University of Toronto - Computer Science - Image

Choosing Test Cases

Choosing tests for `count_lowercase_vowels`

```
def count_lowercase_vowels(s):
    """ (str) -> int

    Return the number of vowels (a, e, i, o, and u) in s.

    >>> count_lowercase_vowels('Happy Anniversary!')
    4
    >>> count_lowercase_vowels('xyz')
    0
    """

    num_vowels = 0

    for char in s:
        if char in 'aeiou':
            num_vowels = num_vowels + 1
        return num_vowels
```

To test `count_lowercase_vowels`, we need to:

- pick values for the string argument, and
- call the function to ensure it returns what we expect for each case.

It is not realistic to test using every single possible string argument. Instead, we create relevant categories, and choose one representative string argument from each category. To choose the string argument, consider:

- the length of the string, and
- the characters that make up the string.

There are many possible string lengths. For this example, we'll consider strings that have these lengths:

- 0 (empty)
- 1 (single character)
- 6 (longer)

Which characters should we use? For this example, we'll choose characters based on whether they are vowels or non-vowels. The actual character doesn't matter.
If we want a non-vowel, we could use `'b'`, `'n'`, `'?'`, or any other character that is not a vowel.

We will make a table of the test cases with the following 3 columns:

- value of the argument to the function
- expected return value of the function
- a description of the test case

s	**Expected Value**	**Description**
`''`	`0`	`empty string`
`'a'`	`1`	`single char, vowel`
`'b'`	`0`	`single char, non-vowel`

'pffffft'	0	several chars, no vowels
'bandit'	2	several chars, some vowels
'aeioua'	6	several chars, all vowels*

* we included all 5 vowels to ensure each one is properly counted.

Choosing tests for `is_palindrome`

```
def is_palindrome(s):
    """ (str) -> bool

    Return True if and only if s is a palindrome.

    >>> is_palindrome('noon')
    True
    >>> is_palindrome('racecar')
    True
    >>> is_palindrome('dented')
    False
    """
```

Because the function returns a Boolean value, we need *at least* 2 test cases: one that returns `True` and one that returns `False`. In this case, we actually need quite a few more than 2 test cases.

As for the previous example, we need to choose different values for the string argument that represent different categories of strings.

Dichotomy

When we developed the code for `is_palindrome()`, we found that whether a string was even or odd affected the code. Therefore, the tests should consider strings that have even and odd lengths. For this example, we'll consider strings that have these lengths:

- 0 and 1 (which are both considered palindromes)
- 2 (smallest possible non-empty even length palindrome, smallest possible non-palindrome)
- 3 (smallest possible multiple-character odd length palindrome)
- 6 (longer, even length string)
- 7 (longer, odd length string)

The test cases are summarized in this table:

s	Expected Value	Description
''	True	empty string
'a'	True	single character
'aa'	True	2 chars, palindrome
'ab'	False	2 chars, not palindrome
'aba'	True	3 chars, palindrome
'abc'	False	3 chars, not palindrome
'redder'	True	longer, even, palindrome
'renter'	False	longer, even, not palindrome
'racecar'	True	longer, odd, palindrome
'banana'	False	longer, odd, not palindrome

General Tips

When choosing test cases, consider the following factors:

- **Size**
  For collections (`strings, lists, tuples, dictionaries`) test with:
  - empty collection
  - a collection with 1 item
  - smallest interesting case
  - collection with several items
- **Dichotomies**
  Consider your situation:

  For example:
  - vowels/non-vowels
  - even/odd
  - positive/negative
  - empty/full
  - etc.
- **Boundaries**
  If a function behaves differently for a value near a particular threshold (i.e. an `if` statement checking when a value is 3; 3 is a threshold), test at that threshold.
- **Order**
  If a function behaves differently when the values are in a different order, identify and test each of those orders.

There is often overlap between the categories, so one test case may fall into more than 1 category.

Jennifer Campbell • Paul Gries
University of Toronto