The *triangle program* takes as input three integers and returns the type of triangle that could be formed with sides of the given lengths. Possible outputs are *Equilateral*, *Isosceles*, *Scalene*, and *NotATriangle*.

Write a set of tests for the triangle program.
Software testing is a form of *dynamic verification* – it involves executing the actual program we built.

Testing as a type of dynamic verification implies the adage:

*You can’t test quality into a product*
Our Main Motivation: $$$

We want to:

- maximize the number of faults detected while minimizing the number of tests we write
- create test cases that provide new information
- test an appropriate amount for our particular problem
Fundamentally, a test case is a tuple: \((input, expected\ output)\).

We will commonly capture more information than just that:

<table>
<thead>
<tr>
<th>ID</th>
<th>Purpose</th>
<th>Setup</th>
<th>Inputs</th>
<th>Expected Outputs</th>
<th>History</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Testing Dichotomy

How do we create a test case?

- Tests created using only the specification are considered *black-box* (sometimes called *functional*) tests.
- Tests created by also considering the source code are considered *white-box* tests.

Which is better?
The first black-box testing technique we’ll discuss is *equivalence partitioning*. The idea with EP is to partition the input such that every input in a partition is processed “equivalently” by our system.

Consider the hypothetical/theoretical example discussed in class.