Methods -Propriétés calculés – Liaison de classe

Objectifs

- Utilisation de
  - Computed
  - Method
  - Class binding

TP 2a

- Créer une fonction de tri et l'utiliser

```
export default{
    data(){
        return{
            company_name: "Sahaza Group",
            //topics: ['introduction', 'théorie', 'pratique']
            topics:[
                {'name': 'introduction', 'duration': 5},
                {'name': 'théorie', 'duration': 15},
                {'name': 'pratique', 'duration': 63}
            ]
        }
    },
    computed: {
        durationSortedByDurationDesc(){
            return this.topics.sort((a, b) => b.duration - a.duration);
        }
    }
}
```

```
<li v-for="(topic,index) in durationSortedByDurationDesc" :key="index">
```

TP 2a

- Explication
  - **this.topics** : Cela fait référence à une liste (ou un tableau) d'objets appelée **topics**, où chaque objet possède une propriété **duration**.
  - **sort((a, b) => b.duration - a.duration)** : La méthode **sort()** prend une fonction de comparaison en argument. Cette fonction compare deux éléments, **a** et **b**, pour déterminer leur ordre.
  - **b.duration - a.duration** : La comparaison soustrait la durée de **a** de celle de **b**. Cela entraîne un tri par ordre décroissant :
    - Si le résultat est positif (durée de **b** plus grande que **a**), **b** est placé avant **a**.
    - Si le résultat est négatif (durée de **a** plus grande), **a** est placé avant **b**.
    - Si le résultat est 0, les éléments sont considérés comme égaux et leur ordre reste inchangé.

TP 2b

- Mettre un CSS différent sur la durée la plus longue
- Etape 1 : mettre un css sur les « li »

```
<style scoped>
li{
    border: 2px solid blue;
}
</style>
```

- Etape 2 : Mettre une classe « maxDuration » sur tous les « li »

```
<ul>
    <li class="maxDuration" v
</ul>
</template>

<style scoped>
li.maxDuration{
    border: 2px solid blue;
}
</style>
```

TP 2b

- Créer une fonction qui retourne true si c'est la durée maximum

```
methods: {
    isMaxDuration(topic){
        return topic.duration == this.durationSortedByDurationDesc[0].duration;
    }
}
```

- Utiliser la fonction dans le binding de class

```
<li :class="{'maxDuration': isMaxDuration(topic)}" v-for="(topic,index) in
durationSortedByDurationDesc" :key="index">{{ topic.name }} - {{ topic.duration }}mn</li>
```

En résumé

- Computed: données calculées
- Methods : fonctions associées à un composant
- Class binding : rajouter des classes à un élément

```
    computed: {
        durationSortedByDurationDesc(){
            return this.topics.sort((a, b) => b.duration - a.duration);
        }
    },
    methods: {
        isMaxDuration(topic){
            return topic.duration == this.durationSortedByDurationDesc[0].duration;
        }
    }
}
</script>

<template>
<h1>Formation vue js - <small> {{ company_name }} </small></h1>
<ul>
    <li :class="{'maxDuration': isMaxDuration(topic)}" v-for="(topic,index) in
    durationSortedByDurationDesc" :key="index">{{ topic.name }} - {{ topic.duration }}mn</li>
</ul>
</template>

<style scoped>
li.maxDuration{
    border: 2px solid blue;
}
</style>
```

Methods

Définition

- Un composant a besoin de fonctions
  - Methods : en option API
  - Fonction simple : en composition API
- Methods a accès aux datas et computed
- Mis en évidence des produits moins chers

Déclarations

Option API

- Mettre dans « methods »

```
<script>
export default {
  data() {
    return {
      count: 0
    }
  },
  methods: {
    increment() {
      this.count++
    }
  }
}
</script>
```

Composition API

- Fonction JS simple

```
function calculateBooksMessage() {
  return author.books.length > 0 ? 'Yes' : 'No'
}
```

Appel de fonctions

- Dans une autre fonction

```
<script>
export default {
  data() {
    return {
      count: 0
    }
  },
  methods: {
    increment() {
      this.add(1)
    },
    add(number_to_add) {
      this.count += number_to_add
    }
  }
}
</script>
```

- A partir d'un évenement

```
<button @click="increment">{{ count }}</button>
```

Asynchrone

- On peut faire de l’asynchrone (fetch par exemple) dans les méthodes
- Ce n’est pas recommandé dans les autres éléments du composant (

Computed

Définition

- Données calculées : variables qui dépendent d'autres variables
- Exemple
  - Tableau filtré -> on part d'un tableau existant
  - Somme des produits d'un panier -> somme d'autres variables
- Mise à jour automatique si les variables d'origines changent

Déclarations

Option API

- Mettre dans « computed »

```
computed: {
  // a computed getter
  publishedBooksMessage() {
    // `this` points to the component instance
    return this.author.books.length > 0 ? 'Yes' : 'No'
  }
}
```

Composition API

- Importer computed

```
import { reactive, computed } from 'vue'


// a computed ref
const publishedBooksMessage = computed(() => {
  return author.books.length > 0 ? 'Yes' : 'No'
})
```

Exemples (option API)

```vue
<script>
export default {
  data() {
    return {
      author: {
        name: 'John Doe',
        books: [
          'Vue 2 - Advanced Guide',
          'Vue 3 - Basic Guide',
          'Vue 4 - The Mystery'
        ]
      }
    }
  },
  computed: {
    publishedBooksMessage() {
      return this.author.books.length > 0 ? 'Yes' : 'No'
    }
  }
}
</script>

<template>
  <p>Has published books:</p>
  <span>{{ publishedBooksMessage }}</span>
</template>
```

Has published books:

Yes

Exemples (composition API)

```vue
<script setup>
import { reactive, computed } from 'vue'

const author = reactive({
  name: 'John Doe',
  books: [
    'Vue 2 - Advanced Guide',
    'Vue 3 - Basic Guide',
    'Vue 4 - The Mystery'
  ]
})

// a computed ref
const publishedBooksMessage = computed(() => {
  return author.books.length > 0 ? 'Yes' : 'No'
})
</script>

<template>
  <p>Has published books:</p>
  <span>{{ publishedBooksMessage }}</span>
</template>
```

Has published books:

Yes

Computed VS Methods

Computed

```
computed: {
  // un accesseur calculé
  publishedBooksMessage() {
    // `this` pointe sur l'instance du composant
    return this.author.books.length > 0 ? 'Yes' : 'No'
  }
}
```

```
<span>{{ publishedBooksMessage }}</span>
```

Methods

```
methods: {
  calculateBooksMessage() {
    return this.author.books.length > 0 ? 'Yes' : 'No'
  }
}
```

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

- Cache

Erreurs à ne pas faire

- Ne pas utiliser de l'asynchrone dans computed
- Computed : Lire des données mais ne modifie pas les données

Class binding

Définition

- Rajouter des classes de manières dynamiques
- Utilisation de v-bind :class ou directement :class
- Exemple : on retourne isActive si active est true

```
<div :class="{ active: isActive }"></div>
```

- On passe un objet où
  - Clé est la classe
  - Valeur est la condition
- On peut passer plusieurs éléments

```
<div :class="{ active: isActive, 'text-danger': hasError }"></div>
```

Utilisation avec class

- On peut utiliser en même temps class et :class

```
<div
  class="static"
  :class="{ active: isActive, 'text-danger': hasError }"
></div>
```

```
<div class="static active"></div>
```

Stocker l’objet dans data ou computed

Data

```js
data() {
  return {
    classObject: {
      active: true,
      'text-danger': false
    }
  }
}
```

```template
<div :class="classObject"></div>
```

Computed

```js
data() {
  return {
    isActive: true,
    error: null
  }
},
computed: {
  classObject() {
    return {
      active: this.isActive && !this.error,
      'text-danger': this.error && this.error.type === 'fatal'
    }
  }
}
```

```template
<div :class="classObject"></div>
```

Utiliser style

```js
data() {
  return {
    activeColor: 'red',
    fontSize: 30
  }
}
```

```template
<div :style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

```js
data() {
  return {
    styleObject: {
      color: 'red',
      fontSize: '13px'
    }
  }
}
```

```template
<div :style="styleObject"></div>
```

Utiliser un tableau pour le bind

```
data() {
  return {
    activeClass: 'active',
    errorClass: 'text-danger'
  }
}
```

```template
<div :class="[activeClass, errorClass]"></div>
```