Nom du langage : Rust

Créateur : Graydon Hoare

Première apparition : 2010

Paradigmes :

- Impératif (pas purement impératif)
- Fonctionnel

Typage :

- Statique
- Fort
- Inféré

Peut-on créer de nouveau type ? :

- Oui

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

Différents types :

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

Gestion de la mémoire :

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

Sécurité :

- Sécurité mémoire garantie par le compilateur
- Vérification des emprunts à la compilation
- Absence de data races

Interopérabilité :

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

Utilisations courantes :

- Développement système

- Applications embarquées
- WebAssembly
- Services web
- Jeux vidéo

Niveau d'abstraction :

- Élevé

Popularité :

- En forte croissance

Mutabilité :

- Rust est un langage immuable par défaut
- Mutabilité explicite

Syntaxe :

```
---------------------------------------- Print "Hello, world!"

fn main() {
    println!("Hello, world!");
}
---------------------------------------- Fonctions
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);
}
---------------------------------------- Structures
struct Point {
    x: f64,
    y: f64,
```

```rust
}

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());
}
---------------------------------------- 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 à gauche"),
        Direction::Right => println!("Le joueur va à droite"),
    }
}

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

```rust
fn main() {
    let chien = Chien;
    let chat = Chat;

    chien.make_sound();
    chat.make_sound();
}
---------------------------------------- Gestion des erreurs
fn division(a: f64, b: f64) -> Result<f64, String> {
    if b == 0.0 {
        Err("Division par zéro".to_string())
    } else {
        Ok(a / b)
    }
}

fn main() {
    match division(9.0, 3.0) {
        Ok(result) => println!("Résultat : {}", result),
        Err(err) => println!("Erreur : {}", err),
    }
}
---------------------------------------- Itérateurs
fn main() {
    let numbers = vec![1, 2, 3, 4, 5];

    for number in numbers.iter() {
        println!("{}", number);
    }
}
---------------------------------------- Pattern matching
fn main() {
    let number = 42;

    match number {
        0 => println!("Zéro"),
        1..=100 => println!("Entre 1 et 100"),
        _ => println!("Autre"),
    }
}
---------------------------------------- Gestion de la mémoire
fn main() {
    let s1 = String::from("hello");
    let s2 = s1;

    println!("{}", s1); // Erreur : s1 a été déplacé
}
// avec type simple :
fn main() {
    let x = 5;
    let y = x;

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