CMSC 201 Fall 2015

CMSC 201 - Python Coding Standards

Every programming department has a set of standards or conventions that programmers are expected to follow. The purpose of these standards is make programs readable and maintainable. After all, you may be the programmer who maintains your own code more than 6 months after having written the original. Programming standards vary by department, therefore, it is important that all members of the department follow the same standards.

Neatness counts!

At UMBC, the following standards have been created and are followed in CMSC 201.

Part of every project and homework grade is how well these standards are followed.

It is your responsibility to understand these standards. If you have any questions, ask any of the TAs or the instructors.

Naming Conventions

- Use meaningful variable names!!
  - For example, if your program needs a variable to represent the radius of a circle, call it 'radius', NOT 'r' and NOT 'rad'. The use of single letter variables is forbidden except in loops.
  - The use of obvious, common, meaningful abbreviations is permitted. For example, 'number' can be abbreviated as 'num' as in 'numStudents'.
- Begin variable and function names with lowercase letters
- Names of constants should be in all caps, i.e., EURO_TO_USD = 1.41374
- Do not use global variables
- Separate "words" within identifiers with underscores or mixed upper and lowercase. Example: grandTotal or grand_total
  - Be consistent! If you choose to use mixed case, always use mixed case.

Use of Whitespace

The prudent use of whitespace (blank lines as well as spaces) goes a long way to making your program readable.

- Use blank lines to separate major parts of a function.
- Indentation should be 4 spaces long. Hitting Tab in emacs will accomplish this.
- Use spaces around all operators. For example, write `x = y + 5`, NOT `x=y+5`

break & continue

Using `break` and `continue` is not allowed in any of your code for this class. Using these statements damages the readability of your code. Readability is a quality necessary for easy code maintenance.

Use of Constants

To improve readability, you should use constants whenever you are dealing with values. Your code shouldn't have any "magic numbers", numbers whose meaning is unknown. Here's an example:

```
total = subtotal + subtotal * .06
```

.06 is a magic number. What is it? We don't know. So, at the very least this line of code would require a comment. However, if we use a constant, the number's meaning becomes obvious, the code more readable, and no comment is required. Constants are typically placed near the top of the program so that if their value ever changes they are easy to locate to modify and so that they stay in scope throughout the program.

```
TAX_RATE = .06

... (lots of code goes here)
total = subtotal + subtotal * TAX_RATE
... (other code goes here)
print "Maryland has a sales tax rate of %.2f percent" % (100 * TAX_RATE)
```

Comments

Programmers rely on comments to help document the project and parts of the project.

Generally, we break out comments into three types:

1. File Header Comments,
2. Function Header Comments, and
3. Code Comments (in-line).

Comments for files, functions and code are described below.

1. File Header Comments

Every file should contain an opening comment describing the contents of the file and other pertinent information. This "file header comment" **MUST** include the following information.

- The file name
- Your name
- The date the file was created
- Your section number
- Your UMBC e-mail address
- A description of the contents of the file

For example:

```
# File:    hw3.py
# Author:  Sue Evans
# Date:    9/22/09
# Section: 4
# E-mail:  bogar@cs.umbc.edu
# Description:
# This file contains python code that will decode an English
# message that was encoded using a Caesar cypher with a fixed
# rotation length.  It makes use of letter frequencies in
# the English language to determine the rotation length.
```

2. Function Header Comments

EACH FUNCTION must have a header comment that includes the following:

- function name
- a description of what the function does
- a description of the function's inputs
- a description of the function's outputs
- side effect of the function (if any -- this should be rare)

For example:

```
# findCircleArea() calculates the area of a circle from the radius
# Input:  the circle's radius
# Output: the circle's area
```

3. In-Line Comments

- "In-line" comments are used to clarify **what** your code does, NOT **how** it does it.
- Well-structured code will be broken into logical sections that perform a simple task. Each of these sections of code (typically starting with an 'if' statement, or a loop) should be documented.

- Any confusing-looking code should also be commented.
- **Do not comment every line of code.** Trivial comments (# increment x) are worse than no comments at all.

An in-line comment appears above the code to which it applies and is indented to the same level as the code.
For example:

```
# check every member of a list
for num in numList :

    # if it's odd, print it, if even, do nothing
    if num % 2 == 1 :
        print num
```