

2. Constantes, Variáveis e Mutabilidade

Ao começar a programar em Rust, um dos primeiros conceitos que surpreendem os programadores vindos de Python é a diferença entre como as variáveis e a mutabilidade são tratadas. Embora ambas linguagens lidem com variáveis de forma eficaz, a abordagem e os objetivos são fundamentalmente diferentes.

2.1. Constantes: O Que Não Muda, Não Estraga (Geralmente)

Ao contrário das variáveis, constantes têm seus valores definidos em tempo de compilação e não podem ser alteradas durante a execução do programa. Elas são úteis para armazenar valores que permanecem fixos ao longo da execução do programa.

Em Rust:

Constantes são declaradas com a palavra-chave `const` e sempre exigem anotação de tipo. Esta definição de tipo se deve ao fato que, durante a compilação, os locais onde a constante é utilizada são substituídas por seus respectivos valores.

```
const PI: f64 = 3.14159;
fn main() {
    println!("O valor de PI é: {}", PI);
}
```

Em Python:

Não possui suporte nativo para constantes imutáveis, mas convencionou-se usar nomes em letras maiúsculas para indicar variáveis que devem ser tratadas como constantes. Essa convenção é cultural, e nada impede que o valor seja alterado.

```
PI = 3.14159      # Convenção para constantes
print(f"O valor de PI é: {PI}")
# Não há restrição para alteração, mas é desencorajado
```

2.2. Declaração de Variáveis: Onde Seus Dados Ganham Vida (e um Tipo!)

Uma variável é um espaço nomeado na memória que pode armazenar dados e cujo valor pode ser alterado durante a execução de um programa. Em linguagens como Rust, as variáveis são mutáveis quando explicitamente marcadas como `mut`. Já em Python, todas as variáveis são mutáveis, a menos que se trate de um tipo imutável, como tuplas.

Rust é fortemente tipado e requer que o tipo seja declarado ou inferido no momento da inicialização ou do primeiro uso da variável. Além disso, por padrão, as variáveis são imutáveis.

Exemplo em Rust:

```
fn main() {  
    let mut x = 5;           // Variável mutável e inferida como inteiro  
    let y: i32 = 20;        // Variável imutável e com tipo explícito i32 (integer 32bits)  
  
    println!("O valor de x é: {}", x);  
    x = 10;                 // Alteração do valor de x  
    println!("Agora o valor de x é: {}", x);  
}
```

Python é uma linguagem de tipagem dinâmica, o que significa que é possível declarar uma variável sem especificar seu tipo, e este tipo pode mudar durante a execução do programa.

Exemplo em Python:

```
x = 5                        # Variável em Python  
print(f"O valor de x é: {x}")  
x = 10                      # Alteração do valor de x  
print(f"Agora o valor de x é: {x}")  
x = "Hello"                 # Agora x é uma string  
print(f"X agora é o texto: {x}")
```

Rust não permite que uma variável mude de tipo após sua declaração.

```
let x = 10;                 // Inteiro  
// x = 6; // Erro de compilação: variável imutável  
// x = "Hello"; // Erro de compilação: tipo incompatível  
  
// Esses prints são equivalentes pois não precisam computar o valor de x  
println!("The value of x is: {}", x);  
println!("The value of x is: {x}");
```

Para cenários onde é necessário computar o valor, é necessário passar o valor para formatação como argumento, ao invés de dentro das chaves:

```
println!("The value of 2*x is: {}", 2 * x);
```

Passar o valor dentro das chaves resulta em um erro de compilação:

```
println!("The value of 2*x is: {2 * x}");
error: invalid format string: expected `}`, found `*`
--> src/main.rs:9:39
|
9 |     println!("The value of 2*x is: {2 * x}");
    |                                     - ^ expected `}` in format string
    |                                     |
    |                                     because of this opening brace
    |
= note: if you intended to print `{`, you can escape it using `{{`
error: could not compile `main` (bin "main") due to 1 previous error
```

Outro ponto que será bastante citado durante os próximos capítulos é sobre os erros de compilação providos pelo Rust. São muito descritivos, amigáveis e legíveis.

Para o exemplo acima, caso a atribuição `x = 6` seja descomentada, o erro se apresenta como:

```
error[E0384]: cannot assign twice to immutable variable `x`
--> src/main.rs:3:5
|
2 |     let x = 10; // Inteiro
    |               - first assignment to `x`
3 |     x = 6; // Erro de compilação: variável imutável
    |     ^^^^^ cannot assign twice to immutable variable
    |
help: consider making this binding mutable
|
2 |     let mut x = 10; // Inteiro
    |               +++

For more information about this error, try `rustc --explain E0384`.
error: could not compile `main` (bin "main") due to 1 previous error
```

Acima é possível identificar várias informações úteis para a solução do problema:

- **src/main.rs:3:5:** Arquivo onde o compilador pegou o erro com número da linha e a coluna, neste exemplo na linha 3 e coluna 5;
- **cannot assign twice to immutable variable:** O motivo do erro encontrado pelo compilador;
- **help: consider making this binding mutable:** Uma possível sugestão de como resolver, neste caso transformar a variável `x` em mutável;
- **let mut x = 10;** Como fazer a sugestão proposta;
- **For more information about this error, try `rustc --explain E0384`** : O link da doc onde é possível obter mais detalhes, neste caso proporciona a saída abaixo:

```
// An immutable variable was reassigned.
```

```
// Erroneous code example:
```

```
fn main() {  
    let x = 3;  
    x = 5; // error, reassignment of immutable variable  
}
```

```
// By default, variables in Rust are immutable. To fix this error, add the keyword mut  
// example:
```

```
fn main() {  
    let mut x = 3;  
    x = 5;  
}
```

```
// Alternatively, you might consider initializing a new variable: either with a new bou  
// existing variable. For example:
```

```
fn main() {  
    let x = 3; let  
    x = 5;  
}
```

2.3. Mutabilidade

Em Python:

Por padrão, todas as variáveis em Python são mutáveis (exceto tipos imutáveis, como tuplas e strings). É possível alterar o valor de uma variável a qualquer momento, sem restrições.

```
x = 10  
x = 20    # Permitido
```

Em Rust:

A palavra-chave **let** é utilizada em várias linguagens para declarar variáveis. O Rust exige que a mutabilidade seja declarada explicitamente, o que promove segurança e evita alterações acidentais nos valores.

```
let x = 10;  
// x = 20; // Erro de compilação: x não é mutável
```

```
let mut y = 10;      // Variável mutável  
y = 20;             // Agora permitido
```

2.4. Segurança em Concorrência

Rust adota a imutabilidade por padrão como parte de seu modelo de segurança em memória e concorrência. Programadores de Python estão acostumados com mutabilidade irrestrita, mas em sistemas concorrentes, isso pode levar a condições de corrida.

Em Python, alterar uma variável global em um ambiente multi-threaded pode levar a resultados inesperados.

Exemplo em Python:

```
counter = 0  
  
def increment():  
    global counter  
    counter += 1    # Pode causar problemas em threads simultâneas
```

Exemplo em Rust:

Rust impede que múltiplas threads modifiquem dados simultaneamente sem proteção explícita, como mutexes ou referências imutáveis compartilhadas.

```
use std::sync::Mutex;  
  
let counter = Mutex::new(0);  
{  
    let mut num = counter.lock().unwrap();  
    *num += 1;    // Seguro e protegido  
}
```

Não se preocupem com o **Mutex** agora, será abordado mais adiante.

2.5. Shadowing (Sombreamento)

O Shadowing é um conceito em Rust que permite declarar uma nova variável com o mesmo nome de uma variável já existente em um escopo, efetivamente "escondendo" a variável anterior, algo que Python não suporta diretamente. Quando isso acontece, a variável original ainda existe, mas não

pode mais ser acessada pelo nome dentro do escopo onde o shadowing ocorreu, pois o nome agora se refere à nova variável.

Em Rust:

é possível reutilizar o mesmo nome de variável para um novo valor.

```
let x = 10;
let x = x + 5;           // Sombra criada; x agora vale 15
```

Em Python:

Reatribuir o nome da variável simplesmente sobrescreve o valor anterior.

```
x = 10
x = x + 5  #Agora x vale 15
```

A diferença é que, em Rust, o sombreamento cria uma nova variável com o mesmo nome, enquanto em Python, o valor antigo é descartado.

2.6. Desempenho e Segurança

Rust, ao enfatizar a imutabilidade e a segurança explícita, previne bugs difíceis de rastrear e melhora o desempenho em sistemas concorrentes. Em Python, a mutabilidade irrestrita é mais flexível, mas pode levar a bugs em aplicações maiores.

Aspecto	Python	Rust
Mutabilidade padrão	Todas as variáveis são mutáveis, salvo tipos imutáveis (ex.: tuplas, strings)	Variáveis são imutáveis por padrão. Para torná-las mutáveis, usa-se <code>mut</code>
Constantes	Não existem constantes nativas; apenas convenções com letras maiúsculas	Declaradas com <code>const</code> e sempre imutáveis
Tipo	Python deduz o tipo automaticamente	Rust exige anotação de tipo para constantes
Declaração de tipo	Dinâmica	Estática
Sombreamento	Não suportado diretamente	Suportado
Segurança em memória	Sem restrições explícitas	Segurança garantida pelo compilador

A diferença central entre as duas linguagens é que Python prioriza a flexibilidade, enquanto Rust prioriza a segurança e eficiência. Ao entender como Rust lida com variáveis e mutabilidade, programadores de Python podem aprender a estruturar melhor seus programas e evitar problemas comuns em aplicações maiores.

2.7. Conclusões

- Use variáveis quando precisar alterar o valor ao longo do programa.
- Prefira constantes para valores fixos e frequentemente acessados, especialmente em linguagens como Rust.
- Em Python, mesmo sem constantes imutáveis, adote convenções para melhorar a legibilidade e manutenção do código.