Introducing Ruby

Ruby is an incredibly powerful and flexible programming language. It is a modern language - created in the 1990s by Yukihiro Matsumoto (Matz). A large part of his design philosophy is centered around optimizing for developer happiness - making a language that is a pleasure to program in. Its simple syntax makes it ideal for first-time programmers.

Ruby is a popular language among startups, due to its simplicity and the fast development time it allows - AirBnB, GitHub, Twitter, Groupon have all used Ruby to build their sites.

In this course, we'll be learning the basics of Ruby, in a web development context. We will be using a simple web framework called **Sinatra**.

RubyGems

RubyGems is a package management system used to install and manage software packages written in Ruby. We will be using RubyGems to install things like Sinatra and any other dependencies we want to make use of. You should already have RubyGems installed as per the course preparatory work.

Using Ruby interactively: irb

```
roberts-mbp:~ Rob$ irb
>>
```

You can simply type irb (short for *interactive Ruby*) inside your Terminal / Command Line and hit enter to find yourself in an interactive Ruby session

Copyright 2017 Entrepreneur First CIC (T/A Code First Girls) - *Not to be circulated without prior consent from Code First: Girls*

You can start typing some Ruby commands - for example you can use Ruby's puts command:

```
roberts-mbp:~ Rob$ irb
>> puts 'Hello, world!'
Hello, world!
=> nil
>>
```

You can also do some maths - give it a go!

```
>> 5 + 5
=> 10
>> 2 * (10 + 3)
=> 26
>> 2**4
=> 16
>>
```

In a nutshell, you can interactively code in Ruby, without having to write a program, save it and run it. It's especially useful when you want to try something out.

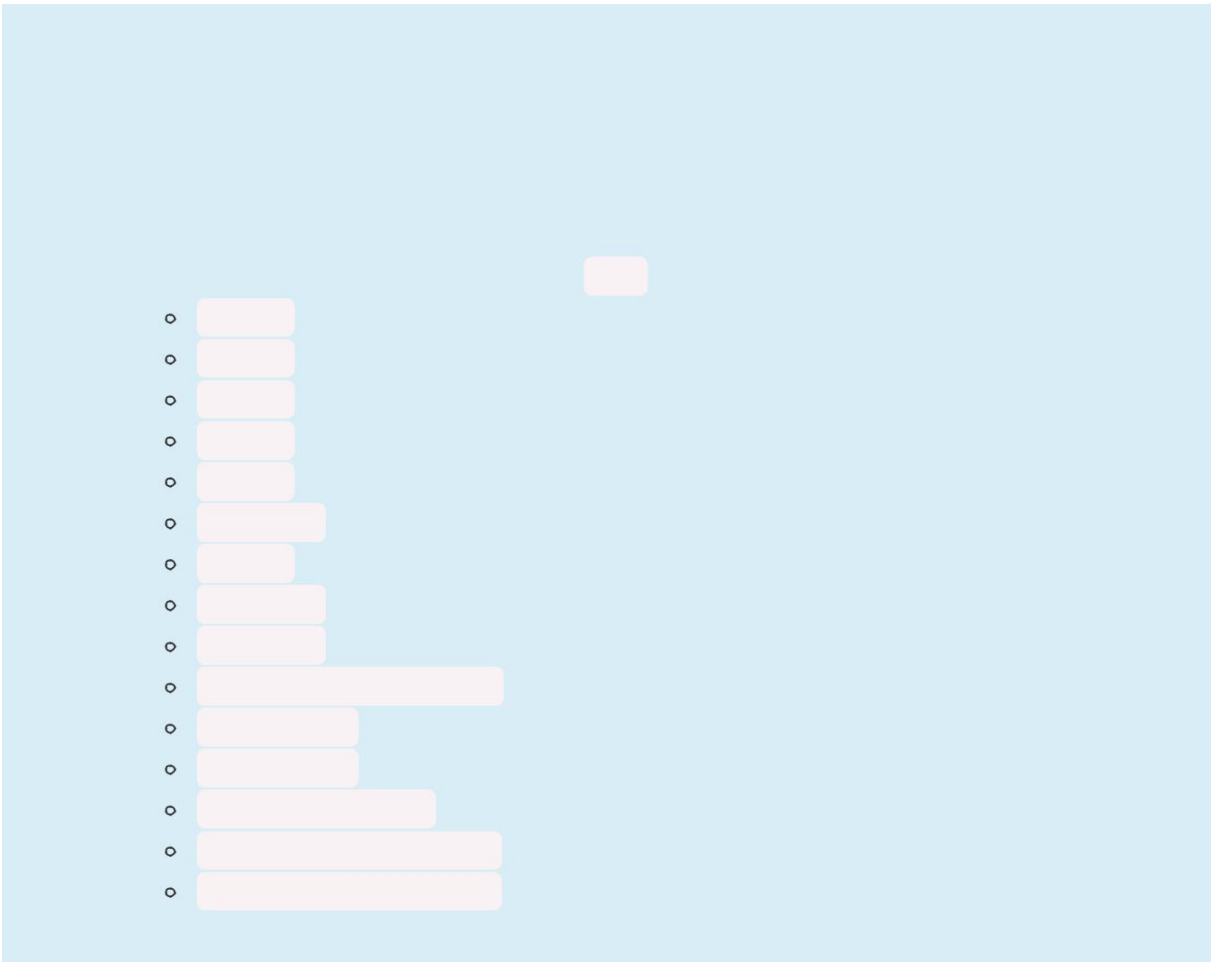
Comments

In Ruby, any part of a line that comes after a # is ignored. This is useful when you are writing complicated programs, as it allows you to write human-readable comments to document your code - this makes it easier for others to follow your code.

You can see comments being used in action:

Copyright 2017 Entrepreneur First CIC (T/A Code First Girls) - *Not to be circulated without prior consent from Code First: Girls*

```
roberts-mbp:~ Rob$ irb
>> puts 'Hello' # This just prints out hello, nothing more!
Hello
=> nil
>> puts 'Hello,' + ' World' # Oh look, we can add strings!
Hello, World
=> nil
>>
```



Task:

1. Pair up with someone
2. Open an interactive Ruby session on one of your laptops
3. For each of the following Ruby expressions, try to agree what value you think they

   evaluate to. Check if you are right in irb .

   1 + 2

   5 - 6

Copyright 2017 Entrepreneur First CIC (T/A Code First Girls) - *Not to be circulated without prior consent from Code First: Girls*

```
8 * 9

6 / 2

5 / 2

5.0 / 2

5 % 2

2.even?

3.even?

"hello " + "world"

"Bob" * 3 "Bob" +
3 "hello".upcase

"GOODBYE".downcase

"hello".capitalize
```

Task Summary

Before we move on to variables, we'll quickly look at a few of the things that you found out in the previous exercise:

```
> 6 / 2
=> 3

> 5 / 2
=> 2

> 5.0 / 2
=> 2.5
```

Copyright 2017 Entrepreneur First CIC (T/A Code First Girls) - *Not to be circulated without prior consent from Code First: Girls*

```
> 5 %
2 => 1
```

If you give Ruby integers (whole numbers), it will do integer division.

For example, `5 / 2` gives `2` as it is the largest whole number of times you can remove 2 from 5. The partner to integer division is the remainder `5 % 2`, giving `1`. Together these two operations tell you that 2 goes twice into 5 with remainder 1. If you give Ruby decimals (floats) it will do normal division.

```
>>
2.even?
=> true
```

The line above is actually pretty special. We've just asked the value `2` a question and it's responded. In programming terms, this works because `2` is an *object* - something that contains both data and *methods* that can query/manipulate the data. In this case, `even?` is the method that we called.

```
> "hello " + "world"
=> "hello world"

> "hello".upcase

=> "HELLO"
```

Here you met another type of value: `"hello"` is a string. As you see here, you can add strings together. Like all Ruby values, strings are also objects and therefore have methods. Here we used the `upper` method, to change the string into uppercase. You can find out more about the methods that strings have in the Ruby docs.

Copyright 2017 Entrepreneur First CIC (T/A Code First Girls) - *Not to be circulated without prior consent from Code First: Girls*

```
>> "Bob" + 3

TypeError: can't convert Fixnum into String
from (irb):1:in `+'
```

Turns out that you can't add a string to an integer. Have another read of the message that irb gave you. Can you figure out what it is saying? When something goes wrong, Ruby tries to be as helpful as it can (and show where the problem was).

Learning to interpret the errors that Ruby gives is an important part of learning to become a programmer.

Names & Variables

So far we have used irb as a clever calculator, by working with values directly. Programming becomes a lot more powerful when you're able to give values names. A name is just something that you can use to refer to a value. In Ruby you create a name by using the assignment operator, = .

```
age = 5
```

You can change the value associated with a name at any point. It doesn't even have to be the same type of value as the first value assigned to the name. Here we change the value associated with age to a string:

```
age = "almost three"
```

Variables should start with a lower-case letter.

String interpolation

String interpolation is a way of taking a variable and putting it inside a string. To write a string in Ruby you can either use ' (*single quote*) or " (*double quote*).

Copyright 2017 Entrepreneur First CIC (T/A Code First Girls) - *Not to be circulated without prior consent from Code First: Girls*

```
string1 = 'hello'
string2 = "hello"
```

In the code above, string1 and string2 are exactly the same.

```
age = 5

like = "painting"

age_description = "My age is #{age} and I like #{painting}."

#=> "My age is 5 and I like painting."
```

So what just happened here? Essentially, `#{}` let us insert (or *interpolate*) our variables into into the surrounding string.

As well as variables, we can also add some Ruby code:

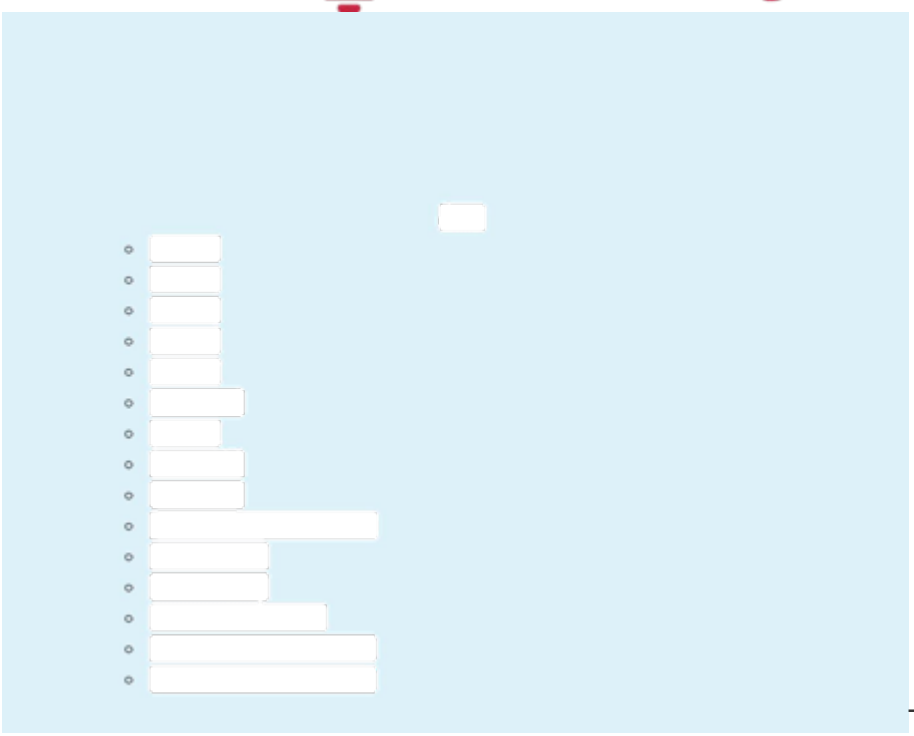
```
age_description = "My age is #{age * 4} and I like #{painting.upcase}."

#=> "My age is 20 and I like PAINTING."

"3 times 2 is #{3 * 2}"

#=> "3 times 2 is 6"
```

Copyright 2017 Entrepreneur First CIC (T/A Code First Girls) - *Not to be circulated without prior consent from Code First: Girls*



Task:

With your pair, decide what each of the following instructions will do. Test to see if you're right in irb .

```
a = 1 a
+ 1 a

a = a + 1 a

b = "hello" b

c = b.capitalize b

c

d = "hello"

e = d.capitalize d

e

name = "Dave"

f = "Hello #{name}!" f

name = "Sarah" f

f * 5
```

Copyright 2017 Entrepreneur First CIC (T/A Code First Girls) - *Not to be circulated without prior consent from Code First: Girls*

Task summary

```
> x =
1 => 1

> x = x +
1 => 2
```

This might seem really obvious, but it's worth pointing out: = is an assignment operator; it means 'set name on the left equal to the value on the right'. It isn't the same equals as you see in maths! In maths x = x + 1 doesn't really make sense - it's an equation with no (finite) solutions. In Ruby x = x + 1 makes perfect sense - just set x to be one more than it was before.

```
> name =
"Dave" =>
"Dave"

> f = "Hello
#{name}!" => "Hello
Dave!"

> name = "Sarah"

=>
"Sarah"
>> f

=> "Hello Dave!"
```

Copyright 2017 Entrepreneur First CIC (T/A Code First Girls) - *Not to be circulated without prior consent from Code First: Girls*

The above shows that string interpolation happens when you write it down. When you first write `f = "Hello #{name}!"` Ruby immediately looks up `name` and bakes it straight into the string. Setting `name` to something different later on won't change this.

Homework

- Finish all exercises from today's class
- Make sure everything we went through makes sense
- Head over to this online exercise book and work your way from Exercise 2 up to and including Exercise 10

  Get excited for our next class

Copyright 2017 Entrepreneur First CIC (T/A Code First Girls) - *Not to be circulated without prior consent from Code First: Girls*