NEWBIE CODER *problem solving* SCHOOL

A SELF-PACED ONLINE COURSE TO HELP YOU SOLVE CODING PROBLEMS WITH CONFIDENCE AND EASE!

BY NICOLE ARCHAMBAULT

WELCOME

Welcome to Newbie Coder Problem Solving School!

I'm Nicole Archambault, and I'm going to be your instructor and guide as we explore the world of programmatic problem solving and algorithmic challenges—on a newbie-friendly, 101 level!

First, a little about myself. Plus, I'll be interlacing more of my story throughout the course. My name is Nicole Archambault, and I'm kind of a lot of things, among course instructor. But in general, I'm a front-end web developer, and also an educational technology enthusiast. With this passion, I've created the La Vie en Code community, including the La Vie en Code Podcast, blog, and online courses. It's very important for you to understand how I got to the point where I created this course, because you'll understand so much more about me. Lots of courses out there are churned out, and not a true labor of love focused the people we help. Students are just students, rather than the complex human beings we actually are. You're more than that to me, and this is why.

I've always been a lover of tech, since I was a little kid. I loved to take things apart with no intention whatsoever of putting them back together. I had, as many others of my generation, found deep fascination with building websites through online journals and early social media pages like MySpace.

I began my journey as an aspiring Computer Science major in 2003 at Wellesley College. Unfortunately, my passion was quickly thwarted by a total overwhelm in the classroom in general. With difficult concepts, it made the overwhelm even worse. I struggled on problem sets and ended up frustrated beyond belief by my sophomore year.

Like, what was wrong with me? Wasn't I good with computers? And if so... why was this so damn hard? It felt like I couldn't truly understand or absorb anything past basic data structures and algorithms. It was awful, and I had to figure out ways to understand things and make them click for me. I started to associate learning with struggling and self-loathing. And little did I know, that classroom environment and the trauma it brought for me would follow me into my adulthood. I switched majors and didn't look back... until I was 29 years old.

I graduated in 2007 a Political Science and Spanish double major—a Wellesley survivor, so to speak. I had gotten stuck in a string of dead-end administrative jobs. Gradually, I build my career and built some leverage, before I left an archaic-ass maritime organization. Finally I found Customer Service employment in a tech startup, and left my previous job. It felt good to finally have some control over my career. And wouldn't you know, I was seduced into my love of tech all over again. I was excited to go into work every day. I wanted to learn everything I could about what I had missed over the years.

This job was a really important stepping stone, and I didn't expect it to be forever. Startups are volatile. After I lost that job back in 2015, I was more of less at a total loss of what to do from that point forward. Should I stick with mediocre, boring jobs that frowned on spending time on things like automation that could bring their business into the future—or being brushed off by the engineers at a tech company, as I had been at the startup? With unemployment checks guaranteed for the next several months... I decided to teach myself to code. Initially it was just "to build websites", but I quickly learned that the web development world was so much more than that. I had no idea the wonderful world I was about to step into.

Over the next several months, I studied obsessively. My relationship with coding was so new, so special, and still honeymoon-like. I spent my time finding and leveraging online resources—both free and paid—to build up my knowledge, with a pretty loose goal of becoming a back-end Ruby or PHP programmer. Soon after, I kinda pivoted and fell in love with the front-end. I started a little blog called La Vie en Code to talk about my experience, and tell the story of my struggle into the technology themselves. People found me, and we connected on shared experience. Turns out, being unapologetically honest and transparent really opens you up to new possibilities.

With my new familiarity with both sides of web development, I got my first job at a local web development shop after 10 months of learning largely on my own. It was just a contract, and I had to leave because of family complications, but god damn, it was developer money. I was paid to learn and build, and my dreams were coming true, tiny step by tiny step. But how would I make money in this industry without working for an actual company? By this point, I had given new life to the blog in the form of the La Vie en Code Podcast. I began doing bigtime tech community work, starting a local Meetup group for freeCodeCampers. I tried freelancing for a little while in 2017, but still felt deeply unfulfilled. There was something else—something greater—I was meant to be doing. I was supposed to be creating something greater.

During my time learning to code, I had became aware of my learning process and noticed patterns in my thinking... the errors I made, and why I made them. This is part of why I started the blog back in 2015—to start sharing my growth notes with others via the blog and podcast.

It felt like I was building my own instruction manual. Instead of just reading a book, which didn't typically work well for me, I was able to teach myself how to code using video-based courses, and be given challenges to use what I had just learned. Then, once I truly understood what I was learning, I could read the books to further solidify my knowledge and help me to expand it from there.

Once I discovered what it was capable of, my appreciation for educational technology quickly turned to passion, and I decided to embark on a mission to help new coders like myself to learn the fundamentals that I was constantly finding myself needing to go back to. Fundamentals like autodidactic (or self-teaching) skills, to help learners understand how their brains absorb and retain information to speed up their process learning to code. I wanted to help people learn to solve actual problems, because I experienced so much blank screen paralysis, I'm honestly amazed I stuck through with my self-education. Code-agnostic problem solving isn't taught in online courses, and especially not in Computer Science degree programs and coding bootcamps. So someone had to fill those gaps, and it's me!

Driven by my passion and new goals, I last year refocused in on La Vie en Code as a brand of online courses to help people just like me, and designed to also help as many others as possible in the places I had messed up. We can learn in one of two ways: by doing it correctly, or doing it incorrectly and seeing what doesn't work, then figuring it out with that knowledge in hand. This is the ultimate way to connect with people—being completely transparent about the entire experience, and willing to learn from others as I teach them what I know.

So there you have it: I'm an educational technology stan, because it's changed my life for the better. I keep getting feedback that I'm really strong with storytelling and sharing with others, so I'm finally building the confidence to trust that I know enough, and not hold back out of fear of being wrong. I'm truly hoping that confidence will rub off on you in the process of taking this course, in addition to the confidence you build simply from knowing what to do after taking Newbie Coder Problem Solving School. Ok, so you know who I am, and I want to get to know you, too. Let's jump into the meat and bones of this course.

Are you excited?

Hell yeah!

I'll see you in the next lesson, my friend.

Nicole

LESSON 1

WHY WE NEED PROBLEM SOLVING SKILLS

KEY POINTS FROM THIS LESSON

- Problem solving skills will help you through challenges faster, and help you advance sooner
- Programming is 80% problem solving, 20% coding
- You can't build if you don't understand your tools and what they do
- This course has no code, because you don't need code to learn to solve problems effectively
- Don't build on shaky foundation

PROBLEM SOLVING IS THE FOUNDATION OF EVERYTHING ELSE PROGRAMMING-RELATED

NOTES FROM THIS LESSON

LESSON 2
"THINKING LIKE A PROGRAMMER"

TRAITS OF PROGRAMMATIC THINKERS

So, here's a list of traits/thinking you might see associated with someone who “thinks like a programmer”. (and I'd love to hear any more you come up with!) All of these traits are able to be learned! It just requires embracing your growth mindset, and testing out a new perspective.

- Curious about how things work
- Wants to fix things that don't work
- Automatically thinks of situations in which something *wouldn't* work (what we would consider edge/corner cases)
- Envisions themselves creating things to accomplish various tasks.
- Sees problems as a challenge and opportunity for growth, not an obstacle.
- Tends to see many options where those who don't think programmatically see few or none.
- Sees complex things and wants to break or whittle them down into smaller, simpler bits.
- Can quickly spot redundancy.
- Has great appreciation for:
  - Effectiveness: making things work well
  - Efficiency: making them work quickly
  - Automation: building processes
- Able to come up with solutions on their own—and may be addicted to the rush that comes with it!
- Sees (and hears) patterns in the world.
- Willing to start small with the goal of building upward as their skills and knowledge grows
- Tends to try lots of different ways to solve a problem on their own before asking for help.
- You might have some rogue tendencies (haha I know I do!!) if you see a better option than what's being given to you.

LESSON 2
"THINKING LIKE A PROGRAMMER"

YOU CAN SEE THAT YOUR ABILITY TO THINK PROGRAMMATICALLY IS HEAVILY INFLUENCED BY YOUR CURRENT THINKING, BUT AGAIN—IT CAN BE LEARNED! JUST ENSURE YOU'RE AWARE OF THESE SPECIAL WAYS OF SEEING THE WORLD AROUND US, AND CHALLENGE YOURSELF TO THINK DIFFERENTLY!

KEY POINTS FROM THIS LESSON

- Problem solving can be fun!!
- Programmatic thinking operates off "on/off" or "true/false", there's no grey area
- Another example: counting starts at 0, not 1
- Some people naturally think programmatically—they have distinct traits!

NOTES FROM THIS LESSON

LESSON 3
VARIABLES

VARIABLES ARE ONE OF THE FIRST PROGRAMMING TOPICS WE ENCOUNTER WHILE LEARNING TO CODE.

WHAT CAN YOU STORE IN A VARIABLE?

WHERE CAN YOU USE A VARIABLE?

KEY POINTS FROM THIS LESSON

- Variables are like "buckets" that reference different types of data—individual units of data, or even data structures with many elements in them
- Place data in variables when you want to reference them in multiple places
- Remember to declare your variables before you put stuff in them, or the computer won't know what you're referencing
- Name your variables semantically to avoid confusion over what they're referencing

LESSON 3
VARIABLES

NOTES FROM THIS LESSON

LESSON 4
FUNCTIONS

HOW FUNCTIONS MAKE UP ALGORITHMS

Functions are, like everything else in this course, an important building block of our programmatic problem solving. When you use them wisely, they'll give you much more control over the code you write. And you can recognize early on in your problem solving—often before you even write code—where you'll be using them.

HOW CAN YOU SPOT FUNCTIONS IN ADVANCE?

KEY POINTS FROM THIS LESSON

- Functions allow you to put multiple steps into one easy-to-reference, reusable chunk
- Be sure to start out by ensuring you have all the different elements you're using—like our car we want to wash
- Use functions when you see multiple steps you may want to reuse

LESSON 4
FUNCTIONS

NOTES FROM THIS LESSON

LESSON 5
ALGORITHMS

</>WHAT ARE ALGORITHMS?

</>WHAT DO WE USE ALGORITHMS FOR?

</>WHAT KINDS OF ALGORITHMS ARE THERE?

LESSON 5
ALGORITHMS

NOTES FROM THIS LESSON:

KEY POINTS FROM THIS LESSON

- Algorithms are pretty powerful, and a central concept you'll need to be familiar with in order to move along smoothly in your education.
- But you don't need to be afraid of them. In fact, the more you get to know them, the easier your problem solving process will be overall.

LESSON 5
ALGORITHMS - SEARCHING

WHEN DO WE USE LINEAR SEARCH OVER BINARY SEARCH?

WHAT YOU NEED TO KNOW ABOUT LINEAR AND BINARY SEARCHES IS THIS:

1. Linear searches go through the data set one by one, and evaluate it according to some set of logic (for example, in our guessing game: does it match the number the computer is expecting?)
2. Binary searches evaluate a data set by selecting a point of reference, evaluating the logic response, and then narrowing down the set of data from there.

NOTES FROM THIS LESSON:

LESSON 5
ALGORITHMS - SORTING

WHAT ARE THE TWO TYPES OF SORTING ALGORITHMS

KEY POINTS FROM THIS LESSON:

- Sorting algorithms shouldn't be difficult to understand from a basic concept perspective. We take a set of values, and reorder them according to those values. Sometimes, in the case of strings, those values will first be changed to a value that the computer can easily compare—like integers.
- You'll learn more and more about sorting algorithms as you delve deeper into your education. Often, algorithms will require that you first sort data before doing something with it.
- There are many different algorithms that sort data in a specific way. We'll discuss in the "Measuring efficiency" video how we determine the efficiency of an algorithm, to ensure we choose the right one for the job.

NOTES FROM THIS LESSON:

LESSON 5
ALGORITHMS - MEASURING EFFICIENCY

WHAT IS EFFICIENCY ?

KEY POINTS FROM THIS LESSON:

- There are a lot of factors that go into determining whether or not an algorithm/program is efficient! We leverage the level of efficiency when we make decisions about the type of algorithm to use in our problem solving approach.
- It's useful to be familiar with algorithmic efficiency, which is why I've included it in this course. You should be aware of the fact that the code you write will eventually be measured against other code in terms of how efficiently it runs. But when you first start coding, you should focus entirely on just getting it working—Big O notation can wait!

NOTES FROM THIS LESSON:

LESSON 6
LOOPS

LOOPS ARE ONE OF THE BACKBONES OF PROGRAMMING, AND THEY'RE A CRITICAL—AND MASSIVELY USEFUL!—TOOL IN YOUR CODING TOOLBOX.

WHY DO WE USE LOOPS?

LESSON 6
LOOPS - ITERATORS

WHAT 3 THINGS DO ITERATORS DO?

KEY POINTS FROM THIS LESSON

- Understanding iterators is so important to understanding loops overall, because it affects your problem otherwise your code simply won't run as expected.
- Each programming language will have a different syntax for loops, but these basic elements will exist no matter where you end up.

LESSON 6
LOOPS - FOR LOOPS

THE FOR LOOP IS A TYPE OF LOOP CENTERED AROUND ITERATORS, SO IT'S APPROPRIATE TO HIGHLIGHT IT AFTER ITERATORS.

WHERE DO YOU USE FOR LOOP?

KEY POINTS FROM THIS LESSON

- FOR loops are centered around iterators, which dictate where we begin, and how we move through the data set. And the code within the FOR loop does something with the code.
- You're going to be using FOR loops a lot in your code while solving code challenges and building software. There's no escaping them, so hopefully you feel more comfortable with them!

NOTES FROM THIS LESSON:

LESSON 6
LOOPS - WHILE LOOPS

WHILE LOOPS ARE ANOTHER TYPE OF LOOP YOU'LL USE WHILE PIECING TOGETHER YOUR CODE CHALLENGE SOLUTIONS, AND BUILDING UP YOUR OWN SOFTWARE.

WHERE DO YOU USE WHILE LOOP?

KEY POINTS FROM THIS LESSON

- You don't need to know ALL the details of each type of loop, but at least being familiar with them will get you thinking about what kind of approach you could use.
- But the most important part is recognizing from the start of your problem solving process which tools from your toolbox you'll use. This allows you to go in with some direction and structure after planning out the code you'll write.
- But in short, seeing repeating actions while building your approach —which we'll be walking through soon—should ultimately direct you toward loops, and you'll be able to choose the right loop for the job!

NOTES FROM THIS LESSON:

LESSON 7
CONDITIONALS

ONE OF THE MOST POWERFUL FUNDAMENTALS OF PROGRAMMING IS LOGIC FLOW—KNOWN AS "CONDITIONALS" IN THE PROGRAMMING WORLD.

HOW DO CONDITIONALS AFFECT YOUR PROBLEM SOLVING?

KEY POINTS FROM THIS LESSON

- Conditionals and logic flow are absolutely unavoidable, and are going to be a huge part of your core programming tools. So get comfortable with them!
- One of the best ways to fine-tune your logic flow is to practice code challenges. Yes, really. In fact, the best way to practice all of these tools—loops, conditionals, algorithms—is just to write them.
- Being aware of these tools will change the way you see each of these building blocks. When something doesn't work, you can better think in terms of where it might be going wrong. Perhaps you didn't evaluate the correct conditionals, or by the appropriate parameters?
- Does your loop know where to stop? Should you isolate and test it—we'll be talking about this shortly—to ensure it works as a stand-alone entity?
- If you're doing a search or sort, is it producing the correct output at each step? Are you searching/sorting for the correct element?Simply being aware will change the way you solve problems, and make you far more competent when it comes down to producing a working solution to your code challenges!

LESSON 7
CONDITIONALS - IF STATEMENTS

IF STATEMENTS ARE ONE OF THE FIRST CONDITIONAL OPERATORS YOU'LL USE WHILE PROGRAMMING.

KEY POINTS FROM THIS LESSON

- Here's the thing to remember: most problems won't be written out this way, structured in English and starting with "IF". Often, you'll have to determine these IF statements on your own. When you notice a comparison that needs to be made between two values, isolate it as a small problem, and make sure you're getting the expected output.
- That's a small building block that we'll be using to build the bigger machine. Just being able to identify them is key.

NOTES FROM THIS LESSON:

LESSON 7

CONDITIONALS - IF...ELSE STATEMENTS

IF...ELSE STATEMENTS ALLOW US TO DEFINE EXACTLY WHAT THE PROGRAM SHOULD DO IF THE FIRST STATEMENT RETURNS FALSE!

KEY POINTS FROM THIS LESSON

- When you need to tell your program what to do in case your IF statement returns "false", use an ELSE statement!
- Depending on your language, the ELSE statement will conventionally be placed right after you tell the computer to do if the evaluation returns "true".
- IF...ELSE statements allow us to do one thing and one thing only: provide a backup case if our IF statement evaluation returns "true.

NOTES FROM THIS LESSON:

LESSON 7
CONDITIONALS - ELSE IF STATEMENTS

AS I MENTIONED IN THE LAST LESSON... WHAT IF WE WANT TO RETURN SOMETHING SPECIAL WHEN THE NUMBER WE'RE EVALUATING AGAINST 25 IS 4? ELSE IF STATEMENTS LET US MAKE THAT HAPPEN.

KEY POINTS FROM THIS LESSON

- ELSE IF statements allow us to do more than just define what happens when our IF condition evaluates to "false". They allow us to define other evaluations, beyond just the first IF statement we use. They're powerful, and you'll be using them a lot in your problem solving!
- Use ELSE IF statements when you want to tell the computer to perform different evaluations. You'll be able to spot them early on when you identify your IF statement in your problem solving approach... but you also need to evaluate a value against more than one other value.

NOTES FROM THIS LESSON:

LESSON 7
CONDITIONALS - SWITCH STATEMENTS

SWITCH STATEMENTS ARE SIMILAR TO ELSE IF STATEMENTS, BUT ALLOW FOR A MUCH MORE STREAMLINED EVALUATION PROCESS.

KEY POINTS FROM THIS LESSON

- SWITCH statements are powerful, and I included them because you should know about them as a tool in your toolbox.
- Again, the entire purpose of both the ELSE IF and SWITCH statements are to evaluate something, then provide cases for what happens based on what the value of that "something" is. If it's this, do that. If it's not, skip that case and move on.
- ELSE IF statements can be a bit more taxing because they require going through the entire chain of ELSE IFs, and it can be hard to see where you may have gone wrong. You'll also use SWITCH statements when you want to keep your evaluation-handling options readable and maintainable.
- My ultimate goal by including all of these toolbox topics is to get you thinking about recognizing these elements while solving your problem. Even if you manage to define the problem and plan your approach perfectly before even writing code, you can only do what the language lets you do—the way it lets you do it.
- The next time you use an ELSE IF statement while handling different options in your problem solving, also ask yourself if a SWITCH statement with different cases might be a better way to do it!

NOTES FROM THIS LESSON:

LESSON 8
DATA STRUCTURES

DATA STRUCTURES ARE USED IN ALL KINDS OF PROGRAMMATIC APPROACHES. YOU'LL PARTICULARLY USE THEM WHEN YOU'RE TEMPORARILY HOLDING DIFFERENT KINDS OF DATA WHILE YOU DO THINGS WITH IT.

WHAT ARE THE 4 MAIN FUNCTIONS OF DATA STRUCTURES?

WHAT ASPECTS DO WE CONSIDER WHEN EVALUATING OUR DATA STRUCTURE CHOICES?

KEY POINTS FROM THIS LESSON

- You'll be using data structures at every turn of your problem solving approaches—separating and putting data together, modifying data we already have, and holding data while we get ready to do something else with it.
- We ask ourselves lots of questions before ultimately deciding which data structure to use, depending on the type of output we're looking for and what we need to do with it.

LESSON 8
DATA STRUCTURES - ARRAYS

ARRAYS ARE A POWERFUL DATA STRUCTURE THAT YOU'LL FIND EVERYWHERE IN PROGRAMMING. KNOWING WHERE THEY ARE WILL HELP YOU KNOW HOW TO BETTER MANAGE YOUR DATA.

WHY ARE ARRAYS IMPORTANT TO PROBLEM SOLVING?

KEY POINTS FROM THIS LESSON

- Arrays are one of the most basic data structures you will use
- Arrays are great when you want to be able to access data by its "slot" in the array
- You can also add, remove, and modify data easily within an array
- Arrays can hold other data structures

NOTES FROM THIS LESSON:

LESSON 8
DATA STRUCTURES - OBJECTS

WHAT ARE OBJECTS?

WHEN WILL YOU USE OBJECTS?

KEY POINTS FROM THIS LESSON:

- Objects offer 4 main benefits: encapsulation, abstraction, inheritance, and polymorphism
- Abstraction keeps the inner workings of our objects "under-the-hood", accessible by its name
- Encapsulation is like the opaque "container" that holds your entire object together
- Inheritance is an object's ability to automatically receive all of a parent class's traits
- Polymorphism refers to your object's ability to accept different types of input, and run different code for each
- An object's characteristics are called its properties, and its abilities are methods
- Individual objects, or instances, are created from classes
- Classes are like rubber stamps, or blueprints—they can be further customized, but always come from a pre-defined template
- The process of making a new instance is called instantiation
- Use objects when you want to create a prototype of something with fully contained data, or manage a big code base

LESSON 8
DATA STRUCTURES - LISTS

HOW ARE LISTS DIFFERENT FROM ARRAYS?

KEY POINTS FROM THIS LESSON

- Arrays use an index value for each item in the array
- Lists use nodes, containing data and a reference point to the next node
- Lists can hold a mixture of different data types, unlike arrays
- Both can be iterated through
- Lists insert items in constant time, arrays do not
- Inserting data into a list doesn't change the rest of the list, whereas arrays shift indices

NOTES FROM THIS LESSON:

BEFORE WE GET STARTED

In this section of the course, we'll get into the meat and bones of problem solving, and the step-by-step process of getting from problem to solution.

Ok, now comes the part you probably signed up for! All of the previous review of our toolbox was extremely important, though—it will help you identify the tools you'll be using while you're working through your problem solving approach. When you go to write the actual code, you'll have a much better sense of which tools you'll use.

The problem solving process covered in this course isn't particularly unique, but I have personally fine-tuned it to a system that works for me. It's really amazing how many professionals—especially new coders— don't go through this process. They end up either spending way too much time on the solution... or worse yet, solving the wrong problem entirely.

In the next lesson, I'll be introducing you to an actual coding problem from the amazing online learn-to-code platform, freeCodeCamp. This platform has been built by actual devs using open source technology, so it reflects the types of problems you'll actually need to solve both in your learning and your future career. I recommend freeCodeCamp highly for new devs who want to poke their toe in the water without committing any actual money to the process.

As a reminder, there will be no actual code in any language used in this course. I've intentionally hand-picked problems that can be solved without using code. My overall goal is to help you understand why we are using those particular concepts (as mentioned in our Toolbox section) to solve our problem.

Ok, I'm excited, and you should be too!

THE CHALLENGE
SUM ALL NUMBERS IN A RANGE

WALKING THROUGH THESE PROBLEM SOLVING STEPS WOULD BE HARD IF WE DIDN'T HAVE AN EXAMPLE TO WALK THROUGH. LET'S TAKE AN ACTUAL PROBLEM YOU MIGHT ENCOUNTER IF YOU WERE BUILDING UP YOUR SKILLS OUT IN THE WILD!

KEY POINTS FROM THIS LESSON

- You can find these kinds of problems on all kinds of platforms. I'll review them in my bonus material!
- Sample input is great because it can help you re-phrase the problem, which is Step 2
- Sometimes the function will be pre-defined. That's okay! Use it if it's provided.

NOTES FROM THIS LESSON:

STEP 1
READ AND UNDERSTAND THE PROBLEM

BEFORE WE BEGIN WRITING ANY CODE, WE NEED TO START WITH READING AND UNDERSTANDING THE PROBLEM!

WHAT IS THE FIRST STEP, BEFORE YOU WRITE CODE?

KEY POINTS FROM THIS LESSON

- You NEED to read the problem out loud! Not just in your head. Out loud, at least three times!
- If you're not given the problem in written form, ask for it to be written down
- Ask yourself questions while reading
- Write down any questions or uncertainty regarding the problem immediately to avoid problems down the road!

NOTES FROM THIS LESSON:

STEP 2
RE-PHRASE THE PROBLEM

REPHRASING A PROBLEM IS A GREAT WAY TO VIEW IT FROM A DIFFERENT ANGLE. THIS IS SO IMPORTANT, BECAUSE YOU'LL BEGIN TO THINK DIFFERENTLY, AND SEE APPROACHES YOU MAY NOT HAVE SEEN BEFORE. AND IT WILL ONLY HELP TO SOLIDIFY YOUR UNDERSTANDING OF THE PROBLEM.

HOW SHOULD YOU RE-PHRASE THE QUESTION?

KEY POINTS FROM THIS LESSON

- Re-phrasing problems will help you see a different perspective—our brains are cool like that!
- Consider:
  - Moving around the order of question parts
  - Highlighting key words you want to be able to reference easily
  - Using bullet points so the criteria you need to satisfy will stand out
- Feel free to use my re-phrasing as an example, but absolutely try things to figure out what works for you. They're little brain hacks.

NOTES FROM THIS LESSON:

STEP 3

IDENTIFY YOUR INPUT, OUTPUT, AND VARIABLES

GOING BACK TO OUR ORIGINAL PROBLEM, SUM ALL NUMBERS IN A RANGE, LET'S IDENTIFY THE DIFFERENT ELEMENTS OF OUR PROBLEM, WHICH WE'LL LEVERAGE WHEN BUILDING A SOLUTION.

WHAT'S YOUR INPUT?

WHAT 3 AREAS DO YOU NEED TO IDENTIFY VARIABLES IN?

KEY POINTS FROM THIS LESSON

- Identifying your input and output is critical to ensuring you approach further steps in the problem solving process correctly
- Spotting places that you could potentially assign variables will get you thinking about the structure and flow of your program
- Consider major milestones in your program—what would you need to save to use again in the future?

NOTES FROM THIS LESSON:

STEP 4
BREAK DOWN THE PROBLEM

BREAKING DOWN PROBLEMS MAY REQUIRE MORE THAN ONE PASS. IT COULD BE TRICKY TO SEE THE SMALLEST LEVEL TO WHICH A STEP COULD BE BROKEN DOWN, AND GIVING THE ENTIRE PROBLEM BREAKDOWN A SECOND OR THIRD PASS WILL REVEAL NEW OPPORTUNITIES TO MAKE THE PROCESS EASIER FOR YOU.

WHAT ARE THE 4 SMALLER PROBLEMS IN A MAIN PROBLEM?

KEY POINTS FROM THIS LESSON

- Breaking down the problem helps you begin visualizing the type of code you'll write to accomplish the task
- Try to write your problem breakdown as comments in your code editor
- Input should be checked as a first step
- Your program's output should be the last step

STEP 5
PSEUDOCODE YOUR SOLUTION

PSEUDOCODE IS A PROBLEM SOLVING APPROACH WHERE YOU'LL WRITE PLAIN LANGUAGE THAT REPRESENTS THE CODE OF YOUR PROGRAM. THERE WON'T BE ANY CODE SYNTAX INVOLVED IN THIS STEP, BUT IT'S JUST AS IMPORTANT AS WRITING THE CODE ITSELF.

HOW CAN SHOULD YOU PSEUDOCODE YOUR SOLUTION?

KEY POINTS FROM THIS LESSON

- Pseudocoding is the process of taking your problem breakdown and translating it to "not quite code"
- You'll consider what kinds of tools and data structures you'll be using to solve the problem, BEFORE you even write code!
- Look at one step at a time
- Say out loud what you're trying to do. It will help with identifying your tools, like a Loop or an IF statement
- Returns are the last step in delivering our output
- Be sure to save your return value to a variable—it's valuable!
- If things aren't optimal, don't worry! We'll do that in the next step.

STEP 6
OPTIMIZE YOUR PSEUDOCODE

GOING BACK TO OUR ORIGINAL PROBLEM, SUM ALL NUMBERS IN A RANGE, LET'S IDENTIFY THE DIFFERENT ELEMENTS OF OUR PROBLEM, WHICH WE'LL LEVERAGE WHEN BUILDING A SOLUTION.

WHAT'S YOUR INPUT?

WHAT 3 AREAS DO YOU NEED TO IDENTIFY VARIABLES IN?

KEY POINTS FROM THIS LESSON

- Identifying your input and output is critical to ensuring you approach further steps in the problem solving process correctly
- Spotting places that you could potentially assign variables will get you thinking about the structure and flow of your program
- Consider major milestones in your program—what would you need to save to use again in the future?

NOTES FROM THIS LESSON:

STEP 7
WRITE YOUR ACTUAL CODE

THE GOAL THIS TIME IS TO TRANSPOSE YOUR PSEUDOCODE INTO PROGRAMMING CODE. YOU'LL LEARN A LOT IN THIS STEP, BOTH PERTAINING TO CODE, AS WELL AS HOW THIS PHASE OF THE PROBLEM SOLVING PROCESS WORKS.

KEY POINTS FROM THIS LESSON

- Start by taking it line-by-line, translating your intended action into code to accomplish it
- Learn strong research skills that will help you find what you need, faster
- Don't worry about code not working! Just do your best, and debug later
- Ask for help after trying everything you can think of, but before you get frustrated and burn out
- Effectively communicate to others: what you want, what you're getting, and what you've tried while trying to fix the problem

NOTES FROM THIS LESSON:

STEP 8
DEBUG

"DEBUGGING" IS THE PROCESS OF FIXING CODE THAT EITHER:

1. ISN'T UNDERSTOOD BY THE COMPUTER AT ALL, OR
2. IS UNDERSTOOD, BUT THE COMPUTER ISN'T GIVING YOU THE OUTPUT YOU WANT

YOU'LL TYPICALLY FACE TWO DIFFERENT TYPES OF BUGS:

1. SYNTACTICAL BUGS
2. UNEXPECTED OUTPUT BUGS

KEY POINTS FROM THIS LESSON

- Debugging is the process of fixing code that doesn't work
- There are two types of bugs you'll encounter: syntactical errors, and unexpected output errors
- There are lots of different types of tools you'll use to debug, including: logging, error handling, isolating, and testing—we'll talk about them all soon! This lesson ended up pretty long
- Read your code out loud, and develop your own methods for describing the different code elements in a way you'll better contextualize them
- Compare your pseudocode to your actual code
- Make friends with a supportive rubber ducky, and talk to it about your program when you run into problems.
- Explore IDEs and learn how to use their debugging
- Pay attention to your error messages, and Google them so others' experience can help you
- Take a break! You'll be able to see new perspectives when you return.
- Revisit the tips in this lesson when you need a little direction. Even if you have to use all of them, I promise that one of them will help you get that code running.

STEP 8
DEBUG

NOTES FROM THIS LESSON

RESEARCHING EFFECTIVELY

WHILE YOU'RE LEARNING TO CODE, YOU'RE GOING TO HAVE LOTS OF QUESTIONS TO ANSWER. THE ENTIRE PROCESS OF PROGRAMMING IS ASKING YOURSELF QUESTIONS, AND ARRIVING AT ANSWERS. SOMETIMES, WE ALREADY KNOW THE ANSWERS. BUT A LOT OF THE TIME, WE NEED TO FIND THEM SOMEWHERE. BECAUSE SOMEBODY OUT THERE KNOWS THE ANSWER WE NEED, AND THEY'VE SHARED IT WITH US. WE UNDERESTIMATE EXACTLY HOW VAST THE INTERNET IS, AND HOW MUCH INFORMATION IT CONTAINS.

KEY POINTS FROM THIS LESSON

- Programming consists largely of asking questions and finding answers
- Build Google skills to find resources QUICKLY so you can keep moving forward
- Form strong questions to help you find exactly what you need
- Specify your language to see actual code examples
- Leverage Stack Overflow results—you likely won't need to post!

NOTES FROM THIS LESSON:

LOGGING

ONE VALUABLE TECHNIQUE TO INTRODUCE TO YOUR DEBUGGING PROCESS IS TO LOG YOUR CODE AT DIFFERENT POINTS. LOGGING YOUR CODE MEANS THAT WE'LL INSTRUCT THE COMPUTER TO TELL US WHAT'S PRODUCED AT WHATEVER STEP WE CHOOSE.

WHERE AND WHEN SHOULD YOU LOG?

KEY POINTS FROM THIS LESSON

- Debugging is tough—use every tool at your disposal!
- Logging works alongside commenting out code to help isolate problem points
- You can log virtually anything that has a value, even if it's null
- Check out your loops to make sure you're getting what you expect
- Log variables to see their values

GOOD COMMENTING PRACTICES

ONE OF THE MOST POWERFUL TOOLS IN OUR PROBLEM SOLVING TOOLBOX IS COMMENTING. LEARNING TO COMMENT THE RIGHT THING IN THE RIGHT PLACE IS A SKILL THAT MANY DEVELOPERS DON'T PICK UP AT ALL. AS A NEW DEV, YOU'LL WANT TO LEARN HOW TO LEVERAGE THEIR POWER AS EARLY ON AS POSSIBLE BECAUSE IT WILL MAKE YOUR DEV EXPERIENCE MORE ENJOYABLE OVERALL. WHEN USED INTELLIGENTLY, THEY WILL HELP YOU—AS WELL AS OTHERS—UNDERSTAND AND NAVIGATE YOUR CODE WITH LESS FRUSTRATION.

KEY POINTS FROM THIS LESSON

- Leverage comments wisely, and they'll be your best friend
- Use comments to isolate code blocks for debugging OR note-taking on your approach
- Comments don't affect your program, so comment away!
- As a new coder, don't worry about "best practices"—do what makes sense for YOU, and benefits YOU in the development of your solution
- Try using comments as placeholders for your weak areas of syntactical knowledge
- Code tells you how, comments tell you why

NOTES FROM THIS LESSON:

TESTING

TESTING IS A SOLUTION TO THIS PROBLEM. WE WRITE A SET OF RULES THAT YOUR CODE MUST SATISFY IN ORDER TO PASS THE TEST. IF AT ANY POINT SOMETHING UNEXPECTED HAPPENS, WE'LL BE ALERTED TO THE EXACT PROBLEM. WHEN YOU WRITE THESE RULES AND TEST AROUND WHAT YOU EXPECT YOUR PROGRAM TO DO, YOU CAN ENSURE YOUR PROGRAM RUNS WITHOUT ERRORS OR UNEXPECTED RESULTS.

KEY POINTS FROM THIS LESSON

- Testing: writing a set of rules that your code must satisfy in order to pass the test
- Helps ensure your programs run without errors or unexpected results and identify bugs at an early stage
- Writing tests can be fun!
- Different types of tests: unit, integration, and user interface (UI)Try testing as you build your program

NOTES FROM THIS LESSON:

OBJECT-ORIENTED DESIGN PATTERNS

DESIGN PATTERNS ARE REUSABLE SOLUTIONS TO COMMON PROBLEMS FACED IN PROGRAMMING. WITH DESIGN PATTERNS, YOU DON'T NEED TO REINVENT THE WHEEL—THE HARD WORK HAS ALREADY BEEN DONE FOR YOU. AND NO, IT'S NOT CONSIDERED CHEATING TO USE CODE THAT OTHERS HAVE WRITTEN TO ACT AS A TEMPLATE FOR YOUR OWN SOLUTION.

KEY POINTS FROM THIS LESSON

- Design patterns are reusable solutions to common problems faced in programming, represented by objects
- Christopher Alexander created the concept of design patterns—not a programmer at all!
- Now-famous book released: Design Patterns: Elements of Reusable Object-Oriented Software authored by Gang of Four—who originated design patterns in programming
- Don't need to reinvent the wheel
- Language-agnostic, with new patterns emerging constantly
- Design patterns are "discovered", not "created"Incomplete by design to help promote customization
- Communicate easily with other devs by mentioning the design pattern by name—they'll know why you did what you did because it's a common approach

NOTES FROM THIS LESSON:

OBJECT-ORIENTED DESIGN PATTERNS NOTES

NICOLE'S FINAL ADVICE

Hey, guess what? You made it! You should be so proud of yourself, because this course had some really valuable information to help you improve not just your coding experience—but your life. Yes, this stuff can get you a job, and change your life. And by this point, I hope you trust me enough to believe that.

If you weren't even able to start code challenges before, you now officially have a first step. Just read the problem out loud three times! And after that, a second step. And a third. You know to ask for help, and leverage tools already available to you. Before you know it, you'll be at the final step with great code that attests to your knowledge, and convinces yourself—and others—that you know what you're doing. They'll trust you to find solutions to the problems you face, and that's exactly what we want!

In this course you learned:

- What problem solving entails, and why we need to get comfortable with it.
- How to work through a multi-step problem from start to finish, with an example (and there will be more to come!).
- Some handy problem solving tools to help you along.

If you don't feel that you fully learned any of these concepts, feel free to go back through the course a second time! The wonderful part about online courses is that you can re-take them as many times as it takes to learn the content, then reference it later as you use those skills you picked up.

Your problem solving is going to be different from here on in, simply because you have a different perspective now. You don't have to jump right in and drown. There are answers right in front of us, if we just look for them.

I have a few parting words of advice for you in terms of problem solving:

NICOLE'S FINAL ADVICE

1. ENJOY THE PROCESS

Problem solving is fun, dude! I mean, it can be, if you learn to see it that way. You can train your brain to get excited and start breaking down the problem mentally upon first seeing it. You'll get comfortable asking for help and others' thoughts.

Enjoy the process—you won't get very far if you're not interested in what you're doing.

2. IT'S OKAY TO BE FRUSTRATED

You're going to get frustrated and overwhelmed sometimes when you don't know build your next step, or even completely understand the problem in general. It's okay, and it's normal to feel like you want to walk away or switch to a new problem.

Sometimes, that space will give you more clarity in the long run.

3. CONSIDER WHAT YOU MAY BE ASSUMING

Assumptions wreck us a lot of the time. We don't even realize we're making assumptions while solving a problem, but we are. We're relying on our gut sometimes, and that's great. But, as you go along, get in the habit of identifying where you might not quite be certain you used something correctly, or went the right path.

Not only will these areas be the best starting places while debugging, but you also have an opportunity to fix your knowledge and avoid confusion in the future.

4. FIND WHAT WORKS FOR YOU

Everyone is different, which means your problem solving approach will ultimately be different, too. Use the content in this course as a guideline, and contextualize my advice as it makes sense to you.

NOTES

NOTES

NOTES

NOTES

NOTES

@LAVIEENCODE

WWW.LAVIEENCODE.NET