

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

مدرس: یونس قربانی موضوع: ساخت بازی حدس عدد (Programming a Guessing Game)
مرجع: The Rust Programming Language (Latest Edition)

🎯 چک لیست تدریس در استریم (Presenter Checklist)

- ✓ ۱. ساخت پروژه با Cargo و بررسی ساختار اولیه
- ✓ ۲. دریافت ورودی از کاربر (`io::stdin`) و مفهوم Prelude
- ✓ ۳. متغیرها: تفاوت پیش فرض `immutable` و کلمه کلیدی `mut`
- ✓ ۴. مدیریت خطاهای مقدماتی با `Result` و متد `expect()`
- ✓ ۵. چاپ متغیرها با Placeholders در ماکرو `println!`
- ✓ ۶. افزودن Grates خارجی (Grates.io) و پکیج `rand` در `Cargo.toml`
- ✓ ۷. درک `Cargo.lock`، بازتولید بیلدها (Reproducible Builds) و دستور `cargo update`
- ✓ ۸. تولید عدد تصادفی با `rand::thread_rng().gen_range(1..=100)`
- ✓ ۹. مقایسه اعداد با `match` و اینام `Ordering`
- ✓ ۱۰. تبدیل نوع داده (Type Shadowing & Parsing) از `String` به `u32`
- ✓ ۱۱. ساخت حلقه بی نهایت با `loop`
- ✓ ۱۲. مدیریت خطای ورودی کاربر با `match parse` (Pattern Matching) به جای کرش (`expect`)
- ✓ ۱۳. خروج تمیز از حلقه با `break` در زمان برد

سورس کد نهایی بازی حدس اعداد

```
1 use rand::Rng;
2 use std::cmp::Ordering;
3 use std::io;
4
5 fn main() {
6     println!("🎮 Welcome to the Guessing Game!");
7
8     // Generate a random number between 1 and 100 (inclusive)
9     let secret_number = rand::thread_rng().gen_range(1..=100);
10
11     loop {
12         println!("Please input your guess:");
13
14         let mut guess = String::new();
15
16         io::stdin()
17             .read_line(&mut guess)
18             .expect("Failed to read line!");
19
20         // Convert string to number + handle invalid input error without
           crashing the program
```

```

21     let guess: u32 = match guess.trim().parse() {
22         Ok(num) => num,
23         Err(_) => {
24             println!("⚠ Please enter a valid number!");
25             continue;
26         }
27     };
28
29     println!("You guessed: {guess}");
30
31     // Compare the guessed number with the secret number
32     match guess.cmp(&secret_number) {
33         Ordering::Less => println!("📉 Too small!"),
34         Ordering::Greater => println!("📈 Too big!"),
35         Ordering::Equal => {
36             println!("🎉 You win! 🏆");
37             break;
38         }
39     }
40 }
41 }

```

🧠 تقلب‌نامه و نکات کلیدی برای دانشجویان (Key Concepts)

۱. متغیرها و سیستم تغییرپذیری (Mutability)

- در Rust همه متغیرها به صورت پیش فرض **غیرقابل تغییر (Immutable)** هستند:

```

1 let apples = 5; // Immutable (trying to modify causes a compiler error)
2 let mut bananas = 5; // Mutable

```

- **سوال:** «چرا مثل پایتون یا JS همه چیز آزاد نیست؟»
- **جواب:** امنیت حافظه و همزمانی بدون فدا کردن پرفورمنس در زمان اجرا!

۲. مفهوم Prelude و کتابخانه استاندارد

- بخش‌های بسیار پرکاربرد کتابخانه استاندارد به طور خودکار به هر فایلی import می‌شوند که به آن **Prelude** می‌گویند.
- برای استفاده از ابزارهایی که در Prelude نیستند، از **use** استفاده می‌کنیم:

```

1 use std::io;

```

۳. ارجاع‌ها (References) و &mut

- در `io::stdin().read_line(&mut guess)`
- علامت **&** یعنی داریم یک **Reference (ارجاع)** به متغیر پاس میدیم تا کل دیتا در حافظه کپی نشه.

- فرانس‌ها هم به صورت پیش‌فرض Immutable هستند. پس برای اینکه تابع بتونه مقدار رو دستکاری کنه، باید صراحتاً بنویسیم: `&mut guess`.

۴. مدیریت خطا با Result و متد expect

- متد `read_line` یک `Result` برمیگرداند. `Result` یک `Enum` است با دو وضعیت:
 - `Ok`: عملیات موفقیت‌آمیز بوده و مقدار نهایی را در خود دارد.
 - `Err`: عملیات شکست خورده و دلیل خطا را در خود دارد.
- متد `expect("...")`: اگر وضعیت `Err` باشد، برنامه را متوقف (`Panic`) میکند و پیام ارور را چاپ میکند. اگر نویسی‌دش، کامپایلر یک `Warning` زرد نگران‌کننده بهتون میده.

۵. کریت‌ها (Crate) و فایل Cargo.lock (اکوسیستم کارگو)

- **Crate**: کوچکترین واحد کد اجرایی یا کتابخانه‌ای در Rust.
- اضافه کردن وابستگی به فایل `Cargo.toml`:

```
1 [dependencies]
2 rand = "0.8.5"
```

- تفاوت فایل‌ها:
 - `Cargo.toml`: جایی که شما وابستگی‌ها و نگارش‌های درخواستی را مشخص میکنید.
 - `Cargo.lock`: قفل‌کننده نسخه دقیق وابستگی‌ها برای تضمین **Reproducible Builds** (برنامه همیشه روی سیستم دیگران هم دقیقاً مثل سیستم شما بیلد شود).
 - برای به‌روزرسانی زیرنسخه‌ها بدون تغییر فایل TOML فایل `cargo update`.

۶. سیستم Type Shadowing (سایه‌زنی متغیرها)

زبان برنامه نویسی Rust به شما اجازه میده متغیری با همان نام قبلی بسازید تا نوع آن را تغییر دهید بدون اینکه مجبور شوید اسامی عجیبی مثل `guess_str` و `guess_int` انتخاب کنید:

```
1 let guess = String::new();
2 // ...
3 let guess: u32 = guess.trim().parse().expect("...");
```

- متد `.trim()`: حذف فاصله‌ها و کاراکتر `\n` انتهای خط اینپوت.
- متد `.parse()`: تبدیل رشته به نوع عددی مورد انتظار (`u32`).

۷. کنترل جریان و تطبیق الگو (match & Ordering)

- اینام (`Enum`) `std::cmp::Ordering` سه حالت دارد: `Less`, `Greater`, `Equal`.
- دستور `match` در Rust برخلاف `switch` در زبانهای دیگر، **Exhaustive** است. یعنی باید تمام حالات ممکن پوشش داده شوند وگرنه کد اصلاً کامپایل نمیشود.

```
1 match guess.cmp(&secret_number) {
2     Ordering::Less => println!("Too small!"),
3     Ordering::Greater => println!("Too big!"),
4     Ordering::Equal => println!("You win!"),
5 }
```

۸. مدیریت حرفه‌ای خطا: `match` روی `Result` به جای `expect`

به جای کرش دادن کل برنامه با `expect` وقتی کاربر حرف بی‌ربط وارد کرد:

```
1 let guess: u32 = match guess.trim().parse() {
2     Ok(num) => num,          // If it's a number, extract its value
3     Err(_) => continue,     // If it's an error, skip this iteration and continue
4                             // to the next
5 };
```
