<table>
<thead>
<tr>
<th>Total points: 13/13</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1/1</strong> Of the following below, which would qualify as an implementation detail (generally NOT desired) if included in a doc string?</td>
</tr>
</tbody>
</table>

- [ ] Using the data in "data.csv", computes the average integer in the 'age' column.
- [x] Loops through the given list to find the average integer.
- [ ] Prints the most common age.
- [ ] Computes the average age assuming that all ages are positive.

**Correct Answer:**
Loops through the given list to find the average integer.

**Feedback:**
"Loops through the given list to find the average integer" is an implementation detail because it is feasible for a function to find an average integer without using a loop.

Mention of "data.csv" is not an implementation detail because it is a relevant input that the caller of said function might use to change the output of the function.

<table>
<thead>
<tr>
<th>1/1</th>
<th>Given the following function specification, which of the tests below are <strong>NOT</strong> appropriate?</th>
</tr>
</thead>
</table>

```python
def mode(numbers):
    """ Given a list of integers, returns the mode, defined as the value that appears the most times. If there are multiple modes, then any one might be returned. Returns None if the list is empty """

    assert(mode([1, 1, 2, 2]) == 1 or mode([1, 1, 2, 2]) == 2)
    assert(mode([1, 2, 3, 3]) == 3)
    assert(mode([]) == None)
    assert(mode([1, 1.5, 2, 2.5, 2]) == 2)
```


Correct

Answer:
assert(mode([1, 1.5, 2, 2.5, 2]) == 2)

Feedback:
assert(mode([1, 1.5, 2, 2.5, 2]) == 2) is not an appropriate test because the function specification assumes that the function will be passed a list of integers. The function does not need to be tested for a case that involves non-integers because it makes no guarantees about the return values of non-integers.

Consider the following function. Of the lines of code labelled A, B, C, and D, which would be the most appropriate to move to a helper function?

```python
def extract_firstnames(filename):
    """ Given the name of a file, where in the file all lines are of the format: 
    "lastname", "firstname"
    Returns a list of all of the Firstname strings
    ""
    file = open(filename)  # A
    names = []  # B
    for line in file:
        firstname = line.split(" ", ")[1][1:-1]  # C
        names.append(firstname)  # D
    return names
```

- file = open(filename)
- names = []
- line.split(" ", ")[1][1:-1]
- names.append(firstname)

Correct

Answer:
line.split(" ", ")[1][1:-1]

Feedback:
`firstname = line.split(" ", ")[1][1:-1]` is the most appropriate to move to a helper function. Even though it is only one line of code (like the others), it is fairly complex and is a sub problem that distracts from the main problem at hand.

You wrote a simple program to calculate standard deviation for a given data list:
import math

sum = 0.0
for item in data:
    sum = sum + item
average = sum / len(data)
diff_sq = 0.0
for item in data:
    diff_sq = diff_sq + (item - average) ** 2
std = math.sqrt(diff_sq / (len(data) - 1))

Now you want to merge your work with your friend's. However, it seems that your friend has already used "data" for some other variable's name and the standard deviation calculation is needed in various places. How would you refactor your program in order to merge?

- Copy the code into each place that you need to calculate standard deviation.
- Put all the calculations into functions to hide implementation details.
- Change all variables named "data" to some other non-used name, for example, "data1."

**Correct Answer:**

**Answer:**
Put all the calculations into functions to hide implementation details.

**Feedback:**
This is enough code that it should be placed into a function and given a name, independent of the fact that you have a name conflict with the variable data. You definitely don't want to copy the code multiple places. Changing the variable name will work but there is no need to do this and the code is enough to be in a function which solves the name conflict problem.

1/1 True/False: A dictionary is an abstract data type.

- True
- False

**Correct**

**Answer:**
True

1/1 True/False: Python classes are tools for implementing data abstractions.

- True
- False
True/False: An interface should specify both the behavior of an operation, and the details of how the operation is implemented.

☐ True
☐ False

Correct
Answer:
False

2/2

Given the following class:

class IntSet(object):

    """An intSet is a set of integers""
    # Information about the implementation (not the abstraction)
    # The value of the set is represented by a list of ints, self.vals.
    # Each int in the set occurs in self.vals exactly once.
    def __init__(self):
        """Create an empty set of integers""
        self.vals = []

    def insert(self, e):
        """Assumes e is an integer and inserts e into self""
        if not e in self.vals:
            self.vals.append(e)

    def member(self, e):
        """Assumes e is an integer Returns True if e is in self, and False otherwise""
        return e in self.vals

    def remove(self, e):
        """Assumes e is an integer and removes e from self
        Raises ValueError if e is not in self""
        try:
            self.vals.remove(e)
        except:
Write code that will create an instance of the IntSet class called my_int_set.

```python
my_int_set = IntSet()
```

**Correct Answer:**
```
my_int_set = IntSet()
```
**Feedback:**
This calls the constructor.

Given the class shown above, and an instance of IntSet called my_int_set, add the integer 7 to my_int_set.

```python
my_int_set.insert(7)
```

**Correct Answer:**
```
my_int_set.insert(7)
```
**Feedback:**

Given the class shown above, and an instance of IntSet called `big_int_set`, write code that will print "Found 8" if the integer 8 is in `big_int_set` (and print nothing if it is not).

```python
if big_int_set.member(8):
    print "Found 8"
```

**Correct Answer:**
```
if big_int_set.member(8):
    print "Found 8"
```
**Feedback:**

**Total points:**
13/13