Open in new tab

What is Programming?

Lesson Duration: 90 minutes

Learning Objectives

- Know what a programming language is
- Understand the difference between a class and an object
- Be able to run Ruby code in IRB on the command line

What Is a Programming Language?

Our task as programmer is to communicate thoughts/ideas/structures to a computer.

> Draw image of communicating thoughts to a computer

Let's imagine we live in a world with no programming languages. We have to talk to our computer in machine code. 0010101001010. This would not be fun. It would be hard to write and hard to read. Expressing high level ideas would be almost impossible.

Thankfully we live in the lovely world where lovely people have created programming languages, and more importantly interpreters to speak to the computer at a low level language.

> Break down computer into CPU and language interpreter.

> Draw robot on whiteboard

The programmer can then speak in the more expressive/readable/maintainable high level language. Examples of this languages are Ruby, Python, Javascript, Java, C, C++, C#, Swift, Go, Pascal, Scala….

Ruby

The language we are going to start using is Ruby was created in 1995 by Yukihiro Matsumoto

> "I hope to see Ruby help every programmer in the world to be productive, and to enjoy programming, and to be happy. That is the primary purpose of Ruby language."

Why Teach Ruby?

Ruby has many traits which are transferable to other languages.

- Structure similar to many other languages (class based object orientated language)
- Community tends to follow best practices (e.g. how code is tested)
- Clean (everything is an object)

Working with Ruby

We all have Ruby installed on our laptops.

```
# terminal

ruby -v
```

`irb` will open an Interactive Ruby environment in the console (we exit it by typing `quit` or `exit`).

`irb` stands for "Interactive RuBy", and typing that into a terminal launches a ruby REPL.

- REPL = Read, evaluate, print, loop => good for experimenting with short" snippets of code.

`irb` allows to quickly try snippets of ruby outside of your application.

Use it frequently to test lines of code or to experiment outside of the bigger programs we will write later.

Let's write our first Ruby program

```
# irb

"hello"
```

So what was going on here?

Objects

Ruby is an Object Oriented language. 'Everything is an object'.

Here our Ruby interpreter created an object to represent text for us.

Ruby is class based object orientated language. All objects are instances of classes. The class that represents text is called String. We just created an object of the String class. We can find out the class of an object by asking it.

```
# irb

"hello".class
```

The idea of classes and objects is the key to object orientated programming. It is a concept that can be hard to grasp. Let's look at some examples.

> It's useful to use a real word analogy here to explain class vs object. eg Person class, can create person objects (such as us)!

> Optional depending how you get on, in Text Editor write up example of Classes and corresponding Objects, get the class to create them.

Ruby provides a number of Classes for us out the box, we can then create objects from them.

Where did this String class come from? We can produce these objects because Ruby provides these classes out the box.

```
# irb

"Hello"
"Good Day"
"Howdy"
```

> Show on diagram we are creating multiple objects of the same class.

Learning to program in an object orientated language like Ruby is a combination of understanding how we construct programs (much like grammar) and how we can use the Classes it provides for us (much like vocabulary)

Everything in Ruby is an object, and objects have a "class" (a 'type'). There's is a set of types that we can use in Ruby which make up the bedrock of our programs.

Types

We need a way to communicate ideas to our Ruby robot. Maybe we want to make some text, do a numerical calculation or something else useful to us. To achieve this, we need some basic building blocks and we call these "types". The basic types are just classes that come out of the box with Ruby.

String

The String Class is given to us in Ruby and we can use it to represent text. We are given a special syntax for creating Strings in Ruby, using single or double quotes. There are differences in these that we will cover later.

```
# irb

"some text in here"
"some text".class
```

Integer

Integer objects are ready for us to access using the numbers. We are given methods on these objects.

```
# irb

24
13.class
```

> Versions of Ruby before 2.4.0 will return 'Fixnum', which has since been deprecated (along with Bignnum).

Variables

Assigning objects to variables

Lets create another string object

```
# irb

"This is a string object I have told Ruby to create"
```

How can I get this string back? We create this string but it was lost to us immediately. To hold on to a string we need to assign a variable to it.

```
#irb

my_string = "A brand new string object"
```

We create the object as before, but this time we hold a reference to it and call it my_string.

> Draw on board the variable referencing the object. Show this for each example Can use analogy of cloakroom and ticket here.

We can then access the object using the variable we have assigned to it.

```
# irb

my_string
```

We can store values in "variables" - they're variable because the values can change (we also call this 'mutability')

```
# irb

my_string = "Another String I have defined"
my_string
```

The name of a variable is almost totally up to you to define. Pick a name, and tell it what value it equals:

Have a play with assigning objects to variables. Eg

```
# irb
```

```
character = ""
age = 24
```

Variable Naming Conventions

Ruby has fairly strong opinions about how you should write your code, but very little in the way of enforcement. So you're free to break the rules if you wish, with no real worry about consequences…

… apart from tutting and disapproving headshakes from your peers.

Keeping to style guides means that other developers can more quickly understand your code (and you theirs), and contribute quickly to a code base without making it hard for others to follow.

We use **CamelCaseForClasses**, ****, and ****lower_case_snake_case_for_everything_else**.

Ruby style guide naming conventions

BREAK

Using Objects

We've seen objects, and holding on to them with variables. We've created String objects to represent text, and Integer objects to represent numbers.

Okay nice, but we often want to do things with our objects.

Let's create two numbers.

```
# irb

number_1 = 22
number_2 = 12
```

We can find the sum of this using the + operator

```
#irb

number_1 + number_2
```

Again we have just lost this value! What can we do with it? Yes, store it to a variable.

```
#irb

sum = number_1 + number_2
sum
```

Integer has many operators we can use. Try using -, *

Strong Typing

What do you think will happen if we do this?

```
#irb

1 + "2"
1 + "2".to_i()
```

String

Combining Strings

Strings also have the + operator this will combine the strings.

```
# irb

word_1 = "hello"
word_2 = "world"
sentence = word_1 + word_2
sentence
```

Calling Methods

The objects created by the out the box Ruby class have a bunch of 'things' we can do to them.

```
# irb

my_string = "hello"
my_string.length()
```

These things are called methods. (Again what is actually going on will become clearer when we start writing our own classes next week) We call a method on an object using . and then the name of the method we want to call. Let's see useful methods on strings.

To see all the methods available on a String check out the docs.

String class

Summary

We have learned about Classes, and how Ruby provides useful Classes out the box. How we can use Classes to create objects, and how we can assign objects to variables so we can reuse them. We have seen that objects have methods on them which we can call using dots(.)

Creating objects from classes, and calling methods on these objects is what we as Object Orientated programmers will spend most of our time doing. It may seem strange as first but it will become second nature to us.

Further Practice

If time, practice creating objects, assigning variables, calling methods.

Create a String object, assigning a variable and call a method we haven't used yet. Create an Integer object, assigning a variable and call a method we haven't used yet.

Published with GitHub Pages