Documentation of coding.

Code documentation is the process through which software programmers document their code. It is a combination of clear pictures and textual explanations that describe what a codebase does and how it may be used.

01_04 Rendering Lists

v-for directive, Getting an Index

1. Use `v-for` to repeat 10 times

```
<div v-for="myNumber in 10"></div>
```

2. Modify to display `products`

```
v-for="(myAlias, myIndex) in myArray"
```

3. Display item number with `index`

```
<div v-for="(myAlias, myIndex) in myArray">

  :

</div>
```

4. Pass `products.id` as a `key`

```
<div v-for="myAlias in myArray" :key="myAlias.id"></div>
```

01_05 Conditional Statements

Family of conditional statements: v-if v-else v-else-if v-show

1. `v-if` to show `products` cheaper than `$99`

```
<span class="badge bg-success" v-if="item.price>=90">premier</span>
```

2. Use `v-else` and `v-else-if` to show badges depending on price

```
<span class="badge bg-primary" v-else-if="item.price<90 && item.price>10">value</span>
```

02_01 User Input

Update data on user input

1. Use `v-model` to control max price

   <input class="form-control" id="max-price">

2. Add a `.number` filter

```
<input v-model.number="max" class="form-control" id="max-price" >

<input v-model="max" type="range" class="form-range" min="0" max="130" >
```

3. Create a `displayLabels` boolean

   ```
   <input v-model="displayLabels" class="form-check-input" type="checkbox" id="showLabels">
   ```

   ```
               displayLabels: true,
   ```

4. Modify code to show/hide labels

```
        <span class="badge bg-success" v-if="item.price>=90 && displayLabels">premier</span>

        <span class="badge bg-primary" v-else-if="item.price<90 && item.price>10 &&
displayLabels">value</span>

        <span class="badge bg-danger" v-else-if="item.price<=10 && displayLabels">sale</span>

        ${{item.price}}</div>
```

```
 <input v-model.number="max" class="form-control" id="max-price" >
```

02_02 Computed Content

Show all prices on two decimals and display count based on set value “max” and show label with selected qty value.

1. Create a currency filter

```
 methods: {

   currency(value) {
```

```
    return `$${Number.parseFloat(value).toFixed(2)}`;

  }

}
```

2. Display count based on `max`

```
computed: {

  filteredProducts() {

    return this.products.filter( item => (item.price < this.max))

  }

},
```

3. Use filtered array on `v-for`

```
<div v-for="(item, index) in filteredProducts" :key="item.id" id="item-list" class="row
align-items-center">
```

4. Show label with qty based on price selection

```
<div class="badge bg-success ml-3">results: {{filteredProducts.length}}</div>
```

02_03 Event Handling

You can add one or more modifiers that control how the events are handled.

1. Create a `cart` array

```
    cart: [],
```

2. Click `plus` to add items

```
    <button @click="addToCart(item)" class="btn btn-success">+</button>
```

```
methods: {

  addToCart(products) {

    this.cart.push(products)

  },
```

```
computed: {

cartTotal() {

    return this.cart.reduce((inc, item)=> Number(item.price) + inc, 0)

  }
```

```
      <span class="font-weight-bold bg-white">{{currency(cartTotal)}}</span>
```

3. Display cart items

```
    <div v-if="displayCart" class="list-group" aria-labelledby="cartDropdown">
```

4. Make cart icon toggle menu

```
       <button

        @click="displayCart= !displayCart"
```

02_04 Lifecycle Hooks

Question use different hooks in data cycle
Events
beforeCreate created beforeMount mounted beforeUpdate updated activated

Sometimes you'll need to take care of things at **different points in the life of a Vue application**, and to do that, you use **LifeCycle hooks.**

1. Clear out `products`
2. Create a `lifecycle hook`
3. Use javascript's [Fetch API]

```
created() {

  fetch("https://hplussport.com/api/products/order/price")

  .then(response => response.json())

  .then(data => {

    this.products = data
```

```
})
```

4. Load from
   `https://hplussport.com/api/products/order/price`
5. Try different hooks

03_01 Using Inline Styles

Passing object not attributes

1. Change + button radius to circle

```
<button :style="{borderRadius:'50%'}" @click="addToCart(item)" class="btn btn-success">+</button>
```

2. Add a black border to + button

```
    <button :style="{borderRadius:'50%', border:'1px solid black'}" @click="addToCart(item)"
class="btn btn-success">+</button>
```

3. Modify cart background when cartTotal >=100

```
methods: {

  addToCart(products) {

    this.cart.push(products);

    if (this.cartTotal >=100) {this.warningObject.backgroundColor = 'red'}

  },
```

03_02 Using CSS Classes

Connect classes to variables.
You can animate using CSS styles and classes. https://animate.style/

1. Modify `cart` button to use classes

```
      <button

        :style="warningObject"

        @click="displayCart= !displayCart"

        class="btn btn-sm ml-3"

        :class="cartBtn"
```

2. Use a variable to control classes
3. Try a computed property

```
computed: {

  cartBtn() {

    return {

      'btn-secondary': this.cartTotal <=100,

      'btn-success': this.cartTotal >100,

      'btn-danger': this.cartTotal >200

    }
```

03_03 Transitions

1. Animate the `cart` dropdown
2. Use the `<transition>` tag

```
    <transition name="dropdown"

                @name="transitonColor"

                @after-leave="resetColor">
```

3. Add `name` property to the transition

```
methods: {

  transitionColor(el){

    this.totalColor = 'text-danger';

  },

  resetColor() {

    this.totalColor = 'text-secondary';

  },
```

4. Make `total` red on dropdown

03_04 Transitions Lists

1. `transition-group` for products

```
<transition-group name="products" appear>
```

2. Use the `appear` option
3. Use different `from`/`to` transforms

04_01 Components

```
const App = Vue.createApp({})
App.component('name', { template: `template` })
App.mount('#App')
```

1. Rewrite `App` calls

```
App.mount('#app')
```

2. Create `Currency` Component
3. Replace template calls

```
          <div class="ml-3 font-weight-bold"><curr :amt="item.price"></curr></div>
```

```
const App = Vue.createApp({
  data() { return { myVar: 'myValue' } }
})

App.component('myComponent', {
  props: ['myProp'], template: ``
})
App.mount('#App')
```

```
App.component('curr', {

 props: ['amt'],

 template: `{{dollar(amt)}}`,

 methods: {

   dollar(value) {

     return '$' + Number.parseFloat(value).toFixed(2);

   }
```

```
 }

})



App.mount('#app')
```

04_02 Components Events

When you create an event inside a component, it can trigger something in the parent and passes along some information as well.

1. Create a `product` component

```
App.component('product', {

 props: ['item'],

 emits: ['addToCart'],

 template: `

   <div class="col-2 m-auto">

     <button @click="$emit('addToCart', item)" class="btn btn-success">+</button>

   </div>

   <div class="col-sm-4">

     <img class="img-fluid d-block" :src="item.image" :alt="item.name">

   </div>

   <div class="col">

    <h3 class="text-primary">{{item.name }}</h3>

    <p class="mb-0">{{ item.description }}</p>

    <div class="h5 float-right">

      <span class="label"></span><curr :amt="item.price"></curr></div>

   </div>

 `


})
```

2. Emit an `addToCart` method

04_03 Slots

1. Create a `custom-alert` component
2. Include option to specify color
3. Include option to close alert
4. Add alert when `cartTotal > 100`

```
<custom-alert type="danger" close='true' v-if="cartTotal>100">
```