a beginner's guide to:

RUBY

g.brown

skillstopractice.com

WARNING: This is not meant to be read like a book.
Instead, treat it like a system of interconnected thoughts.

It’s meant to be explored, tinkered with, and ruminated over.
To be traced, retraced, and bounced around within.

The important ideas covered here will show up again and again, with increased nuance the deeper into the materials you go.

Allow them to slowly and joyfully unfold.

Ready? Let’s get into some code.

-g

BGR v0.0.1 0x01 – 0x0F

0x01 – A <RUBY PROGRAM> SAYS “HELLO WORLD”

0x02 – A <RUBY PROGRAM> TELLS YOU YOUR LUCKY NUMBER.

0x03 – A <RUBY PROGRAM> GETS REPETITIVE.

0x04 – A <RUBY PROGRAM> GETS *VERY* REPETITIVE.

0x05 – A <RUBY PROGRAM> VISITS THE FARMER’S MARKET.

0x06 – TO BE REVEALED

0x07 – TO BE REVEALED

0x08 – TO BE REVEALED

0x09 – TO BE REVEALED

0x0A – TO BE REVEALED

0x0B – TO BE REVEALED

0x0C – TO BE REVEALED

0x0D – TO BE REVEALED

0x0E – TO BE REVEALED

0x0F – TO BE REVEALED

0x01 – A <RUBY PROGRAM> SAYS “HELLO WORLD”

```
puts "Hello World"
```

→

```
Hello World
```

We use a METHOD named puts to display a STRING on the CONSOLE by writing to a STREAM called STDOUT (“Standard Output”).

METHOD A useful programmed action that runs when called by its name.

(This often also involves passing along some additional info that the METHOD will make use of in some way to do its job)

STRING A free-form message which can be almost anything.

(i.e. plain English, specially formatted text like a phone number or email address, or even arbitrary data that’s machine-readable only.)

CONSOLE The software tool that lets you run system commands and see their results. This is where you typically run Ruby programs from.

STREAM Something computer programs can read from and write to that allows them to interact with the outside world.

Streams can be used to work with files, to send and receive information over the internet, and also to interact with developer via their CONSOLE.

STDOUT The “Standard Output” stream. Anything written to it will show up on the developer’s CONSOLE.

0x02 - A <RUBY PROGRAM> TELLS YOU YOUR LUCKY NUMBER.

```ruby
puts "Your lucky number is #{rand(1..100)}."
```

```
Your lucky number is 42.
```

Q1 There are two METHODS that are run in this tiny Ruby program, what are they?

Q2 Which of the two METHODS runs first?

Q3 What additional information does each METHOD make use of in order to do its job?

Q4 What is the purpose of the #{ } notation in the STRING shown in this code sample?

Q5 If the #{ } notation was left out and the string was written as “Your lucky number is rand(1..100)” - what would show up on the CONSOLE after this program ran?

RANGE

Represents a span of values between a lower and upper limit.

E.g. (1..100) means “from 1 to 100”

#{ }

This notation allows you to run some code and then take its end result and stick it into a STRING.

The technical term for this handy STRING building technique is Interpolation.

0x03 — A <RUBY PROGRAM> GETS REPETITIVE

```ruby
rand(3..5).times do
  puts "I like foxes."
end
```

Q1 What three METHODS are used in this program?

Q2 What will be displayed on the CONSOLE when this program is run?

Q3 What is the lower limit of the RANGE used in this program?

Q4 What is the upper limit of the RANGE used in this program?

Q5 What is the purpose of the `do  end` notation?

E1 Update the program so that it ends by saying: “And that’s all you need to know for now.”

(This should be shown only *once* per run)

E2 Use what you’ve learned to write a small program that simulates rolling a six sided die ten times.

BLOCK

A chunk of code that can be passed along to a METHOD in order to do something useful. The METHOD gets to decide if and when the code in the BLOCK gets run, and how many times to run it. The METHOD also decides what information to share with the BLOCK (if any), to help it get its job done.

A BLOCK can be defined by using `do  end` to mark where its code begins and ends.

It's also possible to use `{  }` to define blocks, and while the two approaches are not *exactly* identical in how they work, in most cases you can treat them like they're just two different ways of doing the same thing.

NOTE: BLOCKS ARE ONE OF THE MOST IMPORTANT FEATURES IN RUBY, AND MUCH OF THE LANGUAGE IS BUILT AROUND THEM!

The name, in part, hints at how they're meant to be reusable building blocks that you can create pretty much anything from.

0x04 - A <RUBY PROGRAM> GETS *VERY* REPETITIVE

```ruby
3.times do
  puts "Apple"

  2.times {
    puts "Banana"
    puts "Cake"
  }

  0.times {
    puts "Dragonfruit"
  }

  puts "Eggs"
end
```

Q1 How many BLOCKS are defined in this code sample?

Q2 How many times in total will the word Apple appear on the CONSOLE when this code is run?

Q3 How many times will the word Banana appear on the CONSOLE before the word Cake shows up for the first time?

Q4 How many times will the word Banana appear on the CONSOLE before the word Eggs shows up for the first time?

Q5 What will the full final results of this program look like on the CONSOLE after it runs?

0x05 — A <RUBY PROGRAM> VISITS THE FARMER'S MARKET.

```ruby
items_in_stock = ["Apples", "Tomatoes", "Eggs", "Pints of Milk"]

puts "We sell #{items_in_stock.count} types of items"

items_in_stock.each do |item|
  puts "We have #{rand(20..40)} #{item} in stock"
end
```

Q1 What four METHODS are used in this program?

Q2 What will be the first line of text shown on the CONSOLE when this program is run?

Q3 What is the purpose of the `[ , , , ]` notation used on the first line of this program?

Q4 How many lines of text in total will be shown on the CONSOLE after this program runs?

Q5 If `items_in_stock` was renamed to `items_for_sale`, would the program's behavior change or would it stay the same?

Q6 Which METHOD is in this program is making use of a BLOCK? And what is that METHOD using the BLOCK to `do` ?

Q7 What does `item` refer to in this program?

Q8 What will the full final results of this program look like on the CONSOLE after it runs?

OBJECT

The basic “raw material” in a Ruby program, that allows you to model real world problems and develop useful programs to solve them.

Ruby provides many types of OBJECTS to serve as a starting point for building programs.

(I.e. a STRING is a type of Ruby object, and so is a RANGE, but each serves a very different purpose in your programs.)

You can also create your own OBJECTS by starting with the ones Ruby provides, and then combining them to model something more specific to your program.

ARRAY

A list of OBJECTS, stored in a particular order:

```
[1,2,4,8,16,32]
```

```
[“Cat”, “Dog”, “Fish”]
```

Because an ARRAY is itself an OBJECT, you can also create an ARRAY made up of other ARRAYS:

```
[ [12, “Cats”], [3, “Dogs”], [13, “Fish”] ]
```

NOTE: Ruby is a list processing language, and so you’ll find that it provides many useful METHODS for working with ARRAYS as well as other types of collections.