

Fundamentals | Ownership

■ Managing memory

- ◆ Programs must track memory
 - If they fail to do so, a “leak” occurs
- ◆ Rust utilizes an “ownership” model to manage memory
 - The “owner” of memory is responsible for cleaning up the memory
- ◆ Memory can either be “moved” or “borrowed”

Example – Move

```
enum Light {  
    Bright,  
    Dull,  
}  
  
fn display_light(light: Light) {  
    match light {  
        Light::Bright => println!("bright"),  
        Light::Dull   => println!("dull"),  
    }  
}  
  
fn main() {  
    let dull = Light::Dull;  
    display_light(dull);  
    display_light(dull);  
}
```

■ Example – Borrow

```
enum Light {  
    Bright,  
    Dull,  
}  
  
fn display_light(light: &Light) {  
    match light {  
        Light::Bright => println!("bright"),  
        Light::Dull => println!("dull"),  
    }  
}  
  
fn main() {  
    let dull = Light::Dull;  
    display_light(&dull);  
    display_light(&dull);  
}
```

■ Recap

- ◆ Memory must be managed in some way to prevent leaks
- ◆ Rust uses “ownership” to accomplish memory management
 - The “owner” of data must clean up the memory
 - This occurs automatically at the end of the scope
- ◆ Default behavior is to “move” memory to a new owner
 - Use an ampersand (&) to allow code to “borrow” memory