//枚举与模式匹配

//枚举允许我们列举所有可能指来定义类型

//枚举所有可能指叫枚举的变体,枚举变体都位于标识符命名空间下使用::分隔

// enum IpAddrKind{

//   v4,

//   v6,

// }

// // let four = IpAddrKind::v4;

// // let six = IpAddrKind::v6;

// // fn main(){

// //   let four = IpAddrKind::v4;

// //   let six = IpAddrKind::v6;

// // }

// fn route(ip\_kind: IpAddrKind){}

// //将数据附加到枚举变体中

// struct IpAddr{

//   kind: IpAddrKind,

//   adress: String,

// }

// fn main(){

//   let home = IpAddr{

//     kind:IpAddrKind::v4,

//     adress:String::from("127.0.0.1"),

//   };

//   let loopback = IpAddr{        //使用struct来存贮

//     kind:IpAddrKind::v6,

//     adress:String::from("::1"),

//   };

// }

//优点:不需要额外使用struct,每个变体都有不同类型以及相关数据源

// enum  IpAddr{

//   v4(u8,u8,u8,u8),

//   v6(String),

// }

//为枚举定义方法

//impl

// enum Message{

//   Quit,//无关联

//   Move{x:i32,y:i32},//匿名结构体

//   Write(String),

//   ChangeColor(i32,i32,i32),

// }

// impl Message{

//   fn call(&self){}

// }

// fn main(){

//   let g = Message::Quit;

//   let m = Message::Move{x: 12,y:34};

//   let w = Message::Write(String::from("Hello"));

//   let c = Message::ChangeColor(0,255,255);

//   m.call();

// }

//option枚举

//其它语言中,null表示一个值,

//rust没有null,有一个类似的枚举option

//标准库中定义:

// enum Option<T>{

//   Some(T),//关联一个数据,类型是泛型,表示存在值,防止值为空

//   None,

// }

// fn main() {

//     let some\_number = Some(5);

//     let some\_number = Some("A String");

//     let absent\_number:Option<i32> = None;

//   }0

//Option<T>和T是不同类型

// fn main(){

//   let x:i8 = 5;

//   let y:Option<i8> = Some(5);

//   let sum = x + y;会直接报错,所以不会发生null和一个值和null相加的情况

// }

//match---控制流运算符

//允许一个值和一系列模式进行匹配,并执行匹配的模式对应的代码

//模式可以是字面值,变量名,通配符等

// enum Coin {

//     Penny,

//     Nickel,

//     Dime,

//     Quarter,

// }

// fn value\_in\_cents(coin: Coin) -> u8 {

//     match coin {

//         Coin::Penny => {

//           println!("Penny lene");

//           1

//         }

//         Coin::Nickel => 5,

//         Coin::Dime => 10,

//         Coin::Quarter => 25,

//     }

// }

// fn main() {}

//绑定值的模式

//匹配的分支可以匹配到绑定对象的部分值,因此可以从enum变体中提取提取值

// #[derive(Debug)]

// enum UsState{

//   Alabama,

//   Alaska,

// }

// enum Coin {

//     Penny,

//     Nickel,

//     Dime,

//     Quarter(UsState),//关联一个数据

// }

// fn value\_in\_cents(coin:Coin) -> u8{

//   match coin{

//     Coin::Penny => {

//       println!("Penny!");

//       1

//     }

//     Coin::Nickel => 5,

//     Coin::Dime => 10,

//     Coin::Quarter(state) => {//绑定一个值let state = USState

//       println!("State quarter from {:?}!",state);

//       25

//     }

//   }

// }

// fn main(){

//   let c = Coin::Quarter(UsState::Alaska);

//   println!("{}",value\_in\_cents(c));

// }

//匹配Option<T>

// fn main(){

//   let five = Some(5);

//   let six = plus\_one(five);

//   let none = plus\_one(None);

// }

// fn plus\_one(x:Option<i32>) -> Option<i32>{

//   match x{

//     None => None,

//     Some(i) => Some(i + 1),

//   }

// }

//match模式匹配必须穷举所有可能

//也可以用\_通配符来代替

// match{

//   1 => println!("one"),

//   2 => println!("two"),

//   3 => println!("three"),

//   \_ => println!("something else"),

// }

//if let

//只关心一种匹配而忽略其他匹配的情况

// fn main() {

//     let v = Some(0u8);

//     match v {

//         Some(3) => println!("three"),

//         \_ => (),

//     }

//     if let Some(3) = v {

//         println!("three");

//     }else{

//       println!("others");

//     }

// }

//

//

//

//

//Package Create Module

//模块系统:

// Package(包):Cargo特性,用来构建,测试,共享crate

  //包含一个Cargo.toml,讲述如何构建这些Crates

      //mian.rs源代码文件,就是package的入口文件,但是在toml中没有提到这个文件

          //因为src/main.rs是binary crate的crate root,crate名与包名相同

          //src/lib.rs说明package包含一个library crate,这个文件是library crate的crate root,crate名与包名相同

      //cargo会把crate root文件交给rustc编译器来构建library或binary crate

  //只能包含0-1个library crate,任意数量binary crate

  //至少包含一个crate

// Crate(单元包):树形模块,产生一个library或可执行文件

    //binary:二进制

    //library:库

    //crate root:源代码文件,rust编译器从这里开始,组成你的Crate的根Module

    //可以将相关功能组合到一个作用域内,便于项目之间共享

//Module(模块),use:让你控制代码组织,作用域,私有路径

    //在一个crate内,将代码进行分组,增加可读性,易于复用,可以控制item是public还是private

    //可嵌套,可包含其他项(struct,function,enum,trait,常量等)

       //在lib.rs中

       //mod front\_of\_house{

      //   mod hosting{

      //     fn add\_to\_waitlist(){}

      //     fn seat\_at\_table(){}

      //   }

      //   mod serving{

      //     fn take\_order(){}

      //     fn serve\_order(){}

      //     fn take\_payment(){}

      //   }

      // }                         src/main.rs,src/lib.rs,叫做crate root,这两个文件内容形成了一个crate,位于整个模块树位于根部

//Path(路径):为struct,function或module等项目命名