

Rust Cheat Sheet

Feuille de triche pour connaître les bases du RUST

Projet Cargo

Créer projet :

```
1 cargo new projet
2 cd projet
3 cargo run
```

Commandes utiles :

```
1 cargo build
2 cargo check
3 cargo test
4 cargo doc --open
```

Ajouter dépendance :

```
1 [dependencies]
2 serde = "1.0"
3 tokio = "1"
```

Types de Variables

Types entiers :

```
1 i8 i16 i32 i64 i128 isize
2 u8 u16 u32 u64 u128 usize
```

Float :

```
1 f32
2 f64
```

Booléen :

```
1 let b: bool = true;
```

Caractère :

```
1 let c: char = 'a';
```

String :

```
1 let s1: &str = "hello";
2
3 let s2: String =
4     String::from("hello");
```

Tuple :

```
1 let t = (1, 3.14, true);
2
3 let (a,b,c) = t;
```

Array :

```
1 let arr = [1,2,3];
2
3 let zeros = [0;10];
```

Slice :

```
1 let slice = &arr[0..2];
```

Variables

```
1 let x = 5;
2 let mut y = 10;
3
4 const PI: f64 = 3.14;
```

Shadowing :

```
1 let x = 5;
2 let x = x + 1;
```

Struct

```
1 struct User {
2     name: String,
3     age: u32,
4 }
5
6 impl User {
7
8     fn new(name:String,
9           age:u32) -> Self {
10         Self { name, age }
11     }
12
13     fn is_adult(&self)->bool{
14         self.age >= 18
15     }
16 }
17
```

Enum

```
1 enum Message {
2
3     Quit,
4
5     Move {x:i32,y:i32},
6
7     Write(String)
8 }
9
```

Option

Option représente une valeur qui peut être absente.

```
1 enum Option<T> {
2
3     Some(T),
4
5     None
6 }
7
```

Exemple :

```
1 let x: Option<i32> = Some(5);
2
3 let y: Option<i32> = None;
```

Match :

```
1 match x {
2
3     Some(v) => println!("{}", v),
4
5     None => println!("{}", "None")
6 }
7
```

if let :

```
1 if let Some(v) = x {
2
3     println!("{}", v);
4 }
5
```

Méthodes utiles :

```
1 x.unwrap()
2
3 x.unwrap_or(0)
4
5 x.is_some()
6
7 x.is_none()
```

Pattern Matching

```
1 match msg {
2
3     Message::Quit => {}
4
5     Message::Move{x,y} => {
6         println!("{}",x);
7     }
8
9     _ => {}
10 }
11
```

Ownership

Règles :

- 1 propriétaire
- move par défaut
- drop automatique

```
1 let s1 = String::from("hello");
2
3 let s2 = s1; // move
```

Clone :

```
1 let s2 = s1.clone();
```

Borrowing

```
1 fn len(s: &String) -> usize {
2
3     s.len()
4
5 }
```

Mutable :

```
1 fn change(s: &mut String){
2
3     s.push_str("!");
4
5 }
```

Generics

```
1 fn largest<T: PartialOrd>
2 (list: &[T]) -> &T {
3
4     let mut max = &list[0];
5
6     for item in list {
7
8         if item > max {
9
10             max = item;
11         }
12     }
13
14     max
15 }
16
17
```



Traits

```
1 trait Animal {  
2  
3 fn speak(&self);  
4  
5 }  
6  
7 impl Animal for Dog {  
8  
9 fn speak(&self){  
10  
11 println!("Woof");  
12  
13 }  
14  
15 }
```

Collections

Vector :

```
1 let mut v = vec![1,2,3];  
2  
3 v.push(4);
```

HashMap :

```
1 use std::collections::HashMap;  
2  
3 let mut map = HashMap::new();  
4  
5 map.insert("a",1);
```

Gestion des erreurs

Rust utilise Result :

```
1 enum Result<T,E> {  
2  
3 Ok(T),  
4  
5 Err(E)  
6  
7 }
```

Exemple :

```
1 use std::fs::File;  
2  
3 let file = File::open("test.txt");
```

Match :

```
1 match file {  
2  
3 Ok(f) => println!("File open"),  
4  
5 Err(e) => println!("{}", e)  
6  
7 }
```

Propagation d'erreur :

```
1 use std::fs::File;  
2  
3 fn open_file()  
4 -> Result<File,std::io::Error>{  
5  
6 let f = File::open("test.txt");  
7  
8 Ok(f)  
9  
10 }
```

? retourne automatiquement l'erreur au caller.

Async

```
1 async fn hello() {  
2  
3 println!("hello");  
4  
5 }  
6  
7 #[tokio::main]  
8  
9 async fn main(){  
10  
11 hello().await;  
12  
13 }
```

