

# Fundamentals | Type Aliases

# ■ Type Aliases

- ◆ Give a new name to an existing type
  - Basic text substitution
- ◆ Simplifies complicated types
- ◆ Makes code easier to read & write
- ◆ Multiple aliases for the same type will work together, but maybe not as intended

# ■ Syntax

```
type Name = Type;
```

# ■ Examples

```
type ContactName = String;
```

```
type Miles = u64;
```

```
type Centimeters = u64;
```

```
type Callbacks = HashMap<String, Box<Fn(i32, i32) -> i32>>;
```

# ■ Usage

```
struct Contact {  
    name: String,  
    phone: String,  
}
```

```
type ContactName = String;
```

```
type ContactIndex = HashMap<ContactName, Contact>;
```

```
fn add_contact(index: &mut ContactIndex, contact: Contact) {  
    index.insert(contact.phone.to_owned(), contact);  
}
```

# ■ Generics/Lifetimes

```
type BorrowedItems<'a> = Vec<&'a str>;
```

```
type GenericThings<T> = Vec<Thing<T>>;
```

# ■ Recap

- ◆ Type aliases are a way to declare another name for a type
- ◆ Can have generic parameters and lifetime annotations
- ◆ Prefer to use type aliases on longer types
  - Helps avoid bugs
  - Smaller types should be limited to a single module
- ◆ Use newtypes for stricter type checking