Rust Syntax and Data Types

Zachary Bennett
Lead Software Developer

@z_bennett_ | zachbennettcodes.com

Variables are immutable by default

```
// Variables

let my_number = 1;

let mut my_other_number = 2;

const IMPORTANT_NUMBER: i32 = 42;


let my_number = 2;

// “my_number” now points to the number ”2”
```

◀ **Defining an immutable variable**

◀ **Defining a mutable variable**

◀ **Defining a constant. A constant is always immutable and valid throughout the life of your program.**

◀ **Shadowing a variable is possible. This is how to perform a transformation on a value where the variable remains immutable.**

Comments

Simple

Documentation

```
// Comments
```

◀ A simple comment looks like this

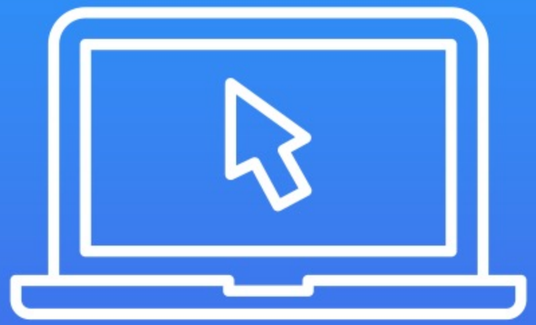
```
let my_number = 2; // Another comment
```

◀ Comments can also be placed at the end of a line of code

```
/// Subtracts “y” from “x”

pub fn subtract(x: i32, y: i32) -> i32 {
    x - y
}
```

◀ Documentation comments are created with three forward slashes. These are used when creating external crates. Documentation is generated from comments like these.

Demo



Rust Syntax

- Variables
- Immutability and mutability
- Comments

Primitive Data Types in Rust

Date Type Categories

Scalar

These are types that represent a single value

Compound

Types that represent multiple values

Scalar Types

Integer

Floating Point

Boolean

Character

```
// The Integer Type

let my_number: i32 = 1;

let my_8_bit_int: i8 = -128;

let other_int: u8 = 128;
```

◂ Defining integers

Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

```
// More Primitive Scalar Types


let my_floating_point: f64 = 2.5;

let other_float: f32 = 2.0;


let this_is_true: bool = true;

let this_is_false = false;


let my_char: char = 'A';
```

◀ **Defining floating point values**

◀ **Defining booleans**

◀ **Defining a single character literal**

Compound Types

Tuples

Arrays

```
// Primitive Compound Types

let my_tuple = (‘A’, 1, 1.2);

let my_arr = [1, 2, 3];

let my_arr_typed: [i32; 3] = [1, 2, 3];
```

◀ **Defining a tuple**

◀ **Defining an array**

◀ **Defining an explicitly typed array**

Creating Custom Types

Custom Types

Structs

A combination of different data types which create a custom type

Enums

A way to define a custom type by explicitly defining its states

```
// Example struct


struct Coffee {
  id: i64,
  name: String
}


let coffee = Coffee {
  id: 123456,
  name: String::new(“Latte”)
}
```

- Structs are defined by using the “struct” keyword alongside a name
- Structs are your means of combining different data types together underneath the umbrella of a custom type
- Instantiating a struct looks like this

```
// Example enum

enum CoffeeType {

  HOT,

  ICED

}

// Example usage

let my_coffee = Coffee {

  id: 123456,

  coffee_type: CoffeeType::HOT,

  name: String::new(“Latte”)
}
```

◀ **An enum is defined via the “enum” keyword**

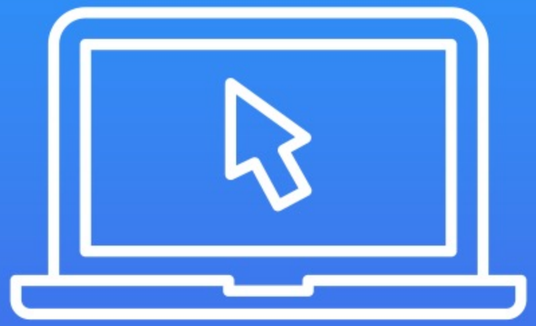
◀ **Using an enum looks like this**

```
// Including data with enums

enum CoffeeType {
  HOT(f32),
  ICED(f32)
}

// Example usage

let my_coffee = Coffee {
  id: 123456,
  coffee_type: CoffeeType::HOT(102.5),
  name: String::new(“Latte”)
}
```

- An enum can have data attached to it as well

- Example of an enum with data attached. Here we have the temperature attached to it.

Demo



Primitive data types

Custom data types

- Structs
- Enums