

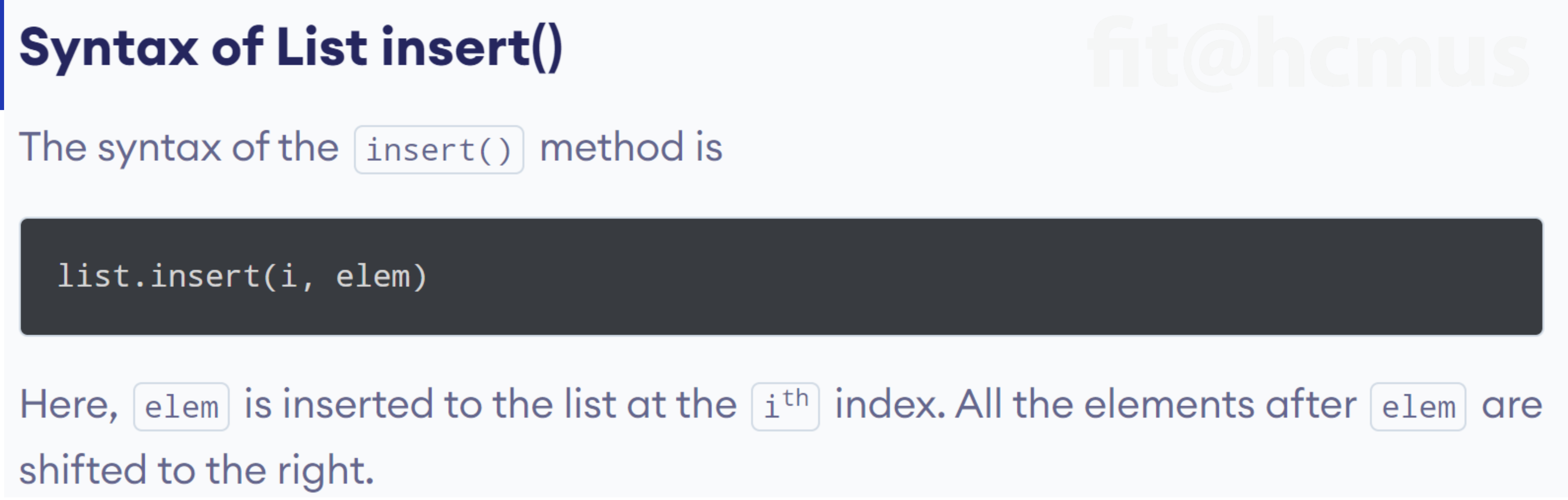
Parameter Values

Parameter	Description
elmnt	Required. An element of any type (string, number, object etc.)

```
a = ["apple", "banana", "cherry"]

b = ["Ford", "BMW", "Volvo"]

a.append(b)
```

```
['apple', 'banana', 'cherry', ["Ford", "BMW", "Volvo"]]
```

Insert

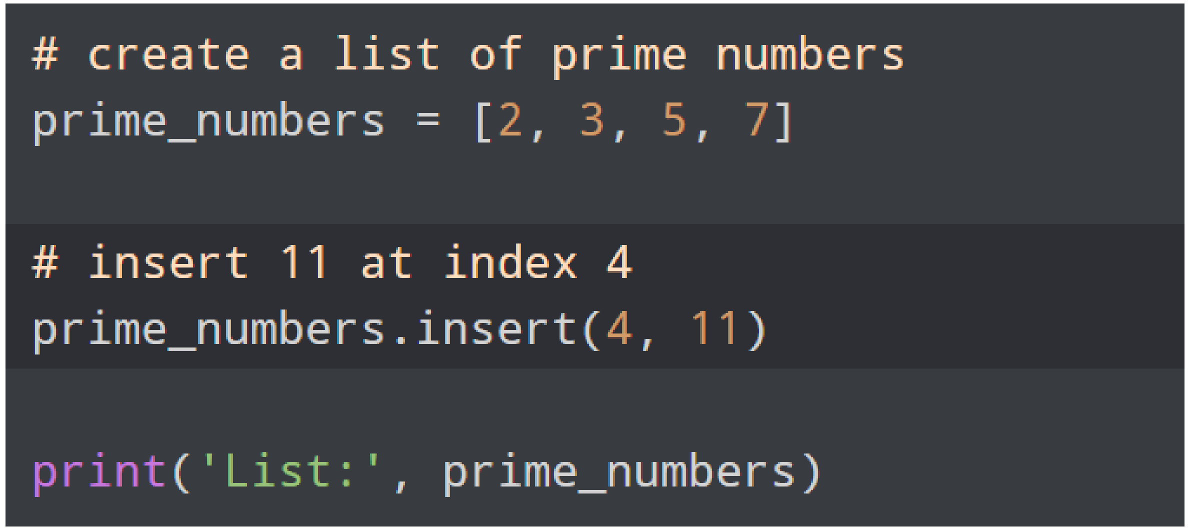


Syntax of List insert()

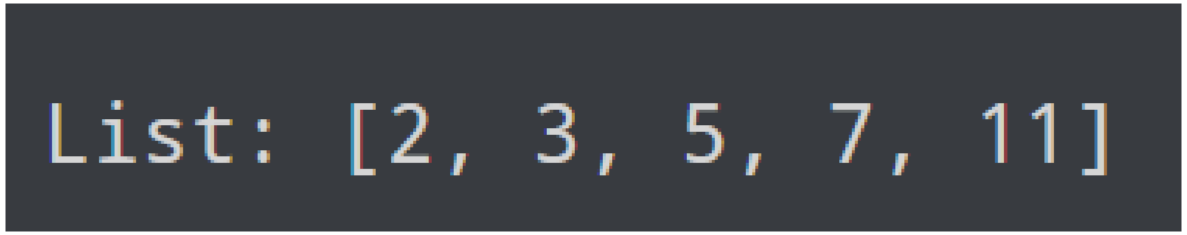
The syntax of the `insert()` method is

```
list.insert(i, elem)
```

Here, `elem` is inserted to the list at the `i^th` index. All the elements after `elem` are shifted to the right.



```
# create a list of prime numbers
prime_numbers = [2, 3, 5, 7]

# insert 11 at index 4
prime_numbers.insert(4, 11)

print('List:', prime_numbers)
```

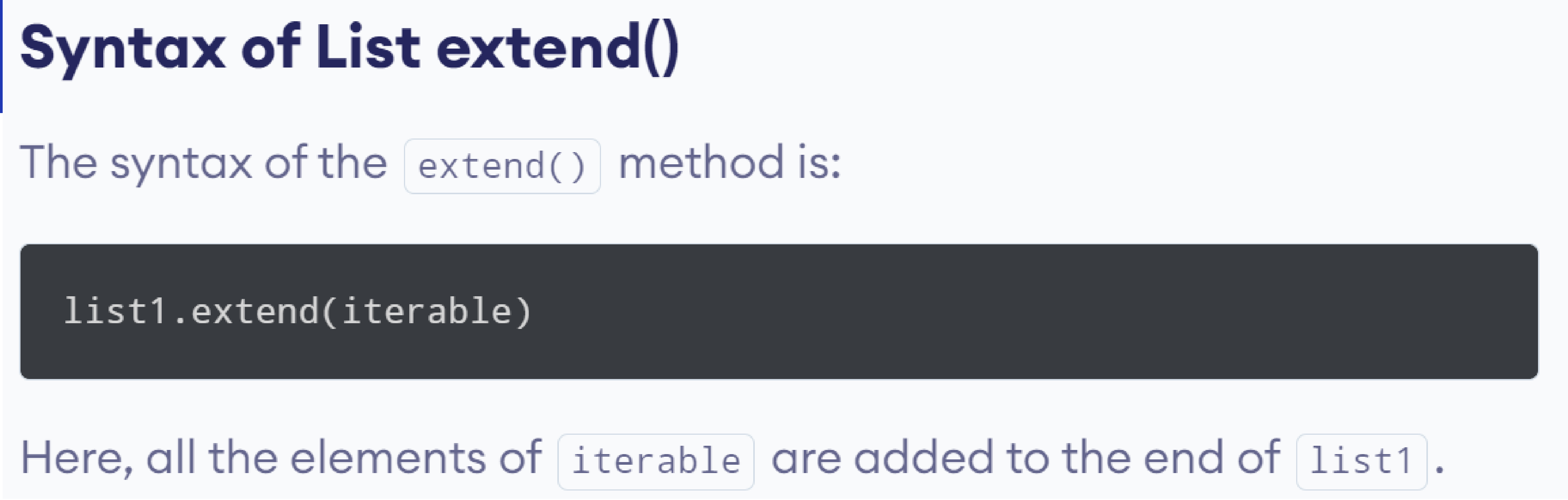


```
List: [2, 3, 5, 7, 11]
```

Extend

iterable:

- list
- tuple
- string

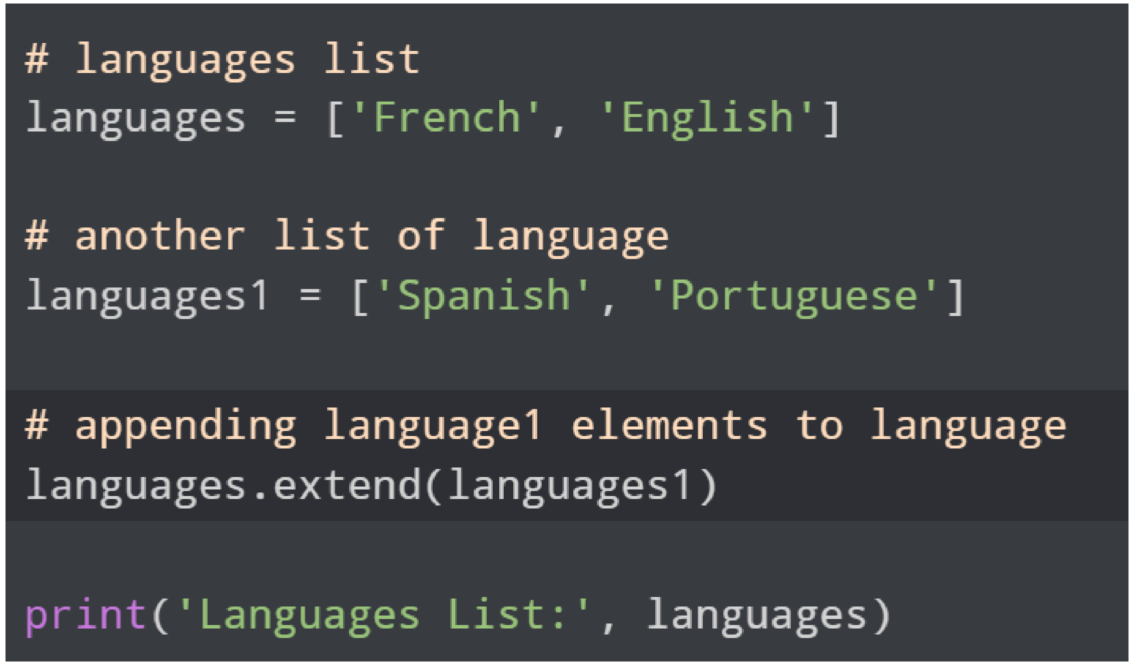


Syntax of List extend()

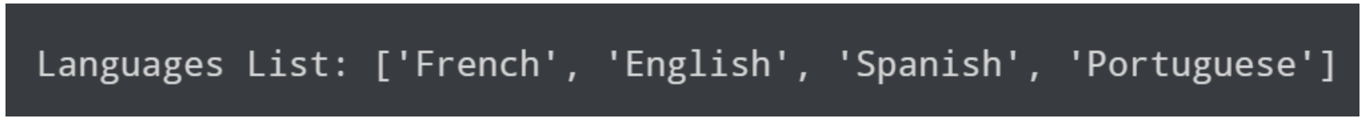
The syntax of the `extend()` method is:

```
list1.extend(iterable)
```

Here, all the elements of `iterable` are added to the end of `list1`.



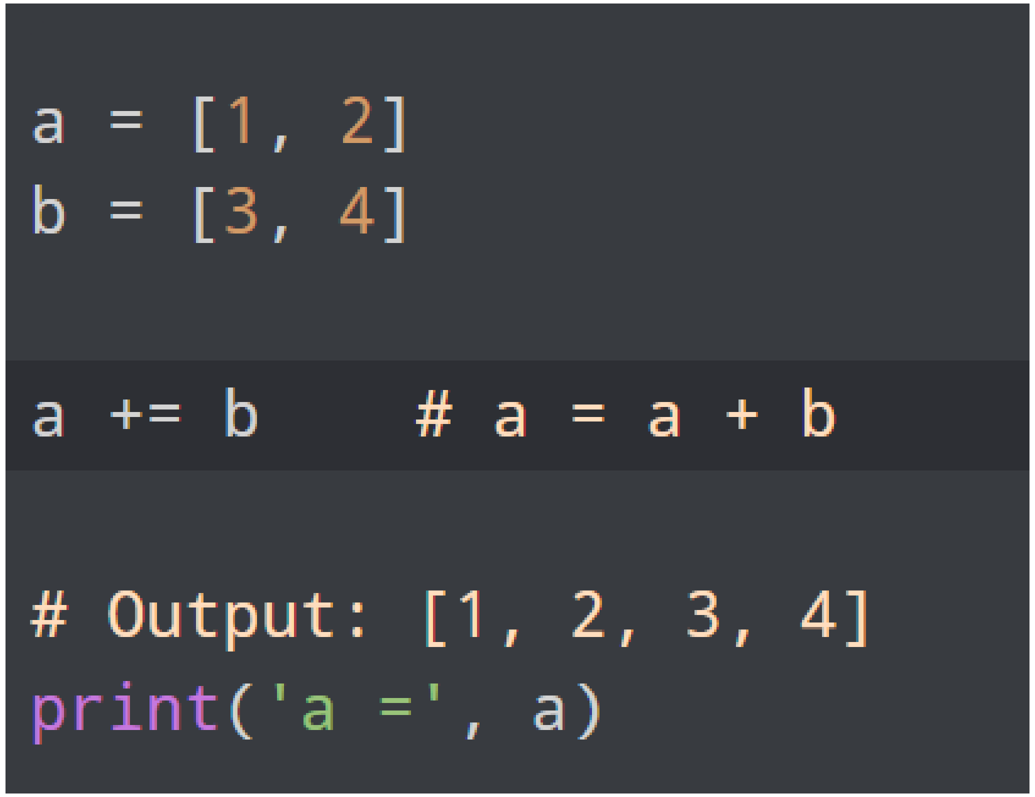
```python
# languages list
languages = ['French', 'English']

# another list of language
languages1 = ['Spanish', 'Portuguese']

# appending language1 elements to language
languages.extend(languages1)

print('Languages List:', languages)
```



```
Languages List: ['French', 'English', 'Spanish', 'Portuguese']
```

Operator +

- we can concatenate two lists by operators +
- a + b means a.extend(b)



```
a = [1, 2]
b = [3, 4]

a += b     # a = a + b

# Output: [1, 2, 3, 4]
print('a =', a)
```

Exercise

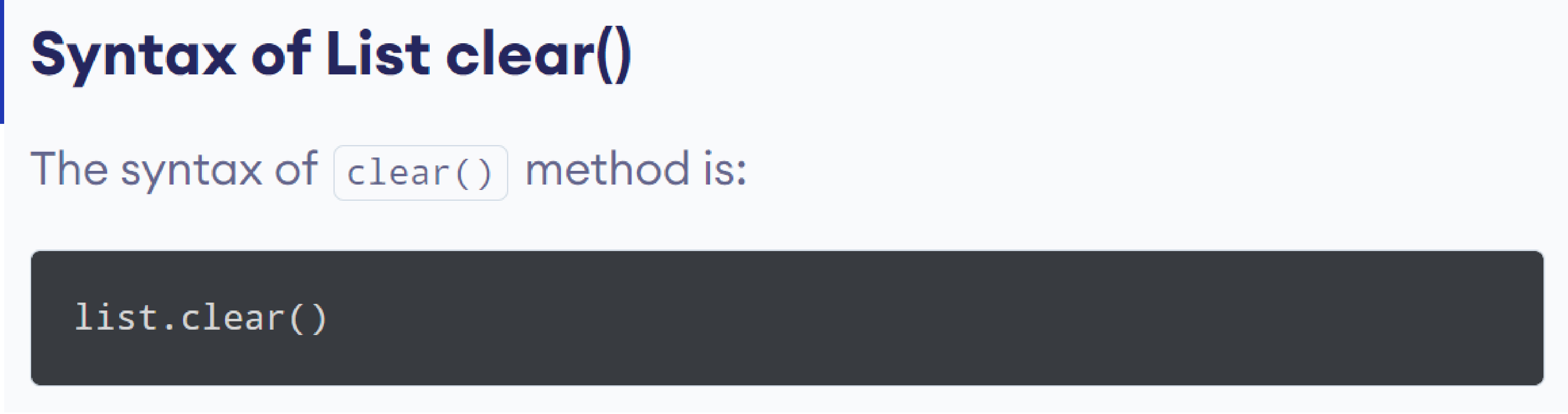
fit@hcmus

- If two continuous items are both odd, add their sum between them.
- For example, if we have the list [1, 3, 5, 3],
- the output is: [1, 4, 3, 8, 5, 8, 3]

Remove element from List

- Pre-defined functions:
  - clear()
  - pop()
  - remove()
  - del

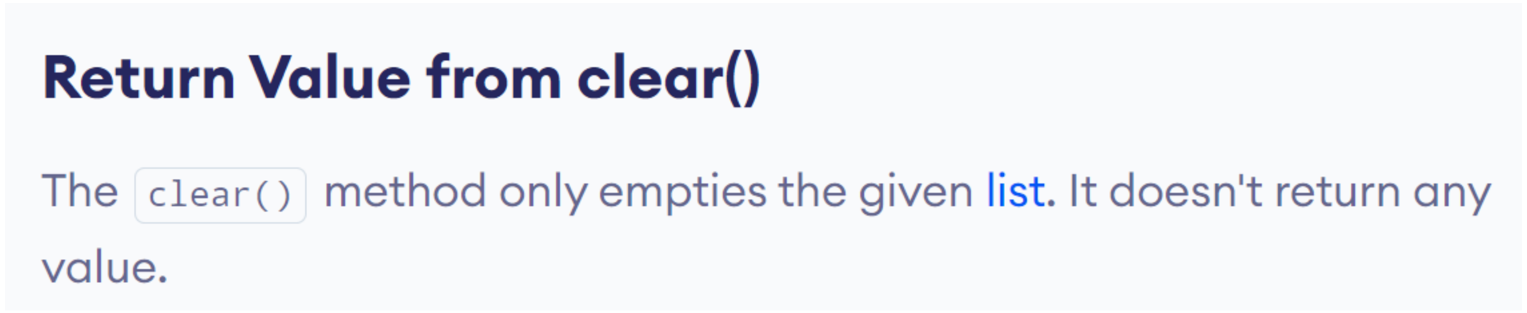
clear



Syntax of List clear()

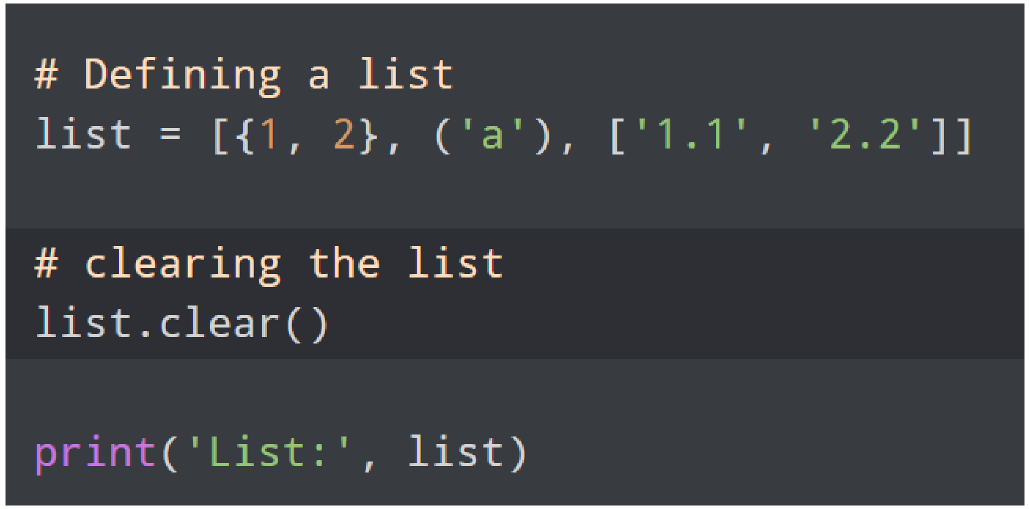
The syntax of `clear()` method is:

```
list.clear()
```



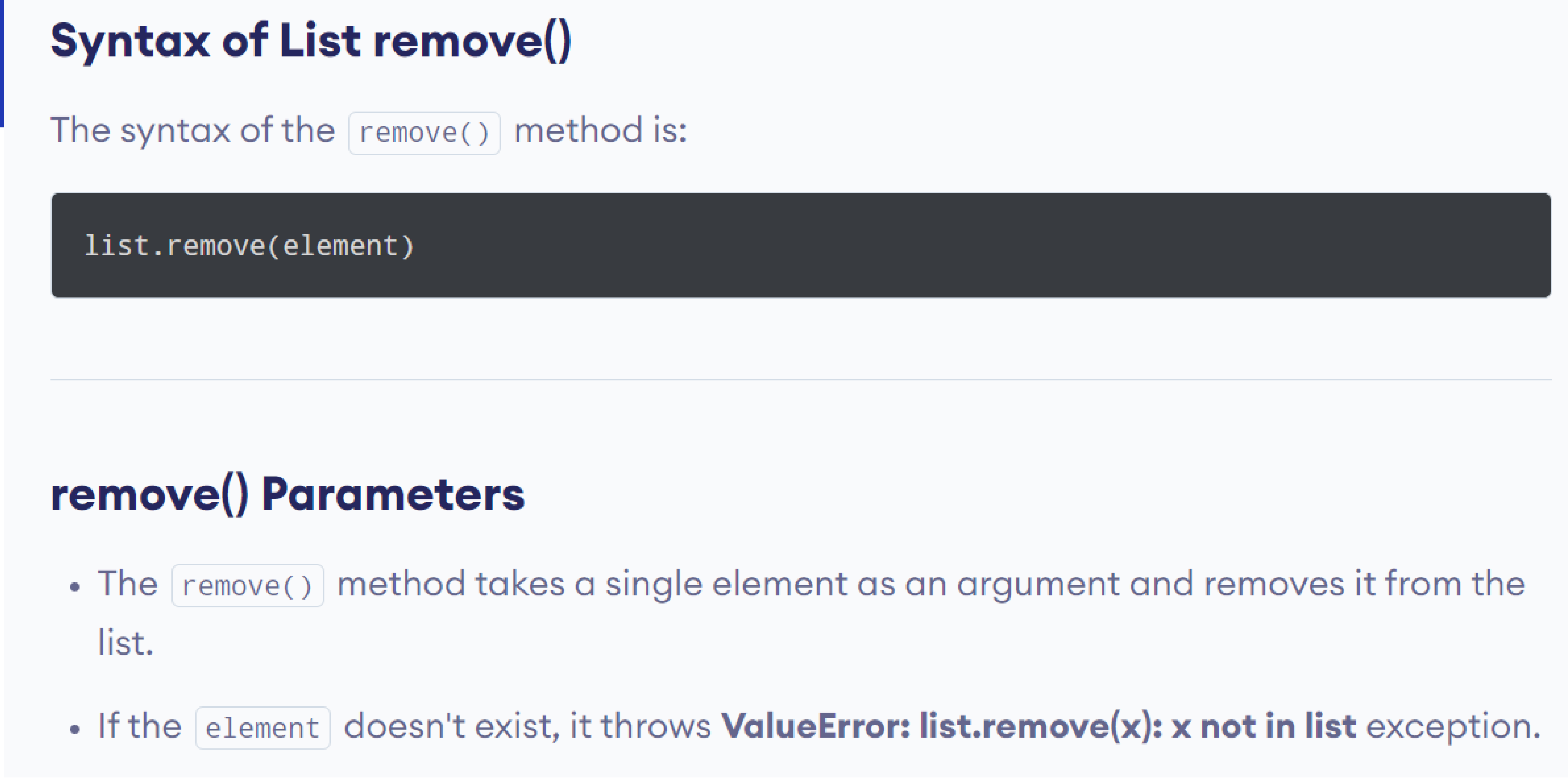
Return Value from clear()

The `clear()` method only empties the given list. It doesn't return any value.



```
# Defining a list
list = [{1, 2}, ('a'), ['1.1', '2.2']]

# clearing the list
list.clear()

print('List:', list)
```

remove



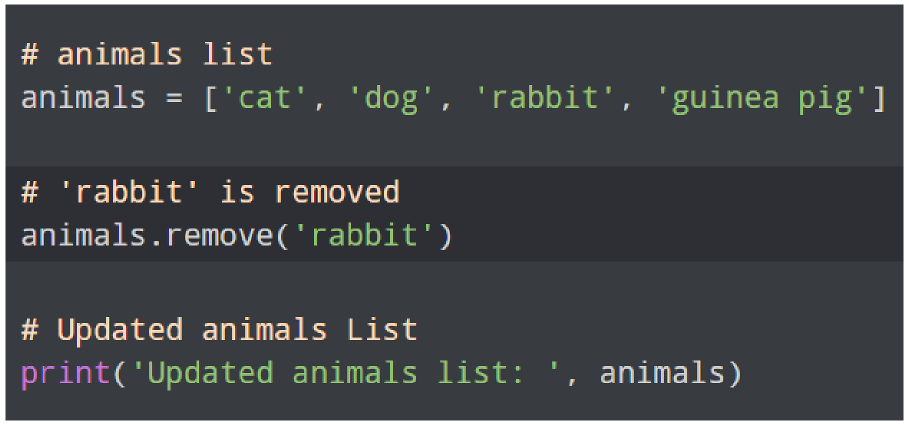
Syntax of List remove()

The syntax of the `remove()` method is:

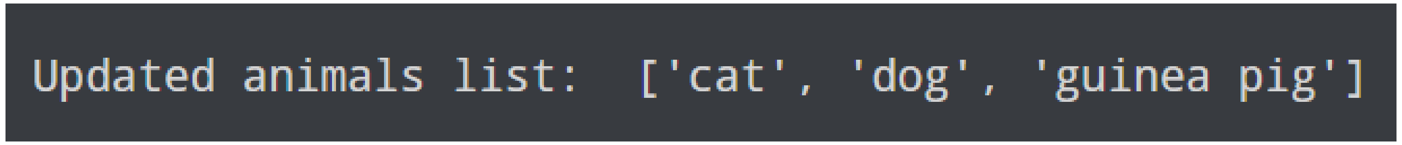
```
list.remove(element)
```

remove() Parameters

- The `remove()` method takes a single element as an argument and removes it from the list.
- If the `element` doesn't exist, it throws **ValueError: list.remove(x): x not in list** exception.



```
# animals list
animals = ['cat', 'dog', 'rabbit', 'guinea pig']

# 'rabbit' is removed
animals.remove('rabbit')

# Updated animals List
print('Updated animals list: ', animals)
```



```
Updated animals list:  ['cat', 'dog', 'guinea pig']
```

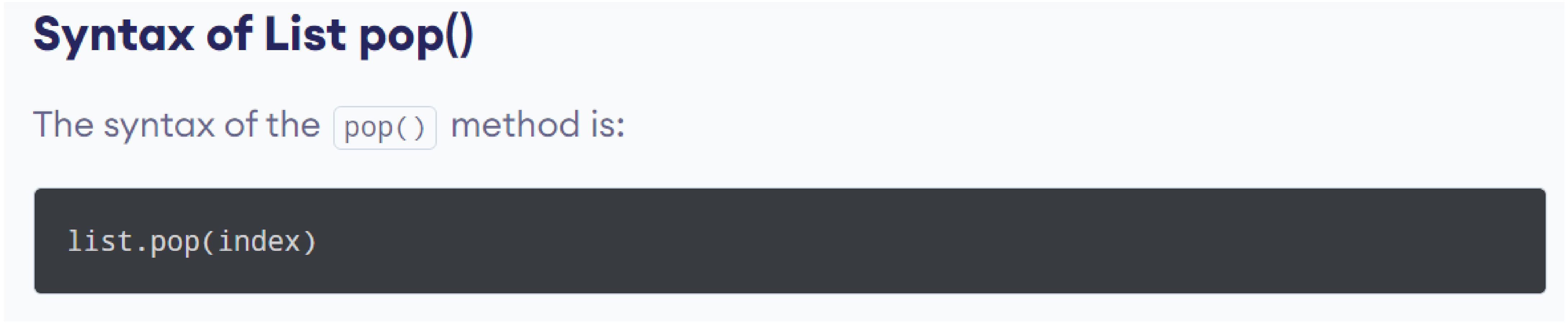
remove

```python
# animals list
animals = ['cat', 'dog', 'dog', 'guinea pig', 'dog']

# 'dog' is removed
animals.remove('dog')

# Updated animals list
print('Updated animals list: ', animals)
```

```
Updated animals list:  ['cat', 'dog', 'guinea pig', 'dog']
```

- If a list contains duplicate elements, the **remove()** method only removes the **first** matching element

pop()

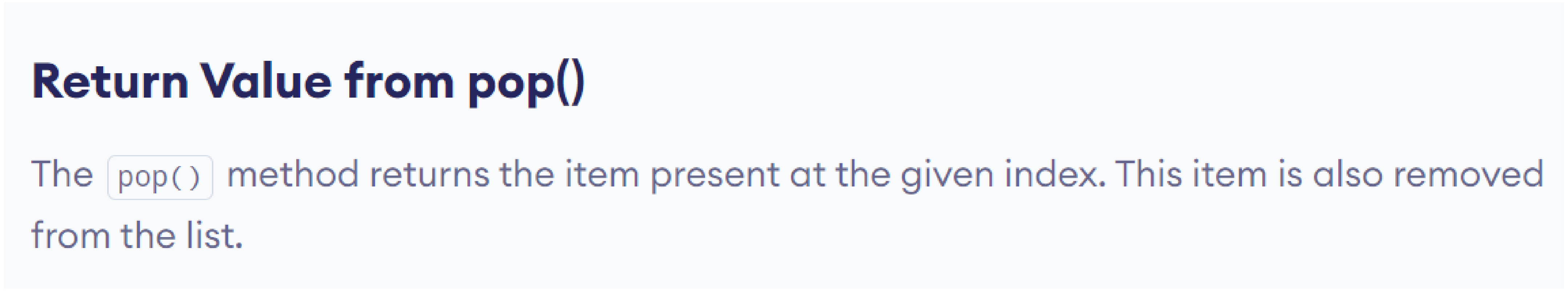


Syntax of List pop()

The syntax of the `pop()` method is:

```
list.pop(index)
```

- Default: index = -1



Return Value from pop()

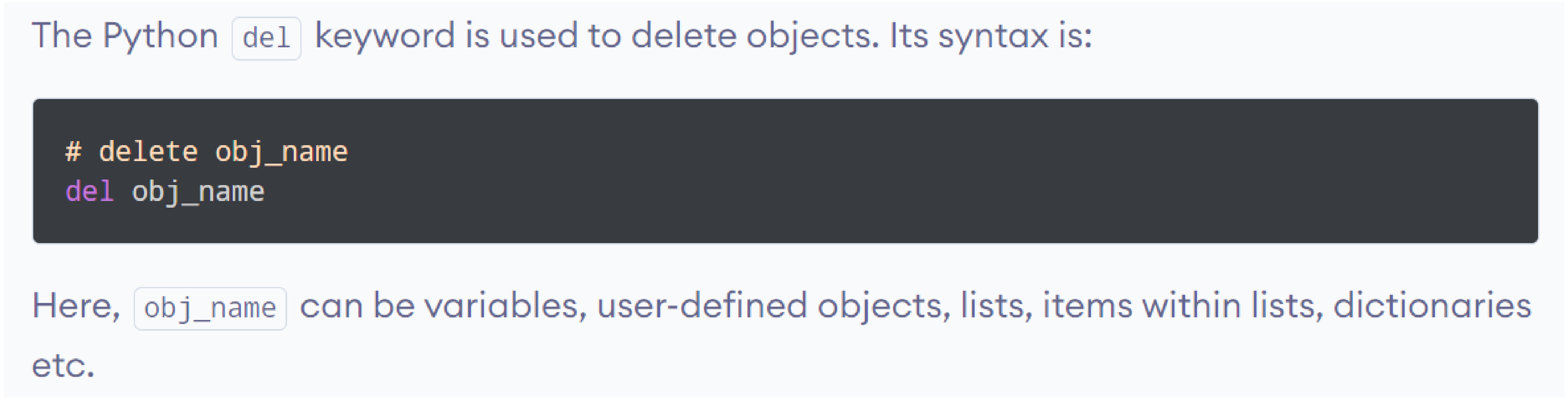
The `pop()` method returns the item present at the given index. This item is also removed from the list.

pop()

```
# programming languages list
languages = ['Python', 'Java', 'C++', 'French', 'C']

# remove and return the 4th item
return_value = languages.pop(3)
print('Return Value:', return_value)

# Updated List
print('Updated List:', languages)
```

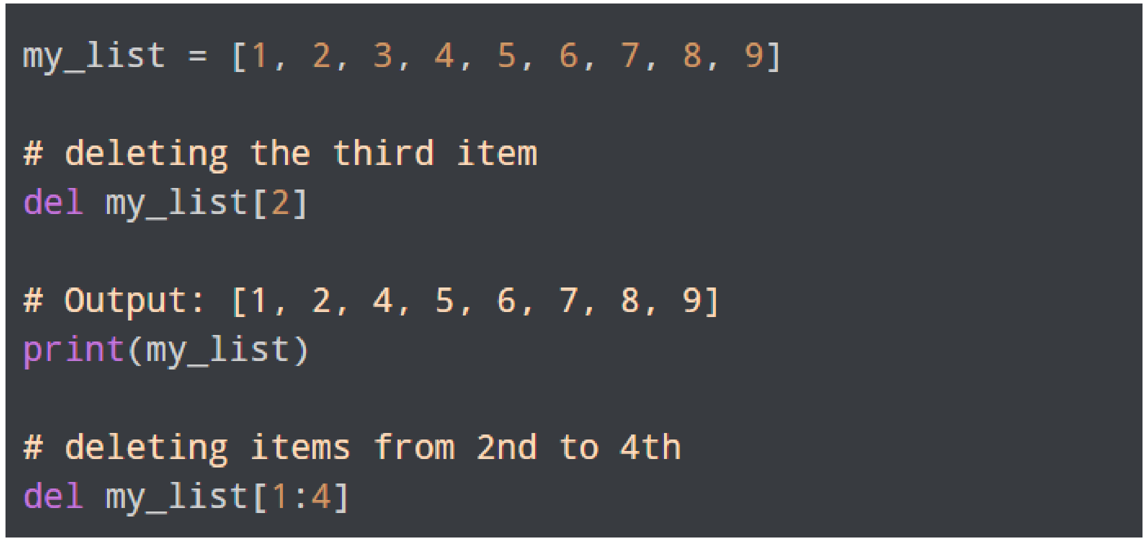
```
Return Value: French
Updated List: ['Python', 'Java', 'C++', 'C']
```

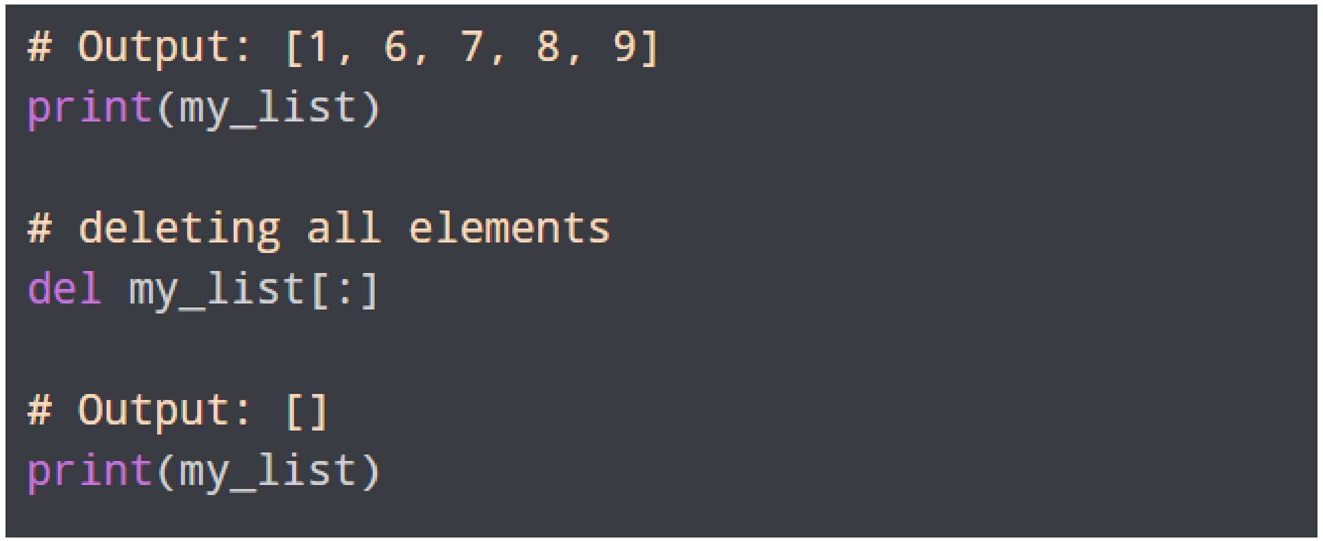
del statement


The Python `del` keyword is used to delete objects. Its syntax is:

```
# delete obj_name
del obj_name
```

Here, `obj_name` can be variables, user-defined objects, lists, items within lists, dictionaries etc.

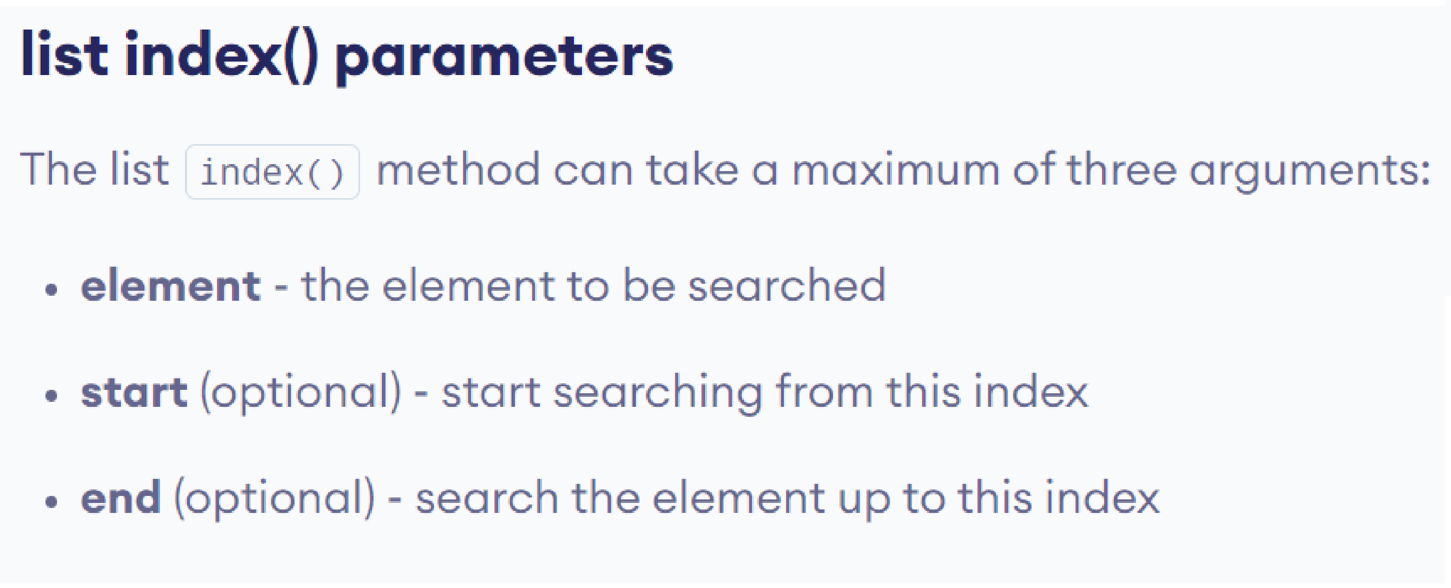

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# deleting the third item
del my_list[2]

# Output: [1, 2, 4, 5, 6, 7, 8, 9]
print(my_list)

# deleting items from 2nd to 4th
del my_list[1:4]
```


```
# Output: [1, 6, 7, 8, 9]
print(my_list)

# deleting all elements
del my_list[:]

# Output: []
print(my_list)
```

Other methods

- index()
- count()
- len()
- sort()
- reverse()
- copy()

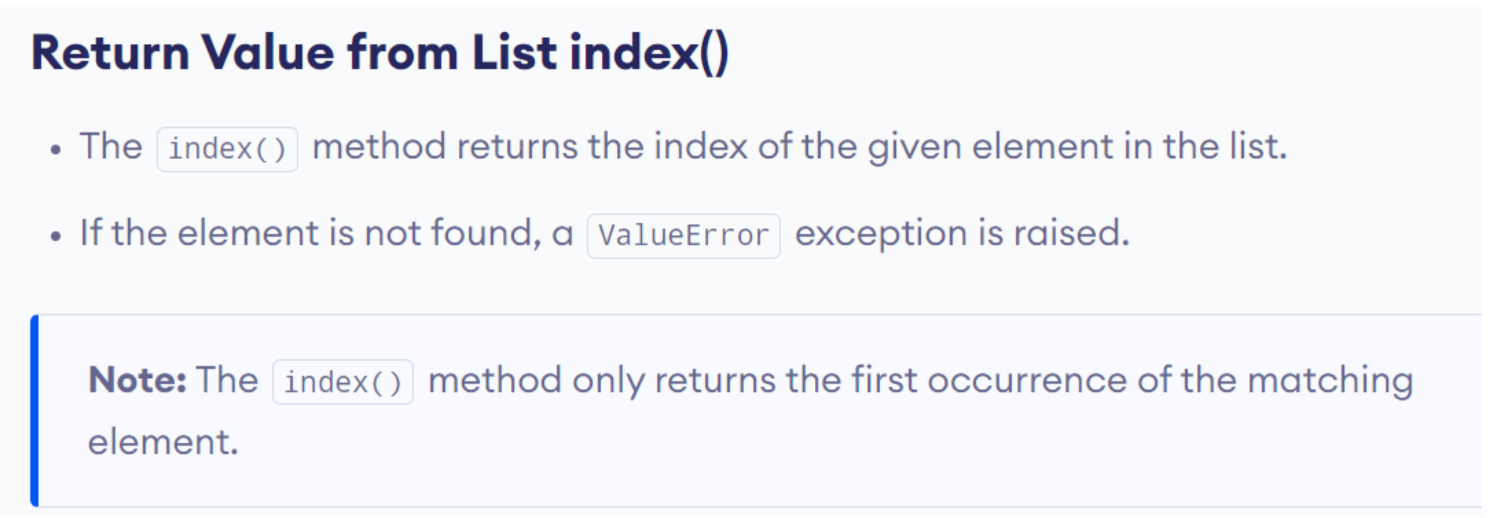
index



list index() parameters

The list `index()` method can take a maximum of three arguments:

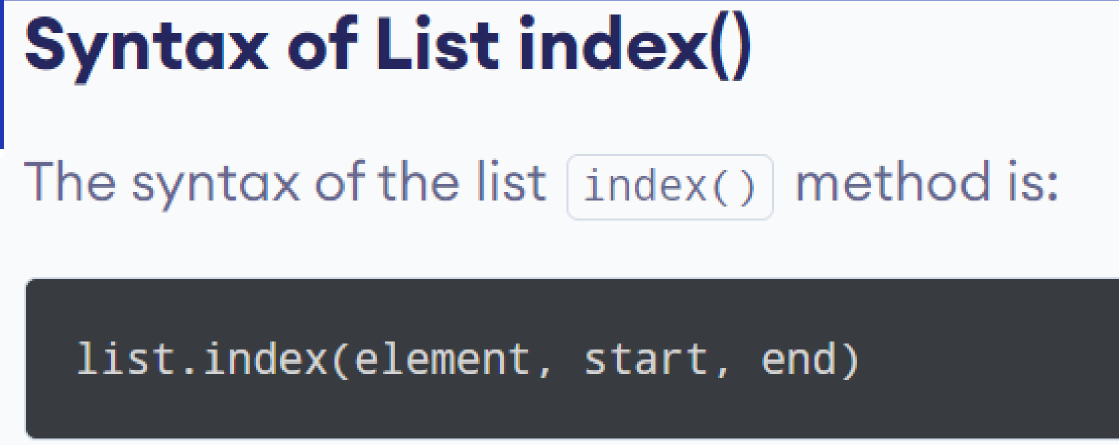
- **element** - the element to be searched
- **start** (optional) - start searching from this index
- **end** (optional) - search the element up to this index



Return Value from List index()

- The `index()` method returns the index of the given element in the list.
- If the element is not found, a `ValueError` exception is raised.

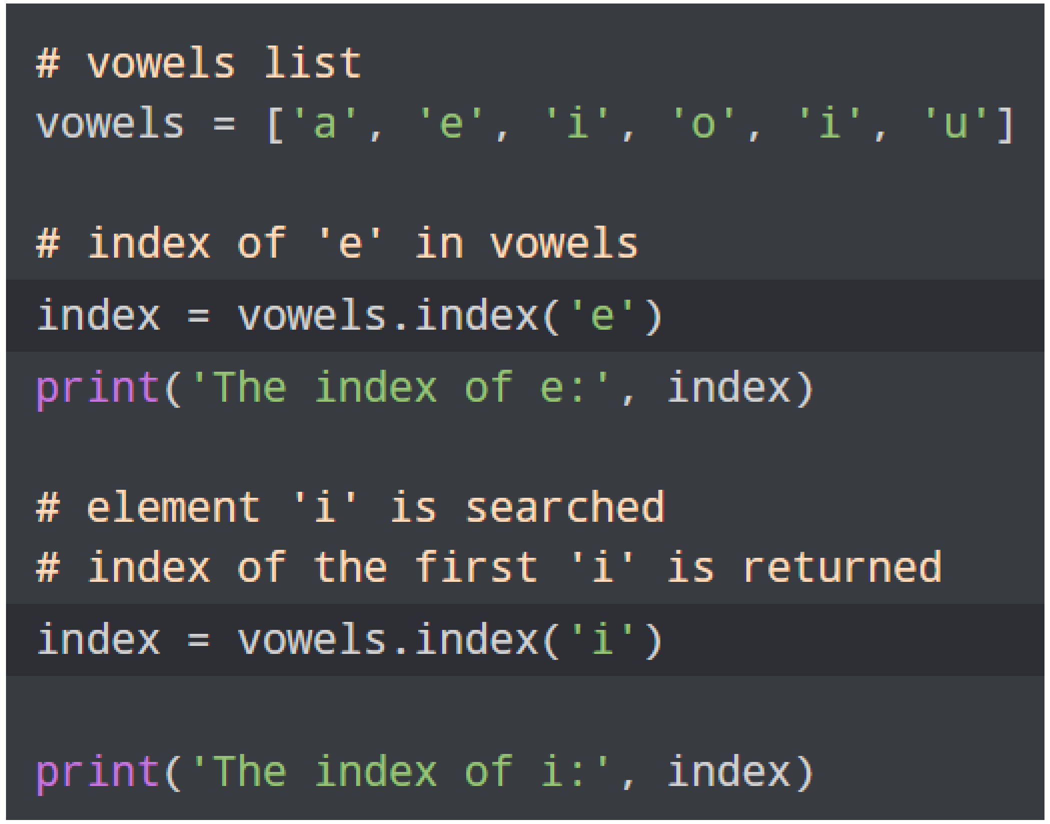
Note: The `index()` method only returns the first occurrence of the matching element.



Syntax of List index()

The syntax of the list `index()` method is:

```
list.index(element, start, end)
```



```python
# vowels list
vowels = ['a', 'e', 'i', 'o', 'i', 'u']

# index of 'e' in vowels
index = vowels.index('e')
print('The index of e:', index)

# element 'i' is searched
# index of the first 'i' is returned
index = vowels.index('i')

print('The index of i:', index)
```

Exercise

- Write a program to return the index of all occurrences of the matching element in list

A = [1, 2, 2, 3, 2, 2]

Element = 2

Return [1, 2, 4, 5]

count()

Syntax of List count()

The syntax of the `count()` method is:

```
list.count(element)
```

count() Parameters

The `count()` method takes a single argument:

- **element** - the element to be counted

Return value from count()

The `count()` method returns the number of times `element` appears in the list.

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'i', 'u']

# count element 'i'
count = vowels.count('i')

# print count
print('The count of i is:', count)

# count element 'p'
count = vowels.count('p')

# print count
print('The count of p is:', count)
```

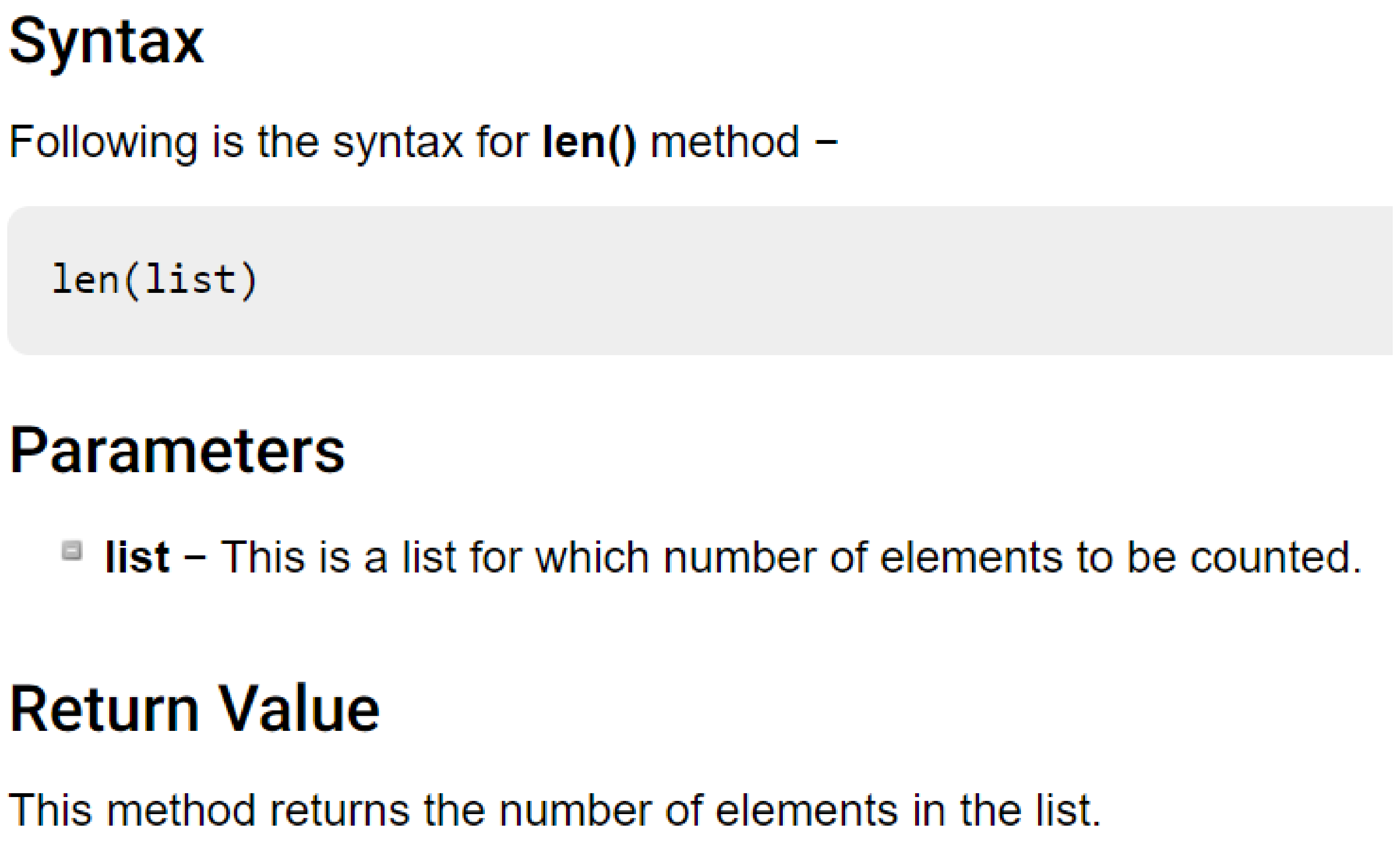
Output

```
The count of i is: 2
The count of p is: 0
```

Exercise

- Write a program to count all prime numbers in a list of positive integers

len()



Syntax

Following is the syntax for **len()** method –

```
len(list)
```

Parameters

- **list** – This is a list for which number of elements to be counted.

Return Value

This method returns the number of elements in the list.

sort()

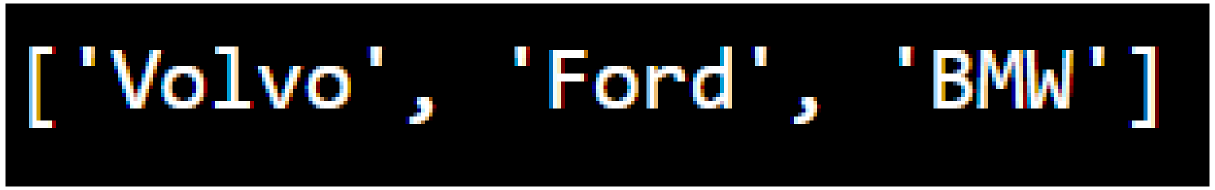
Syntax

```
list.sort(reverse=True|False, key=myFunc)
```

Parameter Values

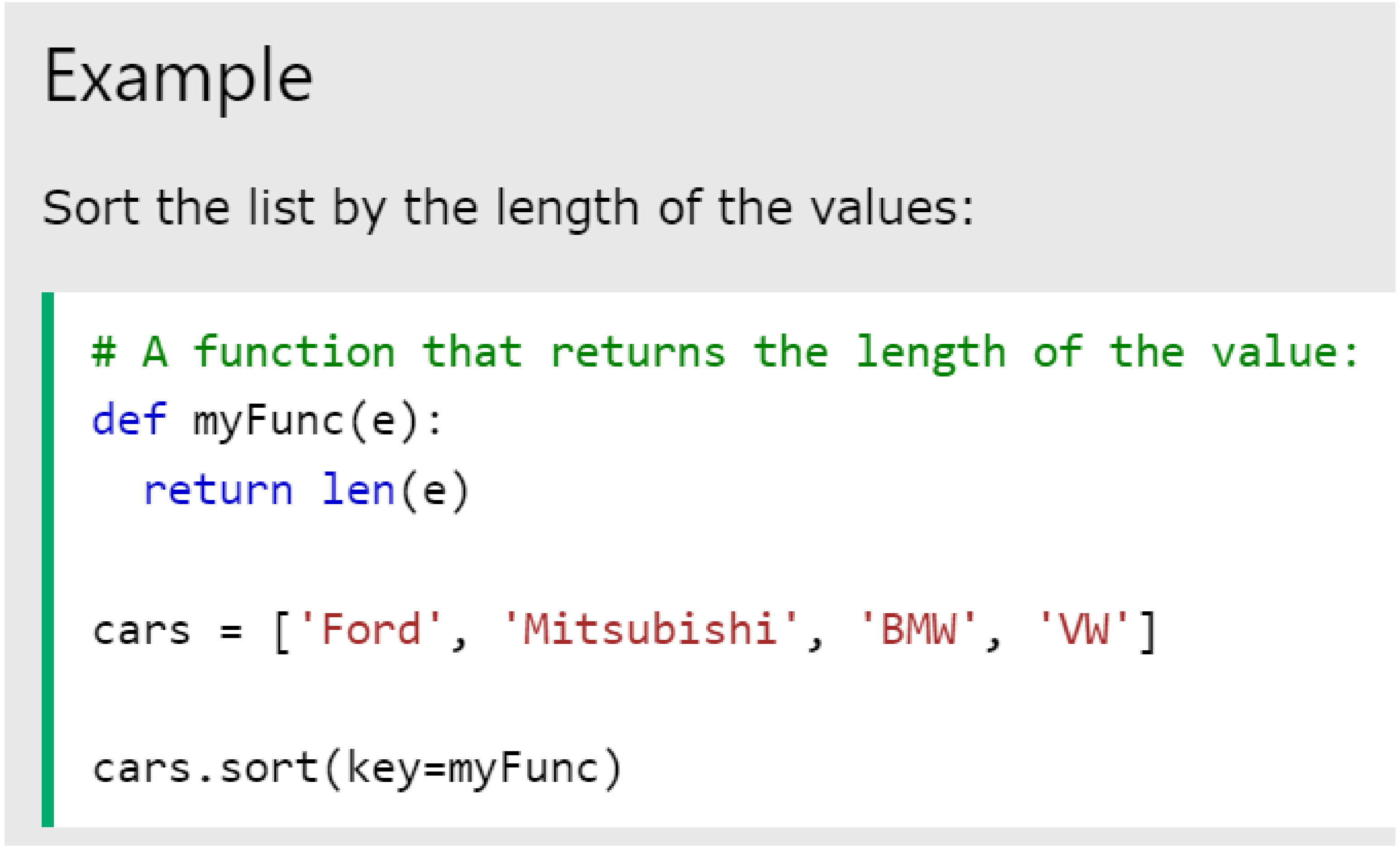
Parameter	Description
reverse	Optional. reverse=True will sort the list descending. Default is reverse=False
key	Optional. A function to specify the sorting criteria(s)

```
cars = ['Ford', 'BMW', 'Volvo']

cars.sort(reverse=True)

print(cars)
```



```
['Volvo', 'Ford', 'BMW']
```

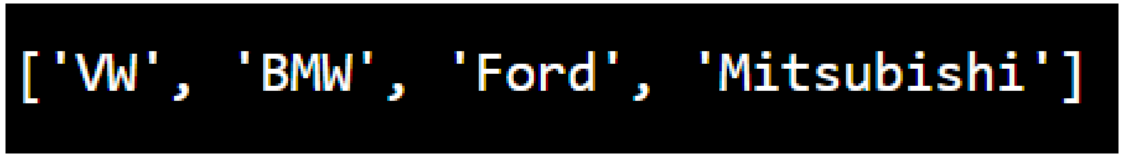
sort()



Example

Sort the list by the length of the values:

```python
# A function that returns the length of the value:
def myFunc(e):
  return len(e)

cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']

cars.sort(key=myFunc)
```

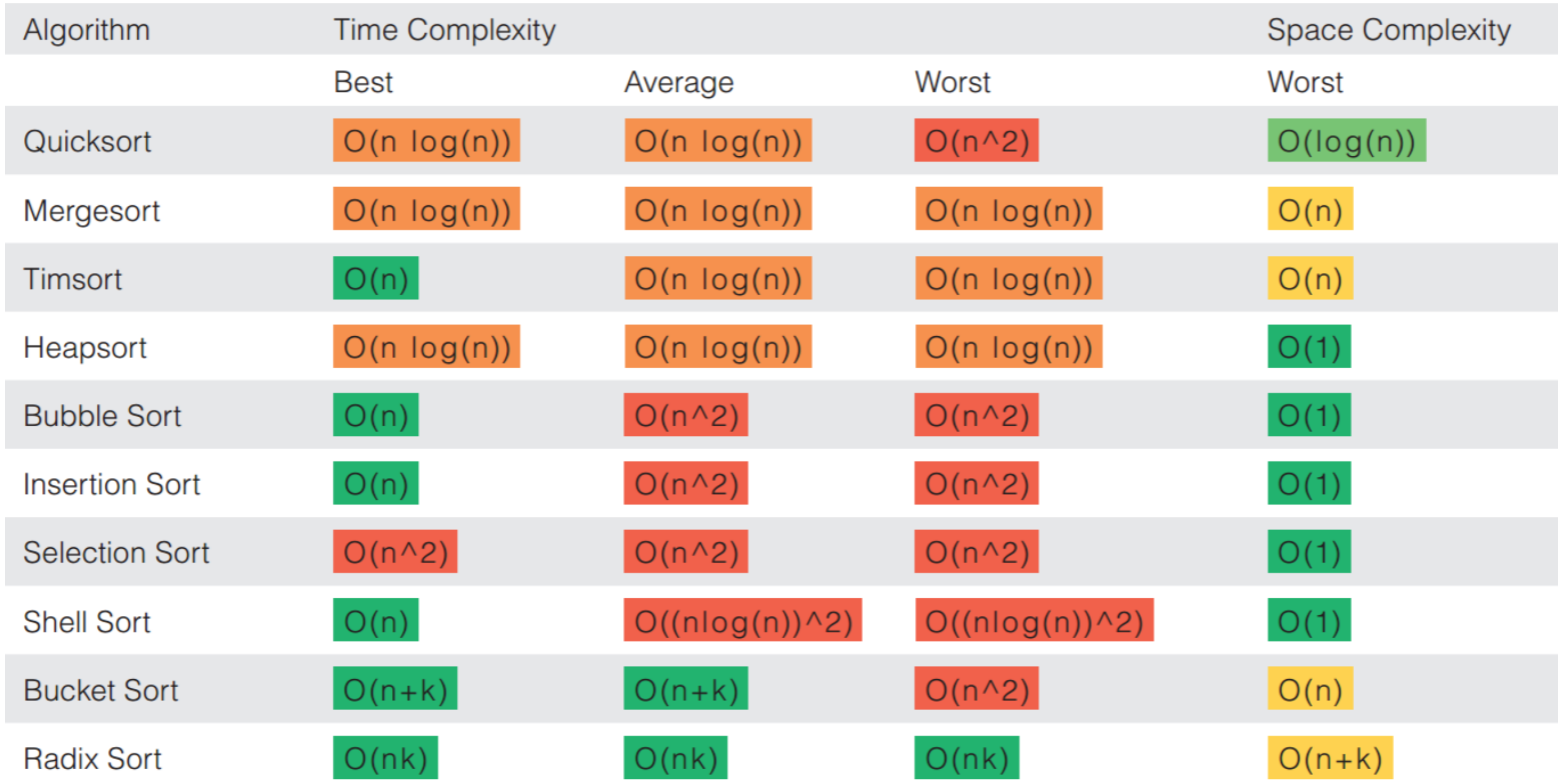


```
['VW', 'BMW', 'Ford', 'Mitsubishi']
```

Part 3: Sorting Algorithms

Sort List without pre-defined func

- In this course, we only introduce
  - selection sort
  - bubble sort



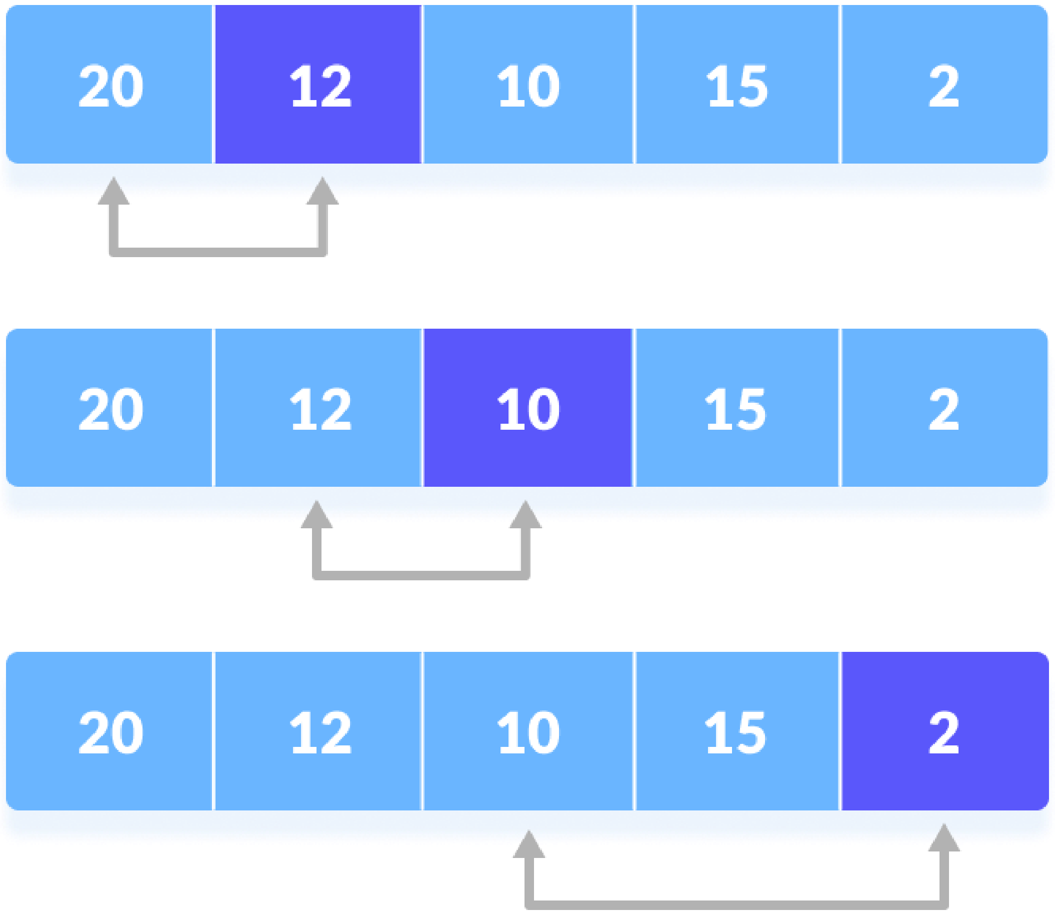
Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	O(n log(n))	O(n log(n))	O(n^2)	O(log(n))
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Timsort	O(n)	O(n log(n))	O(n log(n))	O(n)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)

Selection Sort

- Main idea: repeatedly doing the following procedure:
    - finding the minimum element (considering ascending order) from unsorted part
    - putting it at the beginning

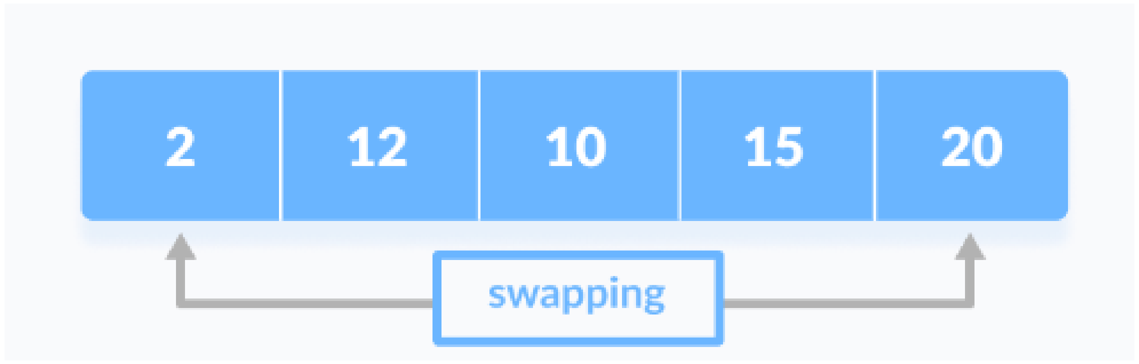
Selection Sort

- Step 1: find the minimum value of the list

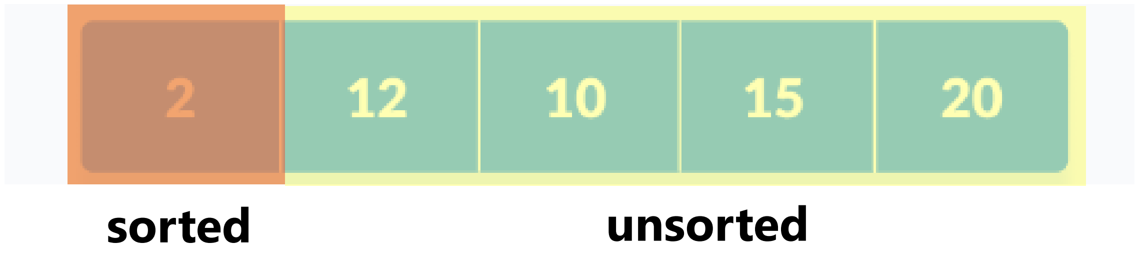


Selection Sort

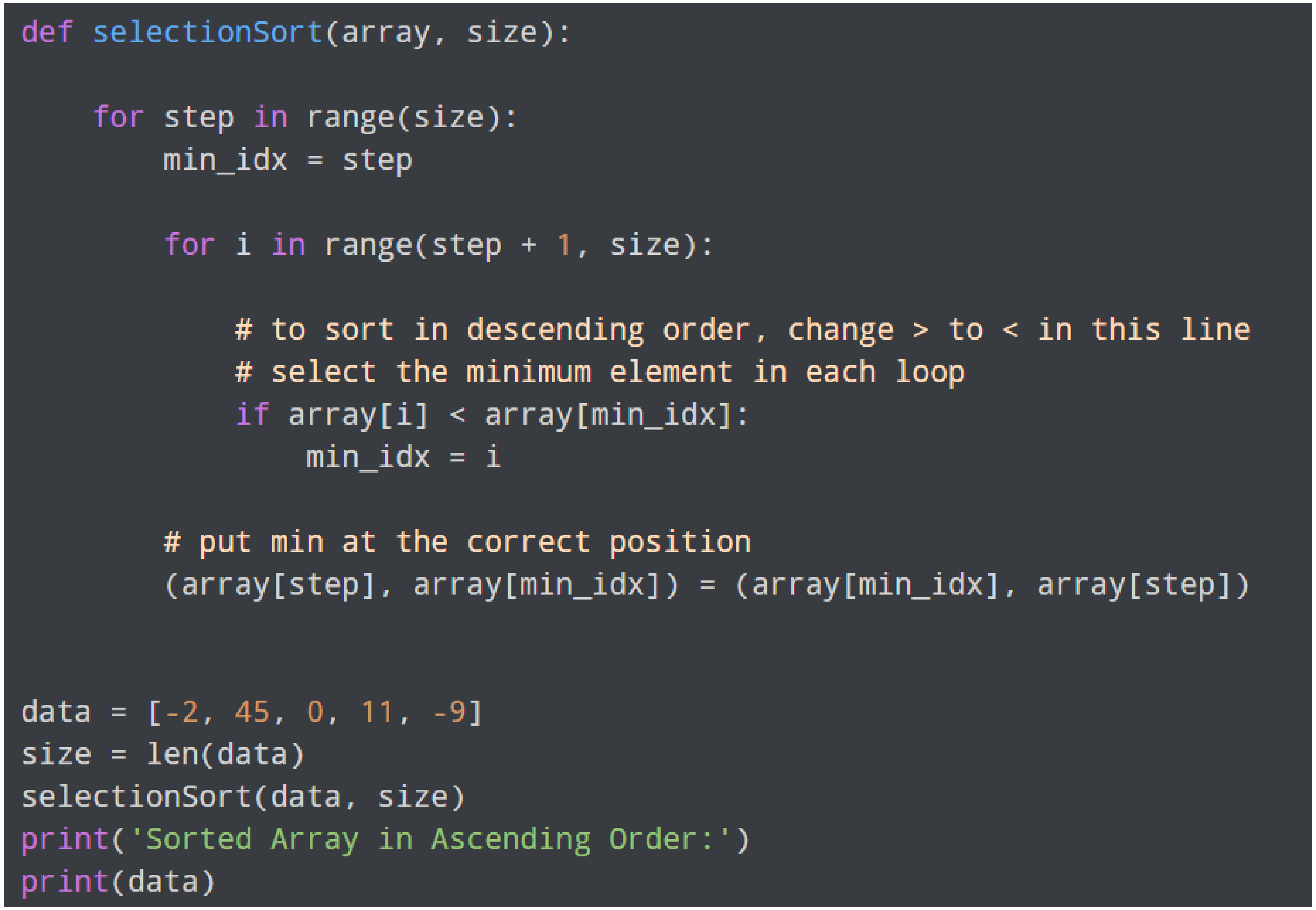
- Step 2: min val is placed in the front of the unsorted list



- Step 3: repeatedly step 1-2 for the unsorted parts



Selection Sort



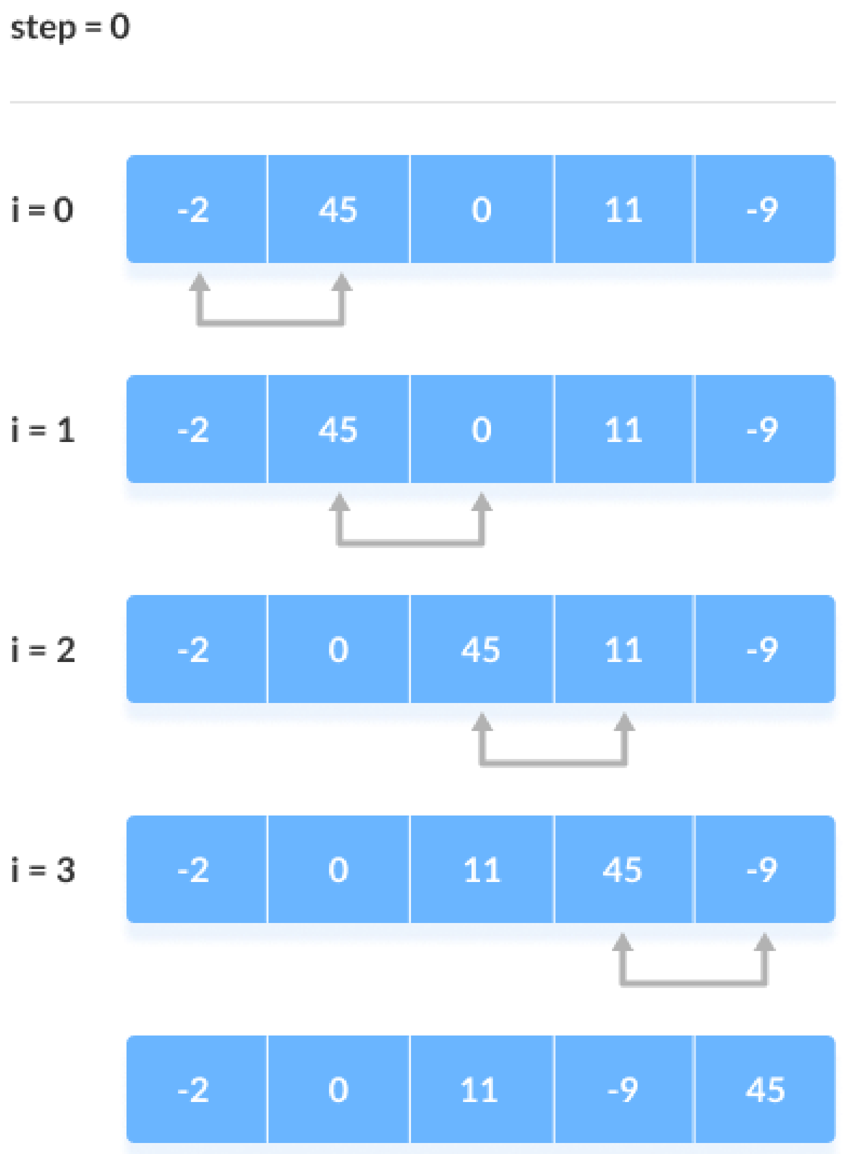
```python
def selectionSort(array, size):

    for step in range(size):
        min_idx = step

        for i in range(step + 1, size):

            # to sort in descending order, change > to < in this line
            # select the minimum element in each loop
            if array[i] < array[min_idx]:
                min_idx = i

        # put min at the correct position
        (array[step], array[min_idx]) = (array[min_idx], array[step])


data = [-2, 45, 0, 11, -9]
size = len(data)
selectionSort(data, size)
print('Sorted Array in Ascending Order:')
print(data)
```

Bubble Sort

- Main idea: compares two adjacent elements and swaps them until they are in the intended order

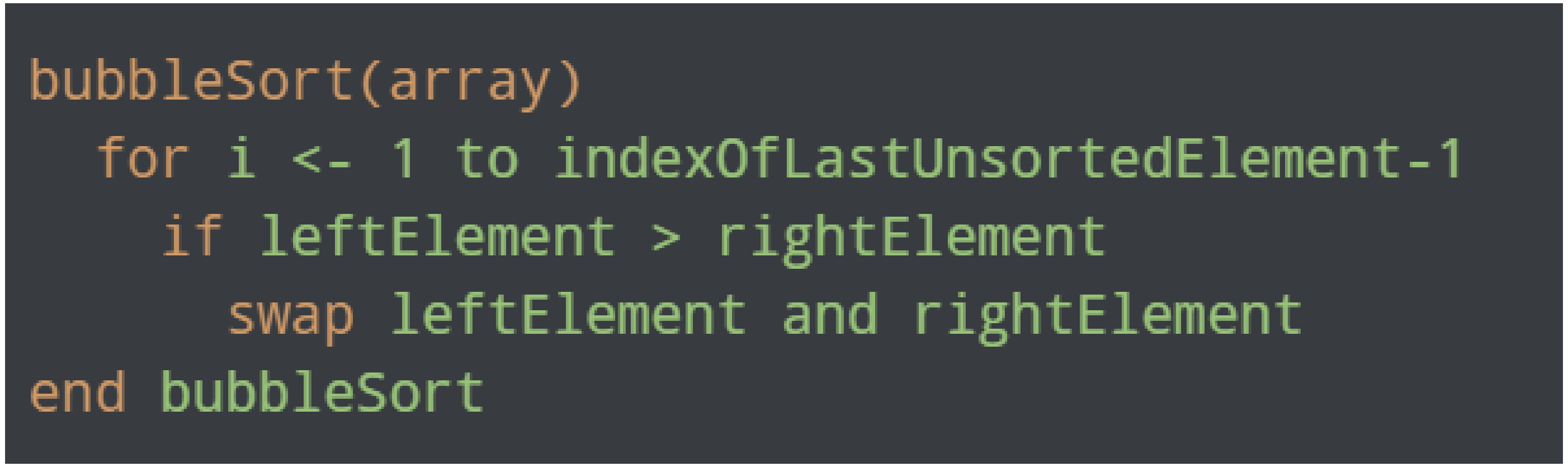
Bubble Sort

- Step 1: Compare and Swap
  - If i^{th} and $(i+1)^{th}$ elements are in the incorrect positions, swap them



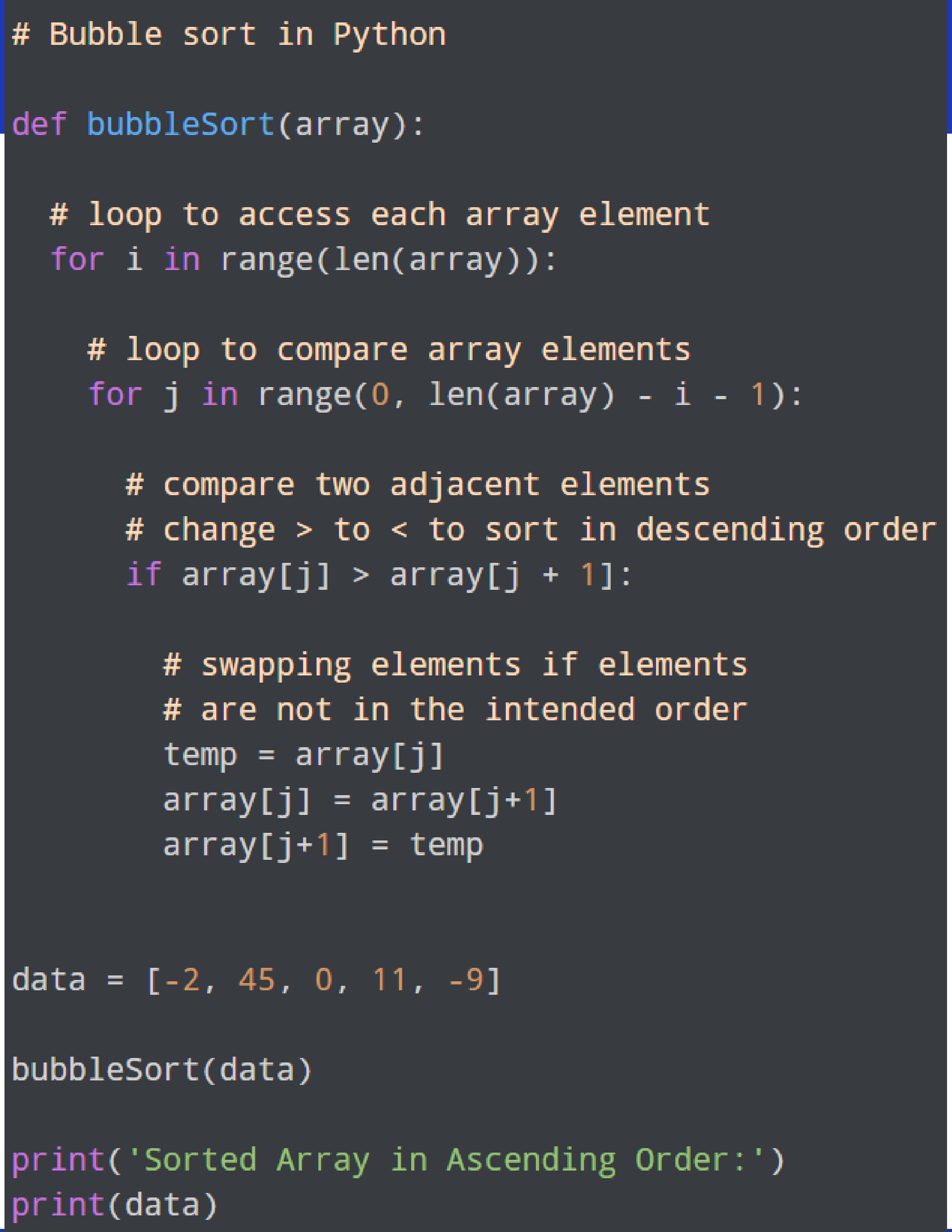
Bubble Sort

- Step 2: Remaining Iteration
  - Repeat Step 1
  - Until sorted list or len(list) – 1 times



```
bubbleSort(array)
  for i <- 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap leftElement and rightElement
end bubbleSort
```

Bubble Sort



```python
# Bubble sort in Python

def bubbleSort(array):

 # loop to access each array element
 for i in range(len(array)):

  # loop to compare array elements
  for j in range(0, len(array) - i - 1):

   # compare two adjacent elements
   # change > to < to sort in descending order
   if array[j] > array[j + 1]:

    # swapping elements if elements
    # are not in the intended order
    temp = array[j]
    array[j] = array[j+1]
    array[j+1] = temp


data = [-2, 45, 0, 11, -9]

bubbleSort(data)

print('Sorted Array in Ascending Order:')
print(data)
```

Exercise

1. Write a program to move all the odd numbers into the left-hand side and the even numbers into the right-hand size. Note that the relative positions of all elements are remained and not using the temporary list (in-place algorithm)

e.g:

the unsorted list: [4, 5, 1, 7, 9, 3, 0, 2]

the sorted list: [5, 1, 7, 9, 3, 4, 0, 2]

THANK YOU
for YOUR ATTENTION