good habits a developer should start using early

1. **Focus on good code decomposition.**

   Don't cram all your code into one function or class -- rather, have each function be in charge of one primary responsibility, and make sure each class makes sense as a singular "thing".

   If your function is complicated, delegate: move subproblems into helper functions and use them in the main one. If you do this correctly, your functions will be fairly short -- about 5-15 lines long (ignoring blank lines, comments, etc).

   Likewise, make your classes "cohesive": they should contain data and functionality that make sense together (don't just toss things together).

   A good rule of thumb I use is to give functions verb clauses as names (functions do things), and classes noun classes as names (objects are things). If I can't come up with a good name, or if I need to resort to using compound statements, my code is poorly structured/is trying to do too much.

   Try and do this decompositional process as you're writing code, instead of writing all your code at once and doing it after. (For example, if you're not sure how to solve a subproblem, it's fine to just write an empty method + pretend it can solve the subproblem so you can focus on getting the overall logic of your function correct, then return back to flesh out the empty helper function later).

2. **Learn how to rigorously and systematically debug code.**

   Many beginners I see tend to code "by trial and error" -- they write code, don't understand why it's failing, try changing something, see it doesn't work, try again, and keep tweaking their code until it starts working.

   Never be aimless when debugging. Think like a scientist -- conduct experiments to either try fixing the bug or to narrow down where the bug might be, and record your results. If your hypothesis/attempted change turned out to be wrong, or if something worked when you didn't think it would, stop, pause, and figure out why.

   Bugs happen when your assumptions don't match up to reality. If something feels "off", don't ignore it -- trace that down until you pin down what the issue is/what incorrect assumption you made.

   Don't just "do" random shit until you get lucky -- be systematic about it.

3. **Start writing tests.**

   Get into the habit of writing tests for your code. Write code to test your various functions and classes so that if something goes wrong, you'll automatically have a sense of where the problem might be.

This pairs well with my previous point about writing many smaller functions -- you can write tests for each small function to help you pinpoint where an error is.

The basic idea is that you want to avoid being merely reactive when it comes to bugs. Instead of waiting for your code to break, be proactive as well and actively try and make it break. Try using your functions and classes in unusual ways, and see what happens. Do you get the expected results? Does the code handle unusual inputs correctly? Does the code break? Does the code not break when you expected it to?

(Then, when you make a change, re-run all of your old tests and see if they all pass. This will help give you confidence that your new change has not introduced any obvious bugs into your code, and will help make you more fearless when coding.)

4. **Have a good sense of craftsmanship.**

   Comment your code (cleanly). Give your variables good names. Use consistent formatting. Try and do all of these things as you're writing your code, not after. (Try googling "X style guide" or "X best practices", where "X" is the programming language you're using).

   Review and refactor your code after writing it. It's just like writing an essay -- you need a proofreading process. Don't expect your code to be perfect the first time around, especially if you weren't sure what you were doing in the first place.