> *Fundamentos básicos de Ruby Professor Alexandre Calaça*

# Characteristics of Variables in Ruby

## Variables in Ruby

### Characteristics

- **Scope:** Determines where the variable is accessible.
- **Dynamic Typing:** Variables can change types dynamically.
- **Reassignment:** Variables can be reassigned to different values.
- **Multiple Assignments:** Assign multiple values to multiple variables in a single line.
- **Value Swapping:** Swap values between variables efficiently.
- **Sigils:** Prefixes that indicate the variable's scope.
- **Immutability:** Some objects in Ruby are immutable, meaning they cannot be modified.

---

## 1. Scope

Scope determines where a variable can be accessed within a program. In this material, we'll cover only two forms of scopes, we'll dive into those topics in another file.

**Example 1: Local Scope**

```ruby
def example
  name = "Alice"
  puts name # Accessible inside the method
end

example
puts name # Error: undefined local variable
```

**Example 2: Instance Scope**

```ruby
class Person
  def initialize(name)
    @name = name
  end
  def greet
    puts "Hello, my name is #{@name}"
  end
end

p = Person.new("Alice")
p.greet # Outputs: Hello, my name is Alice
```

---

## 2. Dynamic Typing

Variables in Ruby can hold any object and their types can change dynamically.

**Example 1: Changing Variable Type**

```
var = 10
puts var.class # Integer
var = "Hello"
puts var.class # String
```

**Example 2: Dynamic Method Invocation**

```
num = 42
puts num.to_s # Converts Integer to String
```

---

## 3. Reassignment

Variables can be reassigned to hold different values.

**Example 1: Reassigning a Variable**

```
x = 5
x = 10
puts x # Outputs: 10
```

**Example 2: Reassigning with Expressions**

```
count = 2
count += 1 # Equivalent to count = count + 1
puts count # Outputs: 3
```

---

## 4. Multiple Assignments

Assign multiple values in a single line.

**Example 1: Basic Multiple Assignments**

```
a, b, c = 1, 2, 3
puts a, b, c # Outputs: 1, 2, 3
```

**Example 2: Swapping Values with Multiple Assignments**

```
a, b = 5, 10
a, b = b, a
puts a, b # Outputs: 10, 5
```

---

## 5. Value Swapping

Easily swap values between variables.

**Example 1: Simple Swap**

```ruby
x, y = 1, 2
x, y = y, x
puts x, y # Outputs: 2, 1
```

**Example 2: Swapping with Temporary Variable**

```ruby
temp = x
x = y
y = temp
puts x, y # Outputs: 2, 1
```

---

## 6. Sigils

Ruby uses different prefixes (sigils) to denote variable scope.

**Example 1: Local Variable**

```ruby
name = "Ruby"
puts name # Accessible only within this scope
```

**Example 2: Instance and Class Variables**

```ruby
class Example
  def initialize(name)
    @name = name # Instance variable
  end
  def self.set_class_var
    @@class_var = "Class-wide"
  end
end
```

---

## 7. Immutability

Certain objects in Ruby, such as symbols and frozen objects, are immutable.

**Example 1: Symbols**

```ruby
sym1 = :hello
sym2 = :hello
puts sym1.object_id == sym2.object_id # Outputs: true
```

**Example 2: Freezing Objects**

```ruby
name = "Ruby"
name.freeze
name << " on Rails" # Raises error: FrozenError
```

---

## Challenges

### Easy mode

1. Declare a variable and assign different types of values to it (Integer, String, Boolean). Print its type after each assignment.
2. Swap two variable values without using a third variable.

### Regular mode

1. Write a method that takes two numbers as arguments, swaps their values, and returns them.
2. Create a class with an instance variable and a method that modifies the instance variable. Instantiate the class and call the method.
3. Implement a program that demonstrates scope by using local, instance, and global variables inside different methods and classes.