Style Guide for ENGI E1006

Note: This style guide borrows heavily from the PEP-8 guidelines, but focuses on aspects of the guidelines that are most relevant to the course. Students are *encouraged* to view the original PEP-8 guidelines at https://www.python.org/dev/peps/pep-0008/, and familarize themselves with it.

Why Style?

Style is often considered secondary to functional code. However, as you will realize, while debugging code, programmers often spend more time reading code than writing it. Thus, having well written code that follows specific style conventions simplifies the task of reading one's own code, as well as code written by teammates or coworkers. Let's take a look at two pieces of code to see how style improves the readability of code:

Sample 1

```
def euclid(a, b):

    if a < b:
        t = a
        a = b
        b = t
    while b != 0:
        r = a % b
        a = b
        b = r
    return a

def main():
    print "(i,j) -> GCD"
    for i in range(1, 10):
        for j in range(i, 10):
            print "(" + str(i) + "," + str(j) + ") ->", euclid(i,j)

main()
```

Sample 2

```
##########################
# Name : Amar Dhingra
# UNI  : asd2157
#
# File contains function that computes the GCD of two numbers and tester
##########################

def euclid(a, b):
    ''' Function that computes the greatest common divisor of two numbers using
    the Euclidean algorithm'''
    # determine which of the two numbers is larger
    if a > b:
```

```
        larger = a
        smaller = b
    else:
        larger = b
        smaller = a

    # running euclids algorithm to determine the gcd
    while not smaller == 0:
        remainder = larger % smaller
        larger = smaller
        smaller = remainder

    # returning the gcd
    return larger


def main():
    ''' Tester for euclidean algorithm'''

    # printing out the gcd for all pairs of numbers from 1 to 9
    print "(i,j) -> GCD"
    for i in range(1, 10):
        for j in range(i, 10):
            print "(" + str(i) + "," + str(j) + ") ->", euclid(i,j)

main()
```

Both of these pieces calculate the greatest common divisor of two numbers using the Euclidean algorithm (http://en.wikipedia.org/wiki/Euclidean_algorithm) and then prints the divisor out.

However, Sample 2 is clear to read and easy to follow in such a manner that even the most inexperienced programmers can follow and understand what the program is doing. Sample 1 is probably unreadable to most people, except for the author, and it would even take experienced programmers a few minutes to understand it.

Below, we will examine what coding practices and conventions make the code from *Sample 1* easier to read, as well as the style guidelines you will be expected to follow for this class.

1. Header

A header is the first thing someone sees when they open a Python file (disregarding the shebang line). As such, the header should provide the user with information about the author of the file, as well as a short overview of what the functions are in the file.

If the whole file is one program, the header should have a summary of what the program does. If the file holds helper functions for some other external program (not contained in the file), it should state what program the helper functions belong to and an overview of what the functions do.

For This Class

Your header should contain at least three things:

1. Your Name
2. Your UNI, and
3. 1 - 2 line overview describing what the functions in the file do

2. Import Statements

Programmers (both amateur and professional) very rarely work on projects that do not require the use of built-in libraries. Python allows a programmer to use functions from external files by importing the file using an `import` statement. Functions from an external file can be used anywhere within the scope of an import statement, so it is often convenient to call an import statement within a funciton. **HOWEVER**, this should be avoided as much as possible. It is poor style to import external modules in a function; it is good style to have all import statements at the top of the file, directly under the header. This allows the imports to be global and affect the whole program, rather than one specific function.

For This Class

You will be expected to have all your `import` statements declared globally; that is, at the top of the file, directly under the header.

3. Whitespace

Adding whitespace to code is the easiest way to make unreadable code readable. There are three areas in which whitespace should be used to maximize readability:

1. single line operations
2. within functions, and
3. between functions

Single Line Operations

Single line operations (such as varaible assignments or algebraic operations) can be made more readable by adding appropriate amounts of whitespace. By convention, binary operators such as `=`, `+`, `-`, `*`, `/`, and `%` should have a single space around one side. Unary operators such as `-` should only have one space, in order to distinguish them from other types.

Example

```
test=(a*2+-4)/6
```

can be more easily read as

```
test = (a * 2 + -4) / 6
```

Within Functions

Within functions, whitespace should be used to break code into logical blocks; thus, making it easier to follow what the function is doing. This sectionalization has the added benefit of making debuggign easier, as the astute programmer can now locate bugs in one logical block of a (sometimes) larger function.

In Sample 2, we broke the Euclidean algorithm into three sections: Determine which number is larger, apply the algorithm, and return the appropriate value.

Between Functions

As per Sample 2, whitespace should be used to clearly demarcate functions by adding two lines between each function.

For This Class

You are expected to include space inbetween assignment and algebraic operations within single lines of code. Similarly, you should include one line of whitespace between logical blocks of code in functions, and two whitespaces between different functions.

4. Docstring Comments

Docstring comments are single or multi-line comments added after the function definition. These comments are meant to concisely describe to the user what the function does. These comments are written using Python's multi-line comments (triple quotation marks, such as `''' blah blah blah'''` or `""" Sample text"""`)

For This Class

You are expected to include docstring comments for **EVERY** function you write. This includes both functions given to you in the assignment, along with helper functions you write on your own. Please refer back to Sample 2 for an example of the sort of description the TA's will be looking for.

5. Line Length

Most IDE's (Integrated Devlopment Environment) by default open with windows that are 79 characters wide. To prevent users from having to scroll to the right while reading code, a programmer should limit each line to 79 characters or less. If a line is over 79 characters, it can often be broken into multiple short lines that provide the same functionality.

If a line HAS to be over 79 characters long, it can be extended onto multiple lines by adding a "\" character as the last character or each line.

For This Class

Your code will be expected to be less than 79 characters long. You can test the length of your code by opening a text editor and resizing your window to 80 characters. If the text wraps around the window, your line is too long.

6. Comments

Comments are added to explain the functionality of a single line or a block of code. As such, comments should be used to explain logical blocks of code, so that the users can understand said block by just reading the comment. **BE CAREFUL**, a common beginner mistake is to overcomment code (i.e. explain every line of code), this actually makes code *more difficult* to understand, as a programmer may change code and forget to change her comments. Thus, use comments to help explain code that seems tricky to understand.

For This Class

You are expected to put comments in every program you write. This does not mean you should have more comments than acutal code. Use comments to explain any logic you believe will be difficult to explain through the code itself. Also, a la Sample 2, place comments above blocks to explain what the blocks are doing.

7. Indentation

Python uses indentation to recognize different scopes as well as to identify code in loops or conditional statements. Python recognizes either tabs or any multiple of 4 spaces as a single indentation block. The entire file must use a uniform identation size, otherwise an indentation error will occur. Similarly, a programmer cannot mix tabs and spaces within the same program as that will create an indentation error as well!

By convention, most programmers use 4 spaces for a single indentation, as different text editors read the tab symbol as different lengths (e.g. 4 spaces to 8 spaces). 4 spaces allows code to maintain readability and proper indentation, while keeping lines short.

For This Class

You are expected to maintain proper indentation throughout your programs. That means you must have 4 spaces per indentation (either spaces or tab), and no identation errors.

8. Variable And Function Naming

Variables should be named to make it clear to readers what the function of the variable is. For example, in Sample 2, naming the numbers `larger`, `smaller`, and `remainder` makes it easy to identify what each variable does. Similarly, naming the function `euclid` makes it clear that the funciton runs the Euclidean algorithm on the two number provided. Having clear variable and function names allows the programmer the luxury of not having to use comments to document her code, making it much more approachable for other programmers.

For This Class

You are expected to use clear variable names on **EVERY** function and variable other than counters or indexes. Please refer to Sample 2 for examples.

9. Global Variables

Global variables are variable that can be accessed and changed by any function in the file. Global variables are usually defined at the top of the file and are used to store constants or numbers that have to be frequently accessed but never changed. Writing global variables is expensive to the system running the program and should be avoided as often as possible.

For This Class

Unless otherwise stated, you are expected to not use global variables in your programs.

10. File Structure

The structure of a file can make a big different on how easy it is to understand the code within a file. The structure of a file is a combination of the other sytle guidelines seen so far.

Example

```
##########
# Header
##########

import statements

def function_1(a, b):
    ''' docstring comments'''

    # Comment over code block 1
    ... Code block 1 ...

    # Comment over code block 2
    ... Code block 2 ...

def function_2():
    """ docstring comments"""
    ... Code here ...
```

For This Class

You will be expected to maintain file structure similar to the example given above.