How to Pass a Function as a Prop in Vue

It’s a pretty common question that newer Vue developers often ask.

You can pass strings, arrays, numbers, and objects as props.

But can you pass a function as a prop?

While you can pass a function as a prop, this is almost always a bad idea. Instead, there is probably a feature of Vue that is designed exactly to solve your problem.

If you keep reading you’ll see what I mean.

In this chapter I’ll show you:

- How to pass a function as a prop — even though you probably shouldn’t
- Why React and Vue are different when it comes to passing methods as props
- Why events or scoped slots might be better solutions
- How you can access a parent’s scope in the child component

First, even though I probably shouldn’t, I’ll show you how to do it.

Passing a function as a prop

Taking a function or a method and passing it down to a child component as a prop is relatively straightforward. In fact, it is exactly the same as passing any other variable:

```
<template>
  <ChildComponent :function="myFunction" />
</template>
```

```
export default {
  methods: {
    myFunction() {
      // ...
    }
  }
};
```

But as I said earlier, this is something you shouldn't ever be doing in Vue.

Why?

Well, Vue has something better...

React vs Vue

If you're coming from React you're used to passing down functions all of the time.

In React you'll pass a function from a parent to a child component, so the child can communicate back up to the parent. Props and data flow down, and function calls flow up.

Vue, however, has a different mechanism for achieving child -> parent communication.

We use events in Vue.

This works in the same way that the DOM works — providing a little more consistency with the browser than what React does. Elements can emit events, and these events can be listened to.

So even though it can be tempting to pass functions as props in Vue, it's considered an anti-pattern.

Using events

Events are how we communicate to parent components in Vue.

Here is a short example to illustrate how events work.

First we'll create our child component, which emits an event when it is created:

```
// ChildComponent
export default {
  created() {
    this.$emit('created');
  }
}
```

In our parent component, we will listen to that event:

```
<template>
  <ChildComponent @created="handleCreate" />
</template>
```

```
export default {
  methods: {
    handleCreate() {
      console.log('Child has been created.');
    }
  }
};
```

There is a lot more that can be done with events, and this only scratches the surface. I would highly recommend that you check out the official Vue docs to learn more about events. Definitely worth the read!

But events don't quite solve **all** of our problems.

Accessing a parent's scope from the child component

In many cases the problem you are trying to solve is accessing values from different scopes.

The parent component has one scope, and the child component another.

Often you want to access a value in the child component from the parent, or access a value in the parent component from the child.

Vue prevents us from doing this directly, which is a good thing.

It keeps our components more encapsulated and promotes their reusability. This makes your code cleaner and prevents lots of headaches in the long run.

But you may be tempted to try and pass functions as props to get around this.

Getting a value from the parent

If you want a child component to access a parent's method, it seems obvious to just pass the method straight down as a prop.

Then the child has access to it immediately, and can use it directly.

In the parent component we would do this:

```
<!-- Parent -->
<template>
  <ChildComponent :method="parentMethod" />
</template>
```

```
// Parent
export default {
  methods: {
    parentMethod() {
      // ...
    }
  }
}
```

And in our child component we would use that method:

```
// Child
export default {
  props: {
    method: { type: Function },
  },
  mounted() {
    // Use the parent function directly here
    this.method();
  }
}
```

What's wrong with that?

Well, it's not exactly **wrong**, but it's much better to use events in this case.

Then, instead of the child component calling the function when it needs to, it will simply emit an event. Then the parent will receive that event, call the function, and then the props that are passed down to the child component will be updated.

This is a much better way of achieving the same effect.

Getting a value from the child

In other cases we may want to get a value from the child into the parent, and we're using functions for that.

For example, you might be doing this. The parent's function accepts the value from the child and does something with it:

```
<!-- Parent -->
<template>
  <ChildComponent :method="parentMethod" />
</template>
```

```
// Parent
export default {
  methods: {
    parentMethod(valueFromChild) {
      // Do something with the value
      console.log('From the child:', valueFromChild);
    }
  }
}
```

Where in the child component you pass the value in when calling it:

```
// Child
export default {
  props: {
    method: { type: Function },
  },
  data() {
    return { value: 'I am the child.' };
  },
  mounted() {
    // Pass a value to the parent through the function
    this.method(this.value);
  }
}
```

Again, this isn't completely **wrong**.

It will work to do things this way.

It's just that it isn't the best way of doing things in Vue. Instead, events are much better suited to solving this problem.

We can achieve this exact same thing using events instead:

```
<!-- Parent -->
<template>
  <ChildComponent @send-message="handleSendMessage" />
</template>
```

```
// Parent
export default {
  methods: {
    handleSendMessage(event, value) {
      // Our event handler gets the event, as well as any
      // arguments the child passes to the event
      console.log('From the child:', value);
    }
  }
}
```

And in the child component we emit the event:

```
// Child
export default {
  props: {
    method: { type: Function },
  },
  data() {
    return { value: 'I am the child.' };
  },
  mounted() {
    // Instead of calling the method we emit an event
    this.$emit('send-message', this.value);
  }
}
```

Events are extremely useful in Vue, but they don't solve 100% of our problems either.

There are times when we need to access a child's scope from the parent in a different way.

For that, we have scoped slots!

Using scoped slots instead

Scoped slots are a more advanced topic, but they are also incredibly useful.

In fact, I would consider them to be one of the most powerful features that Vue offers.

They let you blur the lines between what is in the child's scope and what is in the parent's scope. But it's done in a very clean way that leaves your components as composable as ever.

If you want to learn more about how scoped slots work — and I really think that you should — check out Adam Wathan's great post on the subject.