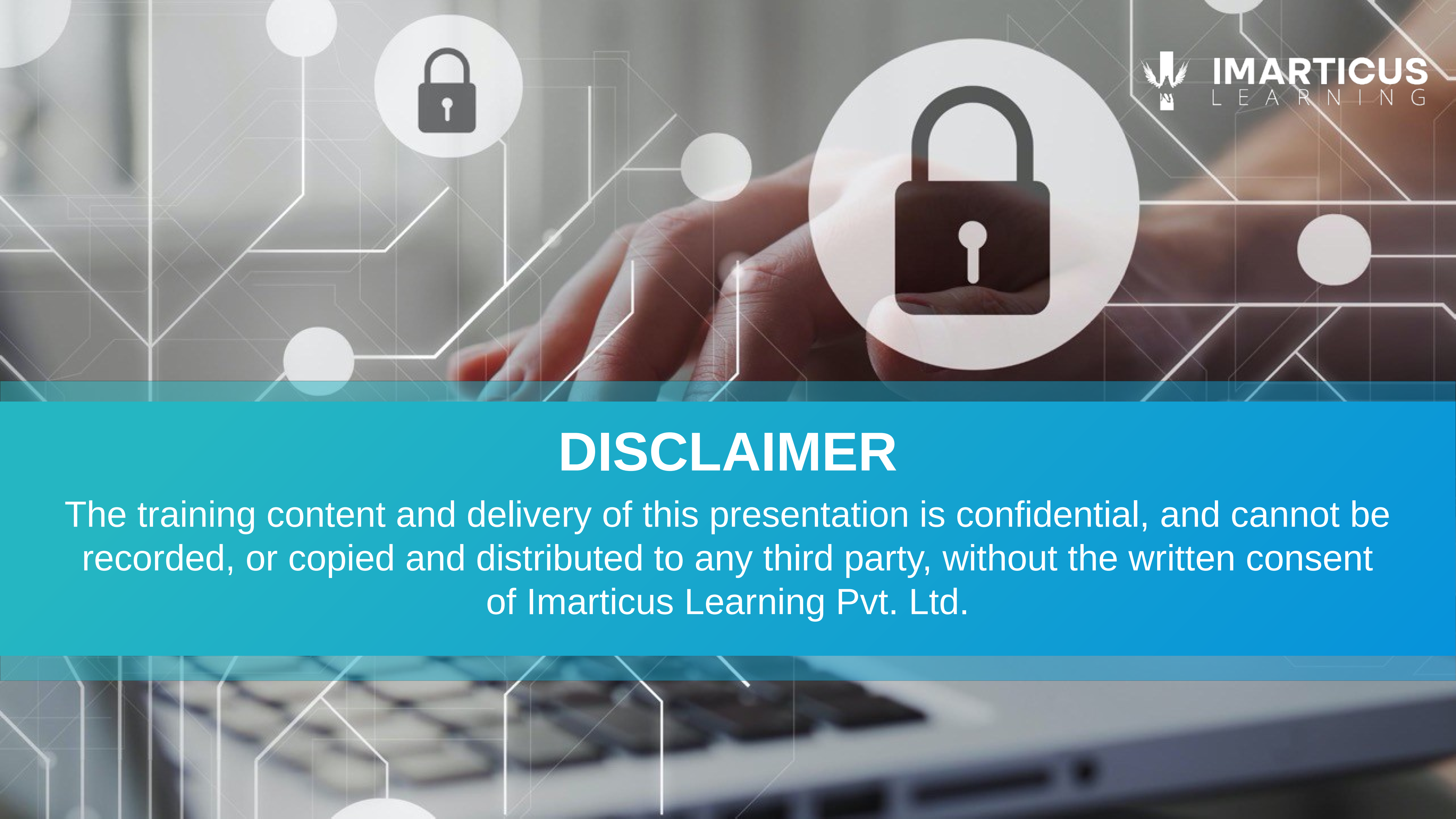
Python Programming

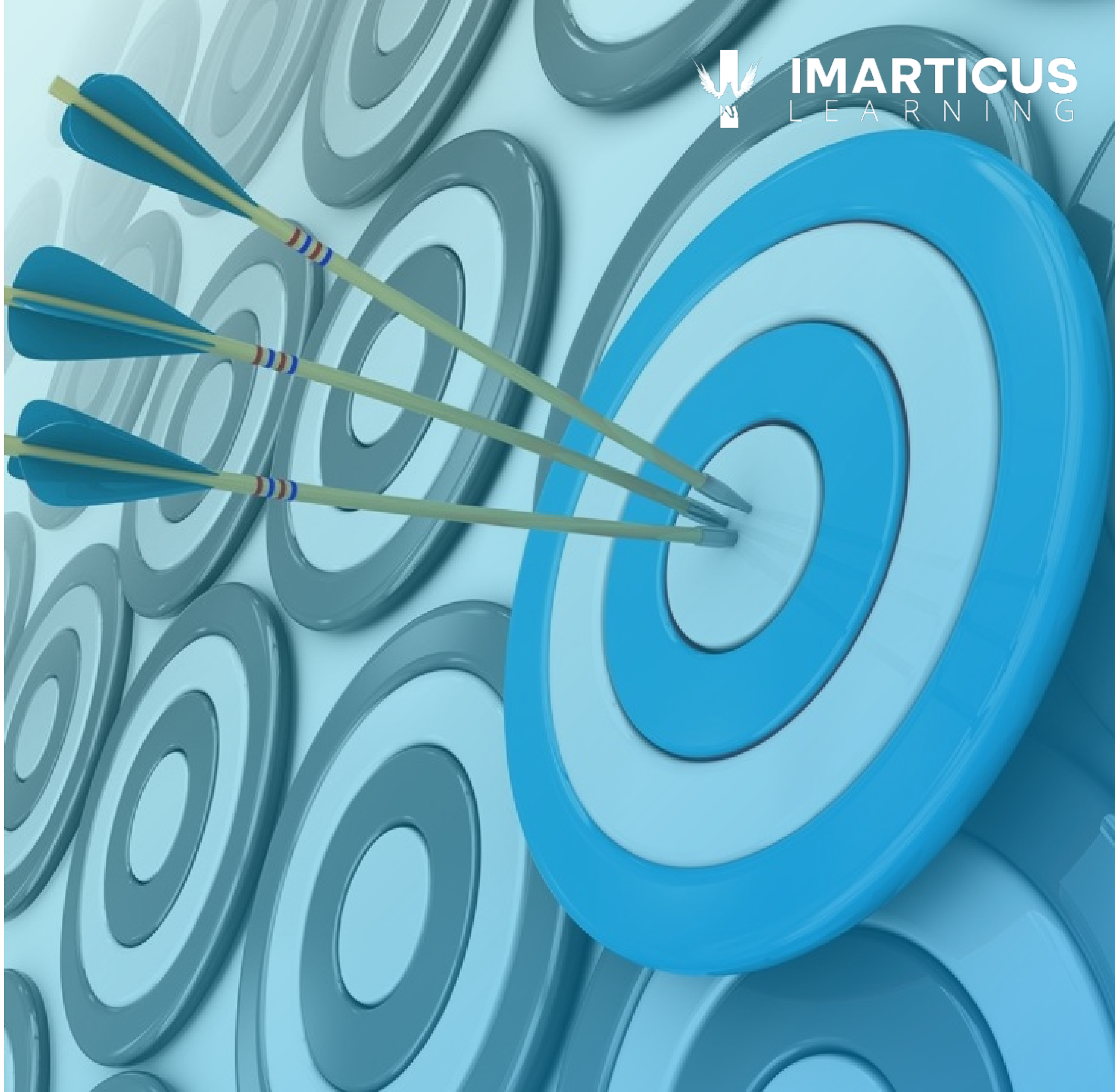
Python Objects



DISCLAIMER

The training content and delivery of this presentation is confidential, and cannot be recorded, or copied and distributed to any third party, without the written consent of Imarticus Learning Pvt. Ltd.

LEARNING OBJECTIVES



At the end of this session, you will learn:

- Python Collection Objects
- Strings
- List
- Tuple
- Dictionary
- List Comprehension

Python Collection Objects

PYTHON COLLECTION OBJECTS

- Collection objects are objects that can hold any number of arbitrary elements
- Collections or container objects provides a way to access elements and iterate over them

Some collection objects are:

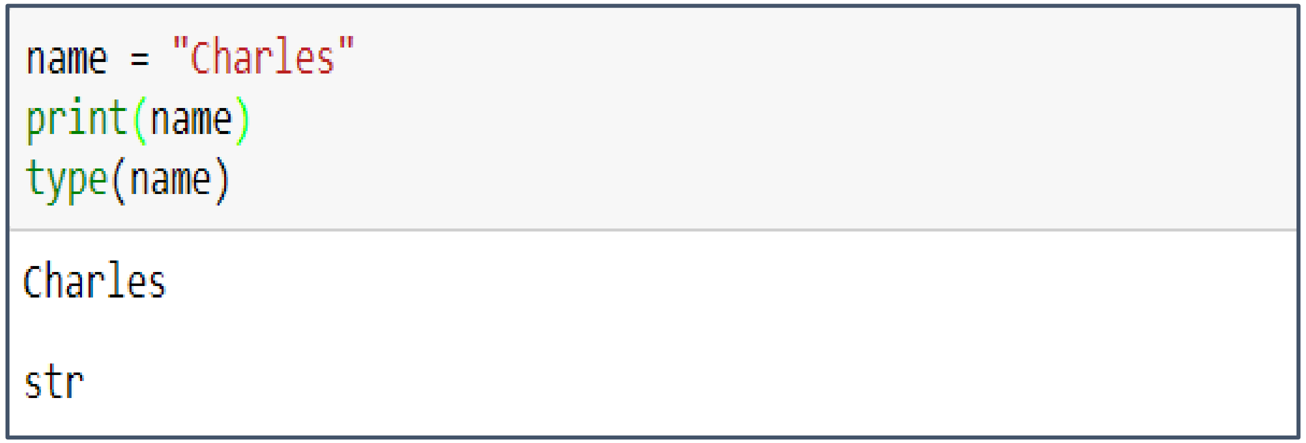
- String
- List
- Tuple
- Dictionary

Strings

CREATING STRING VARIABLES

- String variables are variables that hold zero or more characters such as letters, numbers, spaces, commas and many more

Use type(variable_name) to check the data type of variable declared



```
name = "Charles"
print(name)
type(name)
```

```
Charles

str
```

INDEXING AND SLICING

IMARTICUS LEARNING

1

Indexing is a process of referring an element of an iterable (string, list, tuple, etc) by its position, called index, within the iterable

2

Slicing is a process of getting a subset of elements from an iterable based on their indices

3

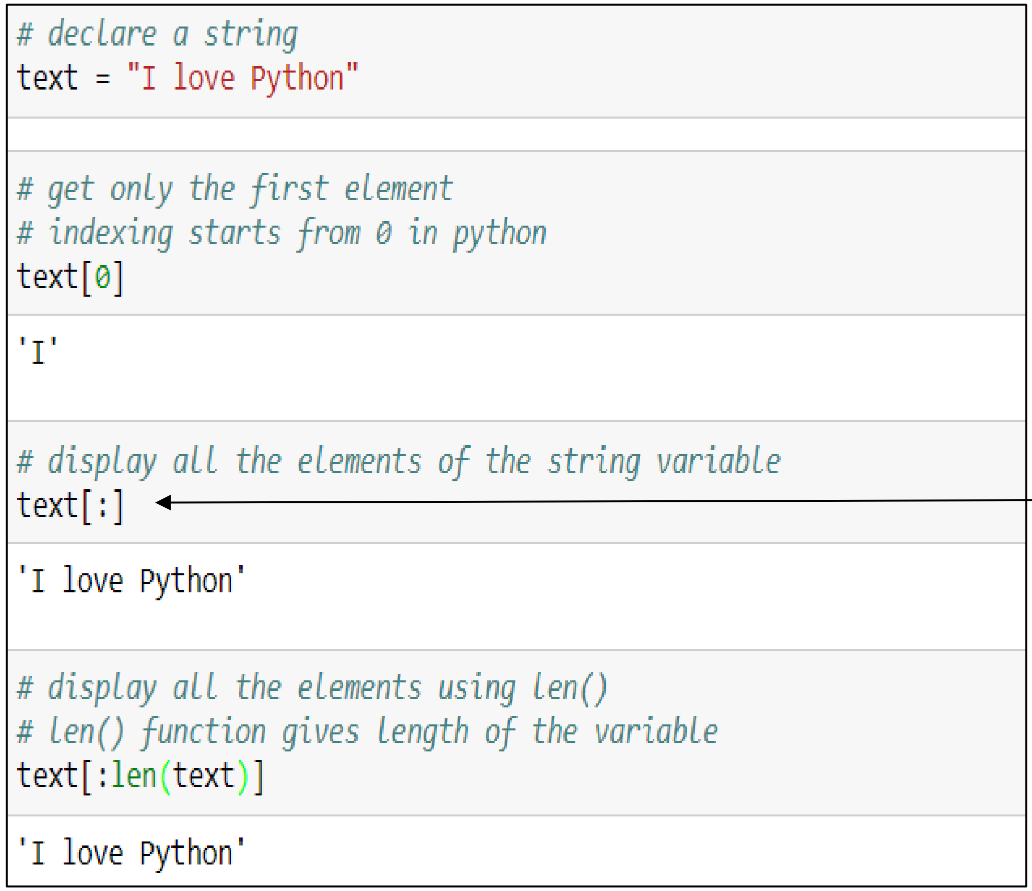
To retrieve an element of the list, we use the index operator []. Iterables are “zero indexed”, so [0] returns the zero-th (i.e. the left-most) item in the list, and [1] returns the one-th item

4

Python also allows you to index from the end of the list using a negative number

INDEXING AND SLICING

Example



```
# declare a string
text = "I love Python"
```

```
# get only the first element
# indexing starts from 0 in python
text[0]
```

```
'I'
```

```
# display all the elements of the string variable
text[:]
```

```
'I love Python'
```

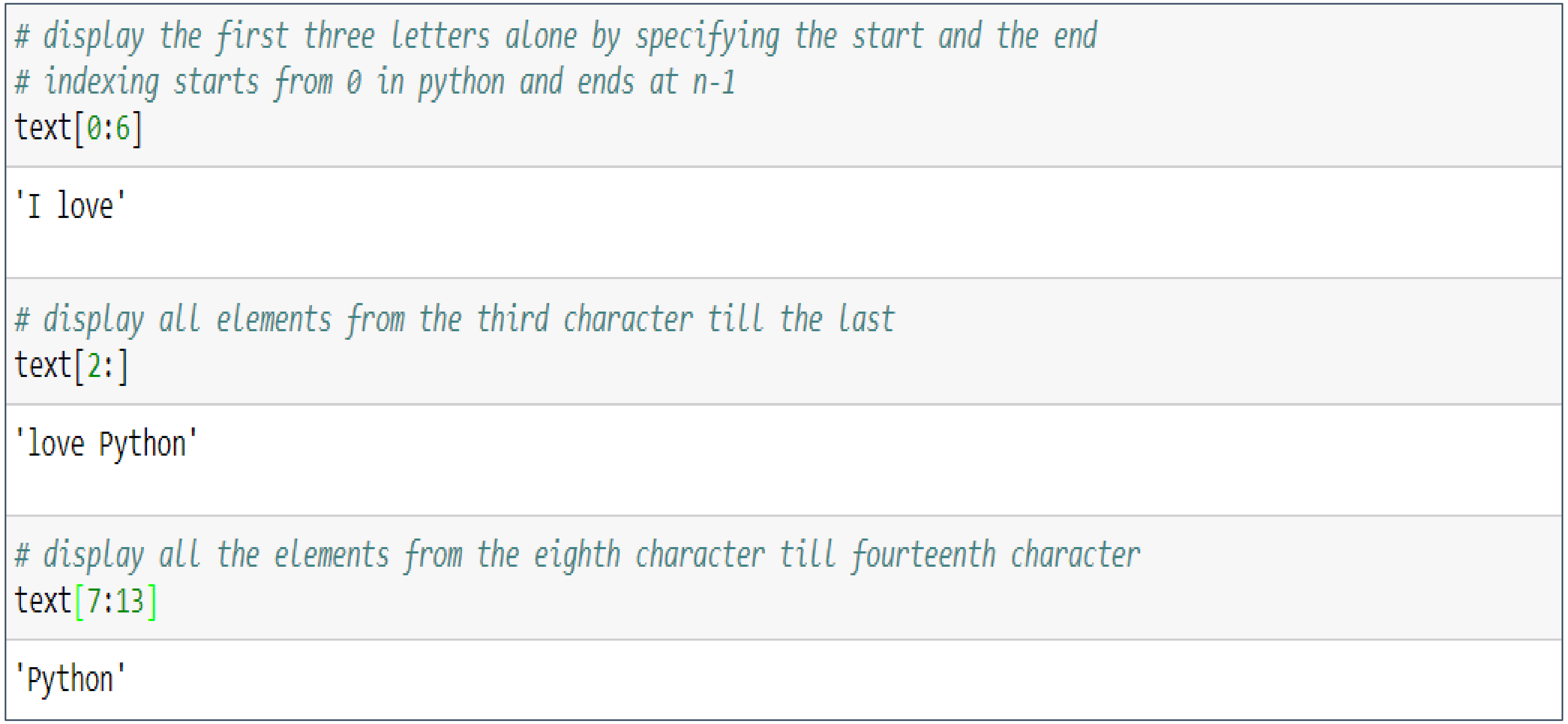
```
# display all the elements using len()
# len() function gives length of the variable
text[:len(text)]
```

```
'I love Python'
```

‘:’ is used to specify range of the sequence to be sliced

INDEXING AND SLICING

Example



```
# display the first three letters alone by specifying the start and the end
# indexing starts from 0 in python and ends at n-1
text[0:6]
```

```
'I love'
```

```
# display all elements from the third character till the last
text[2:]
```

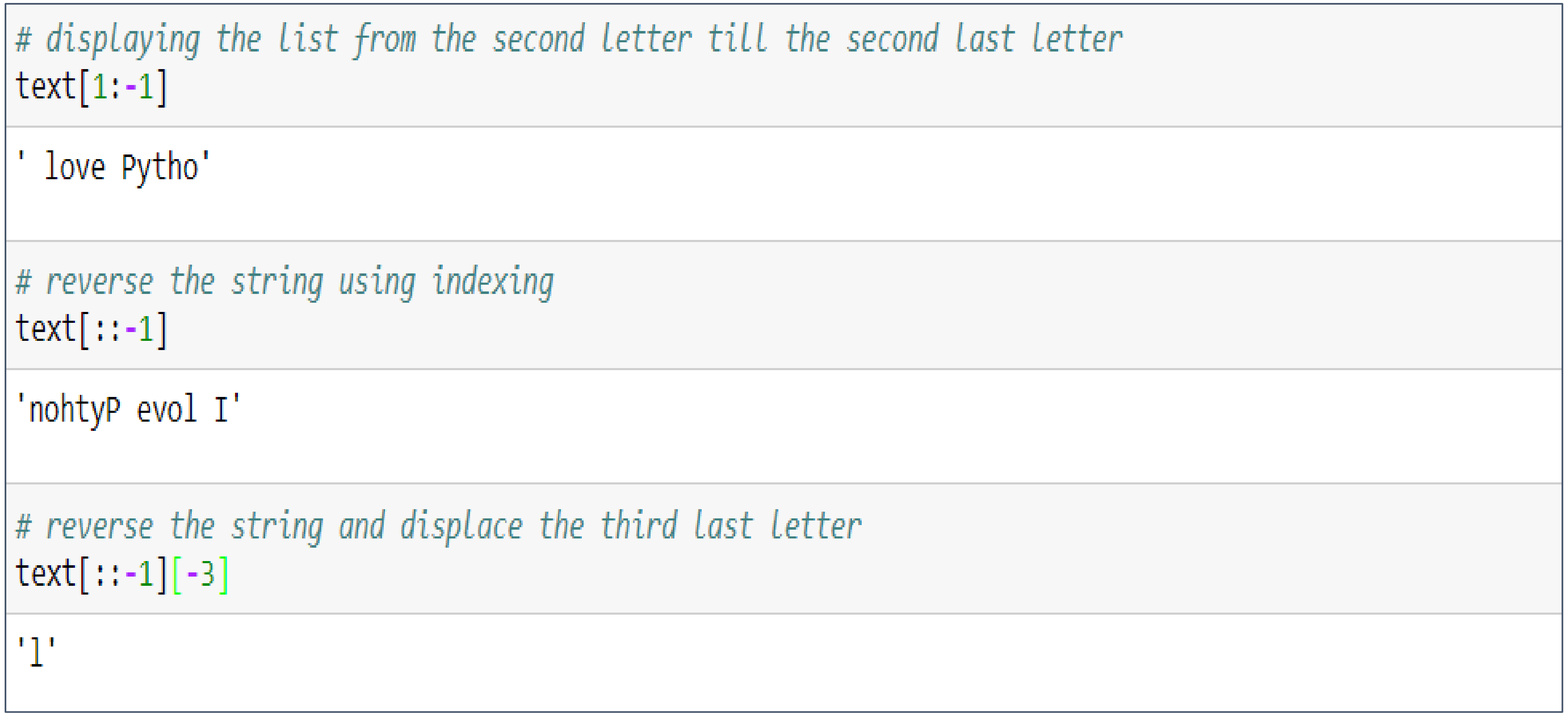
```
'love Python'
```

```
# display all the elements from the eighth character till fourteenth character
text[7:13]
```

```
'Python'
```

INDEXING AND SLICING

Example



```
# displaying the list from the second letter till the second last letter
text[1:-1]
```

```
' love Pytho'
```

```
# reverse the string using indexing
text[::-1]
```

```
'nohtyP evol I'
```

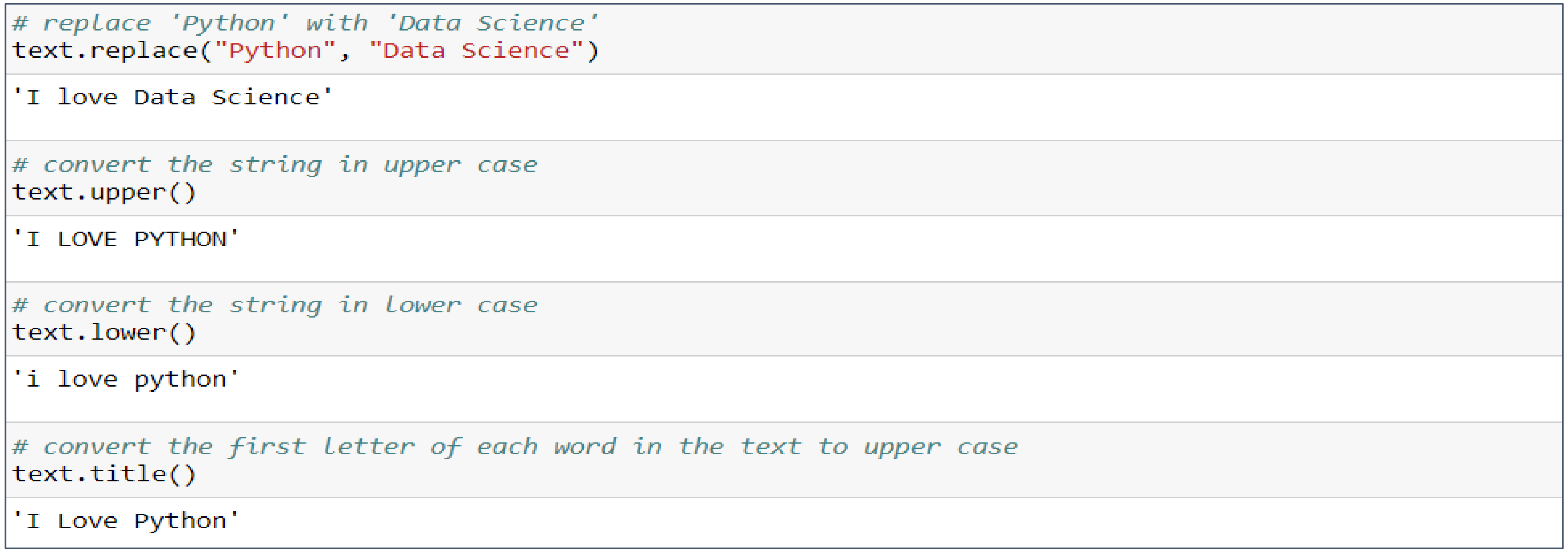
```
# reverse the string and displace the third last letter
text[::-1][-3]
```

```
'l'
```

INDEXING AND SLICING

Python has a set of built-in methods that you can use on strings

Some of the string built-in methods are as follows:



```
# replace 'Python' with 'Data Science'
text.replace("Python", "Data Science")
```

```
'I love Data Science'
```

```
# convert the string in upper case
text.upper()
```

```
'I LOVE PYTHON'
```

```
# convert the string in lower case
text.lower()
```

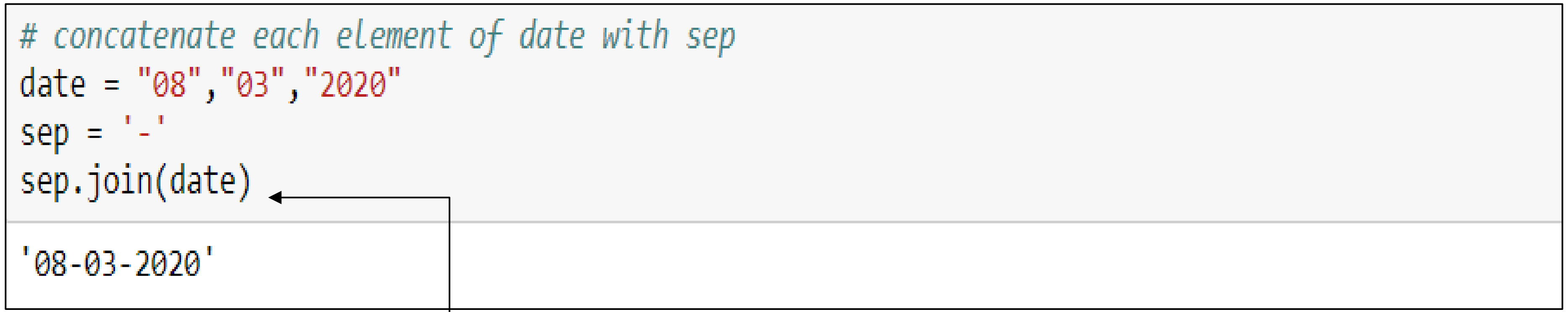
```
'i love python'
```

```
# convert the first letter of each word in the text to upper case
text.title()
```

```
'I Love Python'
```

STRING HANDLING

Example



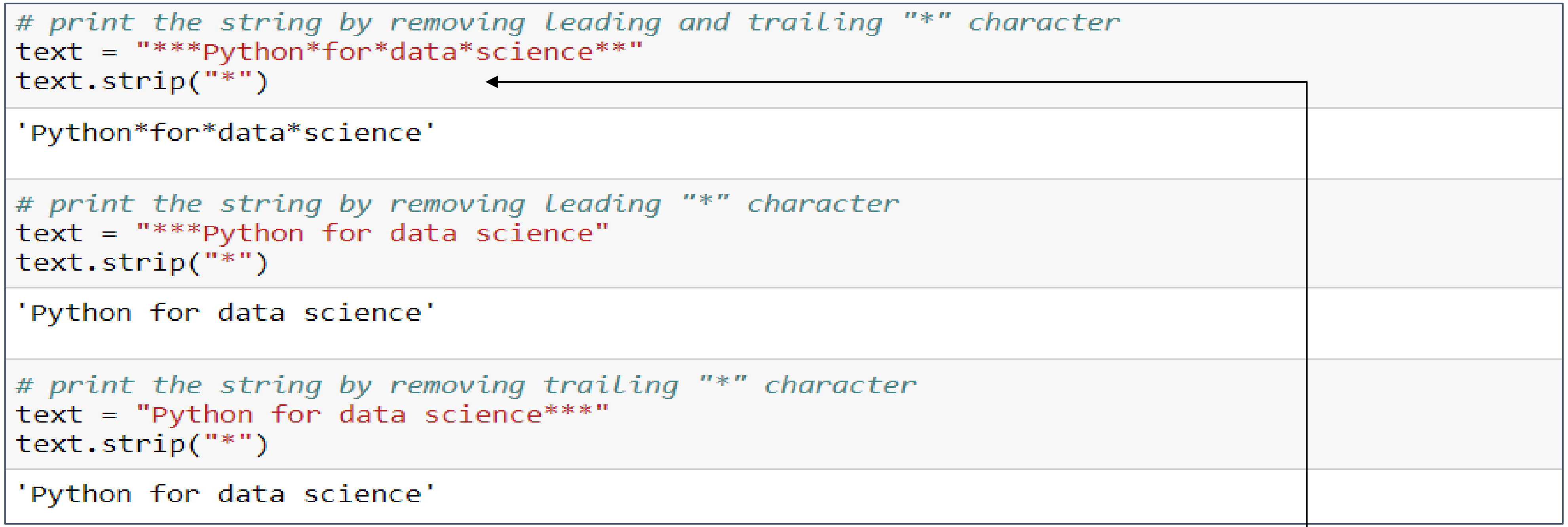
```
# concatenate each element of date with sep
date = "08","03","2020"
sep = '-'
sep.join(date)
```

```
'08-03-2020'
```

Returns a concatenated string where each character of date is separated with sep string which is ‘-’

STRING HANDLING

Example



```
# print the string by removing leading and trailing "*" character
text = "***Python*for*data*science**"
text.strip("*")
```

```
'Python*for*data*science'
```

```
# print the string by removing leading "*" character
text = "***Python for data science"
text.strip("*")
```

```
'Python for data science'
```

```
# print the string by removing trailing "*" character
text = "Python for data science***"
text.strip("*")
```

```
'Python for data science'
```

Returns a copy of the string with both leading and trailing characters removed (based on the string argument passed)

List

LIST

Heterogeneous

Can have duplicates

Can be changed

Elements are separated by comma

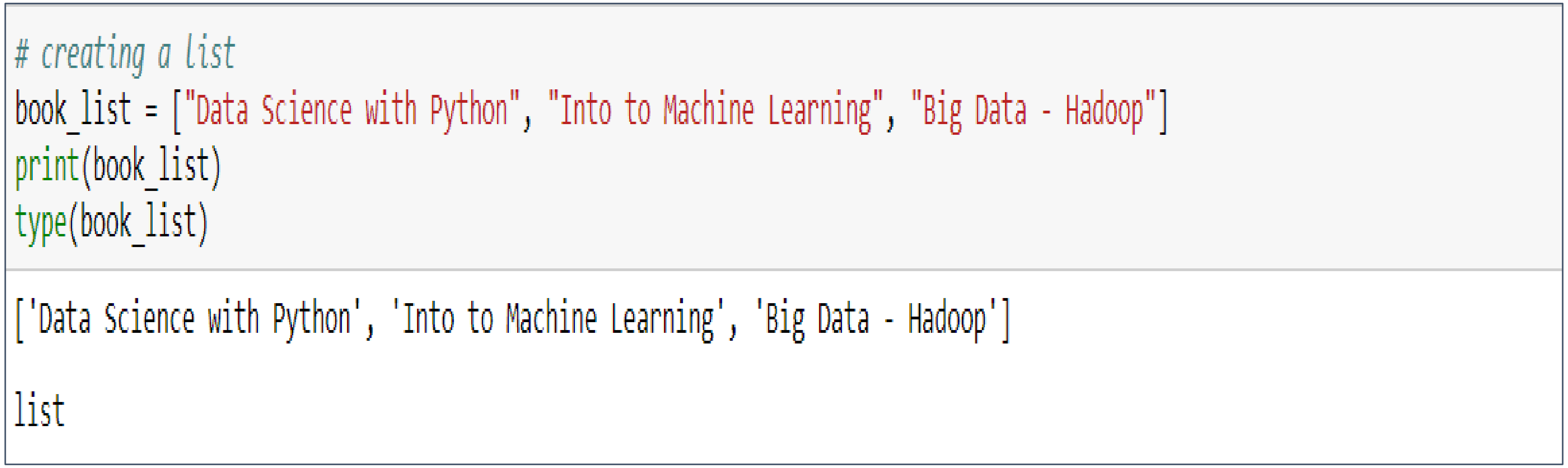
Elements are enclosed in [ ] parenthesis

LIST

Supports the following:

1. Appending Elements
2. Concatenation
3. Indexing and Slicing
4. Min(), Max(), len()
5. in, not in

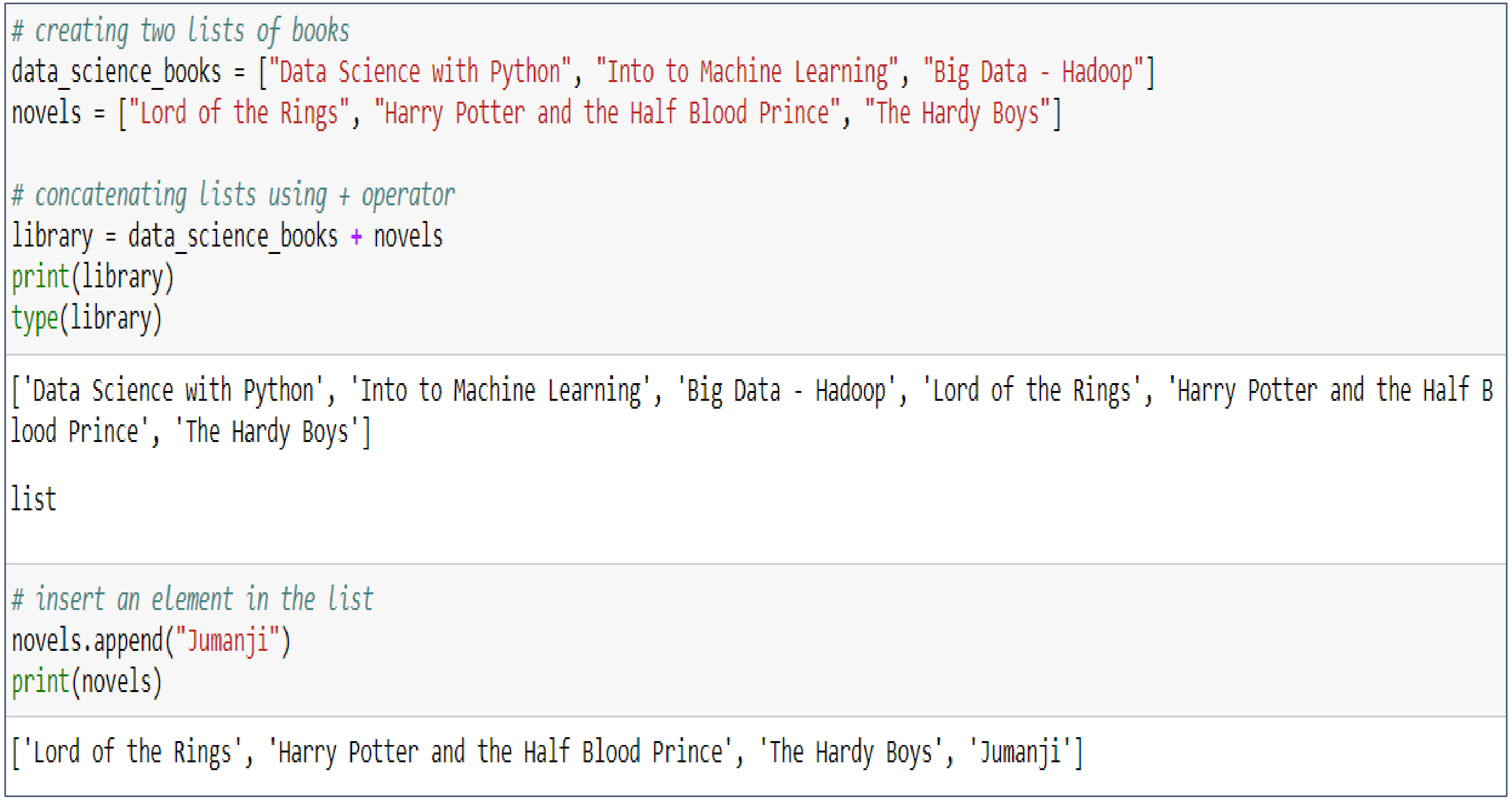
LIST CREATION



```
# creating a list
book_list = ["Data Science with Python", "Into to Machine Learning", "Big Data - Hadoop"]
print(book_list)
type(book_list)
```

```
['Data Science with Python', 'Into to Machine Learning', 'Big Data - Hadoop']

list
```

CONCATENATE AND APPEND



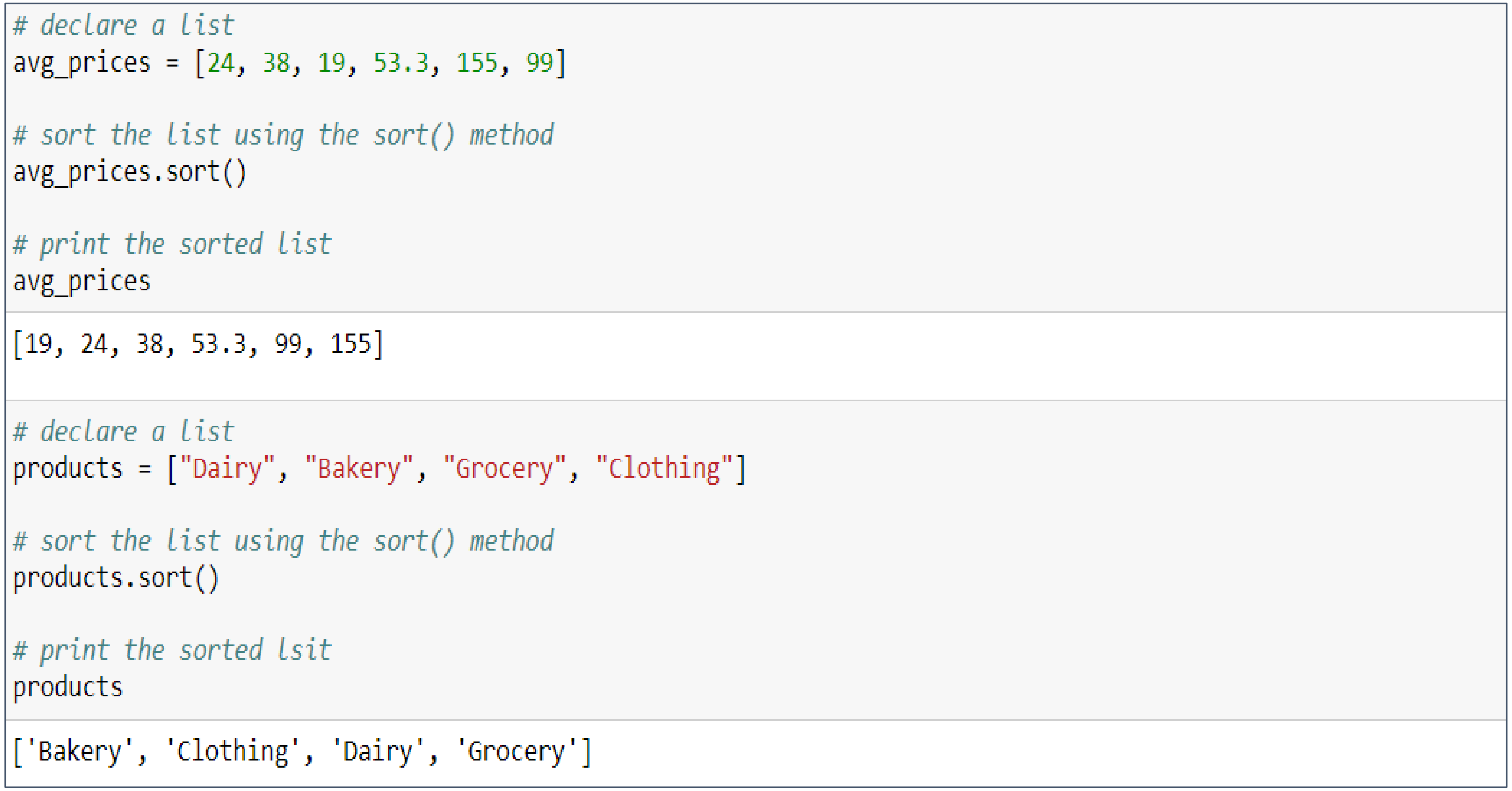
```
# creating two lists of books
data_science_books = ["Data Science with Python", "Into to Machine Learning", "Big Data - Hadoop"]
novels = ["Lord of the Rings", "Harry Potter and the Half Blood Prince", "The Hardy Boys"]

# concatenating lists using + operator
library = data_science_books + novels
print(library)
type(library)
```

```
['Data Science with Python', 'Into to Machine Learning', 'Big Data - Hadoop', 'Lord of the Rings', 'Harry Potter and the Half B
lood Prince', 'The Hardy Boys']

list
```

```
# insert an element in the list
novels.append("Jumanji")
print(novels)
```

```
['Lord of the Rings', 'Harry Potter and the Half Blood Prince', 'The Hardy Boys', 'Jumanji']
```

SORTING ELEMNTS IN THE LIST



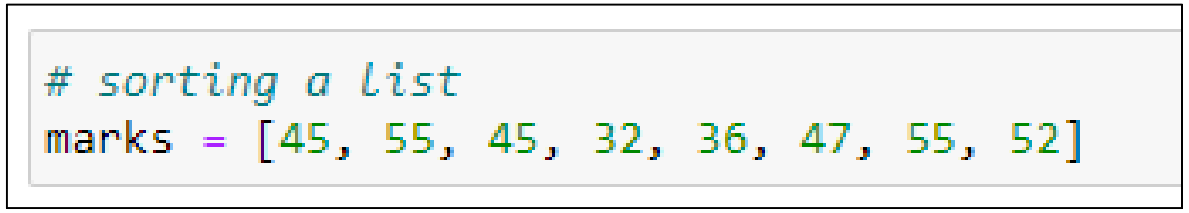
```
# declare a list
avg_prices = [24, 38, 19, 53.3, 155, 99]

# sort the list using the sort() method
avg_prices.sort()

# print the sorted list
avg_prices
```

```
[19, 24, 38, 53.3, 99, 155]
```

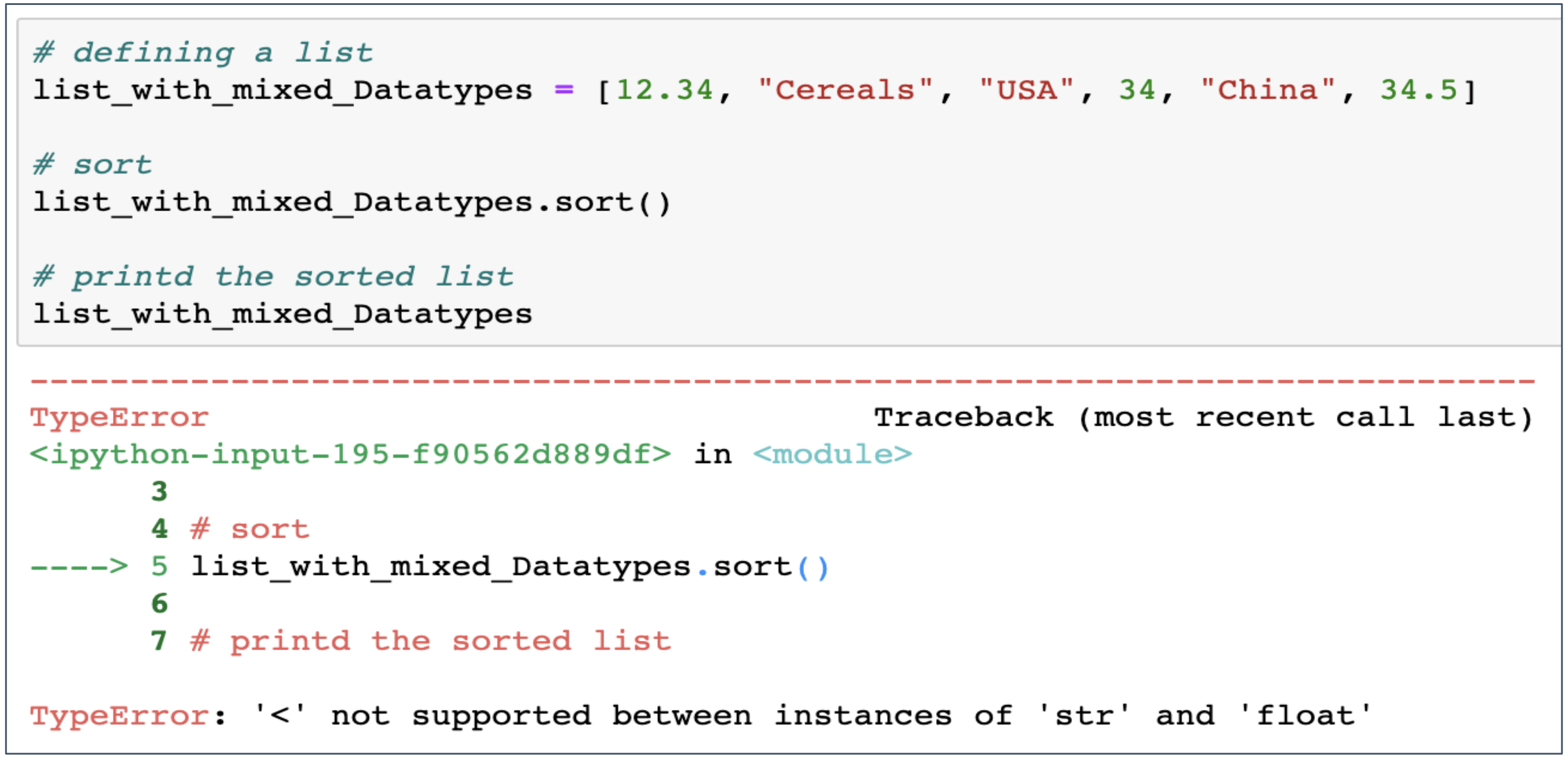
```
# declare a list
products = ["Dairy", "Bakery", "Grocery", "Clothing"]

# sort the list using the sort() method
products.sort()

# print the sorted lsit
products
```

```
['Bakery', 'Clothing', 'Dairy', 'Grocery']
```

- Sort the list in reverse order by passing the parameter ‘reverse = True’ to the sort function
- By default reverse is set to false



```
# sorting a list
marks = [45, 55, 45, 32, 36, 47, 55, 52]
```

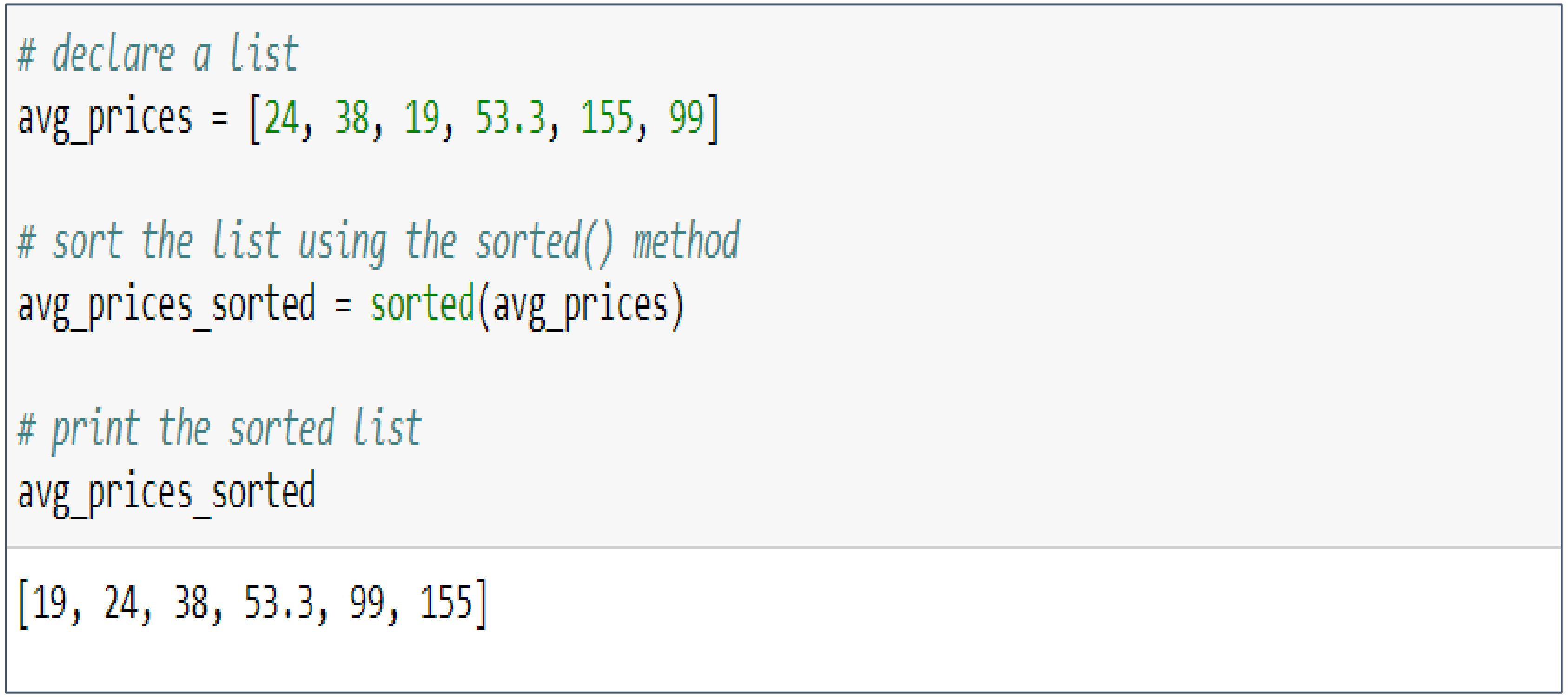


```
# defining a list
list_with_mixed_Datatypes = [12.34, "Cereals", "USA", 34, "China", 34.5]

# sort
list_with_mixed_Datatypes.sort()

# printd the sorted list
list_with_mixed_Datatypes
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-195-f90562d889df> in <module>
      3 
      4 # sort
----> 5 list_with_mixed_Datatypes.sort()
      6 
      7 # printd the sorted list

TypeError: '<' not supported between instances of 'str' and 'float'
```

You cannot sort a list with mixed data types

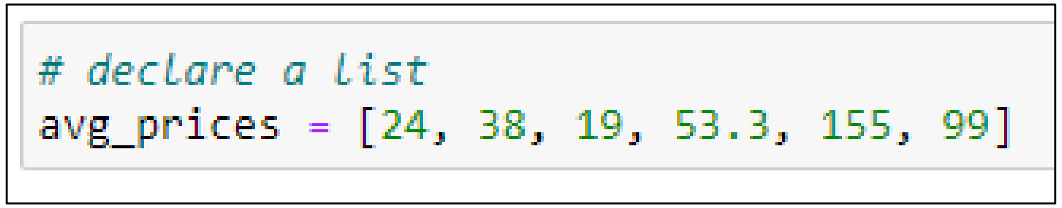
SORTING A LIST USING THE SORTED() FUNCTIONS



```
# declare a list
avg_prices = [24, 38, 19, 53.3, 155, 99]

# sort the list using the sorted() method
avg_prices_sorted = sorted(avg_prices)

# print the sorted list
avg_prices_sorted
```

```
[19, 24, 38, 53.3, 99, 155]
```

Sort the list in reverse order by passing the parameter reverse = True to the sorted function



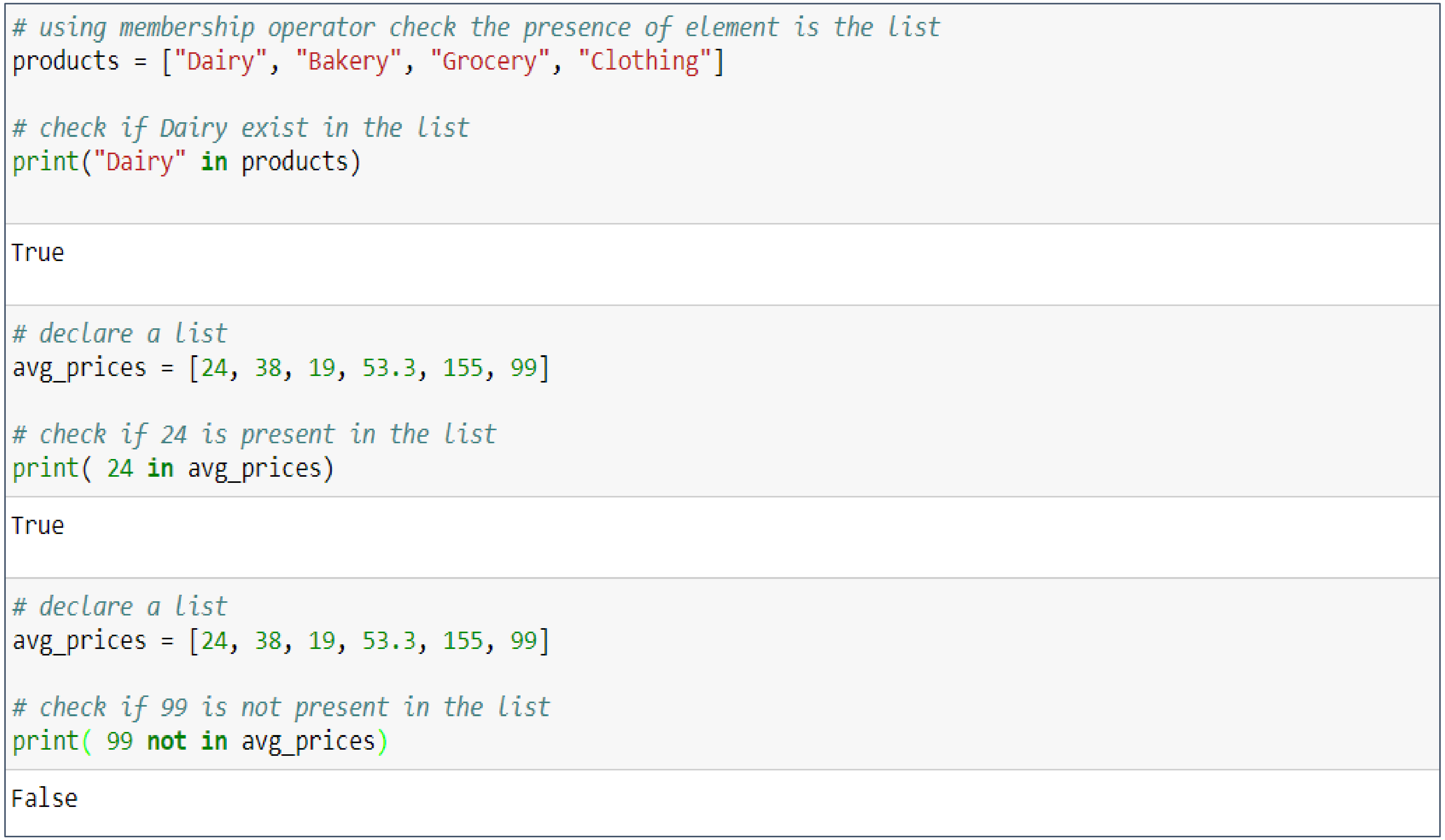
```
# declare a list
avg_prices = [24, 38, 19, 53.3, 155, 99]
```

Did You Know?

What is the difference between sort and sorted?

The sort() will modify an existing list but sorted() will create a new list.

CHECKING PRESENCE OF AN ELEMENT IN THE LIST



```
# using membership operator check the presence of element is the list
products = ["Dairy", "Bakery", "Grocery", "Clothing"]

# check if Dairy exist in the list
print("Dairy" in products)
```

```
True
```

```
# declare a list
avg_prices = [24, 38, 19, 53.3, 155, 99]

# check if 24 is present in the list
print( 24 in avg_prices)
```

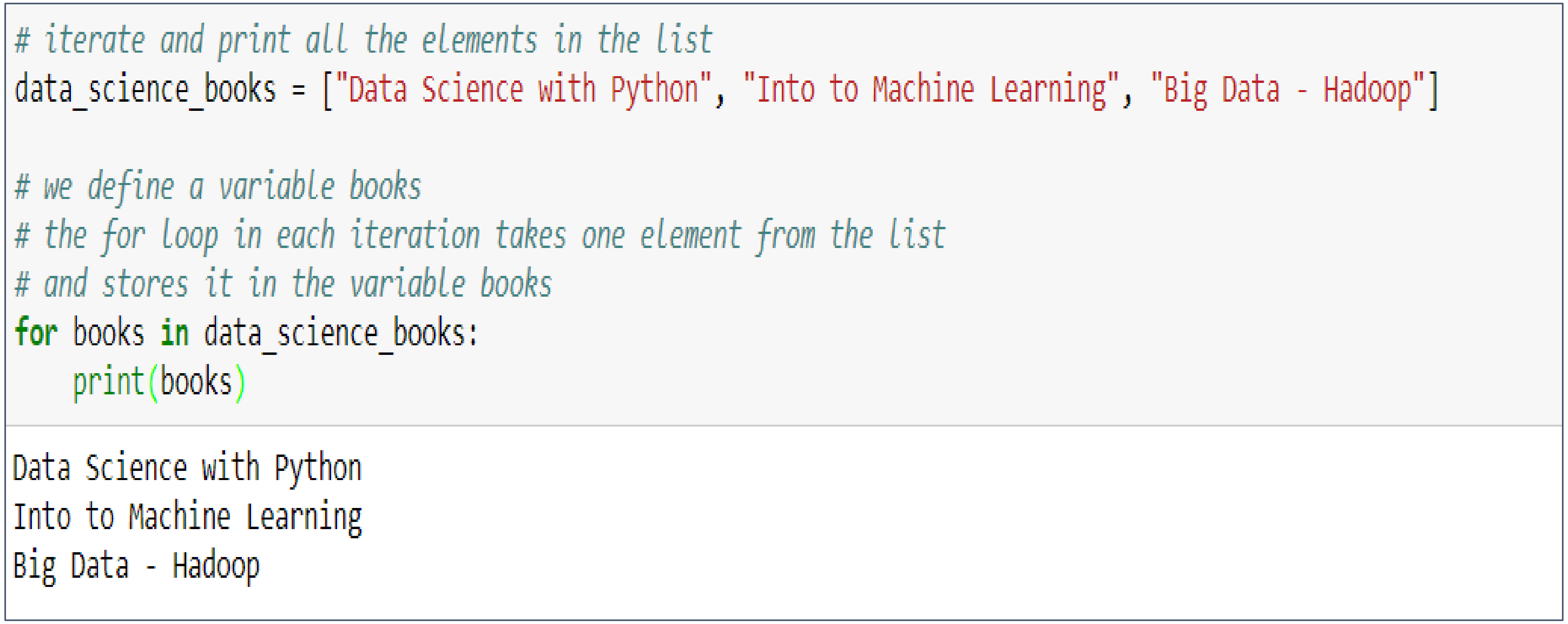
```
True
```

```
# declare a list
avg_prices = [24, 38, 19, 53.3, 155, 99]

# check if 99 is not present in the list
print( 99 not in avg_prices)
```

```
False
```

MORE LIST OPERATIONS: ITERATING THROUGH A LIST

Iterating through all the elements in the list



```
# iterate and print all the elements in the list
data_science_books = ["Data Science with Python", "Into to Machine Learning", "Big Data - Hadoop"]

# we define a variable books
# the for loop in each iteration takes one element from the list
# and stores it in the variable books
for books in data_science_books:
    print(books)
```

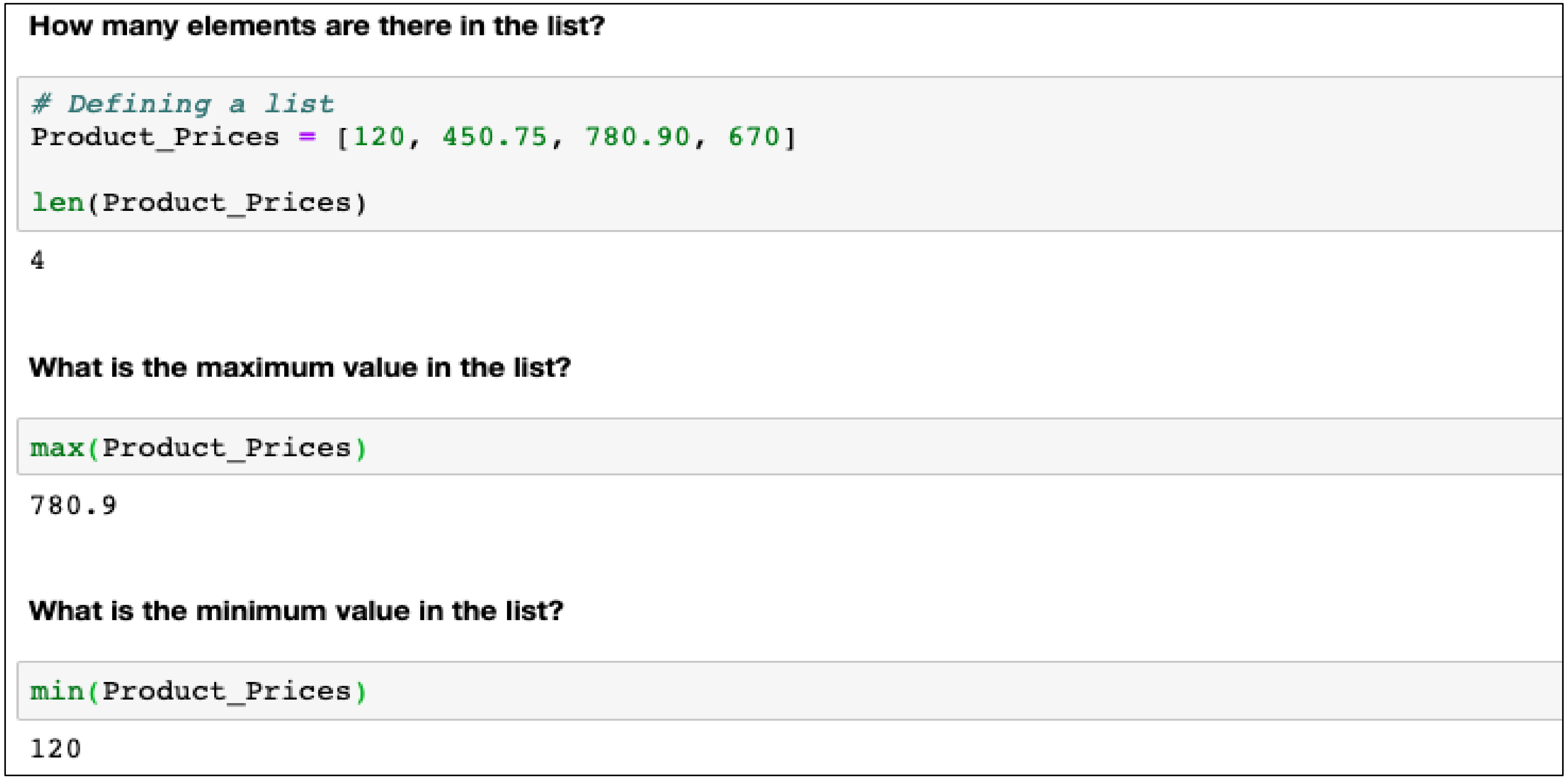
```
Data Science with Python
Into to Machine Learning
Big Data - Hadoop
```

MORE LIST OPERATIONS

We learn more operations that can be performed on a list:

- Finding the number of elements in a list
- Finding the maximum & minimum value of a list with numeric values
- Append a new element to a list
- Remove all elements from a list
- Create a copy of the list
- Count the number of occurrences of a specified element
- Return an index position of an object
- Insert a new element to a list at a specified index position
- Delete an element from a list
- Reverse a list
- Sort a list

MORE LIST OPERATIONS



How many elements are there in the list?

```
# Defining a list
Product_Prices = [120, 450.75, 780.90, 670]

len(Product_Prices)
```

```
4
```

What is the maximum value in the list?

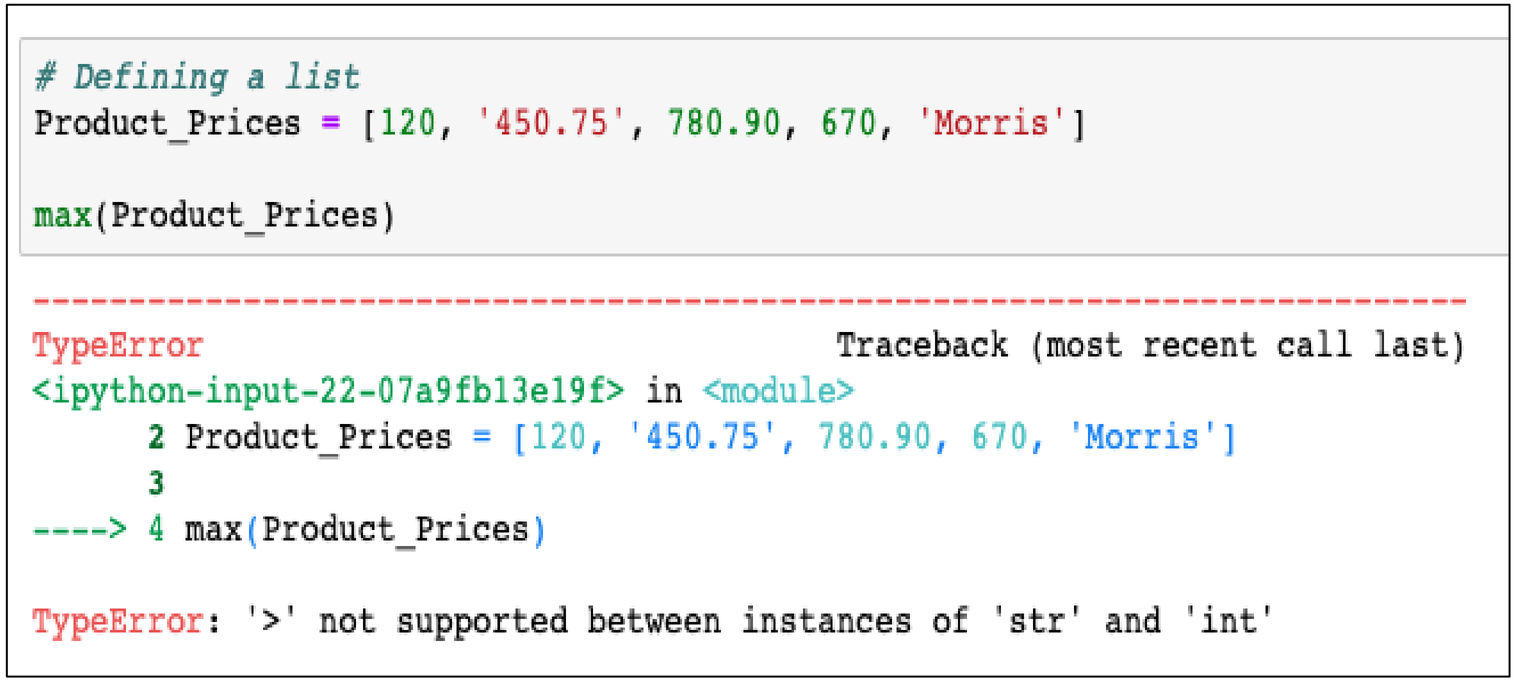
```
max(Product_Prices)
```

```
780.9
```

What is the minimum value in the list?

```
min(Product_Prices)
```

```
120
```

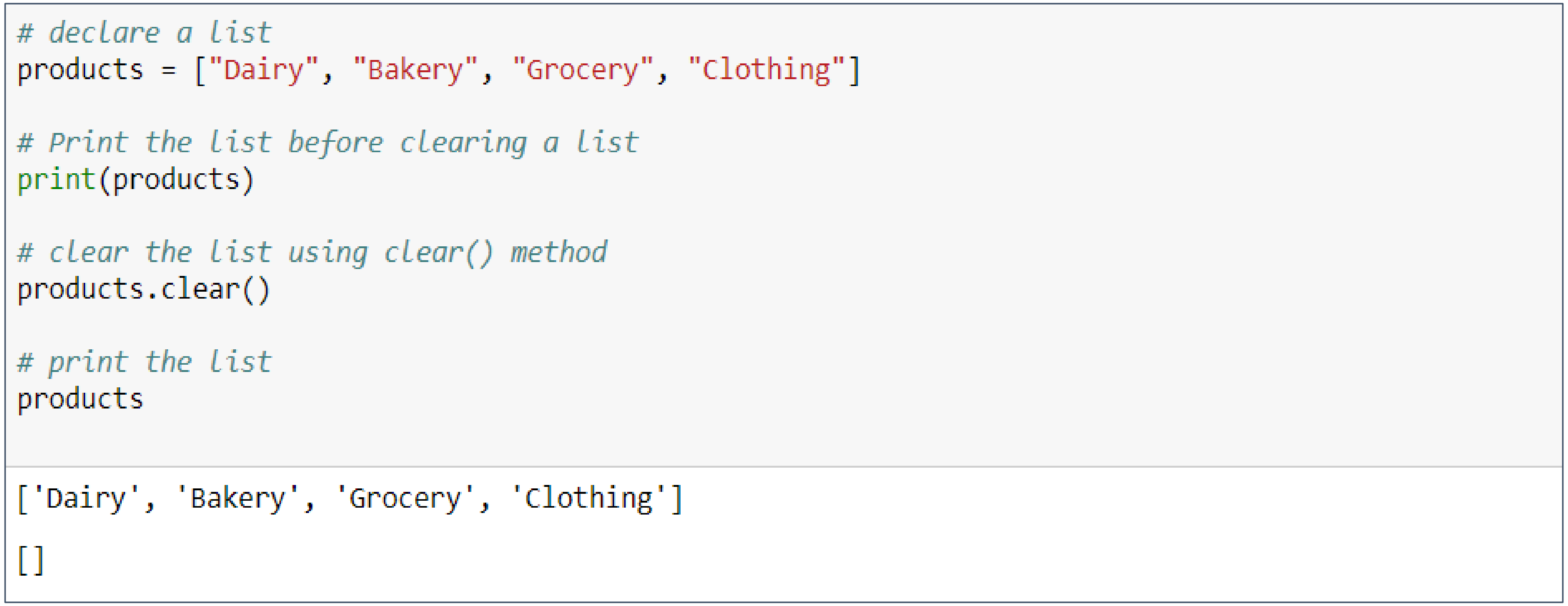


```
# Defining a list
Product_Prices = [120, '450.75', 780.90, 670, 'Morris']

max(Product_Prices)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-07a9fb13e19f> in <module>
      2 Product_Prices = [120, '450.75', 780.90, 670, 'Morris']
      3 
----> 4 max(Product_Prices)

TypeError: '>' not supported between instances of 'str' and 'int'
```

You cannot use max() & min() functions on a list with mixed data types

REMOVING ALL ELEMENTS OF A LIST



```python
# declare a list
products = ["Dairy", "Bakery", "Grocery", "Clothing"]

# Print the list before clearing a list
print(products)

# clear the list using clear() method
products.clear()

# print the list
products
```

```
['Dairy', 'Bakery', 'Grocery', 'Clothing']

[]
```

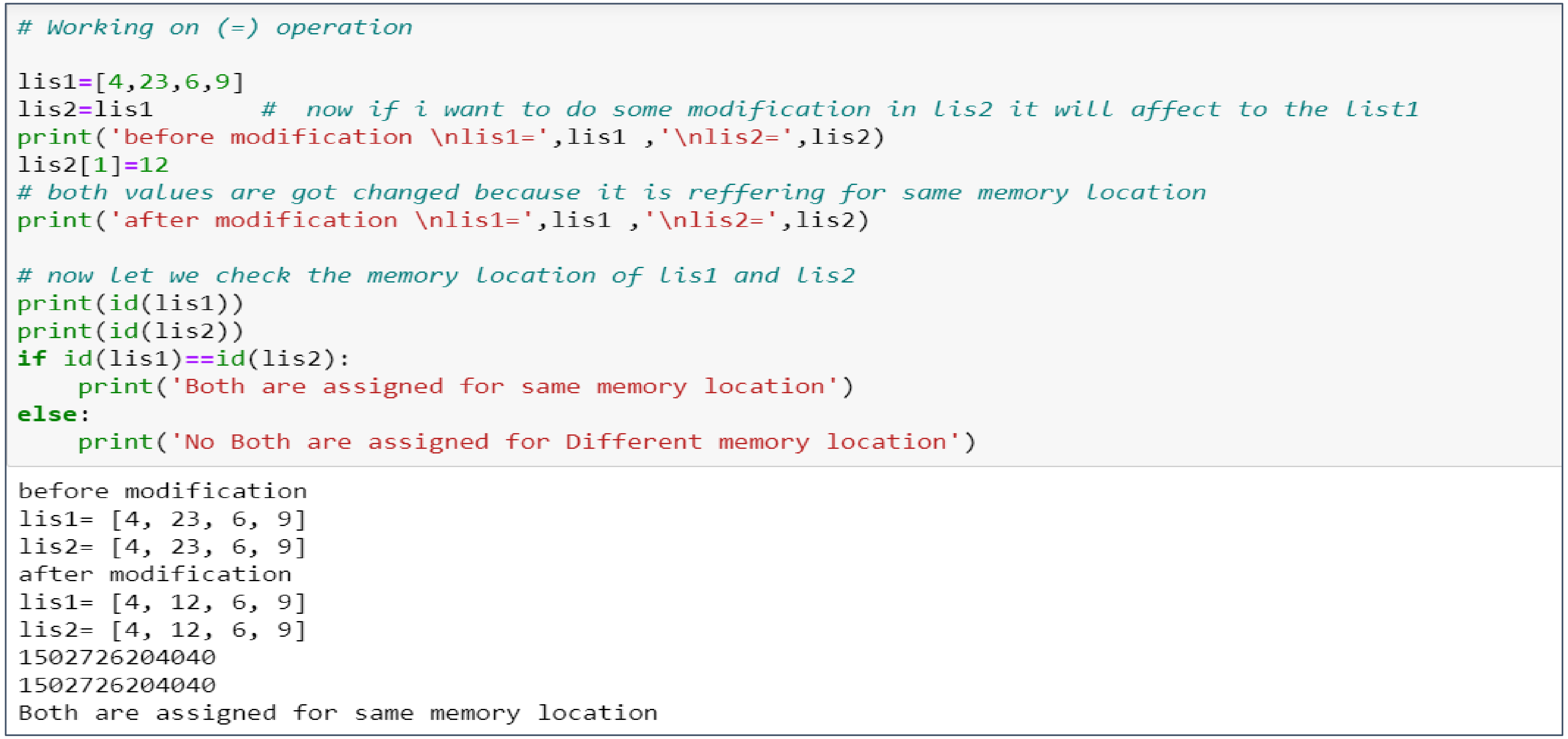
Note that though all elements are removed but the list still exists with no elements

The list products do not have any members now. Can you add new elements to the blank list?

Hint: Use append() function

Introduction to Shallow copy and Deep copy

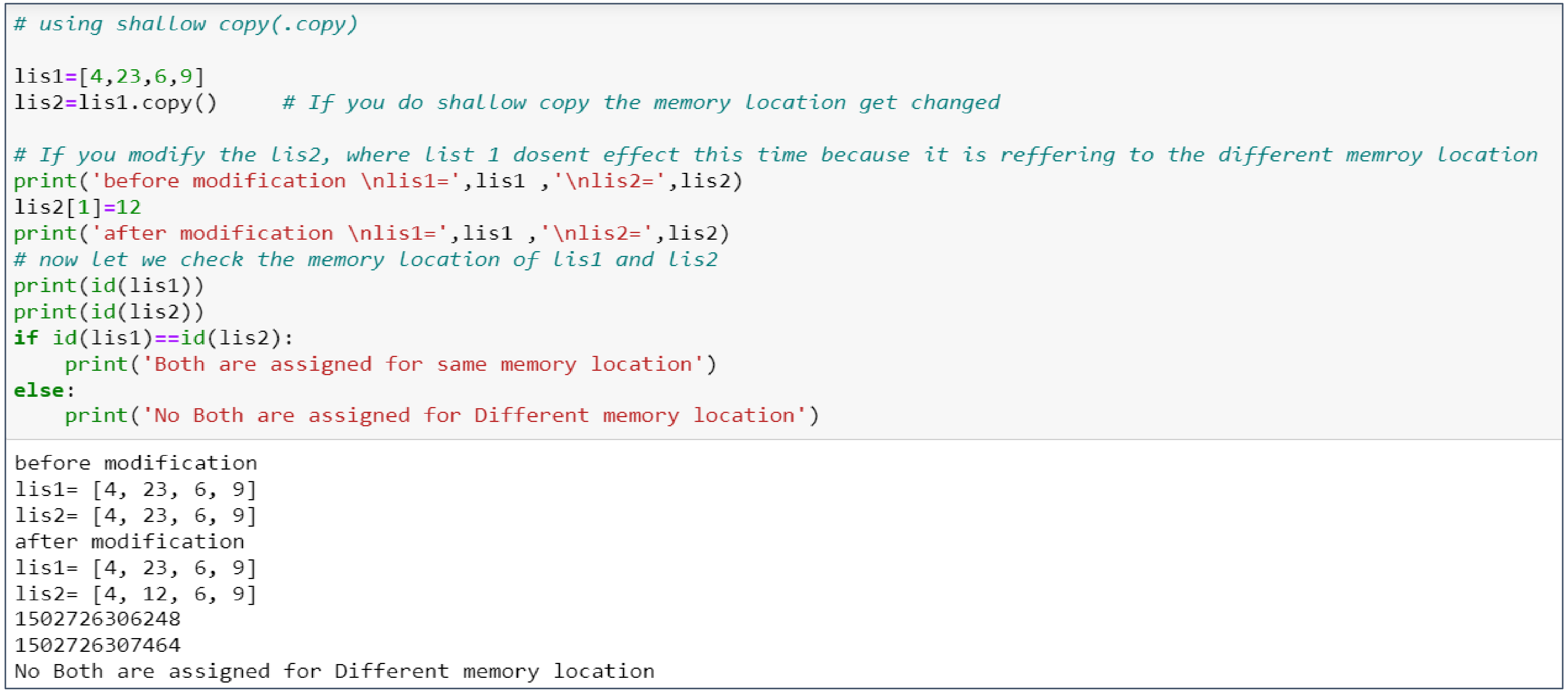
When ever we are going to work on copy the data should be collection because the collection of element is mutable



```
# Working on (=) operation

lis1=[4,23,6,9]
lis2=lis1       #  now if i want to do some modification in lis2 it will affect to the list1
print('before modification \nlis1=',lis1 ,'\nlis2=',lis2)
lis2[1]=12
# both values are got changed because it is reffering for same memory location
print('after modification \nlis1=',lis1 ,'\nlis2=',lis2)

# now let we check the memory location of lis1 and lis2
print(id(lis1))
print(id(lis2))
if id(lis1)==id(lis2):
    print('Both are assigned for same memory location')
else:
    print('No Both are assigned for Different memory location')
```

```
before modification
lis1= [4, 23, 6, 9]
lis2= [4, 23, 6, 9]
after modification
lis1= [4, 12, 6, 9]
lis2= [4, 12, 6, 9]
1502726204040
1502726204040
Both are assigned for same memory location
```

Different between copy and shallow copy

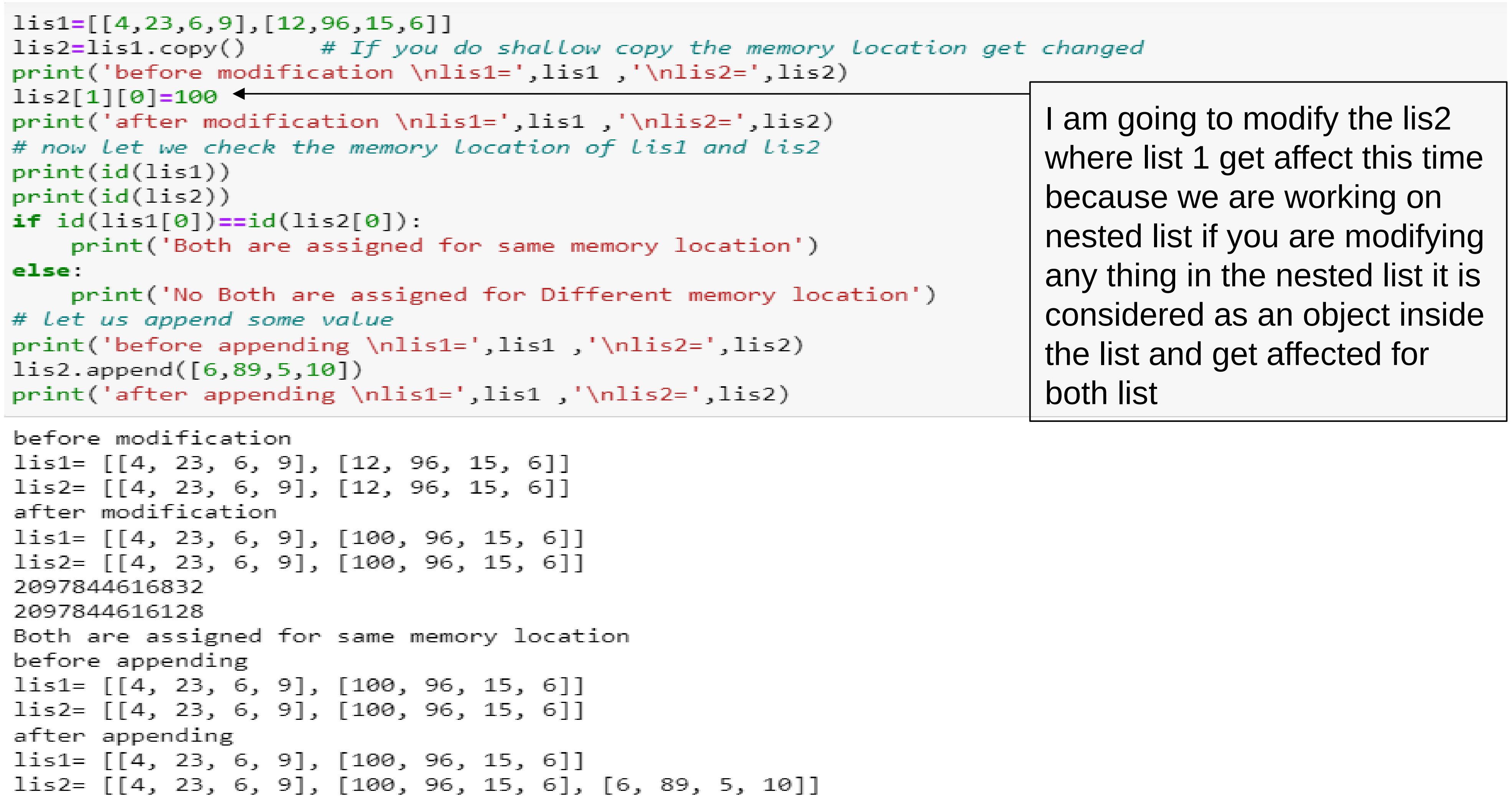


```
# using shallow copy(.copy)

lis1=[4,23,6,9]
lis2=lis1.copy()     # If you do shallow copy the memory location get changed

# If you modify the lis2, where list 1 dosent effect this time because it is reffering to the different memroy location
print('before modification \nlis1=',lis1 ,'\nlis2=',lis2)
lis2[1]=12
print('after modification \nlis1=',lis1 ,'\nlis2=',lis2)
# now let we check the memory location of lis1 and lis2
print(id(lis1))
print(id(lis2))
if id(lis1)==id(lis2):
    print('Both are assigned for same memory location')
else:
    print('No Both are assigned for Different memory location')
```

```
before modification
lis1= [4, 23, 6, 9]
lis2= [4, 23, 6, 9]
after modification
lis1= [4, 23, 6, 9]
lis2= [4, 12, 6, 9]
1502726306248
1502726307464
No Both are assigned for Different memory location
```

Problem in Shallow copy

If we want to work on nested list we cannot use shallow copy because it acts like an copy it store in same memory location



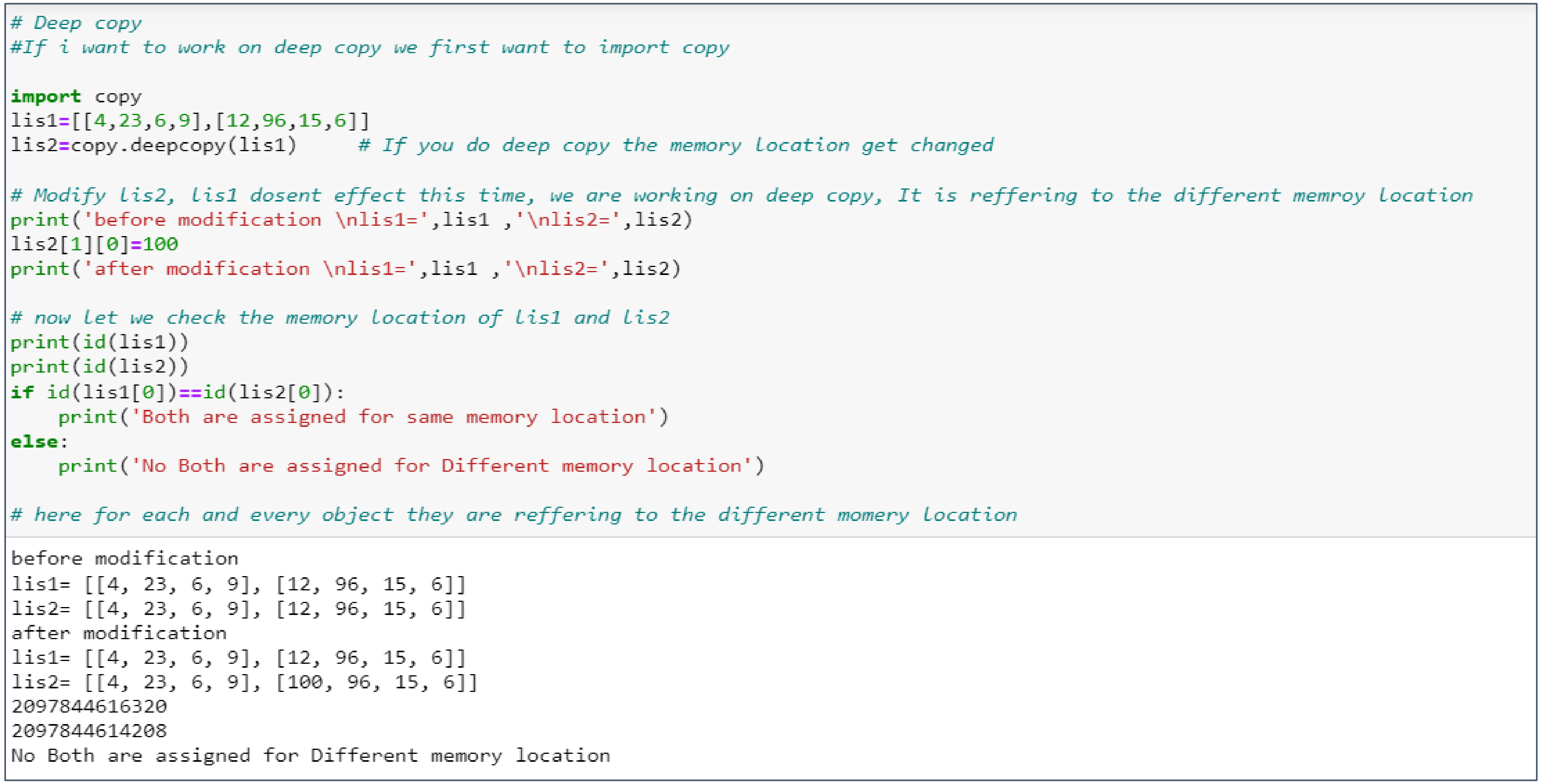
```
lis1=[[4,23,6,9],[12,96,15,6]]
lis2=lis1.copy()     # If you do shallow copy the memory location get changed
print('before modification \nlis1=',lis1 ,'\nlis2=',lis2)
lis2[1][0]=100
print('after modification \nlis1=',lis1 ,'\nlis2=',lis2)
# now let we check the memory location of lis1 and lis2
print(id(lis1))
print(id(lis2))
if id(lis1[0])==id(lis2[0]):
    print('Both are assigned for same memory location')
else:
    print('No Both are assigned for Different memory location')
# let us append some value
print('before appending \nlis1=',lis1 ,'\nlis2=',lis2)
lis2.append([6,89,5,10])
print('after appending \nlis1=',lis1 ,'\nlis2=',lis2)
```

```
before modification
lis1= [[4, 23, 6, 9], [12, 96, 15, 6]]
lis2= [[4, 23, 6, 9], [12, 96, 15, 6]]
after modification
lis1= [[4, 23, 6, 9], [100, 96, 15, 6]]
lis2= [[4, 23, 6, 9], [100, 96, 15, 6]]
2097844616832
2097844616128
Both are assigned for same memory location
before appending
lis1= [[4, 23, 6, 9], [100, 96, 15, 6]]
lis2= [[4, 23, 6, 9], [100, 96, 15, 6]]
after appending
lis1= [[4, 23, 6, 9], [100, 96, 15, 6]]
lis2= [[4, 23, 6, 9], [100, 96, 15, 6], [6, 89, 5, 10]]
```

I am going to modify the lis2 where list 1 get affect this time because we are working on nested list if you are modifying any thing in the nested list it is considered as an object inside the list and get affected for both list

Deep copy

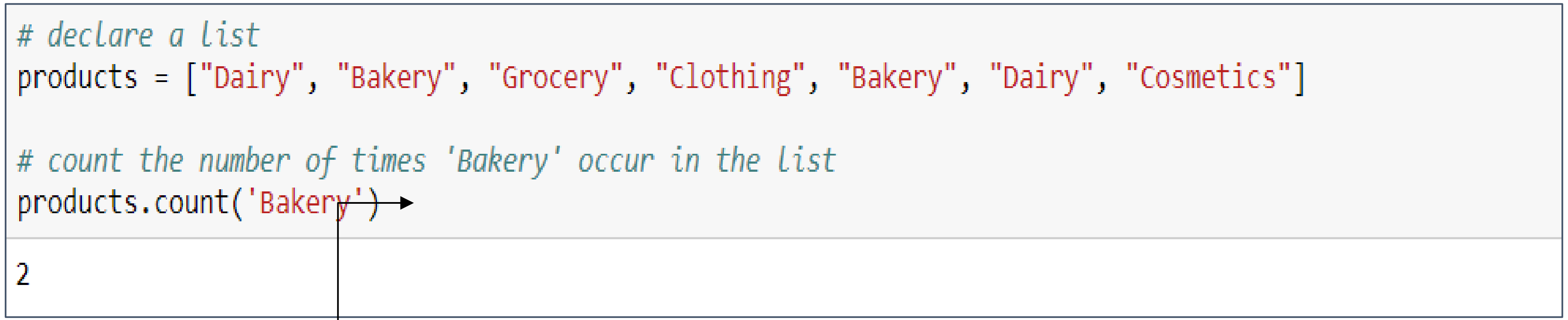
In Deep copy every object referring to the different memory location so we should use deep copy in nested list



```
# Deep copy
#If i want to work on deep copy we first want to import copy

import copy
lis1=[[4,23,6,9],[12,96,15,6]]
lis2=copy.deepcopy(lis1)     # If you do deep copy the memory location get changed

# Modify lis2, lis1 dosent effect this time, we are working on deep copy, It is reffering to the different memroy location
print('before modification \nlis1=',lis1 ,'\nlis2=',lis2)
lis2[1][0]=100
print('after modification \nlis1=',lis1 ,'\nlis2=',lis2)

# now let we check the memory location of lis1 and lis2
print(id(lis1))
print(id(lis2))
if id(lis1[0])==id(lis2[0]):
    print('Both are assigned for same memory location')
else:
    print('No Both are assigned for Different memory location')

# here for each and every object they are reffering to the different momery location
```

```
before modification
lis1= [[4, 23, 6, 9], [12, 96, 15, 6]]
lis2= [[4, 23, 6, 9], [12, 96, 15, 6]]
after modification
lis1= [[4, 23, 6, 9], [12, 96, 15, 6]]
lis2= [[4, 23, 6, 9], [100, 96, 15, 6]]
2097844616320
2097844614208
No Both are assigned for Different memory location
```

COUNT THE NUMBER OF OCCURRENCES OF AN ELEMENT



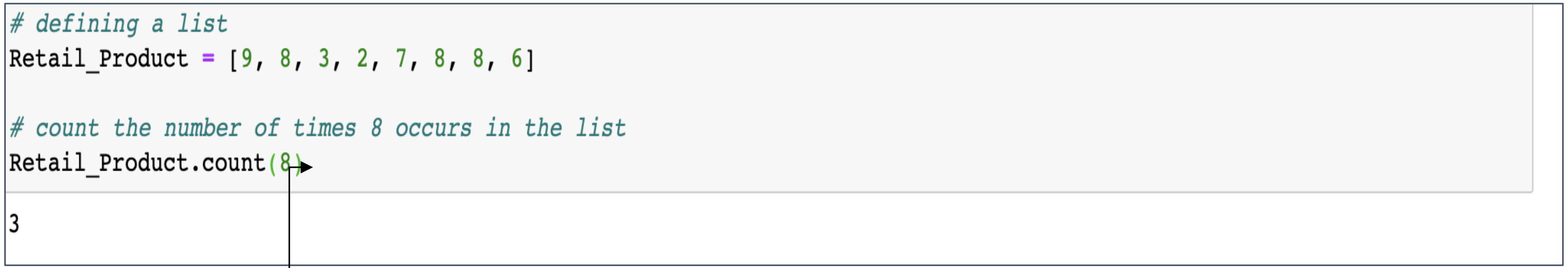
```
# declare a list
products = ["Dairy", "Bakery", "Grocery", "Clothing", "Bakery", "Dairy", "Cosmetics"]

# count the number of times 'Bakery' occur in the list
products.count('Bakery')
```

```
2
```

Pass the element as an input parameter to the count() function that returns the count how many times ‘Bakery’ appear in the list

COUNT THE NUMBER OF OCCURRENCES OF AN ELEMENT

The count() method works on numeric elements as well

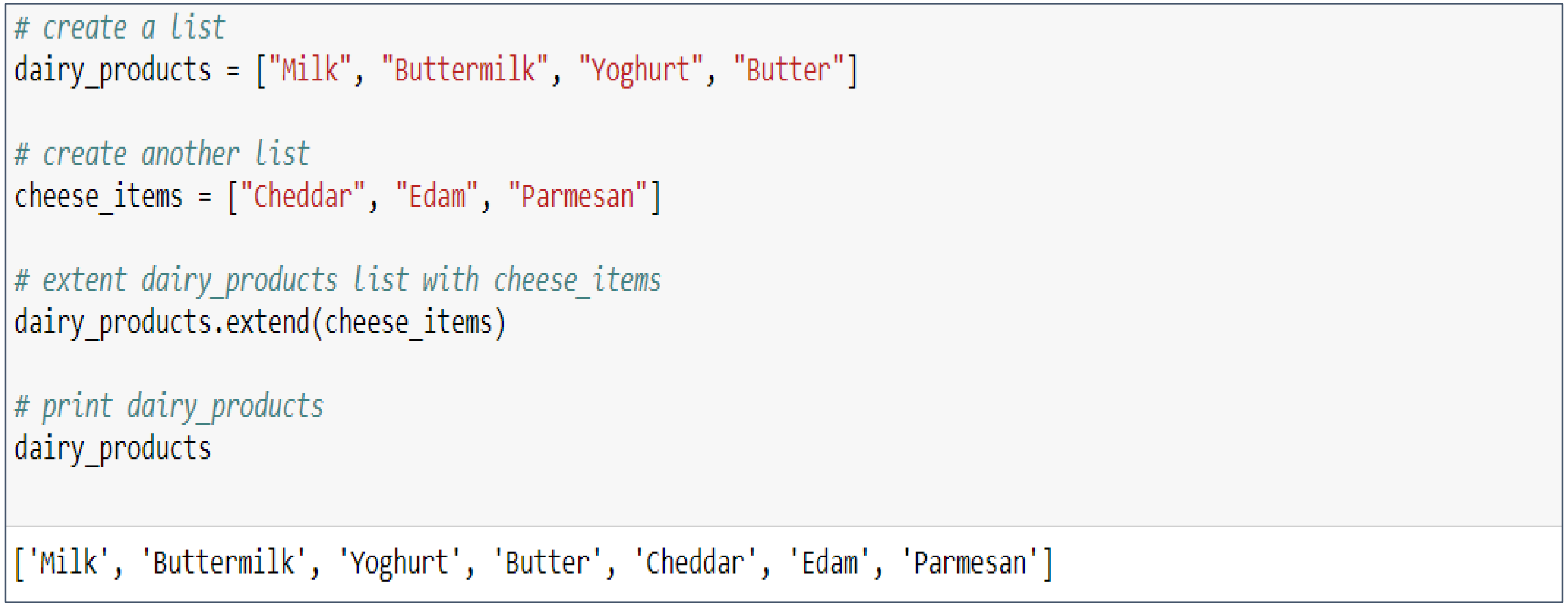


```
# defining a list
Retail_Product = [9, 8, 3, 2, 7, 8, 8, 6]

# count the number of times 8 occurs in the list
Retail_Product.count(8)

3
```

Pass the element as an input parameter to the count() function and it will return the count of how many times 8 appear in the list

EXTENDING A LIST

- The extend() extends the current list by adding all items of the the specified list (passed as an argument) to an end

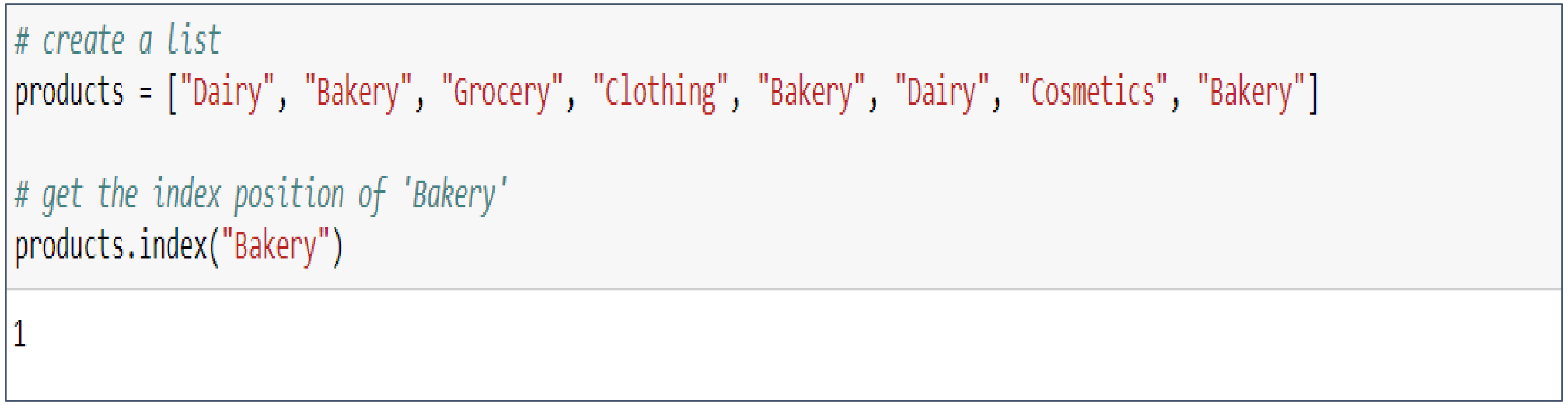


```
# create a list
dairy_products = ["Milk", "Buttermilk", "Yoghurt", "Butter"]

# create another list
cheese_items = ["Cheddar", "Edam", "Parmesan"]

# extent dairy_products list with cheese_items
dairy_products.extend(cheese_items)

# print dairy_products
dairy_products
```

```
['Milk', 'Buttermilk', 'Yoghurt', 'Butter', 'Cheddar', 'Edam', 'Parmesan']
```

RETURNING THE INDEX POSITION OF AN OBJECT

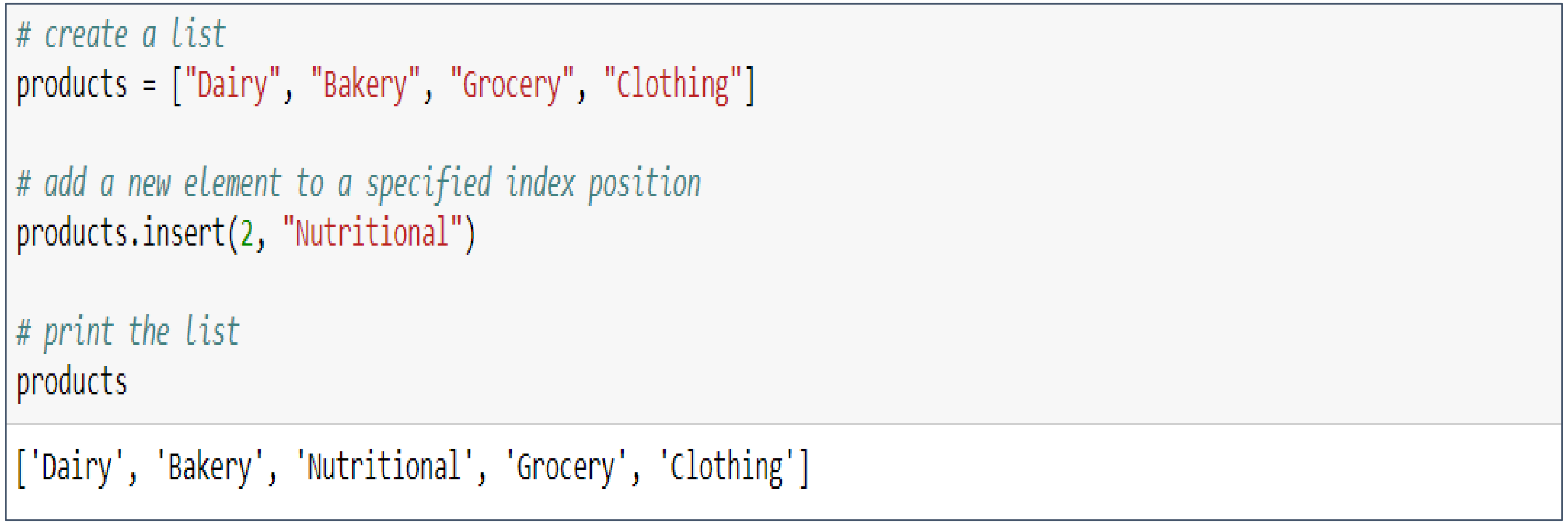
This returns the lowest index position in case of multiple occurrence



```
# create a list
products = ["Dairy", "Bakery", "Grocery", "Clothing", "Bakery", "Dairy", "Cosmetics", "Bakery"]

# get the index position of 'Bakery'
products.index("Bakery")
```

```
1
```

INSERT A NEW ELEMENT AT A SPECIFIED INDEX POSITION



```
# create a list
products = ["Dairy", "Bakery", "Grocery", "Clothing"]

# add a new element to a specified index position
products.insert(2, "Nutritional")

# print the list
products
```

```
['Dairy', 'Bakery', 'Nutritional', 'Grocery', 'Clothing']
```

REMOVE ELEMENT FORM A LIST

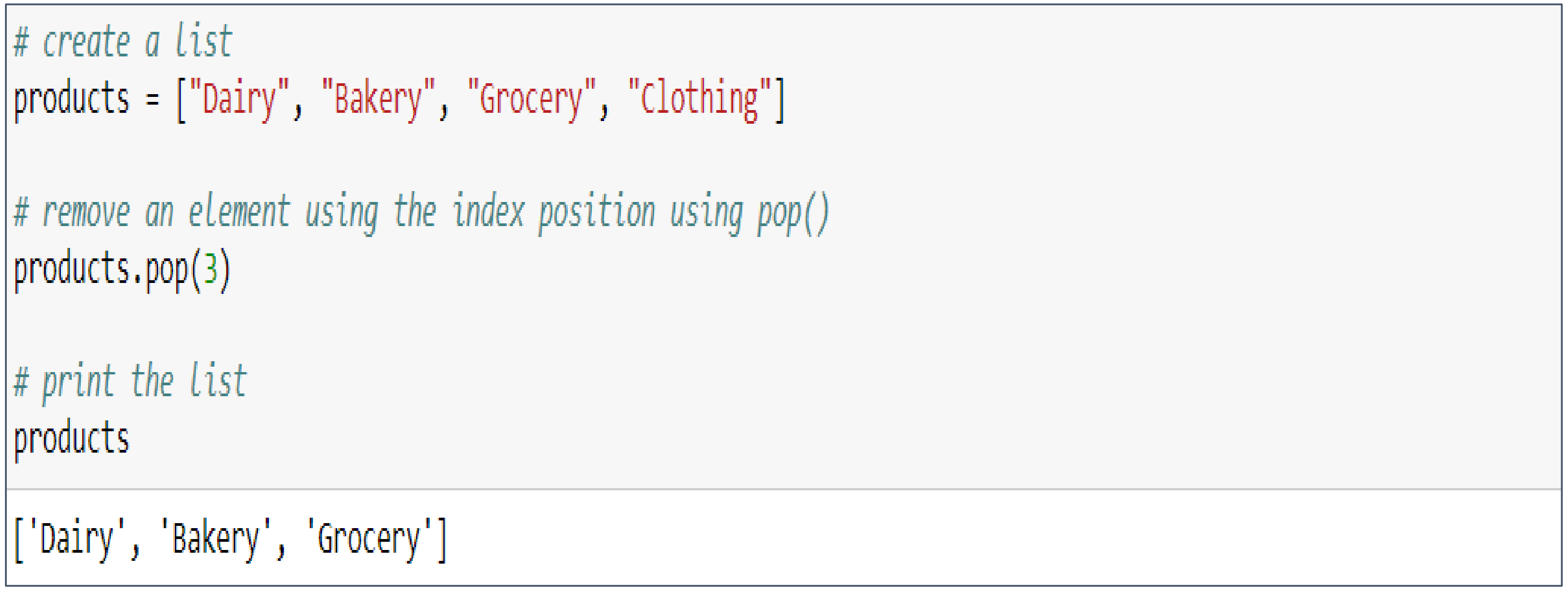
There are two ways to remove the elements from the list:

1. Using pop() method, where the elements are removed by passing their index position

2. By using the remove() method, where the elements are removed by passing the element name

REMOVE ELEMENT FORM A LIST

Using pop() method

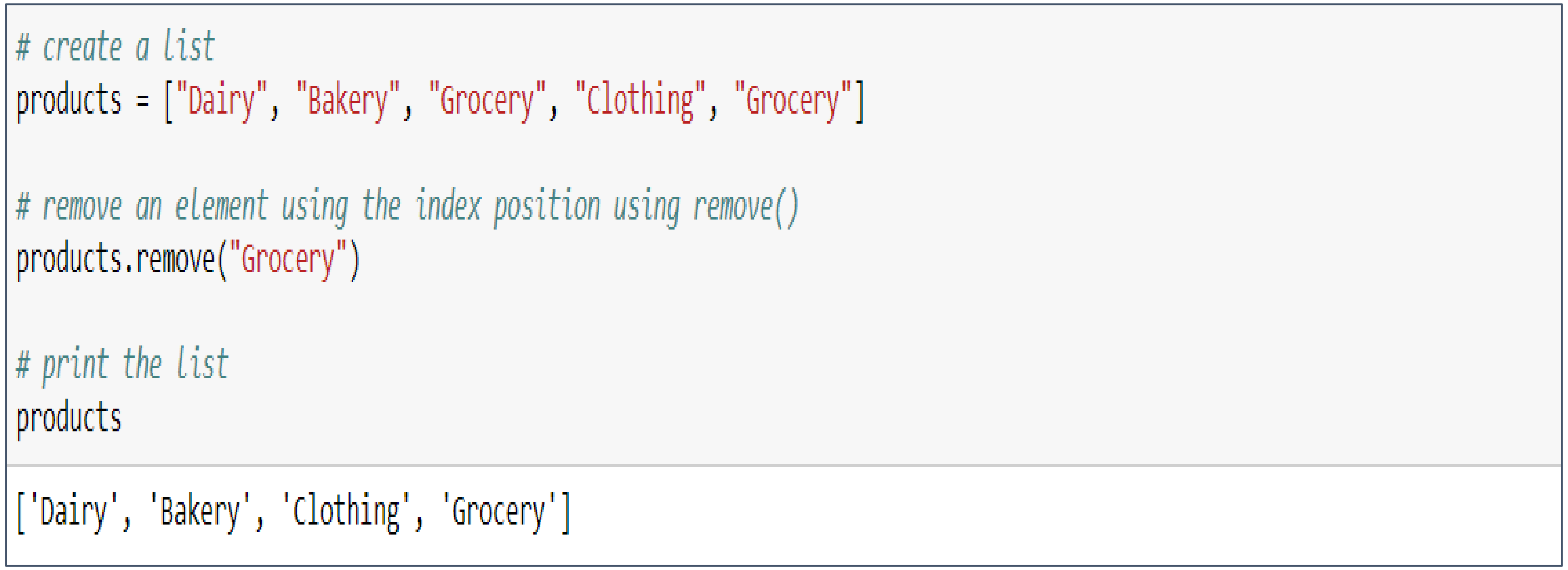


```
# create a list
products = ["Dairy", "Bakery", "Grocery", "Clothing"]

# remove an element using the index position using pop()
products.pop(3)

# print the list
products
```

```
['Dairy', 'Bakery', 'Grocery']
```

REMOVE ELEMENT FORM A LIST

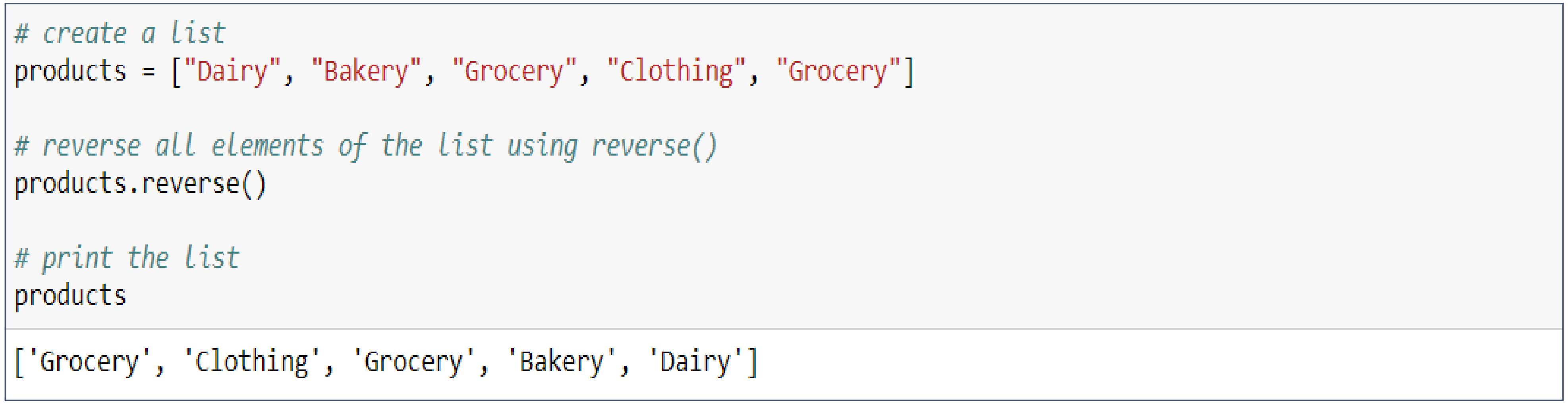
Using remove() method



```
# create a list
products = ["Dairy", "Bakery", "Grocery", "Clothing", "Grocery"]

# remove an element using the index position using remove()
products.remove("Grocery")

# print the list
products
```

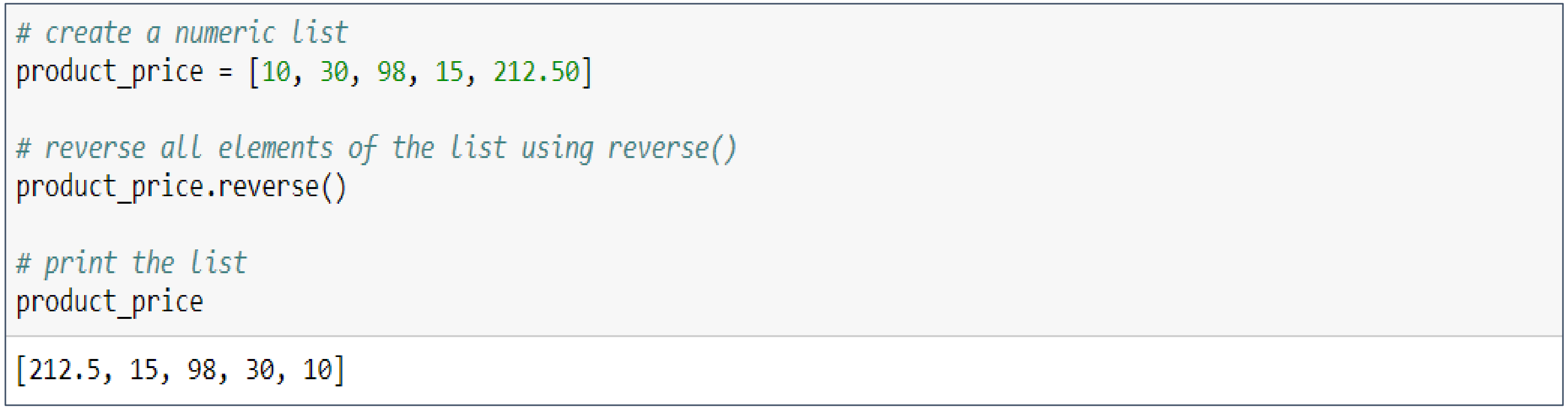
```
['Dairy', 'Bakery', 'Clothing', 'Grocery']
```

REVERSING A LIST



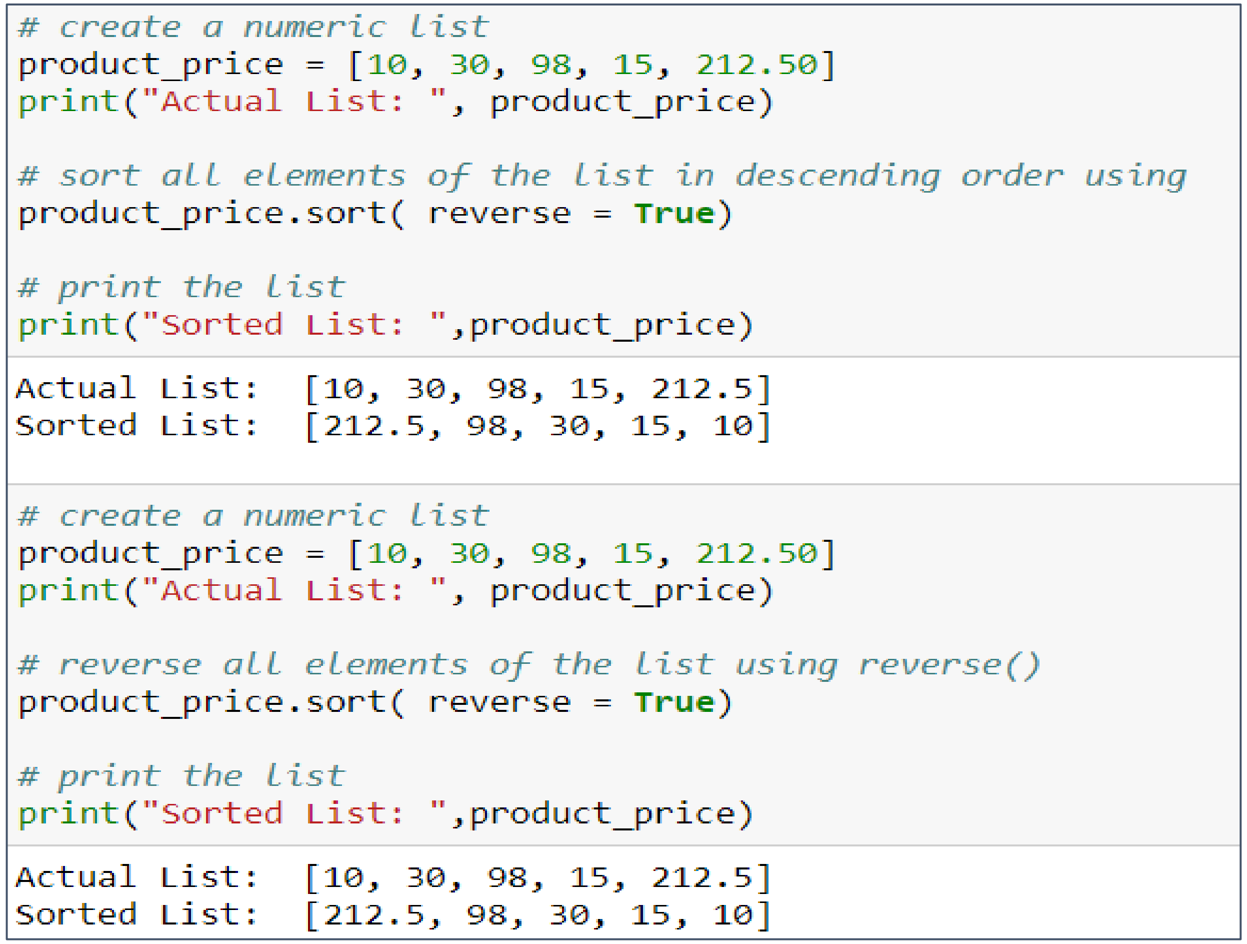
```python
# create a list
products = ["Dairy", "Bakery", "Grocery", "Clothing", "Grocery"]

# reverse all elements of the list using reverse()
products.reverse()

# print the list
products
```

```
['Grocery', 'Clothing', 'Grocery', 'Bakery', 'Dairy']
```



```python
# create a numeric list
product_price = [10, 30, 98, 15, 212.50]

# reverse all elements of the list using reverse()
product_price.reverse()

# print the list
product_price
```

```
[212.5, 15, 98, 30, 10]
```



```
# create a numeric list
product_price = [10, 30, 98, 15, 212.50]
print("Actual List: ", product_price)

# sort all elements of the list in descending order using
product_price.sort( reverse = True)

# print the list
print("Sorted List: ",product_price)
```

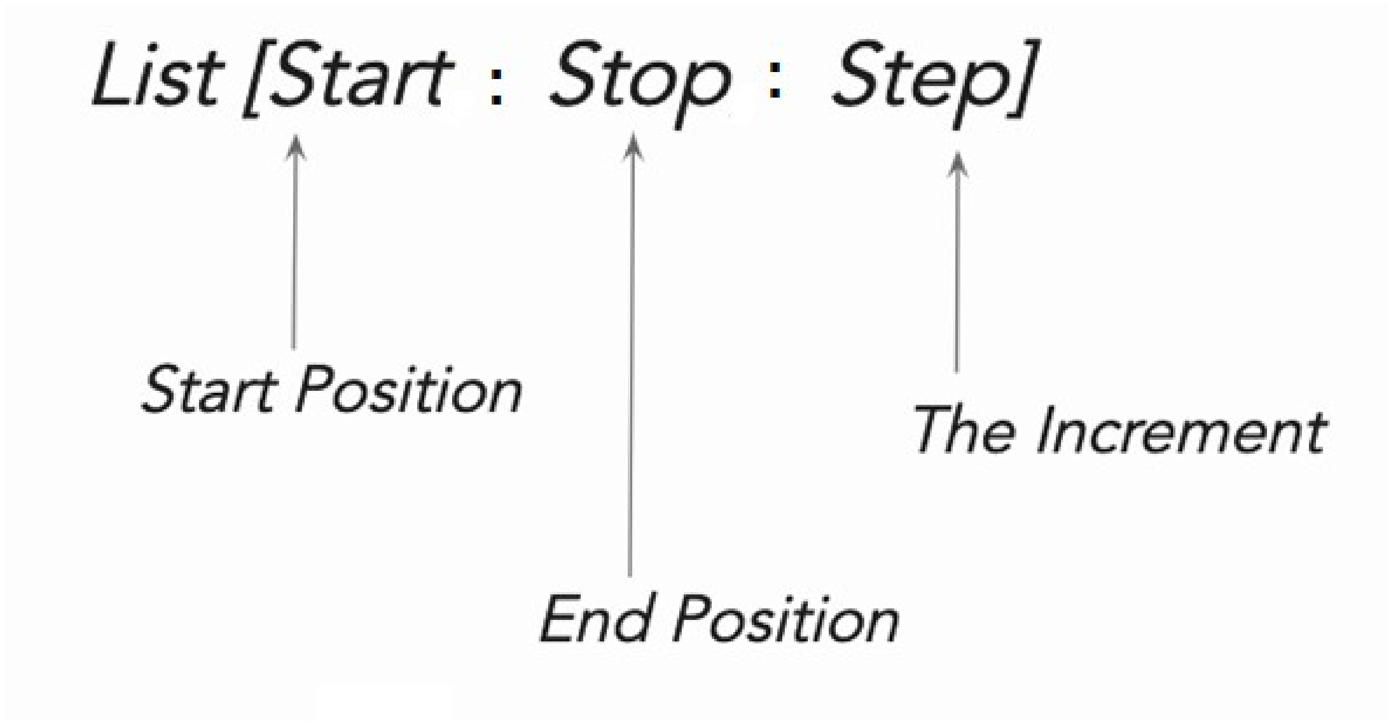
```
Actual List:  [10, 30, 98, 15, 212.5]
Sorted List:  [212.5, 98, 30, 15, 10]
```

```
# create a numeric list
product_price = [10, 30, 98, 15, 212.50]
print("Actual List: ", product_price)

# reverse all elements of the list using reverse()
product_price.sort( reverse = True)

# print the list
print("Sorted List: ",product_price)
```

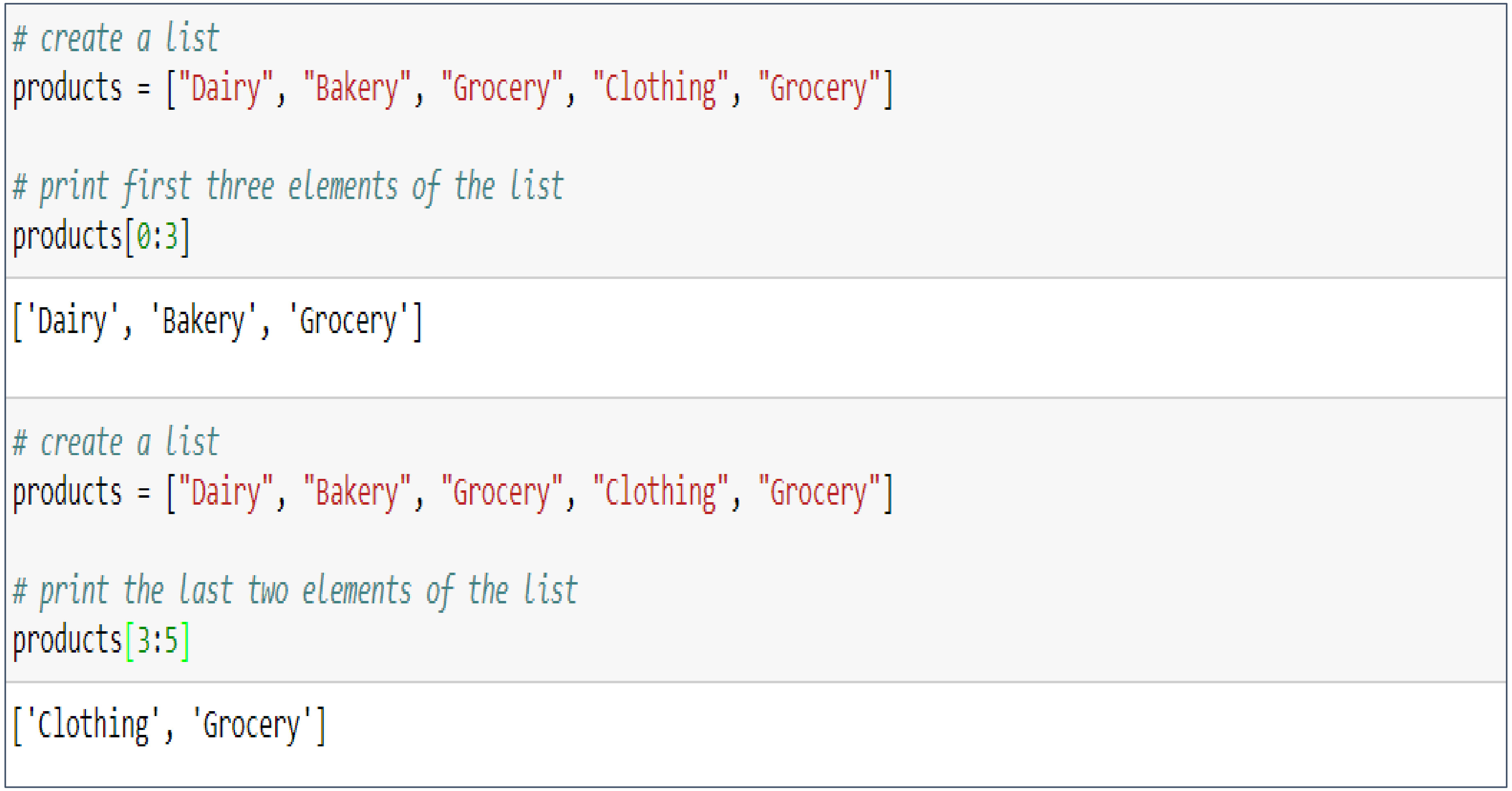
```
Actual List:  [10, 30, 98, 15, 212.5]
Sorted List:  [212.5, 98, 30, 15, 10]
```

The list reverse() method and the parameter reverse in the sort() method are two different operations

INDEXING AND SLICING LIST



SLICE A LIST– BASIC EXAMPLE



```
# create a list
products = ["Dairy", "Bakery", "Grocery", "Clothing", "Grocery"]

# print first three elements of the list
products[0:3]
```

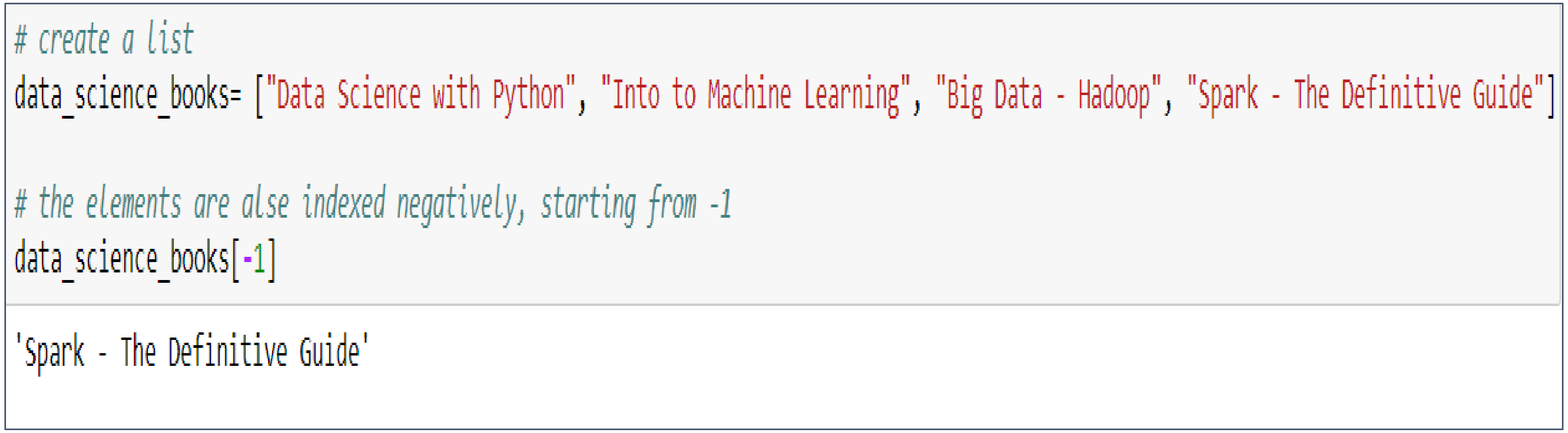
```
['Dairy', 'Bakery', 'Grocery']
```

```
# create a list
products = ["Dairy", "Bakery", "Grocery", "Clothing", "Grocery"]

# print the last two elements of the list
products[3:5]
```

```
['Clothing', 'Grocery']
```

SLICE A LIST USING NEGATIVE INDEX

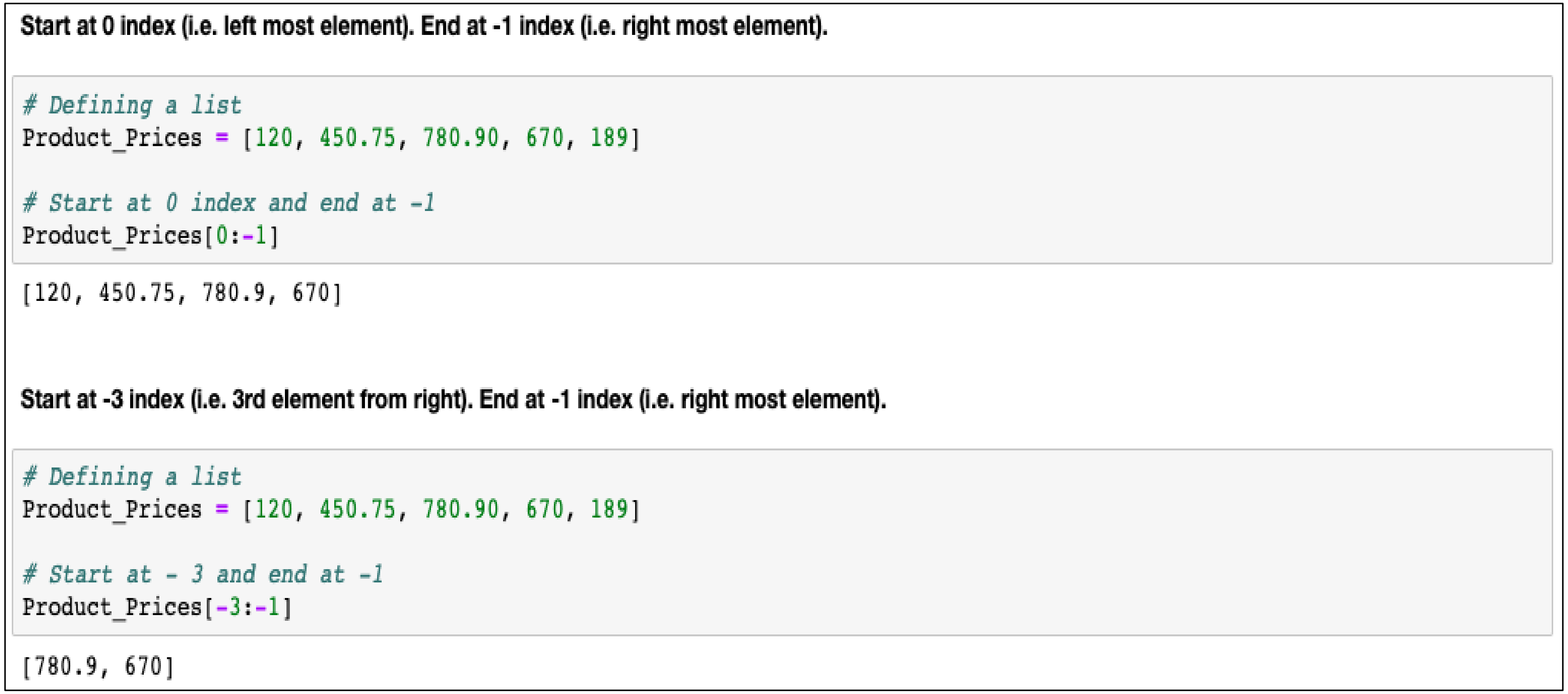
- With negative index the elements are selected from right hand side of the list. The index of the right most element is -1



```
# create a list
data_science_books= ["Data Science with Python", "Into to Machine Learning", "Big Data - Hadoop", "Spark - The Definitive Guide"]

# the elements are alse indexed negatively, starting from -1
data_science_books[-1]
```

```
'Spark - The Definitive Guide'
```

SLICE A LIST USING NEGATIVE INDEX



Start at 0 index (i.e. left most element). End at -1 index (i.e. right most element).

```
# Defining a list
Product_Prices = [120, 450.75, 780.90, 670, 189]

# Start at 0 index and end at -1
Product_Prices[0:-1]
```

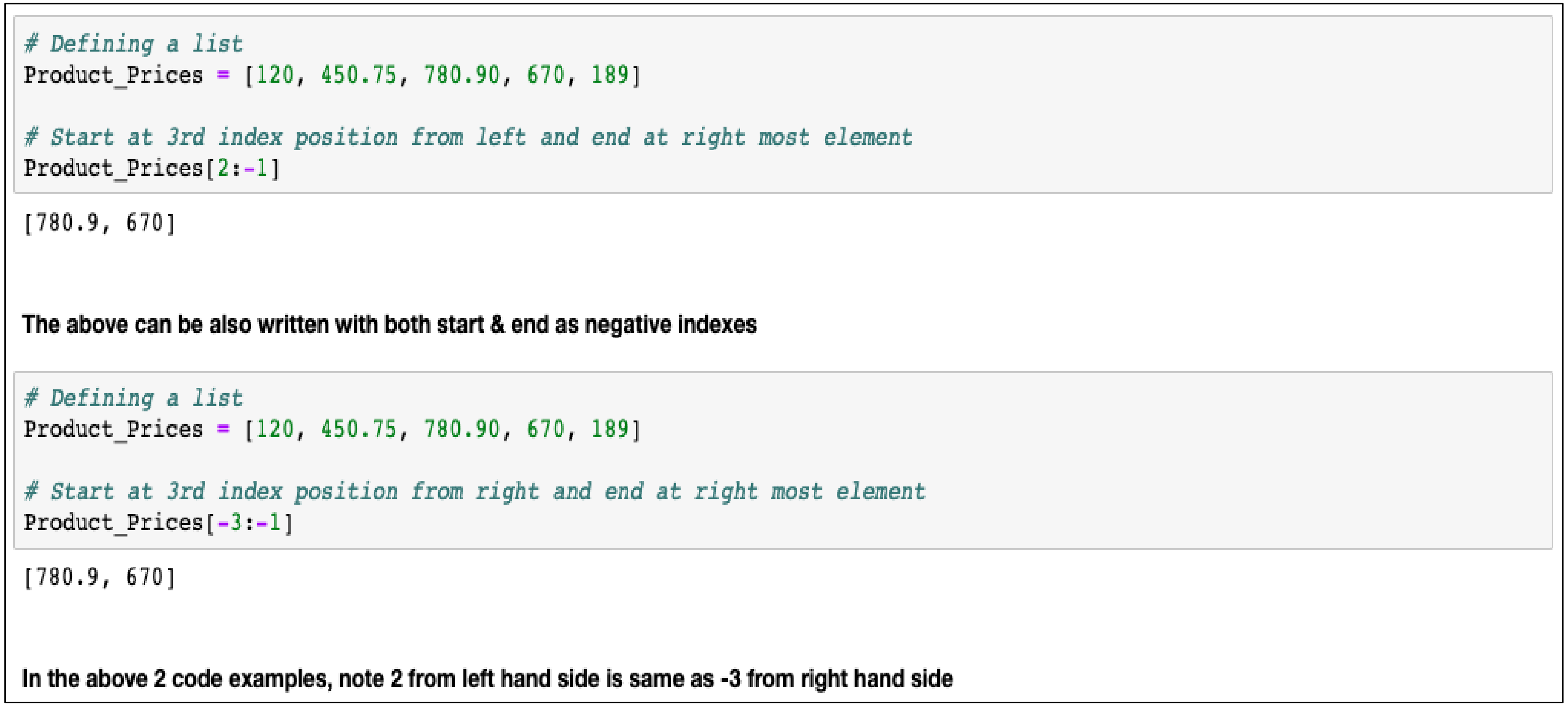
```
[120, 450.75, 780.9, 670]
```

Start at -3 index (i.e. 3rd element from right). End at -1 index (i.e. right most element).

```
# Defining a list
Product_Prices = [120, 450.75, 780.90, 670, 189]

# Start at - 3 and end at -1
Product_Prices[-3:-1]
```

```
[780.9, 670]
```

SLICE A LIST USING NEGATIVE INDEX



```
# Defining a list
Product_Prices = [120, 450.75, 780.90, 670, 189]

# Start at 3rd index position from left and end at right most element
Product_Prices[2:-1]
```

```
[780.9, 670]
```

The above can be also written with both start & end as negative indexes

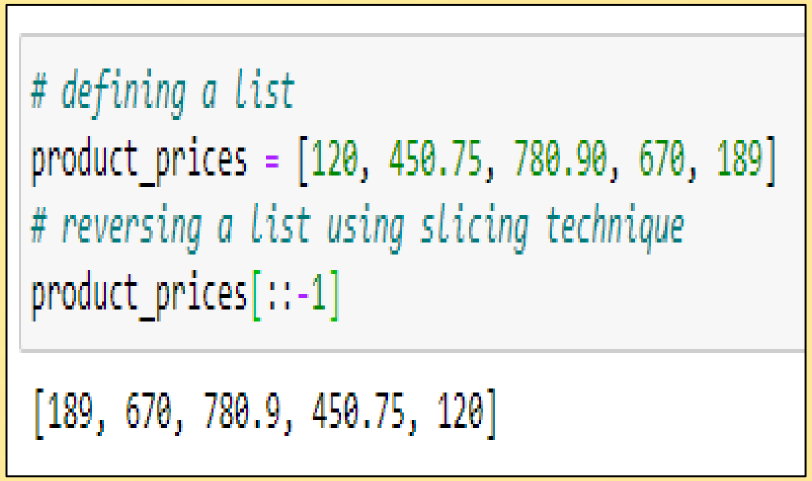
```
# Defining a list
Product_Prices = [120, 450.75, 780.90, 670, 189]

# Start at 3rd index position from right and end at right most element
Product_Prices[-3:-1]
```

```
[780.9, 670]
```

In the above 2 code examples, note 2 from left hand side is same as -3 from right hand side

Did You Know?

You can reverse a list with slicing technique



```
# defining a list
product_prices = [120, 450.75, 780.90, 670, 189]
# reversing a list using slicing technique
product_prices[::-1]
```

```
[189, 670, 780.9, 450.75, 120]
```

Tuple

TUPLES

Container object

Heterogeneous sequence of elements

Can have duplicates

Are immutable

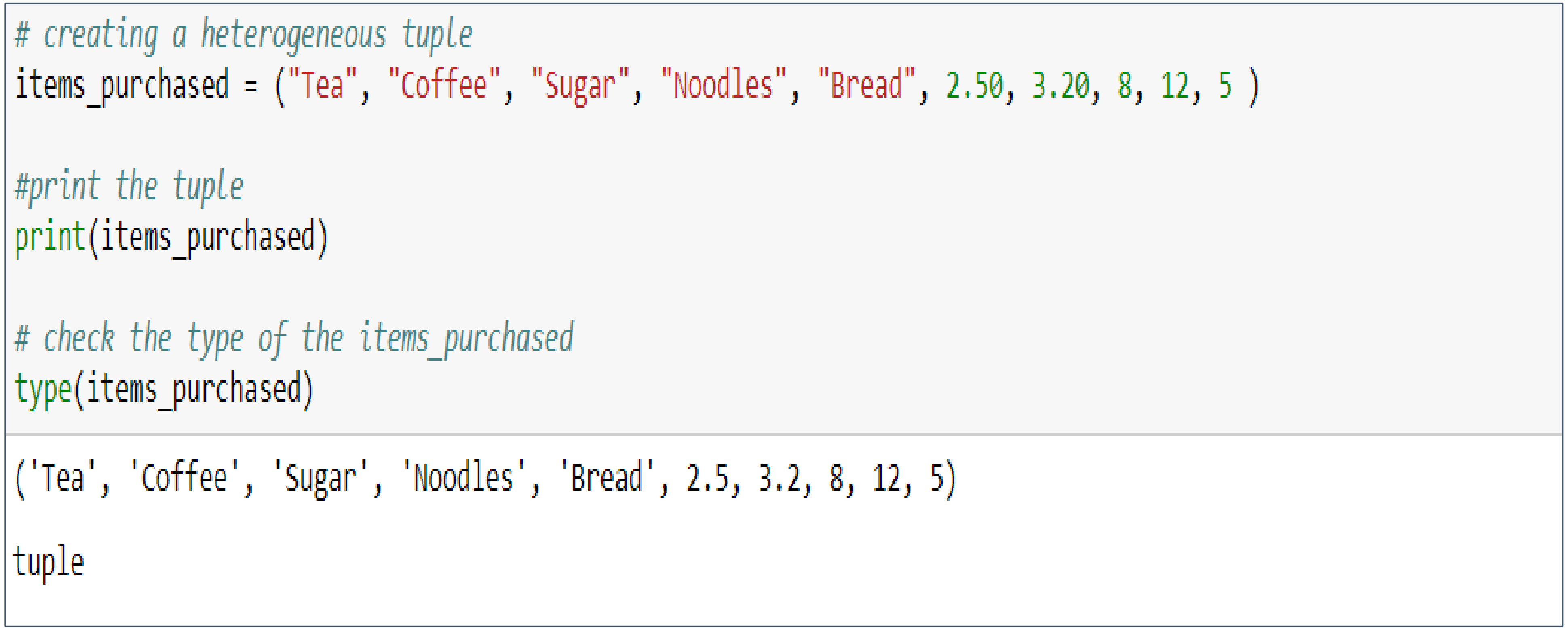
Elements are separated by comma, enclosed in ( ) parenthesis

TUPLES

Tuple supports the following operations:

- in, not in
- Concatenation
- min(), max(), len()
- Indexing and Slicing

CREATE TUPLE

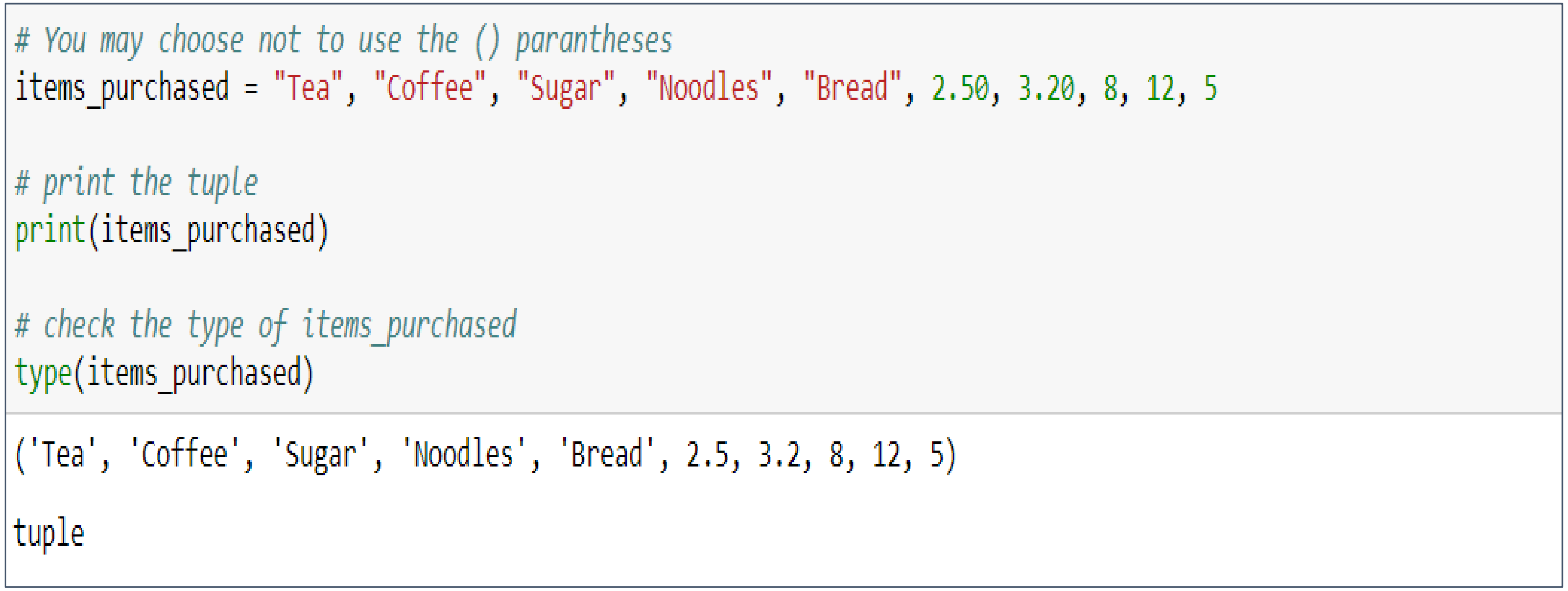


```python
# creating a heterogeneous tuple
items_purchased = ("Tea", "Coffee", "Sugar", "Noodles", "Bread", 2.50, 3.20, 8, 12, 5 )

#print the tuple
print(items_purchased)

# check the type of the items_purchased
type(items_purchased)
```

```
('Tea', 'Coffee', 'Sugar', 'Noodles', 'Bread', 2.5, 3.2, 8, 12, 5)

tuple
```

CREATE TUPLE

Although we mostly use () parentheses to create a tuple, it is not mandatory to use it

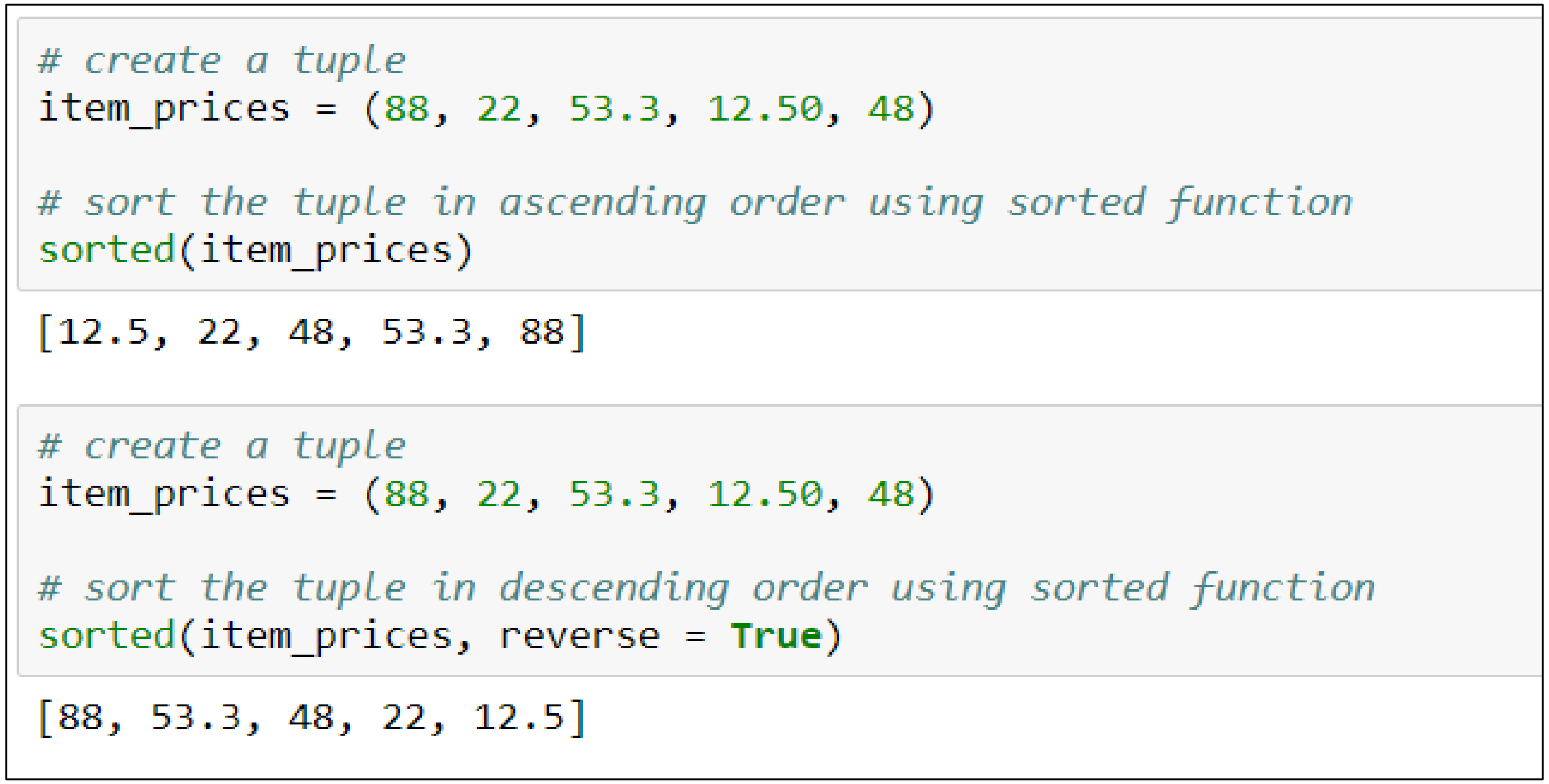


```
# You may choose not to use the () parantheses
items_purchased = "Tea", "Coffee", "Sugar", "Noodles", "Bread", 2.50, 3.20, 8, 12, 5

# print the tuple
print(items_purchased)

# check the type of items_purchased
type(items_purchased)
```

```
('Tea', 'Coffee', 'Sugar', 'Noodles', 'Bread', 2.5, 3.2, 8, 12, 5)

tuple
```

TUPLE OPERATIONS

Sorting the tuple elements using the sorted() function



```
# create a tuple
item_prices = (88, 22, 53.3, 12.50, 48)

# sort the tuple in ascending order using sorted function
sorted(item_prices)
```

```
[12.5, 22, 48, 53.3, 88]
```

```
# create a tuple
item_prices = (88, 22, 53.3, 12.50, 48)

# sort the tuple in descending order using sorted function
sorted(item_prices, reverse = True)
```

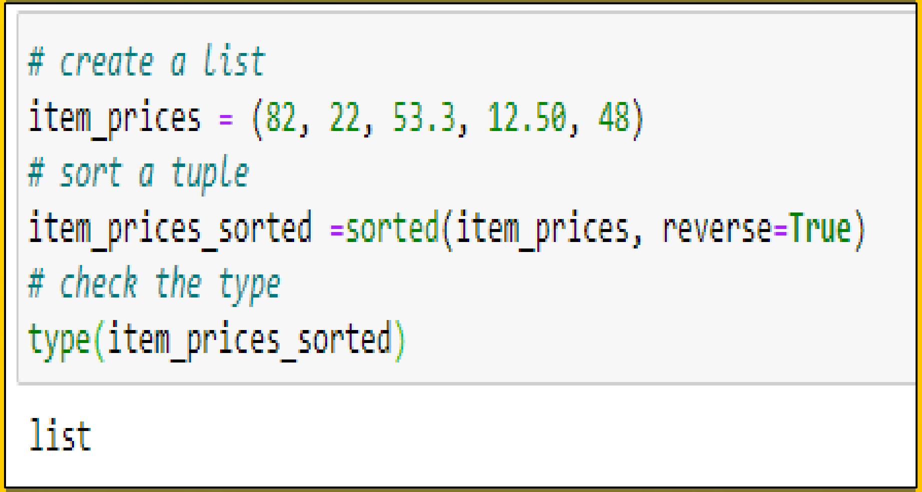
```
[88, 53.3, 48, 22, 12.5]
```

Did You Know?

List has a sort() function but a tuple does not have a sort() function. However, you can use the sorted() function to sort both a list & a tuple

Did You Know?

The sorted() returns a list even if you pass a tuple to it.

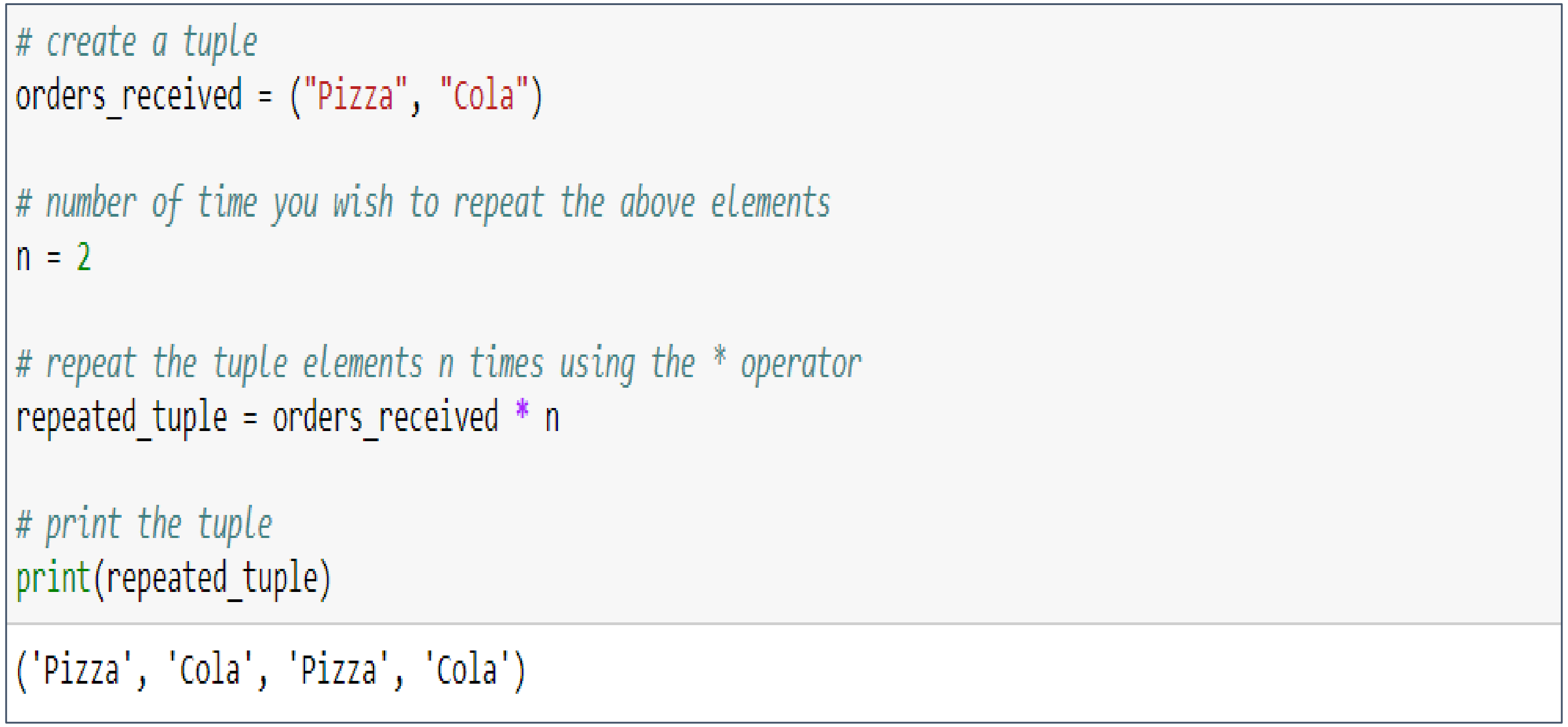


```
# create a list
item_prices = (82, 22, 53.3, 12.50, 48)
# sort a tuple
item_prices_sorted =sorted(item_prices, reverse=True)
# check the type
type(item_prices_sorted)
```

```
list
```

TUPLE OPERATIONS

Repeating the tuple elements

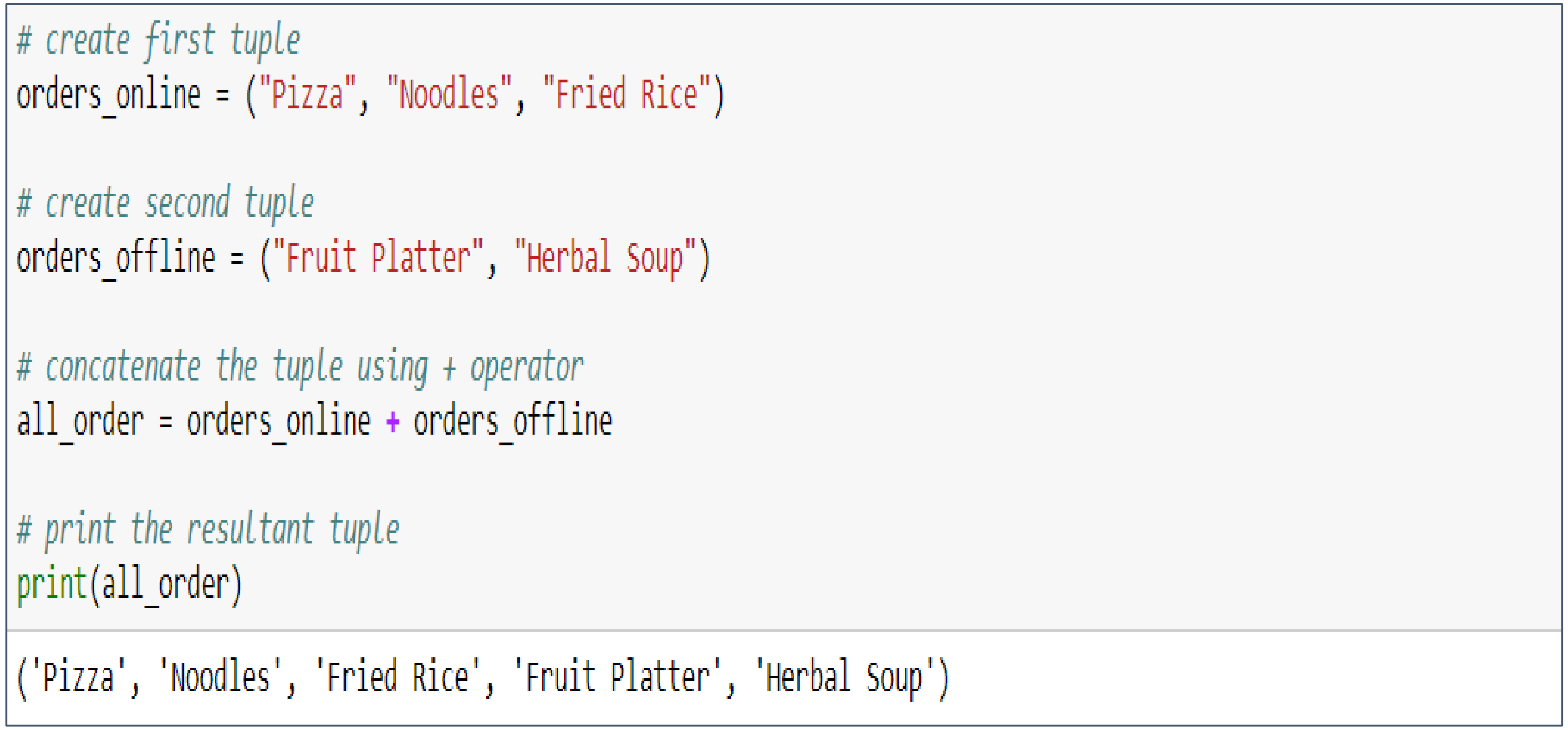


```
# create a tuple
orders_received = ("Pizza", "Cola")

# number of time you wish to repeat the above elements
n = 2

# repeat the tuple elements n times using the * operator
repeated_tuple = orders_received * n

# print the tuple
print(repeated_tuple)
```

```
('Pizza', 'Cola', 'Pizza', 'Cola')
```

TUPLE OPERATIONS

Concatenating two tuples

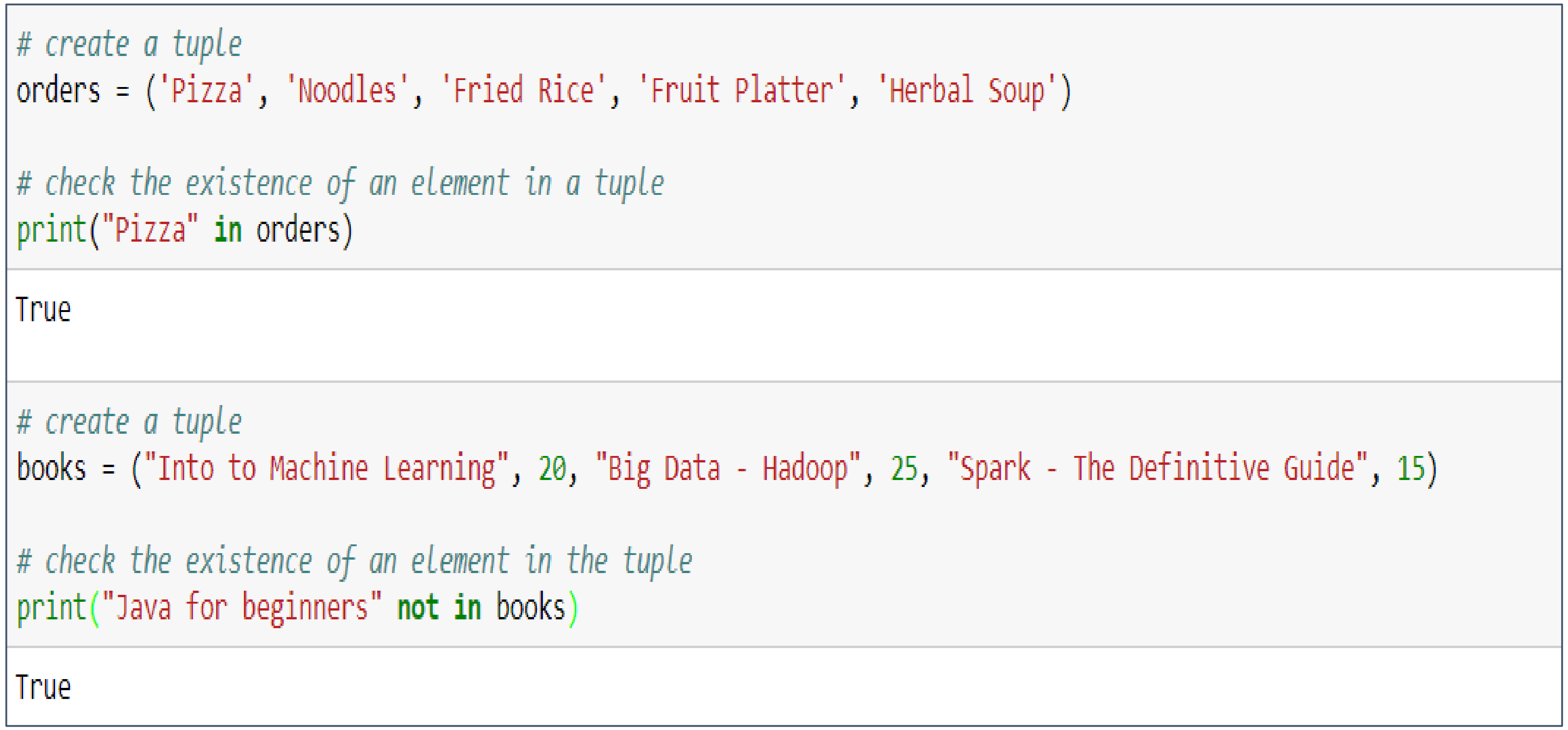


```
# create first tuple
orders_online = ("Pizza", "Noodles", "Fried Rice")

# create second tuple
orders_offline = ("Fruit Platter", "Herbal Soup")

# concatenate the tuple using + operator
all_order = orders_online + orders_offline

# print the resultant tuple
print(all_order)
```

```
('Pizza', 'Noodles', 'Fried Rice', 'Fruit Platter', 'Herbal Soup')
```

TUPLE OPERATIONS

Checking existence of an element using membership operators



```
# create a tuple
orders = ('Pizza', 'Noodles', 'Fried Rice', 'Fruit Platter', 'Herbal Soup')

# check the existence of an element in a tuple
print("Pizza" in orders)
```

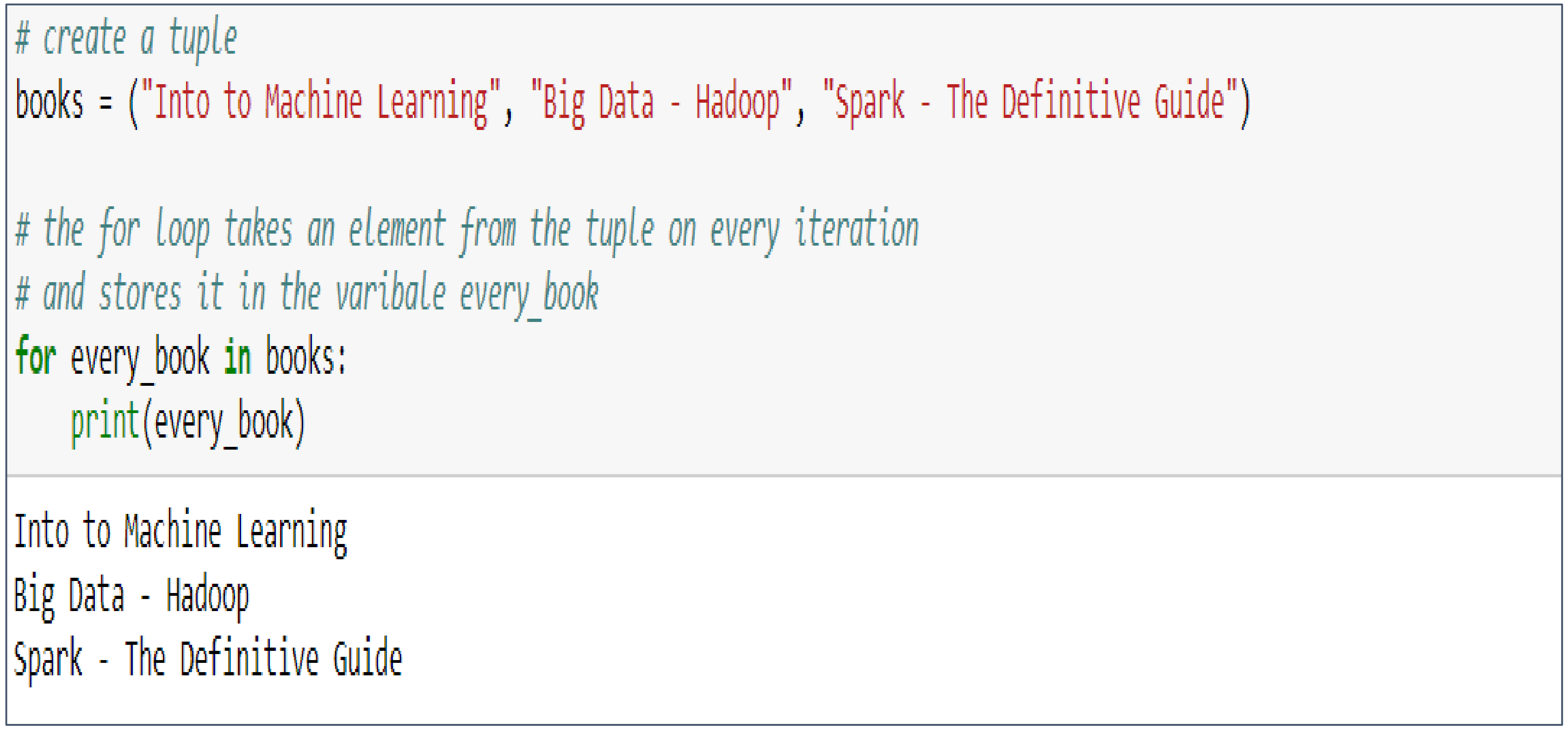
```
True
```

```
# create a tuple
books = ("Into to Machine Learning", 20, "Big Data - Hadoop", 25, "Spark - The Definitive Guide", 15)

# check the existence of an element in the tuple
print("Java for beginners" not in books)
```

```
True
```

TUPLE OPERATIONS

Iterating through the element of a tuple

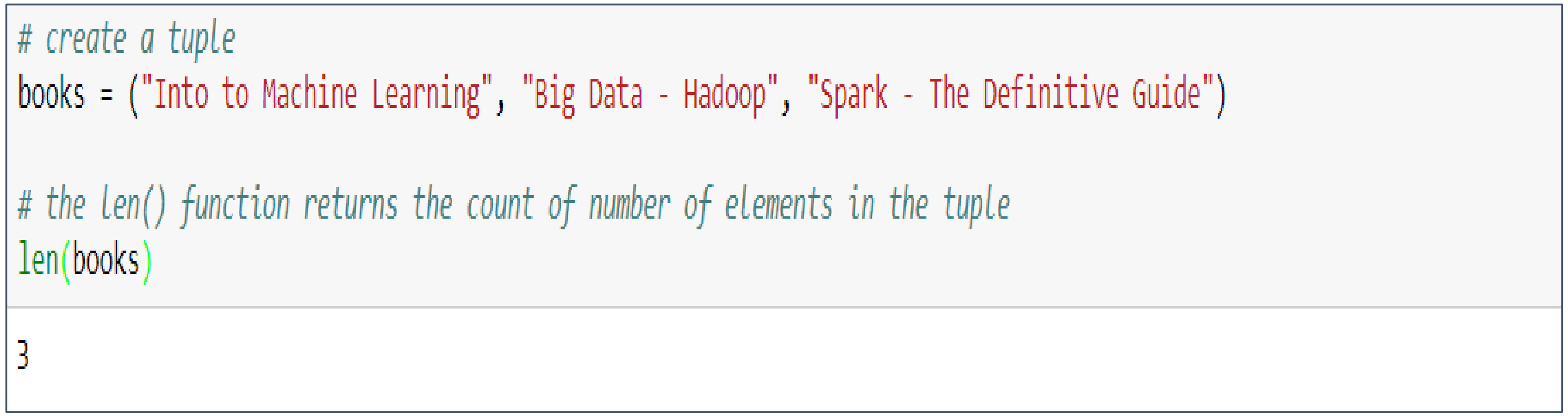


```
# create a tuple
books = ("Into to Machine Learning", "Big Data - Hadoop", "Spark - The Definitive Guide")

# the for loop takes an element from the tuple on every iteration
# and stores it in the varibale every_book
for every_book in books:
    print(every_book)
```

```
Into to Machine Learning
Big Data - Hadoop
Spark - The Definitive Guide
```

TUPLE OPERATIONS

Counting the number of elements in the tuple

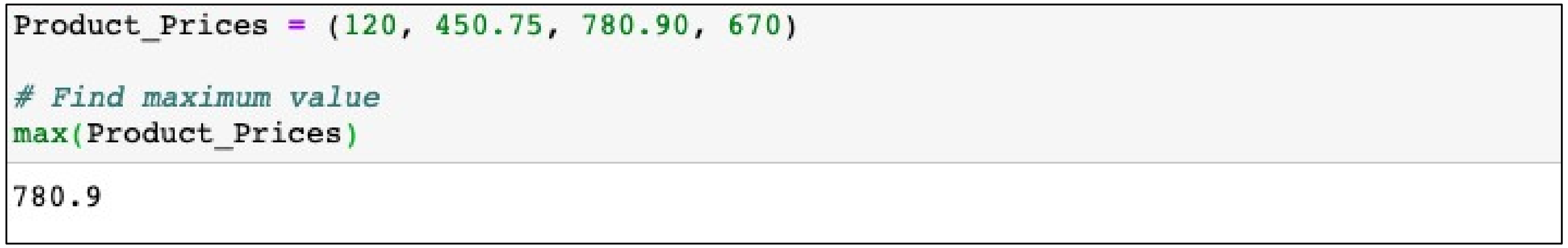


```
# create a tuple
books = ("Into to Machine Learning", "Big Data - Hadoop", "Spark - The Definitive Guide")

# the len() function returns the count of number of elements in the tuple
len(books)
```

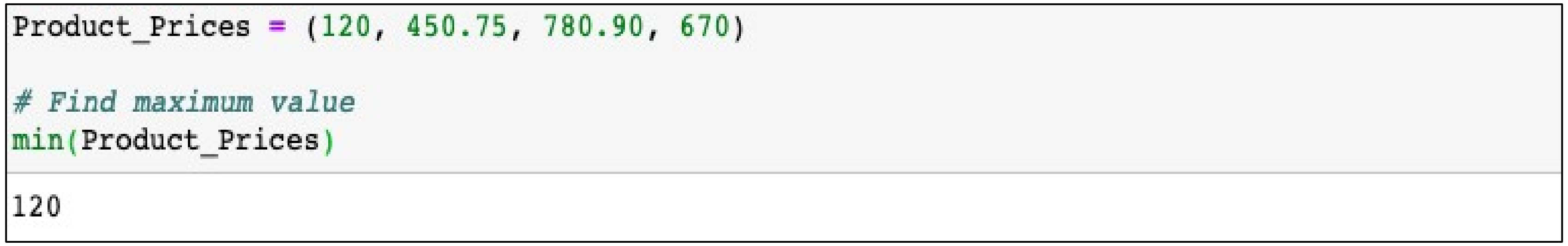
```
3
```

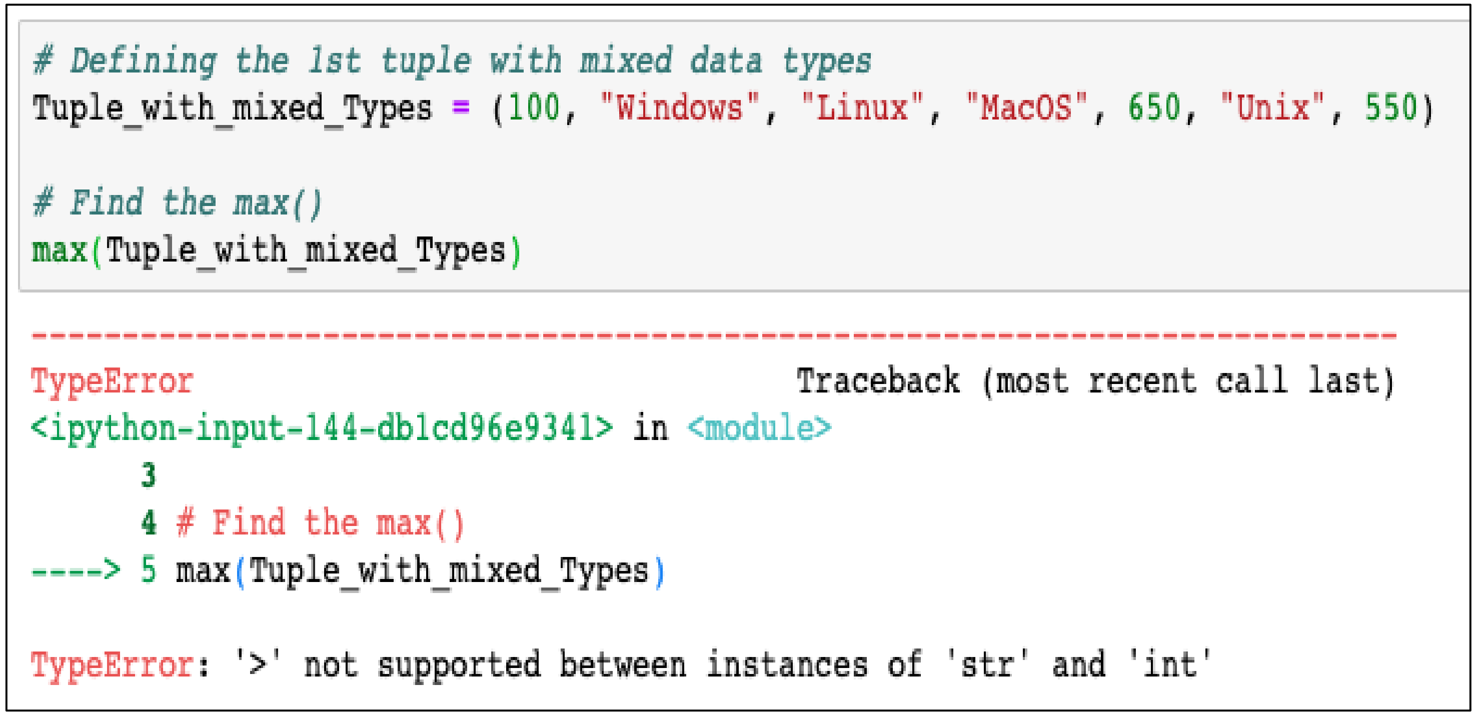
TUPLE OPERATIONS

Get the maximum value from the tuple


```
Product_Prices = (120, 450.75, 780.90, 670)

# Find maximum value
max(Product_Prices)
```
```
780.9
```

Get the minimum value from the tuple

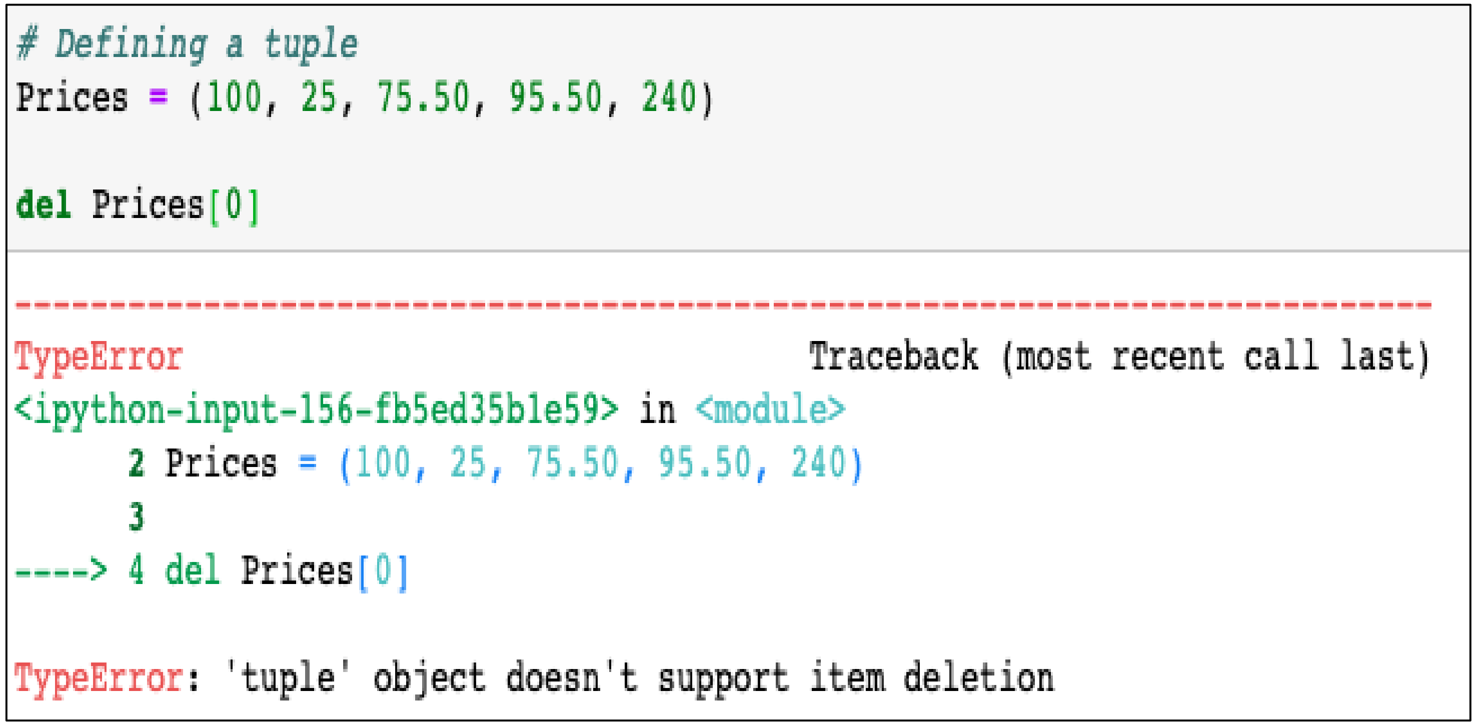

```
Product_Prices = (120, 450.75, 780.90, 670)

# Find maximum value
min(Product_Prices)
```
```
120
```



```
# Defining the 1st tuple with mixed data types
Tuple_with_mixed_Types = (100, "Windows", "Linux", "MacOS", 650, "Unix", 550)

# Find the max()
max(Tuple_with_mixed_Types)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-144-db1cd96e9341> in <module>
      3
      4 # Find the max()
----> 5 max(Tuple_with_mixed_Types)

TypeError: '>' not supported between instances of 'str' and 'int'
```

The max() & min() cannot be used on a tuple with mixed data types



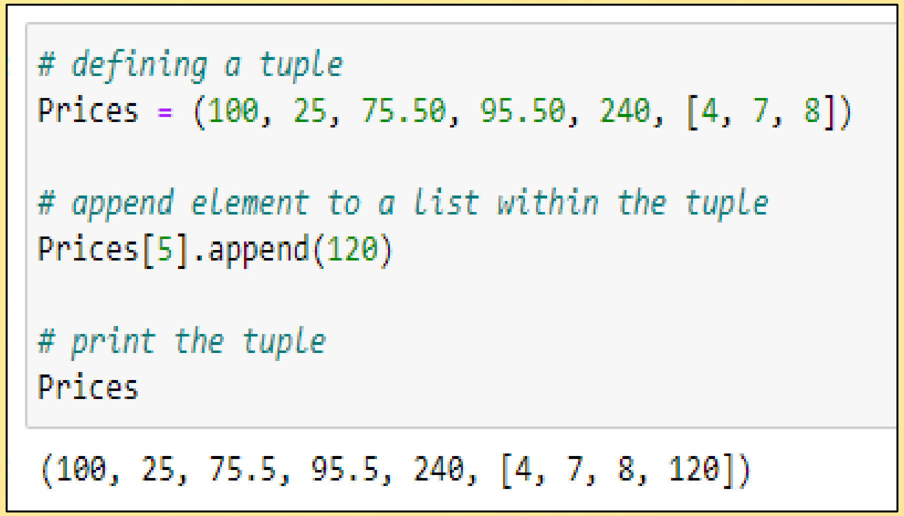
```
# Defining a tuple
Prices = (100, 25, 75.50, 95.50, 240)

del Prices[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-156-fb5ed35b1e59> in <module>
      2 Prices = (100, 25, 75.50, 95.50, 240)
      3 
----> 4 del Prices[0]

TypeError: 'tuple' object doesn't support item deletion
```

Since a tuple is immutable, a new element cannot be modified

Did You Know?

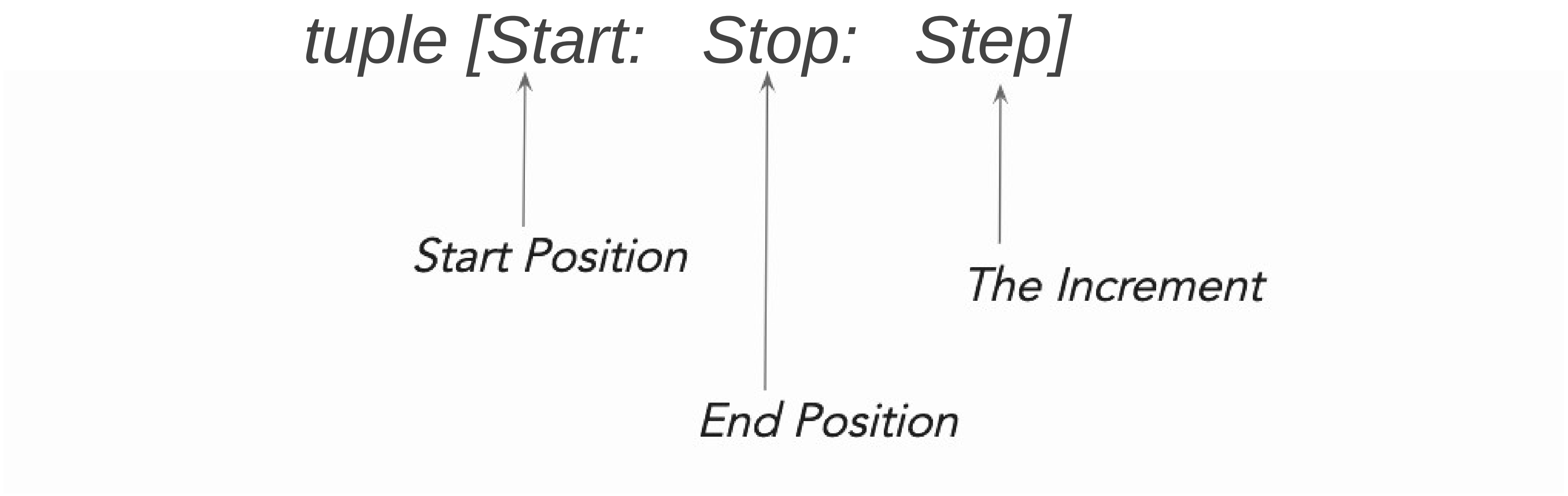
You can have a list within a tuple
And you can append elements to a list within the tuple



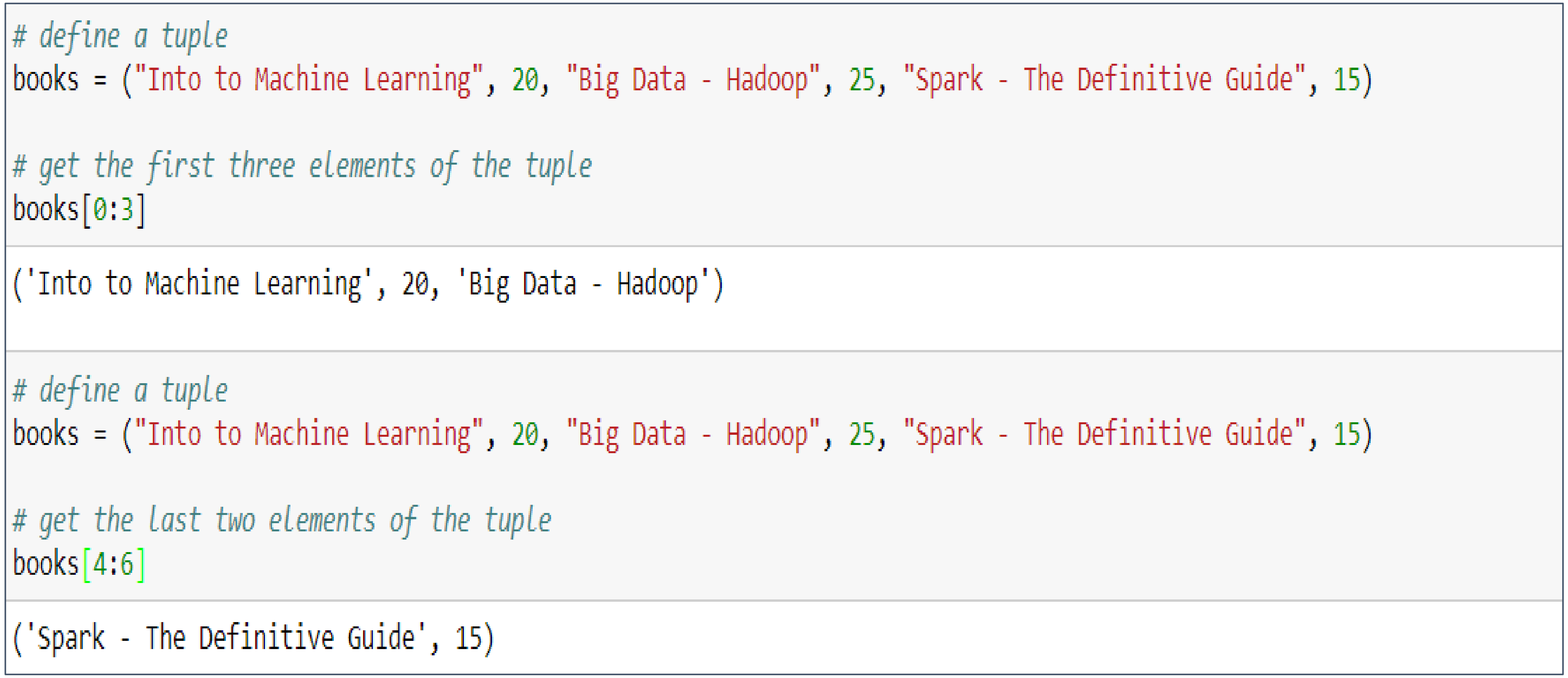
```
# defining a tuple
Prices = (100, 25, 75.50, 95.50, 240, [4, 7, 8])

# append element to a list within the tuple
Prices[5].append(120)

# print the tuple
Prices
```

```
(100, 25, 75.5, 95.5, 240, [4, 7, 8, 120])
```

INDEXING AND SLICING TUPLES



INDEXING AND SLICING TUPLES



```
# define a tuple
books = ("Into to Machine Learning", 20, "Big Data - Hadoop", 25, "Spark - The Definitive Guide", 15)

# get the first three elements of the tuple
books[0:3]
```

```
('Into to Machine Learning', 20, 'Big Data - Hadoop')
```

```
# define a tuple
books = ("Into to Machine Learning", 20, "Big Data - Hadoop", 25, "Spark - The Definitive Guide", 15)

# get the last two elements of the tuple
books[4:6]
```

```
('Spark - The Definitive Guide', 15)
```

Did You Know?

Because tuple is immutable, a tuple needs lesser memory than lists.

Did You Know?

Lesser memory fragmentation is better in terms of performance. Python can reuse a tuple which results in reduction of memory fragmentation and can speed up memory allocations. If a tuple is no longer required and has less than 20 elements, instead of deleting it permanently, Python reserves it temporarily.

Dictionary

DICTIONARY

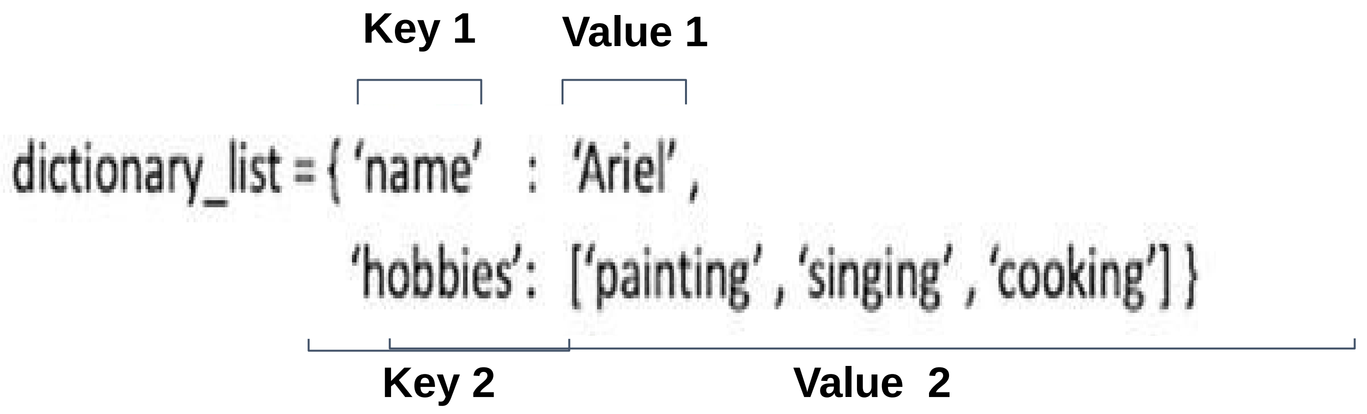
Elements exists in key-value pairs

Use {} curly brackets for creating a dictionary

[] square brackets for indexing it

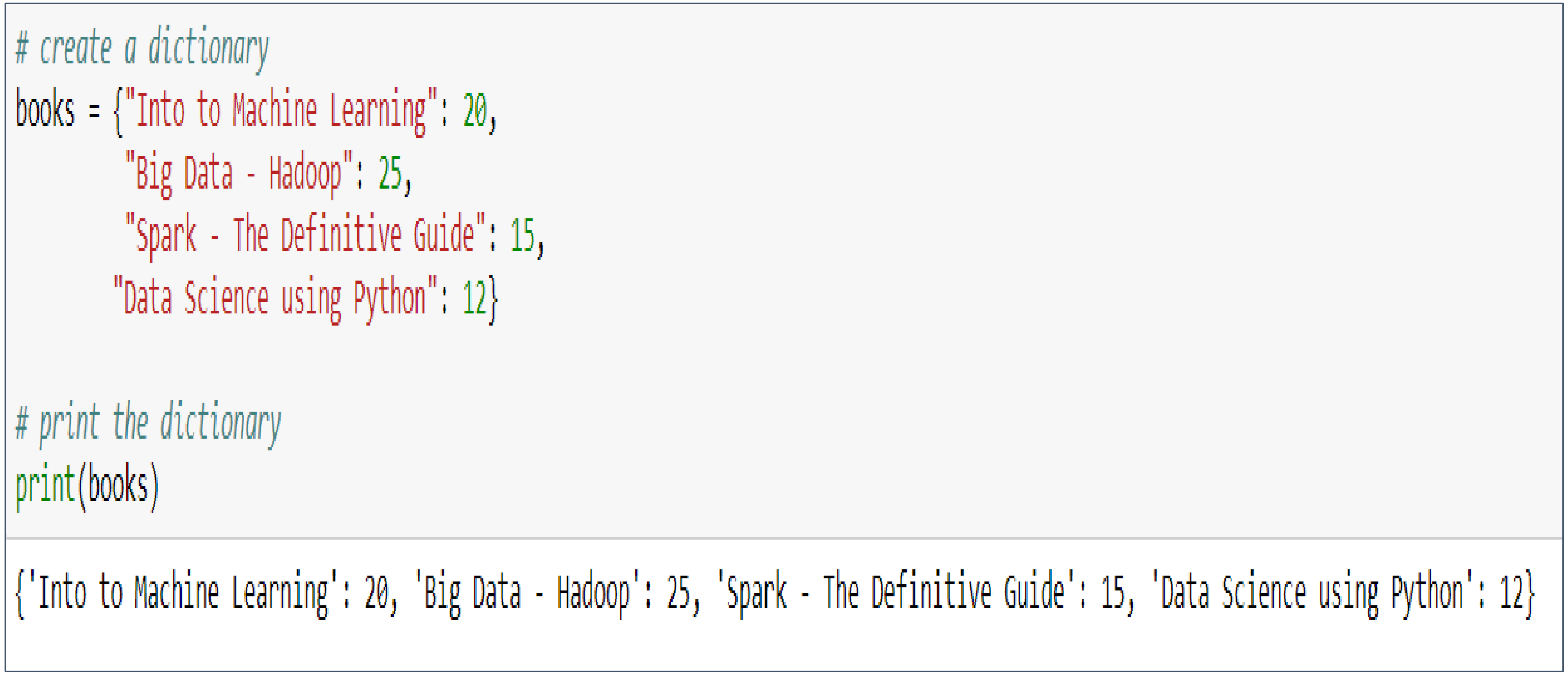
Dictionaries are mutable and unordered

HOW DOES A DICTIONARY OBJECT LOOK LIKE?



```
dictionary_list = { 'name'   :  'Ariel' ,
                   'hobbies':  ['painting' , 'singing' , 'cooking'] }
```

CREATING A DICTIONARY OBJECT

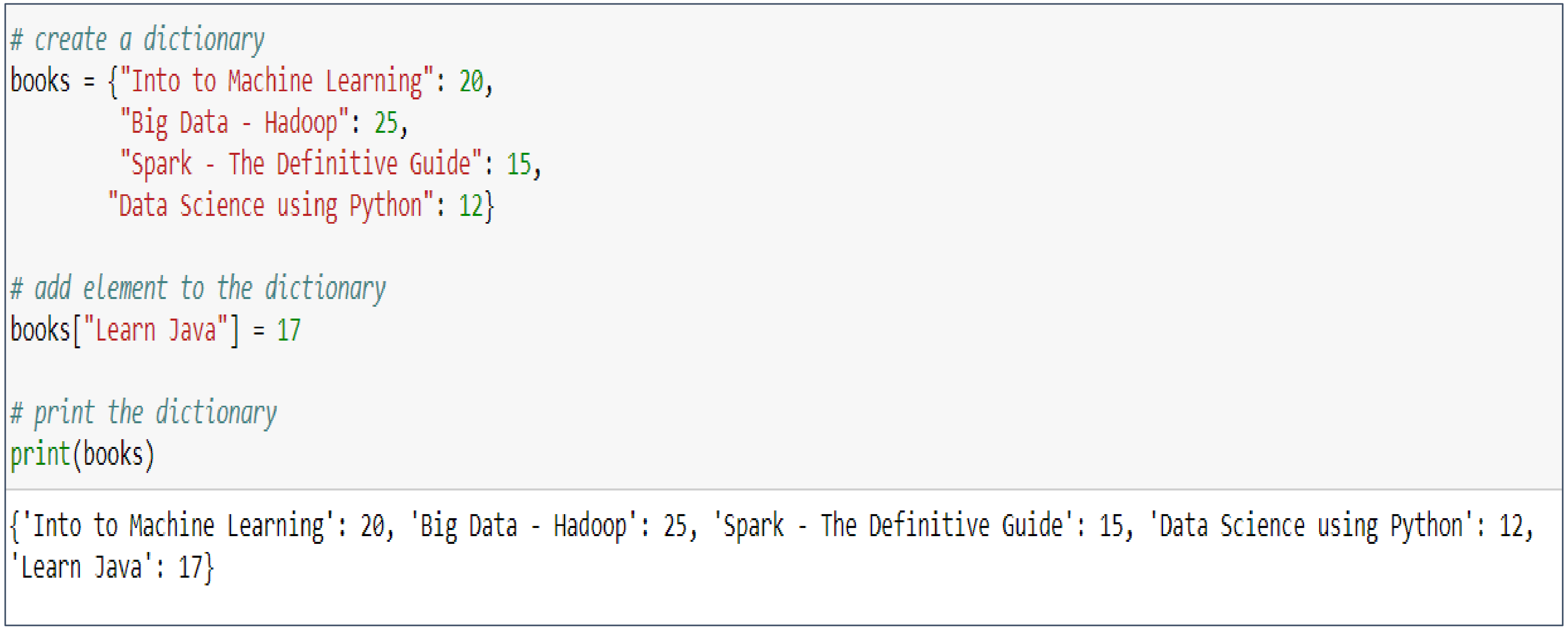


```
# create a dictionary
books = {"Into to Machine Learning": 20,
         "Big Data - Hadoop": 25,
         "Spark - The Definitive Guide": 15,
        "Data Science using Python": 12}


# print the dictionary
print(books)
```

```
{'Into to Machine Learning': 20, 'Big Data - Hadoop': 25, 'Spark - The Definitive Guide': 15, 'Data Science using Python': 12}
```

DICTIONARY OPERATIONS

Adding an element in a dictionary

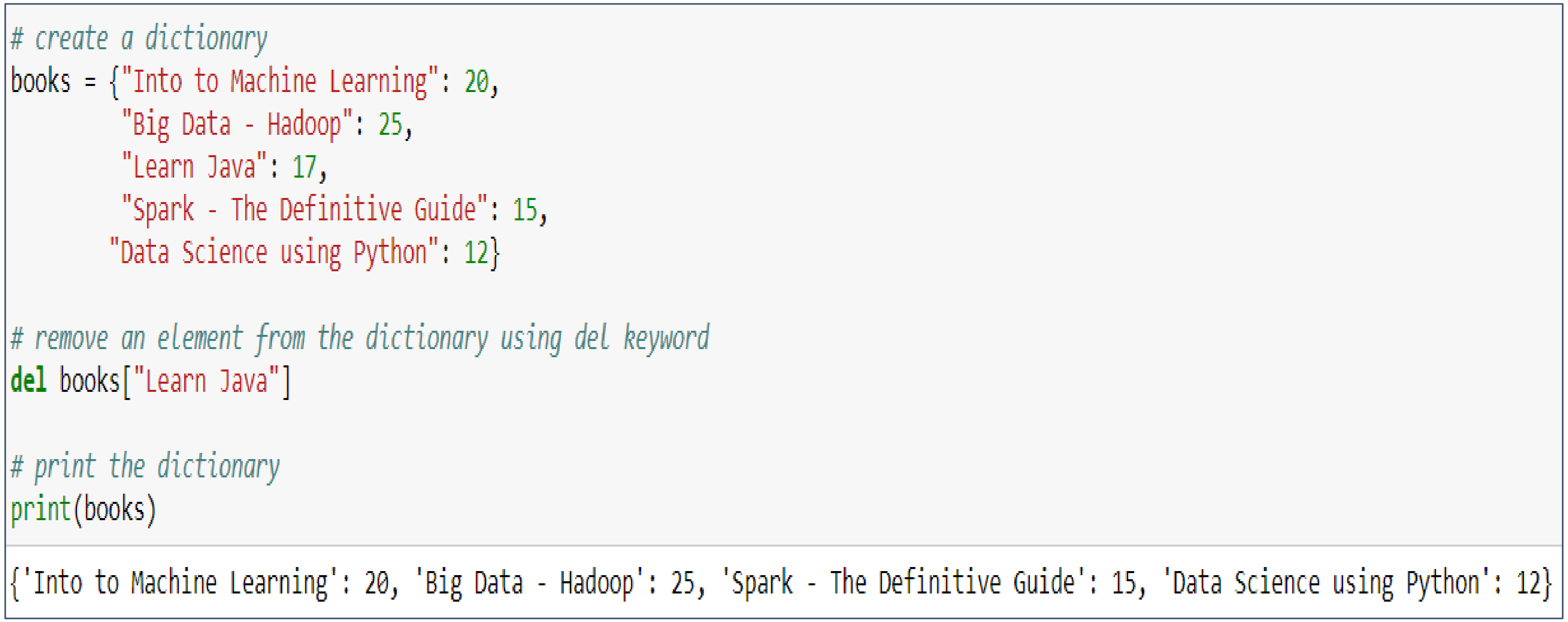


```
# create a dictionary
books = {"Into to Machine Learning": 20,
         "Big Data - Hadoop": 25,
         "Spark - The Definitive Guide": 15,
        "Data Science using Python": 12}

# add element to the dictionary
books["Learn Java"] = 17

# print the dictionary
print(books)
```

```
{'Into to Machine Learning': 20, 'Big Data - Hadoop': 25, 'Spark - The Definitive Guide': 15, 'Data Science using Python': 12,
'Learn Java': 17}
```

DICTIONARY OPERATIONS

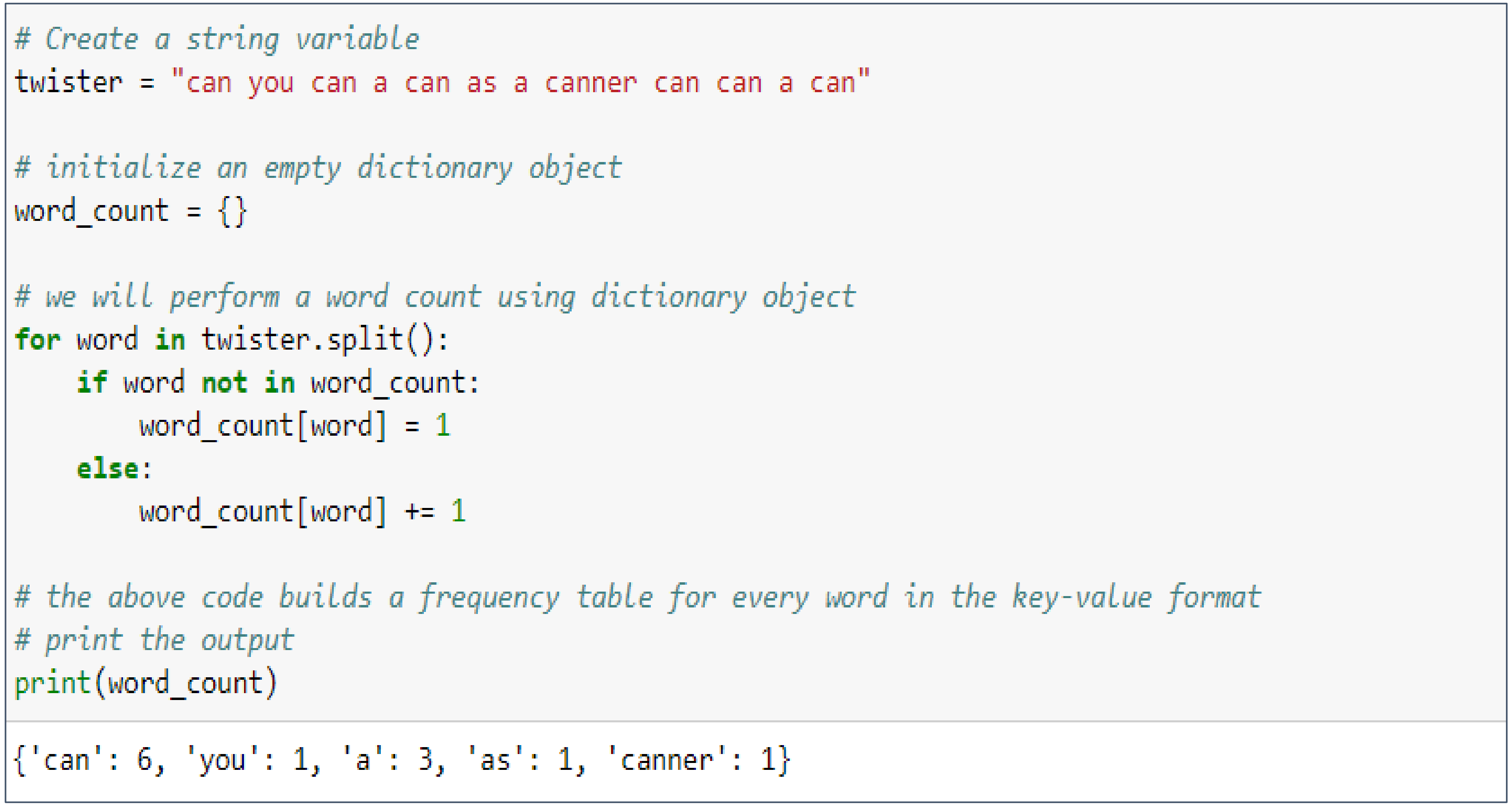
Removing an element from a dictionary



```
# create a dictionary
books = {"Into to Machine Learning": 20,
         "Big Data - Hadoop": 25,
         "Learn Java": 17,
         "Spark - The Definitive Guide": 15,
        "Data Science using Python": 12}

# remove an element from the dictionary using del keyword
del books["Learn Java"]

# print the dictionary
print(books)
```

```
{'Into to Machine Learning': 20, 'Big Data - Hadoop': 25, 'Spark - The Definitive Guide': 15, 'Data Science using Python': 12}
```

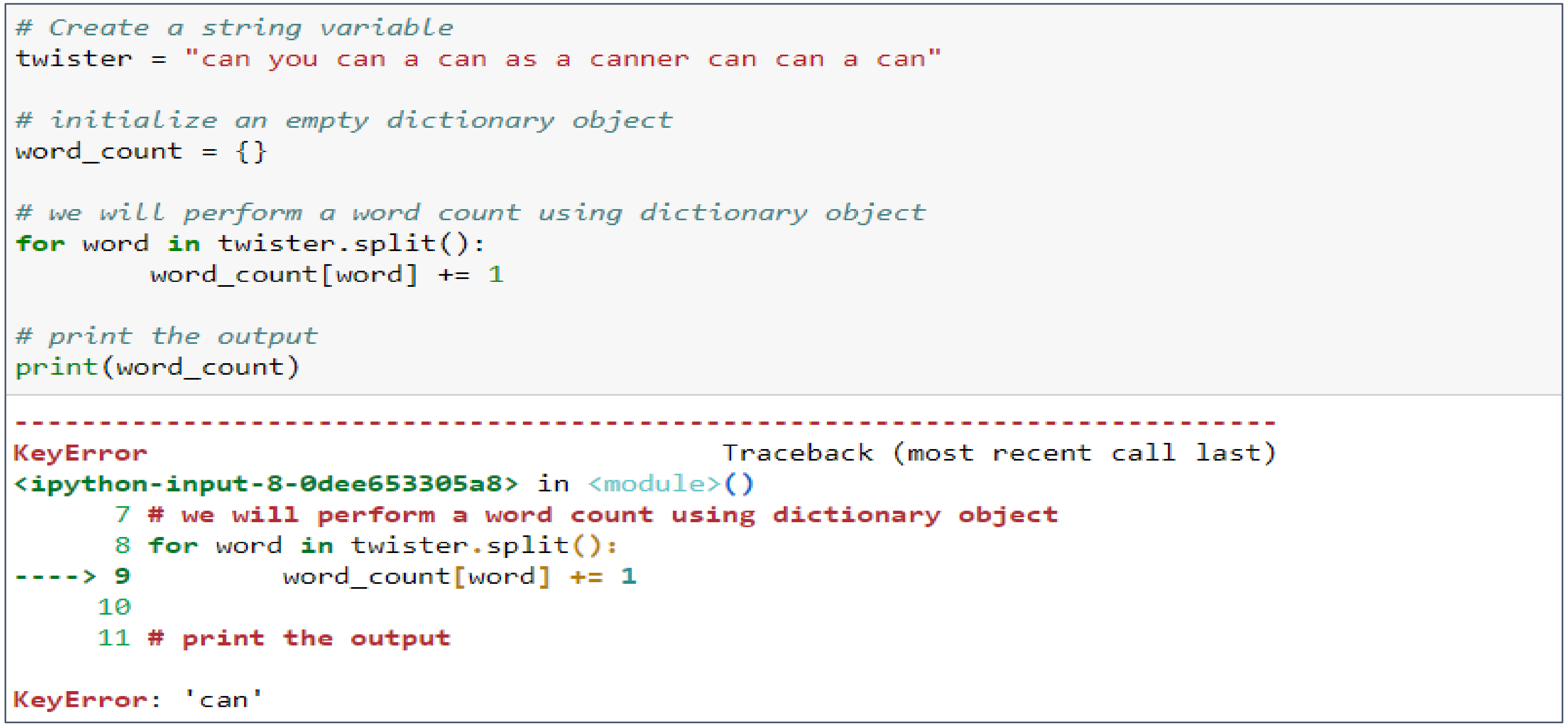
DICTIONARY WORD COUNT EXAMPLE



```python
# Create a string variable
twister = "can you can a can as a canner can can a can"

# initialize an empty dictionary object
word_count = {}

# we will perform a word count using dictionary object
for word in twister.split():
    if word not in word_count:
        word_count[word] = 1
    else:
        word_count[word] += 1

# the above code builds a frequency table for every word in the key-value format
# print the output
print(word_count)
```

```
{'can': 6, 'you': 1, 'a': 3, 'as': 1, 'canner': 1}
```

CONSIDER THE WORD CONT PROGRAM BELOW



```
# Create a string variable
twister = "can you can a can as a canner can can a can"

# initialize an empty dictionary object
word_count = {}

# we will perform a word count using dictionary object
for word in twister.split():
        word_count[word] += 1

# print the output
print(word_count)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-8-0dee653305a8> in <module>()
      7 # we will perform a word count using dictionary object
      8 for word in twister.split():
----> 9         word_count[word] += 1
     10 
     11 # print the output

KeyError: 'can'
```

The above code throws an error because the key is not initialized and we are trying to add 1 to an the uninitialized key's value

DEFAULT DICTIONARY

In order to avoid the error for uninitialized values, we use defaultdict()

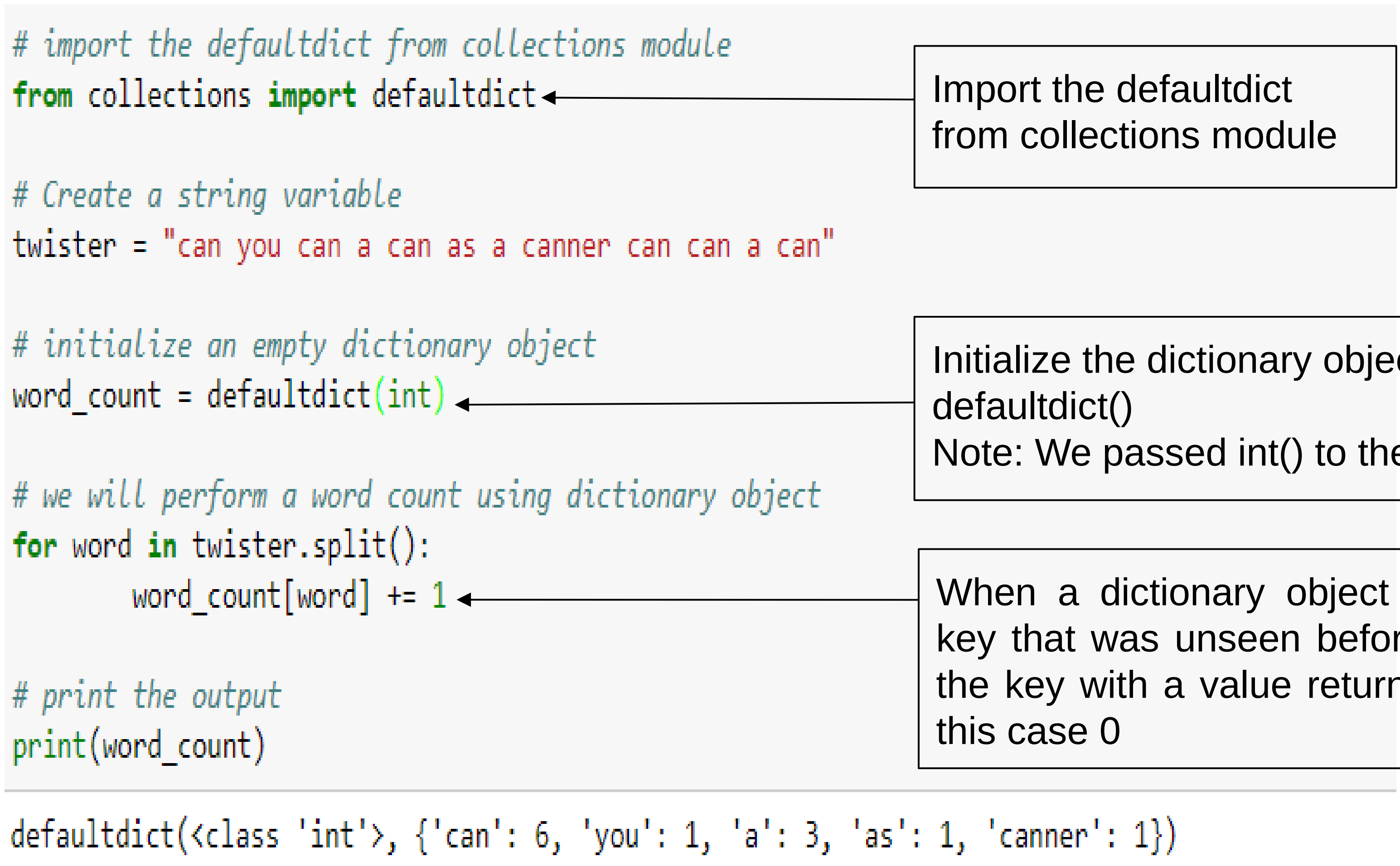
Import the defaultdict from collections module

Initialize the dictionary object with defaultdict()

Note: We passed int() to the defaultdict()

When the dictionary object encounters a key that was not seen before, it initializes the key with a value returned by int(), in this case 0 (zero)

DICTIONARY OBJECT: DEFAULT DICTIONARY



```python
# import the defaultdict from collections module
from collections import defaultdict

# Create a string variable
twister = "can you can a can as a canner can can a can"

# initialize an empty dictionary object
word_count = defaultdict(int)

# we will perform a word count using dictionary object
for word in twister.split():
        word_count[word] += 1

# print the output
print(word_count)
```

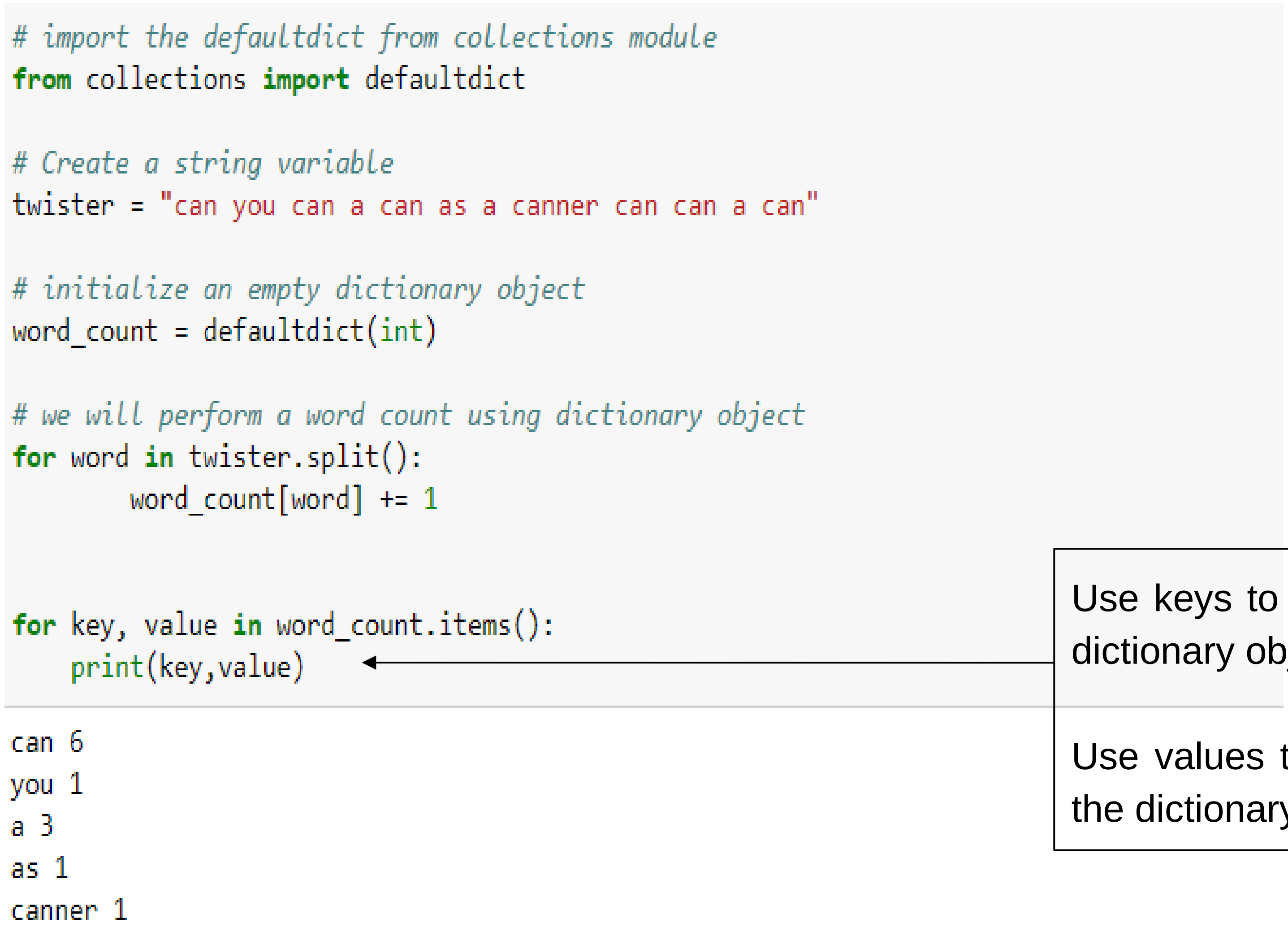
```
defaultdict(<class 'int'>, {'can': 6, 'you': 1, 'a': 3, 'as': 1, 'canner': 1})
```

Import the defaultdict from collections module

Initialize the dictionary object with defaultdict()
Note: We passed int() to the defaultdict()

When a dictionary object encounters a key that was unseen before, it initializes the key with a value returned by int(), in this case 0

LOOPING THROUGH THE DICTIONARY



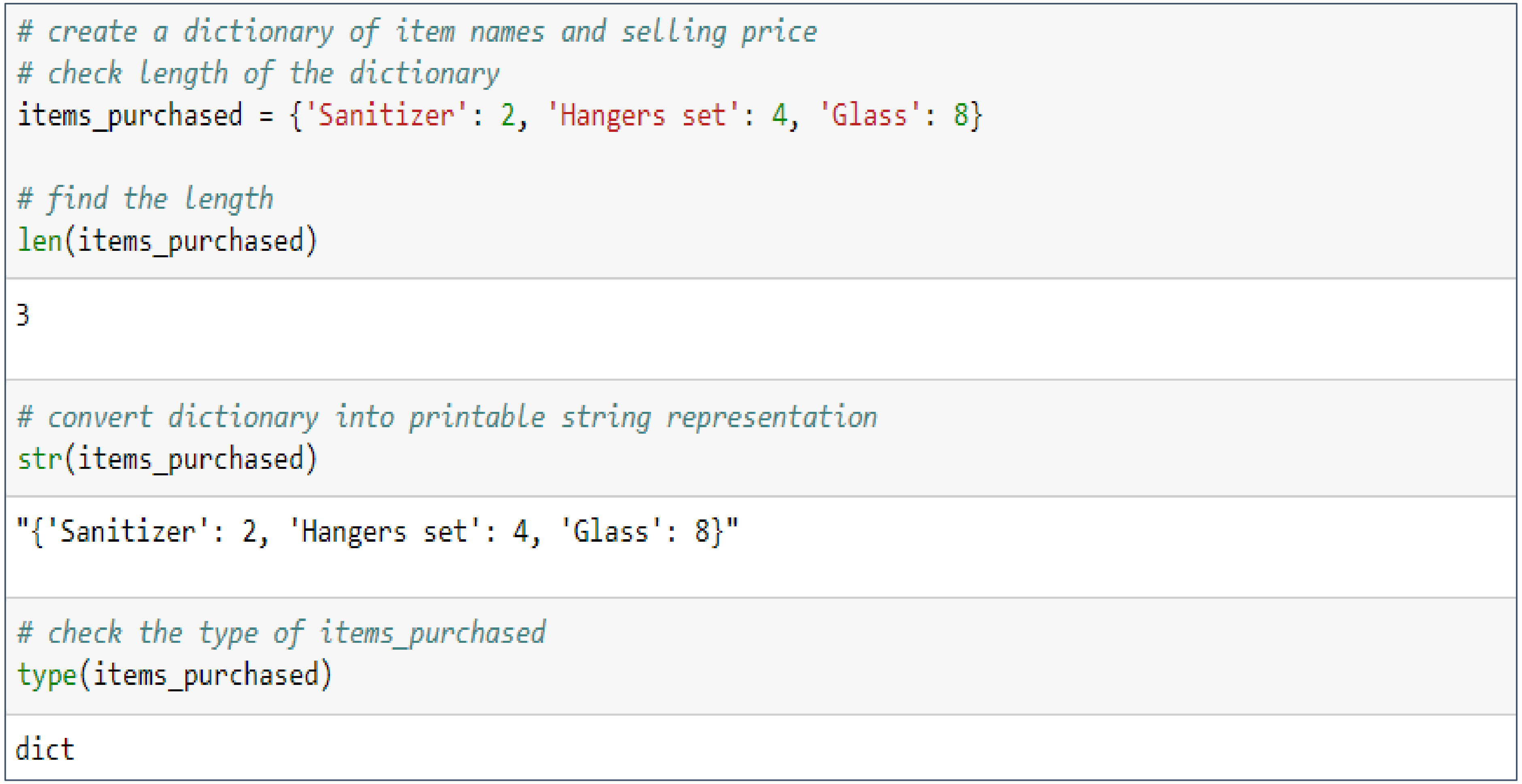
```
# import the defaultdict from collections module
from collections import defaultdict

# Create a string variable
twister = "can you can a can as a canner can can a can"

# initialize an empty dictionary object
word_count = defaultdict(int)

# we will perform a word count using dictionary object
for word in twister.split():
        word_count[word] += 1


for key, value in word_count.items():
    print(key,value)
```

```
can 6
you 1
a 3
as 1
canner 1
```

Use keys to loop through the keys in the dictionary object

Use values to loop through the values in the dictionary object

DICTIONARY OPERATIONS



```
# create a dictionary of item names and selling price
# check length of the dictionary
items_purchased = {'Sanitizer': 2, 'Hangers set': 4, 'Glass': 8}

# find the length
len(items_purchased)
```

```
3
```

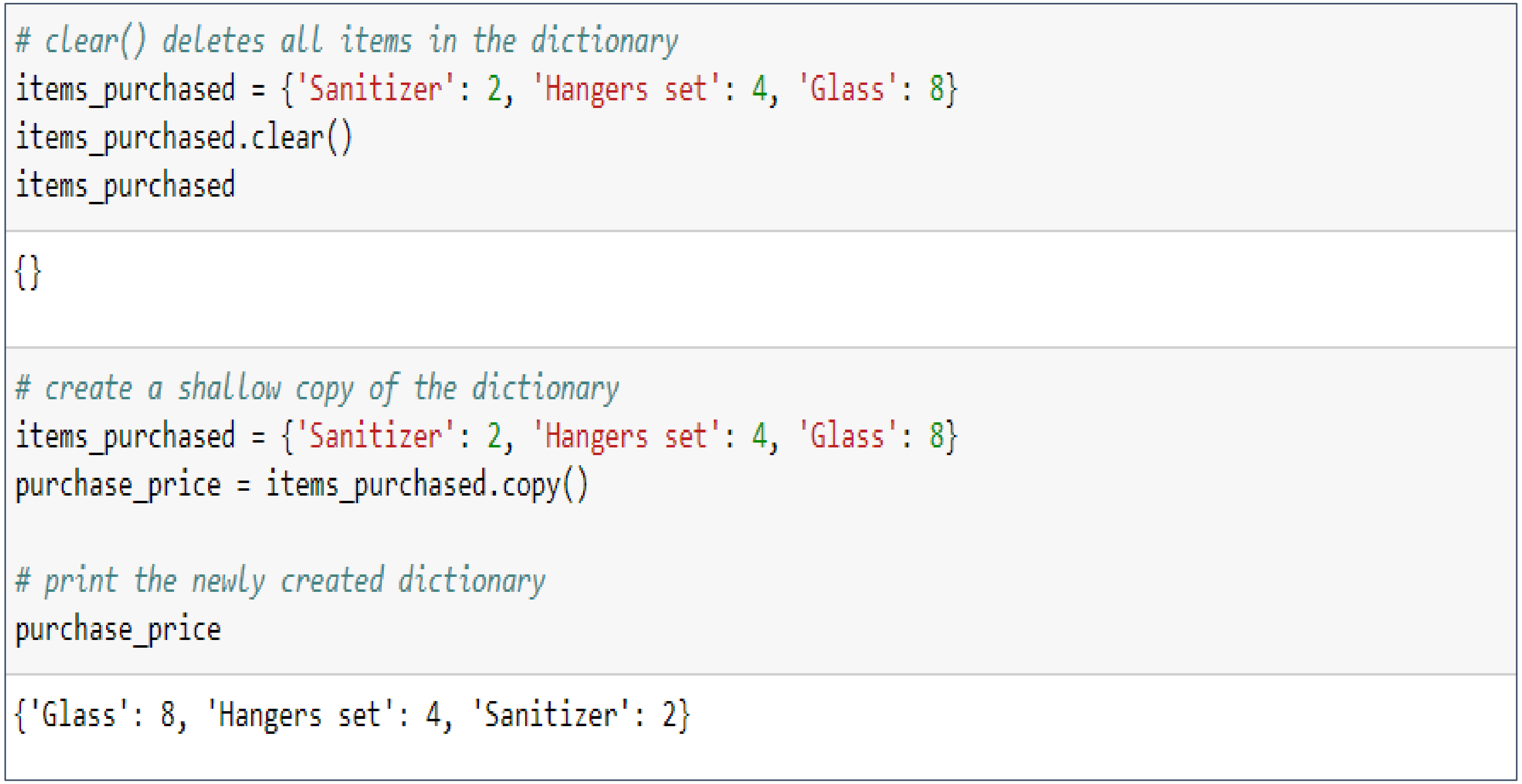
```
# convert dictionary into printable string representation
str(items_purchased)
```

```
"{'Sanitizer': 2, 'Hangers set': 4, 'Glass': 8}"
```

```
# check the type of items_purchased
type(items_purchased)
```

```
dict
```

DICTIONARY OPERATIONS



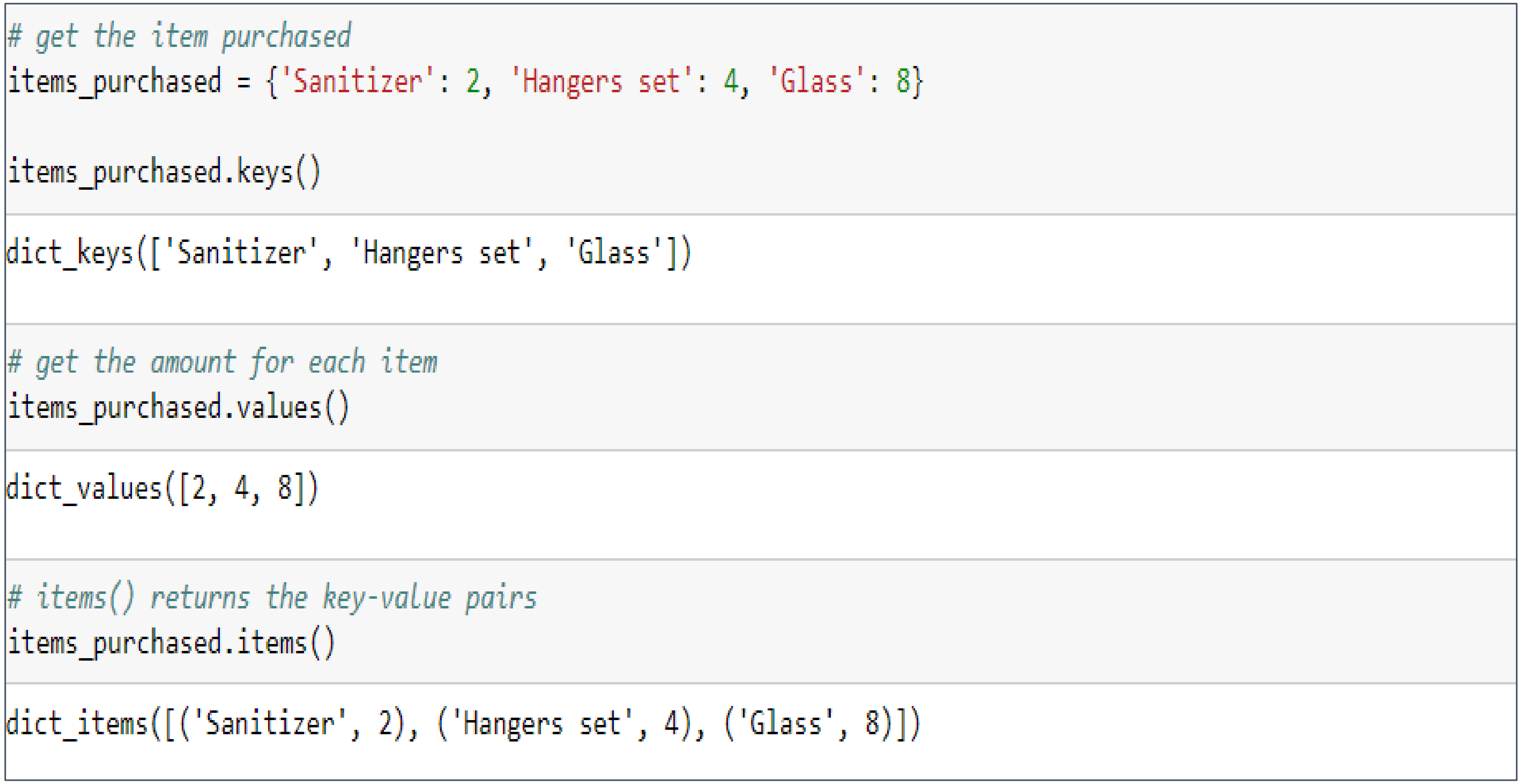
```python
# clear() deletes all items in the dictionary
items_purchased = {'Sanitizer': 2, 'Hangers set': 4, 'Glass': 8}
items_purchased.clear()
items_purchased
```

```
{}
```

```python
# create a shallow copy of the dictionary
items_purchased = {'Sanitizer': 2, 'Hangers set': 4, 'Glass': 8}
purchase_price = items_purchased.copy()

# print the newly created dictionary
purchase_price
```

```
{'Glass': 8, 'Hangers set': 4, 'Sanitizer': 2}
```

DICTIONARY OPERATIONS



```
# get the item purchased
items_purchased = {'Sanitizer': 2, 'Hangers set': 4, 'Glass': 8}

items_purchased.keys()
```

```
dict_keys(['Sanitizer', 'Hangers set', 'Glass'])
```

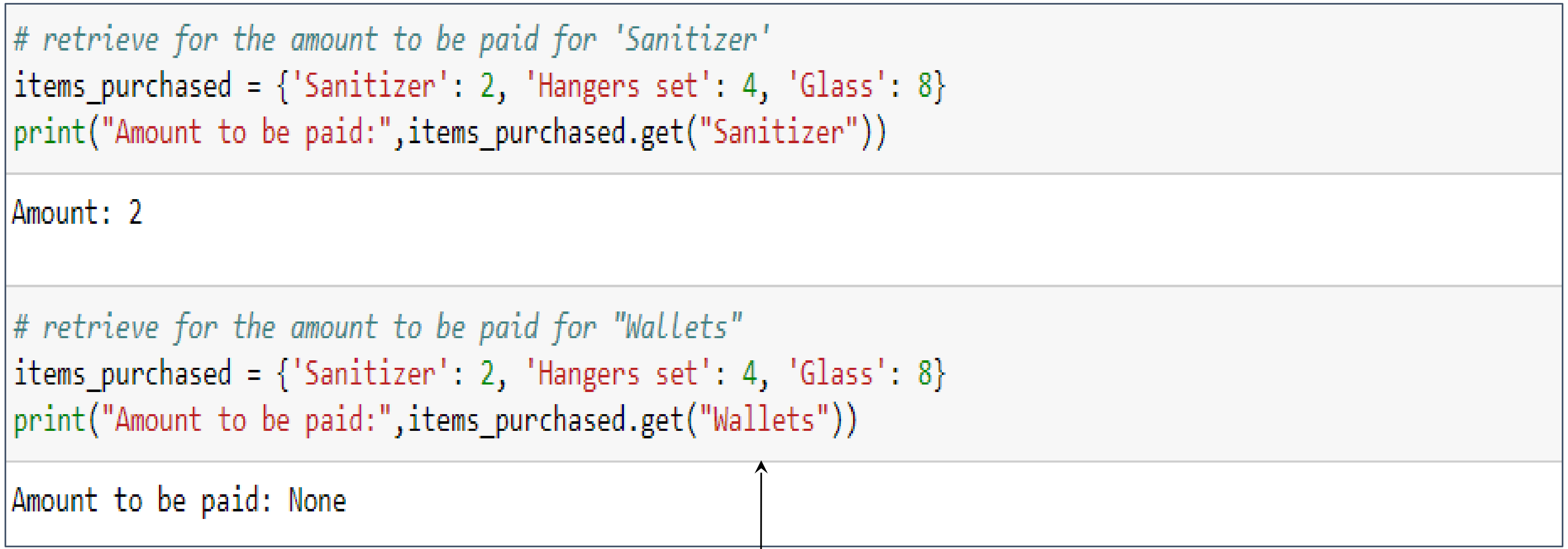
```
# get the amount for each item
items_purchased.values()
```

```
dict_values([2, 4, 8])
```

```
# items() returns the key-value pairs
items_purchased.items()
```

```
dict_items([('Sanitizer', 2), ('Hangers set', 4), ('Glass', 8)])
```

DICTIONARY OPERATIONS



```
# retrieve for the amount to be paid for 'Sanitizer'
items_purchased = {'Sanitizer': 2, 'Hangers set': 4, 'Glass': 8}
print("Amount to be paid:",items_purchased.get("Sanitizer"))
```

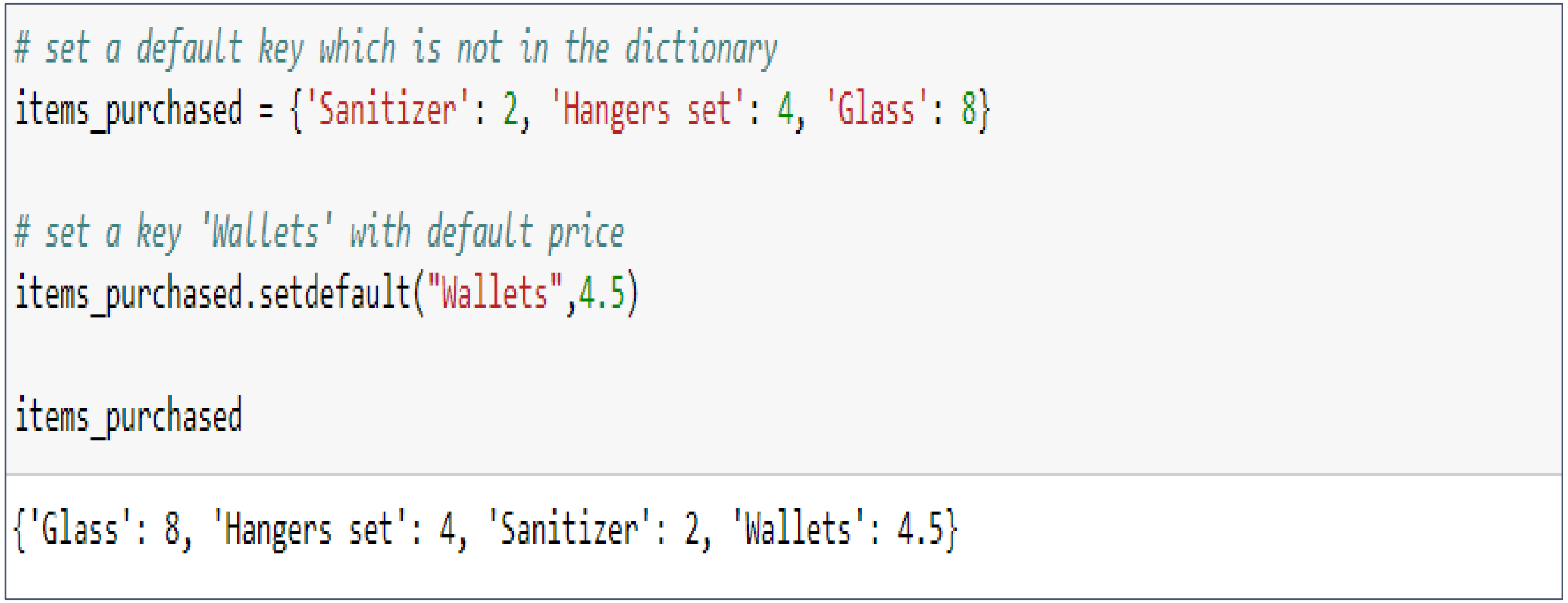
```
Amount: 2
```

```
# retrieve for the amount to be paid for "Wallets"
items_purchased = {'Sanitizer': 2, 'Hangers set': 4, 'Glass': 8}
print("Amount to be paid:",items_purchased.get("Wallets"))
```

```
Amount to be paid: None
```

There is no key called ‘Wallets’ in item_purchased. Thus, get() returns ‘None’ as output

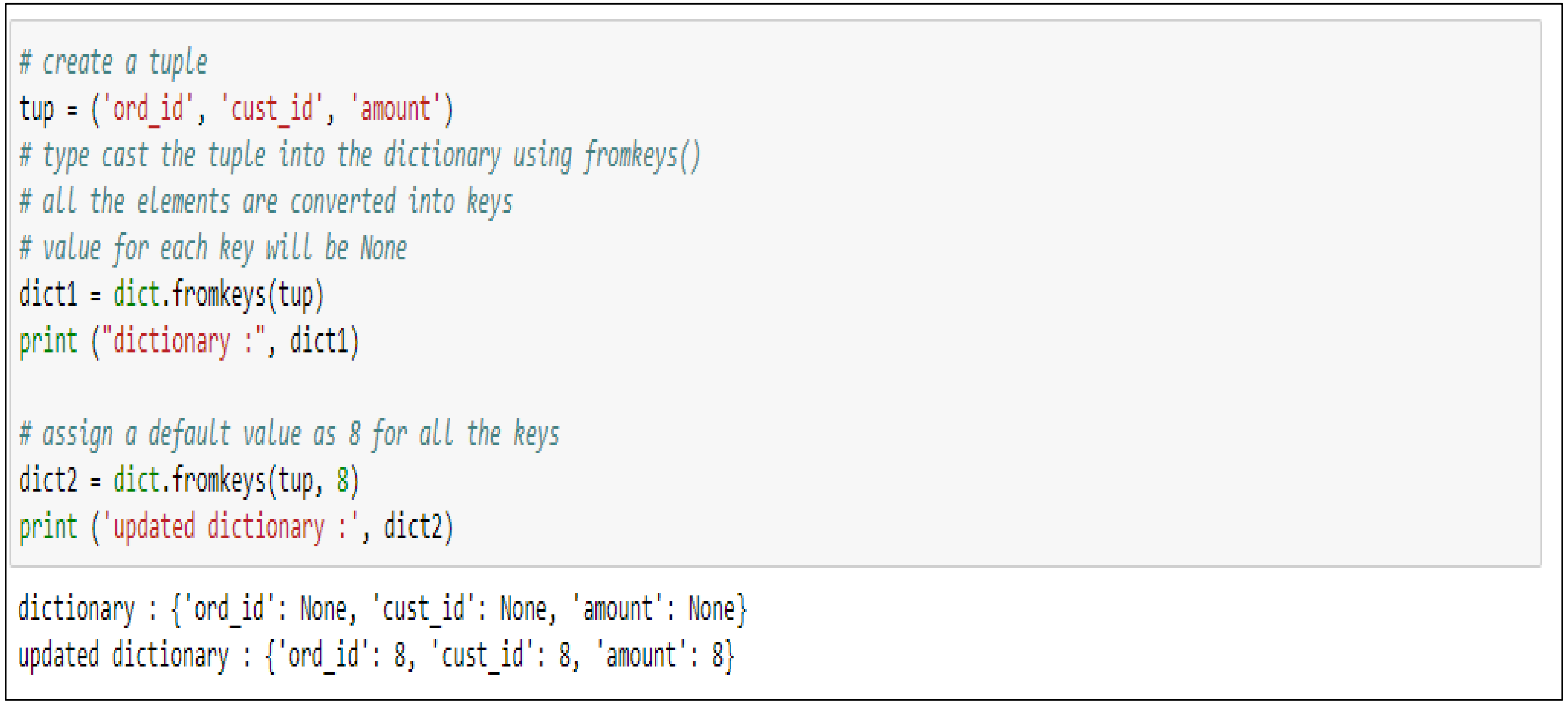
DICTIONARY OPERATIONS



```
# set a default key which is not in the dictionary
items_purchased = {'Sanitizer': 2, 'Hangers set': 4, 'Glass': 8}

# set a key 'Wallets' with default price
items_purchased.setdefault("Wallets",4.5)

items_purchased
```

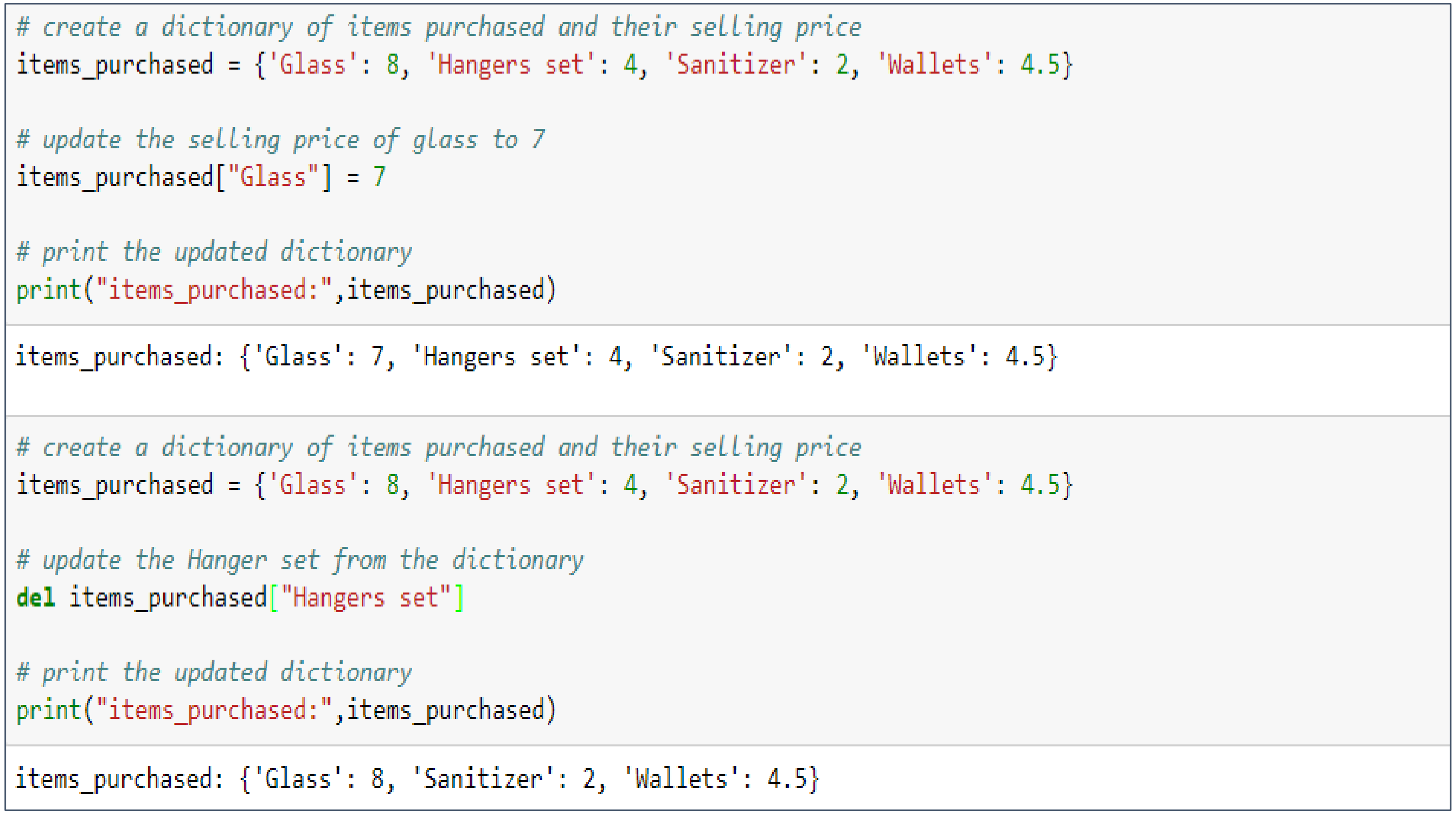
```
{'Glass': 8, 'Hangers set': 4, 'Sanitizer': 2, 'Wallets': 4.5}
```

DICTIONARY OPERATIONS


```
# create a tuple
tup = ('ord_id', 'cust_id', 'amount')
# type cast the tuple into the dictionary using fromkeys()
# all the elements are converted into keys
# value for each key will be None
dict1 = dict.fromkeys(tup)
print ("dictionary :", dict1)

# assign a default value as 8 for all the keys
dict2 = dict.fromkeys(tup, 8)
print ('updated dictionary :', dict2)
```

```
dictionary : {'ord_id': None, 'cust_id': None, 'amount': None}
updated dictionary : {'ord_id': 8, 'cust_id': 8, 'amount': 8}
```

DICTIONARY OPERATIONS



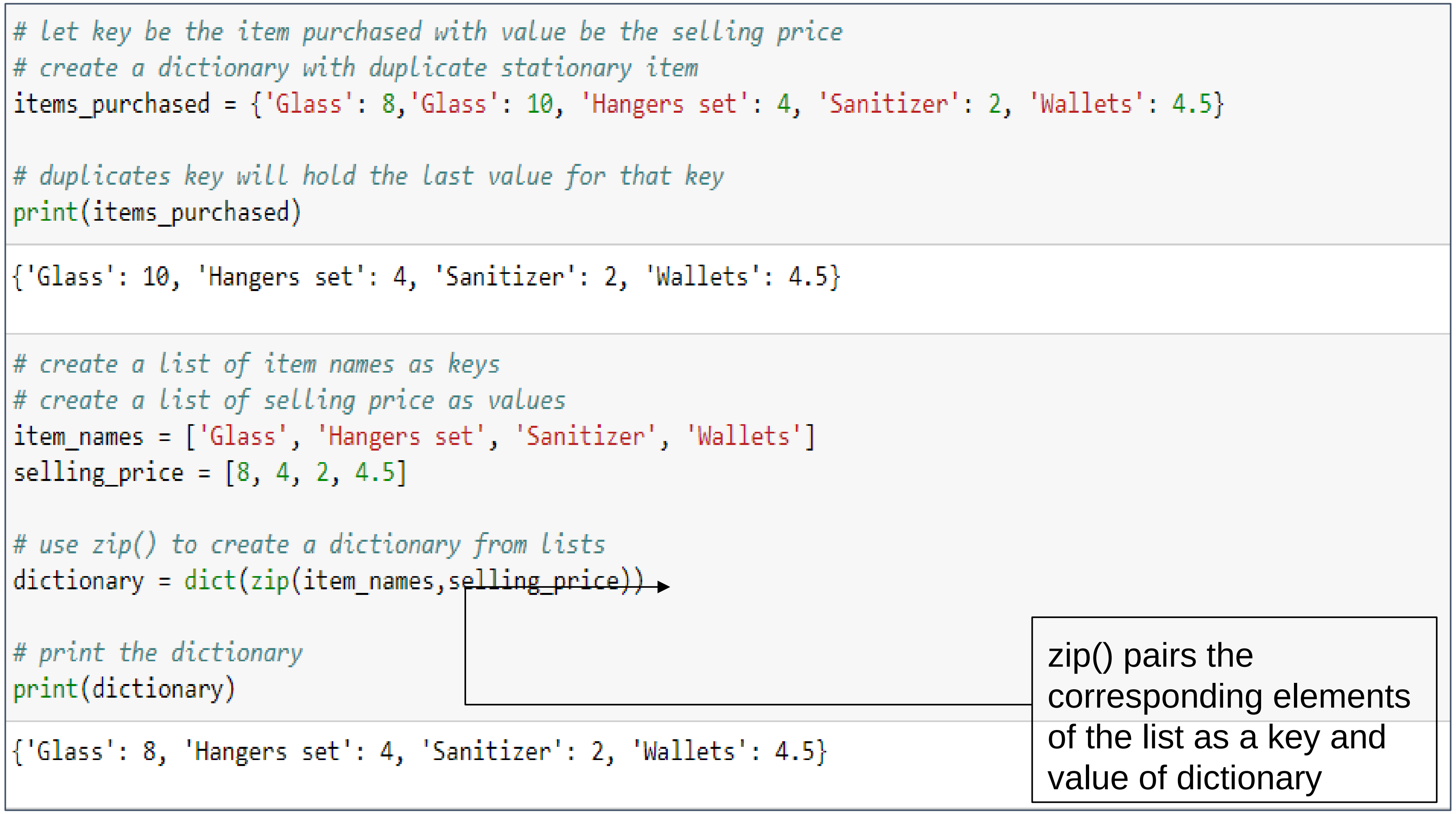
```
# create a dictionary of items purchased and their selling price
items_purchased = {'Glass': 8, 'Hangers set': 4, 'Sanitizer': 2, 'Wallets': 4.5}

# update the selling price of glass to 7
items_purchased["Glass"] = 7

# print the updated dictionary
print("items_purchased:",items_purchased)
```

```
items_purchased: {'Glass': 7, 'Hangers set': 4, 'Sanitizer': 2, 'Wallets': 4.5}
```

```
# create a dictionary of items purchased and their selling price
items_purchased = {'Glass': 8, 'Hangers set': 4, 'Sanitizer': 2, 'Wallets': 4.5}

# update the Hanger set from the dictionary
del items_purchased["Hangers set"]

# print the updated dictionary
print("items_purchased:",items_purchased)
```

```
items_purchased: {'Glass': 8, 'Sanitizer': 2, 'Wallets': 4.5}
```

DICTIONARY OPERATIONS



```python
# let key be the item purchased with value be the selling price
# create a dictionary with duplicate stationary item
items_purchased = {'Glass': 8,'Glass': 10, 'Hangers set': 4, 'Sanitizer': 2, 'Wallets': 4.5}

# duplicates key will hold the last value for that key
print(items_purchased)
```

```
{'Glass': 10, 'Hangers set': 4, 'Sanitizer': 2, 'Wallets': 4.5}
```

```python
# create a list of item names as keys
# create a list of selling price as values
item_names = ['Glass', 'Hangers set', 'Sanitizer', 'Wallets']
selling_price = [8, 4, 2, 4.5]

# use zip() to create a dictionary from lists
dictionary = dict(zip(item_names,selling_price))

# print the dictionary
print(dictionary)
```

```
{'Glass': 8, 'Hangers set': 4, 'Sanitizer': 2, 'Wallets': 4.5}
```

zip() pairs the corresponding elements of the list as a key and value of dictionary

Did You Know?

A tuple is immutable. However, if the tuple contains a list or a dictionary inside it, then the list or the dictionary that is contained within the tuple can be modified.

List Comprehension

WHAT IS LIST COMPREHENSION?

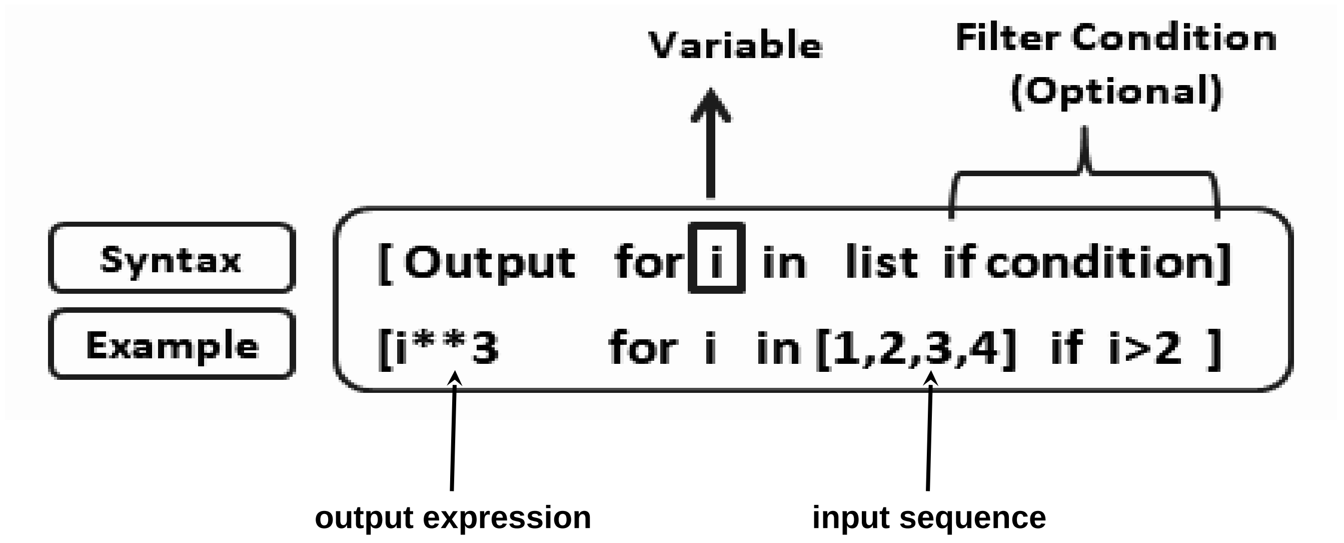
A list comprehension provides a concise way to create a list from another iterable

They consist of brackets containing the expression

It is possible to use conditions and loop in a list comprehension

The output is always a list

WHAT IS LIST COMPREHENSION?


Variable
Filter Condition (Optional)
Syntax
Example
[Output for i in list if condition]
[i**3 for i in [1,2,3,4] if i>2]
output expression
input sequence

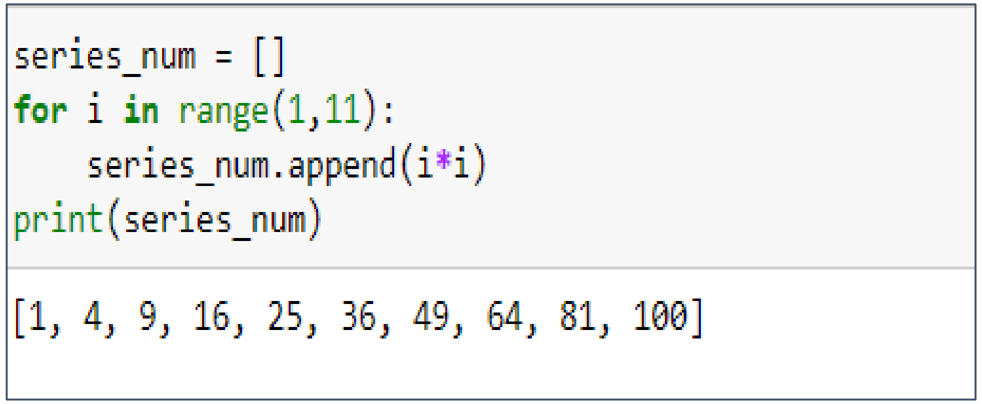
It consists of:

- An output expression sequence
- A for loop statement that iterates through the iterable
- a conditional expression (optional)

LIST COMPREHENSION vs LOOPS

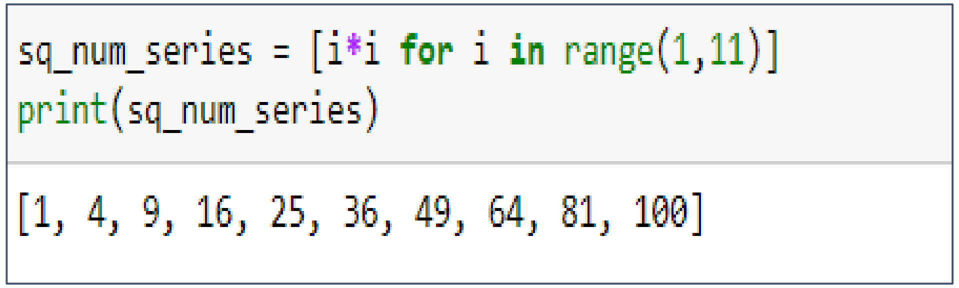
Creating a list using for loop

Creating a list using list comprehension



```
series_num = []
for i in range(1,11):
    series_num.append(i*i)
print(series_num)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```



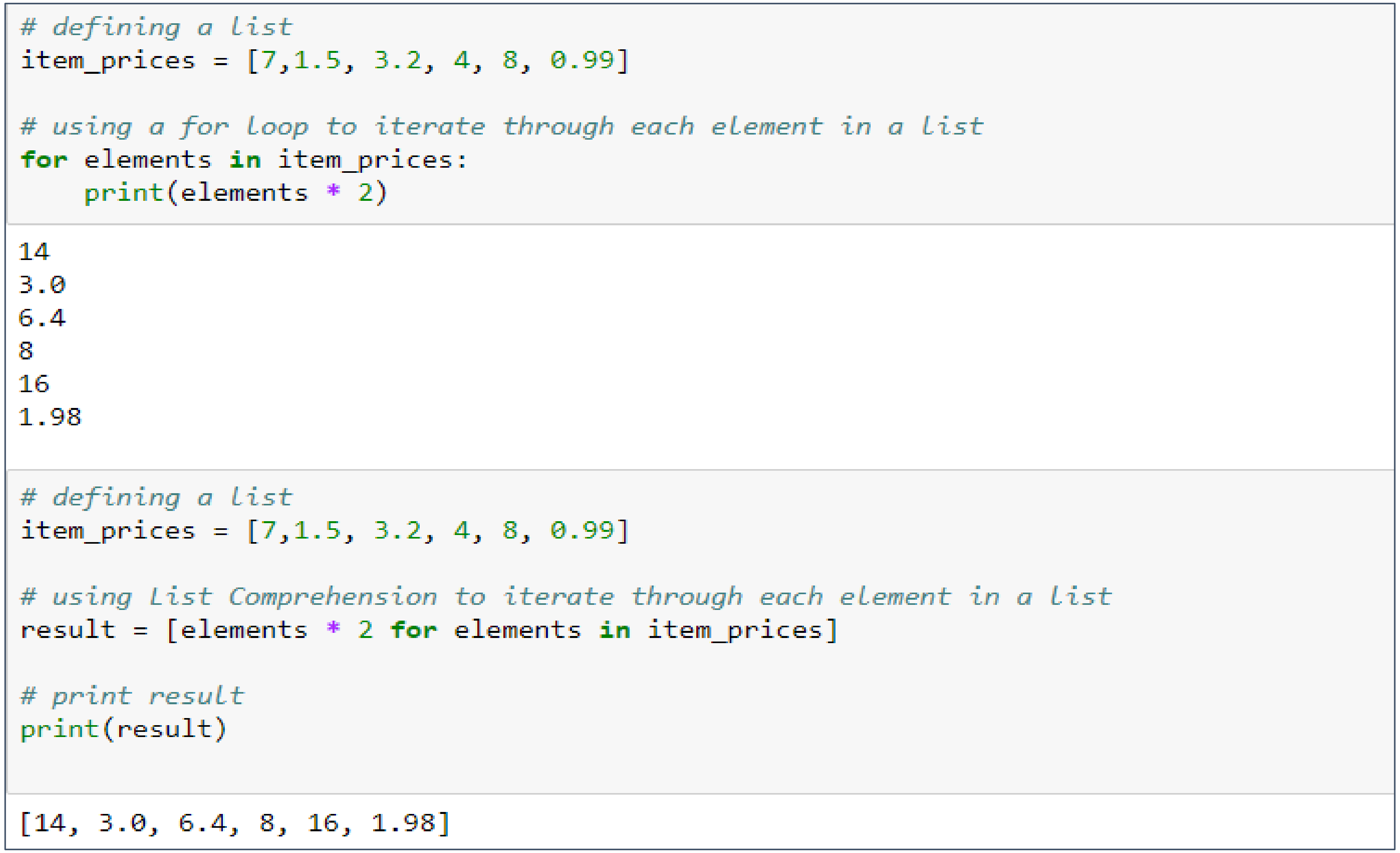
```
sq_num_series = [i*i for i in range(1,11)]
print(sq_num_series)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Create empty list and append squared values one by one using for loop

Create a list using list comprehension

FOR LOOP IN LIST COMPREHENSION



```
# defining a list
item_prices = [7,1.5, 3.2, 4, 8, 0.99]

# using a for loop to iterate through each element in a list
for elements in item_prices:
    print(elements * 2)
```

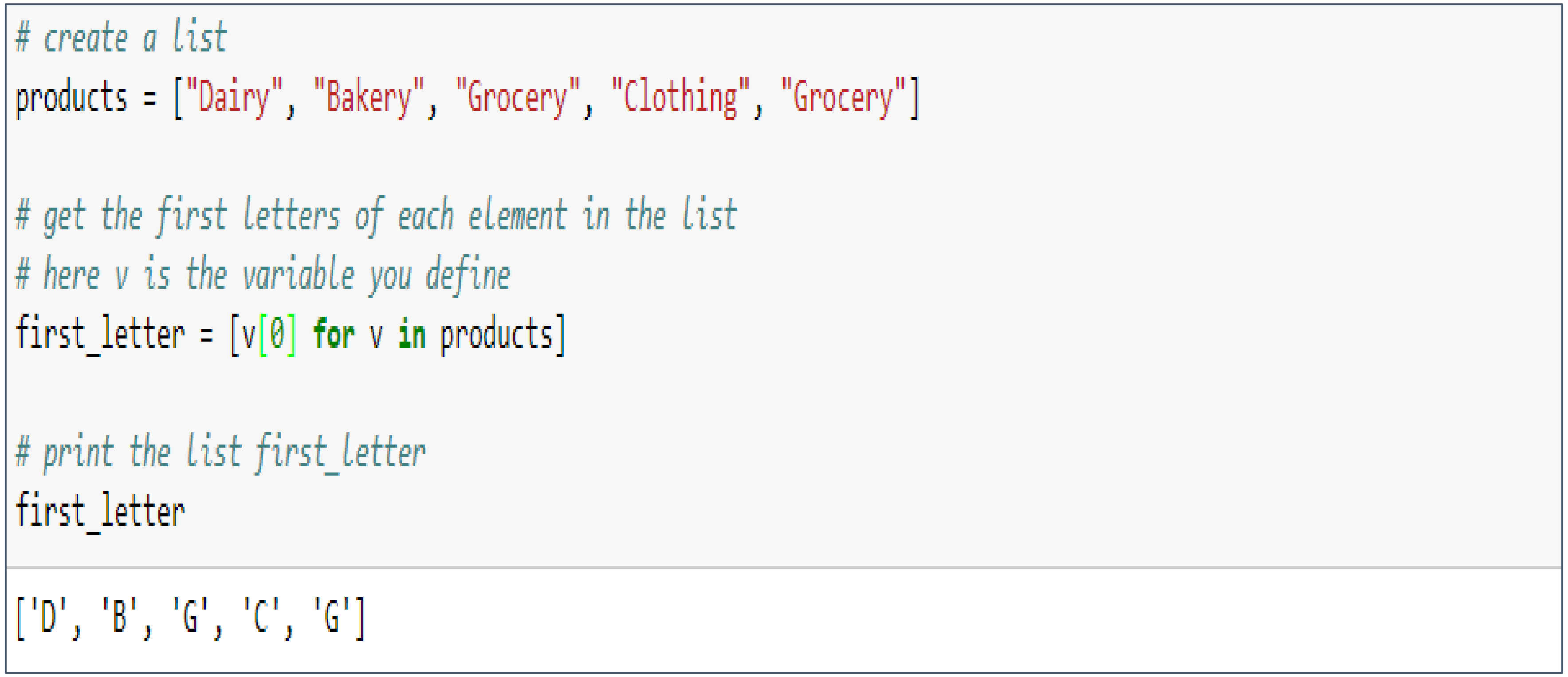
```
14
3.0
6.4
8
16
1.98
```

```
# defining a list
item_prices = [7,1.5, 3.2, 4, 8, 0.99]

# using List Comprehension to iterate through each element in a list
result = [elements * 2 for elements in item_prices]

# print result
print(result)
```

```
[14, 3.0, 6.4, 8, 16, 1.98]
```

LIST COMPREHENSION: EXAMPLES

Reading first letter of each word in a list

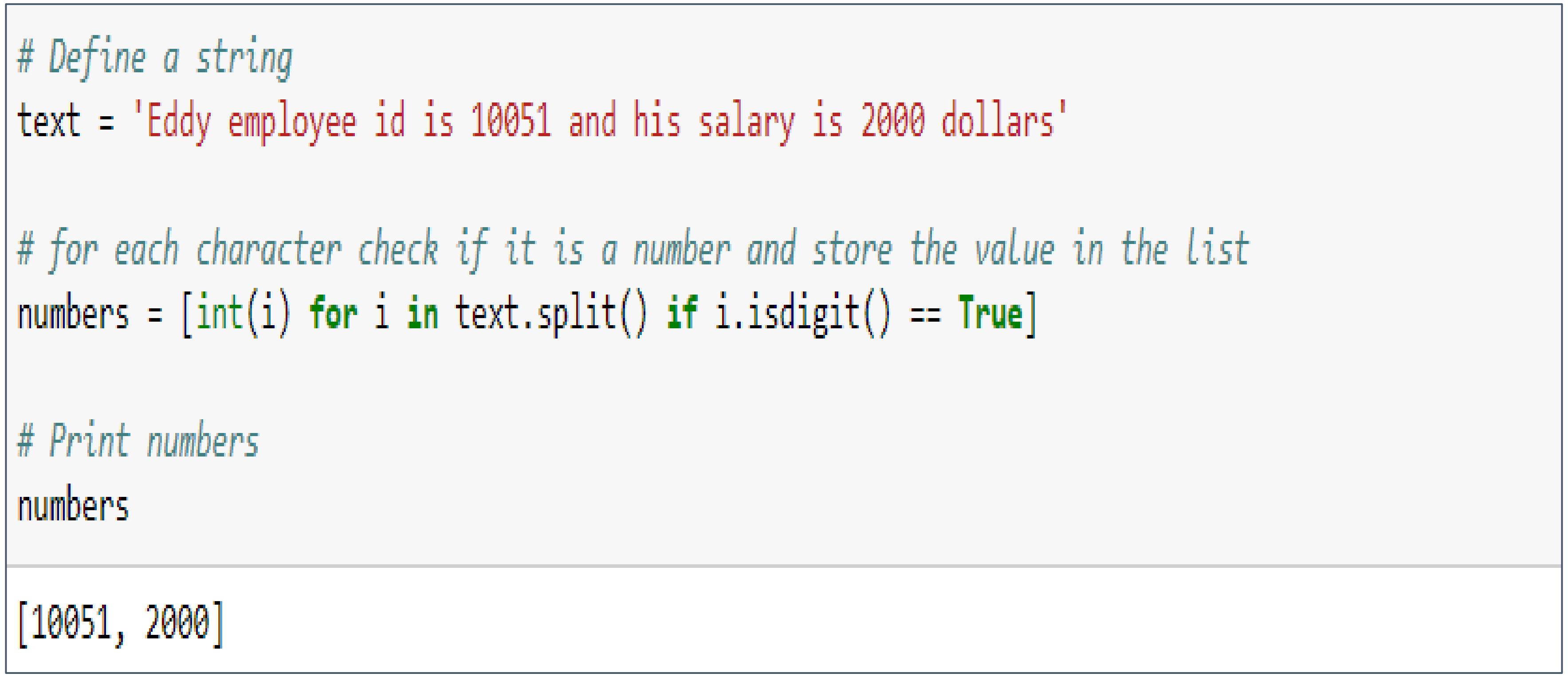


```
# create a list
products = ["Dairy", "Bakery", "Grocery", "Clothing", "Grocery"]

# get the first letters of each element in the list
# here v is the variable you define
first_letter = [v[0] for v in products]

# print the list first_letter
first_letter
```

```
['D', 'B', 'G', 'C', 'G']
```

LIST COMPREHENSION: EXAMPLES

Extract all the numbers from the string using list comprehension

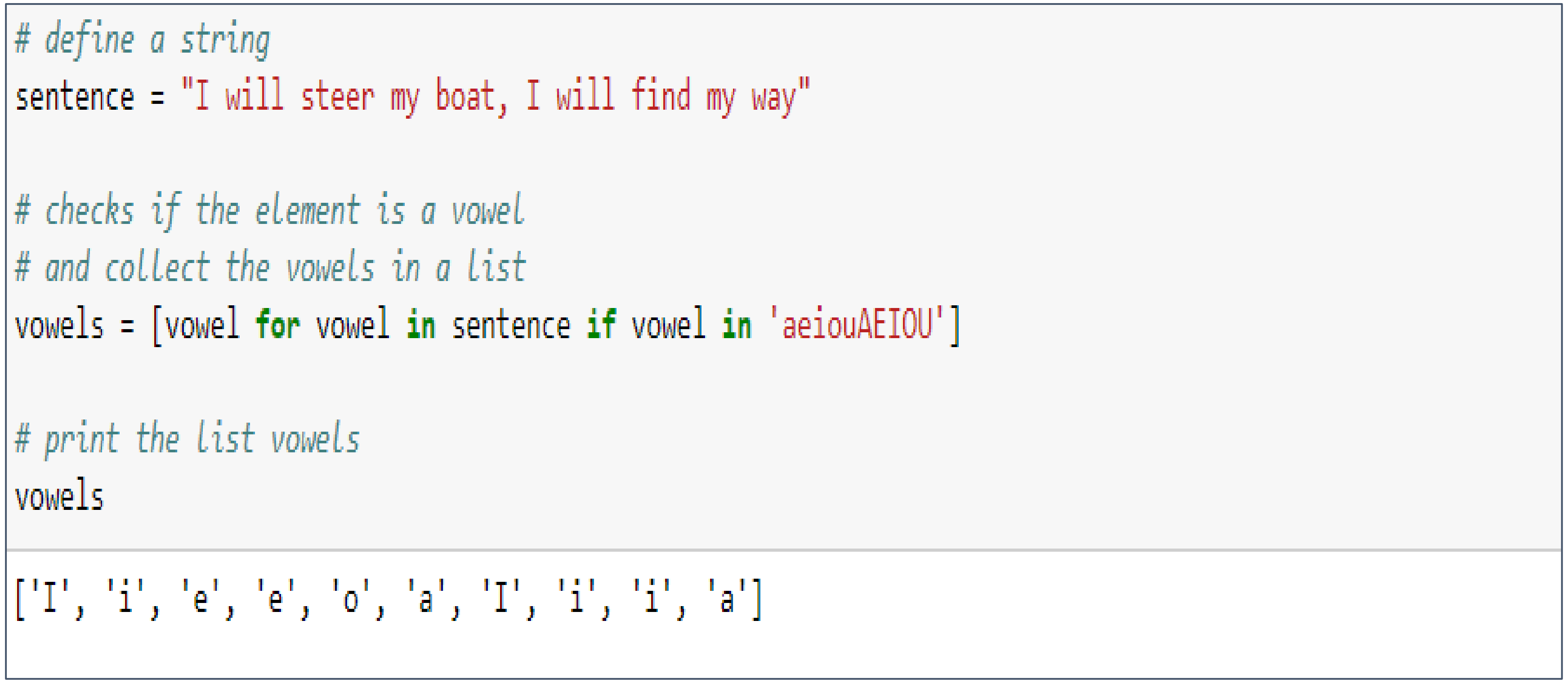


```python
# Define a string
text = 'Eddy employee id is 10051 and his salary is 2000 dollars'

# for each character check if it is a number and store the value in the list
numbers = [int(i) for i in text.split() if i.isdigit() == True]

# Print numbers
numbers
```

```
[10051, 2000]
```

LIST COMPREHENSION: EXAMPLES

Extracting vowels from the string

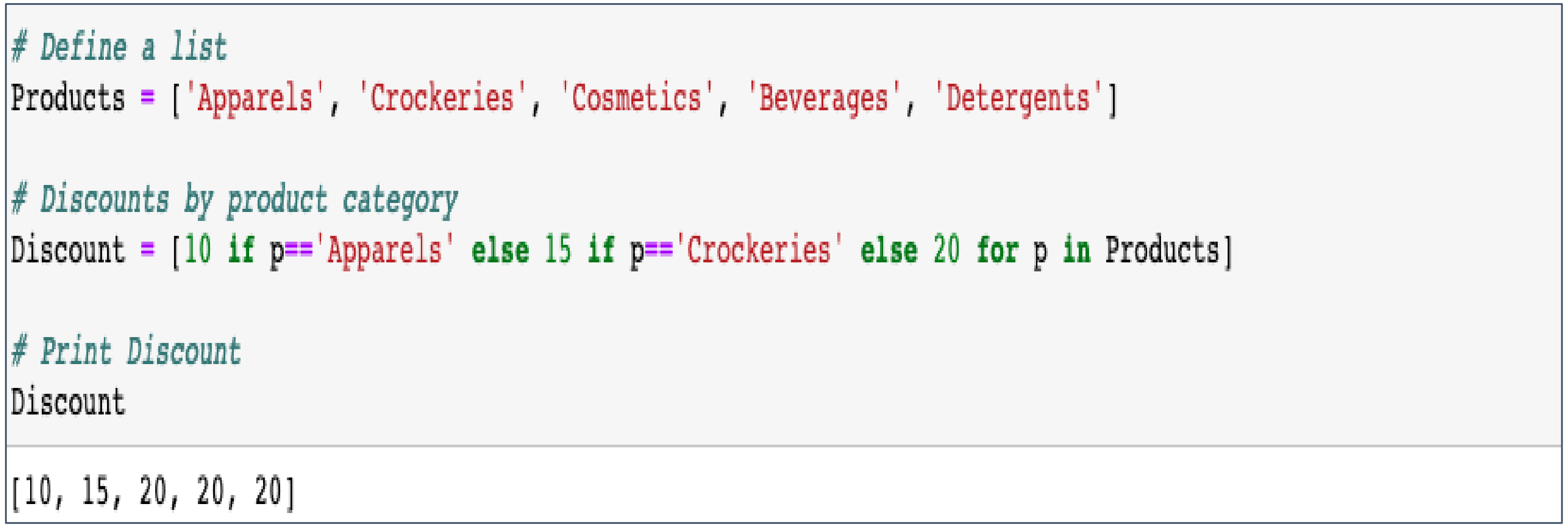


```
# define a string
sentence = "I will steer my boat, I will find my way"

# checks if the element is a vowel
# and collect the vowels in a list
vowels = [vowel for vowel in sentence if vowel in 'aeiouAEIOU']

# print the list vowels
vowels
```

```
['I', 'i', 'e', 'e', 'o', 'a', 'I', 'i', 'i', 'a']
```

LIST COMPREHENSION: EXAMPLES

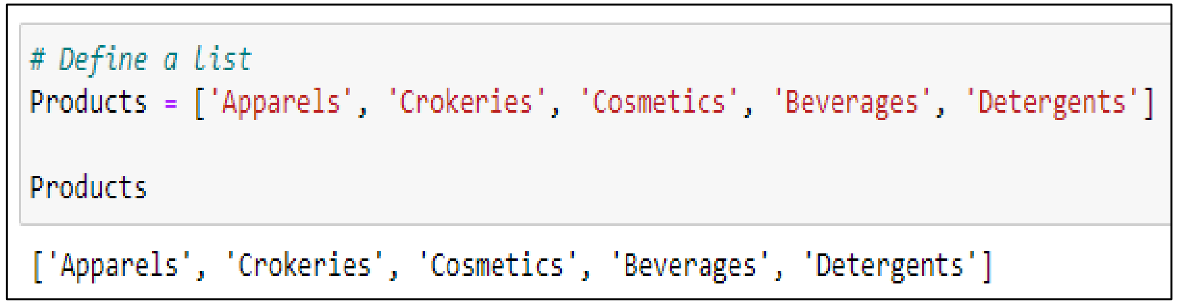
Using nested conditional statements in list comprehension



```
# Define a list
Products = ['Apparels', 'Crockeries', 'Cosmetics', 'Beverages', 'Detergents']

# Discounts by product category
Discount = [10 if p=='Apparels' else 15 if p=='Crockeries' else 20 for p in Products]

# Print Discount
Discount
```

```
[10, 15, 20, 20, 20]
```

Find strings in a list that are longer than 9 character using list comprehension



```
# Define a list
Products = ['Apparels', 'Crokeries', 'Cosmetics', 'Beverages', 'Detergents']

Products
```

```
['Apparels', 'Crokeries', 'Cosmetics', 'Beverages', 'Detergents']
```

Did You Know?

- A list comprehension helps reduce three lines of code into a single line of code.
- Python allocates the list's memory before appending the elements to the list. This helps avoid resize of the memory at runtime. This makes a list comprehension faster.

We're committed to empower you to be
#FutureReady
through powerful training solutions.

IMARTICUS
LEARNING

We build the workforce of the future.

250+	30,000+	25000+
Corporate Clients	Learners Trained	Learners Placed

FREE WEBINARS

Sign up for free webinars with industry experts every fortnight!

Enroll Now!

Contact us today!
https://imarticus.org/