Tham Khảo Nhanh Rust

Ownership, kiểu dữ liệu, trait và pattern matching thiết yếu

Cơ Bản

Hello World

```
fn main() {
    println!("Hello, World!");
}
```

Lệnh Cargo

```
cargo new my_project    # create new project
cargo build             # compile (debug)
cargo build --release   # compile (optimized)
cargo run               # build and run
cargo test              # run tests
```

Cấu Trúc Dự Án

`Cargo.toml`	Manifest dự án (dependency, metadata)
`src/main.rs`	Điểm vào binary crate
`src/lib.rs`	Gốc library crate
`tests/`	Thư mục integration test

Biến & Tính Biến Đổi

Binding & Tính Biến Đổi

```
let x = 5;              // immutable by default
let mut y = 10;         // mutable
y += 1;
const MAX: u32 = 100;   // compile-time constant
```

Shadowing

```
let x = 5;
let x = x + 1;          // shadows previous x
let x = "now a string"; // can change type
```

Kiểu Vô Hướng

`i8..i128, isize`	Số nguyên có dấu
`u8..u128, usize`	Số nguyên không dấu
`f32, f64`	Số thực dấu phẩy động (f64 mặc định)
`bool`	**`true`** / **`false`**
`char`	Giá trị scalar Unicode (4 bytes)

Kiểu Ghép

```
let tup: (i32, f64, char) = (42, 6.4, 'z');
let (a, b, c) = tup;        // destructure
let arr: [i32; 3] = [1, 2, 3];
let first = arr[0];
```

Hàm

Định Nghĩa

```
fn add(a: i32, b: i32) -> i32 {
    a + b   // no semicolon = return expression
}
```

Closure

```
let double = |x: i32| x * 2;
let sum: i32 = vec![1, 2, 3]
    .iter()
    .map(|x| x * 2)
    .sum();
```

Con Trỏ Hàm & Trait

`fn(T) -> U`	Kiểu con trỏ hàm
`Fn(T) -> U`	Closure mượn tham chiếu
`FnMut(T) -> U`	Closure mượn tham chiếu có thể thay đổi
`FnOnce(T) -> U`	Closure lấy quyền sở hữu

Luồng Điều Khiển

If / Else

```
let status = if score >= 90 { "A" }
             else if score >= 80 { "B" }
             else { "C" };  // if is an expression
```

Vòng Lặp

```
loop { break; }                 // infinite
while condition { }             // while
for item in &vec { }            // iterator
for i in 0..10 { }              // range
for (i, v) in vec.iter().enumerate() { }
```

Nhãn Vòng Lặp

```
'outer: for i in 0..5 {
    for j in 0..5 {
        if i + j > 6 { break 'outer; }
    }
}
```

Ownership & Borrowing

Quy Tắc Ownership

1. Mỗi giá trị có đúng một owner.
2. Khi owner ra khỏi scope, giá trị bị drop.
3. Giá trị có thể được move hoặc clone.

Move & Clone

```
let s1 = String::from("hello");
let s2 = s1;          // s1 is moved, no longer valid
let s3 = s2.clone();  // deep copy, both valid
```

Borrowing

```
fn len(s: &String) -> usize { s.len() }  // shared ref
fn push(s: &mut String) { s.push('!'); }  // mutable ref
// Rule: many &T OR one &mut T, never both
```

Lifetime

```
fn longest<'a>(a: &'a str, b: &'a str) -> &'a str {
    if a.len() > b.len() { a } else { b }
}
```

Struct & Enum

Struct

```
struct User {
    name: String,
    age: u32,
    active: bool,
}
let u = User { name: String::from("Alice"), age: 30, active: true };
```

Khối Impl

```
impl User {
    fn new(name: &str, age: u32) -> Self {
        Self { name: name.to_string(), age, active: true }
    }
    fn greeting(&self) -> String {
        format!("Hi, {}", self.name)
    }
}
```

Enum

```
enum Shape {
    Circle(f64),
    Rect { w: f64, h: f64 },
    Point,
}
let s = Shape::Circle(5.0);
```

Pattern Matching

Biểu Thức Match

```
match shape {
    Shape::Circle(r) => std::f64::consts::PI * r * r,
    Shape::Rect { w, h } => w * h,
    Shape::Point => 0.0,
}
```

If Let & While Let

```
if let Some(val) = optional {
    println!("{val}");
}
while let Some(top) = stack.pop() {
    println!("{top}");
}
```

Cú Pháp Pattern

`_`	Ký tự đại diện, khớp bất kỳ
`x @ 1..=5`	Gán dải khớp cho **`x`**
`(a, b, ..)`	Destructure tuple, bỏ qua phần còn lại
`Some(x) if x > 0`	Guard điều kiện
`Foo { x, .. }`	Struct, bỏ qua các trường khác

Xử Lý Lỗi

Result & Option

```
enum Result<T, E> { Ok(T), Err(E) }
enum Option<T>    { Some(T), None }
```

Toán Tử ?

```
fn read_file(path: &str) -> Result<String, io::Error> {
    let mut s = String::new();
    File::open(path)?.read_to_string(&mut s)?;
    Ok(s)
}
```

Xử Lý Lỗi

```
match result {
    Ok(val) => println!("{val}"),
    Err(e) => eprintln!("Error: {e}"),
}
let val = result.unwrap_or(0);
let val = result.unwrap_or_else(|_| default());
```

Phương Thức Phổ Biến

`.unwrap()`	Lấy giá trị hoặc panic
`.expect(msg)`	Lấy giá trị hoặc panic với thông báo
`.unwrap_or(default)`	Lấy giá trị hoặc dùng mặc định
`.map(f)`	Biến đổi giá trị Ok/Some
`.and_then(f)`	Nối chuỗi thao tác (flatmap)
`.is_ok() / .is_some()`	Kiểm tra boolean

Tham Khảo Nhanh Rust

Trait

Định Nghĩa & Triển Khai

```
trait Summary {
    fn summarize(&self) -> String;
    fn preview(&self) -> String {   // default impl
        format!("{}...", &self.summarize()[..20])
    }
}
impl Summary for User {
    fn summarize(&self) -> String { self.name.clone() }
}
```

Ràng Buộc Trait

```
fn notify(item: &impl Summary) { }
fn notify<T: Summary + Display>(item: &T) { }
fn notify(item: &(impl Summary + Display)) { }
```

Trait Phổ Biến

`Display`	Định dạng chuỗi cho người dùng
`Debug`	Định dạng debug (**`{:?}`**)
`Clone, Copy`	Sao chép (sâu / theo bit)
`PartialEq, Eq`	So sánh bằng nhau
`PartialOrd, Ord`	So sánh thứ tự
`Iterator`	**`next()`** để duyệt
`From, Into`	Chuyển đổi kiểu
`Default`	Hàm khởi tạo giá trị mặc định

Collection

Vec

```
let mut v: Vec<i32> = vec![1, 2, 3];
v.push(4);
v.pop();              // returns Option<i32>
let first = &v[0];   // panics if empty
let first = v.get(0); // returns Option<&i32>
```

HashMap

```
use std::collections::HashMap;
let mut m = HashMap::new();
m.insert("key", 42);
m.entry("key").or_insert(0);
if let Some(val) = m.get("key") { }
```

String

```
let s = String::from("hello");
let s = "hello".to_string();
let combined = format!("{} {}", s, "world");
for c in s.chars() { }   // iterate characters
```

Iterator

```
let sum: i32 = vec![1, 2, 3].iter().sum();
let doubled: Vec<_> = v.iter().map(|x| x * 2).collect();
let evens: Vec<_> = v.iter().filter(|x| *x % 2 == 0).collect();
```

Đồng Thời

Thread

```
use std::thread;
let handle = thread::spawn(|| {
    println!("from spawned thread");
});
handle.join().unwrap();
```

Channel

```
use std::sync::mpsc;
let (tx, rx) = mpsc::channel();
tx.send(42).unwrap();
let val = rx.recv().unwrap();
```

Trạng Thái Dùng Chung

`Arc<T>`	Đếm tham chiếu nguyên tử (Rc an toàn với thread)
`Mutex<T>`	Loại trừ lẫn nhau, khóa để truy cập giá trị
`RwLock<T>`	Nhiều reader hoặc một writer
`Send`	Trait: an toàn khi chuyển giữa các thread
`Sync`	Trait: an toàn khi chia sẻ tham chiếu giữa thread

Macro & Attribute

Macro Phổ Biến

`println!()`	In kèm xuống dòng
`format!()`	Trả về String đã định dạng
`vec![]`	Tạo Vec từ literal
`todo!()`	Placeholder, panic lúc chạy
`assert!(expr)`	Panic nếu expr là false
`assert_eq!(a, b)`	Panic nếu a != b

Attribute Derive

```
#[derive(Debug, Clone, PartialEq)]
struct Point { x: f64, y: f64 }
// Auto-implements Debug, Clone, PartialEq
```

Attribute Kiểm Thử

```
#[cfg(test)]
mod tests {
    use super::*;
    #[test]
    fn it_works() { assert_eq!(add(2, 2), 4); }
    #[test]
    #[should_panic]
    fn it_panics() { panic!("boom"); }
}
```