Writing Functions

Zachary Bennett
Lead Software Developer

@z_bennett_ | zachbennettcodes.com

Functions in Rust

Name

Parameters/Arguments

Return type

Body

```
fn add(a: i32, b: i32) -> i32 {

    a + b

}

let result = add(1, 2);
```

Function Anatomy

Functions are comprised of four primary parts: a name, parameters, a return type, and a body. The last expression evaluated within the function body becomes the return value.

A function is called by passing in a comma-delimited list of variable names enclosed by parentheses.

Function Anatomy “Gotcha”

```
fn add(a: i32, b: i32) -> i32 {
    a + b; // This will not compile because of the semicolon
}

------------

fn add(a: i32, b: i32) -> i32 {
    return a + b; // Adding a return statement or removing the semicolon works
}
```

Returning Early From Within a Function

```
fn add_or_subtract(a: i32, b: i32) -> i32 {
    if (a <= b) {
        return a + b;
    }

    a - b
}
```

Functions Without An Explicit Return Type

```
// This function returns nothing – it just performs a side effect
fn log(msg: &str) {
    println!(“{}”, msg);
}
```

Function names, like variable names, use “snake_case”

Using Functions

Rust Function Categories

Named Functions

Anonymous Functions (Closures)

Methods

Associated Functions

Named Functions

```
// Simple, named function

fn add(a: i32, b: i32) -> i32 {
    a + b
}
```

Anonymous Function (Closure)

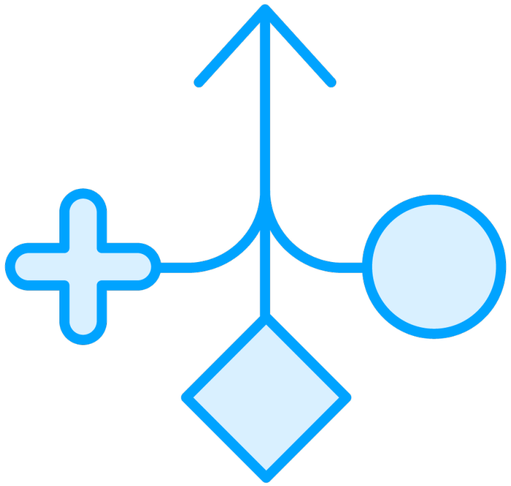
```
let my_anon_fn = |a: i32, b: i32| if a > b { a + b } else { a - b };

my_anon_fn(1, 2);
```

Anonymous functions can omit parameter types and return types.

Closures and Scope

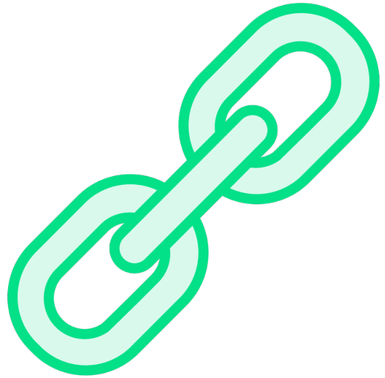
```
let my_num = 42;

// This function has access to a variable defined outside of its own scope
let my_anon_fn = |a: i32| if a < 0 { a + my_num } else { a - my_num };

my_anon_fn(1);
```

Function Associated With a Type



Methods

A method is a function that is associated with an instance of a type.



Associated Functions

An associated function is a function that is directly usable from a type itself

Methods

```
#[derive(Debug)]
struct Point {
    x: i32,
    y: i32
}

impl Point {
   fn print(&self) {
       println!(“X: {} | Y: {}”, self.x, self.y);
   }

   fn add_one(&mut self) {
       self.x += 1;
       self.y += 1;
   }
}
```

Methods always take “self” as the first parameter

Associated Functions

```
struct Coffee {
    temp: i32,
    name: &str
}

impl Coffee {
   fn new(temp: i32, name: &str) -> Coffee {
       Coffee {
           temp: temp,
           name: name
       }
   }
}
```

Associated functions do not require a reference to a particular instance

Demo

Named Functions

Anonymous Functions (Closures)

Methods and Associated Functions