# Computed Props and Watchers — What's the Difference?

Computed properties and watchers are two of the most fundamental concepts in Vue.

But I often see people mixing them up — using one when they should be using the other.

They have a lot of similarities, so it can be hard knowing which one is best for what you're trying to accomplish.

In this chapter you'll learn:

- What a watcher is, and what a computed prop is
- Common use cases for both
- The differences and similarities between them
- How to know which to use

## What is a watcher?

When building components in Vue, we often need to respond to changes in our props.

A watcher — or watched prop — let's us **track a property** on our component and **run a function whenever it changes**.

For example, if the prop `colour` changes, we can decide to log something to the console:

```
export default {
  name: 'ColourChange',
  props: ['colour'],
  watch: {
    colour()
      console.log('The colour has changed!');
    }
  }
}
```

You can put a watcher on any reactive property. This includes computed props, **props**, as well as data that is specified inside of `data()` on your Vue component.

They're really useful for **creating side effects** — things that don't update your application's state immediately.

If you need to fetch data or do some other **ansychronous action**, watched props are really good for that. Or maybe you need to interact with an imperative browser API, such as `localstorage`.

Because watchers aren't expected to be pure functions, we can do all sorts of things like this with them!

Watched props are really powerful, but many times I accidentally use a watcher when all that I needed was a computed property.

## What is a computed prop?

Computed props let us compose new data from other data.

Let's say we have a component that takes a person's `firstName` and `lastName`. We can create `fullName` as a computed prop:

```
export default {
  name: 'FullName',
  props: ['firstName', 'lastName'],
  computed: {
    fullName() {
      return this.firstName + ' ' + this.lastName;
    }
  }
}
```

Because computed props are **reactive**, whenever `firstName` or `lastName` are changed, `fullName` will be recomputed and will always be up to date.

Already we can see an advantage of computed props over watchers.

Composing this data together **can** be done with watchers, but it's much cleaner and **more declarative** using computed props:

```
export default {
  name: 'FullName',
  props: ['firstName', 'lastName'],
  data() {
    return {
      fullName: this.firstName + ' ' + this.lastName,
    };
  },
  watched: {
    firstName() {
      this.fullName = this.firstName + ' ' + this.lastName;
    },
    lastName() {
      this.fullName = this.firstName + ' ' + this.lastName;
    }
  }
}
```

This is a fairly simple example — computed props can depend on as many other properties as you need them to. You can even watch other computed props, as well as reactive data in `data()`!

Computed properties are required to be **pure functions**.

This means that they return a new value, but aren't allowed to change anything, and they must be synchronous.

Once we've created our computed prop, we can access it like we would any other prop. This is because computed props are reactive properties, along with regular props and data.

Now that we've covered how to use watchers and computed props, **where do we use them?**

## Common use cases for a watcher

The most common use case for a watched prop is in **creating side effects.**

A **side effect** is anything that affects state outside of the component, or affects state in an asynchronous way.

Common examples are:

- Fetching data
- Manipulating the DOM
- Using a browser API, such as local storage or audio playback

None of these things affect your component directly, so they are considered to be **side effects**.

We’ll look at an example of fetching data.

Let’s say we have a `MovieData` component, and it fetches data from the server based on what the `movie` prop is set to:

```
export default {
  name: 'MovieData',
  props: {
    movie: {
      type: String,
      required: true,
    }
  },
  data() {
    return {
      movieData: {},
    }
  },

  watch: {
    // Whenever the movie prop changes, fetch new data
    movie(movie) {
      // Fetch data about the movie
      fetch(`/${movie}`).then((data) => {
        this.movieData = data;
      });
    }
  }
}
```

Now, this component will work wonderfully. Whenever we change the `movie` prop, our watcher will fire, and it will fetch us the new data.

You can also use watchers to recompute data in certain cirumstances.

While computed props are generally the better way of doing this (as we’ll see in a moment), there are cases where it doesn’t make sense to use a computed prop.

> **NOTE: Be careful with using a `watch` to update state. This means that both your component and the parent component are updating — directly or indirectly — the same state. This can get very ugly very fast.**

## Common use cases for computed props

The main use case for computed props, as discussed before, is in composing data from other data.

This use case is pretty straightforward, but extremely useful. In my Vue code I use computed props far more than I use watchers, and almost always they are used to compose new data together.

I’ll say this again because it’s so important.

**Using computed props to compose new data is one of the most useful patterns to learn when building Vue applications**.

I use them literally everywhere.

Another one is allowing us **easier access to nested data**.

If we had some deeply nested data in our component that we needed access to, like `data.nested.really.deeply.importantValue`, we can simplify this a whole lot:

```
computed() {
  importantValue() {
    return this.data.nested.really.deeply.importantValue;
  },
}
```

And now we only need to write `this.importantValue` to get ahold of it!

## Differences and Similarities

Let’s see how they are different:

- Computed props are **more declarative** than watched properties
- Computed props should be **pure**: return a value, synchronous, and have no side-effects

- Computed props create new **reactive properties**, watched props only call functions
- Computed props can react to changes in **multiple props**, whereas watched props can only watch one at a time
- Computed props are **cached**, so they only recalculate when things change. Watched props are executed every time
- Computed props are **evaluated lazily**, meaning they are only executed when they are needed to be used. Watched props are executed whenever a prop changes

But they also have some similarities.

They aren’t exactly twins, but they both:

- **react to changes** in properties
- can be used to **compute new data**

But how do you know **which one to use** in a given situation?

## Which one do you use?

Watched properties can often be confused with computed properties, because they operate in a similar way.

It’s even trickier to know when to use which one.

But I’ve come up with a good rule of thumb.

**Watch is for side effects. If you need to change state you want to use a computed prop instead.**

Most of the time you’ll want a computed prop, so **try to use that first**. If it doesn’t work, or results in weird code that makes you feel like taking a shower, then you should switch to using a watched prop.

## Learn more

If you want to learn more about this, Sarah Drasner has a great article comparing computed props, watchers, and methods.

You can also check out the official documentation, which addresses this issue specifically.