

Redux Devtools:

- provide us debugging platform for Redux apps. It allows us to
- perform time-travel debugging and live editing.
- Some Features:
 1. it lets you inspect every state and action payload
 2. it lets you go back in time by 'cancelling' actions
 3. if you change the reducer code, each staged action will be re-evaluated
 4. if the reducers throw, we can identify the error and also during which action this happened
 5. with persistState() store enhancer, you can persist debug sessions across page reloads

React reactjs.org

A JavaScript library for building user interfaces

1. Declarative

Painless to create interactive UIs
update and render just the right components when your data changes
easier to debug

2 Component-Based

3. Learn Once, write Anywhere.

npx create-react-app pr1

npm -v
node -v
nodejs -v

cd pr1

npm start

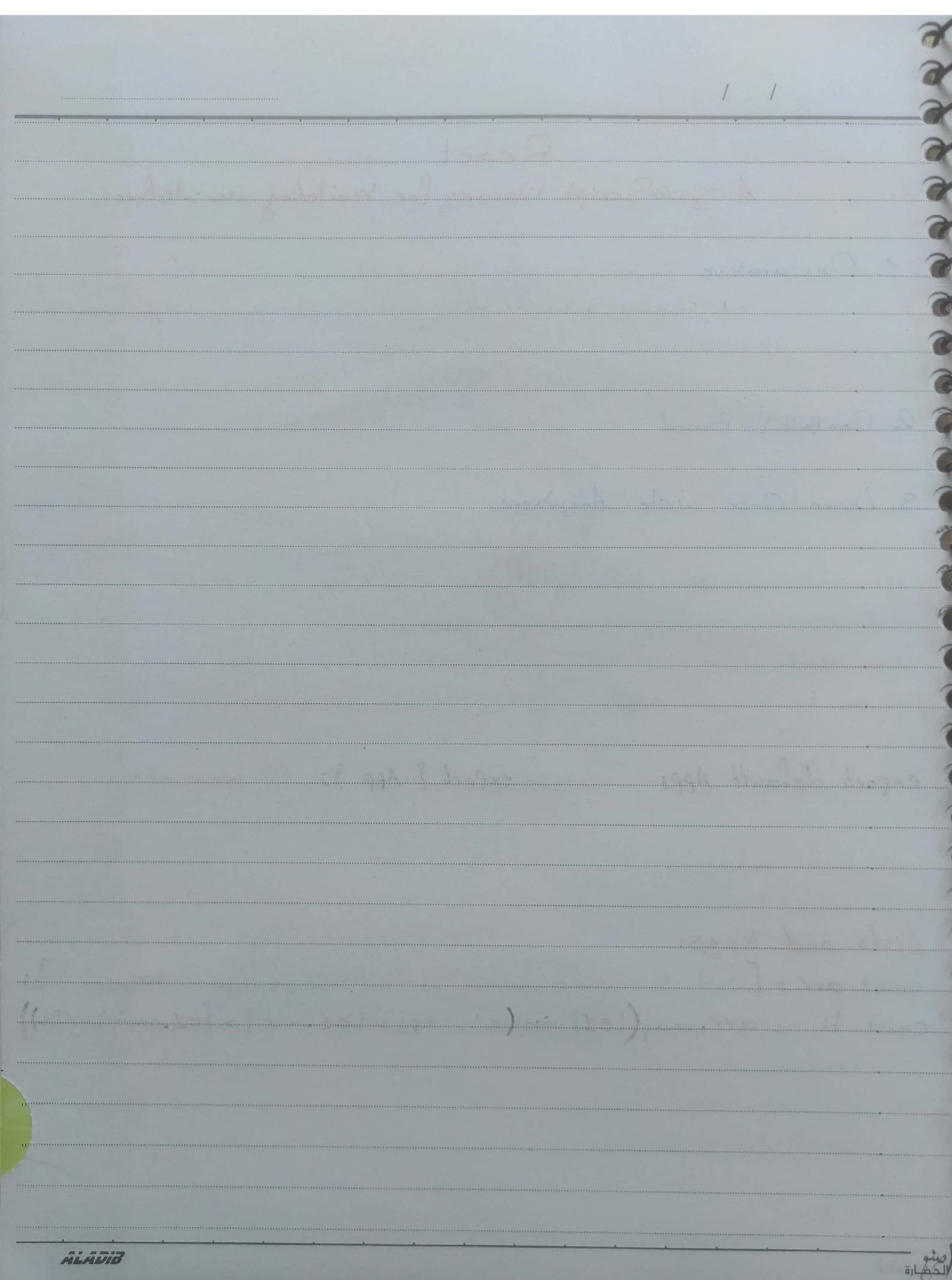
npm init

export default App; in export { App };

→ </> → Fragment

Lists and Keys:

```
const arr = [{id: 1, name: 'Anas'}, {id: 2, name: 'Attoum'}];
const li = arr.map((el) => (<li key={el.id}>{el.name}</li>))
```





Center Vertically:

1- Using line-height:

<div class="center">

<p> I am vertically and horizontally centered </p>

</div>

.center {

.center p {

① line-height: 200px; ② line-height: 150px;
② height: 200px; ③ display: inline-block;
text-align: center; ④ vertical-align: middle; } } }

2- Using position & Transform:

.center {

height: 200px;

① position: relative; } } }

.center p {

② margin: 0;

③ position: absolute;

④ top: 50%; left: 50%;

⑤ transform: translate(-50%, -50%); } } }

3- Using Flexbox:

.center {

① display: flex;

② justify-content: center;

③ align-items: center;

height: 200px; } } }

CSS Combinators:

div p { ... } جميع div يليه div p

div > p { ... } جميع div يليه div child

div + p { ... } أول div يليه p

div ~ p { ... } جميع div تلي div p

CSS Pseudo-class:

- a:hover MUST come after a:link and a:visited in the CSS definition in order to be affective!
- a:active MUST come after a:hover in the CSS definition in order to be affective!
- Pseudo-class names are not case-sensitive.

p : first-child { ... } أول p

<q lang="no" > quote </q> q:lang(no) { quotes: "n" "n"; }

CSS Pseudo-element:

p::first-line {

color: #ff0000;

font-variant: small-caps;

}

p.intro::first-letter {

color: #ff0000;

font-size: 200%;

}

w::before {

content: url(smiley.gif);

}

::marker {

color: red;

font-size: 23px;

}

::selection {
outline: 2px solid red;}

color: red;

background: yellow;

}

If you do not want to apply opacity to child element use **RGBA** color values

background: rgba(4, 170, 109, 0.6)

CSS Vertical Navigation Bar:

ul >

 Home

 News

 contact

 About

ul {

list-style-type: none;

margin: 0;

padding: 0;

width: 200px;

background-color: #f1f1f1;

}

li a {

display: block;

color: #0000;

padding: 8px 16px;

text-decoration: none;

}

li a.active {

background-color: #00A6D1;

color: white;

}

li a:hover: not(.active) {

background-color: #999;

color: white;

}

CSS Horizontal Navigation Bar:

1. Inline List Items:

```
ul {  
    list-style-type: none; display: inline;  
    margin: 0;  
    padding: 0;  
}  
li {
```

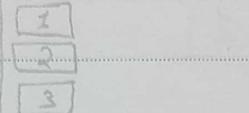
2. Floating List Items:

```
ul {  
    list-style-type: none;  
    margin: 0;  
    padding: 0;  
    overflow: hidden;  
}  
li {  
    float: left; display: block;  
    padding: 8px;  
    background-color: #cccccc;  
}  
li:last-child { float: right; } ← اداً يعنى في آخر عاليه
```

CSS Flexbox:

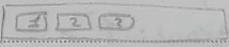
```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```

flex-container {
 ① display: flex;
 ② flex-direction: column;
 background-color: DodgerBlue; margin: 10px;
 text-align: center;
 line-height: 75px;
 font-size: 30px;
}



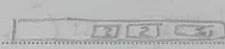
flex-direction: column

row



column-reverse

row-reverse



flex-wrap: wrap

1 2 3 4 5 6
7 8 9 10 11 12

nowrap

7 8 9 10 11 12

wrap-reverse 1 2 3 4 5 6

flex-flow: row wrap

→ shorthand

justify-content: center

1 2 3

space-around

1 2 3 4

flex-start

1 2 3

space-between

1 2 3 4

flex-end

1 2 3

stretch

1 2 3

stretch

align-items: center

top/bottom

stretch

1 2 3

to fill container

flex-start

1 2 3

baseline

1 2 3 4

flex-end

1 2 3

align-content: center

1 2 3

stretch

1 2 3

flex-start

1 2 3 4 5 6
7 8 9 ...

space-around

1 2 3 4 5 6

flex-end

1 2 3 4 5 6
7 8 9 ...

space-between

1 2 3 4 5 6

Use the `order` property to sort the flex items as you like:

<div class="flex-container">

1

2

```
<div style="order: 2">3</div>
```

</div>

2 3 1

Use the **flex-grow** property to specifies how much flex will grow relative to the rest of the flex items.

```
style="flex-grow: 1">>
```

```
style: "flex-grow: 1">...
```

style = "flex-grow: 2;"

2

③ style: "flex-shrink: 0" → 

١. الافتراضات ٢. اهمية الافتراضات ٣. امثلة على الافتراضات

④ style = "flex-basis: 200px"

style: "flex: 0 0 200px" → "flex: flex-grow flex-shrink flex-basis"

⑥ `style = "align-self: center"` → overrides the default alignment set by container's align-items property

CSS Variables - The var() Function:

identifying variables

Used to insert the value of a CSS variable.

Variable → Global (entire document)

→ Local (only inside the selector where it is declared)

```
:root {  
    --mainColor: #...;  
}  
  
⇒ h2 {  
    border-bottom: 2px solid var(--mainColor);  
}
```

h2 { local

```
--mainColor: #...  
}
```

Notes:

- 1 The variable name must begin with 2 dashes (--)
- 2 --color: ... is case sensitive
- 3 Local → Global (Local first)

Var(--name, value)

Required Optional

↳ The fallback value

(used if the variable is not found)

CSS 2D Transforms: تعلم الـ CSS

1. transform: translate(50px, 100px);

$$(50\text{px}) \Rightarrow x \overset{\%}{\rightarrow} 50\%$$

الـ 50px

2. transform: rotate(20deg)

3. transform: scale(2, 3) $\overset{x \text{ axis}}{\text{scale}(2)} \Rightarrow \text{original width}$

$$\text{original width} = 100\text{px} \Rightarrow 20\text{px} \quad \text{width increases} \leftarrow \text{scale(2)}$$

scaleX(2)

scaleY(0.5)

4. transform: skew(20deg, 10deg) skew(20deg) \Rightarrow بعدي X 10

skewX(20deg)

skewY(10deg)

transform-origin: right top; default $\rightarrow 50\%, 50\%$

transform: rotate(20deg) scale(2)

Learn about

transform: matrix(1, -0.3, 0, 1, 0, 0)

scaleX skewY skewX scaleY Tx Ty

implies

CSS 3D Transforms:

transform: rotateX(180deg) \rightarrow 90deg \rightarrow 180deg

rotateY(180deg)

rotateZ(90deg)

css Transitions: → allows you to change property values smoothly over a given duration

أي جملة لا تدخل منها الـ 2000 او ما فوق

transition-property: width;

transition-duration: 5s;

transition-delay: 5s;

معنـى يـمـكـنـكـ تـعـدـ لـ 5ـسـ اـخـرـ

transition-timing-function: ease; → سـلـسلـةـ سـلـسلـةـ

linear → سـلـسلـةـ

ease-in → سـلـسلـةـ

ease-out → سـلـسلـةـ

ease-in-out → سـلـسلـةـ

steps(8, start)

cubic-bezier(0.1, 0.7, 1.0, 0.1) cubic-bezier.com

transition: width 2s, height 2s, transform 2s;

shorthand:

transition: width 2s linear 1s;

property duration function delay

→ all

CSS Animations:

@keyframes example?

from { } 0%, 10%.

$t_0 \{ - \}$

3

animation-name: example;

اسناد

بـ ذـهـبـ بـرـجـعـ لـلـمـكـ

animation-duration: 4s;

دیانت

animation-delay: 2s;

إذا فتحت أي باب \Rightarrow يُبَيَّنُ نَاسٌ الْمُهَاجِرَةِ إِلَى الْمَغَالِ (أَكَلُوا مَا لَمْ يُحِلُّ لَهُ)

animationIterationCount: 3;

infinite

animation-direction: reverse;

alternate normal → reverse

alternate - reverse reverse - normal

animation-timing-function: (transitionevis)

animation-fill-mode: none انتهاء التحريك forwards (آخر تجربة حمل على آخر تجربة الائتمان) | animation-direction: alternate (تسلق تسلق الائتمان) |
Especially style when animation both backwards (دلالة (تبين انتشار التسلق)) delay
is not playing

shorthand:

animation: example 5s linear infinite alternate forwards;

CSS Attribute Selectors:

^{attribute}
a[target] { ... }

a[target = "blank"] { ... }

[title ~="flower"] → ~ contain → flower ✓ / summer_flower ✓ / my_flower ✗ / flowers ✗

[class |= "top"] | → exactly or followed by hyphen(-) [Case-Sensitivity: top vs top]

[class ^= "top"] ^ → start → top_content ✓ / top-content ✓ / content-topX

[class \$= "top"] \$ → end

[class *= "top"] * → contain →

button
input[type="text"] : focus [Not Working]

CSS Counters:

```
body { counter-reset: section; }
h2::before {
    counter-increment: section;
    content: "Section " counter(section) ". ";
}

```

Section 1: HTML Tutorial

Nested Counters:

```
body { counter-reset: section; }
h1 { counter-reset: subsection; }
h1::before {
    counter-increment: section;
    content: "Section " counter(section) ". ";
}
h2::before {
    counter-increment: subsection;
    content: counter(section) ". " counter(subsection) ". ";
}

```

Section 1: HTML/CSS Tutorials

1.1 HTML

1.2

Counters: → to insert a string between different levels of nested counters

```
ol { counter-reset: section;
    list-style-type: none; }
li::before {
    counter-increment: section;
    content: counters(section, ". ") " ";
}
<ol>
    <li> item </li>
        <li> item
            <ol>
```

1 item

2 item

2.1 item

11

CSS Grid Layout Module:

نیز (اندھی)

Parent:

`display: grid; (block)`

display: inline-grid;

2 column-gap: 50px; row-gap: 50px; gap: 50px; 10ms

gap: 50px 100px;

③ `grid-template-columns: 200px 300px;`

50% 50%

عند المحتوى (رسائل) الـ auto و (الـ repeat) يعطى بخطاب $1fr$ $auto \rightarrow$ $repeat(4, 100px)$
 $1fr \quad 1fr \quad 1fr \quad 1fr \rightarrow repeat(4, 100px)$

grid-template-rows: ↑ في

5 justify-content: → يتيح لك تحديد موضع المحتوى (الكلمات) في العنصر

align-content: ↗

Child:

grid-column: 1 / 5; 1 → 4

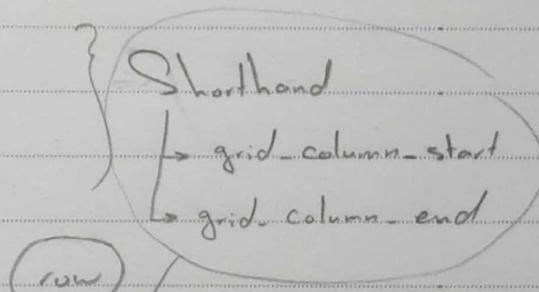
1 / span^{length} 5 ; 1 → s

من الابن 2 يذهب الى 5 و 6

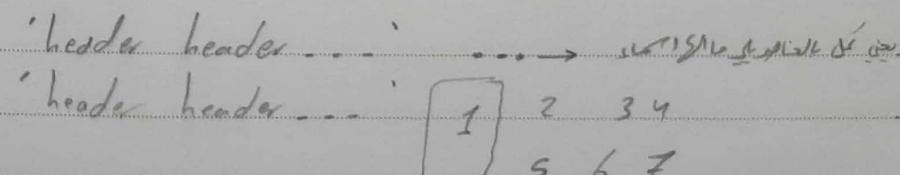
② grid row: ↑ inner

child grid-area: header;

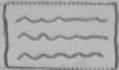
Parent grid template areas: 'header header header'



6 Shorthand salmon
row start start
grid-area: 1 / 2 / (6) / 16;
span 4



writing-mode: vertical-rl; → ↗



max-width: 20rem;

display: -webkit-box;
-webkit-line-clamp: 3;

-webkit-box-orient: vertical;

overflow: hidden;

background: linear-gradient (to right, rgba(67, 124, 205), rgba(69, 214, 202));

-webkit-background-clip: text;
-webkit-text-fill-color: transparent;

pointer-events: none; →

opacity: 0

نحوه مع الـ opacity ← Strong Combination

JS

Leetcode.com

is not required, but highly recommended

typeof null → object

This is a well-known bug in JavaScript
and has historical reasons.

JavaScript

Window.alert()

(Window).alert('hello world')

document.write('hello') (ancient HTML injection vulnerability)

console.log()

.error()

.table(['a', 'b', 'c'])

console.log("Eng. Anas AlHoum, color:red;font-size:30px;"
 ,"color:blue;font-size:30px;")

typeof (variable_name)

(undefined) → undefined

(null) → object

typeof Null / number !!

Var

Redeclare

yes

Access before declare

Undefined

Variable scope Drama

Yes, added to window

let

No

Error

no

const

No

Error

no

+ Unary plus: Return number if its not number.

- Unary negation: - - - - - and negates it

Datatypes

String - Number - BigInt - Boolean

Undefined - Null - Symbol - Object

`SafeInteger → -(263-1) + (263-1)`

3007199254740991 ✓

3007199254740992 X

عن `toFixed` + بحسب المقدار
انه المطلوب الدقيق (غير رقم العاشر)
وادع بـ `toFixed(10)` يتحقق رقم العاشر

Number Methods:

(100). `toString()` ↔ 100...`toString()` ⇒ 100 as a String

(100.5555).`toFixed(2)` ⇒ 100.56 اذا سأكتب مثل العاشرة فلن يتحقق
Number("100 Anas") ⇒ NaN (رسالة إنما تتحقق في المثلث) مثال
(+"100 Anas") ⇒ NaN

`parseInt("100 Anas")` ⇒ 100 بحث عن كل رقم ويشكل الأصفحة (رسالة إنما تتحقق)

("Anas 100") ⇒ NaN

(100.50) ⇒ 100

`parseFloat(100.50)` ⇒ 100.5 او ان فاتحة تبدا بـ . او اعدها مثلاً

`isSafeInteger`

Number.`isInteger(100)` ⇒ true false ← المطلوب

Number.`isNaN("Anas")` ⇒ false of datatype Number NaN is true حرج

Number.`isNaN("Anas"/20)` ⇒ true

`isNaN()` ↳ 'Hello' → false ↳ 'Hello.' → true

Math Methods:

`Math.round(99.2)` → 99

(99.5) → 100

`Math.ceil(99.2)` → 100

(99.5) → 100

`Math.floor(99.2)` → 99

`Math.min(100, 1, -100, 99)` → -100

.`max` → 100

`Math.pow(2, 4)` → 16

`Math.random()`

`Math.trunc(99.5)` → 99 بحث عن المثلث

String Methods:

let a = "Anas front developer";

console.log(a.indexOf("front"));

.indexOf("Front")

.indexOf("Anas", 8) مقدمة بـ 8

.indexOf("o")

diagram. lastIndexOf('o')

default string length

⑧ → 9 o index

.slice(2)

as front developer

(2, 5)

(2, 6)

last char (-5)

top

skip char 2 (-5, -3)

top

.repeat(2)

Anas front developer Anas front developer

.split('')

Array(3) ["Anas", "front", "developer"]

("", 6)

Array(6) ["A", "n", "a", "s", "f", "r", "e", "v", "e", "l", "o", "p", "e", "r"]

(' ', 2)

Array(["Anas", "Front"])

إذا تم تقطيعه في موضع المثلث سيدخل في كل قطعة

.length

20

let a = "Anas front developer";

Console.log(a.substring(2, 6));

swap ↪ previous (6, 2)

if size ↪ current (10, 6)

(a.length - 6)

(a.length - 6, a.length - 3)

a ↪ a.length - 1
substr(0, 6)

(-6)

(-5, 2)

.includes("s fro")

("s fro", 2) → default 0 → otherwise true.

.startsWith("z")

("s", 2) → default 0 → otherwise false.

("as", 2)

.endsWith("s")

("o", 6) → default string length → false.

("ro", 6)

true.

false.

false.

let newObj = Object.assign(..., ...); concat

/ /

JS Objects: Unordered Collection of Properties (can be changed, added, deleted)
and some are read-only)

const person = { firstName: "Anas", age: 23 }; full: Function
myVar.firstName // return this.firstName;

const person = {};
new Object() → no need to use it

Person.firstName = "Anas"; Person myVar → undefined

person[age] = 23 Person [myVar] → Anas

In JavaScript, almost "everything" is an object.

Objects Maths Functions Dates Arrays Maps Sets are objects.
All JavaScript values, except primitives, are objects.

JavaScript Primitives: is a value that has no properties or methods

A primitive data type is data that has a primitive value

JavaScript defines 7 types of primitive data types:

string - number - boolean - bigint - symbol - undefined - null

Primitive values are immutable (they are hardcoded and cannot be changed)

x = 3.14 → you can change the value of x but you cannot change the value of 3.14

Objects are Mutable → They are addressed by reference, not by value

const x = person; The object x is NOT a copy of person. The object x is person

x.age = 10; → change

Deleting Properties: delete person.age; or delete person[age];
→ deletes both value and property

Some Solutions to display JavaScript objects are:

1. Displaying the Object Properties by name.
2. : : : : in a loop.
3. : : : using Object.values()
4. : : : using JSON.stringify()

```
let text = "";  
for (let x in person)
```

text += person[x] + " is a " + gender + " from " + country + " and he is " + age + " years old.";

Note: you must use person[x] in the loop. Person.x will not work (Because x is the loop variable).

```
const myArray = Object.values(person); // Anas, 23, Syria
```

```
let text = "";  
for (let [fruit, value] of Object.entries(fruits))  
    text += fruit + ": " + value + "<br>";
```

```
let myString = JSON.stringify(person); // {name: "Anas", age: 23}
```

Object Constructor Functions:

Sometimes we need to create many objects of the same type

function Person (first, age) {

 this.firstName = first;

 this.age = age;

} this.nationality = "English" → Default

const mySelf = new Person ("Anas", 23);

mySelf.nationality = "English";

(mySelf now) nationality: English

Person.nationality = "English"; X

Person.prototype.nationality = "English"; ✓

Person = null; // Now value is null, but type is still an object

Person = undefined; // Now both value and type is undefined

(fruits instanceof Map)

Set

Array

Date

/ /

The void Operator: evaluates an expression and returns undefined.

This operator is often used to obtain the undefined primitive value

using "void(0)" useful when evaluating an expression without using the return value

```
<a href="javascript:void(document.body.style.backgroundColor='red');">
```

Click me

```
</a>
```

JavaScript Events:

HTML Events can be something the browser does, or something a user does.

```
<element event='some JavaScript'>
```

Common HTML Events:

onchange, onclick, onmouseover, onmouseout, onkeydown, onload

Arrays → use numbered indexes
Objects → use named keys

/ /

JavaScript Arrays:

const cars = ["Saab", "Volvo", "BMW"]; // easiest way (array literal)

or const cars = [];

cars[0] = "Saab";

cars[1] = "Volvo";

or const cars = new Array("Saab", "Volvo", "BMW"); // No need to use it

^{new} cars.all[0];

Access Element → let car = cars[0]; change Element → cars[0] = "Opel";

Array to String: // Saab, Volvo, BMW

cars.toString(); or cars.join();

String (سلسلة) ← المدخلات نفس النتيجة → cars.join()

to Recognize an Array:

Array.isArray(cars); or (cars instanceof Array);

Array Methods:

cars.length;

cars.toString();

cars.join(); ^{" "} → Separator

cars.push("a"); ^{a, b} يرجع المدخلات

cars.pop(); ^{last item} يخرج المدخلات

cars.unshift("a", "b"); ^{a, b}

cars.shift(); ^{first item} يخرج المدخلات

Easy way to append a new element to an array → cars[cars.length] = 'add';

delete cars[0]; // leaves undefined holes. So use pop() or shift instead.

cars.concat(fruitst); ^{a, b → push(), b} (المدخلات الجديدة) Fruits → cars duplication

cars.copyWithin(2, 0); copy to index 2, all elements from index 0

// Saab, Volvo, Saab → array "of ice cream" يكتب المدخلات غير المدخلات الناتج

Q374

Const myArr = [[1, 2], [3, 4]];

Const newArr = myArr.flat(); create new array with sub-array element concatenated to as specified depth.

Cars.splice(2, 0, "a", "b")
مقدمة مرفق دين حيث

برفع العناصر المختضنة / تغير العناصر

(0, 1) → وعكسه

toSpliced() → نفس اسماً تغير عناصر

Cars.slice(1);

Array Search:

cars.indexOf('a');

cars.includes('a');

numbers.find(c => {return c > 18});

ترجمة: تجد العنصر الأكبر من

findIndex(...);

findLast(...);

findLastIndex(...);

Sorting Arrays:

Cars.sort(); ترتيب عناصر ابتداءاً من العنصر الأول

(a, b) => {a - b};

Cars.reverse(); ترتيب عناصر ابتداءاً من العنصر الأخير

Cars.toSorted();

Cars.toReversed();

Cars.sort((a, b) => {a - Math.random()});

array亂序

The Fisher Yates Method

الخطوة 4

مراجع مفهومية
array.forEach (value, index, array)

map

filter / every / some

total / reduce / reduceRight

loop:

forEach →
map →
جج

filter →
array

find →
xegerValue

Math.min.apply(null, numbers)

array

Sorting Object Arrays:

const cars = [{ type: "Volvo", year: 2016 },
{ type: "BMW", year: 2010 }];

cars.sort((a,b) => {
return a.year - b.year
});

cars.sort((a,b) => {
let x = a.type.toLowerCase();
let y = b.type.toLowerCase();
if (x < y) return -1;
if (x > y) return 1;
return 0;});

numbers.filter(x => { return value > 18; })

18 هي الحد الأقصى للقيمة

numbers.reduce((x,y) => { return x+y; })

sum

reduceRight(...)

works from right to left

numbers.every(x => { return x > 18; })

18 هي الحد الأقصى للقيمة

numbers.sum(...)

18 هي الحد الأقصى للقيمة

const myArr = Array.from("ABCDEFG"); // A.B.C.D.E.F.G

const keys = cars.keys();

let text = ":

for (let x of keys) { text += x + '
' ; } // :

for (let x of cars.entries()) { text += x[0] + ' ' + x[1]; } // [0, "Volvo"]
[1, "BMW"]

const my = cars.with(2, "a"); // safe way to update elements in array without altering the original

Array Spread (...)

const q1 = [1, 2, 3];

const q2 = [4, 5, 6];

const q = [...q1, ...q2]; // 1 2 3 4 5 6

JavaScript HTML DOM

JavaScript - HTML Dom Methods.

In the Dom, all HTML elements are defined as objects

Property → value that can get or set

method → action you can do

Method	Property
<code>document.getElementById("demo").innerHTML = "Hello World!"</code>	

Finding HTML Elements:

1. Finding HTML Elements by id

2. by tag name

3. by Class name

4. by CSS selectors

5. by HTML object collections

```
const x = document.forms["form"];
```

```
let text = "";
```

```
for (let i = 0; i < x.length; i++) {
```

```
    text += x.elements[i].value + "<br>" ;
```

```
}
```

```
document.forms["myForm"]["fname"].value
```

DOM Navigation

- ↳ `nodeName` → tag name uppercase/read-only
- ↳ `nodeValue` → for element is null
- ↳ `nodeType` → read-only

`navigator.cookieEnabled` → True/false

/ /

JS HTML DOM Events.

1. `onclick`
2. `onload/onunload`
3. `oninput`
4. `onchange`

[source code body.innerHTML] [input type="text"] [input type="password"] [input type="checkbox"] [input type="radio"] → focus/focused

5. `onmouseover`
6. `onmouseout`
7. `onmousedown`
8. `onmouseup`

9. `onfocus`
10. `mousemove`

DOM Event Listener

`addEventListener(event, function, useCapture)`

↳ `false` (default) ⇒ bubbling inner first

↳ `true` ⇒ capturing outer first

`element.removeEventListener("mousemove", myFunction);`

Navigating Between Nodes:

`parentNode` `childNodes[Nodenumber]` `firstChild` `lastChild`
`nextSibling` `previousSibling` `nodeValue`

DOM Root Nodes:

allow access to the full document

1. `document.body.innerHTML`
2. `document.documentElement.innerHTML`

The most important `nodeType` properties are:

`ELEMENT_NODE` → 1 `ATTRIBUTE_NODE` → 2

`TEXT_NODE` → 3 `COMMENT_NODE` → 8

`DOCUMENT_NODE` → 9 The HTML Document itself (the parent of HTML)

`DOCUMENT_TYPE_NODE` → 10 `<!Doctype html>`

Creating New HTML Elements (Nodes)

```
const para = document.createElement('p');
const node = document.createTextNode('This is new');
para.appendChild(node);
document.getElementById('div1').appendChild(para);
```

element.appendChild(node) تجذب الـ node إلى element

element.insertBefore(node, element2) element2 هو node لا ينضم

element.removeChild() في ما يتعلّق بالـ browser

parent.removeChild(child) ← يحيط بالـ child

parent.replaceChild(para, child)

document.getElementsByTagName('div') → Returns an HTMLCollection

let x = document.getElementsByTagName('div') → Returns an HTMLCollection
Array.from(x).map() → Returns an Array

document.querySelectorAll('div') → Returns a static NodeList

childNodes Property Returns a live NodeList

الفرق بين document.querySelectorAll و document.querySelector

The Difference Between an HTMLCollection and a NodeList:

Both:

- are array-like collections (lists) of nodes (elements)
- have a length property

HTMLCollection:

- is a collection of document elements.
- can be accessed by their name, id, or index number.
- "always live collection Example: if you add a `<div>` element to a list in the DOM, the list in the HTMLCollection will also change.

NodeList: methods → `forEach()`, `map()`, `entries()`

- is a collection of document nodes (element nodes, attribute nodes and text nodes).
- can be accessed by their index number.
- "most often a static collection Example: if you add a `<div>` element to a list in the DOM, the list in NodeList will not change.

```
<div class="js">  
  JS <span>Div</span> &lt;i>span</i>  
</div>  
  
<a href="#">Google</a>  
  
<script>  
document.querySelector('.js').innerHTML: 'In JS <span>Div</span>&lt;i>span</i>'  
  .textContent: 'In JS <span>Div</span>'  
  .innerText: 'JS Div <span>'  
  
document.images[0]  
  .src  
  .getAttribute('src')  
  .setAttribute('src', 'hello') // return undefined  
  .attributes  
  .hasAttribute('src') true || false  
  .removeAttribute('src') // return undefined  
  .hasAttributes() true || false
```

JavaScript Destructuring does not change the original object

```
let {firstName, lastName} = person;          // firstName و lastName هما مسماً جديداً
let [firstName, lastName] = person;          // firstName و lastName هما مسماً جديداً
                                                    country: 'US'  
                                                    last name]
```

```
let name = 'W3Schools';
let [a, b, c, d, e] = name;
```

```
const fruits = ['Bananas', 'Oranges', 'Apples', 'Mangoes'];
let [fruit1, fruit2] = fruits;
let [fruit1, , , fruit2] = fruits;
let {[0]: fruit1, [1]: fruit2} = fruits;
const [a, b, ...rest] = numbers;
```

```
const fruits = new Map([['apple', 500], ['banana', 300]]);
for (const [key, value] of fruits) { }
```

[firstName, lastName] = [lastName, firstName]; Swapping Variables

JavaScript Browser BOM:

`window.innerHeight` } in Pixels / the browser viewport // 752 px
`.innerWidth` Not including toolbars and scrollbars // 576 px

`window.open()` open a new window

`.close()` close the current window

`.moveTo()` move the current window

`.resizeTo()` resize the current window

`(window).screen.width`

`.height`

`.availWidth` returns the width of the visitor's screen, in Pixels, minus interface features like the Windows Taskbar.

`.availHeight`

`.colorDepth` returns number of bits used to display one color

modern computers use 24 bit or 32 bit hardware for color resolution

24 bits: 16 777 216 different "True Colors"

32 bits: 4 294 967 296 different "Deep Colors"

Older Computer → 16 bits "High Color" resolution / Very old computers & old cell phones → 256 "VGA colors"

`.pixelDepth`

for modern computers Color Depth and Pixel Depth are equal

```
    alert('I am an alert box!');  
    window.confirm('Press a button') or Abbrechen  
    Prompt('please enter your name: ', Ans Moryn);  
    input #username
```

prev.

(Window).location.href: returns the URL of the current page

//https://asp

.hostname returns the name of the internet host (of the current page)

//www.w3schools.com

.pathname

site hostname

// /is/is.window.location.asp

.protocol //https

.port returns the number of the internet host port (of the current page)

if the port number is default, most browsers will display or nothing

default → 80 for http and 443 for https

.assign("https://com") load a new document

please see notes

(Window).history.back() // same as clicking the Back button in the browser

forward()

information about visitors browser

(Window).navigator.cookieEnabled; true || false

.appName; //Netscape Strange Enough

↳ is the application name for IE11, Chrome, Firefox and Safari

.appCodeName; Mozilla

.product; //Gecko product name of the browser engine

.appVersion; //5.0 (Windows)

.userAgent; returns the user-agent header sent by the browser to the server

.platform; //Win32

.language; //de

.online; //true

.javaEnabled(); //false

ALAAIB

gio
الجهاز

1 / 1

myVar =
(Window).setTimeout (()=> { alert('Hello') } , 3000) , milliseconds
.clearTimeout(myVar)

يُوَتَّفِدُ إِلَيْهَا مُنْفَلِّ

.setInterval (...);

.clearInterval (myVar);

document.cookie = "username=Anas Attoum; expires=Thu, 18 Dec 2013 12:00:00 UTC;
path=/";

what path the cookie belongs to. By default, the cookie belongs to the current page

let x=document.cookie; return all cookies in one string

(جِئْ إِلَيْكُوكِيَّةَ تُرْكِيَّةً) كُوكِيَّةَ فِي إِنْتِرْنَيْتُ

وَكُوكِيَّةَ لَيْكَيَّا فِي إِنْتِرْنَيْتُ

JavaScript JSON: JavaScript Object Notation

is a lightweight data-interchange format.

JSON is used to send data between computers.

JSON format is text only

JSON makes it possible to store JavaScript objects as text.

JSON names require double quotes

JSON names → must be strings, written with double quotes

JavaScript keys → can be strings, numbers, or identifier names

JSON values → must be string or number or object or array or boolean or null

JavaScript values → can be all of the above and function, date, undefined

new Date(), strings and 35w, etc.

```
const obj = JSON.parse('{"name": "Anas", "age": 23}');
```

reviver parameter: is a second parameter of the JSON.parse, checks each property, before returning the value

```
const obj = JSON.parse(text, (key, value) => {
  if (key === "birth") { return new Date(value); }
  else { return value; }
});
```

```
const obj = JSON.parse(text);
```

```
obj.age = eval("(" + obj.age + ")");
```

you should avoid using functions in JSON, the functions will lose their scope, and you would have to use eval() to convert them back into functions.

```
const myJSON = JSON.stringify(obj);
```

```
obj.age = obj.age.toString();
```

← جملة ابتدائية
stringify: value of string type

JavaScript Sets :

is a collection of unique values

Each value can only occur once in a Set.

The values can be of any type, primitive values or objects.

```
const letters = new Set(["a", "b", "c"]);
```

or

```
const letters = new Set();
```

letters.add("a"); // if you add equal element, only the first will be saved

```
letters.add("b");
```

```
letters.add("c");
```

• entries() →
• letters.keys() → because it has no keys
• letters.values()

```
for (const x of letters) { ... } / letters.forEach(...)
```

```
typeof letters; // Object
```

```
letters instanceof Set; // true
```

```
answer = letters.has("d"); // false
```

Object	Map	objects have a default keys but map doesn't
size x	✓	
directly iterable x	✓	
keys string or symbol	any datatype ordered	/ /

JavaScript Maps:

A Map holds key-value pairs where the keys can be any datatype.

A Map remembers the original insertion order of the keys.

```
const fruits = new Map([
```

```
  ["apples", 500],
```

```
  ["bananas", 300],
```

```
  ["oranges", 200]
```

```
]);
```

or

```
const fruits = new Map();
```

```
fruits.set("apples", 500);
```

```
fruits.get("apples"); // Return 500
```

`typeof fruits;` // Object

`fruits instanceof Map;` // true

```
fruits.delete("apples");
```

```
fruits.clear(); // removes all elements from a map
```

```
fruits.has("apples"); // true
```

entries , keys , values , forEach ,
key , object

TS

TypeScript Tutorial: Static Typing

- TypeScript is JavaScript with added syntax for types.
- TypeScript being a "Syntactic Superset" means that it shares the same base syntax as JavaScript, but adds something to it.
- TypeScript uses compile time type checking which means it checks if the specified types match before running the code, not while running the code.

installing the Compiler:

npm install typescript --save-dev → install the compiler

npx tsc → run

Configuring the compiler:

npx tsc --init → create tsconfig.json with the recommended settings with

```
{
  "include": ["src"],
  "compilerOptions": {
    "outDir": "./build"
  }
}
```

{
 you can open the file in an editor
 to add those options. This will configure the
 TypeScript compiler to transpile TypeScript files
 located in the src directory of your project.
 into JavaScript files in the build directory

TypeScript Simple Types: (Primitives)

main primitives

boolean - number - string - bigint - symbol

numbers

floating

symbol

used to create globally

unique identifiers

Type Assignment:

1. Explicit Type writing out the types (easier to read and more intentional)

```
let firstName: string = "Anas"
```

2. Implicit Type (guess the type) → Infer / shorter/faster to type and often used when developing and testing

```
const icon = JSON.parse("ss");
console.log(typeof icon) // number
```

This behavior can be disabled by enabling `noImplicitAny` as an option in a TypeScript's project `tsconfig.json`. That is a JSON config file for customizing how some of TypeScript behaves.

• boolean != Boolean

For this tutorial just know to use the lower-cased values, the upper-case ones are for very specific circumstances.

TypeScript Special Types:

Type: `any` → is a type that disables type checking and effectively allows all types to be used

```
let v: any = true;
```

```
v: string;
```

```
Math.round(v): // NaN
```

`any` can be a useful way to get past errors since it disables type checking, but TypeScript will not be able to provide type safety, and tools which rely on type data, such as auto completion, will not work.

Remember, it should be avoided at "any" cost...

Type: `unknown` is a similar, but safer alternative to "any"

TypeScript will prevent `unknown` types from being used.

`unknown` is best used when you don't know the type of data being typed. To add a type later, you'll need to cast it.

Casting is when we use the "as" keyword to say property or variable is of the casted type.

Type: `never` effectively throws an error whenever it is defined

rarely used, especially by itself, its primary use is in advanced generics

Type: `undefined & null` don't have much use unless "strictNullChecks" is enabled in

```
let y: undefined = undefined;
```

```
let z: null = null;
```

the `tsconfig.json` file

const names: string[] = [];
names.push("Anas");

const names: readonly string[] = ["Anas"]
names.push("Attoum"); //Error

Typed Arrays (tuple)

- A tuple is a typed array with a pre-defined length and types for each index.
- Tuples are great because they allow each element in the array to be known type of value.

let ourTuple: [number, boolean, string] = [5, false, "Anas"] //Error
ourTuple[2] = "Coding God was here" //Error

const ourreadonlyTuple: readonly [number, boolean, string] = [5, true, "The Real Coding God"]

Named Tuples: allow us to provide context for our values at each index

const graph: {x: number, y: number} = {ss: 2, gg: 3}

Destructuring Tuples:

const graph: [number, number] = [ss, gg]
const [x, y] = graph

TypeScript Object Types:

```
const car: {type: string, model: string, year?: number} = {
    type: "Toyota",
    model: "Corolla",
    year: 2009
};
```

{ is object literal
; is optional colon
? is optional value

Type Inference:

```
const car: { type: "Toyota" } = {
    type: "Ford";
    type: 2; // Error
```

Index Signatures:

can be used for objects without a defined list of properties.

```
const nameAgeMap: {[index: string]: number} = {}
nameAgeMap.Anas = 23;
nameAgeMap.Ahmed = "23"; // Error
```

TypeScript Enums:

`enum` is a special "class" that represents a group of constants (unchangeable variables).
 Enums come in two flavors: string and numeric.

```
enum CardinalDirections {
```

North → 1 ➔ cardinal

East ➔ cardinal

South

West

}

```
let currentDirection: CardinalDirections = North;
```

```
let currentDirection = "North"; // Error
```

string not allowed here

Technically, you can mix and match string and numeric enum values, but it is recommended not to do so.

TypeScript Union Types:

```
function printStatusCode (code: string | number) {  
  console.log(code);
```

```
  printStatusCode(404);  
  printStatusCode('404');
```

TypeScript Type Aliases and Interfaces: allow types to be easily shared between different variables/objects.

Type Aliases: allow defining types with a custom name (an alias) / string objects arrays

```
type carYear: number;
```

```
type CarType: string;
```

```
type Car = {
```

```
    year: CarYear;
```

```
    type: CarType;
```

```
}
```

```
const carYear: CarYear = 2001;
```

```
const carType: CarType = "Toyota";
```

```
const car: Car = {
```

```
    year: carYear;
```

```
    type: carType;
```

```
};
```

Interfaces:

interface Rectangle {

height : number,

width : number

}

const rectangle: Rectangle = {

height : 20,

width : 10

};

Extending Interfaces:

interface ColoredRectangle extends Rectangle {

color : string

};

const coloredRectangle: ColoredRectangle = {

height : 20,

width : 10,

color : "red"

};

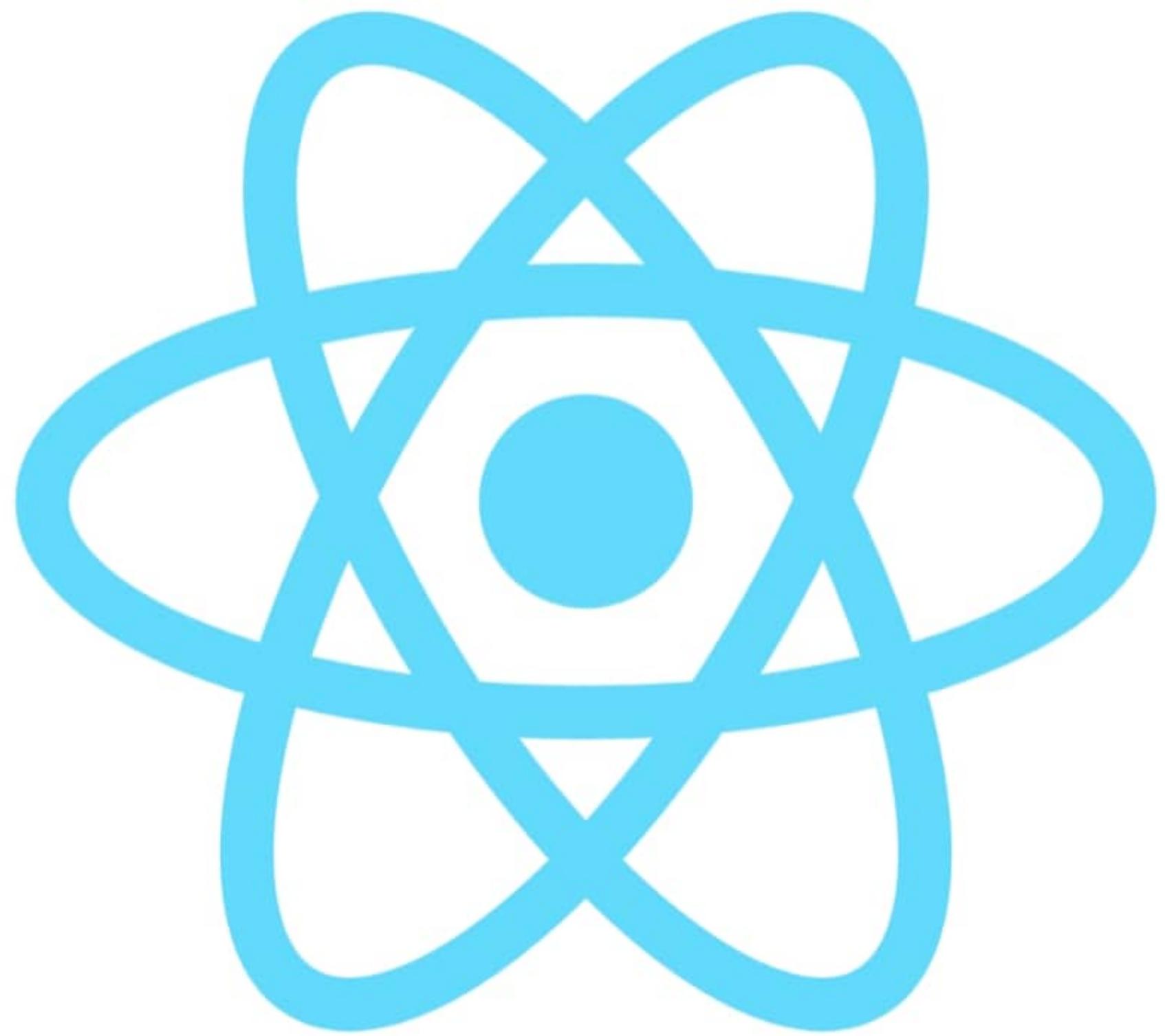
/ /

TypeScript Functions:

```
function getTime(): number {  
    return new Date().getTime();  
}
```

```
function printHello(): void {  
    console.log('Hello!');  
}
```

```
function multiply(a: number, b?: number): number {  
    return a * (b || 1);  
}
```



React: created by Facebook / is a tool for building UI component
is a JavaScript library for building user interfaces.
is used to build single-page applications.
allows us to create reusable UI components.

tool
`npx create-react-app my-react-app`
`cd my-react-app`
`npm start`

React create a VIRTUAL DOM in memory.
React only change what needs to be changed.

If you've previously installed `create-react-app` globally, it is recommended that you uninstall the package to ensure `npx` always uses the latest version of `create-react-app`.
To uninstall, run this command: `npm uninstall -g create-react-app`

To Upgrade to React 18

1. `npm i react@latest react-dom@latest`
2. use the new `rod` API (you can see it on React Upgrade tab)

JSX: stands for JavaScript XML [JSX not required, but easier]

allows us to write HTML in React.

- JSX is an extension of the JavaScript language based on ES6, and is translated into regular JavaScript at runtime.

- with JSX you can write expressions inside curly braces {}.

expression → React Variable or Property or any other valid JavaScript expression

const myElement = <h1>React is {5+5} times better with JSX</h1>;

- To write HTML on multiple lines, put the HTML inside parentheses:

const myElement = (

 Apples

 Bananas

);

- The HTML code must be wrapped in ONE top level element.

Alternatively, you can use a "fragment" to wrap multiple lines. This will prevent unnecessarily adding extra nodes to the DOM.

- Elements must be closed.

- ClassName (class keyword is reserved word in JavaScript) attributes translates into class attributes when JSX is rendered.

- React supports **if** statements, but not inside JSX.

To be able to use conditional statements in JSX:

I] put if statements outside of the JSX:

```
const x=5
```

```
let text = "Goodbye";
```

```
if (x<10) {
```

```
    text = "Hello";
```

```
}
```

```
const myElement = <h1> {text} </h1>;
```

R] Use **ternary expressions** instead:

```
const x=5;
```

```
const myElement = <h1> { (x) <10 ? "Hello" : "Goodbye" } </h1>;
```

React Components:

- are like functions that return HTML elements.
- independent and reusable bits of code.
- Components types
 - Function Components.
 - Class Components.
- In older React code bases, you may find class components primarily used. It is now suggested to use function components along with Hooks, which were added in React 16.8. There is an optional section on class components for your reference.
- Component's name MUST start with an upper case letter.

Class Component:

MUST include the `extends React.Component` statement.
it also requires a `render()` method

```
class Car extends React.Component {
  render() {
    return <h2> Hi, I am a Car! </h2>;
  }
}
```

Function Component:

```
function Car() {
  return <h2> Hi, I am a Car! </h2>;
}
```

The fileName MUST start with an uppercase character.

1 /

Props: stands for properties

function Car(props) {

return <h2> I am a {props.color} Car! </h2>;

}

<Car color="red"/> → هي امثلة لـ Component
↳ not a String? ⇒ color: { carName }

export default Car;

import Car from './Car.js';

Read Props are read-only!

React event handler are written inside curly braces:

onClick={shoot} instead of onclick="shoot()".

const shoot = (a, b) => { alert(b.type); } Return

return (

<button onClick={(event) => shoot("Goal", event)}>Take the shot!</button>
);

Pass an argument to an event handler. سيتم تفريغ

لذلك يُمكننا إدخال متغير إلى المثلث

function Goal(props) { 1. If Statement

```
const isGoal = props.isGoal;
if (isGoal) {
  return <MadeGoal/>;
}
```

```
return <MissedGoal/>;
```

}

2. Logical && Operator

```
{cars.length > 0 && <h2> You have {cars.length} cars in your garage! </h2> }
```

3. Ternary Operator

```
{isGoal ? <MadeGoal/> : <MissedGoal/>}
```

React Lists with Keys: without key → warning

Keys allow React to keep track of elements. This way, if an item is updated or removed, only that item will be re-rendered instead of the entire list.

function Garage() {

```
const cars = [{id: 1, brand: 'Ford'}, ...];
```

return (

<>


```
{cars.map((car) => <Car key={car.id} brand={car.brand}/>)}
```


</>

);

you can use the array index as a key.

Handling Forms:

- is about how you handle the data when it changes value or gets submitted
- In HTML, form data is usually handled by the DOM.
- In React, form data is usually handled by the components.
- when the data is handled by the components, all the data is stored in the component state.

```
import {useState} from 'react';
function MyForm() {
  const [name, setName] = useState("");
  return (
    <form>
      <label> Enter your name:</label>
      <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
      </label>
    </form>
  )
}

<form onSubmit={handleSubmit}>
  <input type="submit" value="Submit" />
</form>

const handleSubmit = (event) => {
  event.preventDefault();
  alert(`The name you entered was: ${name}`);
}
```

Multiple Input Fields:

```
const [inputs, setInputs] = useState({});  
  
const handleChange = (event) => {  
    const name = event.target.name;  
    const value = event.target.value;  
    setInputs(values => ({...values, [name]: value}))  
}  
  
return (  
            onChange={handleChange}  
);
```

React Router: is the most popular solution
create react app doesn't include page routing

npm i -D react-router-dom

@latest

← (in VS Code instead of VS Webpack)

src\pages\:

Layout.js

import { BrowserRouter, Routes, Route } from "react-router-dom";

return (

 <BrowserRouter>

 <Routes>

 <Route path="/" element={<Layout>} >

 default route for parent

 <Route index element={<Home>} />

 <Route path="blogs" element={<Blogs>} />

 <Route path="contact" element={<Contact>} />

 undefined URLs. This is great for <Route path="*" element={<NoPage>} />

 404 error page

 </Routes>

 </BrowserRouter>

);

Layout.js The layout route is a shared component that inserts common content on all pages
 import { Outlet, Link } from "react-router-dom";
 such as navigation menu

↳ used to set the URL and keep track of browsing history

const Layout = () => { Anytime we link to an internal path, we will use <Links>
 return (
 instead of

<>

<nav>

<Link to="/"> Home </Link>

</nav>

<Outlet /> — renders the current route selected

</>

)

};

export default Layout;

React Memo:

- Using memo will cause React to skip rendering a component if its props have not changed.
- This can improve performance.

```
import {memo} from "react"
```

```
export default memo(Todos);
```

- But react may still re-render it: memorization is a performance optimization, not a guarantee.

React Hooks: added to React in version 16.8

- allow function components to have access to state and other react features. Because of this, class components are generally no longer needed.
- allows us to 'hook' into react features such as state and lifecycle methods.

• 3 rules for hooks:

1. Hooks can only be called inside React function components.
2. ... at the top level of a component.
3. ... cannot be conditional.

- Hooks will not work in React class components.

useState Hook:

allow us to track state in function component.

- State generally refers to data or properties that need to be tracking in an application.

```
import {useState} from 'react';
```

:

```
const [color, setColor] = useState("red");
```

```
const [car, setCar] = useState({
```

```
  brand: "Ford",
```

```
  model: "Mustang",
```

```
});
```

```
setCar(values => { return { ...values, color: "blue" } });
```

useEffect Hook:

- allows you to perform side effects in your components.
- some examples of side effects are: fetching data, directly updating the DOM, and timers.

```
import {useState, useEffect} from 'react';
```

```
const [count, setCount] = useState(0);
useEffect(() => {
  setTimeout(() => {
    setCount(count + 1);
  }, 1000);
});
```

3); But wait!! It keeps counting even though it should only count once! useEffect runs on every render. That means that when the count changes, a render happens, which then triggers another effect.

- we should always include the second parameter which accepts an array. We can optionally pass dependencies to useEffect in this array.

1. No dependency passed \Rightarrow runs on every render.

2. An Empty array \Rightarrow runs only on the first render.

3. Props or state values \Rightarrow runs on the first render and any time any dependency value ^{change} value \neq

Effect Cleanup:

```
return () => clearTimeout(timer)
```

useEffect(
 () => {
 // cleanup code
 }, []);

useContext Hook:

- is a way to manage state globally.
- It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

The Problem:

- State should be held by the highest parent component in the stack that requires access to the state.

```
import {useState, createContext} from 'react';
```

```
const UserContext = createContext()
```

```
:
```

```
function Component1() {
```

```
  const [user, setUser] = useState({name: "Aman"});  
  return (
```

```
    <UserContext.Provider value={user}>
```

```
      <h1> Hello ${user.name}! </h1>
```

```
    </UserContext.Provider>
```

```
  );
```

```
}
```

```
import {useContext} from 'react';
```

```
:
```

```
const user = useContext(UserContext);
```

useRef Hook :

- allows you to persist values between renders.
- It can be used to store a mutable value that does not cause a re-render when updated.
- It can be used to access a DOM element directly.

import { useRef } from 'react';

```
const count = useRef(0); live const count = { current: 0 }
```

```
useEffect(() => { count.current + 1; }, []);
```

after render first

useRef() only returns one item. It returns an object called current.

Accessing DOM Elements:

```
const inputElement = useRef();
```

```
const focusInput = () => { inputElement.current.focus(); };
```

```
return (
```

```
  <input type='text' ref={inputElement} />
```

```
  <button onClick={focusInput}>Focus Input</button>
```

```
</> );
```

Tracking State Changes:

useRef || useState || useState & useEffect » لـ *لـ* *لـ*

useReducer Hook:

- Similar to the useState Hook.
- It allows for custom state logic.
- If you find yourself keeping track of multiple pieces of state that rely on complex logic, useReducer may be useful.

```
import {useReducer} from 'react';
const initialTodos = [{id: 1, title: 'Todo 1', complete: false}, ...];
const reducer = (state, action) => {
  switch (action.type) {
    case "COMPLETE":
      return state.map(todo => {
        if (todo.id === action.id) {
          return {...todo, complete: !todo.complete};
        } else {
          return todo;
        }
      });
    default:
      return state;
  }
};
```

function Todos() {

```
  const [todos, dispatch] = useReducer(reducer, initialTodos);
  const handleComplete = (todo) => {
    dispatch({type: "COMPLETE", id: todo.id});
  };
  return (
    <ul>
      {todos.map((todo) => (
        <div key={todo.id}>
          <label><input type="checkbox" checked={todo.complete} />
          onChange={(e) => handleComplete(todo)} />
          {todo.title} </label> </div>
      ))} </ul>
  );
}
```

- This is just the logic to keep track of the todo complete status.
- All of the logic to add, delete, and complete a todo could be contained within a single `useReducer`. ~~House~~ by adding more actions.

useCallback Hook:

- returns a memorized callback function.
- Think of memoization as caching a value so that it does not need to be recalculated.
- This allows us to isolate resource intensive functions so that they will not automatically run on every render.
- The useCallback Hook only runs when one of its dependencies update.
- This can improve performance.
- The useCallback and useMemo Hooks are similar. The main difference is that useMemo returns a memoized value and useCallback returns a memoized function.

Problem:

one reason to use useCallback is to prevent a component from re-rendering unless its props have changed.

useMemo Hook:

- The React useMemo Hook returns a memoized value.
- Think of memoization as caching a value so that it does not need to be recalculated.
- only runs when one of its dependencies update.
- This can improve performance.

Performance impact

5:27 abends | 4,0KB/s 📻



← src/UseReducerHook.js



```
1 import { useReducer } from "react";
2
3 export default function UseReducerHook() {
4     function reducer(count, action) {
5         if (action.type === 'increment') {
6             return ++count
7         }
8     }
9     const [count, dispatch] = useReducer(reducer, 0)
10
11    return (
12        <div>
13            <p>{count}</p>
14            <button onClick={() => { dispatch({ type:
15                'increment' }) }}>+</button>
16        </div>
17    );
}
```

[←](#) src/Context.js

```
1 import { createContext, useState } from "react";
2
3 export const contextData = createContext();
4
5 export function Context(props) {
6
7     const [a, setA] = useState('ljlknkl')
8     return (
9         <contextData.Provider value={[a, setA]}>
10            {props.children}
11        </contextData.Provider>
12    )
13
14 }
```

← src/index.js

```
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import App from './App';
4 import { Context } from './Context';

5

6 const root = ReactDOM.createRoot(document.
7 getElementById('root'));
8 root.render(
9   <React.StrictMode>
10     <Context>
11       <App/>
12     </Context>
13   </React.StrictMode>
14 );
```

[←](#) src/App.js

```
1 import { useContext } from "react";
2 import {contextData} from "./Context";
3
4 import UseReducerHook from "./UseReducerHook";
5
6 function App() {
7
8     const[a, ]=useContext(contextData)
9     // console.log("🚀 ~ App ~ useContext(contextData):",
10     useContext(contextData))
11
12     return (
13         <div>
14             Hello {a}
15             <UseReducerHook/>
16         </div>
17     );
18 }
19 export default App;
```



Redux Tutorial:

Redux Managing application's state with a single global object called Store.

Redux can have only a single store in your application.

State is Read-only → The only way to change the state is to emit an action, an object describing what happened. This means nobody can directly change the state of your application.

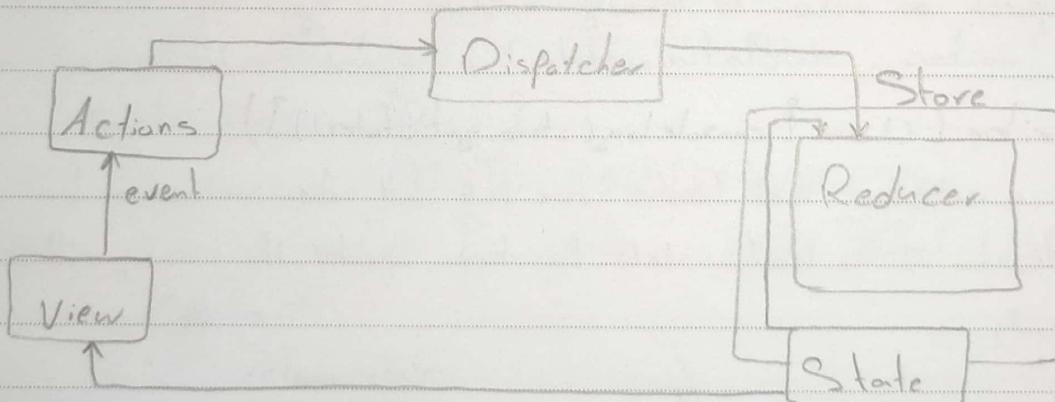
when ever the store is created → you need to specify the reducer.

`npm install --save redux`

`npm install --save react-redux`

`npm install --save-dev redux-devtools` If you do not want to install redux dev tools and integrate it into your project, you can install (Redux DevTools Extension) for Chrome and Firefox.

Redux reduces the complexity of the codes by enforcing the restriction on how and when state update can happen.



- An action is dispatched when a user interacts with the application.
- The root reducer function is called with the current state and the dispatched action.
- The store notifies the view by executing their callback functions.
- The view can retrieve updated state and re-render again.

```
import { createStore } from 'redux';
import reducer from './reducers/reducer';

const store = createStore(reducer);
```

createStore(reducer, [preloadedState], [enhancer])
is a function that returns the next state of app
↳ optional, it will help you enhance store with third-party capabilities.

A store has three important methods:

1. store.getState()

helps you retrieve the current state of your Redux store

2. store.dispatch({type: 'ITEMS_REQUEST'})

allows you to dispatch an action to change a state in your application.

const unsubscribe =

3. store.subscribe(() => {console.log(store.getState())})

Helps you register a callback that Redux store will call when an action has been dispatched. As soon as the Redux state has been updated, the view will re-render automatically.

Note: subscribe Function returns a function for unsubscribing the listener.

unsubscribe();

Actions :

are the only source of information for the store as per Redux official documentation. It carries a payload of information from your application to store.

{ type: 'GET_ORDER_STATUS', payload: { orderId, userId } }

- The structure of an action is totally up to the developer.

- It is recommended to keep your action object as light as possible and pass only the necessary information.

Pure Functions:

```
const initialState = {
  isAddedToCart: false
}
```

```
const addToCartReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'ADD_TO_CART_SUCCESS':
      return { ...state, isAddedToCart: !state.isAddedToCart }
    default:
      return state
  }
}
```

```
export default addToCartReducer;
```

Redux - Reducers: (state, action) \Rightarrow newState

are a pure function in Redux. Pure functions are predictable.

Reducers are the only way to change states in Redux.

```
const initialState = {
```

```
  isLoading: false,
```

```
  items: []
```

```
}
```

undefined state \rightarrow new

```
const reducer = (state = initialState, action)  $\Rightarrow$  {
```

```
  switch (action.type) {
```

```
    case 'ITEMS_REQUEST':
```

```
      return Object.assign({}, state, {
```

```
        isLoading: action.payload.isLoading
```

```
)
```

```
    case 'ITEMS_REQUEST_SUCCESS':
```

```
      return Object.assign({}, state, {
```

```
        items: state.items.concat(action.items),
```

```
        isLoading: action.isLoading
```

```
)
```

```
    default:
```

```
      return state;
```

```
)
```

```
)
```

```
export default reducer;
```

/reducer/orderStatusReducer.js

```
import { GET_ORDER_STATUS } from '../constants/appConstant';
GET_WISHLISTITEMS o similar to ...  
export default function (state = {}, action) {
  switch (action.type) {
    case GET_ORDER_STATUS:
      return { ...state, orderStatusData: action.payload.orderStatus };
    default:
      return state;
  }
}
```

/reducer/index.js

```
import { combineReducers } from 'redux';
import OrderStatusReducer from './orderStatusReducer';
import GetWishlistDataReducer from './GetWishlistDataReducer';

const rootReducer = combineReducers({
  orderStatusReducer: OrderStatusReducer,
  getWishlistDataReducer: GetWishlistDataReducer
});

export default rootReducer;
```

```
export default rootReducer;
```

```
const store = createStore(rootReducer);
```

???

/ /

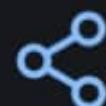
Redux - Middleware : syntax → (`getState, dispatch`) → next ⇒ action
let you write an action dispatcher which returns a function instead of
an action object.

```
function getUser() {  
    return function() {  
        return axios.get('/get-user-details');  
    }  
}
```

Conditional dispatch can be written inside middleware. Each middleware receives store's dispatch so that they can dispatch new action, and `getState` functions as arguments so that they can access the current state and return a function. Any return value from an inner function will be available as the value of dispatch function itself.



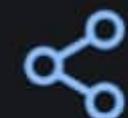
.../Reducers/Counter.js



```
1 const Counter = (state = 0, action) => {
2     switch (action.type) {
3         case 'INCREMENT':
4             return state + action.payload
5         case 'DECREMENT':
6             return --state
7         default:
8             return state
9     }
10 }
11 export default Counter;
```



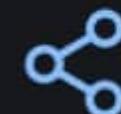
...Reducers/IsLogged.js



```
1 const IsLogged = (state = false, action) => {
2     switch (action.type) {
3         case 'SIGN_IN':
4             return !state
5         default:
6             return state
7     }
8 }
9 export default IsLogged
```



src/Reducers/index.js



```
1 import { combineReducers } from 'redux'  
2  
3 import Counter from './Counter';  
4 import IsLogged from './IsLogged';  
5  
6 export const AllReducers = combineReducers({  
7     Counter: Counter,  
8     IsLogged: IsLogged  
9 });
```

← src/index.js



```
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import App from './App';
4
5 import { legacy_createStore as createStore,
6   applyMiddleware } from 'redux'
7 import { Provider } from 'react-redux'
8
9
10
11
12 import { persistStore, persistReducer } from
13   'redux-persist'
14 import storage from 'redux-persist/lib/storage';
15 import { PersistGate } from 'redux-persist/integration/
16   react';
17 const config = {
18   key: 'main-root',
19   storage
20 }
21
22 const persistRed = persistReducer(config, AllReducers)
23
24 const store = createStore(
25   persistRed,
26   applyMiddleware(),
27   // window.__REDUX_DEVTOOLS_EXTENSION__ && window.__RE
28   DUX_DEVTOOLS_EXTENSION__(),
29 )
30
31 const persistor = persistStore(store)
32
33
34 const root = ReactDOM.createRoot(document.
35   getElementById('root'));
36 root.render(
37   <Provider store={store}>
38     <PersistGate loading={null} persistor={persistor}>
39       /* PersistGate delays the rendering o
40       f your app's UI until your persisted state
41       has been retrieved and saved to redux. */
42     </PersistGate>
43   </Provider>
44 )
```

[←](#) src/App.js

```
1 import { useSelector, useDispatch } from 'react-redux'  
2  
3 function App() {  
4  
5     const counter = useSelector(state => state.Counter)  
6     const isLoggedIn = useSelector(state => state.IsLoggedIn)  
7     const dispatch = useDispatch()  
8  
9     return (  
10         <div className="App">  
11             <h1>HELL0</h1>  
12             <h1>counter is {counter}</h1>  
13  
14             <button onClick={() => dispatch({ type:  
15                 'INCREMENT', payload: 5 })}>+</button><br /><br />  
16             <button onClick={() => dispatch({ type:  
17                 'DECREMENT' })}>-</button><br /><br />  
18             {isLoggedIn ? <h1>00H, I am logged in</h1> : <  
19                 button onClick={() => dispatch({ type: 'SIGN_IN' })}>  
20                     LOG IN</button>}  
21  
22         </div>  
23     );  
24 }  
25  
26 export default App;
```