ESGI/B3/S2/Rust

Installation

- ## Installation de rustup
  - **Linux**
    - `$ curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh`
  - **Windows**
    - Installation graphique : https://www.rust-lang.org/tools/install
    - Installer Visual Studio

Concepts Clés

- ## Structure de programme basique
  - ```
    fn main() {
    // instructions
    }
    ```
- ## Variables
  - ```
    fn main() {
      // Déclaration
        let test = 10;
      // Affichage du contenu de la variable
        println!("test = {test}");
      // Autre façon d'afficher le contenu de la variable
      println!("test = {}", test);
    }
    ```
- ## Types de données
  - **Types scalaires**
    - Il y a 4 types scalaire en Rust
      - Les entiers
      - Les nombres à virgule flottante
      - Les booléens
      - Les caractères
  - **Types d'entiers**

- On a plusieurs variantes d'entiers définissant leurs tailles et chacune de ces variantes peuvent être des entiers signés ou non signés.
Taille	Signé	Non signé
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	isize	usize
- Exemple :
  - ```
    fn main() {
        // Déclaration entier signé
        let test_signed: i32 = 10.6;
        // Déclaration entier non signé
        let test_unsigned: u32 = 45.3;
    }
    ```

- **Type virgule flottante**
  - Rust propose deux types de nombre à virgule flottante.
Taille	Signé	Précision
32-bit	`f32`	simple
64-bit	`f64`	double
- **Type booléen**
  - Le booléen occupe 1 octet. Il peut avoir deux valeurs : True ou False
  - Exemple :
    - ```
      fn main() {
          // Déclaration du booléen
          let test_bool: bool = true;
      }
      ```
- **Type caractère**
  - Le type car est le seul type scalaire de caractrère en Rust. il est toujours sur 32 bits.
  - Exemple :
    - ```
      fn main() {
          // Déclaration entier signé
          let test_char: char = "Test";
      }
      ```

Fonctions

- On utilise "fn" pour déclarer une fonction.
- Il est obligatorie de typer les arguments entre ()
      ```

- On peut aussi préciser le type du retour, la dernière expression sans point-virgule est la valeur retrounée.
- Exemple :
- ```
  fn addition(a: i32, b: i32) → i32 {
      a + b
  }
  ```

Contrôle de flux

If

- Le code sera exécuter si la condition est remplie.
- Exemple :
- ```
  if nombre == 20 {
      println!("condition remplie")
  };
  ```

For

- Boucle qui itère sur une colleciton, un tableau, un range etc...
- Exemple :
- ```
  for i in 1..=5 { // de 1 à 5 inclus
      println!("i vaut {}", i);
  }

  for i in 1..5 { // de 1 à 4
      println!("i vaut {}", i);
  }

  let fruits = ["pomme", "banane", "cerise"]; // Avec un tableau

      for fruit in fruits.iter() {
          println!("Fruit : {}", fruit);
      }
  ```

While

- Exécute le bloc de code tant que la condition est vraie
- Exemple :
- ```
  let mut i = 0;

      while i < 10 {
          println!("Bonjour ! i = {}", i);
          i += 1;
      }
  ```

Loop

- Une boucle infinie qui tourne sans limite à moins d'être arrêtée explicitement.
- Exemple :

- ```
  let mut compteur = 0:

  loop {
      compteur += 1;

      if compteur == 5 {
          break;
      }
  }
  ```

- # Structures
  - Elles permettent de définir des types composés regroupant plusieurs champs de données comparables aux classes sans méthodes par défaut.
  - Exemple :
  - ```
    struct Article {
        nom: String,
        marque: String,
        prix: f64,
    }
    ```
- # Traits
  - Les traits définissent un ensemble de méthodes qu'un type doit implémenter, ressemblant aux interfaces en Java
  - Exemple :
  - ```
    trait Afficher {
        fn afficher(&self);
    }
    ```
- # Collections dynamiques : vecteurs
  - Tableau redimensionnable et typé, très utilisé en Rust
  - Exemple :
  - ```
    let mut comptes = Vec::new();
    comptes.push(CompteBancaire { nom: String::from("Alice"), solde: 1000.0 });
    ```
- # Lecture interactive et gestion des entrées utilisateur
  -

  Lecture de lignes depuis la console via `std::io::stdin().read_line(&mut buffer)` — cette fonction bloque jusqu'à appui sur **Entrée**.

  -

  Nettoyage des chaînes avec `trim()` pour enlever espaces et retour chariot.

  -

  Transformation sécurisée en valeurs numériques avec `parse()` et gestion d'erreur via `match` .

  -

Validation d’entrées au moyen de boucles jusqu’à une saisie correcte.