Numbers in IRB

We're going to have a look at using **numbers** in IRB. Open up your terminal and start up IRB.

For example, we can type

```
4 + 5
```

to sum those numbers.

What might not be as clear at first though, is that `4` and `5` are both `Integer` objects, and the plus symbol is actually a method.

Dot Syntax

We saw in the previous lesson how to call a method using the dot syntax, and you can do the same here:

```
4.+(5)
```

So, `4` is the object calling the method, the plus sign is the name of the method, and `5` is an argument. This however doesn't look much like a sum, so Ruby lets us write this in a more natural way, just like how we would normally write arithmetic. This is known as **syntactic sugar**, where a method can be written in a nicer looking way.

Arithmetic Operations

Ruby has lots of other arithmetic operations, such as subtraction (`-`), multiplication (`*`) and division (`/`).

Float

Have a look at this example

```
9 / 2 # returns "4"
```

This is incorrect, because `4.5` should be returned not `4`. This is because `9` and `2` are both integers, and this means that the solution to `9 / 2` will only give the integer part of the answer.

To get the correct answer, we need to use a different type of number object, called a **float**. To use floats instead of integers, all we need to do is write `9` as `9.0` instead:

```
9.0 / 2
```

Now Ruby returns the correct answer of `4.5`.

Odd or Even

Ruby also gives us methods to find out if a number is odd or even:

```
42.even?
33.odd?
```

These are examples of **boolean methods** that only return the results of `true` or `false`, and they will usually have a question mark character on the end, which makes them easier to spot.

GCD method.

Ruby can also do some quite cool mathematical stuff, like finding the greatest common divisor of two numbers using the `gcd` method. So, if we wanted to know the largest integer that divides into both `12` and `20`, we would write:

```
12.gcd(20)
```

LCM Method

There is also a similar `lcm` method, which finds the lowest common multiple of two integers:

```
15.lcm(20)
```

to_s method

One last integer method that we're going to look at is the `to_s` method, with stands for "to string". It takes an integer and converts it into a string representation of that integer:

```
7.to_s
```

This allows us to treat the number like a string so that we can use the string methods on it that we learned in the last lesson.

to_i Method

There is also an equivalent method for strings called `to_i`, which turns a string into an integer:

```
"42".to_i
```

You have to be careful with this method, though, because if the string doesn't actually contain any numbers, then it will just return `0`:

```
"Hi".to_i
```

However, if you enter a string that starts with a number value, then it will return just that number from the beginning:

```
"221 B Baker St".to_i # returns 221
```

Rand method

It's often useful to generate a random number. This is easy to do in Ruby using the `rand` method. This will generate a floating point number between `0` and `1.0`:

```
rand
```

If you provide an integer as the argument to this method, then it will generate a random integer between `0` and an integer up to the argument provided, but not including that argument:

```
rand(6)
```

This going to return a random number from `0` to `5`, so if you instead wish to return a number from `1` to `6`, you'll have to add `1`:

```
rand(6) + 1
```

However Ruby provides a much nicer way of generating a random number from `1` to `6`. What we can do is enter a range of values as the argument to the `rand` method:

```
rand(1..6)
```

This means "choose a random number from 1 to 6, inclusive".