General Rubric & Style Guide
DS2000
Spring 2023

We'll use the same general approach for grading all of your homework submissions this semester. The rubric below gives an overall sense of what we're looking for, though the items on gradescope will be tweaked for each assignment.

General Homework Rubric

Style Guide - Readability

Style Guide - Documentation

Keep this rubric and styleguide as you're working on your DS2000 homeworks. They'll help ensure you've got everything covered. We'll also publish grading notes for every homework; review this document and the grading notes every week, before you submit your homework.

The 30-Minute Guideline
If you get stuck on a homework problem, come by office hours or post on Piazza! We recommend you spend about 30 minutes trying to figure out a problem and then ask for help. Enough time that you can try a few things to get unstuck, but not SO much time that you're banging your head against the wall. Try for 30 minutes, then ask us. :)

General Homework Rubric

Before submitting your homework or visiting office hours, look through each item on this general rubric and compare it to your code. Do you have everything that is expected?

Each homework will also have a rubric of its own that is more specific to the requirements - mostly from the correctness category. Take a look at those as well, to make sure your code accomplishes everything required of the assignment. If you're not sure about anything here, ask us on Piazza or in office hours!

Group (Points)	**Component**	**What We're Looking for**
Correctness (9)	Gather Data	Prompting/reading is correct and all instances use appropriate data types
	Computations	Computations are correct as specified in the homework Data is not hard-coded unless specified in the homework Numerical values are not rounded in this step
	Communication	Requested data/results are reported and formatted as specified by the homework Values are rounded or labeled/colored correctly Interaction with user (prompts, outputs, plots) are easy to read and understand
Readability (6)	Spacing	No line goes over 80 columns Spaces around items (e.g., `x = y` not `x=y`, and `# comment` not `#comment`) Related code is grouped together, and unrelated sections are separated with vertical space Strings are enclosed in either double or single quotes, never mixed *print* uses commas to separate arguments, not + or %
	Variables and	`def main` is used to organize code

	Functions	Variable/function names are clear and concise Variable/function names are lowercase only, and underscores are used when needed The variables *i, j* are used for indexes, not for values Constants are named in all caps and initialized above main Functions, loops, and other code blocks are well-organized and easy to follow
Documentation (3)		Header comment includes name, class, HW number Test cases are included, if required Inline comments are above the code they reference Every function has a comment with parameters, return value, and description Approximately 1-2 inline comments per section of code (not needed within short functions) Comments do not repeat code or variable names, and instead provide explanation/clarity Filename is correct, as specified in the homework spec

Style Guide - Readability

The readability section of the homework rubric is tied to this section of the style guide.

We'll write *all* our Python programs using the same structure, with the heart of our code inside a main function. Before you write any other code, type `def main():` at the very top and `main()` at the very bottom. Your program's code will go in between.

```
def main():
      # Your code goes here!
      # Your code goes here!

main()
```

Vertical space
Group related code together, and use vertical space to separate chunks of code. Like this:

```
# here is a comment describing the next three lines of code,
# which are all related to each other
Code line 1
Code line 2
Code line 3

# here is a comment describing the next two lines, which are
# separate from the lines above
Code line 4
Code line 5
```

Horizontal Space
Limit your code to 80 columns or less. In Spyder, it'll show you a gray vertical line at the correct stopping point.

If you need to continue a line onto the next one, indent the following line so the spacing makes sense:

```
spam = long_function_name(var_one, var_two,
                          var_three, var_four)
```

You can also use Python line continuation, by putting a blackslash (\) where you need the line to carry over:

```
a = 1 + 2 + 3 + 4 + \
```

```
5 + 6 + 7
```

Put an extra space between operators and variables.

Do this:
```
x = y + 5
```
Not this:
```
x=y + 5
```
Not this:
```
x = y+5
```

Do this:
```
if x == y:
```
Not this:
```
if x==y:
```

Put an extra space after the inline comment hashmark:

Do this:
```
# comment
```
Not this:
```
#comment
```

Put an extra space after commas, like this
```
print("Hello", first_name)
result = func(18, 19, "hello")
```

Variable and Function Names
Variable and function names must be short and descriptive. Use lowercase letters, and use underscores to separate words. Do not use camel case.

Here are some acceptable variable names:
```
age = 44
birth_year = 1978
first_name = "Laney"
```

And some crummy ones:
```
a = 44
x = 1978
variableName = "Laney"
```

Constants, whose values never change once initialized, should be uppercase:
```
FILENAME = "file.txt"
```

Constants are the only permissible global variables. They may be defined at the very top of your program, below your comments but above all your functions. All other variables you use in DS2000 must be local -- i.e., defined within a function.

Strings
In Python, single-quotes and double-quotes are the same. You can use either one, but be consistent throughout an entire program.

Separate literals and variables using a comma. Don't use the % string-formatting operator or the string concatenator.

Do this:
```
print("Hello", name)
```

Not this:
```
print("Hello %s" %name)
```

Not this:
```
print("Hello" + name)
```

You can tweak the look of your outputs by using optional arguments to print, like this:

Prints `Hello , Laney !`
```
name = "Laney"
print("Hello", name, "!")
```

Prints `Hello, Laney!`
```
name = "Laney"
print("Hello", name, "!", sep = "")
```

Style Guide - Documentation

The readability section of the homework rubric is tied to this section of the style guide.

Every program you ever write needs to have comments. Before you write any code, put a block comment at the top of every program with your name, the course, the assignment, the date, and the name of the file.
```
'''
    Laney Strange
    DS2000
    Homework 1
    Spring 2023
    pets.py
```

```
'''
```

Comments explaining your code should appear throughout your program. You'll get the feel of what's too much or too little with experience. You don't need to comment every single time you introduce a new variable (that's too much!), but you also don't want to leave the reader to figure out all your code for themselves (that's too little!). Typically, you'll put a few comments above a block of code that does something interesting.

Comments go above Python statements, not beside them.

Do this:

```
# comment describing my code
python statement
```

Not this:

```
python statement # comment describing my code
```

Put a space between the crosshatch and the comment itself.

Do this:

```
# comment
```

Not this:

```
#comment
```

Function Comments

Your functions should be generic -- write a function to add two numbers, rather than a function to add 3 + 4. Function names follow the same style as variable names, and the name of a function describes what it does. Try using verbs for function names.

Functions should be concise; keep them under 30 lines of code. Functions should also accept a limited number of parameters; five of them at the absolute max.

You'll also need to comment every function you write. Function comments should include the parameters and return type, and they should describe the **what** of a function as well. Use Python docstrings to comment your functions, like this:

```
def print_message(message):
    """
      Parameters: message to print, a string
      Returns: nothing
      Does: Print supplied string message plus an extra linebreak
    """
    print(message)
    print("\n")
```