CS 102: Python Style Guide

This documents provides some standards that will help ensure you are writing good code in this course. Good coding standards allow students to clearly demonstrate their plan for a program, and they help avoid confusion and errors. Each coding assignment in this course will be assessed partly on the items below.

Heading Comment at Top

Each Python file you submit must have a comment at the top that includes your name, the date of the assignment, and the assignment number. Here is an example:

```
# Kowshik Bhowmik

# Assignment 1: Style Guide

# 08/01/2020
```

Descriptive Names

Variables, parameters, and functions must have descriptive names. While naming all your variables after your favorite fast food menu items or your favorite Pokemon is fun, it won't make for an understandable program. A good variable or parameter name describes what the variable stores and its role in the program. A good function name describes what the function does. Variable names tend to be nouns while function names are often verbs.

Variable and function names will use the python standard. Names will be lower case with multiple words separated by an underscore. Variables that will hold values that are not meant to change are named in all upper case (this has no effect per say, but it is a stylistic convention). For example:

```
total = 10

sales_tax = 20

search()
```

```
calculate_amount()

CONSTATNT = 10

MY_CONSTANT = 20
```

Descriptive Comments

Every function you write should have a comment before it briefly describing what the function does, what the parameters are (if any), what side effects it has (if any), and what it outputs (if anything). It can also be useful to place a comment near a section of code you found especially tricky or difficult to understand to document its purpose and why it works.

Remove Debugging Code Before Submission

While working through a problem or trying to debug some code, it is very common to use functions such as `print` to display information. Make sure that any output features in your assignment relate to intended functionality described in the program assignment and not left over testing code.

Remove Commented Out Code

It is not uncommon to try a few different solutions to a solve a problem. Sometimes instead of removing the code right away, developer will temporarily comment out the code so it is ignored while running the program. This way they can easily enable the code again by removing the comment `#` character. However, if there is code that is commented out that you no longer in tend to use, remove this text from your program before submission.

Don’t Assign to Parameters

Do not assign new values to parameters of a function. For example:

```
def my_function(x, y):

  y = y + x
```

This can lead to very subtle bugs when you attempt to later use the variable `y` only to find that you have since changed its value. It may seem redundant, but if you need to change the value of a function parameter, do so in a new variable like this:

```
def my_function(x, y):

    z = y + x
```

Assign Values Just Before They Are Needed

An older tradition of programming was to declare/assign all variables at the top/beginning of a function and use them later in the function. Modern programming practice states that a variable should be created near where it is first used. This keeps readers of a program from constantly navigating between variable declaration sections and then back to other locations of interest.

Use Whitespace Organization for Readability

Deliberate use of whitespace can help organize your code and make it much easier to read. Consider the following example:

```
SALES_TAX = 0.20

pre_tax_total = 200

total = pre_tax_total + (pre_tax_total * SALES_TAX)

NAME = "Drew"

print(NAME + " paid: " + str(total))
```

This code has no meaningful organization. It is a block of text that readers must read in order to comprehend the relations between each line. An alternative could be:

```
SALES_TAX = 0.20

pre_tax_total = 200

total = pre_tax_total + (pre_tax_total * SALES_TAX)
```

```
NAME = "Drew"

print(NAME + " paid: " + str(total))
```

While this is a trivial example, notice how the calculation for sales tax is separated from the printing output by a new line. This visual separation isolates the code for the calculation from the statements that process the output. A simple rule of thumb is if you find yourself straining to see what's happening in your code, try to logically group related statements using whitespace.

Do not waste your time with fancy formatting:

```
##### VARIABLES ######

x           = 1000

y           =  13

global_constant =  9.8

######################
```

This kind of formatting may have some visual appeal, but it lacks maintainability. If formatting would need continual adjusting for the length of variable names or data values, it becomes more of a burden than anything. Keep it simple.

Do Not Use "Magic Numbers"

A magic number general term for any literal value present in your code that has no associated name. Consider the following code:

```
a = 10 * 12
```

In the expression above there is little context for the purpose of the values. Instead assign these values to a variable (or if the value doesn't change a constant) like this:

```
LENTH = 10
```

```
WIDTH = 12

a = LENGTH * WIDTH
```

While on the surface this seems redundant, the advantages are two fold. First, if you have used the magic number in multiple locations in your code, you can change it in one place instead of manually changing all occurances. Second, a name allows you to provide additional context to people reading your code (or for yourself when you come back to your code after some time has passed).

Use Relative File Paths

A file path is some text (a string) that describes the location of a file on your computer. File paths can be either absolute or relative. An absolute file path describes the complete location of the file on your computer. On Windows, an absolute file path begins with a drive letter, a colon, and a backslash (e.g. C:\Drew\My Documents\CS102\media.py). On Mac, an absolute file path begins with a slash (e.g. /Users/Drew/Documents/CS102/media.py). Absolute are problematic when you want someone else to run your code on their computer. If I give you code with the file path /Users/Drew/Documents/CS102/media.py, running the program will most likely result in an error as that exact same path will not exist on your computer.

A relative file path lets you specify the location of a file on your computer relative to some other file. In this course, that "other file" will be the program you are running with Thonny. We will save each assignment in its own folder. Each folder will contain all the code, media content, and the media.py module (when needed). If all file paths in your code are relative to these assets, your program should work on any computer that has access to the complete assignment folder.

NOTE: Backslash is a special character in strings in Python. So, if you are on Windows, file paths you use in Python will actually have to use `\\` instead of `\`.