Chapter 6 : Enums and Pattern Matching

6. Gliederung

- 6.1 Enum Erstellung und Umgang
- 6.2 Das Option Enum
- 6.3 Match Control Flow Operator
- 6.4 Zusamenfassung
- 6.5 Quellen

6.1 Enum Erstellung und Umgang

- **Typ Definierung durch Enumerierung (Aufzählung) aller möglichen Werte**
- **Beispiel: IP ist entweder IPv4 oder IPv6**

```
enum IpAddrKind {
    V4,
    V6,
}

let four = IpAddrKind::V4;
let six = IpAddrKind::V6;

fn route(ip_type: IpAddrKind) { }

route(IpAddrKind::V4);
route(IpAddrKind::V6);
```

6.1 Enum Erstellung und Umgang

Daten hinzufügen: Tatsächliche IP(Wert) durch Structs

```
enum IpAddrKind {
    V4,
    V6,
}

struct IpAddr {
    kind: IpAddrKind,
    address: String,
}

let home = IpAddr {
    kind: IpAddrKind::V4,
    address: String::from("127.0.0.1"),
};

let loopback = IpAddr {
    kind: IpAddrKind::V6,
    address: String::from("::1"),
};
```

6.1 Enum Erstellung und Umgang

Daten hinzufügen: Tatsächliche IP(Wert) direkt in jede Enum Variante

+ Pregnanter und kürzer !

```
enum IpAddr {
    V4(String),
    V6(String),
}

let home = IpAddr::V4(String::from("127.0.0.1"));

let loopback = IpAddr::V6(String::from("::1"));
```

6.1 Enum Erstellung und Umgang

+ Vorteil (direkt): Jede Variante kann verschiedene Typen und Anzahl der assoziierten Daten haben

```
enum Message {
    Quit,
    Move { x: i32, y: i32 },
    Write(String),
    ChangeColor(i32, i32, i32),
}
```

- **Alternativ könnten auch vier structs definiert werden**
- **Aber: Functions Definierung schwieriger**

6.1 Enum Erstellung und Umgang

- **Enum Methoden-Definierung: (Äquivalent zu Structs)**

```
impl Message {
    fn call(&self) {
        // method body would be defined here
    }
}

let m = Message::Write(String::from("hello"));
m.call();
```

6.2 Das Option Enum

Encoded häufiger Fall indem Wert invalide oder absent ist

Vom Konzept her ähnlich wie Null in anderen Sprachen

+ Verhindert Bugs

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

```
let some_number = Some(5);
let some_string = Some("a string");

let absent_number: Option<i32> = None;
```

6.2 Das Option Enum

Besser als null, da Option einen zwingt den NULL Fall zu berücksichtigen.

→ Option<T> zu T entpacken (durch match)

→ Option<T> ist ein anderer Typ als T

```
let x: i8 = 5;
let y: Option<i8> = Some(5);

let sum = x + y;
```

```
error[E0277]: cannot add `std::option::Option<i8>` to `i8`
 --> src/main.rs:5:17
  |
5 |     let sum = x + y;
  |                 ^ no implementation for `i8 + std::option::Option<i8>`
  |
  = help: the trait `std::ops::Add<std::option::Option<i8>>` is not implemented for `i8`
```

6.3 Match Control Flow Operator

Wertvergleich mit Patterns

Code Ausführung basierend auf passende Pattern.

Pattern bestehen aus Werten, Variablen, Wildcards etc... (Kapitel 18)

6.3 Match Control Flow Operator

=> Operator trennt Pattern (links) und Code (rechts)

Mehrere ausführbare code Zeilen durch {}

Enum Varianten Wert extrahierbar durch Binden der state Variable an den Wert des Quarter states

Beispiel aufruf:
value_in_cents(Coin::Quarter(UsState::Alaska))
→ “State quarter from Alaska!”

```rust
#[derive(Debug)]
enum UsState {
    Alabama,
    Alaska,
    // --snip--
}

enum Coin {
    Penny,
    Nickel,
    Dime,
    Quarter(UsState),
}

fn value_in_cents(coin: Coin) -> u8 {
    match coin {
        Coin::Penny => {
            println!("Lucky penny!");
            1
        },
        Coin::Nickel => 5,
        Coin::Dime => 10,
        Coin::Quarter(state) => {
            println!("State quarter from {:?}!", state);
            25
        },
    }
}
```

6.3 Match Control Flow Operator

Compiler stellt sicher, dass alle möglichen Fälle behandelt werden

```
fn plus_one(x: Option<i32>) -> Option<i32> {
    match x {
        Some(i) => Some(i + 1),
    }
}
```

```
   Compiling playground v0.0.1 (/playground)
error[E0004]: non-exhaustive patterns: `None` not covered
 --> src/lib.rs:2:11
  |
2 |     match x {
  |           ^ pattern `None` not covered
```

6.3 Match Control Flow Operator

Placeholder _ für alle nicht zu behandelnden Fälle

```
let some_u8_value = 0u8;
match some_u8_value {
    1 => println!("one"),
    3 => println!("three"),
    5 => println!("five"),
    7 => println!("seven"),
    _ => (),
}
```

6.3 Match Control Flow Operator

If let ist Syntax Zucker, wenn kürzer Code ein besseres Trade-off bietet, als vollständiger. Ein else ist Äquivalent zu einem Placeholder

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


if let Some(3) = some_u8_value {
    println!("three");
}
```

6.4 Zusamenfassung

Enum custom Typ Erstellung, die eine der enumerierten Werte sein kann

Option<T> Typ verhindert Errors durch benutzen des Typ Systems

Enum Daten Extrahierung / Benutzung durch match oder if let

6.5 Quellen

https://doc.rust-lang.org/book/ch06-01-defining-an-enum.html

https://doc.rust-lang.org/book/ch06-02-match.html

https://doc.rust-lang.org/book/ch06-03-if-let.html