CONTROL FLOW IN RUBY

How It Works

You may have noticed that the kinds of programs we've written so far in Ruby aren't very flexible. Sure, they can take user input, but they always produce the same result based on that input; they don't change their behavior in reaction to the **environment** (the collection of all variables and their values that exist in the program at a given time).

Control flow gives us the flexibility we're looking for. We can select different outcomes depending on information the user types, the result of a computation, or the value returned by another part of the program.

Instructions

1.
Check out the code in the editor. There's some new syntax there, but we'll bet you can guess what it does. Click Run to see the program in action! (Go ahead and give Ruby an integer—that is, a positive or negative number with no decimal bit.)

Hint
Note: Be sure to give input for the terminal. When it expects input but never receives it an error will be displayed after some time. This prevents it from running indefinitely.

script.rb

```
print "Integer please: "
user_num = Integer(gets.chomp)

if user_num < 0
  puts "You picked a negative integer!"
elsif user_num > 0
  puts "You picked a positive integer!"
else
  puts "You picked zero!"
end
```

If

Ruby's `if` statement takes an **expression**, which is just a fancy word for something that has a value that evaluates to either `true` or `false`. If that expression is `true`, Ruby executes the block of code that follows the `if`. If it's not true (that is, `false`), Ruby doesn't execute that block of code: it skips it and goes on to the next thing.

Here's an example of an `if` statement in action:

```
if 1 < 2
  print "I'm getting printed because one is less than two!"
end
```

Ruby doesn't care about **whitespace** (spaces and blank lines), so the indentation of the print statement isn't *necessary*. However, it's a convention that Rubyists (Ruby enthusiasts) follow, so it's good to get in the habit now. The block of code following an `if` should be indented two spaces.

When you're done with your `if`, you have to tell Ruby by typing `end`.

Instructions

1.
Write your own `if` statement in the editor. It can take any expression you want (even just `true`!), but it should evaluate to `true`. When it does, it should print a string of your choice to the console (using `print` or `puts`).

script.rb

```
if "four".length > "two".length
  puts "The word four is longer than the word two"
  end
```

Else

The partner to the `if` statement is the `else` statement. An `if`/`else` statement says to Ruby: "If this expression is true, run this code block; otherwise, run the code after the `else` statement." Here's an example:

```
if 1 > 2
  print "I won't get printed because one is less than two."
else
```

```
  print "That means I'll get printed!"
end
```

Instructions

1.
Try it yourself in the editor! Use any expression you like in your `if`/`else` statement, but make sure both branches print a string of your choice to the console.

script.rb

```
if "four".length > "two".length
  puts "The word four is longer than the word two"
else
  puts "The word four is shorter than the word two"
end
```

Elsif

What if you want more than two options, though? It's `elsif` to the rescue! The `elsif` statement can add any number of alternatives to an `if`/`else` statement, like so:

```
if x < y  # Assumes x and y are defined
  puts "x is less than y!"
elsif x > y
  puts "x is greater than y!"
else
  puts "x equals y!"
end
```

Instructions

1.
Add an `elsif` block to your `if`/`else` statement in the editor.

script.rb

```
if "four".length > "two".length
  puts "The word four is longer than the word two"
elsif "two".length > "four".length
  puts "The word two is longer than the word four"
else
```

```
  puts "The length of the two words is the same"
end
```

Unless

Sometimes you want to use control flow to check if something is *false*, rather than if it's true. You could reverse your `if`/`else`, but Ruby will do you one better: it will let you use an `unless` statement.

Let's say you don't want to eat *unless* you're hungry. That is, while you're not hungry, you write programs, but if you *are* hungry, you eat. You might write that program in Ruby like this:

```
unless hungry
  # Write some sweet programs
else
  # Have some noms
end
```

Instructions

1.
We've started you off in the editor. Replace the `___`s with the correct `unless` statement code so your program prints out `"I'm writing Ruby programs!"`

script.rb

```
hungry = false

unless hungry
  puts "I'm writing Ruby programs!"
else
  puts "Time to eat!"
end
```

Equal or Not?

In Ruby, we assign values to variables using `=`, the **assignment operator**. But if we've already used `=` for assignment, how do we check to see if two things are equal? Well, we use `==`, which is a **comparator** (also called a **relational operator**). `==` means "is equal to." When you type

```
x = 2
y = 2
if x == y
  print "x and y are equal!"
end
```

you're saying: "if x equals y, print 'x and y are equal!'" You can also check to see if two values are *not* equal using the `!=` comparator.

Instructions

1.
We've got two variables in the editor: `is_true` and `is_false`. Replace the `__` with `==` or `!=` to make `is_true` evaluate to `true` and `is_false` evaluate to `false`.

script.rb

```
is_true = 2 != 3


is_false = 2 == 3
```

Less Than or Greater Than

We can also check to see if one value is less than, less than or equal to, greater than, or greater than or equal to another. Those operators look like this:

- Less than: `<`
- Less than or equal to: `<=`
- Greater than: `>`
- Greater than or equal to: `>=`

script.rb

```
test_1 = 17 > 16
```

```
test_2 = 21 < 30

test_3 = 9 <= 9

test_4 = -11 < 4
```

Practice Makes Perfect

Great work so far! You know what they say: practice makes perfect. Let's try a few more comparators to make sure you've got the hang of this.

Instructions

1.
For this round, we'll show you the comparators and you set each variable to `true` or `false` depending on what value you expect the expression to return. Remember: no quotes around `true` and `false`!

script.rb

```
# test_1 = 77 != 77
test_1 =  false

# test_2 = -4 <= -4
test_2 = true

# test_3 = -44 < -33
test_3 = true

# test_4 = 100 == 1000
test_4 = false
```

And

Comparators aren't the only operators available to you in Ruby. You can also use **logical** or **boolean operators**. Ruby has three: and (`&&`), or (`||`), and not (`!`). Boolean operators result in boolean values: `true` or `false`.

The boolean operator **and**, `&&`, only results in `true` when **both** expression on either side of `&&` are `true`. Here's how `&&` works:

```
true && true # => true
true && false # => false
false && true # => false
false && false # => false
```

For example, `1 < 2 && 2 < 3` is `true` because it's true that one is less than two **and** that two is less than three.

Instructions

1.

Let's practice a bit with `&&`. Check out the boolean expressions and set each variable to `true` or `false` depending on what value you expect the expression to return.

script.rb

```
# boolean_1 = 77 < 78 && 77 < 77
boolean_1 = false

# boolean_2 = true && 100 >= 100
boolean_2 = true

# boolean_3 = 2**3 == 8 && 3**2 == 9
boolean_3 = true
```

Or

Ruby also has the **or** operator (`||`). Ruby's `||` is called an **inclusive or** because it evaluates to `true` when one or the other *or both* expressions are true. Check it out:

```
true || true # => true
true || false # => true
false || true # => true
false || false # => false
```

Instructions

1.

Set each variable to `true` or `false` depending on what value you expect the expression to return.

script.rb

```
# boolean_1 = 2**3 != 3**2 || true
boolean_1 = true

# boolean_2 = false || -10 > -9
boolean_2 = false

# boolean_3 = false || false
boolean_3 = false
```

Not

Finally, Ruby has the boolean operator **not** (`!`). `!` makes `true` values `false`, and vice-versa.

```
!true # => false
!false # => true
```

Instructions

1.
Set each variable to `true` or `false` depending on what value you expect the expression to return.

script.rb

```
# boolean_1 = !true
boolean_1 = false

# boolean_2 = !true && !true
boolean_2 = false

# boolean_3 = !(700 / 10 == 70)
boolean_3 = false
```

Combining Boolean Operators

You can combine boolean operators in your expressions. Combinations like

```
(x && (y || w)) && z
```

are not only legal expressions, but are extremely useful tools for your programs.

These expressions may take some getting used to, but you can always use parentheses to control the order of evaluation. Expressions in parentheses are always evaluated before anything outside parentheses.

Instructions

1.
Last one! Set each variable to `true` or `false` depending on what value you expect the expression to return.

script.rb

```
# boolean_1 = (3 < 4 || false) && (false || true)
boolean_1 = true

# boolean_2 = !true && (!true || 100 != 5**2)
boolean_2 = false

# boolean_3 = true || !(true || false)
boolean_3 = true
```

If, Else, and Elsif

All right! You're all on your lonesome. (Well, not *quite*. We'll just leave this example here.)

```
a = 10
b = 11
if a < b
  print "a is less than b!"
elsif b < a
  print "b is less than a!"
else
  print "b is equal to a!"
end
```

Instructions

1.
Create an `if`/`else` statement in the editor. Make sure to include at least one `elsif`. Each branch of the statement should print something to the console.

Hint
The syntax for `if`/`elsif`/`else` looks like this:

```
if expression
  # Do something
elsif expression
  # Do something else
else
  # Do yet another thing
end
```

script.rb

```
name = "Andres"

last_name = "Bucheli"

if name.length < last_name.length
  print "The name is shorter than the last name"
elsif last_name.length > name.length
  print "The name is longer than the last name"
else
  print "The name and the last name have the same length"
end
```

Unless

Good! Now let's review the `unless` statement.

```
problem = false
print "Good to go!" unless problem
```

Remember, this is basically a short hand `if` statement. It will do whatever you ask `unless` the condition is `true`. In our example, `problem` is `false`, so we don't have a problem. We print `Good to go!`

Instructions

1.
Create an `unless` statement in the editor. The statement should print something to the console.

Hint
Remember, `unless` syntax looks like this:

```
unless condition
  # Do something!
end
```

For `unless`, the `# Do something!` bit will execute if the condition evaluates to `false`.

script.rb

```
name = "Andres"
print "My name is Fernando" unless (name != "Andres")
```

Dare to Compare

Now let's review comparators / relational operators. We've turned the tables a bit!

Remember, comparators need to compare two values to each other to result in `true` or `false`

```
10 > 8 # true
8 > 10 # false
8 == 10 # false
8 != 10 # true
```

Instructions

1.
We're letting you know what value (`true` or `false`) we want each variable to have, and your job is to add an expression that evaluates to the correct value using comparators.

Hint
Remember, comparators are `==`, `!=`, `>`, `>=`, `<`, and `<=`.

script.rb

```ruby
# test_1 should be false
test_1 = 8 == 9

# test_2 = should be false
test_2 = 7 > 14

# test_3 = should be true
test_3 = true == (2*5 == 10)
```

Billions of Booleans

Home stretch! Let's go over boolean operators.

```ruby
( 1 == 1 ) && ( 2 == 2 )  # true
( 1 == 2 ) || ( 2 == 2 ) # true
!( false ) # true
```

1. With `&&` both comparisons on the left and right must evaluate to `true` for the entire statement to return `true`. If the left side does not return `true` it will not bother trying the right side
2. With `||` either the right or left side must evaluate to `true`. If the left side evaluates to `true`, the right side will not be tried because it has met the condition of one side being `true`.
3. With `!` you reverse the result. If you're `false` you're now `true`. if you're `true` you're now `false`! Just think of it as opposite day!

Instructions

1.

The code in the editor indicates what value (`true` or `false`) we want each variable to have, and your job is to add an expression that evaluates to the correct value using boolean operators (`&&`, `||`, or `!`).

script.rb

```ruby
# test_1 should be true
test_1 = true && (2 == 4/2)

# test_2 = should be true
test_2 = (9 == 3**2) || (1 == 3)
```

```
# test_3 = should be false
test_3 = !(6*3 == 9*2)
```
