exercise01.rb

As stated in the tutorial "Both or and || return their first argument unless it is false, in which case they evaluate and return their second argument."

```
x = true; puts "x = #{x.class}  object id = #{x.object_id}"
y = false; puts "y = #{y.class}  object id = #{y.object_id}"
z = true; puts "z = #{z.class}  object id = #{z.object_id}"
```

My first guess was "true"

```
x = y or z
puts "input : x = y or z    output : x = #{x.class}  object id = #
{x.object_id}"
```

This is reversed

```
x = z or y
puts "input : x = z or y    output : x = #{x.class}  object id = #
{x.object_id}"
```

1. My first guess was "true"

```
(x = y) or z
puts "input :(x = y) or z   output : x = #{x.class}  object id = #
{x.object_id}"
```

1. This is reversed

```
(x = z) or y
puts "input :(x = z) or y   output : x = #{x.class}  object id = #
{x.object_id}"
```

2. My first guess was "true"

```
x = (y or z)
```

```
puts "input : x = (y or z)  output : x = #{x.class}  object id = #
{x.object_id}"
```

2. This is reversed

```
x = (z or y)
puts "input : x = (z or y)  output : x = #{x.class}  object id = #
{x.object_id}"
```

"and and or are control flow operators, not boolean operators."

devblog.avdi.org/2010/08/02/using-and-and-or-in-ruby/

Try

```
puts "----------------- Using the right boolean operators -----------------"
x = y || z
puts "input : x = y || z    output : x = #{x.class}  object id = #
{x.object_id}"

x = z || y
puts "input : x = z || y    output : x = #{x.class}  object id = #
{x.object_id}"

(x = y) || z
puts "input :(x = y) || z   output : x = #{x.class}  object id = #
{x.object_id}"

(x = z) || y
puts "input :(x = z) || y   output : x = #{x.class}  object id = #
{x.object_id}"

x = (y || z)
puts "input : x = (y || z)  output : x = #{x.class}  object id = #
{x.object_id}"

x = (z || y)
puts "input : x = (z || y)  output : x = #{x.class}  object id = #
{x.object_id}"
puts
# Uncomment the last two lines and see what happens.
# true = 1
# false = 0
```