chem-bla-ics

Programming in the Life Sciences #8: coding standards

Egon Willighagen

Published October 29, 2013

Citation

Willighagen, E. (n.d.). In *chem-bla-ics*. chem-bla-ics. https://doi.org/10.59350/ckryb-b4v19

Keywords

Pra3006

Copyright

Copyright © Egon Willighagen 2013. Distributed under the terms of the Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Never underestimate the power of lack of coding standards in code obfuscation. Just try randomly to read code you wrote a year ago or four years ago. You'll be surprised with what you find. Coding standards are like the grammar in writing: they ensure that our message gets understood. Of course, the primary goal is that the CPU understands what you mean, but because programming languages are not your native language, you may not always say what you think you are saying.

Copyright and Licensing

First standard is attribution: if you use the solution of someone else, you write in your source code whom wrote the solution. Secondly, you must allow others to do the same. Therefore, you always add your name (and normally email address) to your source code, and under what conditions people may use your code. This is commonly done by assigning a license. Open Source licenses promote (scientific) collaboration, and give others the rights to use your solution, redistribution modifications, etc. They may explicitly require attributions, but often not. In a scholarly setting, you always give attribution, even if not required by the license. Remember, that software falls under copyright but algorithms typically not. Copyright/author and license information is typically added to source code using a header.

Documentation

The second thing is to document what your code is supposed to do, what assumptions are made, how people should use it, and preferably under what conditions it will fail. Comments in your source are just as much documentation as a tutorial in Word format. They are complementary, and documentation must not only be targeted at users, but also at yourself so that you understand why you added that weird check. You will not (have to) remember in two years.

Coding standards

Just like English has coding standards, programming language have too. Both also have styles, and a selection of a style is up to the author, but consistency is important. What coding standards should you be thinking about, include consistent use of variable and method names, keeping code blocks small, etc. For example, compare the following two code examples which do the same thing:

```
var method = function(string) {
  number = 0
  for (var i=0; i<string.length; i++) {
    if (string[i] == "A") number = number +1
  }
  return number
}
```

chem-bla-ics

And this version:

```
var countTheANucleotides = function(dnaSequence) {
  count = 0
  // iterate over all nucleotides in the DNA string
  for (var i=0; i<dnaSequence.length; i++) {
    if (dnaSequence[i] == "A") count = count +1
  }
  return count
}
```

Which one do you find easier to understand the function of?

1. use clear, descriptive variable and method names
2. use source code comments to describe the intention of source code
3. keep source code lines short enough that you can read the full line without (horizontal) scrolling
4. keep code blocks short enough that the fit a single screen (say, 25 lines max)

Unit testing

It is important to realize that what you intend to have the computer to calculate is something different that what your source code actually tells the computer to do. Even more important is to realize that it is not always your fault if the calculation goes wrong; in particular, the input you pass to some program can always be crafted such, that it will fool your code in doing unintended things.

But, a common cause of misbehaving code is the author itself. At first (and many, many times after that) it's just getting the code to compile: missing semi-colons, typos in variable names, etc, etc. After a bit, and hunting you down to your grave, are bugs caused by unintuitive features of programming language, libraries you're using, etc. Common (and often expensive) mistakes include for-loops missing the first or the last element, incorrect conversion of units (125 M$ expensive!), etc.

Fortunately, we can call in the help of computers for this too. We have code checking tools, and importantly, libraries to help us define (unit) tests. These tests call running code, and check if the calculated results are matching our expectation. For example, for JavaScript we could use the MIT-licensed qunit. For example, we could write the following tests (in qunit):

```
test( "counting tests", function() {
  equal(1, countTheANucleotides("AGCT"));
  equal(4, countTheANucleotides("AAAA"));
  equal(0, countTheANucleotides("GCGC"));
});
```

chem-bla-ics

OK, you get the idea. That other scientists really start to care about these things, is shown by these two recent papers:

- Ten simple rules for the open development of scientific software
- Ten simple rules for reproducible computational research