&mut self

Kevin Martin S1’18

&mut self

Kevin Martin S1'18


struct Dog {
awoo: String,
}
impl Dog {
fn new() -> Self {
Self { awoo: String::from("awoo") }
}
}

```rust
struct Dog {
    awoo: String,
}

impl Dog {
    fn new() -> Self {
        Self { awoo: String::from("awoo") }
    }

    fn awoo(&self) {
        println!("{}", self.awoo);
    }



}
```

```rust
struct Dog {
    awoo: String,
}

impl Dog {
    fn new() -> Self {
        Self { awoo: String::from("awoo") }
    }

    fn awoo(&self) {
        println!("{}", self.awoo);
    }

    fn awooo(&mut self) {
        self.awoo.push('o');
        self.awoo();
    }
}
```

What do we do?

What we do

What we do

The Recurse Center runs educational programming retreats in New York City. The retreats are free, self-directed, project based, and for anyone who wants to get dramatically better at programming.

What did I do?

What did I do?

I became a dramatically better programmer!

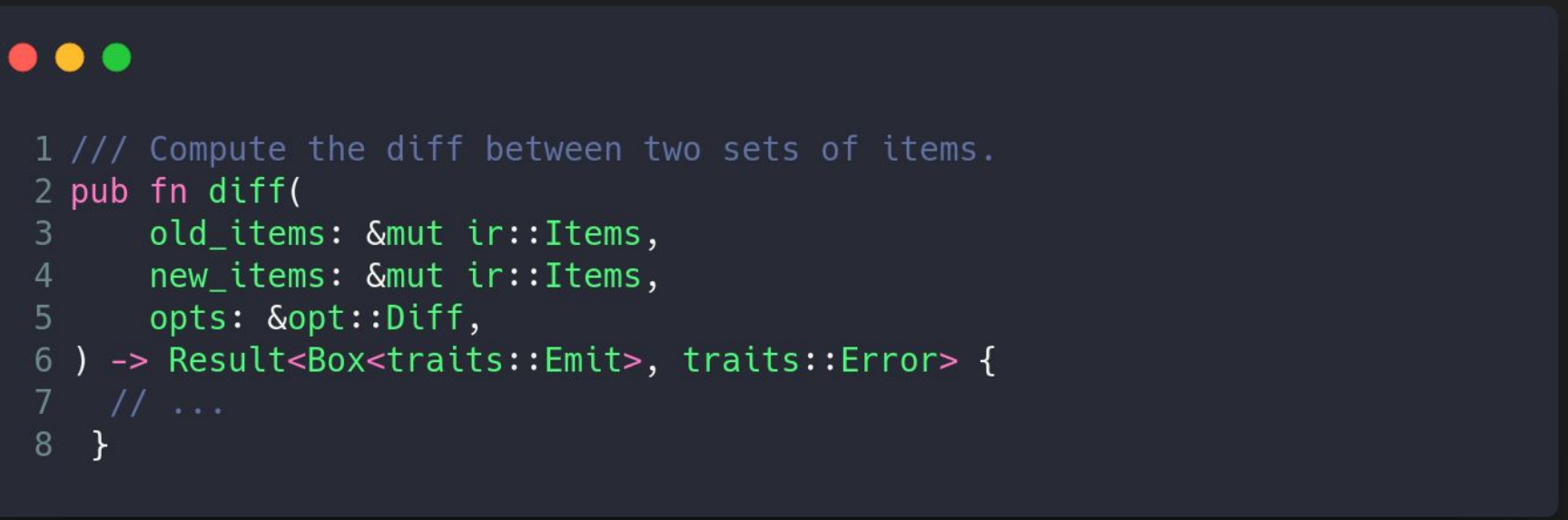
How?

How?

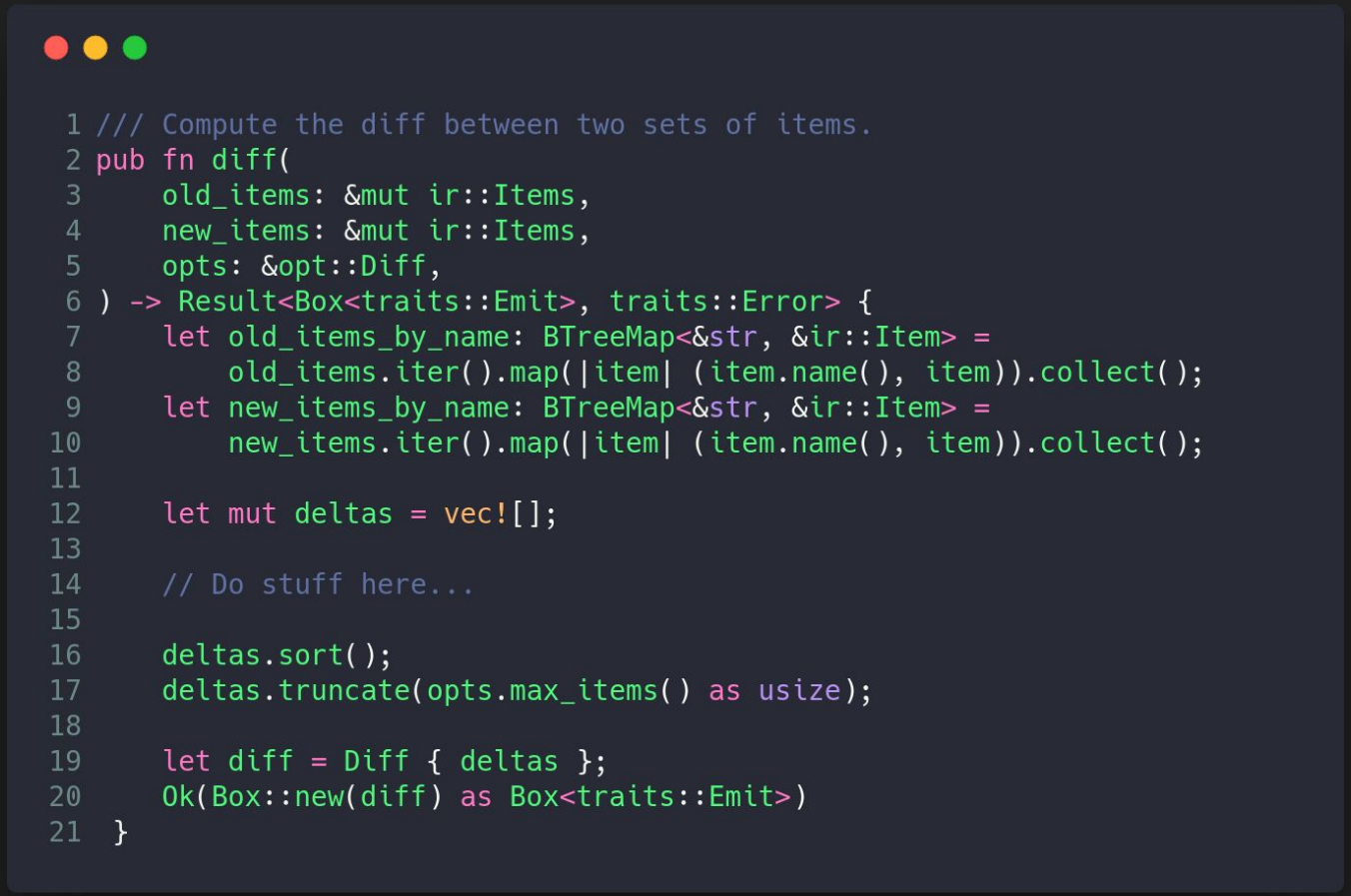
How?

There might not be an easy answer to that question.

So let’s talk about Rust instead!



```rust
/// Compute the diff between two sets of items.
pub fn diff(
    old_items: &mut ir::Items,
    new_items: &mut ir::Items,
    opts: &opt::Diff,
) -> Result<Box<traits::Emit>, traits::Error> {
  // ...
 }
```



```rust
/// Compute the diff between two sets of items.
pub fn diff(
    old_items: &mut ir::Items,
    new_items: &mut ir::Items,
    opts: &opt::Diff,
) -> Result<Box<traits::Emit>, traits::Error> {
    let old_items_by_name: BTreeMap<&str, &ir::Item> =
        old_items.iter().map(|item| (item.name(), item)).collect();
    let new_items_by_name: BTreeMap<&str, &ir::Item> =
        new_items.iter().map(|item| (item.name(), item)).collect();

    let mut deltas = vec![];

    // Do stuff here...

    deltas.sort();
    deltas.truncate(opts.max_items() as usize);

    let diff = Diff { deltas };
    Ok(Box::new(diff) as Box<traits::Emit>)
}
```

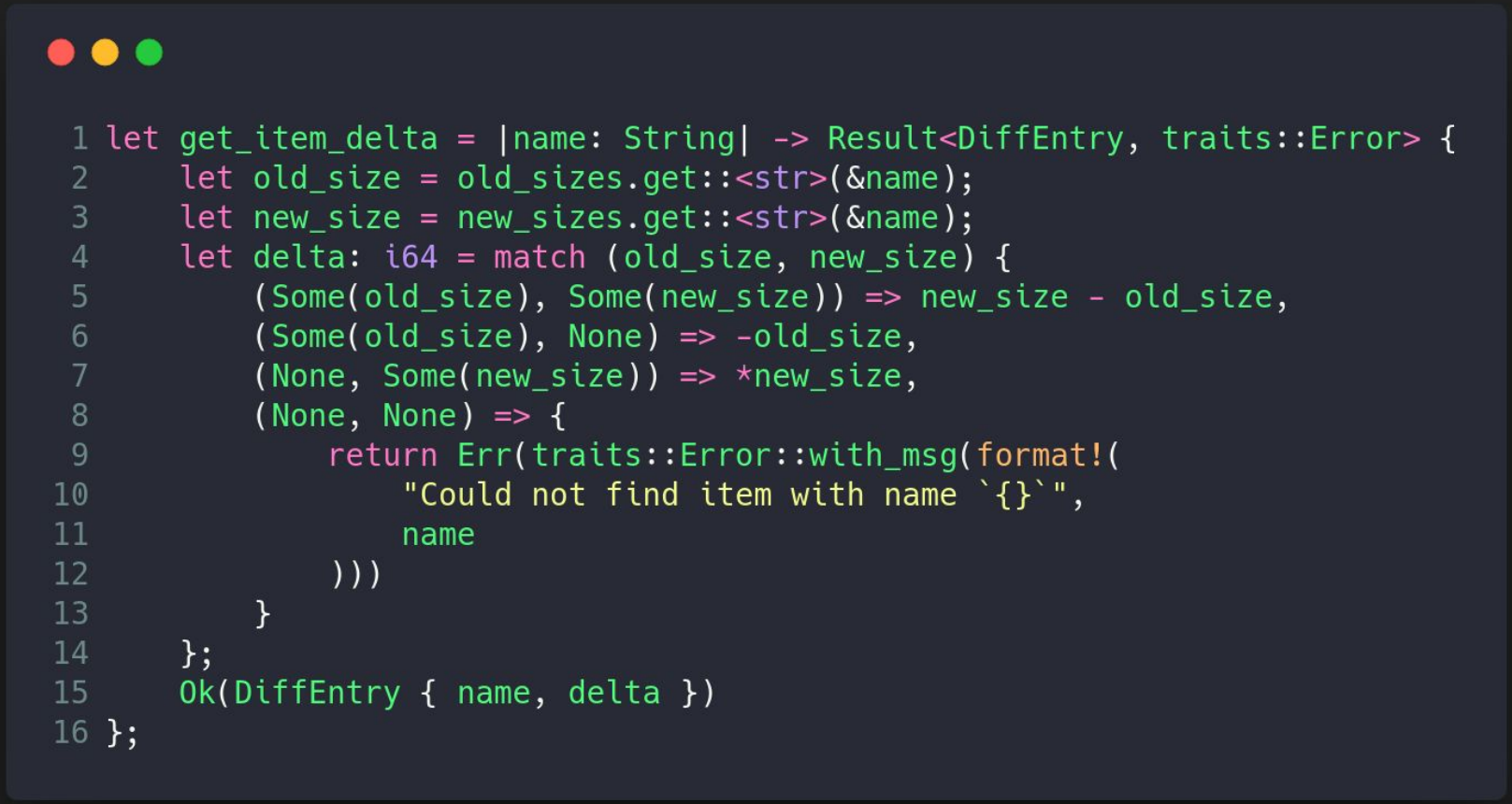
Calculating a diff imperatively...

```
for (name, old_item) in &old_items_by_name {
    match new_items_by_name.get(name) {
        None => deltas.push(DiffEntry {
            name: name.to_string(),
            delta: -(old_item.size() as i64),
        }),
        Some(new_item) => {
            let delta = new_item.size() as i64 - old_item.size() as i64;
            if delta != 0 {
                deltas.push(DiffEntry {
                    name: name.to_string(),
                    delta,
                });
            }
        }
    }
}

for (name, new_item) in &new_items_by_name {
    if !old_items_by_name.contains_key(name) {
        deltas.push(DiffEntry {
            name: name.to_string(),
            delta: new_item.size() as i64,
        });
    }
}

```

Calculating a diff functionally...



```
let get_item_delta = |name: String| -> Result<DiffEntry, traits::Error> {
    let old_size = old_sizes.get::<str>(&name);
    let new_size = new_sizes.get::<str>(&name);
    let delta: i64 = match (old_size, new_size) {
        (Some(old_size), Some(new_size)) => new_size - old_size,
        (Some(old_size), None) => -old_size,
        (None, Some(new_size)) => *new_size,
        (None, None) => {
            return Err(traits::Error::with_msg(format!(
                "Could not find item with name `{}`",
                name
            )))
        }
    };
    Ok(DiffEntry { name, delta })
};
```

```
// Iterate through the set of item names, and use the closure above to map
// each item into a `DiffEntry` object. Then, sort the collection.
let mut deltas = names
    .into_iter()
    .map(get_item_delta)
    .filter(unchanged_items_filter)
    .collect::<Result<Vec<_>, traits::Error>>()?;
deltas.sort();
```

What did I learn?

What did I learn?

- Becoming fluent in a new programming language

What did I learn?

- Becoming fluent in a new programming language

What did I learn?

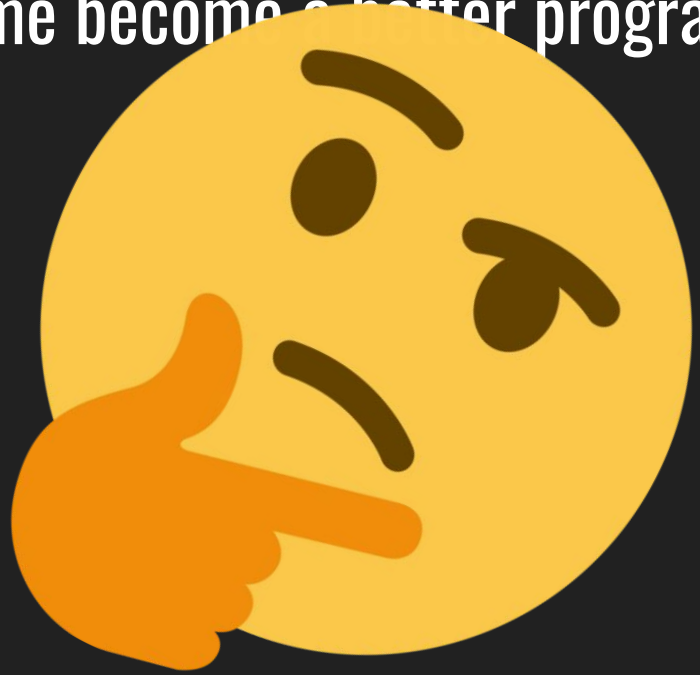
- Becoming fluent in a new programming language
- Working with new programming paradigms

What did I learn?

- Becoming fluent in a new programming language
- Working with new programming paradigms
- Working in previously unexplored domains

How did these things help me become a better programmer?

How did these things help me become a better programmer?




Eval
Apply
λ

“I think that it’s extraordinarily important that we in computer science keep fun in computing. When it started out it was an awful lot of fun. [After a while] we began to feel as if we really were responsible for the successful error-free perfect use of these machines. I don’t think we are.

I think we’re responsible for stretching them setting them off in new directions and keeping fun in the house. I hope the field of computer science never loses its sense of fun. [...]

What you know about computing other people will learn. Don’t feel as if the key to successful computing is only in your hands. What’s in your hands I think and hope is intelligence: the ability to see the machine as more than when you were first led up to it that you can make it more.”

— Alan J. Perlis