Introduction

Now that we've learned a little bit about how numbers and logic statements work, we're going to write a short program that invites the user to guess a randomly chosen integer from `1` to `100`.

Create a new file called *guess_the_number.rb*:

```
number = rand(1..100)
puts "I'm thinking of a number between 1 and 100, try and guess what it is"
guess = gets.to_i
attempts = 1
until guess == number do
  if guess < number then puts "Too small, try again"
  elsif guess > number then puts "Too big, try again"
  end
  attempts += 1
  guess = gets.to_i
end
puts "Well done, you guessed my number in #{attempts} attempt#{'s' if attempts > 1}!"
```

In the first line we use the `rand` method with a range of 1 to 100 as the argument to choose a random integer between `1` and `100` and store it in a variable called `number`. This will be the number that the player needs to guess in the game.

Next we use the `puts` method to output a string explaining what the player has to do.

Then we use the `gets` method to obtain a guess from the player, and store it in a variable called `guess`. Even if the player enters a number, it will still be stored as a string, so we need to add the `to_i` method at the end, to ensure that the value is treated as an integer.

In the next line we create a variable to keep track of the number of attempts the player has had. We set it to `1` initially, since this is the minimum number of guesses that a player can take.

Next comes the main part of the program. `until guess == number do` is actually an example of a loop, and we'll be looking at these in more detail in the next lesson. Inside the block is the actual code that will keep running during the game.

Inside we have are a number of `if` and `else` statements.

The Loop

So first of all, we have an `if` statement that says if the guess that the player makes is less than the number, then we'll use `puts` to output a message. It says the number is too small.

Then we use the `elsif` statement to check if the guess is bigger than the number. If it is, we'll output the message, "it's too big, try again".

After we've given the feedback about the guess, we're going to increase the value of the attempt variable by `1`, because obviously, the player has had another attempt. `+=` is used to increment the value of a variable.

Then we use `gets` again to ask for another value and this will be stored in the guess variable.

Then the whole process starts again, until the player guesses correctly. Once that happens, the program will break out of the whole loop and jump to the last line of the program. There we use `puts` to show a little message that says "well done, you guessed the number correctly". It will also say how many attempts were done, using string interpolations.

Also notice that we have a neat way of using an `if` statement to pluralize the number of guesses, if it was more than `1`.

String Interpolations

What this is saying is add an `s` on the end if the number of attempts is bigger than `1`, otherwise this won't run. Remember, when you're using interpolation you don't actually just have to use it to insert variables into a string. You can actually evaluate Ruby code inside the curly braces as well, which we're doing here.

The `s` character will only show if the variable `attempts` is bigger than `1`. So that's a nice little method to remember for future.

Running the Program

Now open terminal, type

```
ruby guess_the_number.rb
```

and observe the result!