

UFO 演算子 について

(What's the Story) Spaceship Operator?





Hirofumi Wakasugi (@5t111111)

プログラマー

Spaceship Operator (<=>)

- Basic operator for comparing the receiver against another object
- Also known as “UFO operator” (least in Japan)
- Classes mix-ins Comparable must implement <=>

```
class SizeMatters
  include Comparable

  def initialize(str)
    @str = str
  end

  def <=>(other)
    str.size <=> other.str.size
  end
end
```

Implementation of <=>

It should return

- Positive number if self is greater than other
- Zero if self is equal to other
- Negative number if self is less than other

```
class Foo
  ...
  def <=>(other)
    return -1 if size < other.size
    return 0 if size == other.size
    return 1 if size > other.size
  end
end

foo1 < foo2 #=> true/false
```

A tip for using <=>

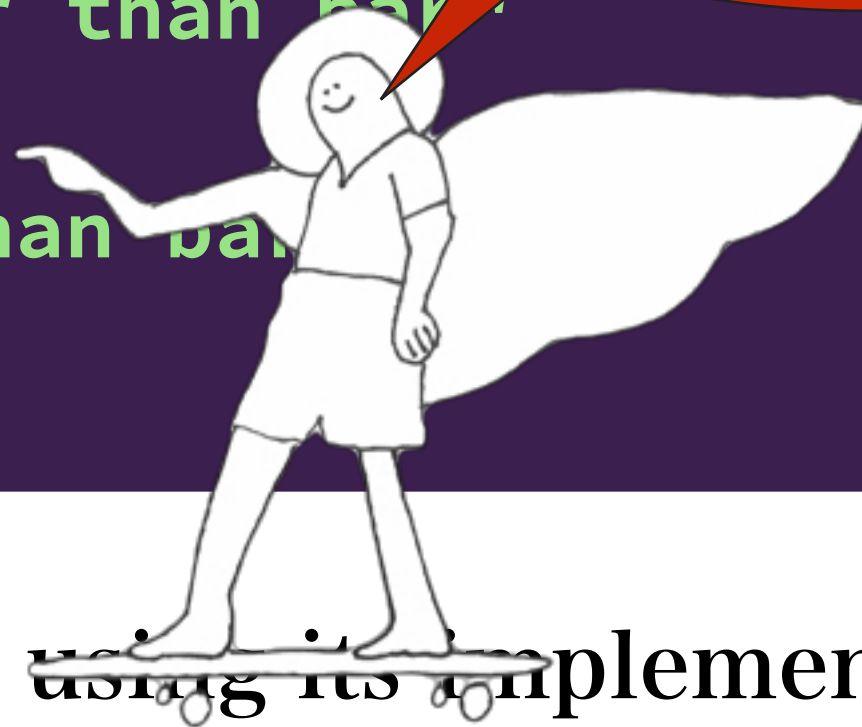
```
case (foo <=> bar) <=> 0
when 0
  puts "foo is equal to bar"
when 1
  puts "foo is greater than bar"
else
  puts "foo is less than bar"
end
```

- You can write like this using its implementation
- Looks pretty convenient, doesn't it?

A tip for using `<=>`

```
case (foo <=> bar) <=> 0
when 0
  puts "foo is equal to bar"
when 1
  puts "foo is greater than bar"
else
  puts "foo is less than bar"
end
```

0?1?
Greater if positive?
Or less if positive?
Or positive if positive?
WTF?



- You can write like this using its implementation
- Looks pretty convenient, doesn't it?

So I always do the following steps when I want to use comparison operators with my classes

1. Include the module in a class
2. Start writing ``def <=>(other); end``
3. be anguished trying to remind me to if greater/less must return positive/negative number
4. Googling for a while
5. ~~It does not work...~~
6. ~~Uh oh I include Enumerable instead of Comparable~~



**So I think Implementation of `<=>`
exceptionally lacks descriptive power
in Ruby world**

almost seems like magic numbers

It reminds me of something like

```
int compare_int(const void *a, const void *b)
{
    return *(int*)a - *(int*)b;
}

int main()
{
    int data[] = {4, 2, 3, 6, 3, 2, 1, 3, 5, 7};
    qsort(data, 10, sizeof(int), compare_int);
}
```

- Old school C-ish flavor
- Java (compareTo) / .NET (CompareTo)



(I am sure you can use if...elsif... though)

I wish I could write like

```
case foo <=> bar
when EQ
  puts "foo is equal to bar"
when GT
  puts "foo is greater than bar"
else
  puts "foo is less than bar"
end
```

- It would be easier to know the codes' intension

Haskell

Ord

```
class Eq a => Ord a where  
  compare :: a -> a -> Ordering
```

Ordering

```
data Ordering = LT | EQ | GT
```

compare returns Ordering

```
2 `compare` 3  
LT
```

Rust

Ordering

```
pub enum Ordering {  
    Less,  
    Equal,  
    Greater,  
}
```

cmp returns Ordering

```
let result = 1.cmp(&2);  
assert_eq!(Ordering::Less, result);
```

```
let result = 1.cmp(&1);  
assert_eq!(Ordering::Equal, result);
```



What if you simply use constants

```
module Ordering
  GT = 1.0..Float::INFINITY
  LT = -Float::INFINITY..-1
  EQ = 0
end

case foo <=> bar
when Ordering::EQ
  puts "3 foo is equal to bar"
when Ordering::GT
  puts "3 foo is greater than bar"
when Ordering::LT
  puts "3 foo is less than bar"
end
```

- Range using INFINITY would work with case expression

What if these constants are available in global namespace...

```
def <=>(other)
  if size > other.size
    Ordering::GT
  elsif size == other.size
    Ordering::EQ
  else
    Ordering::LT
  end
end
```

- It does not work because of their types
- Maybe it has some overhead



I am sure implementation requirement
of `<=>` lets you to write like

```
def <=>(other)  
  size - other.size  
end
```

I love this as it is clean and concise enough (as long
as you understand its requirement correctly)

Conclusion (?)

- In almost all cases the implementation that $\leq$ returns -n/0/n is enough and also has advantage
- More descriptive options would be very welcome especially for newbies
- I don't have any good ideas yet 😞 (considering if singleton objects like nil/true/false work)



THANK YOU

I would also like to say thank you to

- All RubyKaigi staffs and speakers
 - All rubyists I have never/ever met
 - (with emphasis) Drecom. RubyKaraoke was just EPIC
- … and @MakeNowJust for giving me inspiration (<https://github.com/crystal-lang/crystal/pull/2913>)