7 Powerful Reasons Why You’re Struggling to Code & How to Fix Them.

Are you struggling to code? Wow! That is a great achievement because it means you are becoming a software developer already.

Learning to code can be very hard especially for beginners. That is why you are struggling currently. You might be confused about which approach to follow because there are a lot of conflicting approaches and ideas.

You might be memorizing code or overloading yourself with a lot of things. Now, you are distracted and confused.

That is a very common experience to all programmers. When I started learning to code, I had the same experience. I was confused about what to follow because I had access to a lot of styles and opinions on the internet just like you.

I kept learning whatever came my way because I felt all of them were important and I might lose out on what I really needed to grow if any of the information was ignored.

See, you’re smart, brilliant and capable. Nothing is wrong with you. You only need to develop a programming mindset and learn with resources that don’t make everything unnecessary difficult for you. Anyway, below is all you need to know to **reduce** your struggle.

Paradox of teaching: Teachers may be Ignorant.

I am always afraid of sharing my knowledge without having a “fully formed” understanding of the subject matter because it will hunt hundreds of thousands of people in the future but everyone these days is advocating **teaching as you’re learning**.

Though that is a great thing to do for a learner, it might be dangerous to a lot of people that would learn from it in the future.

Look, no tutorial is perfect but it shouldn’t be unnecessarily difficult. Recently, I studied a lot of JavaScript tutorials on the internet and, honestly; I struggled to understand what many of the tutorials taught.

See, I have a sound understanding of JavaScript, yet I struggle with the tutorials.

In fact, it is possible to build an application anyhow in as much as it works and satisfies its intended purpose but it is a different ball game when it comes to teaching how to build it.

A complex code base is hard to explain. It could be explained on the surface but it hardly leads to a deep understanding of the subject matter.

For example, below is a carousel code.

```
var slideIndex = 1;
showSlides(slideIndex);

// Next/previous controls
function plusSlides(n) {
  showSlides(slideIndex += n);
```

```javascript
}

// Thumbnail image controls
function currentSlide(n) {
  showSlides(slideIndex = n);
}

function showSlides(n) {
  var i;
  var slides = document.getElementsByClassName("mySlides");
  var dots = document.getElementsByClassName("dot");
  if (n > slides.length) {slideIndex = 1}
  if (n < 1) {slideIndex = slides.length}
  for (i = 0; i < slides.length; i++) {
        slides[i].style.display = "none";
  }
  for (i = 0; i < dots.length; i++) {
      dots[i].className = dots[i].className.replace(" active", "");
  }
  slides[slideIndex-1].style.display = "block";
  dots[slideIndex-1].className += " active";
}
```

Read and digest it. Easy or hard? Stop! Just read another code that does the same thing below:

```javascript
let images = [
  //list of images
];

let currentImageIndex = 0;
let lastSlide = images.length - 1;

const display = (currentImageIndex) => {
 const sliderImage = document.getElementById("slider_image")
 sliderImage.src = images[currentImageIndex];
}

display(currentImageIndex);

const isPositive = number => Math.sign(number) === Number(number);
const isNegative = number => Math.sign(number) === Number(-number);

const next = (increment) => {
  if(currentImageIndex < lastSlide && isPositive(increment)) {
    currentImageIndex += increment;
  }
  display(currentImageIndex);
}

const previous = (decrement) => {
 if( currentImageIndex > 0 && isNegative(decrement)) {
   currentImageIndex += decrement;
 }
 display(currentImageIndex);
}

//creates page list
const paginate = (items) => {
 const list = document.getElementById("pagination");
 list.innerHTML = items.map((item, index) => `<span class="dot"
onclick="display(${index })"> ${index + 1} </span>`).join(" ");
}
```

If you have read the code above. Both of them do the same thing. **Do any of them make it easy or hard?**

Now, make your judgement on how your learning materials can be a major reason why you are struggling to code. In short, **learning to code starts with finding the right resources to learn from**. Don't take it for granted.

PS: https://youtoocancode.com has a well researched and complete solution for you already. Just check it out**.**

You don't know what you're doing if you memorize code?

It is natural to memorize code when you don't understand what you are doing. That is expected as the resources you are learning from make everything difficult for you to understand.

All I can tell you now is don't memorize code or it will break your heart 💔! It will force you to struggle unnecessarily.

Here is why:

Code memorization gets you confused whenever you apply your knowledge because you don't really understand what you're doing.

Code memorization occurs whenever you store some code in your head without understanding what the code represents.

Since you don't understand concepts, you will struggle to build projects.

You might even give up…

For example,

```
let students = ['Ope', 'Ayo', 'Ola'];
let count;
for ( count = 2; count >= 0;  --count ) {
  console.log(students[count]);
}
```

What will this log in the console? See, if you don't truly understand how for loop, increment,

and decrement work, you will find it hard to understand the code above.

Then, what do you have to do?

Whenever you realize you don't really understand a piece of code, break the code down into pieces

and research the use of each of the pieces. Let's use the code above as an example.

1st: You have to understand forLoop

```
for (...){ }
```

What is it used forLoop?

It is used to repeat an action or a series of actions several times

2nd: You need to understand initialization.

```
for ( initialization; conditional; increment or decrement ) { }
```

What is it used for?

It is to set the beginning of the loop. It determines the position or index to start with.

3rd: You need to understand conditionals.

What is it used for?

It is used to set the condition that must be met for the loop to continue running.

If the condition evaluates to true, the loop will keep running but if it evaluates to false, the loop will terminate (stop).

4th: You need to understand increment and decrement.

What is increment used for? It is used to add 1 to a number.

Increment can be divided into pre-increment **(++increment)** and post-increment **(increment++)**.

How are they different? After adding 1 to a number, pre-increment returns the result as in:

```
let number = 5;
console.log(++number)// 6
```

Six is returned after one was added to five. Post-increment will return the number will added one to instead of the result as in:

```
let number = 5
console.log(number++) // 5
console.log(number) //6
```

You see, post-increment returns 5 after adding 1 to 5 and when we check the number, it is now 6.

In short, whenever pre-increment is used with a number, its result is returned. But if post-increment is used, the number itself is returned instead of its result. What they return makes all the difference.

So, what is the application of that in a loop?

Whatever the increment or decrement returns doesn't affect the loop as the loop only deals with the result we get after applying post or pre increment to number.They both add one to the initializer...

But we also have decrements...they subtract one from count. That means, in this case, count will be decreased by one at every iteration.

```

let students = ['Ope', 'Ayo', 'Ola'];
let count;
for ( count = 2; count >= 0;  --count ) {
  console.log(students[count]);
}

```

So students [count] will return 'Ola' at the first iteration because its index is 2.

It will return 'Ayo' at the second iteration because its index is 1 and 'Ope' will be returned at the third iteration as its index is zero then the loop will terminate because count will be less than zero and condition will evaluate to false.

That is it…

Stop overloading yourself!

Overloading is dealing with more tasks than you can handle at a given time. Several times, you have the urge to learn everything quickly and gain every possible opportunity programming can offer.

So, you end up overloading yourself with tasks or assign yourself a complex task within a short period of time. Oh, no! That is one of the reasons why you're struggling to code.

I can still remember how learning JavaScript was very hard for me because the silly me overloaded myself. I learnt **VAR, LET, CONST**, function, array, object and conditional statement in less than 30 minutes thinking I was ready to become a world-class software developer.

At the same time, I was learning other things in Python, PHP and still reading John Resig books "Professional JavaScript '' that explains a lot of advanced concepts in JavaScript and how he developed or borrowed solutions to create jQuery.

One more thing...I finished a 100 pages text book and a 6-hour long video tutorial in just an hour.

Honestly, there is nothing wrong with reading or learning widely but there is time for everything. You can't know everything -- it is the only impossibility I believe. So, don't overload yourself.

Choose a part of your project or a programming language you know is stupidly simple to build or understand and ask a lot of reasonable and stupid questions about it because there is a **sense** in non**sense**. Hey wait! Let me give you an instance:

Imagine you want to learn JavaScript and you just decide…hey, I am going to learn just LET and CONST today, and nothing else. And you ask a lot of stupid and reasonable questions about them.

By doing that, your learning pace will seem slow but you're most likely going to outpace your counterparts that rushes everything except they have special brains which is a rare case.

Yes, handling several tasks at the same time or doing a thing with insufficient time makes coding difficult for you. It can also force you to quit if you achieve a reasonable result along the line. So, keep whatever you learn stupid simple and **tame your curiousity because it is the opposite of focus**.

Remove yourself from conflict of perspectives.

There are a lot of conflicting opinions about programming languages, libraries, frameworks, techniques and approaches such as **"Don't Repeat Yourself (DRY)**", "**Test Driven Development (TDD)"**, "Profit Driven Development (PDD)", "You Ain't Gonna Need It (YAGNI)", "Clean Code", "Always Be Coding (ABC)", "Favour Composition over Inheritance" and many others.

It is tempting to take sides based on the opinions of some software developers you respect. See, don't take any side until you truly understand what you're doing. Your job currently is to understand the fundamentals and build projects with it.

Then you will come across some challenges that will force you to use some approaches, libraries and frameworks. All approaches or techniques are useful based on context. **No approach is either right or wrong, it is context that makes it so**.

All of them are useful in solving real world problems and all of them have downsides. So, don't worry too much about them until you have understood the fundamentals of your chosen programming language and built some basic project with it.

Develop research mindset

Programming, most of the time, is like researching; you start with observation and then test several things until you achieve a satisfactory result. That is what programming looks like. You don't really know what you're doing until it works.

Once you know that, you would be content with **necessary** struggles because most of the time, you have to find out how to fix a bug you have never seen in your career and it might really demand too much effort from you.

In that case, you may struggle to fix it and to be honest, that is a necessary struggle. It is acceptable because no software developer is free from such a struggle. Well, maybe **10x engineers (smiles)** are free from it but I am not. Being an expert is not about having answers to all questions, it is about knowing how to find it.

Deal with your distractions first

We all have distractions, so what is your distraction? Is it your current job, husband, wife, girlfriend or survival struggle? You need to figure out your distractions and schedule your coding activities and time in such a way that you will not be distracted whatsoever.

You can't really be that effective with coding if you have a lot of distractions in your life, so you need to identify them and fix them one by one. Then, you might increase your chance of achieving your programming dreams.

Don't work too hard

Relax! Don't overwork yourself because you really need to understand the concepts you are dealing with at this stage and it takes an effective mind to understand. So, don't overwork yourself to affect your learning effectiveness.

Programming is hard for experienced software engineers, let alone a beginner. Working too hard without getting reasonable results may discourage you easily. When you are a bit drained mentally, stop to continue later.

Quit. Quit that task for a few minutes. Work away from your computer. Sleep and free your mind but don't forget to try one more time...one more time until you understand it.

Conclusion

Trust me, I am honest. Struggling is a good thing when it is necessary. It makes you a better person or a better software developer. Don't get me wrong - I am not telling you to struggle unnecessarily.

Although no software developer, I repeat; no software developer is free from struggling to code but it is worse to struggle unnecessarily by learning with the wrong resources or

not asking for help from a mentor or someone who can help you whenever you are stuck.

If you **need a reliable mentor and a community of like-minded people who will always help you out whenever you are stuck**, get “JavaScript For A Total Novice” **paid course + mentoring** at **https://youtoocancode.com** or **chat with Ayobami On WhatsApp**.