Feedback — Week 2 Exercise

You submitted this homework on **Thu 11 Apr 2013 3:34 AM PDT -0700**. You got a score of **10.00** out of **10.00**.

This exercise is about choosing test cases and also about writing unit tests. Plan to spend time thinking carefully about each question and each possible answer.

As a warning, each time you attempt this exercise, the options may be reordered, and you may be presented with different options.

Question 1

Consider this code:

```
def high_low(guess, actual):
    """ (int, int) -> str

    Return "your guess is low" if guess is lower than actual, "your
 guess is high" if guess is higher
    than actual, and "correct" if guess is equal to actual.

    >>> high_low(4, 10)
    "your guess is low"
    """
```

Which set of test cases is the *best* choice of tests cases for this function? (Think about the **Boundaries** category for choosing test cases.)

Your Answer	Score	Explanation
○ • `guess` and `actual` refer to negative values • `guess` and `actual` refer to zero • `guess` and `actual` refer to positive values		
○		

- `guess` and `actual` refer to odd values
- `guess` and `actual` refer to zero
- `guess` and `actual` refer to non-zero even values

○
- `guess` refers to `"higher"` and `actual` refers to `"lower"`
- `guess` refers to `"correct"` and `actual` refers to `"correct"`
- `guess` refers to `"lower"` and `actual` refers to `"higher"`

● ✓ 1.00
- `guess` refers to a value that is less than the value referred to by `actual`
- `guess` refers to a value equal to the value referred to by `actual`
- `guess` refers to a value that is greater than the value referred to by `actual`

Total 1.00 / 1.00

Question 2

Consider this code:

```
def first_two_items(L):
    """ (list of str) -> list of str

    Return the first two items in L. If there are fewer than two it
ems in L,
    return all of the items.

    >>> first_two_items(["apple", "pear", "grape"])
    ["apple", "pear"]
    """
```

Which of these sets of values for `L` is the *best* choice of tests cases for this function?

(Think about the **Size** category for choosing test cases.)

Your Answer	Score	Explanation
○ • `[]` • `["one"]`		
○ • `[]` • `["one"]` • `["one", "two"]` • `["one", "two", "three"]` • `["one", "two", "three", "four"]` • `["one", "two", "three", "four", "five"]` • `["one", "two", "three", "four", "five", "six"]` • More tests like this, with lists up to 20 items long.		
○ • `["one", "two"]` • `["one", "two", "three"]` • `["one", "two", "three", "four"]`		
● • `[]` • `["one"]` • `["one", "two"]` • `["one", "two", "three", "four"]`	✔ 1.00	
Total	1.00 / 1.00	

Question 3

Consider this code:

```
def is_preschooler(age):
    """ (int) -> bool

    Precondition: age >= 0

    Return True if and only if age is between 3 and 5 inclusive.

    >>> is_preschooler(4)
    True
    """
```

Which set of test cases is the *best* choice of tests cases for this function? (Think about which test case category you might want to consider.)

Your Answer		Score	Explanation
○ • one number between `0` and `2` • `4` • one number over `5`			
○ • one number under `0` • `0` • one number over `0`			
○ • one number between `0` and `2` • one number between `3` and `5` • one number over `5`			
● • `0` or `1` • `2` • `3` • `4` • `5` • `6` • one number over `6`	✔	1.00	
Total		1.00 / 1.00	

Question 4

Consider this code:

```
def count_occurrences(s, ch):
    """ (str, str) -> int

    Precondition: len(ch) == 1

    Return the number of occurrences of ch in s.

    >>> count_occurrences("hello", "l")
    2
    """
```

Which of these sets of values for `s` and `ch` is the *best* choice of tests cases for this function? (There are several test case categories for this question, including at least **Size** and **Dichotomies**.)

Your Answer	Score	Explanation

○

- `s` refers to `''`, `ch` refers to `'a'`
- `s` refers to `'a'`, `ch` refers to `'a'`
- `s` refers to `'aaaaaa'`, `ch` refers to `'a'`

○

- `s` refers to `''`, `ch` refers to `'a'`
- `s` refers to `'a'`, `ch` refers to `'a'`
- `s` refers to `'1'`, `ch` refers to `'1'`
- `s` refers to `'a'`, `ch` refers to `'b'`
- `s` refers to `'abc'`, `ch` refers to `'b'`
- `s` refers to `'123'`, `ch` refers to `'2'`
- `s` refers to `'abc'`, `ch` refers to `'d'`
- `s` refers to `'abcabca'`, `ch` refers to `'a'`

○

- `s` refers to `''`, `ch` refers to `'b'`
- `s` refers to `'a'`, `ch` refers to `'b'`
- `s` refers to `'aaaaaa'`, `ch` refers to `'b'`

●	✓	1.00

- `s` refers to `''`, `ch` refers to `'a'`
- `s` refers to `'a'`, `ch` refers to `'a'`
- `s` refers to `'a'`, `ch` refers to `'b'`
- `s` refers to `'abc'`, `ch` refers to `'b'`
- `s` refers to `'abc'`, `ch` refers to `'d'`
- `s` refers to `'abcabca'`, `ch` refers to `'a'`

Total	1.00 / 1.00

Question 5

Consider this code:

```
def can_vote(age):
    """ (int) -> bool

    Precondition: age >= 0

    Return True if and only if a person aged age can vote in Canada
.
    The legal voting age in Canada is 18 years and older.
    """

    return age > 18
```

There is a bug in the implementation of the function above (the code does not work as described). Select the test(s) that reveal the bug. (Think about which test case category might help you reveal this bug.)

Your Answer		Score	Explanation
☐ `age` is between `0` and `18`, exclusive	✓	0.25	
☐ `age` is `0`	✓	0.25	
☐ `age` is over `18`	✓	0.25	
☒ `age` refers to `18`	✓	0.25	

Total	1.00 / 1.00

Question 6

Consider this code:

```
def contains_item(L, s):
    """ (list, object) -> bool

    Return True if and only if s is an item of L.
    """

    for item in L:
        if item == s:
            return True
        else:
            return False
```

There are bugs in the implementation of the function above (the code does not work as described). Select the test(s) that reveal the bugs. (Think about the **Order** category for choosing test cases.)

Your Answer		Score	Explanation
☒ `L` refers to `[]`, `s` refers to `1`	✔	0.25	
☐ `L` refers to `[1, 2, 3]`, `s` refers to `1`	✔	0.25	
☒ `L` refers to `[1, 2, 3]`, `s` refers to `3`	✔	0.25	
☒ `L` refers to `[1, 2, 3]`, `s` refers to `2`	✔	0.25	
Total		1.00 / 1.00	

Question 7

Consider this code:

```
def sum_items(L):
    """ (list of number) -> number

    Return the sum of the items in L.
    """

    total = 0

    for item in L:
        total = item

    return total
```

There is a bug in the implementation of the function above (the code does not work as described). Select the test(s) that reveal the bug.

Your Answer		Score	Explanation
☐ `L` refers to `[1]`.	✓	0.20	As expected, `total` refers to `1`.
☒ `L` refers to `[1, 2]`.	✓	0.20	`total` refers to `2` (the item `L[-1]`), but it should refer to `3`.
☒ `L` refers to `[1, 0, 1]`.	✓	0.20	`total` refers to `1` (the item `L[-1]`), but it should refer to `2`.
☐ `L` refers to `[]`.	✓	0.20	As expected, `total` refers to `0`.
☐ `L` refers to `[0, 0, 2]`.	✓	0.20	As expected, `total` refers to `2`.
Total		1.00 / 1.00	

Question 8

Consider this code:

```
def can_afford(item_cost, wallet_money):
    """ (float, float) -> bool

    Return True if and only if wallet_money is greater or equal to
item_cost.

    >>> can_afford(3.42, 10.00)
    True
    >>> can_afford(27.32, 5.00)
    False
    """
```

You are implementing a `unittest` test class to test `can_afford`. Which of the following names can be used as the name of one of your test methods?

Your Answer		Score	Explanation
☒ `test_can_afford`	✔	0.25	
☐ `can_afford_test`	✔	0.25	
☐ `Test_Can_Afford`	✔	0.25	
☐ `TEST_CAN_AFFORD`	✔	0.25	
Total		1.00 / 1.00	

Question 9

Consider this code, which is in a file called `words.py`:

```
def word_frequency(letter_to_words):
    """ (dict of {str: list of str}) -> dict of {str: int}

    Precondition: the length of each list in letter_to_words is at
least 1,
    each key in letter_to_words is a lowercase letter, and each val
ue is a
    list of lowercase words beginning with that letter.

    Return a dictionary where the keys are the letters from letter_
to_words,
    and the values are the number of words beginning with that lett
er in
    letter_to_words.

    >>> d = {'a': ['apple'], 'b': ['beet', 'banana'], 'c': ['carrot
', 'cucumber']}
    >>> expected = {'a': 1, 'b': 2, 'c': 2}
    >>> word_frequency(d) == expected
    True
    """
```

Consider this `unittest` method header:

```
def test_word_frequency(self):
    """ A dictionary with several items."""

    # Add body here.
```

Select the `unittest` method body(ies) that are equivalent to the `doctest` in the `word_frequency` docstring. You can assume that `words` has been imported.

Your Answer		Score	Explanation

```
    d = {'a': ['
apple'], 'b': ['
beet', 'banana']
, 'c': ['carrot'
, 'cucumber']}
    actual = wor
ds.word_frequenc
y(d)
    expected = {
'a': 1, 'b': 2,
'c': 2}
    self.assertE
qual(actual, exp
ected)
```
	✔	0.33	

<pre> d = {'a': ['</pre>	✔	0.33	The order of the key-value pairs is irrelevant: `{'c': 2, 'b': 2, 'a': 1}` and `{'a': 1, 'b': 2, 'c': 2}` are considered equal.

```
apple'], 'b': ['
beet', 'banana']
, 'c': ['carrot'
, 'cucumber']}
    actual = wor
ds.word_frequenc
y(d)
    expected = {
'c': 2, 'b': 2,
'a': 1}
    self.assertE
qual(actual, exp
ected)
```

☐

```
    d = {'a': ['
apple'], 'b': ['
beet', 'banana']
, 'c': ['carrot'
, 'cucumber']}
    words.word_f
requency(d)
    expected = {
'a': 1, 'b': 2,
'c': 2}
    self.assertE
qual(d, expected
)
```

✔ 0.33

This code calls `words.word_frequency`, but doesn't remember the return value in a variable. It needs to compare the actual result to the expected one.

Total 1.00 / 1.00

Question 10

Consider this code, which is in a file called `words.py`:

```
def make_uppercase(L):
    """ (list of str) -> NoneType
    Convert all letters in each string in L to uppercase.
    """
```

Consider this `unittest` method header:

```
def test_make_uppercase(self):
    """ A list with several items."""

    # Add body here.
```

Select the method body(ies) that correctly test the function with the list `['Ada Lovelace', 'Grace Hopper', 'Alan Turing']`. You can assume that `words` has been imported.

Your Answer		Score	Explanation
☐ `actual = words.make_uppercase(['Ada Lovelace', 'Grace Hopper', 'Alan Turing'])` `expected = ['ADA LOVELACE', 'GRACE HOPPER', 'ALAN TURING']` `self.assertEqual(actual, expected)`	✓	0.33	
☒ `L = ['Ada Lovelace', 'Grace Hopper', 'Alan Turing']` `words.make_uppercase(L)` `expected = ['ADA LOVELACE', 'GRACE HOPPER', 'ALAN TURING']` `self.assertEqual(L, expected)`	✓	0.33	
☐ `L = ['Ada Lovelace', 'Grace Hopper', 'Alan Turing']` `actual = words.make_uppercase(L)` `expected = ['ADA LOVELACE', 'GRACE HOPPER', 'ALAN TURING']` `self.assertEqual(actual, expected)`	✓	0.33	

Total	1.00 / 1.00