

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, 2015
```

```
    @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

```
myVar1, myVar2 = 0, 1
```

```
import os, sys
```

OK

```
myVar1 = 0
myVar2 = 1
```

```
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.]

Do not span strings over more than one physical line. In cases where a string won't fit on a physical line, break the string into individual parts that will fit on a line and concatenate the parts. [Rationale: This simplifies construction of tools that count lines of code.]

Not OK	OK
<pre>myVar1 = myVar2 + myVar3 + \ myVar4</pre>	<pre>myVar1 = myVar2 + myVar3 + \ myVar4;</pre>
<pre>myString = "multi-line string"</pre>	<pre>myString='multi-line ' + \ 'string'</pre>

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. For example:
- Identifiers should be given meaningful names that indicate purpose or use.
 - Single letters should only be used as names for loop indices when mirroring a mathematical equation specified in the requirements, otherwise, loop indices should be given names that indicate their use.
 - 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.
 - Comments should be used sparingly as an aid to understanding non-intuitive code structure, complex calculations, unique circumstances, etc. It should not be a substitute for unreadable code.

[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()

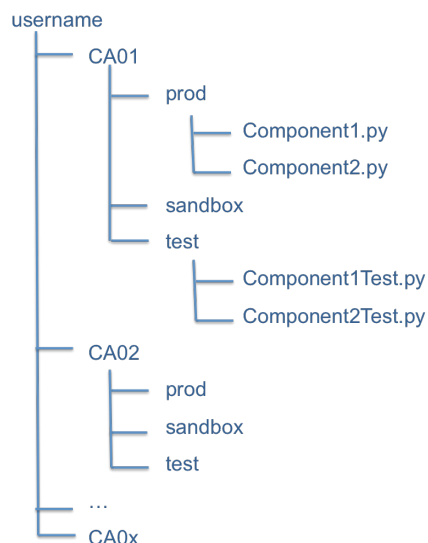
Identifiers designated to be constant should be nouns in all upper case with words separated by an underscore. All other identifiers should be nouns in camel-case, with the first letter in lower case.

10. Do not use multiple inheritance. [Rationale: This practice decreases understandability.]
11. Code should be developed in Eclipse using the directory/file structure below.

Note that we are differentiating “production” code from “test” code. The former fulfills requirements and the latter verifies that the requirements 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.

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. For example, class “MyComponent” should be located in a file named “MyComponent.py”.

Eclipse file structure



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.

```
"""
    Created to demonstrate the counting standard
    Baselined: 1 Aug 2009
    Modified: 29 Sep 2009
    @author: D. Umphress
"""
def fib(n):
    """ fib(n) calculates the
    Fibbanacci value of

        n """
    prev = 0
    next = 1

    for i in range(n - 1):
        temp = next
        next = next + prev
        prev = temp
    return next

# The code below is a test driver for fib()
if __name__ == "__main__":
    print "Enter a positive integer:"
    inputValue = input()
    fibNum = fib(inputValue)
    print "The Fibbanacci " + \
        " " + \
        "number is " + str(fibNum)
```

```
doc string ... not a line of code
doc string ... not a line of code
doc string ... not a line of code
doc string ... not a line of code
doc string ... not a line of code
doc string ... not a line of code
blank line ... not a line of code
line of code
doc string ... not a line of code
doc string ... not a line of code
doc string ... not a line of code
doc string ... not a line of code
line of code
line of code
blank line ... not a line of code
line of code
line of code
line of code
line of code
line of code
line of code
blank line ... not a line of code
comment ... not a line of code
line of code
line of code
line of code
line of code
line of code
line of code
line of code
line of code
```

total lines of code: 15

total number of components: 1

```
name: fib
methodCount = 1
locCount= 15
```