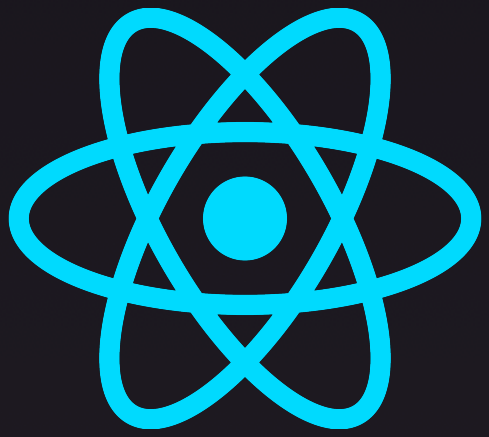
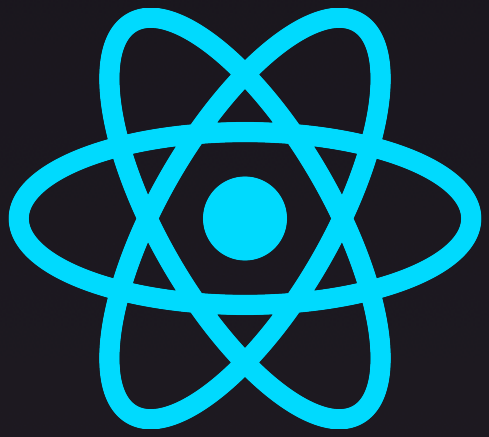
ACADEMIND



What is React?

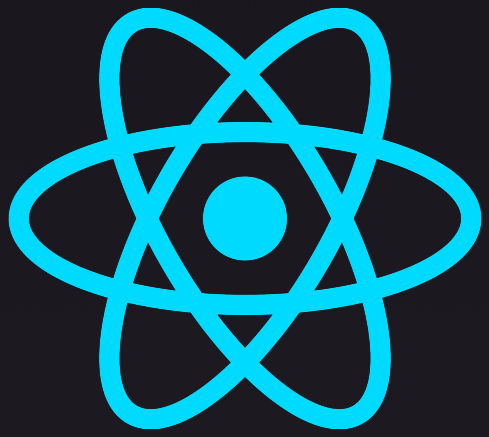
And why exactly would you use it?

ACADEMIND



React is a library for building user interfaces

ACADEMIND

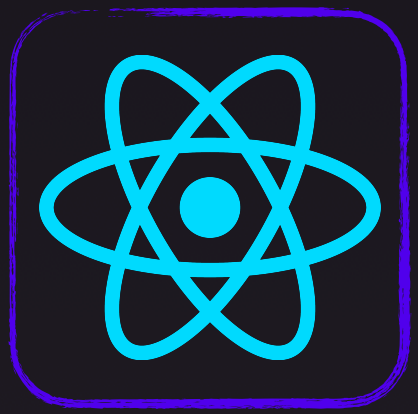


A JavaScript library for building user interfaces

But why would you use it?

ACADEMIND

Displays & manages website content & UI



React builds up on JavaScript (it's a library!)

As a React developer, you'll write React-specific JavaScript code

In the end, it's JavaScript doing "the magic"

React just uses JavaScript features

ACADEMIND

Why not just JavaScript?

ACADEMIND

Using "Just JavaScript" Typically Isn't A Great Option

- Writing complex JavaScript code quickly becomes **cumbersome**
- Complex JavaScript code quickly becomes **error-prone**
- Complex JavaScript code often is **hard to maintain or edit**
- React offers a **simpler mental model**

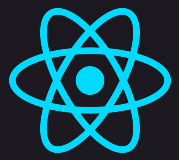
ACADEMIND

Basic Web Dev & JavaScript Knowledge is Assumed!

React = Declarative UI Programming

With React, you define the target UI state(s) — not the steps to get there!

Instead, React will figure out & perform the necessary steps



Declarative

Define the goal, not the steps

```
let content;

if (user.isLoggedIn) {
  content = <button>Continue</button>
} else {
  content = <button>Log In</button>
}

return content;
```

Imperative

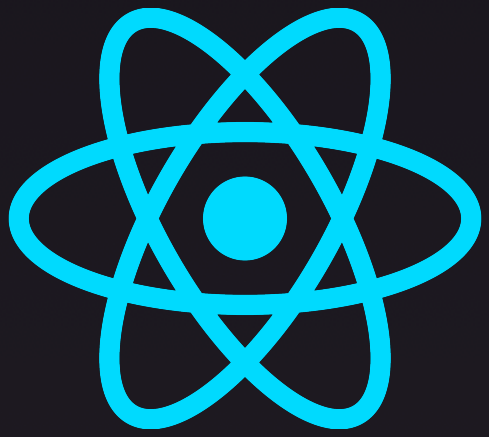
Define the steps, not the goal

```
let btn = document.querySelector('button');

if (user.isLoggedIn) {
  button.textContent = 'Continue';
} else {
  button.textContent = 'Log In';
}

document.body.append(btn);
```

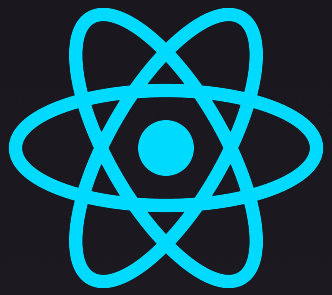
ACADEMIND



Our First React App

Time to get your hands dirty!

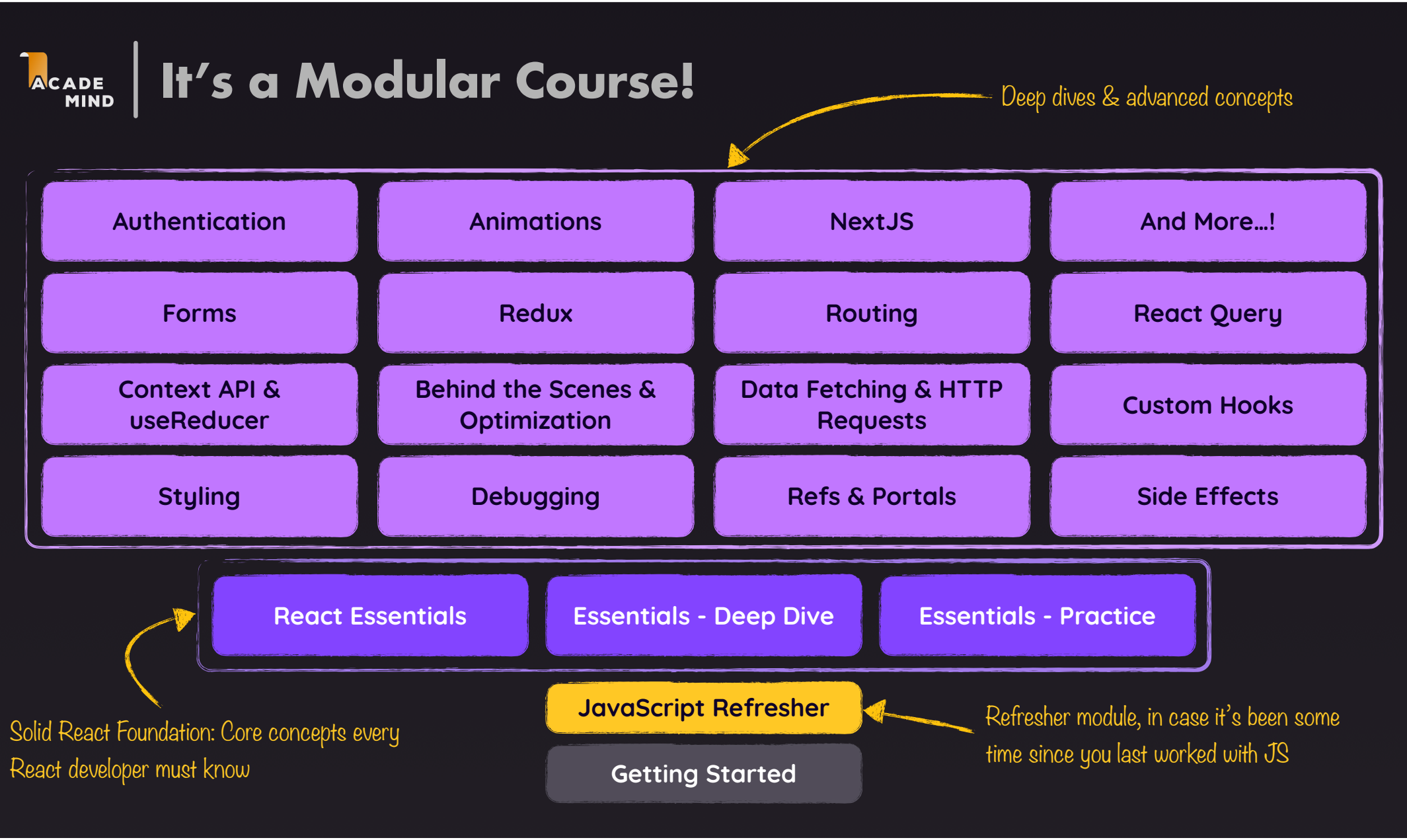
ACADEMIND



Our First React App

Time to get your hands dirty!

- Use the attached (updated!) demo app
- Add a fourth button
- Load the content for this button from the “content” array


ACADEMIND
It's a Modular Course!
Deep dives & advanced concepts
Authentication
Animations
NextJS
And More...!
Forms
Redux
Routing
React Query
Context API & useReducer
Behind the Scenes & Optimization
Data Fetching & HTTP Requests
Custom Hooks
Styling
Debugging
Refs & Portals
Side Effects
React Essentials
Essentials - Deep Dive
Essentials - Practice
Solid React Foundation: Core concepts every React developer must know
JavaScript Refresher
Getting Started
Refresher module, in case it's been some time since you last worked with JS

ACADEMIND

One Course, Two Paths

Standard Path

Recommended

Start with lecture 1 in section 1

Complete the course lecture by lecture & section by section

You'll learn React step-by-step, from the ground up & in great detail!

Summary Path

If you got limited time

Complete the "Summary" section

You'll get a good overview of basic & advanced React concepts

It's also a great refresher after finishing the course

ACADEMIND

How To Get The Most Out Of This Course

Meet the Prerequisites

Basic web dev & JavaScript knowledge is required

Use the JavaScript refresher if needed

Watch the Videos

Watch them at your pace

Slow me down or speed me up

Pause & rewind

Repeat sections

Practice!

Complete coding exercises

Pause & practice on your own

Build dummy demo projects

Help each other

Use code attachments if stuck

Ask & answer in the Q&A section

Find fellow developers on our Discord

React Code Must Be Transformed!

React Code

JavaScript code that typically uses JSX ("HTML in JavaScript")

Code is **transformed** & **optimized** (e.g., unnecessary whitespace is removed)

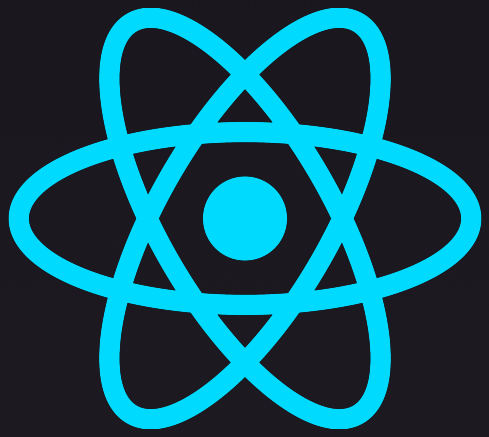
Code that runs in the browser

JavaScript code without JSX

Handled by build tool (e.g., Vite)

→ That's why a "more complex" project setup (which includes such a build tool) is needed

ACADEMIND



You Need A React Project

If you want to write React code

JavaScript Refresher

In case it's been some time...

Maximilian Schwarzmüller

This course section is optional!

It's recommended if you haven't used JavaScript in a while or if you don't have a lot of JavaScript experience

This section does not replace a JavaScript course!

But it will revisit crucial JavaScript concepts needed for building React apps

JavaScript Refresher

In case it's been some time since you last worked with JavaScript

- Core Syntax & Rules
- Essential, Modern JavaScript Features
- Key JavaScript Features Used In React Apps

Maximilian Schwarzmüller

JavaScript Can Be Executed In Many Environments

In the Browser
(i.e., as part of websites)

JavaScript code can be included in any website

The code then executes inside the browser (i.e., on the machine of the website visitor)

On any Computer
(e.g, server-side code)

Thanks to Node.js or Deno, JavaScript code can be executed outside of the browser, too

The code then executes directly on the machine

On mobile Devices
(e.g., via embedded websites)

With extra technologies like Capacitor or React Native, you can build mobile apps based on JavaScript

The code then executes on the mobile device

Adding JavaScript Code To A Website

Between <script> Tags

```
<script>
alert('Hello')
</script>
```

Can quickly lead to unmaintainable & complex HTML files

Typically only used for very short scripts

Via <script> Import

```
<script src="script.js"></script>
```

Separates HTML & JavaScript code

Maintaining complex JS-powered apps becomes easier

JavaScript code is just plain text!

How Code Is Executed

Code is read top to bottom,
left to right

JavaScript Code Consists Of “Statements”

Statement

The “thing” that gets executed

Keywords

Built into JavaScript

Required to “tell” JavaScript that a certain feature is used

`let`, `if`, `for`, …

Identifiers

Defined by developers

Used to identify commands (functions), variables (values) etc.

`alert()`, `age`, …

Values / Expressions

Defined by developers

Hardcoded values or expressions that produce new values

‘Hello world’, `5 - 3`, …

Keywords

Keywords enable language features

```
let, const, if, for, function, …
```

Identifiers

Identifiers identify “things”

Variables

Functions
(“Commands”)

Parameters

Property

JavaScript code is case-sensitive!

Identifiers Must Follow Certain Rules & Recommendations

#1

Must not contain whitespace or special characters (except $ and _)

Valid: $userName, age, user_name, data$, ...

Invalid: %userName, age/, user name, ...

#2

May contain numbers but must not start with a number

Valid: user3, us3r, ...

Invalid: 3user, 11players, ...

#3

Must not clash with reserved keywords

Valid: user, age, data, ...

Invalid: let, const, if, ...

#4

Should use camelCasing

Recommended: userName, isCorrect, ...

Uncommon: user_name, iscorrect, ...

#5

Should describe what the "thing" it identifies contains or does

Recommended: userName, isCorrect, loadData, ...

Uncommon: userDataPoint, correctness, dataLoader, ...

Semicolons are optional
(in most cases)!

Whitespace is ignored in many cases!

Use it to format your code & improve readability

But avoid adding too much whitespace

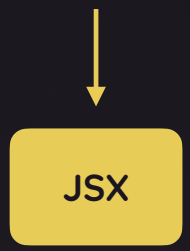
React projects use a build process

The code **you write** is **not** the code that gets **executed** (like this) in the browser

Your code is **transformed** before it's handed off to the browser

React Projects Use A Build Process

1. Raw, unprocessed React code **won't execute** in the browser



JSX is not a default JavaScript feature

2. In addition, the code would **not be optimized for production** (e.g., not minified)

React projects require a build process that transforms your code

create-react-app, vite etc. give you such a build process (no custom setup or tweaking needed)

There Are Different Types Of Values

String

Text values

Wrapped with single or double quotes

Can also be created with backticks (`)

```
"Hello World"

'Max'

`Hi there`
```

Number

Positive or negative

With decimal point (float) or without it (integer)

```
5
-23

3.14
-8.12
```

Boolean

True or false

A simple “Yes” or “No” value type

Typically used in conditions

```
true

false
```

Null & undefined

“There is no value”

undefined: Default if no value was assigned yet

null: Explicitly assigned by developer (reset value)

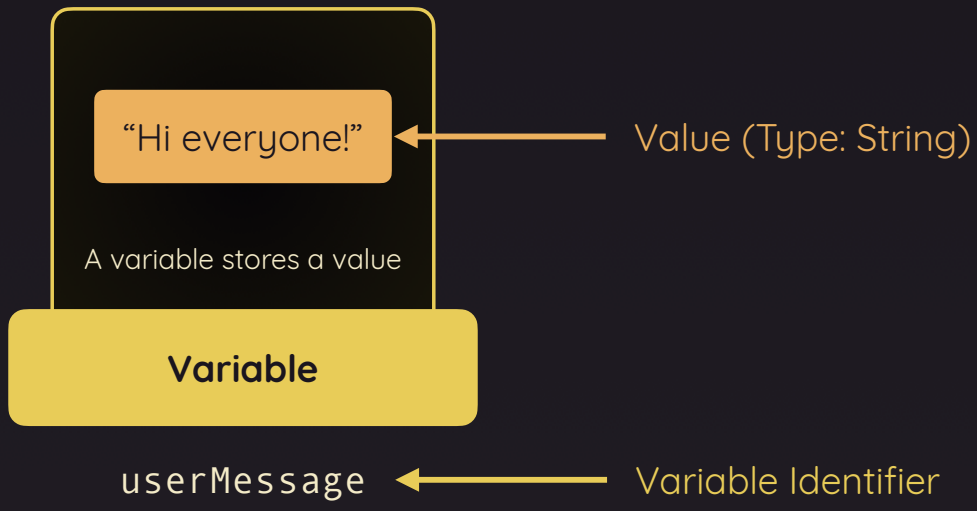
```
undefined

null
```

Additionally → **Objects**

Variables store Values

Variables Are Data Containers



Why Use Variables?

1

Reusability

Store a value in a variable once and use it as often and in as many places as needed

2

Readability

Organize your code over several lines rather than cramming everything into a single line

Variables vs Constants

Variables

Defined via `let`

Can be **re-assigned**
(i.e., the stored value can be overwritten)

```
let age = 34;
```

Allowed → `age = 29;` ✓

Constants

Defined via `const`

Cannot be **re-assigned**
(i.e., the stored value can't be overwritten)

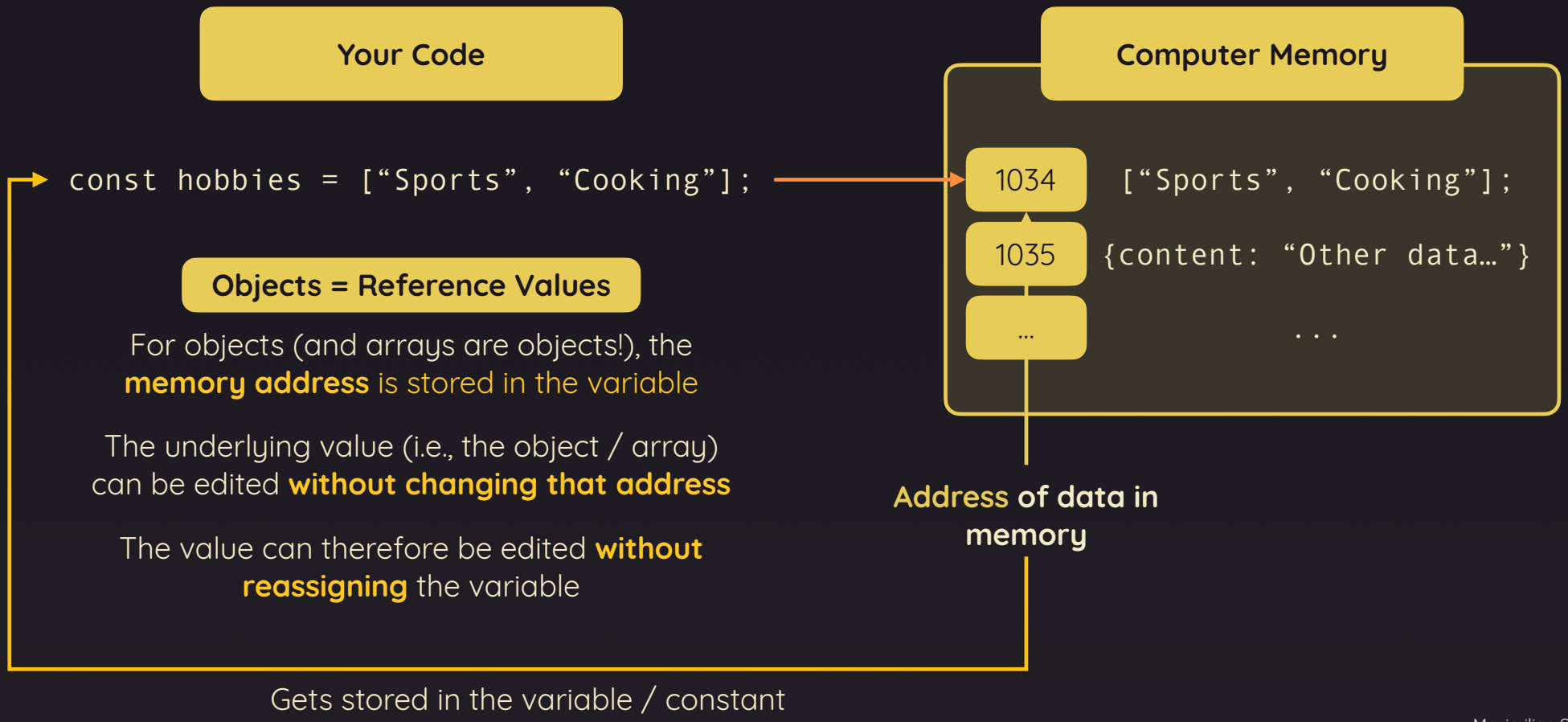
```
const age = 34;
```

✗ `age = 29;` ← Error

Values can be hardcoded

But they can also be derived via Expressions & Operators

Reference Values



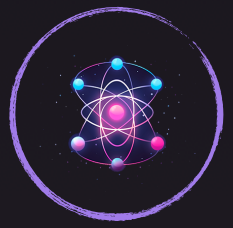
Objects = Reference Values

For objects (and arrays are objects!), the **memory address** is stored in the variable

The underlying value (i.e., the object / array) can be edited **without changing that address**

The value can therefore be edited **without reassigning** the variable

ACADE MIND



React Essentials

Components, JSX & State

- Building User Interfaces with **Components**
- Using, Sharing & Outputting **Data**
- Handling User **Events**
- Building Interactive UIs with **State**

About This Section

What you'll learn in this section

- How to build web user interfaces with React Components
- How to make those user interfaces interactive with State
- How to output static, dynamic & conditional data

What you'll be able to do after finishing this section

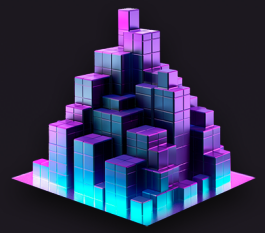
- Build basic, interactive React web applications
- Use React Components to build user interfaces with ease
- Handle user events & output data

What you should know before starting this section

- You must have basic JavaScript knowledge (variables, objects, arrays, functions, control structures, DOM)
- NO prior React knowledge is required!


ACADEMIND
E2E Testing
Props
JSX
Refs
Custom Hooks
Classes
Reducers
Components
Routing
Effects
Fragments
Redux
DevTools
Suspense
Hooks
Portals
State
Animations
Strict Mode
Styling
RSC
Context
NextJS
Unit Testing

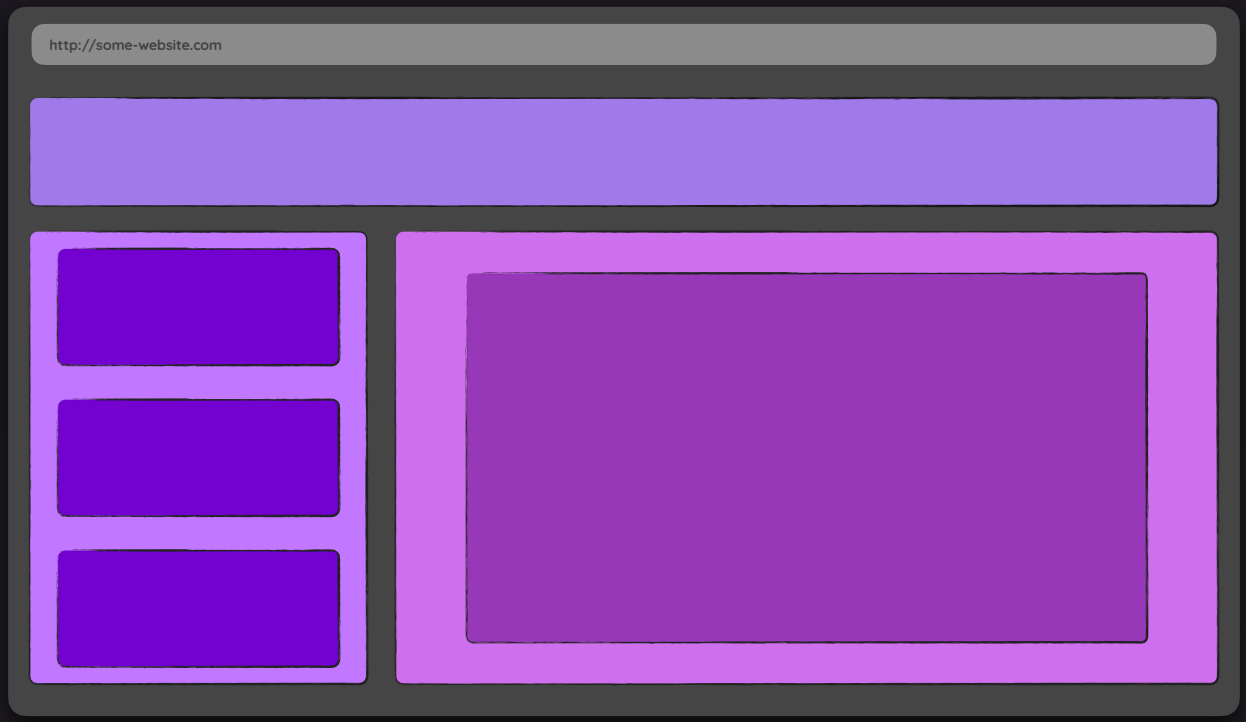
ACADEMIND



Components

ACADEMIND

Components Are UI Building Blocks

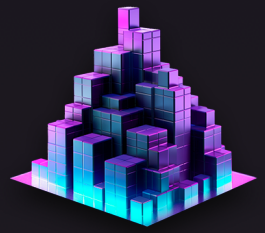


ACADEMIND

React Apps Are Built By Combining Components

ACADEMIND

Components Are The Foundation



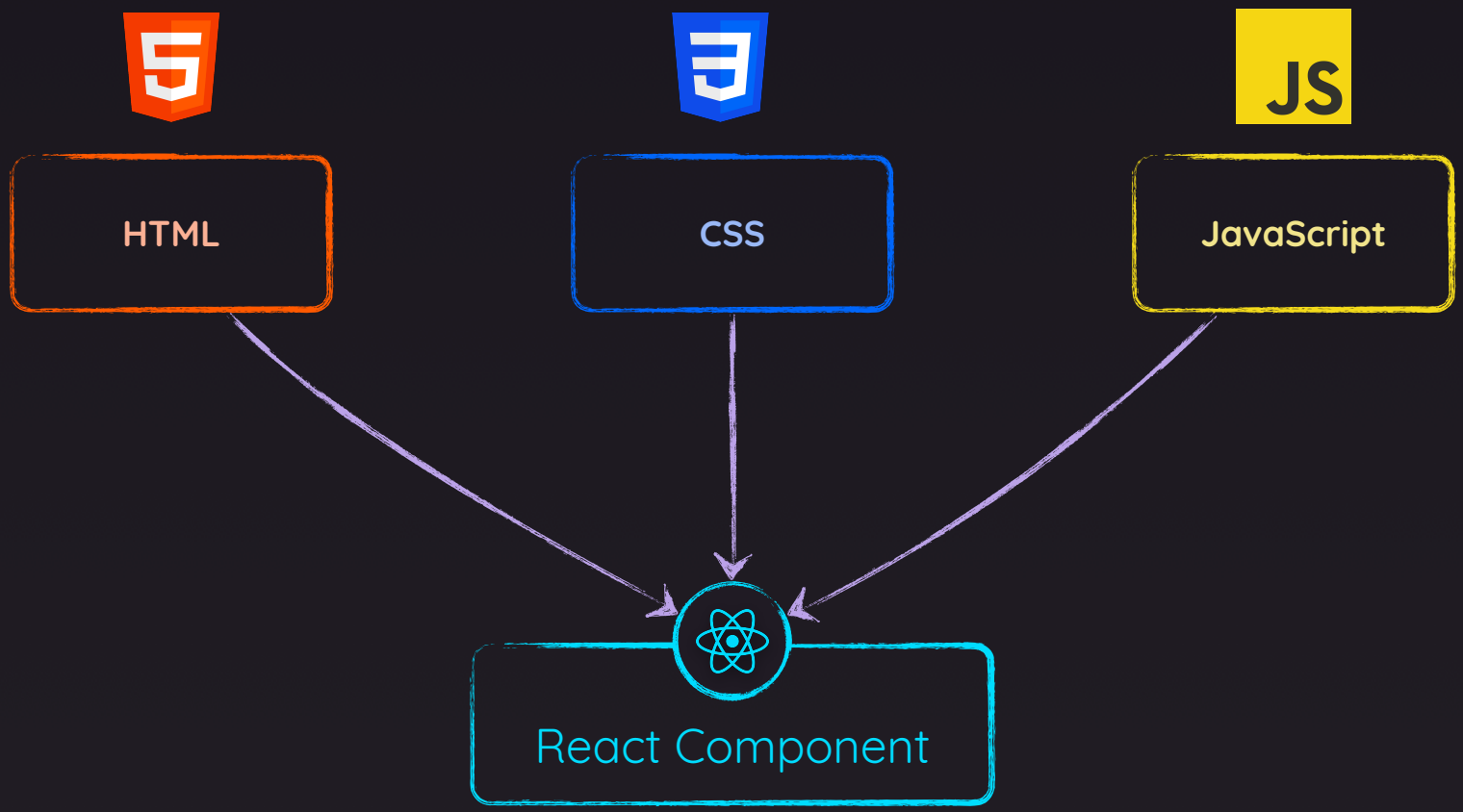
Components

JSX

State

Props

Creating Components



Build User Interfaces With Components

Any website or app can be broken down into
smaller building blocks: **Components**

It can therefore also be built by creating & combining such components

Header

Core Concept Items

Interactive Tabs

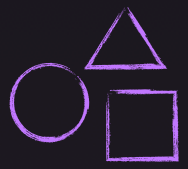
ACADEMIND

Why Components?

Reusable Building Blocks

Create **small building blocks** & **compose** the UI from them

If needed: **Reuse** a building block in different parts of the UI (e.g., a reusable button)



Related Code Lives Together

Related HTML & JS (and possibly CSS) code is **stored together**

Since JS influences the output, storing JS + HTML together makes sense

Separation of Concerns

Different components handle different data & logic

Vastly **simplifies** the process of working on complex apps

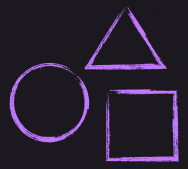
ACADEMIND

Why Components?

Reusable Building Blocks

Create **small building blocks** & **compose** the UI from them

If needed: **Reuse** a building block in different parts of the UI (e.g., a reusable button)



Related Code Lives Together

Related HTML & JS (and possibly CSS) code is **stored together**

Since JS influences the output, storing JS + HTML together makes sense

Separation of Concerns

Different components handle different data & logic

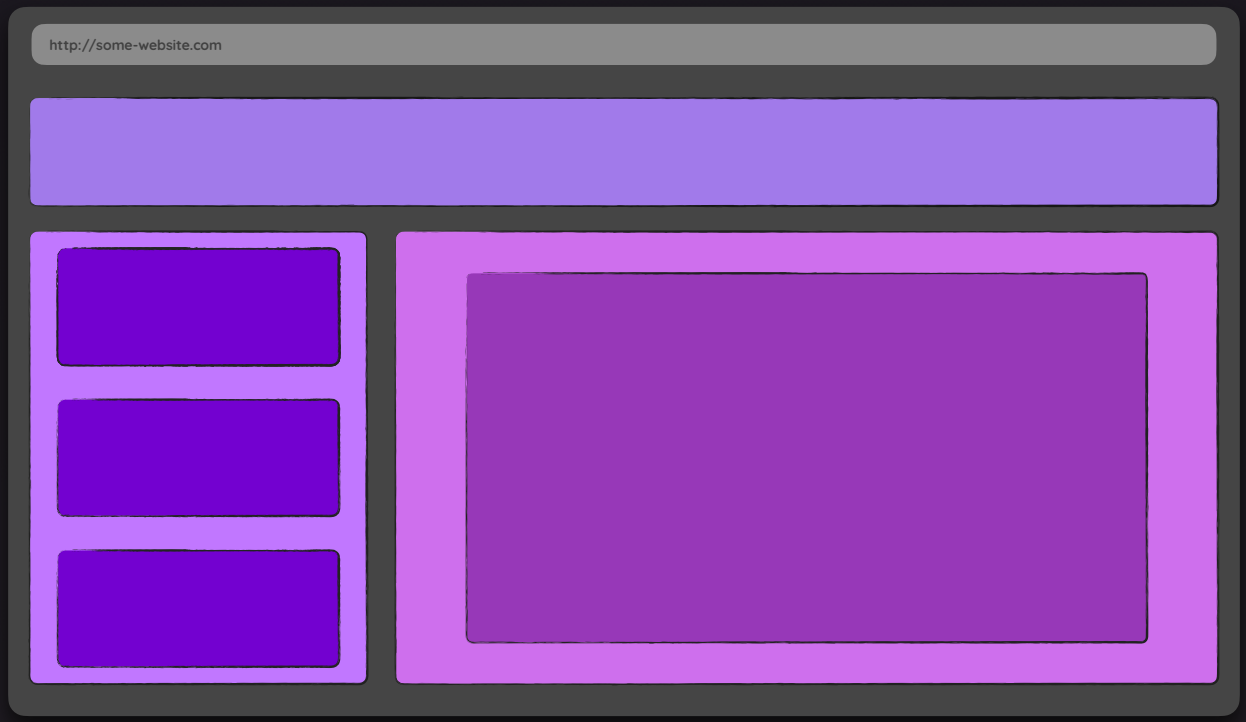
Vastly **simplifies** the process of working on complex apps

ACADEMIND

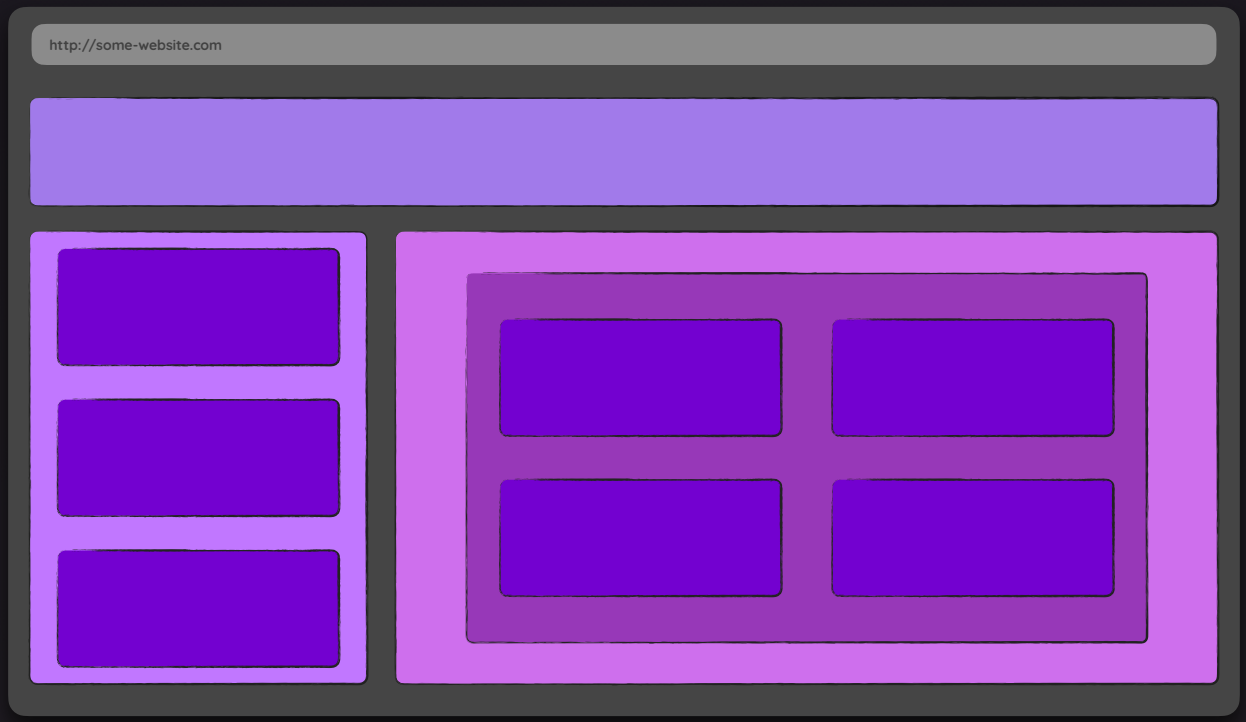
Components Can Potentially Be Reused

ACADEMIND

Components Can Potentially Be Reused



Components Can Potentially Be Reused



Components Can Potentially Be Reused

Header

Core Concept Items

The same component is used multiple times with different input data

Interactive Tabs

Different Components, Different Responsibilities

Header

Displaying main title, subtitle & image

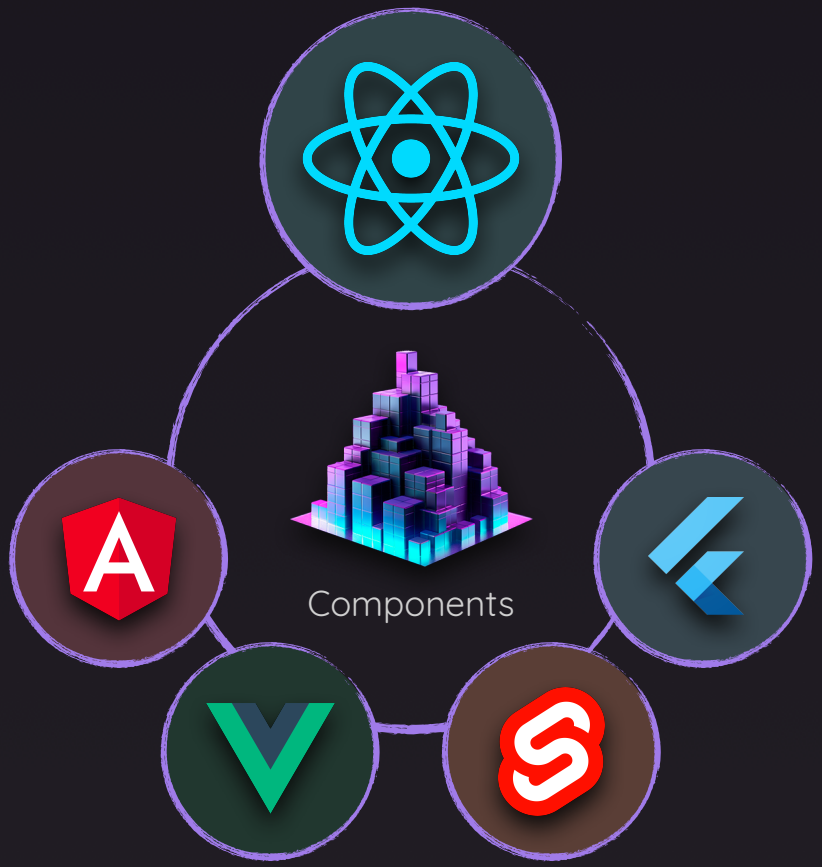
Core Concept Items

Outputting core concept information

Interactive Tabs

Handling user input & switching between tabs

Components Are A Fundamental Concept



Describe The Target UI With JSX

JavaScript Syntax eXtension

JSX

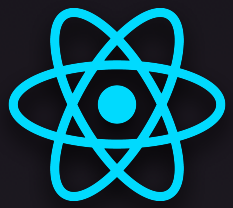
```
<div>
  <h1>Hello World!</h1>
  <p>JSX code is awesome!</p>
</div>
```

Used to describe & create HTML elements in JavaScript in a **declarative** way

BUT → **Browsers do not support JSX!**

React projects come with a **build process** that **transforms** JSX code (behind the scenes) to code that **does work** in browsers

ACADEMIND



With React, You Write Declarative Code

You define the target HTML structure & UI — not the steps to get there!

Component Functions Must Follow Two Rules

Name Starts With Uppercase Character

The function name **must start** with an **uppercase** character

Multi-word names should be written in **PascalCase** (e.g., “MyHeader”)

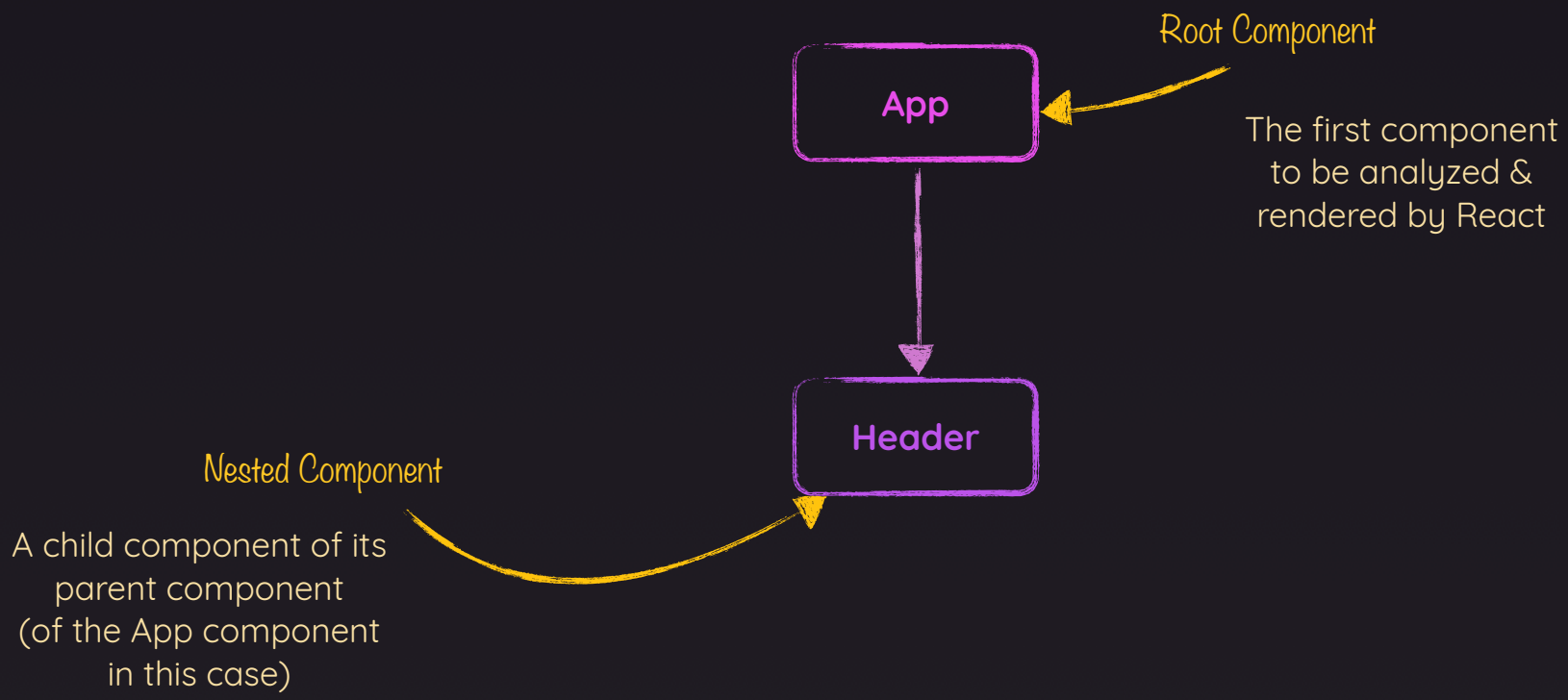
It’s **recommended** to pick a name that **describes** the UI building block (e.g., “Header” or “MyHeader”)

Returns “Renderable” Content

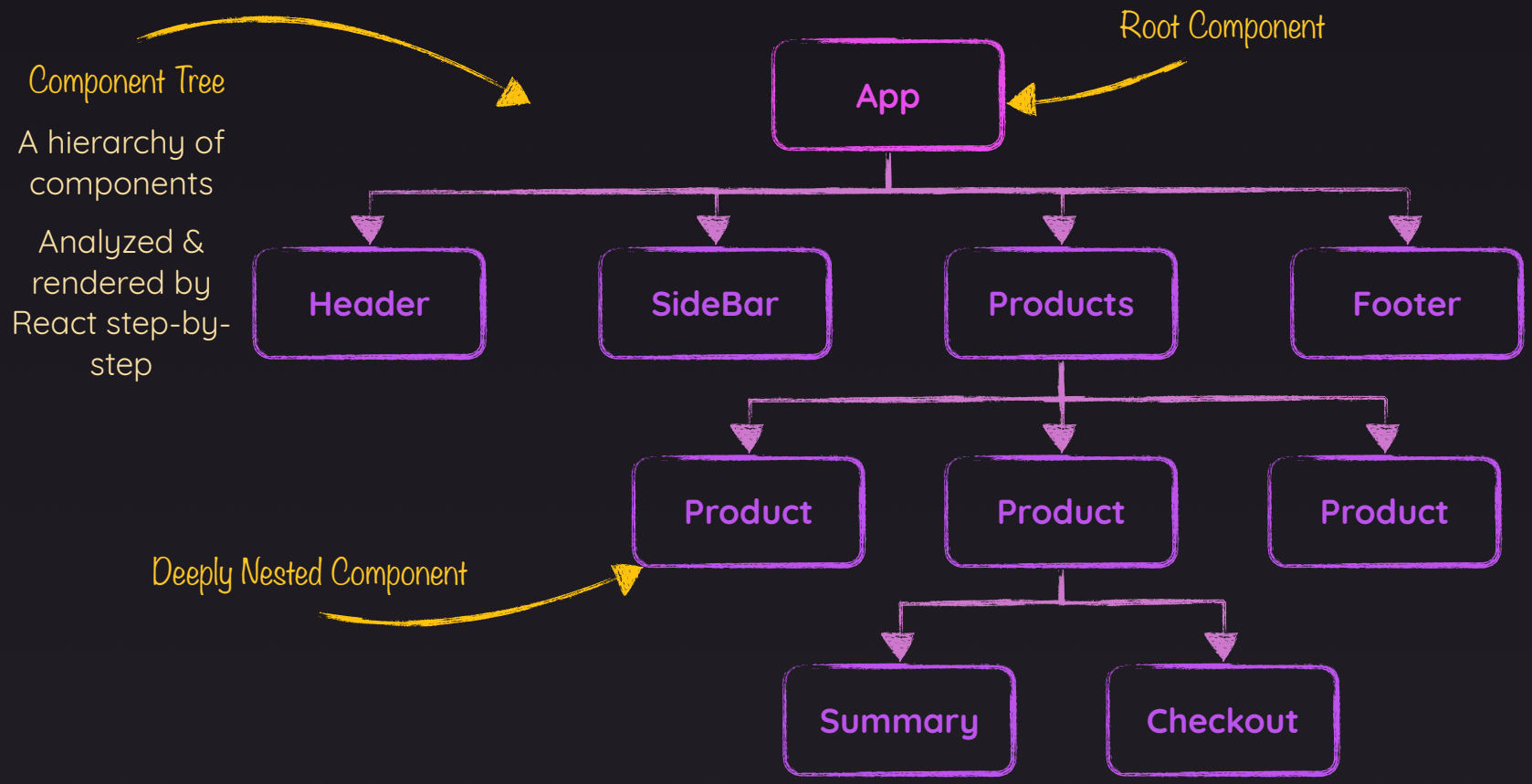
The function must **return** a value that can be **rendered** (“displayed on screen”) by React

In **most** cases: Return **JSX**
Also allowed: string, number, boolean, null, array of allowed values

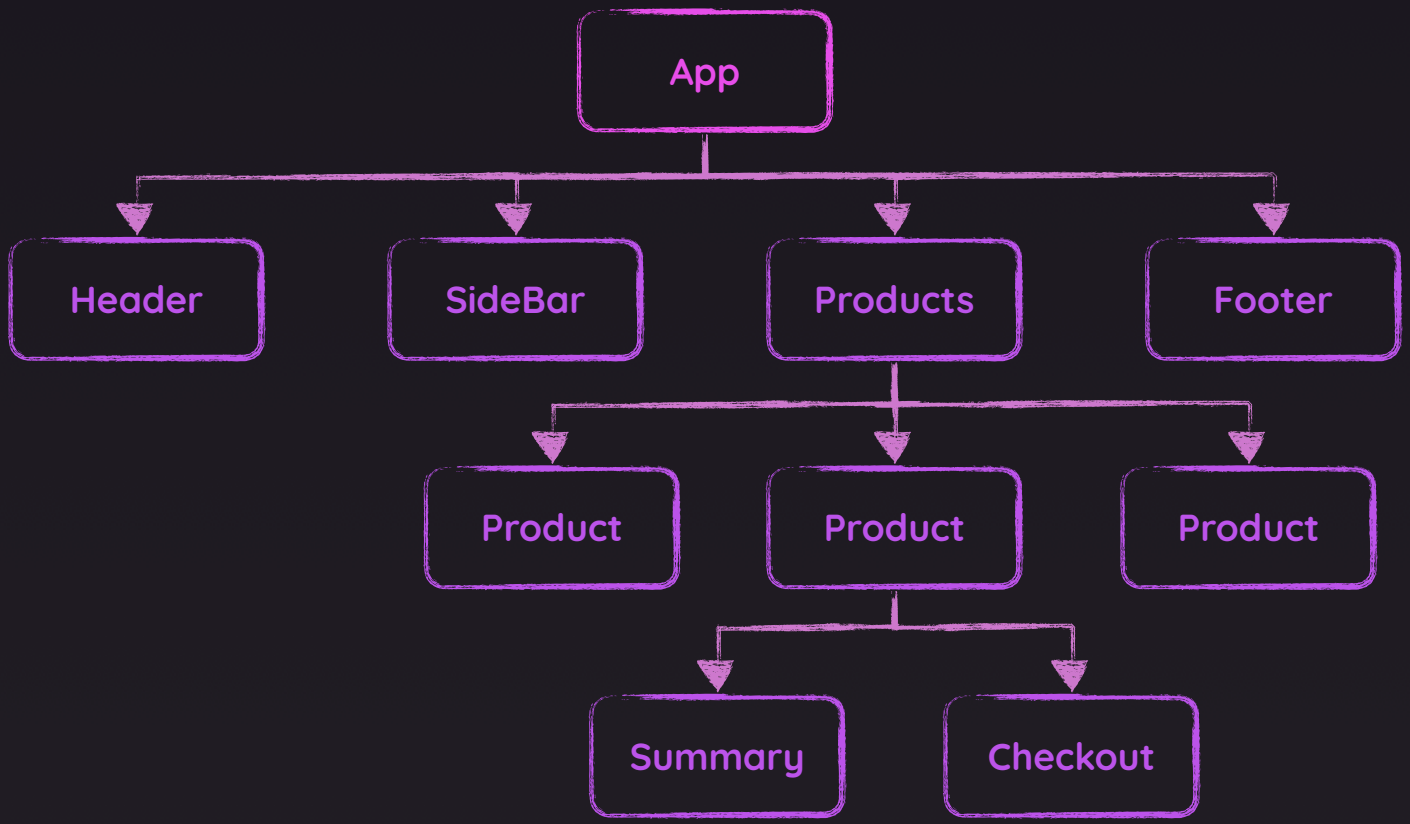
How Components Get Rendered



Building a Component Hierarchy



Building a Component Hierarchy



From Component Tree To DOM Node Tree

```
<div>
  <header>
    <img src="…" alt="…">
    <h1>React Essentials</h1>
    <p>Learn about all the core React concepts!</p>
  </header>
  <main>
    <p>Time to get started!</p>
  </main>
</div>
```

Built-in Components vs Custom Components

Built-in Components

Name starts with a **lowercase** character

Only **valid, officially defined** HTML elements are allowed

Are **rendered as DOM nodes** by React (i.e., displayed on the screen)

Custom Components

Name starts with **uppercase** character

Defined by you, "**wraps**" built-in or other custom components

React "**traverses**" the component tree until it has only built-in components left

Outputting Dynamic Content in JSX

Static Content

Content that's **hardcoded** into the JSX code

Can't change at runtime

Example

```
<h1>Hello World!</h1>
```

Dynamic Content

Logic that **produces the actual value** is added to JSX

Content / value is **derived** at runtime

Example

```
<h1>{userName}</h1>
```

React Projects & "The Build Process"

React projects must be "built" (via a build process) before deployment

The React code you write & test

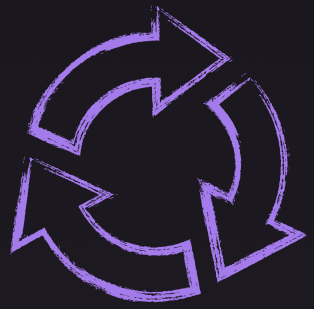
Changes & optimizes your code

Transforms it such that it runs in the browser

Also (potentially) optimizes other assets like CSS & image files

A collection of generated files that include your optimized code and any other extra assets (e.g., CSS code files, optimized images etc)

ACADEMIND

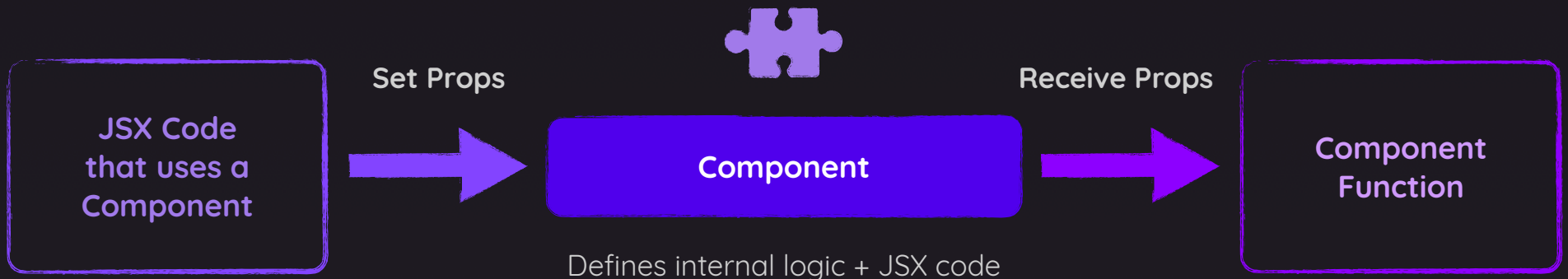


You Can Reuse React Components

But You Don't Have To!

Configuring Components With "Props"

React allows you to **pass data to components** via a concept called **"Props"**



Set component input data via "custom HTML attributes" (props)

Defines internal logic + JSX code that should be rendered

Receives props parameter with configuration data

Props Accept All Value Types

You're not limited to text values!

String value

Number value

Curly braces are required to pass the value as a number.

If quotes were used, it would be considered a string.

```
<UserInfo
  name="Max"
  age={34}
  details={{userName: 'Max'}}
  hobbies={['Cooking', 'Reading']}
/>
```

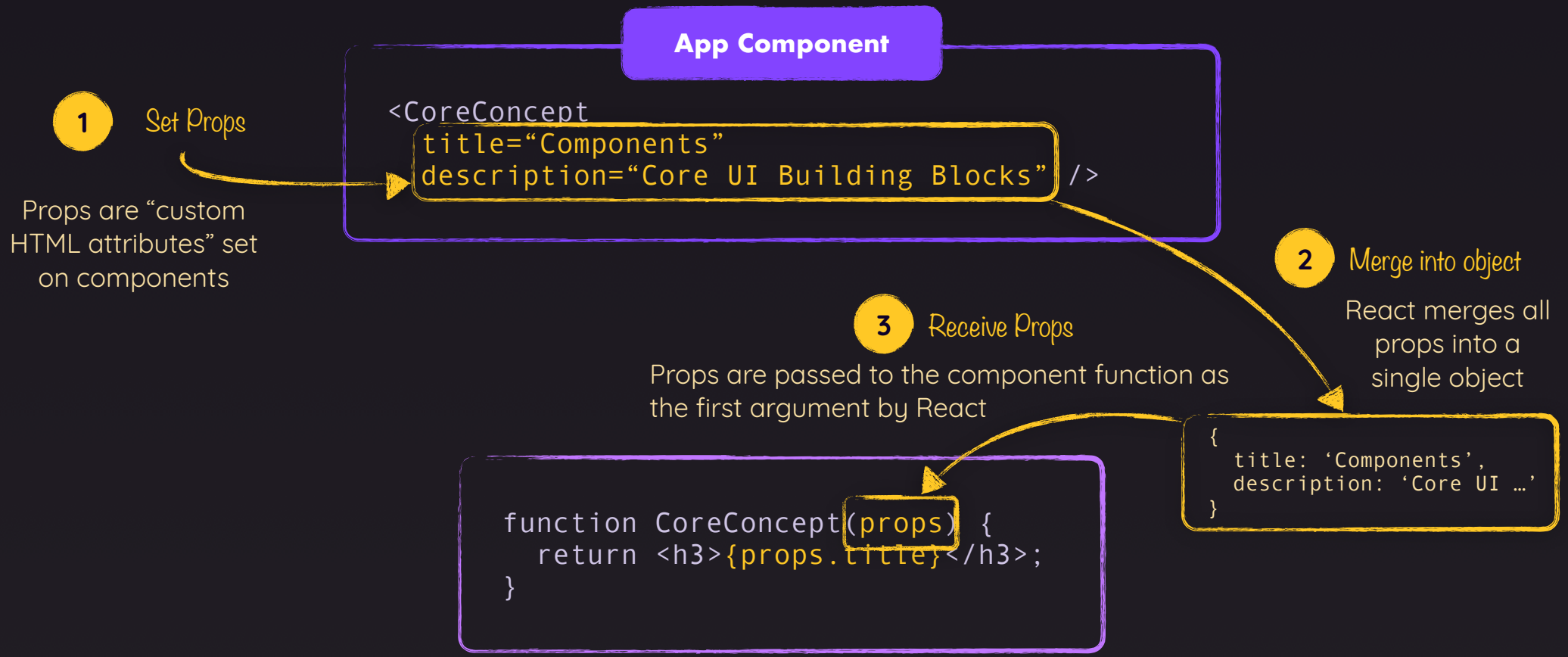
Object value

This is **no special "double curly braces" syntax!**

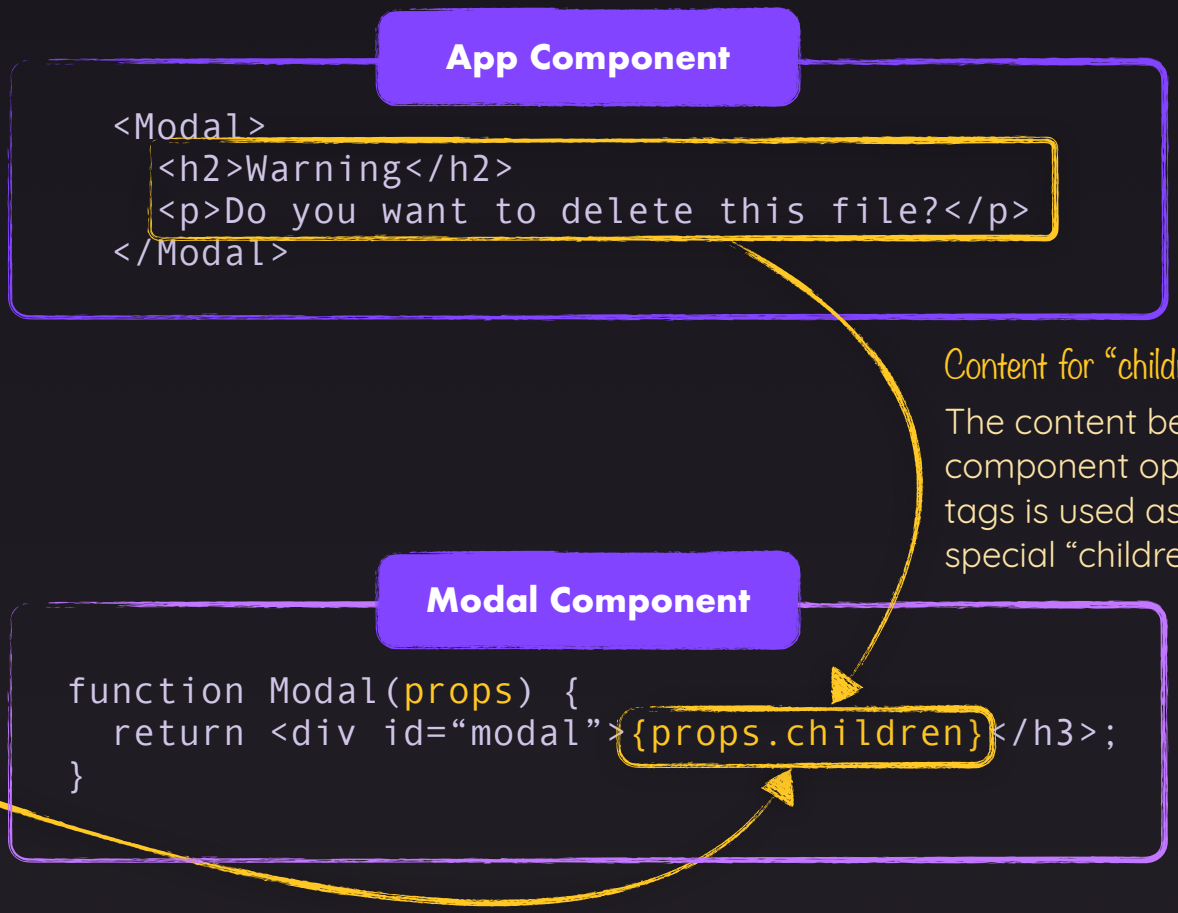
It's simply a JS object passed as a value.

Array value

Understanding Props



The Special “children” Prop



“children” Prop vs “Attribute Props”

Using “children”

```
<TabButton>Components</TabButton>
```

```
function TabButton({ children }) {
  return <button>{children}</button>;
}
```

For components that take a **single piece of renderable content**, this approach is closer to “normal HTML usage”

This approach is **especially convenient** when passing **JSX code as a value** to another component

Using Attributes

```
<TabButton label=“Components”></TabButton>
```

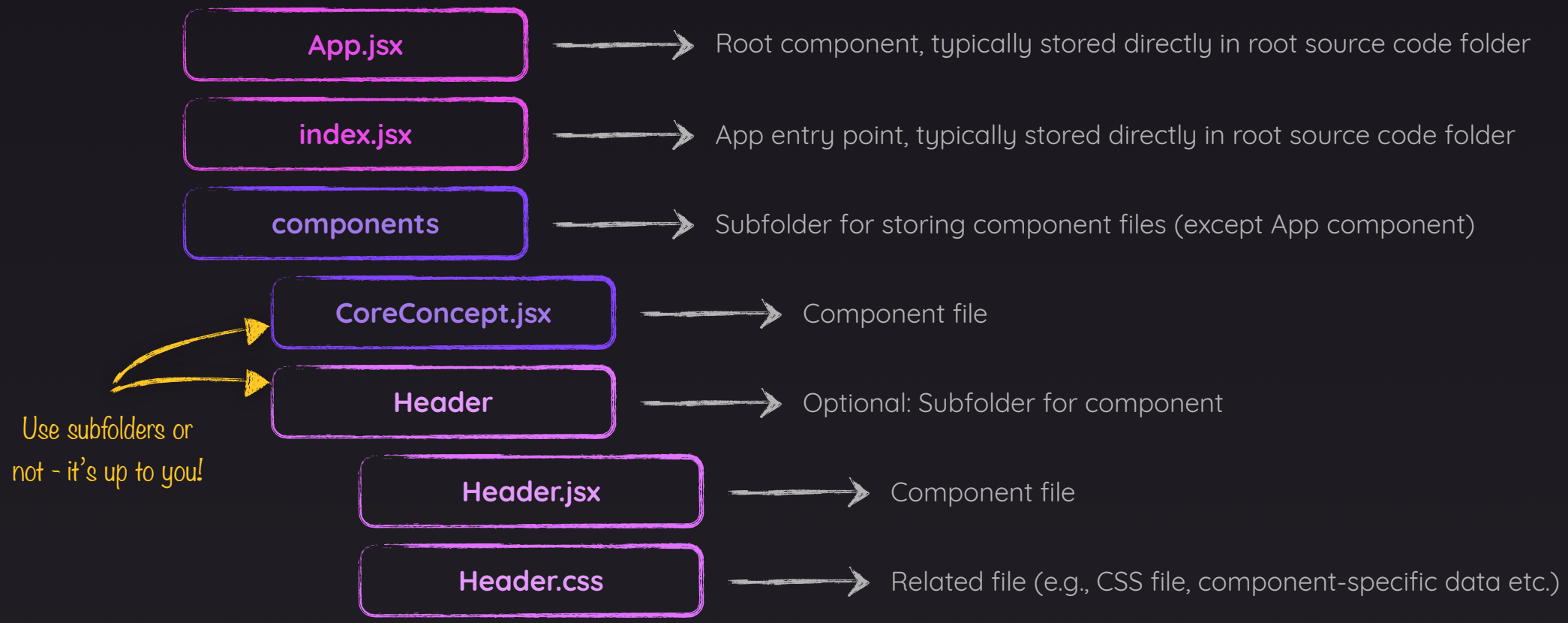
```
function TabButton({ label }) {
  return <button>{label}</button>;
}
```

This approach makes sense if you got **multiple smaller pieces of information** that must be passed to a component

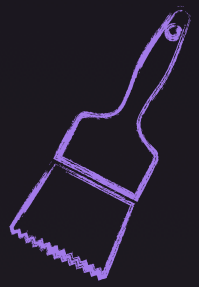
Adding **extra props** instead of just wrapping the content with the component tags mean **extra work**

Ultimately, it comes down to your use-case and personal preferences.

Structuring React Projects - The "src" Folder



ACADEMIND

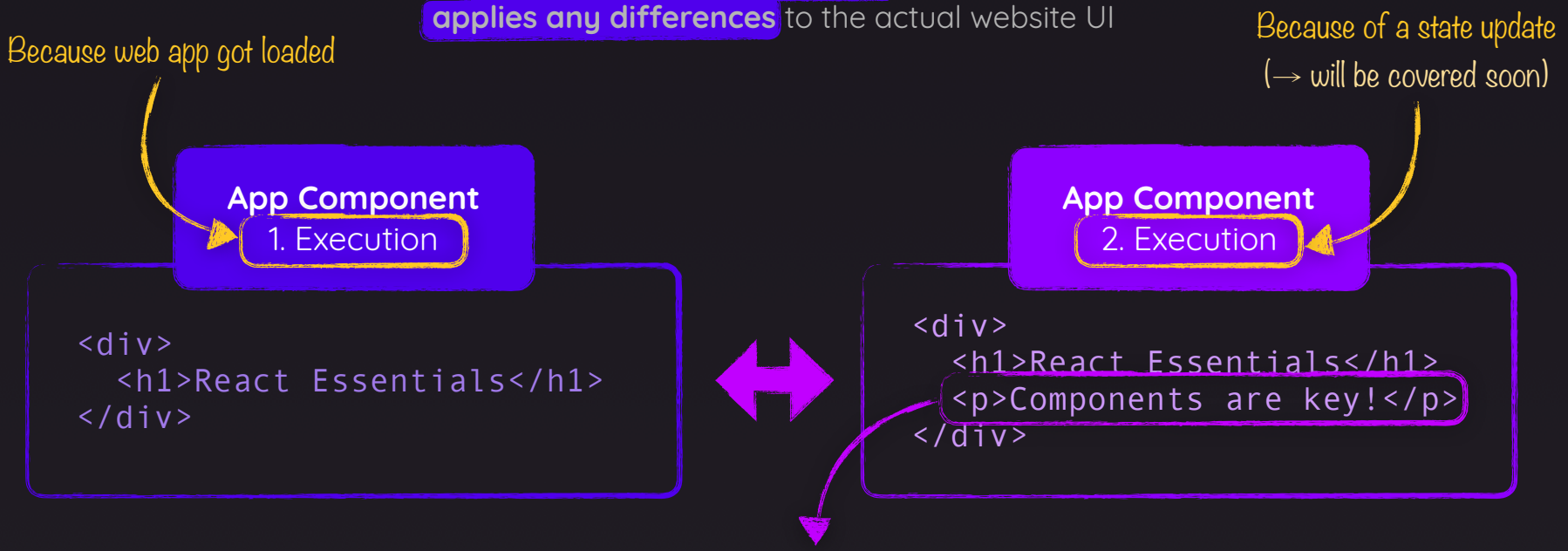


By Default, React Components Execute **Only Once**

You have to **"tell"** React If A Component Should be Executed Again

How React Checks If UI Updates Are Needed

React **compares** the **old output** ("old JSX code") of your component function to the **new output** ("new JSX code") and **applies any differences** to the actual website UI



Necessary updates will be applied to the real DOM, ensuring that the visible UI matches the expected output.

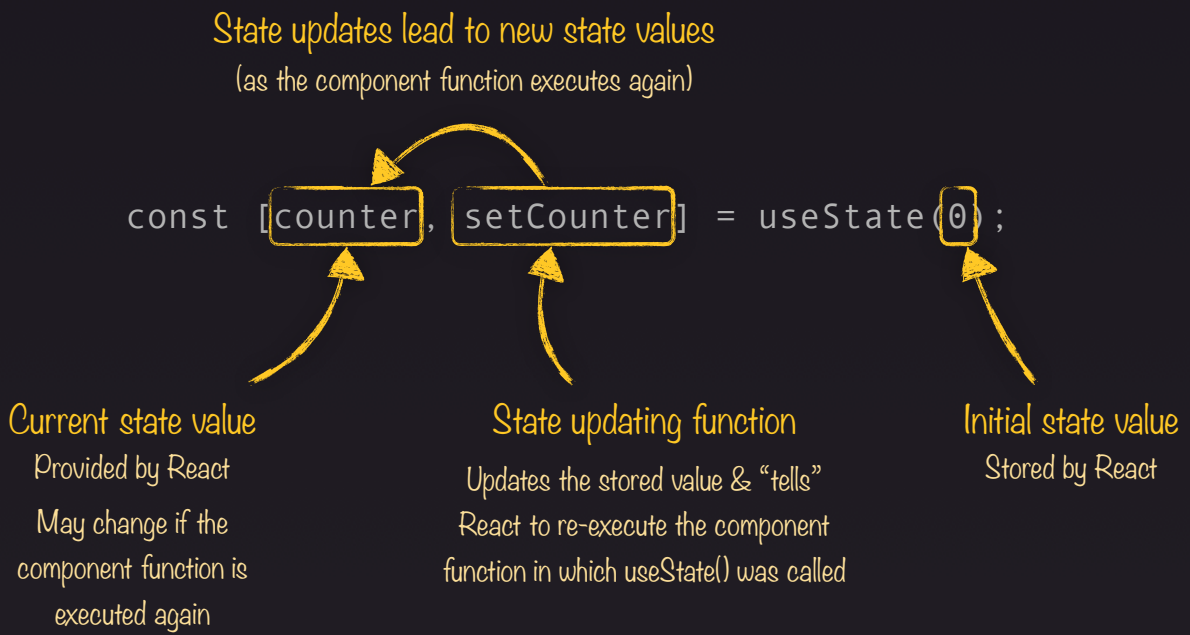
useState() Yields An Array With Two Elements

And it will always be exactly two elements

```
const stateArray = useState('Please click a button');
```

Manage State

Manage data & **"tell"** React to **re-execute** a component function via React's **useState()** Hook



ACADEMIND

Rules of Hooks

1 Only call Hooks inside of Component Functions

React Hooks must not be called outside of React component functions

```
function App() {
  const [val, setVal] = useState(0);
}
```

```
const [val, setVal] = useState(0);

function App() { … }
```

2 Only call Hooks on the top level

React Hooks must not be called in nested code statements (e.g., inside of if-statements)

```
function App() {
  const [val, setVal] = useState(0);
}
```

```
function App() {
  if (someCondition)
    const [val, setVal] = useState(0);
}
```

Transforming Data To JSX

```
[
  { userName: 'Max', age: 34 },
  { userName: 'Manuel', age: 35 },
  { userName: 'Marie', age: 38 },
]
```

Transform

```
[
  <User {...user[0]} />,
  <User {...user[1]} />,
  <User {...user[2]} />,
]
```

Output in JSX

```
<ul>
  <User {...user[0]} />
  <User {...user[1]} />
  <User {...user[2]} />
</ul>
```

ACADEMIND

Essential React Concepts

Components

Reusable building blocks which are used to build the overall app UI

Props

Component “attributes” (input data) used to configure components

JSX

JS syntax extension to describe HTML in JavaScript

State

React-managed data which, when changed, causes React to re-execute the related component function