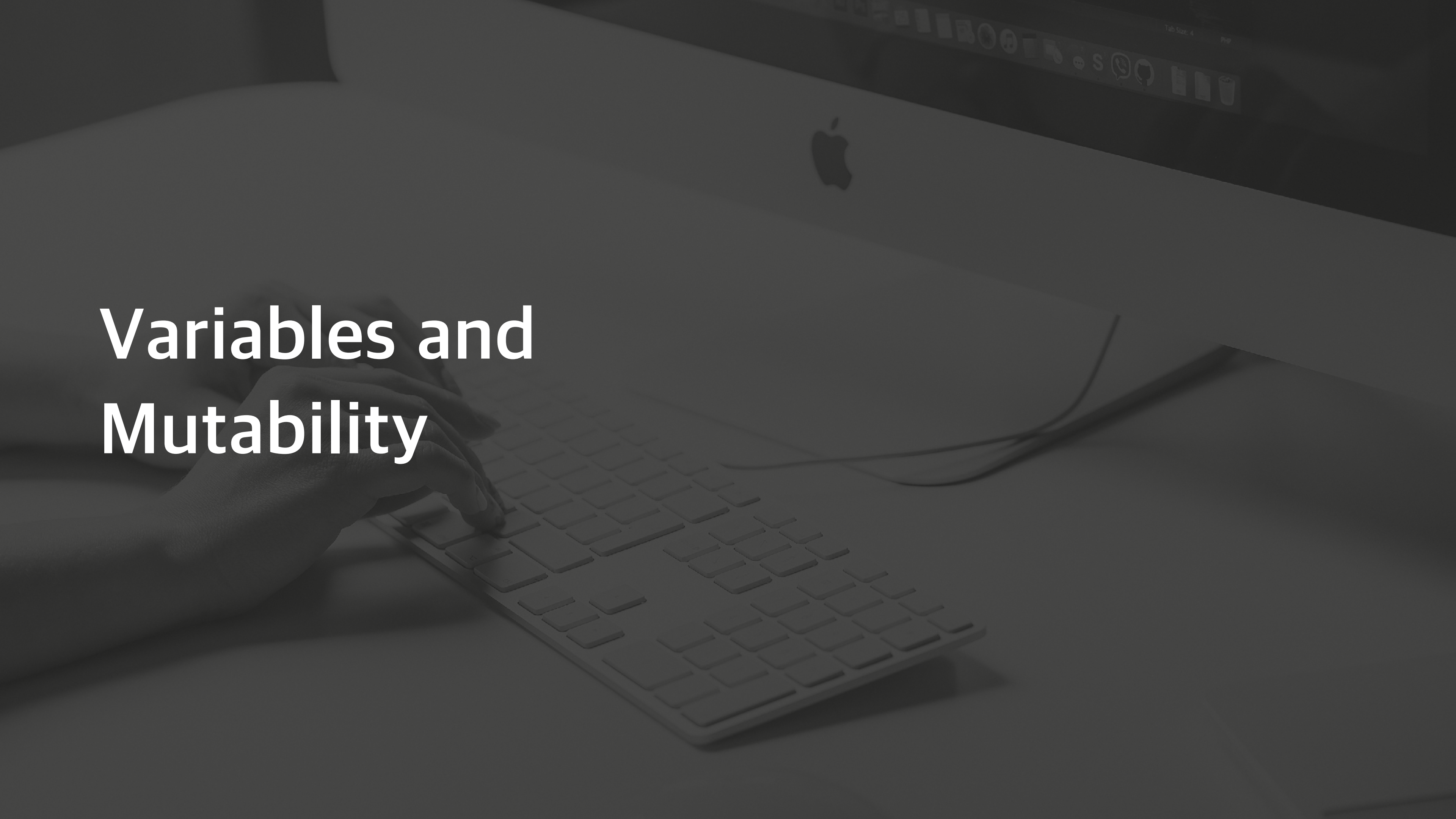
한 번에 끝내는 블록체인 개발 A to Z

Chapter 1

Rust Introduction

Chapter 1 Rust Introduction

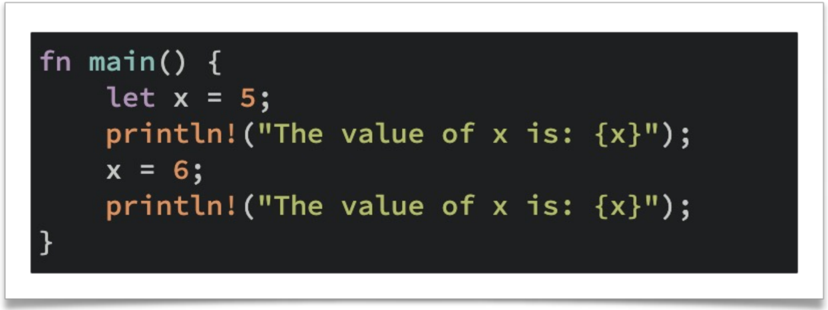
Common Programming Concept



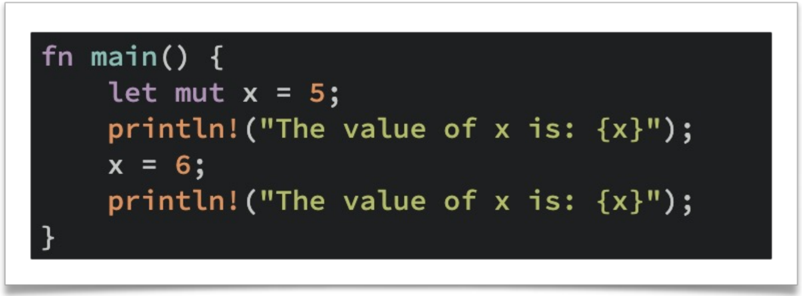
Variables and Mutability

Variable Mutability

Variables are immutable only by default. you can make them mutable by adding mut in front of the variable name. Adding mut also conveys intent to future readers of the code by indicating that other parts of the code will be changing this variable’s value.

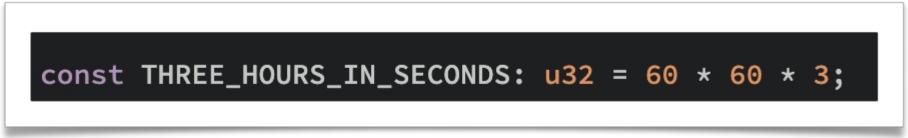


```
fn main() {
    let x = 5;
    println!("The value of x is: {x}");
    x = 6;
    println!("The value of x is: {x}");
}
```



```
fn main() {
    let mut x = 5;
    println!("The value of x is: {x}");
    x = 6;
    println!("The value of x is: {x}");
}
```

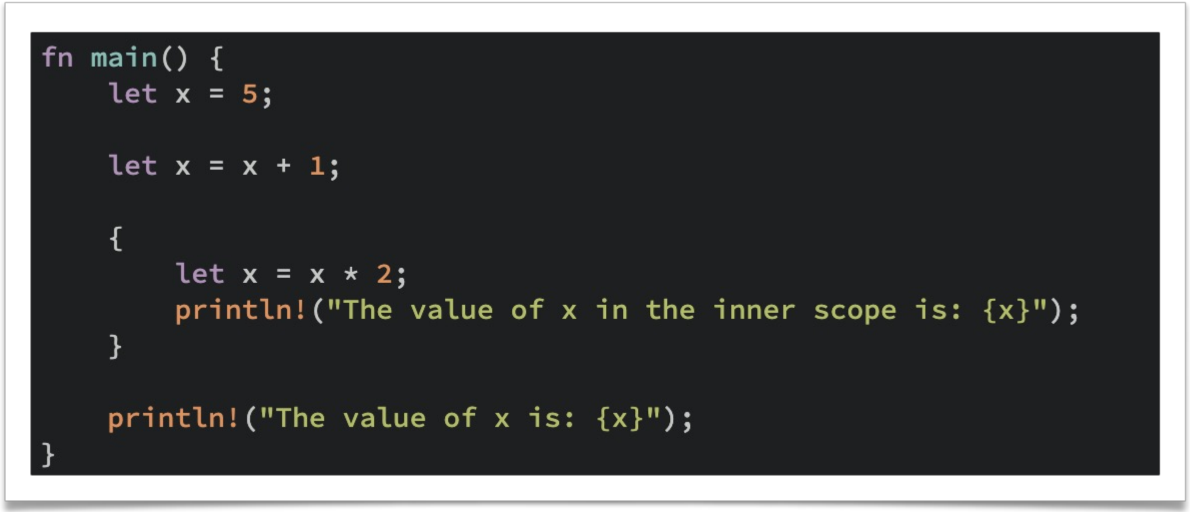
Constants & Shadowing



```
const THREE_HOURS_IN_SECONDS: u32 = 60 * 60 * 3;
```

Constants aren’t just immutable by default—they’re always immutable.

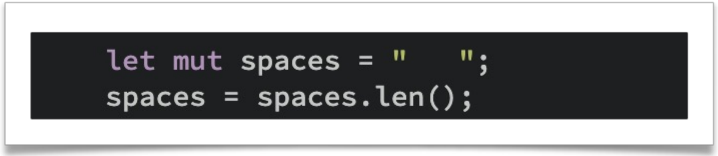
We can declare a new variable with the same name as a previous variable. Then we say that the first variable is shadowed by the second.



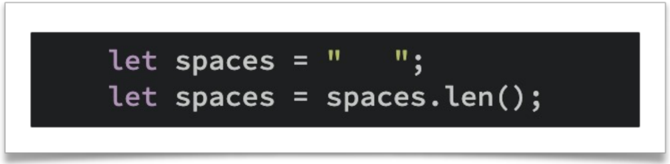
```
fn main() {
    let x = 5;

    let x = x + 1;

    {
        let x = x * 2;
        println!("The value of x in the inner scope is: {x}");
    }

    println!("The value of x is: {x}");
}
```

Shadowing vs Mutability

Shadowing is different from marking a variable as mut, because we're effectively creating a new variable when we use the let keyword again, we can change the type of the value but reuse the same name.



```
let mut spaces = "   ";
spaces = spaces.len();
```

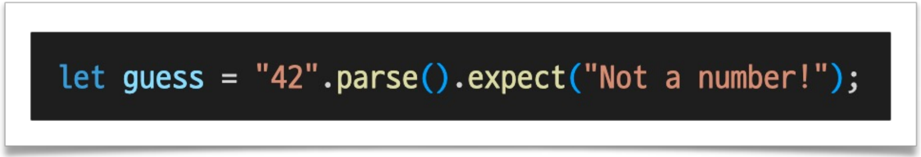


```
let spaces = "   ";
let spaces = spaces.len();
```

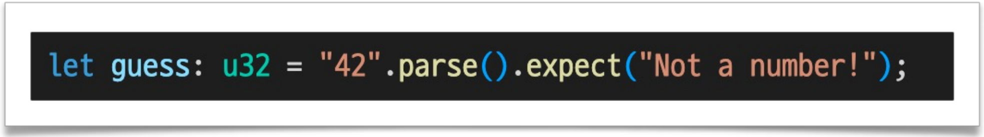
Data Types

Typed Language

Keep in mind that Rust is a statically typed language, which means that it must know the types of all variables at compile time. The compiler can usually infer what type we want to use based on the value and how we use it.



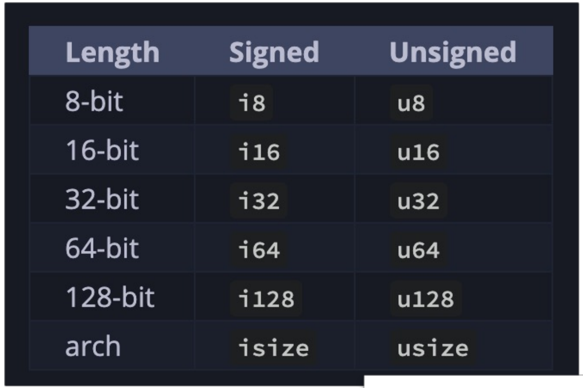
```
let guess = "42".parse().expect("Not a number!");
```



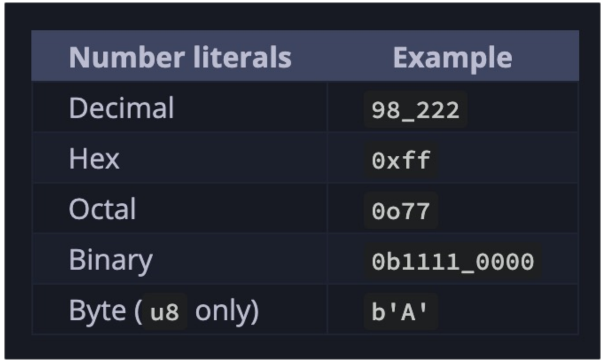
```
let guess: u32 = "42".parse().expect("Not a number!");
```

Scalar Types

Rust has four primary scalar types: integers, floating-point numbers, Booleans, and characters.



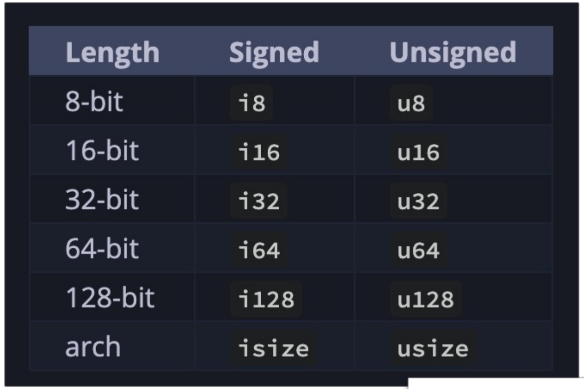
Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`



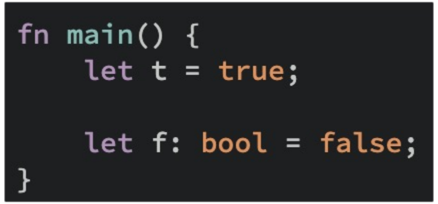
Number literals	Example
Decimal	`98_222`
Hex	`0xff`
Octal	`0o77`
Binary	`0b1111_0000`
Byte (`u8` only)	`b'A'`

Scalar Types

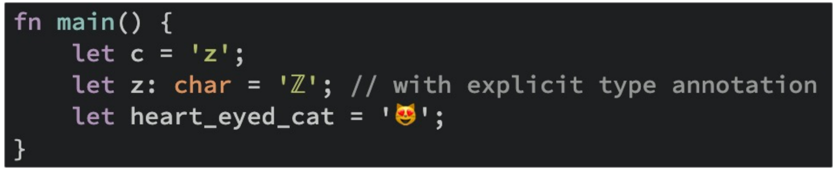
Rust has four primary scalar types: integers, floating-point numbers, Booleans, and characters.



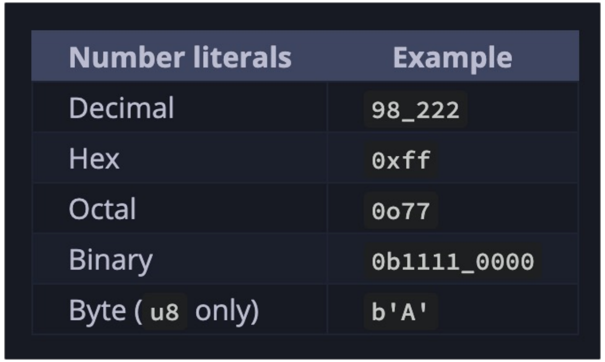
Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`



```
fn main() {
    let t = true;

    let f: bool = false;
}
```



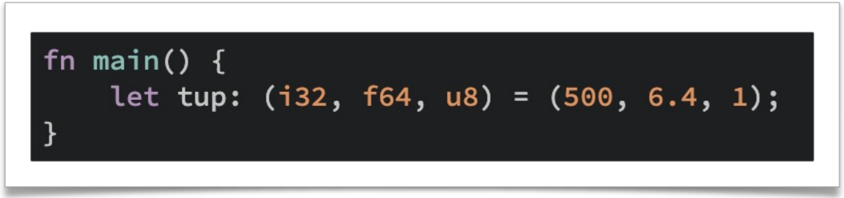
```
fn main() {
    let c = 'z';
    let z: char = 'ℤ'; // with explicit type annotation
    let heart_eyed_cat = '😻';
}
```



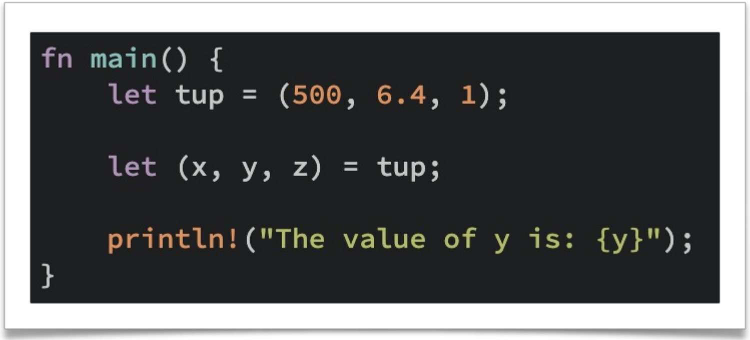
Number literals	Example
Decimal	`98_222`
Hex	`0xff`
Octal	`0o77`
Binary	`0b1111_0000`
Byte (`u8` only)	`b'A'`

Compound Types

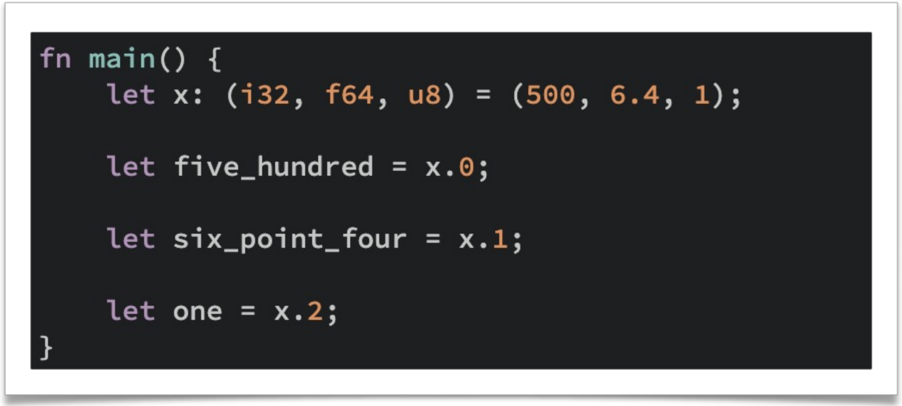
Compound types can group multiple values into one type. Rust has two primitive compound types: tuples and arrays.



```rust
fn main() {
    let tup: (i32, f64, u8) = (500, 6.4, 1);
}
```



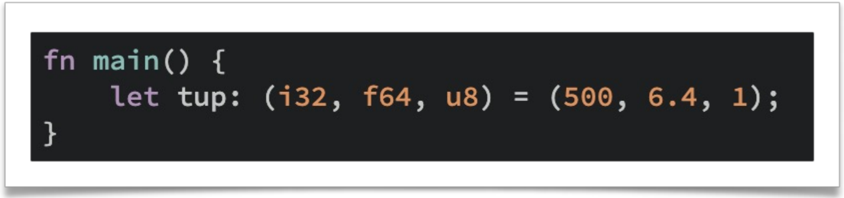
```rust
fn main() {
    let tup = (500, 6.4, 1);

    let (x, y, z) = tup;

    println!("The value of y is: {y}");
}
```



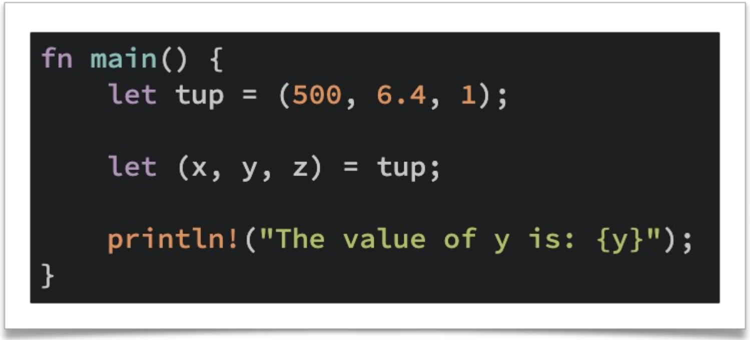
```rust
fn main() {
    let x: (i32, f64, u8) = (500, 6.4, 1);

    let five_hundred = x.0;

    let six_point_four = x.1;

    let one = x.2;
}
```

Compound Types

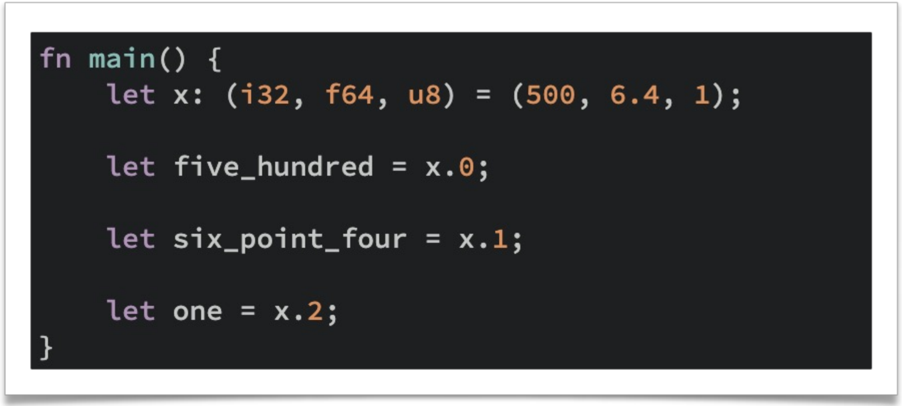
Compound types can group multiple values into one type. Rust has two primitive compound types: tuples and arrays.



```
fn main() {
    let tup: (i32, f64, u8) = (500, 6.4, 1);
}
```



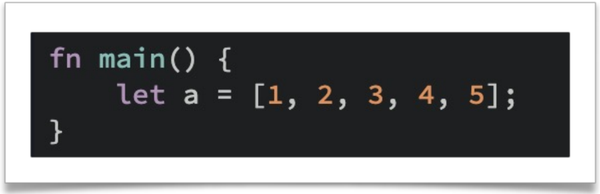
```
fn main() {
    let tup = (500, 6.4, 1);

    let (x, y, z) = tup;

    println!("The value of y is: {y}");
}
```



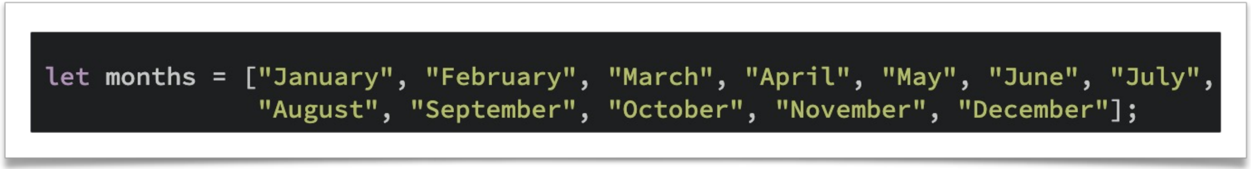
```
fn main() {
    let x: (i32, f64, u8) = (500, 6.4, 1);

    let five_hundred = x.0;

    let six_point_four = x.1;

    let one = x.2;
}
```

Array Type

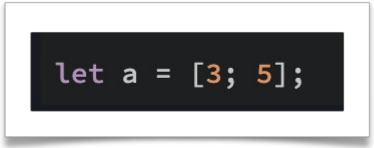
Another way to have a collection of multiple values is with an array. Unlike a tuple, every element of an array must have the same type. Unlike arrays in some other languages, arrays in Rust have a fixed length.



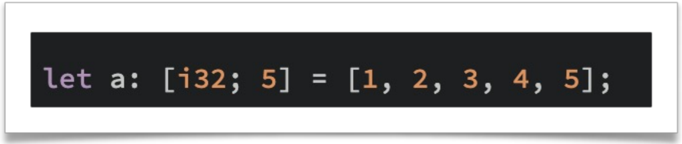
```
fn main() {
    let a = [1, 2, 3, 4, 5];
}
```



```
let months = ["January", "February", "March", "April", "May", "June", "July",
              "August", "September", "October", "November", "December"];
```



```
let a = [3; 5];
```

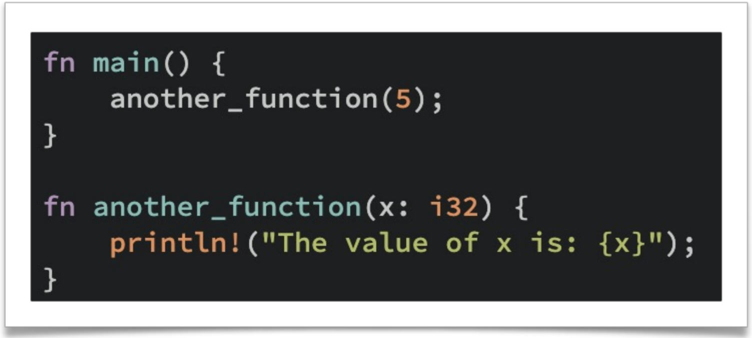


```
let a: [i32; 5] = [1, 2, 3, 4, 5];
```

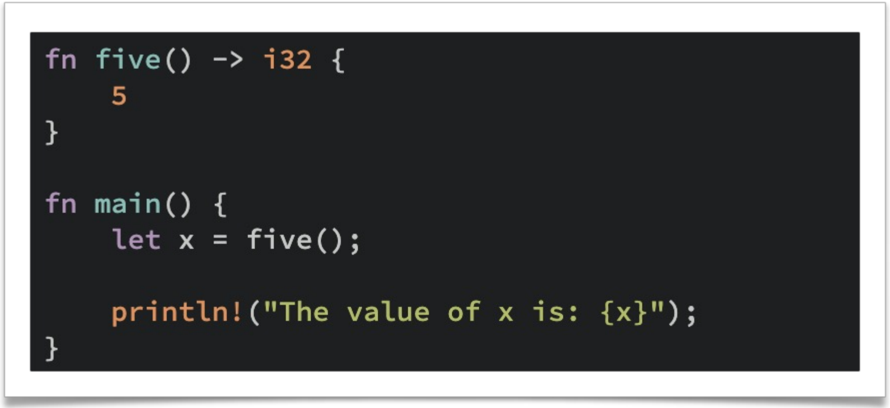
Functions & Comments

Functions

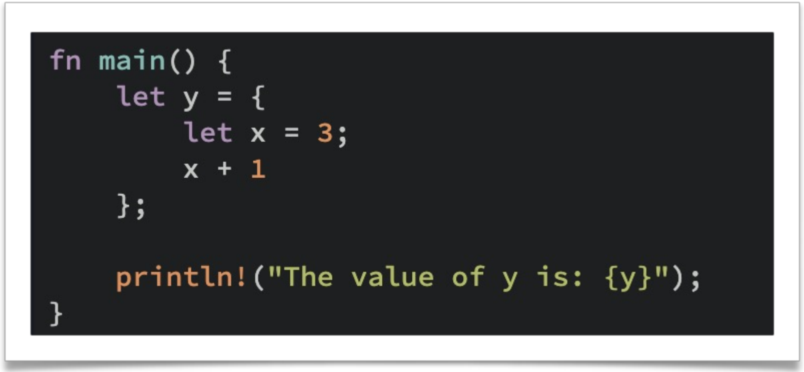
Rust code uses snake case as the conventional style for function and variable names, in which all letters are lowercase and underscores separate words.



```rust
fn main() {
    another_function(5);
}

fn another_function(x: i32) {
    println!("The value of x is: {x}");
}
```

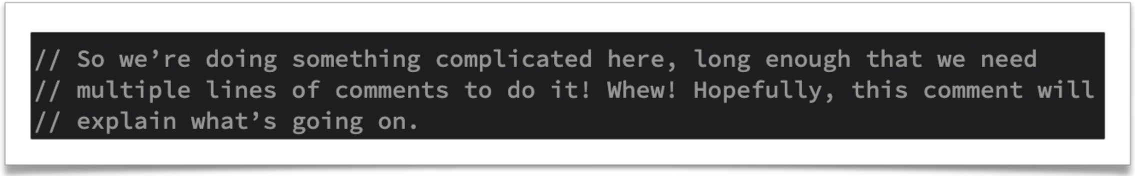


```rust
fn five() -> i32 {
    5
}

fn main() {
    let x = five();

    println!("The value of x is: {x}");
}
```



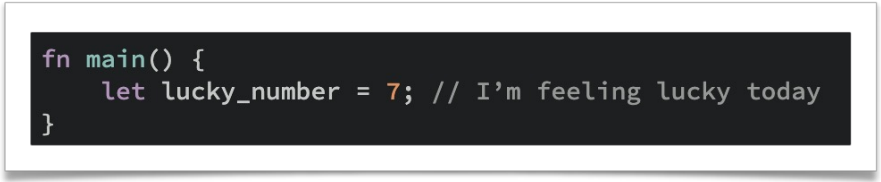
```rust
fn main() {
    let y = {
        let x = 3;
        x + 1
    };

    println!("The value of y is: {y}");
}
```

Comments

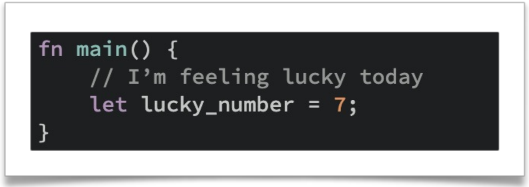


```
// So we’re doing something complicated here, long enough that we need
// multiple lines of comments to do it! Whew! Hopefully, this comment will
// explain what’s going on.
```

In Rust, the idiomatic comment style starts a comment with two slashes, and the comment continues until the end of the line. For comments that extend beyond a single line, you’ll need to include // on each line.



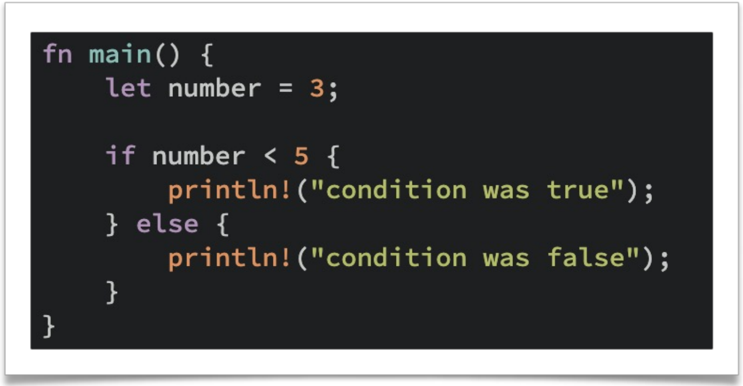
```
fn main() {
    let lucky_number = 7; // I’m feeling lucky today
}
```



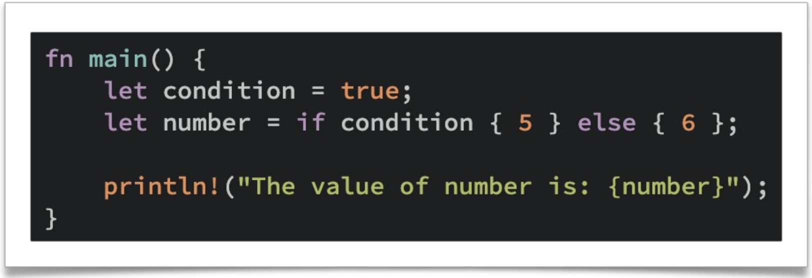
```
fn main() {
    // I’m feeling lucky today
    let lucky_number = 7;
}
```

Control Flow

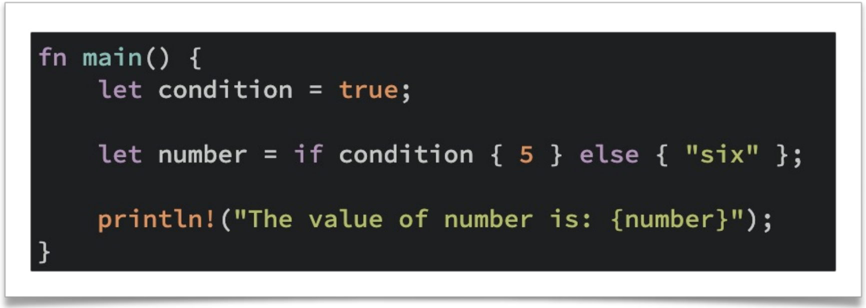
if



```rust
fn main() {
    let number = 3;

    if number < 5 {
        println!("condition was true");
    } else {
        println!("condition was false");
    }
}
```

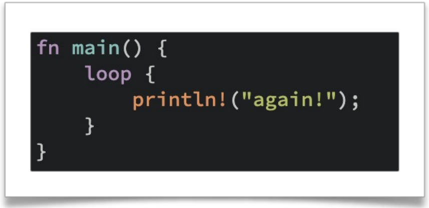


```rust
fn main() {
    let condition = true;
    let number = if condition { 5 } else { 6 };

    println!("The value of number is: {number}");
}
```

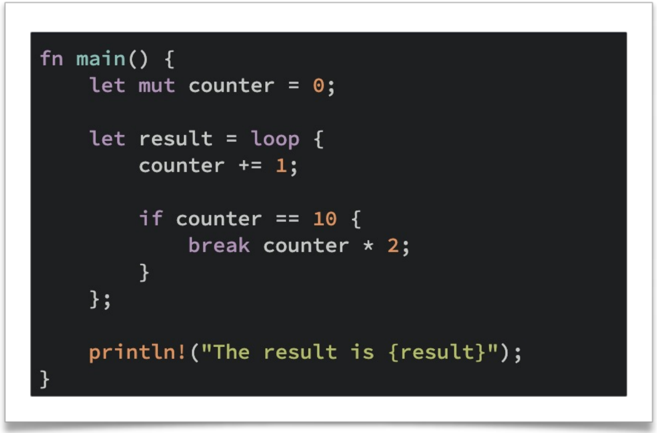


```rust
fn main() {
    let condition = true;

    let number = if condition { 5 } else { "six" };

    println!("The value of number is: {number}");
}
```

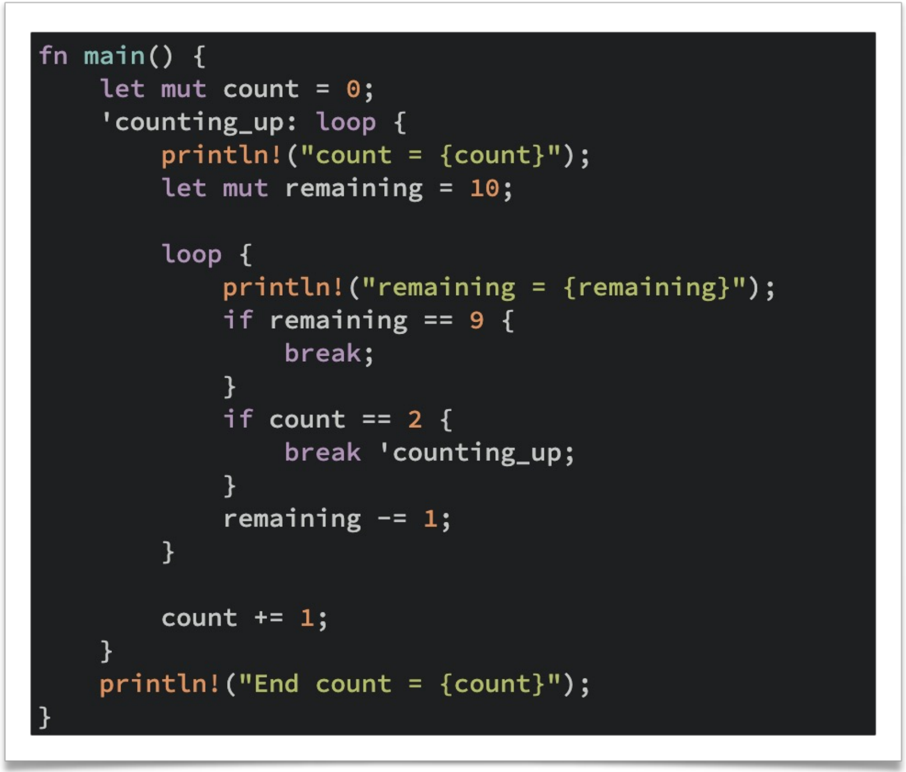
loop



```rust
fn main() {
    loop {
        println!("again!");
    }
}
```

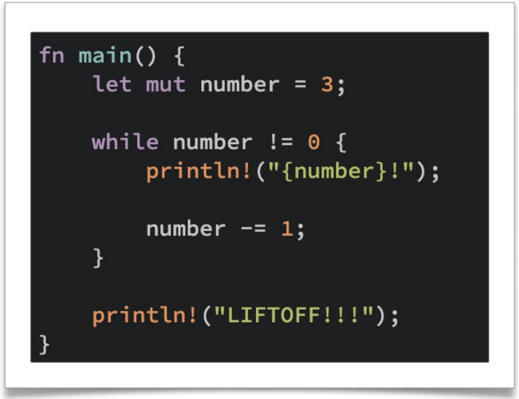


```rust
fn main() {
    let mut counter = 0;

    let result = loop {
        counter += 1;

        if counter == 10 {
            break counter * 2;
        }
    };

    println!("The result is {result}");
}
```

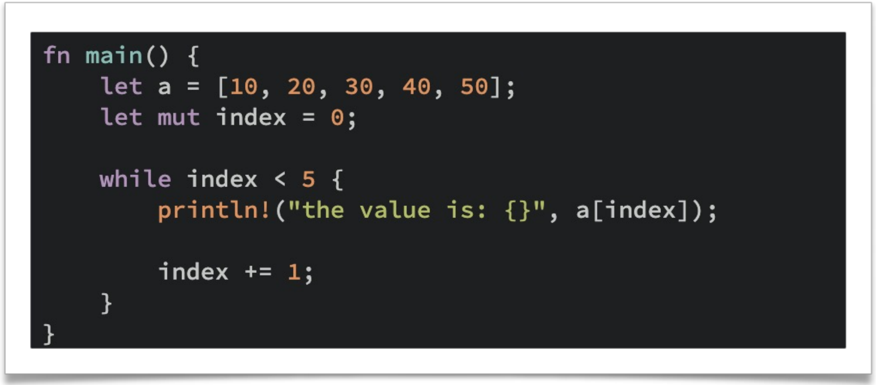


```rust
fn main() {
    let mut count = 0;
    'counting_up: loop {
        println!("count = {count}");
        let mut remaining = 10;

        loop {
            println!("remaining = {remaining}");
            if remaining == 9 {
                break;
            }
            if count == 2 {
                break 'counting_up;
            }
            remaining -= 1;
        }

        count += 1;
    }
    println!("End count = {count}");
}
```

While

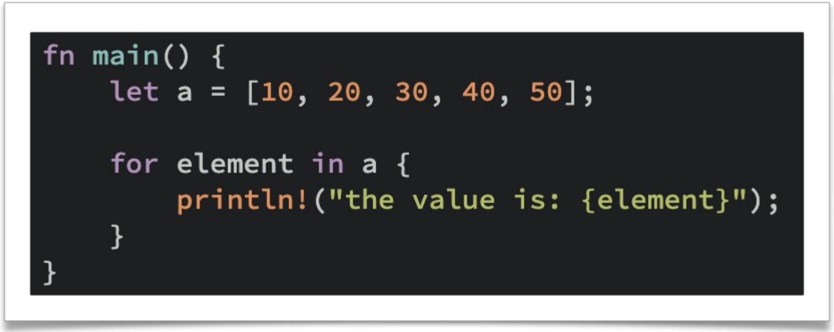


```
fn main() {
    let mut number = 3;

    while number != 0 {
        println!("{number}!");

        number -= 1;
    }

    println!("LIFTOFF!!!");
}
```

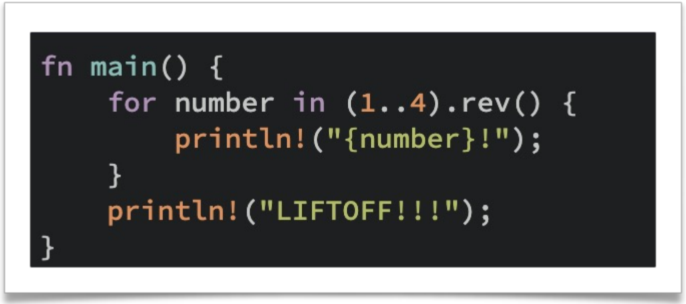


```
fn main() {
    let a = [10, 20, 30, 40, 50];
    let mut index = 0;

    while index < 5 {
        println!("the value is: {}", a[index]);

        index += 1;
    }
}
```

a Collection Iteration



```rust
fn main() {
    let a = [10, 20, 30, 40, 50];

    for element in a {
        println!("the value is: {element}");
    }
}
```



```rust
fn main() {
    for number in (1..4).rev() {
        println!("{number}!");
    }
    println!("LIFTOFF!!!");
}
```