Rust是静态强类型语言，Rust 的每个值都有确切的数据类型，总的来说可以分为两类：基本类型和复合类型。 基本类型意味着它们往往是一个最小化原子类型，无法解构为其它类型(一般意义上来说)。

1. 整数（演示代码在`main()`中）

1.1. 整数类型

i代表的是integer，u代表的是unsigned integer。写一个整数，它默认是i32类型的数，除非添加类型标识。isize 和 usize 类型取决于程序运行的计算机 CPU 类型： 若 CPU 是 32 位的，则这两个类型是 32 位的，同理，若 CPU 是 64 位，那么它们则是 64 位。

长度	有符号	无符号
8 bit	i8	u8
16 bit	i16	u16
32 bit	i32（默认）	u32
64 bit	i64	u64
128 bit	i128	u128
arch	isize	usize

1.2. 整数字面量的表示

数字字面量	示例
十进制	98_222
十六进制	0xff
八进制	0o77
二进制	0b1111_0000
字节 (仅限于 u8)	b’A’

1.3. 整数溢出

以debug模式编译时，会检查溢出，如果溢出则panic!()，以release模式编译时，不检查溢出。如果要显示处理可能的溢出，可以使用整数类型的方法：

- 使用 wrapping_* 方法在所有模式下都按照补码循环溢出规则处理，例如 wrapping_add
- 使用 checked_* 方法时发生溢出，则返回 None 值
- 使用 overflowing_* 方法返回该值和一个指示是否存在溢出的布尔值
- 使用 saturating_* 方法使值达到最小值或最大值

2. 浮点数（演示在`float_type()`中）

2.1. 浮点数类型

浮点类型数字 是带有小数点的数字，在 Rust 中浮点类型数字也有两种基本类型： f32 和 f64，分别为 32 位和 64 位大小。默认浮点类型是 f64，在现代的 CPU 中它的速度与 f32 几乎相同，但精度更高。

长度	类型
32 bit	f32
64 bit	f64（默认）

2.2. 浮点数陷阱

- 浮点数往往是你想要数字的近似表达：浮点数类型是基于二进制实现的，但是我们想要计算的数字往往是基于十进制，例如 0.1 在二进制上并不存在精确的表达形式，但是在十进制上就存在。
- 浮点数在某些特性上是反直觉的：浮点数是可以比较的，比较运算实现的是 std::cmp::PartialEq 特征，但是并没有实现 std::cmp::Eq 特征。

2.3. NaN

对于数学上未定义的结果，例如对负数取平方根 -42.1.sqrt() ，会产生一个特殊的结果：Rust 的浮点数类型使用 NaN (not a number)来处理这些情况。所有跟 NaN 交互的操作，都会返回一个 NaN，而且 NaN 不能用来比较。

3. 数字运算（演示在`operations()`中）

Rust 支持所有数字类型的基本数学运算：加法、减法、乘法、除法和取模运算。

4. 位运算（演示在`bitwise_operations()`中）

运算符	位运算
&	相同位置均为1时则为1，否则为0
|	相同位置只要有1时则为1，否则为0
^	相同位置不相同则为1，相同则为0
!	把位中的0和1相互取反，即0置为1，1置为0
<<	所有位向左移动指定位数，右位补0
>>	所有位向右移动指定位数，带符号移动（正数补0，负数补1）

5. 总结

- Rust 拥有相当多的数值类型. 因此你需要熟悉这些类型所占用的字节数，这样就知道该类型允许的大小范围以及你选择的类型是否能表达负数
- 类型转换必须是显式的. Rust 永远也不会偷偷把你的 16bit 整数转换成 32bit 整数
- Rust 的数值上可以使用方法. 例如你可以用以下方法来将 13.14 取整：13.14_f32.round()，在这里我们使用了类型后缀，因为编译器需要知道 13.14 的具体类型