



JONAS.IO

SCHMEDTMANN

Subscribe here

THE COMPLETE COURSE
JAVASCRIPT



TABLE OF CONTENTS: THEORY

- 1 Watch before you start!
- 2 A Brief Introduction to JavaScript
- 3 Data Types
- 4 Boolean Logic
- 5 JavaScript Releases: ES5, ES6+ and ESNext
- 6 Functions Calling Other Functions
- 7 Reviewing Functions
- 8 Learning How to Code
- 9 How to Think Like a Developer
- 19 Summary
- 20 First-Class Functions
- 21 Closure
- 22 Data Structures
- 23 Summary
- 24 How they work
- 25 Event Properties
- 26 Efficiency
- 27 What is

ON

O

C

L

W

WW



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

SOME QUICK CONSIDERATIONS BEFORE

👉 This course is for all of you! So please don't worry about progressing too slow, or too fast for you. To make things easier, you can always pause and re-watch the course with slower or faster playbacks.



Awful, not what I expected



SOME QUICK CONSIDERATIONS BEFORE

You need to code along with me! You will learn
code YOURSELF!



SOME QUICK CONSIDERATIONS BEFORE

Try all the coding challenges! Try to do your best if you can't figure it out! Just rewatch the lesson and move on.



SOME QUICK CONSIDERATIONS BEFORE

If you want the course material to stick, take |



- 1) Board personality
 - use archetypes for both audience and board person
 - Align board personality to audience person if it doesn't have to be the same. We need to align board personality that solves their problem, but a personality that looks like the person that we want. Think of board as a game
 - Get an image that looks like the person

SOME QUICK CONSIDERATIONS BEFORE

If this is your first time ever programming, please consider everything at the beginning. Just don't think "I



SOME QUICK CONSIDERATIONS BEFORE

In the first sections of the course, don't bother
stress about efficient code, or fast code, or cl
understand the **WHY** later in the course.



SOME QUICK CONSIDERATIONS BEFORE

Before moving on from a section, make sure that we wrote, review your notes, review the project



The notebook contains two main sections:

1) Core message framework

- Core message: what you want your audience to understand about your brand will influence all messaging, marketing, identity, social media, etc.
- Communicates who brand is, beliefs, commitments, values, differentiators, etc.
- Binary: why you're differently marked portion, the value vein, like salt...
→ Define words and tone of voice the brand will use.

2) Brand personality

- Use analogies for both audience and brand personality
- Align brand personality to audience personality, but it doesn't have to be the same. We need to choose a personality that solves their problem, that they are looking for an image that looks like the personality for what we want. Think of brands as a person.
- Define words and tone of voice the brand will use.

```
10 // We listen  
click is sur  
11 // Checking  
document.querySelector()  
12  
13  
14  
15  
16  
17  
if (!guess  
document
```

SOME QUICK CONSIDERATIONS BEFORE

!?!? If you have an error or a question, start by trying it, check the Q&A section. If that doesn't help,

1



SOME QUICK CONSIDERATIONS BEFORE

- I recorded this course on a Mac, but everything work on your computer, it's NOT because you're



SOME QUICK CONSIDERATIONS BEFORE

Most importantly, have fun! It's so rewarding - frustrated, stop whatever you're doing, and co



Java



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

WHAT IS JAVASCRIPT?

JAVASCRIPT

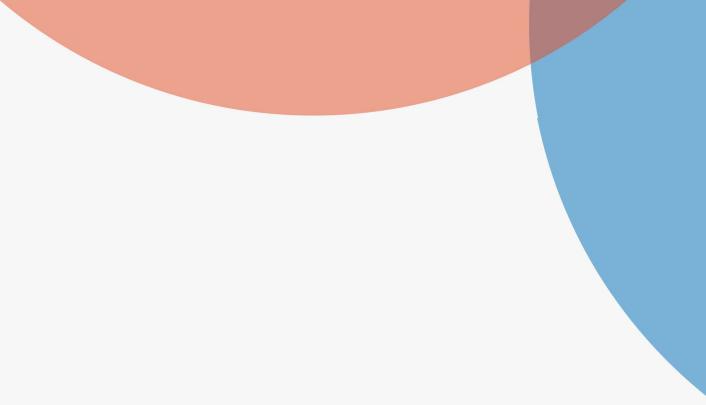
JAVAS

OBJECT-O

Based on objects, for
storing most kinds of data



THE ROLE OF JAVASCRIPT IN WEB



EXAMPLE OF DYNAMIC EFFECTS

The image shows a mobile application interface with a dark background. At the top, there is a navigation bar with icons for search, notifications, and account settings. Below this, a large green header features a circular profile picture of a man with brown hair and a beard, wearing a blue shirt. To the right of the profile picture, the name "Jonas Schmedtmann" is displayed in bold black text, followed by the handle "@jonasschmedtmann". Below the name, it says "Developer. Designer. On" and "Algarve, Portugal". To the right of the handle, there are statistics: "37 Following" and "19.4K F". A red diagonal line starts from the top right corner and extends downwards towards the bottom left, crossing over the profile picture and the stats.

Jonas Schmedtmann
@jonasschmedtmann
Developer. Designer. On
Algarve, Portugal
37 Following 19.4K F

Home Explore Notifications Messages Bookmarks Lists Profile More

1,317 Tweets

Show spinner + loading
data in the background

THERE IS NOTHING YOU CAN'T DO WHEN

Dynamic effects and
web applications in the
browser 😍

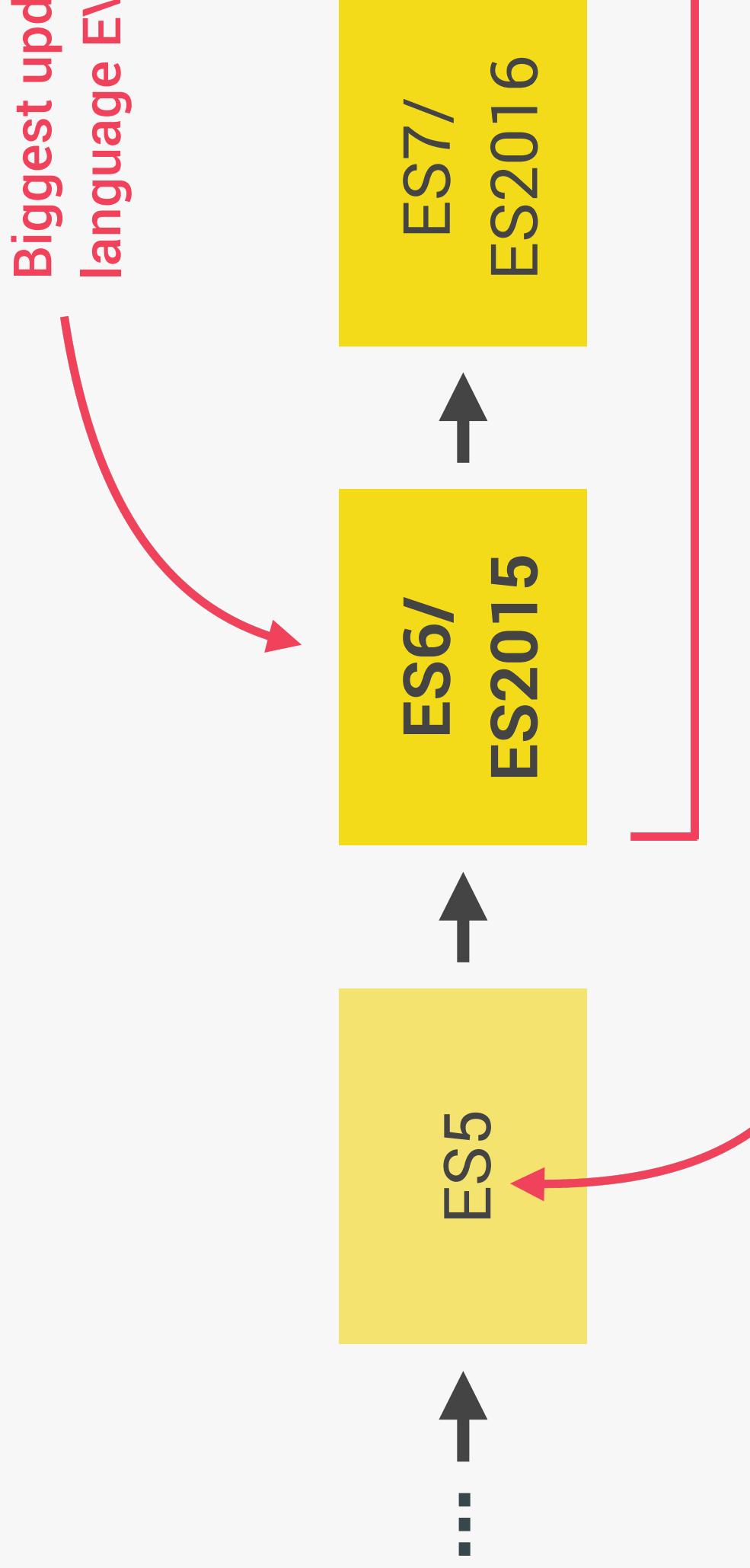


JS

THIS COURSE



JAVASCRIPT RELEASES... (MORE ABC)



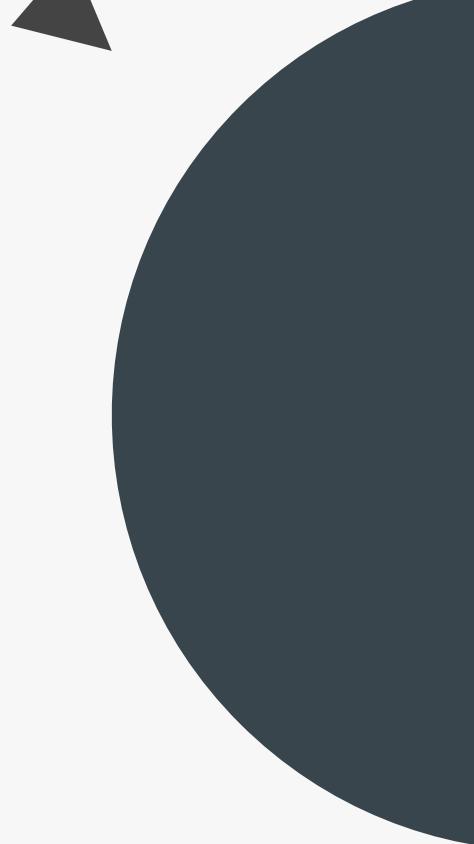


JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

OBJECTS AND PRIMITIVES



THE 7 PRIMITIVE DATA TYPES

1. **Number:** Floating point numbers 
2. **String:** Sequence of characters 
3. **Boolean:** Logical type that can only be 
4. **Undefined:** Value taken by a variable that has not been assigned
5. **Null:** Also means 'empty value'



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

BASIC BOOLEAN LOGIC: THE AND, OR

A AND B

*"Sarah has a driver's license
AND good vision"*

Possible values

A

| | | |
|-----|------|-------|
| AND | TRUE | FALSE |
| | | |

AN EXAMPLE



BOOLEAN VARIABLES

- 👉 A: Age is greater or equal 20
- 👉 B: Age is less than 30

LET'S USE OPERATORS!

👉 !A

true



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

A BRIEF HISTORY OF JAVASCRIPT

1995

👉 Brendan Eich creates the **very first** Mocha, but already had many funda-

1996

👉 Mocha changes to LiveScript and the

However, **JavaScript has almost no**

👉 Microsoft launches IE, **copying** Java

1997

👉 With a need to standardize the lang-

BACKWARDS COMPATIBILITY: DON'T



```
// ES1 Code
function add(n) {
  var x = 5 + add.arguments[0];
  return x;
}
```

1997

DO

HOW TO USE MODERN JAVASCRIPT

- ⌚ **During development:** Simply use the latest features.
- 🚀 **During production:** Use Babel to transpile back to ES5 to ensure browser compatibility.



ES5

👉 Fully supported

👉 Ready to help

MODERN JAVASCRIPT FROM THE BE

Learn modern JavaScript



But, also learn how some
(e.g. const & let vs val)

Java



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

CALLING A FUNCTION INSIDE A FUNCTION

```
const cutPieces = function() {
    return fruit * 4;
};

const fruitProcessor = () => {
    const applePieces = cutPieces();
    const orangePieces = cutPieces();
    const totalFruit = applePieces + orangePieces;
    return totalFruit;
};
```

8



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

FUNCTIONS REVIEW: 3 DIFFERENT F

👉 **Function declaration**

Function that can be used before it's declared



👉 **Function expression**

Essentially a function



FUNCTIONS REVIEW: ANATOMY OF /

Parameters values. Like

Function name

```
function calcAge(  
    const age = 20  
    console.log(`  
        return age;  
    
```

return statement to output a value from the function and terminate execution

о

и

ш

е

ш

д



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

HOW TO FAIL AT LEARNING HOW TO FAIL



He did
He did
He did
He did

code

code

code

code



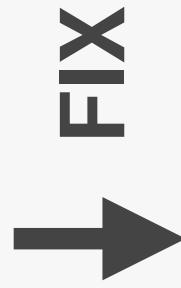
HOW TO SUCCEED



AT LEARNING



He didn't have a clear goal at the beginning of his journey



👉 Set a specific, measurable, realistic and time-based goal



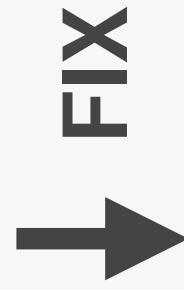
👉 Know exactly why you are



HOW TO SUCCEED AT LEARNING



**He didn't practice coding, and
didn't come up with his own
project ideas**



Practicing on your own is the
most important thing to do

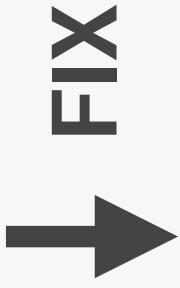


HOW TO SUCCEED



AT LEARNING

He was learning in isolation

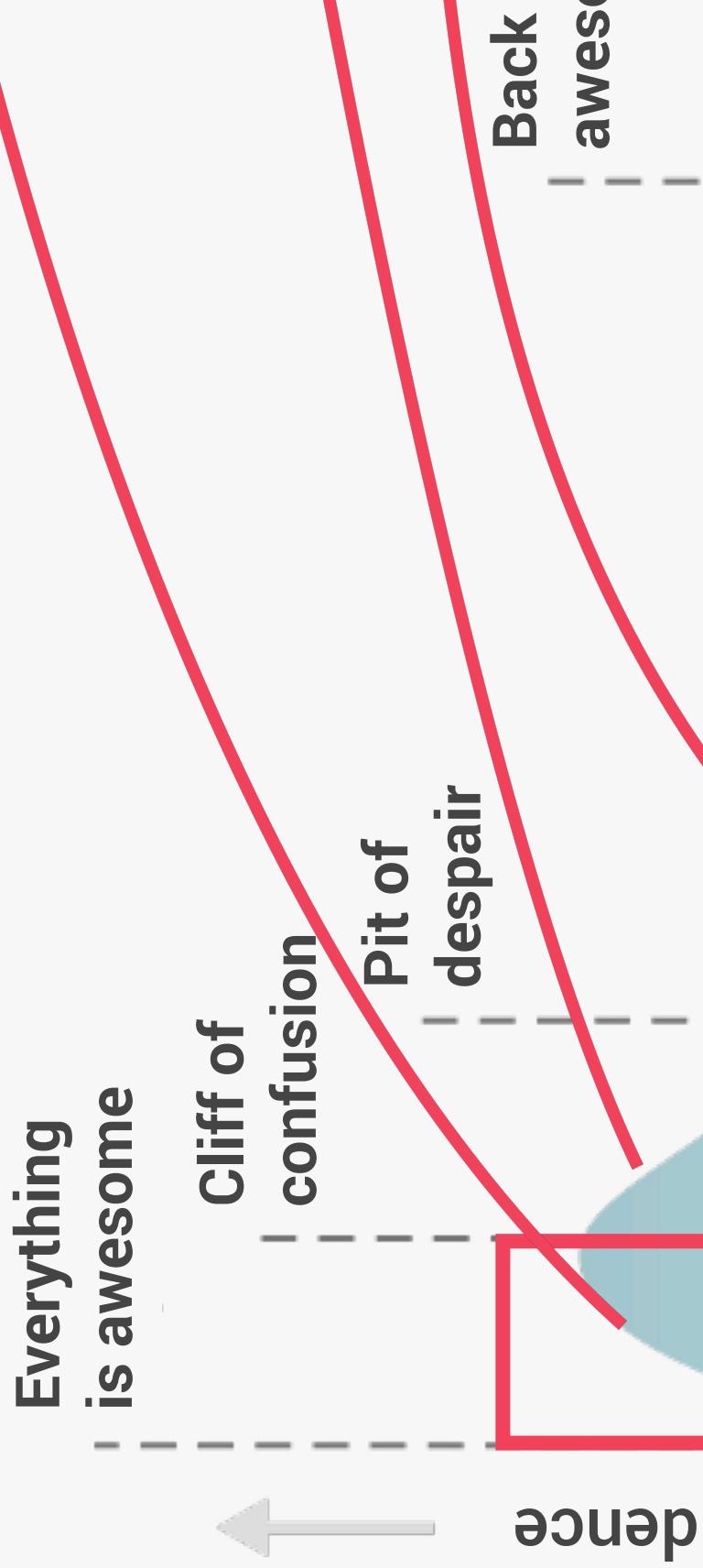


👉 Explain new concepts to people. If you can explain you truly understand it!



Share your goals to make

LEARNING HOW TO CODE IS HARD,



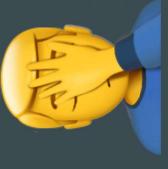


JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

HOW TO FAIL AT SOLVING PROBLEMS



WHENEVER

He ju



He im



He ge



He is



4 STEPS TO SOLVE ANY PROBLEM

Make sure you 100% understand the problem. Ask the right questions to get a clear picture of the problem

1

4 STEPS TO SOLVE ANY PROBLEM

Make sure you 100% understand the problem. Ask the right questions to get a clear picture of the problem



Divide and conquer: Break a big problem into smaller sub-problems.

1

2

4 STEPS TO SOLVE ANY PROBLEM

Make sure you 100% understand the problem. Ask the right questions to get a clear picture of the problem



Divide and conquer: Break a big problem into smaller sub-problems.

1

2

4 STEPS TO SOLVE ANY PROBLEM

Make sure you 100% understand the problem. Ask the right questions to get a clear picture of the problem



Divide and conquer: Break a big problem into smaller sub-problems.

1

2



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

WHAT IS A SOFTWARE BUG?

- 👉 **Software bug:** Defect or problem in a code. Basically, any unexpected or unintended computer program is a software bug.
- 👉 Bugs are completely normal in software

👉 Previous example: “We need a function that

THE DEBUGGING PROCESS

Becoming aware
that there is a bug

IDENTIFY



Is
ex-
ha-

FIND



JAVA
JAVA
JAVA
JAVA

m
r
o
w
s
u
d



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

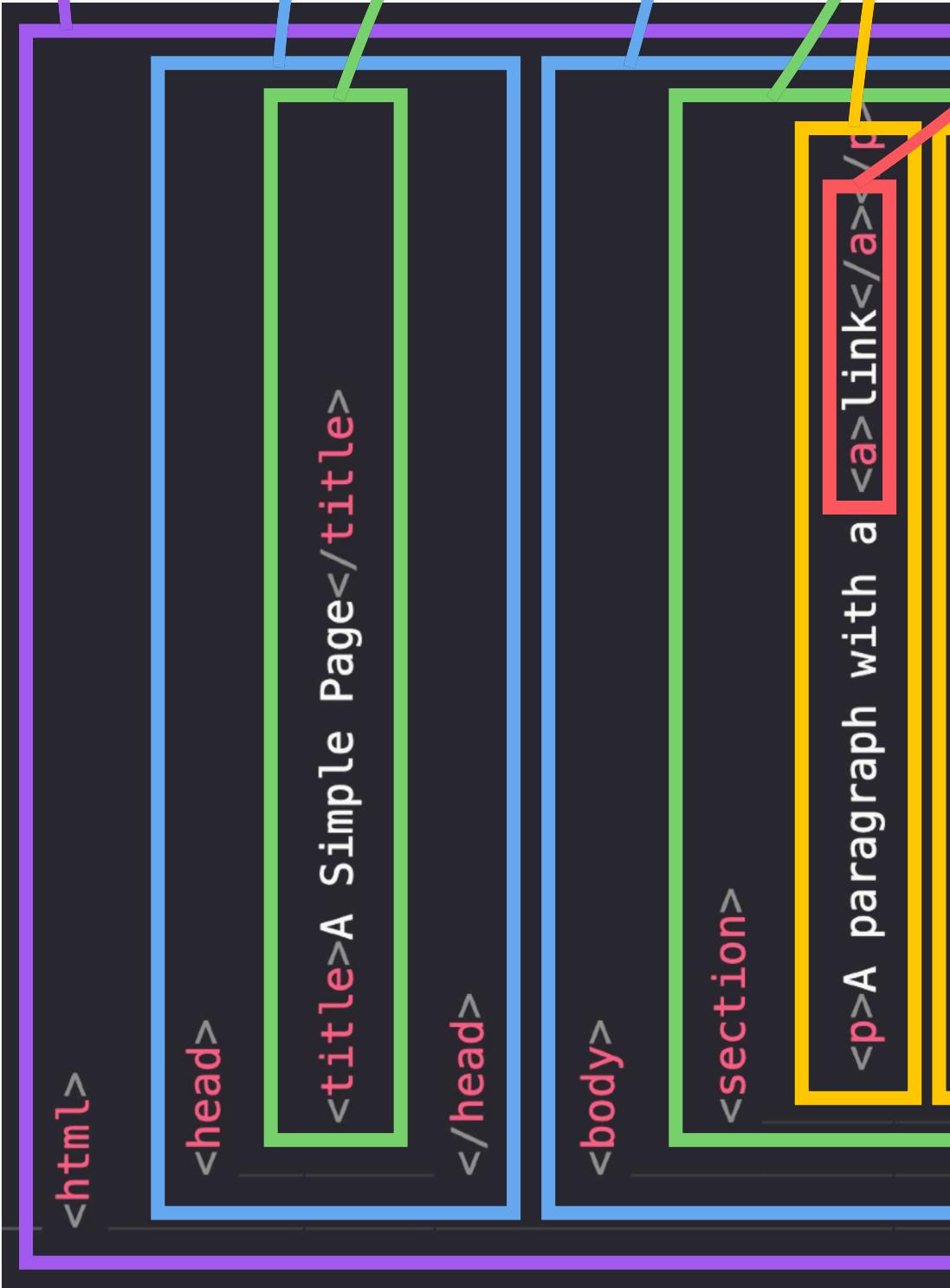
WHAT IS THE DOM?



DOCUMENT OBJECT MODEL

REPRESENTATION OF HTML DO

THE DOM TREE STRUCTURE



DOM ==> JAVASCRIPT



DOM Methods and Properties for DOM Manipulation



For example

`document.querySelector()`

API: Application Program

now
to
you



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

WHAT IS JAVASCRIPT: REVISTED

JAVASCRIPT

JAVAS

OBJECT-O

WHAT IS JAVASCRIPT: REVISITED



JAVASCRIPT IS A HIGH-LEVEL

MULTI-PARADIGM INTERFACE

DYNAMIC SINGLE-THREAD

DECONSTRUCTING THE MONSTER DEFINITION

High-level

Garbage-collected

Interpreted or just-in-time compiled

Multi-paradigm

Prototype-based object-oriented

DECONSTRUCTING THE MONSTER DEFINITION

High-level

Garbage-collected

Interpreted or just-in-time compiled

Multi-paradigm

Prototype-based object-oriented

DECONSTRUCTING THE MONSTER DEFINITION

High-level

Garbage-collected

Interpreted or just-in-time compiled

Multi-paradigm

Prototype-based object-oriented

DECONSTRUCTING THE MONSTER DEFINITION

High-level

Garbage-collected

Interpreted or just-in-time compiled

Multi-paradigm

Prototype-based object-oriented

DECONSTRUCTING THE MONSTER DEFINITION

Paradigm



directive

1

2

High-level

Garbage-collected

Interpreted or just-in-time compiled

Multi-paradigm

Prototype-based object-oriented

DECONSTRUCTING THE MONSTER DEFINITION

High-level

Garbage-collected

Interpreted or just-in-time compiled

Multi-paradigm

Prototype-based object-oriented

Our array
inherits me

from protot

DECONSTRUCTING THE MONSTER DEFINITION

High-level

In a language
as variable
from

Garbage-collected

Interpreted or just-in-time compiled

Multi-paradigm

Prototype-based object-oriented

DECONSTRUCTING THE MONSTER DEFINITION

High-level

Garbage-collected

Interpreted or just-in-time compiled

Multi-paradigm

No
bec

Prototype-based object-oriented

DECONSTRUCTING THE MONSTER

Code has



High-level

Garbage-collected

Interpreted or just-in-time compiled

Multi-paradigm

Prototype-based object-oriented

Java



Scalability

... b



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

WHAT IS A JAVASCRIPT ENGINE?



PROGRAM THAT EXECUTES
JAVASCRIPT CODE.

COMPUTER SCIENCE SIDENOTE: COMPILER

- 👉 **Compilation:** Entire code is converted into machine

Source code

STEP 1
—
COMPIRATION

- 👉 **Interpretation:** Interpreter runs through the source code

MODERN JUST-IN-TIME COMPILATION

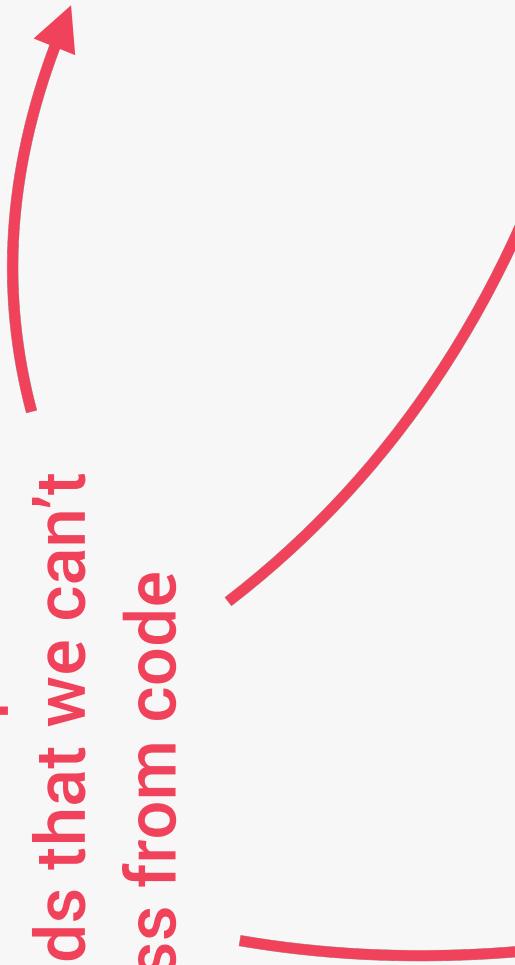
ENGINE

JS

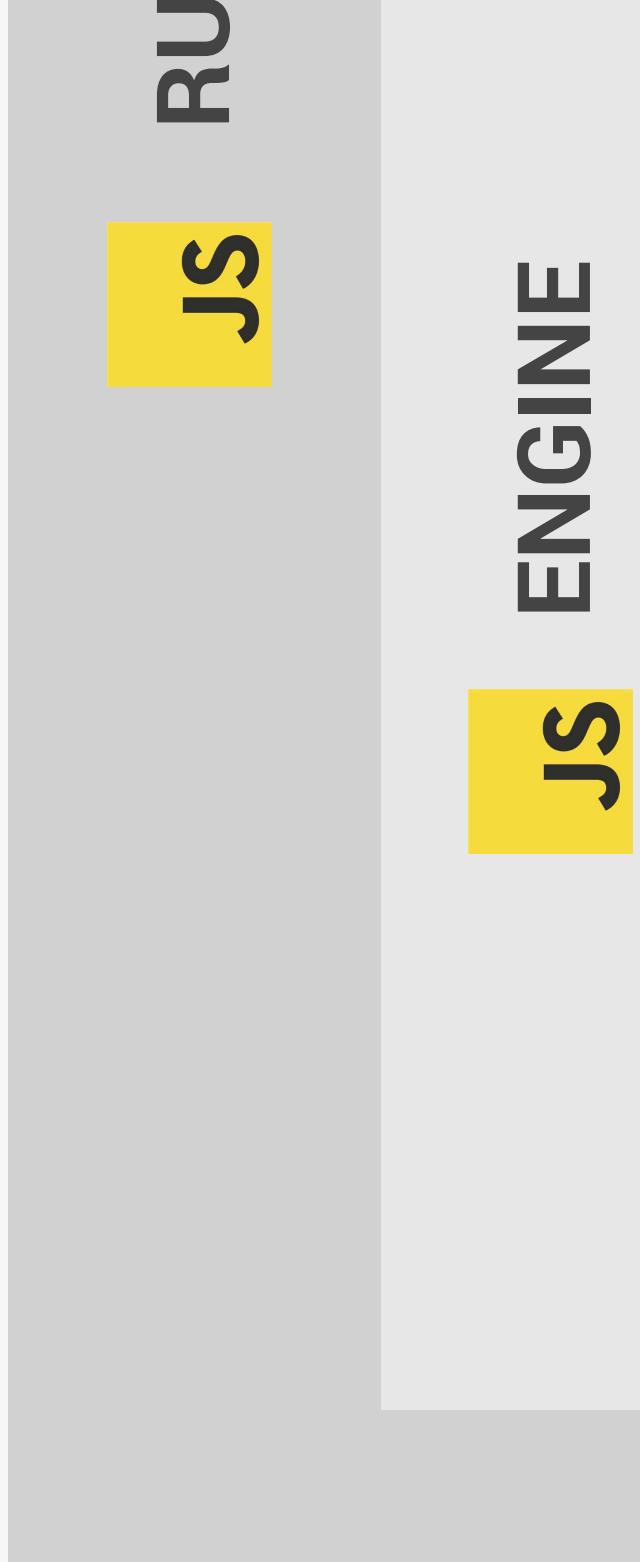
```
document.querySelector(".again").  
  document.querySelector(".message").  
  document.querySelector(".number").  
  document.querySelector(".guess").  
});
```



Happens in special
threads that we can't
access from code



THE BIGGER PICTURE: JAVASCRIPT



THE BIGGER PICTURE: JAVASCRIPT

RU

JS

ENGINE

JS





JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

WHAT IS AN EXECUTION CONTEXT?

Com

Creation of g
context (for

Human-readable code:



```
const name = 'Jonas';
const first = () => {
  let a = 1;
  const b = second();
  a = a + b;
  return a;
};
```

NO

EXECUTION CONTEXT IN DETAIL

WHAT'S INSIDE EXECUTION CONTEXT?

1

Variable Environment

- let, const and var declarations



- Functions



~~arguments object~~



THE CALL STACK

- 👉 Compiled code starts execution

```
const name = 'Jonas';

const first = () => {
  let a = 1;
  const b = second(7, 9);
  a = a + b;
  return a;
```




JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

SCOPING AND SCOPE IN JAVASCRIPT

SCOPE CONCEPTS

- 👉 **Scoping:** How our program's values live? Or "Where can we access a
- 👉 **Lexical scoping:** Scoping is cor

THE 3 TYPES OF SCOPE

GLOBAL SCOPE

```
const me = 'Jonas';
const job = 'teacher';
const year = 1989;
```

```
function()
  const
  const
  return
}
```

console

THE SCOPE CHAIN

```
const myName = 'Jonas';
```

```
function first() {
```

```
    const age = 30;
```

```
    if (age >= 30) { // true
```

```
        const decade = 3;
```

```
        var millennial = true;
```

```
}
```

let and const are **block-scoped**

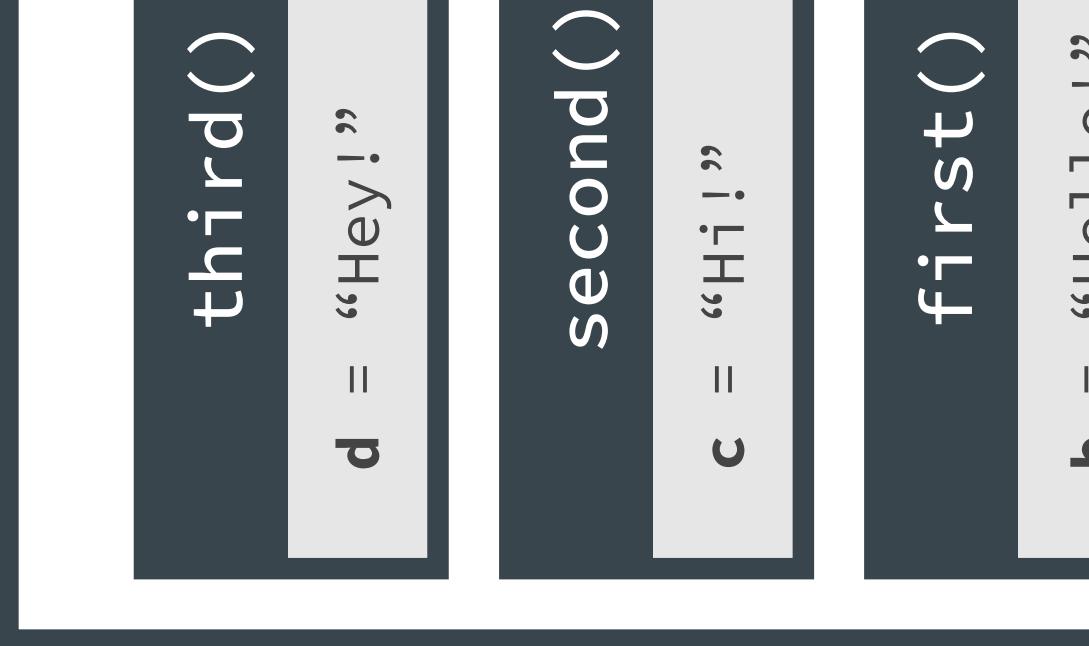
Variables not
current scope

var is **function-scoped**

```
function second() {
```

SCOPE CHAIN VS. CALL STACK

```
const a = 'Jonas';
first();
function first() {
  const b = 'Hello!';
  second();
}
function second() {
  const c = 'Hi!';
  third();
}
third();
```



SUMMARY



- 👉 Scoping asks the question “Where do variables live?”
- 👉 There are 3 types of scope in JavaScript: the global scope, the function scope, and the block scope.
- 👉 Only let and const variables are block-scoped.
- 👉 In JavaScript, we have lexical scoping, so the variable declared inside a function can be accessed from inside another function.
- 👉 Every scope always has access to all the variables declared in its parent scope.



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

HOISTING IN JAVASCRIPT

- 👉 **Hoisting:** Makes some types of variables actually declared. “Variables lifted to the top”

↓ **BEHIND THE SCENES**

Before execution, code is scanned for variable property is created in the **variable environment**

HOIST



TEMPORAL DEAD ZONE, LET AND CONST

```
const myName = 'Jonas';
if (myName === 'Jonas') {
  console.log(`Jonas is a ${job}`);
  const age = 2037 - 1989;
  console.log(age);
  const job = 'teacher';
  console.log(x);
}
```




JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

HOW THE THIS KEYWORD WORKS

- 👉 **this keyword/variable:** Special variable that is created.
Takes the value of (points to) the “owner” of the function.
- 👉 **this is NOT static.** It depends on how the function is actually called.

Method ➡ `this = <Object that is calling the method>`



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

REVIEW: PRIMITIVES, OBJECTS AND

PRIMITIVES

Number



String



Boolean



Undefined

STORED IN



PRIMITIVE VS. REFERENCE VALUES

- 👉 Primitive values example:

```
let age = 30;  
let oldAge = age;  
age = 31;  
console.log(age); // 31  
console.log(oldAge); // 30
```

- 👉 Reference values example:

```
const me = {
```

HOW JAVASCRIPT WORKS BEHIND THE SCENES

Prototypal Inheritance

1

2

Event Loop



Asynchronous



DATA

S

DATA
STRUCTURE
DESIGN
ALGORITHM
PROGRAMMING



JONAS.IO

SCHMEDTMANN

THE COMPLETE
JAVASCRIPT COURSE

DATA STRUCTURES OVERVIEW

SOURCES OF DATA

- 1 From the program itself: Data written directly
- 2 From the UI: Data input from the user or data
- 3 From external sources: Data fetched for exa



ARRAYS VS. SETS AND OBJECTS VS.

ARRAYS

VS.

SETS

```
tasks = ['Code', 'Eat', 'Code'];
// ["Code", "Eat", "Code"]
```

```
tasks = new Set(['Code'],
// {"Code", "Eat", "Eat"})
```

- 👉 Use when you need **ordered** list of values (might contain duplicates)
- 👉 Use when you need **unique** values

- 👉 Use when you need **unique** values
- 👉 Use when **high-per-**

S

O

L

G

A



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

FIRST-CLASS VS. HIGHER-ORDER FU

FIRST-CLASS FUNCTIONS

- 👉 JavaScript treats functions as **first-class citizens**
- 👉 This means that functions are **simply values**
- 👉 Functions are just another “**type**” of **object**

👉 Store functions in **variables** or **properties**:

```
const add = (a, b) => a + b;
```

```
const counter = {
```




JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

„CREATING“ A CLOSURE

Variable
Environment

secureBooking() EC

```
const secureBooking = function(...args) {  
    let passengerCount = args.length;  
    return function(...args) {  
        passengerCount += args.length;  
        console.log(`Passenger count: ${passengerCount}`);  
    };  
};
```

UNDERSTANDING CLOSURES

```
secureBooking()
```

```
passengerCount = ()
```

```
const secureBooking = () => {  
    let passengerCount = 0;  
    return function(...args) {  
        passengerCount += args.length;  
        if (passengerCount > 10) {  
            console.error(`Overbooked! Only ${passengerCount} passengers allowed`);  
        }  
        return `Secure booking for ${passengerCount}`;  
    };  
};
```

```
bookings[0]();
```

UNDERSTANDING CLOSURES

- 👉 A function has access to the variable environment (VE)
- 👉 **Closure:** VE attached to the function, exactly as it was at

```
const secure  
let passen
```

```
return fu  
passeng  
conceal
```

```
hooked up to
```

CLOSURES SUMMARY



- 👉 A closure is the **closed-over variable environment** execution context is gone;

↓ **Less formal**

- 👉 A closure gives a function access to all the variables function keeps a **reference** to its outer scope, which

↓ **Less formal**

- 👉 A closure makes sure that a function doesn't loose

Less formal

—
R

O

W



JONAS.IO

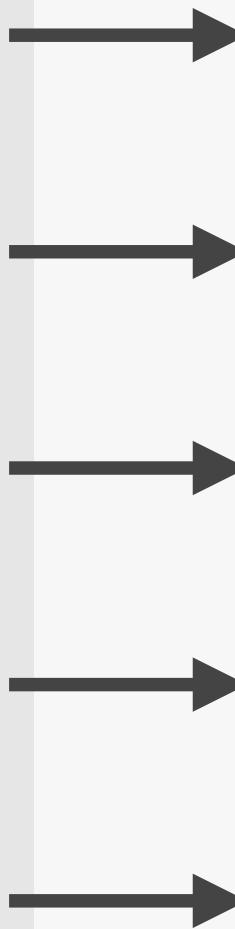
SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

DATA TRANSFORMATIONS WITH MAP

Original array

```
3  
1  
4  
3
```



MAP

Example

```
current * 2
```




JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE



WHICH ARRAY METHOD TO USE?

To mutate original array

👉 Add to original:

• **push** (end)

• **unshift** (start)

👉 Remove from original:

• **pop** (end)

• **shift** (start)

A new array

👉 Computed from original:

• **map** (loop)

👉 Filtered using condition:

• **filter**

👉 Portion of original:

• **slice**

ADVANZ
G



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

REVIEW: WHAT IS THE DOM?

JS



- Allows us to make JS
- 👉
- We can write JavaS
- 👉



HOW THE DOM API IS ORGANIZED



MDN web docs

[moz://a](https://developer.mozilla.org/en-US/docs/Web/API/Element)

Represented by
JavaScript object

- `textContent`
- `childNodes`
- `parentNode`
- `cloneNode()`

• `innerHTML`

• `classList`

• `children`

• `parentElement`

• `append()`

• `remove()`

• `insertAdjacentHTML()`

• `querySelector()`

• `closeset()`

<code>Paragraph</p>

Element

Text



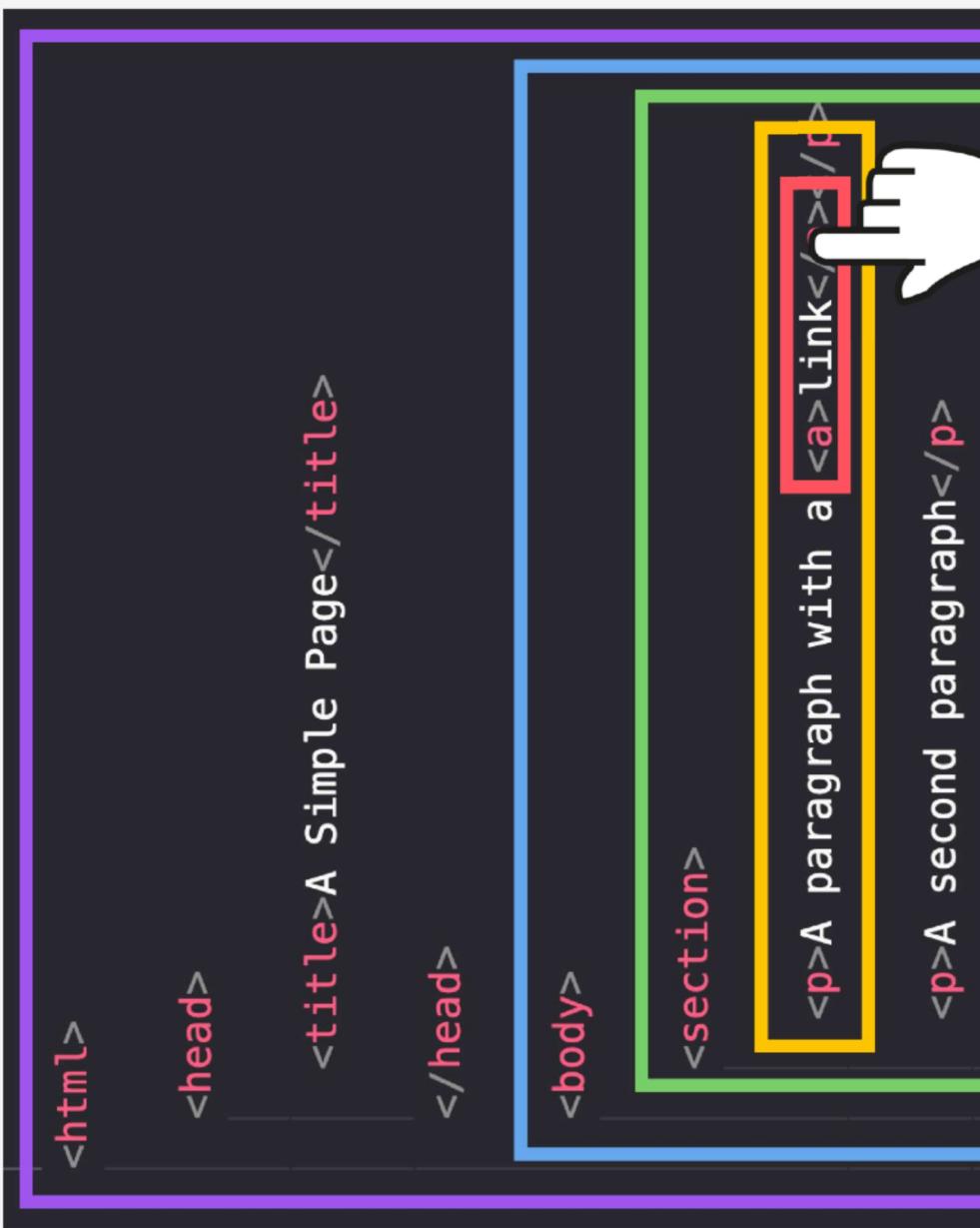
JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

BUBBLING AND CAPTURING

CAPTURING PHASE



1

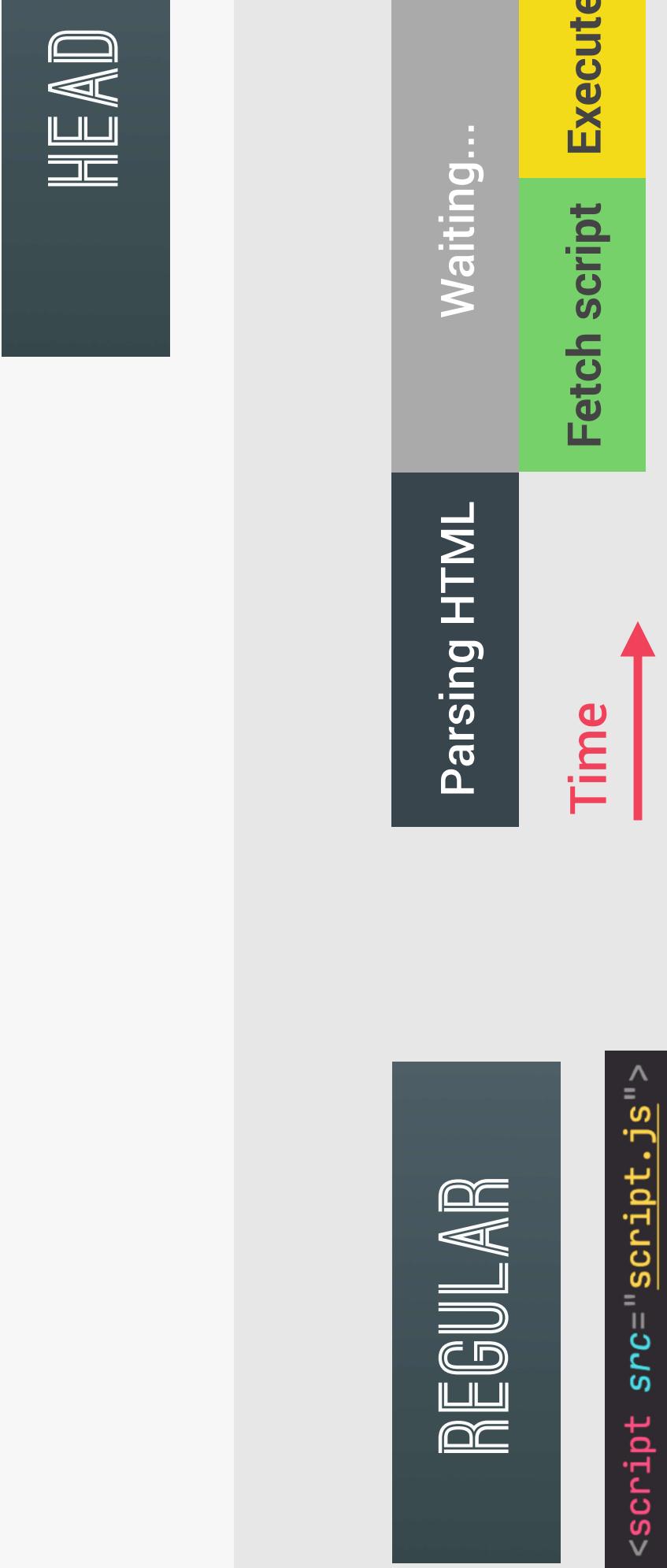


JONAS.IO

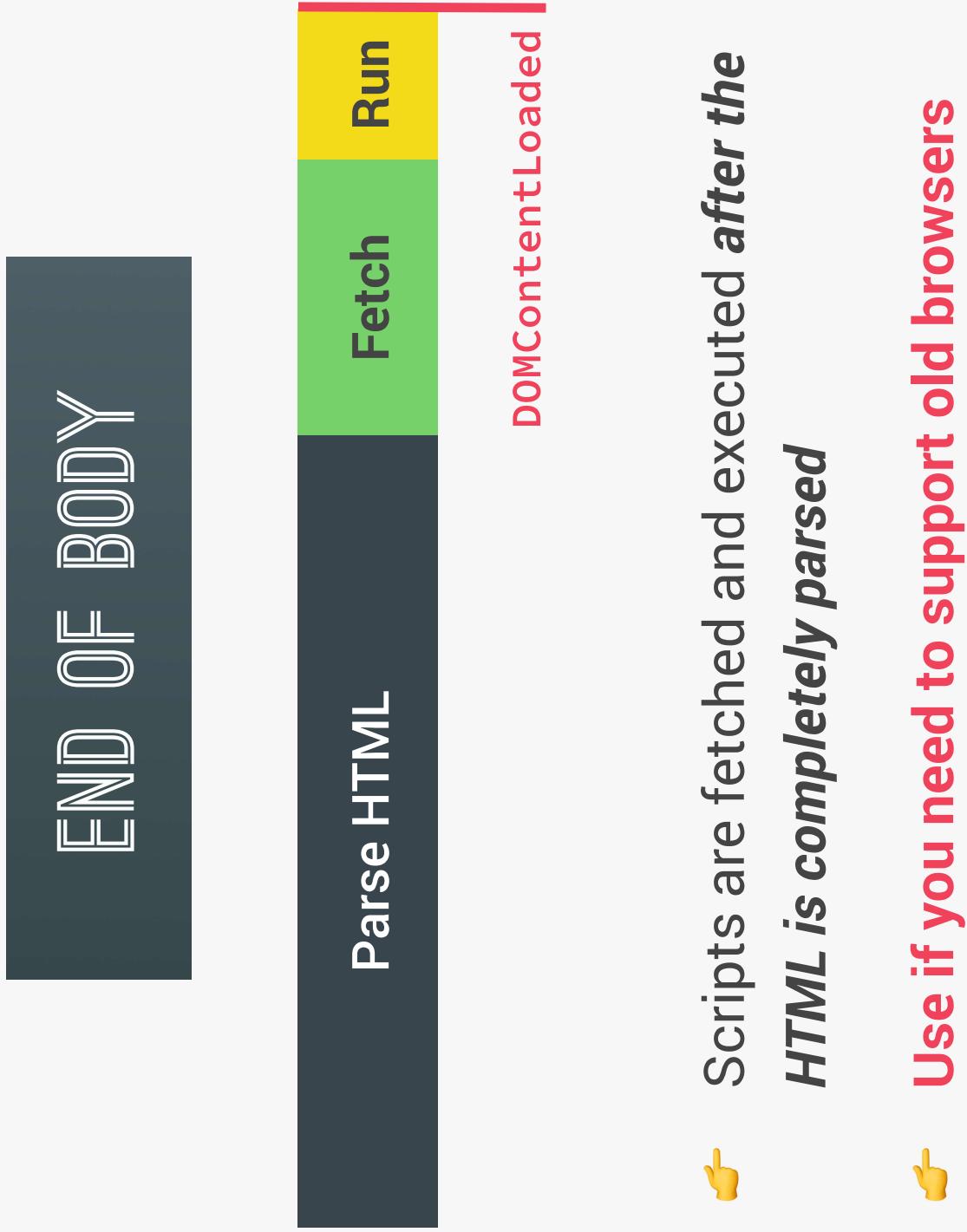
SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

DEFER AND ASYNC SCRIPT LOADING



REGULAR VS. ASYNC VS. DEFER



Five
years
ago
I
had
no
idea
what
I
was
doing
with
my
life.

Now
I
have
no
idea
what
I
was
doing
with
my
life.



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

WHAT IS OBJECT-ORIENTED PROGRAMMING



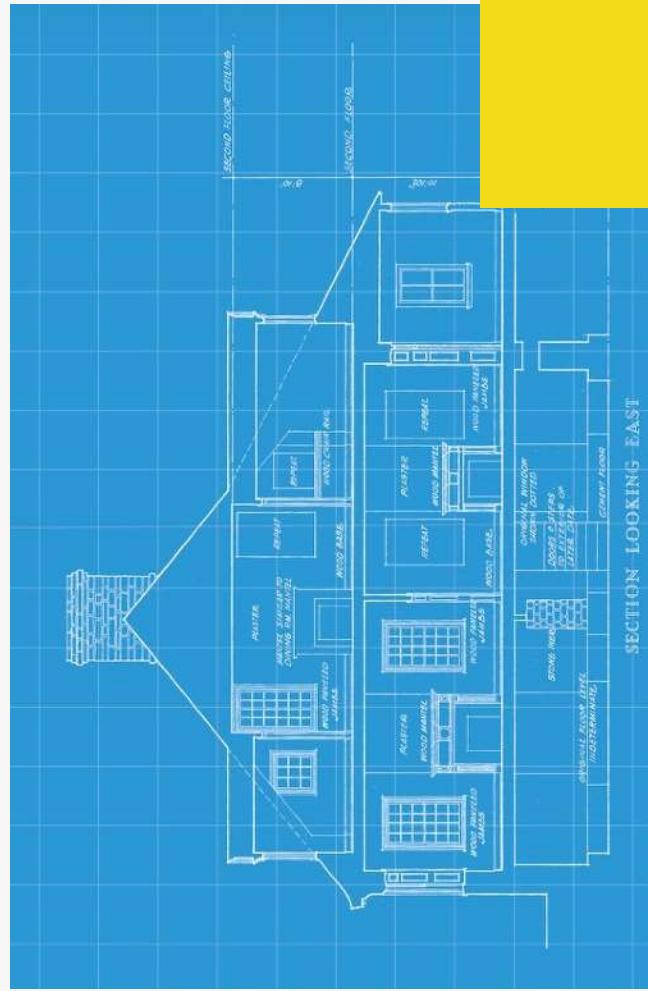
Data

- Object-oriented programming concept of objects;
- We use objects to model (
- Objects may contain data and the corresponding pack data and the corresponding

```
const user = {  
  user: 'jonas',  
  password: 'dk23s',  
  login(password) {  
    if (password === this.password) {  
      console.log('Access granted!');  
    } else {  
      console.log('Access denied!');  
    }  
  }  
};
```

CLASSES AND INSTANCES (TRADITION)

Like a blueprint
which we can create
new objects



User {

THE 4 FUNDAMENTAL OOP PRINCIPLES

The 4 fundamental principles
of Object Oriented Programming



Abstraction

Encapsulation

Inheritance

PRINCIPLE 1: ABSTRACTION

Phone {

charge
volume
voltage
temperature

Abstraction

homeBtn()
volumeBtn()
screen() {
verifyVolt
verifyTemp
vibrate()
soundSpeak()
soundEar()

Encapsulation

Inheritance

PRINCIPLE 2: ENCAPSULATION

Abstraction

Encapsulation

Inheritance

NOT accessible from
outside the class!

STILL accessible fro
within the class!

STILL accessible fro
within the class!

NOT accessible from

PRINCIPLE 3: INHERITANCE

```
User {  
    user  
    password  
    email  
  
    Login(password)  
    // Login logic  
}  
  
sendMessage()  
// Sending logic  
}
```

Abstraction

Encapsulation

Inheritance

PRINCIPLE 4: POLYMORPHISM

INHERITANCE

```
Admin {
```

```
    user  
    password  
    email  
    permissions
```

```
Login(password,  
// DIFFERENT /
```

Abstraction

Encapsulation

Inheritance



JONAS.IO

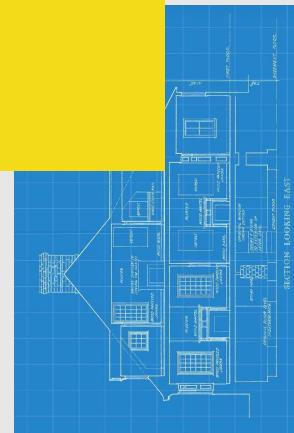
SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

OOP IN JAVASCRIPT: PROTOTYPES

"CLASSICAL OOP": CLASSES

Class



↓
INSTANTIATION

Instance

0

3 WAYS OF IMPLEMENTING PROTOTOT

“How do we actually create *prototypes*? How can we *create new objects*? 🤔

1

Constructor functions

- 👉 Technique to create objects from a function
- 👉 This is how built-in objects like Arrays, Number, String, etc. are implemented



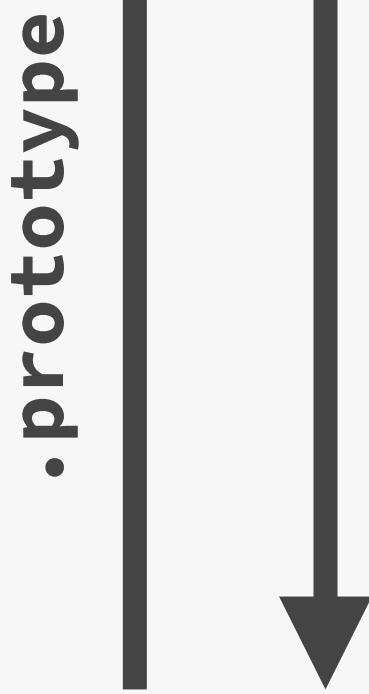
JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

HOW PROTOTYPAL INHERITANCE //

Constructor function
[Person()]



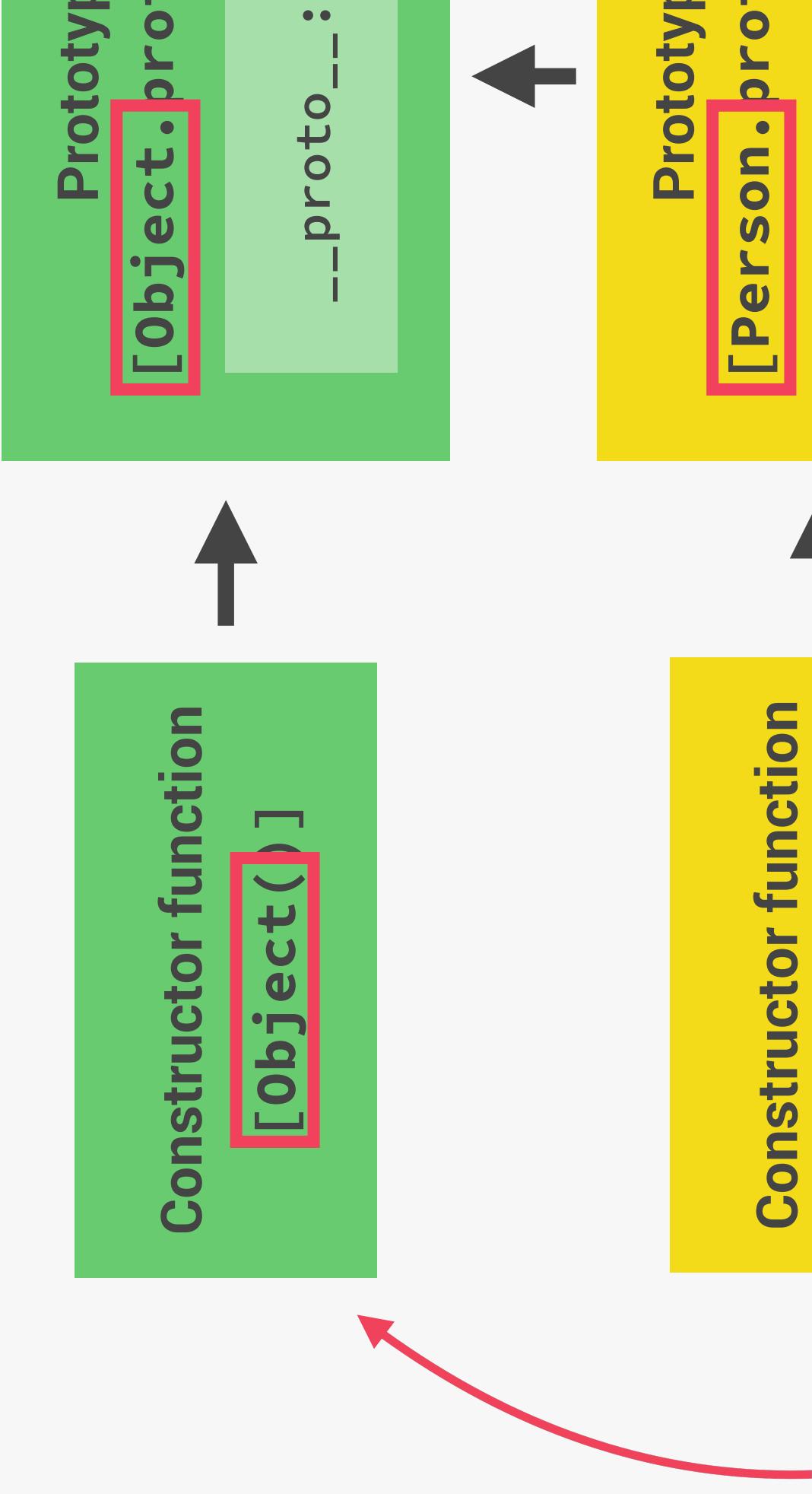
```
const Person = function(name, birthYear) {  
  ...  
  this.name = name;  
  this.birthYear = birthYear;  
};
```

•constructor

PROTOTYPE

CHAIN

THE PROTOTYPE CHAIN





JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

HOW OBJECT.CREATE WORKS

Prototype

[PersonProto]

calcAge: function

```
const PersonProto = {  
  calcAge() {  
    console.log(2037 - this.year);  
  },  
};
```

• `--proto--`

```
const steven = Object.create(PersonProto)
```




JONAS.IO

SCHMEDTMANN

THE COMPLETE
JAVASCRIPT COURSE

INHERITANCE BETWEEN "CLASSES"

Prototypal

Constructor
Function



INHERITANCE BETWEEN "CLASSES"

Constructor function

[Person ()]

Proto

[Person.prototype]

```
Student.prototype = Object.create(Person.prototype);
```

Proto

[Student.prototype]

Constructor function

INHERITANCE BETWEEN CLASSES

```
Student.prototype = Object.create(Person.prototype);
```

Prototype
[Person.prototype]

Constructor function
[Person()]



Prototype
[Student.prototype]

Constructor function



INHERITANCE BETWEEN "CLASSES"

Constructor function

[Person ()]

Proto

[Person.prototype]

--proto--
Object.prototype

```
Student.prototype = Object.create(Person.prototype);
```

Proto

[Student.prototype]

Constructor function



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

INHERITANCE BETWEEN "CLASSES":

```
const StudentProto = Object.create(PersonProto)
```




JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

Public field (similar to property, available on created object)

Private fields (not accessible outside of class)

Static public field (available only on class)

Call to parent (super) class (necessary with extend). Needs to happen before accessing this

Instance property (available on created object)

Redefining private field

Public method

Referencing private field and method

Private method (⚠ Might not yet work in your

MAP

GO



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

PROJECT PLANNING

Description of the application's functionality from the user's perspective. All user stories put together describe the entire application.

1. USER STORIES

2. FEATURES

1. USER STORES

- 👉 User story: Description of the user's goal.
- 👉 Common format: As a *type* of user, I want to *verb* my *noun*.

1

As a user, I want to log my steps/minute, so I can keep track of my progress.

2

As a user, I want to log my meals and track my nutrition intake.

2. FEATURES

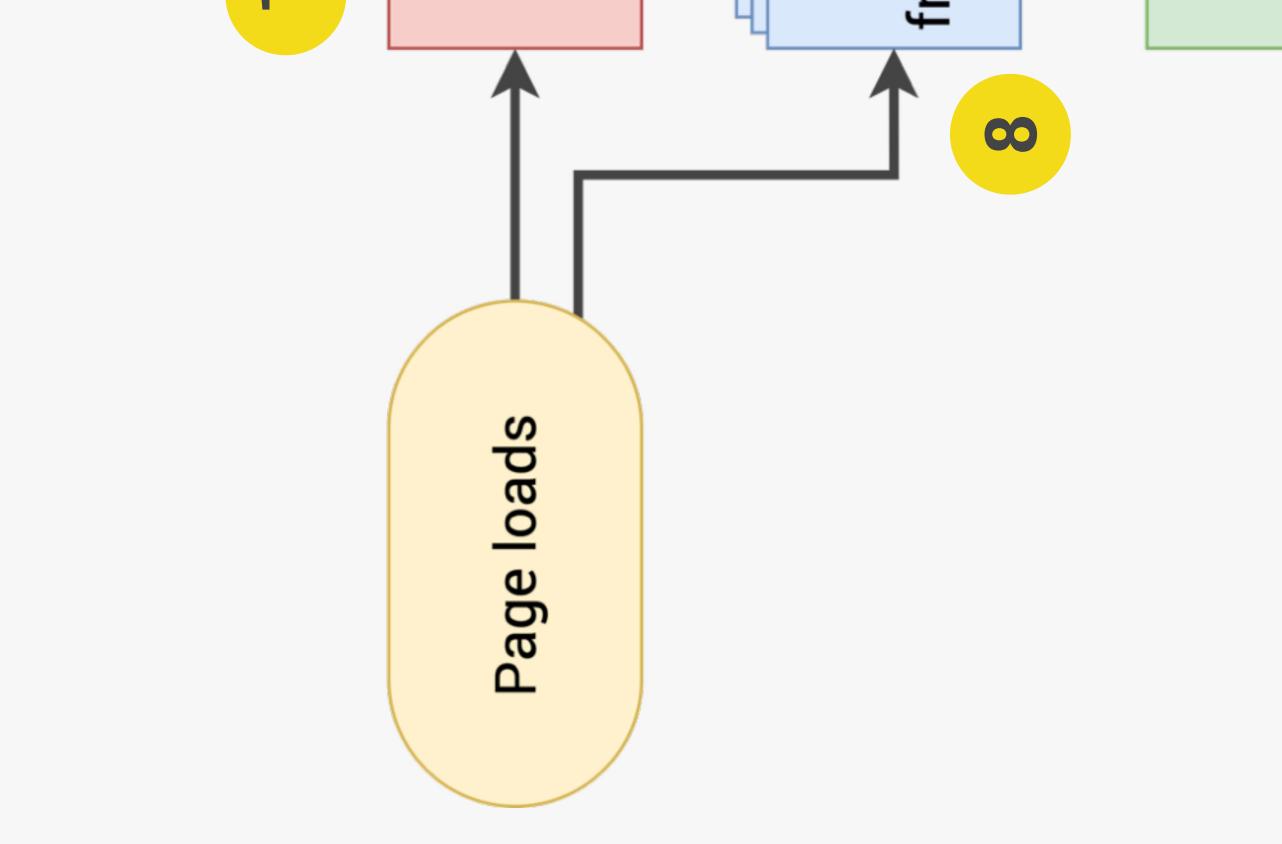
USER STORIES

- 1 Log my running workouts with location, distance, time, pace and steps/minute
- 2 Log my cycling workouts with location, distance, time, speed and elevation gain

3. FLOWCHART

FEATURES

1. Geolocation to display map at current location
2. Map where user clicks to add new workout
3. Form to input distance, time, pace, steps/minute
4. Form to input distance, time, speed, elevation gain



A. ARCHITECTURE

FOR N



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

10 ADDITIONAL FEATURE IDEAS: CHARTS

- 👉 Ability to **edit** a workout;
- 👉 Ability to **delete** a workout;
- 👉 Ability to **delete all** workouts;
- 👉 Ability to **sort** workouts by a certain field (e.g.
- 👉 **Re-build** Running and Cycling objects coming
- 👉 More realistic error and confirmation messages

O
N
Z
Y
S
A



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

SYNCHRONOUS CODE

```
const p = document  
p.textContent = 'M  
alert('Text set!')  
p.style.color = 'r
```



BLOCKING

ASYNCHRONOUS CODE

Asynchronous

```
const p = document  
setTimeOut(function  
    p.textContent =  
}, 5000);  
p.style.color = 'r'
```

CALLBACK WILL
RUN AFTER TIMER

Example: Timer with callback



[1 - 2 - 3] man(v => v *

ASYNCHRONOUS CODE

Asynchronous

```
const img = document  
img.src = 'dog.jpg'  
img.addEventListener  
img.classList.add('dog')  
p.style.width = '300px'
```

Example: Asynchronous in



**CALLBACK WILL RUN
AFTER IMAGE LOADS**

WHAT ARE AJAX CALLS?



Asynchronous JavaScript
remote web servers in a
request data from web :

WHAT IS AN API?



- 👉 Application Programming Interface: Piece that can be used by another piece of software to talk to each other;
- 👉 There are many types of APIs in web development.

DOM API

Geolocation API

Own Class API



JONAS.IO

SCHMEDTMANN

THE COMPLETE
JAVASCRIPT COURSE

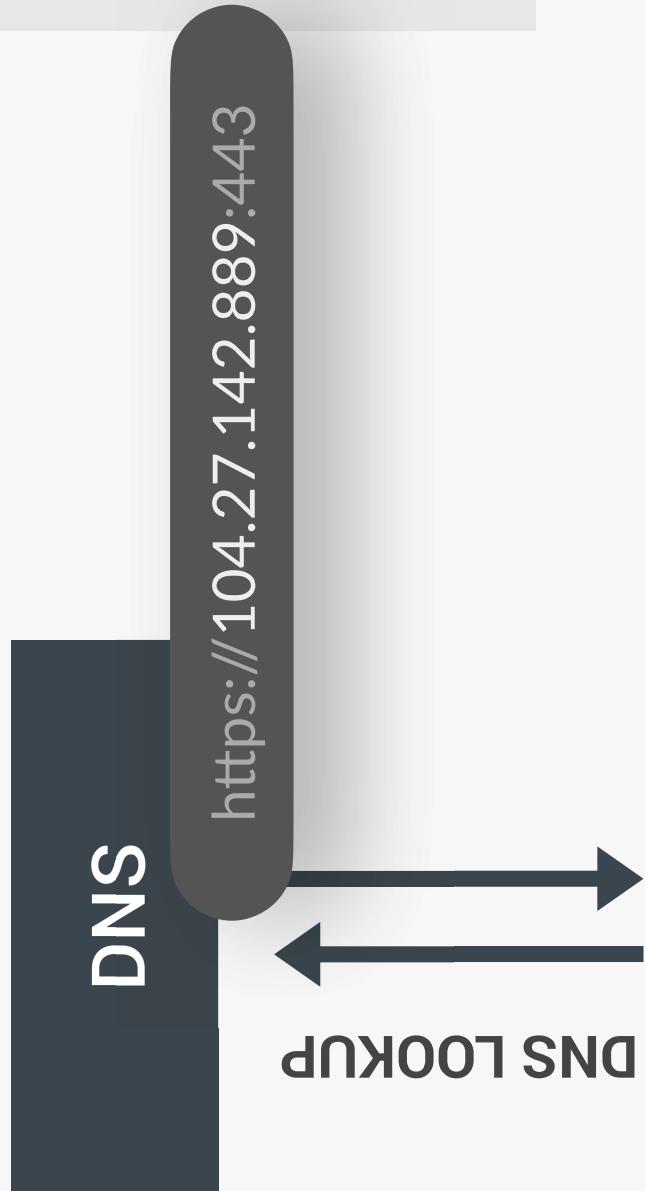
WHAT HAPPENS WHEN WE ACCESS

Request-response



CLIENT
(e.g. browser)

WHAT HAPPENS WHEN WE ACCESS



1

CLIENT
(e.g. browser)

2



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

WHAT ARE PROMISES?



- 👉 **Promise:** An object that is used as a result of an asynchronous operation.
👉 **Less formal** 
- 👉 **Promise:** A container for a future value.
👉 **Less formal** 

THE PROMISE LIFE CYCLE

Before the future
value is available



PENDING

ASYNC TASK





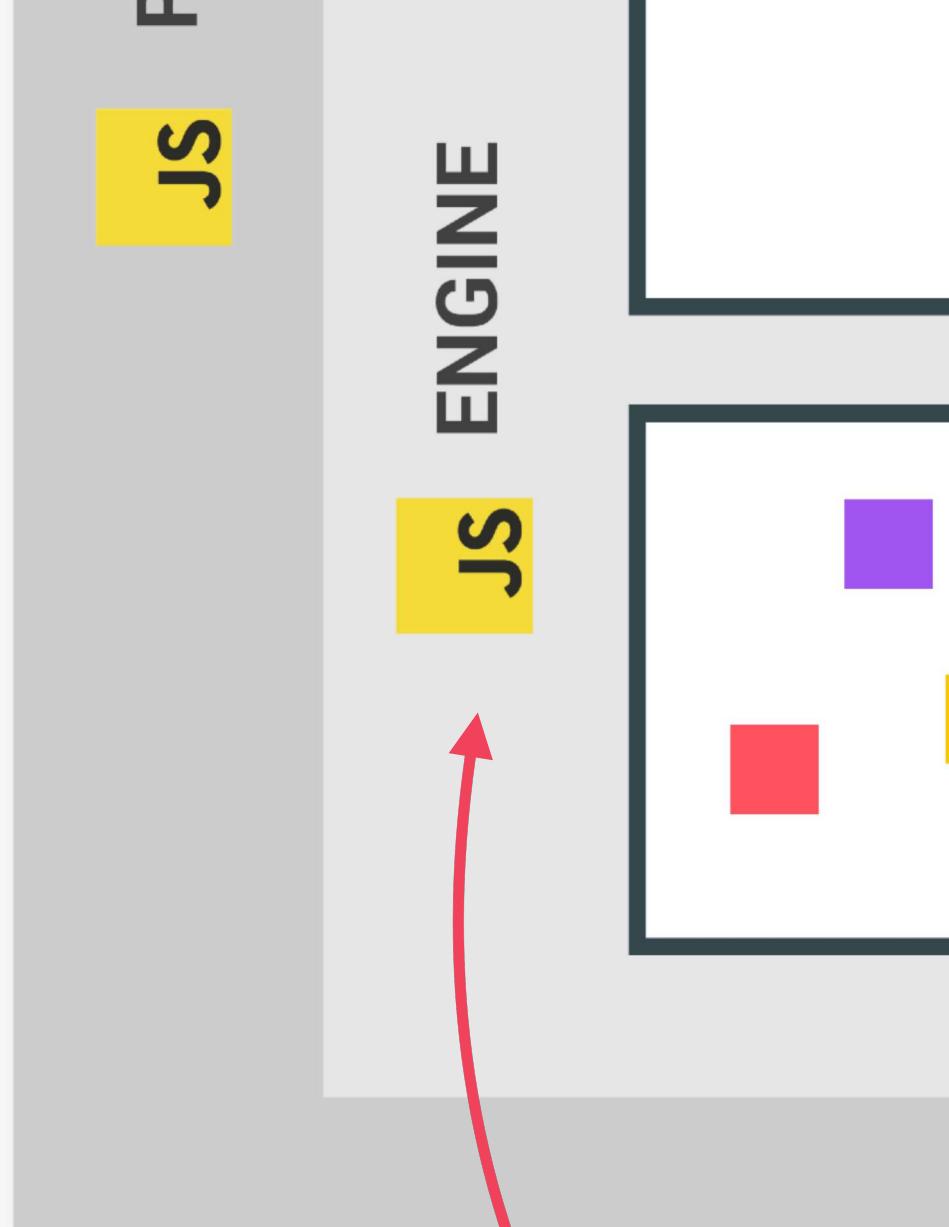
JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

REVIEW: JAVASCRIPT RUNTIME

- 👉 **Concurrency model:** How JavaScript handles multiple tasks happening at the same time.

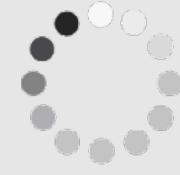


“Heart” of
the runtime

HOW ASYNCHRONOUS JAVASCRIPT

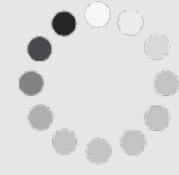
Where asynchronous tasks run —

```
○ => {  
    el.classList  
}
```



Loading image

```
res => con
```



Fetching data

Execution context
log()

Z

W

R

V

W

W

D

D

O

W



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

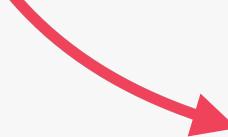
MODERN JAVASCRIPT DEVELOPMENT

MODULE

MODULE

3RD-PARTY
PACKAGE

BUN





JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

AN OVERVIEW OF MODULES



- 👉 Reusable piece of code that **encapsulates** implementation details;
- 👉 Usually a **standalone file**, but it doesn't have to be.

NATIVE JAVASCRIPT (ES6) MODULES

ES6 MODULES

Modules stored in files, exactly
one module per file.

HOW ES6 MODULES ARE IMPORTED

```
import { rand } from './math.js';
import { showDice } from './dom.js';
const dice = rand(1, 6, 2);
showDice(dice);
```

index.js
👉

```
const rand = () => {
  // Random number between 1 and 6
  return Math.floor(Math.random() * 5) + 1;
};

export { rand };
```

math.js
👉

Live connection,
NOT copies



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

REVIEW: MODERN AND CLEAN CODE

READABLE CODE

- 👉 Write code so that **others** can understand it
- 👉 Write code so that **you** can understand it in 1 year
- 👉 Avoid too “clever” and overcomplicated solutions
- 👉 Use descriptive variable names: **what they contain**
- 👉 Use descriptive function names: **what they do**

REVIEW: MODERN AND CLEAN CODE

AVOID NESTED CODE

- 👉 Use early return (guard clauses)
- 👉 Use ternary (conditional) or logical operators instead of if/else-if
- 👉 Use multiple if instead of if/else-if



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

IMPERATIVE VS. DECLARATIVE

IMPERATIVE CODE

Two types of code:

IMPERATIVE

Programmer explains “**HOW** to do things”



We explain the computer every *single step* it has to follow to achieve a result



FUNCTIONAL PROGRAMMING PRINCIPLES

FUNCTIONAL PROGRAMMING

- 👉 Declarative programming paradigm
- 👉 Based on the idea of writing software by combining **pure functions**, avoiding **side effects** and mutating external variables
- 👉 **Side effect:** Modification (mutation) of any data outside of the function (mutating external variables, logging)

—
r
o
l

—
z
—
d
—
—
—
j
—
m



JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

PROJECT PLANNING

1.
USER STORIES

2.
FEATURES

1. USER STORES

- 👉 **User story:** Description of the user's goal or task.
- 👉 **Common format:** As a *type* of user, I want to *search* for *meal* for different number of people.

1

2

2. FEATURES

USER STORIES

1

Search for recipes

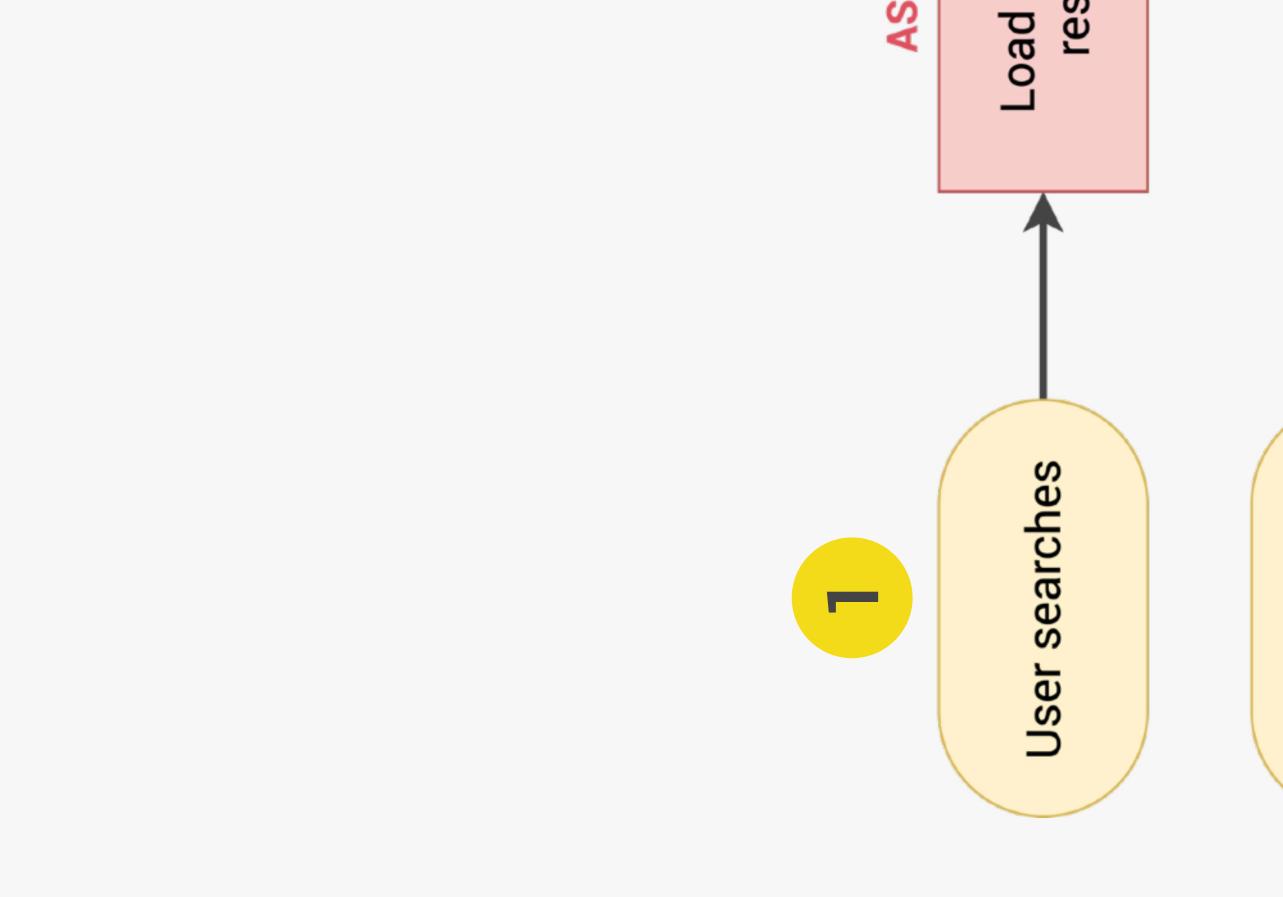
2

Update the number of servings

3. FLOWCHART (PART 1)

FEATURES

1. Search functionality: API search request
2. Results with pagination
3. Display recipe





JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

WHY WORRY ABOUT ARCHITECTURE

- 👉 Like a house, software needs a structure: the way we **organize our code**

COMPONENTS OF ANY ARCHITECTURE

BUSINESS LOGIC

STATE

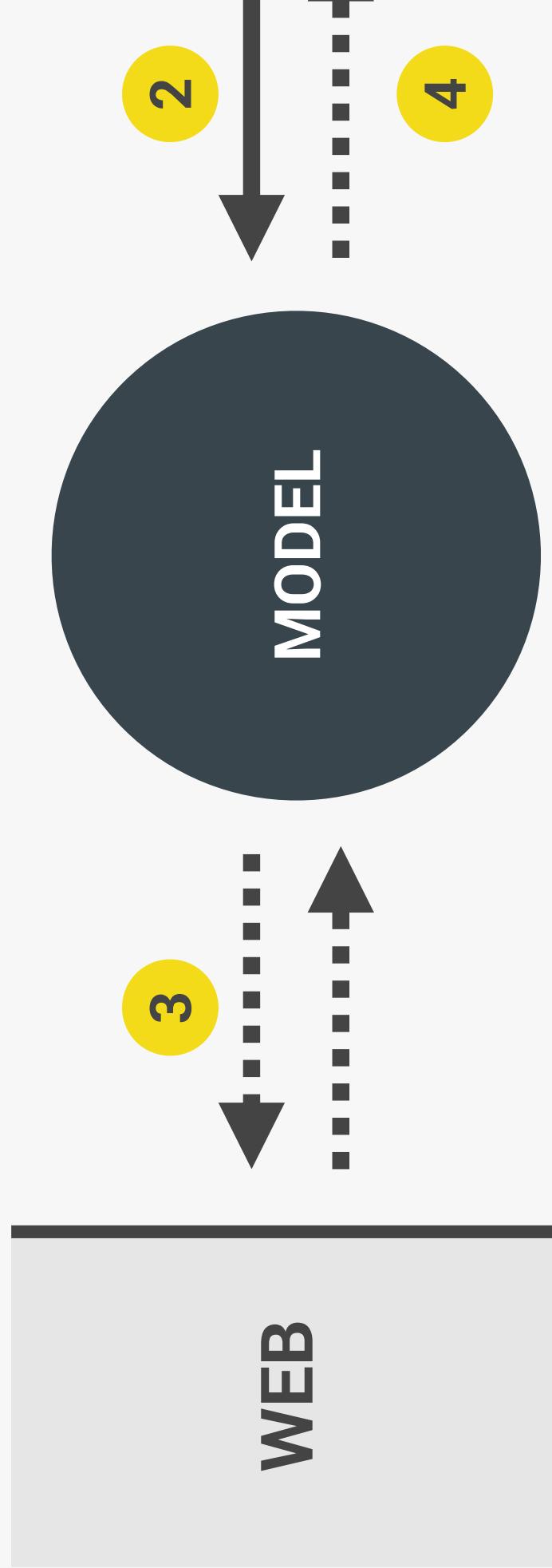
- 👉 Code that solves the actual business problem;

- 👉 Essentially stores all the data about the application

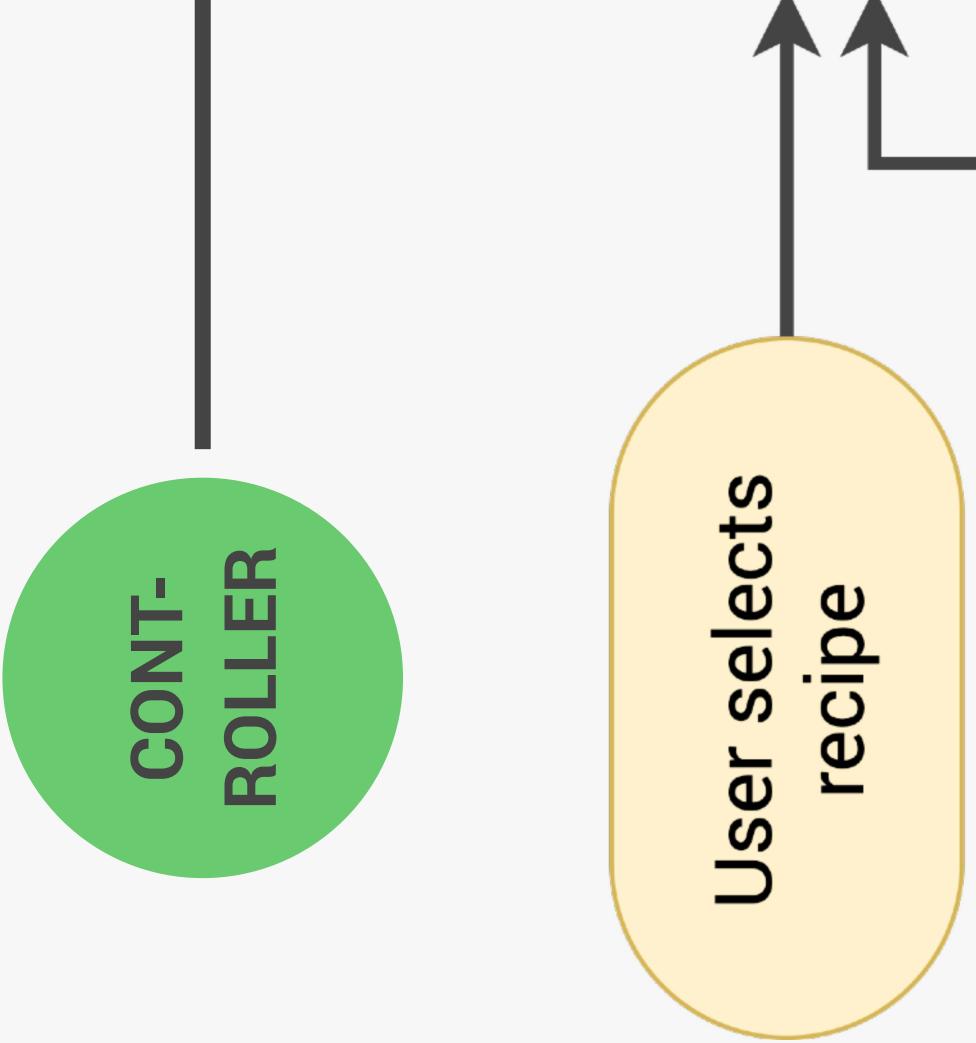
Directly related to

Should be the

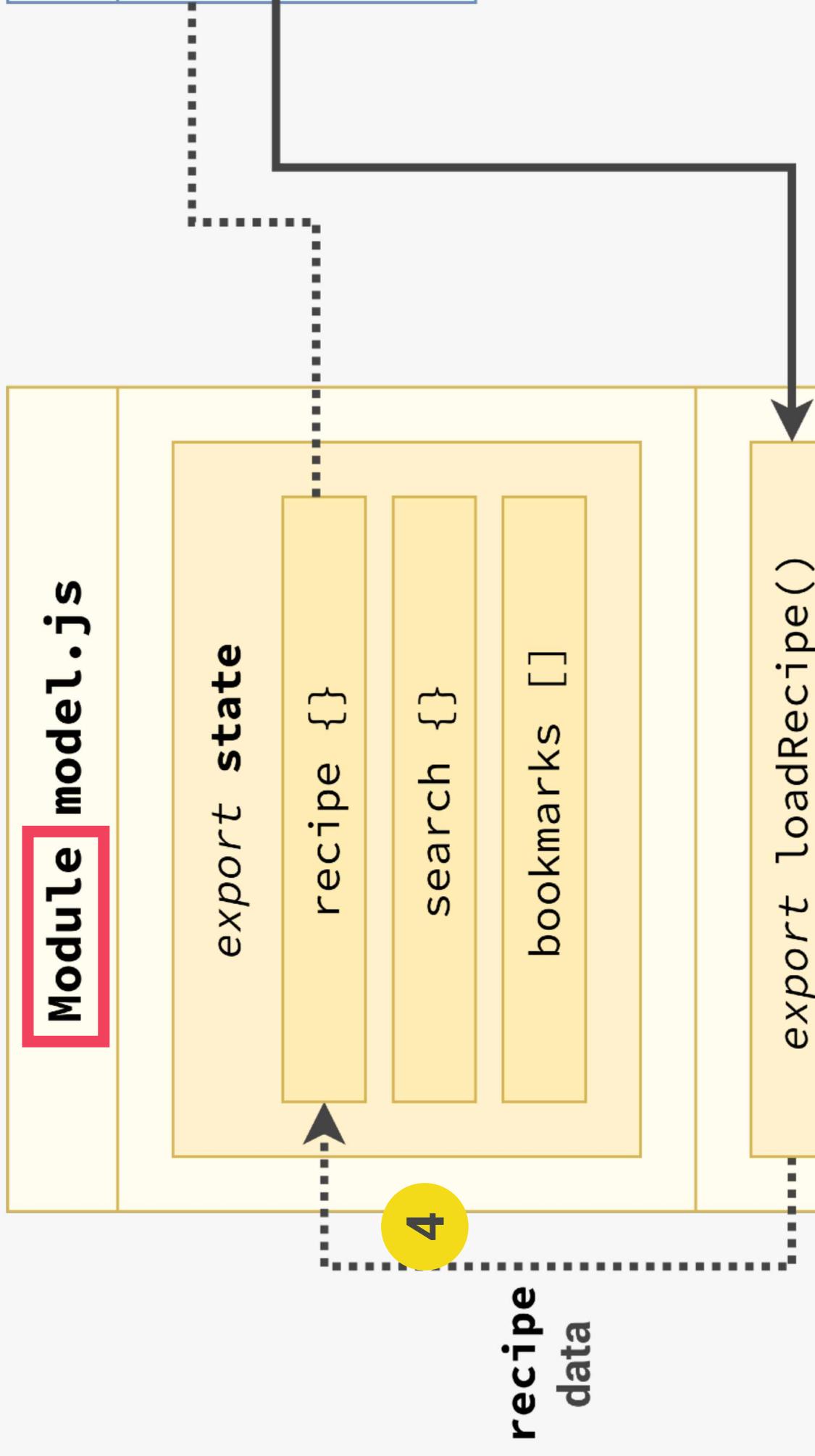
THE MODEL=VIEW=CONTROLLER (MVC)



MODEL, VIEW AND CONTROLLER IN F



MVC IMPLEMENTATION (RECIPE DISP)





JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

EVENT HANDLING IN MVC: PUBLISH

Code that wants to
react: **SUBSCRIBER**

Module controller.js

controlRecipes()

init()

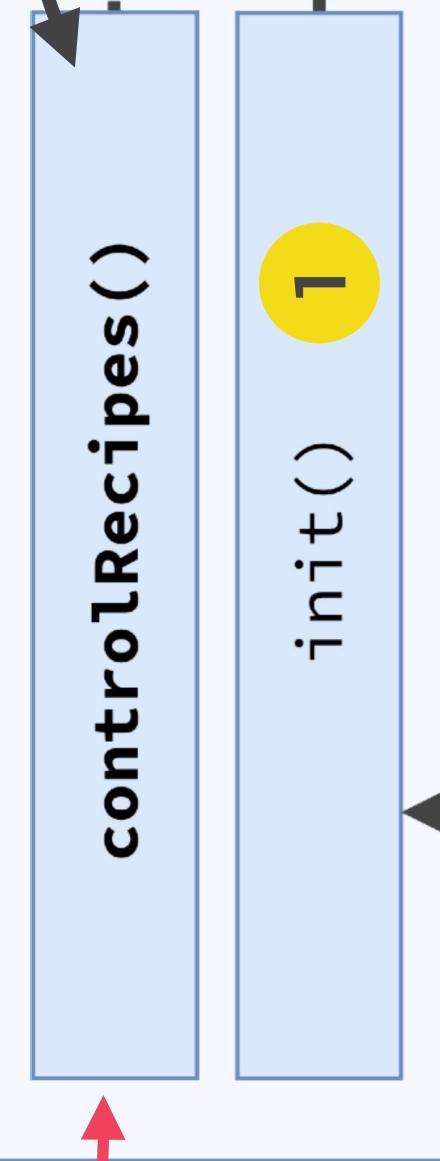
0

Program
starts

A() → B

→ Function call

.....→ Data flow





JONAS.IO

SCHMEDTMANN

THE COMPLETE JAVASCRIPT COURSE

IMPROVEMENT AND FEATURE IDEAS:

- 👉 Display number of pages between the pages
- 👉 Ability to sort search results by duration or number
- 👉 Perform ingredient validation in view, before saving
- 👉 Improve recipe ingredient input: separate in more than 6 ingredients;

