How Ruby Handles Logic

Now we're going to have a look at how Ruby handles **logic**.

Create a new file *logic.rb* with the following code:

```
number = rand(1..6)

puts "You Rolled a #{number}"
```

Here we use the `rand` method to create a random number from `1` to `6` to model a number rolled on a die. Then we're using the `puts` command to output this string, which employs interpolation to say which number was rolled.

KeyWords

Now let's have a look at some logical statements that we can use to output some information about the number depending on if certain conditions are met.

For example, let's check if the number is `6`:

```
puts "You rolled the highest number possible" if number == 6
```

The `if` command comes before the condition that has to be `true`, if we want this message to be displayed.

Notice that we're using two equals signs to test for quality. This is different from what we did at the beginning, where we only used one equal symbol to assign a value to a variable.

We could also use a similar structure to test if a condition is not met:

```
puts "You didn't roll the lowest number possible" if number != 1
```

The `!=` operator means "not equal to". You'll only get the message, if that condition is not met.

Bang Method

We can also put the condition at the beginning:

```
if number < 5 then puts "You rolled a number less than 5" end
```

The < operator means “less than” (strictly, so 5 isn’t actually included). We also use another keyword, `then`, and next we put the code that we want to run, if this condition here is met. Lastly we have to finish with an end command just to say that the `if` statement has finished.

Writing Multiple Conditions

We can also write out multiple conditions on multiple lines:

```
if number.even?
  puts "You rolled an even number"
else
  puts "You rolled an odd number"
end
```

`else` is used to say what we want to happen if the condition isn’t met.

Elsif

Using `elsif` we can have as many “if-else” conditions as we like.

```
if    number==1 then puts "You rolled a one"
elsif number==2 then puts "You rolled a two"
elsif number==3 then puts "You rolled a three"
else puts "You rolled a number bigger than three"
end
```

Case Statement

There is another statement we can use, which is a `case` statement:

```
case number
  when 4 then puts "You rolled a four"
  when 5 then puts "You rolled a five"
  when 6 then puts "You rolled a six"
  else puts "You rolled a number less than four"
end
```

This works in a very similar way to the example above, but we use `case` instead of `if`.

The important thing here, is we put the variable `number`, because in this `case` statement all the conditions refer to the value of this variable. Like with `if` statements, we need to finish with an `end` to say that the `case` statement has finished.

Else

Also note that inside `case` statements we can use `else` as well. If none of the statements inside this block of code happen, then we just want to say, "You rolled a number less than four".

The `case` statement just a little bit neater than `elsif` and saves you writing.

If Statements

We can also chain conditions together:

```
if number== 2 or number==3 or number==5
  puts "You rolled a prime number"
end
```

The code will run if either of those three conditions are `true`.

There's an alternative way of writing such conditions:

```
puts "You rolled a square number" if number==1 || number==4
```

The `||` operator represent the word `or`.

Double Pipe Symbol

There are some slight differences in using the `||` and `or` operators, but they are essentially doing the same thing.

There's also the `and` operator that returns `true` only if both statements are `true`:

```
if number.odd? and number >= 4
  puts "You rolled a five"
end
```

Greater Than

>= represents “greater than or equal to”, so 4 will be included. Because we’re also testing if the number is odd and use and, then both of our conditions have to be true.

There’s also a symbol version of and operator that is written as &&:

```
if number.even? && number <= 3
  puts "You rolled the only even prime number"
end
```

Again, there are some subtle differences between these.