# Style Guide for DAT-119 Coding Assignments

"**Programs must be written** **for people to read**, and only incidentally for machines to execute." — Abelson & Sussman

1. Write out your plan for the program, listing inputs, outputs, and intermediate steps before you start programming. You're always going to turn this in. It doesn’t have to be super neat or 100% match what you end up coding—though I’d like to see some reflection on why it changed, when that happens—but it does have to show that you put in thought before you started writing code.
2. Begin each file with a block that lists your name, the date you started writing the code, the assignment it's for, and a brief description of what it does. Here is a template:

'''

Firstname Lastname

9/21/2019

Python 1 - DAT-119 - Fall 2019

Homework 3

Takes a list of five test scores from the user, computes the average, and

outputs it nicely.

'''

(You don't have to use my *exact* format, but I expect yours to be equally readable and contain roughly the same information.)

If you start putting your program plan into that code block, instead of a separate file, that is fine. That’s what I do in the homework solutions.

1. Give variables useful names that describe what they hold. Named constants should be in all caps, with underscores between words. For every other variable (this will be most of your variables), use "snake\_case": everything is lowercase, and multi-word variables have underscores between them. This rule will also apply to function names when we start using those.

* **Good constants:** INTEREST\_RATE, PERCENTAGE
* **Good variables:** average\_grade, first\_name, favorite\_animal

1. **Bad variables:** x, a, i, youareapotato, someCamelCaseNonsense
2. **Use comments**. Your planning document can help guide what comments you need to use: was it worth writing down as a step? It's worth commenting. If you handed your code to a classmate, would a comment help them understand what it does? It's worth commenting. When we start writing functions, you’re going to need to have a comment for every function, listing (at minimum) what kind of arguments it expects and what kind of thing it returns.
3. Put spaces around binary and assignment operators. There's no good reason to say something like 3+5/4 when 3 + 5 / 4 is more readable. Also put spaces around your equal signs.

* **Bad:** item=item+1
* **Good:** item = item + 1

1. Save your files as **your\_surname\_homework\_number.py**. If there are multiple assignments, label them with letters. For instance, I might submit Sheldon-Hess\_homework\_1a.py and Sheldon-Hess\_homework\_1b.py (You may want to give them more descriptive names while you're working on them and then just change the filename to submit the homework.)
2. **Think of the user of your program, always**. Do nice things for them, like adding the space in your input strings and always making your outputs readable.
   * **Bad:**input(“type a sentence”)
   * **Good:**   
       
     print(“This program will [some details for the user]”)   
     print(“You should enter a complete sentence and hit ‘enter’ to proceed.”)  
     input(“Your sentence: ”)
3. Don't let your lines get too long. There are hardliners who insist on 80-character lines. I'm not them—my own code sometimes strays closer to 100 characters wide. But if an expression won't fit on a reasonable-size area without scrolling right, go ahead and break it up for readability. Use common sense.
4. For homework assignments and projects (really, anything except informal in-class assignments where I explicitly say "you don't have to"), *all user-supplied inputs need to be validated*, **or** you need to use exceptions. You need to make sure bad user input doesn’t break the program messily.
   * 1. Test your code with unexpected inputs before you turn it in. Think about edge cases.\*
5. There should be two blank lines between the import statements and any function definitions. There should be two blank lines between functions. Single blank lines should be used within functions as needed to increase readability.

\* There’s a joke related to this: “A QA tester walks into a coffee shop. Orders a coffee. Orders 0 coffees. Orders 99999999999 coffees. Orders a lizard. Orders -1 coffees. Orders a ueicbksjdhd.”