“HomeWork15” Justin Minsk

4.

def is\_sorted(lst):  
 lst\_temp = lst  
 lst\_temp = sorted(lst\_temp)  
 if lst == lst\_temp:  
 return True  
 else:  
 return False  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print(is\_sorted([]))

import is\_sorted as is\_sorted  
import unittest  
  
  
class Test\_list\_mag(unittest.TestCase):  
  
 def test\_list\_mag\_empty(self): # 1  
 *"""Test an empty list."""* argument = []  
 expected = True  
 argument = is\_sorted.is\_sorted(argument)  
 self.assertEqual(expected, argument, "The list is empty.")  
  
 def test\_running\_list\_mag\_item(self): # 2  
 *"""Test a one-item list."""* argument = [5]  
 expected = True  
 argument = is\_sorted.is\_sorted(argument)  
 self.assertEqual(expected, argument, "The list contains one item.")  
  
 def test\_running\_list\_mag\_items(self): # 3  
 *"""Test a two-item list."""* argument = [5, 2]  
 expected = False  
 argument = is\_sorted.is\_sorted(argument)  
 self.assertEqual(expected, argument, "The list contains two items.")  
  
 def test\_running\_list\_mag\_negative(self): # 4  
 *"""Test a list of negative values."""* argument = [-1, -5, -3, -4]  
 expected = False  
 argument = is\_sorted.is\_sorted(argument)  
 self.assertEqual(expected, argument, "The list contains only negative values.")  
  
 def test\_running\_list\_mag\_zeros(self): # 5  
 *"""Test a list of zeros."""* argument = [0, 0, 0, 0]  
 expected = True  
 argument = is\_sorted.is\_sorted(argument)  
 self.assertEqual(expected, argument, "The list contains only zeros.")  
  
 def test\_running\_list\_mag\_positive(self): # 6  
 *"""Test a list of positive values."""* argument = [4, 2, 3, 6]  
 expected = False  
 argument = is\_sorted.is\_sorted(argument)  
 self.assertEqual(expected, argument, "The list contains only positive values.")  
  
 def test\_running\_list\_mag\_mix(self): # 7  
 *"""Test a list containing mixture of negative values, zeros and positive values."""* argument = [-5, -2, 0, 1, 3, 4]  
 expected = True  
 argument = is\_sorted.is\_sorted(argument)  
 self.assertEqual(expected, argument, "The list contains a mixture of negative values, zeros and positive values"  
 ".")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

5.

def find\_min\_max(values):  
 *"""(list of numbers) -> NoneType  
  
 Print the minimum and maximum value from values.  
 """* min = values[0] # start at the first number  
 max = values[0] # start at the first number  
 for value in values:  
 if value > max:  
 max = value  
 if value < min:  
 min = value  
 print('The minimum value is {0}'.format(min))  
 print('The maximum value is {0}'.format(max))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 find\_min\_max([0, 1, 3, 4, 0, 2, -1])  
  
 # None type screws up the list and produces an error  
 # change line 6 and 7 to the first item on the list. This still does not help with mixed lists (string and num).

6.

import average as avg  
import unittest  
  
  
class Test\_list\_mag(unittest.TestCase):  
  
 def test\_running\_list\_mag\_item(self): # 2  
 *"""Test a one-item list."""* argument = [5]  
 expected = 5  
 argument = avg.average(argument)  
 self.assertEqual(expected, argument, "The list contains one item.")  
  
 def test\_running\_list\_mag\_items(self): # 3  
 *"""Test a two-item list."""* argument = [5, 2]  
 expected = 3.5  
 argument = avg.average(argument)  
 self.assertEqual(expected, argument, "The list contains two items.")  
  
 def test\_running\_list\_mag\_negative(self): # 4  
 *"""Test a list of negative values."""* argument = [-1, -5, -3, -4]  
 expected = -3.25  
 argument = avg.average(argument)  
 self.assertEqual(expected, argument, "The list contains only negative values.")  
  
 def test\_running\_list\_mag\_zeros(self): # 5  
 *"""Test a list of zeros."""* argument = [0, 0, 0, 0]  
 expected = 0  
 argument = avg.average(argument)  
 self.assertEqual(expected, argument, "The list contains only zeros.")  
  
 def test\_running\_list\_mag\_positive(self): # 6  
 *"""Test a list of positive values."""* argument = [None, 2, 3, 6]  
 expected = 3.6666666666666665  
 argument = avg.average(argument)  
 self.assertEqual(expected, argument, "The list contains positive values and None.")  
  
 def test\_running\_list\_mag\_mix(self): # 7  
 *"""Test a list containing mixture of negative values, zeros and positive values."""* argument = [-5, -2, 0, 1, 3, 4]  
 expected = 0.16666666666666666  
 argument = avg.average(argument)  
 self.assertEqual(expected, argument, "The list contains a mixture of negative values, zeros and positive values"  
 ".")  
  
 def test\_running\_list\_mag\_mix(self): # 7  
 *"""Test a list containing mixture of negative values, zeros and positive values."""* argument = [-5, None, -2, 0, 1, 3, 4, None]  
 expected = 0.16666666666666666  
 argument = avg.average(argument)  
 self.assertEqual(expected, argument, "The list contains a mixture of negative values, zeros, None, and positive"  
 " values.")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

import doctest  
  
def average(values):  
 *"""(list of numbers) -> number  
  
 >>> average([20, 30])  
 25.0  
 >>> average([None, 20, 30])  
 25.0  
 """* count = 0  
 total = 0  
 for value in values:  
 if value is not None:  
 total += value  
 count += 1  
 return total / count  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 doctest.testmod()  
 print(average([-5, -2, 0, 1, 3, 4]))

Part B.

ef list\_by\_mag(num\_list):  
 for nums in range(len(num\_list) - 1, 0, -1):  
 for i in range(nums):  
 if abs(num\_list[i]) > abs(num\_list[i + 1]):  
 temp = num\_list[i]  
 num\_list[i] = num\_list[i+1]  
 num\_list[i+1] = temp  
 if abs(num\_list[i]) == abs(num\_list[i + 1]):  
 if num\_list[i] > num\_list[i + 1]:  
 temp = num\_list[i]  
 num\_list[i] = num\_list[i + 1]  
 num\_list[i + 1] = temp  
 else:  
 temp = num\_list[i + 1]  
 num\_list[i + 1] = num\_list[i]  
 num\_list[i] = temp  
 if abs(num\_list[i]) == abs(num\_list[i - 1]):  
 if num\_list[i] > num\_list[i - 1]:  
 temp = num\_list[i]  
 num\_list[i] = num\_list[i - 1]  
 num\_list[i - 1] = temp  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 num\_list = [-2, -1, 2, 3, -5, 4, 2, -2, -4]  
 list\_by\_mag(num\_list)  
 print(num\_list)

import list\_by\_magnitude as ibm  
import unittest  
  
  
class Test\_list\_mag(unittest.TestCase):  
  
 def test\_list\_mag\_empty(self): # 1  
 *"""Test an empty list."""* argument = []  
 expected = []  
 ibm.list\_by\_mag(argument)  
 self.assertEqual(expected, argument, "The list is empty.")  
  
 def test\_running\_list\_mag\_item(self): # 2  
 *"""Test a one-item list."""* argument = [5]  
 expected = [5]  
 ibm.list\_by\_mag(argument)  
 self.assertEqual(expected, argument, "The list contains one item.")  
  
 def test\_running\_list\_mag\_items(self): # 3  
 *"""Test a two-item list."""* argument = [2, 5]  
 expected = [2, 5]  
 ibm.list\_by\_mag(argument)  
 self.assertEqual(expected, argument, "The list contains two items.")  
  
 def test\_running\_list\_mag\_negative(self): # 4  
 *"""Test a list of negative values."""* argument = [-1, -5, -3, -4]  
 expected = [-1, -3, -4, -5]  
 ibm.list\_by\_mag(argument)  
 self.assertEqual(expected, argument, "The list contains only negative values.")  
  
 def test\_running\_list\_mag\_zeros(self): # 5  
 *"""Test a list of zeros."""* argument = [0, 0, 0, 0]  
 expected = [0, 0, 0, 0]  
 ibm.list\_by\_mag(argument)  
 self.assertEqual(expected, argument, "The list contains only zeros.")  
  
 def test\_running\_list\_mag\_positive(self): # 6  
 *"""Test a list of positive values."""* argument = [4, 2, 3, 6]  
 expected = [2, 3, 4, 6]  
 ibm.list\_by\_mag(argument)  
 self.assertEqual(expected, argument, "The list contains only positive values.")  
  
 def test\_running\_list\_mag\_mix(self): # 7  
 *"""Test a list containing mixture of negative values, zeros and positive values."""* argument = [-2, -1, 2, 3, -5, 4, 2, -2, -4]  
 expected = [-1, -2, -2, 2, 2, 3, -4, 4, -5]  
 ibm.list\_by\_mag(argument)  
 self.assertEqual(expected, argument, "The list contains a mixture of negative values, zeros and positive values"  
 ".")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()