# Incremental Retro-Compatibility Programming Standard (IRCoProS o IRCPS)

Author: Esteban Chacon Martín

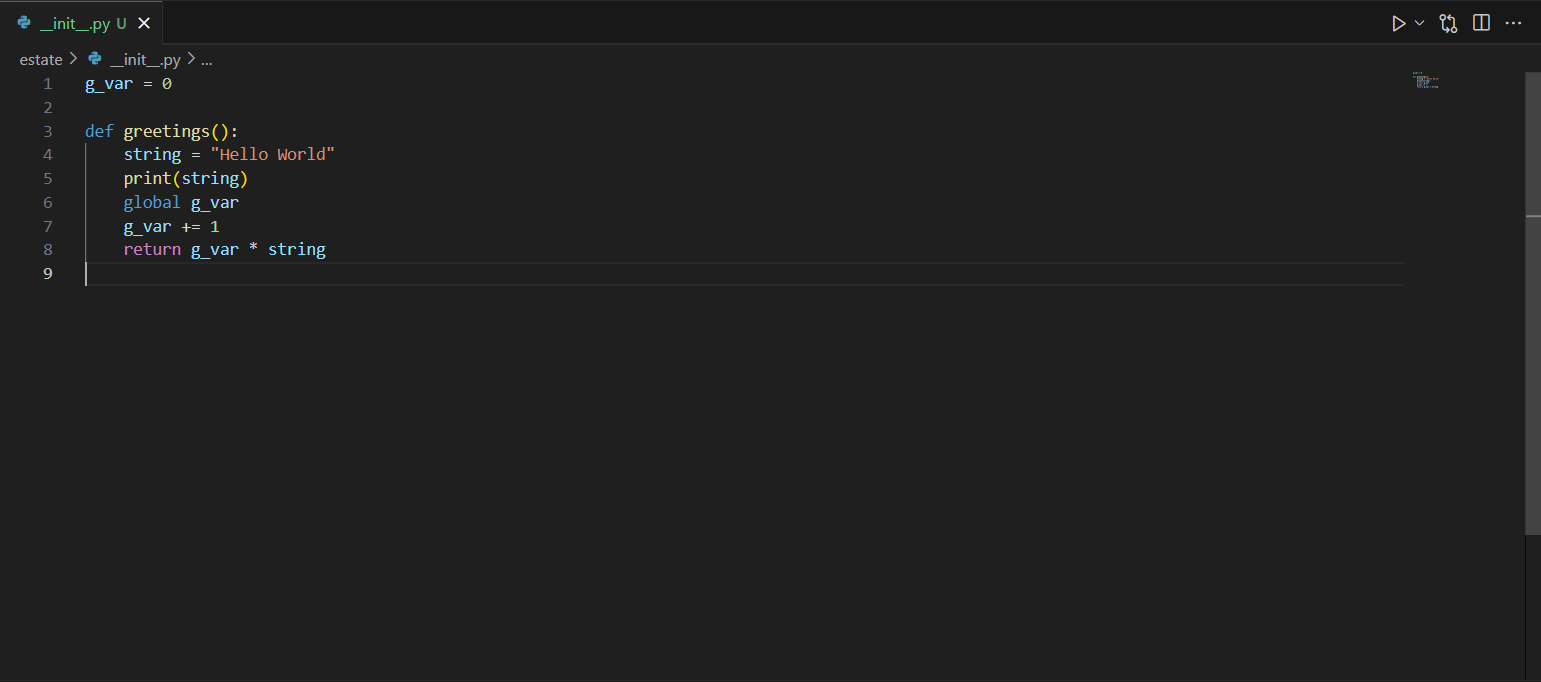
Theme: Code Standards and Retro-compatibility Management

Copyright: MIT

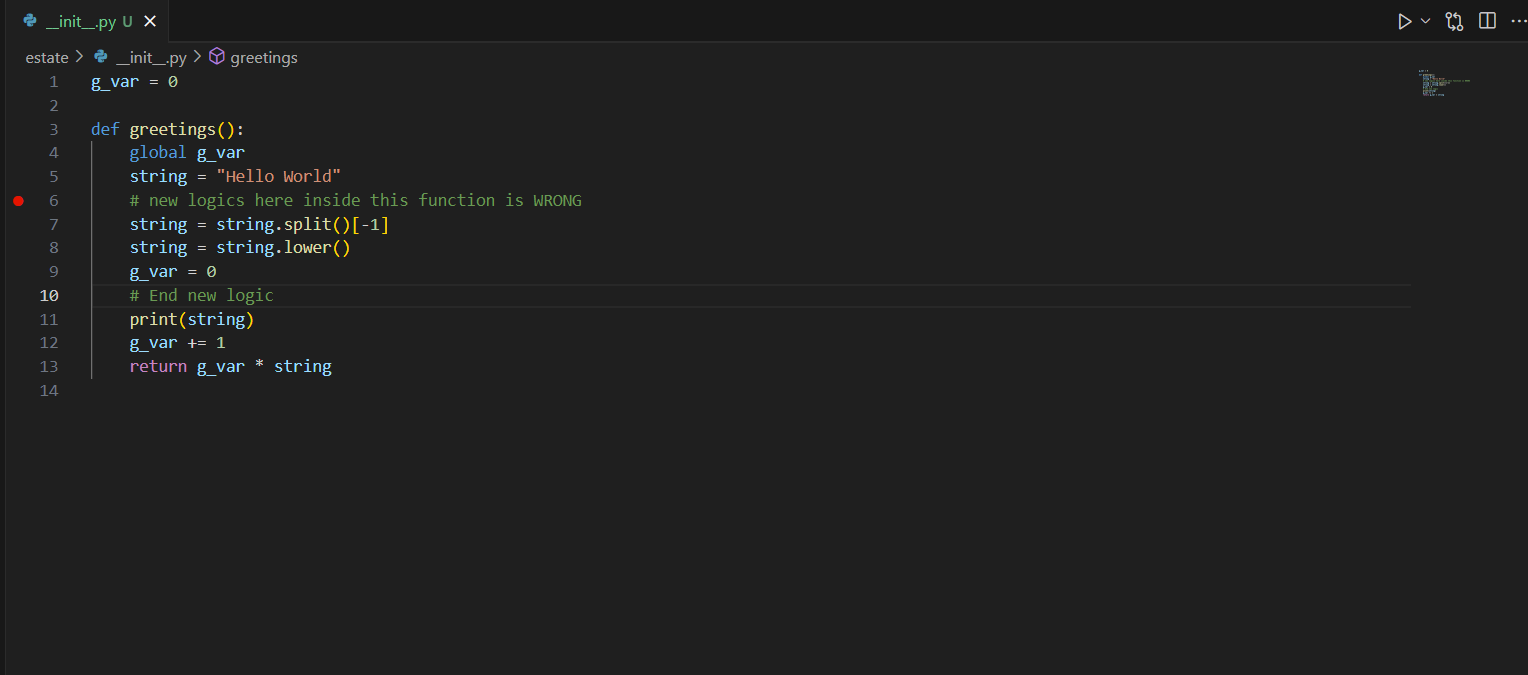
In order to reduce considerably code breaks during new releases, even in the case we are using Unit Testing we present you with a programming standard that intends to avoid most and in some cases all of the code break. In this brief document we are going to use python programming as the example programming language.

1. Avoid in block codes or isolated blocks insertion of multiple or several lines.

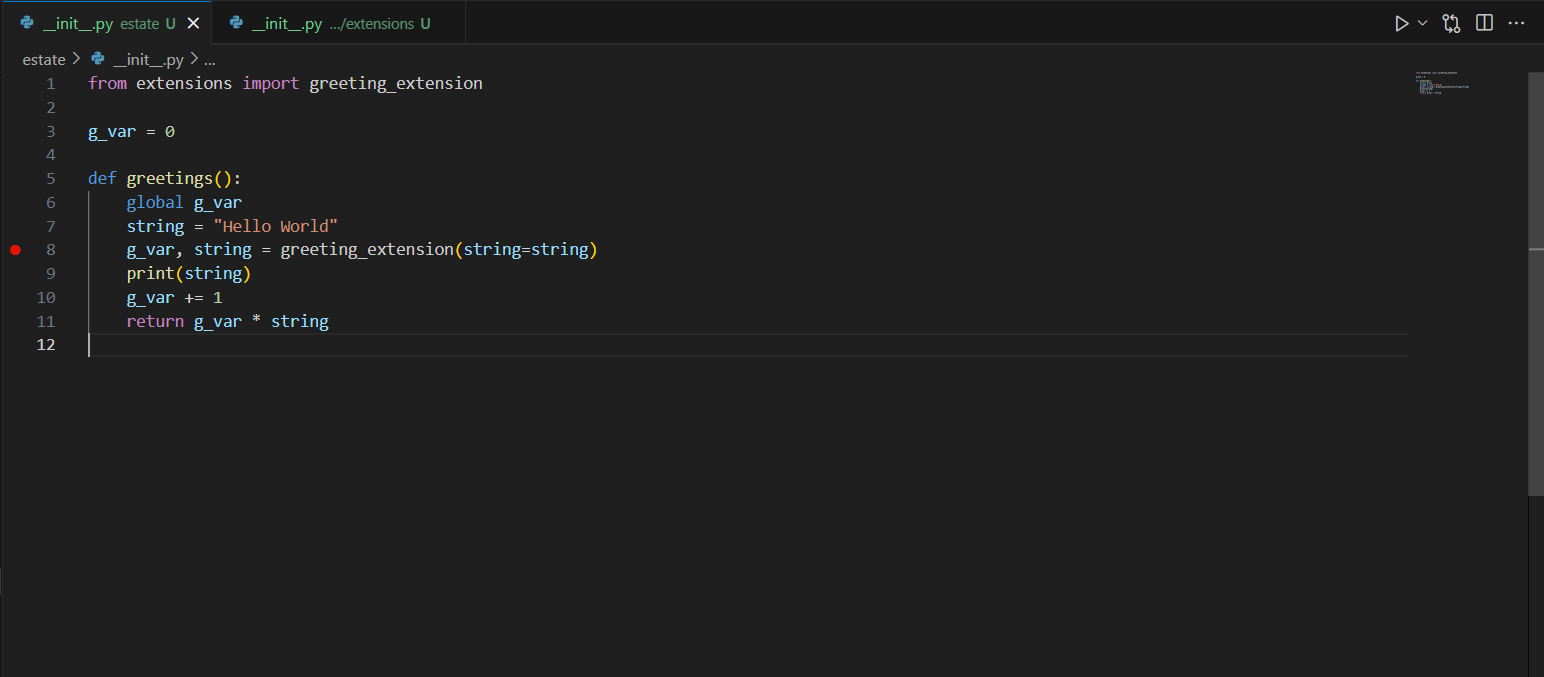
For example in a code like below:

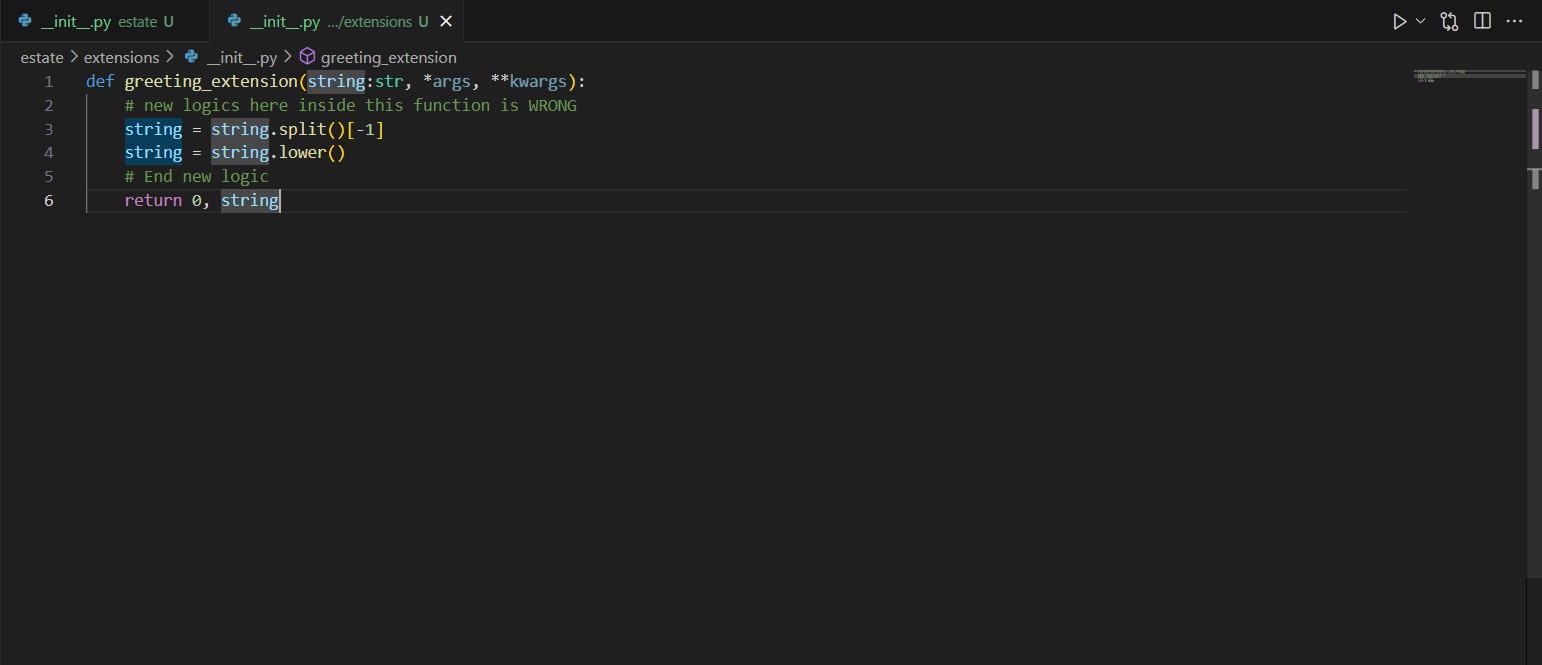


Avoid extending the code by doing something like below (WRONG):



And instead try a cleaner approach like below (CORRECT):





The above reduce PR complexity by focusing the problem in only one side, the new added files or extension methods (you may be immediately thinking about the overmodularization problem, we are going to talk about it in another chapter. For now, let's just say it is not an issue for the first hundred commits unless you are making small, poor-quality commits little by little :-) ).

1. Avoid removing or changing methods definitions, instead extend or override existing methods.

Always think in the people working with you, if you change a model definition both your code and his/her may break at some point because the methods don’t accept the same parameters any more.

(If images are required here for clarity please tell me)

1. Use interfaces to force data compatibility through upgrades, never reduce interface signature, always extend it.

In OOP we always use a base class containing all of the methods needed or attributes if is not an interface but instead an abstract class, in cases for languages like python that don’t enforce in compiling or startup time the classes’s signature you must raise the error on the constructor \_\_new\_\_ or the initializer \_\_init\_\_.

1. Once interfaces or execution stack becomes too large, consider marking them as deprecated and creating a new code under a different module with the vN (e.g. v1, v2, v3 …) naming convention at the start of the new module.

Meaning once we shouldn’t extend any more the classes or the nested code in the functions we may need to think about migrating the code by following the process described above.

1. Don’t rush the deprecation, instead implement units tests for all of the new upgrades created that test compatibility for old deprecated code with existing non deprecated code.
2. For breaking changes you will also use the 5 step approach but this time without caring about retro-compatibility (Is a breaking change after all)