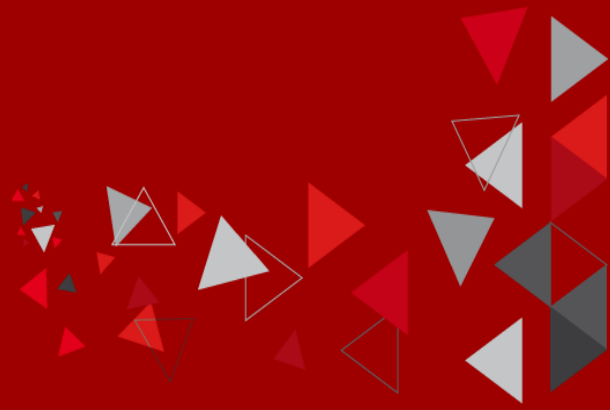
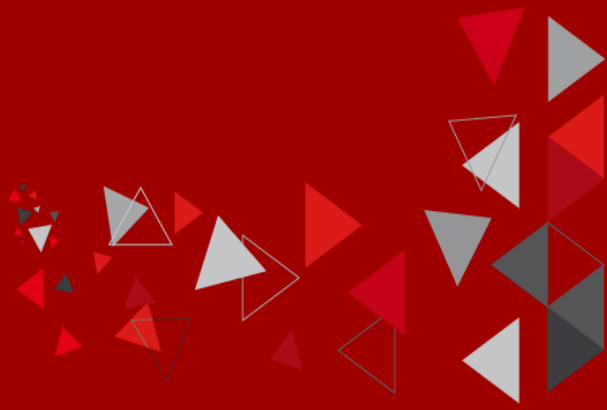
esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

Composants, Props & State

United Nations Educational, Scientific and Cultural Organization
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia
uniTwin
EUR-ACE®
Délivrée par la Commission des Titres d'Ingénieur
CONFÉRENCE DES GRANDES ÉCOLES
cdio
CONCEIVE DESIGN IMPLEMENT OPERATE



- JSX
- Babel
- Composant React
- Props & States
- Cycle de vie d’un composant React



JSX

(JavaScript Syntax eXtension)

JSX

Définition

- JSX c’est l'acronyme de **JavaScript Syntax eXtension** et il est Basé sur **XML.**

- Avec JSX, nous pouvons écrire du HTML dans notre code JavaScript et vice versa:

```
const Hello = ()=>{
        return <h1> Hello React V. {10 + 8 } </h1>;
};
```

- Vu que JSX est un mixte de deux langages différents, il a une syntaxe qui diffère légèrement du HTML que nous connaissons.

JSX

Syntaxe

Let & const dans JSX

```
let name = ‘Jane Doe’;
const element = <h1> Hello, {name}</h1>;
```

Résultat:

Hello, Jane Doe

JSX

Syntaxe

Les objets dans JSX

```
const etudiant = {
            nom : 'Doe',
            prenom : 'Jane',
};
 (...)
return (
  <div>
     <h1>
        Bonjour {etudiant.nom} {etudiant.prenom}
     </h1>
  </div>
);
```

Résultat: **Bonjour Doe Jane**

JSX

Syntaxe

Les fonctions dans JSX

```
function getEtudiant (etudiant) {
        return etudiant;
};
 (...)
return (
<div>
    <h1>
    Bonjour {getEtudiant('Jane Doe')}
    </h1>
</div>
);
```

Résultat: **Bonjour Jane Doe**

JSX

Syntaxe

Les listes dans JSX

```
const listEtudiant =[
{ nom: 'Doe',
  prenom: 'Jane',
  age: '27',
  objectID: 0, },

{ nom: 'Doe',
  prenom: 'John',
  age: '22',
  objectID: 1, },
];
```

```
return (
    <div>
        {
          listEtudiant.map(function(item) {
            return <div key={item.objectID}> {item.prenom} </div>
          }
        }
    </div>
);
```

Résultat:

Jane
John

JSX

Syntaxe

- Pour définir les noms de classe dans JSX, nous utilisons « **className** » et non pas « **class** »:

```
<div className="maClasse">
```

- Pour définir les attributs de la balise « label» dans JSX, nous utilisons « **htmlFor** » et non pas « **for** » :

```
<label htmlFor="name">Name</label>
```

- Nous devons écrire tous les attributs HTML et références d'événements en camelCase. Ainsi, **onclick** devient **onClick**, **onmouseover** devient **onMouseOver**, et ainsi de suite.

  Par exemple :

```
<button onClick={sayHI}>Click</button>
```

- Les commentaires s'écrivent : {/* */}

JSX

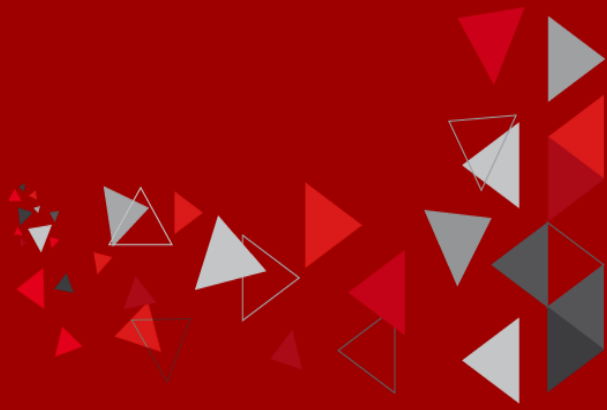
Syntaxe

- Pour définir des styles en ligne dans JSX, vous devez l'écrire en tant qu'objet, avec les propriétés en camelCase, les valeurs entre guillemets, puis vous le transmettez en ligne au JSX, comme dans cet exemple:

- Vous pouvez choisir d'écrire l'objet directement dans l'attribut de style, c'est-à-dire en ouvrant deux accolades et en mettant les propriétés et les valeurs à l'intérieur.

- Mais une méthode plus propre consiste à définir l'objet en dehors du JSX, puis à le passer dans la balise d'ouverture.

```
const Hello = () => {
        const inlineStyle = {
                color: "#2ecc71",
                fontSize: "26px",
        };
 return (
     <>
     <div className="maClasse">
         <p style={inlineStyle}>Hi React Class</p>
     </div>
     </>
 );
};
```



Babel

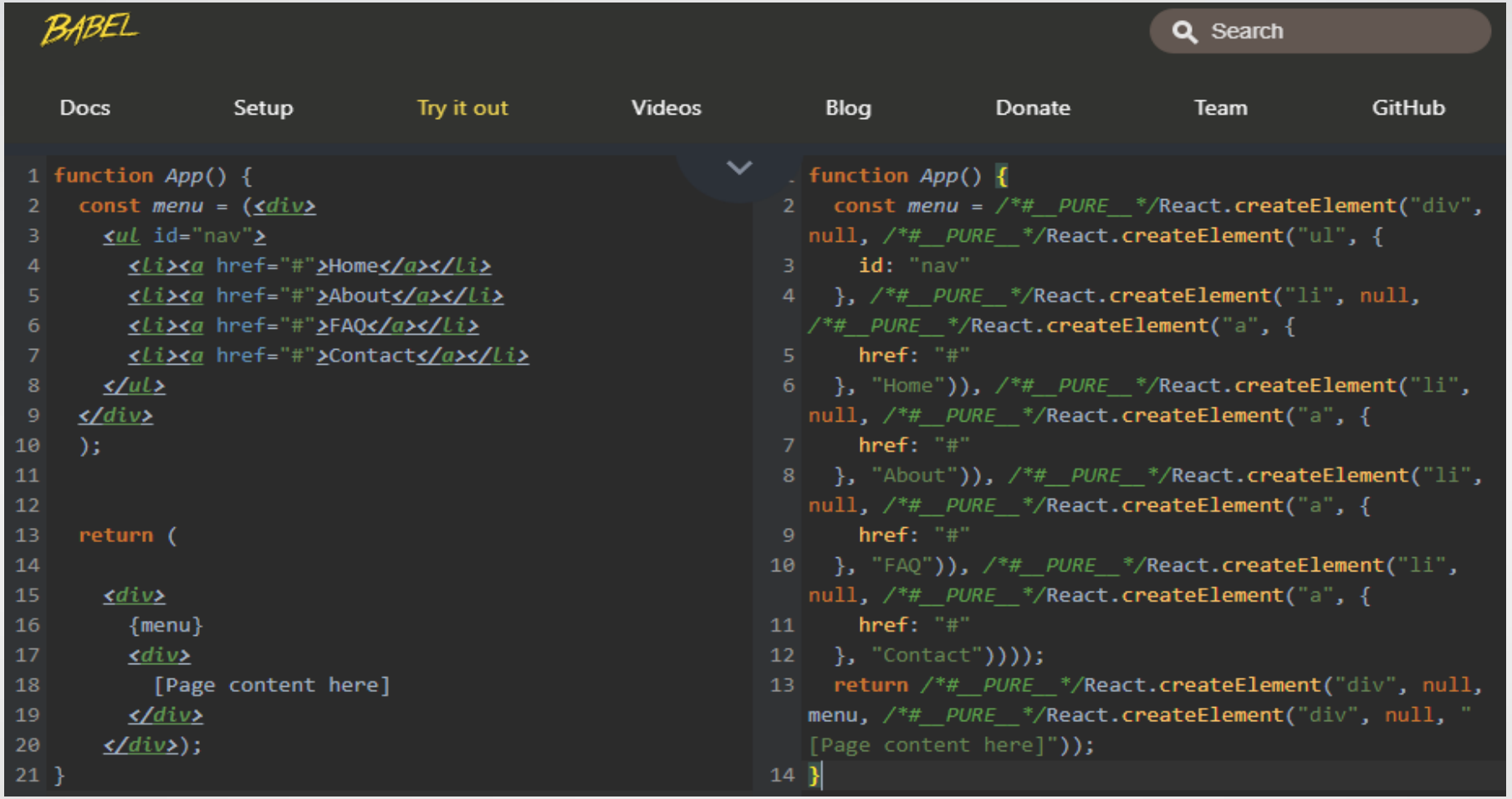
Babel

Définition

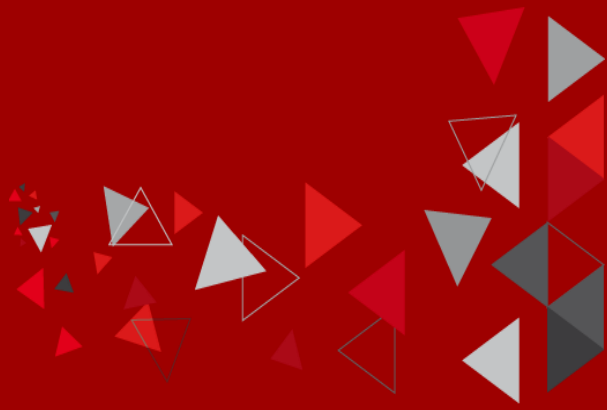
- Babel est un Transpileur, il sert à compiler un code source d'un certain langage de programmation en un code source d'un autre langage.
- Dans notre cas il permettra de convertir du code JavaScript récent (syntaxe ES2015+, JSX etc.) en du code JavaScript capable d'être interprété par les anciens navigateurs.
- Babel peut être installé comme package sur notre projet.
- Nous pouvons aussi utiliser l'utilitaire en ligne de Babel sur l'adresse https://babeljs.io/repl .

Babel

Exemple



Exemple de code JSX transpilé sur https://babeljs.io/repl



Composant React

Composant React

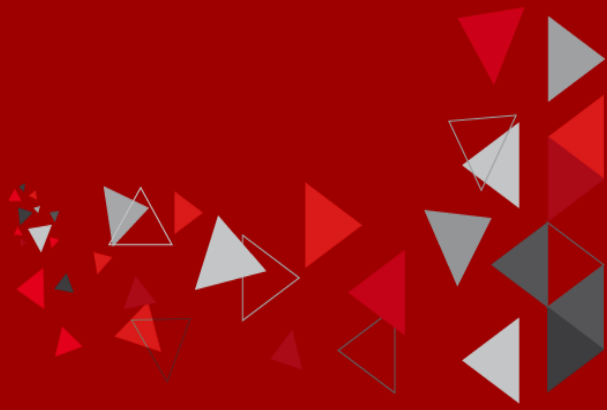
Définition

- Les composants sont essentiellement des ensembles de code qui compartimentent la logique et les fonctions d'une application React.
- Les composants React sont considérés comme les briques sur lesquelles repose l'interface utilisateur.
- Plusieurs composants peuvent exister dans le même espace mais s'exécutent indépendamment les uns des autres.
- Les composants React ont donc leurs propres structures, méthodes, APIs et peuvent être injectés dans une interface au besoin.

Composant React

Pour mieux comprendre, on considère l'ensemble de l'interface utilisateur comme un arbre où chaque composant est un nœud.

Le composant de départ devient la racine et chaque autre composant devient branche et sous branche.



Composant React

Avantages?

Composant React

Avantages

- Réutilisabilité du code.
- Développement Rapide.
- Maintenabilité.

Composant React

Types de Composant

React propose **2 types** de composants :

Composant de classe

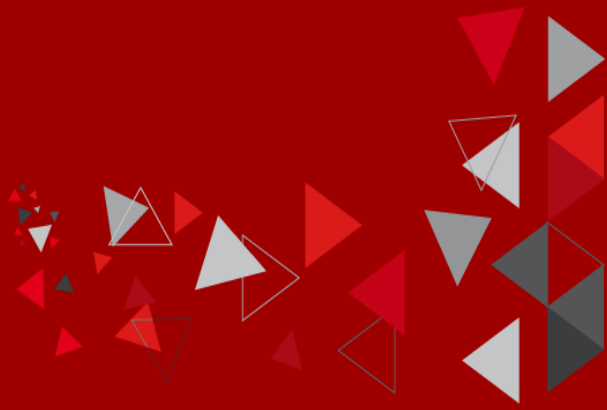
Sous forme de classe et hérite de la classe ***React.Component.***

```
1  import React from "react";
2
3  class ClassComponent extends React.Component {
4   render() {
5     return <h1>Hello, world</h1>;
6   }
7  }
```

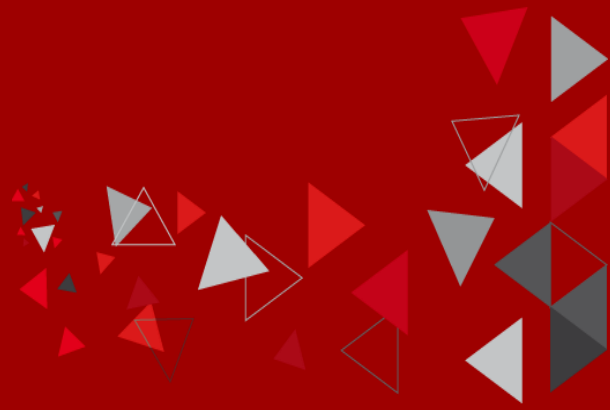
Composant fonctionnel

Composant sous forme d'une ***fonction*** JavaScript

```
1  import React from "react";
2
3  function FunctionalComponent() {
4   return <h1>Hello, world</h1>;
5  }
```



Props & States



Props & States

Props

Props & states

- **Props et states** sont deux variables utilisées par React pour contenir les informations d'un composant.
- Dans une fonction JavaScript, on peut recevoir un ou plusieurs paramètres. C'est le cas pour nos deux types de composants React, c'est ce qu'on appelle des **Props.**
- Les **States** d'un autre côté correspondent à l'état du composant.

La différence fondamentale entre les deux est leur provenance :

- Les **Props** proviennent de l'extérieur du composant
- Les **States** sont des "propriétés", l'état local du composant.

Props & states

Props

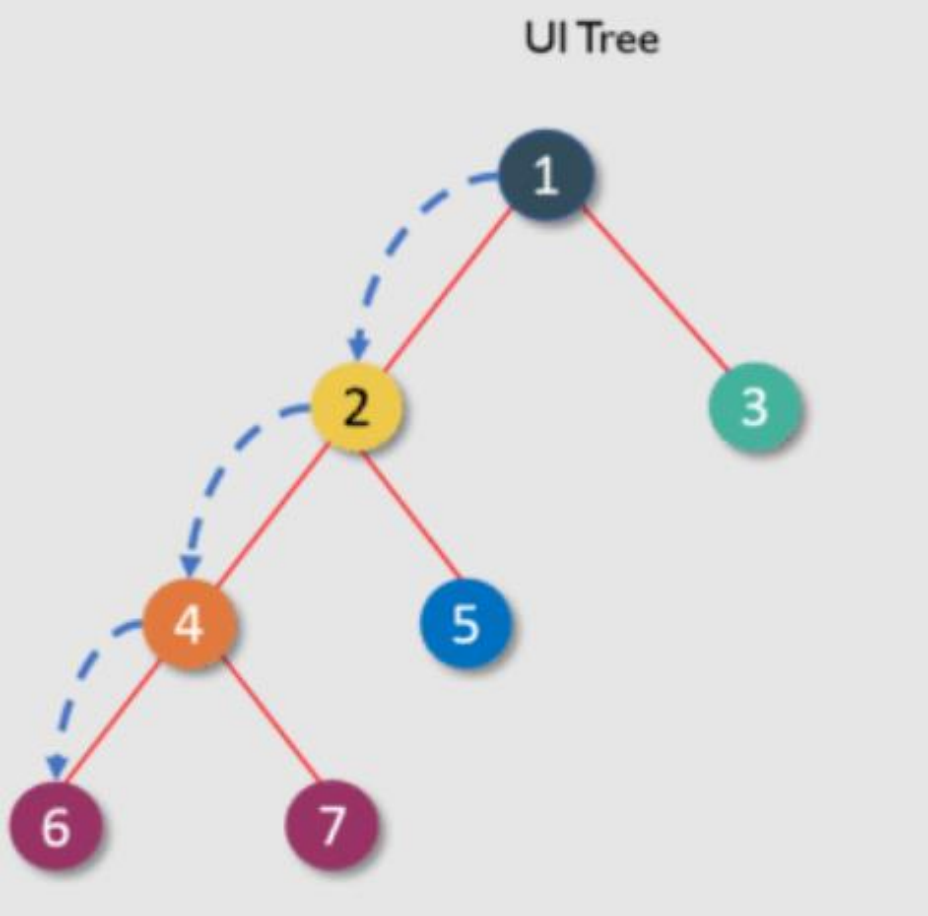
- Les **Props** seront utilisés pour transférer les données d'un composant parent à ses fils.
- Les données des variables **Props** sont en lecture seule pour les fils.
- On ne peut passer une **Props** que d'un composant parent à un composant fils direct.

Props & states

Props

- Pour passer une **props** du composant 1 au composant 6

=> On doit passer par les composants 2 et 4



Props & states

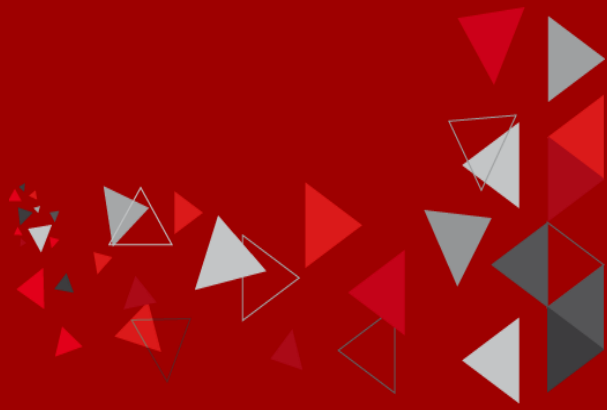
Props dans les Composants de classes

Props dans les Composants fonctionnels

```
const element = <Welcome name="Jane Doe" />;
```

```
class Welcome extends React.Component {
  render() {
    return <h1>Bonjour, {this.props.name}</h1>;
  }
}
```

```
function Welcome(props) {
  return <h1>Bonjour, {props.name}</h1>;
}
```



Props & States

States

Props & states

State

NB: Cette partie concerne les composants de classes.

- Le **state** d'un composant correspond à son état local.
- **State** est une donnée modifiable au cours du temps en réponse à des évènements ou des actions effectués par l'utilisateur.
- **State** est propre au composant actuel, il n'est accessible ou modifiable qu'à l'intérieur du composant.
- Un changement sur le state va engendrer une mise à jour (re-rendering) du composant.
- On utilise la méthode **setState()** pour changer la valeur du state.

Props & states

State

- Il est initialisé dans le constructeur du **composant de classe** ou dans la classe directement.

```
import React from "react";
class Welcome extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value1: "HELLO",
      value2: 0,
      value3: { value31: "works too !!" },
    };
  }
  render() {
    return (
      <div>
        <ul>
          My State: <li> {this.state.value1}</li>
          <li> {this.state.value2}</li>
          <li> {this.state.value3.value31}</li>
        </ul>
      </div>
    );
  }
}
export default Welcome;
```

NB: On aura une erreur si on n'appelle pas le super si on initialise le state dans le constructeur

```
import React from "react";
class Welcome extends React.Component {
  state = {
    value1: "HELLO",
    value2: 0,
    value3: { value31: "works too !!" },
  };

  render() {
    return (
      <div>
        <ul>
          My State: <li> {this.state.value1}</li>
          <li> {this.state.value2}</li>
          <li> {this.state.value3.value31}</li>
        </ul>
      </div>
    );
  }
}
export default Welcome;
```

Merci de votre attention