Coding Standard*

A coding standard provides a map so that the code generated by a group of programmers will be consistent and, therefore, easier to read and maintain.

1. Place a doc string at the beginning of each file. The doc string should state the purpose of the file's contents, the date the file was baselined, the date of any modifications subsequent to being baselined, and the name of the author. Each line of the doc string should be indented by four spaces. It can be delimited by either three double quotes or three single quotes. [Rationale: This provides identifying information for the file.]

   Example

   ```
   '''
       Created on Aug 27, 2012

       @author:    David Umphress
   '''
   ```

   Use of docstrings elsewhere is optional. If used, they may be free form; i.e., this coding standard specifies no format.

2. Use # for comments other than doc strings. [Rationale: This style of commenting makes comments consistent across the languages used in class.]

   Example

   ```
   # Return the answer to the universe
   return 42
   ```

3. Indent code four spaces per block. All indenting should be done with spaces, not tabs. Indentation should indicate nesting level. Do not include comments such as "#end if" to denote end of structural constructs. [Rationale: 1) Tabs can cause spacing problems when source code is opened in an editor other than the one in which the code was created. 2) Indentation gives a visual cue to nesting; comments that indicate end of blocks of code decrease readability.]

4. Format code using Eclipse's Source -> Format feature. [Rationale: code that is consistently formatted is easier to read.]

5. Avoid multiple assignment statements. Use only in situations where readability would be significantly affected by using single assignment statements. [Rationale: Multiple logical lines per physical line result in an inaccurate LOC count.]

Not OK	OK
myVar1, myVar2 = 0, 1	myVar1 = 0 myVar2 = 1
import os, sys	import os import sys

* This document is a distillation and synthesis of Python PEP 8 (http://www.python.org/dev/peps/pep-0008)

6. Logical lines can span multiple physical lines. In such cases, the continuation lines must be indented one indentation beyond the initial line. Break lines after operators or commas. [Rationale: Indentation improves readability.]

Not OK	OK
myVar1 = myVar2 + myVar3 + \ myVar4	myVar1 = myVar2 + myVar3 + \ myVar4;

Exception: Literal strings that are part of an executable line of code must not span more than one physical line. Strings that are too long to fit a physical line must be broken into smaller strings that are concatenated. The concatenation may span multiple physical lines. [Rationale: multiple-line strings cause confusion when counting lines of code.]

7. Use blank lines to break up related chunks of code. Use at least one blank line between all methods. [Rationale: Blank lines visually segment blocks of related code, thus making it more understandable.]

8. Code should be as self-documenting as possible. Identifiers should be meaningful, with the exception of loop indices, which may be "i", "j", etc. Magic numbers – meaning, numbers that appear without explanation in source code -- are to be avoided. If they appear, they should be assigned to identifiers that describe their purpose. [Rationale: These practices improve comprehension of code.]

9. Variables should follow a camel-case convention (i.e., the identifier consists of a series of words with the spaces removed; the beginning letter should be lower case; the beginning letter of each word should be uppercase.)

   Module and class names should be nouns in camel-case, with the first letter in lower case.

   Class names should be nouns in camel-case, with the first letter capitalized.

Example
class MyClass

   Function names should be in camel-case, with the first letter in lower case. The name should connote action and thus should be a verb or verb phrase.

Example
run()
showValue()

   Methods that retrieve a Boolean object state must use the prefix *is* in the method name.

Example
isEnabled()

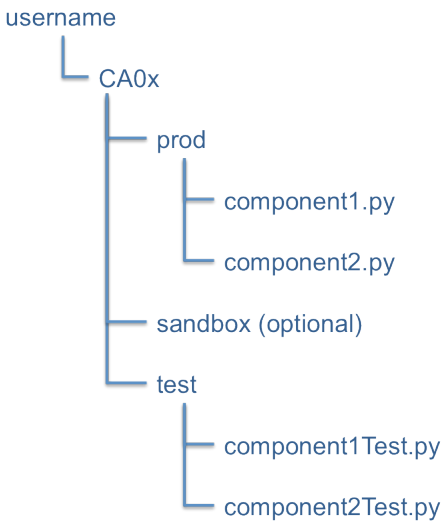
   Identifiers designated to be constant should be nouns in all upper case with words separated by an underscore. All other identifiers should be nouns in camel-case,

with the first letter in lower case.

10. Do not use multiple inheritance. [Rationale: This practice decreases understandability.]

11. Code should be developed in Eclipse using the directory/file structure below.

Eclipse file structure



Note that we differentiating "production" code from "test" code. The former fulfulls requirements and the latter verifies that the requirments are fulfilled. "Sandbox" code is code used to experiment with concepts or programming language constructs. It is not to be used to develop production code.

Code should be delivered as an archived Eclipse project. To export from Eclipse, highlight your username, click File -> Export, click General, click Archive File, click Next, provide a file named: username.zip. Ensure that the all assignments are included and that all assignments include production, sandbox, and test code.

11. Unless specified otherwise, each class should be stored in its own file. The name of the file should be the name of the class followed by a ".py" extension. The file name should be in lower case regardless of the name of the class. For example, class "MyComponent" should be located in a file named "mycomponent.py".

Test code should be placed in a file whose file name clearly reflects what component is being tested. In most cases, the file name will be the component name appended with "Test.py". For example, "mycomponentTest.py" would contain the code that tests "mycomponent.py".

Counting Standard

Each non-blank, non-comment, non-doc string line counts as one line of code.

Example: The LOC count for the code below is 15.

'''	doc string … not a line of code
Created to demonstrate the counting standard	doc string … not a line of code
Baselined: 1 Aug 2009	doc string … not a line of code
Modified: 29 Sep 2009	doc string … not a line of code
@author: D. Umphress	doc string … not a line of code
'''	doc string … not a line of code
	blank line … not a line of code
def fib(n):	line of code
''' fib(n) calculates the	doc string … not a line of code
Fibbanacci value of	doc string … not a line of code
	doc string … not a line of code
n '''	doc string … not a line of code
prev = 0	line of code
next = 1	line of code
	blank line … not a line of code
for i in range(n - 1):	line of code
temp = next	line of code
next = next + prev	line of code
prev = temp	line of code
return next	line of code
	blank line … not a line of code
# The code below is a test driver for fib()	comment … not a line of code
if __name__ == "__main__":	line of code
print '"Enter a positive integer:"'	line of code
inputValue = input()	line of code
fibNum = fib(inputValue)	line of code
print "The Fibbanacci " +	line of code
"number is " + str(fibNum)	line of code
	total lines of code: 14