# Equality Is Hard

## Ruby

### **Structural Equality(Reflexive Equality)**

Structural equality asks if two references are the same value.

In order to compare things Ruby has a bunch of comparison operators.

**Example**

* 1 == 1 \* 1 will return true,
* "A" == "A" also returns true because both strings have the same value.
* [1, 2] == [1, 2] will return true,
* but [1, 2] == [2, 3] and [1, 2] == [2, 1] both will return false.

**Integer and Floating Points**

When comparing numbers of different types (e.g., integer and float), if their numeric value is the same, == will return true.

2 == 2.0 # Output: => true

### **Reference equality.**

Reference equality asks if the two objects being compared are the same object[1]. In other words, does the variable point at the same area of memory?

**Object References**

When we work with objects in Ruby, we are really working with object references[1]. It is not the object itself we manipulate but a reference to it.[[\*](https://www.oreilly.com/library/view/the-ruby-programming/9780596516178/ch03s08.html#ftn.id777333)] When we assign a value to a variable, we are not copying an object “into” that variable; we are merely storing a reference to an object into that variable.

**Example**

Some code makes this clear:

* s = "Ruby" # Create a String object. Store a reference to it in s.
* t = s # Copy the reference to t. s and t both refer to the same object.
* t[-1] = "" # Modify the object through the reference in t.
* print s # Access the modified object through s. Prints "Rub".
* t = "Java" # t now refers to a different object.
* print s,t # Prints "RubJava".

When you pass an object to a method in Ruby, it is an object reference that is passed to the method. It is not the object itself, and it is not a reference to the reference to the object. Another way to say this is that method arguments are passed by value rather than by reference, but that the values passed are object references.

**Case-equality operator**

Also known as **triple equals** [2].

This operator does not test equality, but rather tests if the right operand has an **IS A relationship** with the left operand. As such, the popular name case equality operator is misleading.

This answer describes it thus: the best way to describe a === b is "if I have a drawer labeled a, does it make sense to put b in it?" In other words, does the set a include the member b?

**Examples** [3][4]

(1..5) === 3 # => true

(1..5) === 6 # => false

Integer === 42 # => true

Integer === 'fourtytwo' # => false

/ell/ === 'Hello' # => true

/ell/ === 'Foobar' # => false

So, for example, Module#=== tests whether b.is\_a?(a). If you have Integer === 2 , means Integer module has number 2?

**Program**

**def check\_num(a, b, c)**

**if a==b && b==c**

**return 0**

**elsif a==b**

**return c**

**elsif a==c**

**return b**

**elsif b==c**

**return a**

**else**

**return a+b+c**

**end**

**end**

**//Check Structural equality of integer w.r.t floating point**

**print check\_num(5.0, 5, 5.0),"\n"**

**//Check Structural equality of Character answer will be letter B**

**print check\_num(“A”, “A”, “B”),"\n"**

**print check\_num(5, 7, 5),"\n"**

**print check\_num(7, 5, 5),"\n"**

**print check\_num(1, 2, 3),"\n"**

## **Rust**

### Structural Equality

**In case of Integer**

Rust programming language differentiates between signed and unsigned integer types [6].

Rust comes with multiple integer types: u32, i8, **usize, isize** . They all have something in common, they are prefixed with either u (standing for *unsigned*) or i (standing for, well *integer*).

**Case A:**

**let x: i32 = 4; let y: i16 = 4; println!("{}", x == y);**

**When compiling the snippet above, the compiler prints the following error:**

**error[E0308]: mismatched types --> src/main.rs:5:25 | 5 | println!("{}", x == y); | ^ expected i32, found i16**

**Case B:**

Multiple variables can interact with the same data in different ways in Rust. Let’s look at an example using an integer

let x = 5;

let y = x;

**// answer 5 in case integer**

We can probably guess what this is doing: “bind the value 5 to x; then make a copy of the value in x and bind it to y.” We now have two variables, x and y, and both equal 5. This is indeed what is happening, because integers are simple values with a known, fixed size, and these two 5 values are pushed onto the stack.

**Some important points**

* Numeric cast [5]
  + Casting between two integers of the same size (e.g. i32 -> u32) is not a probelm
  + Casting from a larger integer to a smaller integer (e.g. u32 -> u8) will truncate
  + Casting from a smaller integer to a larger integer (e.g. u8 -> u32) will
    - zero-extend if the source is unsigned
    - sign-extend if the source is signed
  + Casting from a float to an integer will round the float towards zero
  + Casting from an integer to float will produce the closest possible float
  + Casting from an f32 to an f64 is perfect and lossless
  + Casting from an f64 to an f32 will produce the closest possible f32
* Enum cast
  + Casts an enum to its discriminant, then uses a numeric cast if needed.
* Primitive to integer cast
  + false casts to 0, true casts to 1
  + char casts to the value of the code point, then uses a numeric cast if needed.
* u8 to char cast
  + Casts to the char with the corresponding code point.

The complete casting with respect to relative type can be studied in more detail[6].

### **String version:**

let s1 = String::from("hello");

let s2 = s1;// wrong

When we assign s1 to s2, the String data is copied, meaning we copy the pointer, the length, and the capacity that are on the stack. We do not copy the data on the heap that the pointer refers to.

## **References**

1. <https://www.oreilly.com/library/view/the-ruby-programming>
2. <https://github.com/rubocop-hq/ruby-style-guide>
3. <https://riptutorial.com/ruby/example/14092/case-equality-operator>
4. <https://medium.com/@khalidh64/difference-between-eql-equal-in-ruby-2ffa7f073532>
5. <https://doc.rust-lang.org/reference/expressions/operator-expr.html#comparison-operators>
6. <https://medium.com/@marcinbaraniecki/on-integer-types-in-rust-b3dc1b0a23d3>
7. <https://www.reddit.com/r/rust/comments/dbhbxl/a_couple_questions_about_string_comparison/>
8. <https://www.linuxjournal.com/content/text-processing-rust>