How To Code It?

v2017.06.30

Attila Egri-Nagy

Here is an adaptation of the classic problem solving method[1] to computer programming. It applies to coding exercise problems in (functional) programming languages with a REPL (read-eval-print-loop). Problem solving is a messy business, asking these questions during the process may help.

Understanding the problem

No point in writing code without knowing what to do.

What is the input? What is the output? What are the types of these data items? What is the connection between input and output?
Think about a few example cases. Use pen and paper, draw if necessary. Introduce suitable notation. Notation on the paper can be used for names in the code later.

Making a plan

Decomposing the problem into smaller and simpler tasks.

If you are a beginner, play in the REPL! Use functions that work with the data types of the input and output. First just play freely, try many related functions, even without any obvious connection to the problem. This is for refreshing memory, and for simple problems, using an existing function might turn out to be a solution.

Trying to solve a problem in one go is a sign of lacking a plan. Making a plan is identifying the steps needed, i.e. decomposing the problems into simpler ones. *What are the steps of data transformations? Can we combine existing functions to do work? Can we specialize a function to fit the task? Do we process collections? How to process an element? Do we need to extract the required data items from the input?* If needed, by wishful thinking, come up with functions that would make solving the problems easier. *If we had a function that computes x, and another that computes y, would it be easier to solve the problem?*
If the problem is too difficult, try solving a similar, but easier problem first. *Is there a more general problem? or a more special one?*

Carrying out the plan

Write code, test code.

Test each step, each function if possible. Make the write-code-test-code cycle as short as possible. Do not do deep nested calls without checking each function. Mistake in the first step may appear later, disguised as a totally different problem. *Did we get the first step right? The second? …What are the test cases? Can we reason about the correctness of the code?* When confused, go back to function definitions, examine their source code. Throw away code that does not work. Leftover code from dead end attempts and different half solutions turn the original problem into a harder one. *Do we use this line of the code?*

Looking back

Check, reflect and learn. Rewrite for improvement.

Can we check the solution? Does the program do more than what is needed? Is there a way to simplify this solution? Is there a different solution? Can we use the function or its method to solve other problems? Refactor (rewrite) for simplicity, efficiency or wider applicability.

[1] *How to Solve It – A New Aspect of Mathematical Method* by George Pólya, Princeton University Press, 1945.