Array

```
'farooq'.length // 6
'farooq'[1] // 'a'
'farooq'.includes('oos') // false
'farooq'.includes('oo') // true
'farooq'.indexOf('ar') // 1
'farooq'.startsWith('f') // true
'farooq'.endsWith('f') // false
'farooq'.slice(0, 5) // faroo
'farooq'.slice(4) // oq
'farooq'.toUpperCase() // FAROOQ
'FAROOQ'.toLowerCase() // farooq
'farooq'.replace('oo', 'ff') // farffq
'farooq'.repeat(3) // farooqfarooqfarooq
'farooq'.split('') // ['f', 'a', 'r', ' ', 'o', 'o', 'q']
'far ooq'.split(' ') //  ['far', 'ooq']
```

```
[1, 2, 3].push(4) // 4 : [1,2,3,4]
[1, 2, 3].pop() // 3 : [1,2]
[1, 2, 3].shift() // 1 : [2,3]
[1, 2, 3].unshift() // 3 : [1,2,3]
['a', 'b'].concat('c') // ['a','b', 'c']
['a', 'b', 'c'].join('-') // a-b-c
['a', 'b', 'c'].slice(1) // ['b', 'c']
['a', 'b', 'c'].indexOf('b') // 1
['a', 'b', 'c'].includes('c') // true
[3, 5, 6, 8].find((n) => n % 2 === 0) // 6
[2, 4, 3, 5].findIndex((n) => n % 2 !== 0) // 2
[3, 4, 8, 6].map((n) => n * 2) // [6,8,16,12]
[1, 4, 7, 8].filter ((n) => n % 2 === 0) // [4,8]
[2, 4, 3, 7].reduce((acc, cur) => acc + cur) // 16
[2, 3, 4, 5].every((x) => x < 6) // true
[3, 5, 6, 8].some((n) => n > 6) // true
[1, 2, 3, 4].reverse() // [4,3,2,1]
[3, 5, 7, 8].at(-2) // 7
```

1. indexOf() – find an item's index

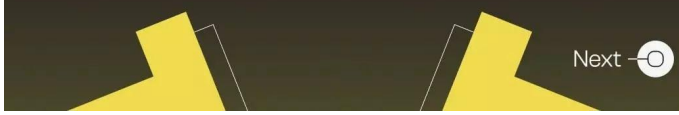
```
const myPets = ['Dog', 'Cat', 'Hamster'];
myPets.indexOf('Cat') -> // 1
```

2. join() – create a string from array items

```
const myPets = ['Dog', 'Cat', 'Hamster'];
myPets.join(' and '); -> // 'Dog and Cat and Hamster'
```

3. slice() – split an array at given index(es)

```
const myPets = ['Dog', 'Cat', 'Hamster'];
myPets.slice(1); -> // ['Cat', 'Hamster']
myPets.slice(1, 2); -> // ['Cat']
```



4. splice() – split an array and/or insert new items

```
const myPets = ['Dog', 'Cat', 'Hamster'];
myPets.splice(1, 2, 'Lizard') -> // ['Dog', 'Lizard']
```

5. concat() – concatenate one or more arrays

```
const myPets = ['Dog', 'Cat', 'Hamster'];
const myFlyingPets = ['Bird'];
const myWaterPets = ['Fish'];

const allPets = myPets.concat(myFlyingPets, myWaterPets);
// allPets -> ['Dog', 'Cat', 'Hamster', 'Bird', 'Fish']
```

6. forEach() – loop over an array and access each item

```
const myPets = ['Dog', 'Cat', 'Hamster'];
myPets.forEach(pet => console.log(pet));
```

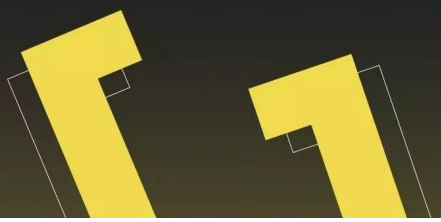
Next

7. filter() – create a new array based on a filter

```
const myPets = ['Dog', 'Cat', 'Hamster'];
const threeLetterPets = myPets.filter(pet => pet.length === 3)
// threeLetterPets -> ['Dog', 'Cat']
```

8. map() – loop over an array and run some operation on each item without mutating the original array

```
const myPets = ['Dog', 'Cat', 'Hamster'];
const lovedPets = myPets.map(pet => `${pet}❤️`)
// lovedPets -> ['Dog❤️', 'Cat❤️', 'Hamster❤️']
```



Next

9. flat() – flatten an array to a single dimension

```
const values = [1, 2, [7], 3, [1, 2], 4]
console.log(values.flat())
// [1, 2, 7, 3, 1, 2, 4]
```

10. reduce() – run a callback on each item and reduce the array to a single value

```
const values = [1, 6, 7, 1, 3, 4];
const total = values.reduce((total, currentVal) => total + currentNum);
// total -> 22
```

11. findIndex() – finds the index of an item based on a condition

```
const people = [{ name: 'David' }, { name: 'Peter'}, { name: 'Alex'}]
const peterIndex = people.findIndex(person => person.name === 'Peter');
// peterIndex -> 1
```

Next

12. every() – check if every item meets a condition

```
const values = [1, 6, 7, 1, 3, 4];
console.log(values.every(val => val < 8))
// true
```

13. find() – find first value that meets a condition

```
const values = [4, 1, 7, 2, 5, 7, 9, 25];
const firstValueOverSeven = values.find(val => val > 7);
// firstValueOverSeven -> 9
```

14. some() – check if some values meet a condition

```
const values = [1, 6, 7, 1, 3, 4];
console.log(values.some(val => val > 7))
// false
```

15. sort() – sorts an array

```
const values = [4, 1, 7, 2, 5];
const names = ['David', 'Alexander', 'Peter'];

values.sort((a, b) => a - b); // [1, 2, 4, 5, 7]
names.sort(); // ['Alexander', 'David', 'Peter']
```

Next

02

FOR

```
const array = [1,2,3,4,5];

for (let index = 0; index < array.length; index++) {
  console.log(array[index]);
}

// 1 2 3 4 5
```

03

WHILE

```
let index = 0;
const array = [1,2,3,4,5];

while (index < array.length) {
  console.log(array[index]);
  index++;
}

// 1 2 3 4 5
```

04

ForEACH

```
const array = [1,2,3,4,5];

array.forEach(function(current_value, index, array) {
  console.log(current_value);
});

// 1 2 3 4 5
```

05

MAP

```
const array = [1,2,3,4,5];
const square = x => Math.pow(x, 2);
const squares = array.map(square);

console.log(`Original array: ${array}`);
console.log(`Squared array: ${squares}`);

// Original array: 1,2,3,4,5
// Squared array: 1,4,9,16,25
```

06

REDUCE

```
const array = [1,2,3,4,5];
const sum = (x, y) => x + y;
const array_sum = array.reduce(sum, 0);

console.log(`The sum of array: ${array} is ${array_sum}`);

// The sum of array: 1,2,3,4,5 is 15
```

07

FILTER

```
const array = [1,2,3,4,5];
const even = x => x % 2 === 0;
const even_array = array.filter(even);

console.log(`Even numbers in array ${array}: ${even_array}`);

// Even numbers in array 1,2,3,4,5: 2,4
```

08

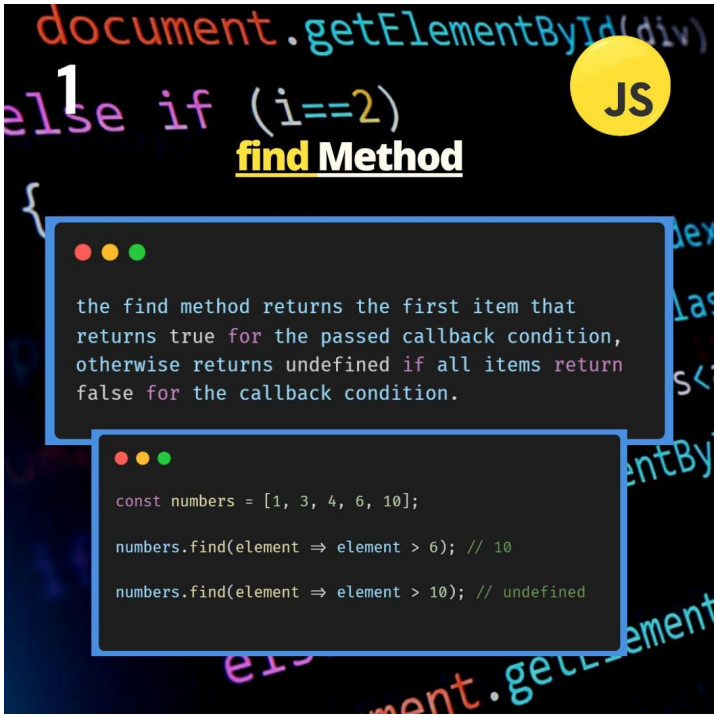
EVERY

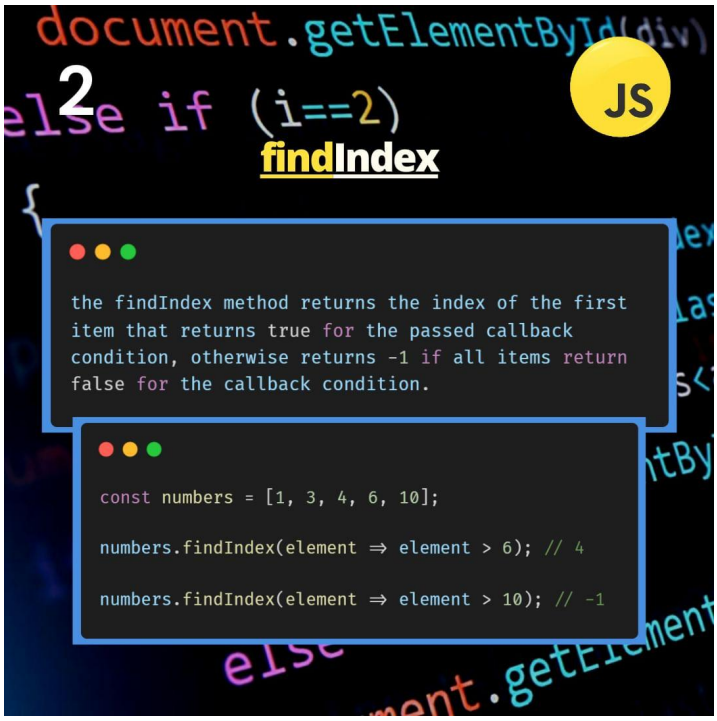
```
const array = [1,2,3,4,5];
const under_seven = x => x < 7;

if (array.every(under_seven)) {
  console.log('less than 7');
} else {
  console.log('bigger than 7');
});

// less than 7
```

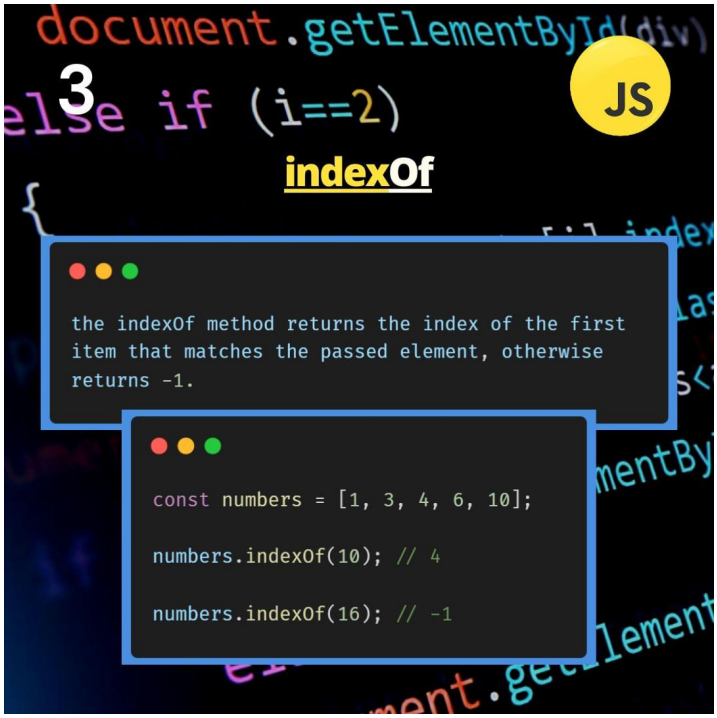
09

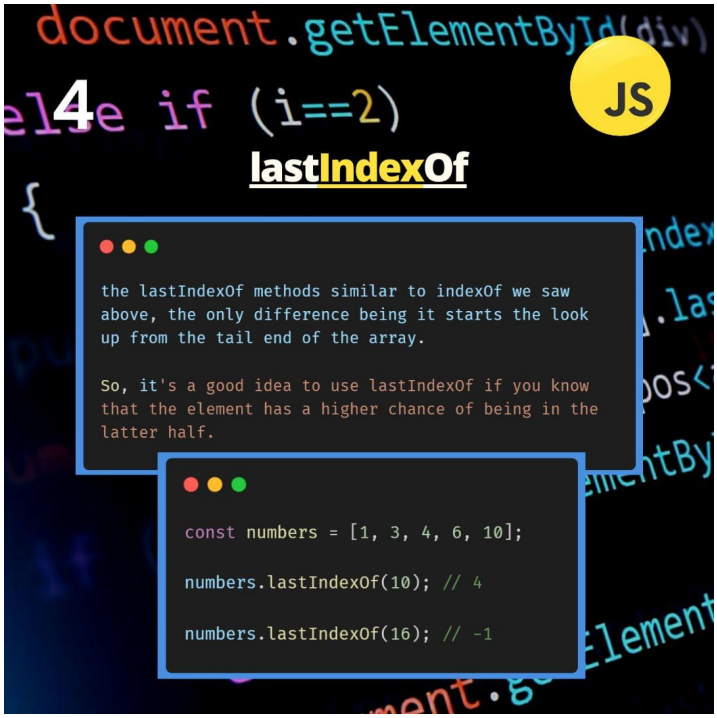
SOME

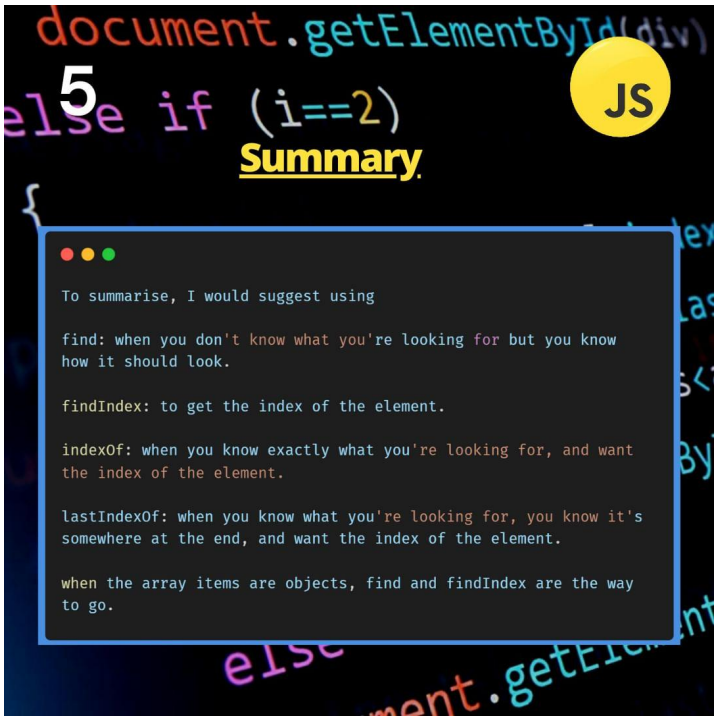
```
const array = [1,2,3,9,5,6,4];
const over_seven = x => x > 7;

if (array.some(over_seven)) {
  console.log('element bigger than 7 was found');
} else {
  console.log('No element bigger than 7 was found');
};

// element bigger than 7 was found
```


1
JS
find Method
the find method returns the first item that returns true for the passed callback condition, otherwise returns undefined if all items return false for the callback condition.
const numbers = [1, 3, 4, 6, 10];
numbers.find(element ⇒ element > 6); // 10
numbers.find(element ⇒ element > 10); // undefined


2
JS
findIndex
the findIndex method returns the index of the first item that returns true for the passed callback condition, otherwise returns -1 if all items return false for the callback condition.
const numbers = [1, 3, 4, 6, 10];
numbers.findIndex(element ⇒ element > 6); // 4
numbers.findIndex(element ⇒ element > 10); // -1


document.getElementById(div)
3
else if (i==2)
JS
indexOf
{
the indexOf method returns the index of the first item that matches the passed element, otherwise returns -1.
const numbers = [1, 3, 4, 6, 10];
numbers.indexOf(10); // 4
numbers.indexOf(16); // -1


4
JS
lastIndexOf
the lastIndexOf methods similar to indexOf we saw above, the only difference being it starts the look up from the tail end of the array.
So, it's a good idea to use lastIndexOf if you know that the element has a higher chance of being in the latter half.
const numbers = [1, 3, 4, 6, 10];
numbers.lastIndexOf(10); // 4
numbers.lastIndexOf(16); // -1


5
JS
Summary
To summarise, I would suggest using
find: when you don't know what you're looking for but you know how it should look.
findIndex: to get the index of the element.
indexOf: when you know exactly what you're looking for, and want the index of the element.
lastIndexOf: when you know what you're looking for, you know it's somewhere at the end, and want the index of the element.
when the array items are objects, find and findIndex are the way to go.

#TiposDeLoopsJS

for

```
const brands = ["Nissan", "Audi", "BMW", "Toyota", "Honda"];

for (let i = 0; i < brands.length; i++) {
  console.log(brands[i].toUpperCase());
}
```

#TiposDeLoopsJS

forEach

```
const brands = ["Nissan", "Audi", "BMW", "Toyota", "Honda"];

brands.forEach((item) => {
  console.log(item.toUpperCase());
});
```

Arrasta

#TiposDeLoopsJS

for of

```
const brands = ["Nissan", "Audi", "BMW", "Toyota", "Honda"];

for (const brand of brands) {
  console.log(brand.toUpperCase());
}
```

Arrasta

#TiposDeLoopsJS

map

```
const brands = ["Nissan", "Audi", "BMW", "Toyota", "Honda"];

const newBrands = brands.map((item) => {
  const brand = item.toUpperCase();
  console.log(brand);
  return brand;
});
```

#TiposDeLoopsJS

while

```
const brands = ["Nissan", "Audi", "BMW", "Toyota", "Honda"];

let i = 0;

while (i < brands.length) {
  console.log(brands[i].toUpperCase());
  i++;
}
```

Arrasta

#TiposDeLoopsJS

do while

```
const brands = ["Nissan", "Audi", "BMW", "Toyota", "Honda"];

let i = 0;

do {
  console.log(brands[i].toUpperCase());
  i++;
} while (i < brands.length);
```

Arrasta


#TiposDeLoopsJS
for in
const brands = {
1: "Nissan",
2: "Audi",
3: "BMW",
4: "Toyota",
5: "Honda",
};
for (const key in brands) {
console.log(brands[key].toUpperCase());
}
Arrasta

String

```
"Hello".charAt(4)                => o
"Hello".concat("", "world")      => Hello world
"Hello".startsWith("H")          => true
"Hello".endsWith("o")            => true
"Hello".includes("x")            => false
"Hello".indexOf("l")             => 2
"Hello".lastIndexOf("l")         => 3
"Hello".match(/[A-Z]/g)          => ['H']
"Hello".padStart(6, "?")         => ?Hello
"Hello".padEnd(6, "?")           => Hello?
"Hello".repeat(3)                => HelloHelloHello
"Hello".replace("llo", "y")      => Hey
"Hello".search("e")              => 1
"Hello".slice(1, 3)              => el
"Hello".split("")                => ['H', 'e', 'l', 'l', 'o']
"Hello".substring(2, 4)          => ll
"Hello".toLowerCase()            => hello
"Hello".toUpperCase()            => HELLO
"  Hello  ".trim()               => Hello
"  Hello  ".trimStart()          => "Hello  "
"  Hello  ".trimEnd()            => "  Hello"
```

Number

```
const num = 12.4;

isNaN(num); // false

isFinite(num); // true

Number.isInteger(num); // false

Number.isSafeInteger(num); // false

parseFloat(num); // 12.4

parseInt(num); // 12

num.toExponential(); // "1.24e+1"

num.toFixed(); // 12

num.toPrecision(3); // "12.4"

num.toString(10); // "12.4"

num.valueOf(); // 12.4

let today = new Date();
today.toLocaleString('en-IN');
// "1/9/2022, 11:03:17 am"
```

```
new Intl.NumberFormat('en-US', {
  maximumSignificantDigits: 3
}).format('123123123')
// -> 123,000,000

new Intl.NumberFormat('en-IN')
  .format('123123123')
// -> 12,31,23,123

new Intl.NumberFormat('en-US', {
  notation: 'compact'
}).format('123123123')
// -> 123M

new Intl.NumberFormat('en-US', {
  style: 'percent'
}).format(0.5)
// -> 50%
```

Grouping objects in array

Grouping array items

BABY WOLF CODES

JavaScript

Grouping array items.

Input

```
const arr = [
  { name: 'foo', category: 'A' },
  { name: 'bar', category: 'A' },
  { name: 'baz', category: 'B' }
]
```

Output

```
{
  A: ['foo', 'bar'],
  B: ['baz'],
}
```

SWIPE

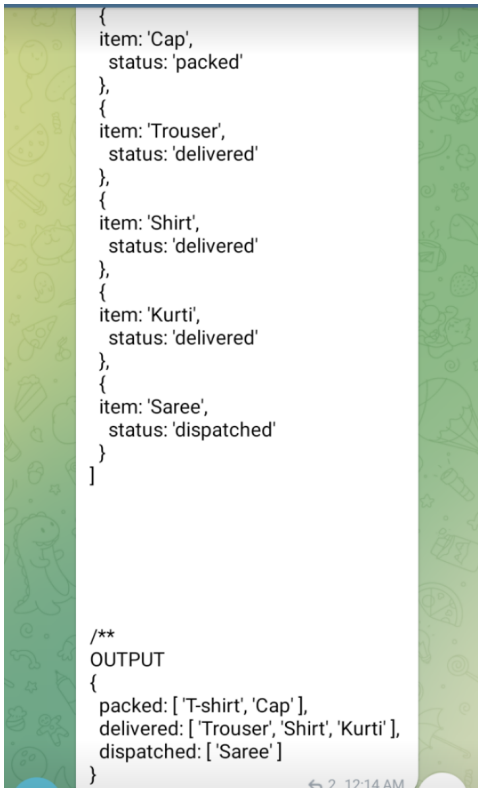
BABY WOLF CODES

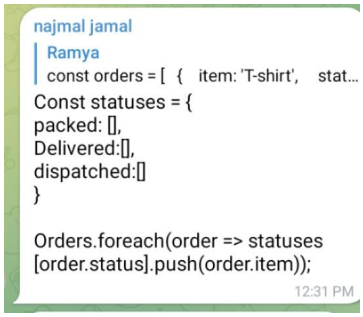
SOLUTION

Create a new array if this is the first time we are encountering this key

```
arr.reduce((groups, item) => {
  if (!groups.has(item.category)) {
    groups.set(item.category, []);
  }

  groups.get(item.category).push(item.name);
  return groups;

}, new Map());
```

Initial value for the accumulator (groups)


{
item: 'Cap',
status: 'packed'
},
{
item: 'Trouser',
status: 'delivered'
},
{
item: 'Shirt',
status: 'delivered'
},
{
item: 'Kurti',
status: 'delivered'
},
{
item: 'Saree',
status: 'dispatched'
}
]
/**
OUTPUT
{
packed: ['T-shirt', 'Cap'],
delivered: ['Trouser', 'Shirt', 'Kurti'],
dispatched: ['Saree']
}


najmal jamal
Ramya
const orders = [{ item: 'T-shirt', stat...
Const statuses = {
packed: [],
Delivered:[],
dispatched:[]
}
Orders.foreach(order => statuses
[order.status].push(order.item));
12:31 PM

```
const students = [
  { name: "Bob", skill: "js" },
  { name: "David", skill: "css" },
  { name: "James", skill: "js" },
  { name: "Jake", skill: "html" },
  { name: "Jack", skill: "js" },
  { name: "Jill", skill: "css" }
];
/*
Output:
[
  { skill: 'html', people: ['Jake'], count: 1 },
  { skill: 'css', people: ['David', 'Jill'], count: 2},
  { skill: 'js', people: ['Bob', 'James', 'Jack'], count: 3}
]
*/

function changeFormat(list) {
  // Your code here
  const obj = {};
  list.forEach((student) => {
    if (!obj[student.skill]) {
      obj[student.skill] = [];
    }
    obj[student.skill].push(student.name);
  });
  const result = Object.keys(obj).map((key) => {
    console.log(key);
    console.log(obj[key]);
    return {
      skill: key,
      people: obj[key],
      count: obj[key].length
    };
  });
  console.log(result);
}
changeFormat(students);

```

03

#JAVASCRIPT

1.Copy to Clipboard

Easily copy any text to clipboard using navigator.clipboard.writeText.

```
const copyToClipboard = (text) => navigator.clipboard.writeText(text);

copyToClipboard("Hello World");
```

2.Convert RGB to Hex

```
const rgbToHex = (r, g, b) =>
  "#" + ((1 << 24) + (r << 16) + (g << 8) +
b).toString(16).slice(1);
rgbToHex(0, 51, 255);
// Result: #0033ff
```

@getbitcode

3.Check if Date is Valid

Use the following snippet to check if a given date is valid or not.

```
const isDateValid = (...val) => !Number.isNaN(new Date(...val).valueOf());

isDateValid("December 17, 1995 03:24:00");
// Result: true
```

4.Generate Random Hex

You can generate random hex colors with Math.random and padEnd properties.

```
const randomHex = () => `#${Math.floor(Math.random() * 0xffffff).
toString(16).padEnd(6, "0")}`;

console.log(randomHex());
// Result: #92b008
```

5.Clear All Cookies

You can easily clear all cookies stored in a web page by accessing the cookie using document.cookie and clearing it.

```
const clearCookies = document.cookie.split(';').forEach(cookie =>
 document.cookie = cookie.replace(/^ +/, '').replace(/=.*/, `=;expires=${new
Date(0).toUTCString()};path=/`));
```

6.Find the day of year

```
const dayOfYear = (date) =>
  Math.floor((date - new Date(date.getFullYear(), 0, 0)) / 1000 / 60 / 60 /
24);
dayOfYear(new Date());
// Result: 272
```

7.Capitalise a String

Javascript doesn't have an inbuilt capitalise function, so we can use the following code for the purpose.

```
const capitalize = str => str.charAt(0).toUpperCase() +
str.slice(1)
capitalize("follow for more")
// Result: Follow for more
```

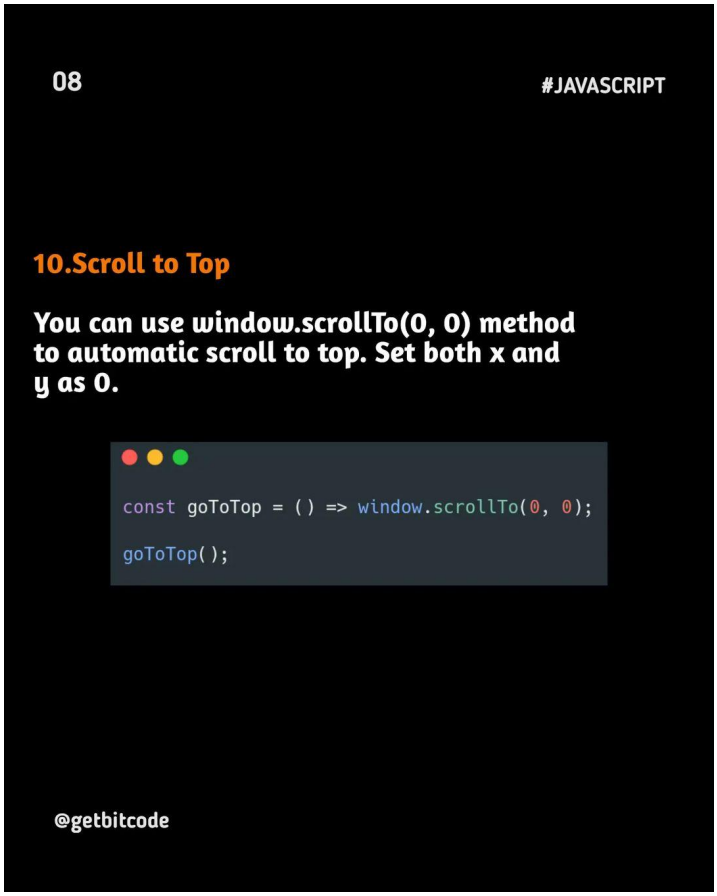
8.Remove Duplicated from Array

You can easily remove duplicates with Set in JavaScript. Its a life saver.

```
const removeDuplicates = (arr) => [...new Set(arr)];

console.log(removeDuplicates([1, 2, 3, 3, 4, 4, 5, 5, 6]));
// Result: [ 1, 2, 3, 4, 5, 6 ]
```

9.Check if a number is even or odd

```
const isEven = num => num % 2 === 0;
console.log(isEven(2));
// Result: True
```


08
#JAVASCRIPT
10.Scroll to Top
You can use window.scrollTo(0, 0) method to automatic scroll to top. Set both x and y as 0.
const goToTop = () => window.scrollTo(0, 0);
goToTop();
@getbitcode

Axios crud

MAKE HTTP REQUEST WITH AXIOS

You can also use JavaScript **fetch API**

Installing **axios** and initialzing new project.

```
npm i axios      //Install Axios
npm init -y      //New Project package.json
touch index.js   //New File to write code
```

Load the **axios** in the **index.js** file

```
const axios = require('axios');
```

GET request with **axios**

GET REQUEST WITH AXIOS

GET **/get** Retrieve or read the data from record

```
const getRequest = async () => {
  try {
    const response = await axios.get('https://anyAPI.com/todos/1');
    console.log(response.data);
  } catch (err) {
      console.error(err);
  }
};

getRequest();
```

The Axios response object contains the following properties.

console.log(response.data)

```
{
  userId: 1,
  id: 1,
  title: 'Please like & save',
  completed: false
}
```

POST REQUEST WITH AXIOS

POST **/post** Create New record

Here's the code snippet that shows how you can pass data with your **POST** request.

```
const postRequest = async () => {
  const newTodo = {
      userId: 1,
      title: 'Like the post',
      completed: false
  }
  try {
    const resp = await axios.post('https://anyAPI.com/todos', newTodo);
    console.log(resp.data);
  } catch (err) {
    console.error(err);
  }
}

postRequest();
```

PUT REQUEST WITH AXIOS

PUT **/put** If the record exists then update else create a new record.

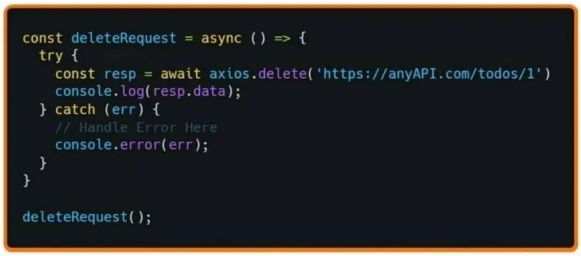
The **PUT** method allows you to update an entire object.

```
const putRequest = async () => {
  // Update TODO object with ID 1
  const updatedTodo = {
    id: 1,
    userId: 1,
    title: 'Updated task title',
    completed: true
  }
  try {
    const resp = await axios.put('https://anyAPI.com/todos/1', updatedTodo);
    console.log(resp.data);
  } catch (err) {
    console.error(err);
  }
}

putRequest();
```

DELETE REQUEST WITH AXIOS

DELETE **/delete** Deletes the record

The **DELETE** method allows you to remove a resource.



```
const deleteRequest = async () => {
  try {
    const resp = await axios.delete('https://anyAPI.com/todos/1')
    console.log(resp.data);
  } catch (err) {
    // Handle Error Here
    console.error(err);
  }
}

deleteRequest();
```

Similarly, there are other methods you can see the documentation for them.

Local storage and session storage

SAVE THIS POST

JOIN US & LEARN
WEB DEVELOPMENT

SESSION STORAGE

- 5MB Storage
- Client side reading only
- Session based
- Working per window or tab

LOCAL STORAGE

- 5MB/10MB Storage
- Client side reading only
- Not session based
- Need to be deleted via js or manually

@developers_community_._ X @gowsami.dev

SAVE THIS POST

JOIN US & LEARN
WEB DEVELOPMENT

BASIC USAGE FOR SESSION STORAGE

```
// Save data to sessionStorage
sessionStorage.setItem('key', 'value');

// Get saved data from sessionStorage
let data = sessionStorage.getItem('key');

// Remove saved data from sessionStorage
sessionStorage.removeItem('key');

// Remove all saved data from sessionStorage
sessionStorage.clear();
```

@developers_community_._ X @gowsami.dev

SAVE THIS POST

JOIN US & LEARN
WEB DEVELOPMENT

BASIC USAGE FOR LOCAL STORAGE

```
// To accesses the current domain's local Storage object
and adds a data item to it using Storage.setItem().
localStorage.setItem('myCat', 'Tom');

// Syntax for reading the localStorage item
const cat = localStorage.getItem('myCat');

// Syntax for removing the localStorage item
localStorage.removeItem('myCat');

// syntax for removing all the localStorage items
localStorage.clear();
```

@developers_community_._ X @gowsami.dev

SAVE THIS POST

JOIN US & LEARN
WEB DEVELOPMENT

SAVING TEXT BETWEEN REFRESHES

The following eg **autosaves** the ***contents of a text field,*** and if browser is refreshed, restores the text field content so that **no writing is lost.**

```
// Get the text field that we're going to track
let field = document.getElementById("field");

// See if we have an autosave value
// (this will only happen if the page is
accidentally refreshed)
if (sessionStorage.getItem("autosave")) {
  // Restore the contents of the text field
  field.value = sessionStorage.getItem("autosave");
}

// Listen for changes in the text field
field.addEventListener("change", function() {
  // And save the results into the session storage
object
  sessionStorage.setItem("autosave", field.value);
});
```

@developers_community_._ X @gowsami.dev

JS

Using Local Storage API

Local storage is a browser API that lets you save data in the browser for a longer duration.

The data is stored against the website address in the form of key/value pairs.

The data is only stored as a string. So to save arrays, objects, etc., we need to first convert them into a string using the JSON.stringify method.

Saad Irfan
@DevWithSaad

JS

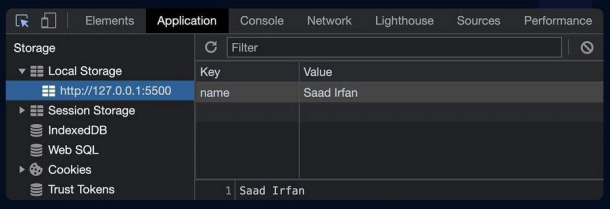
Save a value in browser

You can use local storage API inside the JavaScript of your website.

index.html

```
<script>
    const fullName = "Saad Irfan";
    localStorage.setItem('name', fullName);
</script>
```

You can see the data in the Chrome dev tools.



Saad Irfan
@DevWithSaad

JS

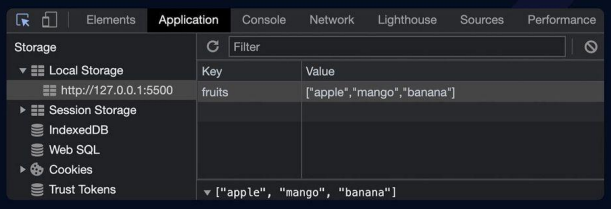
Save an array in browser

You can save an array by first converting it to string.

index.html

```
<script>
    const fruits = ['apple', 'mango', 'banana'];
    localStorage.setItem('fruits', JSON.stringify(fruits))
</script>
```

Stored data



Saad Irfan
@DevWithSaad

JS

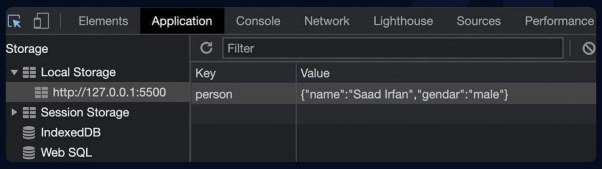
Save an object in browser

You can save an object the same way you saved an array.

index.html

```
<script>
    const person = {
        name: "Saad Irfan",
        gendar: "male"
    }
    localStorage.setItem('person', JSON.stringify(person))
</script>
```

Stored data



Saad Irfan
@DevWithSaad

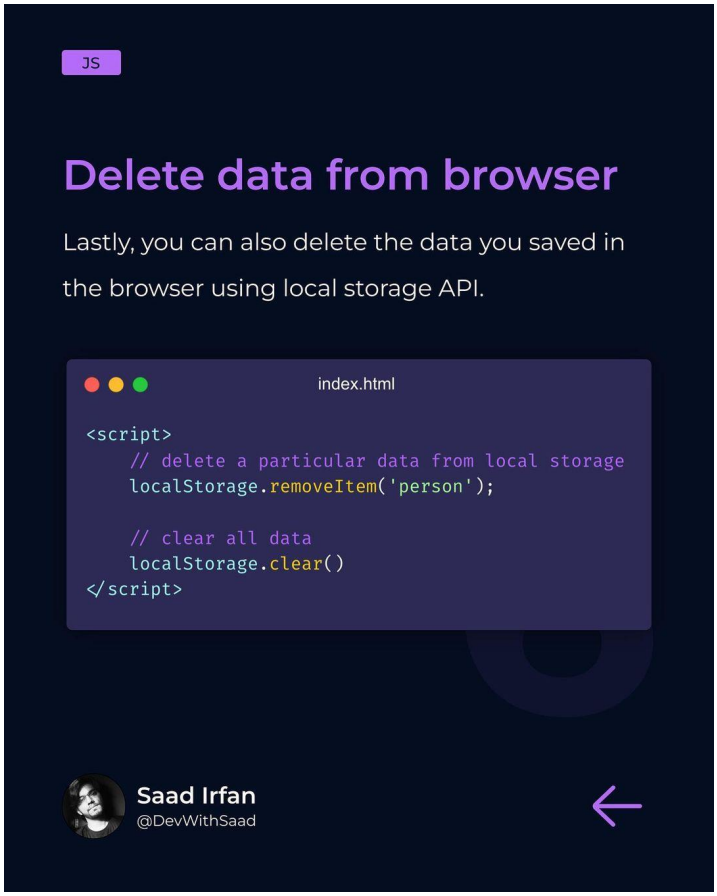
JS

Retrieve data from browser

You can also retrieve the data from the browser. If retrieved data is an object/array, make sure you first parse it to convert it to the object/array.

```
<script>
    const personDataString = localStorage.getItem('person');

    // converting string to object
    const person = JSON.parse(personDataString);
    console.log(person);

    // output →  {name: 'Saad Irfan', gendar: 'male'}
</script>
```

Saad Irfan
@DevWithSaad



Arrow functions

WHAT IS ARROW FUNCTION ?

Arrow function is one of the features introduced in the ES6 version of JavaScript. It allows you to create functions in a cleaner and shorter way as compared to regular functions.

Syntax:-

```
(param1, param2, paramN) => { statement(s) }
```

In fact, if you have only one parameter you can skip the parentheses () as well

```
const hello = name => "Hello" + name;
```

Remove

We can remove a return and { } bracket if the function have only single line statement

If you have parameters, you pass them inside the parentheses ()

```
const hello = (name) => "Hello" + name;
```

Regular function

```
function hello() {
  return "Hello world";
}
```

Arrow function

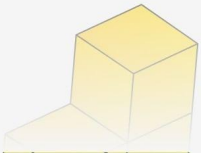
```
const hello = () => {return "Hello world"}
```

1 Arrow Functions Implicit Returns

When you plan to use a simple **function** to return some value, you can avoid the **classic** function syntax and use the **implicit return** instead by avoiding the use of **curly** braces!

```
// Longer

function multiplyNumbers(first, second) {
  return first * second
}

// Shorter

const multiplyNumbers = (first, second) => (first * second)
```

You can **wrap** your return value in **round parentheses** when you want to write more **complex** logic.



Next

2 Multiple String Checks

Instead of **chaining** multiple conditions to check if a **string** is equal to one of your **desired values,** you can make clever use of this array method and get **shorter** code as a result!

```
// Longer

const isFoodILike = (food) => {
  if (
    food === 'pizza' ||
    food === 'burgers' ||
    food === 'pasta' ||
    food === 'steak'
  ) {
    return true;
  }
  return false;
};
```

```
// Shorter

const foodsILike = ['pizza', 'burgers', 'pasta', 'steak'];
const isFoodILike = (letter) => foodsILike.includes(letter);
```

Next

3 Using Ternary Operator

Writing **if statements** can get really messy and are often easily replaced by checks using the **ternary operator.** It can even be **nested** and it still ends up much **shorter** in the end!

```
// Longer

const checkNumber = (value) => {
  let result;
  if (typeof value !== 'number') {
    result = 'Invalid Value';
  } else if (value % 2 === 0) {
    result = 'Value is even';
  } else {
    result = 'Value is odd';
  }

  return result;
}
```

```
// Shorter

const checkNumber = (value) => (
  typeof value !== 'number'
    ? 'Invalid Value'
    : value % 2 === 0 ? 'Value is even' : 'Value is odd'
)
```

Next

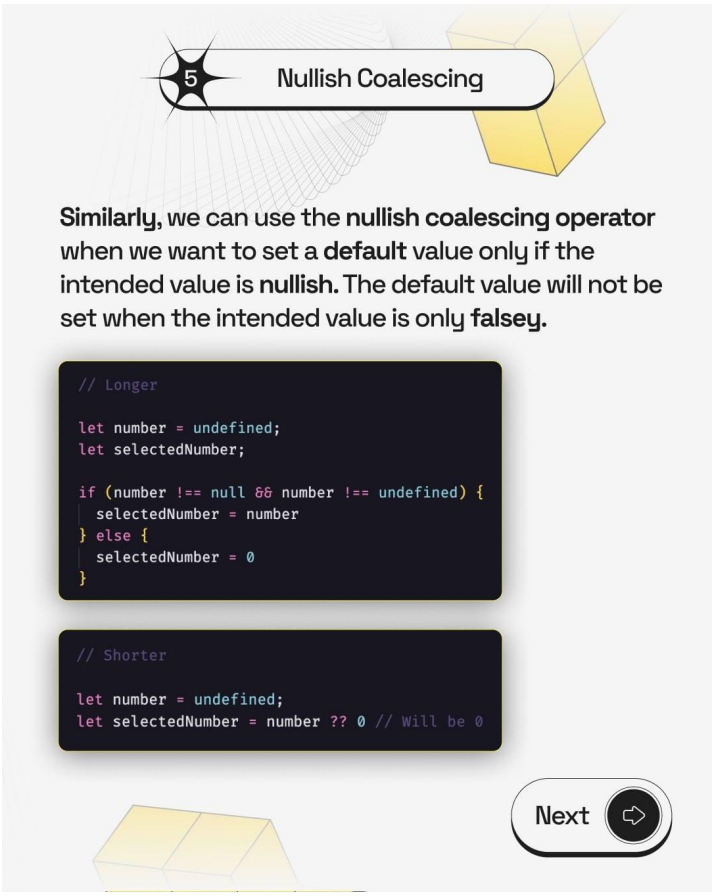
4 Setting Fallback Values

There is no need to go as far as writing an **if statement** for something as simple as setting a value based on the **validity** of another value, we can **easily** set a fallback value using the **"||" operator.**

```
// Longer

let currentUser;
if (user.name) {
  currentUser = user.name;
} else {
  currentUser = "Guest";
}
```

```
// Shorter

const currentUser = user?.name || "Guest";
```

Next



Remove elements from array

Remove the first Element from an array by its value

```
const colors = ['red', 'green', 'blue', 'yellow']

colors.shift()

console.log(colors)
// Output: ['green', 'blue', 'yellow']
```

If you want to delete only the First Element, you can use this.

Remove Element from an array by its index

```
const colors = ['red', 'green', 'blue', 'yellow']
colors.splice(2, 1)

console.log(colors)
// Output: ['red', 'green', 'yellow']
```

In the above example, you want to remove blue color at index 2

Remove object from an array by its value

```
const persons = [
  {
    id: 1,
    name: 'Roger',
  },
  {
    id: 2,
    name: 'Romane',
  },
  {
    id: 3,
    name: 'Bob',
  },
]

const filteredPersons = persons.filter((person) => person.name !== 'Bob')

console.log(filteredPersons)
// Output:
// [
//   { id: 1, name: 'Roger' },
//   { id: 2, name: 'Romane' }
// ]
```

Remove Element from an array by its value

```
const colors = ['red', 'green', 'blue', 'yellow']
const filteredColors = colors.filter((color) => color !== 'green')

console.log(filteredColors)
// Output: ['red', 'blue', 'yellow']
```

If you want to delete any element from an array in JavaScript, you can do it by using its value

Remove the Last Element from an array by its value

```
const colors = ['red', 'green', 'blue', 'yellow']

colors.pop()

console.log(colors)
// Output: ['red', 'green', 'blue']
```

If you want to delete only the last Element, you can use this.

Conditional statements

1

JS

if/else

if/else statements are how programs process yes/no questions. If the first condition evaluates to true, then the program will run the **if** block. Otherwise, it will run the **else** block.

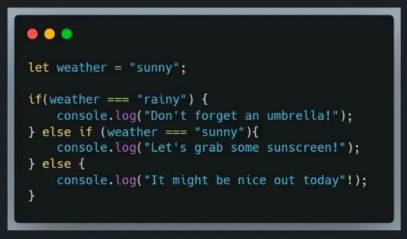
```
let weather = "rainy";
if(weather === "rainy") {
    console.log("Don't forget an umbrella today!");
}
else {
    console.log("It might be nice out today"!);
}
```

2

JS

else if

else if statements are used to add more conditions to an **if/else** statement.



```
let weather = "sunny";

if(weather === "rainy") {
    console.log("Don't forget an umbrella!");
} else if (weather === "sunny"){
    console.log("Let's grab some sunscreen!");
} else {
    console.log("It might be nice out today"!);
}
```

3

JS

True and False values

All JavaScript values have a **truthy** or **falsy** value. Declared variables are automatically given a truthy value unless the variable value contains any of the following:

- false
- 0 and -0
- “” and ‘’ (empty strings)
- null
- undefined
- NaN (not a number)

4

JS

Comparison Operators

Comparison operators are used to comparing two values and **return** true or false depending on the validity of the comparison:

```
console.log(8 !== 8); // false
console.log(5 <= 9); // true
console.log(true === "true"); // false
```

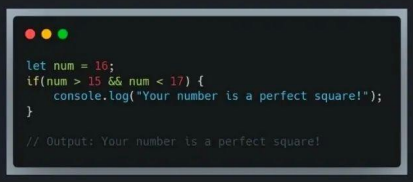
5

JS

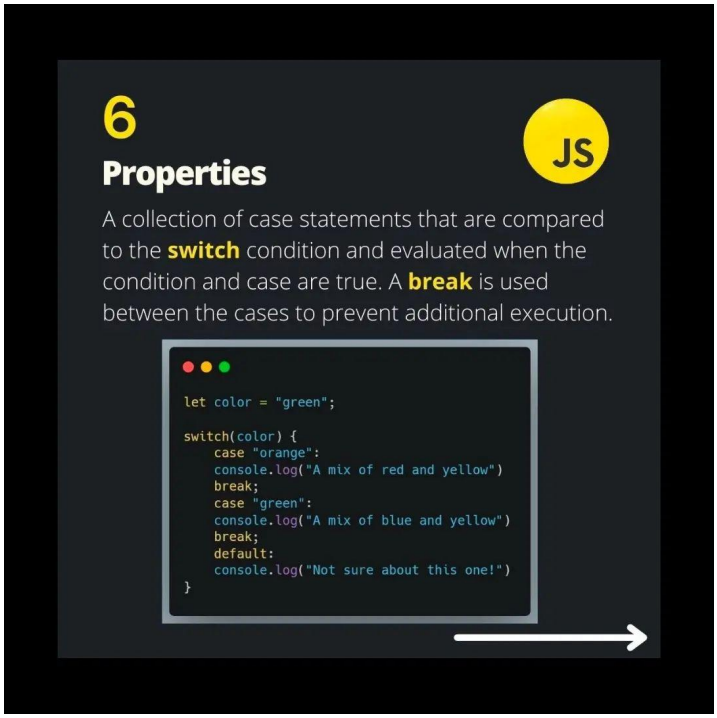
Logical Operators

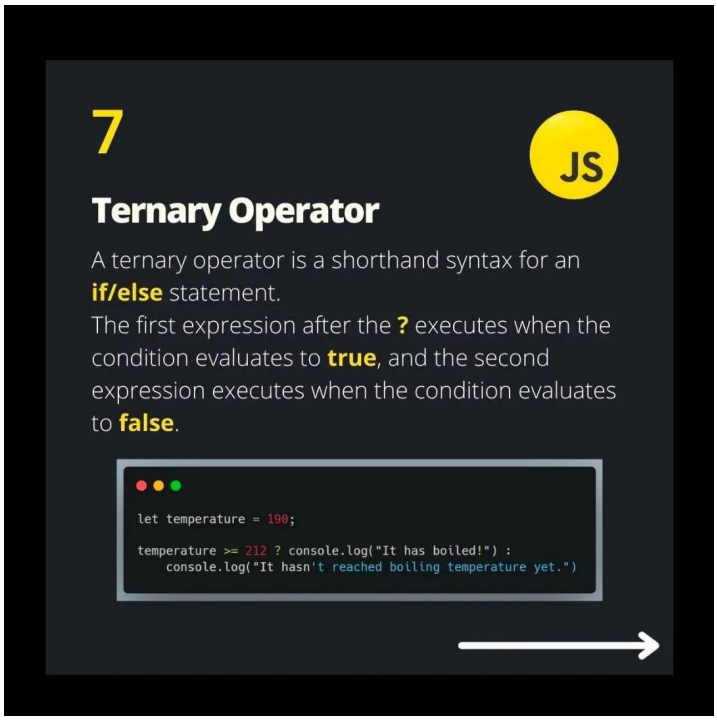
Logical operators allow us to determine if both or either of the compared values are **truthy** or **falsy**.

Use **&&** to check if **both** values are true. Use **||** to check if **either** of the values is true.

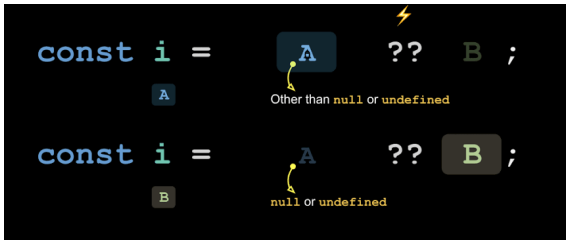


```
let num = 16;
if(num > 15 && num < 17) {
    console.log("Your number is a perfect square!");
}

// Output: Your number is a perfect square!
```

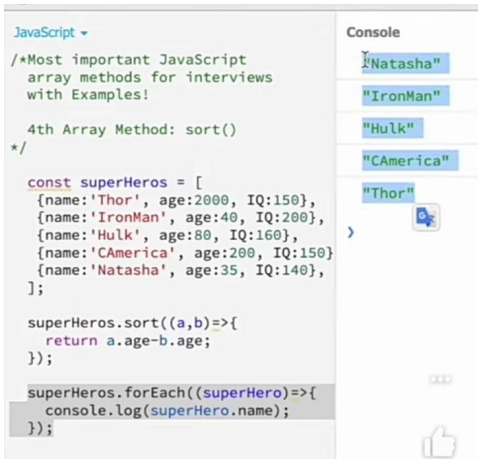




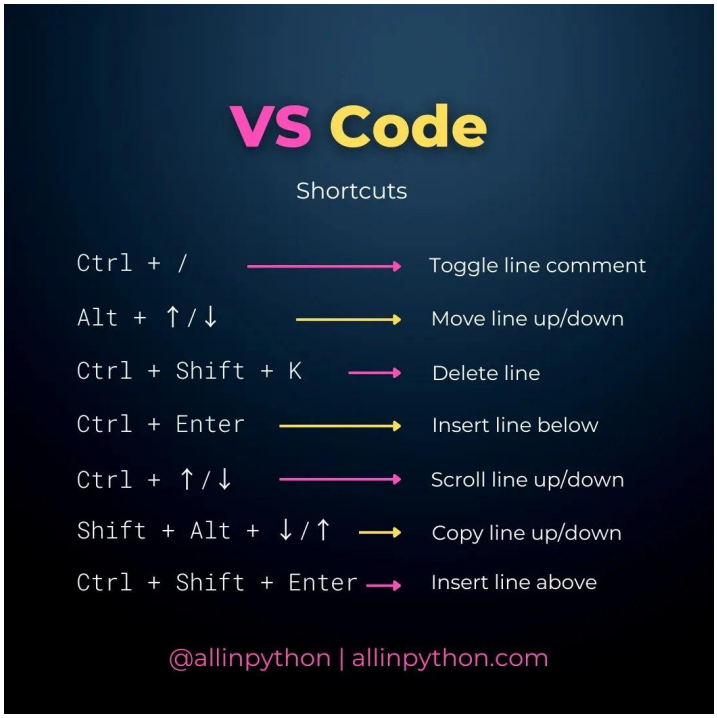
Nullish coalescing operator

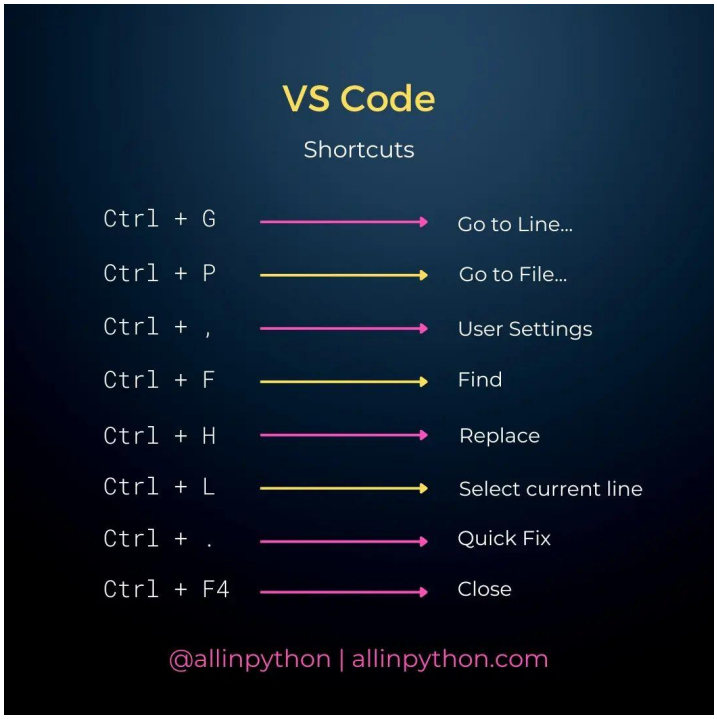


Array of objects sort



Vs code shortcut





Destructering

What is Destructuring

Destructuring was introduced in ES6. It's a JavaScript feature that allows us to extract multiple pieces of data from an array or object and assign them to their own variables.

you can also access the default value here.

1. **Destructuring in Objects**

```
const employee = {
        id= 1,
    name= " abc",
        age=24
        }

const  { name , age } = employeee
```

angular_ui_developer

2. Destructuring in Nested Objects

```
const employee = {
        id:1,
        name:  " abc",
        age:  24,
       address : {  city: "Delhi"  }
        }

const { name , address: { city }  } =
            employeee
```


angular_ui_developer

3. Destructuring in function

```
const employee = {
        id:1,
        name:  " abc",
        age:  24,
          }

const displayEmployee = ( {name, age} ) => {
 console.log(` name is ${name} and age is
                    ${age}
                      }
       displayEmployee(employee)
```


angular_ui_developer

4. Destructuring in Array

```
const fruits =  [ "apple","banana" ,"kiwi" ]


        const [ a , b ]  =  fruits
          console.log(a, b);
```

Output

apple , banana

 angular_ui_developer

5. Destructuring in Array with rest operator

```
const  fruits  =   [  "apple","banana"  ,"kiwi",'
,                        "mango" ]


        const [ ...rest  ]  =  fruits
          console.log(  rest );
```

Output

["apple" ," banana", "kiwi", "mango"]

 angular_ui_developer

6. Destructuring in Array and Object

```
const   comapny =  {
          name: " Google " ,
           location : [" Singapour" ,"India" ]
                }
```

```
const { location : [ loc ]  }  = company
```

angular_ui_developer

Spread operator

What is the spread operator?

The spread operator is a useful and quick syntax for adding items to arrays, combining arrays or objects, and spreading an array out into a function's arguments.

What else can ... do?

1.Copying an array
2.combining arrays
3.Adding an item to a list
4.Combining objects
5.Converting NodeList to an array


angular_ui_developer

Spread Operator with Array

```
const nums = [ 1, 2, 3, 4 ]
const letters = [ 'a','b' ,'c' ,'d' ]

const  newArr = [ ...nums,  ...letters ,  5,6,7  ]
```

Output

[1, 2, 3, 4, 'a', 'b', 'c', ' d', 5, 6, 7]


angular_ui_developer

Spread Operator with Array of Objects

```
const persons =[ { id: 1, name: "X" },
{ id:2, name: "Y" },   { id: 3, name: "Z" } ]

const newPerson = { id : 4 , name : " R" }

const updatedPerson = [ ...persons ,
newPerson  ]
```

output

```
[ { id: 1, name: "X" },
{ id:2, name: "Y" },   { id: 3, name: "Z" } , {id: 4, name:" R" } ]
```

angular_ui_developer

Spread Operator with Objects

```
const book = {
title : " Rich Dad Poor Dad",
author : " Robert T. Kiyosaki" ,
chapters: {
chapter1 : " The Rich don't work for money ",
chapter2 : " The Rich invest money "
}
}
const updatedBook = {
...book,
chapters :{
...book.chapters,
chapter3: " Mind Your Own Business "
}
}
```

angular_ui_developer

Spread Operator with Function

```
function sum (a, b, c){
return console.log( a +b+ c )
}
const num =[ 1, 2 , 3]
sum ( ...num )
```

output

6

angular_ui_developer

Clone objects

Using Spread Operator (...)

The spread operator is useful for adding items to arrays, combining arrays or objects, and spreading an array out into a function's arguments.

```
const person = { firstName: "John",lastName:" Maria" }

const person2 = { ...person }
```

output

```
{ firstName: "John",lastName: "Maria"
```

angular_ui_developer

Using Object.assign

It is used to copy the values and properties from one or more source objects to a target object.

Syntax

```
Object.assign(target, ...sources)
```

target: It is the target object to which values and properties have to be copied.

sources: It is the source object from which values and properties have to be copied.

```
const objOne = { a:1 ,b:2, c:3 }
const objTwo = { d:4,e:5 }

const cloneObj = Object.assign( {}, objOne,  objTwo)
```

output

```
{a: 1, b: 2, c: 3, d: 4, e: 5}
```

angular_ui_developer

Using JSON

We can do the deep clone with JSON

Advantages

1. The Complex Objects are deep cloned using JSON
2. The Object is Deep Cloned using this method..

DisAdvantage

functions are not copied

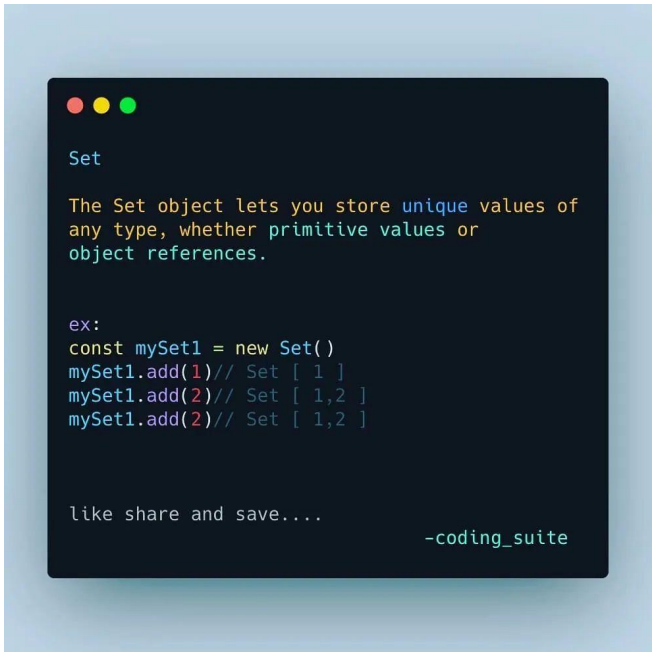
```
const objOne = { a:1, b:2 }

const objTwo = JSON.parse (  JSON.stringify (  objOne ) )
```

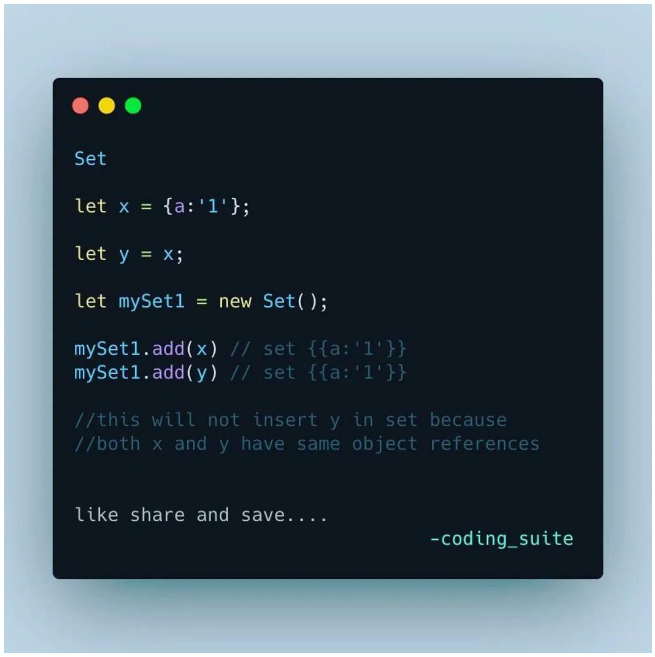
Output

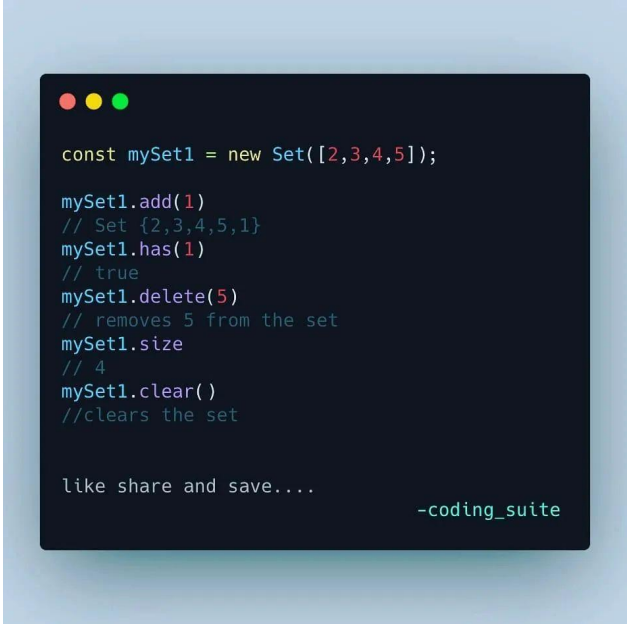
{a: 1, b: 2}

angular_ui_developer

Set method


Set
The Set object lets you store unique values of any type, whether primitive values or object references.
ex:
const mySet1 = new Set()
mySet1.add(1)// Set [1]
mySet1.add(2)// Set [1,2]
mySet1.add(2)// Set [1,2]
like share and save....
-coding_suite


Set
let x = {a:'1'};
let y = x;
let mySet1 = new Set();
mySet1.add(x) // set {{a:'1'}}
mySet1.add(y) // set {{a:'1'}}
//this will not insert y in set because
//both x and y have same object references
like share and save....
-coding_suite

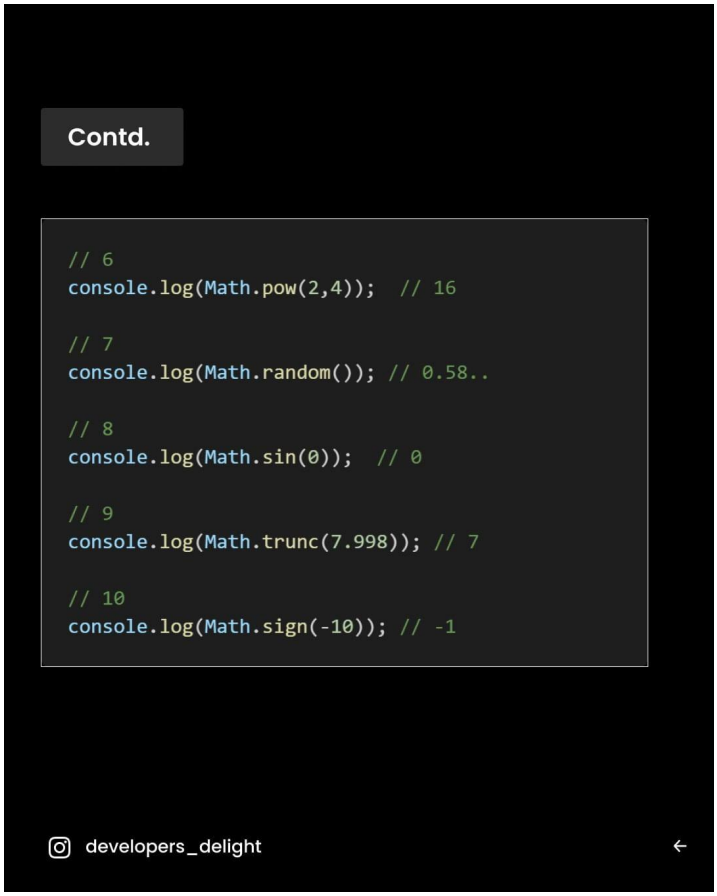


```
const mySet1 = new Set([2,3,4,5]);

mySet1.add(1)
// Set {2,3,4,5,1}
mySet1.has(1)
// true
mySet1.delete(5)
// removes 5 from the set
mySet1.size
// 4
mySet1.clear()
//clears the set


like share and save....
                                -coding_suite
```

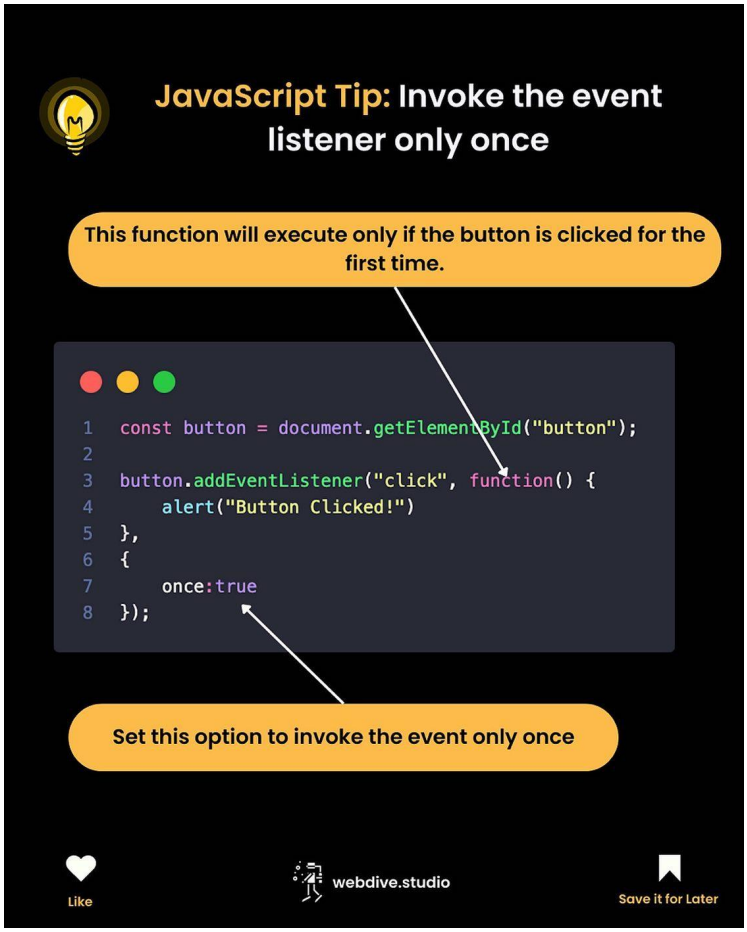
Math property

MATH methods :

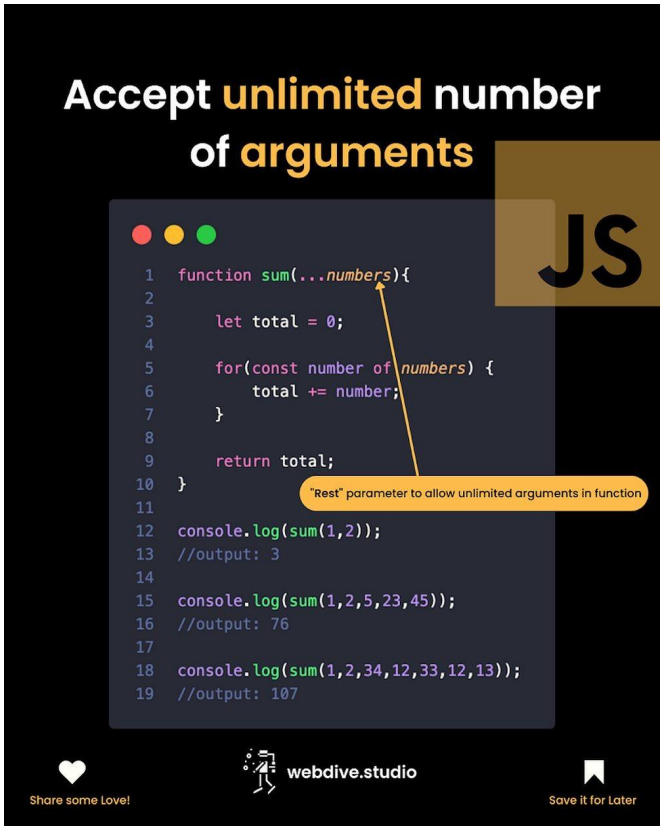
```
// 1
console.log(Math.ceil(23.8));  // 24

// 2
console.log(Math.abs(-11)); // 11

// 3
console.log(Math.floor(23.8));  // 23

// 4
console.log(Math.log(10)); // 2.3..

//5
console.log(Math.max(5, 10)); // 10
```



Button click one time or first time event trigger



Add function using reduce



Dom manipulation by I'd

Creating elements

```
//Syntax
const element = document.createElement('element')
parentElement.appendChild(element);

//Example 1
const div = document.createElement('div')
body.appendChild(div);

//Example 2
const ul = document.createElement('ul');
const li = document.createElement('li');
ul.appendChild(li);
```

@coder_aishya

Removing elements

```
//Syntax
// with removeChild method
parent.removeChild(child)
// with remove method
child.remove()

//Example
ul.removeChild(li)
// or
li.remove()
```

@coder_aishya

Querying elements

- The querySelector() method returns the first element that matches a CSS selector.
- To return all matches (not only the first), use the querySelectorAll() instead.

```
//Syntax
// query by id
const element = document.querySelector('#elementId');

// query group of elements by id
//(which ignores the purpose of having a special id)
const elements.querySelectorAll('#elementId');

// query element by class name
const element = document.querySelector('.className');

// query group of elements by class name
const elements = document.querySelectorAll('.className');
```

@coder_aishya

Modify elements

modifying element's attributes

modifying the style object

```
element.style.fontSize = "18px"
element.style.backgroundColor = "#ffffff"
```

modifying the id

```
element.id = "myId"
```

@coder_aishya

modifying the class

```
element.classList.add('myClass');

element.classList.remove('myClass');
```

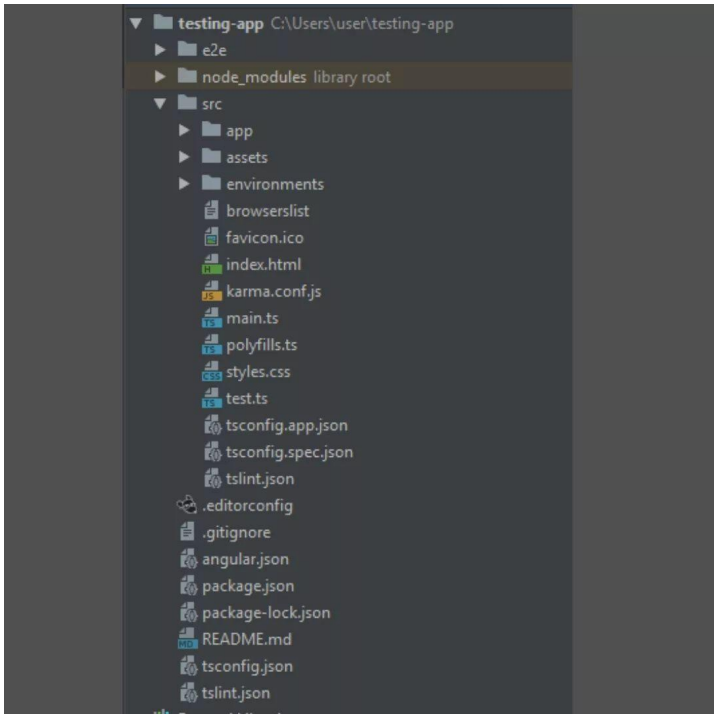
modifying the text

```
const element.innerText = "Hello World"
```

@coder_aishya

Angular

Angular Folder structure



- **dist folder**

The "dist" folder is the build folder that has all the files and folders which **can be hosted in the server**. It has the **compiled code** of our angular project including .js, .html and .css files.

- **e2e folder**

1. "src" folder
    a. app.e2e-spec file
    b. app.po file
2. protractor.conf file
3. tsconfig file

This is the **default configuration for the e2e** folder created by the Angular CLI. e2e stands for end-to-end and it refers to **e2e tests of the website**. It is used to track user behavior on the website while testing.

- **node_modules folder**

This folder is **generated when we run "npm install"** command. This folder contains **third-party libraries** and files. All these files are bundled in our project together. We **don't need this folder while deploying** our application somewhere.

@coder_aishya

- **app folder**

Contains **"directives", "modules" , "services" and components"** for our Angular application.

By default it basically has,

1. **app.component.css** - Contains the CSS code for the component.
2. **app.component.html** - HTML file pointing to the app component. It is a template for the angular application.
3. **app.component.spec.ts** - Unit testing file associated with app component. It can be generated using "ng test" command.
4. **app.component.ts** - Entire functional logic is written in this file.
5. **app.module.ts** - TypeScript file holds all dependencies. Here we will use "NgModule" and define the Bootstrap component when loading the application.

- **assets folder**

Here we will keep resources such as **images, styles, icons**, etc.

- **environments folder**

It contains the **environment configuration constants** that help while building the angular application. It has **environment.ts and environment.prod.ts.** These configurations are used in angular.json file.

- **favicon.io**

The icon appears in the browser tab of our application.

- **index.html**

Basic HTML file & is starting point of Angular application which has app.component.html embeded in it

- **main.ts**

The starting point of our application. It **bootstraps/starts the AppModule** from the app.module.ts file.

- **polyfills.ts**

This file is used to compile our TypeScript to specific JavaScript methods. Provides compatibility support for Browser versions.

- **styles.css**

Global CSS file.

- **tests.ts**

It is the **main test file.** When we **run the "ng test"** command, this file is taken into consideration.

- **.browserslistrc file**

Browser compatibility and versions are mentioned in this file. This configuration is pointed to in our package.json file.

- **.editorconfig**

This file deals with consistency in code editors to organize some basics such as indentation and whitespaces. More like code formatting.

- **angular.json**

This file defines the structure of our application. It includes settings associated with our application. Also, we can specify the environments on this file. For example, development, production, etc.

- **karma.conf.js**

This is the configuration file for the Karma Test Runner. It is used in Unit Testing.

- **package.json**

This is the **npm configuration file.** All the dependencies mentioned in this file. We can modify dependency versions as per our need on this file.

- **package-lock.json**

Whenever we change something on the node_modules or package.json, this file will be generated. It is associated with npm.

- **README.md**

This file is created by default. It contains our project description. It shows how to build and which Angular CLI version has been used.

- **tsconfig.app.json**

This configuration file overrides the tsconfig.json file with relevant app-specific configurations.

- **tsconfig.base.json**

This file has been introduced in Angular 10+. It has the same configuration as compared to tsconfig.json file.

- **tsconfig.json**

TypeScript compiler configuration file. This is responsible for compiling TypeScript to JavaScript so that the browser will understand.

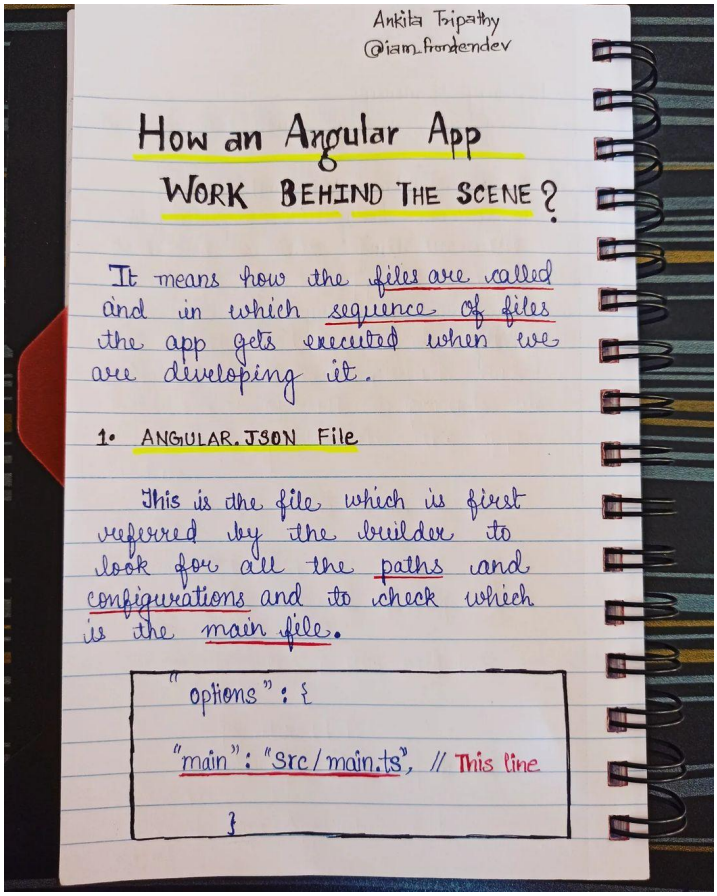
- **tsconfig.spec.json**

This file overrides the tsconfig.json file with app-specific unit test configurations while running the "ng test" command.

- **tslint.json**

tslint.json is a static analysis tool. This file keeps track of the TypeScript code for readability, maintainability, and functionality errors.

Angular work flow


Ankita Tripathy
@iam_frontendev
How an Angular App
WORK BEHIND THE SCENE?
It means how the files are called and in which sequence of files the app gets executed when we are developing it.
1. ANGULAR.JSON File
This is the file which is first referred by the builder to look for all the paths and configurations and to check which is the main file.
"options": {
"main": "src/main.ts", // This line
}

Ankita Tripathy
@iam_frontendev

It has a reference to the main.ts file which tells the builder to start the app from there.

2. MAIN.TS

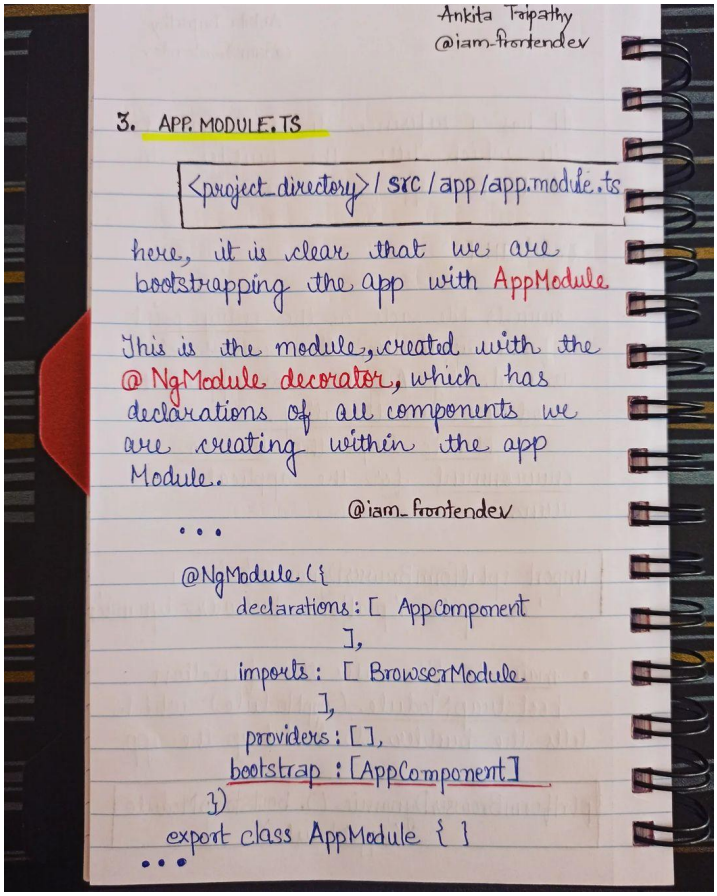
main.ts file acts as the entry point of the application. This entry point is used by Angular to support the modular functionality.

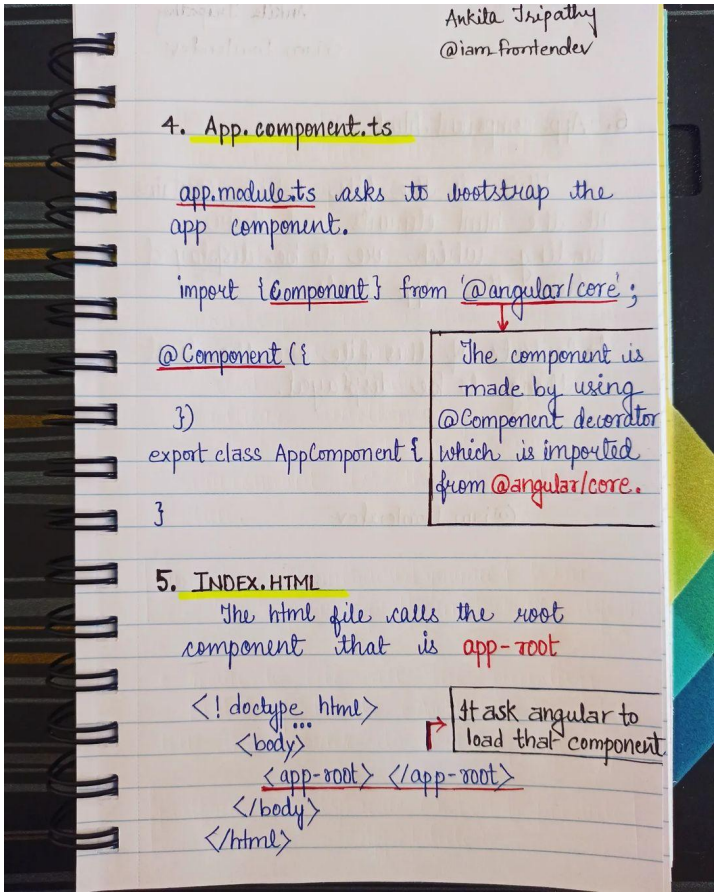
Main.ts helps the browser environment for the application to run.

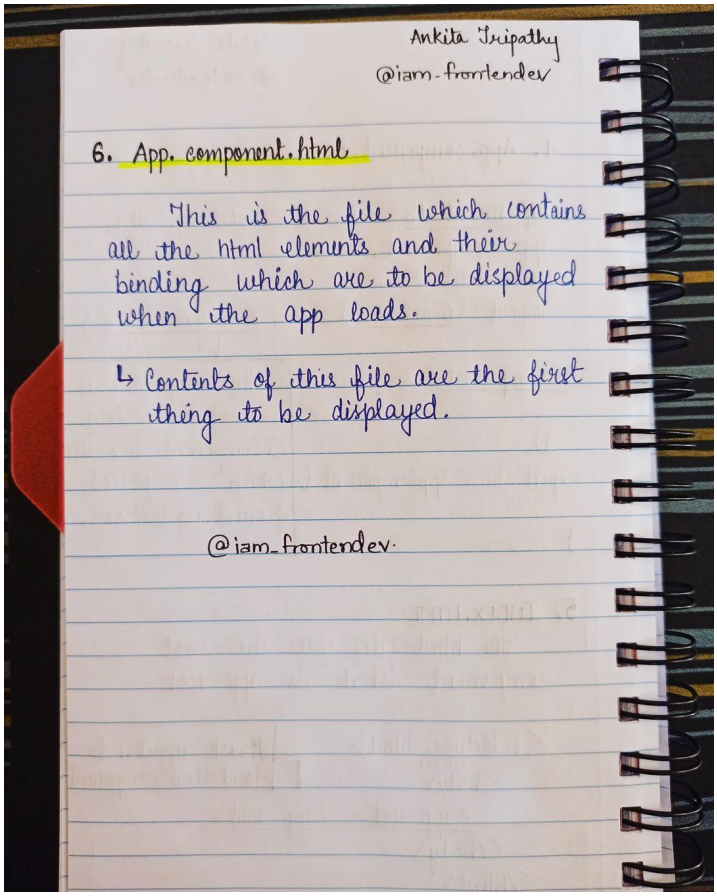
```
import {platformBrowserDynamic} from
'@angular/platform-browser-dynamic';
```

- main.ts file calls the function bootstrapModule (AppModule) which tells the builder to bootstrap the app.

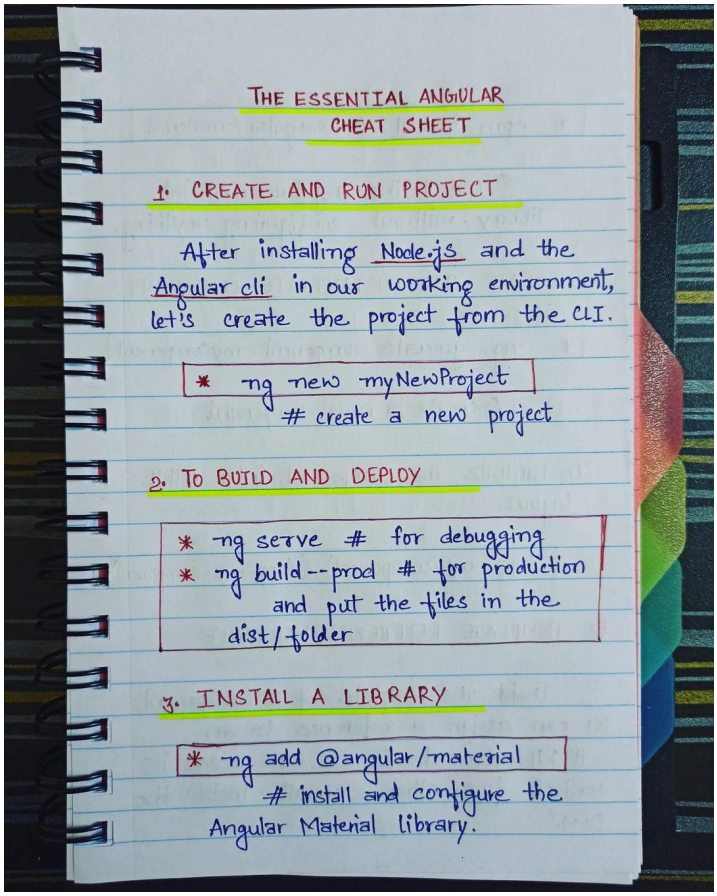
```
platformBrowserDynamic().bootstrapModule
(AppModule)
```

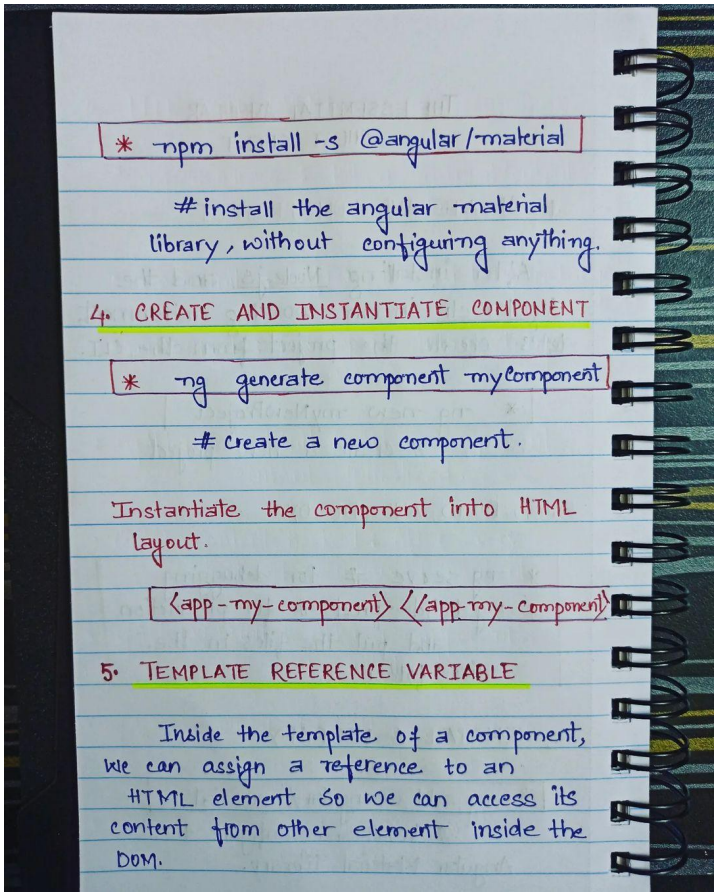

Ankita Tripathy
@iam_frontendev
3. APP.MODULE.TS
<project_directory> / src / app / app.module.ts
here, it is clear that we are bootstrapping the app with AppModule.
This is the module, created with the @NgModule decorator, which has declarations of all components we are creating within the app Module.
@iam_frontendev
...
@NgModule({
declarations: [AppComponent
],
imports: [BrowserModule
],
providers: [],
bootstrap : [AppComponent]
})
export class AppModule { }
...

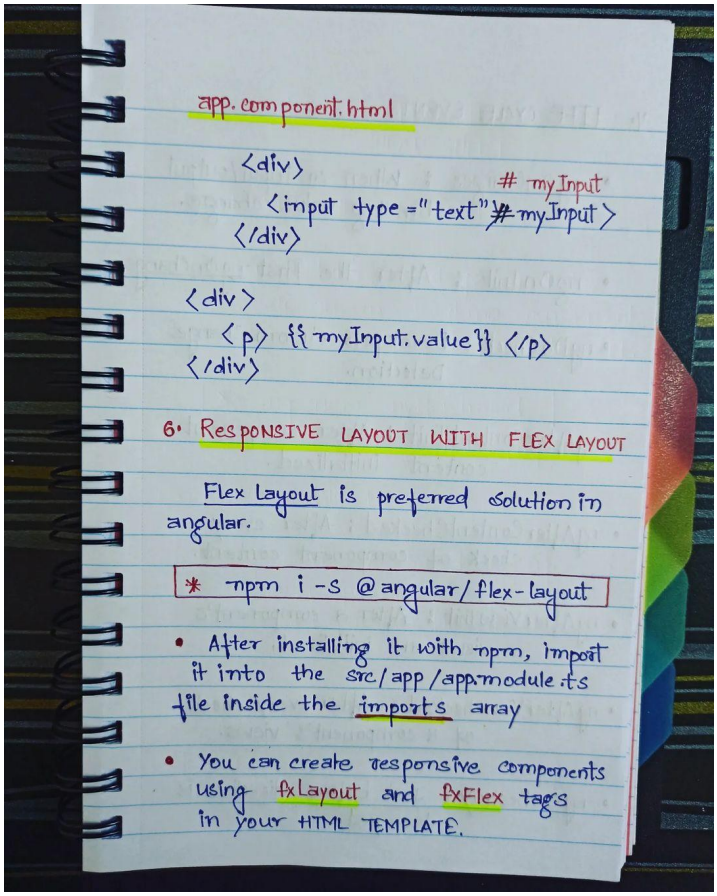

Ankita Tripathy
@iam_frontendev
4. App.component.ts
app.module.ts asks to bootstrap the app component.
import {Component} from '@angular/core';
@Component({
})
export class AppComponent {
}
The component is made by using @Component decorator which is imported from @angular/core.
5. INDEX.HTML
The html file calls the root component that is app-root
<!doctype html>
...
<body>
<app-root></app-root>
</body>
</html>
It ask angular to load that component



Angular command


THE ESSENTIAL ANGULAR CHEAT SHEET
1. CREATE AND RUN PROJECT
After installing Node.js and the Angular cli in our working environment, let's create the project from the CLI.
* ng new myNewProject
# create a new project
2. TO BUILD AND DEPLOY
* ng serve # for debugging
* ng build --prod # for production and put the files in the dist/ folder
3. INSTALL A LIBRARY
* ng add @angular/material
# install and configure the Angular Material library.


* npm install -s @angular/material
# install the angular material library, without configuring anything.
4. CREATE AND INSTANTIATE COMPONENT
* ng generate component myComponent
# create a new component.
Instantiate the component into HTML layout.
<app-my-component></app-my-component>
5. TEMPLATE REFERENCE VARIABLE
Inside the template of a component, we can assign a reference to an HTML element so we can access its content from other element inside the DOM.



app.component.html

```
<div>
    <input type="text" #myInput>
</div>

<div>
    <p> {{ myInput.value }} </p>
</div>
```

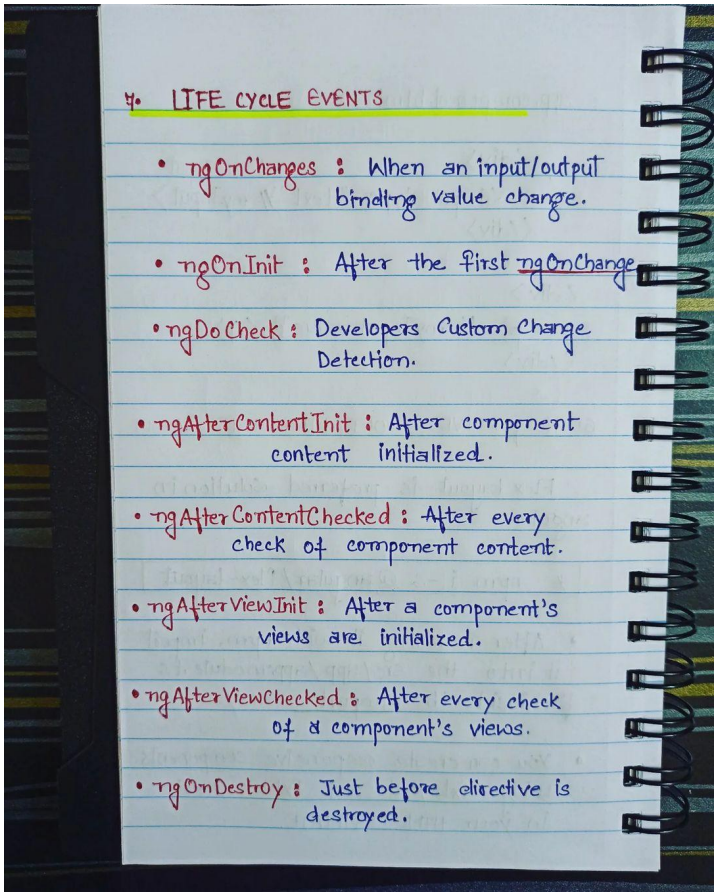
myInput

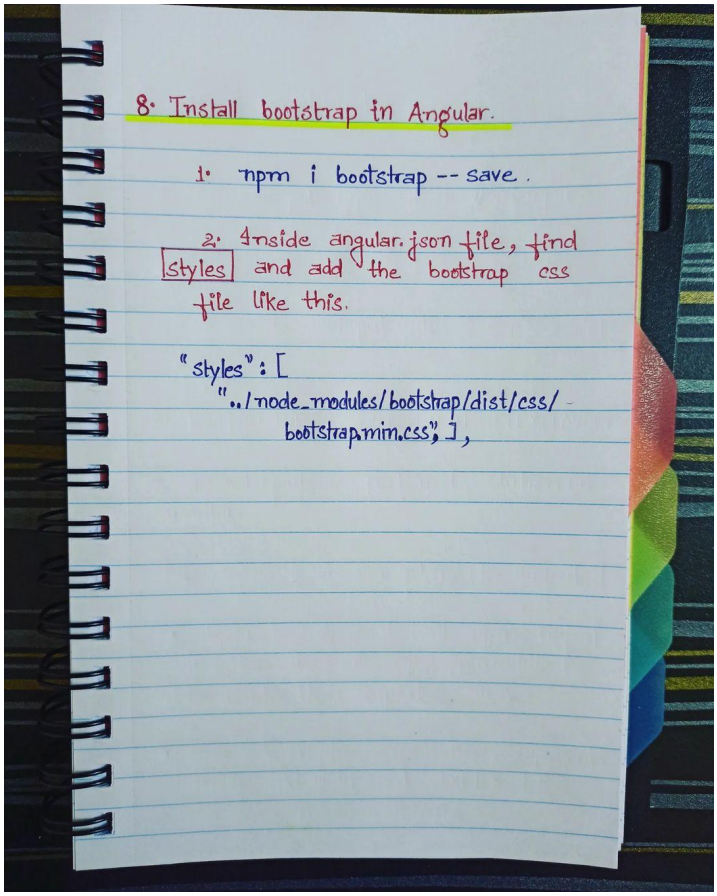
6. RESPONSIVE LAYOUT WITH FLEX LAYOUT

Flex Layout is preferred solution in angular.

* npm i -s @angular/flex-layout

- After installing it with npm, import it into the src/app/app.module.ts file inside the imports array
- You can create responsive components using fxLayout and fxFlex tags in your HTML TEMPLATE.

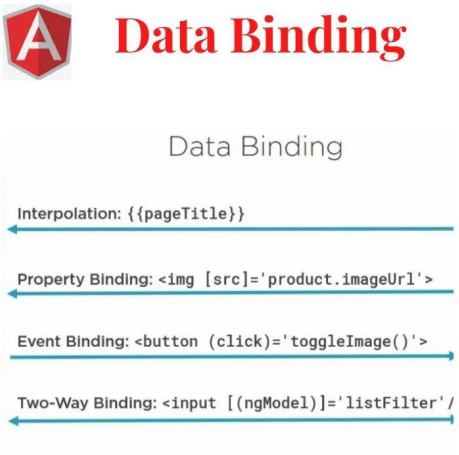

4. LIFE CYCLE EVENTS
• ngOnChanges : When an input/output binding value change.
• ngOnInit : After the first ngOnChange
• ngDoCheck : Developers Custom Change Detection.
• ngAfterContentInit : After component content initialized.
• ngAfterContentChecked : After every check of component content.
• ngAfterViewInit : After a component's views are initialized.
• ngAfterViewChecked : After every check of a component's views.
• ngOnDestroy : Just before directive is destroyed.



Component and module

Module	Component
A module instead is a collection of components, services, directives, pipes and so on.	A component in Angular is a building block of the Application with an associated template.
Denoted by @ngModule	Denoted by @Component
The Angular apps will contain many modules, each dedicated to the single purpose.	Each component can use other components, which are declared in the same module. To use components declared in other modules, they need to be exported from this module and the module needs to be imported.

Data binding



String Interpolation

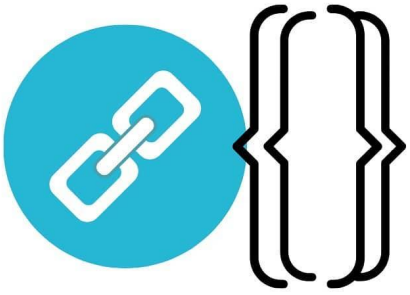
String interpolation is one way data binding technique used to transfer the data from typescript code to an html template. It adds the value of the property from the component to the html template view.

Syntax:

{{component_property}}

angular. guides

Interpolation {{}}
in
Angular Application

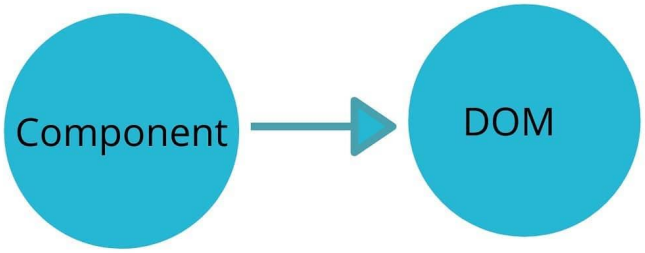


angular_ui_developer

What is Interpolation {{}}

Interpolation allow us to display the value of property from the **component** to **DOM(HTML template)**

It bind the data in **one way**



angular_ui_developer

There are different ways to pass data from component to DOM

1. Interpolation **with variable**

app.component.ts

```
name : string :  "xyz"
```

app.component.html

```
<h1> {{ name }} </h1>
```

angular_ui_developer

2. Interpolation **with Object**

app.component.ts

```
user = {
name: 'xyz' ,
email:'xyz@gmail.com'
    }
```

app.component.html

```
<h1> {{user.name}} </h1>
<h1> {{user.email}} </h1>
```

angular_ui_developer

3. Interpolation **with function**

app.component.ts

```
email: 'xyz@gmail.com';

getEmail() {
return this.email
}
```

app.component.html

```
<h1> {{ getEmail() }} </h1>
```

angular_ui_developer

4. Interpolation with Airthmatic Operation

app.component.ts

```
public  a = 10;
public  b = 50;
```

app.component.html

```
<p> {{ a + b }} </p>
```

angular_ui_developer

4. Interpolation with Array

app.component.ts

```
arr = [ 'abc' , 'pqr' , 'rts' , 'mno' ]
```

app.component.html

```
<p> {{ arr}} </p>
```

It will print all value from an array

abc,pqr,rts,mno

```
<p> {{ arr[1] }} </p>
```

It will print one value at position one **pqr**

```
<p> {{ arr.length }} </p>
```

it will print the length of an array **4**

angular_ui_developer

Property Binding

Property binding in Angular helps you set values for properties of HTML elements or directive
With property binding, you can do things such as toggle button functionality, set paths programmatically, and share values between components.

angular. guides

Syntax:

<element [property]= 'typescript_property'>

Approach

- Define a property element in the app.component.ts file.
- In the app.component.html file, set the property of the HTML element by assigning the property value to the app.component.ts file's element.

angular. guides

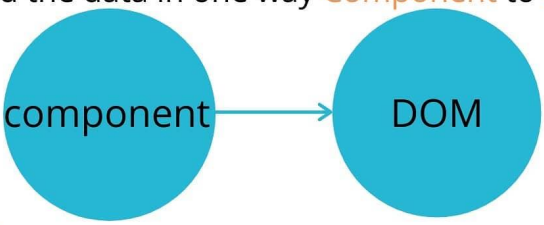
What is Property Binding

Property binding is a technique that allow the user to bind the **properties of DOM Elements** , **Components** and **Directives.**

Syntax

[property] = 'expression'

It bind the data in one way Component to DOM



1. Property Binding with Properties of DOM Elements

app.component.ts

```
isDisabled = true;
student = {
imagePath : ' . /assets/images/abc.png'
}
```

app.component.html

```
<img [src]= ' student.imagePath '/ >
<button [disabled] = ' isDisabled '   >Disabled
Button </button>
```

2. Property Binding between components

app.component.ts

```
titleA = ' Hello! Developers";
```

app.component.html

```
<app-xyz   [ titleB ]=' titleA ' ></app-xyz>
```

xyz.component.ts

```
@Input titleB: string
```

xyz.component.html

```
<h1> {{titleB}} </h1>
```

angular_ui_developer

3. Property Binding with Directives

app.component.css

```
.blueText {
color : red;
}
```

app.componen.ts

```
text = 'blueText' ;
title = 'Welcome! '
```

app.component.html

```
<p  [ngClass]= ' blueText '>{{ title }}</p>
```

angular_ui_developer

Event Binding

event binding is used to handle the events raised by the user actions like button click, mouse movement, keystrokes, etc. When the DOM event happens at an element(e.g. click, keydown, keyup), it calls the specified method in the particular component.

Angular.guides

Syntax

< element (event) = function() >

Approach

- Define a function in the app.component.ts file which will do the given task.
- In the app.component.html file, bind the function to the given event on the HTML element.

Angular.guides

Two way data binding

Angular allows two-way data binding that will allow your application to share data in two directions i.e. from the components to the templates and vice versa. This makes sure that the models and the views present in your application are always synchronized. Two-way data binding will perform two things i.e. setting of the element property and listening to the element change events.

Angular.guides

Syntax

<element [(ngModel)] ="property">

Example

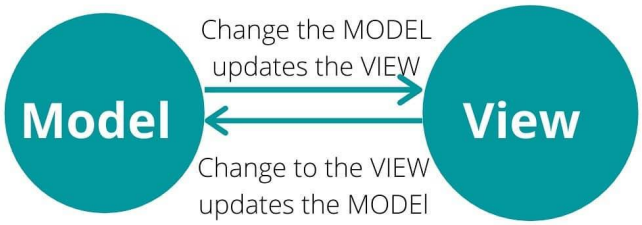
In html we can write :
<input [(ngModel)]="name">

In ts file declare name:
name:string = "angular guides"

Angular.guides

What is Two Way Data Binding

In Two way data Binding Data flow between View To Component .
It is a bi-directional binding.
It is combination of **property binding []** and **event binding ()**



angular_ui_developer

Two way data binding with [(ngModel)]

NgModel directive allows user to display a data property and update that property when the user makes changes

we need to Import FormsModule to use ngModel

```
import  {FormsModule} from '@angular/forms' ;
```

app.component.ts

```
name  = " "
```

app.component.html

```
<input  type="text"  [(ngModel)]= "name">
<h1>{{name}}</h1>
```

angular_ui_developer

Two way data binding without [(ngModel)]

app.component.ts

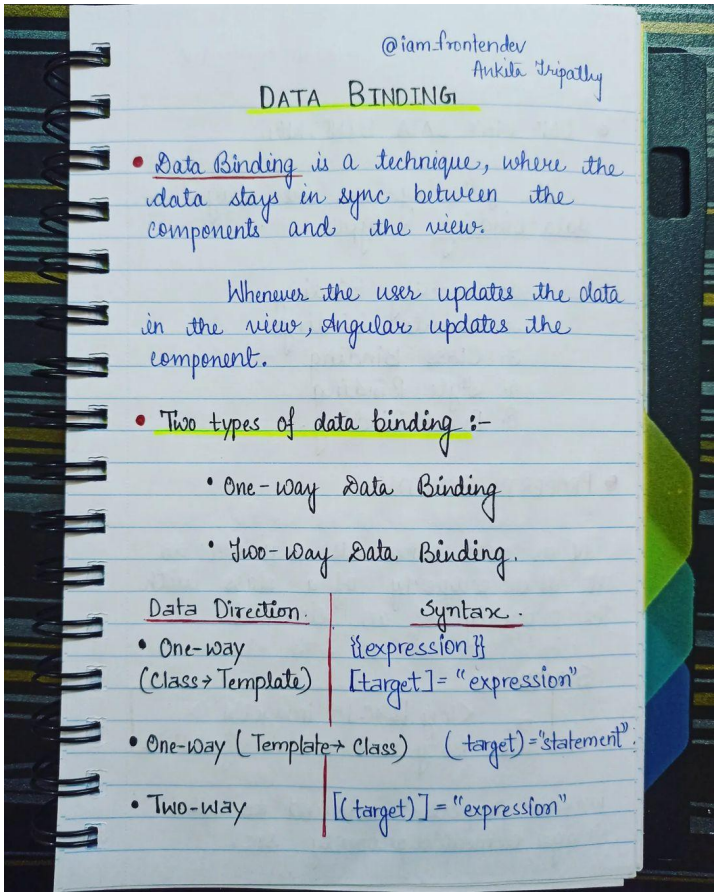
```
data : string = " peter "
```

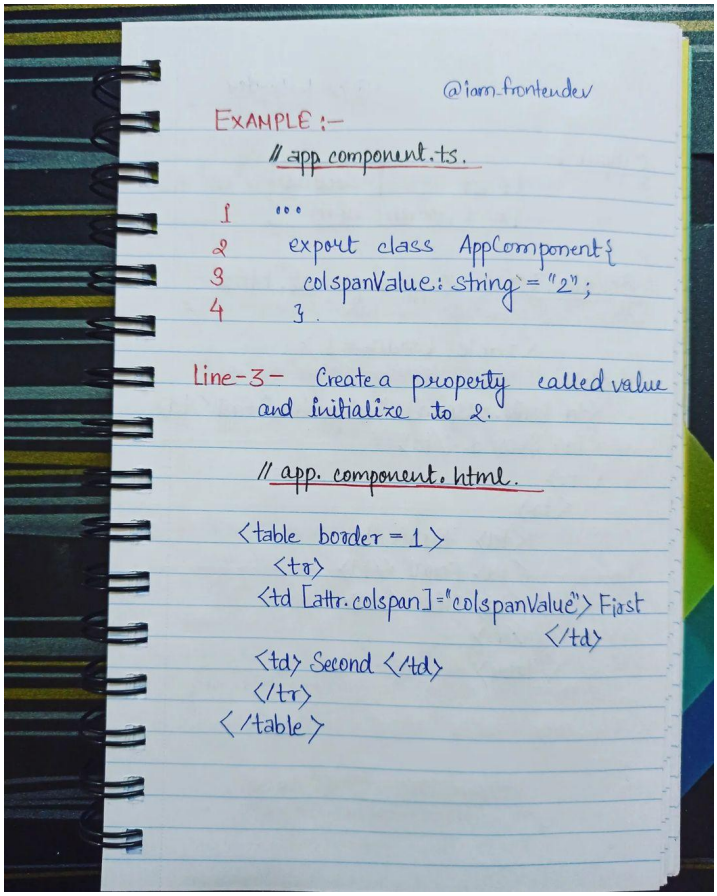
app.component.html

```
<input  type="text" [value]="data"
(input)="data-$event.target.value>
        <h1>{{data}}</h1>
```

angular_ui_developer

Data binding another


@iam_frontendev
Ankita Tripathy
DATA BINDING
• Data Binding is a technique, where the data stays in sync between the components and the view.
Whenever the user updates the data in the view, Angular updates the component.
• Two types of data binding :-
• One-way Data Binding
• Two-way Data Binding.
Data Direction.
Syntax.
• One-way (Class → Template)
{{expression}}
[target] = "expression"
• One-way (Template → Class)
(target) = "statement"
• Two-way
[(target)] = "expression"



@iam_frontendev

EXAMPLE :-

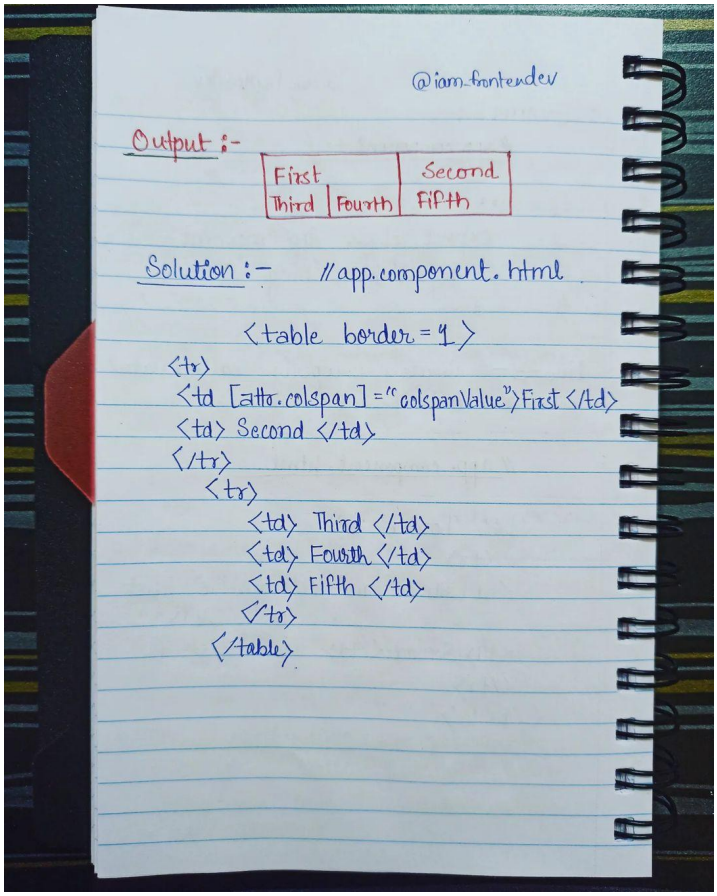
// app.component.ts.

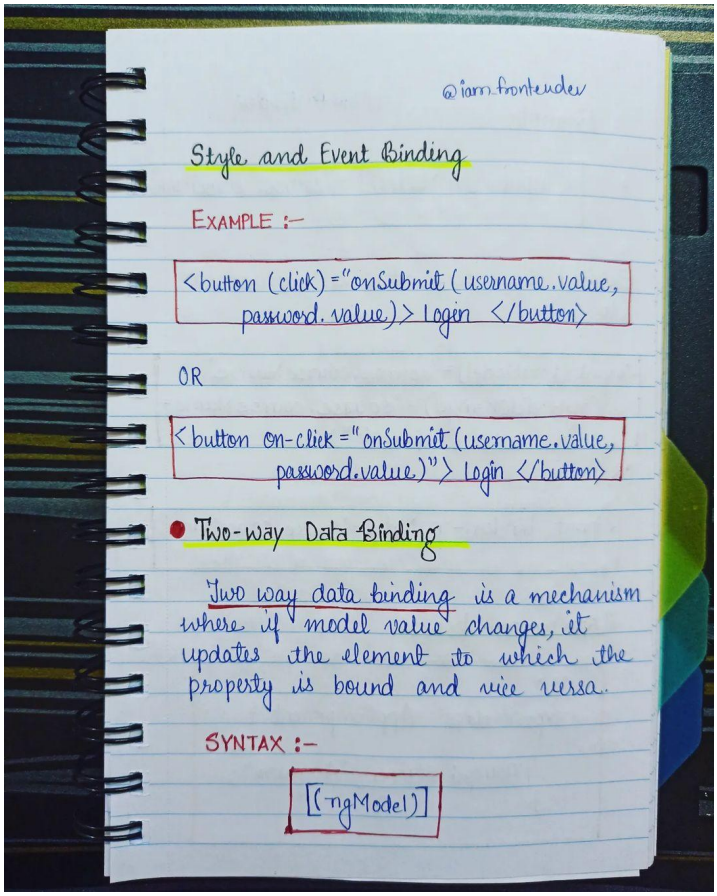
```
1   ...
2     export class AppComponent{
3       colspanValue: string = "2";
4     }.
```

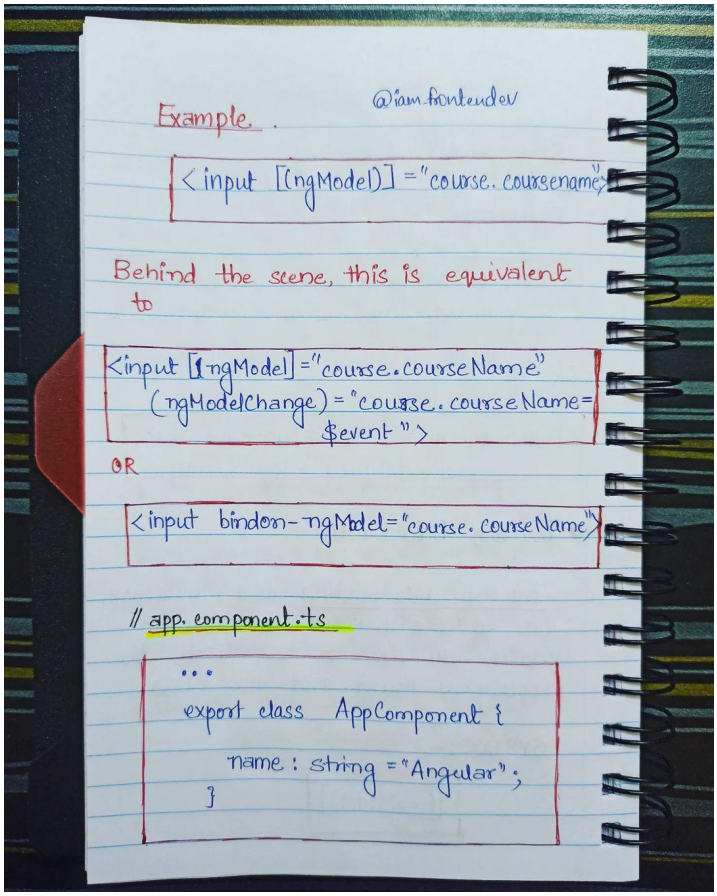
Line-3- Create a property called value and initialize to 2.

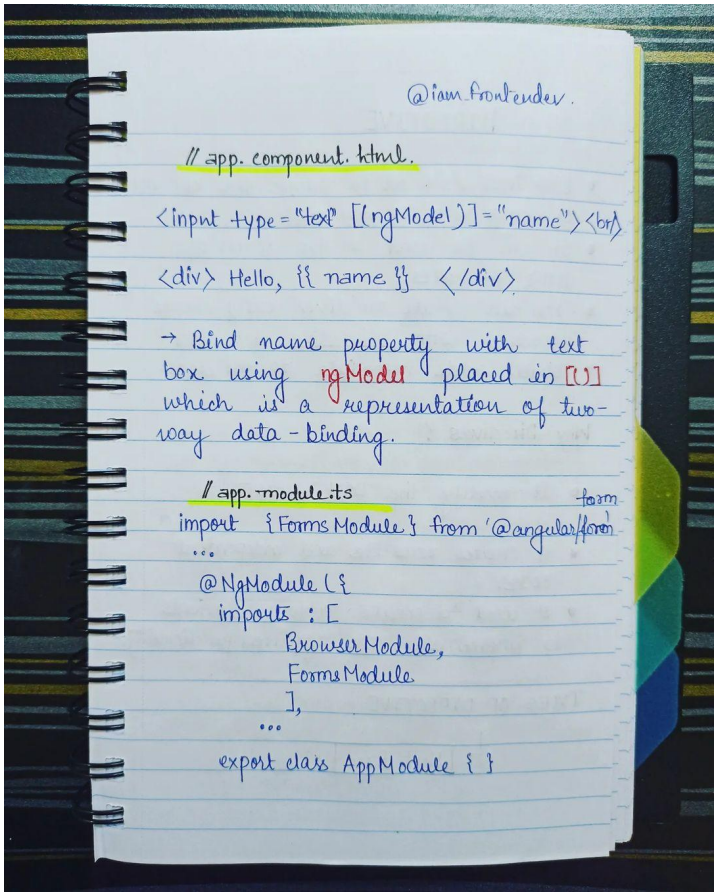
// app.component.html.

```
<table border = 1 >
  <tr>
  <td [attr.colspan]="colspanValue"> First
                                    </td>
  <td> Second </td>
  </tr>
</table>
```

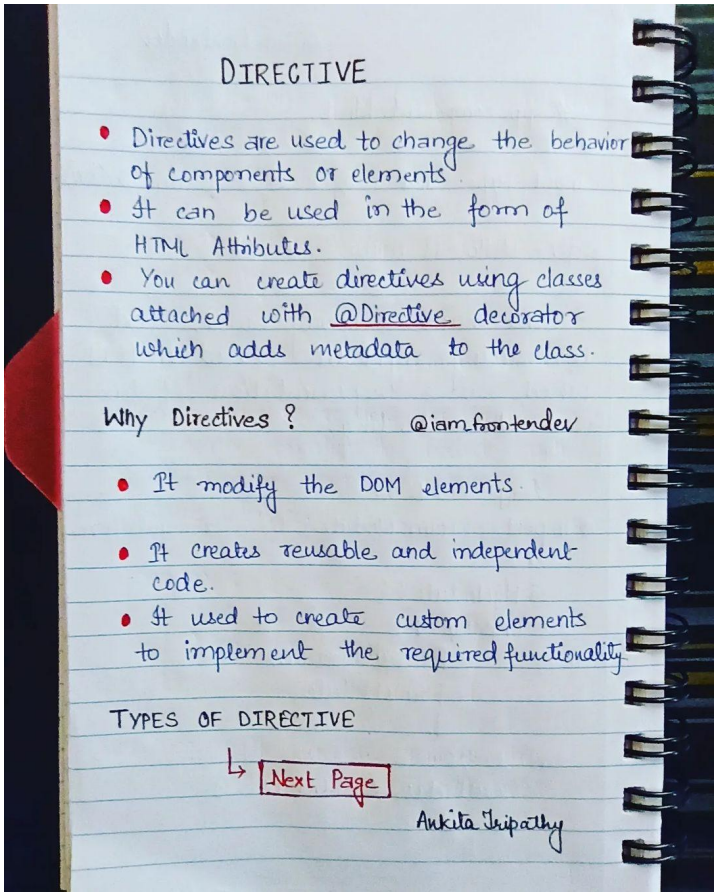

@iam_frontendev
Output :-
First
Second
Third
Fourth
Fifth
Solution :- //app.component.html
<table border = 1>
<tr>
<td [attr.colspan]="colspanValue">First</td>
<td> Second </td>
</tr>
<tr>
<td> Third </td>
<td> Fourth </td>
<td> Fifth </td>
</tr>
</table>

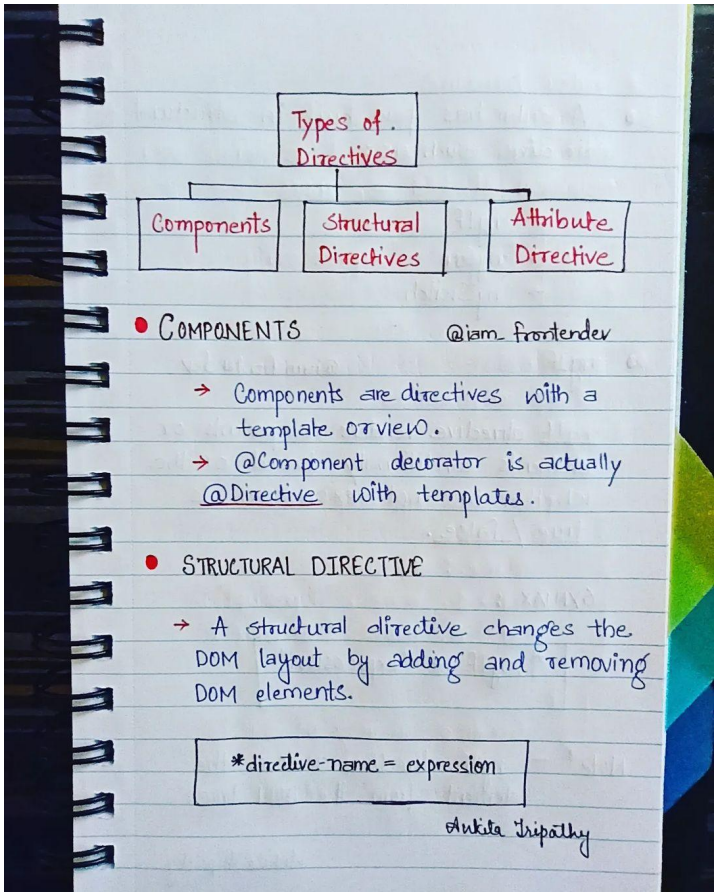

@iamn.frontender
Style and Event Binding
EXAMPLE :-
<button (click)="onSubmit(username.value, password.value)> login </button>
OR
<button on-click="onSubmit(username.value, password.value)"> login </button>
Two-way Data Binding
Two way data binding is a mechanism where if model value changes, it updates the element to which the property is bound and vice versa.
SYNTAX :-
[(ngModel)]


@iam.frontendev
Example.
<input [(ngModel)]="course.coursename">
Behind the scene, this is equivalent to
<input [ngModel]="course.courseName"
(ngModelChange)="course.courseName=$event">
OR
<input bindon-ngModel="course.courseName">
// app.component.ts
...
export class AppComponent {
name: string = "Angular";
}


@iam.frontender.
// app.component.html.
<input type="text" [(ngModel)]="name">
 Hello, {{ name }}
→ Bind name property with text box using ngModel placed in [()] which is a representation of two-way data-binding.
// app.module.ts
form
import { FormsModule } from '@angular/forms'
...
@NgModule({
imports : [
BrowserModule,
FormsModule
],
...
export class AppModule { }

Directive


DIRECTIVE
• Directives are used to change the behavior of components or elements.
• It can be used in the form of HTML Attributes.
• You can create directives using classes attached with @Directive decorator which adds metadata to the class.
Why Directives ?
@iamfrontendev
• It modify the DOM elements.
• It creates reusable and independent code.
• It used to create custom elements to implement the required functionality.
TYPES OF DIRECTIVE
Next Page
Ankita Tripathy



- COMPONENTS @iam_frontendev

  → Components are directives with a template or view.

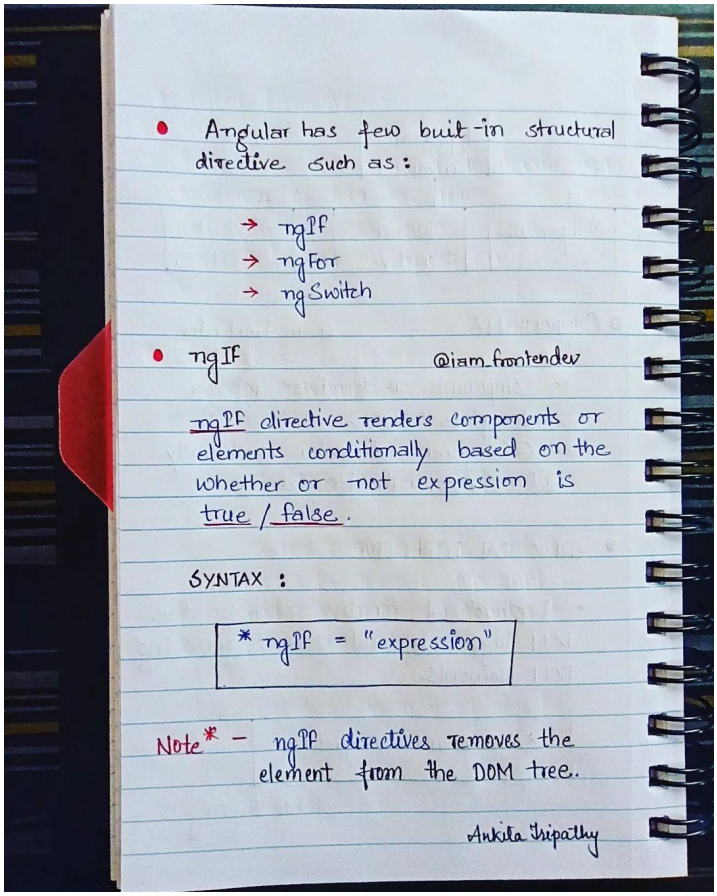
  → @Component decorator is actually @Directive with templates.

- STRUCTURAL DIRECTIVE

  → A structural directive changes the DOM layout by adding and removing DOM elements.

  *directive-name = expression

Ankita Tripathy



- Angular has few built-in structural directive such as:
  - → ngIf
  - → ngFor
  - → ngSwitch

- ngIF @iam_frontendev

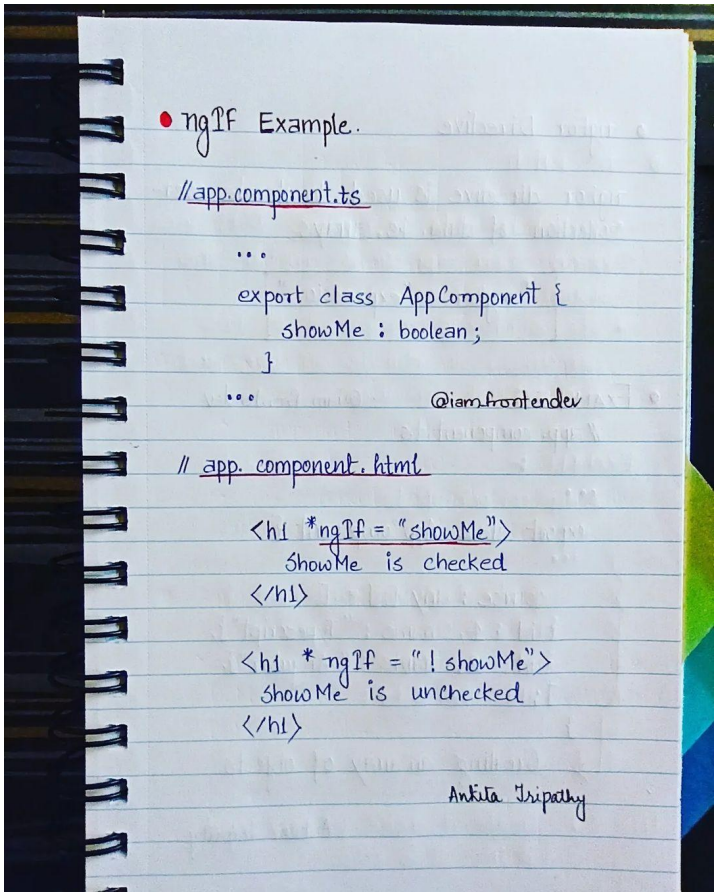
ngIf directive renders components or elements conditionally based on the whether or not expression is true / false.

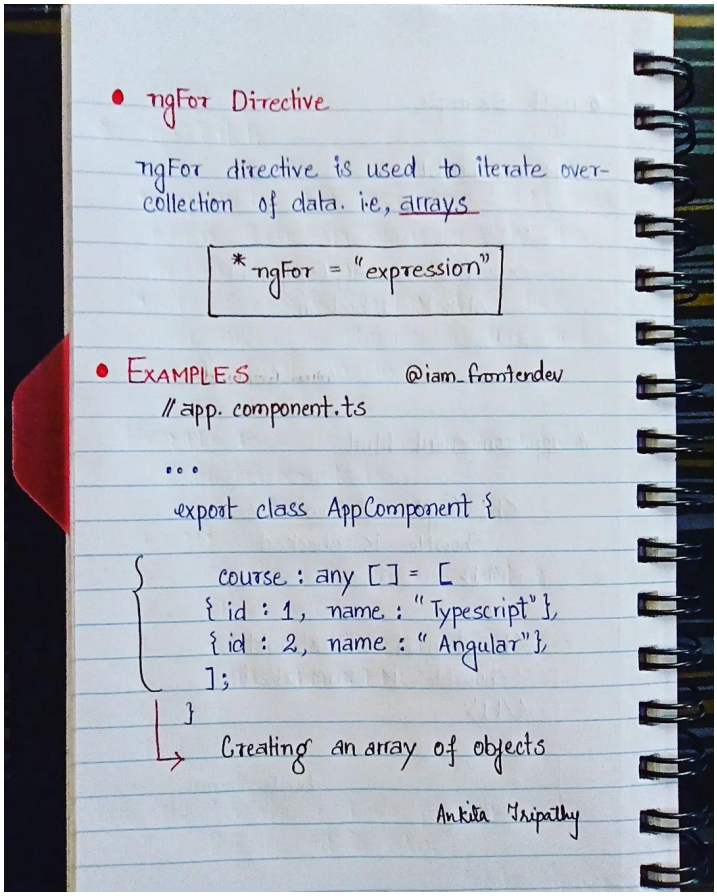
SYNTAX :

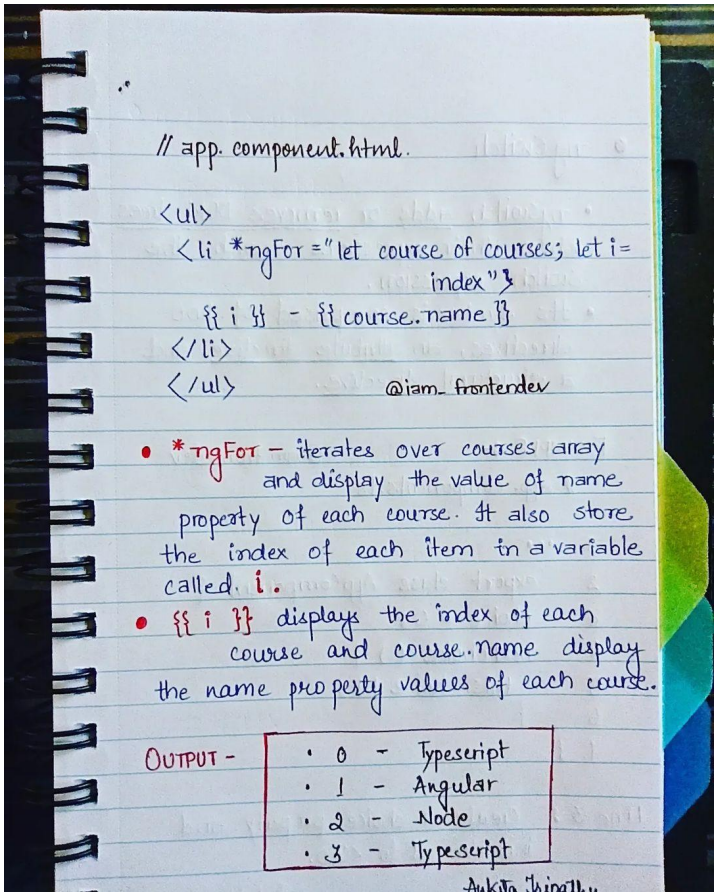
```
* ngIf = "expression"
```

Note* - ngIf directives removes the element from the DOM tree.

Ankita Tripathy


ngIf Example.
//app.component.ts
...
export class AppComponent {
showMe : boolean;
}
...
@iamfrontendev
// app. component. html
<h1 *ngIf = "showMe">
ShowMe is checked
</h1>
<h1 *ngIf = "!showMe">
ShowMe is unchecked
</h1>
Ankita Tripathy


• ngFor Directive
ngFor directive is used to iterate over collection of data. i.e, arrays
*ngFor = "expression"
• EXAMPLES
@iam_frontendev
// app. component.ts
...
export class AppComponent {
course : any [] = [
{ id : 1, name. : "Typescript" },
{ id : 2, name : " Angular" },
];
}
Creating an array of objects
Ankita Tripathy



// app.component.html.

```
<ul>
  <li *ngFor="let course of courses; let i=
                                index">
    {{ i }} - {{course.name }}
  </li>
</ul>
```

@iam_frontendev

- *ngFor – iterates over courses array and display the value of name property of each course. It also store the index of each item in a variable called, i.
- {{ i }} displays the index of each course and course.name display the name property values of each course.

OUTPUT -

- 0 - Typescript
- 1 - Angular
- 2 - Node
- 3 - Typescript

Ankita Thipathy

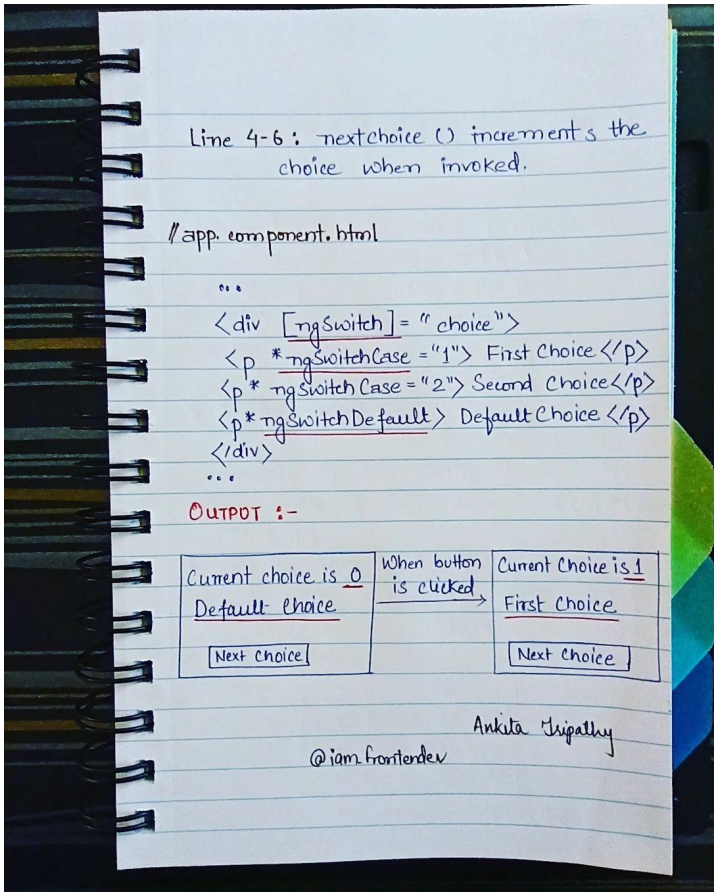
- ng Switch
  - ngSwitch adds or removes DOM trees when their expressions match the switch expression.
  - Its syntax is comprised of two directives, an attribute directive, and a structural directive.

EXAMPLE :- @iam_frontedev

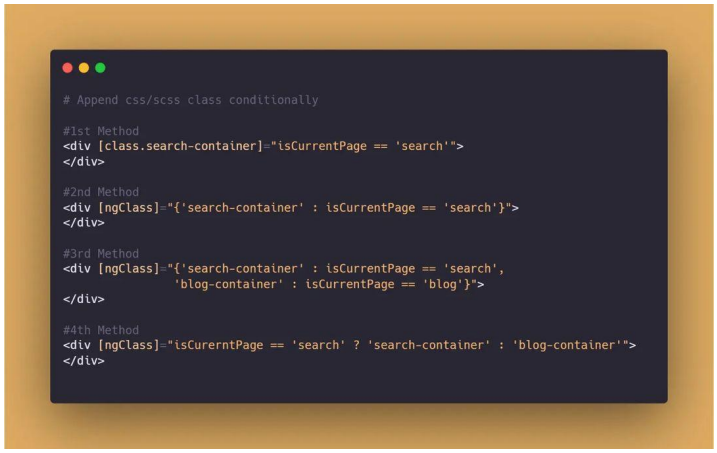
```
// app.components.ts
1  ...
2    export class AppComponent {
3     choice = 0;
4     nextchoice () {
5       this.choice ++;
6     }
7. }
```

Line 3 : Create a choice property and initialize it to zero.

Ankita Tripathy



Ng class attribute directive



```
# Append css/scss class conditionally

#1st Method
<div [class.search-container]="isCurrentPage == 'search'">
</div>

#2nd Method
<div [ngClass]="{'search-container' : isCurrentPage == 'search'}">
</div>

#3rd Method
<div [ngClass]="{'search-container' : isCurrentPage == 'search',
                 'blog-container' : isCurrentPage == 'blog'}">
</div>

#4th Method
<div [ngClass]="isCurerntPage == 'search' ? 'search-container' : 'blog-container'">
</div>
```

Ng style

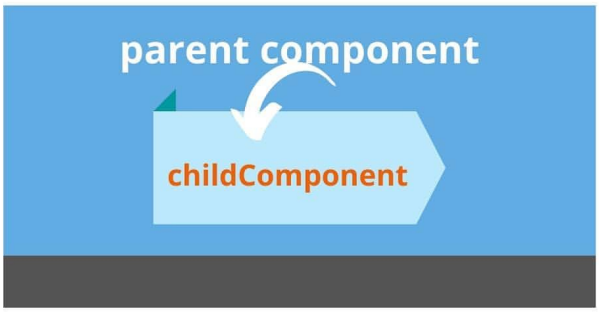
ngStyle

This directive lets you set **dynamically** a given DOM elements style properties. The value in the object that we assign to **ngStyle** can be a **JS expression** which are evaluated and used as value of the css property

Input decorator

The @Input decorator allows to share data with child component from parent component

It receive the data before calling ngOnInIt() method



angular_ui_developer

There are 2 different ways to use Input decorator

1.Using class property
2.Using Alias

There are 2 different ways to Detecting Input Change

1. Using OnChanges life Cycle hook
2. Using Input Setter

angular_ui_developer

@Input decorator
using class property

parent.component.ts

```
parentTitle = "Hello!"
```

parent.component.html

```
<app-child [childTitle]="parentTitle">
<app-child>
```

child.component.ts

```
@Input() childTitle : string
```

child.component.html

```
<p> {{childTitle}}</p>
```

angular_ui_developer

@Input decorator
using Alias

parent.component.html

```
<app-child msg="Welcome!">
<app-child>
```

child.component.ts

```
@Input('msg') childTitle : string
```

child.component.html

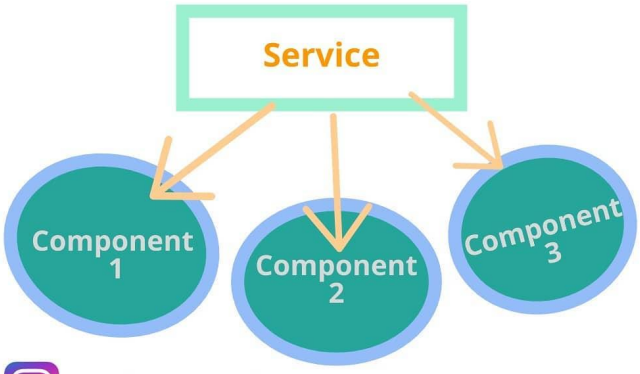
```
<p> {{childTitle}}</p>
```

angular_ui_developer

Service

What is Service

Service is a **reusable typescript class** to share data among the classes that do not know each other.



angular_ui_developer

Why Use Service

A situation where we need to use same property/functions in multiple components. Writing same code in multiple components becomes difficult to manage the code.

To deal this kind of problem,we take help of service class.

By using service we can make testing & debugging simple.

angular_ui_developer

Ways to Register Service

There are 3 ways to register a service

1. Registering a service in a Component

This method should be used when a service in concern, is strickly related to the concerned component and won't be used elsewhere in App.

```
@Component  ({
providers:  [DemoService]
})
```

angular_ui_developer

2. Registering a service in AppModule

This method should be used when you want to have only one instance of your service to be used across the app.
It is called a Singleton Service

```
@NgModel ({
providers: [DemoService]
})
```

Example

Requirement to show toastr message

angular_ui_developer

3. Service Using ProvidedIn Attribute

This method should be used when you want to have only one instance of your service that can be consumed across the app in any Module,Component,Directive,Pipe

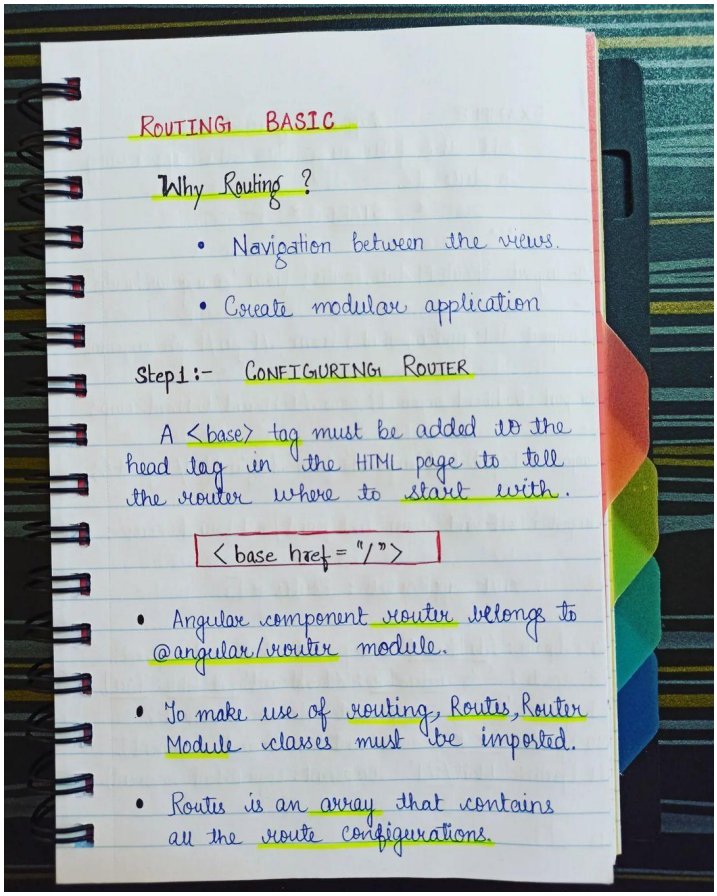
```
@Injectable ({
providedIn: 'root'
})
```

Example

Requirement to use REST API

angular_ui_developer

Routing



ROUTING BASIC

Why Routing ?

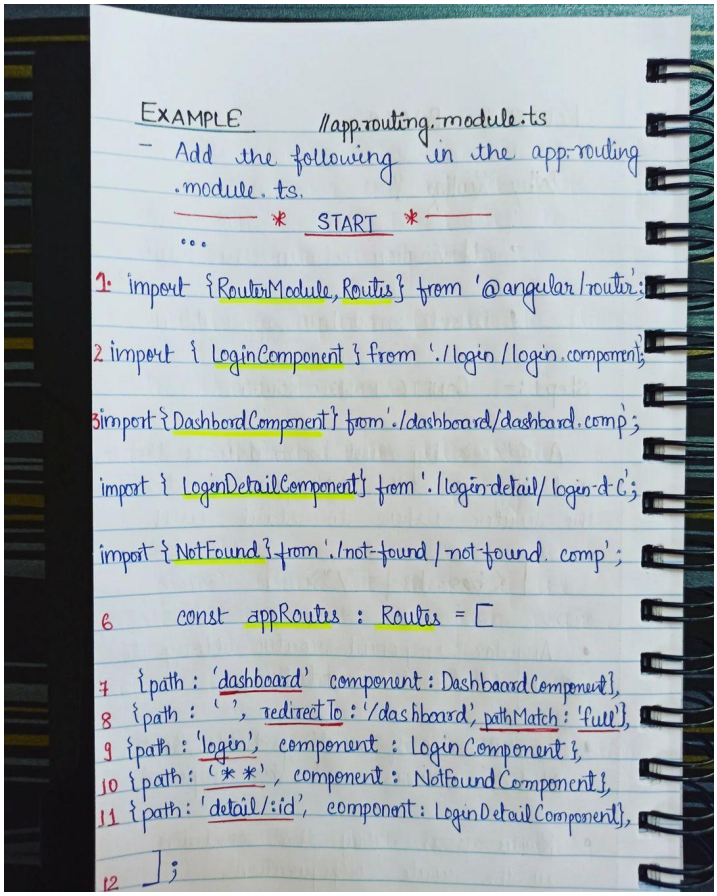
- Navigation between the views.
- Create modular application

Step1:- CONFIGURING ROUTER

A <base> tag must be added to the head tag in the HTML page to tell the router where to start with.

```
<base href="/">
```

- Angular component router belongs to @angular/router module.
- To make use of routing, Routes, RouterModule classes must be imported.
- Routes is an array that contains all the route configurations.



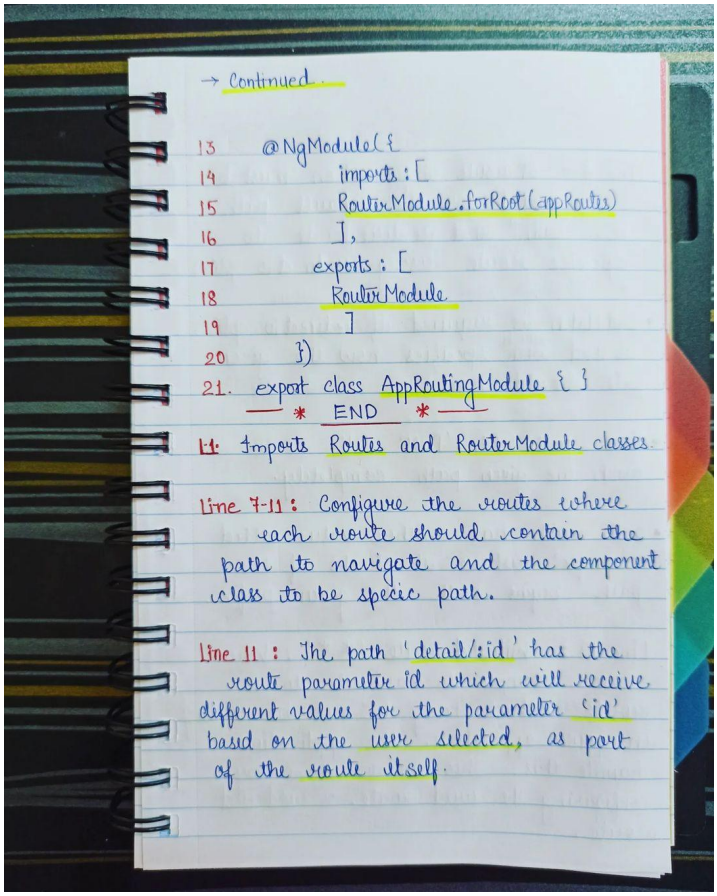
EXAMPLE //app.routing.module.ts

- Add the following in the app-routing .module.ts.

——— * START * ———

...

```
1. import {RouterModule, Routes} from '@angular/router';
2 import { LoginComponent } from './login/login.component';
3import {DashboardComponent} from './dashboard/dashboard.comp';
import { LoginDetailComponent} from './login-detail/login-dtC';
import {NotFound} from './not-found/not-found.comp';
6       const appRoutes : Routes = [
7   {path : 'dashboard' component : DashbaordComponent},
8   {path : '', redirectTo : '/dashboard', pathMatch : 'full'},
9   {path : 'login', component : LoginComponent },
10  {path : '**', component : NotFoundComponent},
11  {path : 'detail/:id', component : LoginDetailComponent},
12  ];
```



→ Continued..

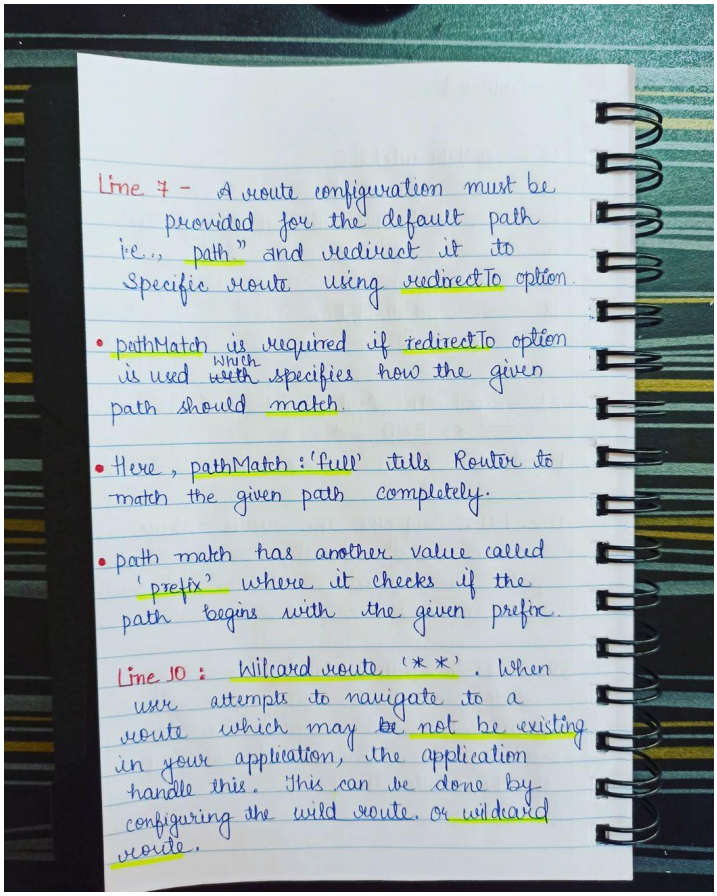
```
13    @NgModule({
14          imports: [
15          RouterModule.forRoot(appRoutes)
16          ],
17          exports: [
18            RouterModule
19            ]
20        })
21. expot class AppRoutingModule { }
```

* END *

1-4: Imports Routes and RouterModule classes.

Line 7-11 : Configure the routes where each route should contain the path to navigate and the component class to be specic path.

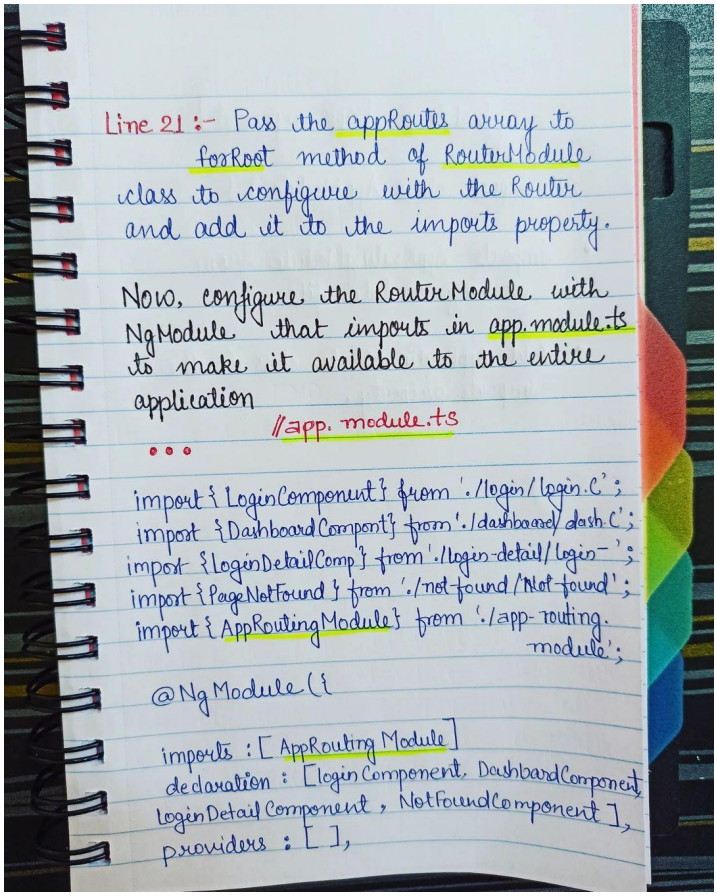
line 11 : The path 'detail/:id' has the route parameter id which will receive different values for the parameter 'id' based on the user selected, as part of the route itself.



Line 7 - A route configuration must be provided for the default path i.e., "path" and redirect it to specific route using redirectTo option.

- pathMatch is required if redirectTo option is used which specifies how the given path should match.
- Here, pathMatch: 'full' tells Router to match the given path completely.
- path match has another value called 'prefix' where it checks if the path begins with the given prefix.

Line 10: Wilcard route '**'. When user attempts to navigate to a route which may not be existing in your application, the application handle this. This can be done by configuring the wild route. or wildard route.



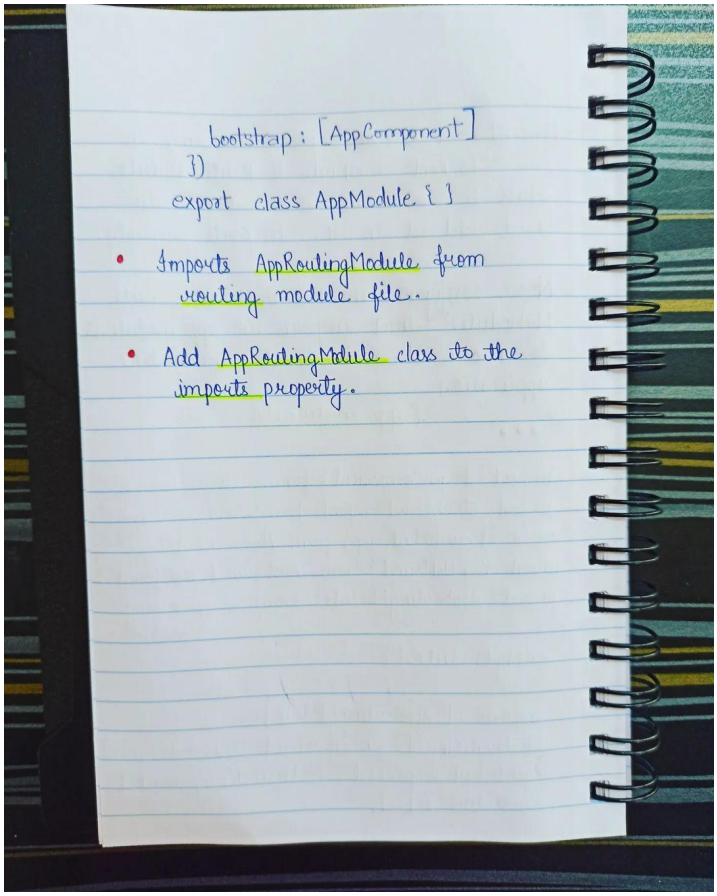
Line 21 :- Pass the appRoutes array to forRoot method of RouterModule class to configure with the Router and add it to the imports property.

Now, configure the RouterModule with NgModule that imports in app.module.ts to make it available to the entire application

//app.module.ts

...

```
import { LoginComponent } from './login/login.C';
import {DashboardCompont} from './dashboard/dash.C';
import {LoginDetailComp} from './login-detail/login-';
import {PageNotFound} from './not-found/Not-found';
import { AppRoutingModule} from './app-routing.
                                module';

@NgModule({

  imports : [ AppRouting Module ]
  declaration : [loginComponent, DashbardComponent,
  LoginDetailComponent , NotFoundComponent ],
  providers : [ ],
```

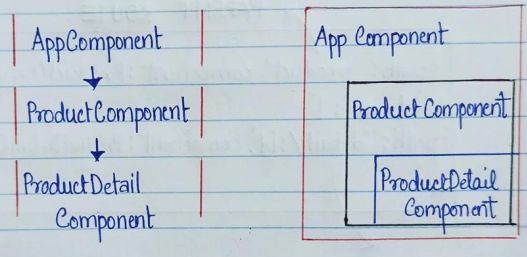


Nested child routes

NESTED ROUTES IN ANGULAR

Consider the following Component tree

The Component Tree



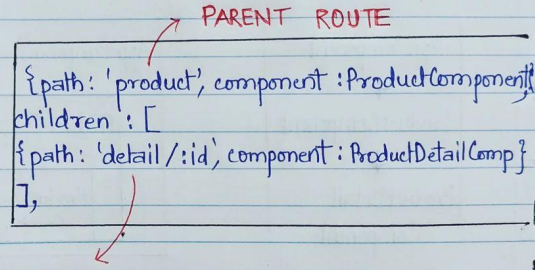
STEP : 1 Define the routes (//app.routing.ts)

```
{path : 'product', component : ProductComponent },
{path : 'product/:id', component : ProductDetailComp },
```

Here, the ProductDetailComponent is defined as the sibling of the Product Component and not as the child.

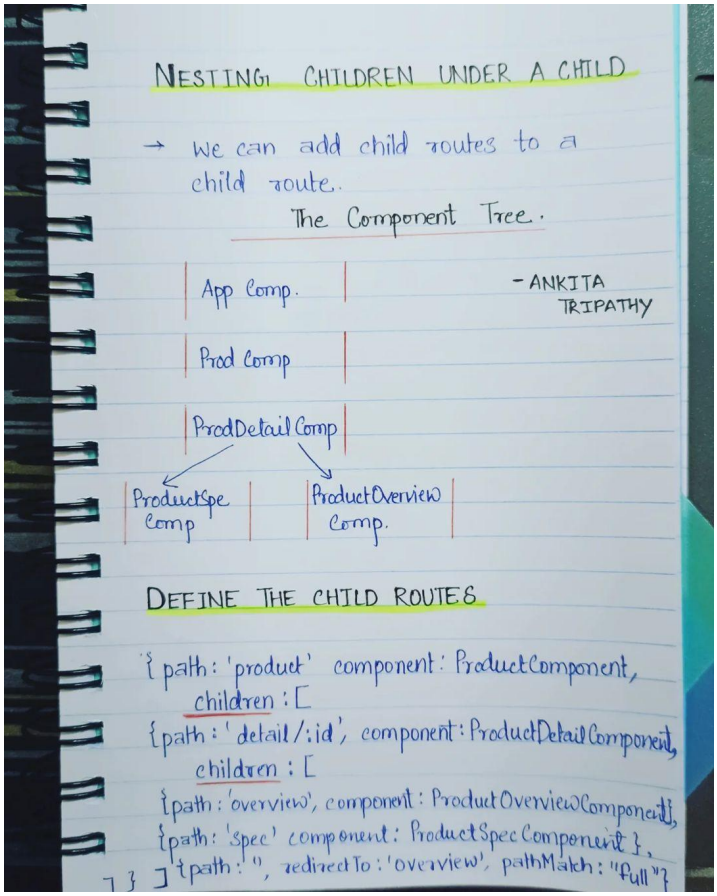
— ANKITA TRIPATHY

STEP:2 To make ProductDetail Comp as the child of the Product Component, We need to add the children key to the product route, which is an array of all child routes.

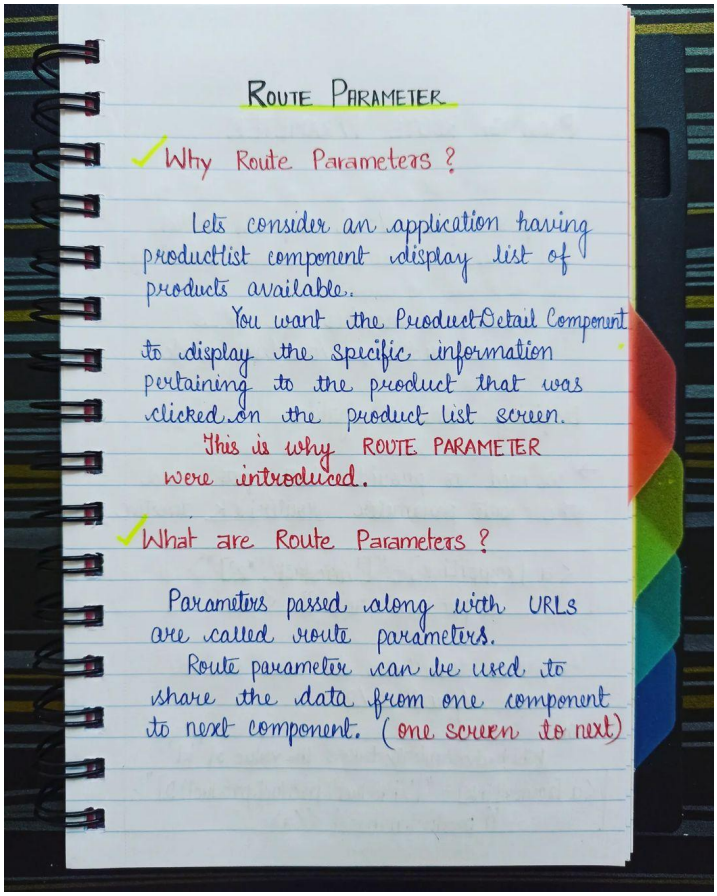


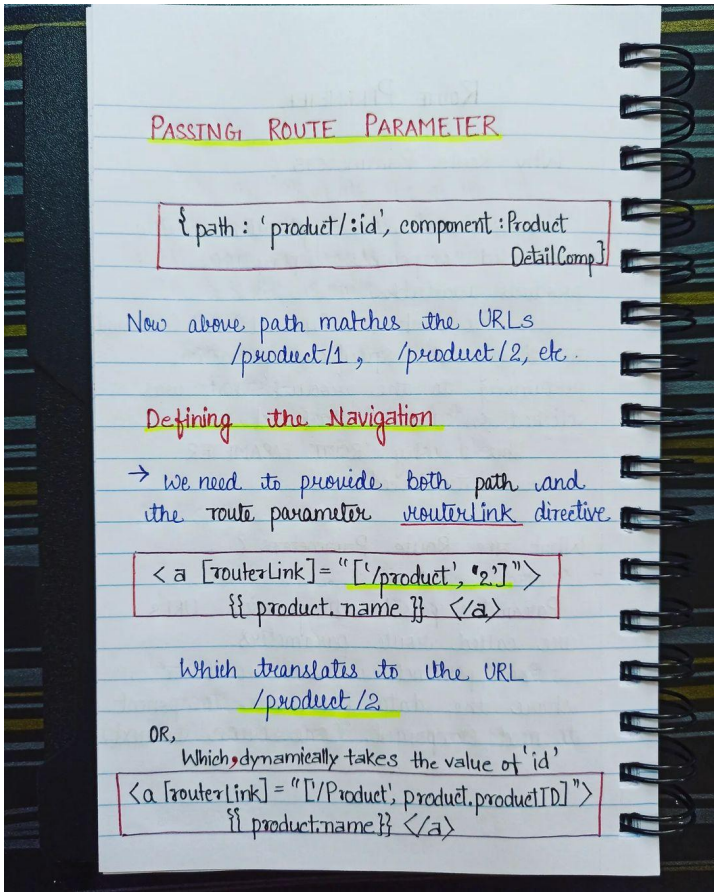
CHILD ROUTE → This will match the URL path "/product/detail/id".

- It start off the first URL segment that is 'product' and finds the match in the path 'product' and instantiates the Product Component.



Route parameters


ROUTE PARAMETER
Why Route Parameters ?
Lets consider an application having productlist component display list of products available.
You want the ProductDetail Component to display the specific information pertaining to the product that was clicked on the product list screen.
This is why ROUTE PARAMETER were introduced.
What are Route Parameters ?
Parameters passed along with URLs are called route parameters.
Route parameter can be used to share the data from one component to next component. (one screen to next)



PASSING ROUTE PARAMETER

```
{path: 'product/:id', component: ProductDetailComp}
```

Now above path matches the URLs
/product/1 , /product/2, etc.

Defining the Navigation

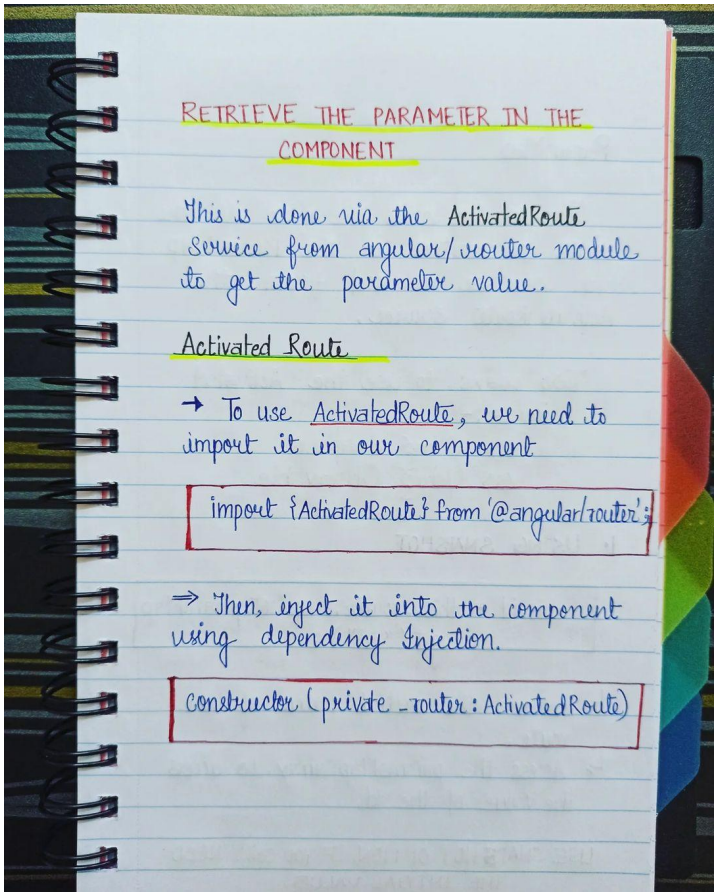
→ We need to provide both path and the route parameter routerLink directive.

```
<a [routerLink]="['/product', '2']">
    {{ product.name }} </a>
```

Which translates to the URL
/product/2

OR,
Which, dynamically takes the value of 'id'

```
<a [routerLink]="['/Product', product.productID]">
    {{ product.name }} </a>
```



RETRIEVE THE PARAMETER IN THE COMPONENT

This is done via the ActivatedRoute Service from angular/router module to get the parameter value.

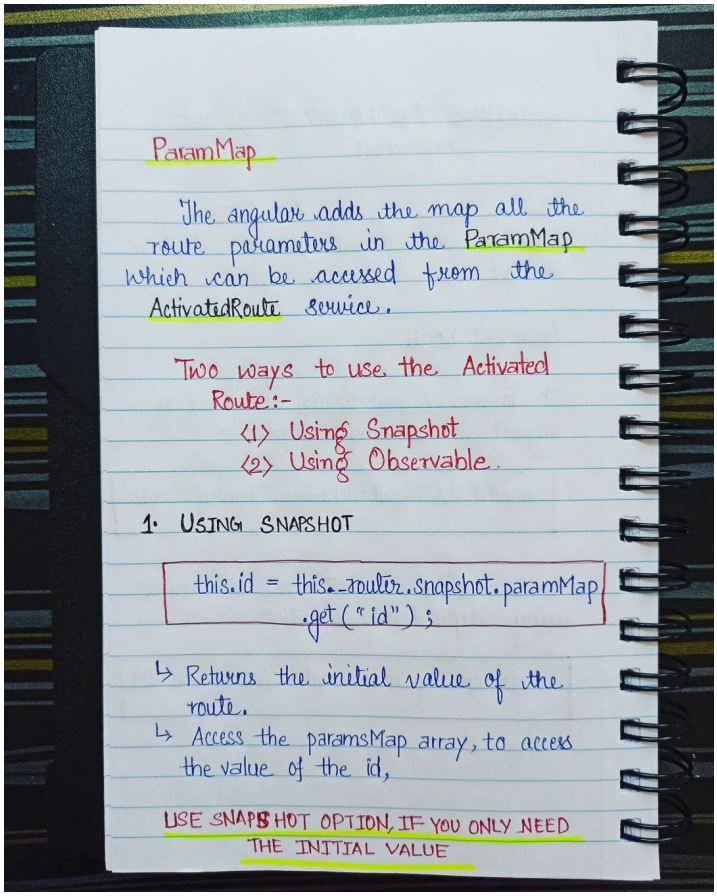
Activated Route

→ To use ActivatedRoute, we need to import it in our component

```
import {ActivatedRoute} from '@angular/router';
```

⇒ Then, inject it into the component using dependency Injection.

```
constructor (private _router : ActivatedRoute)
```



ParamMap

The angular adds the map all the route parameters in the ParamMap which can be accessed from the ActivatedRoute service.

Two ways to use the Activated Route:-

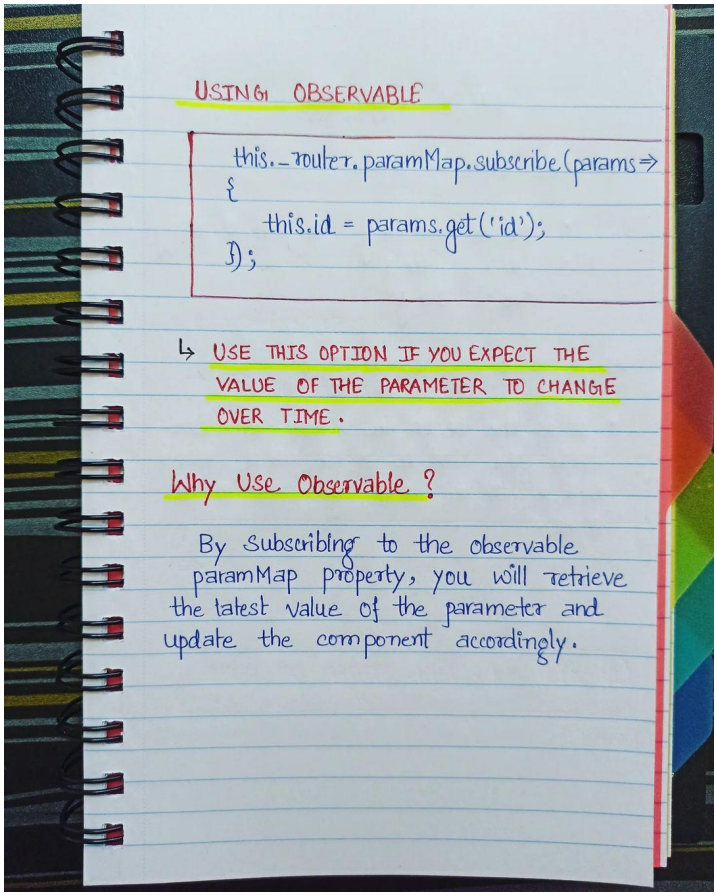
<1> Using Snapshot
<2> Using Observable.

1. USING SNAPSHOT

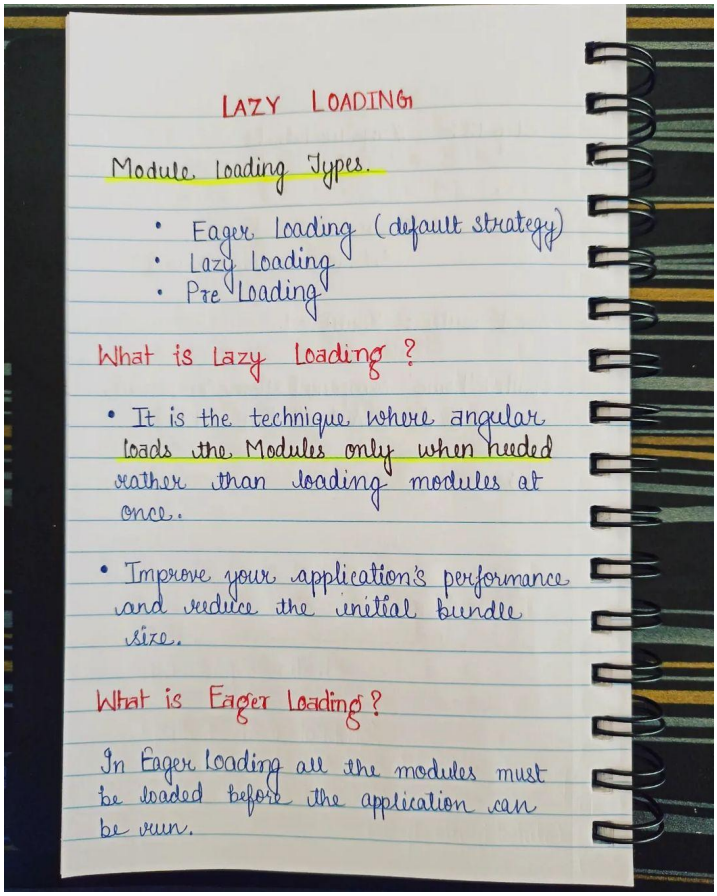
```
this.id = this._router.snapshot.paramMap
          .get("id");
```

↳ Returns the initial value of the route.
↳ Access the paramsMap array, to access the value of the id,

USE SNAPSHOT OPTION, IF YOU ONLY NEED THE INITIAL VALUE



Lazy loading



LAZY LOADING

Module Loading Types.

- Eager Loading (default strategy)
- Lazy Loading
- Pre Loading

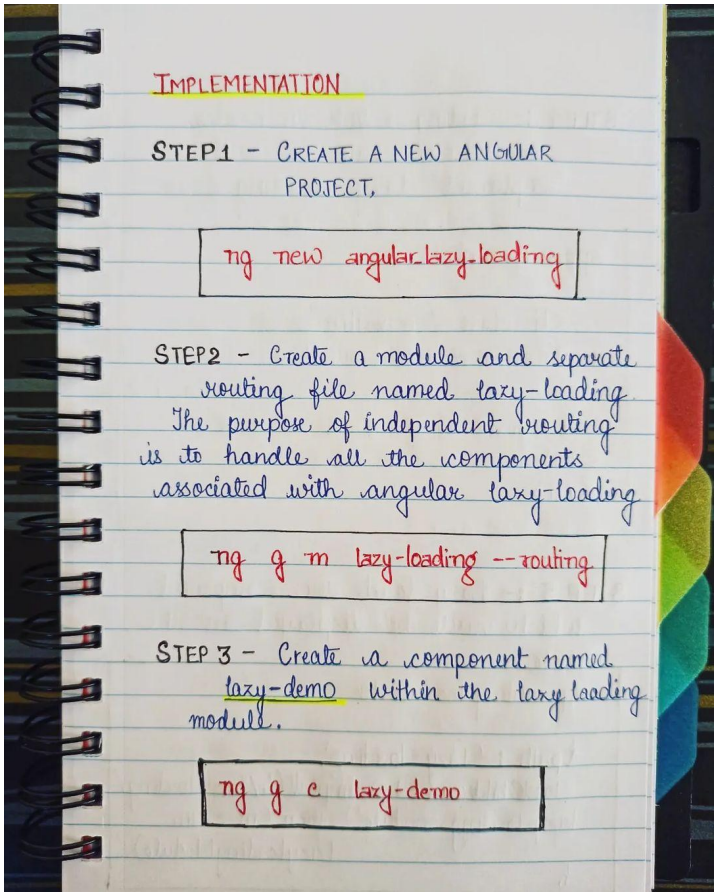
What is Lazy Loading ?

- It is the technique where angular loads the Modules only when needed rather than loading modules at once.

- Improve your application's performance and reduce the initial bundle size.

What is Eager Loading ?

In Eager Loading all the modules must be loaded before the application can be run.



IMPLEMENTATION

STEP 1 - CREATE A NEW ANGULAR PROJECT.

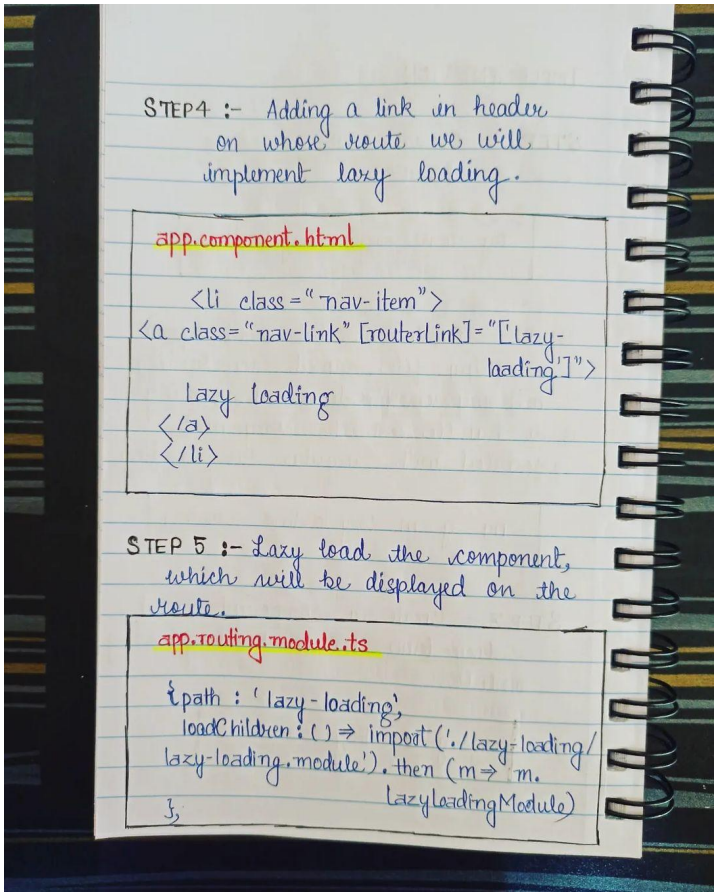
```
ng new angular_lazy-loading
```

STEP 2 - Create a module and separate routing file named lazy-loading The purpose of independent routing is to handle all the components associated with angular lazy-loading

```
ng g m lazy-loading --routing
```

STEP 3 - Create a component named lazy-demo within the lazy loading module.

```
ng g c lazy-demo
```



STEP 4 :- Adding a link in header on whose route we will implement lazy loading.

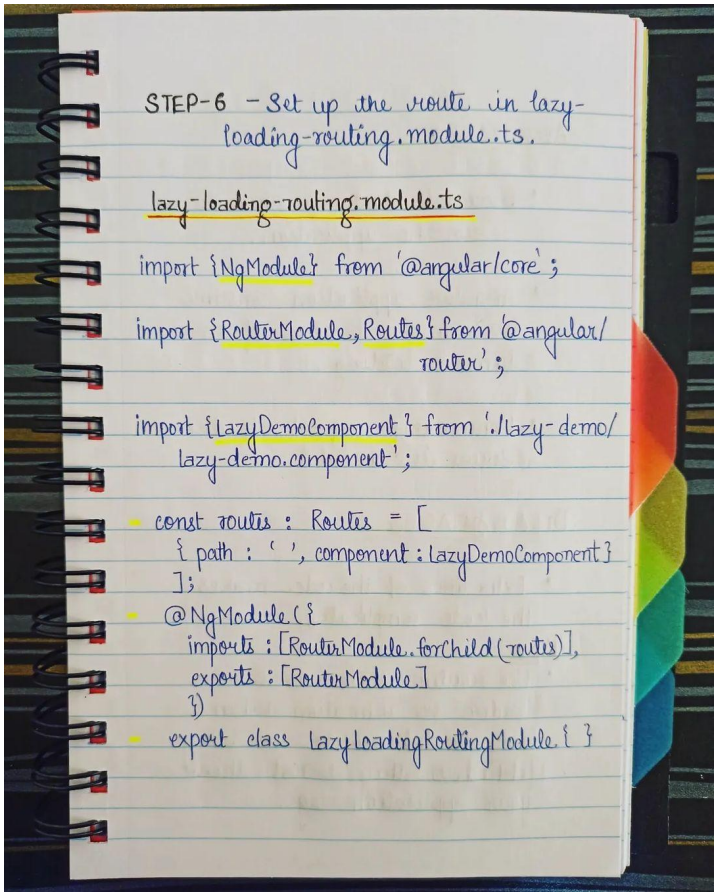
app.component.html

```
<li class="nav-item">
<a class="nav-link" [routerLink]="['lazy-
                                  loading']">
  Lazy loading
</a>
</li>
```

STEP 5 :- Lazy load the component, which will be displayed on the route.

app.routing.module.ts

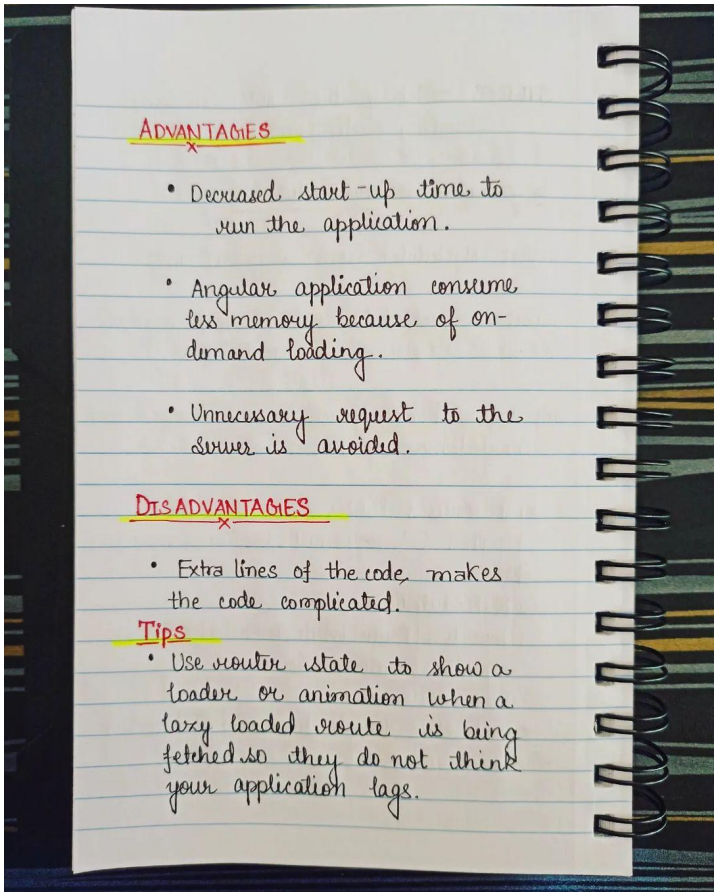
```
{path : 'lazy-loading',
 loadChildren : () => import('./lazy-loading/
lazy-loading.module').then(m => m.
                        LazyLoadingModule)
},
```



STEP-6 - Set up the route in lazy-loading-routing.module.ts.

lazy-loading-routing.module.ts

```
import {NgModule} from '@angular/core';

import {RouterModule, Routes} from '@angular/router';

import {LazyDemoComponent} from './lazy-demo/lazy-demo.component';

const routes : Routes = [
  { path : '', component : LazyDemoComponent}
];
@NgModule({
  imports : [RouterModule.forChild(routes)],
  exports : [RouterModule]
})
export class LazyLoadingRoutingModule { }
```



ADVANTAGES

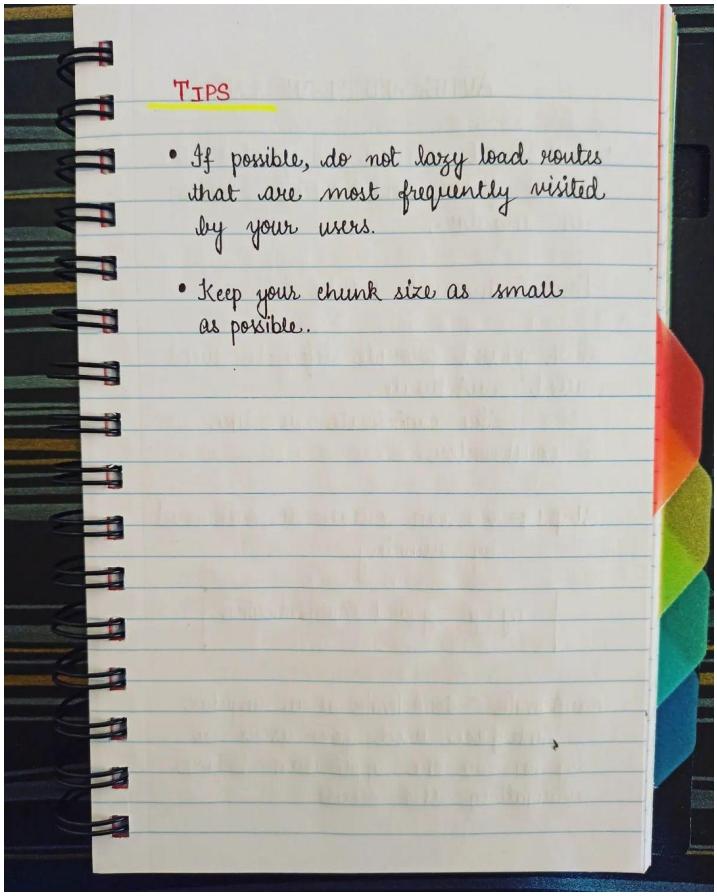
- Decreased start-up time to run the application.
- Angular application consume less memory because of on-demand loading.
- Unnecessary request to the server is avoided.

DISADVANTAGES

- Extra lines of the code makes the code complicated.

Tips

- Use router state to show a loader or animation when a lazy loaded route is being fetched so they do not think your application lags.


TIPS
• If possible, do not lazy load routes that are most frequently visited by your users.
• Keep your chunk size as small as possible.

Auth guard

AUTHGUARD IN ANGULAR.

- AuthGuard is used to protect the routes from unauthorized access in angular.

How AuthGuard works ?

Auth guard provide lifecycle event called canActivate.

The canActivate is like a constructor.

Step 1 :- we can create an AuthGuard by running.

```
ng g guard Authentication
```

canActivate - CanActivate is an angular interface used for users to log in to the application before navigating the route.

Step2 :- Choose the canActivate Inteface.

Step 3 :- Create service using the command.

```
ng g s AuthguardService
```

- The main advantage of service one is that the backlned value shares multiple components.

Step4 : Once you create the service file, you can add app.module.ts with the following code.

```
// app.module.ts
...
- import {AuthguardService} from './authguard-service.service'
...
- providers : [
    AuthguardService
  ],
```

Step 5 - Routing

```
import {NgModule} from '@angular/core';

import {Routes, RouterModule} from '@angular/router';
```

→ Routing makes your application as SPA. That way we can use Routing in our application.

→ RouterModule - Provide required services and directives to use routing and navigation in angular application.

→ Routes - Defines an array of roots that map a path to a component.

Using RouterModule.forRoot()

```
@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})
```

Step 7 :- Let's create the component and add the routing.ts file.

//app-routing.module.ts

```
...
import {LoginComponent} from './login.comp';
import {HomeComponent} from './home.component';

const routes : Routes = [
{path: '', redirectTo: '/login', pathMatch: 'full'},
{path: 'login', component: LoginComponent},
{path: 'home', component: Home.Component}
];
...
```

Step 8 :- Open the authentication.guard.ts file and change the code.

```
export class AuthenticationGuard implements CanActivate{
  canActivate (): boolean
   {
    return true;
   }
}
```

Represents value in two states : true or false.

Step 9: Store the user session detail in your login page and call the ngOnInit,

```
ngOnInit() {
  localStorage.setItem('SessionUser', this.user)
}
```

Step 10: Open AuthguardService.ts file get your local Storage data.

```
import {Injectable} from '@angular/core';
@Injectable({
    providedIn: 'root'
})
export class AuthguardService {
  constructor() {
  }
    gettoken() {
      return !localStorage.getItem("SessionUser");
    }
}
```

LOCAL STORAGE

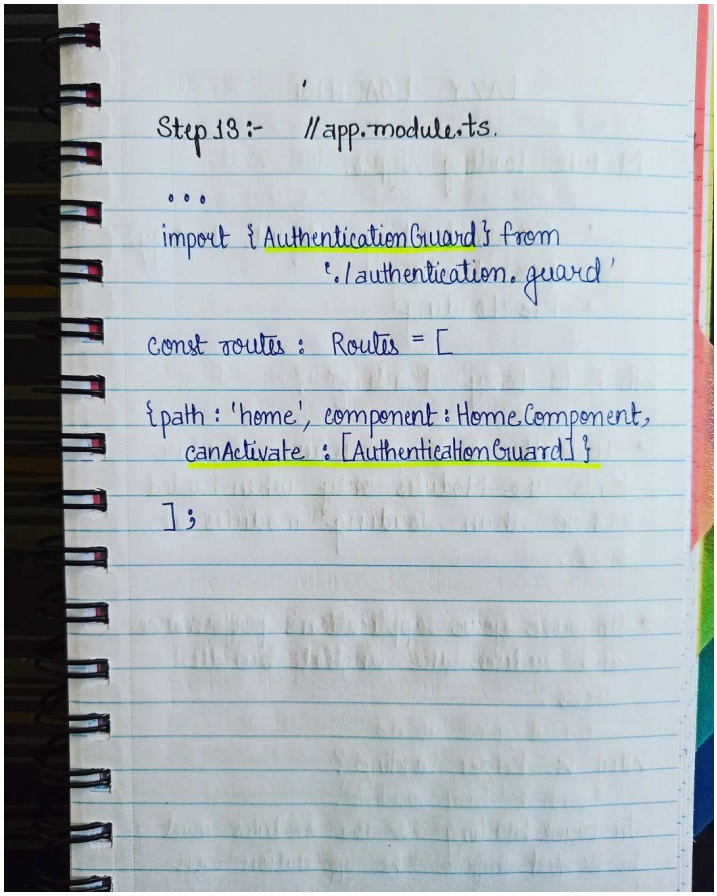
- Permanent Storage
- No expire date unless you remove it
- Domain - Specific

Step 11 :- // Authguard

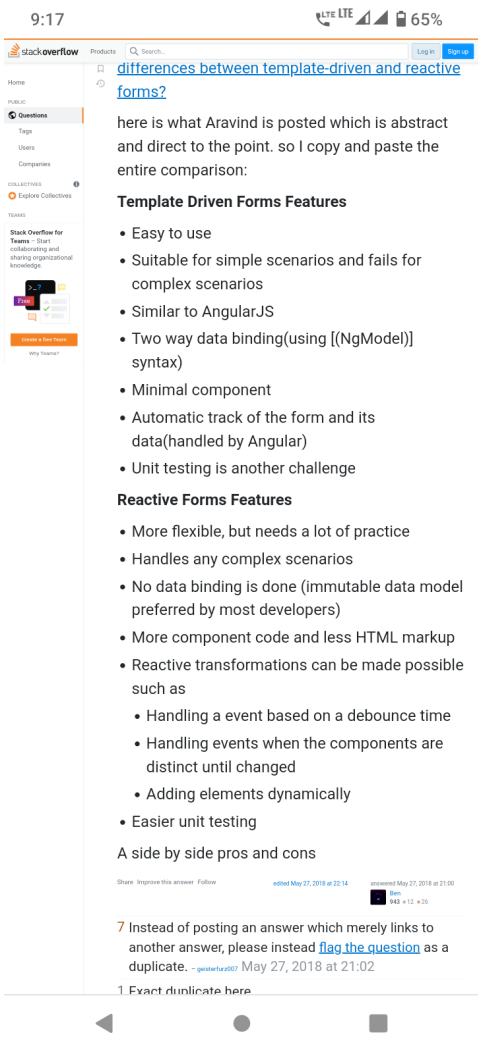
```
constructor ( private Authguardservice : Authguard
    Service , private router : Router ) { }

  canActivate ( ): boolean {
  if ( ! this. Authguardservice. gettoken ( )) {
      this.router.navigateByUrl("/login");
    }
   return this. Authguardservice. gettoken ( );
  }
```

Step 12: // app. component. html.

<router-outlet> </router-outlet>


Step 13 :- //app.module.ts.
...
import { AuthenticationGuard } from
'./authentication.guard'
const routes : Routes = [
{path : 'home', component : HomeComponent,
canActivate : [AuthenticationGuard] }
];

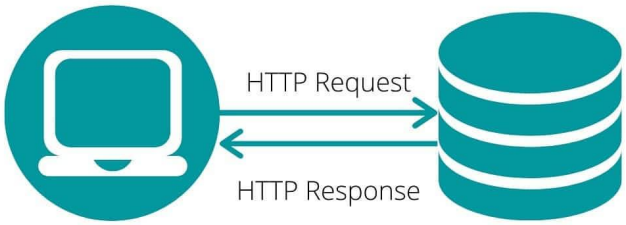
Form



Http methods

What is HTTP Request

The HttpClient is used to perform HTTP Requests .
HTTP Request is a packet of information which is transferred from source to destination.
A Client make an HTTP Request and server sends the information to the client



angular_ui_developer

HTTP Request Methods

Before writing the methods to make HTTP Requests, we need to make the variable of httpClient in Service

constructor(private http: httpClient)

It is the best practice to divide the whole url in to two parts.
url: "https://jsonplaceholder.typicode.com/users

1.main url
"https://jsonplaceholder.typicode.com

2. endpoints
users

angular_ui_developer

1. GET Methods

GET Method fetchs the data from the service and give an array of object.
It take only one parameter i.e backend service url: "https://jsonplaceholder.typicode.com/user

GET Method will provide the data in JSON formate that we need to convert in typescript code .
By using httpClient it convert the data from JSON to an Observable form.

To extract the data from Observable we need to use subscribe method of rxjs


angular_ui_developer

a). Get All Users

```
public getAllUser()  {
this.http.get(this.url).subscribe(data =>{
console.log(data);
});
}
```

b). Get User By Id

```
public getUserById()  {
let id: number =2;
endpoints= "/users/ "+ id;
this.http.get(this.url +
endpoints).subscribe(data =>{
console.log(data);
});
}
```


angular_ui_developer

2. POST Method

The POST Method is used for sending data to the server.It takes two parameter the service url and the request body

EX. Post New User

```
public addUser(userData :object)   {
    endpoints = "/users/ " ;
this.http.post(this.url + endpoints
,userData).subscribe(data =>{
    console.log(data);
    });
    }
```

angular_ui_developer

3. PUT Method

The PUT Method is used for updating an Object which is already saved in database.
For updating the object ,we need to pass the object Id in the URL

EX. Update the User

```
public updateUser(userData :object)  {
endpoints = "/users/3 " ;
this.http.put(this.url + endpoints , ,
,userData).subscribe(data => {
console.log(data);
});
}
```


angular_ui_developer

4. DELETE Method

DELETE Method only require the url which has the ID of the Object.
It check the ID and delete the data from the Database

EX. Delete an User

```
public deleteUser()  {
endpoints = "/users/2 " ;
this.http.put(this.url + endpoints
).subscribe(data => {
console.log(data);
});
}
```


angular_ui_developer

Link

https://interviewant.com/angular-interview-questions