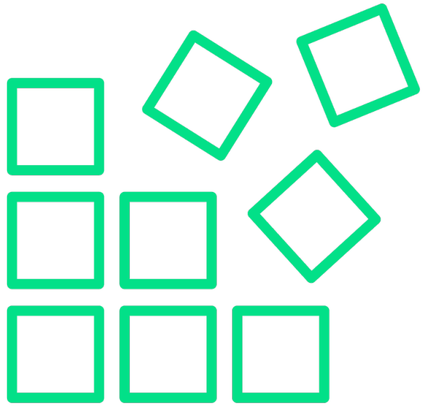
Control Flow

Zachary Bennett
Lead Software Developer

@z_bennett_ | zachbennettcodes.com

Control Flow



Expressions

A combination of values, variables, operations, etc. that the compiler evaluates into a new value

Statements

A standalone unit of code that does not return a value. Think “structure” here.

Rust is expression-based

Conditional Expressions

```
if my_num >= 5 {
    println!(“The number is greater than or equal to 5!”);
} else if my_num > 2 {
    println!(“The number is greater than 3 but less than 5!”);
} else {
    println!(“Now we are running this code!”);
}
```

Conditional Expressions With let Statements

```
let should_use_ten = true;

let my_num = if should_use_ten { 10 } else { 0 };
```

Control Flow Using Loops

loop

while

for

Statement-based Looping Using loop

```
// Basic infinite loops
loop {
    println!(“We are looping!”);
}


// Returning values using loop
let mut my_num = 0;
let final_num = loop {
    my_num += 5;

    if my_num > 10 {
        break my_num + 1;
    }
}
```

Conditional Looping Using while

```
let mut my_num = 0;

while my_num < 10 {
  my_num += 1;
 // This code will run until “my_num” is less than 10
}
```

Iteration Using for/in

```
let my_arr = [1, 2, 3, 4, 5];

for num in my_arr {
    // Here we have access to each element in the array
}

// Mutable looping
let mut my_vec = vec![1, 2, 3, 4, 5];

for num in &mut my_vec {
    *num += 1;
}
```

Demo

Expressions and statements

Control flow using conditional expressions

Control flow using loops

- loop
- while
- for/in

Pattern Matching

Pattern Matching

“... an extremely powerful control flow construct ... that allows you to compare a value against a series of patterns and then execute code based on which pattern matches.”

Citation: Rust Docs, “The match Control Flow Construct”, https://doc.rust-lang.org/book/ch06-02-match.html

Using match

```
enum MessageState {
  Pending,
  Sending,
  Received
}

let msg_state = MessageState::Pending;

match msg_state {
  MessageState::Pending => println!(“Message Pending!”),
  MessageState::Sending => println!(“Message Sending!”),
  MessageState::Received => println!(“Message Received”)
}
```

Using match With Variables

```
let msg_state = MessageState::Pending;

let status_code = match msg_state {
  MessageState::Pending => 1,
  MessageState::Sending => 2,
  MessageState::Received => 3
}
```

Pattern Matching with Bound Values

```
enum MessageState {
  Pending(1),
  Sending(2),
  Received(3)
}

let msg_state = MessageState::Received;

match msg_state {
  MessageState::Pending(status_code) => println!(“Msg status: {}”, status_code),
  _ => println!(“Message is no longer pending”)
}
```

Simple Pattern Matching with if/let

```
enum MessageState {
  Pending(1),
  Sending(2),
  Received(3)
}

let msg_state = MessageState::Received;

if let MessageState::Pending(status_code)  = msg_state {
  println!(“Msg status: {}”, status_code),
}

// Can use an optional “else” block afterwards!
```

Pattern matching is powerful!

Using Comparison and Logical Operators

Logical Operators Breakdown

! - NOT

Reverses the logical state of its operand. True becomes false. False becomes true

&& - AND

If both operands are true, returns true. If any operand is false, returns false.

|| - OR

If any operand is true it returns true, otherwise returns false

Logical Operator Examples

```
let this_is_true = true;
let this_is_false = false;

if !this_is_false {
  println!(“This code will run – false becomes true!”);
}

if this_is_true || this_is_false {
  println!(“This code will run – one of these operands is true!”);
}

if this_is_true && this_is_false {
  println!(“This code will NOT run – one of these operands is false!”);
}
```

Comparison Operators

!=	>	<
==	>=	<=

Comparison Operator Examples

```
let my_num = 5;
if my_num != 3 {
  println!(“This code will run – ‘my_num’ does not equal 3!”);
}

if my_num == 5 {
  println!(“This code will run – ‘my_num’ equals 5!”);
}

if my_num > 6 {
  // This block won’t run
} else if my_num >= 6 {
  // This block won’t run
} else if my_num < 5 {
  // This block won’t run
} else if my_num <= 5 {
  // This block WILL run because ‘my_num’ is equal to 5
}
```