Standard Guidelines in Python Program:

Introduction:

Python has an excellent style guide called PEP8. It covers most of the situations you will step into while writing Python, PEP8 can be considered a *generic Python guideline* rather than strict rules as it allows different approaches to achieve similar goals. The main aim of this guide is to have a code that is clean, consistent and efficient.

Code Layout:

The code layouts contain certain guideline to make the code written in python more effective. These are:

- Use 4 spaces instead of tabs.
- Maximum line length is 120 symbols.
- 2 blank lines between classes and functions.
- 1 blank line within class between class methods.
- No blank line following a `def` line.
- No whitespace inside parentheses, brackets, or braces.
- Surround operators with single whitespace on either side.
- Never use whitespaces around `=` when passing keyword arguments or defining a default parameter value.
- Use blank lines for *logic* separation of functionality within functions/methods wherever it is justified.
- Move function arguments to a new line with an indentation, if they do not fit into the specified line length.
- Place a class' `__init__` at the beginning of each class.
- Use multiline strings, not `\\` since it gets much more readable.
- Use named arguments to improve readability and avoid dummy mistakes in the future.
- Never end your lines with a semicolon, and do not use a semicolon to put two statements on the same line.
- Chaining methods should be broken up on multiple lines for better readability.

Naming Conventions:

The naming should be done in a predefined way so that it can reduce errors in the code. The guideline for naming is as follows:

1. General

- Avoid using names that are too general or too wordy. Strike a good balance between the two.

- Bad: data_structure, my_list, info_map, dictionary_for_the_purpose_of_storing_data_representing_word_definitions
- Good: user_profile, menu_options, word_definitions
- Don't be a jackass and name things "O", "l", or "I"
- When using CamelCase names, capitalize all letters of an abbreviation (e.g. HTTPServer)

2. Packages

- Package names should be all lower case
- When multiple words are needed, an underscore should separate them
- It is usually preferable to stick to 1 word names

3. Modules

- Module names should be all lower case
- When multiple words are needed, an underscore should separate them
- It is usually preferable to stick to 1 word names

4. Classes

- Class names should follow the UpperCaseCamelCase convention
- Python's built-in classes, however are typically lowercase words
- Exception classes should end in "Error"

5. Global (module-level) Variables

- Global variables should be all lowercase
- Words in a global variable name should be separated by an underscore

6. Instance Variables

- Instance variable names should be all lower case
- Words in an instance variable name should be separated by an underscore
- Non-public instance variables should begin with a single underscore
- If an instance name needs to be mangled, two underscores may begin its name

7. Methods

- Method names should be all lower case
- Words in an method name should be separated by an underscore

- Non-public method should begin with a single underscore
- If a method name needs to be mangled, two underscores may begin its name

8. Method Arguments

- Instance methods should have their first argument named ‘self’.

9. Functions

- Function names should be all lower case
- Words in a function name should be separated by an underscore

10. Constants

- Constant names must be fully capitalized
- Words in a constant name should be separated by an underscore

Formatting:

- Use double quotes (") around strings that are used for interpolation or that intended for the end-user to see, otherwise use single quotes (').
- Add trailing commas in sequences of items only when the closing container token],), or } does not appear on the same line as the final element.
- Always start a new block on a new line.

Commenting:

- First of all, if the code needs comments to clarify its work, you should think about refactoring or rewriting it. The best comments to code are the code itself.

- Describe complex, possibly incomprehensible points and side effects in the comments
- Separate comments and # with a whitespace.