Vue.Js Notes Solutions :

**Need to learn & create Program in the same ....**

**# parent to child connectivity for vue js**

**# child to parent**

**# how to make reactive child passing data from parent**

**# lifeCycle in vue Js**

**# which lifeCycle is used when Dom is called**

**# how to pass data from child to parent in lifeCycle method**

**# how to find function is called vue Js**

**# what is watch & how to use watch**

**# what is computed & how to use it**

**# between computed & methods**

**# find , filter , foreach , map method in JS**

**# mixin methods in vue**

**# proto types**

**++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++**

**1 ) Parent to Child Connectivity in Vue.js >>>**

**In Vue.js, you can pass data from a parent component to a child component using props.**

**Example:**

**Parent.vue >>>**

<template>

<ChildComponent :message="parentMessage" />

</template>

<script>

import ChildComponent from './ChildComponent.vue';

export default {

data() {

return {

parentMessage: 'Hello from Parent'

}

}

}

</script>

**ChildComponent.vue >>>**

<template>

<div>{{ message }}</div>

</template>

<script>

export default {

props: ['message']

}

</script>

----------------------------------------------------------------------------------------------------------------------

**2) Child to Parent Connectivity >>>>**

**To pass data from a child to a parent, you can emit an event in the child component and listen for that event in the parent component.**

**Example:**

**Child.vue >>>**

<template>

<button @click="sendToParent">Click me</button>

</template>

<script>

export default {

methods: {

sendToParent() {

this.$emit('childEvent', 'Hello from Child');

}

}

}

</script>

**Parent.vue >>>**

<template>

<ChildComponent @childEvent="handleEvent" />

</template>

<script>

import ChildComponent from './ChildComponent.vue';

export default {

methods: {

handleEvent(data) {

console.log(data); // Output: Hello from Child

}

}

}

</script>

**3) Reactive Child Passing Data from Parent >>>**

If you want to make reactive data flow from the parent to the child, Vue's reactivity system will automatically update the child component whenever the parent's data changes.

**Example:**

**Parent.vue**

<template>

<ChildComponent :counter="counter" />

<button @click="incrementCounter">Increment</button>

</template>

<script>

import ChildComponent from './ChildComponent.vue';

export default {

data() {

return {

counter: 0

}

},

methods: {

incrementCounter() {

this.counter++;

}

}

}

</script>

**ChildComponent.vue >>>**

<template>

<div>Counter: {{ counter }}</div>

</template>

<script>

export default {

props: ['counter']

}

</script>

**4 ) Vue.js Lifecycle Hooks >>>>**

**Vue provides lifecycle hooks that allow you to run code at specific stages of the component's lifecycle.**

**beforeCreate:** Called before the component instance is initialized.

**created:** Called after the component is created, but before mounting**.**

**beforeMount:** Called before the DOM is mounted.

**mounted**: Called after the DOM is mounted (useful when interacting with the DOM).

**beforeUpdate:** Called when data changes before the DOM is re-rendered.

**updated:** Called after the data is updated and the DOM is re-rendered.

**beforeDestroy:** Called before the component is destroyed.

**destroyed:** Called after the component is destroyed.

**5. Lifecycle Hook When the DOM is Available >>>>**

The mounted lifecycle hook is called when the DOM is available for manipulation.

<script>

export default {

mounted() {

console.log('Component is mounted and the DOM is available.');

}

}

</script>

**6. Passing Data from Child to Parent in a Lifecycle Method >>>>**

You can emit an event inside a lifecycle hook in a child component to send data to the parent.

<script>

export default {

mounted() {

this.$emit('childMounted', 'Child Mounted');

}

}

</script>

**7 ) How to Check if a Function is Called in Vue.js >>>>**

You can log messages to the console or use a debugger to check if a function is being called.

<script>

export default {

methods: {

myMethod() {

console.log('myMethod was called');

}

}

}

</script>

**8) Watch in Vue.js >>>>**

The watch option is used to react to changes in specific data properties or computed properties.

<script>

export default {

data() {

return {

message: 'Hello'

}

},

watch: {

message(newValue, oldValue) {

console.log('Message changed from', oldValue, 'to', newValue);

}

}

}

</script>

9) Computed Properties in Vue.js >>>>

Computed properties are reactive and cached based on their dependencies. They are used to compute derived state.

<script>

export default {

data() {

return {

firstName: 'John',

lastName: 'Doe'

}

},

computed: {

fullName() {

return this.firstName + ' ' + this.lastName;

}

}

}

</script>

10) Computed vs. Methods in Vue.js >>>>

**Computed:** Cached based on dependencies and only re-evaluated when dependencies change.

**Methods:** Re-evaluated every time they are called.

<script>

export default {

methods: {

fullName() {

return this.firstName + ' ' + this.lastName;

}

}

}

</script>

11 ) JavaScript Array Methods >>>>

**find:** Finds the first element in the array that satisfies a condition.

**filter:** Returns an array of all elements that satisfy a condition.

**forEach:** Executes a function for each array element.

**map:** Transforms an array by applying a function to all elements.

const numbers = [1, 2, 3, 4, 5];

const found = numbers.find(num => num > 3);

const filtered = numbers.filter(num => num > 3);

numbers.forEach(num => console.log(num));

const mapped = numbers.map(num => num \* 2);

console.log(found); // 4

console.log(filtered); // [4, 5]

console.log(mapped); // [2, 4, 6, 8, 10]

12 ) Mixin in Vue.js >>>>

Mixins are a way to reuse logic between components. A mixin can contain data, methods, lifecycle hooks, etc.

**mixin.js >>**

export const myMixin = {

data() {

return {

mixinMessage: 'Hello from mixin'

}

},

methods: {

mixinMethod() {

console.log('This is from the mixin');

}

}

}

**Component.vue**

<script>

import { myMixin } from './mixin.js';

export default {

mixins: [myMixin],

mounted() {

console.log(this.mixinMessage); // Output: Hello from mixin

this.mixinMethod(); // Output: This is from the mixin

}

}

</script>

13) Prototypes in JavaScript >>>>

Prototypes allow you to add methods or properties to JavaScript objects.

function Person(name) {

this.name = name;

}

Person.prototype.sayHello = function() {

console.log(`Hello, my name is ${this.name}`);

}

const person = new Person('John');

person.sayHello(); // Output: Hello, my name is John

--------------------------------------------------------------------------------------------------------------

Summary (Single Notes):

Parent to Child: Use props.

Child to Parent: Use $emit to trigger an event in the parent.

Reactive Child: Props from the parent will update the child automatically.

Lifecycle Hooks: Use mounted for DOM manipulation.

Watch: Used to reactively watch data or computed changes.

Computed: Cached, reactive properties.

Array Methods: find, filter, forEach, map.

Mixins: Reusable logic between components.

Prototypes: Extend JavaScript objects with new methods.

This should give you a solid foundation to work with Vue.js!