# Week 2 Exercise

测验, 10 个问题

1 point

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.)

- `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"`

- `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`

- `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

---

1 point

# Week 2 Exercise

测验, 10 个问题

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 items 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.)

- [ ]
  - `["one", "two"]`
  - `["one", "two", "three"]`
  - `["one", "two", "three", "four"]`
- [x]
  - `[]`
  - `["one"]`
  - `["one", "two"]`
  - `["one", "two", "three", "four"]`
- [ ]
  - `[]`
  - `["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.

---

1 point

3。

# Week 2 Exercise

测验, 10 个问题

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.)

- 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

- one number under 0
- 0
- one number over 0

- one number between 0 and 2
- 4
- one number over 5

---

1 point

4。

# Week 2 Exercise

测验, 10 个问题

Consider this code:

```
1   def count_occurrences(s, ch):
2       """ (str, str) -> int
3
4       Precondition: len(ch) == 1
5
6       Return the number of occurrences of ch in s.
7
8       >>> count_occurrences("hello", "l")
9       2
10      """
```

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**.)

- `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 `'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'`

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

1 point

5。

# Week 2 Exercise

测验, 10 个问题

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.)

- [ ] `age` is `0`
- [ ] `age` is between `0` and `18` , exclusive
- [ ] `age` is over `18`
- [x] `age` refers to `18`

---

1 point

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.)

- [ ] `L` refers to `[1, 2, 3]` , `s` refers to `1`
- [x] `L` refers to `[1, 2, 3]` , `s` refers to `3`
- [x] `L` refers to `[]` , `s` refers to `1`
- [x] `L` refers to `[1, 2, 3]` , `s` refers to `2`

# Week 2 Exercise

测验, 10 个问题

1 point

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.

- [ ] `L` refers to `[0, 0, 2]` .
- [x] `L` refers to `[1, 0, 1]` .
- [x] `L` refers to `[1, 2]`
- [ ] `L` refers to `[]` .
- [ ] `L` refers to `[1]`

1 point

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?

- [x] `test_can_afford`
- [ ]

- [ ] `TestCanAfford`
- [ ] `TEST_CAN_AFFORD`
- [ ] `can_afford_test`

---

1 point

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 value 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 letter 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_frequencey` docstring. You can assume that `words` has been imported.

- [x]
```
d = {'a': ['apple'], 'b': ['beet', 'banana']
    , 'c': ['carrot', 'cucumber']}

actual = words.word_frequency(d)

expected = {'a': 1, 'b': 2, 'c': 2}

self.assertEqual(actual, expected)
```

- [x]

# Week 2 Exercise

测验, 10 个问题

```
d = {'a': ['apple'], 'b': ['beet', 'banana']
    , 'c': ['carrot', 'cucumber']}

actual = words.word_frequency(d)

expected = {'c': 2, 'b': 2, 'a': 1}

self.assertEqual(actual, expected)
```

```
d = {'a': ['apple'], 'b': ['beet', 'banana']
    , 'c': ['carrot', 'cucumber']}

words.word_frequency(d)

expected = {'a': 1, 'b': 2, 'c': 2}

self.assertEqual(d, expected)
```

---

1 point

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.

- [x]
```
L = ['Ada Lovelace', 'Grace Hopper', 'Alan
    Turing']

words.make_uppercase(L)

expected = ['ADA LOVELACE', 'GRACE HOPPER',
    'ALAN TURING']

self.assertEqual(L, expected)
```

- [ ]
```
L = ['Ada Lovelace', 'Grace Hopper', 'Alan
    Turing']

actual = words.make_uppercase(L)

expected = ['ADA LOVELACE', 'GRACE HOPPER',
    'ALAN TURING']

self.assertEqual(actual, expected)
```

- [ ]

# Week 2 Exercise

测验, 10 个问题

```
actual = words.make_uppercase(['Ada
    Lovelace', 'Grace Hopper', 'Alan
    Turing'])

expected = ['ADA LOVELACE', 'GRACE HOPPER',
    'ALAN TURING']

self.assertEqual(actual, expected)
```

我（**伟臣 沈**）了解提交不是我自己完成的作业 将永远不会通过此课程或导致我的 Coursera 帐号被关闭。 了解荣誉准则的更多信息

提交测试