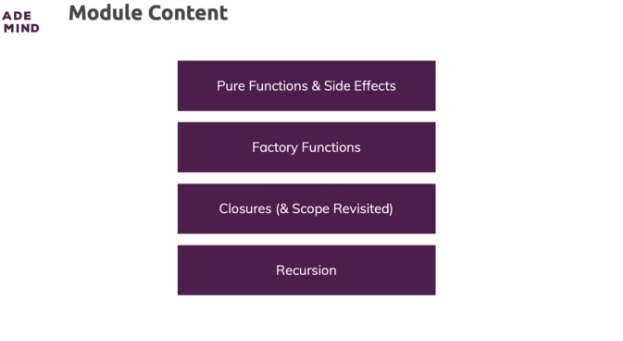
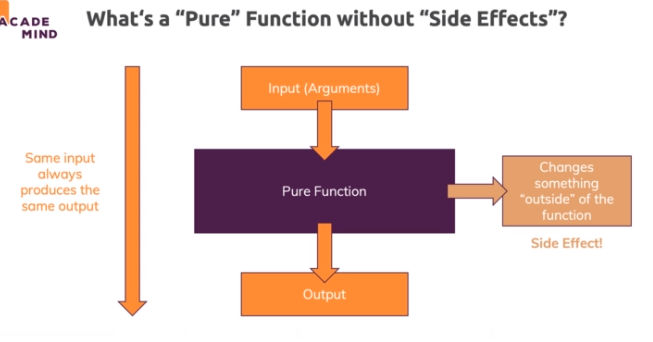
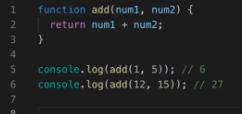
**Advanced Function Concepts**

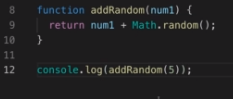


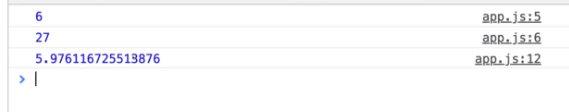
**Pure Functions & Impure Functions**







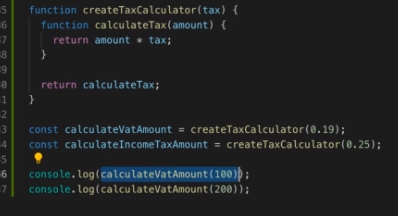


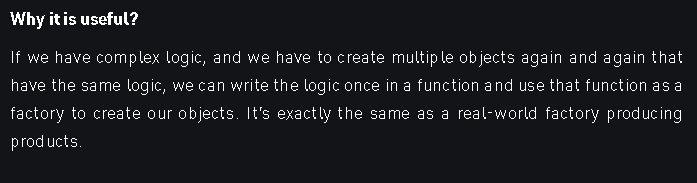


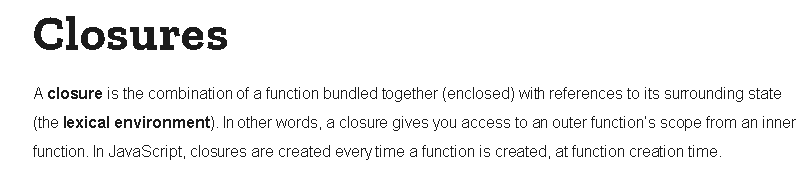
* A function is also not to be considered pure if it introduces side effects

**Factory Functions**

* factory functions is that you have a function that produces another function
* if you needed that amount in different parts of your software? Then you need to call the functions with these tax rates in different parts of the software.









https://paper-attachments.dropbox.com/s_AA3FD6DAB2508AF77F7C4CED7266072CD8E9546A6B6EADAD5FCAFA05F4F5EB7E_1632499506096_screenshot-www.udemy.com-2021-09-24-21-34-59-365.png

**Memory Management**

* If every function logs in all surrounding variables, doesn't that lead to a pretty bad effect in our memory because in big applications where we have many variables, a function might log in a lot of variables which it never uses.
* modern Javascript engines optimize this.
* They basically track variable usage you could say and if a variable obviously isn't getting used course in a safe way so that they don't accidentally crash your program because you need to use that function at some point of time

**Optional: IIFEs**

In JavaScript - especially in older scripts - you sometimes find a pattern described as "**IIFEs**". IIFE stands for "**I**mmediately **I**nvoked **F**unction **E**xpression" and the pattern you might find looks like this (directly in a script file):

    1. (function() {

    2.     var age = 30;

    3.     console.log(age); // 30

    4. })()

    5.

    6. console.log(age); // Error: "age is not defined"

What's that?

We see a function expression which calls itself (please note the () right after the function).

It's NOT a function declaration because it's wrapped in () - that happens on purpose since you can't immediately execute function declarations.

**But why would you write some code?**

Please note that the snippet uses var, NOT let or const. Remember that var does **NOT use block scope** but only differ between global and function scope.

As a consequence, it was hard to control where variables were available - variables outside of function always were available globally. Well, IIFEs solve that problem since the script (or parts of it) essentially are wrapped in a function => Function scope is used.

**Nowadays, this is not really required anymore**. With let and const we got block scope and if you want to restrict where variables are available (outside of functions, if statements, for loops etc - where you automatically have scoped variables since these structures create blocks), you can simply wrap the code that should have scoped variables with {}.

    1. {

    2.     const age = 30;

    3.     console.log(age); // 30

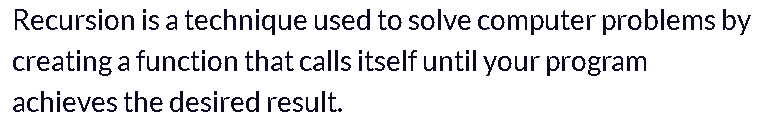
    4. }

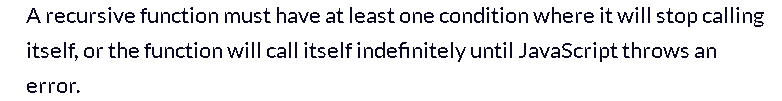
    5.

    6. console.log(age); // Error: "age is not defined"

Not something you see too often but something that is possible.

**Recursion**





* The condition that stops a recursive function from calling itself is known as **the base case**.

