

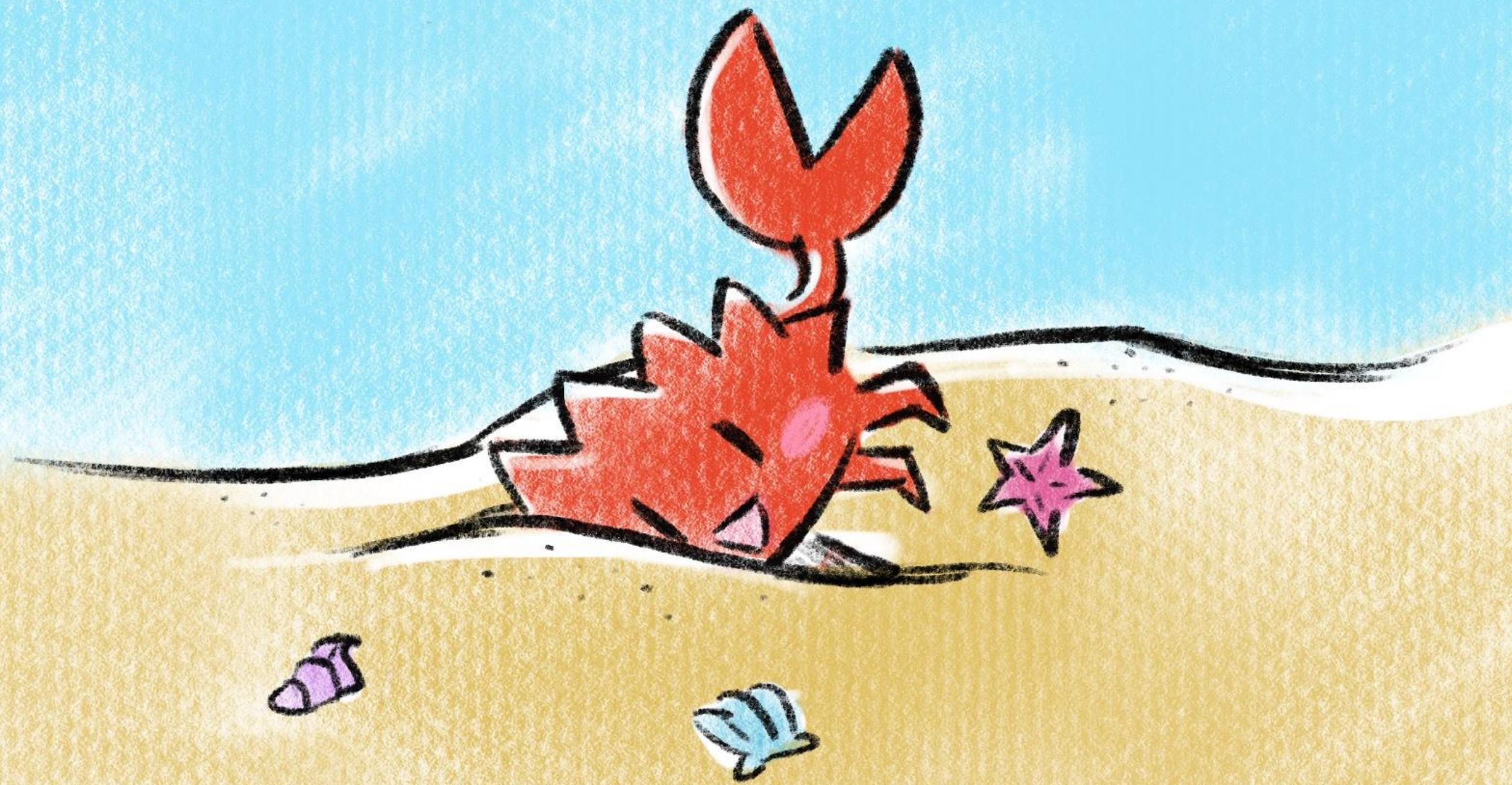


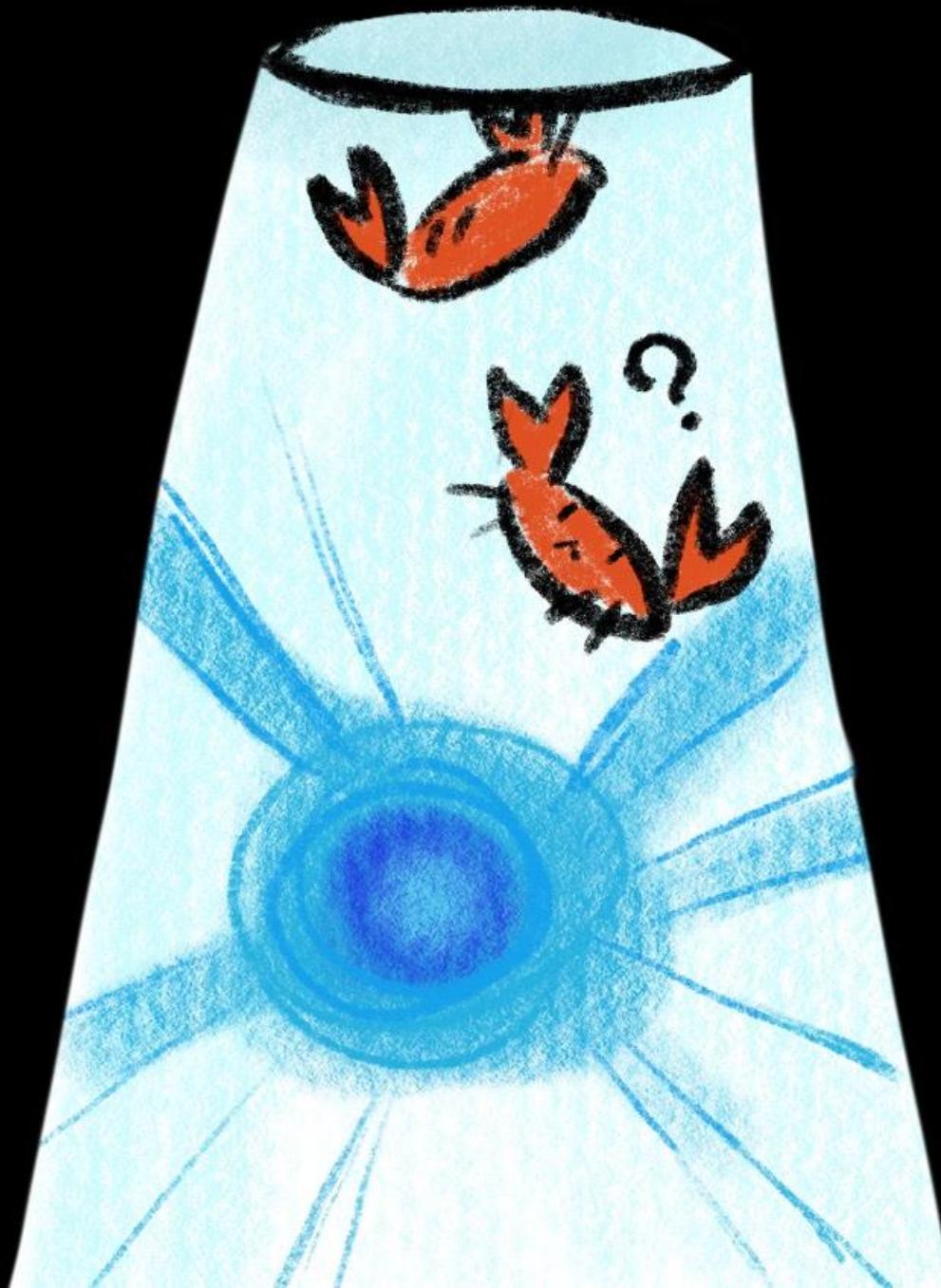
Is this magic!?
Ferris explores rustc!

- J Haigh
- @DebugSteven
- They/them
- Haskell weenie 4lyfe
- Recurse Center (W1 18)
- Colorado Gold Rust

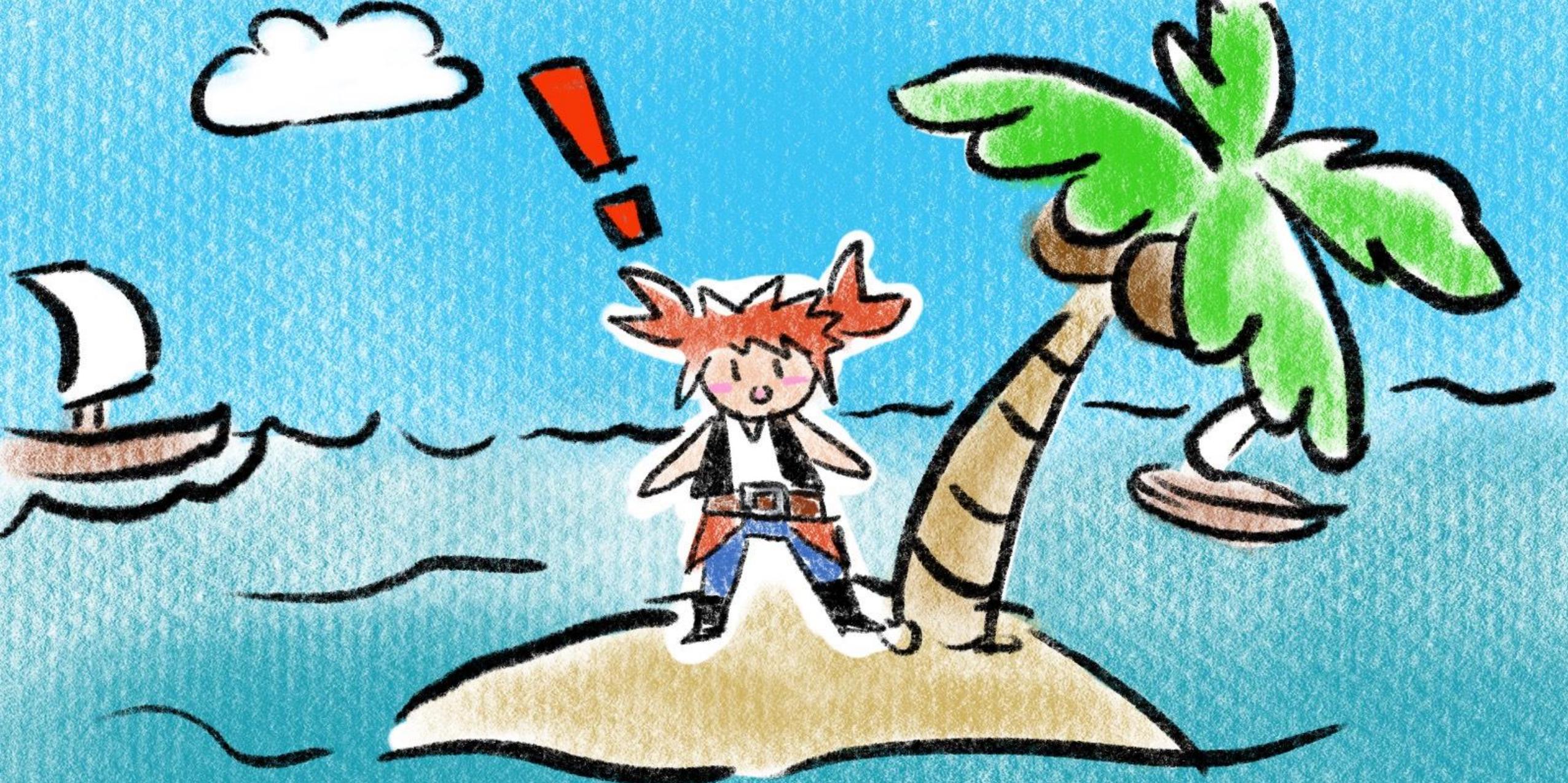
- Grey Mitchell
- @QuietMisdreavus
- She/they
- Rustdoc Team
- Visual Basic fangirl
- Likes docs, a little too much
- Recurse Center (F2 19)















PERSER

+
HOG



CHECK

RESOLVE



LINK

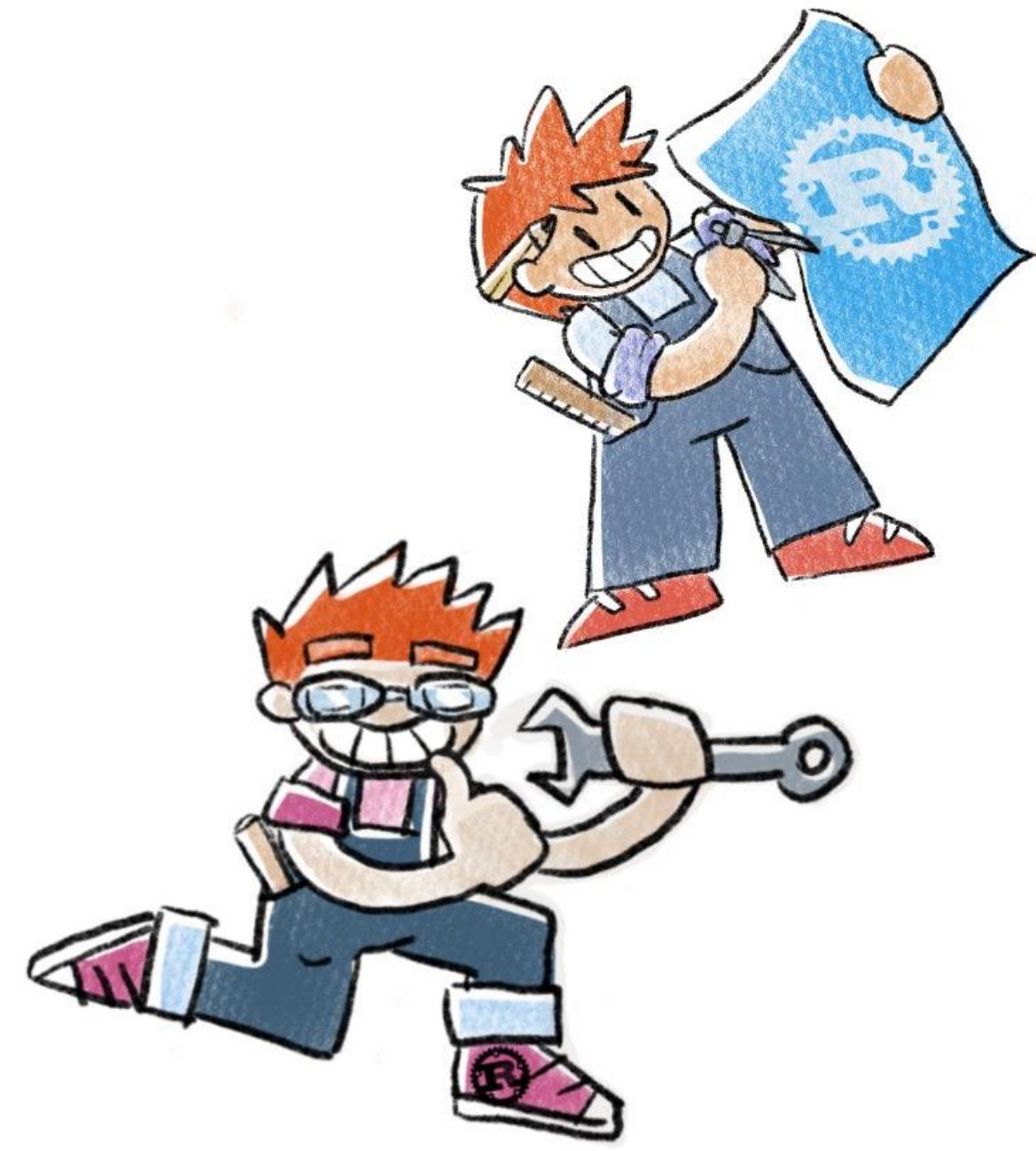


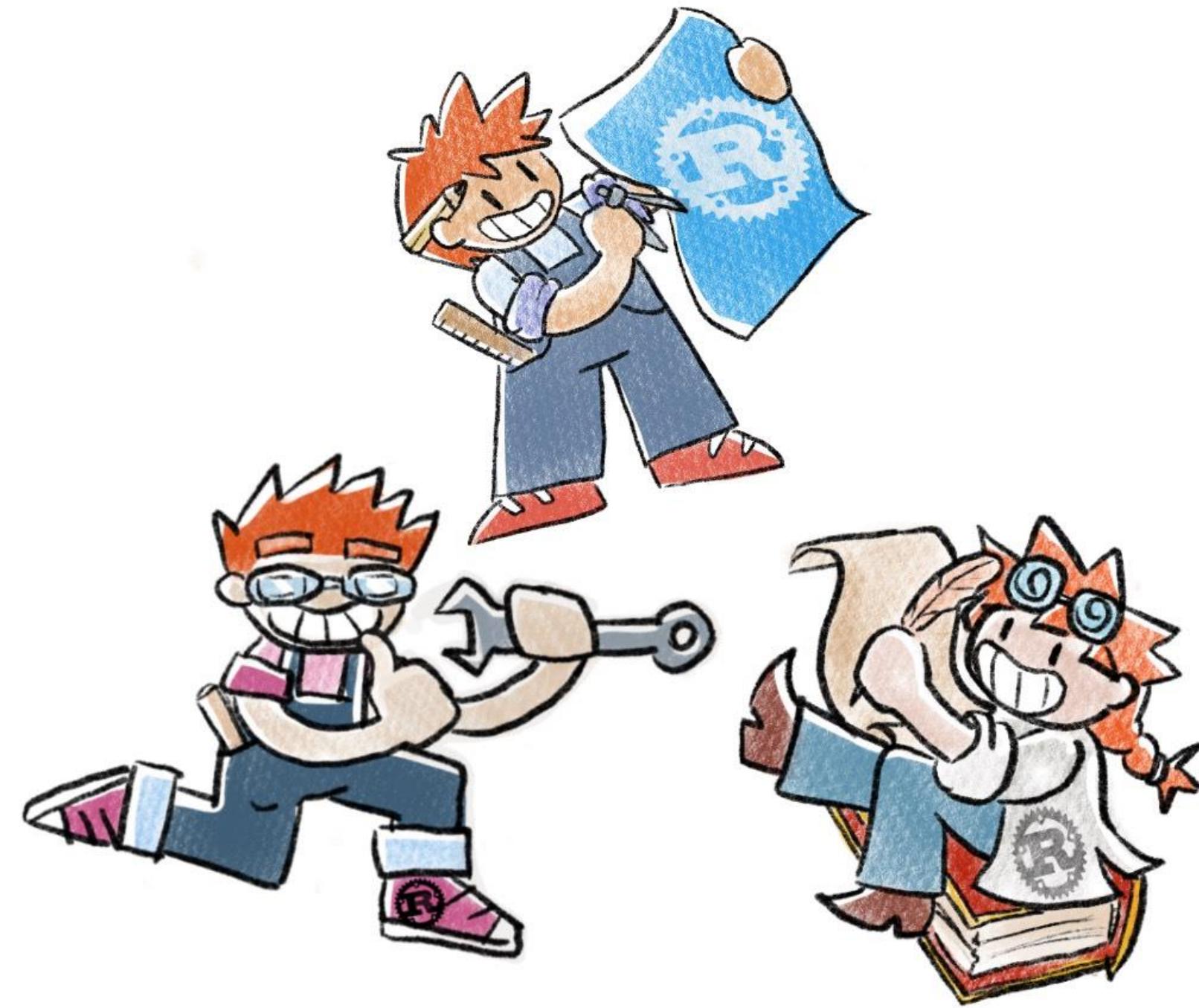
The phases of the compiler

- There are several steps that the compiler takes to go from source code to finished library or binary!
 - Parsing
 - Name resolution
 - Type-checking and borrow-checking
 - Linking to a final binary
 - And many more!
- There are also high/middle/low “Intermediate Representations” that the compiler uses!

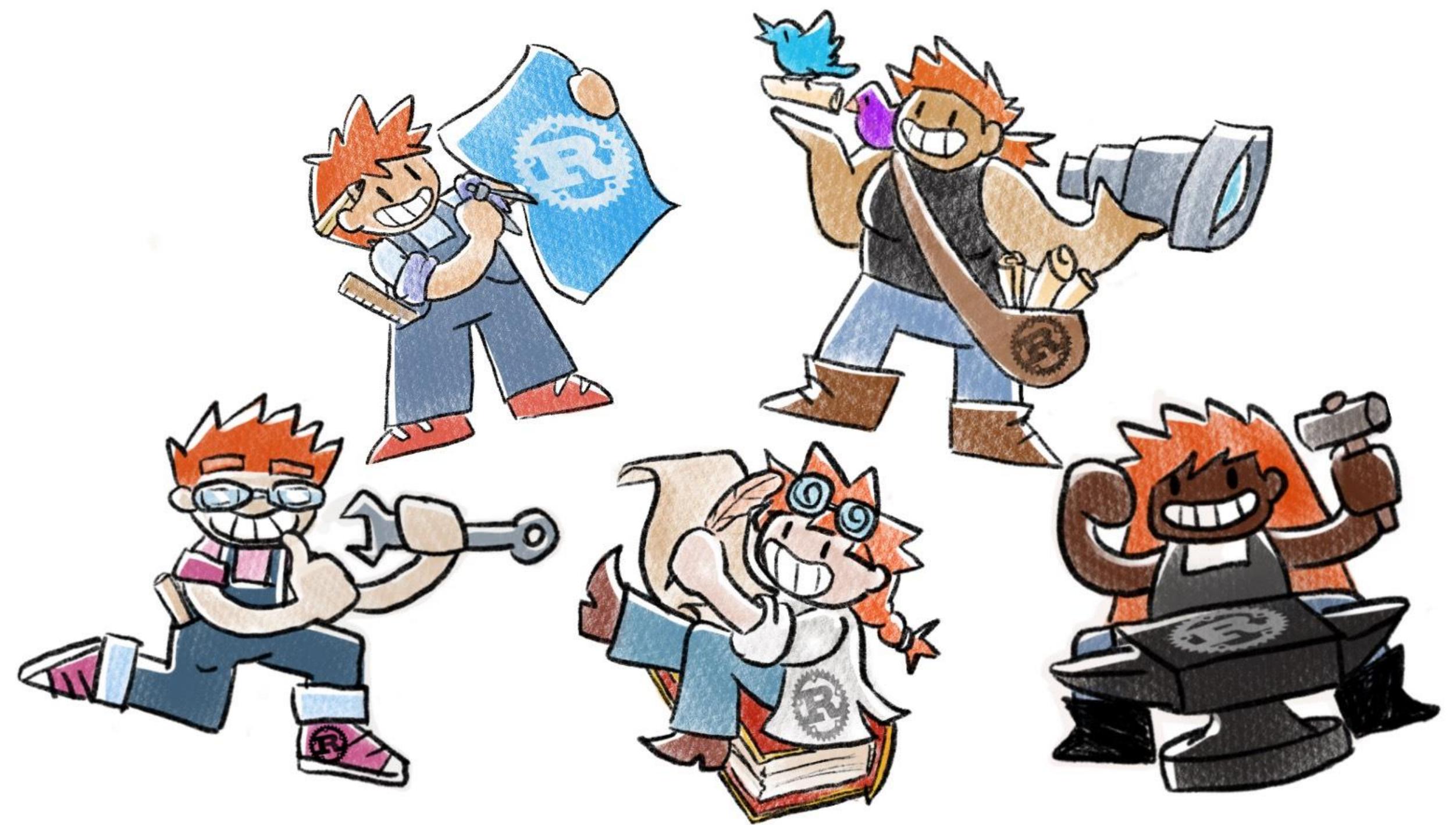




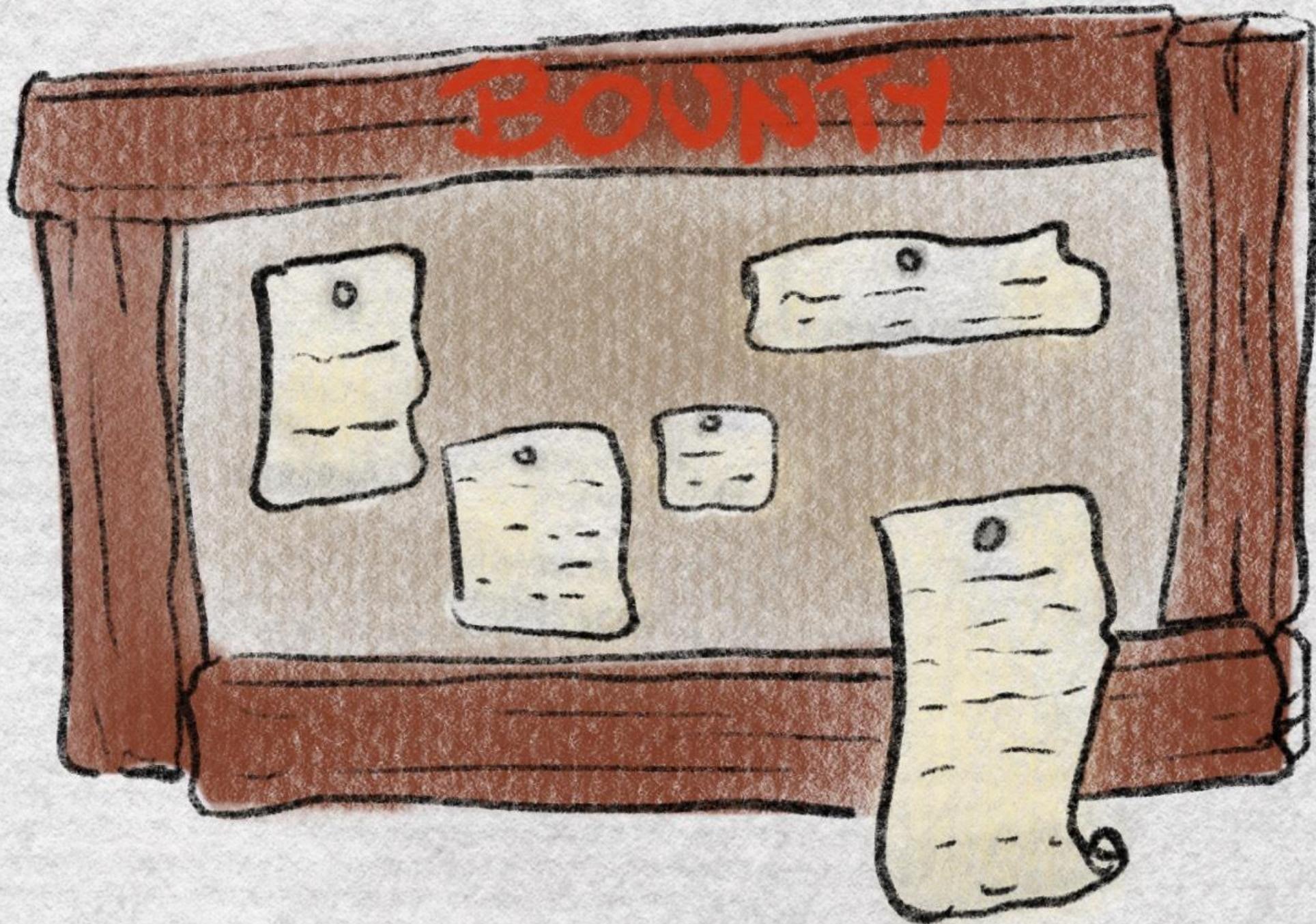








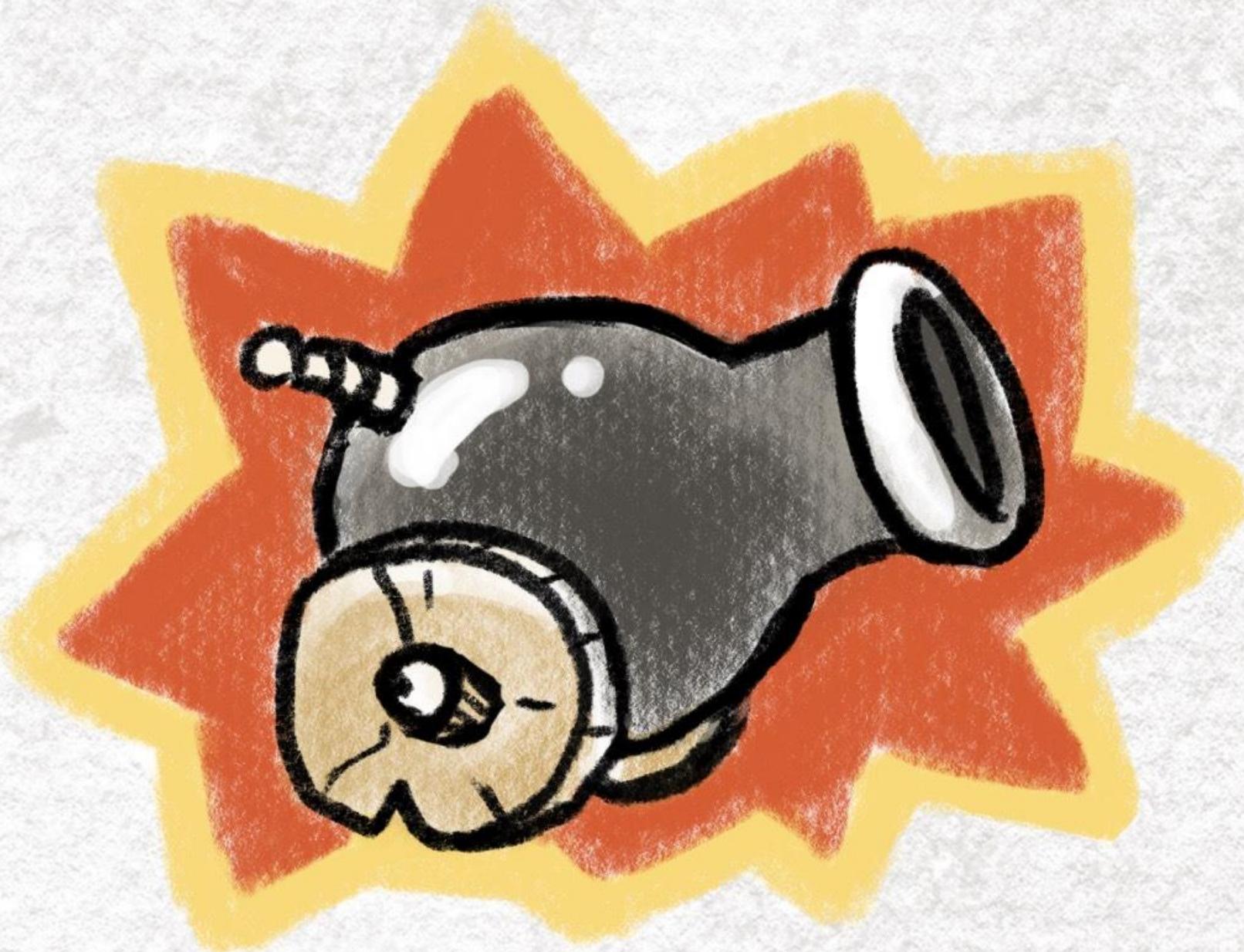


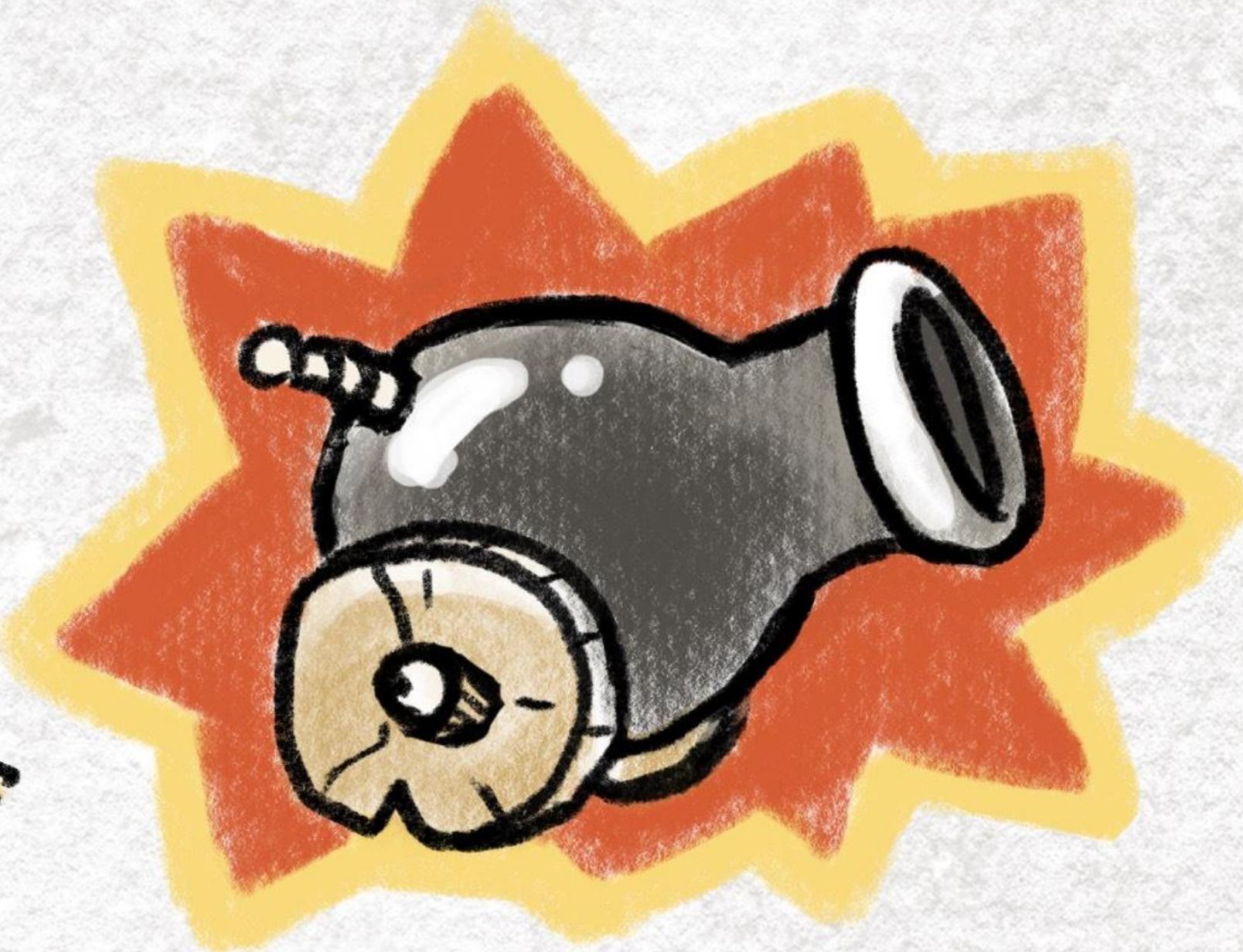


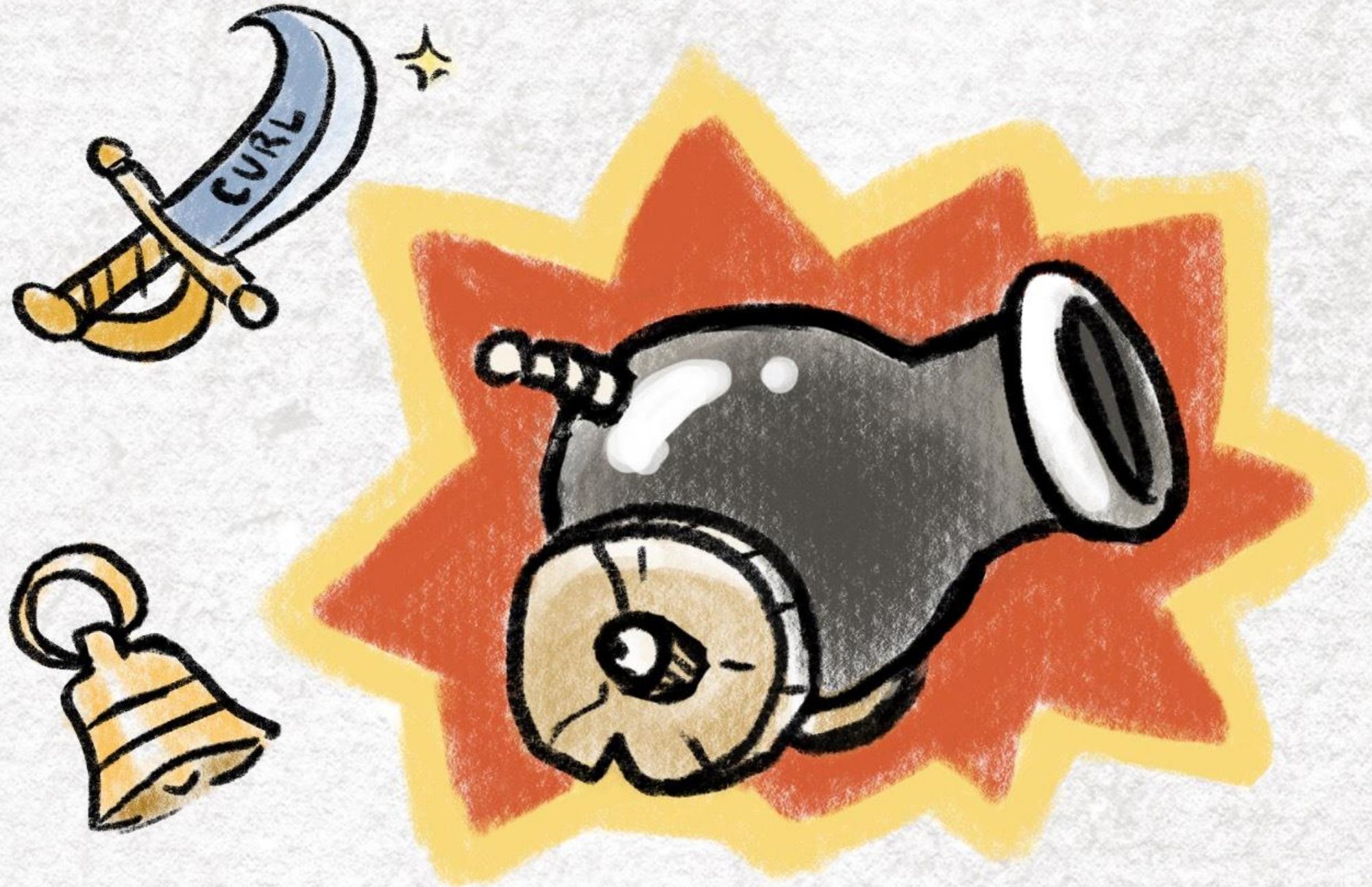


Your First Issue!

- Want to make a contribution? Find an issue on the issue log!
 - Labels like E-easy or E-mentor make great first issues!
- Feeling more social? Go hang out with the team in their chatroom!
 - Some teams use Discord, some use Zulip!





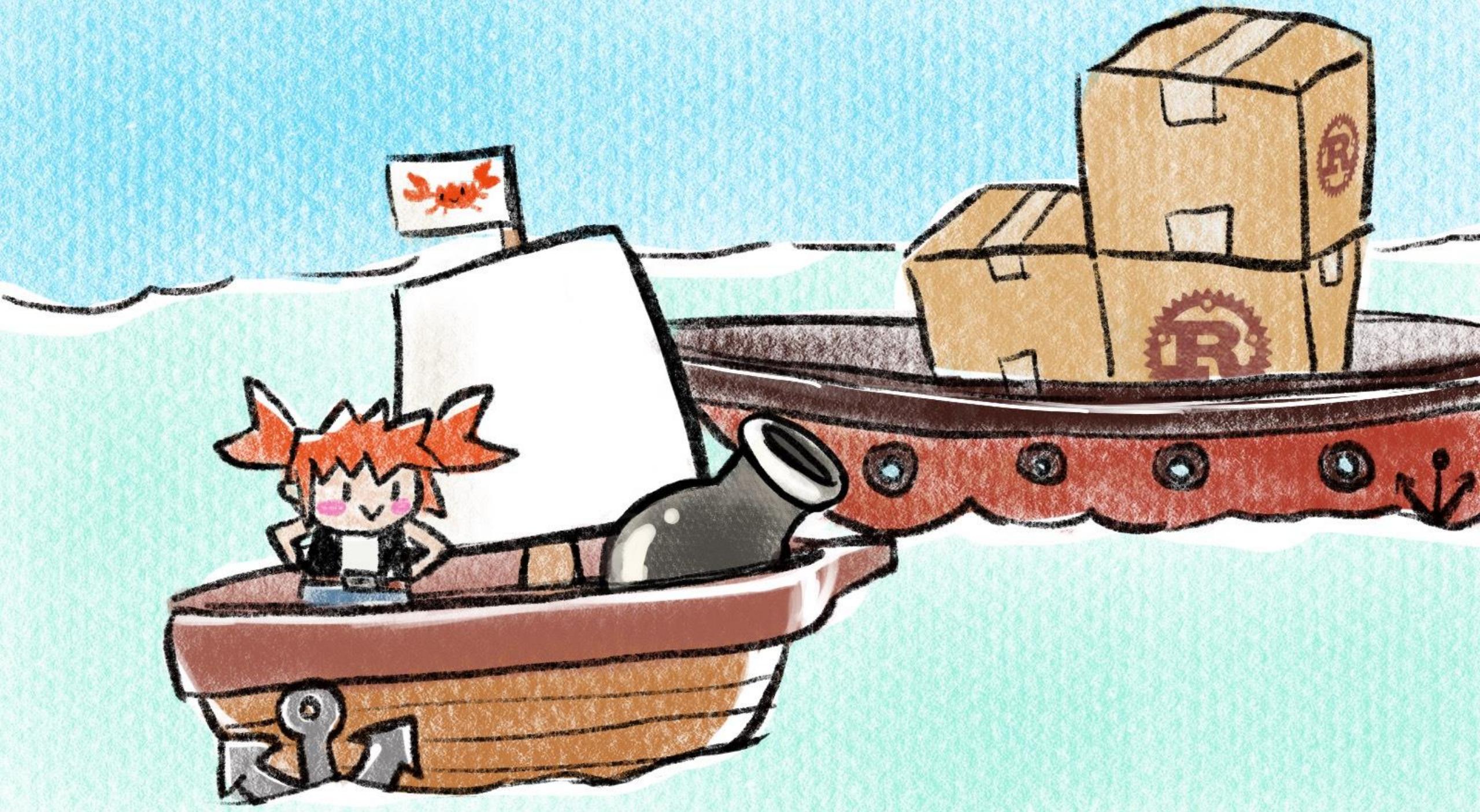


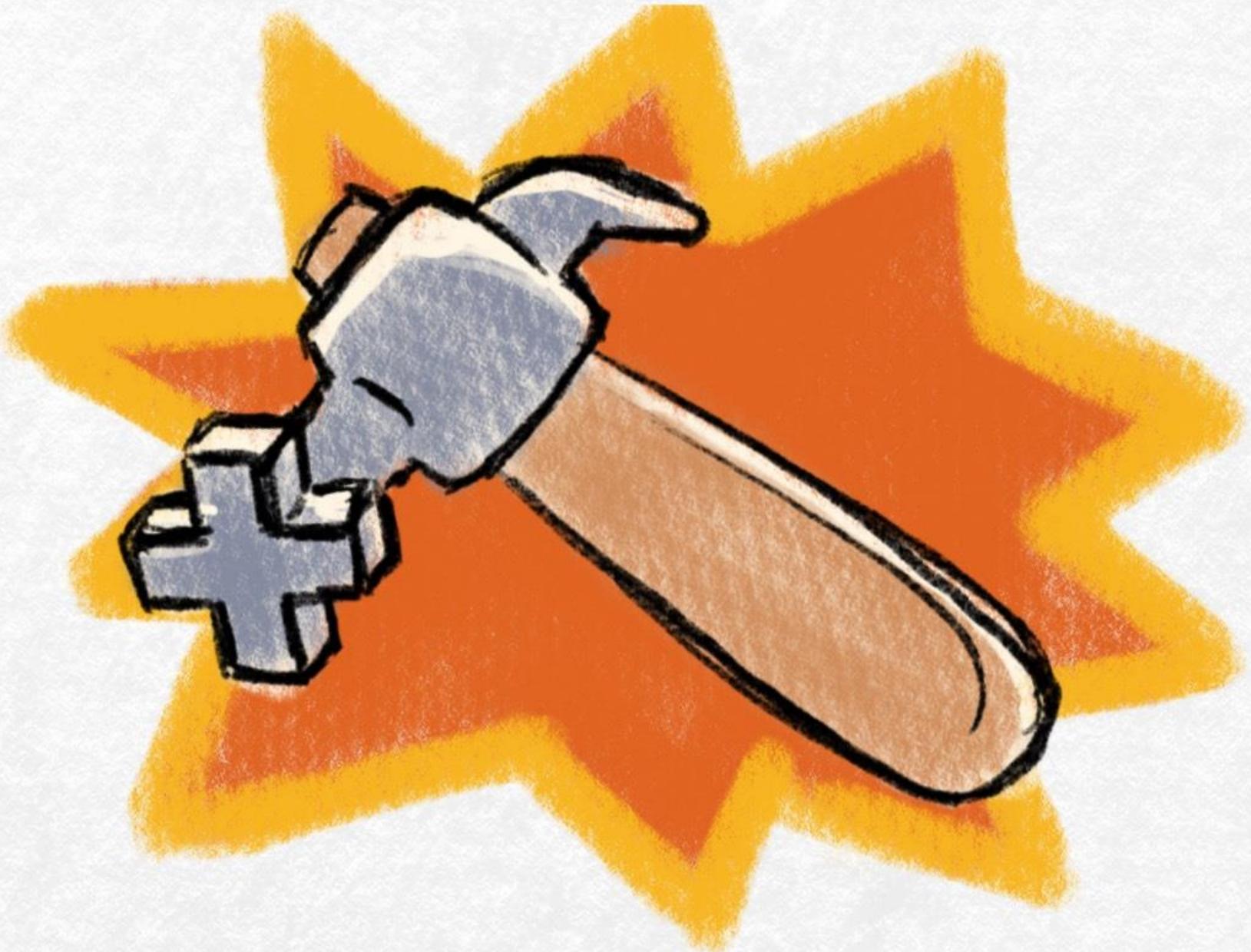




Setting your repo up!

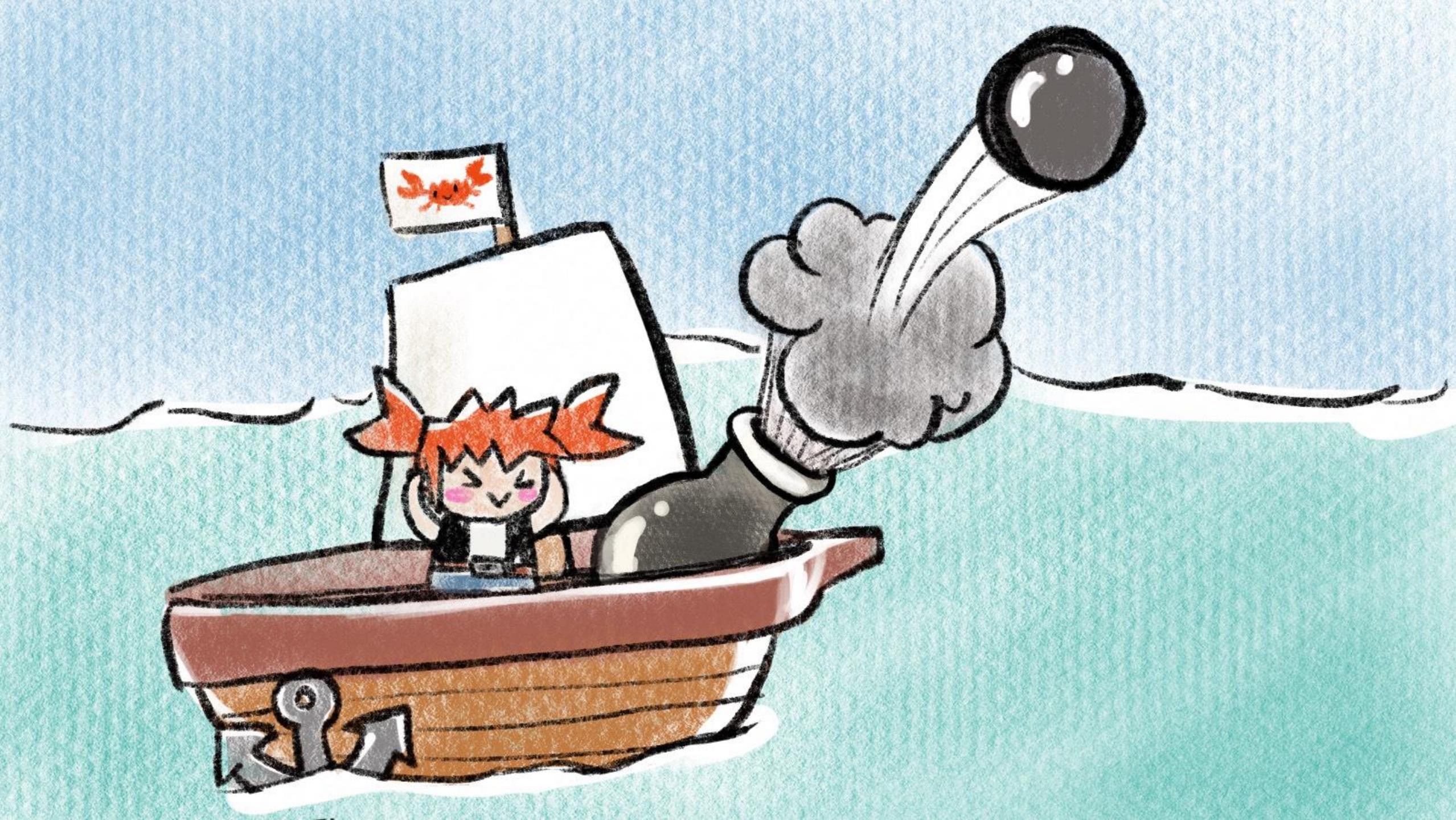
- Rust is built on GitHub! Fork and clone the repo to get the code local!
- You'll need some dependencies to build it:
 - Curl, Git, Cmake, a C++ compiler, Python
- Check `config.toml.example` to see how to tweak the build!
 - Copy it to `config.toml` to tweak it!





Building the compiler!

- To build rust, use `x.py`
 - `x.py check`, `x.py build`, `x.py test`, `x.py doc`
- There are “stages” to the compiler build!
 - The compiler builds itself, so it needs to do this a couple times to make sure everything’s A-OK!





```
#[doc(inline)]
pub extern crate foo;
```

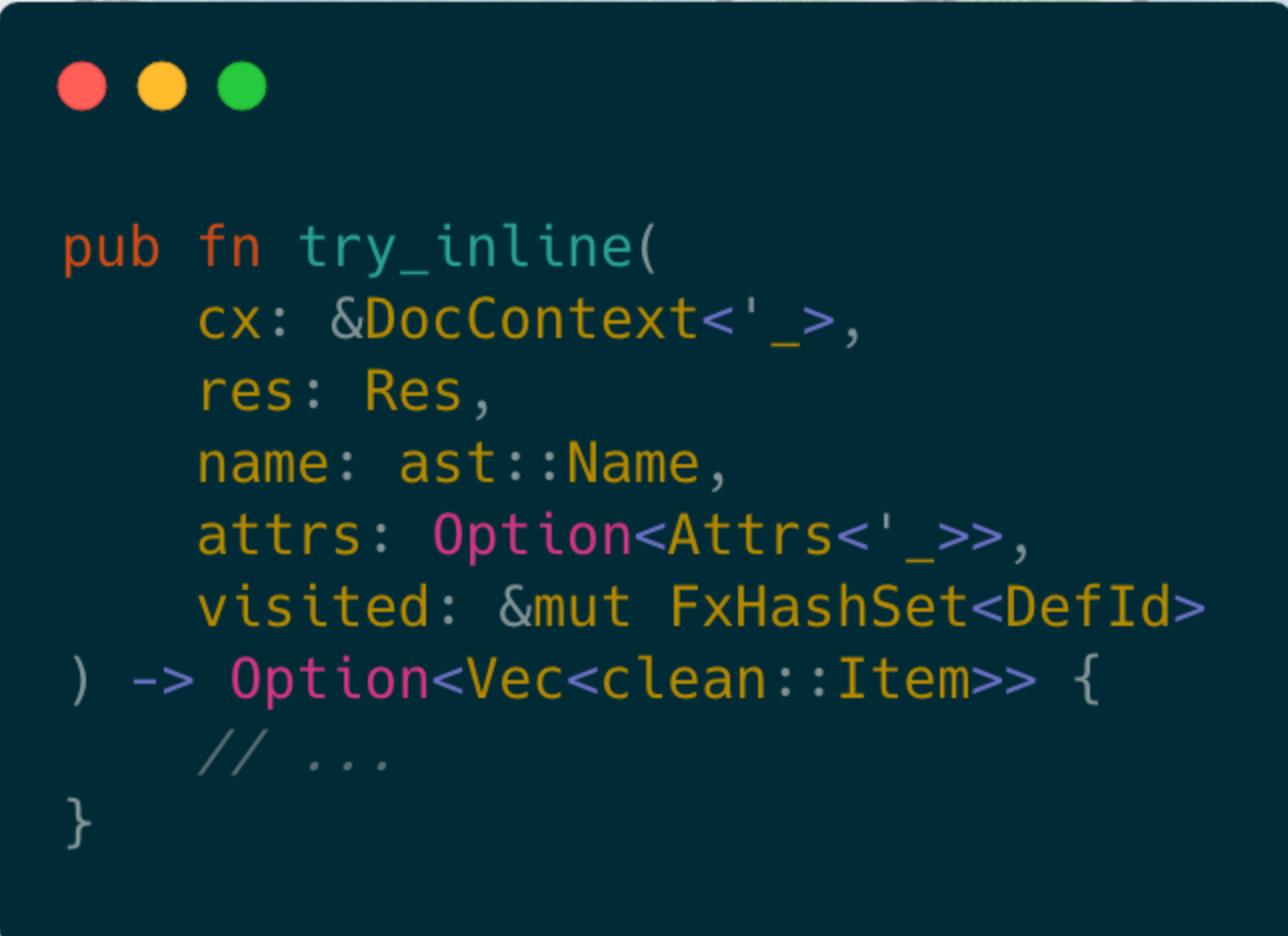


```
extern crate foo;  
#[doc(inline)]  
pub use foo::*;

● ● ●
```



```
impl Clean<Item> for doctree::ExternCrate {
    fn clean(&self, cx: &DocContext) -> Item {
        Item {
            name: None,
            attrs: self.attrs.clean(cx),
            source: self.whence.clean(cx),
            def_id: DefId { krate: self.cnum, index: CRATE_DEF_INDEX },
            visibility: self.vis.clean(cx),
            stability: None,
            deprecation: None,
            inner: ExternCrateItem(self.name.clean(cx), self.path.clone())
        }
    }
}
```



```
pub fn try_inline(  
    cx: &DocContext<'_>,  
    res: Res,  
    name: ast::Name,  
    attrs: Option<Attrs<'_>>,  
    visited: &mut FxHashSet<DefId>  
) -> Option<Vec<clean::Item>> {  
    // ...  
}
```



```
let please_inline = self.vis.node.is_pub() && self.attrs.iter().any(|a| {
    a.name() == "doc" && match a.meta_item_list() {
        Some(l) => attr::list_contains_name(&l, "inline"),
        None => false,
    }
});

if please_inline {
    let mut visited = FxHashSet::default();

    let def = Def::Mod(DefId {
        krate: self.cnum,
        index: CRATE_DEF_INDEX,
    });

    if let Some(items) = inline::try_inline(cx, def, self.name, &mut visited) {
        return items;
    }
}
```



```
impl Clean<Item> for doctree::Module {
    let attrs = self.attrs.clean(cx);

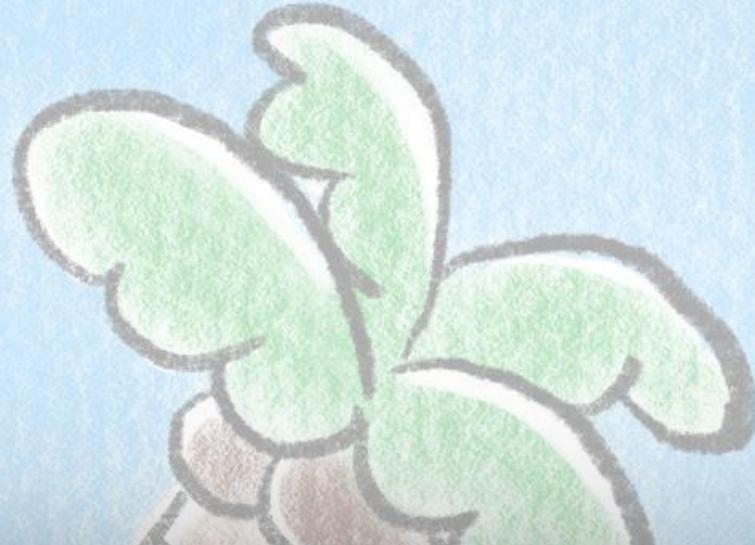
    let mut items: Vec<Item> = vec![];
    items.extend(self.extern_crates.iter().flat_map(|x| x.clean(cx)));
    // ...
}
```



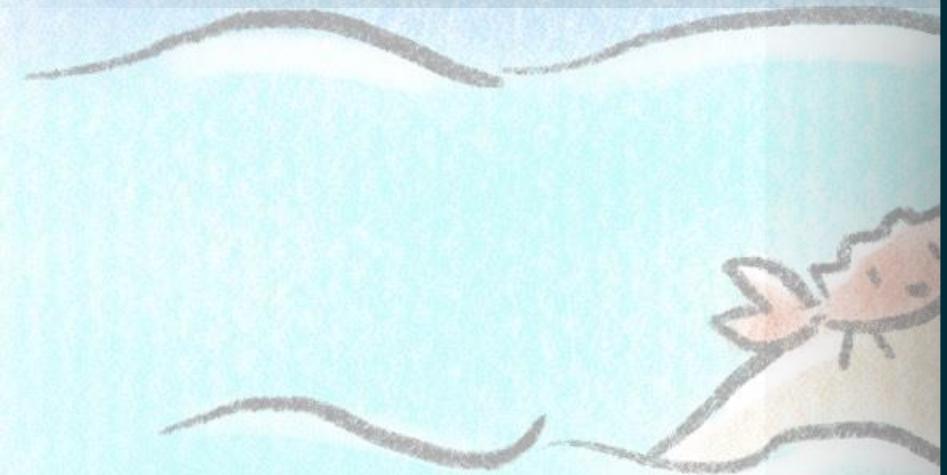
```
impl Clean<Vec<Item>> for doctree::ExternCrate {
    fn clean(&self, cx: &DocContext) -> Vec<Item> {
        // ...
    }
}
```



```
#![crate_name = "inner"]  
pub struct SomeStruct;
```



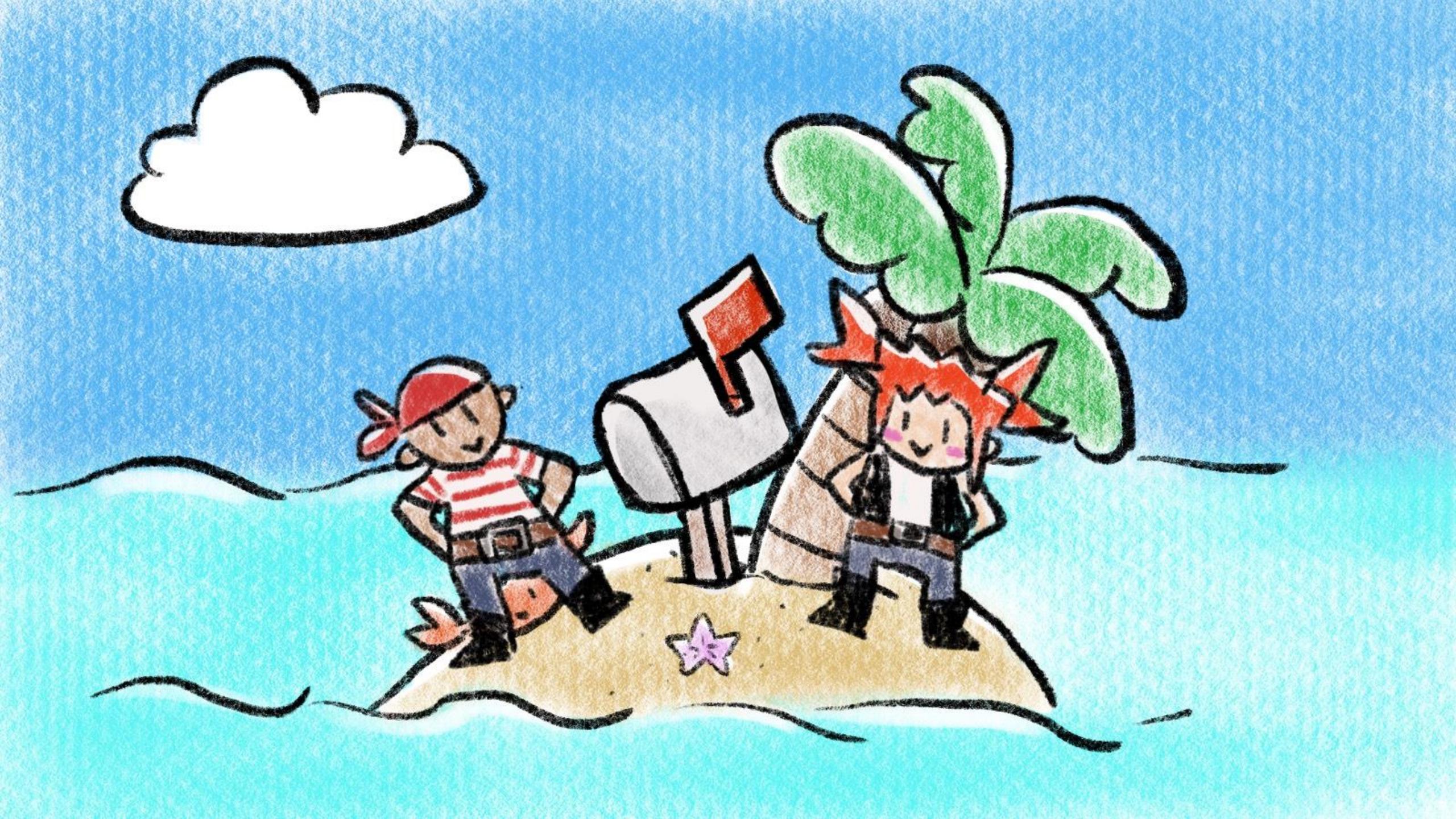
```
// aux-build:pub-extern-crate.rs  
// @has pub_extern_crate/index.html  
// @!has - '//code' 'pub extern crate inner'  
// @has - '//a/@href' 'inner/index.html'  
// @has pub_extern_crate/inner/index.html  
// @has pub_extern_crate/inner/struct.SomeStruct.html  
#[doc(inline)]  
pub extern crate inner;
```





N 11 P

BERG





- <https://optimistictypes.com/>
- <https://quietmisdreavus.net/>
- <https://rust-lang.github.io/rustc-guide/>
- <https://github.com/rust-lang/rust/>
- <https://discord.gg/rust-lang>
- <https://rust-lang.zulipchat.com/>