Computed Values & Vue 3 Source

As we've been building out our Reactivity example, you may have been wondering "why haven't we used `computed` for our values where we've been using `effect`?" When looking at our example:

```
let product = reactive({ price: 5, quantity: 2 })
let salePrice = ref(0)
let total = 0
effect(() => {
  salePrice.value = product.price * 0.9
})
effect(() => {
  total = salePrice.value * product.quantity
})
```

It seems clear to me that if I was coding up Vue I would write both `salePrice` and `total` as computed properties. If you're familiar with the Vue 3 composition API, you're probably familiar with the `computed` syntax. If you're not, maybe take our Vue 3 Essentials course. We might use the computed syntax like so (even though we haven't defined it yet):

```
let product = reactive({ price: 5, quantity: 2 })

let salePrice = computed(() => {
  return product.price * 0.9
})
let total = computed(() => {
  return salePrice.value * product.quantity
})
```

Makes sense right? And notice how my `salePrice` computed property is included inside my `total` computed property, and that we access it using `.value`. This is our first clue to

implementation. It looks like we're creating another reactive reference. Here's how we create our computed function:

```
function computed(getter) {
  let result = ref()  // Create a new reactive
reference

  effect(() => (result.value = getter())) // Set this
value equal to the return value of the getter

  return result // return the reactive reference
}
```

That's all there is to it. You can view / run the code in its entirety over on Github. Our code prints out:

```
Before updated quantity total (should be 9) = 9
salePrice (should be 4.5) = 4.5
After updated quantity total (should be 13.5) = 13.5
salePrice (should be 4.5) = 4.5
After updated price total (should be 27) = 27 salePrice
(should be 9) = 9
```

Vue Reactivity without a Caveat

It's worth mentioning that we can do something with our reactive objects that was impossible with Vue 2. Specifically, we can add new reactive properties. Like so:

```
...
let product = reactive({ price: 5, quantity: 2 })
...

product.name = 'Shoes'
effect(() => {
  console.log(`Product name is now ${product.name}`)
})
product.name = 'Socks'
```

As you might expect, it prints out:

```
Product name is now Shoes
Product name is now Socks
```

This was impossible with Vue 2 because of how Reactivity was implemented, adding getters and setters to individual object properties using `Object.defineProperty`. Now with `Proxy` we can add new properties no problem and they're instantly reactive.

Testing our code against Vue 3 Source

You might be wondering, would this code work against Vue 3 source? So I cloned the vue-next repo (currently alpha 5), ran `yarn install`, then `yarn build reactivity`. This gave me a bunch of files in my `packages/reactivity/dist/` . I then took the `reactivity.cjs.js` file I found there and moved it next to my example files, the ones on github, and wrote this up to use Vue's Reactivity code:

```
var { reactive, computed, effect } =
require('./reactivity.cjs')

// Exactly the same code here from before, without the
definitions

let product = reactive({ price: 5, quantity: 2 })

let salePrice = computed(() => {
  return product.price * 0.9
})

let total = computed(() => {
  return salePrice.value * product.quantity
})

console.log(
```

```
  `Before updated quantity total (should be 9) =
${total.value} salePrice (should be 4.5) =
${salePrice.value}`
)
product.quantity = 3
console.log(
  `After updated quantity total (should be 13.5) =
${total.value} salePrice (should be 4.5) =
${salePrice.value}`
)
product.price = 10
console.log(
  `After updated price total (should be 27) =
${total.value} salePrice (should be 9) =
${salePrice.value}`
)
product.name = 'Shoes'
effect(() => {
  console.log(`Product name is now ${product.name}`)
})
product.name = 'Socks'
```

Ran this with `node 08-vue-reactivity.js`, and as expected, I got all the same results!

```
Before updated quantity total (should be 9) = 9
salePrice (should be 4.5) = 4.5
After updated quantity total (should be 13.5) = 13.5
salePrice (should be 4.5) = 4.5
After updated price total (should be 27) = 27 salePrice
(should be 9) = 9
Product name is now Shoes
Product name is now Socks
```

Wow, so our reactivity system works as well as Vue! Well, at a basic level … yes, but in reality Vue's version is MUCH more complex. Let's take a look through the files that make up Vue 3's Reactivity system, to begin to get familiar.

Vue 3 Reactivity Files

If we take a look inside the Vue 3 source, inside `/packages/reactivity/src/` we'll find the following files. Yes they are TypeScript (ts) files, but you should be able to read them (even if you don't know TypeScript).

- **effect.ts -** Defines the `effect` function to encapsulate code that may contain reactive references and objects. Contains `track` which is called from `get` requests and `trigger` which is called from `set` requests.
- **baseHandlers.ts -** Contains the `Proxy` handlers like `get` and `set`, which call `track` and `trigger` (from effect.ts).
- **reactive.ts -** Contains the functionality for the reactive syntax which creates an ES6 Proxy using `get` and `set` (from basehandlers.ts).
- **ref.ts -** Defines how we create Reactive **Ref**erences, using object accessors (like we did). Also contains `toRefs` which converts reactive objects into a series of reactive references which access the original proxy.
- **computed.ts -** Defines the `computed` function using `effect` and object accessors (a little different than we did).

There are a few more additional files, but these carry the core functionality. If you're feeling like a challenge, you may want to dive into the source code. In the next two lessons we'll speak to Evan You about how he implemented Reactivity in Vue 3 and give us a tour of the source himself.