Chapter 6: Enums and Pattern Matching

6.1 Enum Erstellung und Umgang

- **Auflistung** unterschiedlicher Möglichkeiten (**Variants**)
- nicht im globalen Namespace. Wird Via Typ:: aufgerufen
- **Variante** kann **verschiedene Typen** und Anzahl der assozierten **Daten halten**
- Typen wie Structs → können **impl Blöcke**, **Methoden** und **#[derive] besitzen**

```
enum IpAddr {
        V4(String),
        V6(String),
}
let home = IpAddr::V4(String::from(“127.0.0.1”));
```

6.2 Das Option Enum

- repräsentiert einen **möglichen Wert**. Keine Null in Rust !
- **Generisch** über T
- Typsystem **verhindert T operationen** von T und Option<T>
    - (T valide und Option<T> nicht sicher ob valide)
- **Generell** muss **jede Variante behandelt** werden **→ match expression**
    - Code der läuft, wenn wir ein **Some(T)** Wert haben und der T verwenden kann
    - Code der für die **None** Variante läuft, die kein Zugriff auf T hat.

```
enum Option<T> {
        Some(T),
        None,
}
```

6.3 Match Control flow Operator

- erlaubt Werte mit **mehreren Pattern** zu **vergleichen** und C**ode basierend** auf **übereinstimmende** Pattern **laufen** zu lassen
- Patterns bestehen aus Werten, Variablen, Namen, Wildcards etc ... (Kapitel 18)
- Compiler achtet darauf, dass **alle Fälle behandelt** werden
- Ausführung **mehrerer Zeilen code** im match arm kann man durch **geschweifte Klammern** erreichen.
- **Werte** von Enum Varianten sind **extrahierbar**
- Das Placeholder Pattern _ wird **jeden Wert matchen**
- If let Syntax (Zucker) für den Umgang mit einem Pattern. Der Rest wird ignoriert
    - else fall äquivalent zu Placeholder

```
let some_u8_value = Some(0u8);
match  some_u8_value {
        Some(3) => println!(“three”),
        _ => (),
}
if let Some(3) = some_u8_value {
        println!(“three”);
}
```