Assignment 1

Math 480, Spring 2010

Due: Wednesday, April 7, 2010

Do any four of the following seven problems. (I'll try to make a note at the beginning of any problem that requires some background, such as previous programming experience.)

Problems related to `collatz.py`

1. (This problem requires no previous programming experience.) Find at least two things that are wrong with `collatz.py`. For each of these, give an example of input that leads to the bad behavior. If you're feeling adventurous, try to fix the problems.

2. (This problem probably requires some programming experience.) Imagine that you were using `collatz.py` to experiment with the Collatz conjecture. Think of at least two things that you'd like the program to do that it doesn't already, and implement them.

3. Again, let's say you're going to be trying to understand the Collatz conjecture a little better. Describe, either in words or in words and in code, how you'd try to use a computer to help.

Miscellaneous problems

4. We're going to see just how important indentation is in Python. Create two functions `f` and `g`, let's say both of which take a single integer argument `n`, that have the following properties:

   - `f` and `g` are both valid Python functions,
   - `f` and `g` are identical, except for the indentation of some number of lines in their body, and
   - for some input value `a`, `f(a) != g(a)`.

5. Write a function `cashier` that makes change for you. That is, you should take as input a number between 1 and 100, and return a number of quarters, dimes, nickels, and pennies whose total value is equal to that number. You're welcome to decide on your own output format – however, you **must** describe the output format in the docstring for the function.

6. (This problem is intended for people who are new to programming.) We went through our first program in class, fairly hurriedly. Pick something you didn't understand, and try to find out more about it by reading the Python documentation at `http://docs.python.org`. If you succeed, explain what you were confused about, and what you found in the docs that answered your question. If not, explain what you were *trying* to figure out, and where you looked (and didn't find it).

7. (This problem requires you to have experience with some programming language *other* than Python.) First, get in the mindset of your favorite programming language other than Python. Now recall the "Zen of Python:"

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Which lines jump out at you? Pick a few lines that you think are either remarkably similar or, better yet, remarkably **different** from the "philosophy" of your favorite programming language. Back them up with some explanation and examples (which can include URLs, if that feels appropriate).