**Debugging Strategy Recap**

We went over 5 debugging strategies in this lesson.

1. **Examine error messages when programs crash**

The last line of Python Tracebacks will tell you what went wrong. Reading backwards from there will tell you more about where the problem occurred.

1. **Work from example code**

If your modified code doesn't work, comment it out and do step-by-step modifications to the example code until it does what you want.

1. **Make sure examples work**

Just because you find example code doesn't mean it will work in your system. Check the example code you're using to make sure it behaves the way you expect.

1. **Check (print) intermediate results**

When your code *doesn't* crash, but doesn't behave as expected, add print statements to your program to see where in the code things stop behaving correctly.

1. **Keep and compare old versions**

When you have a working version of your code, save it before you add to the code. This will give you something to go back to if you introduce too many new bugs.

# Commenting well to aid debugging

You may have already seen comments in some of the code you've worked with throughout this program. Comments are lines of code that are ignored by the computer. They exist so that you can make notes (to other programmers or to yourself) about the code you write. Adding comments to your code can help you (and others) debug your code when things go wrong by comparing what the comment claims the code should do and what it actually does.

In Python, you can make a comment by typing #. Everything written after the # on that line will be the comment. Other languages use different characters to denote comments, but **every** language has some way of making them.

Programmers have many guidelines about how to write comments so that they are descriptive but unobtrusive. Throughout this Nanodegree program, and your entire programming career, you will be expected to follow that principle. Here are some tips to help you comment your code meaningfully:

# 1) Don't comment "obvious code"

The meaning of "obvious code" may change for you as you grow as a programmer, but generally if the code is self-explanatory it doesn't need a comment. The following is superfluous:

print "Hello" *# prints hello*

Simply writing print "Hello" is enough.

# 2) Start functions with a comment

All functions should start with a comment describing the expected input(s) and output(s), and explaining what the function does. This helps others determine what the intended purpose of your function is. Here's an example:

**def** **isLeapYear**(year):

*# takes a number as input and outputs True if the number*

*# represents a leap year and False otherwise*

In Python (but not all languages), you can also comment your functions with a docstring. A docstring is a multi-line string that acts as a descriptive comment for a function, but it is retained by the computer as the code executes and can be accessed by users as your code runs.

**def** **isLeapYear**(year):

'''takes a number as input and outputs True if the number

represents a leap year and False otherwise'''

You can choose to use either comments **or** docstrings in IPND, but you should be consistent with your choice. Classes (which you'll learn about in Stage 3) should also follow this rule.

# 3) Keep commments up-to-date

Your code can become very confusing if you make a change without updating the comment(s) as well. Make sure that they stay aligned to avoid confusing others that may be reading your code (or yourself if you're returning to code after a long break). After all, a major purpose of comments is to help others evaluate what your code is supposed to do when things go wrong. That is much harder to do if your comments give incorrect information!

# 4) Be concise

Comments should be short and explain only the most important details of your code. If you find yourself having to write very long comments to clarify confusing parts of your code, you may want to rethink your approach to the problem. Generally, well written code will have sparse comments. Poorly written code may depend on them!