Coding Standards and Style

In this specialization, we will expect you to follow our coding style conventions. For graded projects, these conventions will largely be enforced by the use of Pylint. Pylint is a source code bug and quality checker for the Python programming language. The machine-graders for this specialization will use a customized version of Pylint to check the style of your code. Pylint errors that violate of the style guidelines will result in deductions from the score of your mini-project. **An explanation of common Pylint errors can be found in the course resources here**.

Here are some of the style guidelines that we will follow in this specialization. As you interact with Pylint, you will get more exposure to these guidelines.

Documentation

Documentation strings ("docstrings") are an integral part of the Python language. They need to be in the following places:

- At the top of each file describing the purpose of the program or module.
- Below each function definition describing the purpose of the function.

Docstrings describe **what** is being done in a program or function, **not** how it is being done. A docstring for a function should explain the arguments, what the function does, and what the function returns. Here is an example that includes a docstring at the start :

```
""" An example program that illustrates the use of docstrings """

def welcome(location):
  """
  Takes input string welcome and returns a string of the form
  "Welcome to the location"
  """
  return "Welcome to the " + location
```

These docstrings are treated specially in Python, as they allow the system to automatically give you documentation for programs and functions. At the command prompt, you can type \color{red}{\verb|help(...)|}, and it will return the docstring for whatever the argument you passed to help is. (Note that CodeSkulptor does not provide a command prompt, so you cannot use \color{red}{\verb|help|} in CodeSkulptor.)

Comments

Comments should describe **how** a section of code is accomplishing something. You should not comment obvious code. Instead, you should document complex code and/or design decisions. Comments and docstrings are not interchangeable. Comments start with the "#" character. While you will see some Python programmers do this, you should **not** comment your code by putting a multi-line string in the middle of your program. That is not actually a comment, rather it is just a string in the middle of your program!

A good example:

```
# This is a block comment
# that spans multiple lines
# explaining the following code.
val = some complicated expression
```

A bad example:

```
"""
Somebody told me that a multiline
string is a block comment.

It's not.
"""
val = some complicated expression
```

Note that docstrings are multi-line strings, but they do not violate this convention because docstrings and comments are different and serve different purposes in a program.

Global variables

Global variables should never be used in this specialization. Avoiding their use is good programming practice in any language. While programmers will sometimes break this rule, you should not break this rule in this specialization.

There is one exception to this rule: you may have global constants. Because the Python language does not actually support constants, by convention, Python programmers use global variables with names that are in all capital letters for constants. When you see a variable with a name in all capital letters, you should always assume that it is a constant and you should **never** change it. Again, such global constants are the only global variables that will be allowed in this specialization.

Names

All variable and function names **must** be at least 3 characters. The first character of a name should follow these conventions:

- Variable names should always start with a lower case letter. (Except for variables that represent constants, which should use all upper case letters.)
- Function names should always start with a lower case letter.

Further, we will follow the common Python convention that variable and function names should not have any capital letters in them. You can separate words in a name with an underscore character, as follows: \color{red}{\verb|some_variable_name|}. As previously noted, constants should be in all capital letters, such as: \color{red}{\verb|THIS_IS_A_CONSTANT|}.

Indentation

Each indentation level should be indented by 4 spaces. As Python requires indentation to be consistent, it is important not to mix tabs and spaces. You should never use tabs for indentation. Instead, all indentation levels should be 4 spaces. Note that CodeSkulptor automatically converts all tab indents into 4 spaces.

Scope

You should not use names that knowingly duplicate other names in an outer scope. This would make the name in the outer scope impossible to access. In particular, you should never use names that are the same as existing Python built-in functions. For example, if you were to name one of your local variables \color{red}{\verb|max|} inside of a function, you would then not be able to call \color{red}{\verb|max()|} from within that function.

Arguments and local variables

While there is not necessarily a maximum number of arguments a function can take or a maximum number of local variables you can have, too many arguments or variables lead to unnecessarily complex and unreadable programs. Pylint will enforce maximum numbers of arguments and variables, etc. If you run into limits that Pylint complains about, you should restructure your program to break it into smaller pieces. This will result in more readable and maintainable code.

Further, you should not have function arguments or local variables declared that are never used, except in rare circumstances. Sometimes, you do need to have a variable that you never use. A common case is in a loop that just needs to execute a certain number of times:

```
for num in range(42):
    # do something 42 times
```

In this case, you should name the variable with the \color{red}{\verb|dummy_|} prefix. This indicates clearly to you, others, and Pylint that the variable is intentionally unused.

```
for dummy_num in range(42):
    # do something 42 times
```