Code zone / /

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 ←
cd ←
npm start
npm init

npm -v
node -v node.js

export default App; export { App }; {}

<> </> → fragment

Lists and Keys:

```
Const arr = [{id: 1, name: 'Anas'}, {id: 2, name: 'Attoumi'}];
Const li = arr.map((e1) => (<li key={e1.id}>{e1.name}</li>))
```

/ /

Darrelani: 15/4/2024

Center Vertically:

1- Using line-height:

```
<div class="center">
    <p>I am vertically and horizontally centered</p>
</div>
```

```
.center {
① line-height: 200px;
② height: 200px;
text-align: center; }
```

```
.center p {
③ line-height: 1.5;
④ display: inline-block;
⑤ 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 tranform: 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 الـ p أبناء الـ div حتى لو أولاد الأولاد (غير مباشر)

div > p {...} — جميع الـ p أبناء الـ div وليس أولاد الأولاد

div + p {...} — أول p بعد كل div وليس child ولكن مجرد بعد الـ div (يعني ما في بيناتن عناصر)

div ~ p {...} — جميع p التي تكون sibling وبعد الـ div (حتى لو في بيناتن عناصر)

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 i:first-child {...} — أول i من كل p (الـ i أول ولد للـ p)

p:fist-child i {...} — جميع الـ i بشرط أن يكون الـ p أول ولد

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

CSS Pseudo-element:

```
p::first-line {
  color: #ff0000;
  font-variant: small-caps;
}
```

```
p.intro::first-letter {
  color: #ff0000;
  font-size: 200%;
}
```

```
h1::before {
  content: url(smiley.gif);
}
```

```
::marker {
  color: red;
  font-size: 23px;
}
```

```
::selection {
  color: red;
  background: yellow;
}
```
cursor outline

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

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

CSS Vertical Navigation Bar:

```
<ul>
  <li><a class="active" href="#home">Home</a></li>
  <li><a href="#news">News</a></li>
  <li><a href="#contact">Contact</a></li>
  <li><a href="#about">About</a></li>
</ul>
```

```
ul {
  (1) list-style-type: none;
  (2) margin: 0;
  (3) padding: 0;
  width: 200px;
  background-color: #f1f1f1;
}
```

```
li a {
  (4) display: block;
  color: #000;
  padding: 8px 16px;
  text-decoration: none;
}
```

```
li a.active {
  background-color: #04AA6D;
  color: white;
}
```

```
li a:hover:not(.active) {
  background-color: #555;
  color: white;
}
```

CSS Horizontal Navigation Bar:

1. Inline List Items:

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

```
li {
display: inline;
}
```

2. Floating List Items:

```
ul {
list-style-type: none;
margin: 0;
padding: 0;
overflow: hidden;
}
```

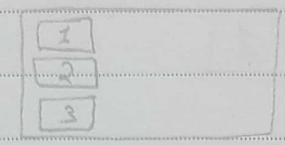
```
li {
float: left;
}
```

```
li a {
display: block; → padding: 8px;
padding: 8px;
background-color: #dddddd;
} text-decoration: none; text-align: center;
```

li:last-child { float: right } ← اذا بدي خلي آخر عنصر طالع عاليمين

CSS Flexbox :

```
<div class="flex_container">
    <div> 1 </div>
    <div> 2 </div>
    <div> 3 </div>
</div>
```




```
.flex_container {
display: flex;
flex_direction: column;
background_color: CadetBlue;
}
```

```
.flex_container > div {
background_color: #f1f1f1;
width: 100 px;
margin: 10 px;
text_align: center;
line_height: 75 px;
font_size: 30 px;
}
```

flex_direction: column — row

column_reverse — row_reverse

flex_wrap : wrap — nowrap

wrap_reverse

flex_flow : row wrap → shorthand

justify_content : center — space_around

flex_start — space_between

flex_end

align the flex items

align_items : center — stretch

flex_start — baseline

flex_end

align the flex lines

align_content : center — stretch

flex_start — space_around

flex_end — space_between

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

```
<div class="flex_container">
   <div style="order: 3">1</div>
   <div style="order: 1">2</div>
   <div style="order: 2">3</div>
</div>
```

2	3	1

[1]

Use the flex_grow property to specifies how much a 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" → بيكبر   1 الافتراضي / 2 بيصغر عالنص / 3 بيصير تلت

[4] ... style="flex-basis: 200px"

[5] ... style="flex: 0 0 200px" → "flex: flex_grow flex_shrink flexbasis"

[6] ... style="align-self: center" (flex_start) → overrides the default alignment set by container's align-items (الافتراضي) property

/ /

CSS Variables - The var() Function:

النطاق بيحدد الـ identity

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)
```

```
h1 {   local
  --mainColor : #...
}
```

Notes:

1. The variable name must begin with 2 dashes(--)
2. " " " is a case sensitive
3. اذا تم استخدام الـ Global والـ Local سوا ← ناخد بالـ Local

```
Var(--name , value)
```

Required — Optional

→ The fallback value (used if the variable is not found)

CSS 2D Transforms: ننقل من حالة لحالة

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

(50px) => X بيمثل محور

إذا سالب بالعكس

2. transform: rotate (20deg)

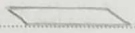
3. transform: scale (2, 3) (ضرب الـ X)   scale(2) => بضاعف المحورين

  0.2 width = 100px => 20px   الـ scale كسر بيتم ضرب الكم الحالي ويقل الـ width

scaleX(2)

scaleY (0.5)

4. transform: skew (20deg, 10deg) (الانحراف)   skew(20deg) => على X مقدم

skewX(20deg) 

skewY (10deg) 

*transform-origin: right top;   default → 50% 50%

transform: rotate(20deg) scale(2)   ← إذا بدي طبق كذا خاصية

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

scaleX skewY skewX scaleY TX TY

CSS 3D Transforms:

transform: rotateX (150deg)   → 90deg → بيختفي

rotateY (150deg)

rotateZ (90deg)

/ /

CSS Transitions: → allows you to change property values smoothly, over a given duration.

transition-property: width;
transition-duration: 5s;
transition-delay: 5s;

transition-timing-function: ease → افتراضي بطيء سريع بطيء
linear → متوسط السرعة
ease-in → بطيء عادي
ease-out → عادي بطيء
ease-in-out → بطيء عادي بطيء
steps(8, start)
cubic-bezier(x1 y1 x2 y2) (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 {        تعريف
    from {...}   0%, 10%...
    to {...}
}
```

animation-name: example;   استدعاء   [بس يخلص برجع للأصلي]

animation-duration: 4s; → بمدنا ما ينفذ

animation-delay: 2s; → إذا قيمتها (بالسالب) ← بيلش كأنو الوثانيتين شغال (مثل انو النص بالنص)

animation-iteration-count: 3;   infinite   (2 ← alternate)

animation-direction: reverse
alternate   normal → reverse
alternate-reverse   reverse → normal

animation-timing-function: (نفس transition)

animation-fill-mode: none افتراضي النسق الأصلي
forwards (آخر قيمة وصل عليها (باخذ بعين الاعتبار animation-direction & animation-iteration-count)
backwards (أول قيمة بتبين أثناء ال delay قبل التنفيذ)
both
↳ specify style when animation is not playing

shorthand:

animation: example 5s linear infinite alternate forwards;
(name duration function count direction fill-mode)

Css Attribute Selectors:

a[target] {...} — attribute لـ

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

[title ~= "flower"] — ~ → contain ⇒ flower ✓ / summer flower ✓ / my-flower ✗ / flowers ✗

[class |= "top"] — | → exactly or followed by hyphen(-) (Case-Sensitive)

[class ^= "top"] — ^ → start ⇒ top-content ✓ / top-content ✓ / content-top ✗

[class $= "top"] — $ → end

[class *= "top"] — * → contain ⇒

input[type="text"]:focus — button

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
⋮

CSS Grid Layout Module: شبكة (أسطر وأعمدة)

Parent:

1 display: grid; (block) display: inline-grid;

2 column-gap: 50px; row-gap: 50px; gap: 50px; بكلاهما

gap: 50px (row) 100px (column);

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

50% 50%

قد ما بيقدر ياخد ياخد ← 1fr auto → على قد المحتوى (بس اذا كلن auto بتم تقسيم المساحة عالعناصر)

100px 100px 100px 100px ⇒ repeat(4, 100px)

1fr 2fr → أكبر

4 grid-template-rows: نفس ↑

5 justify-content: → عادة auto تبع العناصر بتاخد على قد المحتوى (في حال كلن auto ما بيوزعوا) / 1fr ما بتوزع

6 align-content: ↗

Child:

1 grid-column: 1 / 5; 1 → 4

1 / span 5; (length) 1 → 5

2 / span 5; من البداية 2 بياخد 5 أعمدة

2 grid-row: نفس ↑

Shorthand
→ grid-column-start
→ grid-column-end

(row) → Shorthand

grid-area: 1 / 2 / 5 / 6; (row start, column start, span 4)

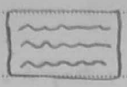
child grid-area: header;

Parent grid-template-areas: 'header header header'
' ... ';
'header header ...' ... → يعني كل العناصر يلي بالأسماء
'header header ...'

1 2 3 4
5 6 7

writing-mode: Vertical-lr; → اتجاه

max-width: 20rem;



display: -webkit-box; عرض 3

-webkit-line-clamp: 3;

-webkit-box-orient: vertical;

overflow: hidden;

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

-webkit-background-clip: text;

-webkit-text-fill-color: transparent;

pointer-events: none; → ماعد يتحدد العنصر

opacity: 0 Strong Combination — opacity تكون مع ال

leetcode.com

; → not required, but highly recommended

typeof null → object

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

/ /

JavaScript

Window. Print()

(Window.)alert ('hello world')

document.write('hello') إذا استخدمتها بعد ما تخلص الصفحة بتمسح كل الـ HTML (تنفيذ بعد onClick)

console.log()

.error()

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

console.log("Eng. %cAnas %cAttoum", "color:red;font-size:30px;"
, "color:blue;font-size:30px;")

typeof (variable_name)

(undefined) → undefined

(null) → object

typeof NaN // number !!

	Redeclare	Access before declare	Variable scope Drama
Var	Yes	Undefined	Yes, added to Window
let	No	Error	no
Const	No	Error	no

+Unary plus: Return number if it's not number.

-Unary negation: " " " " " " and negates it.

Datatype:

String - Number - Bigint - Boolean

Undefined - Null - Symbol - Object

ALADIB

SafeInteger → -(2^{53} - 1) to (2^{53} - 1)

9007199254740991 ✓
9007199254740992 ✗

نفس toFixed بس الفرق toPrecision()
أنو الطول الاجمالي للرقم مع الفاصلة)
واذا ما بعت arg بيرجع الرقم الاصلي

Number Methods:

(100).toString() ⇔ 100..toString() ⇒ 100 as a String

(100.55555).toFixed(2) ⇒ 100.56 — اذا بعت arg بتحدد بشيل الفاصلة لانو بيقرب

Number("100 Anas") ⇒ NaN — بتحول لرقم (بشرط انو يكون رقم يعني يكون نظيف ما في حروف)

(+"100 Anas") ⇒ NaN

parseInt("100 Anas") ⇒ 100 — برجع اول رقم وبشيل الفاصلة (بشرط الاول يكون رقم)

("Anas 100") ⇒ NaN

(100.50) ⇒ 100

parseFloat(100.50) ⇒ 100.5 — اذا في فاصلة بيتركها بس اذا ما في ما بحط فاصلة

Number.isInteger(100) ⇒ true (isSafeInteger) — اذا في فاصلة ⟸ false

Number.isNaN("Anas") ⇒ false (NaN → true) — ترجع true اذا NaN of datatype Number

Number.isNaN("Anas"/20) ⇒ true ('NaN' → false)

isNaN() ⟶ 'Hello' → false, 'Hello' → true — تحول لـ number وتشوف اذا f/t

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 — برجع الـ int بدون تقريب

String Methods:

let a = " Anas front developer ";

```
console.log ( a.indexOf ("front") );          5.
            .indexOf ("Front")                -1.
            .indexOf ("Anas" ,8)              -1.
            .indexOf ("o")                    7.

            .lastIndexOf ('o')                16.
            .(8)

            .slice (2)                        as front developer.
                   (2,5)                      as
                   (2,6)                      as f
                   (-5)                       loper.
                   (-5,-3)                    lo.

            .repeat (2)                       Anas front developerAnas front developer.

            .split (' ')                      Array(3) ["Anas", "front", "developer"]
                   ('',6)                     Array(6) ["A", "n", "a", "s", " ", "f"]
                   (' ',2)                    Array ["Anas", "front"]

            .length                           20.
```

let a = "Anas front developer";

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

swap ← لانه اقل من اكبر (6, 2) as f

لانه اقل سالب ← بيعتبره صفر (-10, 6) Anas f

(a.length - 6) eloper

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

a.length - 1 آخر حرف

.substr(0, 6) Anas f

(-6) eloper

(-5, 2) lo

.includes("s fro") true

("s fro", 2) ← default 0 من دون ما نحدد true

.startsWith("z") false

("s", 2) ← default 0 من دون ما نحدد false

("as", 2) false

.endswith("s") true

("o", 6) ← default string length false

("ro", 6) 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 JavaScripts 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] + " ";
```

Note: you must use person[x] in the loop.
person.x will not work (Because x in 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 ما تنطبق على الكل (بس هيك

Person.nationality = "English";  ✗
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

Array → use numbered indexes
Object → " named "

JavaScript Arrays:

```
const cars = ["Saab", "Volvo", "BMW"]; //easiest way (array literal)
or const cars = [];
   cars[0]= "Saab";
   cars[1]= "Volvo";
                                            avoid it → new Array(4);
or const cars = new Array("Saab", "Volvo", "BMW"); // No need to use it
```

Access Element → let car = cars[0]; (cars.at(0); آخر عنصر ← -1)   change Element → cars[0]= "Opel";

Array to String: //Saab,Volvo,BMW

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

لوطبعة cars رح تطلع نفس النتيجة ↑ بس الفرق الوحيد أنو صار String

to Recognize an Array:

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

Array Methods:

```
cars.length;          cars.toString();          cars.join("*"); → Seperator
cars.push("a"); بترجع الطول الجديد ('a','b')          cars.pop(); بترجع العنصر الأخير المحذوف
cars.unshift("a","b"); " " "                       cars.shift(); بالأول " "
```

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(fruits, arr) ('a','b' → push) بتدمج cars مع ال fruits (وما بتعدل على المصفوفات الأصلية)
cars.copyWithin(2,0); copy to index 2, all elements from index 0
//Saab Volvo Saab
```

اذا العناصر كتار [illegible] بالفنكشن لا ما بزيد عدد عناصر ال array

ALADIB

/ /

```
            الطول 2
Const myArr = [ [1,2] , [3,4] ];
            الطول 4
Const newArr = myArr.flat();
```
create new array with sub-array element concatenated to a specified depth.

```
                من وين نضيف   كم واحد نحذف   شو حنضيف
cars.splice ( 2 ,  0 ,  "a" , "b" )
```
ترجع العناصر المحذوفة / بتغير عالأصلية

(0, 1) → مين بدي أحذف

toSpliced() → نفسها بس ما بتغير عالأصلية

```
cars.slice( 1 );
```

Array Search:

```
cars.indexOf('a');
cars.lastIndexOf('a')
cars.includes('a');
```

```
numbers.find ( (e) => {return e>18} );
```
ترجع أول عدد أكبر من 18

```
        findIndex( ... );
```
أول index

```
        findLast( ... );
        findLastIndex( ... );
```

Sorting Arrays:

```
cars.sort();
```
بتغير عالأصلية

ترتيب أبجدي ويكون الناتج غلط بس بالأرقام

```
        ((a,b) => {a-b})
```
ترتيب تصاعدي

```
cars.reverse();
```
اذا استخدمناها مع sort() بترتب تنازلي

```
cars.toSorted();
```
ما بتغير عالأصلية

```
cars.toReversed();
```

```
cars.sort ((a,b) => {0.5 - Math.random()});
```
خربط الarray

أفضل طريقة لخربطة اسمها The Fisher Yates Method

ما بغير على المصفوفة الأصلية { array.forEach(value, index, array)

(1 = value, 2 = index, 3 = array)

map

filter / every / some

reduce / reduceRight بس بدل واحد total الحاصل وبعدين الرقم

forEach → ما برجع مصفوفة
map → برجع
filter → ترجع array
find → برجع key or value

(loop زي)

لحتى نجيب أصغر عنصر بالمصفوفة إذا تشابهت بأقل عنصر؟ نستخدم:

```
Math.min.apply(null, numbers)
```

(numbers = array)

Sorting Object Arrays:

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

ترتيب حسب السنة

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

ترتيب حسب النوع:

```
Car.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;});
```

برجع عناصر المصفوفة يلي أكبر من 18

يعني قيمة ابتدائية وبزيد 100 على المجموع الكلي (, 100)

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

Sum لكل عناصر المصفوفة

reduceRight: works from right to left

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

برجع true إذا كل العناصر أكبر من 18

```
numbers.sum( .... )
```

برجع true إذا في (زي) قيمة أكبر من 18

/ /

```
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 + "<br>"; }  // 0 1 2

for (let x of cars.entries()) { text += x + "<br>"; } //  0: Volvo   [0, "Volvo"]
                                                         1: BMW     [1, "BMW"]
                                  key/value

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
document.getElementById("demo").innerHTML = "Hello world!"
```

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["frm1"];
let text = "";
for (let i=0; i<x.length; i++) {
    text += x.elements[i].value + "<br>";
}
```

DOM Navigation

nodeName → tagname 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

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] [0] = ↩ 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)` — بتضيف الـ node كآخر عنصر

`element.insertBefore(node, element2)` — بتضيف الـ node قبل element 2

`element.remove()` — بالـ browser القديم هي ما بتشتغل

`parent.removeChild(child)` — البديل عنها ⇐

`parent.replaceChild(para, child)` (para = الجديد, child = القديمة)

`document.getElementsByTagName(...)` (also: ClassName, document.forms, children; .length) ⇒ Return an **HTMLCollection**

لحتى أمشي على HTMLCollection في 3 طرق إما for تقليدية أو let x of myCollection

أو بحولها لـ Array بستخدم map ← Array.from(myCollection).map(...)

`document.querySelectorAll()` ⇒ Return a static NodeList

`childNodes` (getElementsByName) Property Return a live NodeList

الفرق بينهم بالصفحة التالية

/ /

The Difference Between an HTMLCollection and a Node list:

Both:

are array-like collections (lists) of nodes (elements)
have alength 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 <li> element to a list in the DOM, the list in the the HTMLCollection will also change.

NodeList: methods _ forEach / keys / value / 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 <li> element to a list in the DOM, the list in NodeList will not change.

```
<div class="js">
    JS <span>Div</span> &lt;span&gt;
</div>
<img src="s" alt="">
<a href="#">Google</a>

<script>
document.querySelector('.js').innerHTML : '\n JS <span>Div</span>&lt;span&gt;\n'
                        .textContent : '\n JS Div<span>\n'
                        .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 ;
let [ firstName, lastName ] = person ;
                              | country , 'US'

                  , lastName : name ] ...
```

```
let name = 'W3Schools';
let [a1, a2, a3, a4, a5] = name;
```

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

```
const fruits = new Map ([ ['apples', 500], ['bananas', 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 Computer 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 phone → 256 "VGA colors"
.pixelDepth
for modern Computer Color Depth and Pixel Depth are equal

alert ('I am an alert box!');
window.confirm('Press a button!') ou Abbrechen
Prompt ('please enter your name:', 'Anas Alhaya');

(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
// /js/js_window_location.asp
.protocal // https
.port returns the number of the internet host port (of the current page)
if the port number is default, most browsers will diplay 0 or nothing
default ⟶ 80 for http and 443 for https
.assign("https://... .com") load a new document

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

(Window.)navigator: cookieEnabled: true // false
.appName: // Netscape Strange Enough
is the application name for IE11, Chrone, 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

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

.clearTimeout(myVar) بيلغي اذا كان مشغال

.setInterval (...);
.clearInter (myVar);

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

→ expires: By default, the cookie is deleted when the browser is closed

→ 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

لا بدي غير cookie بكتب نفس سطر التعريف (يعني بكتب مثو)

لا حذف cookie بحط التاريخ expires ـــ 1970 مثلاً بس منحط الـ path لتعرف الـ cookie المحدد

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 make 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 : → can be all of the above and function, date, undefined

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();
```

stringify

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");

// letters.keys() → because it has no keys
// letters.values()
// .entries() → a,a  b,b

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

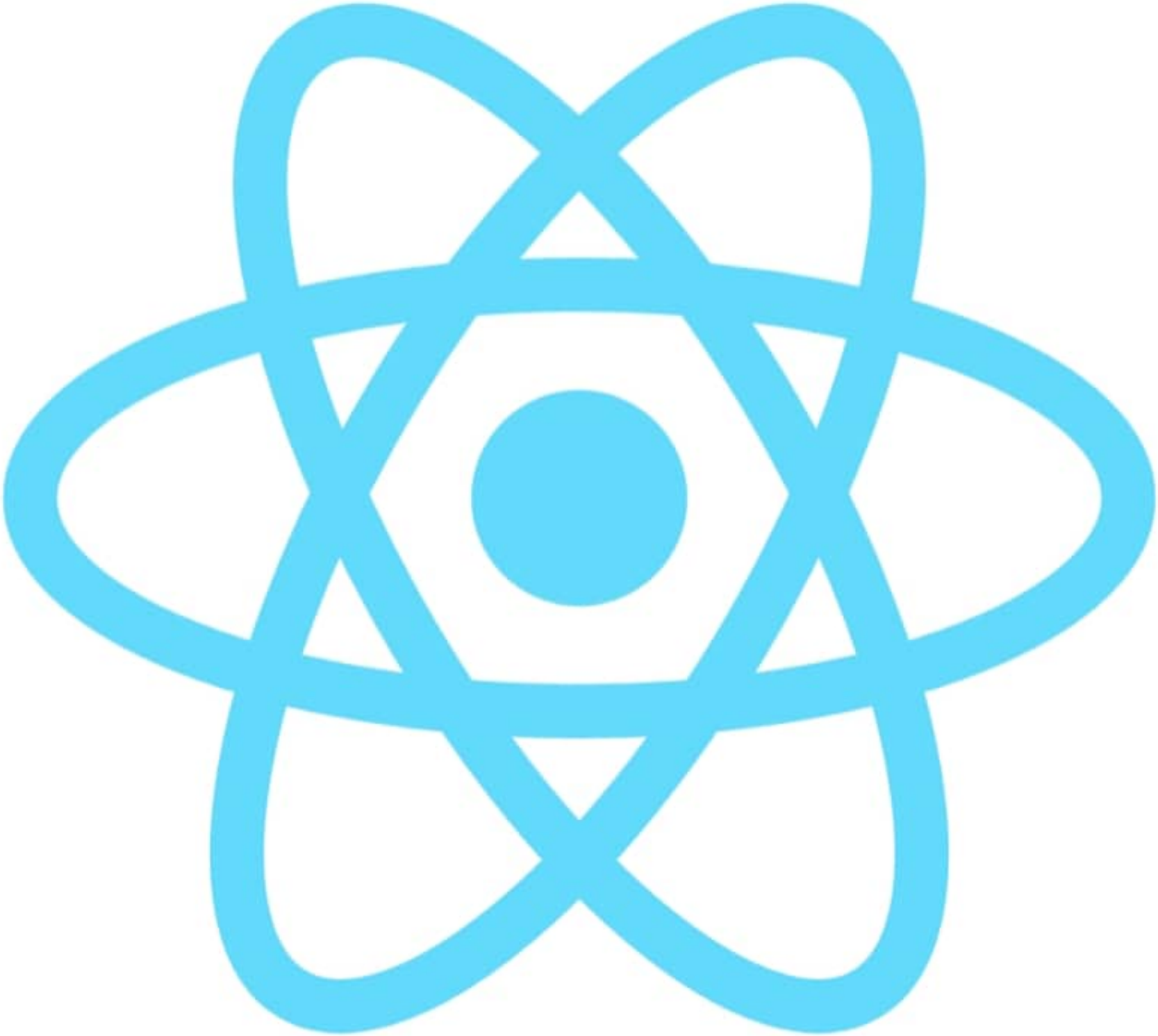
```
const fruits = new Map();
fruits.set("apples", 500);

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 يمكن استخدام
key ، object يمكن استخدام



/ /

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 root API (you can see it on React Upgrade tab)

/ /

JSX : stands for JavaScript XML

JSX not required but it 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 → can be 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 = (
  <ul>
    <li> Apples </li>
    <li> Bananas </li>
  </ul>
);
```

- 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 a 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:

1- put if statements outside of the JSX:

```
const x = 5
let text = "Goodbye";
if (x < 10) {
    text = "Hello";
}
const myElement = <h1> {text} </h1>;
```

2- 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 charachter.

/ /

Props : stands for properties

```
function Car (props) {
    return <h2> I am a {props.color} Car! </h2>;
}
<Car color="red" />; →
```

فيني استدعيه جوا Component آخر

"red" ⤷ not a String? ⟹ color={carName}

```
export default Car;
import Car from './Car.js';
```

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

منستخدم arrow function لما بدي اعمل Pass an argument to an event handler

لانو بدونو بينفذ التابع فوراً (لانو ما في ضغط)

ALADIB

اصنع الحضارة

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

1. if Statement

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

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:
        <input type="text" value={name}
          onChange={(e) => setName(e.target.value)} />
      </label>
    </form>
  )
}
```

<form onSubmit={handleSubmit}> ... <input type="submit" /> ... </label> ... </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 (
    <input type="text" name="username" value={inputs.username || ""}
           onChange={handleChange} />
    :
```

/ /

React Router: is the most popular Solution

create react app doesn't include page routing.

```
npm i -D react-router-dom
```

@latest ⇐ النسخة حالياً V6 إذا بدي حطها من الـ V5 بزيدين

src\pages\:
Layout.js ...

```
import {BrowerRouter, Routes, Route} from "react_router_dom";
return(
    <BrowserRouter>
        <Routes>
            <Route path="/" element={<Layout/>}>
                <Route index element={<Home/>} />            ← default route for Parent/
                <Route path="blogs" element={<Blogs/>} />    ← /blogs
                <Route path="contact" element={<Contact/>} />
                <Route path="*" element={<NoPage/>} />       ← undefined URLs This is great for 404 error page
            </Route>
        </Routes>
    </BrowserRouter>
);
```

Layout.js The layout route is a shared component that inserts common content on all pages such as navigation menus

```
import { Outlet , Link } from "react-router-dom";
```

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

```
const Layout = () => {
  return (
    <>
      <nav>
        <ul>
          <li><Link to="/">Home</Link></li>
          :
        </ul>
      </nav>
      <Outlet /> → render the current route selected
    </>
  )
};
export default Layout;
```

Anytime we link to an internal path, we will use <Link> instead of <a href="">

React Memo:

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

```
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 tracke 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 => count + 1 );
    }, 1000 );
});
```

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 ⇒ runs on every render.
2. An Empty array ⇒ runs only on the first render.
3. Props or state values ⇒ runs on the first render and any time any dependency value change ↑

Effect Cleanup:

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

في حالة استخدام useEffect

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 ("Anas Attoumi");
    return (
        <UserContext.Provider value={user}>
            <h1> {`Hello ${user}!`} </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);   --like-->  const count = {current:0}
useEffect(() => { count.current + 1; });
```

عم يعدّ render عم يصير

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:

فيني استخدمها جوا ال useEffect لما تغيرت قيمة ال useState تغير قيمة ال useRef

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 (<>
        {todos.map((todo) => (
            <div key={todo.id}>
                <label><input type='checkbox' checked={todo.complete}
                    onChange={() => handleComplete(todo)} />
                {todo.title}</label></div>))} </> ); }
```

/ /

- 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 Hook 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 increase.