ACADEMIND

What's It About?

Scope

```
const myName = 'Max';

function greet() {
  const age = 31;
  console.log(myName);
}

greet();
console.log(age);
```

Variables are only accessible in some "places".

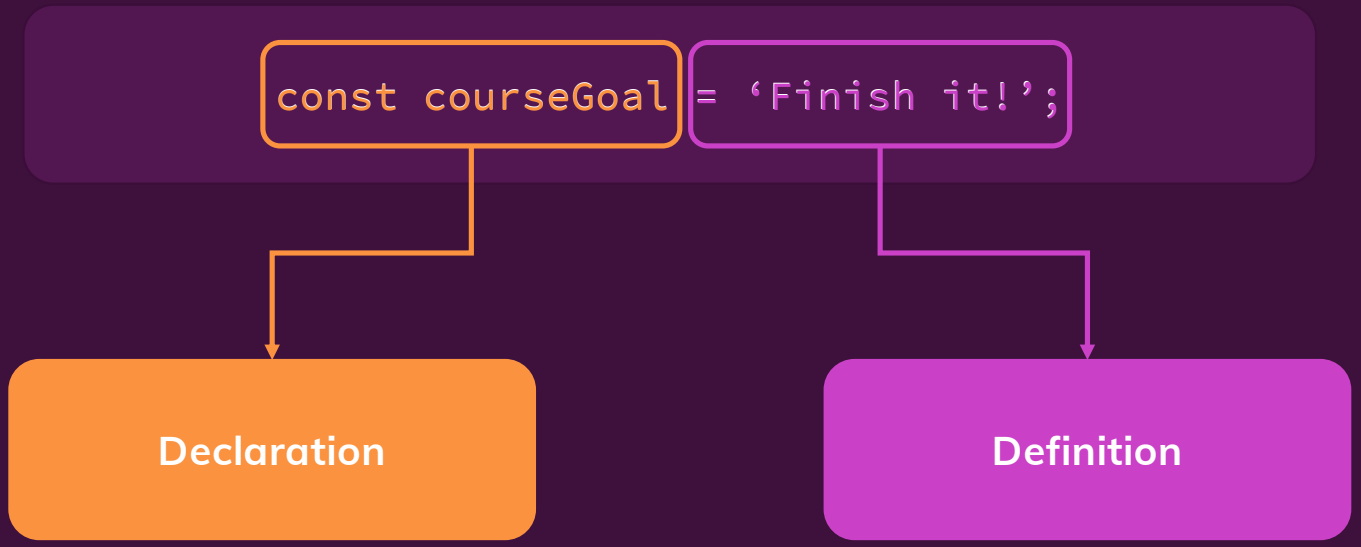
Hoisting

```
const myName = 'Max';

greet();

function greet() {
  console.log(myName);
}
```

Code order does not always dictate "execution order".

ACADEMIND

Declaration & Definition



ACADEMIND

Scopes & Scope Rules

var

Local Scope (Function Scope)

Variables defined inside of functions

Global Scope

Variables defined globally

let, const

Block Scope

Defined inside of {}

Don't confuse with JS objects

Variables in outer scope are accessible from inner scope but not the other way around.

ACADEMIND

What is “Scope”?

Scope is about the “**Visibility of Variables**”

You can only use variables that are visible in the place (i.e. line of code) where you are using them

ACADEMIND

Scope vs Context

Scope is about the "**Visibility of Variables**"

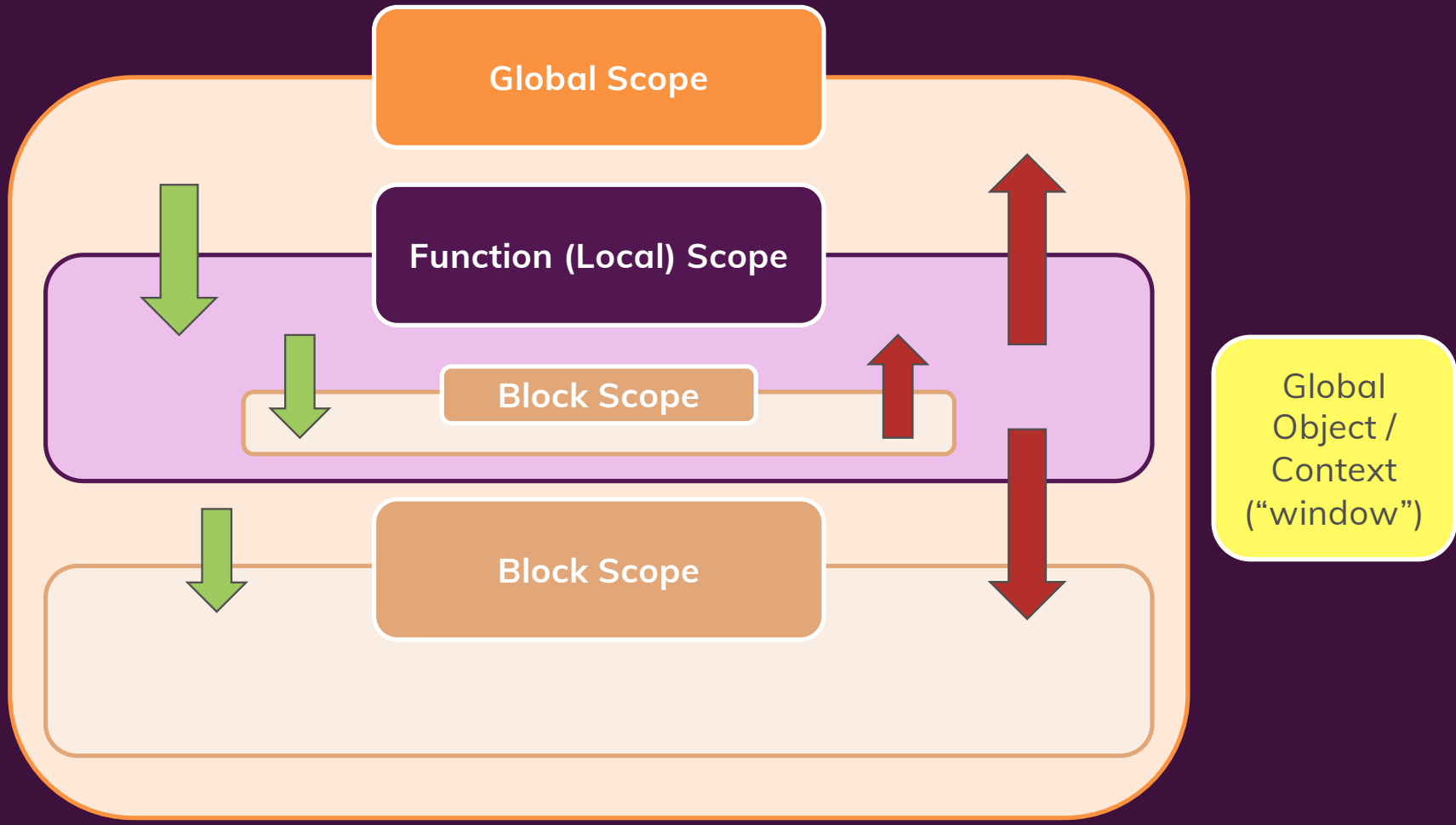
This Module

Context refers to the "**Object a Function belongs to**"

Check the "this" Module

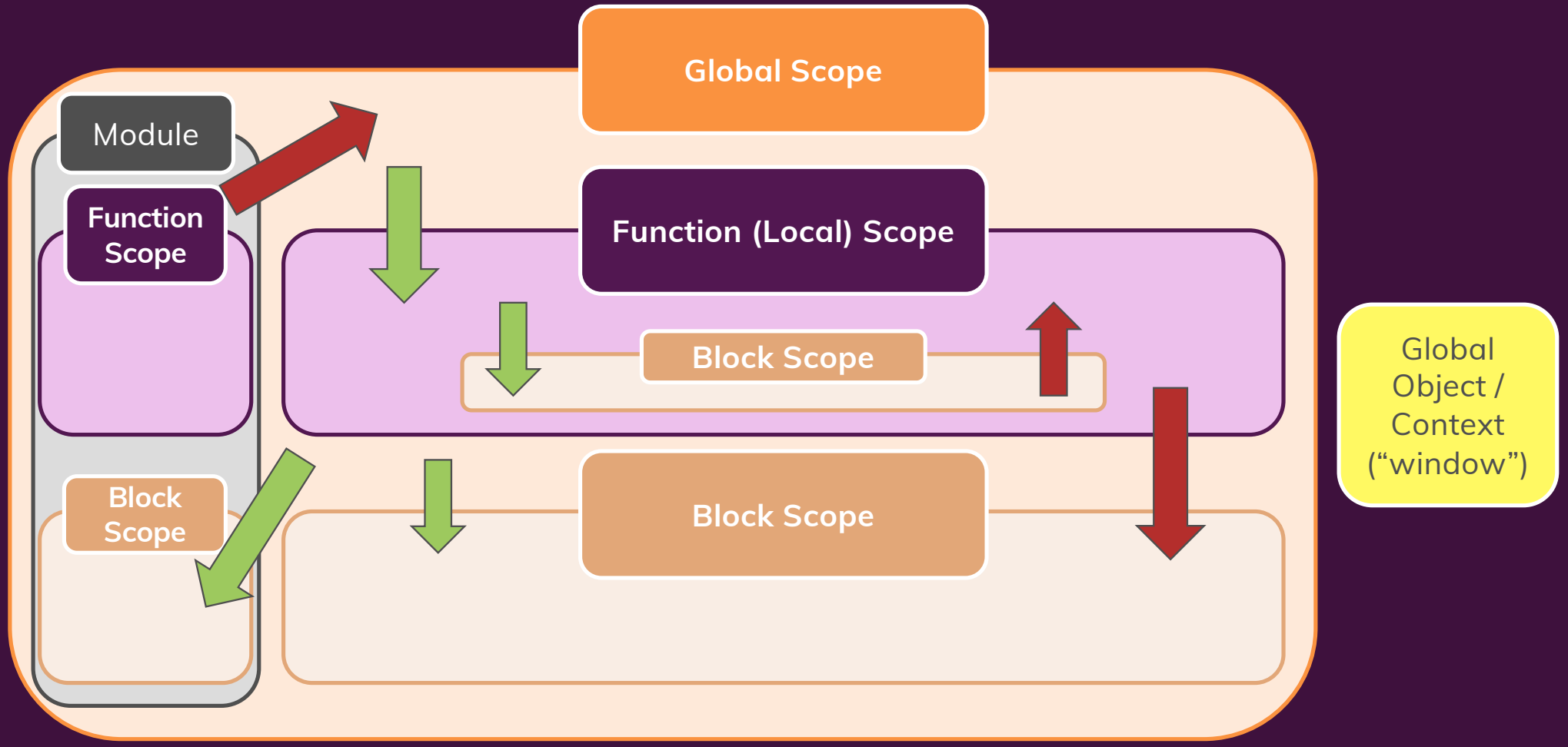
ACADEMIND

Big Picture



ACADEMIND

Big Picture

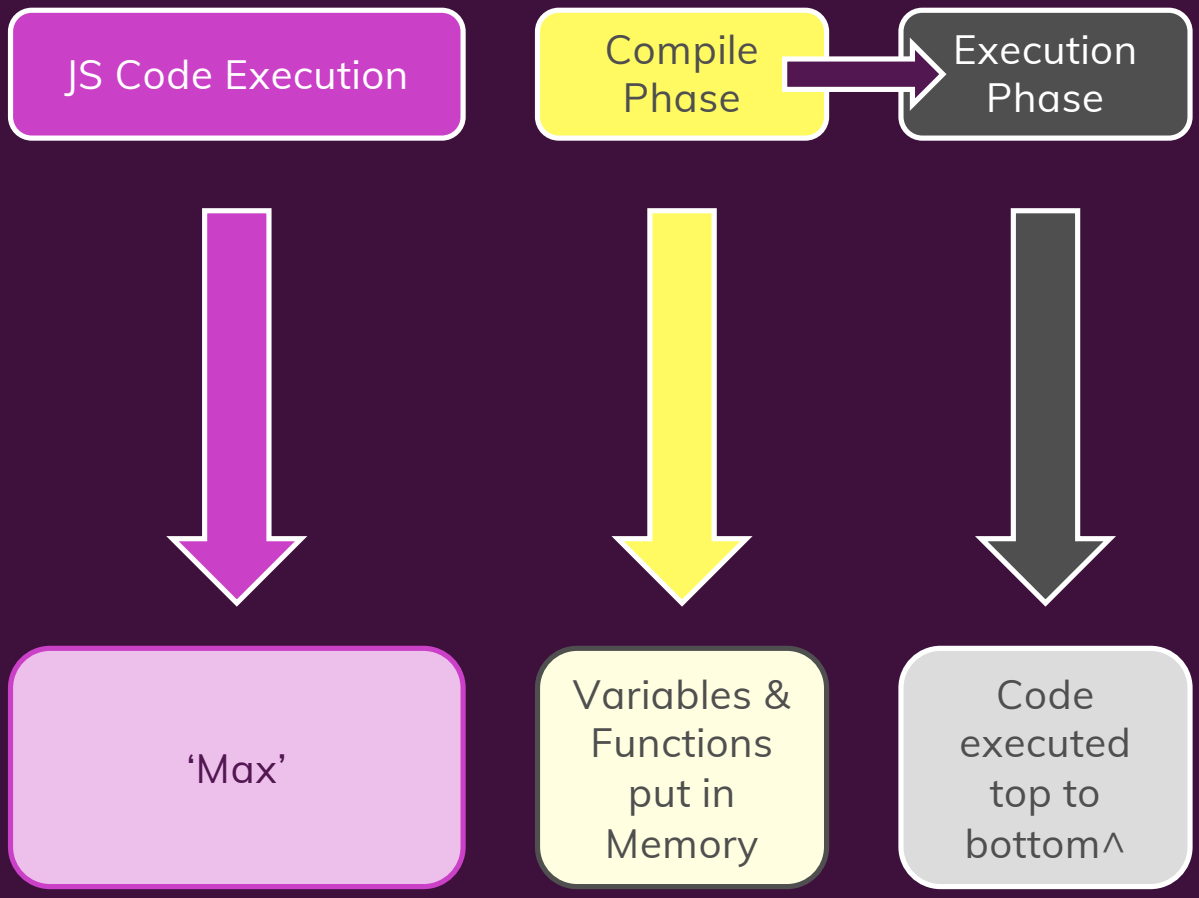


ACADEMIND

Hoisting?

app.js

```
const myName = 'Max';

greet();

function greet() {
  console.log(myName);
}
```



ACADEMIND

Summary - Scope

"Scope" is all about the "**visibility of variables**": **Which variable** can be **accessed where** in your code?

JavaScript "knows" **three types of scope**:

- Global Scope (for global variables)
- Function (Local) Scope (for function variables with `var`) and
- Block Scope (for block variables with `let` or `const`).

A "**block**" is code between curly braces (`{}`).

There is **one global scope** which is **shared across files**. **Modules** (`type="module"`) do have access to that scope but do **not share their variables** with it.

ACADEMIND

Summary - Hoisting

"**Hoisting**" is a concept which describes the process of "**memorizing**" **function and variable declarations** (**not definitions!**) by the JavaScript engine **before the script code is actually executed**.

All **function declarations** (`function someName() { … }`) are hoisted. This **includes** their actual function body.

For **function expressions** (incl. arrow functions) **only the declaration** (not the body) is hoisted.

For **variables**, **only the declaration** is hoisted.

Hoisted code is **not "pulled to the top of the file"** (though that is kind of a mental model you can have about the process). It's simply memorized by the engine instead.