In the last video, we created a lot of helper functions. We put them in a snippet, so we have a lot of fairly loose functions.

Now, one of the issues with this is, if my application had a function called toggleAll, and I ran this, I would be overwriting that code. So, we want to have the ability where we can write functions that help us, but make sure they don't interfere with the application itself.

**To do that, we have to learn a little bit about objects.**

I'm just going to create a new snippet here, rename “vanilla object”, just so that we can learn a little bit.

Now, objects in JavaScript, are really just functions. So, I'm going to create a variable, I'm going to call it “autoToDo” and it's going to be an instantiated object. I'm going to say new function.

var autoTodo = new function() {

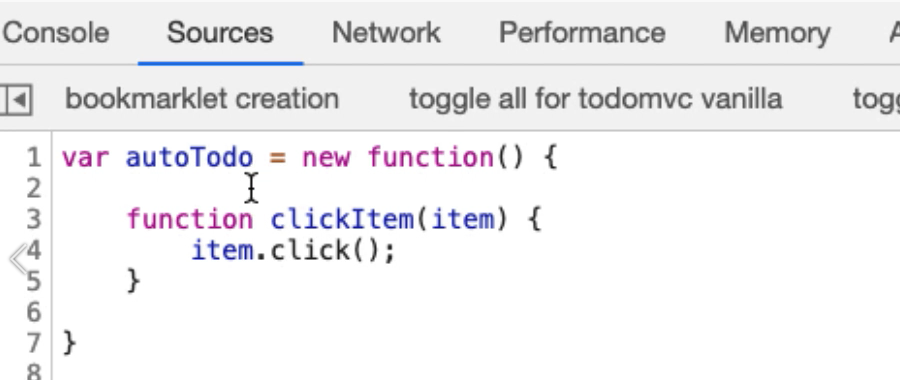
}

Now, at this point, it doesn't do anything, but if I run this, go to the console and I say autoTodo, it just tells me, it’s a function.

I really want to have methods in there. These functions that we've written, I want those to be in my autoTodo.

How do I do that?

I'm going to copy and paste this just to start with.



Do I now have a clickItem function that I can call from autoTodo?

autoTodo.clickItem() — clickItem is not a function, so I haven't managed to create an object with functions.

Now, this is valid code, it's just that this is a private function, so this would run inside autoTodo itself, it's not accessible by anyone else.

To make this a public function, I basically say:

this.clickItem = function(item) {

item.click();

}

If I run this — Console > clickItem(document.querySelector("#toggle-all")); — pass in an object, a web element, and we clicked on it.

Now my clickItem function is public. What we've got here, is we have essentially a namespace, we've got an object that all these functions are on.

If for some reason this application had an object called autoTodo, I could call this “myAutoTodo”. I don't have to rewrite all my code because its wrapped in here. And because it's instantiated when I run this, we're ready to go.

**Useful Tip**

I often use this when I'm creating repeatable code when I'm interacting with the application from the command line, and the code will typically be stored in a snippet so that I can run it. And also, I will have backed up the snippet on a hard drive just in case Chrome does an upgrade and forgets my snippets.

My next action then, really is to take all the helper functions and put them on my actual object.

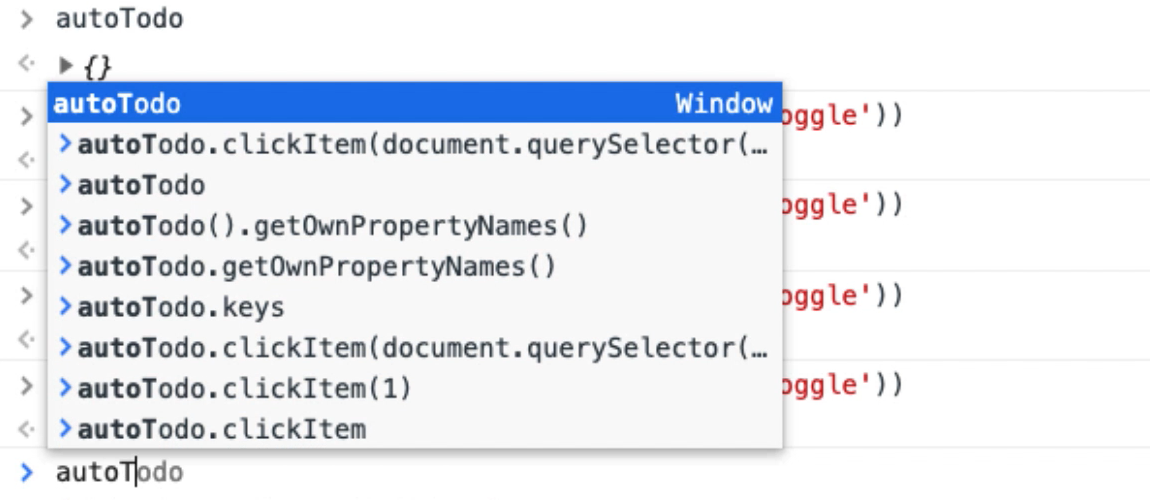
And you saw how I did that. I basically took the function name and then just said “this.functionname = a function”.

this.toggleAll = function() {

document.querySelector('#toggle-all').click();

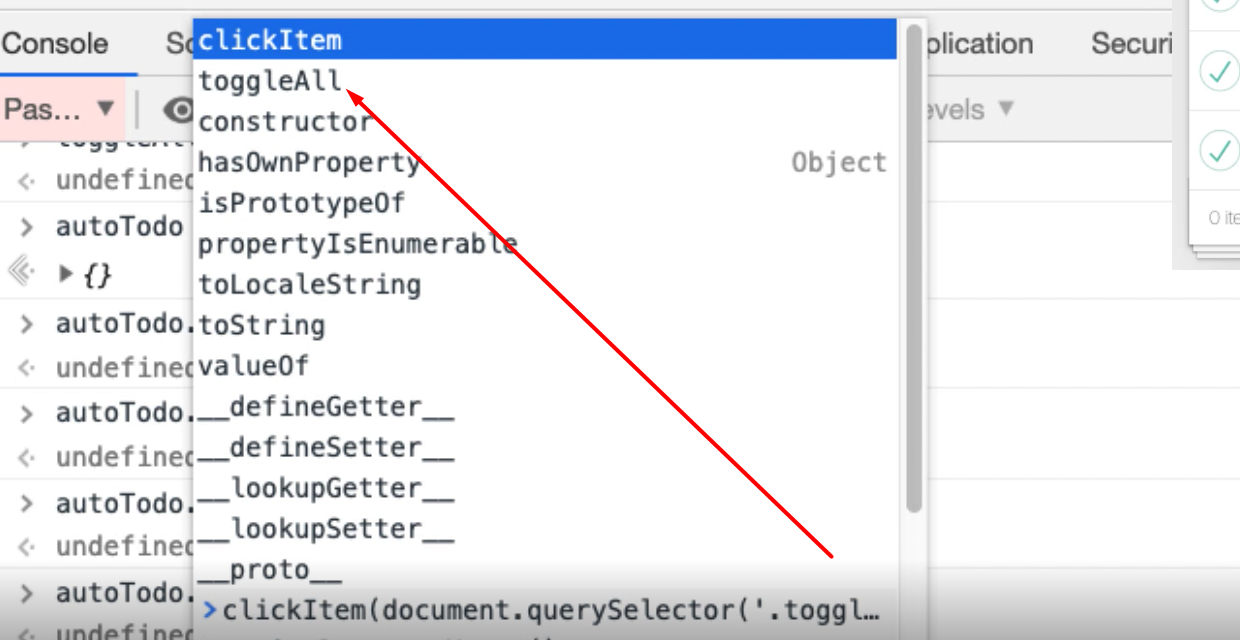
}

Now the toggleAll, should be on.



And you can see that in the code completion I've now got the functions that are available to me.

I didn't write these, these are coming from JavaScript, what's important for me is the toggleAll.



Alright, so I can just keep going ahead and convert these across.

They all take the same pattern, where, instead of it being function, I say: this.thefunctionname = thefunction.

I'm just going to carry along doing this and then I'll explain the code when we're finished. You've pretty much seen everything I'm going to do.

var autoTodo = new function() {

this.clickItem = function(item) {

item.click();

}

this.toggleAll = function() {

document.querySelector('#toggle-all').click();

}

this.selectItemX = function(x) {

document.querySelector("ul.todo-list > li:nth-child(" + x + ") input.toggle").click()

}

this.deleteItemX = function(x) {

document.querySelector('ul.todo-list > li:nth-child(' + x + ') button.destroy').click()

}

this.clearCompleted = function() {

document.querySelector('button.clear-completed').click();

}

this.filterCompleted = function() {

location.hash = "/completed";

}

this.filterAll = function() {

location.hash = "/";

}

this.filterActive = function() {

location.hash = "/active";

}

this.createTodo = function(name) {

document.querySelector('input.new-todo').value = name;

document.querySelector('input.new-todo').dispatchEvent(new Event('change',{

'bubbles': true

}));

}

this.amendTodo = function(x, amendedValue) {

document.querySelector('ul.todo-list > li:nth-child(' + x + ') > div > label').dispatchEvent(new Event('dblclick',{

'bubbles': true

}));

document.querySelector('ul.todo-list > li:nth-child(' + x + ') .edit').value = amendedValue;

document.querySelector('ul.todo-list > li:nth-child(' + x + ') .edit').dispatchEvent(new Event('blur'));

}

}

That was exciting, a lot of copy and pasting and editing, so what have we got here?

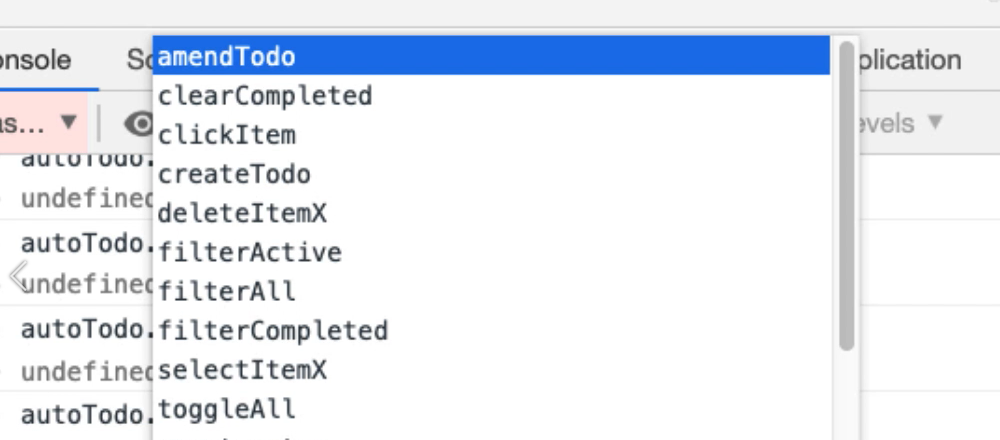
We have got an object called autoTodo. It’s an object, an instantiated object, so new function has created a new instantiate function.

I say this, which is the object, the autoTodo instantiated object, this.clickItem function is a function that takes an “item”, click.

This toggleAll, is a function which does this activity.

And all I've done is copy and pasted in all the helpers and amended them to be actual functions that are accessible on the object. You can see the code associated with this lecture.

Now, in the console, if I do autoTodo., I have access to all these different functions.



I can filterCompleted.

Which it was on. Good choice!

I can autoTodo.filterActive, and there's none. Let's create one — “hello” — there we go.

What I should have done, I'll delete that, autoTodo.deleteItemX(1).

See I don't need to use the GUI at all now.

autoTodo.createTodo(“jings”) — here we go.

So, my autoTodo is going to be pretty helpful when I start writing a bot, because it represents all the functionality that we've been using to interact with the application to do things.

All the code should be associated with this lecture, feel free to paste into snippets, create your own snippets and use these functions.

We're going to start pooling all this together and creating bots that will automatically interact with the application and automate it properly from the console. Because what we've got now are some tactical functions that can help us. And we can keep writing new functions to do whatever we want, but it will be useful to have functions that create a whole bunch of test data for us.

Let's figure out how to do that.

**Quiz: +100 credits**

**1. A function in JavaScript:**

makes it easy to organise code

creates a readable abstraction layer for automating

can take parameters

all of the above

**2. If I have a collection of things in a variable I can iterate over all items in the collection and call a function clickItem by writing:**

things.forEvery(clickItem)

things.forEach(clickItem(this))

things.forEach(clickItem)

clickItem(things).forEach()

**3. I can join strings using what syntax?**

"start " & "middle" & " end"

"start " + "middle" + " end"

"start " . "middle" . " end"

**4. A function that takes a parameter has which of following syntax?**

function withParam(){console.log(theParam)}

function withParam(theParam){console.log(theParam)}

function withParam(:theParam){console.log(theParam)}

var theParam="hello";function withParam(){console.log(theParam)}

**5. Creating objects allows us to:**

create an isolated 'namespace' to avoid interfering with the application under test

organise our functions into even more readable abstraction layers

create 'private' functions in the object that no-one else can use

all of the above

**6. Converting a function to a public function on an object is:**

very complicated and requires a lot of JavaScript knowledge

pretty simple once we realise it is copy paste with a few amendments

Submit