**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.)

Checkpoint 2 Passed

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