Open in new tab

Conditional Logic

Lesson Duration: 60 minutes

Learning Objectives

- To understand how to use conditional operators to control behaviour
- To understand how to check equality
- Exposure to if statements, case select, guard, ternary

What Is Conditional Logic

We often make decisions about whether things are true or false in our daily lives.

> Ask students to think of a question that has a yes no answer, e.g. “Is Craig Hungry?” “Is it raining today?”

When we think about these statements and come to a yes / no answer we are evaluating them. We take in a statement about the world and then give a yes / no answer.

Our programs work in exactly the same way. We can give them a “statement” just like what we have been doing, and it will “evaluate” it to true or false (yes or no).

We use a special operator in Ruby to check if things are equal. If you remember, `=` on it’s own *assigns* a variable. Two `==` together checks if two things are equal.

Examples of statements that evaluate to true or false:

```
# terminal
irb

# IRB

4 == 2 + 2
#=> true

4 == 2 + 3
#=> false

"cat" == "cat"
#=> true

"cat" == "dog"
#=> false
```

Note that equality needs both the *type* and the *value* to be equal

```
"1" == 1
false
```

How do we fix this?

```
"1" == 1.to_s
"1".to_i == 1
```

Control Flow

Similarly, we might make decisions based on whether something is true or false.

> Ask students to think of examples where something happens in your daily life based on the outcome of one of these yes/no stamements.

If it’s raining, I’ll take the car to work. Otherwise, I’ll cycle. If I catch a Pikachu I’ll be happy, otherwise I’ll be sad.

Programming languages also do this to allow us to have different actions and paths our code can go down, otherwise our program could only do one thing and that would be sad.

The easiest way to see this is to try it! Let’s have a go.

```
# terminal
touch what_animal.rb

atom .

# what_animal.rb
p "What animal are you?"
name = gets.chomp
if (name == "chicken")
        p "This is my favourite animal!"
else
        p "Sad, not my favourite."
end
```

Note that we indent the code as we take actions depending on the outcome of the `if`. Ruby doesn’t need this but it makes it much easier for a human to read. It’s important to learn the indentation conventions of a language.

Sometimes, we must do certain things with indents - Python won’t work if your tabs are in weird places!

```
# terminal
ruby what_animal.rb
```

We don't need the brackets in Ruby around the condition, but since we are just learning we are going to keep them to prepare ourselves for the languages we will learn later.

What happens if we want to add a condition for a kitten?

```
# what_animal.rb
p "What animal are you?"
name = gets.chomp

if (name == "chicken")
  p "This is my favourite animal!"
elsif (name == "kitten") # UPDATED
  p "I love kittens!"
else
  p "Sad, not my favourite."
end
```

Note that if we want this to be case insensitive we need to lowercase the input.

```
# what_animal.rb
p "What animal are you?"
name = gets.chomp.downcase

if (name == "chicken")
  p "This is my favourite animal!"
elsif (name == "kitten") # UPDATED
  p "I love kittens!"
else
  p "Sad, not my favourite."
end
```

Let's make a little program that asks you to guess when Ruby was created.

> Give students a few minutes to try this. Give them a hint that the correct answer should be 1995!

```
p "When was ruby created"
guess = gets.chomp

if (guess == 1995.to_s)
  p "Whoo you win!"
else
  p "Nope"
end
```

When Choices Are More Complex

`if` statements work very well for one or two conditions, but when there are multiple conditions they can get a bit messy. We tend to shy away from `elsif`s.

Instead we can use `case` statements (in some languages, these are called `switch` statements).

```
# terminal
touch case_statement.rb
```

```
# case_statement.rb
score = 10

case score
  when 10
    p "genius"
  when 9
    p "merit"
  when 8
    p "pass"
  else
    p "fail"
end
```

```
# terminal
ruby case_statement.rb
```

We might want to have a range of values in our case, so 10 is genius and 8 to 9 is merit, 5 to 7 is pass and everything else fails.

```
score = 6

case score
  when 10
    p "genius"
  when 8..9
    p "merit"
  when 5..7
    p "pass"
  else
    p "fail"
end
```

We have a lot of duplication here with the `p`. We also might want to save the result of the `case` to a variable and do something with it.

```
score = 6

result = case score
  when 10
    "genius"
  when 8..9
    "merit"
  when 5..7
    "pass"
  else
```

```
    "fail"
end

p result
```

[Task:]

- Add a condition for 4 => “resit”
- Ask the user to input the score instead of hard coding it

Control Flow Nice To Haves

When we have simple statements to evaluate, `if` statements can look a bit cumbersome. Ruby has the concept of guards which can neaten things up.

```
score = 6

result = "fail"
result = "pass" if (score >= 6)

p result
```

Ternary

We can also use a construct called a ternary. It’s really nice when you only have a simple choice to make that only have 2 outcomes.

```
score = 6
result = score > 5 ? "pass" : "fail"
p result
```

You can see it takes three arguments, hence why it’s *ternary*. The first is the condition to check, the second (after the `?`) is the action to take when true the third (after the `:`) is the action to take if false.

if students are struggling with this it might be worth pointing out that this is the same as writing an if…else…end statement.

One Last Thing…

Sometimes we want to check if one or more conditons are true. To do this we need to use something special.

```
#terminal
touch more_conditionals.rb

#more_conditionals.rb

craig_hungry = true
craig_tired = true

p "Craig is hangry!" if craig_hungry && craig_tired

craig_tired = false
p "Craig is grumpy!" if craig_hungry || craig_tired
```

We use `&&` for AND and `||` for OR.

Published with GitHub Pages