CMSC 201 Spring 2016
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 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 obvious, common, meaningful abbreviations is permitted. For example, 'number' can be abbreviated as **`num`** as in **`numStudents`**.
  - The use of single letter variables is *forbidden* except in loops.
- Begin variable and function names with lowercase letters.
- Names of constants should be in all caps with underscores between words.
  - *e.g.*, **`EURO_TO_USD = 1.13`** or **`MAX_NUM_STUDENTS = 100`**
- Separate "words" within identifiers with underscores or mixed upper and lowercase.
  - *e.g.*, **`grandTotal`** or **`grand_total`**
  - Be consistent! If you choose to use mixed case (also known as "camel case"), always use mixed case.
  - Do *not* use global variables!

Use of Whitespace

The prudent use of whitespace goes a long way to making your program readable. Horizontal whitespace (spaces between characters) will make it easier to read your code. Vertical whitespace (blank lines between lines of code) will help you to organize it.

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

Use of Constants

To improve readability, you should use constants whenever you are dealing with hard-coded values. Your code shouldn't have any "magic numbers," numbers whose meaning is unknown.

For example:

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

In the code above, **`.06`** is a magic number. What is it? The number itself tells us nothing; at the very least, this code would require a comment. However, if we use a constant, the number's meaning becomes obvious, the code becomes 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. Constants may be placed outside of the main() function – this makes them global constants, which means everything in the file has access to them. (Global variables are only allowed for constants!)

Here's the updated code:

```
TAX_RATE = .06   # TAX_RATE is a global constant

def main():
    # lots of code goes here
    total = subtotal + subtotal * TAX_RATE
    # other code goes here
    print("Maryland has a sales tax rate of", TAX_RATE, "percent")

main()
```

Comments

Programmers rely on comments to help document the project and parts of the project. Generally, we categorize comments as one of three types:

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

1. File Header Comments

Every file should contain a comment at the top 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 brief description of the contents of the file

For example:

```
# File:    proj1.py
# Author:  Katherine Gibson
# Date:    10/14/2015
# Section: 1
# E-mail:  k38@umbc.edu
# Description:
# This file contains python code that will simulate a run of
# Conway's "Game of Life".  It allows the user to choose
# which cells to turn on, and for how many iterations it is run.
```

2. Function Header Comments

Every single 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:

```
# circleArea() 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 comments within the code itself. They are normally comments for the line of code directly below them.

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.
- In-line comments are used to clarify what your code does, not how it does it.

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
```

Built-In Functions

Python has many useful built-in functions that easily let a programmer perform a variety of tasks. Unless specified by the assignment, you are not to use any of the built-in functions we have not covered in class.

Using a built-in function to solve a homework problem for you does not show that you have mastered the concepts behind it, and hence does not fulfill the assignment.

Break and 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.