

Vue.js : directives et binding

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

`elmouelhi.achref@gmail.com`



Plan

1 Introduction

2 Interpolation `{{ ... }}`

- interpoler une variable primitive
- `v-html`
- `v-text`
- interpoler un tableau
- interpoler un objet
- interpoler une expression
- interpoler une méthode

3 Directives structurales

- `v-for`
- `v-if`
- `v-else`
- `v-else-if`
- `v-show`

- 4 Attribute binding : `v-bind`
 - Cas de l'attribut `class`
 - Cas de l'attribut `style`
- 5 Event binding : `v-on`
- 6 Two way binding : `v-model`
- 7 Watchers

Remarque

Dans ce cours, nous considérons le projet `cours-vue-cdn` contenant une **CDN**.

2 modes de liaison (binding) template/script avec **Vue.js**

- One way binding
 - Interpolation `{{ ... }}`
 - Attribute binding `v-bind`
 - Event binding `v-on`
- Two way binding : `v-model`

Interpolation

- s'effectue avec les doubles moustaches {{ ... }}
- permet d'afficher dans le template le contenu d'une donnée déclarée dans la fonction `data`
- la donnée à afficher peut être
 - une variable primitive,
 - un tableau,
 - un objet ou
 - une expression.

Le fichier `script.js`

```
const App = Vue.createApp({
  data() {
    return {
      message: "Hello world"
    };
  }
});
App.mount('#app');
```

Le body d'`index.html`

```
<body>
  <div id='app'>
    {{ message }}
  </div>

  <script src="https://unpkg.com/vue@next"></script>
  <script src="script.js">
  </script>
</body>
```

Le fichier `script.js`

```
const App = Vue.createApp({  
  data() {  
    return {  
      message: "Hello world"  
    };  
  }  
});  
App.mount('#app');
```

Le body d'`index.html`

```
<body>  
  <div id='app'>  
    {{ message }}  
  </div>  
  
  <script src="https://unpkg.com/vue@next"></script>  
  <script src="script.js">  
  </script>  
</body>
```

{{ Interpolation }}

`message` sera remplacé par sa valeur dans la fonction `data()` définie dans la partie (ici `script.js`)

Vue.js

Remarque

L'interpolation interprète les données comme un texte simple (pas comme un contenu **HTML**).

© Achref EL MOUELHI ©

Vue.js

Remarque

L'interpolation interprète les données comme un texte simple (pas comme un contenu **HTML**).

Exemple, ajoutons une balise `span` à notre attribut `message`

```
const App = Vue.createApp({
  data() {
    return {
      message: "<span>Hello world</span>"
    };
  }
});
App.mount('#app');
```

Vue.js

Le résultat affiché sera

```
<span>Hello world</span>
```

© Achref EL MOUELHI ©

Vue.js

Le résultat affiché sera

```
<span>Hello world</span>
```

Pour demander à Vue.js d'interpréter les données comme un contenu HTML, on utilise la directive `v-html`

```
<p v-html="message"></p>
```

Vue.js

Le résultat affiché sera

```
<span>Hello world</span>
```

Pour demander à Vue.js d'interpréter les données comme un contenu HTML, on utilise la directive `v-html`

```
<p v-html="message"></p>
```

Le résultat affiché sera

```
Hello world
```

Vue.js

Une autre alternative de l'interpolation est la directive `v-text` (n'interprète pas le code HTML)

```
<p v-text="message"></p>
```

© Achref EL MOU

Vue.js

Une autre alternative de l'interpolation est la directive `v-text` (n'interprète pas le code HTML)

```
<p v-text="message"></p>
```

Le résultat affiché sera

```
<span>Hello world</span>
```

Vue.js

Pour résumer

- `v-html` appelle `innerHTML`
- `v-text` appelle `textContent`

Vue.js

Ajoutons un tableau d'entiers `tab` dans la fonction `data()` de `script.js`

```
const App = Vue.createApp({
  data() {
    return {
      message: "Hello world",
      tab: [2, 3, 5, 8]
    };
  }
});
App.mount('#app');
```

Afficher le tableau `tab` dans `index.html`

```
<body>
  <div id='app'>
    {{ message }}
    <ul>
      <li>{{ tab[0] }}</li>
      <li>{{ tab['1'] }}</li>
      <li>{{ tab["2"] }}</li>
    </ul>
  </div>
  <!-- + les balises scripts -->
</body>
```

© Achref EL MOUËL

Afficher le tableau `tab` dans `index.html`

```
<body>
  <div id='app'>
    {{ message }}
    <ul>
      <li>{{ tab[0] }}</li>
      <li>{{ tab['1'] }}</li>
      <li>{{ tab["2"] }}</li>
    </ul>
  </div>
  <!-- + les balises scripts -->
</body>
```

Remarques

- Ce code est trop répétitif
- Et si on ne connaissait pas le nombre d'éléments
- Ou si on ne voulait pas afficher tous les éléments

Afficher le tableau `tab` dans `index.html`

```
<body>
  <div id='app'>
    {{ message }}
    <ul>
      <li>{{ tab[0] }}</li>
      <li>{{ tab['1'] }}</li>
      <li>{{ tab["2"] }}</li>
    </ul>
  </div>
  <!-- + les balises scripts -->
</body>
```

Remarques

- Ce code est trop répétitif
- Et si on ne connaissait pas le nombre d'éléments
- Ou si on ne voulait pas afficher tous les éléments

Solution

utiliser les directives (section suivante)

Vue.js

Ajoutons l'objet JavaScript suivant

```
const App = Vue.createApp({  
  data() {  
    return {  
      message: "Hello world",  
      tab: [2, 3, 5, 8],  
      personne: { id: 100, nom: "Wick", prenom: "John" }  
    };  
  }  
});  
App.mount('#app');
```

Vue.js

Pour afficher l'objet dans `index.html`

```
<body>
  <div id='app'>
    {{ message }}
    <ul>
      <li>{{ tab[0] }}</li>
      <li>{{ tab['1'] }}</li>
      <li>{{ tab["2"] }}</li>
    </ul>
  </div>
  <p>{{ personne }}</p>
  <!-- + les balises scripts -->
</body>
```

Vue.js

Pour afficher l'objet dans `index.html`

```
<body>
  <div id='app'>
    {{ message }}
    <ul>
      <li>{{ tab[0] }}</li>
      <li>{{ tab['1'] }}</li>
      <li>{{ tab["2"] }}</li>
    </ul>
  </div>
  <p>{{ personne }}</p>
  <!-- + les balises scripts -->
</body>
```

Résultat

```
{ "id": 100, "nom": "Wick", "prenom": "John" }
```

Vue.js

Pour afficher quelques attributs de l'objet dans le `body`
d'`index.html`

```
<ul>
  <li>Nom : {{ personne.nom }}</li>
  <li>Prénom : {{ personne['prenom'] }}</li>
</ul>
```

© Achret

Vue.js

Pour afficher quelques attributs de l'objet dans le `body` d'`index.html`

```
<ul>
  <li>Nom : {{ personne.nom }}</li>
  <li>Prénom : {{ personne['prenom'] }}</li>
</ul>
```

Résultat

```
Nom : Wick
Prénom : John
```

Vue.js

On peut aussi utiliser une expression (ternaire par exemple) dans une interpolation

```
{{ tab[0] % 2 == 0 ? "pair" : 'impair' }}
```

```
<!-- affiche pair -->
```

© Achref EL MOUELHI

Vue.js

On peut aussi utiliser une expression (ternaire par exemple) dans une interpolation

```
{{ tab[0] % 2 == 0 ? "pair" : 'impair' }}
```

<!-- affiche pair -->

Ou une fonction/méthode JavaScript prédéfinie

```
{{ tab.length }}
```

<!-- affiche 4 -->


```
{{ message.length }}
```

<!-- affiche 11 -->


```
{{ Math.sqrt(4) }}
```

<!-- affiche 2 -->

Vue.js

On peut interpoler une expression mais pas une instruction : ceci génère une erreur

```
{{ var x = 2 }}
```

Vue.js

Ajouter une méthode `direBonjour()` dans une partie `methods`

```
const App = Vue.createApp({
  data() {
    return {
      message: "Hello world",
      tab: [2, 3, 5, 8],
      personne: { id: 100, nom: "Wick", prenom: "John" }
    };
  },
  methods: {
    direBonjour: function() {
      return 'Bonjour Vue.js 3';
    }
  }
});
App.mount('#app');
```

Vue.js

Ou aussi sans le mot-clé `function`

```
const App = Vue.createApp({
  data() {
    return {
      message: "Hello world",
      tab: [2, 3, 5, 8],
      personne: { id: 100, nom: "Wick", prenom: "John" }
    };
  },
  methods: {
    direBonjour() {
      return 'Bonjour Vue.js 3';
    }
  }
});
App.mount('#app');
```

Vue.js

Pour appeler la méthode `direBonjour()` dans le `body` d'`index.html`

```
<body>
  <div id='app'>
    {{ message }}
    <ul>
      <li>{{ tab[0] }}</li>
      <li>{{ tab['1'] }}</li>
      <li>{{ tab["2"] }}</li>
    </ul>
    <p>{{ direBonjour() }}</p>
  </div>

  <script src="https://unpkg.com/vue@next"></script>
  <script src="script.js">
  </script>
</body>
```

Ajouter un attribut `nom` dans une partie `data`

```
data() {  
  return {  
    message: "Hello world",  
    tab: [2, 3, 5, 8],  
    personne: { id: 100, nom: "Wick", prenom: "John" },  
    nom: "Wick"  
  };  
}
```

© Achref EL MOUELHI ©

Ajouter un attribut `nom` dans une partie `data`

```
data() {  
  return {  
    message: "Hello world",  
    tab: [2, 3, 5, 8],  
    personne: { id: 100, nom: "Wick", prenom: "John" },  
    nom: "Wick"  
  };  
}
```

Définissons une méthode `bonjourNom()` dans la partie `methods` et qui référence l'attribut `nom` de la section `data` avec le mot-clé `this`

```
methods: {  
  direBonjour() {  
    return 'bonjour Vue.js 3';  
  },  
  bonjourNom() {  
    return `Bonjour ${this.nom}`;  
  }  
}
```

Ajouter un attribut `nom` dans une partie `data`

```
data() {  
  return {  
    message: "Hello world",  
    tab: [2, 3, 5, 8],  
    personne: { id: 100, nom: "Wick", prenom: "John" },  
    nom: "Wick"  
  };  
}
```

Définissons une méthode `bonjourNom()` dans la partie `methods` et qui référence l'attribut `nom` de la section `data` avec le mot-clé `this`

```
methods: {  
  direBonjour() {  
    return 'bonjour Vue.js 3';  
  },  
  bonjourNom() {  
    return `Bonjour ${this.nom}`;  
  }  
}
```

Pour appeler la méthode `bonjourNom()` dans `index.html`

```
<p>{{ bonjourNom() }}</p>
```

Vue.js

Définissons une méthode `bonjourNom()` avec paramètre dans la partie `methods`

```
methods: {  
  direBonjour() {  
    return 'bonjour Vue.js 3';  
  },  
  //    bonjourNom() {  
  //      return `Bonjour ${this.nom}`;  
  //    },  
  bonjourNom(nom) {  
    return `Bonjour ${nom}`;  
  }  
}
```

Vue.js

Définissons une méthode `bonjourNom()` avec paramètre dans la partie `methods`

```
methods: {  
  direBonjour() {  
    return 'bonjour Vue.js 3';  
  },  
  //    bonjourNom() {  
  //      return `Bonjour ${this.nom}`;  
  //    },  
  bonjourNom(nom) {  
    return `Bonjour ${nom}`;  
  }  
}
```

Pour appeler la méthode `bonjourNom()` avec paramètre dans `index.html`

```
<p>{{ bonjourNom('Dalton') }}</p>
```

Directive ?

- Attribut d'une balise **HTML** commençant par `v-`
- Exemples de directives prédéfinies
 - `v-bind`
 - `v-on`
 - `v-model`
 - `v-html`
 - ...

Vue.js

Directive ?

- Attribut d'une balise **HTML** commençant par `v-`
- Exemples de directives prédéfinies
 - `v-bind`
 - `v-on`
 - `v-model`
 - `v-html`
 - ...

Remarque

Il est possible de définir une nouvelle directive (directive personnalisée).

Deux types de directive

- **Structural directives** : modifient le **DOM**.
- **Attribute directives** : modifient l'apparence d'un élément **HTML** (contenu, couleur...).

Plusieurs directives structurelles

- `v-for`
- `v-if`
- `v-else`
- `v-else-if`
- `v-show`
- ...

v-for

- permet de répéter un traitement (affichage d'un élément **HTML**)
- s'utilise comme un attribut de balise et accepte un tableau, un objet ou un nombre

Afficher les éléments du tableau `tab` en utilisant `v-for`

```
<ul>
  <li v-for="elt in tab">
    {{ elt }}
  </li>
</ul>
```

© Achref EL MOUELHI ©

Afficher les éléments du tableau `tab` en utilisant `v-for`

```
<ul>
  <li v-for="elt in tab">
    {{ elt }}
  </li>
</ul>
```

Remarque

Le code a été compilé mais un message d'erreur est affiché : `error Elements in iteration expect to have 'v-bind:key' directives vue/require-v-for-key.`

Afficher les éléments du tableau `tab` en utilisant `v-for`

```
<ul>
  <li v-for="elt in tab">
    {{ elt }}
  </li>
</ul>
```

Remarque

Le code a été compilé mais un message d'erreur est affiché : `error Elements in iteration expect to have 'v-bind:key' directives vue/require-v-for-key.`

Explication

Vue.js nous demande de spécifier l'identifiant : la clé de chaque élément (à traiter plus tard).

Vue.js

On peut aussi utiliser le `forof` du JavaScript (dans ce cas, `Object.keys()` sera appelée)

```
<ul>
  <li v-for="elt of tab">
    {{ elt }}
  </li>
</ul>
```

Vue.js

Et pour avoir l'indice de l'itération courante

```
<ul>  
  <li v-for="(elt, index) in tab">  
    {{ index }} : {{ elt }}  
  </li>  
</ul>
```

© Achref EL MOU

Vue.js

Et pour avoir l'indice de l'itération courante

```
<ul>
  <li v-for="(elt, index) in tab">
    {{ index }} : {{ elt }}
  </li>
</ul>
```

ou

```
<ul>
  <li v-for="(elt, index) of tab">
    {{ index }} : {{ elt }}
  </li>
</ul>
```

Vue.js

Pour éviter le message d'erreur que Vue.js affiche pour les directives `v-for`, il faut spécifier ce qui pourrait être la clé

```
<ul>
  <li v-for="(elt, index) in tab" :key="index">
    {{ elt }}
  </li>
</ul>
```

© Achref EL

Vue.js

Pour éviter le message d'erreur que Vue.js affiche pour les directives `v-for`, il faut spécifier ce qui pourrait être la clé

```
<ul>
  <li v-for="(elt, index) in tab" :key="index">
    {{ elt }}
  </li>
</ul>
```

ou

```
<ul>
  <li v-for="(elt, index) of tab" :key="index">
    {{ elt }}
  </li>
</ul>
```

Vue.js

Vue.js nous permet également d'itérer sur un objet

```
<ul>
  <li v-for="value in personne">
    {{ value }}
  </li>
</ul>
```

© Achref EL MOU

Vue.js

Vue.js nous permet également d'itérer sur un objet

```
<ul>
  <li v-for="value in personne">
    {{ value }}
  </li>
</ul>
```

Ou avec forof

```
<ul>
  <li v-for="value of personne">
    {{ value }}
  </li>
</ul>
```

Vue.js

Et pour avoir la clé et la valeur

```
<ul>
  <li v-for="(value, key) in personne">
    {{ key }} : {{ value }}
  </li>
</ul>
```

© Achref EL MOU

Vue.js

Et pour avoir la clé et la valeur

```
<ul>
  <li v-for="(value, key) in personne">
    {{ key }} : {{ value }}
  </li>
</ul>
```

ou

```
<ul>
  <li v-for="(value, key) of personne">
    {{ key }} : {{ value }}
  </li>
</ul>
```

Vue.js

Et pour avoir la clé, la valeur et l'indice

```
<ul>  
  <li v-for="(value, key, index) in personne">  
    {{ index }} : {{ key }} : {{ value }}  
  </li>  
</ul>
```

© Achref EL MOU

Vue.js

Et pour avoir la clé, la valeur et l'indice

```
<ul>  
  <li v-for="(value, key, index) in personne">  
    {{ index }} : {{ key }} : {{ value }}  
  </li>  
</ul>
```

ou

```
<ul>  
  <li v-for="(value, key, index) of personne">  
    {{ index }} : {{ key }} : {{ value }}  
  </li>  
</ul>
```

Vue.js

Considérons la liste suivante (à déclarer dans la fonction `data`)

```
personnes: [  
  { id: 100, nom: 'Wick', prenom: 'John' },  
  { id: 101, nom: 'Abruzzi', prenom: 'John' },  
  { id: 102, nom: 'Marley', prenom: 'Bob' },  
  { id: 103, nom: 'Segal', prenom: 'Steven' }  
]
```

© Achref EL

Vue.js

Considérons la liste suivante (à déclarer dans la fonction `data`)

```
personnes: [  
  { id: 100, nom: 'Wick', prenom: 'John' },  
  { id: 101, nom: 'Abruzzi', prenom: 'John' },  
  { id: 102, nom: 'Marley', prenom: 'Bob' },  
  { id: 103, nom: 'Segal', prenom: 'Steven' }  
]
```

Exercice 1

Écrire un code **Vue.js** qui permet d'afficher dans une liste **HTML** les nom et prénom de chaque élément de la liste `personnes` (on n'affiche pas les clés).

Solution

```
<ul>
  <li v-for="elt in personnes">
    {{ elt.prenom }} {{ elt.nom }}
  </li>
</ul>
```

Exercice 2

- Utiliser deux boucles :
 - Une première pour itérer sur le tableau `personnes`
 - Une deuxième pour itérer sur les champs de chaque élément du tableau `personnes`
- Le rendu de ces deux boucles doit être dans une liste **HTML**

Vue.js

Solution

```
<ul>
  <li v-for="elt in personnes">
    <template v-for="value in elt">
      {{ value + " " }}
    </template>
  </li>
</ul>
```

© ACT

Vue.js

Solution

```
<ul>
  <li v-for="elt in personnes">
    <template v-for="value in elt">
      {{ value + " " }}
    </template>
  </li>
</ul>
```

Remarque

La balise `template` n'apparaît pas dans le **DOM**..

Vue.js

`v-for` accepte aussi un nombre (ici 3), elle va donc faire 3 itérations (la valeur initiale est 1)

```
<ul>
  <li v-for="elt in 3">
    {{ elt }}
  </li>
</ul>
```

© Achref EL

Vue.js

v-for accepte aussi un nombre (ici 3), elle va donc faire 3 itérations (la valeur initiale est 1)

```
<ul>
  <li v-for="elt in 3">
    {{ elt }}
  </li>
</ul>
```

Ou avec forof

```
<ul>
  <li v-for="elt of 3">
    {{ elt }}
  </li>
</ul>
```

Vue.js

Pour tester puis afficher si le deuxième élément du tableau est impair

```
<ul>  
  <li v-if="tab[1] % 2 != 0">  
    {{ tab[1] }} est impair  
  </li>  
</ul>
```

© Achref EL MOUL

Vue.js

Pour tester puis afficher si le deuxième élément du tableau est impair

```
<ul>
  <li v-if="tab[1] % 2 !== 0">
    {{ tab[1] }} est impair
  </li>
</ul>
```

Aussi en utilisant `template` qui n'apparaît pas dans le DOM

```
<ul>
  <template v-if="tab[1] % 2 !== 0">
    <li>
      {{ tab[1] }} est impair
    </li>
  </template>
</ul>
```

Exercice

Utiliser les directives `v-if` (et `v-else`) pour afficher le premier élément du tableau (`tab`) ainsi que sa parité (pair ou impair).

Solution avec `v-if` et `v-else`

```
<ul>
  <li v-if="tab[0] % 2 !== 0">
    {{ tab[0] }} est impair
  </li>
  <li v-else>
    {{ tab[0] }} est pair
  </li>
</ul>
```

Remarque

`v-if` et `v-for` ne doivent pas être utilisées dans la même balise.

Exercice

Utiliser les directives **Vue.js** pour afficher sous forme d'une liste **HTML** tous les éléments du tableau précédent (`tab`) ainsi que leur parité. La liste **HTML** doit être **W3C-Valid**

Solution

```
<ul>  
  <template v-for="elt in tab">  
    <li v-if="elt % 2 !== 0">  
      {{ elt }} est impair  
    </li>  
    <li v-else>  
      {{ elt }} est pair  
    </li>  
  </template>  
</ul>
```

Exercice

Utiliser les directives `v-if`, `v-else` et `v-else-if` pour afficher sous forme d'une liste **HTML** les éléments du tableau `moyennes`: `[18, 5, 11, 15]` (à déclarer dans la fonction `data`) avec le message suivant :

- Si $0 \leq \text{valeur} < 10$ alors échec,
- Si $10 \leq \text{valeur} < 13$ alors moyen,
- Si $13 \leq \text{valeur} < 16$ alors bien,
- Sinon très bien.

v-if VS hidden

- `v-if` commentera le contenu si la condition est fausse. L'élément ne sera pas attaché au **DOM**.
- `hidden` attachera l'élément au **DOM** et le marquera avec l'attribut `hidden`.

Remarque

`v-if`, `v-else` et `v-else-if` peuvent être utilisées dans la balise `template`.

v-show

- Similaire à `v-if`.
- N'a pas de `v-else` ni `v-else-if`.
- Ne supporte pas la balise `<template>`.
- Si la condition est fausse, l'élément ne sera pas affiché mais il sera attaché au DOM.

Vue.js

Exemple avec `v-show` (vérifiez que l'élément n'est pas affiché mais qu'il est attaché au DOM avec la propriété CSS : `display : none`)

```
<ul>
  <li v-if="tab[0] % 2 != 0">
    {{ tab[0] }} est impair
  </li>
</ul>
<ul>
  <li v-show="tab[0] % 2 != 0">
    {{ tab[0] }} est impair
  </li>
</ul>
```

Remarque

L'interpolation ne peut pas être utilisée comme valeur d'un attribut **HTML**.

Définissons un attribut lien dans la fonction `data`

```
lien: "http://elmouelhia.free.fr/"
```

On ne peut utiliser l'interpolation pour lier l'attribut HTML à l'attribut de la fonction `data`

```
<p>  
  Voici un lien vers ma page,  
  <a href="{{lien}}"> Cliquez pour visiter</a>  
</p>
```

© Achref EL MOUELHI ©

On ne peut utiliser l'interpolation pour lier l'attribut HTML à l'attribut de la fonction `data`

```
<p>  
  Voici un lien vers ma page,  
  <a href="{{lien}}"> Cliquez pour visiter</a>  
</p>
```

On peut utiliser Attribute binding sur l'attribut `href`

```
<p>  
  Voici un lien vers ma page,  
  <a v-bind:href="lien"> Cliquez pour visiter</a>  
</p>
```

On ne peut utiliser l'interpolation pour lier l'attribut HTML à l'attribut de la fonction `data`

```
<p>  
  Voici un lien vers ma page,  
  <a href="{{lien}}"> Cliquez pour visiter</a>  
</p>
```

On peut utiliser Attribute binding sur l'attribut `href`

```
<p>  
  Voici un lien vers ma page,  
  <a v-bind:href="lien"> Cliquez pour visiter</a>  
</p>
```

Attribute binding

`v-bind:attribute=value` : permet de remplacer `value` par sa valeur dans la fonction `data`

On peut aussi utiliser le raccourci

```
<p>  
  Voici un lien vers ma page,  
  <a :href="lien"> Cliquez pour visiter</a>  
</p>
```

Définissons un attribut `lienTarget` de type objet dans la fonction `data`

```
lienTraget: {href: "http://elmouelhia.free.fr/", target: "_blank"}
```

On peut utiliser l'Attribute binding avec un objet

```
<p>  
  Voici un lien vers ma page,  
  <a v-bind="lienTraget"> Cliquez pour visiter</a>  
</p>
```

© Achref EL M...

On peut utiliser l'Attribute binding avec un objet

```
<p>  
  Voici un lien vers ma page,  
  <a v-bind="lienTraget"> Cliquez pour visiter</a>  
</p>
```

Constat

En cliquant, le lien s'ouvrira dans un nouvel onglet.

Vue.js

`v-bind:class` **ou** `class`

- permet d'attribuer de nouvelles classes à un élément **HTML**.
- permet de récupérer des valeurs définies dans la partie `script` **ou** `style`.

Angular

Ajoutons les trois attributs suivants dans data

```
rouge: true,  
couleur: 'white',  
couleurBg: 'red'
```

© Achref EL MOUELHI ©

Angular

Ajoutons les trois attributs suivants dans data

```
rouge: true,  
couleur: 'white',  
couleurBg: 'red'
```

Définissons la fonction afficherEnRouge dans la partie methods

```
couleurConditionnelle() {  
  if (this.rouge == false) {  
    var resultat = "rouge"  
  } else {  
    var resultat = "bleu"  
  }  
  this.rouge = !this.rouge;  
  return resultat  
}
```

Vue.js

Définissons deux classes rouge et bleu dans style.css qu'on le référence dans index.html

```
.rouge {  
  color: red;  
}  
  
.bleu {  
  color: blue;  
}
```


Vue.js

Définissons deux classes rouge et bleu dans style.css qu'on le référence dans index.html

```
.rouge {  
  color: red;  
}  
  
.bleu {  
  color: blue;  
}
```

Référençons style.css dans index.html

```
<link rel="stylesheet" href="style.css">
```

Vue.js

Pour associer la classe rouge à la balise <p>

```
<p :class="{ 'rouge': true }">
  {{ message }}
</p>
```

© Achref EL MOU

Vue.js

Pour associer la classe rouge à la balise <p>

```
<p :class="{ 'rouge': true }">
  {{ message }}
</p>
```

Ainsi, on peut faire aussi un affichage conditionnel

```
<p :class="{ 'rouge': nom == 'Wick' }">
  {{ message }}
</p>
```

Vue.js

On peut aussi lister les conditions

```
<p :class="{ 'rouge': nom == 'Wick', 'bleu': nom != 'Wick' }">
  {{ message }}
</p>
```

© Achref EL MOUADJID

Vue.js

On peut aussi lister les conditions

```
<p :class="{ 'rouge': nom == 'Wick', 'bleu': nom != 'Wick' }">
  {{ message }}
</p>
```

On peut également utiliser une expression ternaire

```
<p :class="nom == 'Wick' ? 'rouge' : 'bleu'">
  {{ message }}
</p>
```

On peut aussi appeler une méthode dans :class (le premier message sera affiché en rouge et le deuxième en bleu)

```
<p :class="couleurConditionnelle()">
  {{ message }}
</p>
<p :class="couleurConditionnelle()">
  {{ message }}
</p>
```

Exercice

Utiliser `:class` pour afficher en bleu les éléments pairs du tableau précédent (`tab`) et en rouge les éléments impairs.

Vue.js

Première solution

```
<ul>
  <template v-for="elt of tab">
    <li :class="{ 'rouge': elt % 2 !== 0, 'bleu': elt % 2 == 0 }">
      {{ elt }}
    </li>
  </template>
</ul>
```

© Achref EL MOU

Vue.js

Première solution

```
<ul>
  <template v-for="elt of tab">
    <li :class="{ 'rouge': elt % 2 !== 0, 'bleu': elt % 2 == 0 }">
      {{ elt }}
    </li>
  </template>
</ul>
```

Deuxième solution

```
<ul>
  <template v-for="elt of tab">
    <li :class="elt % 2 == 0 ? 'bleu' : 'rouge'">
      {{ elt }}
    </li>
  </template>
</ul>
```

Vue.js

Troisième solution

```
<ul>
  <template v-for="elt of tab">
    <li :class="{ 'rouge': !estPair(elt), 'bleu':
      estPair(elt) }">
      {{ elt }}
    </li>
  </template>
</ul>
```

© Achille

Vue.js

Troisième solution

```
<ul>
  <template v-for="elt of tab">
    <li :class="{ 'rouge': !estPair(elt), 'bleu':
      estPair(elt) }">
      {{ elt }}
    </li>
  </template>
</ul>
```

La méthode `estPair`

```
estPair(elt) {
  return elt % 2 === 0;
}
```

Vue.js

Considérons la liste `personnes` précédente

```
personnes: [  
  { id: 100, nom: 'Wick', prenom: 'John' },  
  { id: 101, nom: 'Abruzzi', prenom: 'John' },  
  { id: 102, nom: 'Marley', prenom: 'Bob' },  
  { id: 103, nom: 'Segal', prenom: 'Steven' }  
]
```

© Achref EL

Vue.js

Considérons la liste `personnes` précédente

```
personnes: [  
  { id: 100, nom: 'Wick', prenom: 'John' },  
  { id: 101, nom: 'Abruzzi', prenom: 'John' },  
  { id: 102, nom: 'Marley', prenom: 'Bob' },  
  { id: 103, nom: 'Segal', prenom: 'Steven' }  
]
```

Exercice

Écrire un script **Vue.js** qui permet d'afficher dans une liste **HTML** les éléments du tableau `personnes` (afficher uniquement nom et prénom). Les éléments d'indice pair seront affichés en rouge, les impairs en bleu.

Solution

```
<ul>
  <template v-for="(elt, index) of personnes">
    <li :class="index % 2 != 0 ? 'rouge' : 'bleu'">
      {{ elt.nom + " " + elt.prenom }}
    </li>
  </template>
</ul>
```

Vue.js

`v-bind:style` ou `style`

- permet de modifier dynamiquement le style d'un élément **HTML**.
- permet de récupérer des valeurs définies dans la partie `script` ou `style`.

Pour attribuer une valeur à la propriété CSS d'une balise

```
<p :style="{ backgroundColor: couleurBg }">
  {{ message }}
</p>
```

© Achref EL MOUL

Pour attribuer une valeur à la propriété CSS d'une balise

```
<p :style="{ backgroundColor: couleurBg }">
  {{ message }}
</p>
```

Pour deux propriétés

```
<p :style="{ backgroundColor: couleurBg, color: couleur }">
  {{ message }}
</p>
```

Il est possible de définir des fonctions dans `methods` qui gèrent le style d'un élément HTML

```
getColor() {  
  return 'white';  
},  
getBgColor() {  
  return 'red';  
}
```

Pour afficher le contenu de l'attribut `message` dans le template avec une couleur de fond rouge

```
<p :style="{ color: getColor(), backgroundColor: getBgColor() }">
  {{ message }}
</p>
```

Vue.js

Définissons `getStyle()` **dans** `methods`

```
getStyle() {  
  return { color: 'white', backgroundColor: 'red'  
  };  
}
```

© Achref EL ME

Vue.js

Définissons `getStyle()` **dans** `methods`

```
getStyle() {  
  return { color: 'white', backgroundColor: 'red'  
    };  
}
```

Nous pouvons attribuer une méthode à `:style`

```
<p :style="getStyle()">  
  {{ message }}  
</p>
```

Vue.js

Définissons l'objet `styles` dans la fonction `data`

```
styles : {  
  color: 'white',  
  backgroundColor: 'red'  
}
```

© Achref EL ME

Vue.js

Définissons l'objet `styles` dans la fonction `data`

```
styles : {  
  color: 'white',  
  backgroundColor: 'red'  
}
```

Nous pouvons attribuer un objet à `:style`

```
<p :style="styles">  
  {{ message }}  
</p>
```

Définissons l'objet `fonts` dans la fonction `data`

```
fonts: {  
  fontWeight: 'bold'  
}
```

© Achref EL MOUADJID

Vue.js

Définissons l'objet `fonts` dans la fonction `data`

```
fonts: {  
  fontWeight: 'bold'  
}
```

Nous pouvons aussi attribuer un tableau à `:style`

```
<p :style="[styles, fonts]">  
  {{ message }}  
</p>
```

Évènement (Event)

- Action appliquée par l'utilisateur ou simulée par le développeur sur un élément **HTML**.
- Évènement déclenché \Rightarrow Fonction, définie dans la partie `methods`, exécutée.

© Achref EL MOU

Vue.js

Évènement (Event)

- Action appliquée par l'utilisateur ou simulée par le développeur sur un élément **HTML**.
- Évènement déclenché \Rightarrow Fonction, définie dans la partie `methods`, exécutée.

Event binding : `v-on`

- Objectif : associer un évènement à une fonction, définie dans la partie `methods`.
- Il suffit de préfixer l'évènement par '`v-on`'.

Vue.js

Modifions la méthode `direBonjour()` **déjà définie dans la partie** `methods`.

```
direBonjour() {  
  alert('bonjour Vue.js 3');  
}
```

© Achref EL MOUELHI

Vue.js

Modifions la méthode `direBonjour()` **déjà définie dans la partie** `methods`.

```
direBonjour() {  
  alert('bonjour Vue.js 3');  
}
```

Pour exécuter la méthode `direBonjour()` **lorsqu'on clique sur un bouton**

```
<button v-on:click="direBonjour()">Dire Bonjour</button>
```

Vue.js

Modifions la méthode `direBonjour()` **déjà définie dans la partie** `methods`.

```
direBonjour() {  
  alert('bonjour Vue.js 3');  
}
```

Pour exécuter la méthode `direBonjour()` **lorsqu'on clique sur un bouton**

```
<button v-on:click="direBonjour()">Dire Bonjour</button>
```

N'oublions pas de commenter

```
<p>{{ direBonjour() }}</p>
```

On peut aussi remplacer `v-on` par le raccourci `@`

```
<button @click="direBonjour()">Dire Bonjour</button>
```

© Achret

Exercice

- Créer deux boutons **HTML** dont l'un est initialement désactivé (`disabled`)
- En cliquant chaque fois sur le bouton activé, ce dernier se désactive et l'autre s'active

Solution

```
<button :disabled="disabled" @click="() => disabled=!disabled">
  bouton 1
</button>

<button :disabled="!disabled" @click="() => disabled=!disabled">
  bouton 2
</button>
```

© Achref

Vue.js

Solution

```
<button :disabled="disabled" @click="() => disabled=!disabled">
  bouton 1
</button>

<button :disabled="!disabled" @click="() => disabled=!disabled">
  bouton 2
</button>
```

Sans oublier de déclarer `disabled` dans `data`

```
disabled: true
```

Remarque

On peut utiliser l'objet `event` pour récupérer des informations sur l'évènement **JavaScript**.

Vue.js

Considérons la zone de saisie suivante dans laquelle nous voudrions uniquement accepter des lettres en minuscule

```
<div>  
  <label for="texte"> Merci de bien saisir un texte :  
</label>  
  <input type="text" id="texte" @input="showValue" />  
</div>
```

© Achref EL MOU

Vue.js

Considérons la zone de saisie suivante dans laquelle nous voudrions uniquement accepter des lettres en minuscule

```
<div>
  <label for="texte"> Merci de bien saisir un texte :
</label>
  <input type="text" id="texte" @input="showValue" />
</div>
```

Pour connaître le caractère saisi, utilisons l'objet `event` dans `showValue()`

```
showValue(event) {
  console.log(event.target.value);
  // affiche tous les caractères saisis

  console.log(event.data);
  // affiche le dernier caractère saisi
}
```

Question

Et si la fonction envoyait des arguments (paramètres) ?.

Considérons la zone de saisie suivante dans laquelle nous voudrions uniquement accepter des lettres en minuscule

```
<div>
  <label for="texte"> Merci de bien saisir un texte :
</label>
  <input type="text" id="texte" @input="showValue('a', $event)" />
</div>
```

© Achref EL M...

Vue.js

Considérons la zone de saisie suivante dans laquelle nous voudrions uniquement accepter des lettres en minuscule

```
<div>
  <label for="texte"> Merci de bien saisir un texte :
</label>
  <input type="text" id="texte" @input="showValue('a', $event)" />
</div>
```

Pour connaître le caractère saisi, utilisons l'objet `event` dans `showValue()`

```
showValue(str, event) {
  console.log(str, event.target.value);
  console.log(str, event.data);
}
```


Exercice

- 1 Déclarer une zone de saisie dans le template.
- 2 La valeur saisie doit s'afficher à coté de la zone de saisie à chaque modification.

Solution

```
<div>
  <label for="message">Message : </label>
  <input type="text" id="message" @input="updateMessage">
    {{ message }}
</div>
```

© Achref EL M...

Vue.js

Solution

```
<div>
  <label for="message">Message : </label>
  <input type="text" id="message" @input="updateMessage">
  {{ message }}
</div>
```

La fonction `updateMessage` à déclarer dans `methods`

```
updateMessage(event) {
  this.message = event.target.value;
}
```

Autres évènements

```
<input type="text" @keyup.enter="showValue()">
```

```
<!-- s'exécute en cliquant sur entrée -->
```

```
<input type="text" @keyup.alt.enter="showValue()">
```

```
<!-- s'exécute en cliquant sur alt + enter -->
```

```
<input type="text" @keyup.up="showValue()">
```

```
<!-- s'exécute en cliquant sur flèche vers le haut -->
```

Two way binding : définition

Un changement de valeur dans le script sera aperçu dans le template et inversement (un changement dans le template sera reçu dans le script).

© Achref EL MOU

Vue.js

Two way binding : définition

Un changement de valeur dans le script sera aperçu dans le template et inversement (un changement dans le template sera reçu dans le script).

Two way binding : comment

Pour la liaison bidirectionnelle, il nous faut la propriété `v-model`.

Vue.js

Le template

```
<div>
  <label for="message">Message : </label>
  <input type="text" id="messsage" v-model="message">
    {{ message }}
</div>
```

© Achref EL ME

Vue.js

Le template

```
<div>
  <label for="message">Message : </label>
  <input type="text" id="message" v-model="message">
    {{ message }}
</div>
```

Constat

Pas besoin de bouton pour envoyer la valeur saisie dans le champ texte, elle est systématiquement mise à jour dans la partie script quand elle est modifiée dans la vue.

Watchers

Concept permettant de surveiller l'état d'une variable :
changement d'état \Rightarrow exécution d'une fonction.

Vue.js

Ajoutons `watch` dans la partie `script`

```
watch: {  
  message(nouvelle, ancienne) {  
    console.log(nouvelle, ancienne)  
  }  
}
```

© Achref EL MOUL

Ajoutons `watch` dans la partie `script`

```
watch: {  
  message(nouvelle, ancienne) {  
    console.log(nouvelle, ancienne)  
  }  
}
```

Explication

- `nouvelle` : paramètre contenant la nouvelle valeur de l'attribut `message`.
- `ancienne` : paramètre contenant l'ancienne valeur de l'attribut `message`.
- Un watcher concerne un seul attribut et un attribut peut avoir un seul watcher.