Lesson 2: Intro to Ruby - Data Types, Variables, & Methods

# Overview

1. Explain the basic **characteristics** of the **Ruby** programming language
2. Work with **Interactive Ruby Shell (IRB)** to interact with Ruby and begin running short programs
3. Review several important Ruby **data types** and how to manipulate / combine them. For now, these will include integers, floats, and strings.
4. Explore the importance of working with **variables** within Ruby
5. Dive into some important **methods** and their purpose within Ruby

[Overview](#h.6vn3xj1m6iz1)

[Characteristics of Ruby](#h.3h8kou9y52yr)

[Let’s Get Started with IRB](#h.b7s5i4oplb7x)

[Data Types](#h.7mjcf6vh2zn1)

[Strings](#h.hcduh9a4ov0f)

[Numbers](#h.i4yjprlgwsoq)

[String Arithmetic](#h.15brmkmwmrih)

[Data Type Conversion](#h.r57y0ej37qa2)

[We Do: Imperial to Metric Program](#h.5i1jfemhwed3)

[Variables](#h.hxbgr8amadv1)

[We Do: Imperial to Metric Program](#h.efjg973xauf9)

[Gets](#h.ywyug18om4az)

[Methods](#h.pc4y519y6xcb)

[We Do: Imperial to Metric Program](#h.afg0da4m2s4f)

[Homework](#h.cdb6i3wca8uz)

**Review Homework**

# Characteristics of Ruby

Ruby is an **object-oriented language**. Everything within a program can be represented in Ruby as an object. In OO programming, computer programs are designed by making them out of objects that interact with one another.

Furthermore, any object-oriented program created with Ruby will contain **classes** and **objects**. A class is the blueprint from which individual objects are created. You can look at it like this: all students in this session are of a **class** called “students.” Each student is an **instance** of the **class** of **objects** known as students.

Can anyone think of another example of classes, objects, and instances?

What are the 2 Commandments of Ruby?

* Thou shalt use convention over configuration.
* Thou shalt keep code D.R.Y.

# Let’s Get Started with IRB

Open your terminal, and type “**irb**”.

IRB = Interactive Ruby Shell

puts “Hello world”

type **exit** to exit

# Data Types

## Strings

“Hello World” is what is called a **string**, a data type frequently used in Ruby. A string is a way of representing a sequence of characters. Show more examples of strings (“my name”, “1234”).

Strings must be in double or single quotes. Note that “1234” is not a number but a string. We can’t perform math on it in a numerical sense.

## Numbers

Ruby has 2 types of numbers - integers and floats (floating point numbers).

Does anyone remember what an integer is?

Which of these are integers and which are floating points?

1

34.5

100.304

1.0

1234

Now let’s do some math. Use +, -, \*, /. IRB is like a calculator. You can perform these functions in Ruby.

## String Arithmetic

Ruby has a fun characteristic in that you can perform some arithmetic on strings. For example:

“hello” + “sia” → This is call concatenation.

“sia” \* 5

1 \* 5 versus “1” \* 5

How would you make IRB print out “RubyRubyRubyRuby”?

## Data Type Conversion

What happens when we try to add a number to a string? (5 + “dollars”)

It errors out. We can only add objects of the same data type.

Luckily, Ruby gives us another way. We can convert numbers to strings with “to\_s”

5.to\_s + “dollars”

Here are the common data type conversion methods:

to\_s = to string

to\_i = to integer

to\_f = to float

Can someone think of a command we could type using to\_i in a useful way?

## We Do: Imperial to Metric Program

Navigate to your desktop or home directory and create a folder named “projects” (whichever you prefer). Inside that folder, create a folder called “tts\_ruby”. Cd into that folder.

How would we create a file named “imperial\_to\_metric.rb” and open in Sublime?

Type “puts 72 \* 2.54” and save.

Go back to terminal and type “ruby imperial\_to\_metric.rb”. What happened?

Now let’s think about our program. We want to get someone’s height and weight and then tell them their height in centimeters and their weight in kg. Can someone break that problem down into English?

(use comments to type in the instructions - explain that # is a comment in Ruby)

# Variables

In Ruby, **variables** are memory locations which hold any data used within a given program. Practice in IRB:

“a = “hello”

b = “world”

a

b

How would we combine hello and world the long way and the short way? “hello” + “world”, a+b, puts a + b

Now you can see why variables are awesome!

## We Do: Imperial to Metric Program

How would we replace 72 with a variable?

a = 72

puts a \* 2.54

Run it in terminal again. We get the same output!

Take a minute with a partner and answer the questions:

* What would happen if we used “72” instead of 72?
* How would we fix this?

Share out answers (to\_i).

Great! Now let’s create variables for a person’s name, their height in inches, their weight in pounds, in to cm conversion factor, lbs to kg conversion factor (0.453592), and finally their height in cm and weight in kg. Go around the room and ask one person each what to type.

user\_name = "Sia"

height\_in = 68

weight\_lbs = 150

in\_to\_cm = 2.54

lbs\_to\_kg = 0.45

height\_cm = height\_in \* in\_to\_cm

weight\_kg = weight\_lbs \* lbs\_to\_kg

Now, how do we output to the terminal screen? Puts! We can use concatenation to string everything together!

puts "Hello " + user\_name + ", your height in cm is " + height\_cm.to\_s + ", and your weight in kg is " + weight\_kg.to\_s

Run the program again. If you have an error, work with your neighbor to figure out what could be wrong. Resolve any remaining errors together.

## Gets

That was really cool, but wouldn’t it be even better if we could ask someone for their info? We can!

puts = put string to the screen

gets = get string from the command line

Let’s go back to our program and change:

user\_name = gets

Run it again. What happened? It captured the return at the end of typing. Fortunately, the chomp method chomps off that return for us:

user\_name = gets.chomp

Let’s also make it more user-friendly by adding a prompt: puts “What is your name?” before user\_name = gets.chomp

Yay!

Classroom challenge (pair program): Use gets and chomp to get and return the user’s height and weight also. (Hint: we will also need one more method that we learned today).

## Center

# Can center header/line by using this method and putting in number of pixels

header\_size = 60

puts “~” \* header\_size

puts “Welcome to this program!”.center(header\_size)

puts “~” \* header\_size

# Methods

Methods, also known as functions in some programming languages, are used to bundle one or more repeatable statements into a single unit. .to\_s is a method we learned.

What are all of the methods we have learned so far?

1. .to\_s
2. .to\_i
3. .to\_f
4. .chomp
5. .center(#)
6. gets
7. puts

Even puts and gets are considered methods. This is because they all must be applied to some sort of input (i.e., argument) in order to produce any meaningful output.

We can find more about methods in the Ruby documentation. Google search “ruby (string, integer) methods”. Do this in class.

Use IRB to play with methods - like a = “sia”, a.reverse, a.reverse!

## We Do: Greeting

Create a file called practice.rb. Cycle through something like this, but start with no arguments, then add arguments:

def greeting

puts "Hello Sia"

end

greeting

Then add arguments:

def greeting name

puts "Hello #{name}! How are you?"

end

greeting("Cory")

b = "sia"

greeting(b)

## You Do: Imperial to Metric Program

Now let’s create some of our own methods! Maybe in the future we want to use these methods multiple times in one program.

Create a method for converting inches to centimeters.

All methods require a **def** and an **end**. If you want to pass in an **argument**, use the parentheses to pass it in. In Ruby, we don’t explicitly **return** something from a method. We just call that operation last. Finally, we have to actually call that method in order to use it.

# [Homework](https://docs.google.com/document/d/1SdXSdW4zCwttoFQ4c2UfWh-Ut5pum85frMElPz_NLP8/edit?usp=sharing)