

CI de Rust

SANNA Thomas, L3STI

5 février 2025

Table des matières

1	Nom du langage	3
2	Créateur	3
3	Première apparition	3
4	Paradigmes	3
5	Typage	3
6	Peut-on créer de nouveau type ?	3
7	Différents types	3
8	Gestion de la mémoire	4
9	Sécurité	4
10	Interopérabilité	4
11	Utilisations courantes	4
12	Niveau d'abstraction	4
13	Popularité	4
14	Mutabilité	4
15	Syntaxe	4
15.1	Print "Hello, world!"	4
15.2	Fonctions	5
15.3	Structures	5
15.4	Enumérations	5
15.5	Traits	6
15.6	Gestion des erreurs	7

15.7	Itérateurs	7
15.8	Pattern matching	7
15.9	Gestion de la mémoire	8
15.10	Booleens	8

1 Nom du langage

Rust

2 Créateur

Graydon Hoare

3 Première apparition

2010

4 Paradigmes

- Impératif (pas purement impératif)
- Fonctionnel

5 Typage

- Statique
- Fort
- Inféré

6 Peut-on créer de nouveau type ?

Oui

```
1 struct Point {  
2     x: f64,  
3     y: f64,  
4 }
```

7 Différents types

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

8 Gestion de la mémoire

- 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 à 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

En forte croissance

14 Mutabilité

- Rust est un langage immuable par défaut
- Mutabilité explicite

15 Syntaxe

15.1 Print "Hello, world!"

```

1 fn main() {
2     println!("Hello, world!");
3 }

```

15.2 Fonctions

```

1 fn add(a: i32, b: i32) -> i32 {
2     a + b
3 }
4
5 fn main() {
6     let a = 5;
7     let b = 10;
8     let c = add(a, b);
9     println!("{}", a + b = c, a, b, c);
10 }

```

15.3 Structures

```

1 struct Point {
2     x: f64,
3     y: f64,
4 }
5
6 impl Point {
7     fn new(x: f64, y: f64) -> Self {
8         Self { x, y }
9     }
10
11     fn distance_from_origin(&self) -> f64 {
12         (self.x.powi(2) + self.y.powi(2)).sqrt()
13     }
14 }
15
16 fn main() {
17     let point = Point::new(3.0, 4.0);
18     println!("Distance de l'origine : {}", point.
19         distance_from_origin());

```

15.4 Enumérations

```

1 enum Direction {
2     Up,
3     Down,
4     Left,
5     Right,
6 }
7
8 fn move_player(d: Direction) {
9     match d {
10         Direction::Up => println!("Le joueur monte"),
11         Direction::Down => println!("Le joueur descend"),
12         Direction::Left => println!("Le joueur va gauche"),
13         Direction::Right => println!("Le joueur va droite"),
14     }
15 }
16
17 fn main() {
18     move_player(Direction::Up);
19 }

```

15.5 Traits

```

1 trait Animal {
2     fn make_sound(&self);
3 }
4
5 struct Chien;
6
7 impl Animal for Chien {
8     fn make_sound(&self) {
9         println!("Ouaf");
10    }
11 }
12
13 struct Chat;
14
15 impl Animal for Chat {
16     fn make_sound(&self) {
17         println!("Miaou");
18    }
19 }

```

```

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 z ro".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!("R sultat : {}",
12                               result),
13         Err(err) => println!("Erreur : {}", err),
14     }
15 }

```

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

```

```

4     match number {
5         0 => println!("Zéro"),
6         1..=100 => println!("Entre 1 et 100"),
7         _ => println!("Autre"),
8     }
9 }

```

15.9 Gestion de la mémoire

```

1 fn main() {
2     let s1 = String::from("hello");
3     let s2 = s1;
4
5     println!("{}", s1); // Erreur : s1 a été déplacé
6 }
7
8 // avec type simple :
9 fn main() {
10     let x = 5;
11     let y = x;
12
13     println!("{}", x); // Ok car i32 est copié
14 }

```

15.10 Booleens

```

1 fn main() {
2     let a = true;
3     let b = false;
4
5     if a && b {
6         println!("a et b sont vrais");
7     } else if a || b {
8         println!("a ou b est vrai");
9     } else {
10         println!("a et b sont faux");
11     }
12 }

```