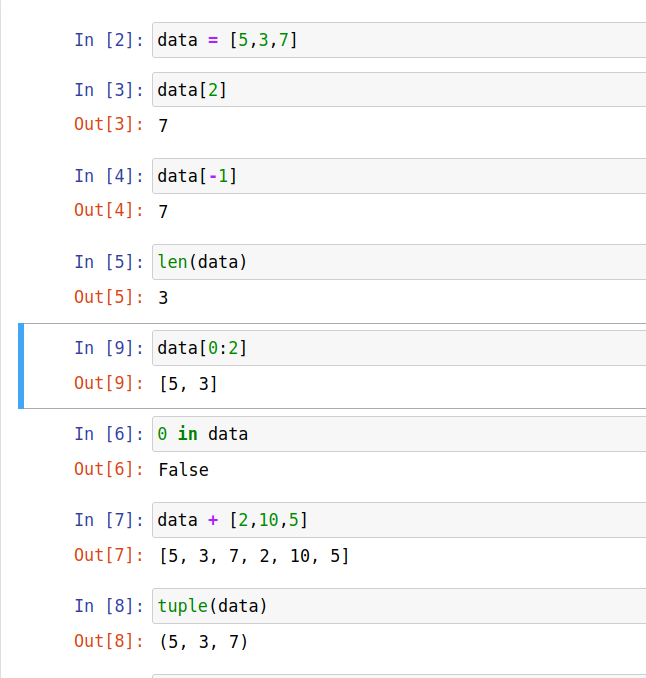
**Chapter 5 – Week 7 – Exercises**

Exercises #1 – page 145-146

1. **Assume that the variable data refers to the list [5, 3, 7]. Write the values of the following expressions:**
2. **data[2]** 7
3. **data[-1]** 7
4. **len(data)** 3
5. **data[0:2]** [5,3]
6. **0 in data** False
7. **data + [2, 10, 5]** [5,3,7,2,10,5]
8. **tuple(data)** (5,3,7)



1. **Assume that the variable data refers to the list [5,3,7]. Write the expressions that perfor the following tasks:**

a) Replace the value at position 0 in **data** with that value’s negation.

data[0] = -data[0]

b) Add the value 10 to the end of **data**.

data.append(10)

c) Insert the value 22 at position 2 in **data.**

data.insert(2,22)

d) Remove the value at position 1 in **data**.

data.pop(1)

e)Add the values in the list **newData** to the end of **data**.

data += newData

f) Locate the index of the value 7 in **data**, safely.

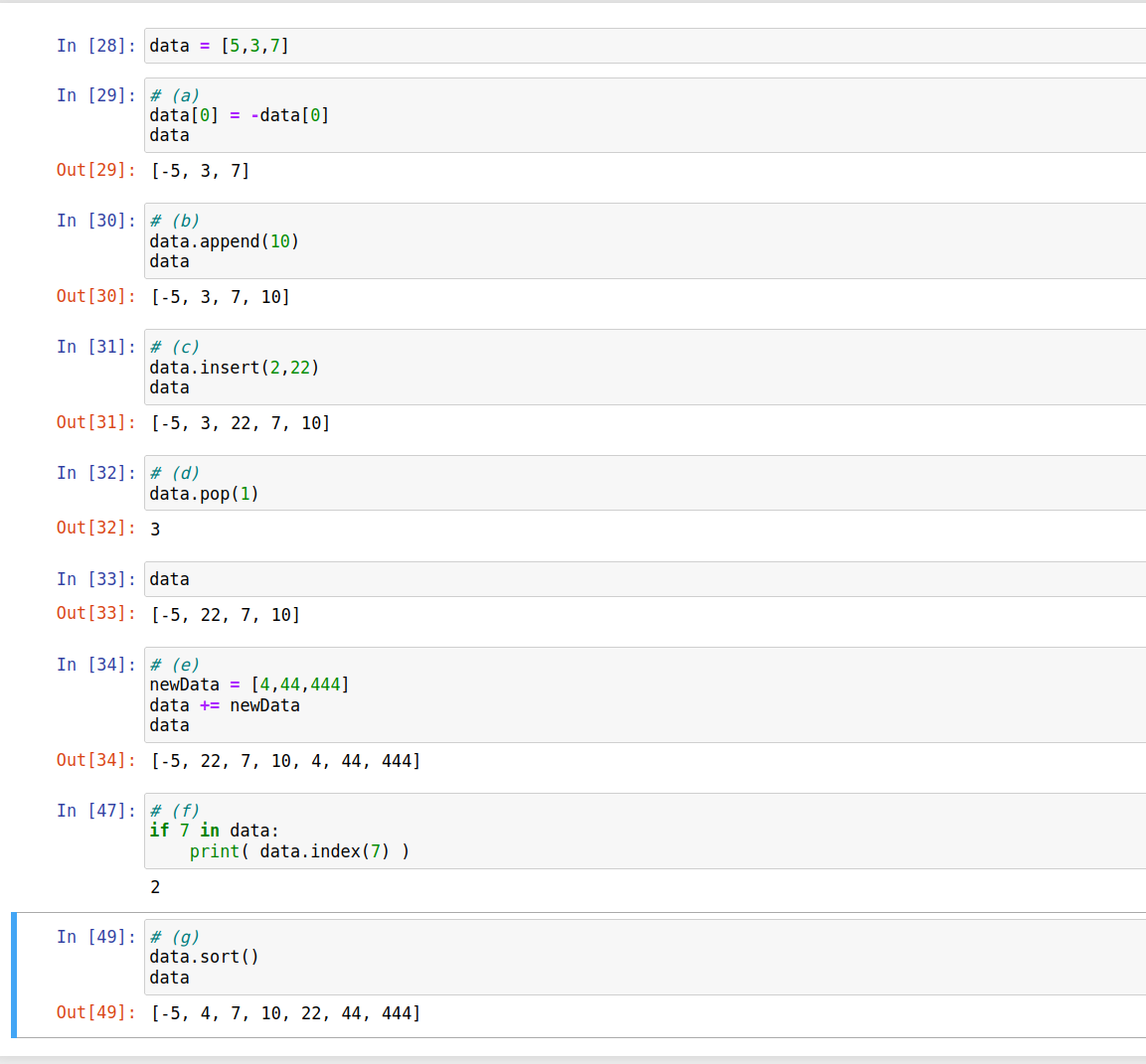
if 7 in data:

data.index(7)

g)Sort the values in **data**.

data.sort()

(Execution of code shown on next page.)



1. **What is the a mutator method? Explain why mutator methods usually return the value None.**

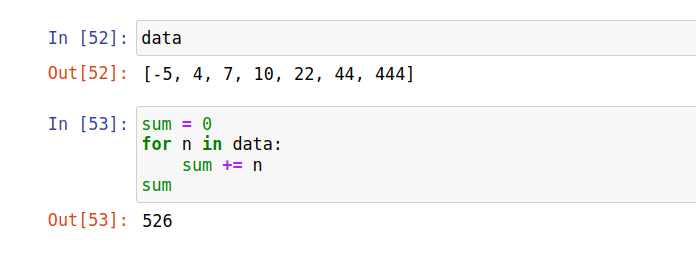
A mutator method is a method devoted entirely to modifying the internal state of the object. Because the change the internal state of the object, it is generally not necessary to return a value to the caller as the expected result is stored in the object the method was called on.

1. **Write a loop that accumulates the sum of the numbers in a list named data.**

sum = 0

for n in data:

sum += n

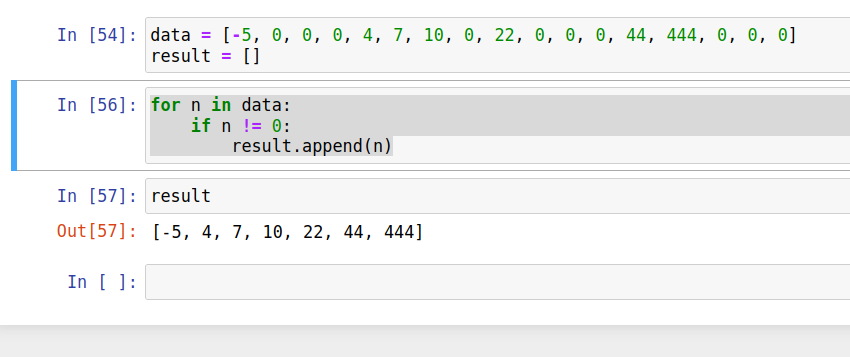
****

1. **Assume that data refers to a list of numbers, and result refers to an empty list. Write a loop that adds the nonzero values in data to the result list, keeping them in their relative positions and excluding the zeroes.**

for n in data:

if n != 0:

result.append(n)

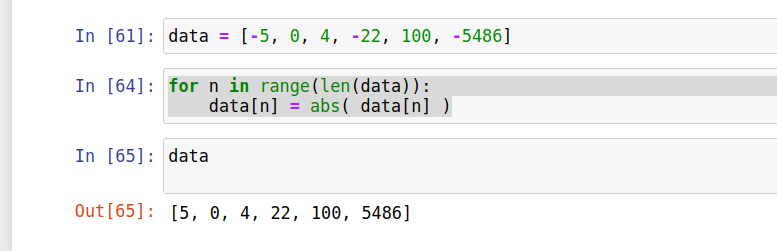


1. **Write a loop that replaces each number in a list named data with its absolute value.**

for n in range(len(data)):

data[n] = abs( data[n] )

(Code execution on next page.)



1. **Describe the costs and benefits of aliasing, and explain how it can be avoided.**

Aliasing is the phenomenon that occurs when multiple variables refer to the same object in memory. This is beneficial in that it only requires the space to store a single object even though multiple references are made to it.

On the other hand, having more that one variable pointing at the same object leads to bugs. These bugs occur because the modifying the object using one variable also modifies the contents of the second variable (because the refer to the SAME data). These bugs are relatively common and sometimes can be quite difficult to find and correct.

We can avoid aliasing by making a duplicate of the object (a second copy). This accomplished by making a new object and setting the necessary values to those stored in the first object.

1. **Explain the difference between structural equivalence and object identity.**

Object identity refers to the aliasing phenomenon discussed in #7. When multiple variables point the the same object, they are “object identity” equivalent. It comes with the pluses and minuses discussed in question #7. Object identity equivalence can be tested using the keyword “is”. For example list1 is list2 returns true only if the two variables refer to the same object in memory.

Structural equivalence is the state of 2 objects having the exact same value. They may be references to the same object or not. However, if they are not, all the values stored in both objects must be the exact same.

Exercises #2 – page 149

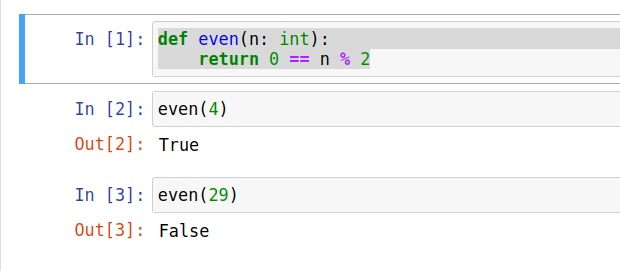
1. **What roles do the parameters and the return statement play in a function definition?**

In a function definition, the parameters represent the input to the function and the return statement represents the output from the function.

1. **Define a function named even. This function expects a number as an argument and returns True if the number is divisible by 2, or it returns False otherwise.**

def even(n: int):

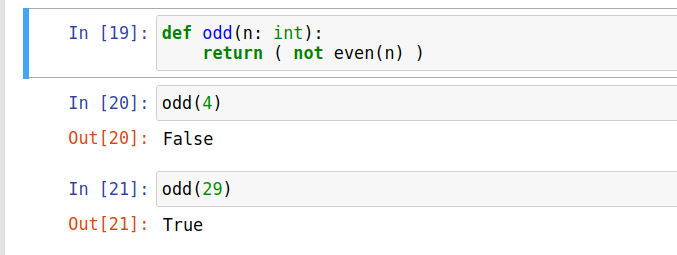
return 0 == n % 2



1. **Use the function even to simplify the definition of the function odd presented in this section.**

def odd(n: int):

return ( not even(n) )



1. **Define a function named summation. This Function expects two numbers, named low and high, as arguments. The function computes and returns the sum of the numbers between low and high inclusive.**

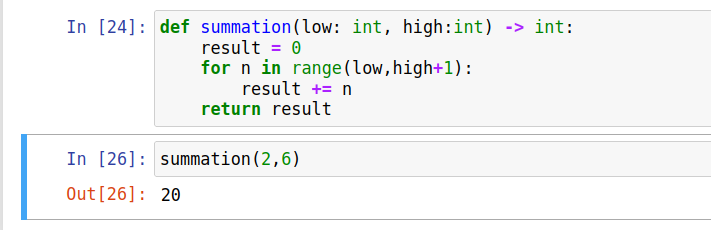
def summation(low: int, high:int) -> int:

result = 0

for n in range(low,high+1):

result += n

return result



1. **What is the purpose of the main function?**

The purpose of the main function is to be the entry point into the code.

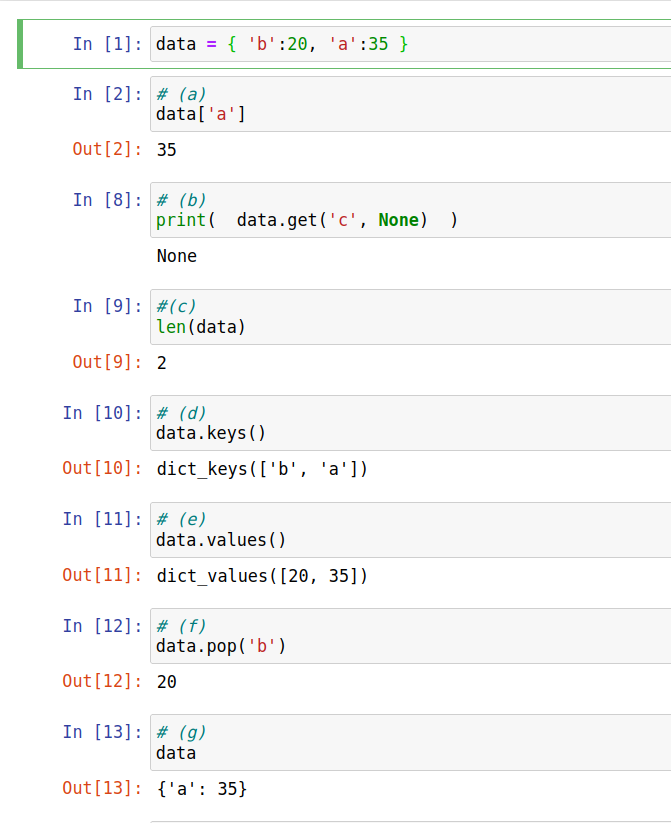
Exercises #3 – page 158

1. **Give three real-world objects that behave like a dictionary.**

Grocery list, contact lists, and actual dictionaries are three objects that behave in a similar manner to a dictionary data structure.

1. **Assume that the variable data refers to the dictionary {‘b’:20, ‘a’:35}. Write the values of the following expressions:**
2. data[‘a’] 35
3. data.get(‘c’, None) None
4. len(data) 2
5. data.keys() dict\_keys( [‘b’, ‘a’] )
6. data.values() dict\_values( [20, 35] )
7. data.pop(‘b’) 20
8. data # after above pop {‘a’:35}

Screenshot on next page.



1. **Assume that the variable data refers to the dictionary {‘b’:20, ‘a’:35}. Write the expressions that perform the following tasks:**
   1. Replace the value at the key **‘b’** in **data** with that value’s negation

data[‘b’] = -data[‘b’]

* 1. Add the key/value pair **‘c’:40** to **data**.

data[‘c’] = 40

* 1. Remove the value at key **‘b’** in **data**, safely.

if 'b' in data:

data.pop('b')

* 1. Print the keys in **data** in alphabetical order.

key\_list = list( data.keys() )

key\_list.sort()

