Щоб почати з VueJS, нам потрібно створити екземпляр Vue, який називається **віртуальним екземпляром Vue** .

Синтаксис

```
var app = new Vue({
   // options
})
```

Давайте розглянемо приклад, щоб зрозуміти, що має бути частиною конструктора Vue.

```
<html>
   <head>
      <title>VueJs Instance</title>
      <script type = "text/javascript" src = "js/vue.js"></script>
   </head>
   <body>
      <div id = "vue_det">
         <h1>Firstname : {{firstname}}</h1>
         <h1>Lastname : {{lastname}}</h1>
         <h1>{{mydetails()}}</h1>
      </div>
      <script type = "text/javascript" src = "js/vue_instance.js"></script>
   </body>
</html>
```

vue_instance.js

```
var  vm = new Vue({
   el: '#vue_det',
   data: {
      firstname : "Ria",
      lastname  : "Singh",
      address   : "Mumbai"
   },
   methods: {
      mydetails : function() {
         return "I am "+this.firstname +" "+ this.lastname;
      }
   }
})
```

Для Vue є параметр під назвою **el** . Вона займає ідентифікатор елемента DOM. У наведеному вище прикладі ми маємо ідентифікатор **#vue_det** . Це ідентифікатор елемента div, який присутній у .html.

```
<div id = "vue_det"></div>
```

Тепер, що ми будемо робити, буде впливати на елемент div і нічого поза ним.

Далі ми визначили об'єкт даних. Вона має значення першого прізвища, прізвище та адреси.

Те саме призначається всередині div. Наприклад,

```
<div id = "vue_det">
   <h1>Firstname : {{firstname}}</h1>
   <h1>Lastname : {{lastname}}</h1>
</div>
```

Значення ім'я: {{firstname}} буде замінено всередині інтерполяції, тобто {{}} з значенням, присвоєним об'єкту даних, тобто Ria. Те саме стосується прізвища.

Далі ми маємо методи, в яких ми визначили функцію mydetails і повертається значення. Він присвоюється всередину div як

```
<h1>{{mydetails()}}</h1>
```

Отже, всередині {{}} викликається функція mydetails. Значення, повернене в екземплярі Vue, буде надруковано всередині {{}}. Перевірте вихід для довідки.

Вихідні дані

Тепер нам потрібно передати параметри конструктору Vue, який, в основному, - дані, шаблон, елемент для монтажу, методи, зворотні виклики тощо.

Давайте подивимося на варіанти, які будуть передані Vue.

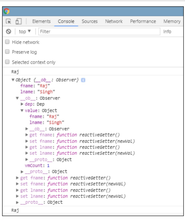
#data - цей тип даних може бути об'єктом або функцією. Vue перетворює свої властивості на геттери, щоб зробити його реактивним.

Давайте подивимося, як дані передаються в опціях.

Приклад

```
<html>
   <head>
      <title>VueJs Introduction</title>
      <script type = "text/javascript" src = "js/vue.js"></script>
   </head>
   <body>
      <script type = "text/javascript">
         var _obj = { fname: "Raj", lname: "Singh"}

         // direct instance creation
         var vm = new Vue({
            data: _obj
         });
         console.log(vm.fname);
         console.log(vm.$data);
         console.log(vm.$data.fname);
      </script>
   </body>
</html>
```

Вихідні дані



console.log (vm.fname); // друкує Радж

console.log (vm. $ data); надрукує повний об'єкт, як показано вище

console.log (vm. $ data.fname); // друкує Радж

Якщо є компонент, об'єкт даних повинен бути відсланий з функції, як показано в наступному коді.

```
<html>
   <head>
      <title>VueJs Introduction</title>
      <script type = "text/javascript" src = "js/vue.js"></script>
   </head>
   <body>
      <script type = "text/javascript">
         var _obj = { fname: "Raj", lname: "Singh"};

         // direct instance creation
         var vm = new Vue({
            data: _obj
         });
         console.log(vm.fname);
         console.log(vm.$data);
         console.log(vm.$data.fname);

         // must use function when in Vue.extend()
         var Component = Vue.extend({
            data: function () {
               return _obj
            }
         });
         var myComponentInstance = new Component();
         console.log(myComponentInstance.lname);
         console.log(myComponentInstance.$data);
      </script>
   </body>
</html>
```

У випадку компонента дані є функцією, яка використовується з Vue.extend, як показано вище. Дані - це функція. Наприклад,

```
data: function () {
   return _obj
}
```

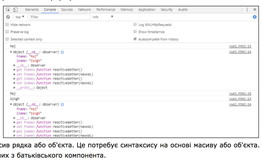
Щоб послатися на дані з компонента, нам потрібно створити його екземпляр. Наприклад,

```
var myComponentInstance = new Component();
```

Щоб отримати дані з даних, ми повинні робити те ж саме, що і з батьківським компонентом вище. Наприклад.

```
console.log(myComponentInstance.lname);
console.log(myComponentInstance.$data);
```

Нижче наведено відомості, які відображаються в браузері.



Підказки - Тип для реквізиту - масив рядка або об'єкта. Це потребує синтаксису на основі масиву або об'єкта. Вони, як кажуть, є атрибутами, які використовуються для прийому даних з батьківського компонента.

Приклад 1

```
Vue.component('props-demo-simple', {
   props: ['size', 'myMessage']
})
```

Приклад 2

```
Vue.component('props-demo-advanced', {
   props: {
      // just type check
      height: Number,

      // type check plus other validations
      age: {
         type: Number,
         default: 0,
         required: true,
         validator: function (value) {
            return value >= 0
         }
      }
   }
})
```

propsData - це використовується для тестування пристроїв.

Type - масив рядка. Наприклад, {[key: string]: будь-який}. Вона повинна бути передана під час створення екземпляра Vue.

Приклад

```
var Comp = Vue.extend({
   props: ['msg'],
   template: '<div>{{ msg }}</div>'
})
var vm = new Comp({
   propsData: {
      msg: 'hello'
   }
})
```

Обчислено - Тип: {[key: string]: Функція | {get: Function, set: Function}}

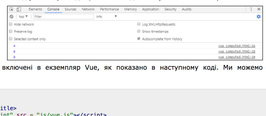
Приклад

```
<html>
   <head>
      <title>VueJs Introduction</title>
      <script type = "text/javascript" src = "js/vue.js"></script>
   </head>
   <body>
      <script type = "text/javascript">
         var vm = new Vue({
            data: { a: 2 },
            computed: {

               // get only, just need a function
               aSum: function () {
                  return this.a + 2;
               },

               // both get and set
               aSquare: {
                  get: function () {
                     return this.a*this.a;
                  },
                  set: function (v) {
                     this.a = v*2;
                  }
               }
            }
         })
         console.log(vm.aSquare);  // -> 4
         vm.aSquare = 3;           // -> 6
         console.log(vm.a);        // -> 6
         console.log(vm.aSum); // -> 8
      </script>
   </body>
</html>
```

Комп'ютер має дві функції **aSum** і **aquare** .

Функція aSum просто повертає це. **A + 2** . Функція aSquare знову **отримує** і **встановлює** дві функції.

Змінна vm є екземпляром Vue, і він викликає aquare і aSum. Також vm.aSquare = 3 викликає задану функцію з aquare і vm.aSquare називає функцію get. Ми можемо перевірити вихід у веб-переглядачі, який виглядає наступним знімком екрана.



Методи - методи повинні бути включені в екземпляр Vue, як показано в наступному коді. Ми можемо отримати доступ до цієї функції, використовуючи об'єкт Vue.

```
<html>
   <head>
      <title>VueJs Introduction</title>
      <script type = "text/javascript" src = "js/vue.js"></script>
   </head>
   <body>
      <script type = "text/javascript">
         var vm = new Vue({
            data: { a: 5 },
            methods: {
               asquare: function () {
                  this.a *= this.a;
               }
            }
         })
         vm.asquare();
         console.log(vm.a); // 25
      </script>
   </body>
</html>
```

Методи є частиною конструктора Vue. Зателефонуйте методу, використовуючи об'єкт Vue **vm.asquare ()** , значення властивості **a** оновлюється в функції **aquare** . Значення а змінюється від 1 до 25, і те, що видно, відображається на наступній консолі веб-переглядача.