Previ

1. Template literals:

```
var a = 1;
var s = `hola ${a+1}
  <--- aquests espais`;
console.log(s);
```

☞ No confondre, els string literals "de tota la vida" (que s'escriuen amb tick: `'`), amb els template literals (que s'escriuen amb backtick: `` ` ``).
☞ Interpolació
☞ Multi-línia (compte amb els espais al començar una línia).

Introducció a Vue.js

1. Que és Vue.js? **Un framework de frontend web.** Framework és un terme difús: "it is just a library", "you call a library, a framework calls you", "re-usable design", etc. Frontend: "del costat del client" (navegador).
2. Perquè triem Vue.js?
3. És pot fer servir de força maneres, però mentre anem agafant una idea del que es pot fer, ens podem imaginar que tenim una single-page application (SPA) que s'executa al client i accedeix a les dades a traves d'una API REST. Aquesta SPA la podem fer amb Vue.js.

   ☞ `https://v3.vuejs.org/guide/`
4. La SPA serà un arxiu html i un parell d'arxius de javascript.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>SPA</title>

    <!-- Això "importa" el vue.js -->
    <script src="vue.js"></script>
  </head>
  <body>
    <h1>SPA</h1>

    <!-- Aquí és on es veurà la nostra aplicació -->
    <div id="app"></div>

    <!-- En aquest arxiu programarem la SPA -->
    <script src="app.js"></script>
  </body>
</html>
```

Notes:

- Fer servir *sempre* UTF-8.
- Podem fer servir
  `<script src="https://unpkg.com/vue@3/dist/vue.global.prod.js"></script>`
  en comptes de
  `<script src="vue.js">`.
- Fer servir **`https://unpkg.com/vue@3/dist/vue.global.js`** per development.

- El del CDN està catchejat però és una dependència externa (+ tema privacitat).
- El **app.js** s'ha d'incloure just abans del **</body>**.

5. Al **app.js** creem una instancia de Vue.

```
let options = {
  template: '<span>Hello World</span>',
}

var app = Vue.createApp(options);
app.mount("#app") // document.getElementById('app')
```

☞ En podem fer tantes com vulguem.

6. Mirem que ha passat...

```
<body>
    <h1>SPA</h1>
    <div id="app" data-v-app="">
        <span>Hello World</span>
    </div>
    <script type="text/javascript" src="app.js"></script>
</body>
```

☞ Perquè el vue.js va just abans del **</body>**?

7. Hi ha un conjunt de dades "reactives" (que quan canvien, les coses que depenen d'aquestes dades també canvien). Es defineixen de la següent forma:

```
const options = {
  data: function() {
    return { a: 3, b: 4 }
  },
  template: '<span>Hello World</span>',
}

const app = Vue.createApp(options)
const vm = app.mount("#app")

console.log(vm.a);
vm.c = 5
console.log(vm.c);
```

☞ Com que data és una funció, podem fer servir les opcions a multiples llocs.
☞ Malauradament, **c** no és una variable reactiva.

8. Aquestes dades reactives es poden *interpolar* als templates:

```
const options = {
  data: function() { // també "data() {" directament
    return { a: 3, b: 4 }
  },
  template: '<span>Hello World {{a}}</span>',
}

const vm = Vue.createApp(options).mount("#app")
```

☞ Mustaches. No són template literals de javascript.

9. I quan canvien aquestes dades, també canvien les interpolacions.

```
setTimeout(() => vm.a = 4, 1000);
```

☞ Aquest és el concepte de *data binding*: lligar les dades amb la UI.

10. Les interpolacions són *expresions* de javascript, en un scope 'especial' on hi ha les dades reactives.

```
 template: '<span>{{a}} {{a * 4}}</span>',
```

☞ Expresions != instruccions. Només si ho pots posar entre parèntesis és una expressió. Això no es pot fer i no és una expressió: `(if(true){ })`.

☞ No confondre amb les interpolacions de javascript.

11. Les interpolacions no es poden fer servir (directament) a dins dels tags html.

```
<input type="text" value="{{a}}"><!-- No! -->
<input type="text" value="a"><!-- Això tampoc és el que volem -->
```

12. Per això existeixen les directives. Són atributs de tags html que comencen per **v-**.

```
<input type="text" v-bind:value="a">
```

Això vol dir: lliga el valor **vm.a** a l'atribut **value** del tag **input**. Si canvia **a**, canvia **value**.

13. Fins ara hem fet *one-way data binding*, però és pot fer *two-way data binding* amb la directiva **v-model**.

```
<input type="text" v-model="a">
```

14. Fem un exemple senzill: un convertidor de litres a cm^3.

```
const options = {
  data() {
    return { litres: 0 }
  },
  template: `<div>
    Litres: <input type="text" v-model="litres"><br />
    cm^3: <input type="text" v-bind:value="litres * 1000">
  </div>`,
};

Vue.createApp(options).mount("#app")
```

15. Recapitulem els conceptes que hem vist:
    - Single-page application (SPA),
    - Templates,
    - Interpolació,
    - Reactivitat,
    - Data binding (one way, two way),

- Directives v-bind i v-model.

16. *Methods*: i si volem fer la conversió fent servir una funció?

```
const options = {
  data() {
    return { litres: 0 }
  },
  methods: {
    litresToCm3: function(litres) {
      return litres * 1000;
    }
  },
  template: `<div>
    Litres: <input type="text" v-model="litres"><br />
    cm^3: <input type="text" v-bind:value="litresToCm3(litres)">
  </div>`,
}
```

Aquesta funció també pot accedir a les dades reactives...

```
const options = {
  data() {
    return { litres: 0 }
  },
  methods: {
    litresToCm3: function() {
      return this.litres * 1000;
    }
  },
  template: `<div>
    Litres: <input type="text" v-model="litres"><br />
    cm^3: <input type="text" v-bind:value="litresToCm3()">
  </div>`,
}
```

☞ Compte amb fer servir arrow functions als methods, que no tenen `this` propi!

17. Podem fer servir *Watchers* per vigilar quan es modifica una dada reactiva.

```
const options = {
  data() {
    return { litres: 0 }
  },
  methods: {
    litresToCm3: function() {
      return this.litres * 1000;
    }
  },
  watch: {
    litres: function(new_value, old_value) {
      console.log(new_value);
    }
  },
  template: `<div>
    Litres: <input type="text" v-model="litres"><br />
```

```
    cm^3: <input type="text" v-bind:value="litresToCm3()">
  </div>`,
};
```

Ara ja podem fer la conversió en les dues direccions.

```
const options = {
  data() {
    return { litres: 0, cm3: 0 }
  },
  watch: {
    litres: function(litres) {
      this.cm3 = this.litres * 1000;
    },
    cm3: function(cm3) {
      this.litres = this.cm3 / 1000;
    }
  },
  template: `<div>
    Litres: <input type="text" v-model="litres"><br />
    cm^3: <input type="text" v-model="cm3">
  </div>`,
};
```

18. Events (esdeveniments). Per exemple amb la directiva **`v-on:click`**.

```
<button v-on:click="litres = 0">Reset</button>
```

19. Renderitzat condicional. Per exemple amb la directiva **`v-if`**.

```
<button v-if="litres != 0" v-on:click="litres = 0">Reset</button>
```

20. Un altre exemple: el teclat virtual.

```
const options = {
    data() {
        return { buffer : "" }
    },
    template: `<div style="border: solid">
      {{buffer}} <br>
      <button v-on:click="buffer += 'A'">A</button>
      <button v-on:click="buffer += 'B'">B</button>
      <button v-on:click="buffer += 'C'">C</button>
    </div>`,
};
```

☞ `<button v-on:click="buffer += &quot;A&quot;">A</button>`.

21. Hi afegim un mètode.

```
const options = {
    data() {
        return { buffer : "" }
```

```
    },
    methods: {
      append: function(letter) {
        this.buffer += letter;
      },
    },
    template: `<div style="border: solid">
      {{buffer}} <br>
      <button v-on:click="append('A')">A</button>
      <button v-on:click="append('B')">B</button>
      <button v-on:click="append('C')">C</button>
    </div>`,
};
```

22. Afegim un botó per esborrar.

```
backspace: function() {
  this.buffer = this.buffer.slice(0,-1);
}
```

```
<button v-on:click="backspace">[DEL]</button>
```

☞ **backspace** o **backspace()** ? Els dos funcionen igual, però no són el mateix!

23. Podem fer servir la directiva **v-for**.

```
data() {
  return { buffer : "", letters: ['A', 'B', 'C', 'D', 'Z'] }
}
```

```
<button v-for="letter in letters"
  v-on:click="append(letter)">{{letter}}</button>
```

24. Que més hem vist?

    - Mètodes,
    - Watchers,
    - Events (**v-on**),
    - Renderitzat condicional (**v-if**, **v-for**).

Components i detalls

1. En episodios anteriores... Per recapitular veiem un exemple nou: un cronometre.

```
const options = {
    data(): () => ({ time : 0 }),
    template: `<div>{{time}}</div>`,
};
const app = Vue.createApp(options);
const vm = app.mount("#app");
setInterval(()=>vm.time++, 1000);
```

2. El concepte de component: una manera d'encapsular elements de la UI i reutilitzar-los.

3. Per exemple, volem poder fer servir aquest cronometre moltes vegades.
Opció 1: fem múltiples instancies de vue.
Opció 2: fem un **<fancy-chronometer>** component que puguem fer servir.

```
const options = {
  template: `<div>
    <fancy-chronometer></fancy-chronometer>
    <fancy-chronometer></fancy-chronometer>
  </div>`,
}
```

4. Com fem el component? **app.component('fancy-chronometer', options)**, on **options** te la mateixa forma que veniem fent servir fins ara per les instancies de vue.

```
Vue.component('fancy-chronometer', {
  data: function() {
    return { time : 0 };
  },
  template: `<div>{{time}}</div>`,
});
```

☞ A partir de que s'ha *registrat* aquest component, ja es pot fer servir a tots els templates de vue.

5. Detall: ja no hi ha la variable **vm** per accedir al component des de fora.

```
created: function() {
  setInterval(()=>this.time++, 1000);
}
```

6. Li afegim un boto de reset.

```
<div style="display:inline-block; padding:10px; margin:10px; border:solid;">
  {{time}}
  <button v-on:click="time=0">Reset</button>
</div>
```

☞ Veiem que efectivament els components funcionen de forma independent.

7. Com podem configurar aquests components que hem fet? O dit d'una altra manera com hi passem informació cap a dintre? ⇒ amb l'opció **props** (de *properties*).

8. Per exemple, volem afegir una etiqueta (diferent) a cada cronometre.

```
<fancy-chronometer label="Crono 1">
```

Afegim a les opcions:

```
props: ['label'],
```

I a la plantilla:

```
{{label}}: {{time}}
```

9. O millor

```
<span v-if="label">{{label}}: </span>{{time}}
```

☞ Atenció: les props van en *camelCase* al javascript (dins del component) i en *kebab-case* al html (fora del component). I.e., **chronoLabel** ⇒ **chrono-label**.

☞ "Culturilla": Hi ha també la *PascalCase* i la *snake_case*.

☞ Les props són per entrar dades al component, les hauriem de fer servir com a read only.

10. També es pot fer binding de les props. Per exemple:

```
data: { label2: "L2" },
```

```
<fancy-chronometer v-bind:chrono-label="label2"></fancy-chronometer>
```

11. Hem vist com entrar dades als components, però com les podem treure?

☞ Farem un textbox que tindra un event que saltarà quan el text sigui 'yes'.

```
const RootComponent = {
  methods: {
    something: function() {
      console.log('yes!!!!');
    }
  },
  template: `<comp v-on:text-is-yes="something()"></comp>`,
};
const vm = Vue.createApp(RootComponent).mount('#app');
```

```
app.component('comp', {
  data: function() {
    return { text: "" };
  },
  watch: {
    text: function() {
      if (this.text == "yes") {
        this.$emit('text-is-yes');
```

```
      }
    }
  },
  template: `<input v-model="text">`,
});
```

12. Continuem amb l'exemple anterior: canviarem el boto de reset per un boto de seguretat. Aquest boto substituirà el botó de reset del cronometre, i tindrà un checkbox d'activar i desactivar.
13. El farem servir així:

```
<secure-button v-on:secure-click="time=0">Reset</secure-button>
```

14. I vindrà a tindre una estructura (bàsica) com la que segueix:

```
app.component('secure-button', {
  data: function() {
    return { /* */ };
  },
  emits: [ 'secure-click' ],
  template: `<div style="display:inline-block; padding:10px;
      margin:10px; border:solid red;">
    <button>Click-me</button>
    </div>`,
});
```

15. Fem que faci alguna cosa al fer click:

```
<button v-on:click="this.$emit('secure-click');"
```

16. Ara que ja ens funciona l'event cap a l'exterior, ja podem afegir la resta de funcionalitat al boto:

```
app.component('secure-button', {
  data: function() {
    return { enabled : false };
  },
  methods:{
    clickMethod() {
      this.enabled = false;
      this.$emit('secure-click');
    }
  },
  emits: [ 'secure-click' ],
  template: `<div style="display:inline-block; padding:10px;
      margin:10px; border:solid red;">
    <button v-bind:disabled="! enabled" v-on:click="clickMethod();">
      Click-me</button>
    <label><input type="checkbox" v-model="enabled">Enable</label>
    </div>`,
});
```

17. Recapitulem els conceptes que hem vist de components:
    - S'han de *registrar*. `app.component('nom-del-component', options);`

- Les **options** són com a **Vue.createApp**.
- Per entrar dades cal un atribut **input-value** al tag html i una *prop* **inputValue** a l'opció **props** a les **options**.
- Per treure events cal emetre'ls amb **$emit('nom-del-event')** dins i capturar-los a fora amb un **v-on:nom-del-event**. "Cal" un *event* **nom-del-event** a l'opció **emits** a les **options**.

18. Seguim millorant l'exemple que teniem...
19. Podem fer servir el tag **<slot>** per passar cap a dintre els elements html de dins del tag del component. És a dir, podem canviar el **Click-me** per un **<slot></slot>** per mostrar el text del botó.
20. Volem treure un valor de dins del component...

```
this.$emit('secure-click', 'yes');
```

```
<secure-button v-on:secure-click="$event == 'yes'? time=0 : '' ">
```

☞ El segón parametre de **$emit** és el valor de l'event i a fora del component el trobem a **$event**.

21. Ara que ja sabem com fer entrar i sortir dades d'un component... **v-model** fa el mateix que:

```
<component v-bind:modelValue="variable" v-on:update:modelValue="variable = $event">
```

```
<component v-model="variable">
```

☞ Aquí només cal entendre que **v-model** és una combinació de prop i event. Quan calqui us mireu els detalls.

22. Un altre exemple: redundant checkbox

```
const RootComponent = {
    data: function() {
        return { checkbox: false },
    },
    template: `<div>
      <redundant-checkbox v-model=checkbox></redundant-checkbox>{{checkbox}}
    </div>`,
}

const RedundantCheckbox = {
  data: function() {
      return {
          checkbox1: false,
          checkbox2: false,
       };
  },
  props: ['modelValue'],
  emits: ['update:modelValue'],
  template: `<span>
    <input type=checkbox v-model=checkbox1>
    <input type=checkbox v-model=checkbox2>
  </span>`,
  watch: {
    checkbox1: function() {
```

```
      this.$emit('update:modelValue', this.checkbox1 && this.checkbox2);
    },
    checkbox2: function() {
      this.$emit('update:modelValue', this.checkbox1 && this.checkbox2);
    },
  },
  created: function() {
      this.checkbox1 = this.value;
      this.checkbox2 = this.value;
  },
}

app.createApp(RootComponent)
app.component('redundant-checkbox', RedundantCheckbox);
app.mount('#app');
```

23. Que hem vist avui?
    - Components,
    - Props i events, i
    - Slots.

Més conceptes importants

24. Concepte: "Virtual DOM".
    - Idea: to speed up, batch updates in next tick.
    - No és millor que fer updates puntuals!
    - `<div v-for="(item, index) in items" v-bind:key="">`

```
data: { variable: true },
template: `<div>
<input v-if="variable" key="1">
<input v-if="!variable" key="2">
</div>`,
```

25. **`computed`** (vs. **`methods`**)

```
data: { a: 1, b: 2 },
computed: { c: function() { return this.a + this.b } },
template: `{{a+b}} {{c}}`,
```

    ☞ És com un **`watch`** però que te la dependencia per la sortida en comptes de per l'entrada. L'exemple anterior s'ha de canviar per dos **`watch`**s.

26. "Change detection caveats"; és a dir quan no funciona la reactivitat.
    - Quan s'afegeixen dades a les opcions a posteriori no són reactives.
    - A l'arrel no s'hi poden afegir propietats reactives (s'han de deixar fetes al començament).
    - Fer servir **`this.$set(target, property, value)`** a nivells més profunds.
    - Passa el mateix als arrays: fer servir **`$set`** o **`splice`** per afegir, treure, o canviar elements (o qualsevol de les altres funcions suportades, com push, etcetera).

27. Seguretat: les interpolacions són segures per defecte, però és pot injectar codi html amb la directiva **`v-html`**.

```
data: { a : '<input>' },
template: `<div v-html="a"></div>`,
```

☞ **NO INTERPOLAR RAW HTML PROPORCIONAT ALGÚ ALTRE! MAI!**

Lectura

☞ https://arstechnica.com/information-technology/2018/10/two-new-supply-chain-attacks-come-to-light-in-less-than-a-week/