The Elements of Programming Style

(Lectures on High-performance Computing for Economists VIII)

Jesús Fernández-Villaverde\textsuperscript{1} and Pablo Guerrón\textsuperscript{2}

January 27, 2022

\textsuperscript{1}University of Pennsylvania

\textsuperscript{2}Boston College
Programming Style
Motivation

- In the same way than when writing in a human language, a good style is paramount when writing code.

- Creates code that is:
  1. Clear.
  2. Robust.
  3. Easier to maintain.
  4. Easier to share.
  5. With less bugs.

- Particularly important when working with coauthors.
• Google coding standard: https://google.github.io/styleguide/cppguide.html


• *The Elements of C++ Style*, by Trevor Misfeldt, Gregory Bumgardner, and Andrew Gray.

• *The Elements of Matlab Style*, by Richard Johnson.

Main ideas

- Two guiding principles:
  1. Consistency. You can have your own rules, but apply them consistently.
  2. Doing it from start (no window dressing).

- Goals from *The Elements of C++ Style*:
  1. Simplicity.
  2. Clarity.
  3. Completeness.
  5. Robustness.
Formatting
- Keep lines short (80 columns).

- Indentation for nested statements.

- White spaces.

- Block code.

- Use parenthesis even if not extremely needed.
Naming
Naming: a motivating example

- What does this code do?
  
  \[ a = b^c \cdot d^e \]

- And this one?
  
  \[ y = (k^a) \cdot (l^{1-a}) \]

- And this third one?
  
  \[ \text{output} = (\text{capital}^{\alpha}) \cdot (\text{labor}^{(1-\alpha)}) \]

- Which one do you want to use?
Naming variables

• Variables should have names that are easy to understand:

  1. output is a good name.

  2. a is not.

• What is a good name is somewhat dependent on the context.

• For instance:

  1. If you are coding an RBC model, names of variables such as $y$, $c$, $i$, or $k$ are probably adequate.

  2. Names for counters can be easier than names for variables.
Variable names and programming languages

- Modern programming languages allow for long names.

- For instance, in Matlab:

```
namelengthmax
ans = 63
```

- Is your programming language case sensitive?

- Does your variable already exist? In Matlab:

```
exist myvariable
ans = 1
```
Five main conventions:

1. lowerCamelCase: consumptionDurablesHousehold.
2. UpperCamelCase: ConsumptionDurablesHousehold.
3. period.separated: consumption.durables.household.
4. underscore_separated: consumption_durables_household.
5. Hungarian notation: doubleconsumption_durables_household,

lowerCamelCase is perhaps the most used.

period.separated: confusion with objects, structures, and dataframes (in R).

underscore_separated: for files names.

Hungarian notation is not very useful with modern IDEs that show workspaces.
Other names

• Best practices:

  1. Use v as first letter in vectors: vCapitalGrid.

  2. Use m as first letter in Matrices: mValueFunction.

  3. Use i as the first letter in iterators: iCapitalNextPeriod.

• Idiosyncratic: I double first letter of greek letters: bbeta.
• Same ideas for functions, subroutines, objects, and classes, etc.

• Best practices:

  1. Use lowercase only for functions.

  2. Use verbs in the name of the functions.

  3. Use nouns to name classes: household.

  4. Reserve get and set for interactions with objects.

  5. is for Boolean functions: isUtilityPositive.
Comments
• In the ideal case, code is understandable without adding comments (although you always want a header).

• However, complicated pieces of code may need clarifying remarks.

• In that case, describe what the code is aimed to do, not how it does it.

• Share with others and with your future self.

• You will probably spend more time reading code than writing code.

• Document early.

• However, realize that just like code, comments have to be maintained.

• So writing code that is readable without comments can save you a lot of time when fixing bugs or updating your compendium.

• It is better to have no comments than comments that are wrong.
• Use TODO/FIXME/NOTE comments: many IDEs will gather them together automatically.

• Automatic systems to Generate documentation from source code.

• Doxygen: https://www.doxygen.nl/index.html

• Literate programming (Knuth): weave.jl, knitr.

• Reproducible Research with R and RStudio (2nd Edition) by Christopher Gandrud.

• Dynamic Documents with R and knitr (2nd Edition) by Yihui Xie.
Functions
Functions

1. Write small functions.
2. Name them properly.
3. Modular use.
4. Hide implementation details.
5. Document immediately.
7. Use function handles.
8. Avoid not passing default parameters.
Optimization
Optimization

Premature optimization is the root of all evil.

- You first want to be sure your code runs properly.
- Then, you optimize.
- Some automatic tools: -O flags in your compiler.
- You want to identify algorithmic bottlenecks and computer hotspots.
- Strategies:
  1. Benchmarking.
  2. Profiling.
  3. Vectorization.
  4. Pre-allocating memory and memoization.
  5. Unrolling loops.
  6. Inlining.