Full Stack Notes

Booleans and Conditions

Boolean values let us script decisions.

Table of Contents

True and False

All variables in Ruby are `true`, unless they are set to `false` or `nil`. This includes empty strings or even the number zero.

```
if (0)
  puts 'Zero is true!'
end

empty_string = ''

if (empty_string)
  puts 'An empty string is true!'
end
```

Output:

```
Zero is true!
An empty string is true!
```

Boolean Expressions

A boolean expression is a mathematical expression that results in either `true` or `false`.

Boolean expressions can contain the following common operators (and more):

Symbol	Meaning
==	equal
!=	not equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
&&	Boolean 'and'
||	Boolean 'or'

Boolean expressions can also contain a single boolean variable.

Let's assume we have a boolean variable named `you_have_had_enough` which has already been set to either true or false:

```
puts 'Enough Already' if you_have_had_enough
```

If Statements

Ruby's if statements work like you would expect.

```
if x > 5
  puts 'Yo, x is still larger than 5.'
end
```

Note that parenthesis around the Boolean expression are optional.

`if` as a modifier:

```
puts 'Great Scott!' if speed_of_delorean > 88
```

If Else Statements

Else statements also work as expected:

```
if (temperature < 0)
  clothing = 'snowsuit'
elsif (temperature < 25)
  clothing = 'pant suit'
else
  clothing = 'bathing suit'
end
```

Note that we use `elsif` and not `else if`.

Unless Statements

The `unless` statement is the bizarro evil twin (the logical inverse) of `if`:

```
unless temperature < 0
  puts 'Let us go for a stroll.'
end
```

Unless is rarely paired with an `else` and can most often be found as a trailing modifier:

```
snow_fort = 'awesome home' unless temperature > 0
```

All `unless` statements can be replaced by an `if` statement and an exclamation mark 'not' modifier. However, this sometimes makes the expression harder to read for humans. (Robots on the other hand love the 'not' modifier.)

```
if !(temperature < 0) # Better written as the equivalent: if temperature >= 0
  puts 'Let us go for a stroll.'
end
```

Case Statements

Case statements are similar to those in Java. The `number_of_chairs` variable is said to be the *target* of the `case` statement.

```
number_of_chairs = 4

case number_of_chairs
when 1
  puts 'Lonely with teardrops in my tea.'
when 2
  puts 'Tea for two.'
  puts 'Two for tea.'
```

```
when (3..10) # COOL: Comparing against a range.
  puts "It's a tea party!"
else
  puts 'I feel claustrophobic.'
end
```

Note: Unlike Java we do not require `break`s after each `when` block.

Case Statment With Expressions

The target of the `case` statement can be left-out:

```
enlightenment = 42


case
when enlightenment > 60
  puts 'You are too hasty, grasshopper.'
when (enlightenment < 40 || enlightenment == nil)
  puts 'You are like the sloth, my friend. Diligence is key!'
when enlightenment == 42
  puts 'Hello, Enlightened One.'
else
```

```
  puts 'Yeah, not quite, pal. Maybe next time.'
end
```

RESOURCES

- Example lovingly ‘borrowed’ from the Humble Little Ruby Book. (See Resources.)
