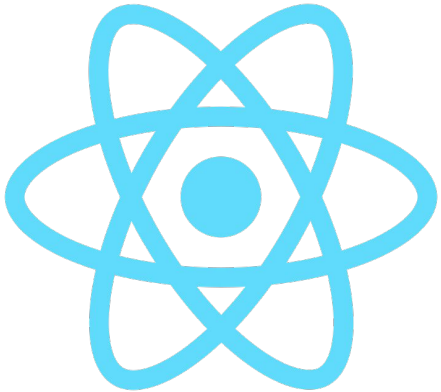
Cours ReactJS

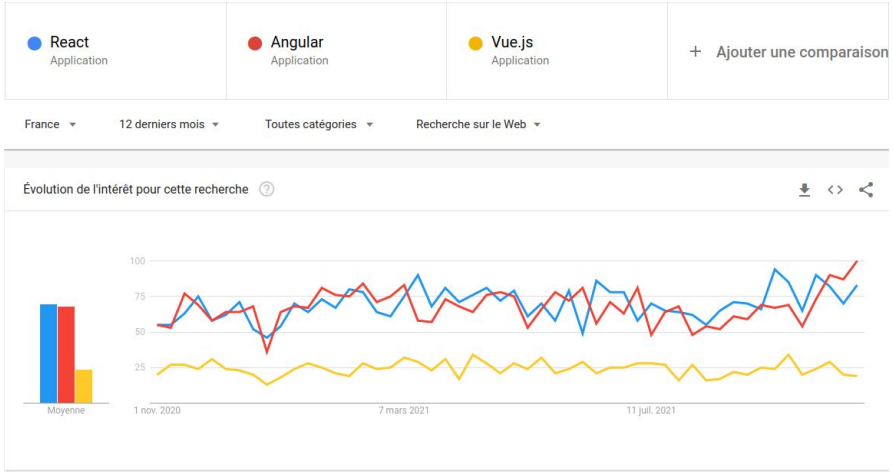


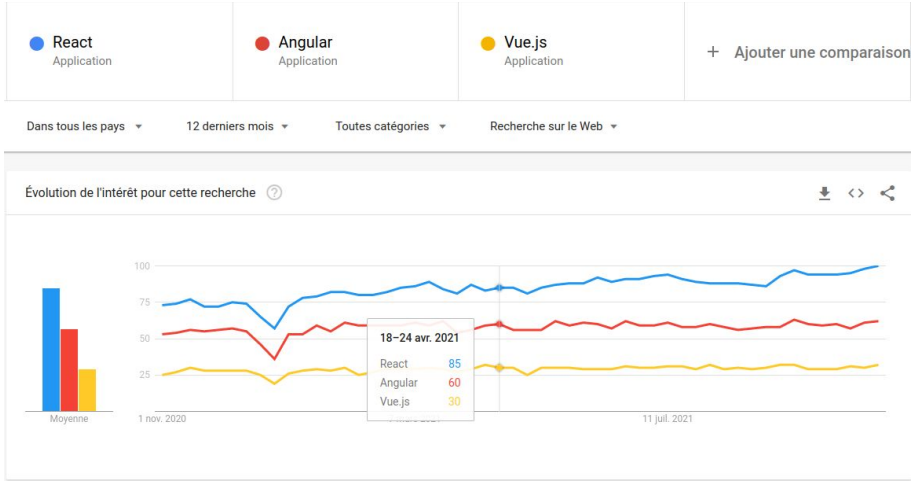
Pourquoi utiliser un framework?

- Profiter du travail partagé par d'autres développeurs. "Ne pas réinventer la roue"
- Disposer d'une boîte à outils utilisée au sein de nombreuses entreprises
- Code mieux structuré, plus lisible, concis
- Pour le cas de React, disposer d'une architecture orientée composant **réutilisables**
- Les inconvénients
  - Apprentissage supplémentaire parfois conséquent
  - Connaissances spécifiques
  - Obsolescence de la librairie au fil du temps (parfois rapide)

ReactJS

ReactJS est un librairie/bibliothèque javascript libre développée par Facebook (depuis 2013) et dédiée à la **création d'interfaces utilisateurs sous forme de composants**





source: [google trends]

Premiers pas avec React. Les composants

React permet de créer des **composants indépendant et réutilisables**

Mais c'est quoi un composant ?

Est un composant HTML représentant une zone de saisie pour l'utilisateur

- Réutilisable
- Indépendant
  - Ne dépend pas d'un autre composant
- Possède des propriétés/attributs
  - type = "text" est une propriété du composant. D'autres valeurs sont possibles
  - Mais elles sont **immutables** une fois le composant créé
- Possède des "écouteurs" sur des interactions utilisateurs "event listeners"

- *"Bon bah parfait y'a besoin de rien si on peut faire des composants en HTML, fin du cours?"*
- *Non...*

Premiers pas avec React. les composants

Bien que les composants proposés par les versions successives d'HTML soient de plus en plus nombreux, leur but est de rester le plus générique possible

- Une très grande majorité des sites et applications web utilisent des boutons, des cases à cocher, des lecteurs vidéos…

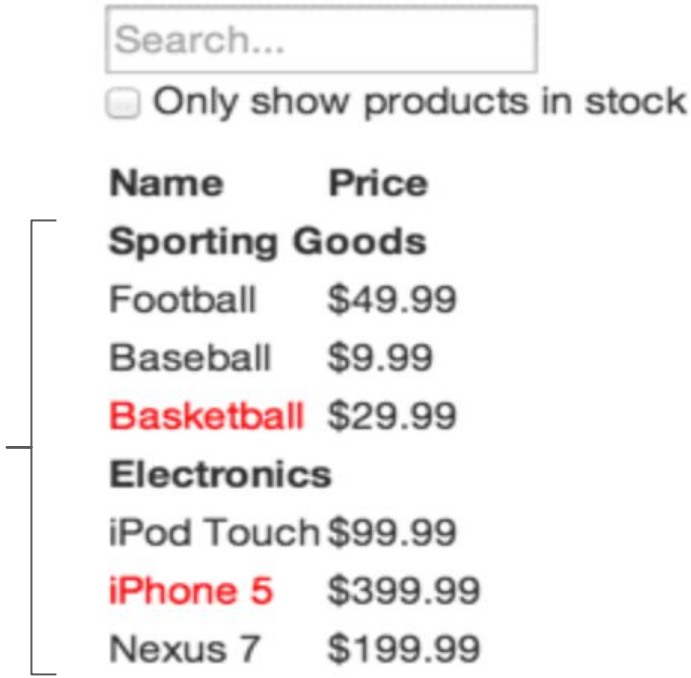
Mais il serait évidemment impossible de créer l'ensemble des composants utiles à n'importe quelle application.

Le rôle du développeur est d'utiliser un ensemble générique de fonctionnalités pour résoudre des problèmes spécifiques

Premiers pas avec React. les composants

Exemple de découpage en composants d'une application listant des produits avec le modèle de donnée suivant :

```
[
  {category: "Sporting Goods", price: "$49.99", stocked: true, name: "Football"},
  {category: "Sporting Goods", price: "$9.99", stocked: true, name: "Baseball"},
  {category: "Sporting Goods", price: "$29.99", stocked: false, name: "Basketball"},
  {category: "Electronics", price: "$99.99", stocked: true, name: "iPod Touch"},
  {category: "Electronics", price: "$399.99", stocked: false, name: "iPhone 5"},
  {category: "Electronics", price: "$199.99", stocked: true, name: "Nexus 7"}
];
```



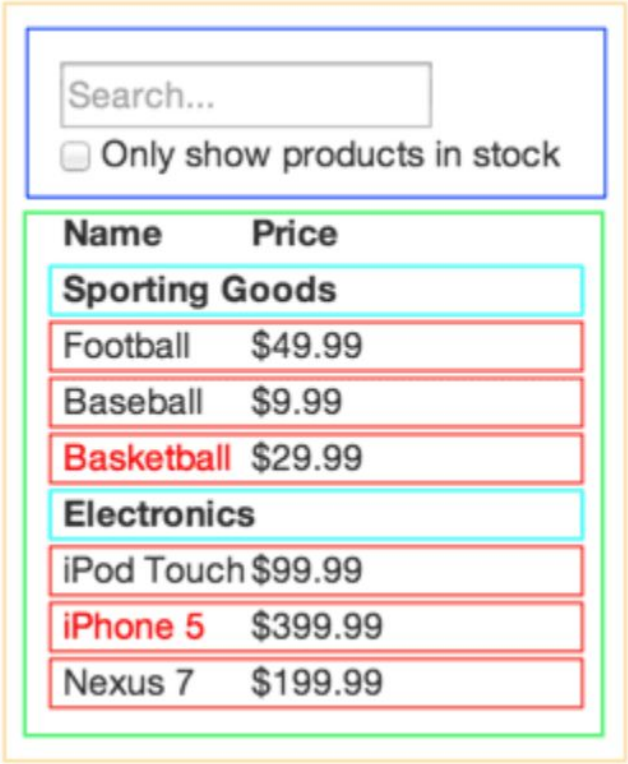
La fonction principale d'un composant est de transformer des données brutes en une interface graphique riche (HTML + javascript)

Premiers pas avec React. les composants

On peut découper cette mini application en 5 composants :

1. **FilterableProductTable (orange) :** contient l'intégralité de l'exemple
2. **SearchBar (bleu) :** reçoit toutes les *données saisies par l'utilisateur*
3. **ProductTable (vert) :** affiche et filtre la *collection de données* en fonction des *données saisies par l'utilisateur*
4. **ProductCategoryRow (turquoise) :** affiche un titre pour chaque *catégorie*
5. **ProductRow (rouge) :** affiche une ligne pour chaque *produit*

Notez la hiérarchie de composants et sous-composants. On parle de "composant parent" et "composant enfant"



source

Premiers pas avec React. les composants

Comment faire des composants en React?


```
class HelloMessage extends React.Component {
   render() {
       return (
           <div>
               Salut {this.props.name}
           </div>
       );
   }
}

ReactDOM.render(
   <HelloMessage name="John Doe"/>,
   document.getElementById('hello-example')
)
```

On va devoir expliquer un peu tout ça...

Premiers pas avec React. les composants

*Tout composant hérite de la classe **React.Component***

*Méthode render = rendu du composant. Un mélange de javascript et de html, du **jsx!***

Fonction "main" à appeler une fois au tout début du code

Déclaration du composant HelloMessage avec la valeur "John Doe" à la propriété name

Réutilisation du composant comme n'importe quel composant HTML!

```
class HelloMessage extends React.Component {
    render() {
        return (
            <div>
                Salut {this.props.name}
            </div>
        );
    }
}

ReactDOM.render(
    <HelloMessage name="John Doe"/>,
    <HelloMessage name="Johnny punchline"/>,
    document.getElementById('hello-example')
)
```

// On accède au propriété d'un composant par son attribut props. Ici la propriété name

Premiers pas avec React. les composants

Récapitulons ce qui se passe dans cet exemple :

1. On appelle ReactDOM.render() avec l'élément <HelloMessage name="John Doe" />.
2. React appelle le composant HelloMessage avec **comme props {name: 'John Doe'}**.
3. Notre composant Welcome retourne un élément Salut John Doe</h1> pour résultat.
4. React DOM met à jour efficacement le DOM pour correspondre à Salut John Doe</h1>.

```
class HelloMessage extends React.Component {
    render() {
        return (
            <div>
                Salut {this.props.name}
            </div>
        );
    }
}

ReactDOM.render(
    <HelloMessage name="John Doe"/>,
    <HelloMessage name="Johnny punchline"/>,
    document.getElementById('hello-example')
)
```

Premiers pas avec React. JSX

JSX pour JavaScript eXtension

- C’est une extension permettant d’écrire des fragments de code HTML et Javascript au même endroit.
- Bien que cela ne soit pas obligatoire, nous utiliserons toujours jsx dans ce cours.
- Après la compilation, les expressions JSX deviennent de simples appels de fonctions JavaScript, dont l’évaluation renvoie des objets JavaScript.

JSX ~ HTML + javascript

```
const element = <h1>Bonjour, monde !</h1>;
```

“Cette drôle de syntaxe n’est ni une chaîne de caractères ni du HTML.

Ça s’appelle du JSX, et c’est une extension syntaxique de JavaScript. Nous recommandons de l’utiliser avec React afin de décrire à quoi devrait ressembler l’interface utilisateur (UI). JSX vous fait sûrement penser à un langage de balisage, mais il recèle toute la puissance de JavaScript.

Au lieu de séparer artificiellement les technologies en mettant le balisage et la logique dans des fichiers séparés, React sépare les préoccupations via des unités faiblement couplées appelées « composants », qui contiennent les deux”

Si l’idée d’injecter des balises dans du JS vous met mal à l’aise, cette présentation vous fera peut-être changer d’avis.

reactjs.org

https://fr.reactjs.org/docs/introducing-jsx.html

Premiers pas avec React. JSX

Attention :

Dans la mesure où JSX est plus proche de JavaScript que de HTML, React utilise la casse **camelCase** comme convention de nommage des propriétés, au lieu des noms d'attributs HTML. Par exemple, **class** devient **className** en JSX, et **tabindex** devient **tabIndex**.

```
<HelloMessage class="ma_class_css" name="John
Doe"/>,
```

*class devient **className** en JSX pour spécifier la class css associée au composant*

```
<HelloMessage className="ma_class_css" name="John
Doe"/>
```

Plus de détails sur

https://fr.reactjs.org/docs/introducing-jsx.html

Premiers pas avec React. La méthode render

Notion fondamentale

La méthode render est la place centrale d'un composant React. Elle est chargée de retourner le rendu du composant en **JSX**

- Seule méthode obligatoire d'une classe composant
- Automatiquement appelée (*voir cycle vie d'un composant*)
  - Au "montage" du composant dans le DOM
  - Au changement d'une propriété
  - A l'appel de la méthode setState
  - A l'appel de la méthode forceUpdate

Premiers pas avec React. les props d’un composants

Les props (abréviation de properties) permettent la configuration d’un composant (ses options).

- Le composant les reçoit de son parent.
- Elles sont **immutables,** càd en lecture seule au sein du composant
- Elles sont transmises au composant par son parent. Un peu comme les arguments d’une fonction
- Si le parent change les props d’un composant enfant, **cela déclenche la méthode render du composant enfant** (voir slides sur le cycle de vie d’un composant)

Premiers pas avec React. Première application React

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Title</title>
   <script src="https://unpkg.com/babel-standalone@6/babel.min.js" ></script>
</head>
<body>
   <div id="hello-example" ></div>
   <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
   <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>
   <script type="text/babel" src="hello-exemple.js" ></script>
</body>
</html>
```

chargement de babel pour pouvoir utiliser jsx

div qui contiendra le composant HelloMessage

chargement de React et du composant

hello-example.js

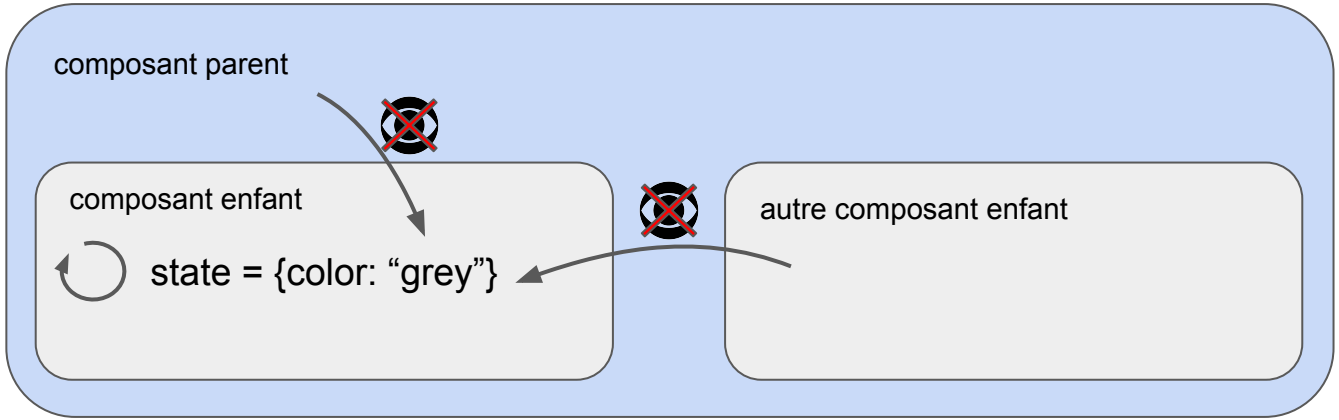
```
class HelloMessage extends React.Component {
   render() {
       return (
           <div>
               Salut {this.props.name}
           </div>
       );
   }
}

ReactDOM.render(
   <HelloMessage class="ma class css" name="John Doe" />,
   document.getElementById('hello-example')
)
```

Premiers pas avec React. L'objet state

Les composants react peuvent disposer d'un état interne manipulable via l'attribut state (lecture) et la méthode setState() (modification)

- L'état d'un composant est **interne au composant (local)**.
- L'état d'un composant n'est accessible et visible qu'à **l'intérieur du composant.**
- **L'état local est réservé à l'interactivité, c'est-à-dire aux données qui évoluent dans le temps**

Notion fondamentale



Premiers pas avec React. L'objet state par l'exemple

```
class ColorSwitcher extends React.Component {

  constructor(props) {
    super(props);
    this.state = {'backgroundColor': 'red'}
  }

  switchColor() {
    const newColor = this.state.backgroundColor === 'blue' ? 'red' : 'blue';
    this.setState({'backgroundColor': newColor}) // appel de la méthode setState pour mettre à jour le composant
  }

  render() {
    return <button onClick={this.switchColor.bind(this)}
                   style={{backgroundColor: this.state.backgroundColor}}>switch color
    </button>;
  }

}


ReactDOM.render(<ColorSwitcher/>, document.getElementById('root'));
```

Premiers pas avec React. L'objet state par l'exemple

1. Initialisation de l'objet state du composant dans le constructeur
   - `this.state = {'backgroundColor': 'red'};`
   -
2. Création de la fonction *SwitchColor pour permuter la couleur*
   - *Notez l'utilisation de la fonction React setState pour changer l'objet*
   - `this.setState({'backgroundColor': newColor})`
3. Appel de la fonction *SwitchColor* au clic sur le bouton
   - `onClick={this.switchColor.bind(this)}`

- Le seul endroit où vous pouvez affecter this.state est **le constructeur,** à tout autre endroit de votre code vous devez utiliser la méthode **setState()**
- Notez ici l'utilisation de la méthode *bind* pour changer la valeur de l'object *this.* Plus d'info sur la doc de mozilla

Premiers pas avec React. la méthode setState

On peut se demander pourquoi utiliser une métode *setState* pour modifier l'état du composant et non en faisant *this.state = {'backgroundColor': 'red'}* comme dans le constructeur.

- A chaque changement de l'objet state, React **doit mettre à jour le composant**
  - On peut alors imaginer le pseudo code de la méthode setState

```
setState(newState) {
  this.state = newState;
  this.render();
}
```

Premiers pas avec React. Cycle de vie d'un composant

Dans une application React, des composants sont créés, mis à jour et détruits périodiquement. Le cycle est toujours le même :

1. Création du composant (on parle de "montage" du composant dans le DOM) : **le constructeur est appelé**
2. Rendu "graphique du composant" : la méthode **render()** est appelée
3. Le composant est monté (présent dans le DOM) : la méthode **componentDidMount()** est appelée
4. Le composant est "démonté" (ex : changement de page) : la fonction **componentWillUnmount()** est appelée

Premiers pas avec React. Cycle de vie d’un composant

```
class ColorSwitcher extends React.Component {

  constructor(props) {
    super(props);
    this.state = {'backgroundColor': 'red'};
    console.log('Appel du constructeur');
  }

  switchColor() {
    const newColor = this.state.backgroundColor === 'blue' ? 'red' : 'blue';
    this.setState({'backgroundColor': newColor})
  }

  render() {
    console.log('Appel de la méthode render');
    return <button onClick={this.switchColor.bind(this)}
                   style={{backgroundColor: this.state.backgroundColor}}>switch color
    </button>;
  }

  componentDidMount() {
    console.log('Méthode Appelée après le constructeur et la méthode render');
  }

  componentWillUnmount() {
    console.log('Méthode Appelée à la destruction du composant')
  }}
```

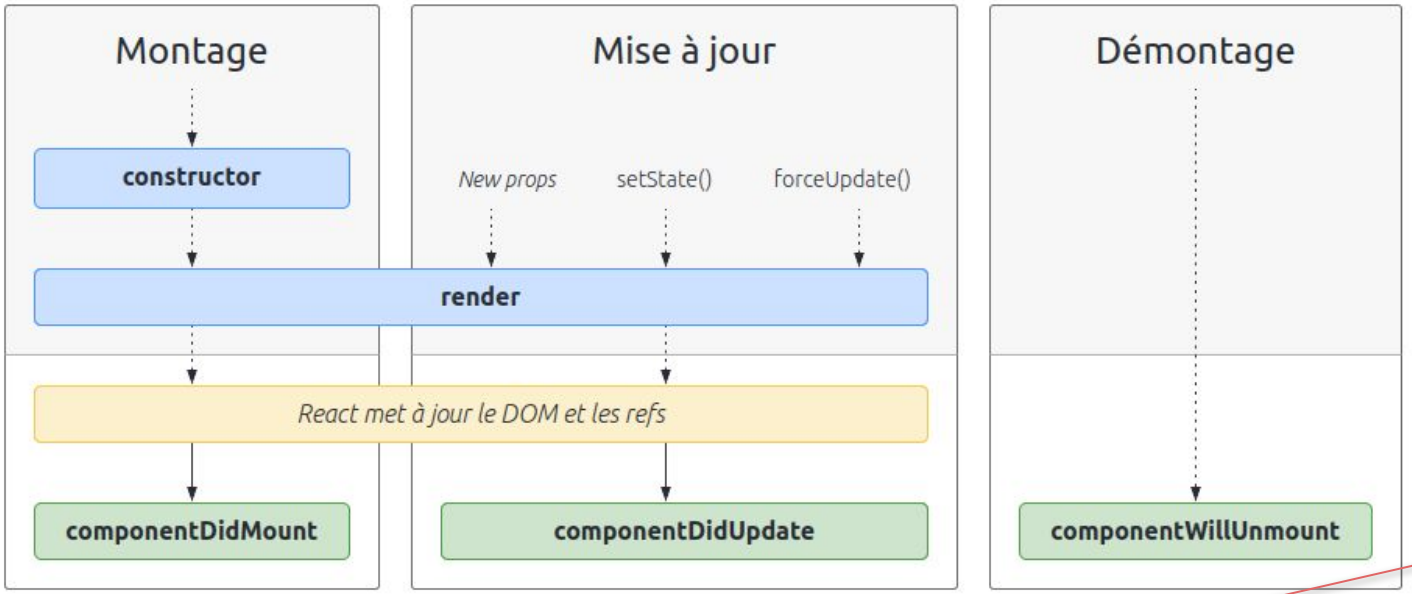
Premiers pas avec React. Cycle de vie d'un composant

"Phase de Render"

Méthodes pures, sans effets secondaires. Peuvent être interrompues, annulées ou redémarrées par React.

"Phase de Commit"

Peuvent opérer sur le DOM, engendrer des effets secondaires, programmer des mise à jour.



Notion fondamentale

https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

Premiers pas avec React. Affichage conditionnel

L'affichage conditionnel en React fonctionne de la même façon que les conditions en Javascript.

On utilise l'instruction Javascript ***if*** ou l'opérateur ternaire pour créer des éléments représentant l'état courant, et on laisse React mettre à jour l'interface utilisateur (UI) pour qu'elle corresponde.

- Voir la documentation react https://fr.reactjs.org/docs/conditional-rendering.html

```
class UserGreeting extends React.Component {
  render() {
    return <h1>Bienvenu</h1>
  }

}

class GuestGreeting extends React.Component {
  render() {
    return <h1>Veuillez vous enregistrer</h1>
  }

}

class Greeting extends React.Component {
  render() {
    const isLoggedIn = this.props.isLoggedIn;
    if (isLoggedIn) {
      return <UserGreeting />;
    }
    return <GuestGreeting />;
  }
}

ReactDOM.render(
  <Greeting isLoggedIn={true} />,  document.getElementById('root')
);
```

Premier pas avec React. Affichage conditionnel

Vous pouvez utiliser n'importe quelle expression dans du JSX en l'enveloppant dans des accolades {}. Ça vaut aussi pour l'opérateur logique Javascript &&. Il peut être pratique pour inclure conditionnellement un élément :

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Bonjour !</h1>
      {unreadMessages.length > 0 &&
      <h2>
        Vous avez {unreadMessages.length} message(s) non-lu(s)
      </h2>}
    </div>
  );
}
```

Premier pas avec React. Affichage conditionnel

Une autre méthode pour l'affichage conditionnel à la volée d'éléments consiste à utiliser **l'opérateur ternaire Javascript**

condition ? trueValue : falseValue.

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Bonjour !</h1>
      {unreadMessages.length > 0
        ? <h2> Vous avez {unreadMessages.length} message(s) non-lu(s)</h2>
        : <h2> Vous n'avez pas de nouveau message</h2>}
    </div>
  );
}
```

Premier pas avec React. Composant fonction

```
class GuestGreeting extends React.Component {
  render() {
    return <h1>Veuillez vous enregistrer {this.props.name}</h1>;
  }
}
```

```
function GuestGreeting(props) {
  return <h1>Veuillez vous enregistrer {props.name}</h1>;
}
```

Ces deux composants sont identiques:

- le retour de la fonction remplace la méthode **render** de la classe
- Les **props** sont un attribut de classe (accès via ***this***). Pour la fonction les **props** sont passées en paramètre

Premier pas avec React. Composant fonction

Quand utiliser l'une ou l'autre forme?

- Si le composant est simple (pas besoin des fonctions du cycle de vie) et entièrement **contrôlé** par son parent (le composant **n'utilise pas d'état "state")**
  - On privilégiera la forme concise des **composants fonction**
- Si le composant est complexe et/ou qu'il a besoin de d'utiliser un état
  - On utilisera la forme des composants **basés sur les classes**

Sachez qu'il est possible d'utiliser la forme "fonction" des composants même si l'on a besoin d'un objet state et/ou d'un accès aux fonctions du cycle de vie grâce au hooks de react .

Premier pas avec React. Conclusion

Le site du projet react https://fr.reactjs.org/ est très bien conçu avec **une bonne documentation**. De nombreux exemples de ce cours sont tirés du site. **N'hésitez pas à le consulter**!

- Il est entièrement traduit en français ce qui ne sera pas souvent le cas dans l'apprentissage de futures librairies

Présentation nextjs: à retenir

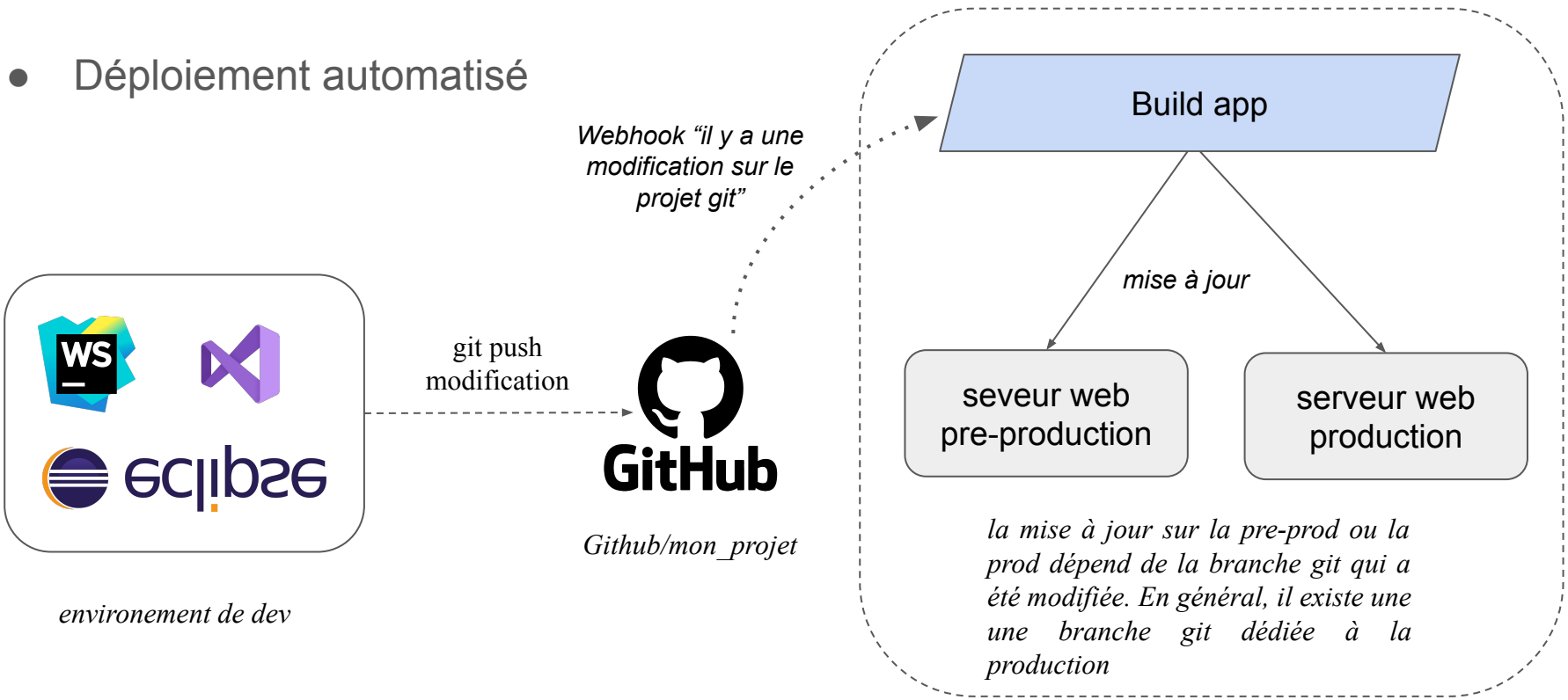
Comme l’outil react-create-app, nextjs permet l’amorçage d’application/site web

- Il permet d’éviter un gros travail de configuration au début du projet
- Il met à disposition un ensemble d’outils très couramment utilisés dans le développement web.

Voyons quelques exemples d’outils indispensables à l’élaboration d’une application web.

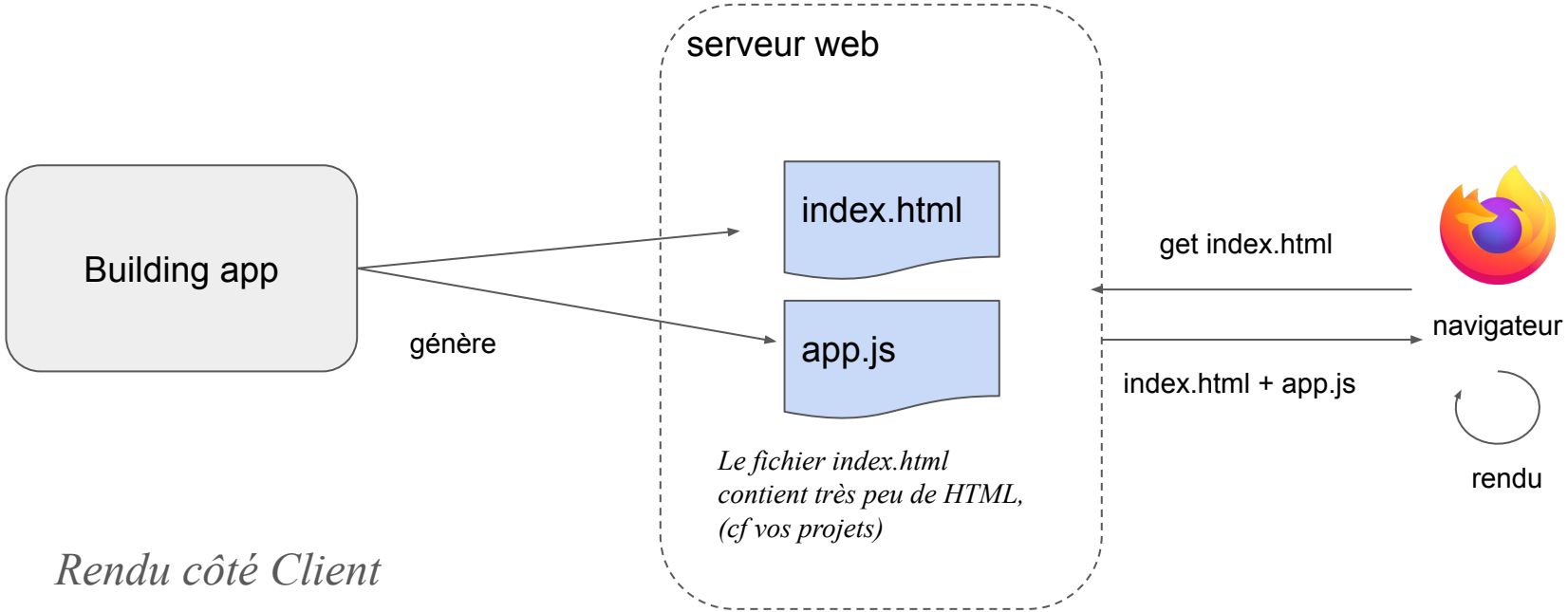
Présentation nextjs: à retenir

- Déploiement automatisé



Présentation nextjs: à retenir

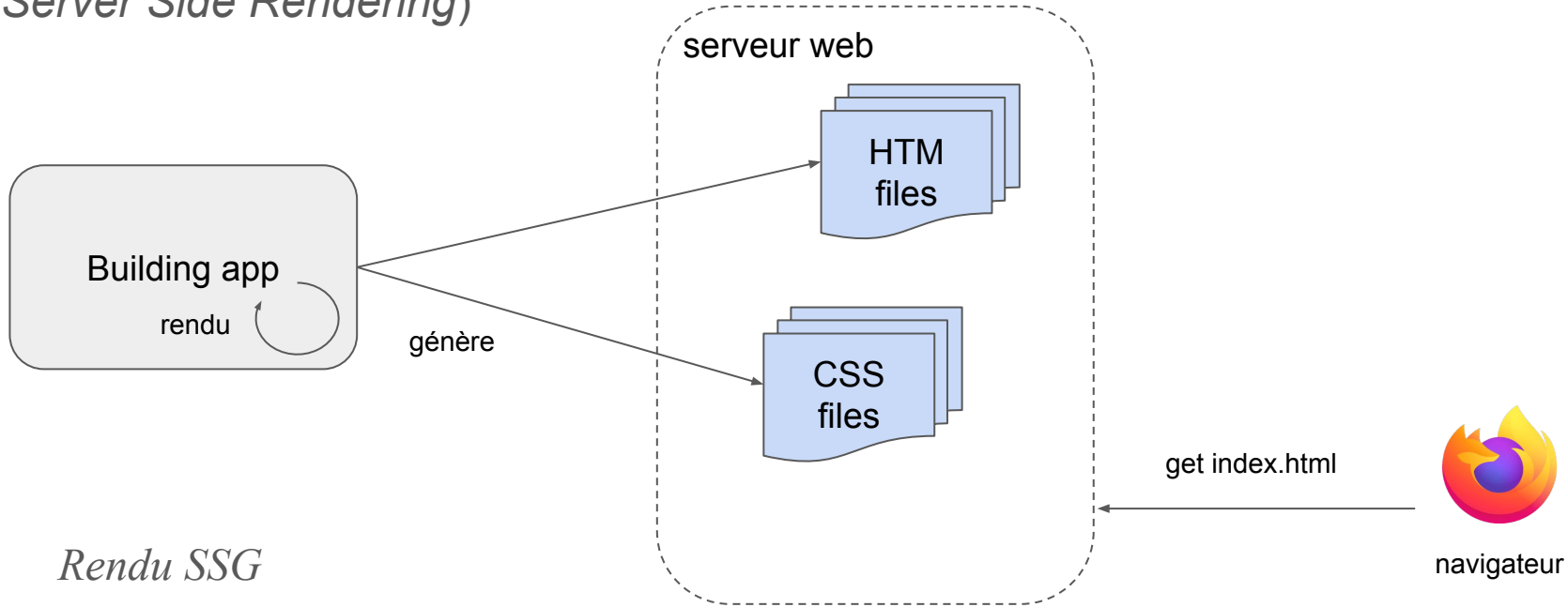
- De base une application React est **rendu côté client**



Rendu côté Client

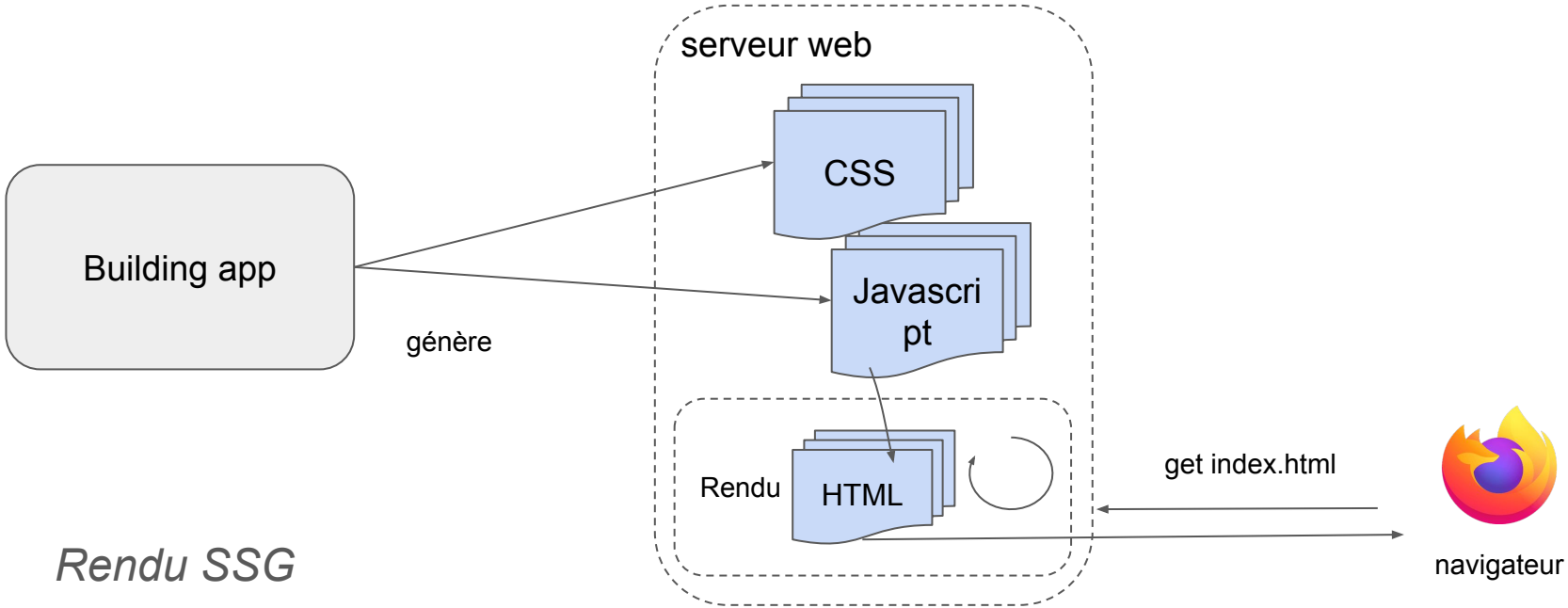
Présentation nextjs: à retenir

- Possibilité de générer un site statique SSG (*Static Site Generation*) ou SSR (*Server Side Rendering*)



Rendu SSG

Présentation nextjs: à retenir



Rendu SSG

Présentation nextjs: à retenir

SSR

- Rendu dynamique en fonction de la requête
  - Réponse personnalisable
- Optimal pour le SEO* par rapport à un rendu client

SSG

- Chargement rapide, page “légère” (très peu de code javascript à charger)
- Optimal pour le SEO* par rapport à un rendu client

**SEO pour Search Engine Optimization. Les robots de Google et autre moteurs de recherches pénalisent les sites où le rendu se fait niveau client, entièrement en javascript, pour des questions de performances dans l’indexation*

Présentation nextjs: à retenir

Quand utiliser client-side, server-side-rendering ou static-site-generation

- Si vous faîtes un site vitrine avec très peu d'interactions utilisateur, privilégiez un site entièrement statique. Il sera plus rapide à charger, plus léger et donc valoriser par Google
- Si vous faîtes un blog, privilégiez un rendu côté serveur qui apportera à la fois une bonne optimisation SEO et la flexibilité d'avoir du contenu dynamique
- Si vous faîtes une application qui ne nécessite pas d'être bien référencée (ex: une app interne à une entreprise, un intranet…), vous pouvez opter pour une application rendu côté client

Méthodes de création d’application web

Il existe trois façons principales de créer des applications mobiles

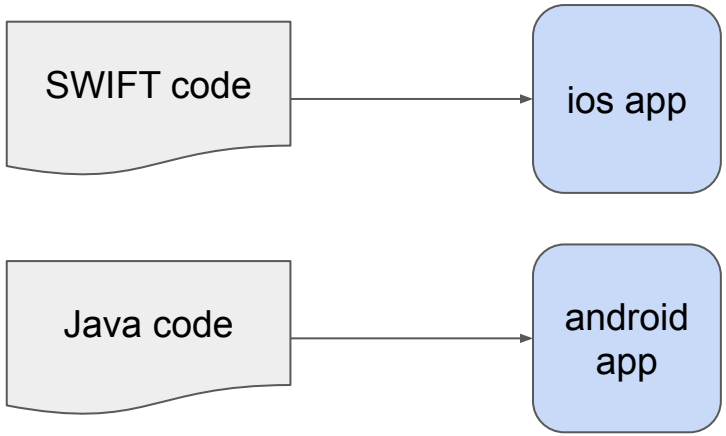
1. La création d’application native en utilisant le langage de programmation de la plateforme
    - swift/objective-c pour ios, java avec le android sdk pour android
2. La création d’application avec les technologies du web
    - Comme **Ionic**
3. La création d’application avec même langage pour toutes les plateforme qui sera “transformé” en code natif
    - **Comme react native**

Méthodes de création d’application web

Avec les outils natifs de chaque plateforme

- Code performant
- Toujours accès aux dernières avancées des plateformes

- Plusieurs sources de code à maintenir
- Connaissances de plusieurs langages
- Coûte beaucoup plus cher

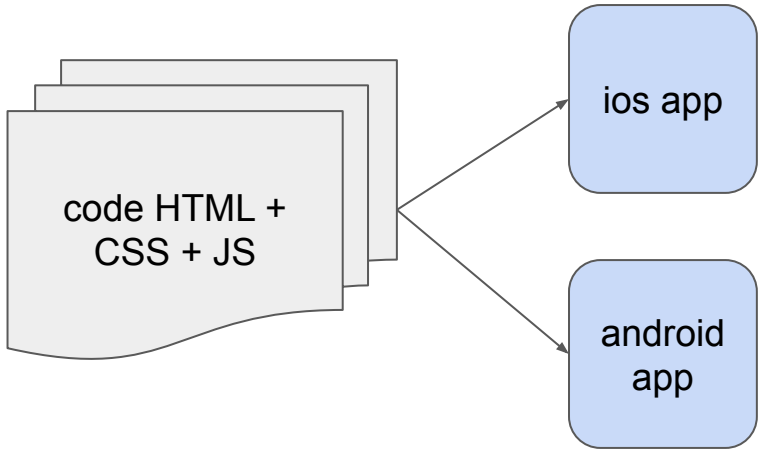


Méthodes de création d’application web

Avec les outils web: “*code once, deploy everywere”*

- Une seule source de code pour toutes les plateformes
- Technologies du web, connu de nombreux développeurs
- Bien moins coûteux

- Application lourde (plusieurs MO) car doit embarquer… Un navigateur web !
- Pas forcément fidèle au rendu graphique de chaque plateforme

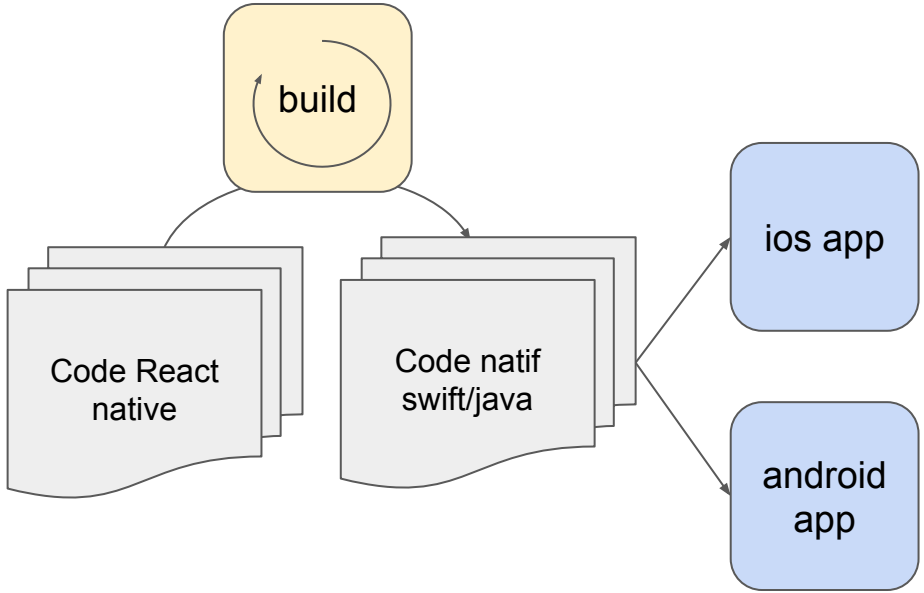


Méthodes de création d’application web

Avec react native: “*learn once deploy everywhere*”

- Une seule source de code pour toutes les plateformes
- Moins coûteux que le natif
- Performances presque similaire
- Rendu graphique respecté par rapport à chaque plateforme

- Nécessite de connaître un langage spécifique (comme Reeact pour Reactnative)



Babel

Babel est un *transcompilateur* principalement utilisé pour assurer la compatibilité des nouvelles fonctionnalités javascript en ancien code javascript.

- **Ne pas attendre la compatibilité des navigateurs pour utiliser les dernières innovations javascript**
- En rapport avec le cours: **pouvoir utiliser JSX** alors que les navigateurs ne le comprennent pas

Exemple babel

```
1 const numbers = [1,2,3,4]
2 const numbers_copy = [...numbers]
3
4 const object = { foo: 1, bar: 3 }
5 const { bar } = object
```

```
3 var numbers = [1, 2, 3, 4];
4 var numbers_copy = [].concat(numbers);
5 var object = {
6   foo: 1,
7   bar: 3
8 };
9 var bar = object.bar;
```

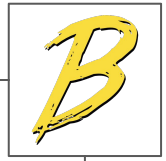
Copier un tableau ou récupérer un attribut d'un objet en EcmaScript 6 (nouvelle version de javascript

Code javascript en version EcmaScript 5 (ancienne version de javascript)

Exemple Babel et et React avec JSX

```
function MyComponent(props) {
  return <div>HEllo world</div>
}
```

Création d'un composant avec JSX dans votre éditeur de code



```
5 function MyComponent(props) {
6   return React.createElement("div", null, "HEllo world");
7 }
```

Composant interprété par votre navigateur

Projet: travail demandé

Création d'une application web utilisant reactjs

- Date de rendu 17 décembre 2021

Un rapport contenant:

1. Votre réflexion avant de vous lancer dans le code. Cela peut prendre plusieurs formes:
   1.1. **obligatoire**: schéma de décomposition en composants react de votre application
   1.2. Story board et/ou Diagrammes UML
2. Une explication succincte des fonctionnalités de votre application
3. Des idées de fonctionnalités supplémentaires
4. Les difficultés rencontrées durant le projet et vos solutions.

- Code disponible sur github. Le lien vers le repo github devra être envoyé dans le mail
- L'application doit impérativement être créé avec l'outil create-react-app ou nextjs

Projet: travail demandé

- Nous sommes dans un cours de React, votre application doit principalement s’appuyer sur react. Ne me faites pas une application avec beaucoup d’algorithmes et peu de composants graphique
- Une application fonctionnelle est primordial pour avoir une bonne note.
  - Je ferai un clone du projet git suivi d’un *npm start*. Si le code ne démarre pas, vous perdrez beaucoup de points
- Une application responsive sera un plus
- Une application avec un design soigné sera apprécié

Mon mail

olivier.semet@protonmail.com