"Indirect" vs "Direct" Function Execution - Summary

It can be confusing to see that there seem to be two ways of executing a function:

```
1. function add() {
2.   something = someNum + someOtherNum;
3. }
```

`add()` vs `add`

It's important to understand why we have these "two ways"!

In general, you call a function that you defined by **using its name** (e.g. `add`) and **adding parentheses** (with any parameters the function might need - or empty parentheses if no parameters are required like in the above example).

=> `add()`

This is how you execute a function from your code. Whenever JavaScript encounters this statement, it goes ahead and runs the code in the function. Period!

Sometimes however, you **don't want to execute the function immediately**. You rather want to "tell JavaScript" that it should execute a certain function **at some point in the future** (e.g. when some event occurs).

That's when you don't directly call the function but when you instead just provide JavaScript with the name of the function.

=> `someButton.addEventListener('click', add);`

This snippet would tell JavaScript: *"Hey, when the button is clicked, go ahead and execute add."*.

`someButton.addEventListener('click', add());` would be wrong.

Why? Because JavaScript would encounter that line when it parses/ executes your script and register the event listener AND immediately execute add - because you added parentheses => That means (see above): *"Please execute that function!"*.

Just writing add somewhere in your code would do nothing by the way:

```
1. let someVar = 5;
2. add
3. alert('Do something else...');
```

Why?

Because you just throw the name of the function in there but **you don't give any other information to JavaScript**. It basically doesn't know what to do with that name (*"Should I run that when a click occurs? After a certain amount of time? I don't know..."*) and hence JavaScript kind of ignores this statement.