

Системне Програмування

З використанням мови програмування **Rust.**
Fundamentals. Flow Control

Flow Control

Структурування коду в будь-якій мові програмування — це основа підтримки його читабельності, масштабованості та логічної послідовності. Rust надає розширені можливості для роботи з умовами, циклами та контрольними конструкціями, що допомагає розробникам будувати ефективні та безпечні програми.

Basic Flow

```
fn f(a: u32) -> f32
fn g(a: f32) -> bool
fn h(a: bool) -> String

fn app(a: u32) -> String {

    let b: f32 = f(a);
    let c: bool = g(b);
    let d: String = h(c);
    d
}
```

Основні конструкції контролю виконання

Rust підтримує різні конструкції для управління потоком виконання програми, серед яких:

- **if/else**
- **match**
- цикли (**for, while, loop**)
- управляючі конструкції **break/continue**

Ці конструкції допомагають реалізувати логіку прийняття рішень та організовувати повторювані процеси в коді.

Базова конструкція if ... else

```
if n < 0 {  
    println!("{}", is negative", n);  
} else {  
    println!("{}", is non-negative", n);  
}
```

Конструкція if ... else комбінована

```
if n < 0 {  
    println!("{}", is negative", n);  
} else if n > 0 {  
    println!("{}", is positive", n);  
} else {  
    println!("{}", is zero", n);  
}
```


Конструкція if може повертати значення

```
let s : &str = if n < 0 {  
    "negative"  
} else if n > 0 {  
    "positive"  
} else {  
    "zero"  
};
```

Умови в середині if можна комбінувати.

Булева алгебра

not, !

a	!a
true	false
false	true

or, a || b

a	b	a b
false	false	false
false	true	true
true	false	true
true	true	true

and, a && b

a	b	a&& b
false	false	false
false	true	false
true	false	false
true	true	true

xor, a ^ b

a	b	a^b
false	false	false
false	true	true
true	false	true
true	true	false

Деякі правила ідентичності та спрощення

- $(a \parallel b) \&\& c == a \&\& c \parallel b \&\& c$
- $a \parallel !a = \text{true}$
- $a \&\& !a = \text{false}$
- $\text{true} \parallel a = \text{true}$
- $a \&\& \text{true} = a$

Повторення певну кількість разів. Не включаючи

```
for n : i32 in 0..  
    println!("{}", n);  
}
```

Включаючи

```
for n : i32 in 0..=5 {  
    println!("{}", n);  
}
```


Повторення

```
for n : i32 in 0.. {  
  println!("{}", n);  
}
```

Повторення по умові

```
let mut n: i32 = 1;
while n < 20 {
    if n % 15 == 0 {
        println!("{}", fizzbuzz, n);
    } else if n % 3 == 0 {
        println!("{}", fizz, n);
    } else if n % 5 == 0 {
        println!("{}", buzz, n);
    } else {
        println!("{}", n);
    }
    n += 1;
}
```


Повторення без умови

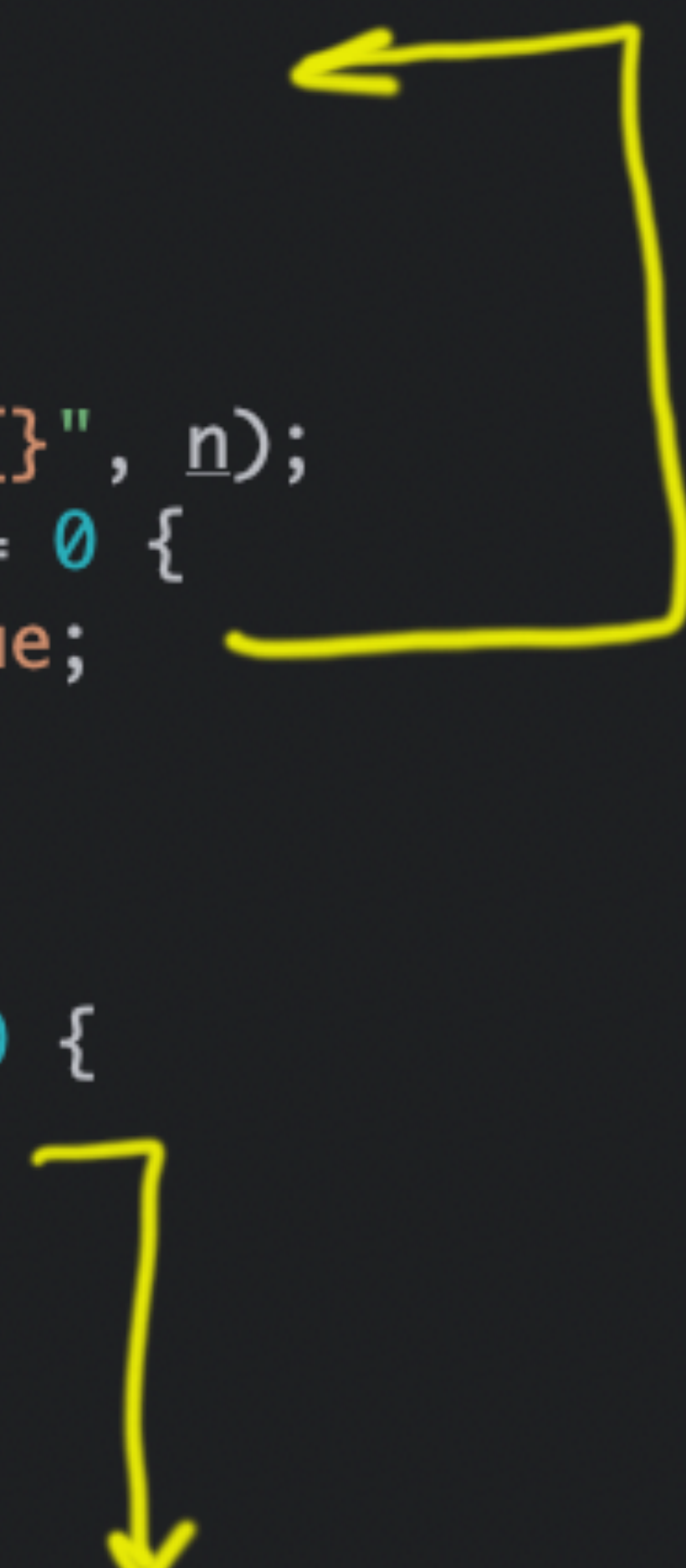
```
let mut n: u32 = 0u32;  
loop {  
    n += 1;  
    println!("{}", n);  
}
```

Повторення без умови, але з виходом

```
let mut n: u32 = 0u32;  
loop {  
    n += 1;  
    println!("{}", n);  
    if n == 10 {  
        break;  
    }  
}
```


Контроль в середині циклу

```
let mut n: u32 = 0u32;  
loop {  
    // ...  
    // ...  
    n += 1;  
    println!("{}", n);  
    if n % 2 == 0 {  
        continue;  
    }  
    // ...  
    // ...  
    if n == 100 {  
        break;  
    }  
    // ...  
    // ...  
}
```



Контроль в середині циклу. `continue`

```
let mut n: i32 = 0;
for i: i32 in 0..<100 {
    n = n + 1; // 100 times
    if i % 2 == 0 {
        continue;
    }
    n = n + 1; // 50 times
}
assert_eq!(150, n);
```


Контроль в середине цикла

```
let mut n: u32 = 0u32;

'a: loop {
    n += 1;

    println!("b");

    if n == 100 { break; };
    continue 'a;
}
```

Loop може повертати значення

```
let mut total : i32 = 1;

loop {
    total *= 2;
    if total > 1000 {
        break;
    }
}

assert_eq!(1024, total);
```

```
let mut total : i32 = 1;

let outcome : i32 = loop {
    total *= 2;
    if total > 1000 {
        break total;
    }
};

assert_eq!(1024, outcome);
```


Переривання може бути в for

```
for n:i32 in 0.. {  
    println!("{}", n);  
    if n == 999 {  
        break;  
    }  
}
```

Переривання може бути в while

```
let mut n: i32 = 0;  
while true {  
    n += 1;  
    if n == 999 {  
        break;  
    }  
}
```


Переривання може бути в loop

```
let mut n: i32 = 0;  
loop {  
    n += 1;  
    if n == 999 {  
        break;  
    }  
}
```

Складні вкладені конструкції

```
let mut x:i32 = 0;
let mut y:i32 = 0;
loop {
    loop {
        x += 1;
        y += 1;
        print!("{}", x);
        if x == 5 { break; }
    }
    loop {
        x -= 1;
        y += 1;
        print!("{}", x);
        if x == 0 { break; }
    }
    if y >= 20 { break; }
}
```

1 2 3 4 5 4 3 2 1 0 1 2 3 4 5 4 3 2 1 0

Pattern Matching

```
enum Solution {  
    NoRoots,  
    OneRoot(f32),  
    TwoRoots(f32, f32),  
}
```

```
fn solve_quadratic(a: f32, b: f32, c: f32) -> Solution
```

```
match solve_quadratic(a, b, c) {  
    NoRoots =>  
        println!("quadratic equation has no roots"),  
    OneRoot(x: f32) =>  
        println!("quadratic equation has one root: {x}"),  
    TwoRoots(x1: f32, x2: f32) =>  
        println!("quadratic equation has two roots: {x1}, {x2}"),  
}
```

**Код з лекцій,
презентації Keypnote,
PDF-файли
знаходяться на GitHub:**

<https://github.com/djnzx/rust-course>