**How to Think Like a Developer: Become a Problem Solver!**

In this lecture,

we're gonna talk about solving problems.

Solving problems is one of the most important things

in programming.

So if your goal is to become a great programmer,

you need to learn how to deal with problems effectively.

So in this video, I will show you how to think

as a developer by solving problems,

using a simple four step framework.

And let's use our friend John again.

So, after some time John can code now,

he is almost job ready at this point,

but he still needs to work on his problem solving skills.

And in programming, when we say solving problems

we mean a real problem that needs a real solution.

For example, in an array or GPS coordinates,

find the two closest to points.

And this problem is not straight forward, right?

There are a lot of steps involved in solving this.

And so we need a way to effectively solve problems

like this one, or even way more complex ones.

So problem solving does not mean to fix coding mistakes

or bucks, all right?

Anyway, whenever John encounters a problem

that he needs to solve,

he usually just jumps at the problem without much thinking.

He also implements his solutions in an unstructured way

without much of a logical approach.

This makes him stressed out

when things don't work right away.

Another problem that John has,

is that he is too proud to research

when he doesn't know how to come up with his own solution.

So, all of this is not effective at all,

and makes John waste a lot of time.

Now, the first fix to this is to always stay calm

and slow down and not just jump at a solution

without having a plan.

Also, when solving problems,

you need to be in a mindset

of taking a logical and rational approach,

after all programming is just logic, all right?

Then when you are in the right mindset,

just use my four step framework

to solve the problem at hand.

And the first step of the framework

is to make sure that you 100% understand the problem.

Step back and take a high-level look at the big picture.

And the most important part of the step

is to ask the right questions

in order to get a clear picture of the whole problem.

And the best way of understanding this,

is probably to look at an example together here.

So let's say you are working at some company

on some project,

and then your project manager comes and tells you,

"We need a function that reverses,

"whatever we pass into it."

And that's all you know about the problem,

but do you understand it 100%?

Well, probably not, right?

I know that I don't.

So let's ask the right questions to get there.

First we can ask,

what does whatever actually mean in this context?

So, what should actually be reversed here?

Well, and if we think about it,

it only makes sense to reverse strings,

numbers, and arrays.

Objects don't have a well-defined order,

so we can't reverse them.

We're also not gonna reverse like undefined,

or null or a Boolean, right?

Then after we have that figured out,

we can ask, what should we do if something else

is passed in that is no string, number, or array?

How are we gonna handle that?

It's also relevant to ask,

what exactly should be returned from the function?

For example, should it always be a string,

or should the tie be the same as was passed in?

Well, in this case, probably we should always

just returned the exact same type that was passed in.

Also we can start to ask more solution oriented questions,

like how to recognize whether the argument is a number,

a string, or an array?

Or how to actually it reverse a number,

a string and an array.

So these are just some of the questions

that we can ask here.

I'm sure there are more,

but this should already give us

a way more clear picture of this problem.

Now, right now, at the beginning of your journey,

you were probably not yet know,

what the right questions actually are, but don't worry,

this will come with time and practice I promise.

The next step is probably the most important one,

which is the divide and conquer strategy.

Divide and conquer means to break up the big problem

into as many small problems as possible

because these small problems are then a lot easier to solve.

And this also ties in with the previous step

because by dividing the problem into sub problems,

you also take the big problem apart,

until you understand it exactly.

So based on the previous step,

if we wanted to break down this problem,

we could define a couple of sub problems.

First, we need to check if the argument

is a number, a string, or an array.

And this is in fact, a small sub problem, right?

So now, and just solve this one

in isolation and then move on.

The next sub problem is to implement reversing a number,

then implement reversing a string,

and then implement reversing an array,

so that we are ready to deal

whatever is passed into the function.

Finally, we then also need of course

to return the reversed value.

So these are our sub problems,

and they kind of look like a task list

that we now need to go ahead and implement.

And this is great because it makes our work

so much easier now.

Now, sure, this example is a pretty simple problem,

but this strategy works just as well

for bigger and way more complex problems.

In fact, divide and conquer isn't essential method

of problem solving that is also used a lot

outside of programming.

Anyway, with this, we have actually already covered

the two most important steps of the framework.

But now let's say that we actually do not know

how to implement one of these sub problems.

Well, in that case, we should not be afraid

of doing as much research as we have to.

Now, of course, we should always first try

to implement a sub problem on our own

using our own coding abilities.

But if we're constantly hitting a wall,

and cannot move on, then we should waste no more time

and just find out how it works using Google,

or a questions and answers websites

like Stack Overflow, or the MDN JavaScript documentation.

We will see how to use these in the next lecture

when we're gonna work together on a real problem.

And actually researching is a huge part

of a programmer's job, as you will find out very soon.

So you really should not be ashamed

when you don't know enough to solve a certain problem.

Let's just completely normal.

Now, in this case some questions that we could Google are,

how to check if a value is a number,

or an array, or a string in JavaScript.

Or we could Google how to reverse a number,

or a string, or an array in JavaScript, okay?

And again, we will do this in the next lecture

on a real problem together.

Finally, in case we are trying to solve bigger problems

we should put all the previous steps together

and write some so-called pseudo-code

before writing the actual code.

And pseudo-code is simply an informal description

of the actual code that we're gonna write.

So it's like code for humans to understand not computers.

For our example, the pseudo-code

might look something like this.

So you see we use some structures of a programming language,

but this is clearly not JavaScript, right?

And there are no real rules

on how to write pseudo-code either,

you just write so that you understand it yourself,

or other people on your team.

So in this example, we just start

by naming the function as a reverse,

and then we basically pass in a value.

Then we say, that if the type of the value

is not a string and not a number,

and also not an array, then simply return the value back.

And that's because as we defined previously,

we only want dysfunction to work for strings

or for numbers or for a race.

And again, you can clearly see

that this is not any valid JavaScript at all.

But anyway, if the type of the value past this initial test,

then we say if the value is a string,

then reverse the string.

If the value is a number, then reverse the number.

And if the value is an array,

then simply reversed the array.

And then at the end,

we want to return to reversed value, okay?

So this is really more like a list of instructions

that we now could go ahead and actually implement

using the JavaScript language.

Okay, and that's it.

That's the four steps to solve any problem

no matter how big.

And now before I leave you, a final tip I can give you,

is to develop a genuine curiosity and passion

for understanding how things actually work,

not only in programming,

but really in the whole world around you.

For example, the famous scientist Richard Feynman,

was repairing radios when he was just a teenager,

and Elon Musk made his first computer game when he was 12.

And there are many more examples like this,

and all of them became geniuses in their fields

in part because of their genuine curiosity

on how the world around them works.

All right, and with that being said,

let's now go ahead and solve a real problem

in the next lecture using this framework

that we just learned about.

So I'll see you there.

