CI de Rust

SANNA Thomas, L3STI

9 février 2025

Table des matières

1 Nom du langage

Rust

2 Créateur

Graydon Hoare

3 Première apparition

2010

4 Paradigmes

— Impératif (pas purement impératif)
— Fonctionnel

5 Typage [1]

— Statique
— Fort
— Inféré

6 Peut-on créer de nouveau type ? [1]

Oui

```
struct Point {
    x: f64,
    y: f64,
}
```

7 Différents types [1]

— i32, i64, u32, u64, f32, f64 (= int32bits, int64bits, unsigned int32bits, unsigned int64bits, float32bits, float64bits)
— bool
— char

8 Gestion de la mémoire [4]

— Pas de garbage collector
— Propriété et emprunt (ownership and borrowing)
— Allocation manuelle et automatique

9 Sécurité

— Sécurité mémoire garantie par le compilateur
— Vérification des emprunts a la compilation
— Absence de data races

10 Interopérabilité

— Interopérabilité avec C et C++
— FFI (Foreign Function Interface)

11 Utilisations courantes

— Développement système
— Applications embarquées
— WebAssembly
— Services web
— Jeux vidéo

12 Niveau d'abstraction

Élevé

13 Popularité [2]

En forte croissance

14 Mutabilité [3]

— Rust est un langage immuable par défaut
— Mutabilité explicite avec le mot-clé **mut**

15 Syntaxe [2]

15.1 Print "Hello, world !"

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

15.2 Fonctions

```rust
fn add(a: i32, b: i32) -> i32 {
    a + b
}

fn main() {
    let a = 5;
    let b = 10;
    let c = add(a, b);
    println!("{} + {} = {}", a, b, c);
}
```

15.3 Structures

```rust
struct Point {
    x: f64,
    y: f64,
}

impl Point {
    fn new(x: f64, y: f64) -> Self {
        Self { x, y }
    }

    fn distance_from_origin(&self) -> f64 {
        (self.x.powi(2) + self.y.powi(2)).sqrt()
    }
}

fn main() {
    let point = Point::new(3.0, 4.0);
    println!("Distance de l'origine : {}", point.
        distance_from_origin());
}
```

15.4 Enumérations

```
enum Direction {
    Up,
    Down,
    Left,
    Right,
}

fn move_player(d: Direction) {
    match d {
        Direction::Up => println!("Le joueur monte"),
        Direction::Down => println!("Le joueur descend
            "),
        Direction::Left => println!("Le joueur va a
            gauche"),
        Direction::Right => println!("Le joueur va a
            droite"),
    }
}

fn main() {
    move_player(Direction::Up);
}
```

15.5 Traits

```
trait Animal {
    fn make_sound(&self);
}

struct Chien;

impl Animal for Chien {
    fn make_sound(&self) {
        println!("Ouaf");
    }
}

struct Chat;

impl Animal for Chat {
    fn make_sound(&self) {
        println!("Miaou");
    }
}
```

```
20
21 fn main() {
22     let chien = Chien;
23     let chat = Chat;
24
25     chien.make_sound();
26     chat.make_sound();
27 }
```

15.6 Gestion des erreurs

```
1  fn division(a: f64, b: f64) -> Result<f64, String> {
2      if b == 0.0 {
3          Err("Division par zero".to_string())
4      } else {
5          Ok(a / b)
6      }
7  }
8
9  fn main() {
10     match division(9.0, 3.0) {
11         Ok(result) => println!("Resultat : {}", result
               ),
12         Err(err) => println!("Erreur : {}", err),
13     }
14 }
```

15.7 Itérateurs

```
1  fn main() {
2      let numbers = vec![1, 2, 3, 4, 5];
3
4      for number in numbers.iter() {
5          println!("{}", number);
6      }
7  }
```

15.8 Pattern matching

```
1  fn main() {
2      let number = 42;
3
```

```
    match number {
        0 => println!("Zero"),
        1..=100 => println!("Entre 1 et 100"),
        _ => println!("Autre"),
    }
}
```

15.9 Gestion de la mémoire [4]

```
fn main() {
    let s1 = String::from("hello");
    let s2 = s1;

    println!("{}", s1); // Erreur : s1 a ete deplace
}

// avec type simple :
fn main() {
    let x = 5;
    let y = x;

    println!("{}", x); // Ok car i32 est copie
}
```

15.10 Booleens

```
fn main() {
    let a = true;
    let b = false;

    if a && b {
        println!("a et b sont vrais");
    } else if a || b {
        println!("a ou b est vrai");
    } else {
        println!("a et b sont faux");
    }
}
```

Références

[1] *Data Types - The Rust Programming Language*. URL : `https://doc.rust-lang.org/book/ch03-02-data-types.html` (visité le 05/02/2025).

[2] *Introduction - Rust By Example*. URL : `https://doc.rust-lang.org/rust-by-example/` (visité le 09/02/2025).

[3] *Variables and Mutability - The Rust Programming Language*. URL : `https://doc.rust-lang.org/book/ch03-01-variables-and-mutability.html` (visité le 05/02/2025).

[4] *What is Ownership ? - The Rust Programming Language*. URL : `https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html` (visité le 05/02/2025).