Activity 2.2 – Testing

Here's my solution to exercise 5.3: Write a function named `is_triangle` that takes three integers as arguments, and that prints either "Yes" or "No," depending on whether you can or cannot form a triangle from sticks with the given lengths.

```
def is_triangle(a, b, c):
    """Print "Yes" or "No", depending on whether the side lengths can form a triangle."""
    # If each of the three sides is less or equal to the sum of the other two,
    # they can form a triangle.
    if a > b + c:
        print "No"
    else:
        print "Yes"
```

Notes:

- A string that appears by itself at the top of a function body is special. It's a **documentation string**, or **doc string** (or **docstring**). A doc string is supposed to tell the *human reader* what the function does or how to use it.
- A string literal that begins and ends with three quotes `"""like this"""`, is similar to a string literal that begins and ends with one quote `"like this"`. A triple-quoted string differs in that (1) it can include `"` without that ending it the string; and (2) it can span more than one line. Both of these features are useful for doc strings.
- The text after a `#` is a **comment**. A comment is supposed to tell the human reader how the code works. Python ignores everything after a `#`. (In the case of `is_triangle`, this is the whole line.)
- Pretend, for now, that `print "No"` means the same thing as `print("no")`. We'll re-visit this below.

Q1. Testing

Terminology:

- The **specification** is what the definition of what the code is supposed to do.
- The **implementation** is the program (or code, or software) itself.

Does the implementation of `is_triangle`, above, match its specification?

Here's a tool for (more) systematically exploring this. The first line in the following table indicates that running the Python code `is_triangle(3, 4, 5)` is *expected* to print `"Yes"` (the "expected" column), and is *observed* to print `"Yes"` (the "actual" column.)

a	b	c	expected	actual
3	4	5	`"Yes"`	`"Yes"`

Without using your computer:

- Add another row where the "expected" and "actual" are both `"Yes"`.
- Add a row where the "expected" and "actual" are both `"No"`.
- Add a row where the "expected" and "actual" entries *differ*.

Q2. Running your tests

1. Create a new Jupyter Notebook.
2. Enter the definition of `is_triangle` into a code cell, and run it.
3. You should get a syntax error:

- If you *do not* get a syntax error, this means that you are running Python 2 instead of Python 3, and will run into issues later in the semester.
- If you *do* get a syntax error, change `print "No"` to `print("No")`, and similarly modify `print "Yes"`.

4. Test whether your predictions about the "actual" values are correct.

Q3. Whitebox Testing

```
def is_triangle(a, b, c):    # line 1
    if a > b + c:            # line 2
        print("No")          # line 3
    elif b > a + c:          # line 4
        print("No")          # line 5
    elif c > a + b:          # line 6
        print("No")          # line 7
    else:                    # line 8
        print("Yes")         # line 9
```

`is_triangle(2, 6, 3)` executes lines ##1, 2, 4, and 5. (You can use Python Tutor to verify this.)

This is shown in the table below.

Add additional rows such that every line number is mentioned somewhere in the table. For example, there are no values of (a, b, c) that will cause both lines ##3 and 5 to run, but you can find values that cause line #3 to run, and separate values that cause line #4 to run.

a	b	c	expected	actual	lines
2	6	3	`"Yes"`	`"Yes"`	1, 2, 4, 5

Going Beyond

This section uses material that has not been covered in class. You are *not* expected to be able to work this problems at this point in the semester if this is your first exposure to Python. We will cover these topics later in the semester.

GB1. Fruitful functions

Modify `is_triangle` to *return* `"Yes"` and `"No"`, instead of *printing* these values.

```
def is_triangle(a, b, c):
    """Return "Yes" or "No", depending on whether the side lengths can form a triangle."""
    ...
```

GB2. Boolean values versus strings

Modify `is_triangle` to return boolean values `True` and `False` instead of string values `"Yes"` and `"No"`.

```
def is_triangle(a, b, c):
    """Return True or False, depending on whether the side lengths can form a triangle."""
    ...
```

GB2. `is_not_triangle`

Write `is_not_triangle`, that returns `True` iff (if and only if) the sides can *not* form a triangle. Write this in two different ways:

1. Copy the source of `is_triangle`, and modify it.
2. Write a function that calls `is_triangle`.

GB3. `is_right_triangle`

Write a function that returns `True` iff the sides can form a right triangle: $a^2 + b^2 = c^2$, $a^2 + c^2 = b^2$, or $b^2 + c^2 = a^2$.

GB4. `is_right_triangle` bonus

Is there a way to implement `is_right_triangle` without using an `if` statement?

This is more a math problem than a programming problem, but the solution can be expressed as code.

GB5. doctest

On Monday we will introduce doctest. You can get started now. Turn your answer to Q1 into docstring examples, and run doctest on them.

GB5+. unit testing

This semester we will (probably) not go over `unittest`. Unit testing is an import part of software engineering (which this course intersects with, but is not about) – our use of doctest is an introduction to that. Read about unittest, and turn your answer to Q1 into a separate set of unit tests.