Лекція 6

Обробка помилок: Option та Result

Безпечне програмування без null та exceptions

Some • None • Ok • Err • ? оператор

🤖 Приклади: пошук дрона, читання сенсорів, валідація команд

Частина 1: Option<T> — відсутність значення

О. С. Бичков • 2025

План лекції (Частина 1: Option)

1. Проблема null в інших мовах
2. Що таке Option<T>?
3. Some та None
4. Створення Option
5. Pattern matching з Option
6. if let та while let
7. unwrap та expect
8. unwrap_or, unwrap_or_else, unwrap_or_default
9. map та and_then
10. ok_or — конвертація в Result
11. Option у структурах
12. 🤖 Пошук дрона
13. 🤖 Читання сенсорів
14. 🤖 Опціональні поля агента
15. Методи Option (таблиця)

Частина 2: Result<T, E> — обробка помилок, оператор ?, власні помилки

Проблема null в інших мовах

"I call it my billion-dollar mistake" — Tony Hoare (винахідник null)

Null — спеціальне значення, що означає "відсутність значення"

❌ Проблеми з null

- NullPointerException в runtime
- Компілятор не попереджає
- Забуті перевірки
- Неочевидно, де може бути null
- "Defensive programming" скрізь

Java / C# / Python

Rust: немає null! Замість цього — Option<T>, що перевіряється компілятором

```
String name = null;
int len = name.length();
// ❌ NullPointerException
```

Що таке Option<T>?

```
enum Option<T> {
    Some(T),  // Значення присутнє
    None,     // Значення відсутнє
}
```

Some(T)

Є значення типу T

Приклад: Some(42), Some("hello")

None

Значення відсутнє

Аналог null, але type-safe!

Option<T> знаходиться в prelude — не потрібно use!

```rust
// Функція, що може повернути значення або нічого
fn find_drone(drones: &[Drone], id: u32) -> Option<&Drone> {
    for drone in drones {
        if drone.id == id {
            return Some(drone);  // Знайшли!
        }
    }
    None  // Не знайшли
}

// Використання
let drones = vec![/* ... */];

match find_drone(&drones, 42) {
    Some(drone) => println!("Знайдено: {:?}", drone),
    None => println!("Дрон не знайдено"),
}

// Компілятор ЗМУШУЄ обробити обидва варіанти!
```

Неможливо "забути" перевірити — компілятор не дасть!

```
// Option повертають багато методів стандартної бібліотеки
let v = vec![1, 2, 3];
v.first()          // Option<&i32> → Some(&1)
v.get(10)          // Option<&i32> → None
v.iter().find(...)  // Option<&T>
"hello".find('e')  // Option<usize> → Some(1)
HashMap::get(&key) // Option<&V>
```

```
// ❌ Компілятор не знає тип
let x = None;  // Error!
```

```rust
let maybe_number: Option<i32> = Some(42);

// match — повна обробка всіх варіантів
match maybe_number {
    Some(n) => println!("Число: {}", n),
    None => println!("Немає числа"),
}

// Деструктуризація з умовою
match maybe_number {
    Some(n) if n > 0 => println!("Позитивне: {}", n),
    Some(n) => println!("Не позитивне: {}", n),
    None => println!("Відсутнє"),
}

// Вкладені Option
let nested: Option<Option<i32>> = Some(Some(42));
match nested {
    Some(Some(n)) => println!("Значення: {}", n),
    Some(None) => println!("Внутрішній None"),
    None => println!("Зовнішній None"),
}
```

```rust
// if let з else
if let Some(drone) = find_drone(&drones, 42) {
    println!("Знайдено: {:?}", drone);
} else {
    println!("Не знайдено");
}

// while let — цикл поки Some
let mut stack = vec![1, 2, 3];
while let Some(top) = stack.pop() {
    println!("{}", top);  // 3, 2, 1
}
```

```rust
if let Some(x) = option {
    println!("{}", x);
}
```

unwrap та expect — швидкий доступ

```
let x = Some(42);
x.unwrap()  // 42

let y: Option<i32> = None;
y.unwrap()  // ❌ panic!
```

⚠️ Тільки коли 100% впевнені!

expect(msg) — паніка з повідомленням

Краще для дебагу

Коли використовувати unwrap/expect?

- В тестах
- Коли логічно неможливий None (і це доведено)
- В прототипах (потім замінити!)

```
let config = get_config();
let port = config
    .expect("Config must exist");
```

```rust
// Практичний приклад
let battery = drone.battery_level.unwrap_or(100);
let name = config.get("name").unwrap_or_else(|| "default".to_string());
```

```rust
// Lazy — f викликається
// тільки при None
let y: Option<i32> = None;
y.unwrap_or_else(|| {
    compute_default()
})
```

```rust
let maybe_string: Option<String> = Some(String::from("hello"));

// map трансформує Some, ігнорує None
let maybe_len: Option<usize> = maybe_string.map(|s| s.len());
// Some(5)

let none_string: Option<String> = None;
let none_len: Option<usize> = none_string.map(|s| s.len());
// None

// Ланцюжок map
let result: Option<i32> = Some("42")
    .map(|s| s.parse::<i32>())  // Option<Result<i32, _>>
    // Hmm, вкладені типи...

// Для структур
let drone: Option<&Drone> = find_drone(&drones, 42);
let battery: Option<u8> = drone.map(|d| d.battery);
```

```rust
// Проблема з map: вкладений Option
let nested: Option<Option<i32>> = Some("42")
    .map(|s| s.parse::<i32>().ok());  // Option<Option<i32>>

// and_then "розгортає" вкладеність
let flat: Option<i32> = Some("42")
    .and_then(|s| s.parse::<i32>().ok());  // Option<i32>
// Some(42)
```

```rust
// Ланцюжок and_then
fn get_user(id: u32) -> Option<User> { /* ... */ }
fn get_email(user: &User) -> Option<&str> { /* ... */ }

let email: Option<&str> = get_user(42)
    .as_ref()
    .and_then(|u| get_email(u));

// Читається як: "якщо є user, то отримай email"
```

```rust
// Практичний приклад: знайти активного дрона з достатнім зарядом
let suitable_drone: Option<&Drone> = find_drone(&drones, 42)
    .filter(|d| d.is_active)
    .filter(|d| d.battery > 20);
```

```rust
None.or(Some(5))    // Some(5)
Some(3).or(Some(5)) // Some(3)

None.or_else(|| Some(5))
```

```rust
// ok_or — None стає Err
let x: Option<i32> = Some(42);
let result: Result<i32, &str> = x.ok_or("Value not found");
// Ok(42)

let y: Option<i32> = None;
let result: Result<i32, &str> = y.ok_or("Value not found");
// Err("Value not found")
```

```rust
// ok_or_else — lazy версія
let result = option.ok_or_else(|| {
    format!("Drone {} not found", id)
});

// Практичний приклад
fn get_drone_or_error(id: u32) -> Result<&Drone, String> {
    find_drone(&drones, id)
        .ok_or_else(|| format!("Drone {} not found", id))
}
```

```rust
// Опціональні поля — звичайна практика
struct Drone {
    id: u32,
    name: String,
    position: Position,
    battery: u8,
    // Опціональні поля
    assigned_mission: Option<MissionId>,  // Може не мати місії
    last_contact: Option<Timestamp>,      // Може бути новий
    commander_id: Option<u32>,            // Може бути незалежний
}

impl Drone {
    fn is_available(&self) -> bool {
        // Доступний якщо немає місії
        self.assigned_mission.is_none() && self.battery > 20
    }

    fn mission_name(&self) -> Option<&str> {
        // Ланцюжок: місія → назва
        self.assigned_mission
            .as_ref()
            .and_then(|m| missions.get(m))
            .map(|m| m.name.as_str())
    }
}
```

```rust
let x: Option<i32> = Some(42);
let y: Option<i32> = None;

// Прості перевірки
x.is_some()  // true
x.is_none()  // false
y.is_some()  // false
y.is_none()  // true

// is_some_and — перевірка з умовою (Rust 1.70+)
Some(5).is_some_and(|x| x > 3)  // true
Some(2).is_some_and(|x| x > 3)  // false
None::<i32>.is_some_and(|x| x > 3)  // false

// Практичний приклад
if drone.assigned_mission.is_none() {
    assign_new_mission(&mut drone);
}

// Краще:
if drone.is_available() {
    assign_new_mission(&mut drone);
}
```

```rust
// insert — вставити і отримати &mut (Rust 1.53+)
let mut x: Option<i32> = None;
let r = x.insert(42);  // &mut 42
*r += 1;  // x = Some(43)
```

```rust
let mut x = Some(42);
let old = x.replace(100);
// old = Some(42)
```

```rust
let text: Option<String> = Some(String::from("hello"));

// Проблема: map споживає Option
let len = text.map(|s| s.len());  // text moved!

// Рішення: as_ref() — Option<&T>
let text: Option<String> = Some(String::from("hello"));
let len = text.as_ref().map(|s| s.len());
// text все ще доступний!

// as_mut() — Option<&mut T>
let mut text: Option<String> = Some(String::from("hello"));
if let Some(s) = text.as_mut() {
    s.push_str(" world");
}
// text = Some("hello world")

// as_deref() — Option<&T::Target> (для String → &str)
let s: Option<String> = Some("hello".to_string());
let slice: Option<&str> = s.as_deref();  // Option<&str>
```

```rust
impl Swarm {
    /// Знайти дрон за ID
    fn find_by_id(&self, id: u32) -> Option<&Drone> {
        self.drones.iter().find(|d| d.id == id)
    }

    /// Знайти найближчий до позиції
    fn find_nearest(&self, pos: &Position) -> Option<&Drone> {
        self.drones.iter()
            .filter(|d| d.is_active)
            .min_by_key(|d| d.position.distance_to(pos) as i64)
    }

    /// Знайти командира групи
    fn find_commander(&self, group: GroupId) -> Option<&Drone> {
        self.drones.iter()
            .find(|d| d.group_id == group && d.is_commander)
    }

    /// Знайти доступного дрона для місії
    fn find_available(&self) -> Option<&Drone> {
        self.drones.iter()
            .find(|d| d.is_active && d.battery > 30 && d.assigned_mission.is_none())
    }
}
```

```rust
struct Sensors {
    gps: Option<GpsReading>,
    radar: Option<RadarReading>,
    camera: Option<CameraFrame>,
    battery_sensor: Option<BatteryReading>,
}

impl Sensors {
    /// Отримати позицію (якщо GPS працює)
    fn get_position(&self) -> Option<Position> {
        self.gps.as_ref().map(|g| Position {
            x: g.latitude,
            y: g.longitude,
            altitude: g.altitude,
        })
    }

    /// Перевірити всі критичні сенсори
    fn all_critical_ok(&self) -> bool {
        self.gps.is_some() && self.battery_sensor.is_some()
    }

    /// Отримати заряд батареї
    fn battery_level(&self) -> Option<u8> {
        self.battery_sensor.as_ref().map(|b| b.percentage)
    }
}
```

```rust
struct Agent {
    id: u32,
    role: AgentRole,
    position: Position,

    // Опціональні компоненти
    communication: Option<CommModule>,
    weapons: Option<WeaponsSystem>,
    cargo: Option<CargoHold>,
    sensors: Option<SensorSuite>,
}

impl Agent {
    /// Може спілкуватись?
    fn can_communicate(&self) -> bool {
        self.communication.as_ref()
            .map(|c| c.is_operational())
            .unwrap_or(false)
    }

    /// Вантажопідйомність
    fn cargo_capacity(&self) -> u32 {
        self.cargo.as_ref()
            .map(|c| c.max_weight)
            .unwrap_or(0)
    }

    /// Озброєний?
    fn is_armed(&self) -> bool {
        self.weapons.is_some()
```

```
// zip у MAC: об'єднати дані з різних джерел
let coords: Option<(f64, f64)> = gps.get_lat().zip(gps.get_lon());
```

```
let a = Some(1);
let b = Some("hi");
a.zip(b)  // Some((1, "hi"))
```

Основні методи Option (1/2)

Метод	Опис	Результат
is_some() / is_none()	Перевірка наявності	bool
unwrap()	Витягти або panic	T
expect(msg)	Витягти або panic з msg	T
unwrap_or(def)	Витягти або default	T
unwrap_or_else(f)	Витягти або обчислити	T
unwrap_or_default()	Витягти або Default::default	T
map(f)	Трансформувати Some	Option<U>
and_then(f)	Ланцюжок (flatMap)	Option<U>

Основні методи Option (2/2)

Метод	Опис	Результат
filter(p)	Залишити якщо predicate true	Option<T>
ok_or(err)	Конвертувати в Result	Result<T, E>
as_ref()	Option<&T>	Option<&T>
as_mut()	Option<&mut T>	Option<&mut T>
take()	Забрати значення, залишити None	Option<T>
replace(val)	Замінити значення	Option<T>
zip(other)	Об'єднати два Option	Option<(T, U)>
flatten()	Option<Option<T>> → Option<T>	Option<T>

Підсумок: Option<T>

Option<T> замінює null:
• Some(value) — значення є
• None — значення відсутнє
• Компілятор змушує обробити обидва варіанти

Основні патерни:
• match / if let — деструктуризація
• map / and_then — ланцюжки трансформацій
• unwrap_or — значення за замовчуванням
• ok_or — конвертація в Result

🤖 MAC застосування:
• Пошук агентів: Option<&Drone>
• Опціональні компоненти
• Читання сенсорів

→ Частина 2: Result<T, E> — обробка помилок

Лекція 6 (продовження)

Обробка помилок: Result<T, E>

Безпечна обробка помилок без exceptions

Ok • Err • ? оператор • власні помилки

🤖 Приклади: виконання команд, валідація, обробка місій

Частина 2: Result та оператор ?

О. С. Бичков • 2025

План лекції (Частина 2: Result)

1. Проблема exceptions
2. Що таке Result<T, E>?
3. Ok та Err
4. Pattern matching з Result
5. Методи Result
6. Оператор ?
7. Проброс помилок
8. Конвертація помилок
9. Власні типи помилок
10. thiserror та anyhow
11. 🤖 Виконання команд
12. 🤖 Валідація даних
13. 🤖 Обробка місій
14. 🤖 Комунікація агентів
15. Option vs Result
16. Практичні поради

Проблема exceptions в інших мовах

❌ Проблеми з exceptions

- Неочевидно, де може виникнути
- Прихований control flow
- Забуті catch блоки
- Performance overhead
- "Exception safety" складно

Java / Python

Rust: немає exceptions! Замість цього — Result<T, E>
- Помилка — це значення, що повертається
- Компілятор змушує обробити

```
try {
    file = open(path);
    data = file.read();
```

```
enum Result<T, E> {
    Ok(T),   // Успіх — містить результат типу T
    Err(E),  // Помилка — містить помилку типу E
}
```

Ok(T)

Операція успішна
Містить результат типу T

Err(E)

Операція провалилась
Містить опис помилки типу E

Result знаходиться в prelude — не потрібно use!

```rust
use std::fs::File;
use std::io::Read;

// Функція, що може провалитись
fn read_file(path: &str) -> Result<String, std::io::Error> {
    let mut file = File::open(path)?;  // ? — проброс помилки
    let mut contents = String::new();
    file.read_to_string(&mut contents)?;
    Ok(contents)
}

// Використання
match read_file("config.txt") {
    Ok(contents) => println!("Вміст: {}", contents),
    Err(error) => println!("Помилка: {}", error),
}

// Компілятор ЗМУШУЄ обробити обидва варіанти!
```

```rust
fn divide(a: i32, b: i32) -> Result<i32, String> {
    if b == 0 {
        Err(String::from("Division by zero"))
    } else {
        Ok(a / b)
    }
}

// match — повна обробка
match divide(10, 2) {
    Ok(result) => println!("Результат: {}", result),
    Err(e) => println!("Помилка: {}", e),
}

// if let — тільки один варіант
if let Ok(result) = divide(10, 2) {
    println!("Результат: {}", result);
}

// let else (Rust 1.65+) — обробка помилки
let Ok(result) = divide(10, 2) else {
    println!("Помилка!");
    return;
};
```

```rust
// Безпечні альтернативи
result.unwrap_or(default_value)      // Ok → value, Err → default
result.unwrap_or_else(|e| handle(e)) // Ok → value, Err → compute
result.unwrap_or_default()           // Ok → value, Err → Default::default()

// ok() — Result → Option (відкидає помилку)
let maybe: Option<i32> = result.ok();  // Err → None
```

```rust
let config = load_config()
    .expect("Failed to load config");
```

```rust
// and_then — ланцюжок операцій (як flatMap)
fn double(x: i32) -> Result<i32, String> { Ok(x * 2) }
fn stringify(x: i32) -> Result<String, String> { Ok(x.to_string()) }

let result: Result<String, String> = Ok(5)
    .and_then(double)
    .and_then(stringify);
// Ok("10")
```

```rust
let x: Result<i32, i32> = Err(5);
let y = x.map_err(|e| e * 2);
// Err(10)
```

Оператор ? — елегантний проброс помилок

```rust
fn process() -> Result<i32, Error> {
    let x = match step1() {
        Ok(v) => v,
        Err(e) => return Err(e),
    };
    let y = match step2(x) {
        Ok(v) => v,
        Err(e) => return Err(e),
    };
    Ok(y)
}
```

3 ? (concise)

? працює тільки у функціях, що повертають Result (або Option)

```rust
fn process() -> Result<i32, Error> {
    let x = step1()?;
    let y = step2(x)?;
    Ok(y)
}
```

```rust
// Стандартний main
fn main() {
    // ? не можна тут, бо main повертає ()
}

// main з Result — можна використовувати ?
fn main() -> Result<(), Box<dyn std::error::Error>> {
    let config = load_config()?;
    let data = read_file(&config.path)?;
    process_data(&data)?;

    Ok(())
}
// При помилці програма завершиться з повідомленням

// Альтернатива: anyhow::Result
use anyhow::Result;

fn main() -> Result<()> {
    // ...
    Ok(())
}
```

```rust
use std::io;
use std::num::ParseIntError;

// Власний тип помилки
#[derive(Debug)]
enum MyError {
    Io(io::Error),
    Parse(ParseIntError),
}

// Імплементуємо From для автоматичної конвертації
impl From<io::Error> for MyError {
    fn from(e: io::Error) -> Self {
        MyError::Io(e)
    }
}

impl From<ParseIntError> for MyError {
    fn from(e: ParseIntError) -> Self {
        MyError::Parse(e)
    }
}

// Тепер ? автоматично конвертує помилки!
fn read_number(path: &str) -> Result<i32, MyError> {
    let contents = std::fs::read_to_string(path)?;  // io::Error → MyError
    let num = contents.trim().parse()?;             // ParseIntError → MyError
    Ok(num)
}
```

```rust
use std::fmt;
use std::error::Error;

#[derive(Debug)]
enum DroneError {
    NotFound(u32),
    LowBattery { id: u32, level: u8 },
    CommunicationFailed(String),
    InvalidCommand,
}

// Display — для користувача
impl fmt::Display for DroneError {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        match self {
            DroneError::NotFound(id) => write!(f, "Drone {} not found", id),
            DroneError::LowBattery { id, level } =>
                write!(f, "Drone {} has low battery: {}%", id, level),
            DroneError::CommunicationFailed(msg) =>
                write!(f, "Communication failed: {}", msg),
            DroneError::InvalidCommand => write!(f, "Invalid command"),
        }
    }
}

// Error trait
impl Error for DroneError {}
```

```rust
use thiserror::Error;

#[derive(Error, Debug)]
enum DroneError {
    #[error("Drone {0} not found")]
    NotFound(u32),

    #[error("Drone {id} has low battery: {level}%")]
    LowBattery { id: u32, level: u8 },

    #[error("Communication failed: {0}")]
    CommunicationFailed(String),

    #[error("Invalid command")]
    InvalidCommand,

    // Автоматична конвертація з std::io::Error
    #[error("IO error: {0}")]
    Io(#[from] std::io::Error),
}

// Використання
fn send_command(drone_id: u32) -> Result<(), DroneError> {
    let drone = find_drone(drone_id)
        .ok_or(DroneError::NotFound(drone_id))?;
    // ...
    Ok(())
}
```

```rust
use anyhow::{Result, Context, anyhow, bail};

// anyhow::Result<T> = Result<T, anyhow::Error>
fn load_mission(path: &str) -> Result<Mission> {
    let contents = std::fs::read_to_string(path)
        .context("Failed to read mission file")?;  // Додає контекст

    let mission: Mission = serde_json::from_str(&contents)
        .context("Failed to parse mission JSON")?;

    if mission.waypoints.is_empty() {
        bail!("Mission has no waypoints");  // Повертає Err
    }

    if mission.priority > 10 {
        return Err(anyhow!("Invalid priority: {}", mission.priority));
    }

    Ok(mission)
}

// anyhow ідеальний для applications (не libraries)
```

thiserror — для бібліотек (структуровані помилки), anyhow — для застосунків

```rust
#[derive(Error, Debug)]
enum CommandError {
    #[error("Drone {0} not found")]
    DroneNotFound(u32),
    #[error("Drone {0} is offline")]
    DroneOffline(u32),
    #[error("Invalid command for drone state: {0}")]
    InvalidState(String),
    #[error("Command timeout after {0}ms")]
    Timeout(u64),
}

impl Swarm {
    fn execute_command(&mut self, drone_id: u32, cmd: Command) -> Result<(), CommandError> {
        let drone = self.drones.get_mut(&drone_id)
            .ok_or(CommandError::DroneNotFound(drone_id))?;

        if !drone.is_online {
            return Err(CommandError::DroneOffline(drone_id));
        }

        if !drone.can_execute(&cmd) {
            return Err(CommandError::InvalidState(format!("{:?}", drone.state)));
        }

        drone.execute(cmd);
        Ok(())
    }
}
```

```rust
#[derive(Error, Debug)]
enum ValidationError {
    #[error("Mission name cannot be empty")]
    EmptyName,
    #[error("At least one waypoint required")]
    NoWaypoints,
    #[error("Waypoint {0} is out of bounds")]
    OutOfBounds(usize),
    #[error("Priority must be 1-10, got {0}")]
    InvalidPriority(u8),
}

fn validate_mission(mission: &Mission) -> Result<(), ValidationError> {
    if mission.name.is_empty() {
        return Err(ValidationError::EmptyName);
    }

    if mission.waypoints.is_empty() {
        return Err(ValidationError::NoWaypoints);
    }

    for (i, wp) in mission.waypoints.iter().enumerate() {
        if !wp.is_valid() {
            return Err(ValidationError::OutOfBounds(i));
        }
    }

    if mission.priority < 1 || mission.priority > 10 {
        return Err(ValidationError::InvalidPriority(mission.priority));
    }
```

```rust
fn process_mission(swarm: &mut Swarm, mission: Mission) -> Result<MissionResult, MissionError> {
    // 1. Валідація
    validate_mission(&mission)
        .map_err(MissionError::Validation)?;

    // 2. Знайти доступних дронів
    let available = swarm.find_available_drones(mission.required_count)
        .ok_or(MissionError::InsufficientDrones)?;

    // 3. Призначити місію
    for drone_id in &available {
        swarm.assign_mission(*drone_id, mission.id)
            .map_err(|e| MissionError::Assignment(*drone_id, e))?;
    }

    // 4. Запустити виконання
    let result = swarm.execute_mission(mission.id)
        .map_err(MissionError::Execution)?;

    // 5. Зібрати результати
    let report = swarm.collect_results(mission.id)?;

    Ok(MissionResult { mission_id: mission.id, drones: available, report })
}
```

```rust
#[derive(Error, Debug)]
enum CommError {
    #[error("Connection to {0} failed")]
    ConnectionFailed(u32),
    #[error("Timeout waiting for response from {0}")]
    Timeout(u32),
    #[error("Invalid message format: {0}")]
    InvalidFormat(String),
    #[error("Encryption error: {0}")]
    Encryption(String),
}

impl Drone {
    fn send_message(&self, target: u32, msg: Message) -> Result<Response, CommError> {
        // Встановити з'єднання
        let conn = self.comm.connect(target)
            .map_err(|_| CommError::ConnectionFailed(target))?;

        // Зашифрувати
        let encrypted = self.crypto.encrypt(&msg)
            .map_err(|e| CommError::Encryption(e.to_string()))?;

        // Відправити і чекати відповідь
        let response = conn.send_and_wait(encrypted, Duration::from_secs(5))
            .map_err(|_| CommError::Timeout(target))?;

        Ok(response)
    }
}
```

```rust
impl Drone {
    /// Отримати позицію з fallback
    fn get_position(&self) -> Position {
        // Спробувати GPS
        if let Ok(pos) = self.gps.read() {
            return pos;
        }

        // Fallback на інерційну навігацію
        if let Ok(pos) = self.ins.estimate() {
            return pos;
        }

        // Остання відома позиція
        self.last_known_position
    }

    /// Виконати з retry
    fn execute_with_retry<F, T, E>(&self, f: F, max_retries: u32) -> Result<T, E>
    where F: Fn() -> Result<T, E>
    {
        let mut last_err = None;
        for _ in 0..max_retries {
            match f() {
                Ok(result) => return Ok(result),
                Err(e) => last_err = Some(e),
            }
        }
        Err(last_err.unwrap())
    }
```

Option vs Result: коли що?

Option<T>

Коли відсутність — НЕ помилка:

- Пошук: може не знайти
- Опціональні поля структур
- Перший/останній елемент
- Значення за замовчуванням

Приклад:
- find_drone() → Option<&Drone>
- HashMap::get() → Option<&V>
- Vec::first() → Option<&T>

Result<T, E>

Коли є конкретна ПОМИЛКА:

- I/O операції
- Парсинг даних
- Валідація
- Мережеві запити

Приклад:
- File::open() → Result<File, Error>
- str::parse() → Result<T, ParseError>
- send_command() → Result<(), CmdError>

Правило: Option коли "може бути", Result коли "може провалитись"

Основні методи Result

Метод	Опис	Результат
is_ok() / is_err()	Перевірка	bool
ok() / err()	Конвертація в Option	Option<T> / Option<E>
unwrap() / expect()	Витягти або panic	T
unwrap_or(def)	Витягти або default	T
map(f)	Трансформувати Ok	Result<U, E>
map_err(f)	Трансформувати Err	Result<T, F>
and_then(f)	Ланцюжок операцій	Result<U, E>
?	Проброс помилки	T або return Err

Практичні поради

✓ Завжди обробляйте помилки явно
✓ Використовуйте ? для проброса
✓ Створюйте власні типи помилок для бібліотек
✓ thiserror для бібліотек, anyhow для apps
✓ Документуйте можливі помилки

❌ Уникайте unwrap() у production коді
❌ Не ігноруйте помилки через .ok()
❌ Не перетворюйте все на String

🤖 Для MAC:
• Структуровані помилки для діагностики
• Graceful degradation де можливо
• Retry з exponential backoff
• Логування всіх помилок

Підсумок лекції

Option<T> — відсутність значення:
- Some(value) / None
- Заміна null
- map, and_then, unwrap_or

Result<T, E> — помилки:
- Ok(value) / Err(error)
- Заміна exceptions
- Оператор ? для проброса

Власні помилки:
- enum з варіантами помилок
- thiserror / anyhow

🤖 MAC: валідація команд, обробка місій, комунікація
→ Наступна лекція: Traits — поліморфізм у Rust

Завдання для самостійної роботи

1. Базове: Створіть функцію divide(a, b) → Result<f64, String>
   Обробіть ділення на нуль.

2. Парсинг: Напишіть parse_drone_status(s: &str) → Result<DroneStatus, ParseError>
   Парсіть формат: "id:123,battery:85,state:active"

3. Валідатор місій: Реалізуйте повну валідацію Mission:
   - Перевірка всіх полів
   - Власний enum ValidationError
   - Колекція помилок (не тільки перша)

4. Команди: Система виконання команд:
   - CommandResult з Ok/Err
   - Retry логіка
   - Timeout handling
   - Logging помилок

Дякую за увагу!

Option<T> • Result<T, E> • Оператор ?

Питання?