

Cours 3 : Langage et intro à



Applications web et mobiles

Christophe Gonzales

Javascript

Exécution de javascript

- ▶ Browsers contiennent un moteur Javascript :



Chakra



Nitro

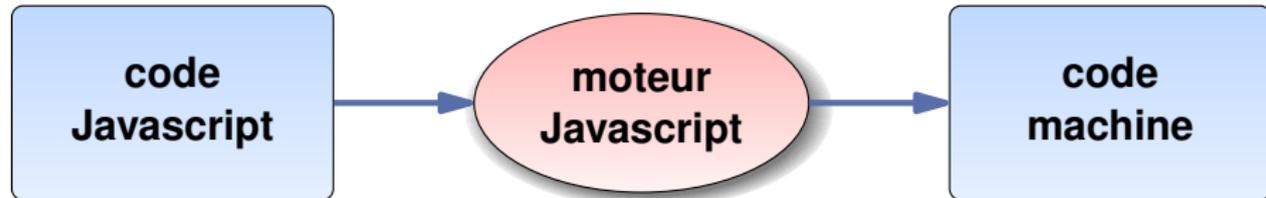


SpiderMonkey



V8

- ▶ Vérifient les normes ECMAScript (ES7 = ES2016) et la plupart sont compatibles avec WebAssembly
- ▶ Principe de fonctionnement :



- ▶ Browser exécute ce code machine
- ▶ Depuis 2009 : compil/exec hors browser :



HTML et javascript

The image shows a development environment with two windows. The left window is the WebStorm IDE, displaying the source code of a file named 'page1.html'. The code is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>intro à javascript</title>
6 </head>
7 <body>
8   <h1>contenu de la page web</h1>
9
10  <!-- insertion d'un code javascript -->
11  <script>
12    console.log('toto');
13  </script>
14 </body>
15 </html>
16
```

The right window is a Chromium browser titled 'intro à javascript'. The address bar shows the file path: '/home/gonzales/enseignement...'. The main content area displays a large heading: 'contenu de la page web'. The browser's developer tools are open, with the 'Console' tab selected, showing a log entry: 'toto'.

At the bottom of the WebStorm IDE, there is a status bar with the following information: 'WebStorm 2019.3.2 available: // Update... (55 minutes ago)', '16:1 LF UTF-8 4 spaces', 'Git: master', and 'Event Log'.

Inclusion d'un fichier javascript

The image shows a development environment with two windows. The left window is a code editor showing the HTML file `page1.html`. The code is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>intro à javascript</title>
6 </head>
7 <body>
8   <h1>contenu de la page web</h1>
9
10  <!-- insertion d'un code javascript -->
11  <script src="page1.js"></script>
12 </body>
13 </html>
14
```

The right window is a browser (Chromium) displaying the page. The title is "intro à javascript" and the main content is a large heading: "contenu de la page web". The browser's developer tools are open, showing the Console tab with the output: `toto`. Below the code editor, the `page1.js` file is open, containing the following code:

```
1 console.log('toto');
2
```

At the bottom of the IDE, there is a status bar with the following information: "WebStorm 2019.3.2 available: // Update... (1 hour ago)", "14:1 LF UTF-8 4 spaces", "Git: master", and "Event Log".

Création de variables en javascript : 4 manières



✓ **let** nom_variable = valeur;

const nom_variable = valeur;

portée de bloc si déclaration dans un bloc
portée de fichier/module sinon

~ **global**.nom_variable = valeur;

~ **window**.nom_variable = valeur;

création/modification d'une variable globale

✗ nom_variable = valeur;

modification de la valeur d'une variable si elle existe déjà
création d'une variable globale sinon

✗ **var** nom_variable = valeur;

portée de fonction si déclaration dans une fonction
variable globale si déclaration hors fonction



Types primitifs :

- ▶ string : `let x = 'toto', y = 'to' + 'to';`
- ▶ number : `let x = 30, y = 4.5;`
- ▶ boolean : `let x = true;`
- ▶ undefined : `let x, y = undefined;`
- ▶ null : `let x = null;`

Types primitifs \implies copie par valeur :

```
let x = 3;
```



```
let y = x;
```



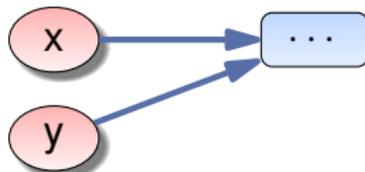
Types référence :

- ▶ Object
- ▶ Array
- ▶ Function

Types référence \implies copie par référence :

```
let x = ...;
```

```
let y = x;
```



▶ **Noms autorisés** : mêmes règles qu'en Java ou C

▶ **Convention** : notation Camel

Exemple : `firstName`

▶ **Attention** : Javascript sensible à la casse :

Exemple : `firstName` \neq `FirstName`

Les fonctions

The screenshot shows a code editor with the following content:

```
1 // les fonctions : fonction nom_fonc (arguments) { code }, comme en php
2 function f(arg1, arg2) {
3     arg1++;
4     console.log(arg1, arg2);
5 }
6
7 // passage de paramètres :
8 // par référence pour les types référence, copie par valeur pour les types primitifs
9 let x = 3;
10 f(x,x); console.log(x);
11 f(x); console.log(x);
12
```

The terminal output shows the execution of the code:

```
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<3> node function.js
4 3
3
4 undefined
3
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<4>
```

Les objets (1/3)

The screenshot shows the WebStorm IDE with the following content:

```
1 // objets = {}
2 // création d'un objet :
3 let guy = {
4     prenom : 'guy', // couples cle-valeur
5     nom : 'nipigue' // séparation des champs
6                 // par des ','
7 };
8
9 console.log(guy);
10
11 // accès aux champs : 2 possibilités :
12 // notation '.' ou '[]'
13 console.log(guy.nom);
14 console.log(guy['prenom']);
15 let field = 'prenom';
16 console.log(guy[field]);
17
```

The terminal window shows the execution of the script:

```
[shalmaneser:~/enseignement/mobile-19-20/prog/j
avascript]<8> node objects.js
{ prenom: 'guy', nom: 'nipigue' }
nipigue
guy
guy
[shalmaneser:~/enseignement/mobile-19-20/prog/j
avascript]<9> 
```

Les objets (2/3)

The screenshot shows the WebStorm IDE interface. The main editor window displays the file `cours2.js` with the following JavaScript code:

```
1 // creation d'un objet avec une méthode
2 let obj = {
3   champ1 : 3,
4   champ2 : 5,
5   // définition d'une méthode avant ES6
6   display1 : function () {
7     console.log(this.champ1);
8   },
9
10  // définition possible depuis ES6
11  display2 () {
12    console.log(this.champ2);
13  }
14 };
15
16 obj.display1();
17 obj.display2();
18
```

The terminal window on the right shows the execution of the command `node cours2.js`, which outputs the values `3` and `5` on separate lines.

Les objets (3/3)

The screenshot shows the Visual Studio Code editor with a file named `obj_class.js` open. The code defines a class `MonObjet` with a constructor and a `display` method. The terminal shows the command `node obj_class.js` being executed, resulting in the output `3 4`.

```
1 // création d'un objet par classe : depuis ES6
2 class MonObjet {
3     // on définit explicitement une méthode constructeur
4     // on ne peut définir qu'un seul constructeur
5     constructor (val1, val2) {
6         // déclaration, initialisation des champs
7         this.champ1 = val1;
8         this.champ2 = val2;
9     }
10
11     // pas besoin de mot clef "function" pour les
12     // méthodes
13     display () {
14         console.log(this.champ1, this.champ2);
15     }
16 }
17
18 let obj = new MonObjet(3,4);
19 obj.display ();
20
```

Terminal: Local x +

```
OFF [shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<1> node obj_class.js
3 4
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<2> |
```

Comparaison types primitifs / référence

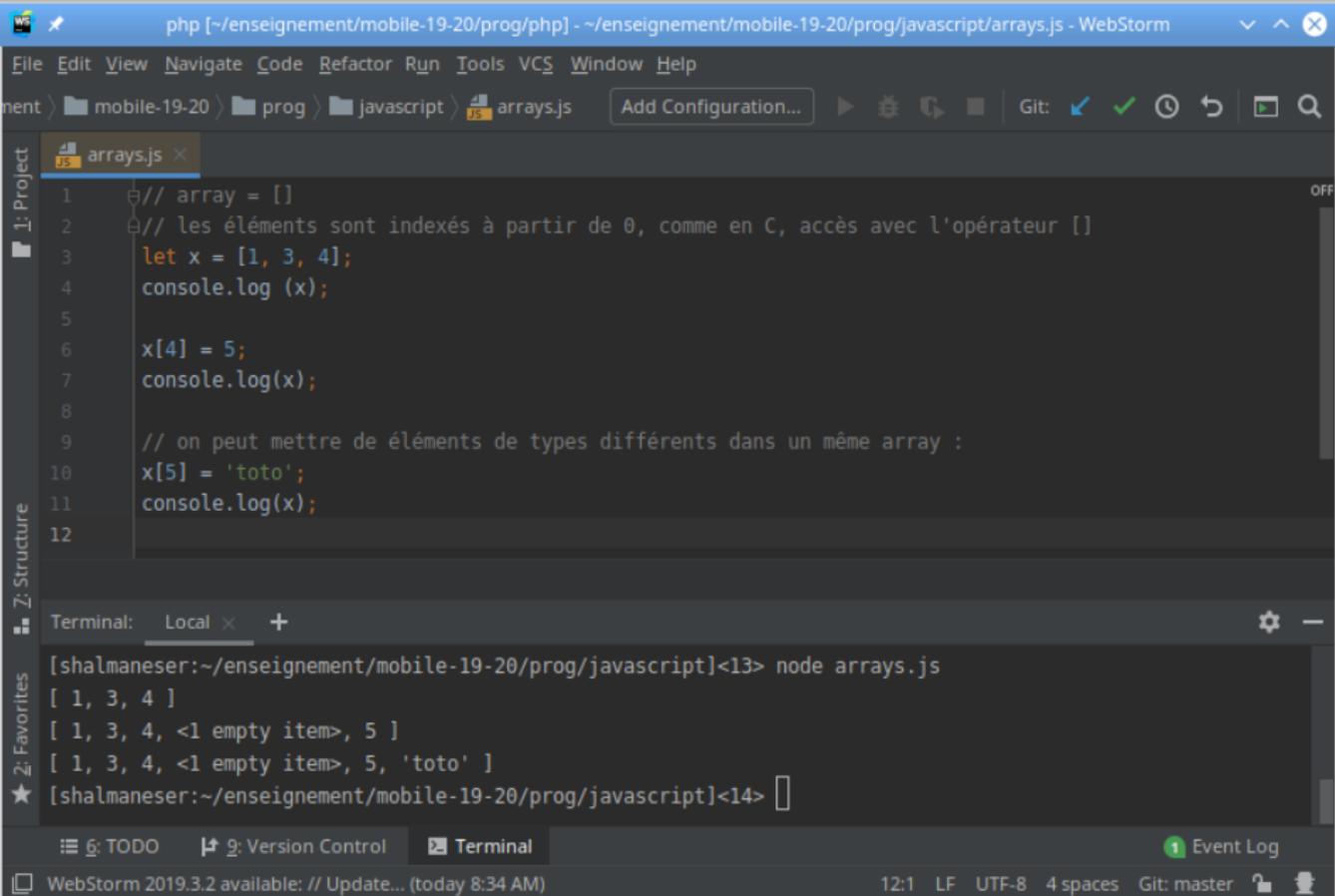
The screenshot shows a code editor with the following content:

```
1 // copie de types primitifs : x != y
2 let x = 4;
3 let y = x;
4 y++;
5 console.log (x,y);
6
7 // copie de types référence ; x = y
8 let a = { val : 4 };
9 let b = a;
10 b.val = 5;
11 console.log (a,b);
12
```

The terminal output shows the execution of the code:

```
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<12> node value_vs_ref.js
4 5
{ val: 5 } { val: 5 }
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<13> 
```

Les tableaux (1/2)



The screenshot shows the WebStorm IDE with the following content:

```
php [~/enseignement/mobile-19-20/prog/php] - ~/enseignement/mobile-19-20/prog/javascript/arrays.js - WebStorm
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help

mobile-19-20 > prog > javascript > arrays.js Add Configuration...

arrays.js

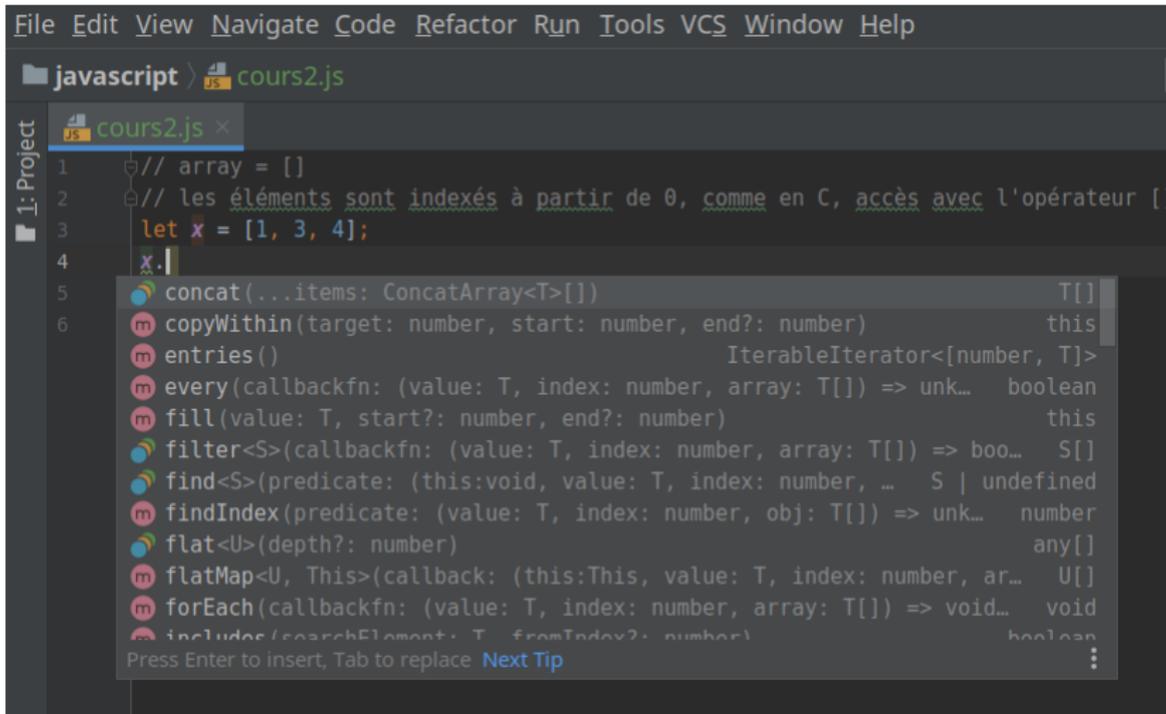
```
1 // array = []
2 // les éléments sont indexés à partir de 0, comme en C, accès avec l'opérateur []
3 let x = [1, 3, 4];
4 console.log(x);
5
6 x[4] = 5;
7 console.log(x);
8
9 // on peut mettre de éléments de types différents dans un même array :
10 x[5] = 'toto';
11 console.log(x);
12
```

Terminal: Local x +

```
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<13> node arrays.js
[ 1, 3, 4 ]
[ 1, 3, 4, <1 empty item>, 5 ]
[ 1, 3, 4, <1 empty item>, 5, 'toto' ]
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<14> █
```

WebStorm 2019.3.2 available: // Update... (today 8:34 AM) 12:1 LF UTF-8 4 spaces Git: master

Les tableaux sont des objets :



The screenshot shows an IDE window with the following content:

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
javascript > cours2.js
cours2.js x
1 // array = []
2 // les éléments sont indexés à partir de 0, comme en C, accès avec l'opérateur []
3 let x = [1, 3, 4];
4 x.
5 concat(...items: ConcatArray<T>[]) T[]
6 copyWithin(target: number, start: number, end?: number) this
  entries() IterableIterator<[number, T]>
  every(callbackfn: (value: T, index: number, array: T[]) => unk... boolean
  fill(value: T, start?: number, end?: number) this
  filter<S>(callbackfn: (value: T, index: number, array: T[]) => boo... S[]
  find<S>(predicate: (this:void, value: T, index: number, ... S | undefined
  findIndex(predicate: (value: T, index: number, obj: T[]) => unk... number
  flat<U>(depth?: number) any[]
  flatMap<U, This>(callback: (this:This, value: T, index: number, ar... U[]
  forEach(callbackfn: (value: T, index: number, array: T[]) => void... void
  includes(searchElement: T, fromIndex?: number) boolean
Press Enter to insert, Tab to replace Next Tip
```

⇒ contiennent des méthodes (length, filter, forEach, etc.)

Typeage dynamique

The screenshot shows a code editor with the following JavaScript code in `typage.js`:

```
1 // chaque variable a un type, qui peut changer dynamiquement
2 let x = 4;
3 console.log(typeof x);
4
5 x = 'toto';
6 console.log(typeof x);
7
8 let y;
9 console.log(y, typeof y); // undefined = type et valeur
10 y = null;
11 console.log(y, typeof y);
12
```

The terminal output shows the result of running `node typage.js`:

```
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<16> node typage.js
number
string
undefined 'undefined'
null 'object'
```

Retour sur les string et les objets

The screenshot shows the WebStorm IDE interface. The top toolbar includes menus like File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The breadcrumb path is: mobile-19-20 > prog > javascript > string_obj.js. The editor window displays the following JavaScript code:

```
1 // strings = type primitif mais si on utilise le '.' => wrappé dans un objet
2 // => intéressant pour les méthodes length, indexOf, trim, replace, etc.
3 let t = 'toto';
4 console.log(t, typeof t);
5 x = t.bold();
6 console.log(x, typeof x);
7 console.log(t.length);
8
9
```

The terminal window at the bottom shows the execution of the script:

```
Local x +
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<21> node string_obj.js
toto string
<b>toto</b> string
4
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<22> 
```

The bottom status bar indicates: WebStorm 2019.3.2 available: // Update... (today 8:34 AM) | 9:1 LF UTF-8 4 spaces Git: master

Template literals

```
javascript [~/enseignement/mobile-19-20/prog/javascript] - .../template_literal.js - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
javascript > template_literal.js Add Configuration... Git: [checked] [checked] [refresh] [undo] [redo] [search]
1 // concat nation de strings : +
2 let x = 4;
3 let y = 'x vaut : ' + x + '\n' + '3 fois' moins que ' + (3 * x);
4 console.log(y);
5
6 // template literals (backquotes : `` ): depuis ES6s
7 let z = `x vaut : ${x}
8 '3 fois' moins que ${3 * x}`;
9 console.log(z);
10
```

Terminal: Local x +

```
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<1> node template_literal.js
x vaut : 4
'3 fois' moins que 12
x vaut : 4
'3 fois' moins que 12
```

Terminal | 6: TODO | Event Log

10:1 LF UTF-8 4 spaces Git: master

Les constantes

```
php [~/enseignement/mobile-19-20/prog/php] - ~/enseignement/mobile-19-20/prog/javascript/constante.js - WebStorm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
mobile-19-20 > prog > javascript > constante.js
Add Configuration...
1 // constante = on ne peut plus réassigner une valeur à la variable
2 // mais si c'est un objet ou un tableau, on peut modifier son contenu
3 const x = 4;
4 // x = 3; => erreur
5
6 const y = { val : 3 };
7 // y = { val : 4 }; => erreur
8 y.val = 5; // valide, ce n'est pas une réaffectation de y
9 console.log(y);
10
11
12
```

Terminal: Local × +

```
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<19> node constante.js
{ val: 5 }
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<20> █
```

Itérer les champs d'un objet (1/2)

The screenshot shows the WebStorm IDE interface. The main editor displays the following JavaScript code in `iter.js`:

```
1 class MonObjet {
2   constructor (val1, val2) {
3     this.champ1 = val1;
4     this.champ2 = val2;
5   }
6
7   // affichage des champs de l'objet
8   display () {
9     // for .. in : parcourt des clefs de
10    // l'objet ou des index pour un tableau
11    for (let key in this)
12      console.log(key, this[key]);
13  }
14 }
15
16 let obj = new MonObjet(3,4);
17 obj.display ();
18
```

The terminal window on the right shows the execution output:

```
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<4> node iter.js
champ1 3
champ2 4
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<5> |
```

The IDE interface includes a menu bar (File, Edit, View, etc.), a toolbar with icons for running and debugging, and a sidebar with project and structure views. The status bar at the bottom shows 'WebStorm 2019.3.2 available: // Update... (5 minutes ago)', '15:1 LF UTF-8 4 spaces', and 'Git: master'.

Itérer les champs d'un objet (2/2)

The screenshot shows a code editor with the following JavaScript code in `iterOf.js`:

```
1 class MonObjet {
2   constructor (val1, val2) {
3     this.champ1 = val1;
4     this.champ2 = ['toto', 'titi', 'tutu'];
5   }
6
7   // affichage des champs de l'objet
8   display () {
9     // for .. of : parcourt des valeurs d'un objet
10    // itérable (par exemple un tableau)
11    for (let key of this.champ2)
12      console.log(key);
13  }
14 }
15
16 let obj = new MonObjet( val1: 3, val2: 4);
17 obj.display ();
18
```

The terminal output shows the execution of `node iterOf.js`, resulting in the following log messages:

```
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<5> node iterOf.js
toto
titi
tutu
[shalmaneser:~/enseignement/mobile-19-20/prog/javascript]<6>
```

The interface includes a sidebar with 'Project', 'Structure', and 'Favorites' views. The bottom status bar shows 'WebStorm 2019.3.2 available: // Update... (21 minutes ago)', '18:1 LF UTF-8 4 spaces Git: master', and 'Event Log'.

Copier (cloner) un objet

The screenshot shows a code editor with a file named `clone.js` open. The code defines an object `obj` with three properties: `champ1` (value 3), `champ2` (array [1, 2, 3]), and a `display` method. It then demonstrates three ways to clone this object: using a loop, a spread operator, and a `const` declaration with a spread operator.

```
1 let obj = {
2   champ1 : 3,
3   champ2 : [1,2,3],
4   display () { console.log(':::', this.champ1);
5 };
6
7 // création d'un objet vide puis recopie de tous
8 // les champs de obj dans obj2 => copie obj
9 const obj2 = {};
10 for (let key in obj) {
11   console.log(key);
12   obj2[key] = obj[key];
13 };
14 obj2.display();
15
16 // ... = spread operator
17 // expanse l'ensemble des champs de obj2
18 const obj3 = { ...obj2 };
19 obj3.display ();
20
```

The terminal output shows the execution of `node clone.js`, which prints the values of `champ1`, `champ2`, and `display` for both the original object and the cloned object `obj2`.

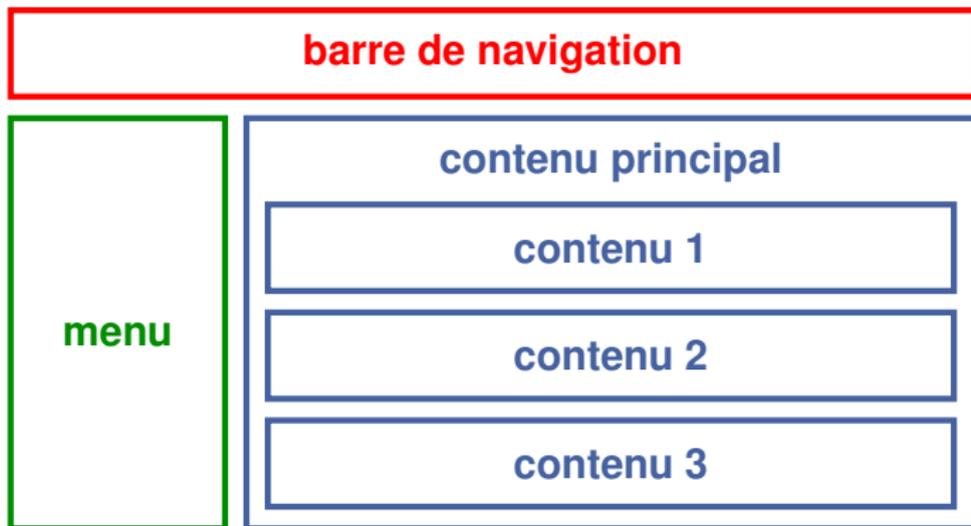
```
[shalmaneser:~/enseignement/mobile-19-20/
prog/javascript]<7> node clone.js
champ1
champ2
display
::: 3
::: 3
[shalmaneser:~/enseignement/mobile-19-20/
prog/javascript]<8>
```

Angular 

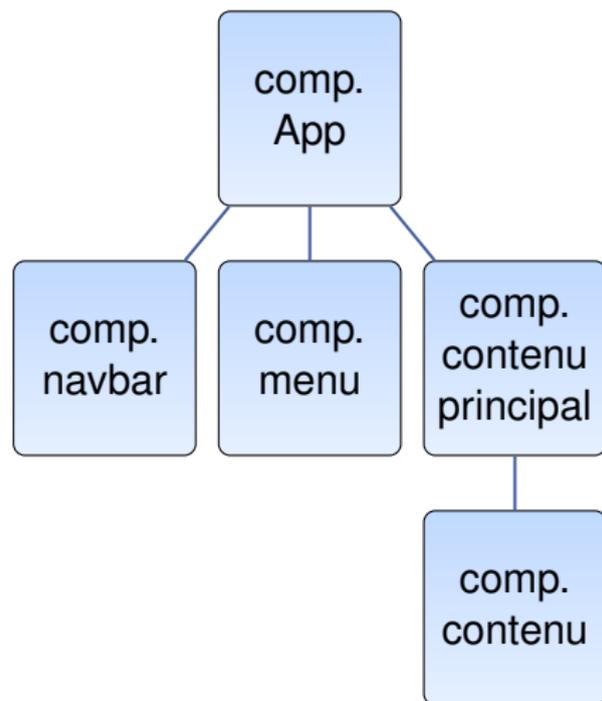
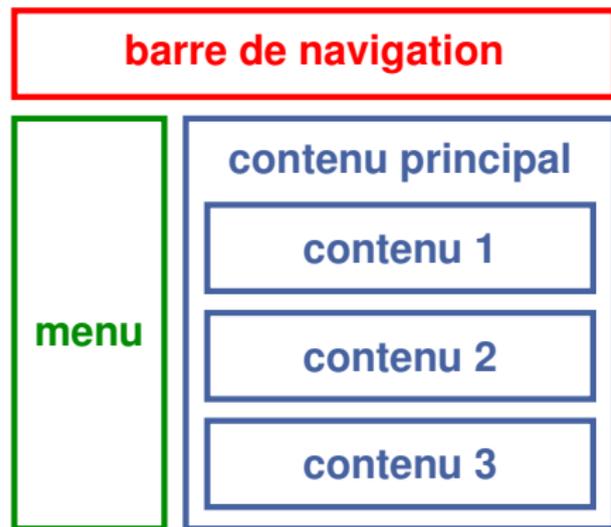
- ▶ Framework pour construire des applications clientes
⇒ front-end
- ▶ Structure l'application
⇒ simplifie programmation/maintenance/débuggage
- ▶ Mise en place de tests simple
- ▶ Utilise TypeScript/Javascript, HTML, CSS

Structure d'une application Angular

Affichage de la page web :



- ▶ Affichage \implies structure
- ▶ Chaque rectangle = **composant Angular**
- ▶ Intérêt des composants : réutilisables plusieurs fois
- ▶ Un composant peut en inclure d'autres



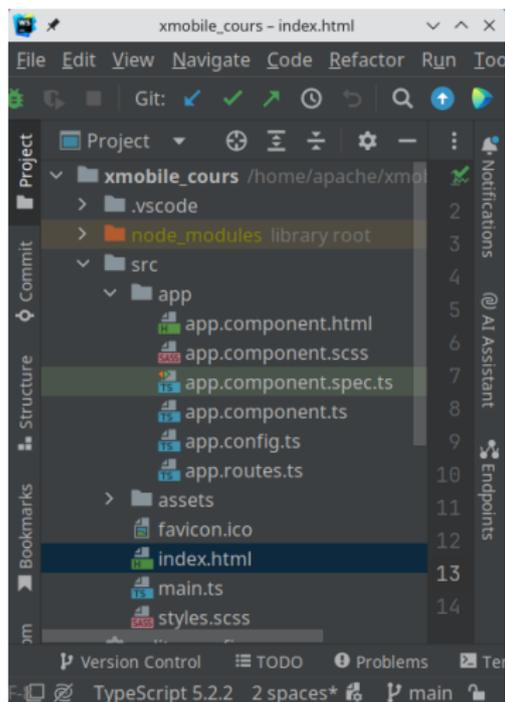
⇒ permet de structurer facilement le code !

Un composant Angular contient essentiellement :

- 1 un fichier TypeScript contenant :
 - ▶ les données du composant
 - ▶ la logique/le comportement du composant
 - 2 un fichier html
 - ▶ contenant le code HTML affiché par le browser
 - ▶ des instructions pour interagir avec le code TypeScript
 - 3 un fichier css contenant le style propre au composant
- ▶ Répertoire `src/app` contient les composants
 - ▶ 1 composant principal appelé `app` ou `root`

Génération d'un projet Angular

- ▶ `ng new mon-projet`
⇒ crée le composant `app` :



- ▶ Dans `src/app` :
 - ▶ `app.component.ts` :
code TypeScript
 - ▶ `app.component.spec.ts` :
pour faire des tests
 - ▶ `app.component.html` :
template HTML
- ▶ Dans `src` :
 - ▶ `index.html` :
point d'entrée de l'appli

Index.html

File Edit View Navigate Code Refactor Run Tools Git Window Help

xmobile_cours src index.html Angular CLI Server

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>XmobileCours</title>
6   <base href="/">
7   <meta name="viewport" content="width=device-width, initial-
8   <link rel="icon" type="image/x-icon" href="favicon.ico">
9 </head>
10 <body>
11   <app-root></app-root>
12 </body>
13 </html>
14
15
```

Project Structure: xmobile_cours /home/apache/xmobile_cours

- .vscode
- node_modules library root
- src
 - app
 - app.component.html
 - app.component.scss
 - app.component.spec.ts
 - app.component.ts
 - app.config.ts
 - app.routes.ts
 - assets
 - favicon.ico
 - index.html
 - main.ts
 - styles.scss
 - .editorconfig
 - .gitignore
 - angular.json
 - package.json
 - package-lock.json
 - README.md
 - tsconfig.app.json
 - tsconfig.json
 - tsconfig.spec.json

External Libraries

Git TODO Problems Terminal Services

Externally added files can be added to Git // View Files // Always Ad... (today 10:32)

15:1 LF UTF-8 TypeScript 5.2.2 2 spaces* main

Insertion of the app component

- 1 Créer les composants (et les modules)
- 2 Les insérer dans l'application via des balises dans les fichiers HTML

Exemple : `<app-root></app-root>`

- ▶ Pour compiler et « servir » votre application (en mode dev) :
`ng serve` OU `npm start`

Le composant App et l'appli servie

The screenshot shows an IDE (Visual Studio Code) with the following components:

- File Explorer:** Shows the project structure for `xmobile_cours`, including `src/app` with files like `app.component.html`, `app.component.spec.ts`, `app.config.ts`, and `app.routes.ts`.
- Code Editor:** Displays the content of `app.component.html`:

```
<h1>Ceci est mon composant App</h1>
<router-outlet></router-outlet>
```
- Browser Preview:** A window titled "XmobileCours" at `localhost:4200` displays the rendered text: **Ceci est mon composant App**.
- Terminal:** Shows the command `ng serve` being executed, with output: `[shalmanser:/home/apache/xmobile_cours]<3> ng serve`, `Node.js version v21.4.0 detected.`, and a warning about Node.js versions: `Odd numbered Node.js versions will not enter LTS status and should not be used for production. For more information, please see https://nodejs.org/en/about/previous-releases/.`
- npm Audit:** A table showing the initial chunk files:

Initial Chunk Files	Names	Raw Size
<code>polyfills.js</code>	<code>polyfills</code>	<code>82.71 KB</code>

Données http \implies servir 127.0.0.1 plutôt que localhost

- ▶ npm start :
 - ▶ objet "scripts" du fichier package.json :

```
"start" : "ng serve --host 127.0.0.1"
```
- ▶ ng serve :
 - ▶ dans l'objet "serve" du fichier angular.json, rajouter :

```
"options" : { "host" : "127.0.0.1" }
```

Le TypeScript du composant app

```
File Edit View Navigate Code Refactor Run Tools Git Window Help
src > app > app.component.ts AppComponent Angular CLI Server Git:
app.component.ts x
// décorateur qui indique à Angular que la classe TypeScript en dessous est le
// code d'un composant
@Component({
  // indique la balise HTML à utiliser pour insérer le composant dans l'application
  selector: 'app-root',

  // indique que le composant est autonome (pas besoin d'appartenir à un module)
  standalone: true,

  // les modules (libraries) dont dépend ce composant
  imports: [CommonModule, RouterOutlet],

  // indique où se trouve le code HTML du composant (celui à indiquer dans l'appli
  // quand on insère le composant)
  templateUrl: './app.component.html',

  // les styles propres, spécifiques, au composant
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  // un composant contient une classe TypeScript qui sert
  // à contenir ses données et sa logique
}
```