codecademy

Control Flow in Ruby

`elsif` Statements in Ruby

In Ruby, an `elsif` statement can be placed between `if` and `else` statements. It allows us to check for additional conditions.

More than one `elsif` can be placed between `if` and `else`.

```
print "enter a number: "
num = gets.chomp
num =  num.to_i;

if num == 5
  print "number is 5"
elsif num == 10
  print "number is 10"
elsif num == 11
  print "number is 11"
else
  print "number is something other than 5, 10, or 11"
end
```

Ruby not Operator

The `!` (not) operator in Ruby flips a boolean value. If a value is `true` then applying `!` to the value changes it to `false` and vice versa.

Else statement in Ruby.

In Ruby, an `if` statement evaluates to either `true` or `false`. The code indented after the `if` portion is executed for `true` while the code indented after the `else` portion is executed for `false`.

```
if number > 50
  print "number is greater than 50"
else
  print "number is not greater than 50"
end
```

Comparison operators in Ruby.

The following *comparison* or *relational* operators are used in Ruby to compare values.

`>` - greater than; `<` - less than; `>=` - greater than or equal to; `<=` - less than or equal to; `==` - equal to

Or operator in Ruby.

The `||` (or) operator is a logical operator which returns `true` if either of the expressions on left-hand side or right-hand side is `true` .

```
grade1 = 50
grade2 = 30
grade3 = 80

if grade1 > grade2 || grade1 > grade3
  puts "Grade 1 is not the lowest score!"
end
```

`if` Statement in Ruby

An `if` statement in Ruby evaluates an expression, which returns either `true` or `false` . If the expression is `true` , Ruby executes the code block that follows the `if` whereas if the expression is `false` , Ruby returns nil.
In this example, the string `"Your condition was true!"` will print because the condition `number == 10` is true.

```
number = 10
if number == 10
  puts "Your condition was true!"
end
```

And operator in Ruby.

`&&` is a logical operator in Ruby which evaluates to `true` only if both expressions on either side of `&&` evaluates to `true` .

```
if score1 > score2 && score1 > score3
  print "Score 1 is the greatest in
value."
else
  print "Score 1 is not the greatest in
value."
end
```

Unless statement in Ruby.

An `unless` statement in Ruby is used to evaluate an expression. If the expression evaluates to `false` , then the code following `unless` is executed.

```
#This construct requires a "number"
variable to be less than 10 in order to
execute:
print "Enter a number"
number = gets.to_i
unless number > 10
  puts "number is less than 10."
end
```