Asynchronous Programming: server does not wait for a certain line of code to respond before moving on

synchronous: total time = sum of both times; asynchronous: total time = length of longer code

ex: axios for fetch request

good for I/O, heavy computations, etc.

After request for promise, moves on, response from promise fed into callback

Promise: Object that may produce a value at some point in value

Pending
Initial state

Settled
Can **fulfilled** or **rejected**

good for keeping UI responsive while running APIs

contagious — if one function is async, functions that call it are also async

Control-Flow Function
executed between async calls

1. Figures out order of execution and data needed
2. Limits concurrency so only one tasks occurs at a time
3. Invokes next step in code

error-first callback function = error object is first parameter, written to handle this object first

Use a **callback function** to handle rejections (.then())

function passed into another function, called by outer function.

do NOT use paranthesis, or will invoke immediatly!

Promise.all() used when want all promises to start concurrently, rejects if even one rejects

If use for loop and async-await, total time is sum of each instead of just the time it takes for longest promise to run

like calling Promise.resolve().then()

await is a good way to make code more synchronous

Async-Await is the more modern way of writing async code

syntatic sugar over promises

Callback Hell
(Pyramid of Doom)

Intensively nested callbacks, result of improper implementation of async code.
Fixes:
- name functions to make code and stack traces easier to read
- modularize code
- handle errors for better debugging

PROS:
- less nested code because don't need anonymous functions inside of your .then()
- can use try/catch statements
- reduced code clutter

when promise fails, catch is not triggered because did not occur in try block

CONS:
- can create callback hell
- harder to debug

when async code met, **event loop** is executed

event ex: onClick, onDone, onLoad

adds events from callback queue to call stack

Core Language

Equality

"**equality**" ==	"**strict equality**" ===
Convert all data types to string '1' == 1	Data types are considered '1' !== 1

object initialization actually references point in memory, so two empty objects are not equal unless reference same point in memory

deep copy = copy entire object to new memory address

template literal = `string ${expr}`

```
let x = 1;    global
func()
{
   x = 2;          local
   for (x in y)
   {
      let z = 1;    block
   }
}
```

shadowing = inner variable overshadows outer variable

immutable

let and **const** block-scope, **var** only local-scopes

types in JS: undefined, NULL, boolean, number, string, symbol

empty object, function, or array is truthy

falsy boolean values:
false
0
empty string
NULL
undefined
NaN — "Not a Number", falsy compared to even itself

can typecast to boolean with !!NaN = false

parse() = JSON to JS Object

stringify() = JS Object to JSON

JSON (JavaScript Object Notation): lightweight format for transporting data

to avoid accidentally defining global variables, put into strict mode so the global this is undefined

global context = window object

'**this**' refers to block of code currently executing (closest parent object)

can use '**bind**' to bind the this of two objects

in JS, **class** is blueprint for creating an object, constructor called on initialization

closures are a way to make global variables local

have an outer function that returns an anonymous function, so can't access the anon function without going through the outer function

variable types are not known at compile time

JavaScript is **dynamically typed**

faster development, but if misspell a variable, will get no warning

polymorphism: objects share an interface but certain functions may differ **inheritance**: objects inherit from others (use Object.call(this, args to inherit))

OOP (Object Oriented Programming): use self-contained objects and classes as building blocks

Typescript = strict superset of JS with optional static typing

Babel converts ES6+ code into backwards compatible forms

Hoisting: declarations and assignments of variables and functions are separated

only declaration hoisted

classes are not hoisted

entire definition is hoisted

EJS (Embedded JavaScript): template to generate HTML with plain JS