THE 'LEARN TO CODE ON THE WEB' ALGORITHM: AN EASY AND INCLUSIVE RECIPE FOR NEWBIES LEARNING ON THE WEB

Kgotso Koete, January 2017 [updated in February 2017]

No more lies...

This learning to code trend conceals a lot of nuances. There is hard and easy content, and there is the hard or the easy way to learn. When the dominant narrative is that computer programming is easy, people are often talking about the easy way to learn the easy part. The opposite can be said for those who say it's hard or that it must come naturally.

I am a newbie who struggled in the beginning, but successfully managed to learn the basics. I am now deep diving into more advanced newbie topics such as algorithms and data structures.

The 3 biggest issues with programming advice that people give is:

> 1) Advice is not empathetic - programmers forget that they were once civilians too.
> 2) The advice often lists too many resources without giving the newcomer a method to prioritize - what's the point of overwhelming people.
> 3) Advice often does not consider that people have different cognitive strengths, weaknesses, and life circumstances. This requires different kinds of resources that accommodate different situations - the curse of knowledge and privilege makes people forget how they learned to code in the first place.

We don't need another list of '30 resources to help you learn to programmed in language x'. While those are helpful, what we need a simple recipe that newbies can use and adjust.

This is a simple 4 step '*learn to code 'recipe:*

> [1] Know your cognitive preferences and learning style and be comfortable with any limitations.
> [2] Learn the basic theory by trying and switching between learning resources till you find a good fit.
> [3] Go to the gym and practice micro-problem solving before switching to macro-problem solving.
> [4] When you have learned the basics and begin to specialize, repeat step [1] - [3], and adjust this recipe.

***

HERE IS THE 4 STEP RECIPE

These steps may seem like a lot, but you can through all of this in a matter of months because you will be switching between resources anyway. Don't worry about the # and the {}, this is a primer on how you will be implementing your algorithms in pseudocode. Besides, I just could not help but write fake code :).

[1] PREFERENCES: Figure out your inclinations and learning **preferences. //** Do you prefer videos vs text, theory vs practical, detail vs visual?

[2] TRY BASIC THEORY & SWITCH: Learn to code the easier way by solving for a platform that will **minimize learning friction,** don't solve for content/language yet. Your starting language does not matter in the medium/long-term, you will learn to switch languages once you understand language structure/feature anyway.

# While you are still learning newbie fundamentals, do the following:
{

# **[A] PRACTICE:** While you are not stuck on basic syntax issues, practice by using your preferred learning method:
{

Try a:

A theory inclined university course: A University Course like Harvard's CS50 on EdX (I WOULD START WITH CS50) or MIT 600.1x (EdX) or similar on Coursera.

Or switch to a:

A practical online course: If you don't like theory, then do Free Code Camp, graduate and work on real projects for non-profits (I am yet to try this, I will only start in Feb 2017).

Or switch to a:

A book: If you really prefer text instead of videos, then read a book like *Learning Python The Hard Way (for example)*. You can crowdsource advice on step [A], by following step [B] below.

}

# **[B] ASK:** When you get stuck because of syntax issues, then:
{

Try:

Crowd source answers: Google or ask a questions in the relevant community channel/page on Stack Overflow (your new bestie), Reddit, Quora, YouTube: In that order.

Or switch to a:

Cheat Sheet: You can look for other people's code for the same problem on GitHub or Geek for Geeks. It's not about copying and pasting, it's about learning how others solved similar problems.

Or switch to a:

Quick and dirty course: If you feel you really need to brush up, then do a Code Academy or Udacity course in the language giving you problems.

Or switch to a:

A visual programming course: If it is really bad and computer code is still hieroglyphics for you, then do a visual programming course from UC Berkeley in a fun language called BYOB. I would not spend too much time on this.

}

# **[C] CONNECT:** While you learn, it's good to accumulate more knowledge outside of your computer and meet people (outside of the internet):
{

Try:

Meetups: Meeting other developers through events from Meetup.com for networks such as Code & Coffee or user groups (Linux User Group or Python User Group etc.).

Try:

Podcasts: Code Newbie by Saron Yitbarek is a cool, inclusive and newbie friendly show you can listen to. In fact I am addicted to that show.

Try:

Blogs: Medium, classic blogs such as Norvig's, and Hackerdom if you want a more pungent geeky flavor. If the cultural paradigms on these blogs make you feel excluded, revert to the step right above this one.

}

}

[3] DELIBERATE PRACTICE AT GYM: Once you have a broad view of what programming is about you can then deliberately practice some key modules to develop a micro-problem solving ability and muscle memory. This will not teach you how to develop software application, instead it will teach you to solve various smaller chunks of problems through computation. It's gym/workout for programmers, each challenge takes anything from 5 min - 3 hours to complete.

// Feel free to skip this step if you are comfortable with **# STEP [2]**. Many of the world best programmers did not do competitive programming.

{

 # [REINFORCE BASICS] Reinforce beginner principles by watching the videos and doing the challenges for topics you feel weak on through My Code School, language syntax, data structures and algorithms are always good places to start.

 # [GET GOOD] Practice the easy challenges by following the beginner path's (just filter for easy problems) on Hacker Rank or Hacker Earth.

 # [BECOME AN EXPERT] Practice more advanced problems and participate in a few of the open coding competitions. I would not spend too much time on advanced deliberate practice if the goal is still to progress to more advanced theory concepts. I have not gotten to this step yet. Try and let me know if this step is a good fit for this meta learning algorithm.

Tweak this algorithm for new needs/circumstances/skills.

 # [GO BACK TO MACRO-PS] While coding challenges are important, the world's most impactful software developers were not necessarily top coders on programming competitions. After gaining muscle in the micro-problem solving gym, go back, and learn macro-problem solving techniques via **# STEP [4]** and the next iteration of **# STEP [2]**.

// Remember, the key to learning anything is to breaking down that craft into smaller modules that you can learn. And then breaking down each module into sub-modules that you can tackle independently.

}

Continues next page...

# [4] SPECIALISE & ADJUST: When you are ready to start solving for deeper/specialized content (web or Machine learning etc.) + related language (JavaScript or Python respectively), then:
{

    # [NEXT PHASE] Rinse, wash, and repeat loop [1] and then [2] above for next phase of deeper/specialized learning.

    # [TWEAK] Tweak this algorithm for new needs/circumstances/skills, or abandon it all together.

    // Remember, learning a new language is very manageable when you have the basics. Programming languages may have different capabilities, but they all share similar structures/ features.
}

***

A FINAL WORD

If this recipe looks like English to you even with the indentations, the # and {}, then it will be easy to talk to machines. Or else, just keep trying different combinations and switching resources from the options above and I promise it will make sense soon.

The key to this recipe is SWITCHING between [A], [B], and [C] in # STEP [2] whenever you are stuck. I did the Harvard CS50 course and I thought I would never be able to programme. I then SWITCHED to Code Academy to learn the basics. I then SWITCHED to the MIT equivalent of CS50 (MIT 600.1x) and did just fine only because it was a more focused curriculum. I also noticed that it took me very long to finish the problem sets because I did not know how to break down and structure small problems. I then SWITCHED to deliberately practicing micro-problem solving on My Code School and Hacker Rank.

Don't let cognitive friction make you feel stupid. Don’t let experienced programmers lie to you about how ‘real programmers learn’. Let’s end the practice of giving absolutist advice in programming, especially to people who have different circumstances.

Feel free to suggest how to improve this algorithm so that newbies are never lied to again. Fork and share as much as you like.

No more lies
Kgotso Koete