控制流：if else 、loop

If 表达式允许根据条件来执行不同的代码分支（**注意：if 是表达式!!!**）
-这个条件必须是 bool 类型
If 表达式中 与条件相关联的代码块就叫做分支（arm）
可选地 在后边可以加上一个 else 表达式

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

条件必须是布尔类型的 不然就会报错（不像 c 会把非布尔类型的值转换成布尔类型）

使用 else if 来处理多重条件

如果使用了多于一个 else if 那么最好使用 match 来重构代码（代码会比较凌乱）

在 let 语句中使用 if

因为 **if 是一个表达式，所以可以将它放在 let 语句中等号的右边**

```
let condition = true;

let number = if condition { 5 } else { 6 };
```

If 和 else 中的类型必须能够提前确定 如果 condition 和 else 中的类型不一样 那么 number1 就无法确定返回什么类型的值 编译器会报错
所以就要求 if else1 每个分支返回的类型必须是一样的 不然 number 就不知道怎么弄了

三种循环：loop while for

Loop 关键字：反复执行一段代码 直到喊停
在 loop 循环中使用 break 来停止

```
fn main() {
    let mut counter = 0;
        i32
    let result = loop {
        counter += 1;

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

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

While 条件循环：每次执行循环体之前都判断一次条件

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

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

        number = number - 1;
    }

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

For 循环遍历集合 安全性和简洁性

Loop 和 while 效率比较低

- 可以使用 while 或 loop 来遍历集合，但是易错且低效（例子）
- 使用 for 循环更简洁紧凑，它可以针对集合中的每个元素来执行一些代码

调用 **iter 办法**

```rust
fn main() {
    let a = [10, 20, 30, 40, 50];
    for element in a.iter() {
        println!("the value is: {}", element);
    }
}
```

Range 类型实现遍历

- 标准库提供
- 指定一个开始数字和一个结束数字，Range 可以生成它们之间的数字（不含结束）
- rev 方法可以反转 Range

src > main.rs > ...

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