Strings in Rust

Kai von dem Fange

Strings vs. String-Literale:

- String:
  - auf dem Heap gespeichert als UTF-8 Bytevektor (Vec<u8>)
  - Veränderbare Größe
  - Haben Owner
  - Reihe von Methoden

- String-Literal (&str):
  - slice (Pointer) der immer auf eine UTF-8 Sequenz zeigt
  - Statische Größe, zur Compile-Zeit bekannt
  - Für Konstanten (unveränderlich)
  - Haben keinen Owner

String-Literale (aka string slice):

- Erstellen: let var_name: &str = „Hello World“;
  - Können auch mutable gesetzt werden,
    aber nicht Sinn von Literalen
- Anhängen: ↯ verwende Owned-Strings
- Verwendung:
  - Parameter
  - Hash-Schlüssel
  - Return-Werte

Strings (die echten):

- Erstellen:
  - Neuer leerer String mit String::new();
  - Aus einem String-Literal mit .to_string();
  - Oder mittels String::from(„…“)
- Anhängen (wenn mit mut deklariert):
  - .push_str(„…“); für Zeichenketten ␣ double quotes „“ verwenden!
    - Parameter sind Literale, keine Strings!
  - .push(„…‘); für chars ␣ single quotes ,‘ verwenden!
  - Konkatenation mit „+“-Operator
- Kein Indexing! ␣ .chars(), .bytes() verwenden

Escapes in Rust:

- Reservierte Zeichen in Strings mittels „\“
  - Bsp: println!(„M&M\‘s“); ␣ Ausgabe: M&M‘s
- Zeichen auch als Hex-Code mit Escapes darstellbar
  - Bsp: println!(„WTF\x3F\x21“); ␣ Ausgabe: WTF?!
- Backslash als Zeichen ␣ „\\“

Raw Strings:

- „rohe“ Zeichenketten
- r“…\…“, r#“…“…“#
- Sonderzeichen wie \ oder „“ und ‚ als normale Zeichen interpretiert
- Um Zeichenketten wie den Delimiter darzustellen weitere #‘s an den Delimiter:
  - r##“…“#…“##
- Bis zu 2^16-1 = 65535 Delimiter

Byte Strings:

- Auch Zeichen außerhalb von UTF-8 (\x00 – \x7f)
- b“\xBFp\xB8\xB2e\xC2 \xBC\B8p!“; ␣ „Hallo Welt!“ in russisch
- Nicht immer &str konvertierbar
- Auch mit Raw String kombinierbar ␣ br#“…“#

Warum eigentlich?

- Fokus auf Sicherheit und Performance
- Keine NullPointer-Dereferenzierung
- Kein Buffer Overflow
- Keine Datenkonflikte

Quellen:

- rust-lang.org (Dokumentation & Forenbeiträge)
- Etwas chatGPT
- Wikipedia