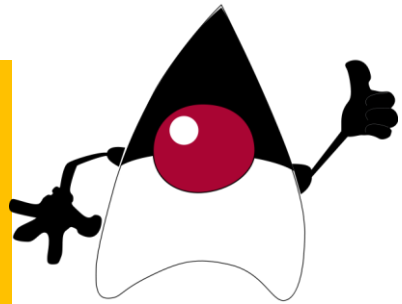
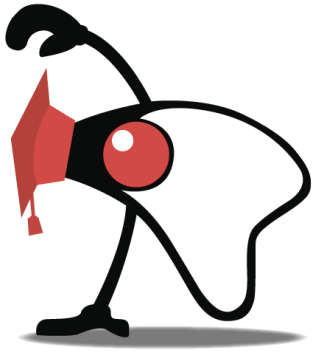
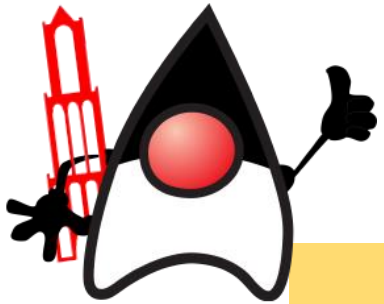
Curso FullStack Python

Codo a Codo 4.0







VUE.js

Parte 2

Agregando elementos a la instancia VUE

Tomaremos el ejemplo del array de frutas y sumaremos una nueva propiedad a la instancia VUE, dentro de la propiedad data:

```
nuevaFruta: '',
```

Así como tenemos datos / información asociada a mi instancia también podemos tener métodos o funciones:

```JS
methods: {
  agregarFruta(){
    this.frutas.push({ nombre: this.nuevaFruta, cantidad: 0 })
  }
}
```

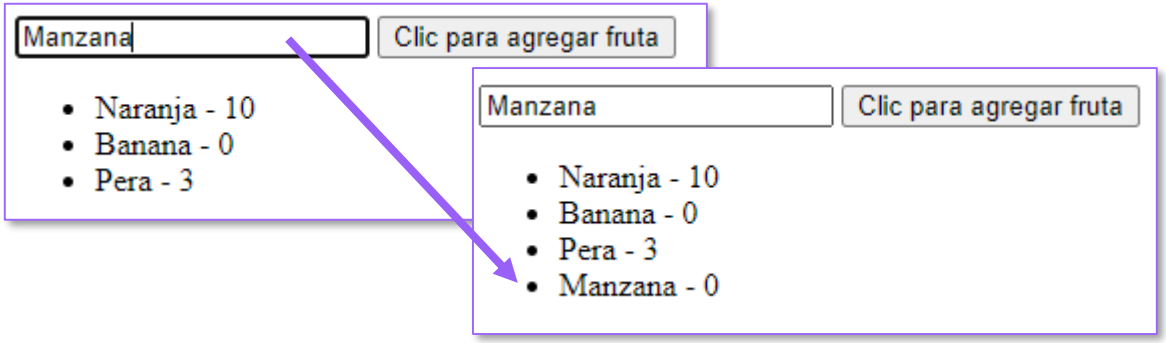
*Crearemos un método que utilizará el método push para agregar un nuevo elemento al array de objetos (frutas) de la misma manera que fueron agregadas antes, con los pares clave: valor (respetando la estructura). El nombre de la fruta es el de **nuevaFruta** (propiedad de la instancia de VUE) y utilizamos el **this** para hacer referencia, justamente, a esa instancia. Además sumamos que la nuevaFruta inicie con cantidad 0.*

Agregando elementos a la instancia VUE

En el documento HTML agregaremos un input y un botón para agregar la fruta que deseamos al array de objetos:

HTML

```
<input type="text" v-model="nuevaFruta">
<button v-on:click=agregarFruta()>Clic para agregar fruta</button>
```



¿Cómo funciona todo esto?

1. *El botón "Clic para agregar..." tiene asociado una directiva v-on que llama al método agregarFruta() de la instancia de VUE creada en JS.*
2. *En JS el método agregarFruta() utilizaba un push para agregar esa nueva fruta al array de objetos.*
3. *Además, a través de la directiva v-model se conectan el input con la propiedad de la instancia de VUE. Lo que suceda en el input se va a ver reflejado en la propiedad y viceversa (**comunicación bidireccional**).*

Agregando elementos a la instancia VUE

Una mejora que se puede hacer es que se limpie la caja de texto cuando agregamos un elemento. Esto se logra agregando en el método agregarFruta() esta instrucción:

```
this.nuevaFruta= '';
```
JS

Podemos hacer una mejora para que no nos permita agregar una fruta hasta que no se haya completado la caja de texto:

```
agregarFrutaConIF(){
if (this.nuevaFruta != "") {
    this.frutas.push({ nombre: this.nuevaFruta, cantidad: 0 });
    this.nuevaFruta= '';
}
```
JS

Ver ejemplo v-model_2 (.html y .js)

Eventos: Modificadores de teclas

Evento keyUp: Nos permite disparar un método una vez que se levanta una tecla. Por ejemplo Enter:

```html
<input type="text" v-model="nuevaFruta" @keyup.enter="agregarFrutaConIF">
```

HTML

Al presionar Enter se dispara el método que me permite agregar una fruta al array de objetos

Ver ejemplo eventos-key (.html y .js)

Fuente: https://vuejs.org/v2/guide/events.html#Key-Modifiers

Otros eventos: https://es.vuejs.org/v2/guide/events.html

Computados

Me van a permitir que VUE agregue una **función** que realice alguna *operación matemática*.
La potencia de los computados es que este método se va a ejecutar cuando haya un cambio en el HTML.
En este ejemplo crearemos una función que realice la sumatoria total de frutas. Va a recorrer el array y sumar las cantidades, luego las va a mostrar.

JS

```
computed: {
  sumarFrutas() {//Muestra sumatoria total de cantidades de frutas.
    this.total = 0;
    for (fruta of this.frutas) {
      this.total += fruta.cantidad; //acumulador
    }
    return this.total;
  }
}
```

*El **for** recorrerá las cantidades del array **frutas**. Se utiliza **fruta.cantidad** y se va agregando a total con **this.total +=fruta.cantidad**;. Además hay un **return** que devuelve el total.*

Dentro de **data** necesitamos inicializar esa propiedad (variable) total.

```
total: 0
```

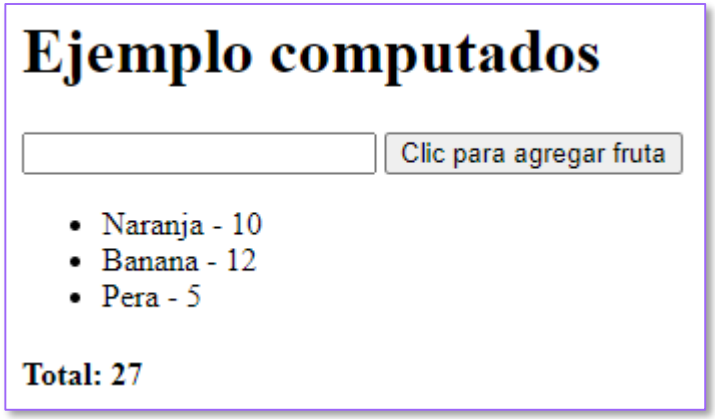
Computados

Agregaremos la etiqueta H4 en el HTML para que me muestre el total calculado:

```html
<h4>Total: {{sumarFrutas}}</h4>
```

HTML

Debemos tomar el valor de retorno del método, no la variable Total



Cuando haga un cambio en el documento HTML la función que está dentro de ***computed*** *se va a ejecutar*

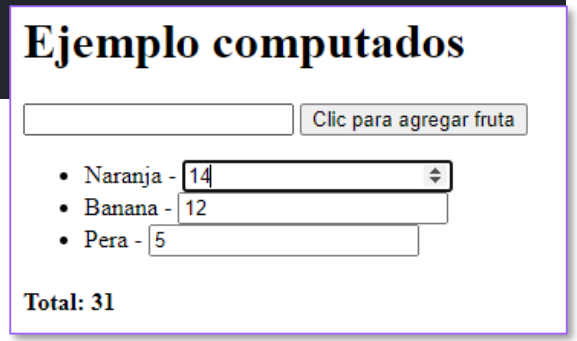
Ver ejemplo computados (.html y .js)

Computados

Aprovechando la capacidad de VUE de actualizar solamente aquello que se modifica en el documento HTML podremos desarrollar un proyecto que sume frutas a medida que se van agregando, como si fuera un “carrito de compras”:

JS

```
<ul>
    <li v-for="fruta in frutas">
        {{ fruta.nombre }} - <input type="number" v-model.number="fruta.cantidad">
    </li>
</ul>
<h4>Total: {{sumarFrutas}}</h4>
```



Si hacemos una modificación en el input se modifican los datos y se ven en el HTML (enlace bidireccional). Utilizamos v-model.number porque lo que introduciremos son valores numéricos. Si no lo colocáramos concatenaría los datos al no estar parseados, los guarda como un string y concatena.

Ver ejemplo computados-2 (.html y .js)

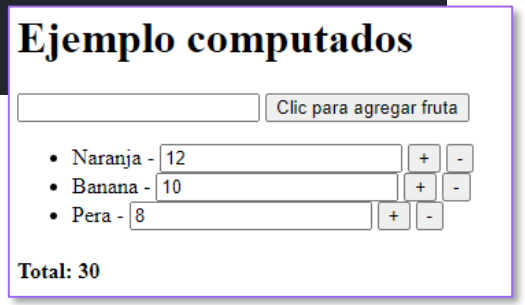
Computados

También podremos agregar botones para sumar y restar valores:

```
<li v-for="fruta in frutas">
    {{ fruta.nombre }} - <input type="number" v-model.number="fruta.cantidad">
    <button @click=fruta.cantidad++>+</button>
    <button @click=fruta.cantidad-->-</button>
</li>
```

JS

Utilizamos @click que reemplaza a v-on y agregando el acumulador de la propiedad cantidad del objeto fruta (para sumar o restar) dentro de cada botón.



Ver ejemplo computados-3 (.html y .js)

Creación de componentes

Los componentes nos permiten dividir en partes nuestro código HTML. Por ejemplo podemos tener resuelta nuestra barra de navegación en un componente externo y utilizarla.
Hasta ahora sabíamos que podíamos crear un objeto de tipo VUE y utilizar sus propiedades en el documento HTML. Ahora vamos a **crear componentes** que son los que van a contener lo que queremos mostrar en el documento HTML.:

Esto ahora vamos a modificarlo con la creación de componentes.

Importante: Aunque creemos un componente de VUE es requisito mantener esta referencia al ID #app (en este caso), aunque no tenga ***data***. En caso contrario no funciona el componente.

Creación de componentes

Vamos a crear un componente, que aprovechando una etiqueta HTML creada por nosotros Podemos crear una etiqueta, por ejemplo **<saludo>** y utilizarla en nuestro documento. Si bien no es válida en HTML la vamos a hacer válida a partir de su creación.

JS

```
Vue.component('saludo', {
    template: "<h1>Hola (estático desde template)</h1>"
})
```

HTML

```
<div id="app">
    <saludo></saludo>
</div>
```

Hola (estático desde template)

```
▼<div id="app">
    <h1>Hola (estático desde template)</h1>
  </div>
```

Como primer parámetro le paso el nombre del componente y como segundo parámetro el objeto. ***Template*** *es una de las propiedades más importantes de los componentes.*

Sin embargo, esta referencia es estática. Cuando trabajamos con componentes al **data**, que se agregaba como propiedad en la instancia de VUE, lo vamos a incluir en el propio componente (antes **data** era una propiedad de la instancia de VUE, pero a la vez era un objeto con propiedades).

Creación de componentes

Entonces **data** ahora será un método que retorna un valor, por ejemplo otro saludo:

JS

```
Vue.component('saludodos', {
    template: "<h1>{{msj}}</h1>",
    data(){
        return {
            msj: 'Hola (dinámica y como componente)'
        }
    }
})
```

En este caso el dato lo toma desde el mismo componente

```
<div id="app">
    <saludo></saludo>
    <saludodos></saludodos>
</div>
```

Hola (estático desde template)

Hola (dinámica y como componente)

```
▼<div id="app">
    <h1>Hola (estático desde template)</h1>
    <h1>Hola (dinámica y como componente)</h1>
```

Utilizar el template de esta forma nos limita a una única línea. Si queremos poner más de una línea en HTML utilizaremos los **backticks** (*comillas invertidas*).

Creación de componentes

Con las comillas invertidas podremos escribir *más de una línea*. Es importante también saber que los templates deben ir dentro de un **contenedor** (en este caso un ***div***):

JS

```
Vue.component('saludotres', {
    template: `
        <div>
            <h1>{{msj}}</h1>
            <h2>{{titulo}}</h2>
        </div>
    `,
    data(){
        return {
            msj: 'Hola (dinámica y como componente)',
            titulo: "Título dinámico"
        }
    }
})
```

```
▼<div id="app"> == $0
  ▼<div>
      <h1>Hola (dinámica y como componente)</h1>
      <h2>Título dinámico</h2>
    </div>
</div>
```

Hola (dinámica y como componente)

Título dinámico

HTML

```
<div id="app">
    <saludotres></saludotres>
</div>
```

Con la extensión **es6-string-html** podemos formatear las etiquetas HTML que estén dentro de un string, eso visualmente ayuda mucho.

Creación de componentes

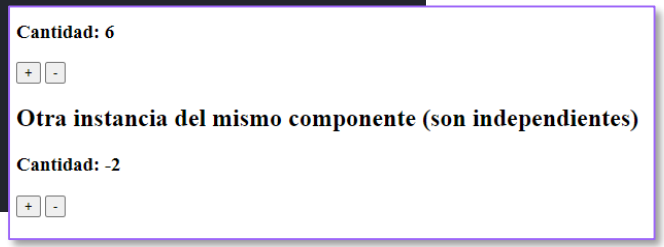
Otro ejemplo de uso de un componente utilizando botones y contadores:

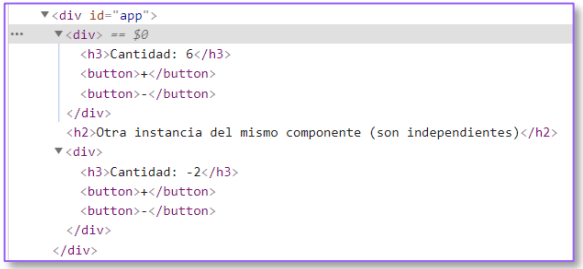
JS

```
Vue.component('contador', {
    template: //html
        `
        <div>
            <h3>Cantidad: {{num}}</h3>
            <button @click="num++">+</button>
            <button @click="num--">-</button>
        </div>
        `,
    data() {
        return {
            num: 0
        }
    }
})
```

HTML

```
<div id="app">
    <contador></contador>
    <h2>Otra instancia del mismo componente
 (son independientes)</h2>
    <contador></contador>
</div>
```





*Creamos un componente llamado contador que lo vinculamos con la etiqueta homónima. Este componente tiene tres líneas: un **h3** que contiene un texto fijo y un elemento variable que se incrementa de acuerdo a la instrucción que tiene cada botón para incrementar o decrementar de a 1.*

Ejemplos, cursos y guías de VUE.js. APIS

- **Guía de VUE,js:** https://es.vuejs.org/v2/guide/index.html#
- **Ejemplos VUE:** https://vuejsexamples.com/
- **Escuela VUE:** https://escuelavue.es/series/
- **¿Qué son las APIs y para qué sirven?:** https://youtu.be/u2Ms34GE14U
- **Curso de Vue JS - Tutorial en Español [Desde Cero]**: https://www.youtube.com/playlist?list=PLPl81lqbj-4J-gfAERGDCdOQtVgRhSvIT
- **VUE Mastery (curso):** https://www.vuemastery.com/courses/intro-to-vue-js/vue-instance/
- **Lenguaje JS - ¿Qué es VUE?**: https://lenguajejs.com/vuejs/introduccion/que-es-vue/