5 Ruby: Conditionals

Programs often need to do different things based on conditional logic. One way to evaluate conditions and control the program is by using `if` statements. `if` statements consider a given condition, and then do something based on the result of the condition. For example, consider the following statement:

```
"If Mary has more than $5, then give her an apple."
```

Let's write a program for this behavior: `can_buy_apple_with?`:

```
# define the method
def can_buy_apple_with?(money)
  if money > 5
    "have an apple"
  end
end

# call the method, passing in a 7
can_buy_apple_with?(7)
#=> "have an apple"

# call the method, passing in a 5
can_buy_apple_with?(5)
#=> nil
```

Remember that in Ruby, the last evaluated line of a method is returned from the method by default. In the first example above, where we call the method and pass a `7`, the last evaluated line is `"have an apple"`. This means that the `"have an apple"` string would be

returned if `money` is `7` (i.e. if the `if` statement evaluates to `true`).

When we pass a `5` to the method in the second example, the `if` condition does not evaluate to `true`, so the method will return `nil`. It returns `nil` because we are not doing anything when `money` is not greater than `5`.

> Always use `end` to close an `if` statement.

Else

If this method was in a real program, returning `nil` if there's not enough money to buy an apple would not create a great user experience. We'd want to tell the user something if they don't have enough money to buy an apple. In the example below, let's offer the user an alternative:

```
def can_buy_apple_with?(money)
  if money > 5
    "have an apple"
  else
    "sorry, how about some gum?"
  end
end

# call the method, passing in a 7
can_buy_apple_with?(7)
#=> "have an apple"

# call the method, passing in a 5
can_buy_apple_with?(5)
#=> "sorry, how about some gum?"
```

In the example above we used an `else` to handle the condition where `money` is not greater than `5`. We didn't need to specify a condition for `else` because it will handle the opposite of the `if`. In other words, when the `if` statement evaluates to `false`, the `else` branch will be executed. In plain English, you could describe the program:

```
"If you have more than $5, you can have an apple; if not you may have some gum."
```

> We still need an `end` to close the `if` statement, but it's placed after the `else` branch.

It's a "best practice" to define an `else` condition for most `if` statements, because an `if` *always implies an* `else`. If we don't specify what happens if the `if` isn't true, a default of `nil` is returned, as in the first example.

Else If

We can expand the logic even further:

```
def can_buy_apple_with?(money)
  if money > 1_000
    "have an Apple computer"
  elsif money > 5
    "have an apple"
  elsif money == 5
    "have some gum"
  else
    "get out of my store"
  end
end

# call the method, passing in a 2_000
can_buy_apple_with?(2_000)
#=> "have an Apple computer"

# call the method, passing in a 6
can_buy_apple_with?(6)
#=> "have an apple"

# call the method, passing in a 5
can_buy_apple_with?(5)
#=> "have some gum"

# call the method, passing in a 4
can_buy_apple_with?(4)
#=> "get out of my store"
```

Using `elsif` (there's no second "e" in "elsif") provides even more options for our program. We still use an `else` statement to handle any condition that doesn't evaluate to `true` in the `if` and `elsif` statements.

`unless` and `!`

`unless` is the opposite of `if`. Use it in simple cases where it makes the conditional easier to read. If you are dealing with a more complex statement, it is better to use a `!` (called a "bang" in programming) to flip the case of the conditional. Consider the following example:

```
def can_buy_apple_with?(money)
  unless money < 1_000
    "have an Apple computer"
  else
    "have an apple"
  end
end
```

As you can see, using `unless` in the example above makes the code a little hard to read. You could refactor using `if` of course, but for the sake of another example you could also use the `!`, as mentioned above. The "bang" basically reads as "not". For example:

```
def can_buy_apple_with?(money)
  if !(money < 1_000)
    "have an Apple computer"
  else
    "have an apple"
  end
end
```

You could read the `if` statement above as:

```
"If money is not less than 1,000, have an apple computer..."
```

> Using a simple `if` and flipping the conditional logic (`if money > 1_000`) is simpler overall, but it's important to understand your options.

One Line Conditionals

You can also use `if` and `unless` at the end of one line statements:

```
gets_discount = true
price *= 0.8 if gets_discount

skip_tax = true
price += price * 0.1 unless skip_tax
```

If `gets_discount` is true (which it is), then you'll discount the price by 20%. Unless `skip_tax` is true, you'll add a 10% tax to the price.

One-line conditionals are helpful, but they can be less clear than their multi-line alternatives.

> When in doubt, favor clarity over brevity.

Clear code is easier to maintain, share, and rewrite. As you will see, it's fun to cram a lot of functionality into a small space — but try to avoid sacrificing clarity for space.

Proper Indentation

Proper indentation becomes **even more important** when handling conditional logic. Before trying the exercises for this checkpoint, read the following code snippet and rewrite it with proper indentation in a message to your mentor.

```
number = rand(25) - 10
if number < 0
puts "The number is negative."
elsif number < 10
if number <= 5
puts "The number is between 0 and 5, inclusive."
else
puts "The number is between 5 and 10, exclusive."
end
else
puts "The number is greater than or equal to 10."
end
```

Compare your solution to the solution below. The indented version is easier to follow than the unindented version.

```ruby
number = rand(25) - 10
if number < 0
  puts "The number is negative."
elsif number < 10
  if number <= 5
    puts "The number is between 0 and 5, inclusive."
  else
    puts "The number is between 5 and 10, exclusive."
  end
else
  puts "The number is greater than or equal to 10."
end
```

5. Ruby: Conditionals

Assignment	Discussion	Submission

Exercises

- ~~If Statements~~
- ~~And / Or~~
- ~~Booleans~~

Solution Walk-Throughs

Do not watch these videos until after you've attempted your own solutions. If you struggle to complete these exercises, submit your best effort to your mentor *before watching a solution walk-through*. Submit your answers even if some tests don't pass, as it's important for your mentor to understand your thought process. Discuss difficult exercises with your mentor.

There are many possible solutions to the exercises. A walk-through is meant to provide insight into the thought process of *one possible solution* per exercise. If your solution is different yet passes all the tests, do not change your solution to match what is seen in the video. Instead, discuss your thought process with your mentor.

1. **If Statements Solution**
2. **And / Or Solution**
3. **Booleans Solution**

assignment completed

?