The Rust programming language

Paolo Bettelini

Contents

1 Basic Types **2**

2 Tuples **2**
2.1 Returning from loops . . . 2
2.2 Labels . . . 2
2.3 Returning from labelled loops . . . 3

3 Pattern Matching **3**
3.1 Basic . . . 3
3.2 Destructuring . . . 4
3.3 Ignoring values . . . 4
3.4 Match guards . . . 4
3.5 @ bindings . . . 5

1 Basic Types

```
// boolean
bool

// signed integers
i8, i16, i32, i64, i128, isize

// unsigned integers
u8, u16, u32, u64, u128, usize

// floating points
f32, f64

// Text
char, String, str
```

2 Tuples

Tuples are a combination of multiple types. Tuples can contain any number of types and/or other tuples.

```
let coordinates = (101, 3, 4);
let person = ("Paolo", "Bettelini", 18);
let status: (bool, (u128, i32)) = (true, (1u128, 2));
```

2.1 Returning from loops

```
let mut counter = 0;

let result = loop {
    counter += 1;

    if counter == 10 {
        break counter;
    }
};
```

2.2 Labels

```
'outer: loop {
    'inner: loop {
        // This breaks the inner loop
        break;
        // This breaks the outer loop
        break 'outer;
    }
}
```

2.3 Returning from labelled loops

```
let mut counter = 0;

let result = 'outer: loop {
    counter += 1;

    if counter == 10 {
        break 'outer counter;
    }
};
```

3 Pattern Matching

3.1 Basic

```
    let x = 5;

    match x {
      // matching literals
      1 => println!("one"),
      // matching multiple patterns
      2 | 3 => println!("two or three"),
      // matching ranges
      4..=9 => println!("within range"),
      // matching named variables
      x => println!("{}", x),
      // default case (ignores value)
      _ => println!("default Case")
    }
```

3.2 Destructuring

```
struct Point {
    x: i32,
    y: i32,
  }

  let p = Point { x: 0, y: 7 };

  match p {
    Point { x, y: 0 } => {
      println!("{}" , x);
    },
    Point { x, y } => {
      println!("{} {}" , x, y);
    },
  }

  enum Shape {
    Rectangle { width: i32, height: i32 },
    Circle(i32),
  }

  let shape = Shape::Circle(10);

  match shape {
    Shape::Rectangle { x, y } => //...
    Shape::Circle(radius) => //...
  }
```

3.3 Ignoring values

```
struct SemVer(i32, i32, i32);

let version = SemVer(1, 32, 2);

match version {
  SemVer(major, _, _) => {
    println!("{}", major);
  }
}

let numbers = (2, 4, 8, 16, 32);

match numbers {
  (first, .., last) => {
    println!("{}, {}", first, last);
  }
}
```

3.4 Match guards

```
let num = Some(4);

match num {
  Some(x) if x < 5 => println!("less than five: {}", x),
  Some(x) => println!("{}", x),
  None => (),
}
```

3.5 @ bindings

```
struct User {
    id: i32
  }

  let user = User { id: 5 };

  match user {
    User {
      id: id_variable @ 3..=7,
    } => println!("id: {}", id_variable),
    User { id: 10..=12 } => {
      println!("within range");
    },
    User { id } => println!("id: {}", id),
  }
```