Rust

Author: Darua Shutina

23-02-07

rustlings 🦀❤️ : https://github.com/rust-lang/rustlings

Crate

A crate is a compilation unit in Rust. Whenever `rustc some_file.rs` is called, `some_file.rs` is treated as the *crate file*.

A crate can be compiled into a binary or into a library. By default, `rustc` will produce a binary from a crate. This behavior can be overridden by passing the flag `--crate-type=lib`.

Cargo

Cargo is **Rust's build system and package manager**. With this tool, you'll get a repeatable build because it allows Rust packages to declare their dependencies in the file `Cargo.toml`.

Это инструмент, который позволяет билдить, запускать, тестить и фиксить проект.

Создать новый проект:

```
$ cargo new new_project
$cd new_project

$cargo init
```

Cargo.toml

В этом файле объявляются имя, версия, сурс, чексумм и зависимости. В начале файла также добавляется версия проекта (?).

```
version = 3
[[package]]
name = "time"
version = "0.1.45"
source =
"registry+https://github.com/rust-lang/crates.ioindex"
checksum =
"1b797afad3f312d1c66a56d11d0316f916356d11158fbc6ca6389ff6bf805a"
dependencies = [
"libc",
"wasi",
"winapi 0.3.9",
]
```

`checksum` - это хеш для цифровой подписи. Когда заливаешь свой пакет в репозиторий, от него формируется Криптографический Хеш и прописывается локально.

Если злоумышленник получит доступ к пакетному менеджеру и попробует подменить пакет, то не пройдет билд, так как сохраненный хеш и хеш пакета не совпадут.

Переменные

```
let par1: String = "aboba";  // cannot be modified
let mut par2: String = "abober"; // can be modified
let par3 = par1;
```

Функции

```
fn f(par1: String) -> String {
    return format!{"{}", par1};
}
fn f(par1: String) {   // <=> void function
    println!{"{}", par1};
}

fn f(mut par1: String) {
// mut => переменную можно изменять внутри функции
    println!{"{}", par1};
}
```

Смысл точки с запятой

Если в конце строки стоит `;`, то строка превращается в statement и ничего не возвращает. Если в такой строке дописать в начале `return`, то тогда она будет что-то возвращать.

Если оставить строку без точки с запятой и без слова `return`, то строка будет что-то возвращать:

```
fn f(par1: String) -> String {
    return format!("{}", par1);
}

fn f(par1: String) -> String {
    format!("{}", par1)
}
```

Макросы

`format!`

Возвращает отформатированный текст в виде строки.

```
format!("the value is {var}", var = "aboba");
```

`print!` и `println!`

То же, что и `format!`, но печататают вывод в `io::stdout`.

`eprint!` и `eprintln!`

То же, что и `format!`, но печататают вывод в `io::stderr`.

`panic!`

Аналог выкидывания исключений. Мы можем кидать панику и перехватывать панику, вот класс!

```
panic!("this is my message");
```

Data types

- Numeric -- всевозможные числа и операции над ними

  Возможные типы:

  - `i8` (int 8 bit), `u8` (unsigned int 8 bit), ..., `i128`, `u128`;
  - `f32`, `f64`, `0xff`;
  - etc.

  Если происходит переполнение, то получим `panic!` в режиме дебага и `overflow` в обычном режиме.

- bool
- char 32bit: `'a'`

  Строка в расте -- это не массив чаров, а какая-то более сложная вещь

- array
- tuple
- etc

If

Фигурные скобочки обязательные

```rust
if a > 0 {
    // do smth
} else {
    // do smth
}

let b: i32 = if a > 0 { 1 } else { 2 };
```

Loop

```rust
loop {
    counter += 1;
    if counter > 42 {
        break
    }
}



let b = loop {
    counter += 1;
    if counter > 42 {
        break counter * 2
    }
};
// `loop` вернет 84, и это значение положится в `b`
// после второй `}` ставится точка с запятой, потому что присваивание -- это всегда statement и требует точку с запятой.



let b = 'main_loop: loop {
    loop {
        counter += 1;
        if counter > 42 {
            break 'main_loop counter * 2
        }
    }
};
// используем лейбл для цикла
```

While/for

```rust
let mut counter = 0;

while counter < 42 {
    counter += 1;
}

for _ in 0..42 {
    println!("we are in a while loop");
}
```

`while` -- это `loop` с условием. Но, в отличие от `loop`, он не может возвращать значение (`loop` может, пример выше).