Beginner Programming Fundamentals With Python

Module 6

List, Tuple, Dictionary and Set

Ali Samanipour

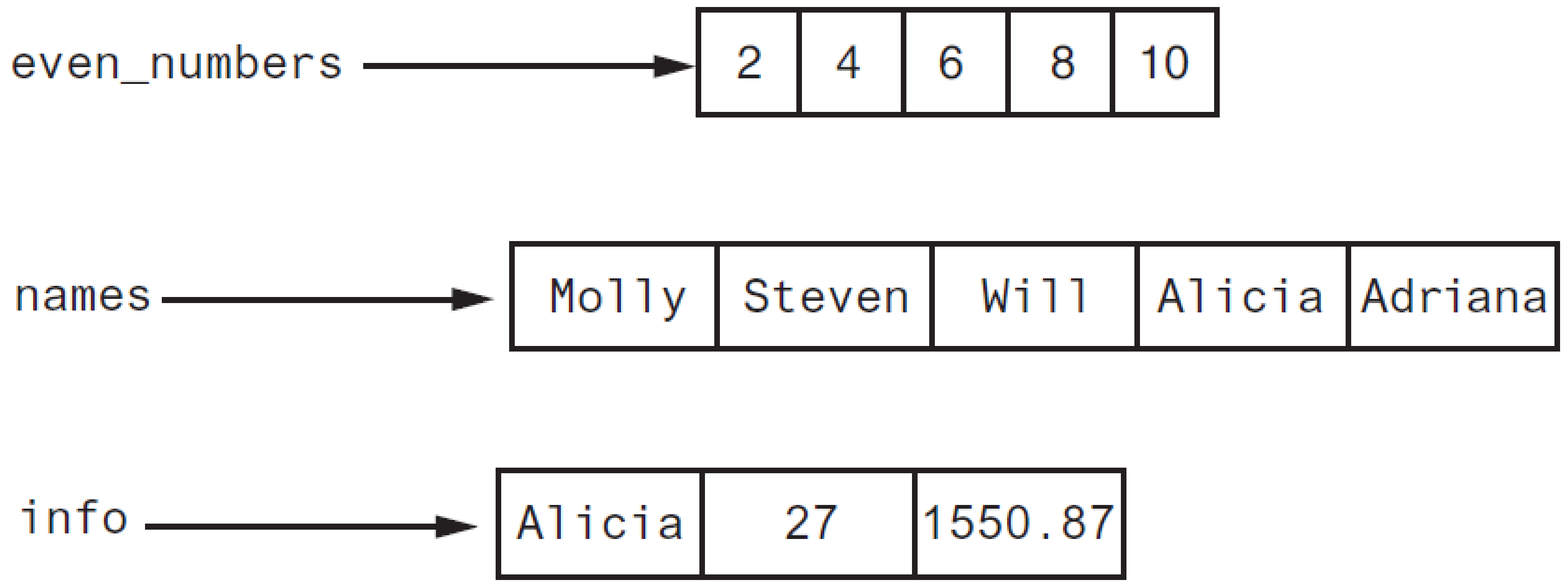
Samani Academy
Apadana OTC
Jan 2022

Module Roadmap

1. List
2. Tuples
3. Dictionaries
4. Sets

Sequences

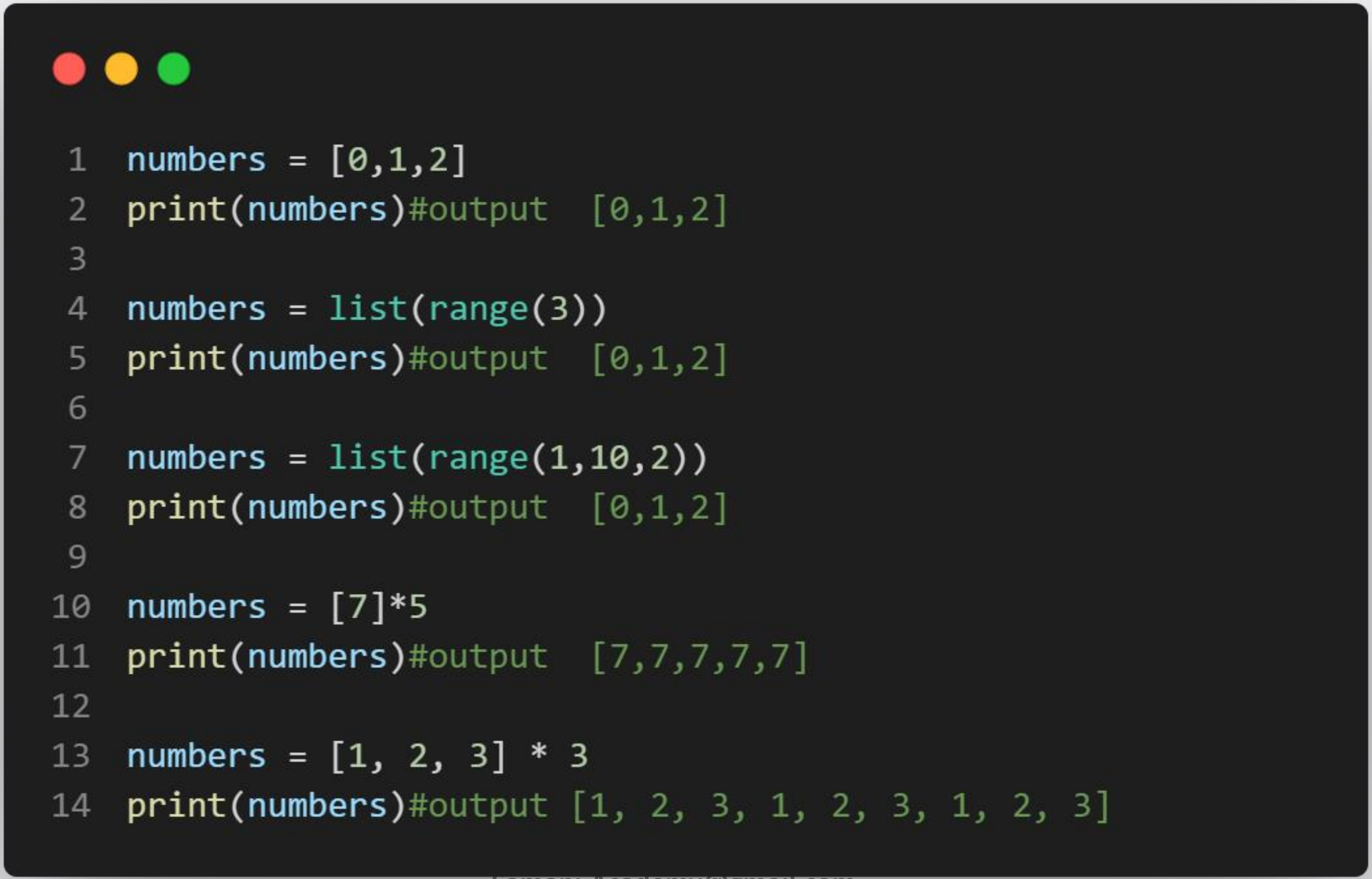
A sequence is an object that holds multiple items of data, stored one after the other. You can perform operations on a sequence to examine and manipulate the items stored in it



Wrap-up

List	[‘Milk’ , ’Honey’ , ‘Milk’]	Mutable, ordered list, duplicates allowed, mostly only one type
Set	{‘Milk’ , ’Honey’}	Mutable, unordered list, duplicates not allowed, mostly only one type
Tuple	(‘Milk’ , ’Honey’)	Immutable, ordered list, duplicates allowed, Often mixed type
Dictionary	{‘name’: ”Ali”, ‘age’ : 25}	Mutable, unordered map, No duplicate keys, Often mixed type

Creating a Lists



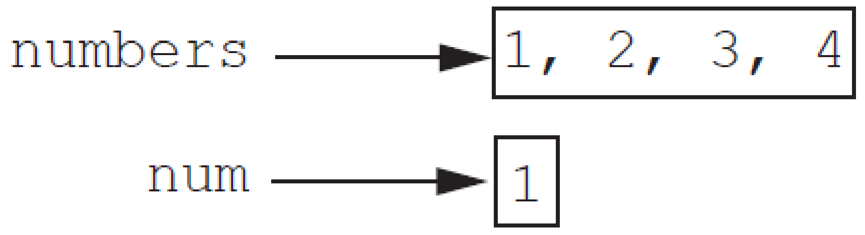
```
numbers = [0,1,2]
print(numbers)#output  [0,1,2]

numbers = list(range(3))
print(numbers)#output  [0,1,2]

numbers = list(range(1,10,2))
print(numbers)#output  [0,1,2]

numbers = [7]*5
print(numbers)#output  [7,7,7,7,7]

numbers = [1, 2, 3] * 3
print(numbers)#output [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Iterating over a List with the for Loop

numbers = [1,2,3,4]

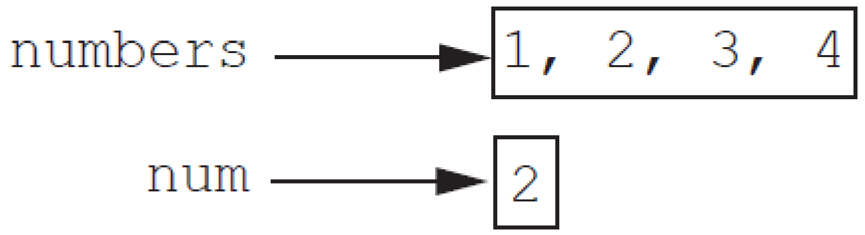
1st Iteration

```
for num in numbers:
    print(num)
```



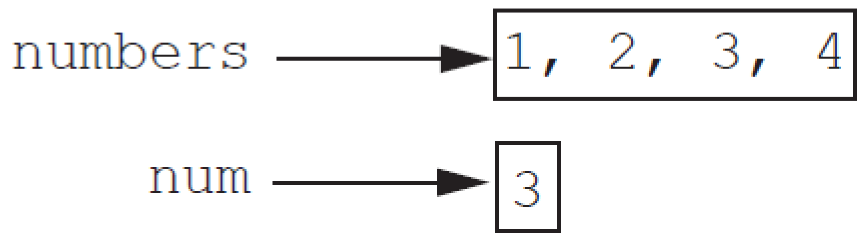
2nd Iteration

```
for num in numbers:
    print(num)
```



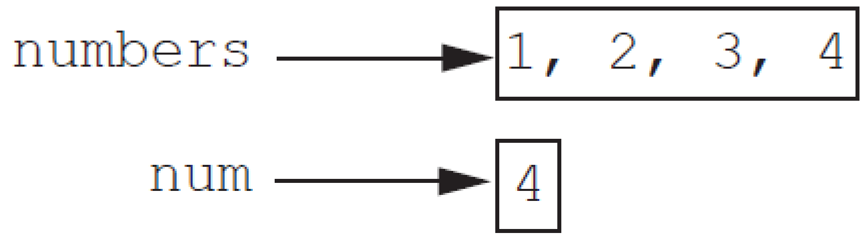
3rd Iteration

```
for num in numbers:
    print(num)
```

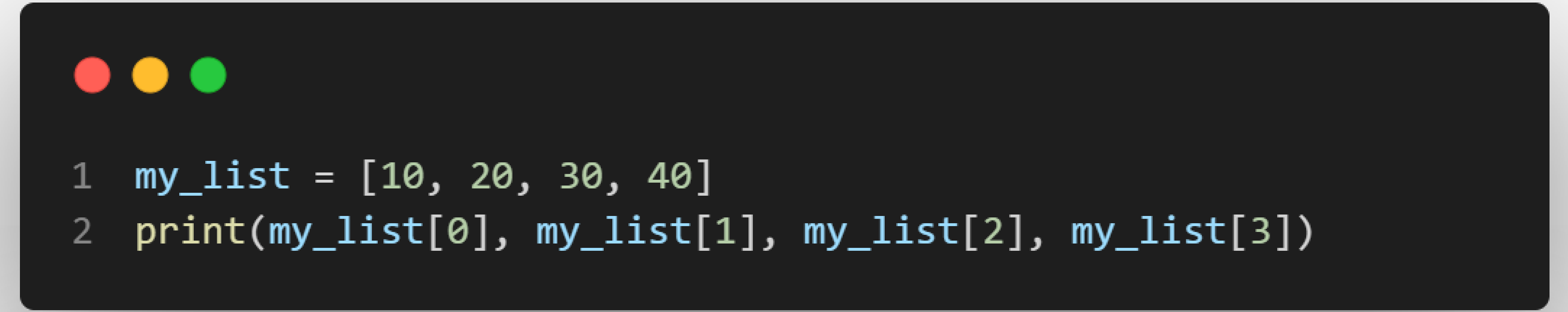


4th Iteration

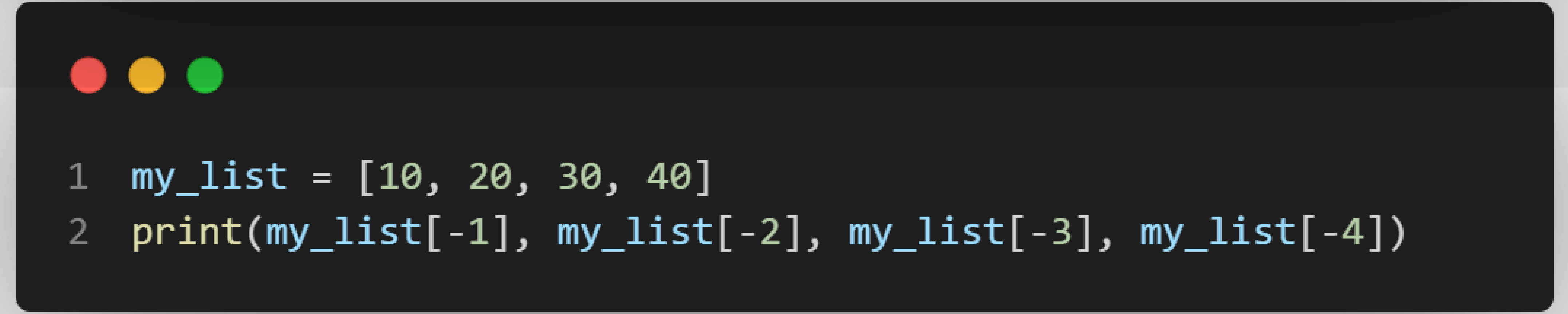
```
for num in numbers:
    print(num)
```



Iterating over a List
Indexing



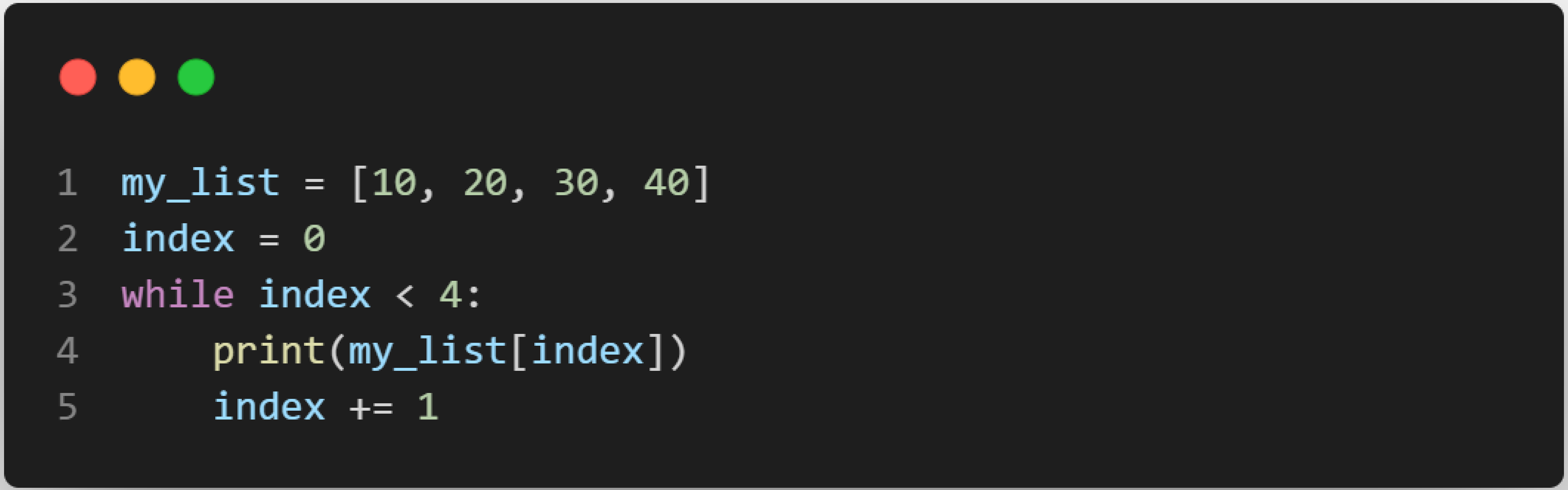
```python
my_list = [10, 20, 30, 40]
print(my_list[0], my_list[1], my_list[2], my_list[3])
```



```python
my_list = [10, 20, 30, 40]
print(my_list[-1], my_list[-2], my_list[-3], my_list[-4])
```

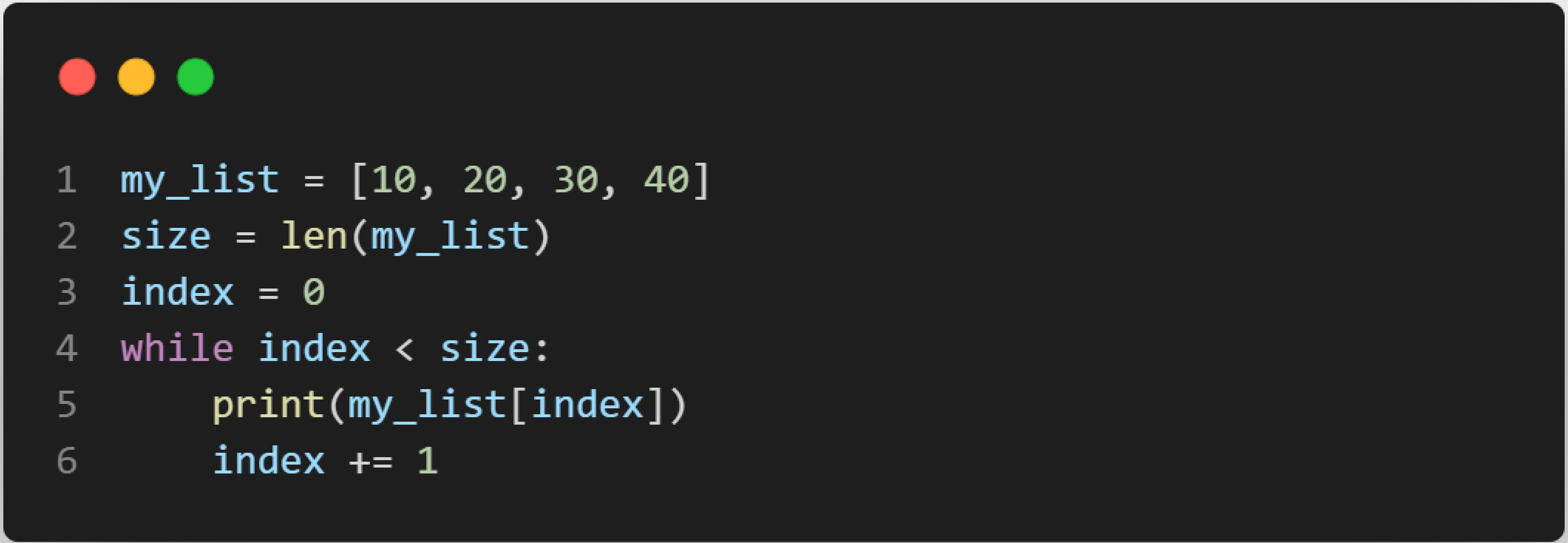
Ali Samanipour
Samani.Academy@gmail.com

Iterating over a List
Indexing



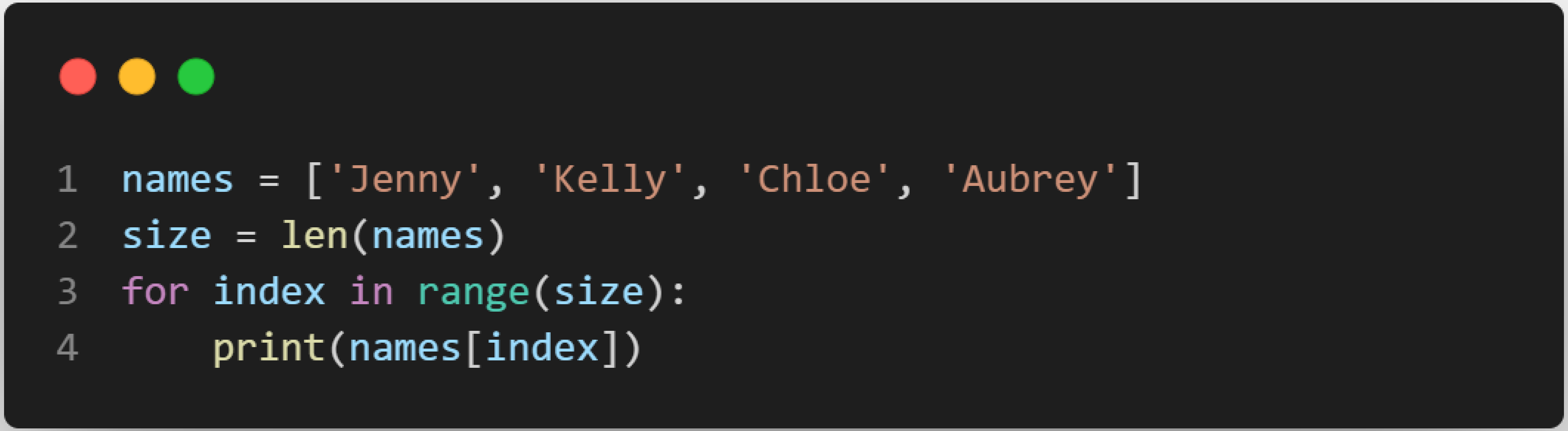
```
my_list = [10, 20, 30, 40]
index = 0
while index < 4:
    print(my_list[index])
    index += 1
```

Iterating over a List
len Function



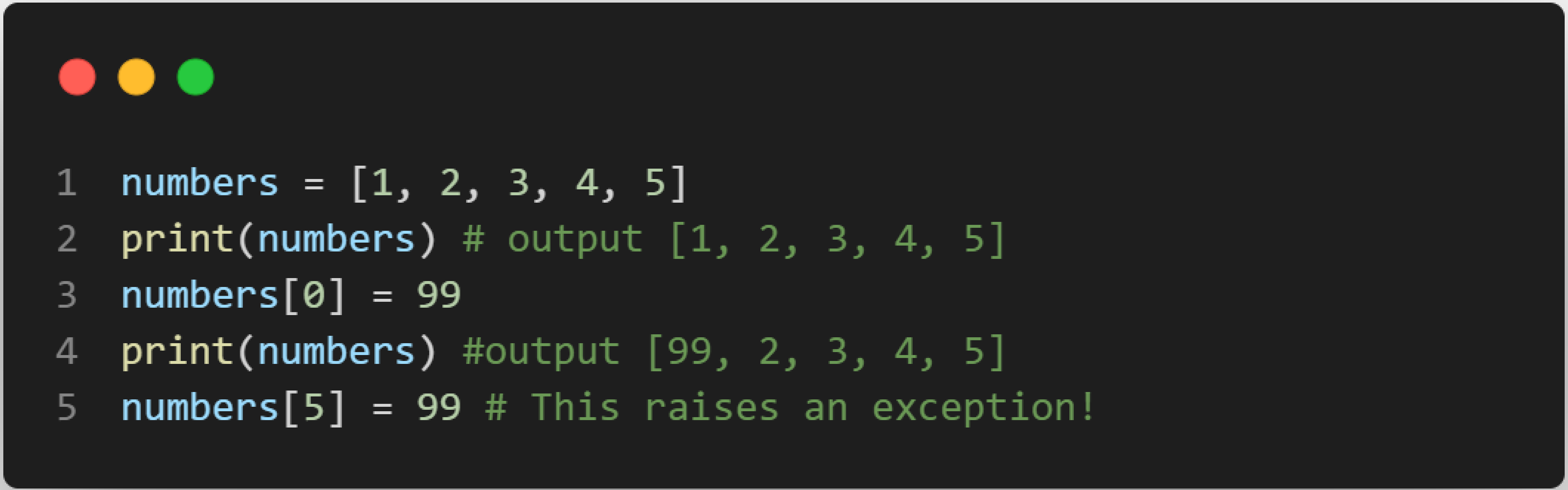
```python
my_list = [10, 20, 30, 40]
size = len(my_list)
index = 0
while index < size:
    print(my_list[index])
    index += 1
```

Iterating over a List
Using a for Loop to Iterate by Index Over a List



```python
names = ['Jenny', 'Kelly', 'Chloe', 'Aubrey']
size = len(names)
for index in range(size):
    print(names[index])
```

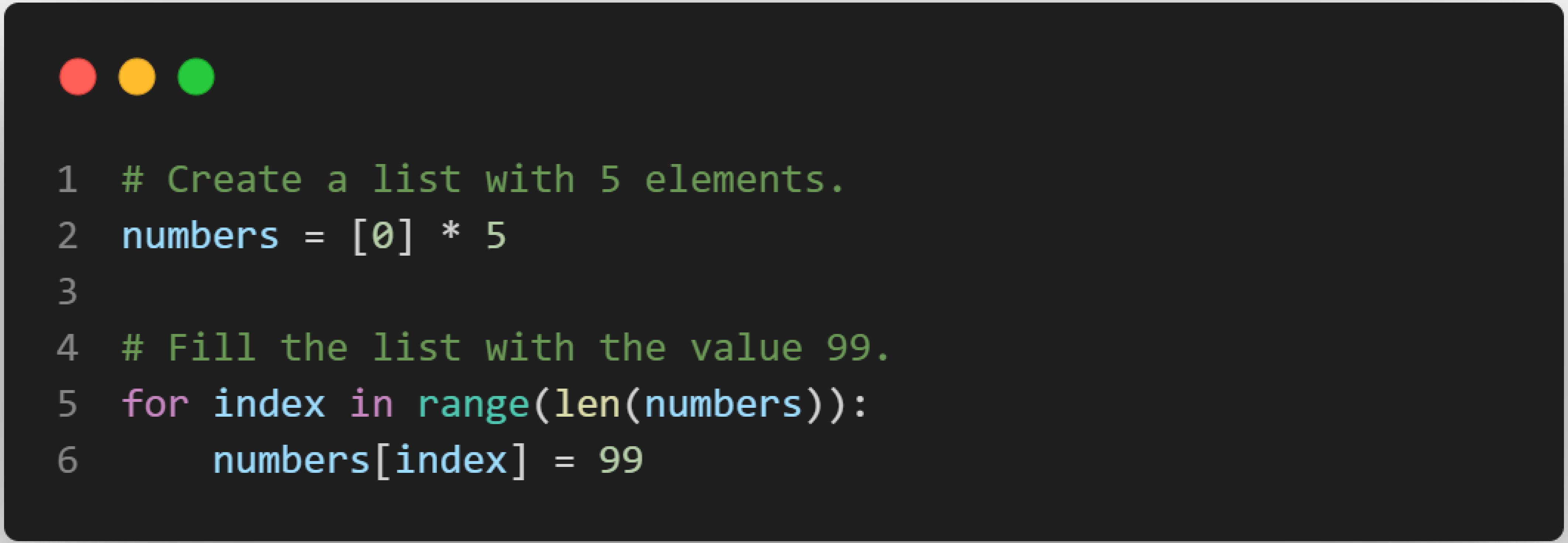
Lists Are Mutable



```python
numbers = [1, 2, 3, 4, 5]
print(numbers) # output [1, 2, 3, 4, 5]
numbers[0] = 99
print(numbers) #output [99, 2, 3, 4, 5]
numbers[5] = 99 # This raises an exception!
```

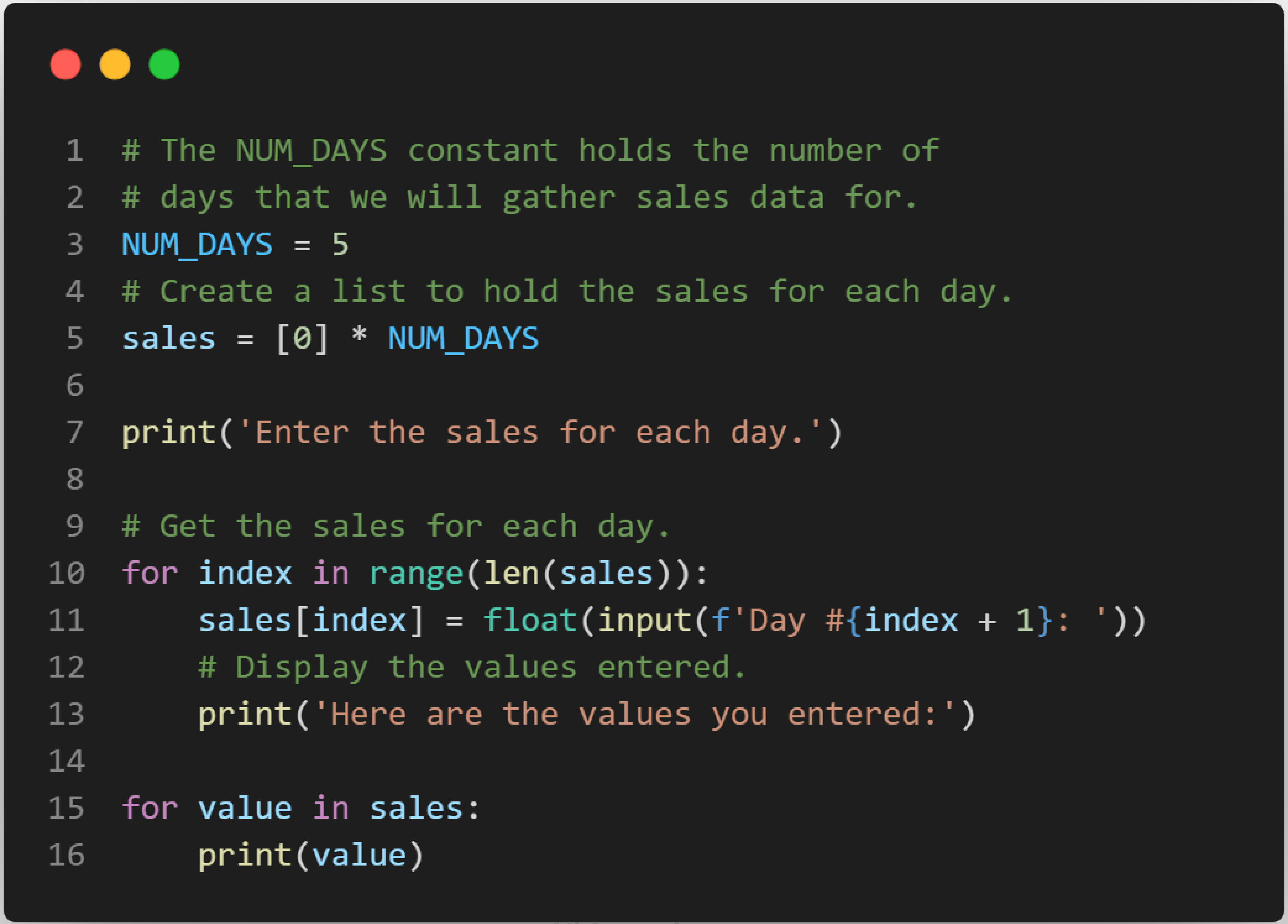
Ali Samanipour
Samani.Academy@gmail.com

Lists Are Mutable



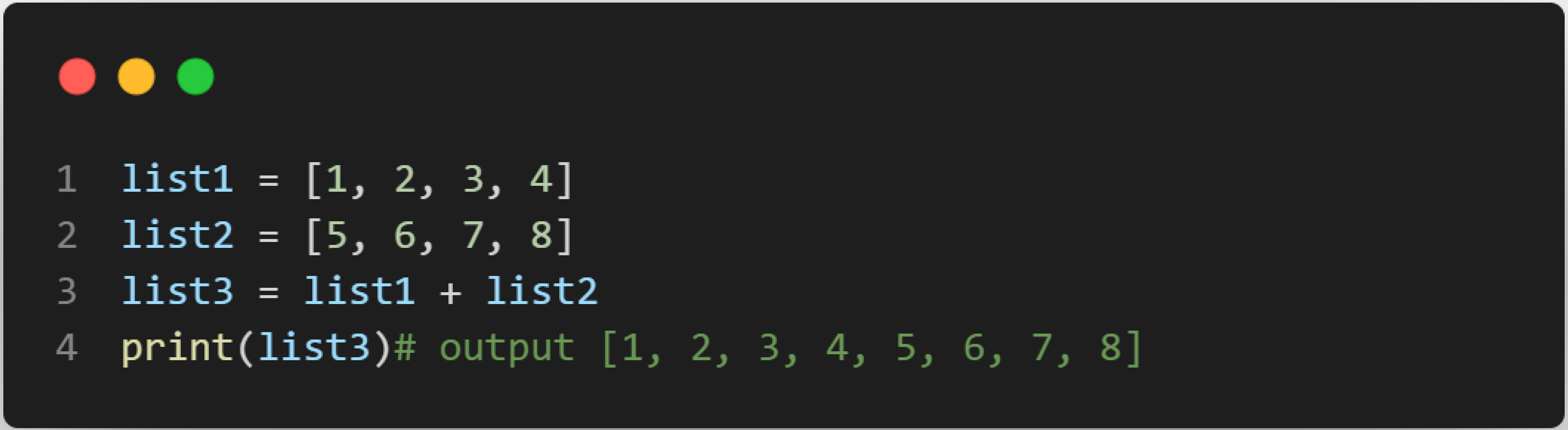
```python
# Create a list with 5 elements.
numbers = [0] * 5

# Fill the list with the value 99.
for index in range(len(numbers)):
    numbers[index] = 99
```

Lets try



```python
# The NUM_DAYS constant holds the number of
# days that we will gather sales data for.
NUM_DAYS = 5
# Create a list to hold the sales for each day.
sales = [0] * NUM_DAYS

print('Enter the sales for each day.')

# Get the sales for each day.
for index in range(len(sales)):
    sales[index] = float(input(f'Day #{index + 1}: '))
    # Display the values entered.
    print('Here are the values you entered:')

for value in sales:
    print(value)
```

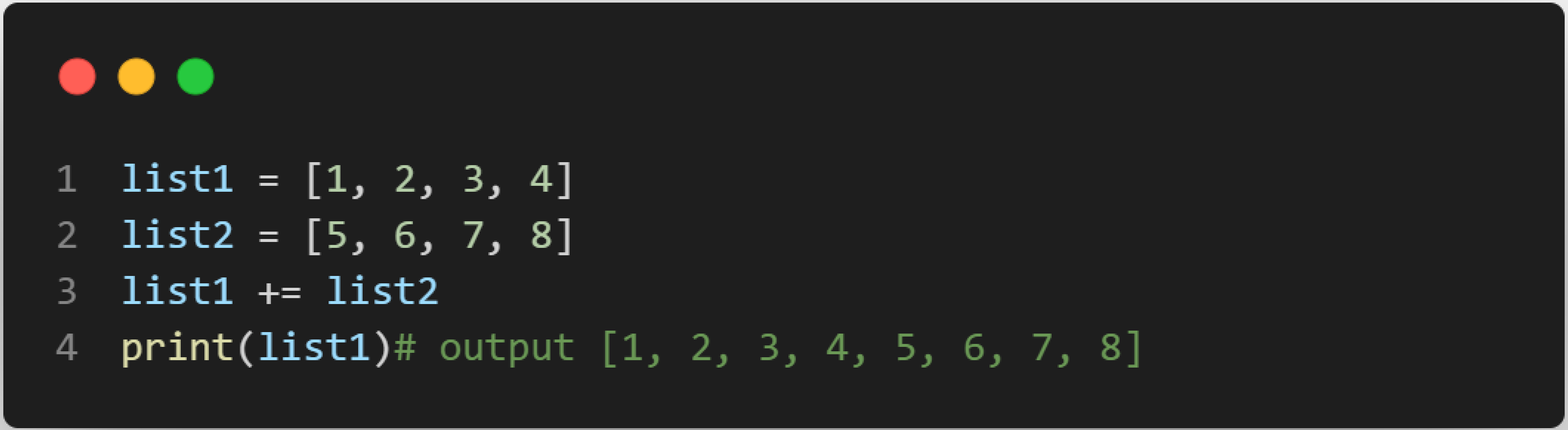
Concatenating Lists



```python
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
list3 = list1 + list2
print(list3)# output [1, 2, 3, 4, 5, 6, 7, 8]
```

Ali Samanipour
Samani.Academy@gmail.com

Concatenating Lists

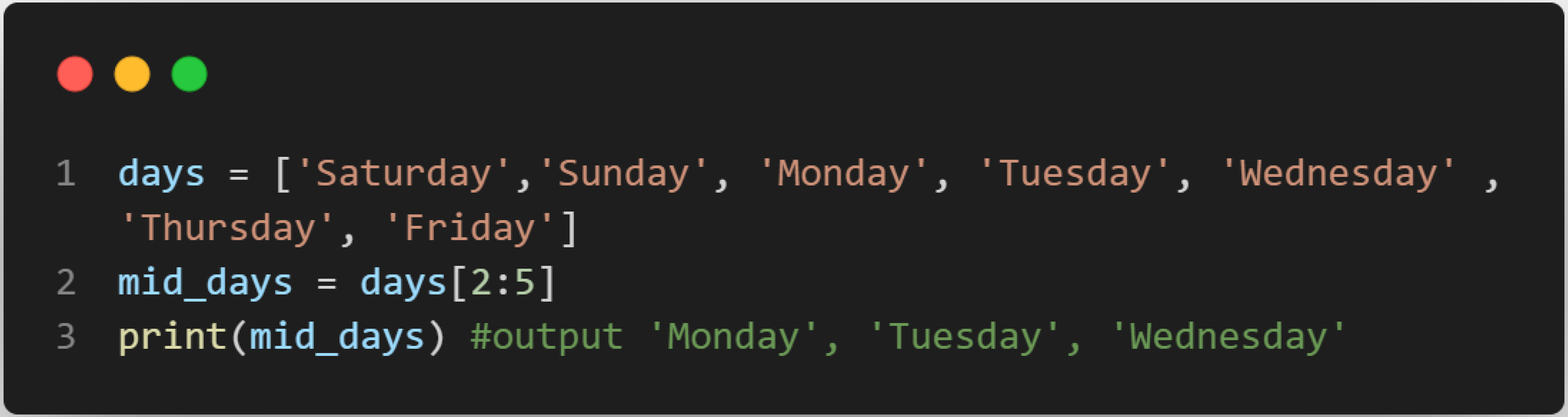


```python
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
list1 += list2
print(list1)# output [1, 2, 3, 4, 5, 6, 7, 8]
```

List Slicing

A slicing expression selects a range of elements from a sequence.

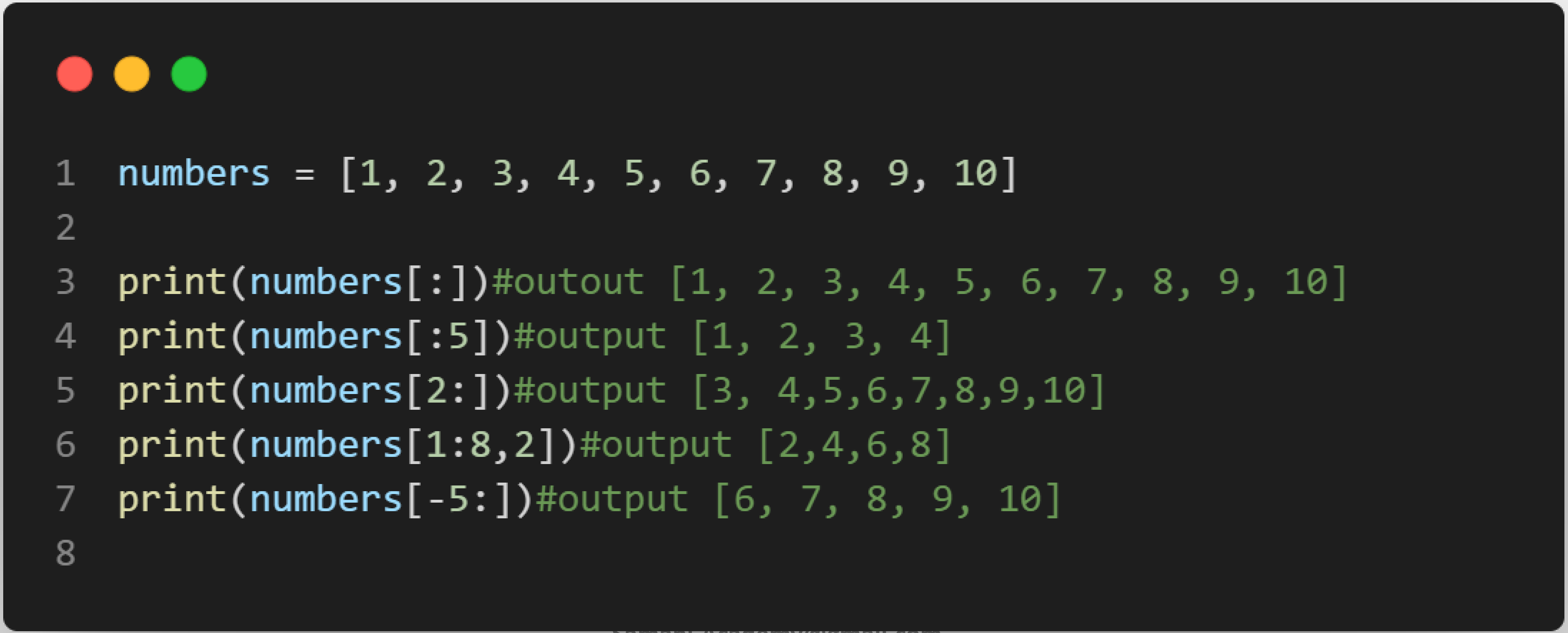
list_name[start : end]



```python
days = ['Saturday','Sunday', 'Monday', 'Tuesday', 'Wednesday' , 'Thursday', 'Friday']
mid_days = days[2:5]
print(mid_days) #output 'Monday', 'Tuesday', 'Wednesday'
```

List Slicing

A slicing expression selects a range of elements from a sequence.



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

print(numbers[:])#outout [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(numbers[:5])#output [1, 2, 3, 4]
print(numbers[2:])#output [3, 4,5,6,7,8,9,10]
print(numbers[1:8,2])#output [2,4,6,8]
print(numbers[-5:])#output [6, 7, 8, 9, 10]
```

List Slicing

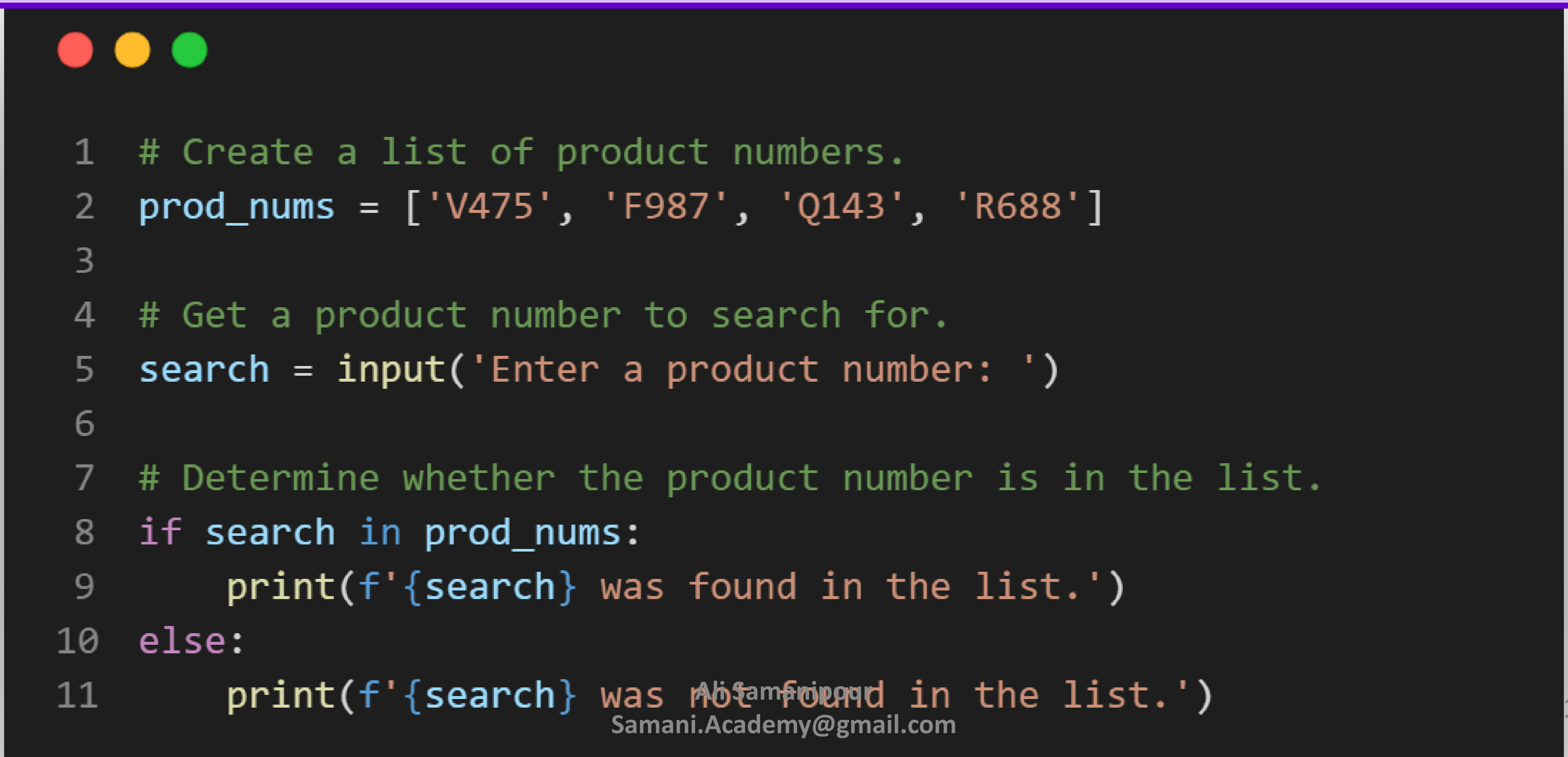
A slicing expression selects a range of elements from a sequence.

Invalid indexes do not cause slicing expressions to raise an exception

- If the end index specifies a position beyond the end of the list, Python will use the length of the list instead.
- If the start index specifies a position before the beginning of the list, Python will use 0 instead.
- If the start index is greater than the end index, the slicing expression will return an empty list.

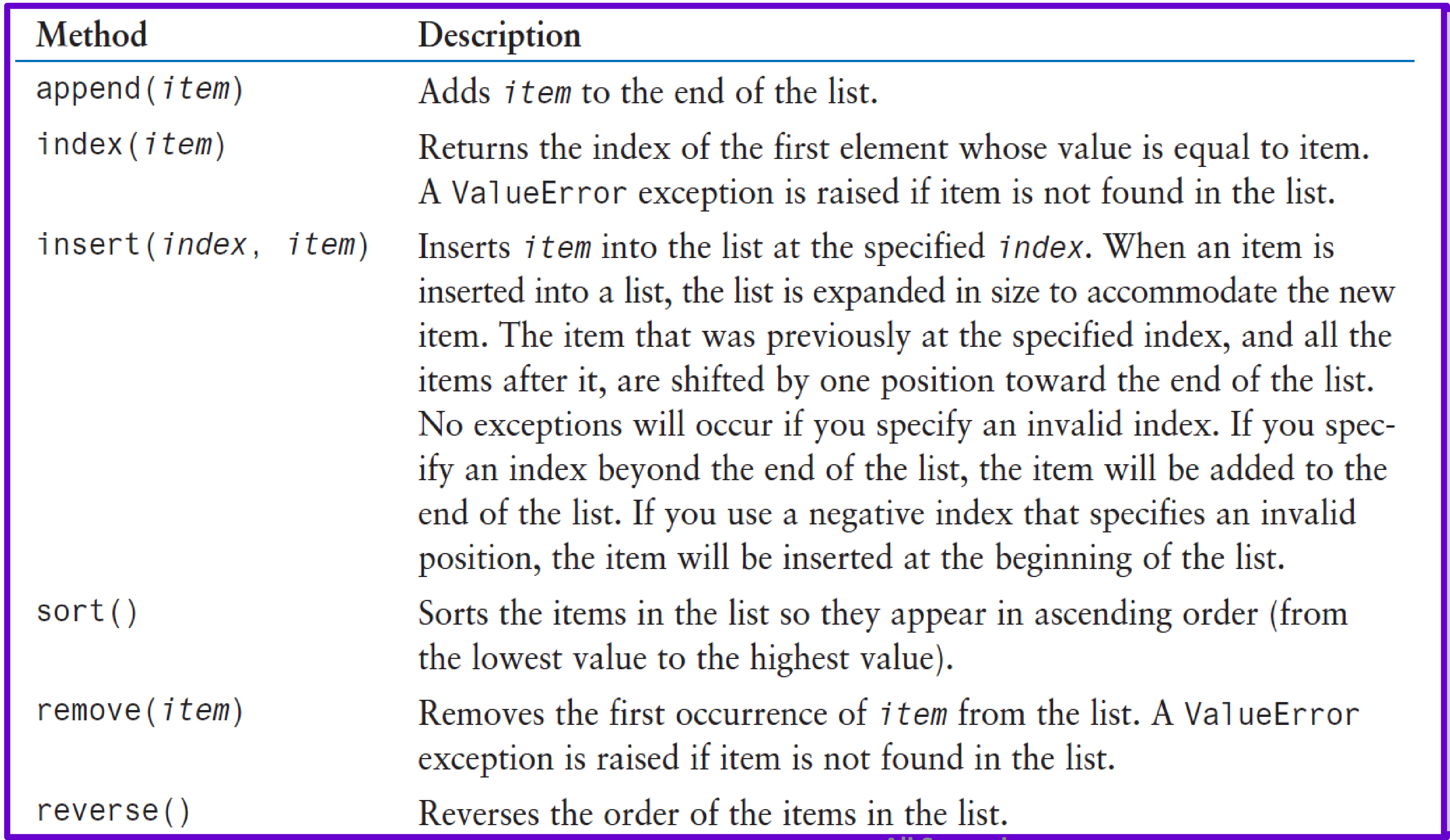
Finding Items in Lists with the in Operator

You can search for an item in a list using the in operator.



```python
# Create a list of product numbers.
prod_nums = ['V475', 'F987', 'Q143', 'R688']

# Get a product number to search for.
search = input('Enter a product number: ')

# Determine whether the product number is in the list.
if search in prod_nums:
    print(f'{search} was found in the list.')
else:
    print(f'{search} was not found in the list.')
```

List Methods and Useful Built-in Functions



Method	Description
`append(item)`	Adds *item* to the end of the list.
`index(item)`	Returns the index of the first element whose value is equal to item. A `ValueError` exception is raised if item is not found in the list.
`insert(index, item)`	Inserts *item* into the list at the specified *index*. When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at the specified index, and all the items after it, are shifted by one position toward the end of the list. No exceptions will occur if you specify an invalid index. If you specify an index beyond the end of the list, the item will be added to the end of the list. If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list.
`sort()`	Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value).
`remove(item)`	Removes the first occurrence of *item* from the list. A `ValueError` exception is raised if item is not found in the list.
`reverse()`	Reverses the order of the items in the list.

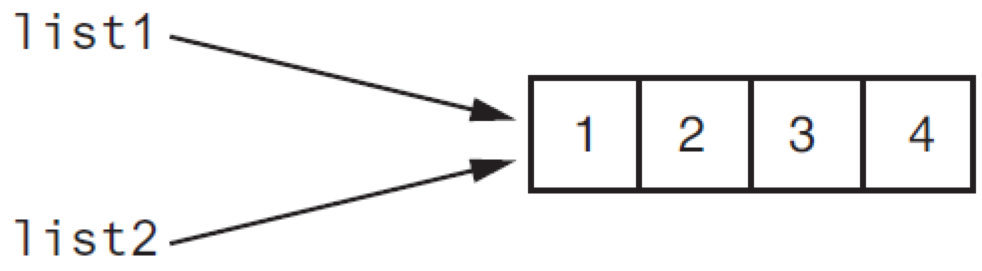
Lists have numerous methods that allow you to work with the elements that they contain. Python also provides some built-in functions that are useful for working with lists.

Reference by value vs Reference by type

To make a copy of a list, you must copy the list’s elements

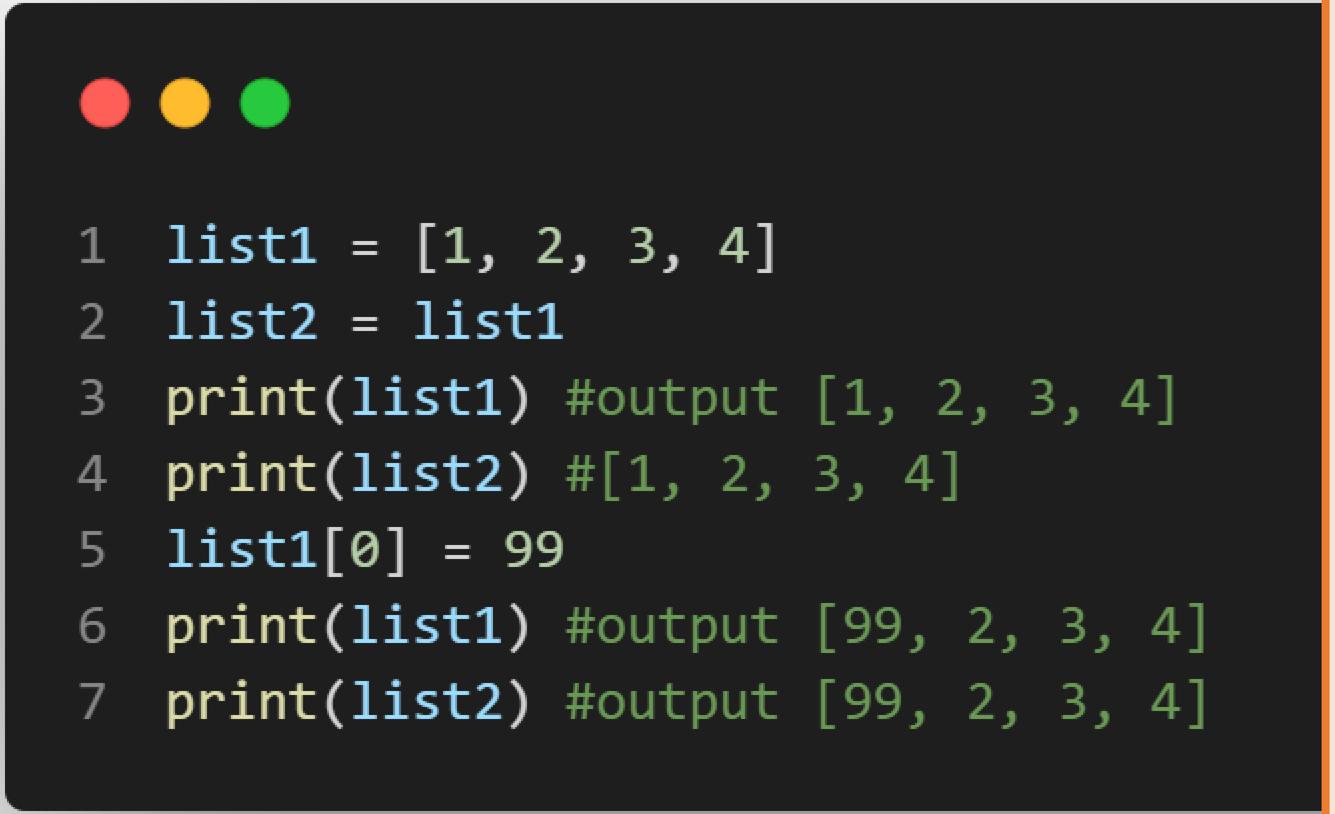
assigning one variable to another variable simply makes both variables reference the same object in memory

```
# Create a list.
list1 = [1, 2, 3, 4]
# Assign the list to the list2 variable.
list2 = list1
```



Ali Samanipour
Samani.Academy@gmail.com

Copying Lists



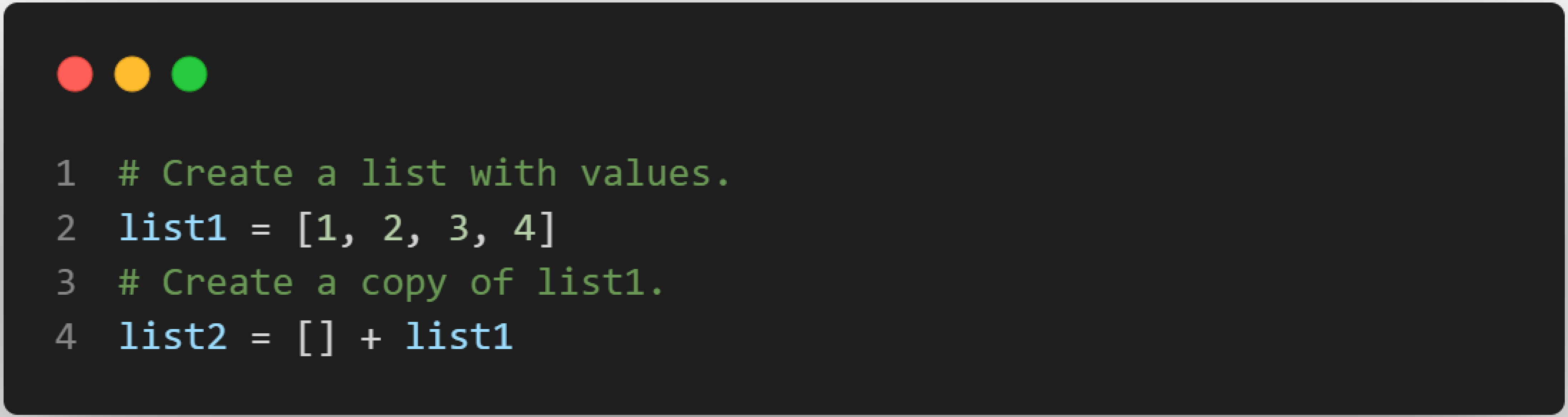
```
list1 = [1, 2, 3, 4]
list2 = list1
print(list1) #output [1, 2, 3, 4]
print(list2) #[1, 2, 3, 4]
list1[0] = 99
print(list1) #output [99, 2, 3, 4]
print(list2) #output [99, 2, 3, 4]
```

Variables of reference types store references to their data (objects), while variables of value types directly contain their data.

assigning one variable to another variable simply makes both variables reference the same object in memory

Copying Lists

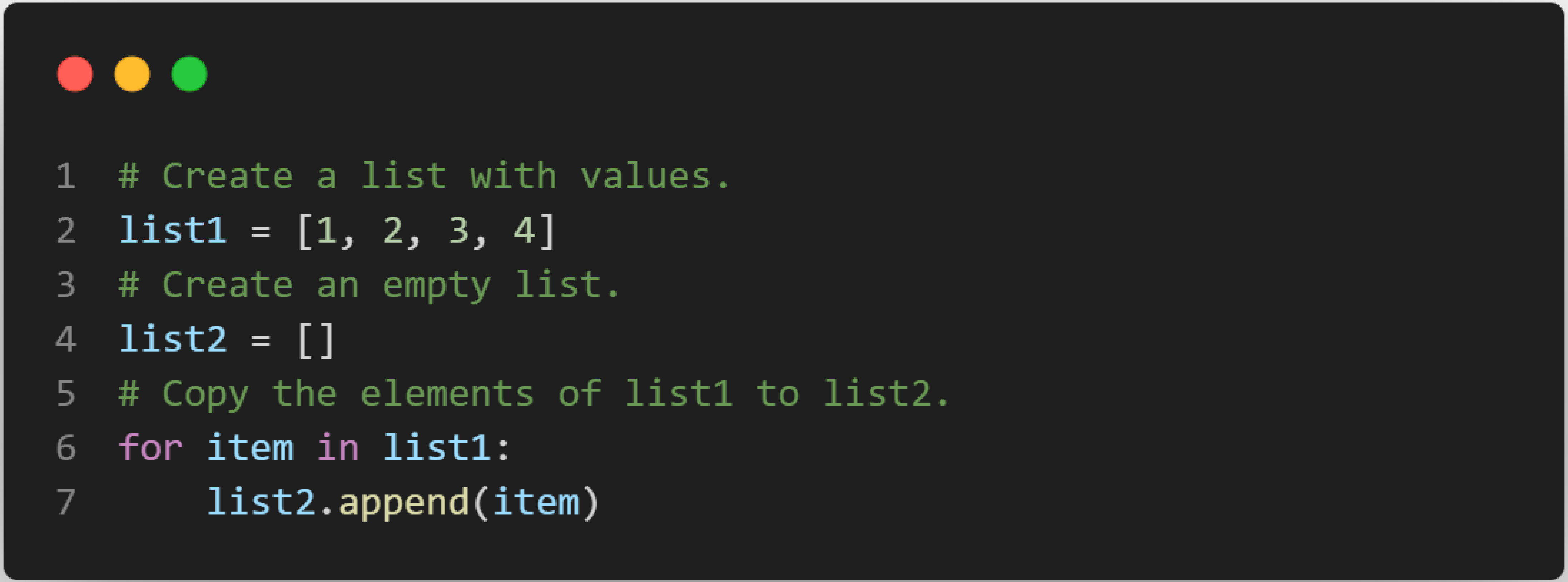
To make a copy of a list, you must copy the list's elements



```python
# Create a list with values.
list1 = [1, 2, 3, 4]
# Create a copy of list1.
list2 = [] + list1
```

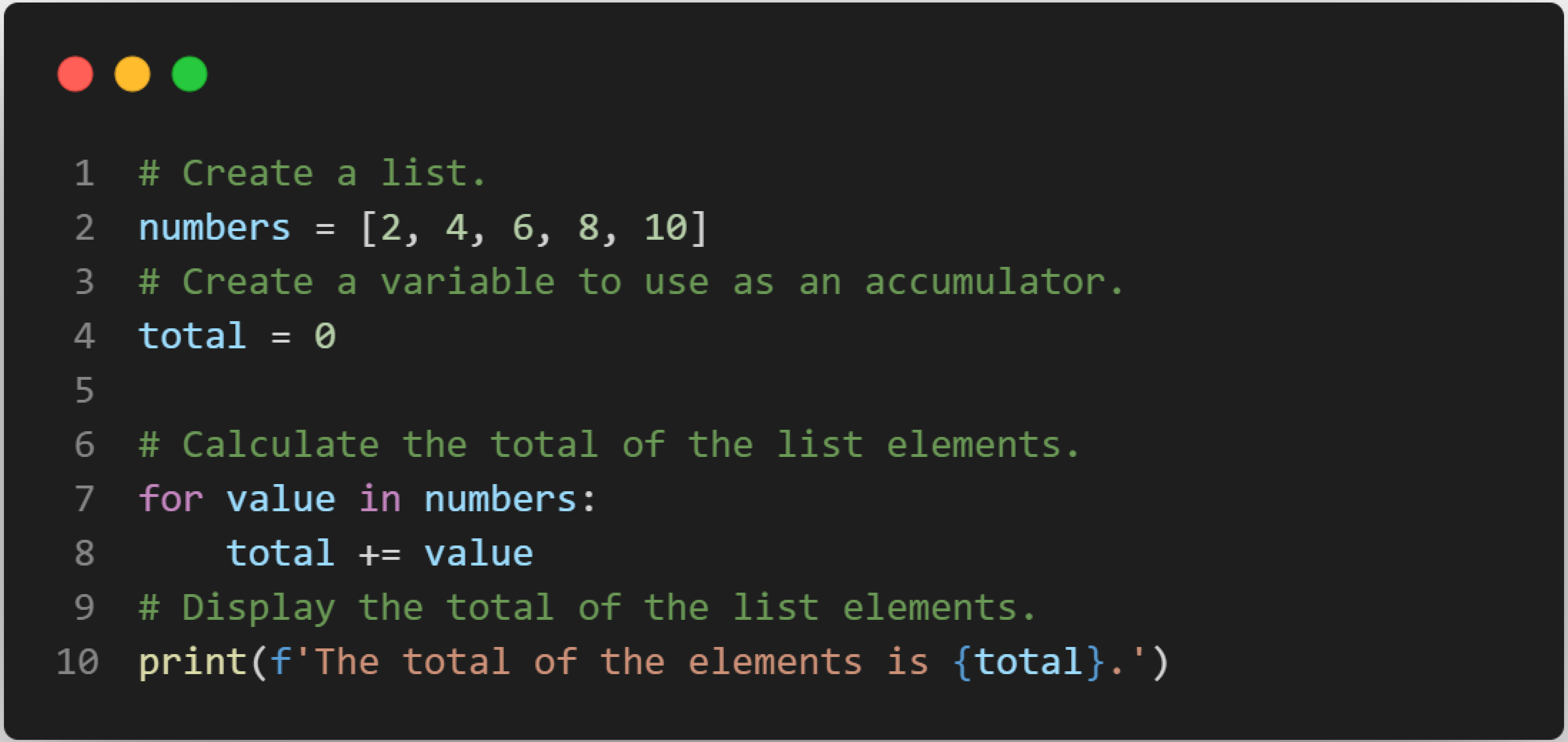
Copying Lists

To make a copy of a list, you must copy the list's elements



```python
# Create a list with values.
list1 = [1, 2, 3, 4]
# Create an empty list.
list2 = []
# Copy the elements of list1 to list2.
for item in list1:
    list2.append(item)
```

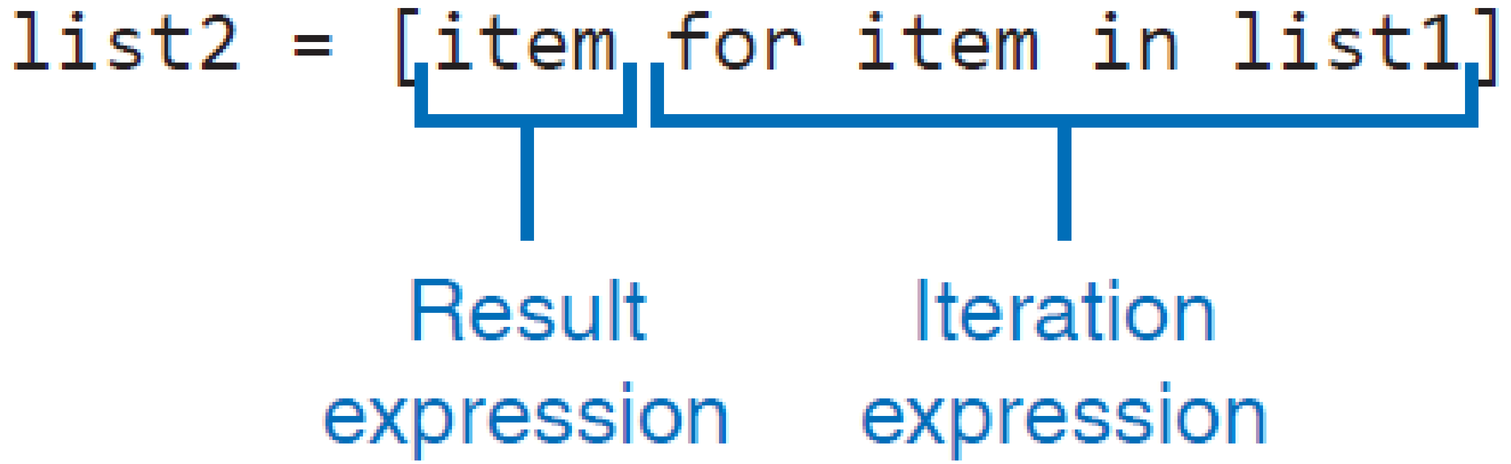
Processing Lists
(Example: Totaling the Values in a List)



```python
# Create a list.
numbers = [2, 4, 6, 8, 10]
# Create a variable to use as an accumulator.
total = 0

# Calculate the total of the list elements.
for value in numbers:
    total += value
# Display the total of the list elements.
print(f'The total of the elements is {total}.')
```

List Comprehensions

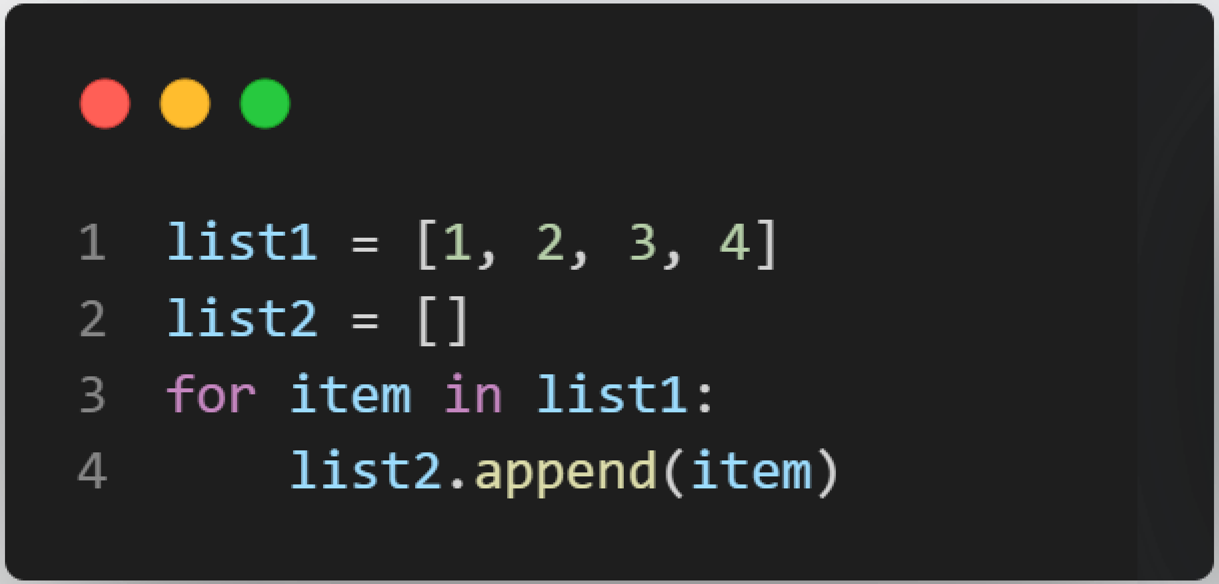
A list comprehension is a concise expression that creates a new list by iterating over the elements of an existing list.



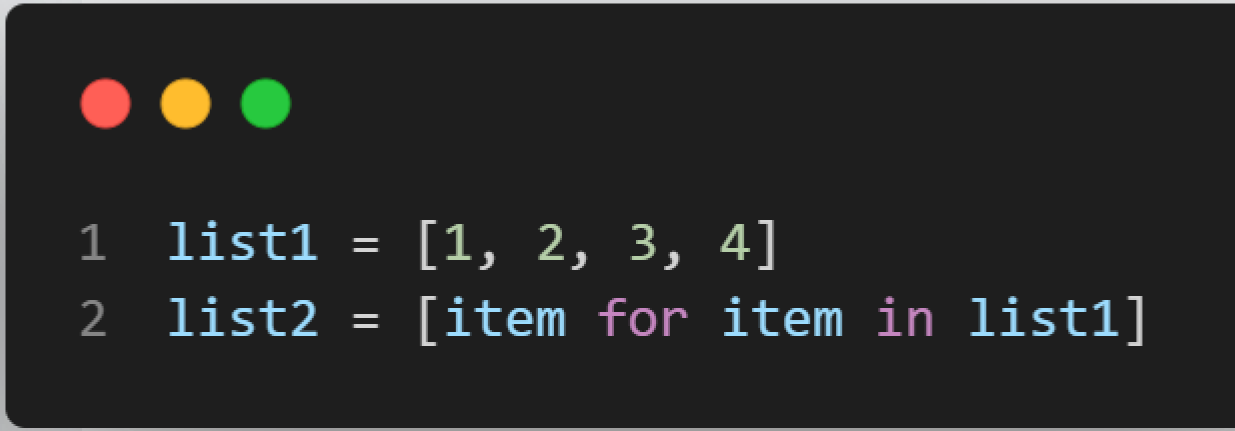
```
list2 = [item for item in list1]
```

List Comprehensions

A list comprehension is a concise expression that creates a new list by iterating over the elements of an existing list.



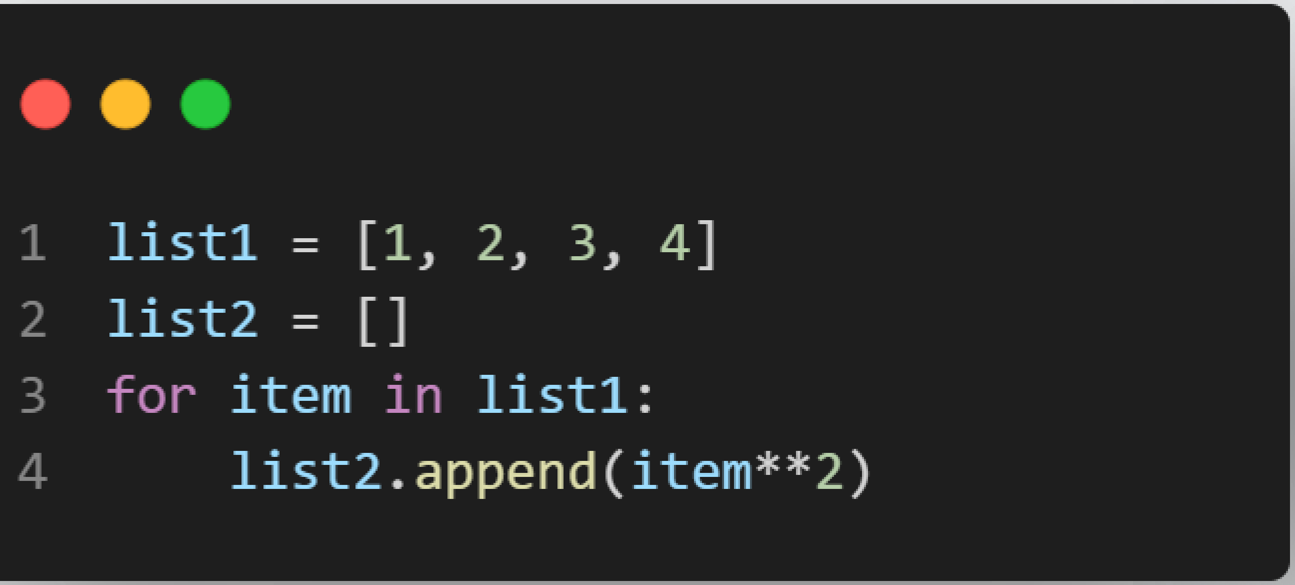
```python
list1 = [1, 2, 3, 4]
list2 = []
for item in list1:
    list2.append(item)
```



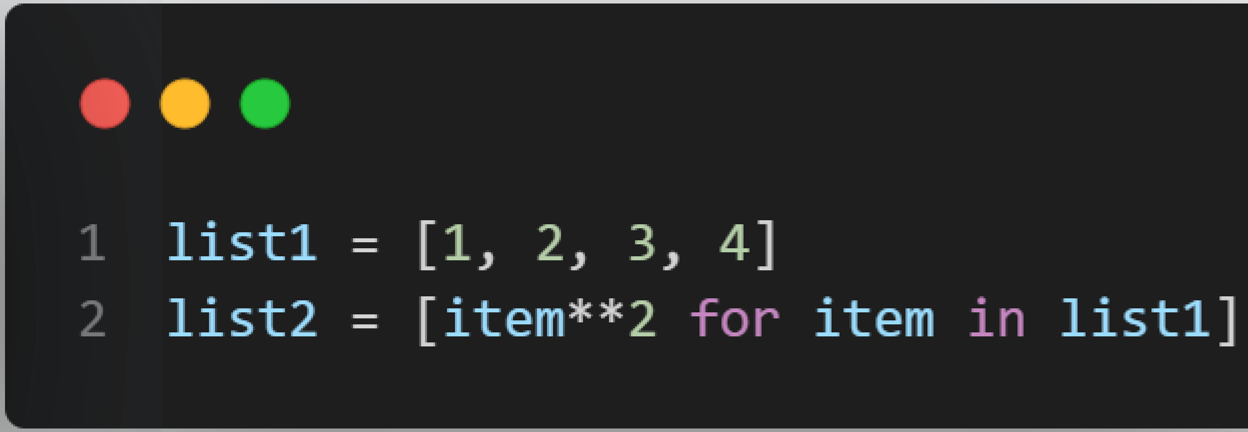
```python
list1 = [1, 2, 3, 4]
list2 = [item for item in list1]
```

List Comprehensions

A list comprehension is a concise expression that creates a new list by iterating over the elements of an existing list.



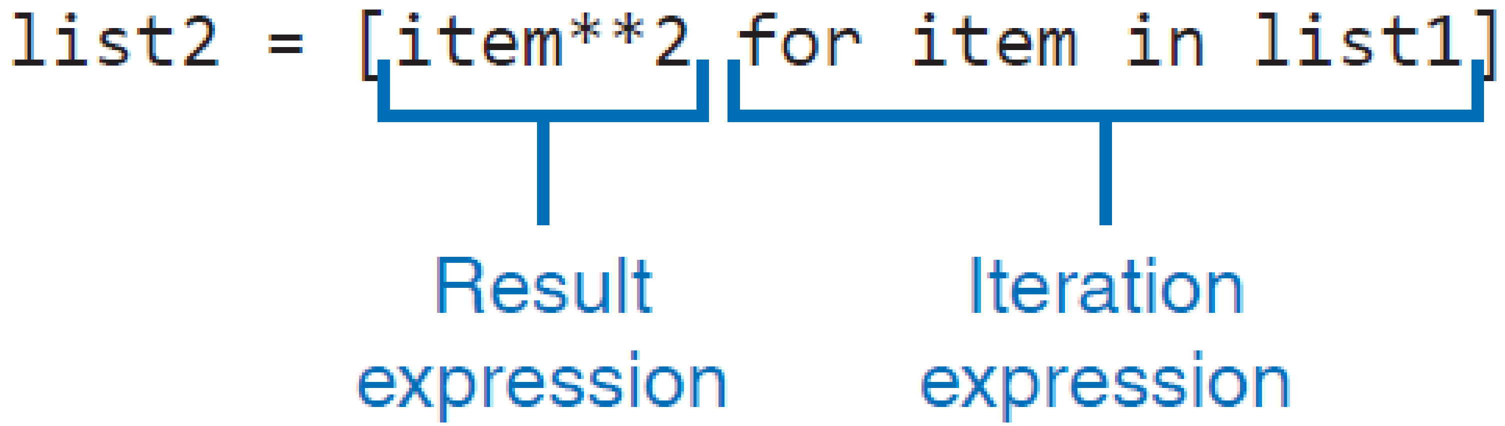
```python
list1 = [1, 2, 3, 4]
list2 = []
for item in list1:
    list2.append(item**2)
```



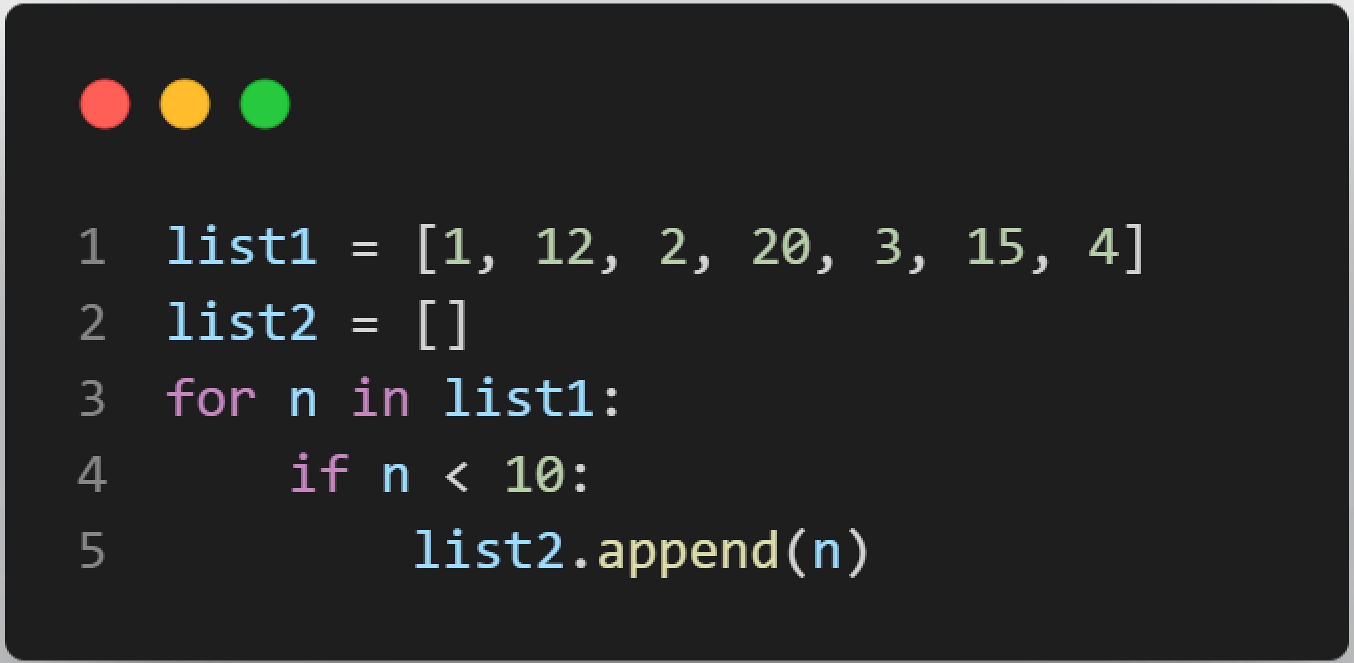
```python
list1 = [1, 2, 3, 4]
list2 = [item**2 for item in list1]
```

List Comprehensions

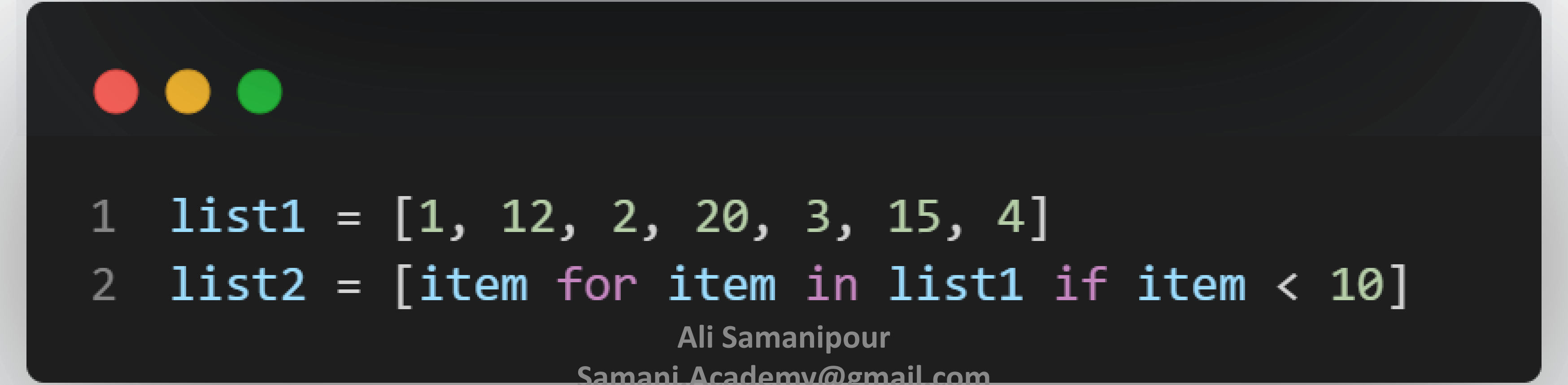
A list comprehension is a concise expression that creates a new list by iterating over the elements of an existing list.


```
list2 = [item**2 for item in list1]
```

List Comprehensions (Using if Clauses with List Comprehensions)



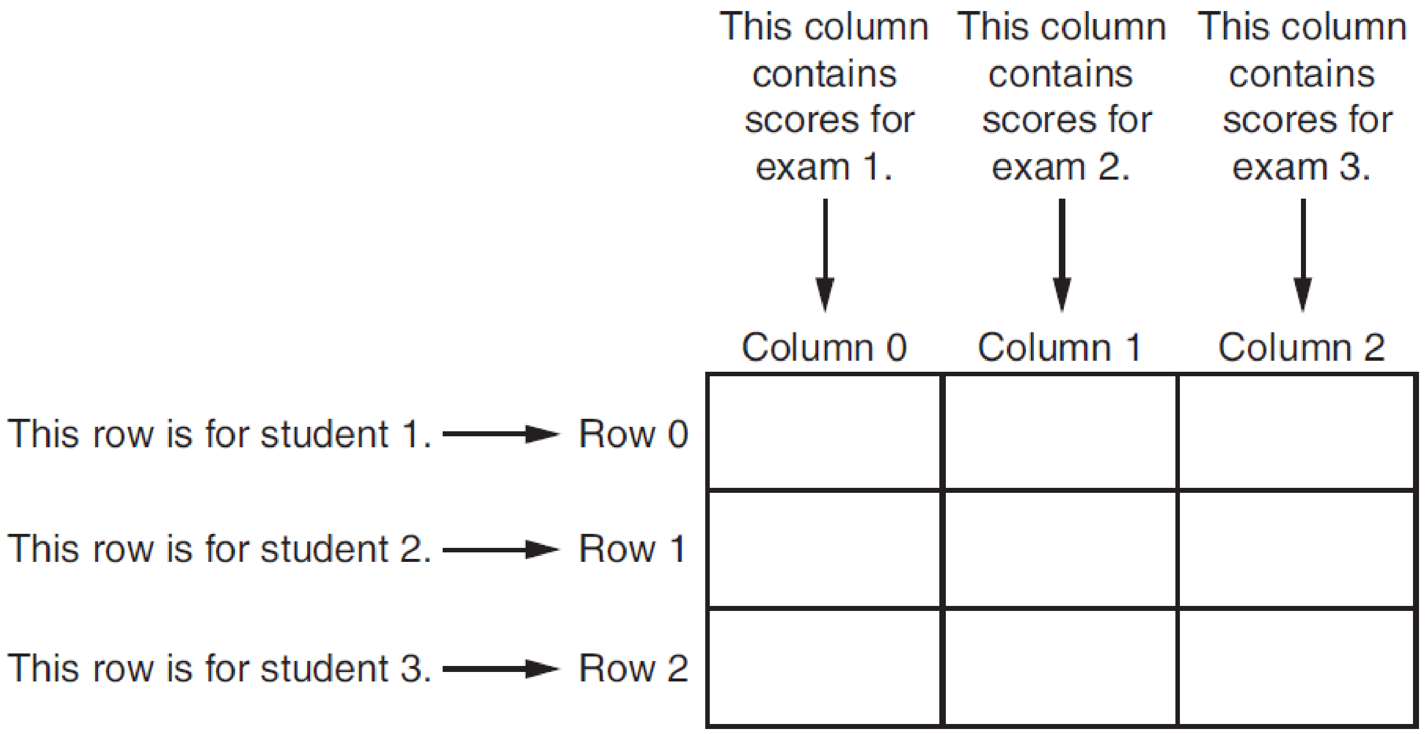
```python
list1 = [1, 12, 2, 20, 3, 15, 4]
list2 = []
for n in list1:
    if n < 10:
        list2.append(n)
```



```python
list1 = [1, 12, 2, 20, 3, 15, 4]
list2 = [item for item in list1 if item < 10]
```

Two-Dimensional Lists

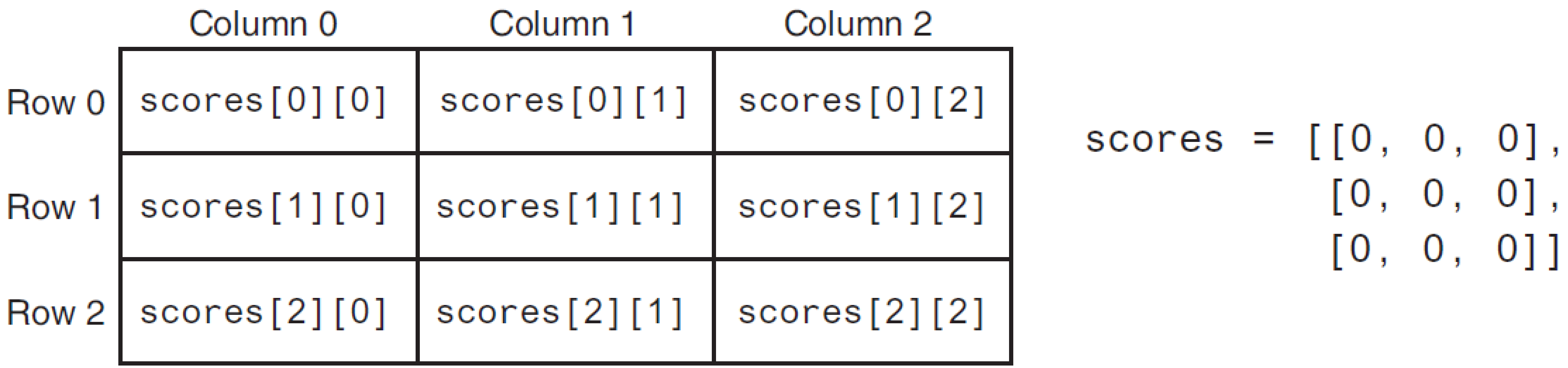
A two-dimensional list is a list that has other lists as its elements.


This column contains scores for exam 1.
This column contains scores for exam 2.
This column contains scores for exam 3.
Column 0
Column 1
Column 2
This row is for student 1.
Row 0
This row is for student 2.
Row 1
This row is for student 3.
Row 2

Ali Samanipour
Samani.Academy@gmail.com

Two-Dimensional Lists

A two-dimensional list is a list that has other lists as its elements.

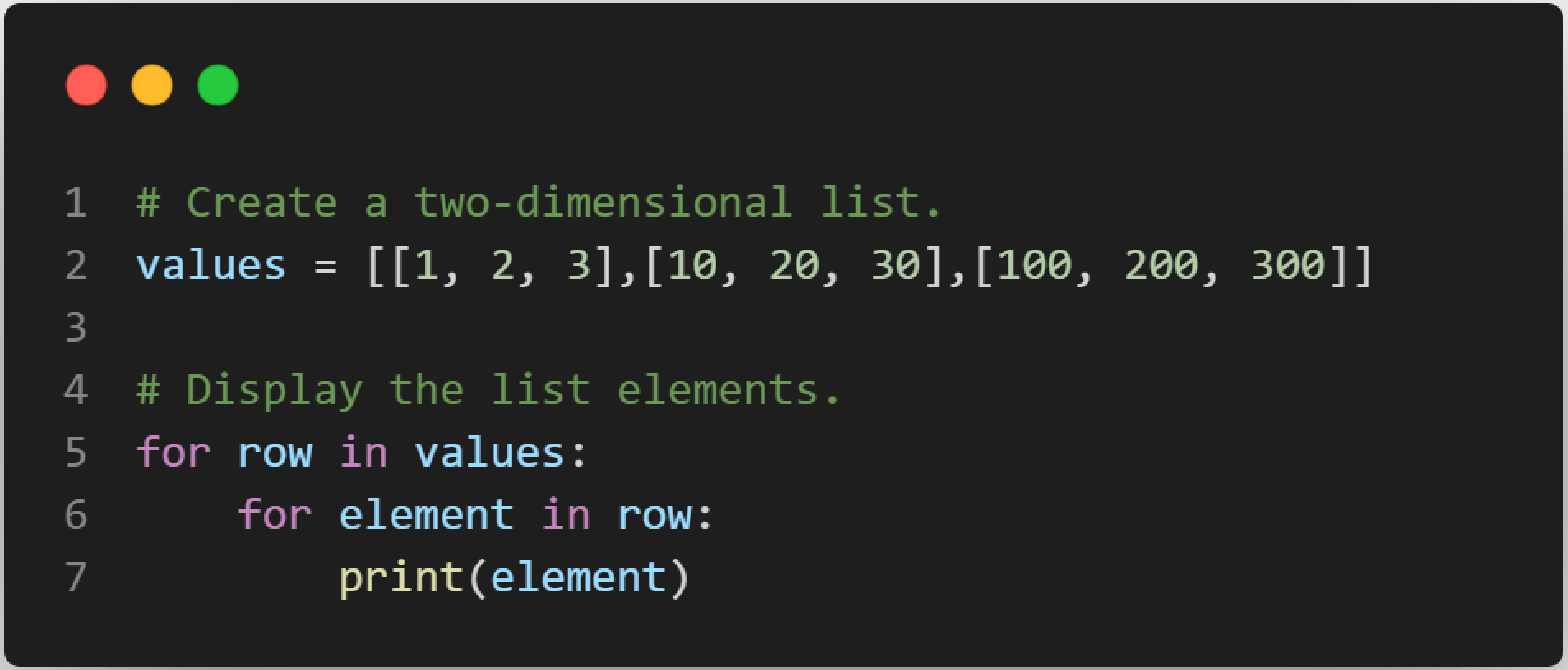


	Column 0	Column 1	Column 2
Row 0	scores[0][0]	scores[0][1]	scores[0][2]
Row 1	scores[1][0]	scores[1][1]	scores[1][2]
Row 2	scores[2][0]	scores[2][1]	scores[2][2]

```
scores = [[0, 0, 0],
          [0, 0, 0],
          [0, 0, 0]]
```

Two-Dimensional Lists

A two-dimensional list is a list that has other lists as its elements.



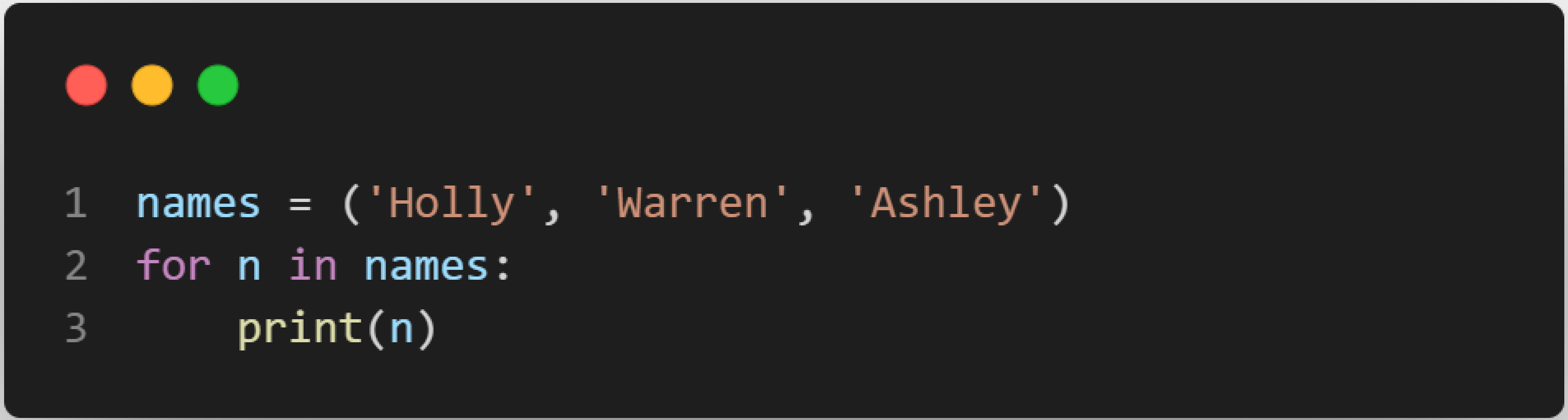
```python
# Create a two-dimensional list.
values = [[1, 2, 3],[10, 20, 30],[100, 200, 300]]

# Display the list elements.
for row in values:
    for element in row:
        print(element)
```

Module Roadmap

1. List
2. Tuples
3. Dictionaries
4. Sets

Tuples

A tuple is an immutable sequence, which means that its contents cannot be changed.



```python
names = ('Holly', 'Warren', 'Ashley')
for n in names:
    print(n)
```

Tuples

Tuples support all the same operations as lists, except those that change the contents of the list

- Subscript indexing (for retrieving element values only)
- Methods such as index
- Built-in functions such as len, min, and max
- Slicing expressions
- The in operator
- The + and * operators

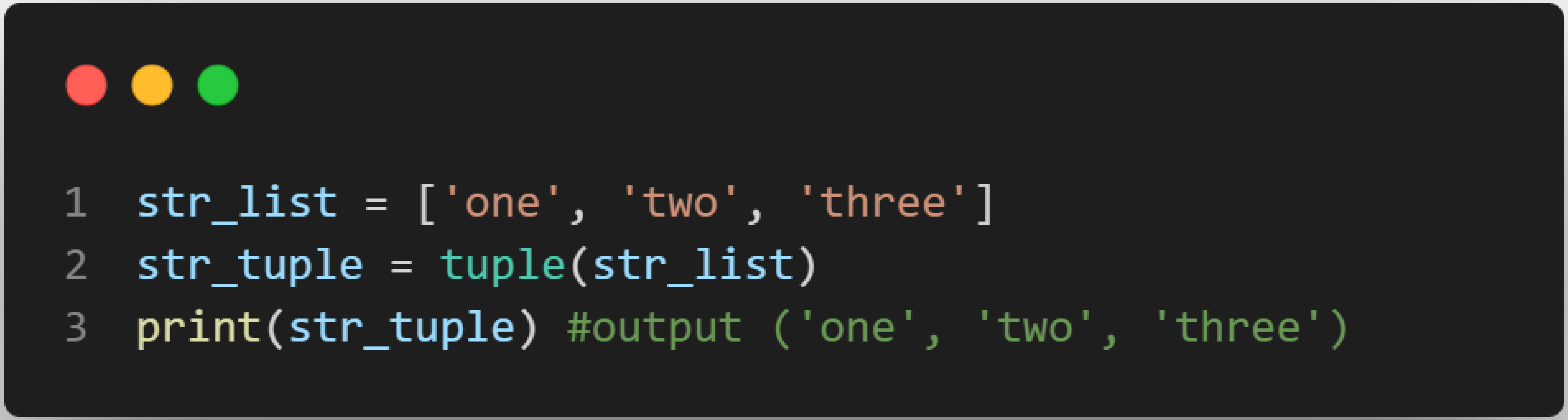
Tuples (What’s the Point?)

Processing a tuple is faster than processing a list, so tuples are good choices when you are processing lots of data

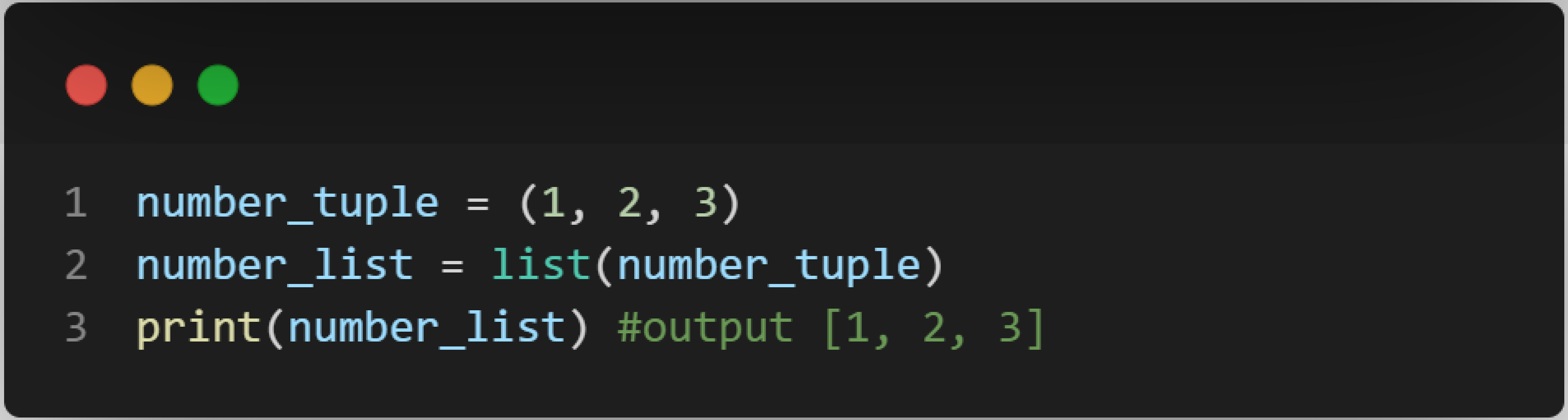
Another reason is that tuples are safe. Because you are not allowed to change the contents of a tuple

Ali Samanipour
Samani.Academy@gmail.com

Converting Between Lists and Tuples



```python
str_list = ['one', 'two', 'three']
str_tuple = tuple(str_list)
print(str_tuple) #output ('one', 'two', 'three')
```

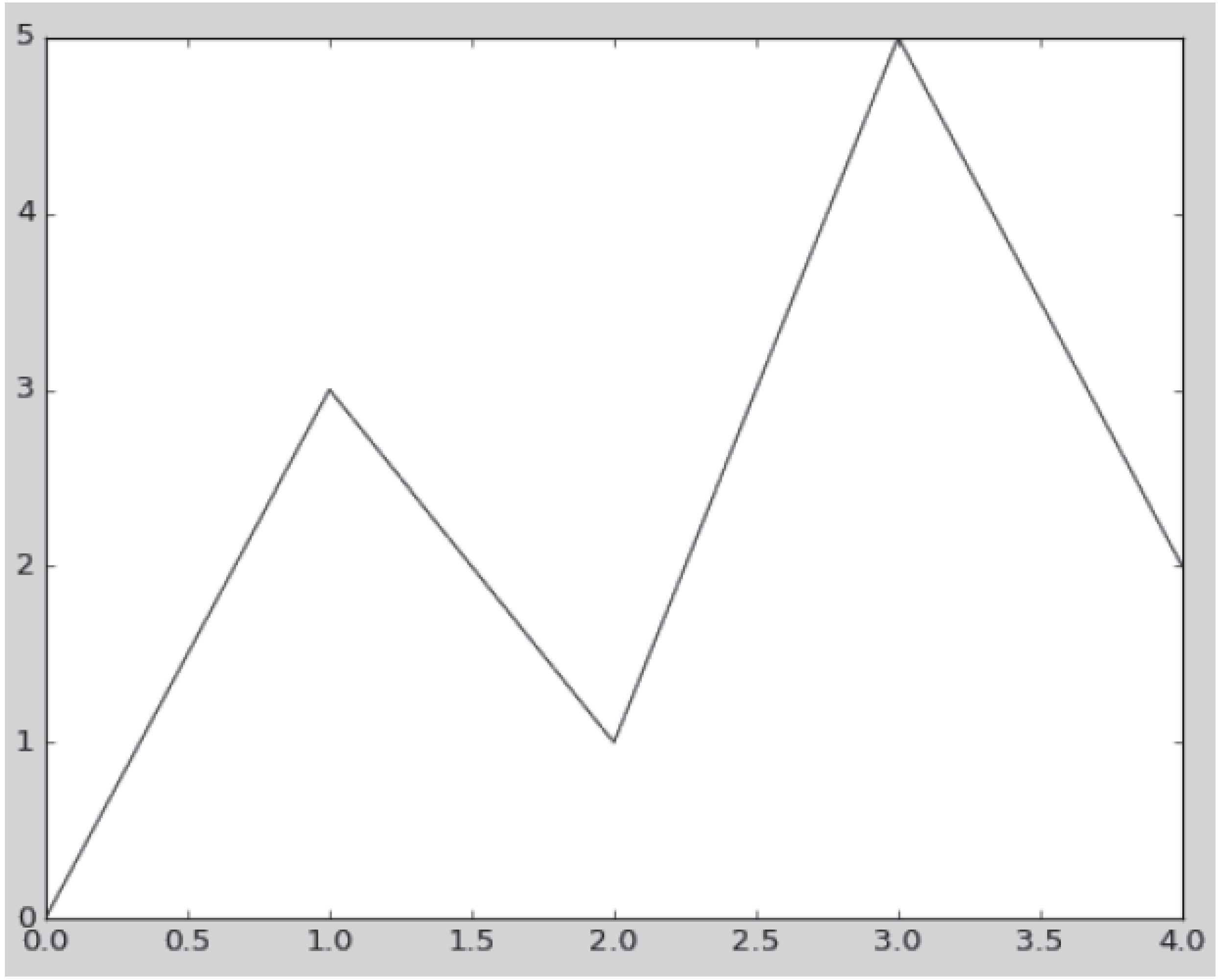


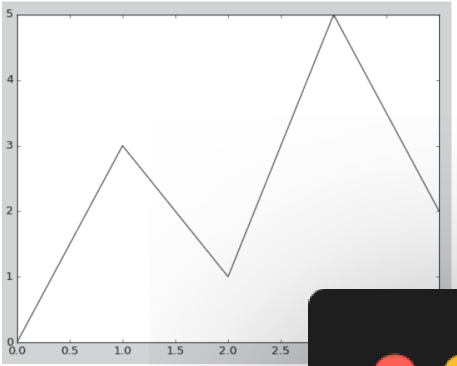
```python
number_tuple = (1, 2, 3)
number_list = list(number_tuple)
print(number_list) #output [1, 2, 3]
```

Ali Samanipour
Samani.Academy@gmail.com

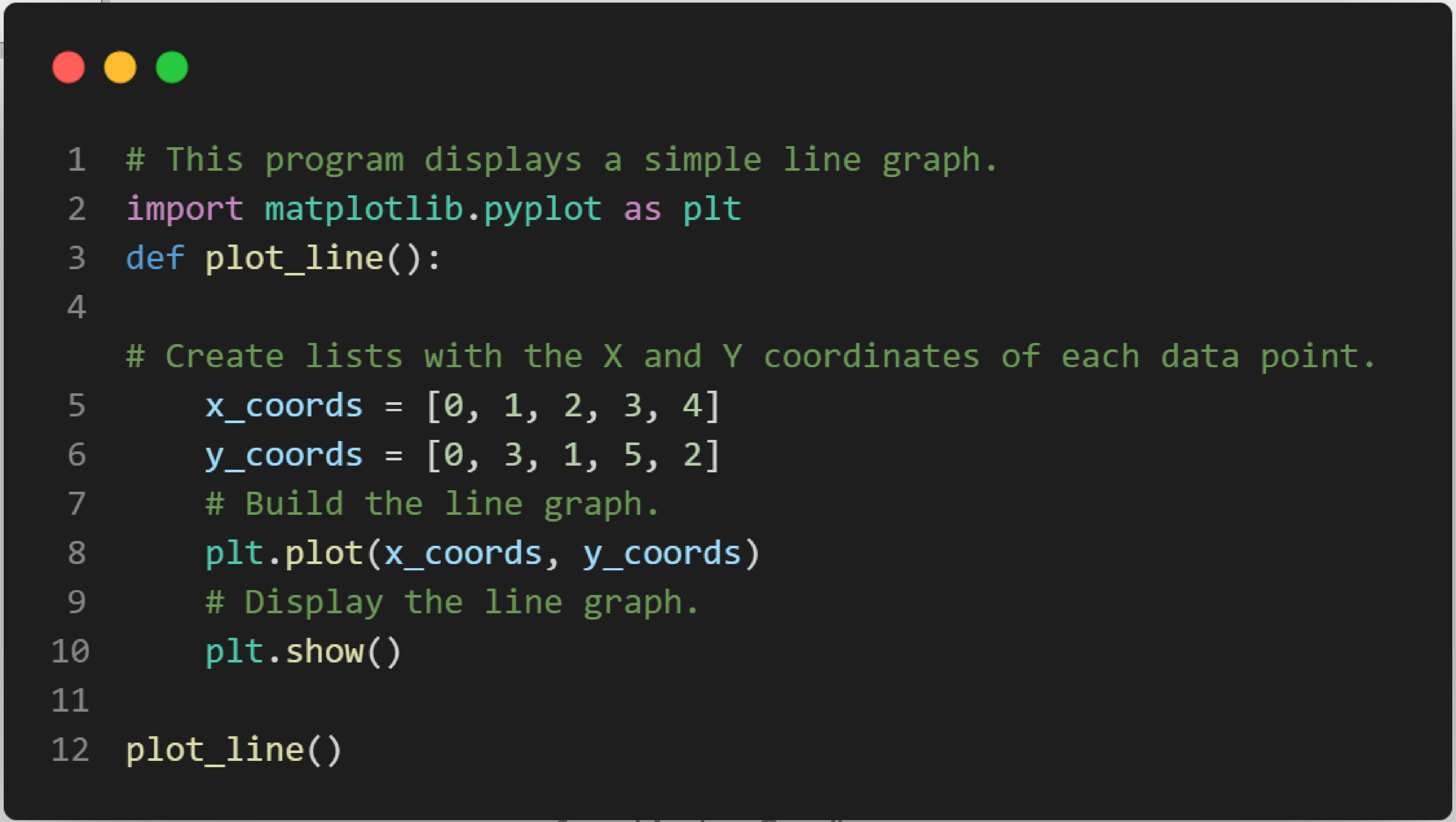
Tuples
Example(Plotting a Line Graph)

(0, 0)
(1, 3)
(2, 1)
(3, 5)
(4, 2)





Tuples
Example(Plotting a Line Graph)



```python
# This program displays a simple line graph.
import matplotlib.pyplot as plt
def plot_line():

# Create lists with the X and Y coordinates of each data point.
    x_coords = [0, 1, 2, 3, 4]
    y_coords = [0, 3, 1, 5, 2]
    # Build the line graph.
    plt.plot(x_coords, y_coords)
    # Display the line graph.
    plt.show()

plot_line()
```

Module Roadmap

1. List
2. Tuples
3. Dictionaries
4. Sets

Dictionaries

A dictionary is an object that stores a collection of data. Each element in a dictionary has two parts: a key and a value. You use a key to locate a specific value

<u>Key-value pairs</u> are often referred to as mappings because each key is mapped to a value.

Creating a Dictionary

You can create a dictionary by enclosing the elements inside a set of curly braces ({}).

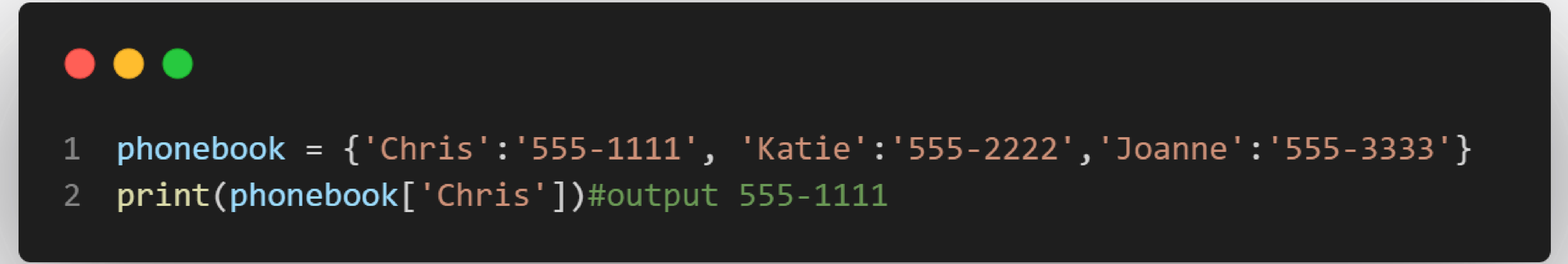
An element consists of a key, followed by a colon, followed by a value.

```
phonebook = {'Chris':'555–1111', 'Katie':'555–2222', 'Joanne':'555–3333'}
```

Retrieve an element by key in Dictionary

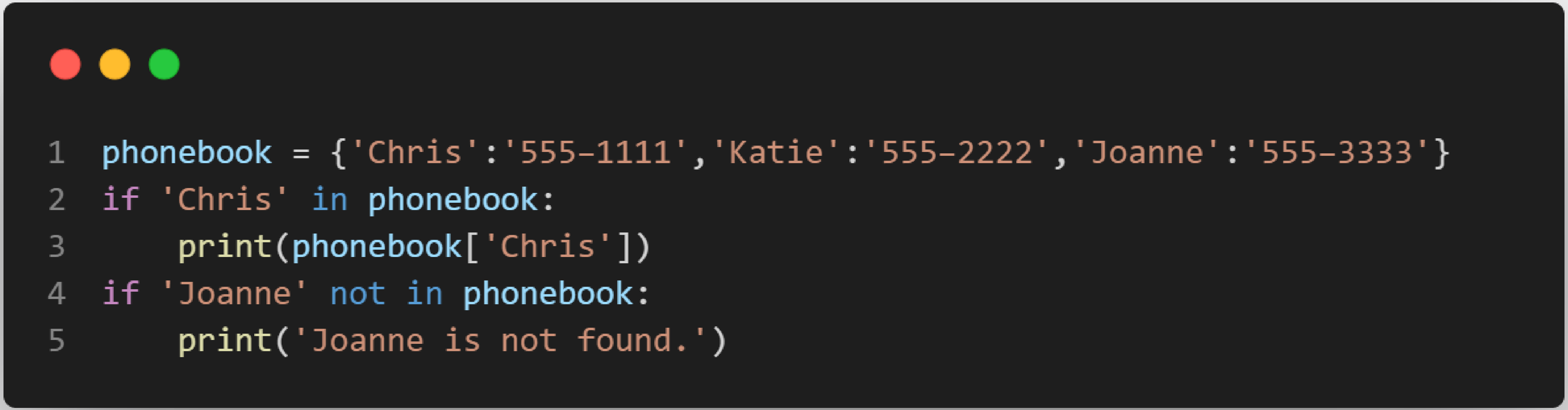
To retrieve a value from a dictionary, you simply write an expression in the following general format:

dictionary_name[key]



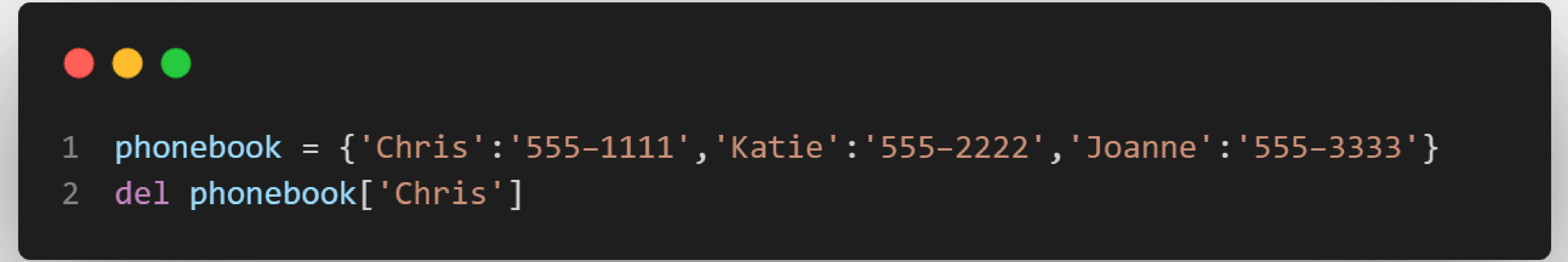
```python
phonebook = {'Chris':'555-1111', 'Katie':'555-2222','Joanne':'555-3333'}
print(phonebook['Chris'])#output 555-1111
```

Using the <u>in</u> and <u>not in</u> Operators to Test for a Value in a Dictionary



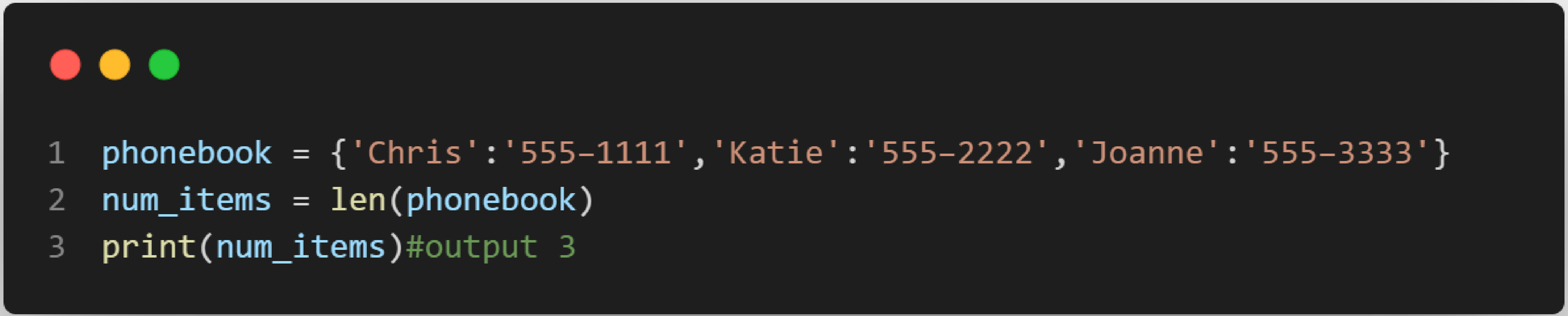
```python
phonebook = {'Chris':'555-1111','Katie':'555-2222','Joanne':'555-3333'}
if 'Chris' in phonebook:
    print(phonebook['Chris'])
if 'Joanne' not in phonebook:
    print('Joanne is not found.')
```

Deleting Elements



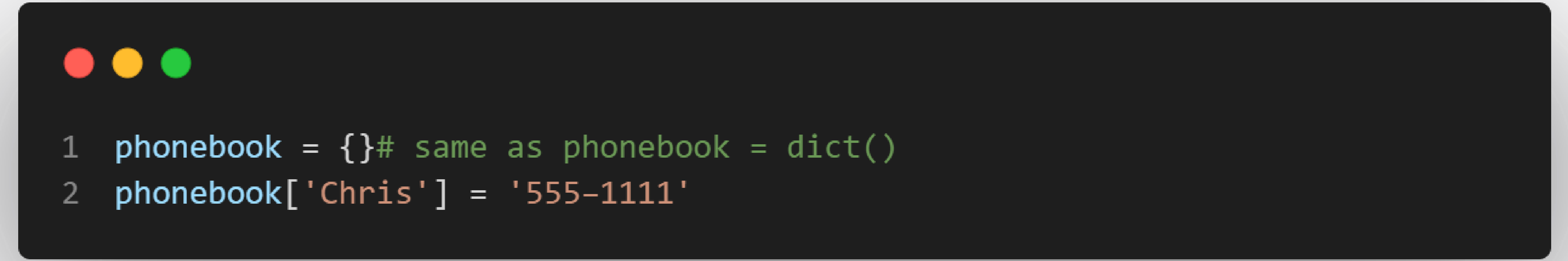
```python
phonebook = {'Chris':'555-1111','Katie':'555-2222','Joanne':'555-3333'}
del phonebook['Chris']
```

Getting the Number of Elements in a Dictionary



```python
phonebook = {'Chris':'555-1111','Katie':'555-2222','Joanne':'555-3333'}
num_items = len(phonebook)
print(num_items)#output 3
```

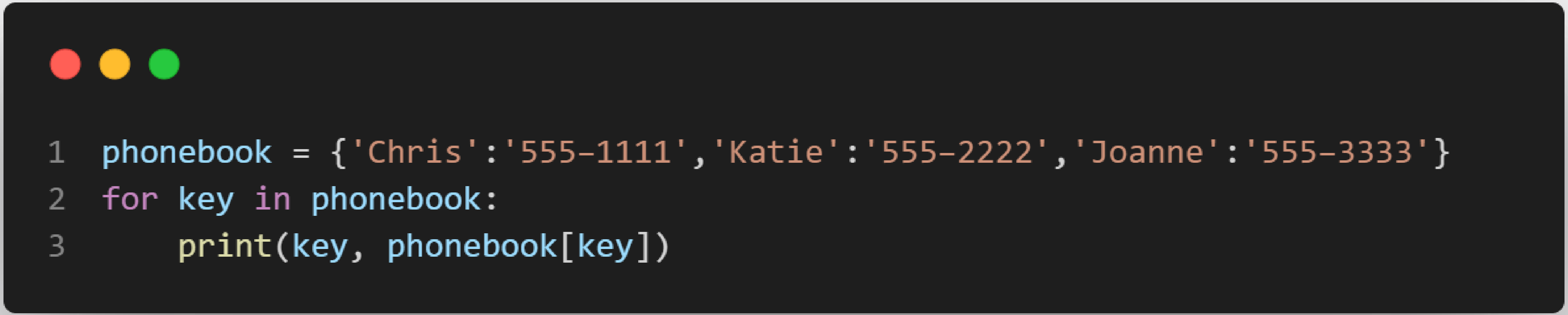
Creating an Empty Dictionary



```python
phonebook = {}# same as phonebook = dict()
phonebook['Chris'] = '555-1111'
```

Ali Samanipour
Samani.Academy@gmail.com

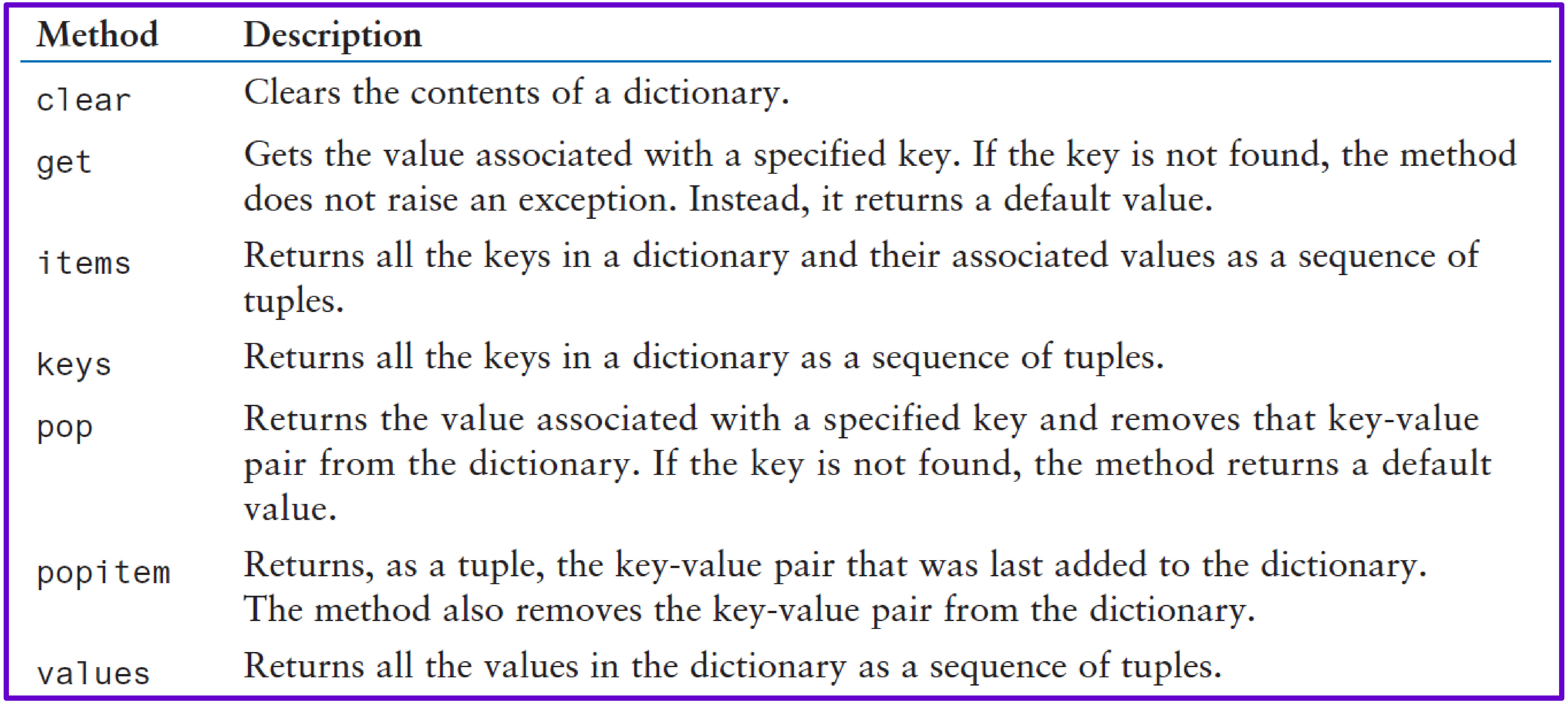
Using the for Loop to Iterate over a Dictionary



```python
phonebook = {'Chris':'555-1111','Katie':'555-2222','Joanne':'555-3333'}
for key in phonebook:
    print(key, phonebook[key])
```

Ali Samanipour
Samani.Academy@gmail.com

Some Dictionary Methods



Method	Description
`clear`	Clears the contents of a dictionary.
`get`	Gets the value associated with a specified key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.
`items`	Returns all the keys in a dictionary and their associated values as a sequence of tuples.
`keys`	Returns all the keys in a dictionary as a sequence of tuples.
`pop`	Returns the value associated with a specified key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.
`popitem`	Returns, as a tuple, the key-value pair that was last added to the dictionary. The method also removes the key-value pair from the dictionary.
`values`	Returns all the values in the dictionary as a sequence of tuples.

Module Roadmap

1. List
2. Tuples
3. Dictionaries
4. Sets

Sets

A set contains a collection of unique values and works like a mathematical set.

- **<u>All the elements in a set must be unique</u>**
- **<u>Sets are unordered</u>**
- **<u>The elements that are stored in a set can be of different data types.</u>**

Creating a Set

To create a set, you have to call the built-in set function

```
myset = set()
```

```
myset = set(['a', 'b', 'c'])
```

```
myset = set('abc’)
```

After this statement executes, the myset variable will reference a set containing the elements 'a', 'b', and 'c'.

Creating a Set

myset = set('aaabc’)

After this statement executes, the myset variable will reference a set containing the elements 'a', 'b', and 'c'.

<u>What if you want to create a set in which each element is a string containing more than one character?</u>

# This is an ERROR!
myset = set('one', 'two', 'three’)

<u>you can pass no more than one argument to the set function</u>

Creating a Set

What if you want to create a set in which each element is a string containing more than one character?

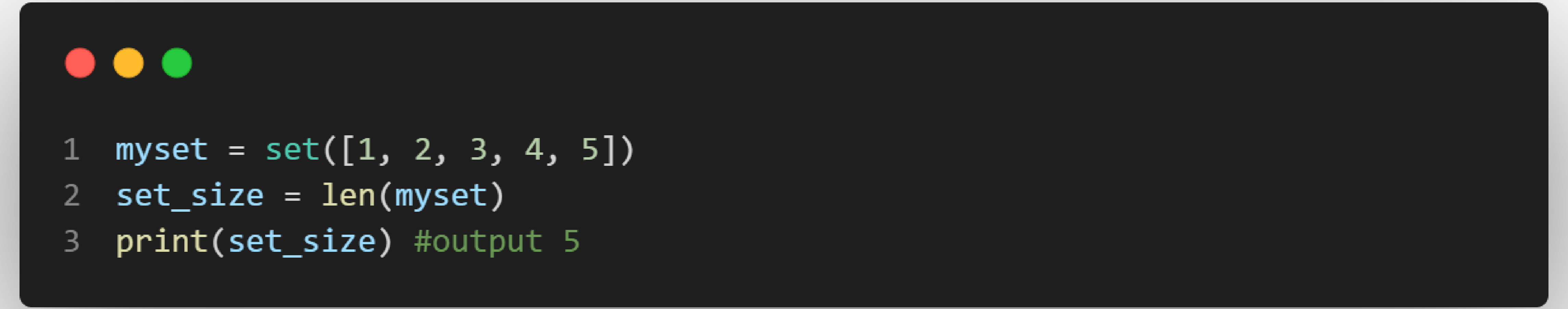
```
# This is an ERROR!
myset = set('one', 'two', 'three’)
```

you can pass no more than one argument to the set function

```
# OK, this works.
myset = set(['one', 'two', 'three'])
```

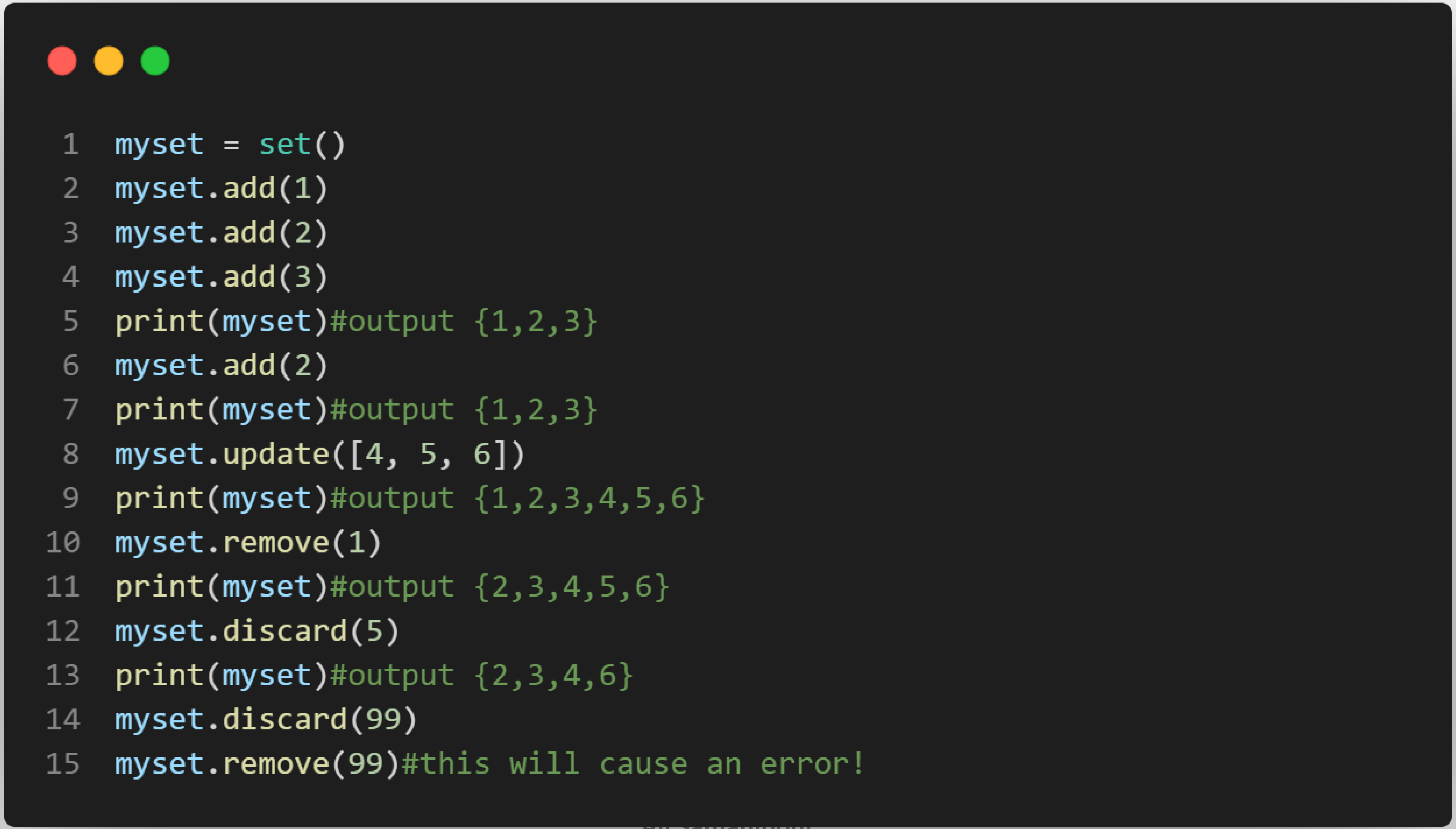
Ali Samanipour
Samani.Academy@gmail.com

Getting the Number of Elements in a Set



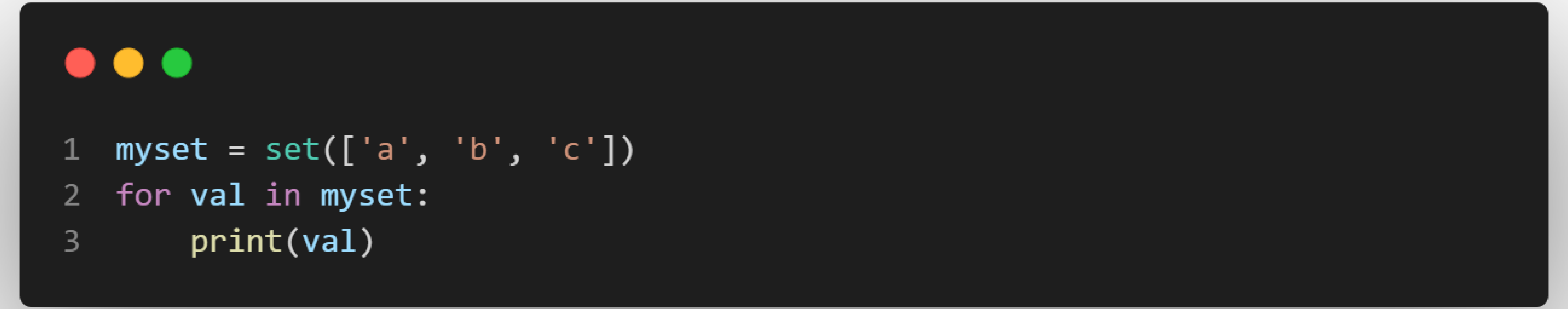
```python
myset = set([1, 2, 3, 4, 5])
set_size = len(myset)
print(set_size) #output 5
```

Adding and Removing Elements



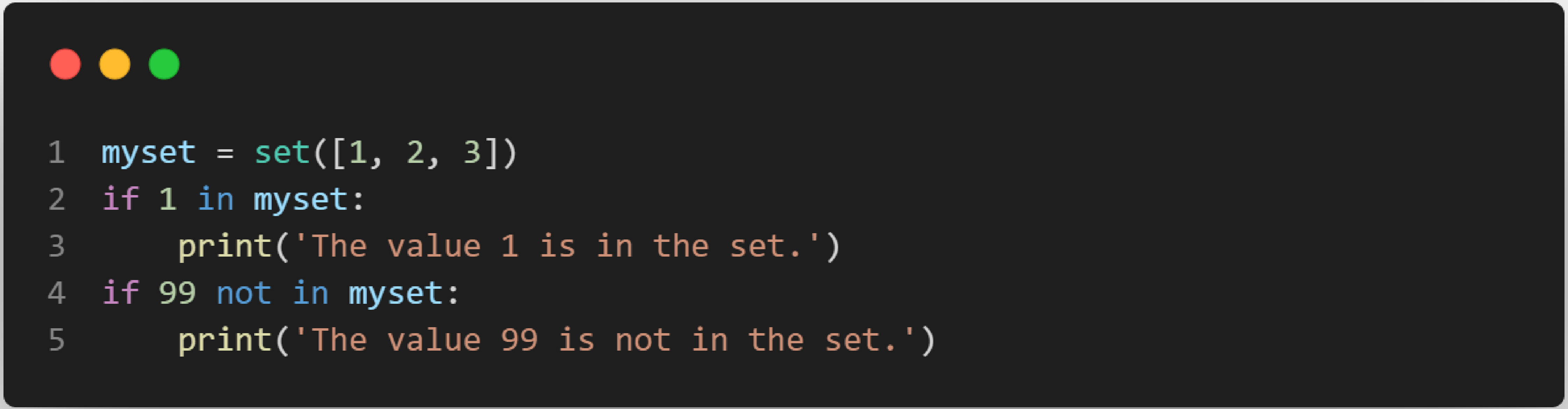
```python
myset = set()
myset.add(1)
myset.add(2)
myset.add(3)
print(myset)#output {1,2,3}
myset.add(2)
print(myset)#output {1,2,3}
myset.update([4, 5, 6])
print(myset)#output {1,2,3,4,5,6}
myset.remove(1)
print(myset)#output {2,3,4,5,6}
myset.discard(5)
print(myset)#output {2,3,4,6}
myset.discard(99)
myset.remove(99)#this will cause an error!
```

Using the for Loop to Iterate over a Set



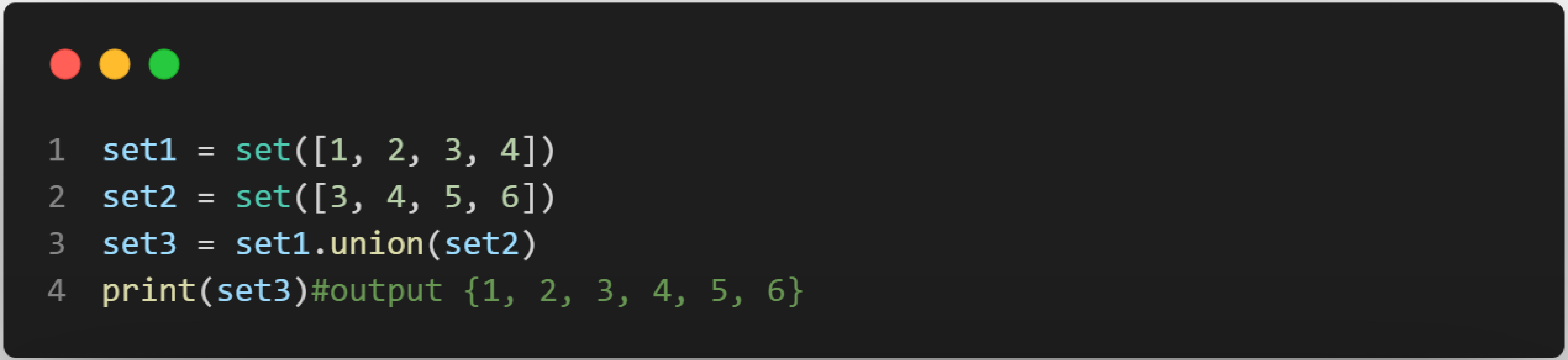
```python
myset = set(['a', 'b', 'c'])
for val in myset:
    print(val)
```

Using the <u>in</u> and <u>not in</u> Operators to Test for a Value in a Set

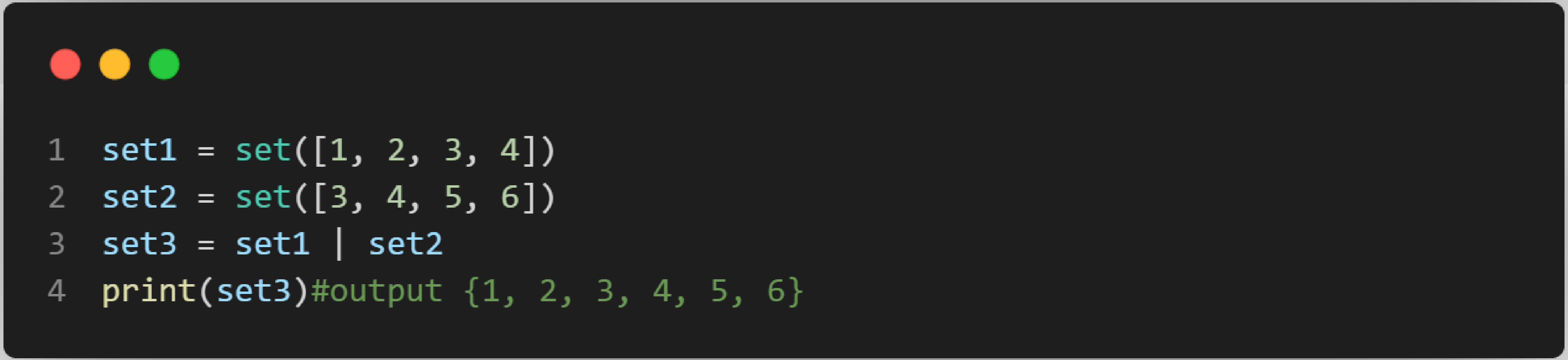


```python
myset = set([1, 2, 3])
if 1 in myset:
    print('The value 1 is in the set.')
if 99 not in myset:
    print('The value 99 is not in the set.')
```

Finding the Union of Sets

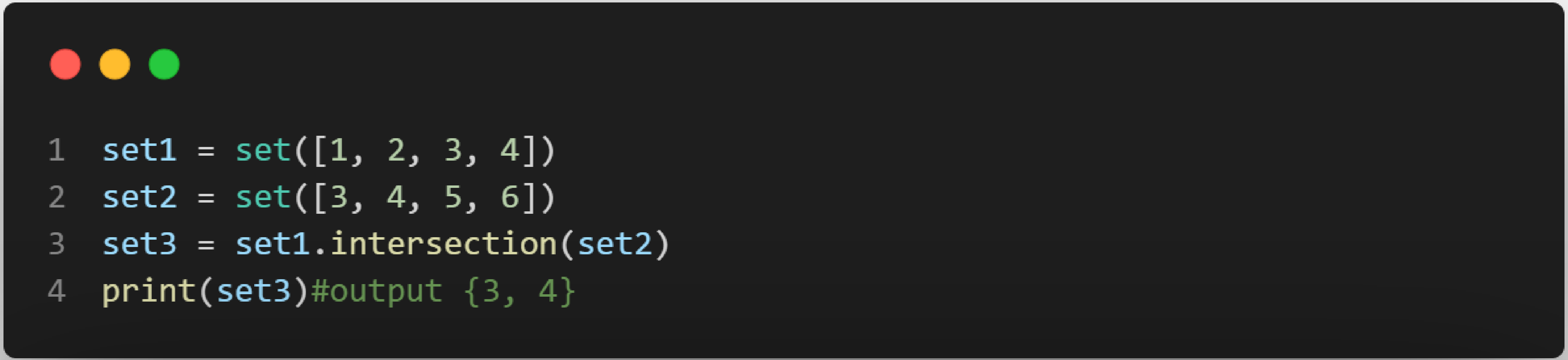


```python
set1 = set([1, 2, 3, 4])
set2 = set([3, 4, 5, 6])
set3 = set1.union(set2)
print(set3)#output {1, 2, 3, 4, 5, 6}
```

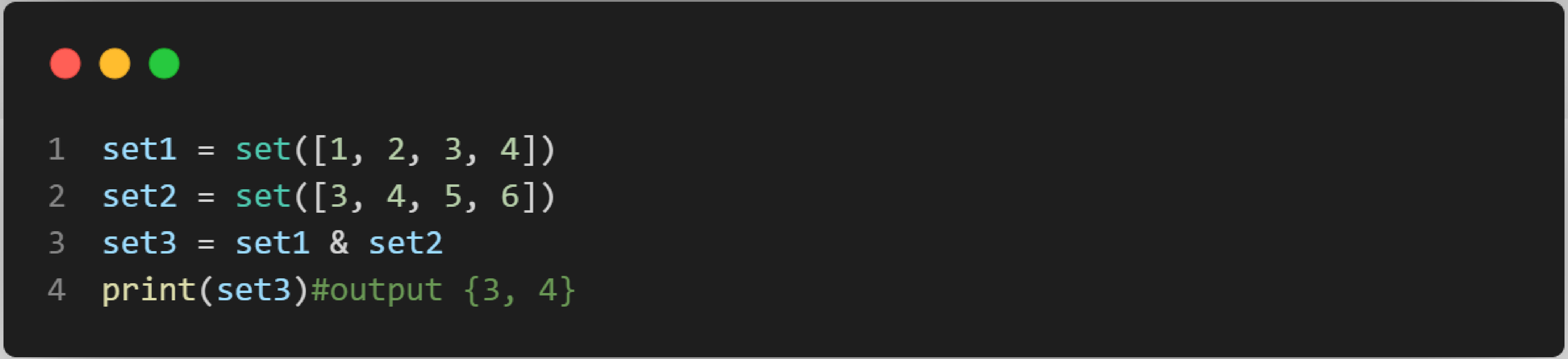


```python
set1 = set([1, 2, 3, 4])
set2 = set([3, 4, 5, 6])
set3 = set1 | set2
print(set3)#output {1, 2, 3, 4, 5, 6}
```

Finding the Intersection of Sets

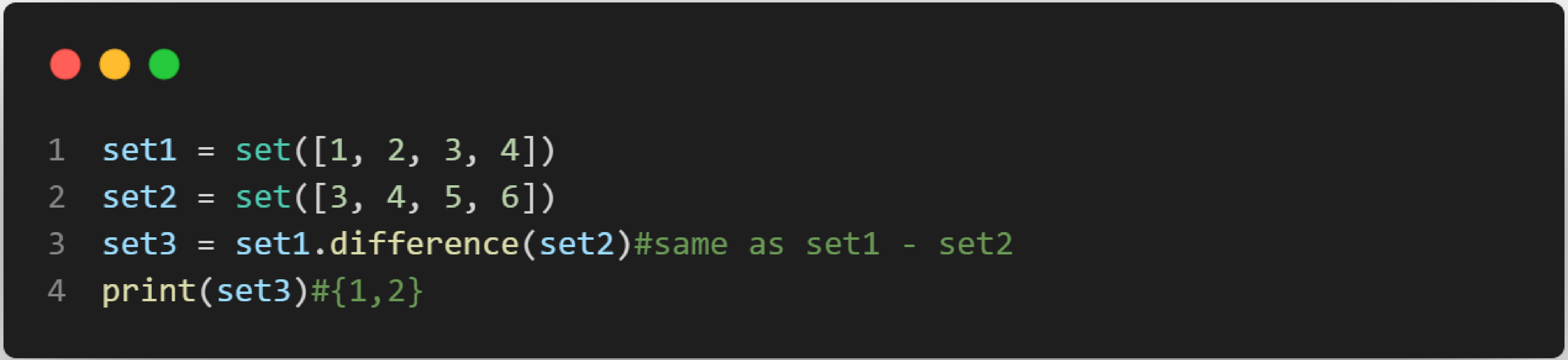


```python
set1 = set([1, 2, 3, 4])
set2 = set([3, 4, 5, 6])
set3 = set1.intersection(set2)
print(set3)#output {3, 4}
```



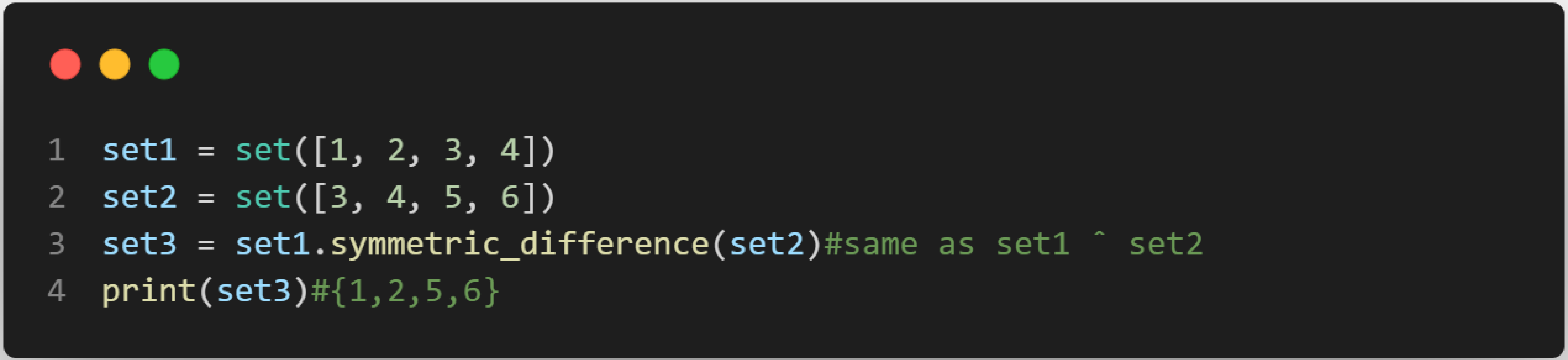
```python
set1 = set([1, 2, 3, 4])
set2 = set([3, 4, 5, 6])
set3 = set1 & set2
print(set3)#output {3, 4}
```

Finding the Difference of Sets



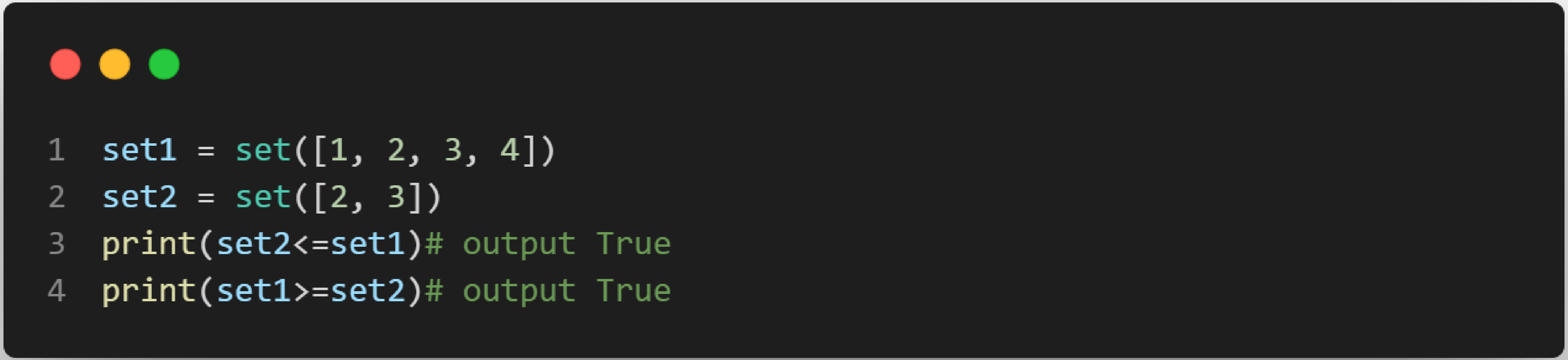
```python
set1 = set([1, 2, 3, 4])
set2 = set([3, 4, 5, 6])
set3 = set1.difference(set2)#same as set1 - set2
print(set3)#{1,2}
```

Symmetric Difference of Sets

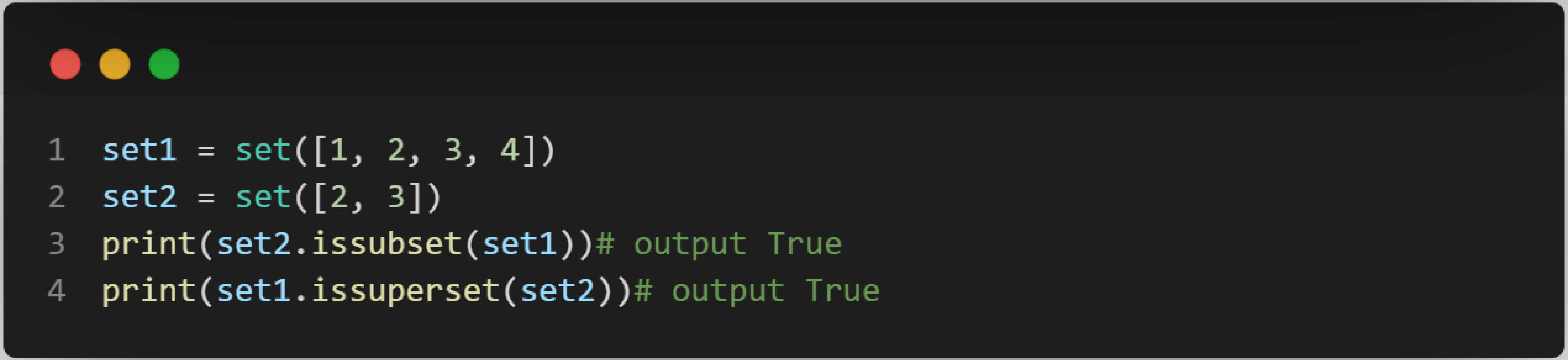


```python
set1 = set([1, 2, 3, 4])
set2 = set([3, 4, 5, 6])
set3 = set1.symmetric_difference(set2)#same as set1 ^ set2
print(set3)#{1,2,5,6}
```

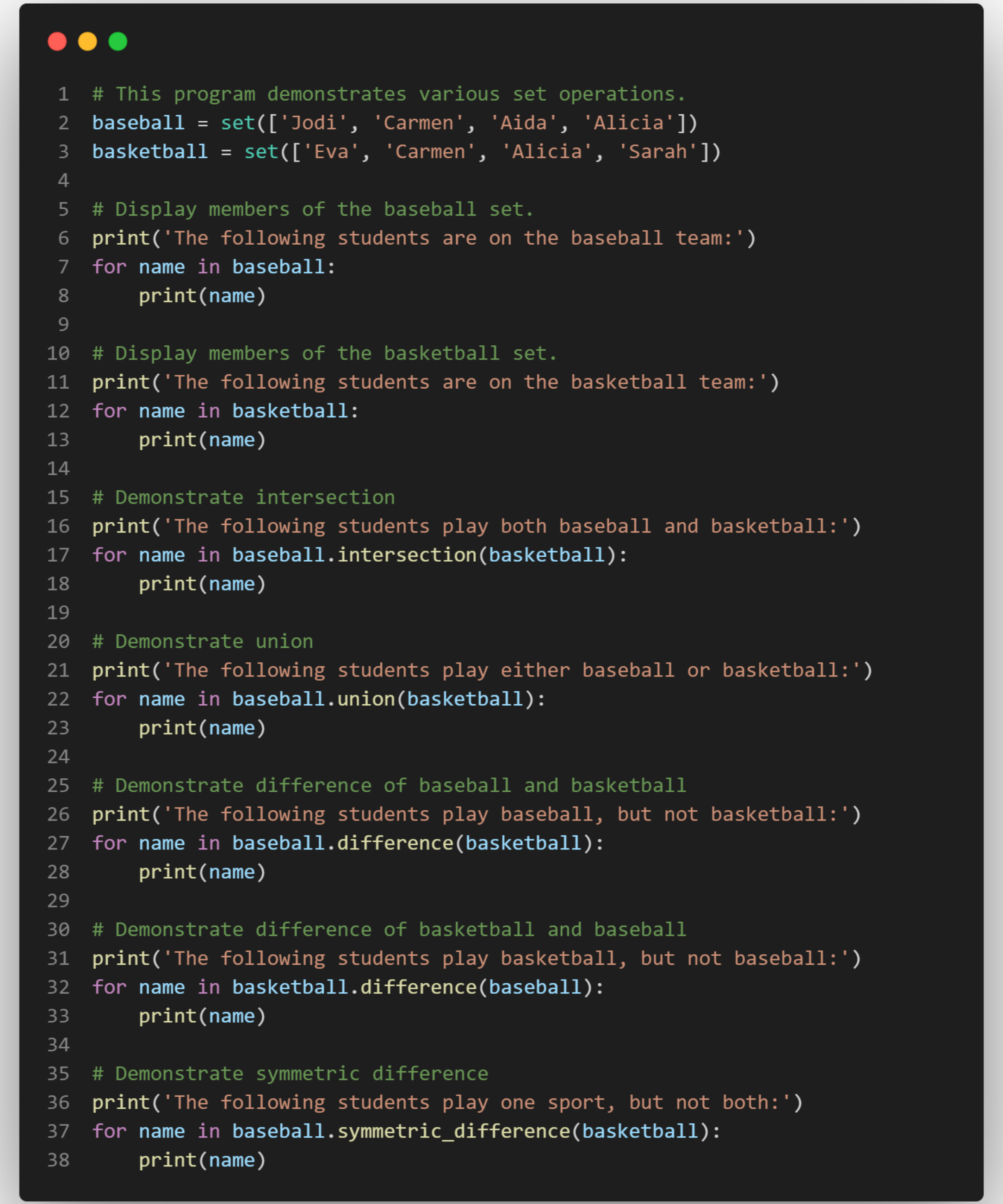
Finding Subsets and Supersets



```python
set1 = set([1, 2, 3, 4])
set2 = set([2, 3])
print(set2<=set1)# output True
print(set1>=set2)# output True
```



```python
set1 = set([1, 2, 3, 4])
set2 = set([2, 3])
print(set2.issubset(set1))# output True
print(set1.issuperset(set2))# output True
```

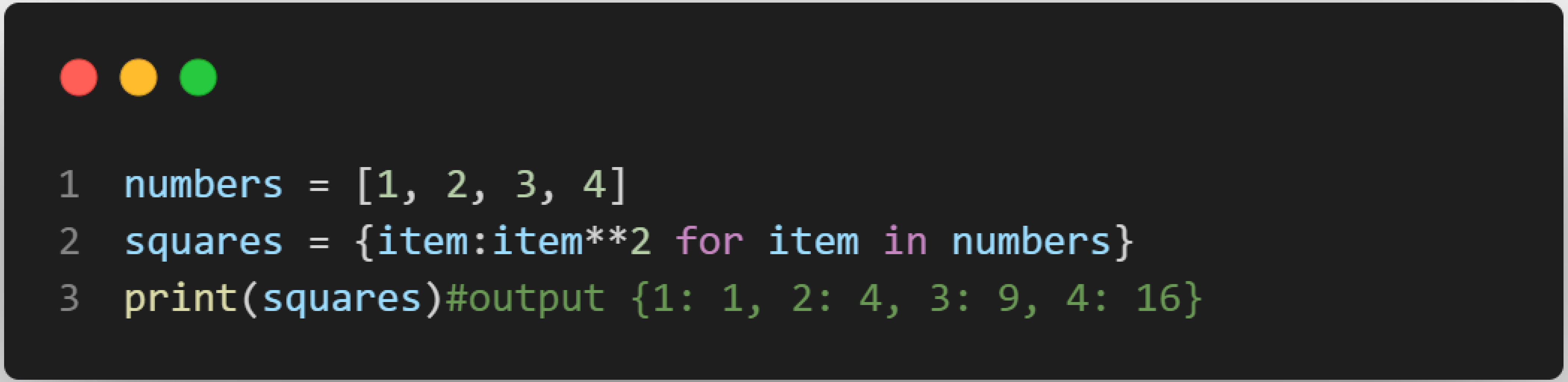


```python
# This program demonstrates various set operations.
baseball = set(['Jodi', 'Carmen', 'Aida', 'Alicia'])
basketball = set(['Eva', 'Carmen', 'Alicia', 'Sarah'])

# Display members of the baseball set.
print('The following students are on the baseball team:')
for name in baseball:
    print(name)

# Display members of the basketball set.
print('The following students are on the basketball team:')
for name in basketball:
    print(name)

# Demonstrate intersection
print('The following students play both baseball and basketball:')
for name in baseball.intersection(basketball):
    print(name)

# Demonstrate union
print('The following students play either baseball or basketball:')
for name in baseball.union(basketball):
    print(name)

# Demonstrate difference of baseball and basketball
print('The following students play baseball, but not basketball:')
for name in baseball.difference(basketball):
    print(name)

# Demonstrate difference of basketball and baseball
print('The following students play basketball, but not baseball:')
for name in basketball.difference(baseball):
    print(name)

# Demonstrate symmetric difference
print('The following students play one sport, but not both:')
for name in baseball.symmetric_difference(basketball):
    print(name)
```

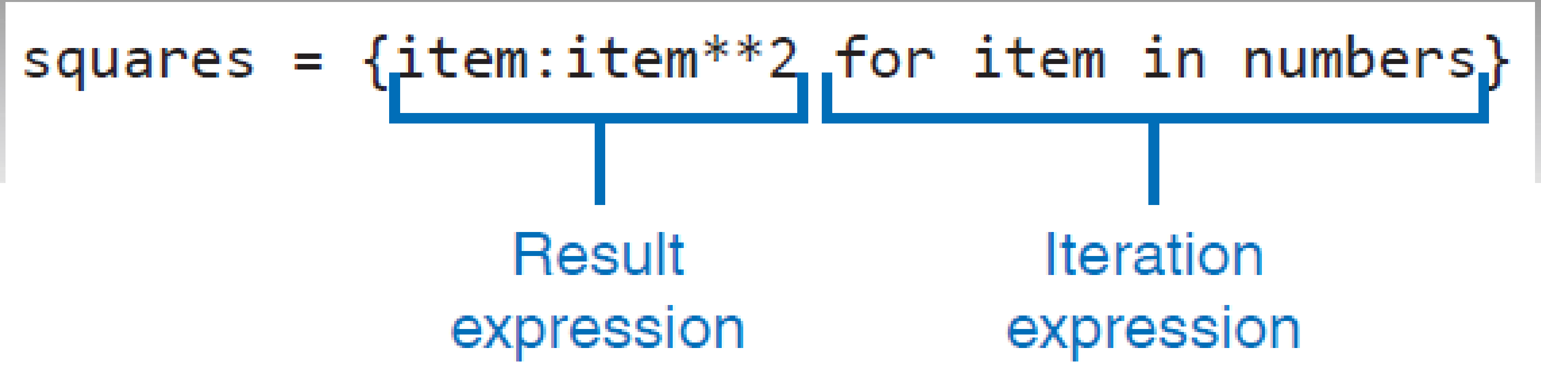
Set Operations Example

Appendix: Dictionary Comprehensions

A dictionary comprehension is an expression that reads a sequence of input elements and uses those input elements to produce a dictionary



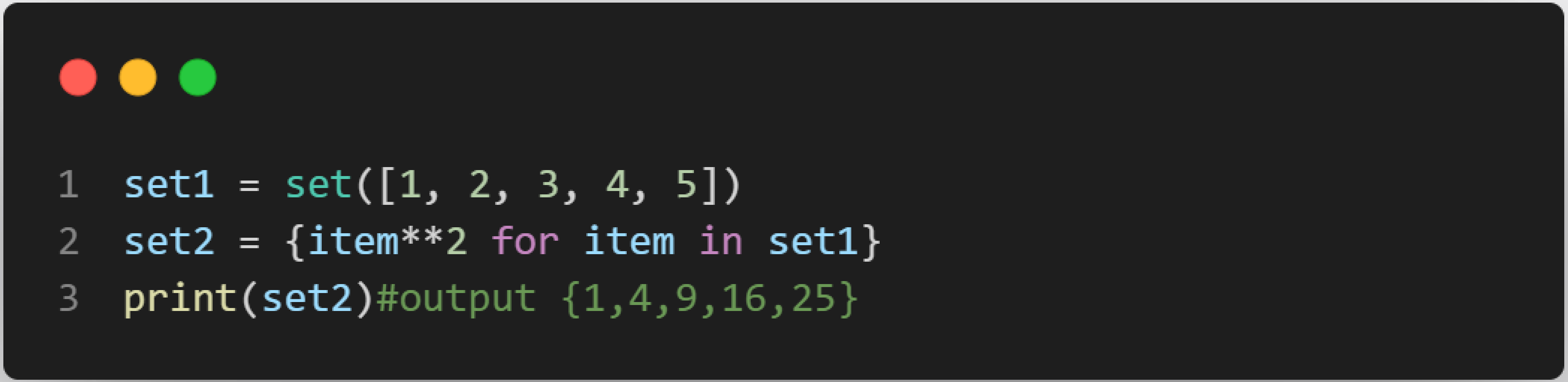
```python
numbers = [1, 2, 3, 4]
squares = {item:item**2 for item in numbers}
print(squares)#output {1: 1, 2: 4, 3: 9, 4: 16}
```



```python
squares = {item:item**2 for item in numbers}
```

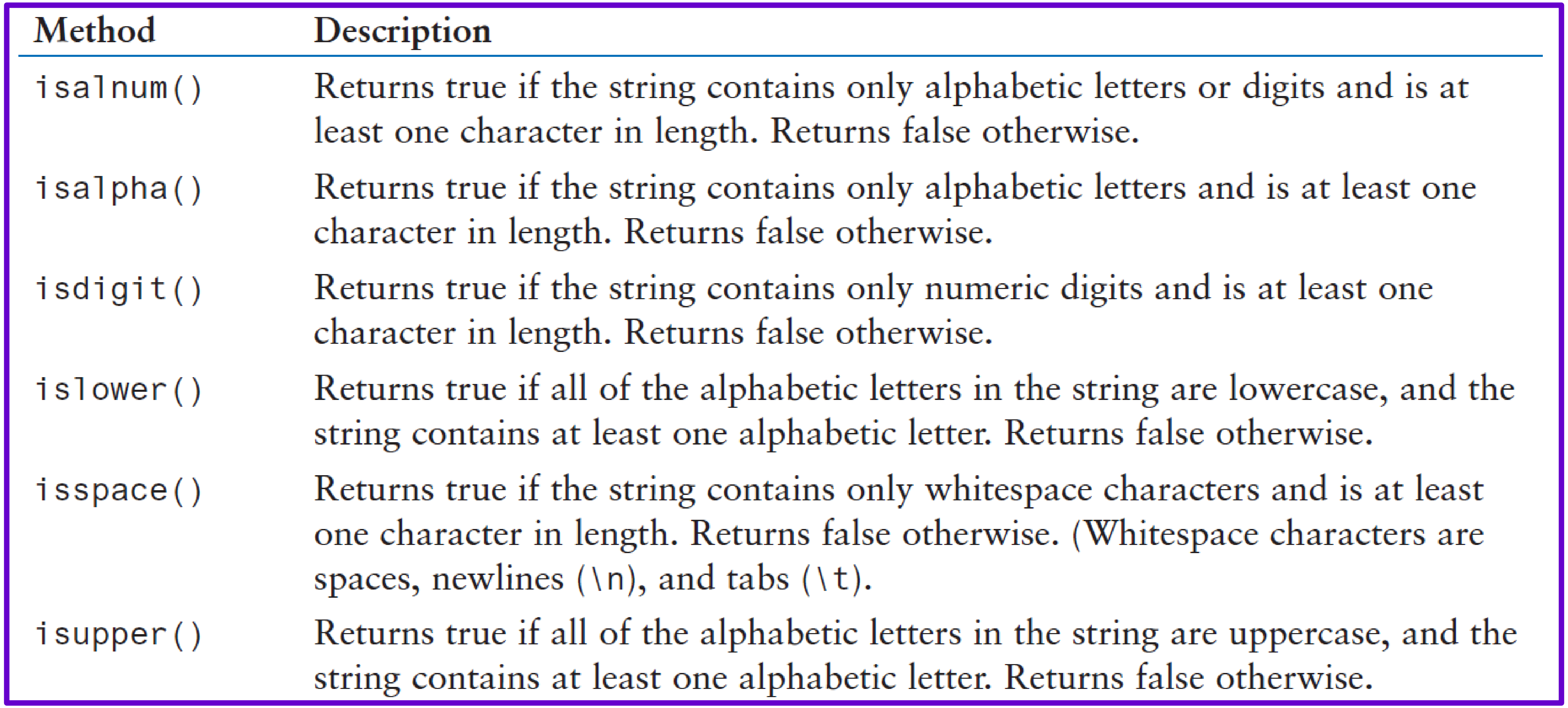
Appendix: Dictionary Comprehensions

A set comprehension is an expression that reads a sequence of input elements and uses those input elements to produce a set.



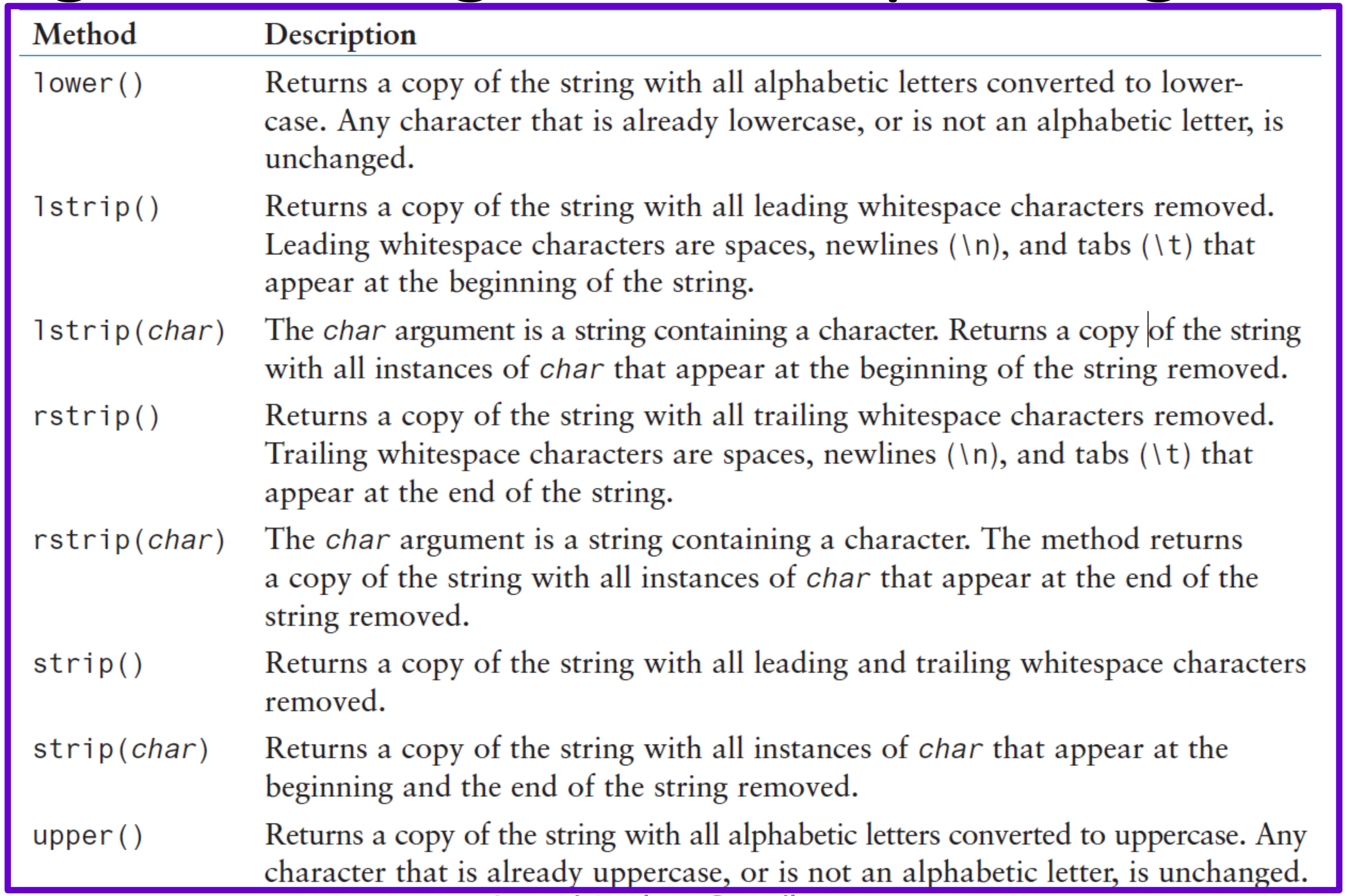
```python
set1 = set([1, 2, 3, 4, 5])
set2 = {item**2 for item in set1}
print(set2)#output {1,4,9,16,25}
```

Appendix : Testing, Searching, and Manipulating Strings



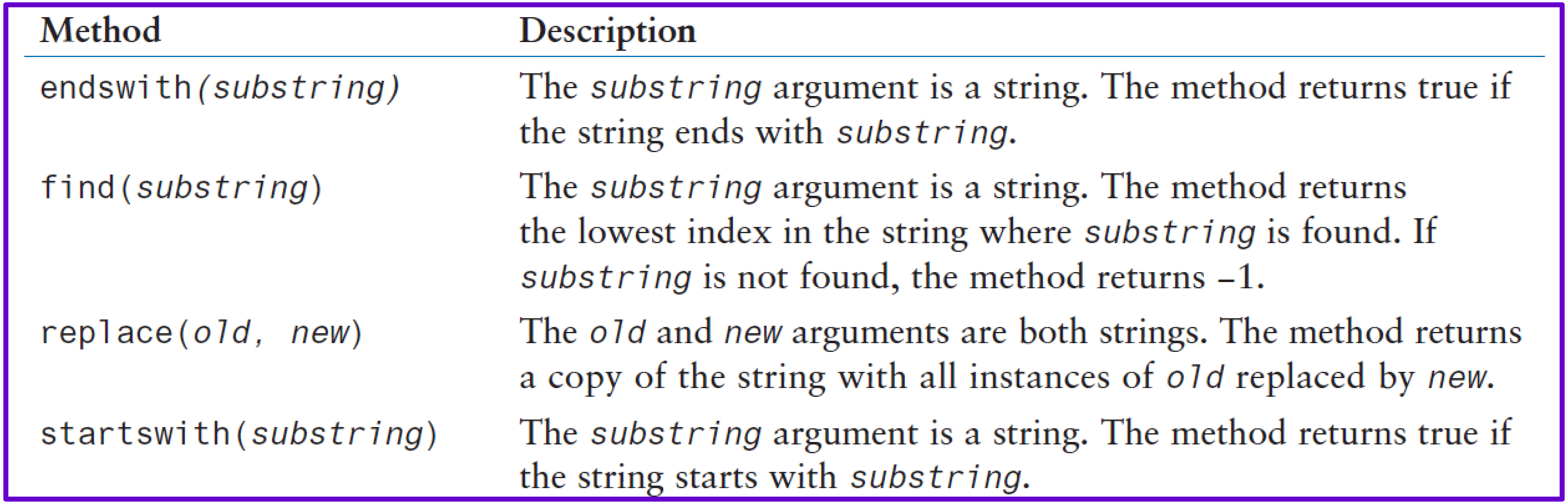
Method	Description
`isalnum()`	Returns true if the string contains only alphabetic letters or digits and is at least one character in length. Returns false otherwise.
`isalpha()`	Returns true if the string contains only alphabetic letters and is at least one character in length. Returns false otherwise.
`isdigit()`	Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.
`islower()`	Returns true if all of the alphabetic letters in the string are lowercase, and the string contains at least one alphabetic letter. Returns false otherwise.
`isspace()`	Returns true if the string contains only whitespace characters and is at least one character in length. Returns false otherwise. (Whitespace characters are spaces, newlines (`\n`), and tabs (`\t`).
`isupper()`	Returns true if all of the alphabetic letters in the string are uppercase, and the string contains at least one alphabetic letter. Returns false otherwise.

Appendix : Testing, Searching, and Manipulating Strings…



Method	Description
`lower()`	Returns a copy of the string with all alphabetic letters converted to lowercase. Any character that is already lowercase, or is not an alphabetic letter, is unchanged.
`lstrip()`	Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines (`\n`), and tabs (`\t`) that appear at the beginning of the string.
`lstrip(`*`char`*`)`	The *`char`* argument is a string containing a character. Returns a copy of the string with all instances of *`char`* that appear at the beginning of the string removed.
`rstrip()`	Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines (`\n`), and tabs (`\t`) that appear at the end of the string.
`rstrip(`*`char`*`)`	The *`char`* argument is a string containing a character. The method returns a copy of the string with all instances of *`char`* that appear at the end of the string removed.
`strip()`	Returns a copy of the string with all leading and trailing whitespace characters removed.
`strip(`*`char`*`)`	Returns a copy of the string with all instances of *`char`* that appear at the beginning and the end of the string removed.
`upper()`	Returns a copy of the string with all alphabetic letters converted to uppercase. Any character that is already uppercase, or is not an alphabetic letter, is unchanged.

Appendix : Testing, Searching, and Manipulating Strings…



Method	Description
endswith(*substring*)	The *substring* argument is a string. The method returns true if the string ends with *substring*.
find(*substring*)	The *substring* argument is a string. The method returns the lowest index in the string where *substring* is found. If *substring* is not found, the method returns –1.
replace(*old*, *new*)	The *old* and *new* arguments are both strings. The method returns a copy of the string with all instances of *old* replaced by *new*.
startswith(*substring*)	The *substring* argument is a string. The method returns true if the string starts with *substring*.