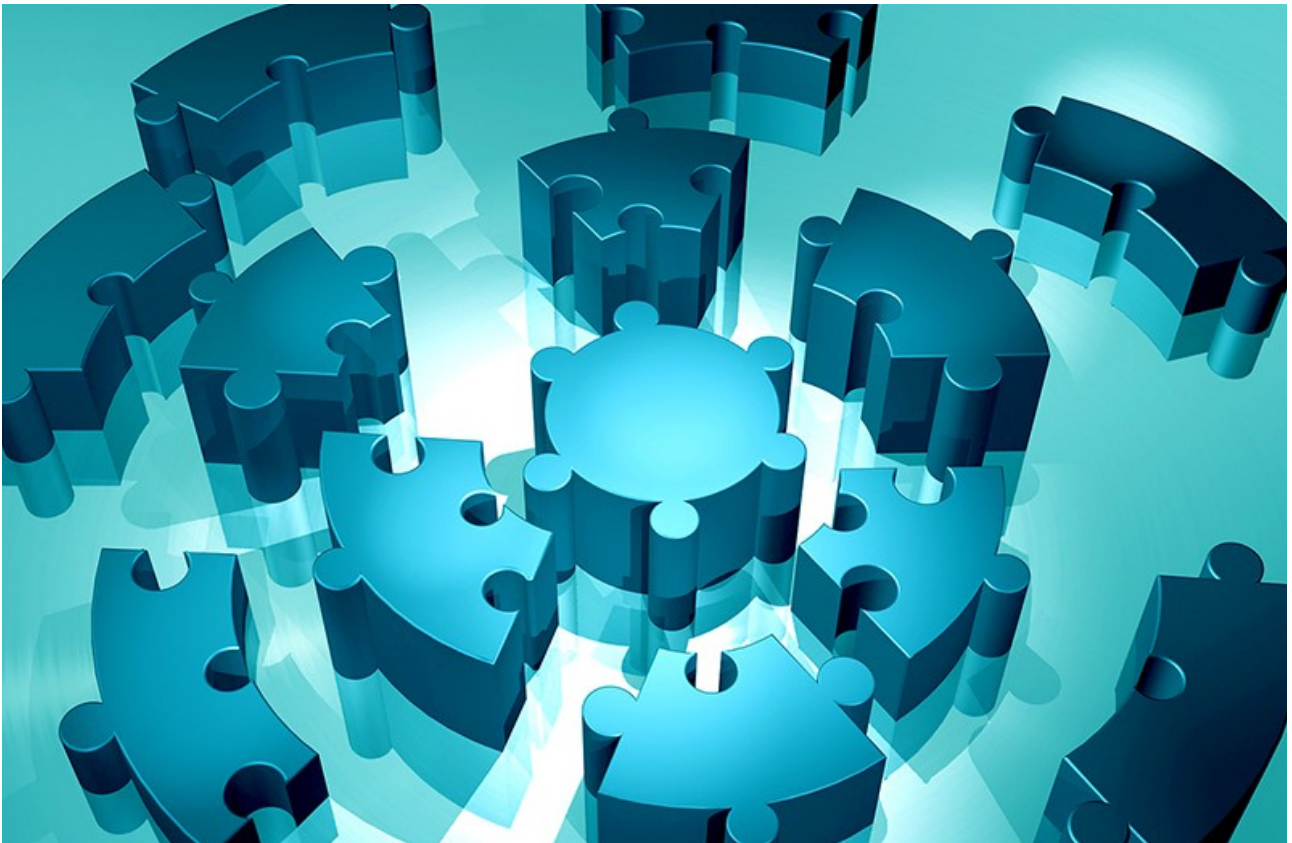


🔍 Search 5,000+ tutorials

/news /for

10 APRIL 2018 / #PROGRAMMING

# How to think like a programmer — lessons in problem solving



by Richard Reis

If you're interested in programming, you

may well have seen this quote before:

“Everyone in this country should learn to program a computer, because it teaches you to think.” — Steve Jobs

You probably also wondered what does it mean, exactly, to think like a programmer? And *how* do you do it??

Essentially, it's all about a **more effective way for problem solving**.

In this post, my goal is to teach you that way.

By the end of it, you'll know exactly what steps to take to be a better problem-solver.

## **Why is this important?**

Problem solving is the meta-skill.

We all have problems. Big and small. How we deal with them is sometimes, well...

pretty random.

Unless you have a system, this is probably how you “solve” problems (which is what I did when I started coding):

1. Try a solution.
2. If that doesn't work, try another one.
3. If that doesn't work, repeat step 2 until you luck out.

Look, sometimes you luck out. But that is the worst way to solve problems! And it's a huge, huge waste of time.

The best way involves a) having a framework and b) **practicing it**.

“Almost all employers prioritize problem-solving skills first.

Problem-solving skills are almost unanimously the most important qualification that employers look

for....more than programming languages proficiency, debugging, and system design.

Demonstrating computational thinking or the ability to break down large, complex problems is just as valuable (if not more so) than the baseline technical skills required for a job.” — Hacker Rank ([2018 Developer Skills Report](#))

## Have a framework

To find the right framework, I followed the advice in Tim Ferriss’ book on learning, “[The 4-Hour Chef](#)”.

It led me to interview two really impressive people: [C. Jordan Ball](#) (ranked 1st or 2nd out of 65,000+ users on [Coderbyte](#)), and [V. Anton Spraul](#) (author of the book “[Think Like a Programmer: An Introduction to Creative Problem Solving](#)”).

I asked them the same questions, and

guess what? Their answers were pretty similar!

Soon, you too will know them.

Sidenote: this doesn't mean they did everything the same way. Everyone is different. You'll be different. But if you start with principles we all agree are good, you'll get a lot further a lot quicker.

“The biggest mistake I see new programmers make is focusing on learning syntax instead of learning how to solve problems.” — V. Anton Spraul

So, what should you do when you encounter a new problem?

Here are the steps:

## 1. Understand

Know exactly what is being asked. Most hard problems are hard because you don't understand them (hence why this is the

first step).

How to know when you understand a problem? When you can explain it in plain English.

Do you remember being stuck on a problem, you start explaining it, and you instantly see holes in the logic you didn't see before?

Most programmers know this feeling.

This is why you should write down your problem, doodle a diagram, or tell someone else about it (or thing... some people use a rubber duck).

“If you can't explain something in simple terms, you don't understand it.” — Richard Feynman

## 2. Plan

Don't dive right into solving without a plan (and somehow hope you can muddle

your way through). Plan your solution!

Nothing can help you if you can't write down the exact steps.

In programming, this means don't start hacking straight away. Give your brain time to analyze the problem and process the information.

To get a good plan, answer this question:

“Given input X, what are the steps necessary to return output Y?”

Sidenote: Programmers have a great tool to help them with this... Comments!

### 3. Divide

Pay attention. This is the most important step of all.

Do not try to solve one big problem. You will cry.

Instead, break it into sub-problems. These

sub-problems are much easier to solve.

Then, solve each sub-problem one by one. Begin with the simplest. Simplest means you know the answer (or are closer to that answer).

After that, simplest means this sub-problem being solved doesn't depend on others being solved.

Once you solved every sub-problem, connect the dots.

Connecting all your “*sub-solutions*” will give you the solution to the original problem. Congratulations!

This technique is a cornerstone of problem-solving. Remember it (read this step again, if you must).

“If I could teach every beginning programmer one problem-solving skill, it would be the ‘reduce the problem technique.’



For example, suppose you're a new programmer and you're asked to write a program that reads ten numbers and figures out which number is the third highest. For a brand-new programmer, that can be a tough assignment, even though it only requires basic programming syntax.

例如，你是個新手，你被要求寫一個程式，能讀入10個數字，然後找出第三大的數字。

If you're stuck, you should reduce the problem to something simpler. Instead of the third-highest number, what about finding the highest overall? Still too tough? What about finding the largest of just three numbers? Or the larger of two?

Reduce the problem to the point where you know how to solve it and write the solution. Then expand the problem slightly and rewrite the solution to match, and keep going until you are back

where you started.” — V. Anton Spraul

## 4. Stuck?

By now, you’re probably sitting there thinking “Hey Richard... That’s cool and all, but what if I’m stuck and can’t even solve a sub-problem??”

First off, take a deep breath. Second, that’s fair.

Don’t worry though, friend. This happens to everyone!

The difference is the best programmers/problem-solvers are more curious about bugs/errors than irritated.

In fact, here are three things to try when facing a whammy:

- Debug: Go step by step through your solution trying to find where you went wrong. Programmers call this *debugging* (in fact, this is all a

debugger does).

“The art of debugging is figuring out what you really told your program to do rather than what you thought you told it to do.” — Andrew Singer

debug的技巧在於，弄清楚「你到底叫程式作什麼」，而不是「你認為你叫程式做什麼」。

- Reassess: Take a step back. Look at the problem from another perspective. Is there anything that can be abstracted to a more general approach?

不同角度思考看看。  
退一步看看，是不是有更通用的方式？  
有時會陷入問題的細節，但用更全局的觀點，  
反而可能用更通用的高度解出這個問題。  
例如，高斯就發現等差數列，頭尾相加會相同，  
用  $n(n+1)/2$  可以不用一個一個相加就得到總和。

“Sometimes we get so lost in the details of a problem that we overlook general principles that would solve the problem at a more general level. [...]

The classic example of this, of course, is the summation of a long list of consecutive integers,  $1 + 2 + 3 + \dots + n$ , which a very young Gauss quickly recognized was

simply  $n(n+1)/2$ , thus avoiding the effort of having to do the addition.”  
— C. Jordan Ball

Sidenote: Another way of reassessing is starting anew. Delete everything and begin again with fresh eyes. I’m serious. You’ll be dumbfounded at how effective this is.

- Research: Ahh, good ol’ Google. You read that right. No matter what problem you have, someone has probably solved it. Find that person/ solution. In fact, **do this even if you solved the problem!** (You can learn a lot from other people’s solutions).

Caveat: Don’t look for a solution to the big problem. **Only look for solutions to sub-problems.** Why? Because unless you struggle (even a little bit), you won’t learn anything. If you don’t learn anything, you wasted your time.

找大問題的答案學不到東西，只找小問題的答案

## Practice

Don't expect to be great after just one week. If you want to be a good problem-solver, **solve a lot of problems!**

Practice. Practice. Practice. It'll only be a matter of time before you recognize that “this problem could easily be solved with <insert concept here>.”

How to practice? There are options out the wazoo!

西洋棋、數學、數獨、圍棋、大富翁、電動、迷戀貓、Coderbyte...

**Chess puzzles, math problems, Sudoku, Go, Monopoly, video-games, cryptokitties, bla... bla... bla....**

In fact, a common pattern amongst successful people is their habit of **practicing “micro problem-solving.”** For example, Peter Thiel plays chess, and Elon Musk plays video-games.

“Byron Reeves said ‘If you want to see what business leadership may look like in three to five years, look

at what's happening in online games.'

Fast-forward to today. Elon [Musk], Reid [Hoffman], Mark Zuckerberg and many others say that games have been foundational to their success in building their companies." — Mary Meeker ([2017 internet trends report](#))

Does this mean you should just play video-games? Not at all.

But what are video-games all about?  
That's right, problem-solving!

So, what you should do is find an outlet to practice. Something that allows you to solve many micro-problems (ideally, something you enjoy).

For example, I enjoy coding challenges. Every day, I try to solve at least one challenge (usually on [Coderbyte](#)).

Like I said, all problems share similar patterns.

## Conclusion

That's all folks!

Now, you know better what it means to “think like a programmer.”

You also know that problem-solving is an incredible skill to cultivate (the meta-skill).

As if that wasn't enough, notice how you also know what to do to practice your problem-solving skills!

*Phew...* Pretty cool right?

Finally, I wish you encounter many problems.

You read that right. At least now you know how to solve them! (also, you'll learn that with every solution, you

每次解題成功，都會學到如何解決問題，然後進步

improve).

“Just when you think you’ve successfully navigated one obstacle, another emerges. But that’s what keeps life interesting. [...]

每解決一困難，就會出現另一個困難，但就是讓生活有趣的地方。

Life is a process of breaking through these impediments — a series of fortified lines that we must break through.

Each time, you’ll learn something.

Each time, you’ll develop strength, wisdom, and perspective.

Each time, a little more of the competition falls away. Until all that is left is you: the best version of you.” — Ryan Holiday ([The Obstacle is the Way](#))

Now, go solve some problems!

And best of luck ?



AND BEST OF LUCK .

**Special thanks** to C. Jordan Ball and V. Anton Spraul. All the good advice here came from them.

Also, all the programming knowledge I've acquired in such a short time wouldn't have happened without Lambda School. Can't thank/ recommend them enough.

Thanks for reading! ? If you enjoyed it, test how many times can you hit ? in 5 seconds. It's great cardio for your fingers AND will help other people see the story.

SAY HELLO ON TWITTER



@richardreeze

"Hello!"

If this article was helpful, [tweet it.](#)

Learn to code for free. freeCodeCamp's open source curriculum has helped more than 40,000 people get jobs as developers. [Get started](#)

Continue reading about

## Programming

How to Make Programming a Daily Habit

---

What I Learned from Reading "The Pragmatic Programmer"

---

How to Learn Flutter in 2020

---

[See all 2735 posts →](#)



#JAVASCRIPT

### Learn React in 5 minutes - A React.js tutorial for beginners

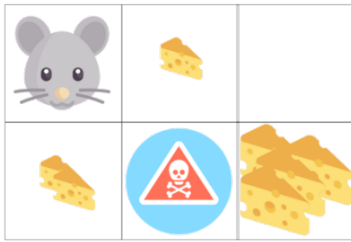


PER HARALD DØRGEN

2 YEARS AGO



TERRENCE DÖRNER · 2 YEARS AGO



#MACHINE LEARNING

## Diving deeper into Reinforcement Learning with Q-Learning

2 YEARS AGO

freeCodeCamp is a donor-supported tax-exempt 501(c)(3) nonprofit organization (United States Federal Tax Identification Number: 82-0779546)

Our mission: to help people learn to code for free. We accomplish this by creating thousands of videos, articles, and interactive coding lessons - all freely available to the public. We also have thousands of freeCodeCamp study groups around the world.

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

You can [make a tax-deductible donation here](#).

### Our Nonprofit

[About](#)[Alumni Network](#)[Open Source](#)[Shop](#)[Support](#)[Sponsors](#)[Academic Honesty](#)[Code of Conduct](#)[Privacy Policy](#)[Terms of Service](#)[Copyright Policy](#)