

Composition API

<https://ua.vuejs.org/guide/introduction.html>

<https://ua.vuejs.org/api/#%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%B7%D0%B8%D1%86%D1%96%D0%B9%D0%BD%D0%B8%D0%B9-api>

Створення додатку

Options API	Composition API
<p>Встановлення Vue CLI (is based on webpack)</p> <pre>npm install -g @vue/cli # OR yarn global add @vue/cli</pre>	<p>Встановлення (is based on Vite)</p> <pre>npm create vue@latest</pre>
<p>Створення додатку</p> <pre>vue create назва-додатку</pre>	<pre>Need to install the following packages: create-vue@3.9.0 Ok to proceed? (y) y Vue.js - The Progressive JavaScript Framework ✓ Project name: ... vue-test-project ✓ Add TypeScript? ... <u>No</u> / Yes ✓ Add JSX Support? ... <u>No</u> / Yes ✓ Add Vue Router for Single Page Application development? ... No / <u>Yes</u> ✓ Add Pinia for state management? ... No / <u>Yes</u> ✓ Add Vitest for Unit Testing? ... <u>No</u> / Yes ✓ Add an End-to-End Testing Solution? » No ✓ Add ESLint for code quality? ... No / <u>Yes</u> ✓ Add Prettier for code formatting? ... No / <u>Yes</u> Scaffolding project in D:\Think\Vue_2023\Completed\18.CompositionAPI\pract_ex\CompApi\vue-test-project... Done. Now run: cd vue-test-project npm install npm run format npm run dev</pre>

Опис реактивних даних

Загальна форма	Приклад	Аналог у Options API
<pre>//створюється об'єкт-обгортка // з властивістю value import { ref } from 'vue' const посилання = ref(поч.знач.)</pre>	<pre>import { ref } from 'vue' //описуємо, навіть якщо null const error = ref(null) //count => {value:0} const count = ref(0) //username => {value: 'Ivan'} const username = ref('Ivan') //age => {value:0} const age = ref(35) //book => {value:{author: 'Petro', // title: 'Super book' } // } let book = ref({ author: 'Petro', title: 'Super book' })</pre>	<pre>export default { data() { return { count: 0, userName: 'Ivan', age: 35, book:{ author: 'Petro', title: 'Super book' } }, } },</pre>

Опис реактивних даних

Загальна форма	Приклад	Аналог у Options API
//створюється об'єкт-обгортка // з властивістю value import { ref } from 'vue' const посилання = ref(поч.знач.)	import { ref } from 'vue' //описуємо, навіть якщо null const error = ref(null) //count => {value:0} const count = ref(0) //username => {value: 'Ivan'} const username = ref('Ivan') //age => {value:0} const age = ref(35) //book => {value:{author: 'Petro', // title: 'Super book' }} // } let book = ref({ author: 'Petro', title: 'Super book' })	export default { data() { return { count : 0, userName : 'Ivan', age: 35, book: { author: 'Petro', title: 'Super book' } } }, }
//об'єкт перетворюється на рективний import { reactive } from 'vue' const посилання = reactive(об'єкт)	import { reactive } from 'vue' let book = reactive({ author: 'Petro', title: 'Super book' })	

Розробка компонентів з використанням Composition API

З використанням setup-функцій

```
<template>
    . . . . .
</template>

<script>
export default {
    setup() {
        . . . . .
    }
}
</script>

<style lang="scss" >
    . . . . .
</style>
```

З використанням <script setup>

```
<template>
    . . . . .
</template>  (Ctrl) ▾

<script setup>
    . . . . .

</script>

<style lang="scss" >
    . . . . .
</style>
```

Експорт реактивних даних для використання у шаблонах

З використанням setup-функцій	З використанням <script setup>	Аналог у Options API
<pre><template> ... використання посилань ... </template> <script> import { ref } from 'vue' export default { // `setup` – задання стану setup() { описание посилань . . // виділення стану до шаблону return { ... список посилань ... } } }</script></pre>	<pre><template> ... використання посилань ... </template> <script setup> import { ref } from 'vue' описание посилань . . </script> <style lang="scss" scoped></style></pre>	<pre><template> ... використання моделей ... </template> <script> export default { data() { return { описание моделей даних . . } }, }</script></pre>

setup-функції. Експорт реактивних даних для використання у шаблонах

Загальна форма	Приклад	Аналог у Options API
<pre><template> ... використання посилань ... </template> <script> import { ref } from 'vue' export default { // `setup` – задання стану setup() { опис посилань . . . // виділення стану до шаблону return { ... спісок посилань ... } } </script> <style lang="scss" scoped></style></pre>	<pre><template> <div> <div>Count value: {{ count }}</div> <div>{{ userName }}</div> <div> {{ book.title }}- {{ book.author }} </div> </div> </template> <script> import { ref } from 'vue' export default { // `setup` – задання стану setup() { const count = ref(0) let userName = ref('Ivan') let book = ref({ author: 'Petro', title: 'Super book' }) // виділення стану до шаблону return { count, userName, book } } </script> <style lang="scss" scoped></style></pre>	<pre><template> <div> <div> Count value: {{ count + 6 }} </div> <div>{{ userName }}</div> <div> {{book.title}}-{{book.author}} </div> </div> </template> <script> export default { data() { return { count: 0, userName:'Ivan', book:{ author: 'Petro', title: 'Super book' } } } </script></pre>

<script setup>. Експорт реактивних даних для використання у шаблонах

Загальна форма	Приклад	Аналог у Options API
<pre><template> ... використання посилань ... </template> <script setup> import { ref } from 'vue' опис посилань . . . </script> <style lang="scss" scoped></style></pre>	<pre><template> <div> <div>Count value: {{ count + 6 }}</div> <div>{{ userName }}</div> <div>{{ book.title }}-{{ book.author }}</div> </div> </template> <script setup> import { ref } from 'vue' const count = ref(0) let(userName = ref('Ivan')) let book = ref({ author: 'Petro', title: 'Super book' }) </script> <style lang="scss" scoped></style></pre>	<pre><template> <div> <div>Count value: {{ count + 6 }}</div> <div>{{ userName }}</div> <div>{{book.title}}-{{book.author }}</div> </div> </template> <script> export default { data() { return { count: 0, userName:'Ivan', book:{ author: 'Petro', title: 'Super book' } }, } </script></pre>

Опис методів

З використанням setup-функцій	З використанням <script setup>	Аналог у Options API
<pre>import { ref } from 'vue' export default { setup() { //опис функції function назва_метода() { } //виділення функції return { назва_метода } } }</pre>	<pre><script setup> import { ref } from 'vue' //опис функції function назва_метода () { } </script></pre>	<pre><script> export default { methods: { назва_метода () { }, } </script></pre>

Опис методів. Приклад

Загальна форма	Приклад	Аналог у Options API
<pre> <template> <div> <div> Count value: {{ count }} </div> <button @click="incrementCount"> Inc </button> </div> </template> <script> import { ref } from 'vue' export default { setup() { const count = ref(0) function incrementCount() { count.value++ } // виділення стану до шаблону return { count, incrementCount } } } </script> </pre>	<pre> <template> <div> <div> Count value: {{ count }} </div> <button @click="incrementCount"> Inc </button> </div> </template> <script setup> import { ref } from 'vue' const count = ref(0) function incrementCount() { count.value++ } </script> </pre>	<pre> <template> <div> <div> Count value: {{ count }} </div> <button @click="incrementCount"> Inc </button> </div> </template> <script> export default { data() { return { count: 0, } }, methods: { incrementCount() { this.count++ }, } } </script> </pre>

Звертання/зміна значень у методах і шаблоні

У JS методах (потрібно використовувати властивість <code>value</code>)	Приклад	Аналог у Options API
<pre>import { ref } from 'vue' const [посилання] = ref([поч.знач.]) [посилання] . value = [нове_значення]</pre>	<pre><script setup> import { ref } from 'vue' //count => { value : 0 } const count = ref(0) function increment() { count.value++ } </script></pre>	<pre>export default { data() { return { count: 0, } }, methods: { increment () { this.count++ }, } }</script></pre>
У шаблоні (об'єкт розгортається, і додатково «value» вказувати не потрібно)	<pre><button @click="count++"> {{ count }} </button></pre>	<pre><button @click="count++"> {{ count }} </button></pre>

Декілька моделей даних. Перевага Composition API (посилання і функції ідуть поруч)

Загальна форма

```
<template>
  <div>
    <div>Count value: {{ count_1 }}</div>
    <button @click="incrementCount_1">Inc</button>

    <div>Count value: {{ count_2 }}</div>
    <button @click="incrementCount_2">Inc</button>
  </div>
</template>

<script setup>
import { ref } from 'vue'
const count_1 = ref(0)
function incrementCount_1() {
  count_1.value++
}

const count_2 = ref(0)
function incrementCount_2() {
  count_2.value++
}
</script>
```

Аналог у Options API

```
<template>
  <div>
    <div>Count value: {{ count_1 }}</div>
    <button @click="incrementCount_1">Inc</button>

    <div>Count value: {{ count_2 }}</div>
    <button @click="incrementCount_2">Inc</button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      count_1: 0,
      count_2: 0,
    }
  },
  methods: {
    incrementCount_1() {
      this.count_1++
    },
    incrementCount_2() {
      this.count_1++
    },
  }
}
</script>
```

Обчислювальні властивості

Загальна форма	Приклад	Аналог у Options API
<pre>import {ref, computed} from 'vue' const [обч._власт] = computed(() => { return [обчислене_значення] }) [Ctrl]</pre>	<pre><template> <div> <div>Count value: {{ count }}</div> <button @click="incrementCount"> Inc </button> <div>{{ oddOrEven }}</div> </div> </template> <script setup> import { ref, computed } from 'vue' const count = ref(0) function incrementCount() { count.value++ } const [oddOrEven] = computed(() => { return count.value % 2 == 0 ? 'Парне' : 'Непарне' }) </script></pre>	<pre><template> <div> <div>Count value: {{ count }}</div> <button @click="incrementCount"> Inc </button> <div>{{ oddOrEven }}</div> </div> </template> <script> export default { data() { return { count: 0, } }, computed: { oddOrEven() { return this.count.value % 2 == 0 ? 'Парне' : 'Непарне' }, }, methods: { incrementCount() { this.count++ }, } }</script></pre>

Обчислювальні властивості з можливістю запису (з `get`, `set`)

Загальна форма	Приклад	Аналог у Options API
<pre>import{ref, computed} from 'vue' const [обч._власт] = computed(get(){ return [обчислене_значення] }, set(newValue){ })</pre>	<pre><script setupcomputed } from 'vue' const firstName = ref('John') const lastName = ref('Doe') const fullName = computed({ // getter get() { return firstName.value + ' ' + lastName.value }, // setter set(newValue) { [firstName.value, lastName.value] = newValue.split(' ') } }) </script></pre>	<pre><script> export default { data() { return { firstName: 'John', lastName: 'Doe', } }, computed: { fullName: { // getter get() { return this.firstName.value + ' ' + this.lastName.value }, // setter set(newValue) { ;[this.firstName.value, this.lastName.value] = newValue.split(' ') } }, } }</pre>

Реєстрація хуків життєвого циклу

`onMounted, onUpdate, onUnmounted, onBeforeMount, onBeforeUpdate, onBeforeUnmount, onErrorCaptured, ...`

Загальна форма	Приклад	Аналог у Options API
<pre>import { ref, onХук_життєвого_циклу } from 'vue' onХук_життєвого_циклу (()=>{ })</pre>	<pre><script setup> import { onMounted } from 'vue' onMounted(() => { console.log(`тепер компонент змонтовано.`) }) </script></pre>	<pre>export default { mounted () { console.log(`тепер компонент змонтовано.`) }, }</script></pre>

Спостерігачі

Загальна форма	Приклад
<pre><script setup> import { ref, watch } from 'vue' const [посилання] = ref([поч.знач.]) watch([посилання], (newValue, oldValue) => { })</script></pre>	<pre><template> <div> <div> <label> User age <input v-model="userAge" type="number" /> </label> </div> <div>Increase status: {{increaseStatus}}</div> </div> </template> <script setup> import { ref, watch } from 'vue' const userAge = ref(null) const increaseStatus = ref(null) watch(userAge, (newVal, oldVal) => { increaseStatus.value = newVal > oldVal }) </script></pre>

Спостерігачі

Загальна форма	Приклад	Аналог у Options API
<pre><script setup> import { ref, watch } from 'vue' const [посилання] = ref(поч.знач.) watch([посилання], (newValue, oldValue) => { })</script></pre>	<pre><template> <div> <div> <label> User age <input v-model="userAge" type="number" /> </label> </div> <div>Increase status: {{increaseStatus }}</div> </div> </template> <script setup> import { ref, watch } from 'vue' const userAge = ref(null) const increaseStatus = ref(null) watch(userAge, (newVal, oldVal) => { increaseStatus.value = newVal > oldVal })</script></pre>	<pre><template> <div> <div> <label> User age <input v-model="userAge" type="number" /> </label> </div> <div>Increase status: {{increaseStatus }}</div> </div> </template> <script> export default { data() { return { userAge: null, increaseStatus: null, }, watch: { userAge(newVal, oldVal) { this.increaseStatus = newVal }, }, }</script></pre>

Вихідні типи спостерігачів

Загальна форма	Приклад
<p>Першим аргументом <code>watch</code> можуть бути різні типи реактивних "джерел":</p> <ul style="list-style-type: none">• це може бути посилання,• обчислювані посилання,• реактивний об'єкт,• функція отримання,• масив із кількох джерел	<pre>const x = ref(0) const y = ref(0) // одиночне посилання watch(x, (newX) => { console.log(`x - це \${newX}`) }) //реактивний об'єкт const obj = reactive({ count: 0 }) watch(() => obj.count, //потрібно описувати функцію!!! (count) => { console.log(`кількість: \${count}`) }) // getter watch(() => x.value + y.value, (sum) => { console.log(`сума x + y: \${sum}`) }) // масив із кількох джерел watch([x, () => y.value], ([newX, newY]) => { console.log(`x це \${newX}, у це \${newY}`) })</pre>

Посилання у шаблонах

Загальна форма	Приклад	Аналог у Options API
<pre><script setup> import { ref } from 'vue' // оголосити посилання на елемент // ім'я має відповідати значенню референції шаблону const посилення = ref(null) </script> <template> <тег ref=" посилення " /> </template></pre>	<pre><script setup> import {ref, onMounted} from 'vue' // оголосити посилання на елемент // ім'я має відповідати значенню референції шаблону const input = ref(null) const input2 = ref(null) onMounted(() => { input.value.focus() input.value.value = 'ok' input2.value.value = 'no' }) </script> <template> <div> <input ref="input" /> <input ref="input2" /> </div> </template></pre>	<pre><script> export default { mounted() { this.\$refs.input.focus() this.\$refs.input.value = 'ok' this.\$refs.input2.value = 'no' }, } </script> <template> <div> <input ref="input" /> <input ref="input2" /> </div> </template></pre>

Посилання у шаблонах, що всередині `v-for`

Загальна форма	Приклад	Аналог у Options API
<pre><script setup> import { ref } from 'vue' // оголосити посилання на елемент const [посилання] = ref([]) // звертання за посиланням [посилання].value[індекс] </script> <template> <тег v-for=" . . . " ref=" [посилання]"> /> </template></pre>	<pre><script setup> import { ref, onMounted } from 'vue' const input = ref([]) onMounted(() => { input.value[1].focus() input.value[0].value = 'ok' input.value[1].value = 'no' input.value[2].value = 'question' }) </script> <template> <div> <input ref="input" v-for="i in 3" :key="i" /> </div> </template></pre>	<pre><script> export default { mounted() { this.\$refs.input[1].focus() this.\$refs.input[0].value = 'ok' this.\$refs.input[1].value = 'no' this.\$refs.input[2].value = 'question' }, } </script> <template> <div> <input v-for="i in 3" :key="i" ref="input" /> </div> </template></pre>

Опис компонентів

Загальна форма <script setup>	Приклад. setup - function	Аналог у Options API
<pre><script setup> import { ref } from 'vue' //Опис вхідних даних-властивостей defineProps(. . . .) //опис посилань const [посилання] = ref([поч.знач]) //опис можливих подій defineEmits(. . . .) </script> <template> </template> <style> ...стилі, що використовуються у компоненті... </style></pre>	<pre><script > export default { //Опис вхідних даних-властивостей props: , //опис посилань const [посилання] = ref([поч.знач]) // опис можливих подій emits: , setup(props) { } } </script> <template> </template> <style> ...стилі, що використовуються у компоненті... </style></pre>	<pre><template> Опис розмітки . . . </template> <script> export default { name: "назва компонента" components: { ...реєстрація компонентів... }, props:{ ...опис вхідних параметрів... }, data () { return { ...моделі даних... } }, computed: { ...обчислювальні властивості... }, watch: { ...спостерігачі... }, methods: { ...методи компонента... } ...хуки життєвого циклу... } </script> <style> ...стилі, що використовуються у компоненті... </style></pre>

Реєстрація компонентів. Глобальна реєстрація

Загальна форма	Приклад
<pre>import { createApp } from 'vue' //імпорт компонента import Компонент from 'шлях' const app = createApp({}) app.component(// зареєстроване ім'я 'ім'я_компоненту', // реалізація компоненту Компонент)</pre>	<pre>import { createApp } from 'vue' //імпорт компонентів import MyComponentA from './App.vue' import MyComponentB from './App.vue' const app = createApp({}) app .component('ComponentA', ComponentA) .component('ComponentB', ComponentB)</pre>

Реєстрація компонентів. Локальна реєстрація. <script setup>

Загальна форма	Приклад
<pre><script setup> //імпорт компонента import Компонент from 'шлях' </script> <template> < Компонент /> </template></pre>	<pre><script setup> import ComponentA from './ComponentA.vue' </script> <template> <ComponentA /> </template></pre>

Реєстрація компонентів. Локальна реєстрація. setup

Загальна форма	Приклад
<pre><script> //імпорт компонента import Компонент from 'шлях' export default { components: { Компонент }, setup() { // ... } } </script > <template> < Компонент /> </template></pre>	<pre><script > import ComponentA from './ComponentA.js' export default { components: { ComponentA }, setup() { // ... } } </script > <template> < Компонент /> </template></pre>

Опис вхідних властивостей

Загальна форма

```
<script setup>
defineProps({
  // Основна перевірка типу
  // (`null` та `undefined` значення дозволяють будь-який
  // тип)
  propA: Number,
  // Кілька можливих типів
  propB: [String, Number],
  // Обов'язковий рядок
  propC: {
    type: String,
    required: true
  },
  // Число зі значенням за промовчанням
  propD: {
    type: Number,
    default: 100
  },
  // Об'єкт зі значенням за промовчанням
  propE: {
    type: Object,
    // Значення за промовчанням для об'єктів або масивів
    // мають бути повернуті фабричною функцією. Функція
    // отримує необроблені реквізити,
    // отримані компонентом як аргумент.
    default(rawProps) {
      return { message: 'Привіт' }
    }
  },
})
```

```
// Спеціальна функція перевірки
propF: {
  validator(value) {
    // Значення має відповідати одному з цих рядків
    return ['успіх', 'увага',
    'небезпека'].includes(value)
  }
},
// Функція зі значенням за промовчанням
propG: {
  type: Function,
  // На відміну від об'єкта чи масиву за промовчанням,
  // це не фабрика
  // функція - це функція, яка слугуватиме значенням за
  // промовчанням
  default() {
    return 'Функція за промовчанням'
  }
}
</script>
```

Опис вхідних властивостей

Приклад <script setup>	Приклад. setup - function
<pre><script setup> const props = defineProps(['foo']) console.log(props.foo) </script></pre>	<pre><script > export default { props: ['foo'], setup(props) { // setup() отримує реквізити як // перший аргумент. console.log(props.foo) } } </script></pre>

Події. <script setup>. Перевірка надсилання події

Загальна форма	Приклад
<pre><script setup> const emit = defineEmits({ submit(payload) { // повертає `true` або `false` // для вказівки // чи пройшла/не пройшла // перевірка } }) </script></pre>	<pre><script setup> const emit = defineEmits({ // Без перевірки click: null, // Перевірка події надсилання submit: ({ email, password }) => { if (email && password) { return true } else { console.warn('Недійсні данні події надсилання!') return false } }) </script></pre>

Події. setup-функції

Загальна форма	Приклад
<pre>export default { emits: ['назав_події', ...], setup(props, ctx) { ctx.emit('назав_події') } }</pre>	<pre>export default { emits: ['inFocus', 'submit'], setup(props, ctx) { ctx.emit('submit') } }</pre>

Композиційні функції

Описуємо у окремому файлі функцію, що містить посилання, функції, ...
та повертаємо стан, що може бути використаний ззовні

```
// mouse.js
import { ref, onMounted, onUnmounted } from 'vue'

// за конвенцією, назви композиційних функцій починаються з "use" (англ. – використо
export function useMouse() {
    // стан, інкапсульований і керований композиційною функцією
    const x = ref(0)
    const y = ref(0)

    // композиційна функція може оновлювати свій керований стан з часом.
    function update(event) {
        x.value = event.pageX
        y.value = event.pageY
    }

    // композиційна функція також може підключитися до свого компонента-власника
    // життєвий цикл для налаштування та демонтажу побічних ефектів.
    onMounted(() => window.addEventListener('mousemove', update))
    onUnmounted(() => window.removeEventListener('mousemove', update))

    // відкрити керований стан як значення, що повертається
    return { x, y }
}
```

Підключаємо де потрібно

```
<script setup>
import { useMouse } from './mouse.js'

const { x, y } = useMouse()
</script>

<template>Координати миші: {{ x }}, {{ y }}</template>
```