George Fox University Introductory Python Style Guide

September 30, 2024

1 Guiding Principles

Three universal principles form the basis of good programming style. They must not be violated:

1. DRY: Do not repeat yourself.
    - Meaning: Avoid repeating portions of code.
    - Reason: Copy/paste coding is surprisingly error prone, difficult to test, and hard to maintain.
    - Remedy: Instead, use function definitions and control flow to avoid repetition.
2. POLA: Principle of least astonishment.
    - Meaning: Apply best practices and commonly used idioms for the language you're using.
    - Reason: Useful code will be expanded and maintained into the future. Applying best practices simplifies code, making this process easier, more efficient, and less prone to bugs.
    - Remedy: Minimize confusion by writing clear and simple code. Learn and apply common idioms. This guide is a good starting point.
3. SEEP: Single entry and exit point.
    - Meaning: Program structures that execute logical control flow (i.e. `if` statements, loops) should have single consistent entry and exit points.
    - Reason: Inconsistent branches in flow control can create bugs that are difficult to isolate and also make code harder to read and maintain.
    - Remedy: Avoid controlling flow with nested and/or multiple `returns`. *Never* use `continue`, `pass`, or `break`.

2 General Python Conventions

- Variable, parameter, and function names are lowercase, with the occasional underscore '_'. This is referred to as `snake_case`.
- Constant names are an exception. Use `UPPERCASE` with a '_' to separate 'words.'
- Place spaces after reserved words and around operators, for example:

```
total=num1+num2           # poor
total = num1 + num2       # good
if some_var<some_value:   # poor
if some_var < some_value: # good
```

- `pass`, `continue`, and `break` statements are *not* allowed.
- `while True` and `while False` loops are also *not* allowed. Use a condition or a `boolean` flag instead.
- Do not 'manually' increment loop counters in a `for`-loop.
- Restrict line length to 100 characters. Your IDE can count for you.
- Comment your code–especially control flow structures like `if` statements.
- Place comments above the line you wish to comment, not at the end. Use a space after the '`#`':

```
user_name = input()  # Get input from user.   # poor

# Get input from user.                        # good
user_name = input()
```

- Avoid too many levels of nesting. Use functions to break up code.
- Do not perform unnecessary conversion operations:

```
user_name = str(input())  # poor
user_name = input()       # good
```

- The proper ordering of a program's components is:
  1. Import statements
  2. Constant definitions
  3. `main()` function declaration
  4. Other function declarations
  5. Invocation of `main()`

3 Variables & Constants

- Global variables are allowed *only* if they are defined as `CONSTANTS`.
- Lists and other objects may be declared as global constants, but they must not be mutated or changed.
- Do not use "magic numbers." A magic number is a numeric constant embedded in code, without a constant definition. The numbers -1, 0, and 1 are not normally considered "magic numbers":
- String representations of magic numbers (i.e. `'20'`) are still magic numbers. Do not use them except when declaring a constant.
- Place constants before any function definitions, but after the import statements:

```
# Definition of a constant:
GRAVITATIONAL_CON = 9.81

def main():
    <. . .>
    # Usage of a constant:
    velocity = GRAVITATIONAL_CON * seconds + initial_vel
    <. . .>
```

- Do not define multiple variables on the same line. Instead use separate definitions:

```
dimes = 0; nickles = 0    # poor

dimes = 0                 # good
nickels = 0
```

- Variable names should be clear and verbose. Abbreviations are permitted where the meaning is unambiguous.
- Historically, `i`, `j`, and `k` have been used for loop counters, and are reasonable exceptions to the rule above when used for that purpose.

4 Functions

- The `main()` function idiomatically governs the control of the entire program, and serves as its single entry point. As such, `main` has specific rules:
  - Every program must have a `main`.
  - `main` must be the first function declared at the top of the code.
  - `main` must be the only function called from the global scope. This invocation must be the last line of a program.
  - `main` cannot be called by another function.
  - `main` cannot have parameters (i.e. `main(foo)`).
  - Programs should include at least one function besides `main`.

The following rules apply to all functions:

- Every function definition (including `main`) must have a 'docstring' comment:

```
def foo(bar):
    """
    A description of the function is written in this
    docstring comment.
    """
    <function body>
```

- Functions cannot contain more than 20 lines of code. Break up complex computations into separate functions whenever possible.
- In general, do not make functions recursive, where a function makes a call to itself.
- Do not allow co-recursion in functions:
  `dog()` calls `chase()` which calls `tail()` which calls `dog() . . .`
- Do not create function definitions inside of functions. Rather, declare all functions in the global scope.
- Do not nest `return` statements inside of control structures.
- Do not have multiple `return` statements in a function.
- `quit` and `exit` can cause confusion in debugging and are also not permitted.