

# Основи програмування на Rust

професор кафедри програмних систем і технологій  
Київського національного університету імені Тараса Шевченка

**Бичков Олексій Сергійович**

# Зміст

Історія. Що таке Rust.

Встановлення. Інструменти. Термінал. Cargo.

Перша програма. Порівняння з C. Розбір програми.

Змінні. Іммутабельність. Імена змінних.

Типи даних. Цілі. Дробові. bool. char. Рядки.

Операції. Арифметичні. Порівняння. Логічні.

Оператори керування. if/else. Цикли loop, while, for.

Функції. Приклади програм. Контрольні питання.

# Цілі

## Дізнатись:

що таке Rust і чим він відрізняється від C/C++  
як встановити Rust та налаштувати середовище

## Написати першу програму

## Навчитись писати прості програми

(використовуючи змінні, типи, операції, умови, цикли, функції)

# Що таке Rust?

Вкратце: Rust — мова системного програмування, яка поєднує швидкість C/C++ з безпекою сучасних мов.

Rust може працювати на рівні операційної системи, писати драйвери, ядра ОС. А може використовуватись для веб-серверів, ігор, блокчайнів.

Щоб програма на Rust працювала, її треба скомпілювати — перетворити текст на машинний код.

## Головна особливість Rust:

Компілятор перевіряє безпеку роботи з пам'яттю ще до запуску програми. Немає segmentation fault, buffer overflow, use-after-free.

# Трохи про Rust

Мова Rust була розроблена Грейдоном Хоаром як особистий проект у 2006 році. Mozilla офіційно підтримала проект у 2010 році. Перша стабільна версія Rust 1.0 вийшла у 2015 році.

Існують три гілки Rust:

Stable — стабільна, рекомендована для продакшену

Beta — тестова версія наступного релізу

Nightly — нічна збірка з експериментальними можливостями

Rust — найулюбленіша мова програмістів за опитуванням Stack Overflow 8 років поспіль.

Інструменти: rustc (компілятор), cargo (збірка), rustup (версії), rust-analyzer (IDE)

# Хто використовує Rust?

Mozilla Firefox

WebRender

Discord

Сервіси

Google Android

Bluetooth

Cloudflare

Edge

Microsoft

Windows kernel

Linux Kernel

Драйвери

Amazon AWS

Firecracker

Dropbox

Sync

70% вразливостей Microsoft — помилки пам'яті. Rust усуває їх на етапі компіляції.

# Що таке термінал?

Термінал (командний рядок) — текстовий інтерфейс для керування комп'ютером. Замість кліків мишкою ви вводите команди текстом.

Як відкрити:

Windows: Win+R → cmd → Enter або PowerShell

macOS: Cmd+Space → Terminal

Linux: Ctrl+Alt+T

```
C:\Users\Student> rustc --version  
rustc 1.75.0 (82e1608df 2023-12-21)
```

↑ ви вводите команду      ↑ комп'ютер відповідає

# Встановлення Rust

Для Windows:

1. Відкрийте <https://rustup.rs>
2. Натисніть "Download rustup-init.exe"
3. Запустіть, натисніть Enter
4. Дочекайтесь (5-15 хв)
5. Перезапустіть термінал!

```
# Для macOS/Linux:  
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

 Після встановлення обов'язково перезапустіть термінал!

# Перевірка встановлення

Відкрийте НОВИЙ термінал і введіть:

```
C:\> rustc --version  
rustc 1.75.0
```

```
C:\> cargo --version  
cargo 1.75.0
```

```
C:\> rustup --version  
rustup 1.26.0
```

rustc — компілятор, cargo — менеджер проектів, rustup — менеджер версій

**Якщо бачите версії — все встановлено правильно!**

# Інструмент cargo

cargo — головний інструмент Rust-розробника:

```
cargo new my_project      # Створити проект
cd my_project             # Перейти в папку (ОБОВ'ЯЗКОВО!)

cargo build               # Скомпілювати
cargo run                  # Скомпілювати І запустити
cargo check                # Швидка перевірка
cargo test                  # Запустити тести
cargo build --release     # Оптимізована збірка
```

# Структура проекту

```
my_project/
├── Cargo.toml
├── src/
│   └── main.rs
└── target/
    └── debug/
        └── my_project.exe
```

Cargo.toml — налаштування: назва, залежності

src/main.rs — ВАШ КОД ТУТ!

target/ — скомпільовані файли (.exe)

# Перша програма: C vs Rust

C

```
#include <stdio.h>

int main()
{
    printf("Hello!\\n");
    return 0;
}
```

Rust

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

Відмінності: немає #include • fn замість int • println! замість printf • немає return 0

# Перша програма: fn main()

```
fn main() {
```

Тіло програми завжди міститься у функції з назвою `main`.

`fn` — ключове слово для оголошення функції (від "function").

Порожні дужки `()` означають, що функція не приймає параметрів.

Фігурна дужка `{` починає тіло функції.

У Rust немає `int` перед `main()`. На відміну від Сі, де `main` повертає `int`, у Rust функція `main` за замовчуванням нічого не повертає.

# Перша програма: `println!`

```
println!("Hello world!");
```

`println!` — макрос для виведення тексту на екран.

Знак оклику ! означає, що це макрос, а не звичайна функція.

Текст береться в подвійні лапки "...".

`\n` — перехід на новий рядок. Але `println!` автоматично додає новий рядок.

Крапка з комою ; — обов'язкова в кінці!

Хороший тон: все в фігурних дужках зсувати вправо (відступи).

# Спецсимволи

`\n` — новий рядок

`\t` — табуляція

`\r` — повернення каретки

`\\"` — зворотна коса

`\"` — подвійна лапка

`\'` — апостроф

`\0` — нульовий символ

`\x41` — hex код (A)

```
println! ("Рядок 1\nРядок 2"); // Виведе два рядки
```

# Форматування виводу

У Rust замість %d, %f використовуються {}:

```
Ci  
printf("x = %d", x);
```

```
Rust  
println!("x = {}", x);
```

```
let a = 5;  
let b = 10;  
println!("{} + {} = {}", a, b, a + b); // 5 + 10 = 15  
println!("point = {:?}", (3, 4)); // point = (3, 4)
```

# Що таке змінна?

Змінна — це іменована область пам'яті, де зберігається значення.

Можна уявити як підписану коробку з даними всередині.

```
let x = 5;
```

let — ключове слово для оголошення змінної

x — ім'я змінної (ідентифікатор)

= — оператор присвоєння

5 — значення

; — кінець інструкції

У Rust змінна — це ВЛАСНИК даних, а не посилання (як у Python).

# Змінні: let та let mut

**ВАЖЛИВО: В Rust змінні за замовчуванням НЕЗМІННІ!**

✗ Без **mut** — помилка:

```
let x = 5;  
x = 10; // ✗ ПОМИЛКА!  
  
// error: cannot assign  
// twice to immutable
```

✓ 3 **mut** — OK:

```
let mut x = 5;  
x = 10; // ✓ OK!  
  
// mut = mutable  
// = змінний
```

Навіщо? Запобігає випадковим змінам • Код легше читати • Краща оптимізація

# Імена змінних

- Починаються з букви або \_
- Містять букви, цифри, \_
- Регистр важливий (x ≠ X)
- Не можуть бути зарезервованими словами

```
// Правильні
let age = 25;
let user_name = "Олексій";

// Неправильні
// let 2point = 5;    // цифра
// let fn = 5;        // зарезервоване
```

Хороший тон: snake\_case (user\_name), зрозумілі імена (age, не a)

# Типи змінних

Тип	Розмір	Діапазон	Приклад
i8	1 байт	-128...127	let a: i8 = -50;
i32	4 байти	±2 млрд	let a = 42; // за замовч.
i64	8 байт	$\pm 9 \cdot 10^8$	let a: i64 = 9999999999;
u8	1 байт	0...255	let a: u8 = 255;
u32	4 байти	0...4 млрд	let a: u32 = 100;
f32	4 байти	$\pm 3.4 \cdot 10^{-8}$	let a: f32 = 3.14;
f64	8 байт	$\pm 1.7 \cdot 10^{-108}$	let a = 3.14; // за замовч.
bool	1 байт	true, false	let a = true;
char	4 байти	Unicode	let a = 'Я';

i = signed, u = unsigned, f = float

# Цілі числа

```
let a = 42;
let million = 1_000_000;    // підкреслення

let hex = 0xFF;            // = 255
let octal = 0o77;          // = 63 (0o!)
let binary = 0b1010;        // = 10

let a = 42i64;             // суфікс типу
let b = 100u8;
```

Відмінність від Сі: вісімкові 0o77 (не 077), двійкові 0b1010, підкреслення 1\_000\_000

# Дробові числа

```
let pi = 3.14159;           // f64 за замовч.  
let e: f32 = 2.718;        // явно f32  
let avogadro = 6.022e23; // експонента
```

 Цілі та дробові НЕ МОЖНА змішувати!

```
let x: i32 = 5;  
let y: f64 = 2.0;  
let z = x + y;      // ✗ ПОМИЛКА!  
let z = x as f64 + y; // ✓ Явне перетворення
```

# Логічний тип bool

```
let is_active: bool = true;
let is_finished = false;

let is_less = 5 < 10;      // true

if is_active {
    println!("Активний");
}
```

## Відмінність від Сі:

- У Сі: 0 = false, інше = true. Можна if (x)
- У Rust: тільки true/false. if x — помилка! Треба if x != 0

# Символьний тип `char`

`char` — один символ Unicode. ОДИНАРНІ лапки!

```
let letter: char = 'A';
let ukrainian = 'Я';
let emoji = '😎';           // Емодзі теж!
let newline = '\n';
```

Cи `char` = 1 байт (ASCII)  
Rust `char` = 4 байти (UTF-32)

⚠ 'A' (char) ≠ "A" (рядок)

# Рядки: &str та String

&str — фіксований  
let s = "Hello";

String — динамічний  
let s = String::from("Hi");

```
// Конкатенація
let hello = String::from("Hello, ");
let world = "World!";
let msg = hello + world; // "Hello, World!"

// format!
let msg = format!("{} {}", "Hello", "World");
```

# Операції

## Арифметичні

+ - \* / %

## Порівняння

== != < > <= >=

## Логічні

&& || !

## Присвоєння

= += -= \*= /= %=

## Бітові

& | ^ << >>

⚠ У Rust НЕМАЄ ++ та -- ! Використовуйте: x += 1

# Арифметичні операції

```
let a = 10;
let b = 3;

println!("{} + {} = {}", a, b, a + b); // 13
println!("{} - {} = {}", a, b, a - b); // 7
println!("{} * {} = {}", a, b, a * b); // 30
println!("{} / {} = {}", a, b, a / b); // 3 (ціле!)
println!("{} % {} = {}", a, b, a % b); // 1 (остача)

let mut x = 10;
x += 5; // 15
x -= 3; // 12
x *= 2; // 24
```

# Операції порівняння

```
let a = 5;
let b = 10;

println!("{} == {} → {}, a, b, a == b); // false
println!("{} != {} → {}, a, b, a != b); // true
println!("{} < {} → {}, a, b, a < b); // true
println!("{} > {} → {}, a, b, a > b); // false
println!("{} <= {} → {}, a, a, a <= a); // true
println!("{} >= {} → {}, a, b, a >= b); // false

if a < b {
    println!("a менше за b");
}
```

# Логічні операції

**&& — І**  
true && true → true  
інше → false

**|| — АБО**  
false || false → false  
інше → true

**! — НЕ**  
!true → false  
!false → true

```
let age = 25;
let has_license = true;
let can_drive = age >= 18 && has_license; // true

let is_hungry = true;
let needs_break = false || is_hungry; // true
```

# Оператор if

```
Cи  
if (x > 5) {  
    printf("Більше");  
} else {  
    printf("Менше");  
}
```

```
Rust  
if x > 5 {  
    println!("Більше");  
} else {  
    println!("Менше");  
}
```

## Відмінності від Сі:

- БЕЗ круглих дужок! if x > 5
- {} ОБОВ'ЯЗКОВІ навіть для одного рядка
- Умова ПОВИННА бути bool! if x — помилка

# if як вираз

if — це вираз, який повертає значення! Замінює тернарний ?:

```
Cи: int s = x > 0 ? 1 : 0;
```

```
Rust: let s = if x > 0 { 1 } else { 0 };
```

```
let score = 75;
let grade = if score >= 90 {
    "Відмінно"
} else if score >= 70 {
    "Добре"      // ← score = 75
} else {
    "Задовільно"
};
println!("Оцінка: {}", grade); // Добре
```

# Цикл `loop`

`loop` — нескінчений цикл. Вихід через `break`.

```
Простий:  
let mut c = 0;  
loop {  
    c += 1;  
    if c >= 5 { break; }  
}  
// 1 2 3 4 5
```

Повертає значення:

```
let mut c = 0;  
let result = loop {  
    c += 1;  
    if c == 10 {  
        break c * 2;  
    }  
}; // result = 20
```

**`break` може повертати значення!**

# Цикл while

```
// Зворотний відлік
let mut n = 5;
while n > 0 {
    println!("{}...", n);
    n -= 1;
}
println!("Старт!");

// 5...
// 4...
// 3...
// 2...
// 1...
// Старт!
```

# Цикл for

```
// 0..5 – НЕ включає 5
for i in 0..5 { print!("{} ", i); } // 0 1 2 3 4

// 1..=5 – ВКЛЮЧАЄ 5
for i in 1..=5 { print!("{} ", i); } // 1 2 3 4 5

// Зворотній порядок
for i in (1..=5).rev() { print!("{} ", i); } // 5 4 3 2 1

// По масиву
let fruits = ["яблуко", "банан"];
for fruit in fruits {
    println!("{}", fruit);
}
```

# break та continue

```
break – вихід:  
for i in 0..10 {  
    if i == 5 {  
        println!("Стоп!");  
        break;  
    }  
    println!("{}", i);  
}  
// 0 1 2 3 4 Стоп!
```

```
continue – пропустити:  
for i in 0..5 {  
    if i == 2 {  
        continue;  
    }  
    println!("{}", i);  
}  
// 0 1 3 4  
// (2 пропущено)
```

# ФУНКЦІЇ

fn оголошує функцію. Типи параметрів ОБОВ'ЯЗКОВІ.

```
fn greet() {
    println!("Привіт!");
}

fn greet_person(name: &str) {
    println!("Привіт, {}!", name);
}

fn add(a: i32, b: i32) -> i32 {
    a + b // БЕЗ ; = повернення!
}

fn main() {
    greet();
    greet_person("Rust");
    let sum = add(5, 3); // 8
}
```

# Приклад: площа круга

```
fn calculate_area(r: f64) -> f64 {
    3.14159265 * r * r
}

fn main() {
    let radius = 5.0;
    let area = calculate_area(radius);

    println!("Радіус: {}", radius);
    println!("Площа: {:.2}", area);
}

// Радіус: 5
// Площа: 78.54
```

# Приклад: конвертер температури

```
fn celsius_to_fahrenheit(c: f64) -> f64 {
    c * 9.0 / 5.0 + 32.0
}

fn main() {
    let temps = [0.0, 20.0, 36.6, 100.0];

    for c in temps {
        let f = celsius_to_fahrenheit(c);
        println!(" {:.1}°C = {:.1}°F", c, f);
    }
}
// 0.0°C = 32.0°F
// 20.0°C = 68.0°F
```

# Завдання

1. Вивести `println!` візерунок:

```
* * ****  
* * *  
**** ***  
* * *  
* * ****
```

Спробуйте одним `println!`

2. Перевести Фаренгейта в Цельсія:  $C = (F - 32) \times 5/9$

3. Написати калькулятор: add, subtract, multiply, divide

# Підсумок

## Інструменти

rustup, cargo new/build/run

## Програма

fn main() { }, println!()

## Змінні

let / let mut

## Типи

i32, f64, bool, char, &str

## Операції

+ - \* / %, == !=, && ||

## Керування

if/else, loop, while, for

## Наступна лекція:

Ownership — система володіння пам'яттю, серце безпеки Rust 