🦀 به راه Rust با یونس : جلسه ششم (فصل ششم کتاب اصلی Rust)

مدرس: یونس قربانی
موضوع: شمارش‌ها و تطبیق الگو (Enums and Pattern Matching)
مرجع: The Rust Programming Language (Latest Edition)

🎯 چک‌لیست تدریس در استریم (Presenter Checklist)

- ◯ ۱. فلسفه Enumها: چرا و چه زمانی به جای Struct از Enum استفاده کنیم؟
- ◯ ۲. تعریف Enum و تعریف مستقیم داده به Variantها (انواع تایپ‌ها: اعداد، رشته‌ها، تاپل‌ها و استراکت‌های ناشناس)
- ◯ ۳. نوشتن متد برای Enumها در بلوک `impl`
- ◯ ۴. معرفی نجات‌بخش جهان برنامه‌نویسی: غیبت `null` در راست و ظهور `Option<T>`
- ◯ ۵. ساختار کنترل جریان فوق‌العاده با `match` و استخراج مقادیر درونی (Binding to Patterns)
- ◯ ۶. پیاده‌سازی الگوهای تطبیق برای `Option<T>`
- ◯ ۷. الزام به جامعیت الگوها (**Exhaustiveness of match**): کامپایلر هیچ حالتی رو جا نمیذاره
- ◯ ۸. الگوهای همه‌گیر و کاراکتر (Catch-all Patterns & Placeholder) `_`
- ◯ ۹. سینتکس خلاصه و شیک `if let` و جایگزینی آن برای حالت‌های تک‌شرطی

💻 تکه‌های کدهای کلیدی جلسه (Code Snippets)

۱. تعریف Enum با دیتای درونی و پیاده‌سازی متد

```rust
// Variants can hold different types and amounts of data
enum Message {
    Quit,                       // Associated with no data
    Move { x: i32, y: i32 },    // Holds named fields like a struct
    Write(String),              // Holds a single String
    ChangeColor(i32, i32, i32), // Holds three i32 values
}

// We can define methods on enums just like structs using `impl`
impl Message {
    fn call(&self) {
        // Method body would be defined here
        println!("Message method called!");
    }
}

fn main() {
    let msg1 = Message::Write(String::from("Hello Rustaceans!"));
    let msg2 = Message::Move { x: 10, y: 20 };

    msg1.call();
}
```

۲. مفهوم `Option<T>` و خداحافظی با ارورهای Null

```rust
fn main() {
    // Standard library Option enum examples
    let some_number = Some(5);
    let some_char = Some('e');

    // If the value is None, we need to annotate the type T
    let absent_number: Option<i32> = None;

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

    // Compiler error! Rust doesn't allow adding an i8 and an Option<i8> directly:
    // let sum = x + y; // ❌ mismatched types: expected `i8`, found `Option<i8>`

    // You must unwrap or match the Option first to safely access the inner value!
}
```

۳. جادوی `match` و استخراج مقادیر با الگو (Coin & UsState Example)

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

enum Coin {
    Penny,
    Nickel,
    Dime,
    Quarter(UsState), // Variant holding another enum's data
}

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

```
    }
}

// Pattern matching with Option<T>
fn plus_one(x: Option<i32>) -> Option<i32> {
    match x {
        None => None,
        Some(i) => Some(i + 1), // Extracting i and wrapping the new value into
Some
    }
}
```

---

۵. ساختار شسته‌رفتهٔ `if let` (شکرِ سینتکسی match)

```
fn main() {
    let config_max = Some(3u8);

    // Verbose way using match:
    match config_max {
        Some(max) => println!("The maximum is configured to be {}", max),
        _ => (), // Boilerplate catch-all branch required by the compiler
    }

    // Clean and concise way using if let:
    if let Some(max) = config_max {
        println!("The maximum is configured to be {}", max);
    } else {
        println!("No maximum configured!");
    }
}
```

🧠 تقلب‌نامه و نکات کلیدی برای دانشجویان (Key Concepts)

۱. قدرت پنهان Enum در راست در برابر سایر زبان‌ها

در زبان‌هایی مثل C یا Java، انوم‌ها (Enum) صرفاً لیستی از اعداد نام‌دار هستند. اما در راست، هر Variant از Enum می‌تواند تایپ، ساختار و دیتای اختصاصی خودش رو حمل کنه (از هیچ دیتا گرفته تا String، Tuple و حتی Struct کامل)!

۲. چرا در راست چیزی به نام `null` وجود ندارد؟

تونی هور (خالق Null) این اختراع رو «اشتباه یک میلیارد دلاری» نامید، چون هر لحظه ممکنه دسترسی به یک متغیر خالی باعث کرش برنامه (NullPointerException) بشه.
راست اصلاً مقدار `null` نداره! در عوض Enum استاندارد `Option<T>` رو داره:

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

```
}
```

اگر متغیری از نوع `i32` باشه، **صددرصد تضمین شده که عدد دارد**. اگر ممکنه عددی وجود نداشته باشه، تایپش باید حتماً `Option<i32>` باشه. سیستم تایپینگ به شما اجازه نمیده با یک `Option` مثل دیتای خام رفتار کنید، مگر اینکه اول با `match` یا توابع امن بررسی کنید که آیا `Some` است یا `None` .

۳. قانون طلایی `match` : جامعیت (Exhaustiveness)

ساختار `match` در راست کاملاً **Exhaustive** است؛ یعنی حتماً باید **تمام حالت‌های ممکن** پوشش داده بشن. اگر حتی یک واریانت یا حالت `None` رو فراموش کنید، کد اصلاً کامپایل نمیشه! این یعنی باگ‌های ناشی از جا افتادن یک شرط در زمان اجرا، به خطای زمان کامپایل تبدیل میشن.

۴. کی از `match` و کی از `if let` استفاده کنیم؟

- **از `match` استفاده کنید وقتی:** با منطق‌های چندگانه سر و کار دارید، چند واریانت مختلف رو هندل می‌کنید و می‌خواهید مطمئن بشید هیچ حالتی جا نیفتاده.
- **از `if let` استفاده کنید وقتی:** فقط و فقط **یک حالت خاص** براتون مهمه (مثلاً فقط `Some` براتون ارزشمنده و بقیه حالت‌ها مهم نیستند). در این صورت `if let` بویلرپلیت و شلوغی `_ => ()` رو حذف می‌کنه و کد رو فوق‌العاده خوانا می‌کنه.