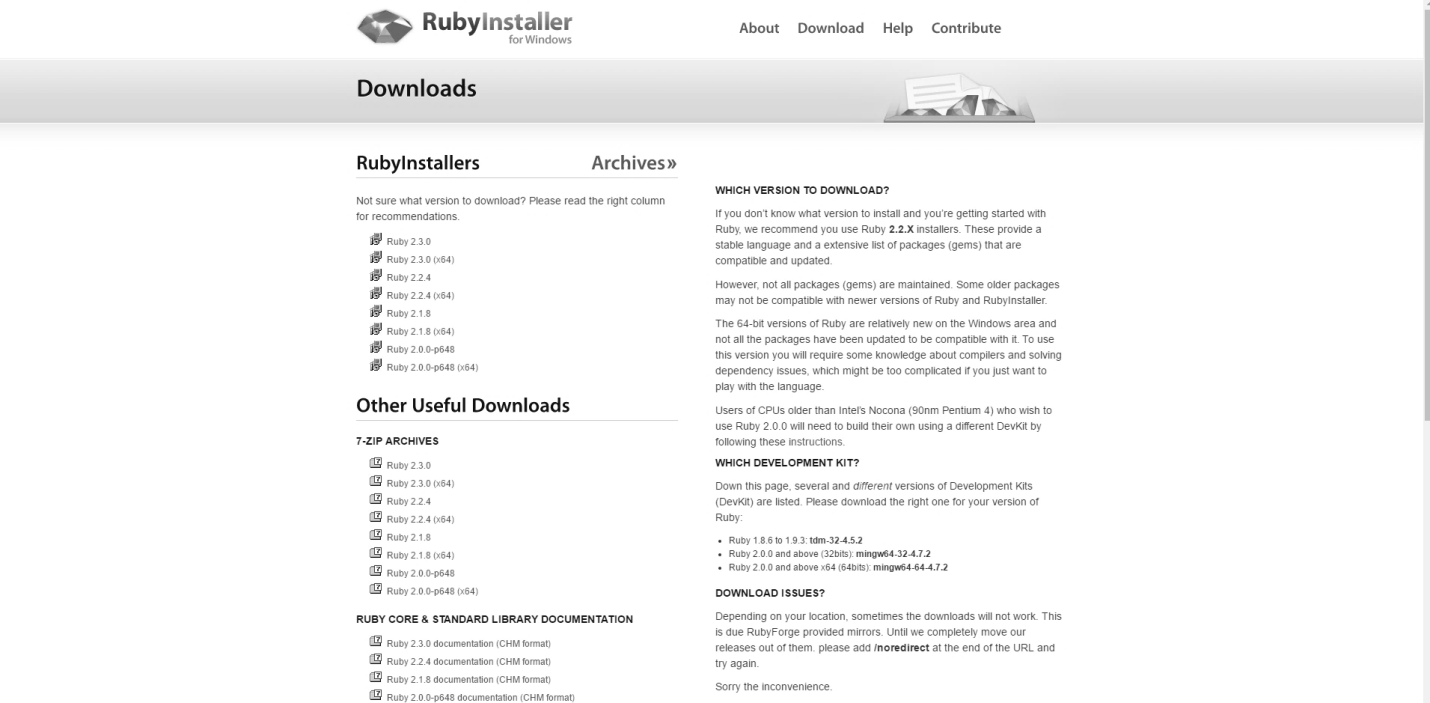
Put down your drills, lay down your chisel, and switch off that head lamp because it’s a different Ruby we’re looking for. The Ruby language may seem foreign or unfamiliar, but it is known as one of the easiest languages to read and write. So gear up … wait, actually don’t gear up, because you’re about to learn everything you need to know and more about Ruby.

**How to obtain Ruby:**

Mac OSX: It’s your lucky day, because your Mac computer came with Ruby already installed! You may have to do some installing eventually, but for now you’re good to go.

Windows: Don’t listen to what I said to those Mac users. Installing Ruby isn’t that bad. Just go to [http://rubyinstaller.org/](http://rubyinstaller.org/%20) and choose the latest build. From there just open the installer, and complete the steps.

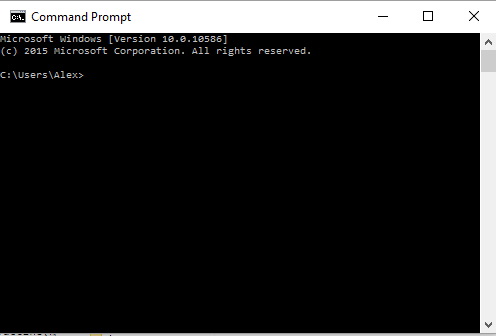


How does that old saying go? “Download Ruby and you’re half way there.” Something like that I think.

**How to use Ruby:**

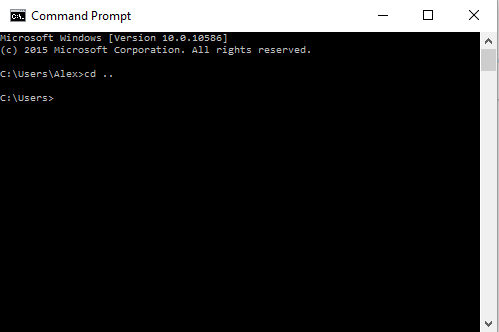
There are two ways to use Ruby. One is through files and the other is directly in your command prompt/terminal. The in-prompt is a sandbox where you can quickly try out code without having to save it. For projects and code you would like to save, Ruby can be executed from a file.

Before we start using Ruby from the command line, I’ll give you a quick overview of how to use it. For Windows users, go to the bottom left of your computer screen and click on Start. Then search ‘cmd’ and press enter. This should open up.

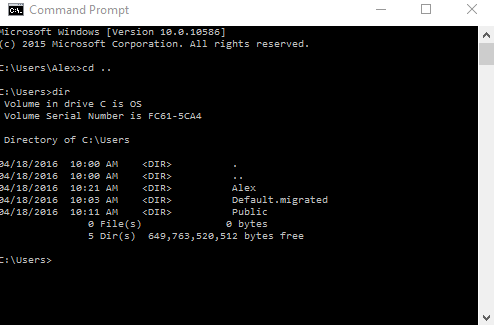


For Mac users, go to your Launch Pad (swipe all five fingers inward if you’re using the trackpad) and search on the top bar for Terminal.

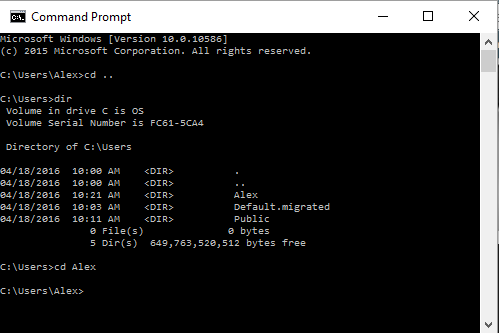
Notice the bit before the ‘>’. This will tell you what folder you are currently in. To change what folder you are in, you can navigate in two ways. First, type ‘cd ..’ . This will make you go back a folder and in my case, it will bring me to the ‘Users’ folder.



Now try typing ‘dir’ (for Mac type ‘ls’). This command will show you what files and other folders are located inside your current folder.



To change your directory (go into a folder inside of your folder), type ‘cd’ followed by the name of the folder.



Let’s start off by opening our command prompt and typing ‘irb’ and pressing enter. Now, you won’t be transported to Narnia, but it may be just as good. You should see this: ‘irb(main):001:0>’

Introduce yourself by typing ‘puts “Hello World” ’ and pressing enter. To exit the session, type ‘exit’ and press enter. That’s all we’ll do for now, but feel free to play around and don’t worry, you won’t break anything because nothing gets saved.

## Now you are about to write your first ever Ruby program. Open your favorite text editor like Sublime Text, Brackets, NotePad++. Open your text editor and create a new file using the file tab at the top. Type ‘puts 1+1’ in your file. Save the file, by using the file tab and clicking save. Name the file ‘first\_script.rb’ and save it in any folder you would like. Switch back over to the command prompt and change directory to the folder where you saved your script. Type ‘ruby first\_script.rb’. You should see the number 2 being displayed. Congratulations! You wrote your very first Ruby script.

(insert first\_script.rb)



**The Basics:**

Whether Ruby is your first language or your tenth, learning it will not be difficult. The basics of Ruby have much in common with other languages. Because Ruby is an object-oriented language, much of it can be defined as classes and objects. Before we get too far, let’s talk about what classes and objects are.

A class is a code template or a blueprint for a particular thing. It includes things like properties and functions. Properties can be thought of as adjectives and functions can be thought of as verbs. A property describes something and a function does something. But how do we use a class? That is where an object comes into play. An object is an *instance* (example) of a class. For example, human would be a class because it is a blueprint and a person would be an object because it is an instance of a human. Now that we cleared that up, let’s continue.

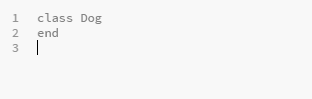
Take, for example, when you wrote ‘puts “Hello World” ’. The output was a string containing that text. A string is simply a sequence of characters (numbers and letters). But let’s look at this through Ruby-colored glasses, and examine what happened. The ‘Hello World’ was an object, an instance of a string. All we did was display the string, but there are a number of other functions we could have applied to that string object. You might be asking what the ‘puts’ is all about. For now you just need to know that puts is used to turn anything into a string and display it as an output. If you must know though, ‘puts’ is a function of the IO (input/output) class.

There are loads of Ruby classes and even more methods (the Ruby term for functions) that you can check out at <http://ruby-doc.org/core-2.3.1/>. You can use this site to learn more about the subjects I talk about, or you can go explore on your own.

**The Fundamentals:**

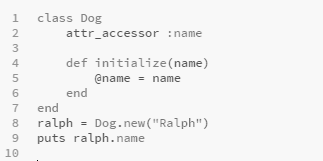
I’m going to approach these fundamentals in an Object Oriented manner, which means everything will be expressed through classes and objects.

Start off by saving a file as ‘dog.rb’ . You can save it anywhere; just make sure you are in the same directory in your command prompt. Start off by making a dog class:



Now let’s create an instance of a dog …

Wait a minute, how can we have a dog without a name? Let’s write some code to give him a name. We don’t want all dogs to be named Spot, so we’ll use the variable ‘@name’ as a place holder. You may be wondering what the ‘@’ is. When placed in front of a variable during assignment, it is known as an *instance variable*. By adding the ‘@’, it will allow us to call the variable throughout our dog instance of Ralph. The initialize method is a preset Ruby method that will run when the object is created. The ‘attr\_accessor :name’ piece I included is extremely important. It allows us to read or change the name of the dog after he is created.

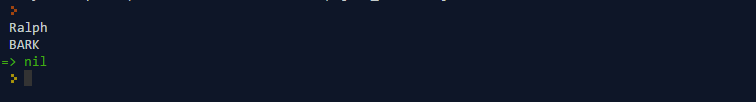


(Insert dog\_1.rb)

Before we move on, let’s put on our programming caps and think about something. Let’s say someone gives you a $20 bill. Can that person change the amount in your wallet? Under the right circumstances, yes they can. In programming, we would say they can *write* to our wallet. But can that same person view the money we have in our wallet? No way! In this case, we don’t want the person being able to *read* from our wallet. If we created a class of a wallet, we would use ‘attr\_writer :money’. This would allow us to have the scenario we just described in which people can write to your wallet but not read from it. The opposite of this would be ‘attr\_reader :money’, where people can read from your wallet but not write to it. Use ‘attr\_accessor :money’ to allow people to both read and write.

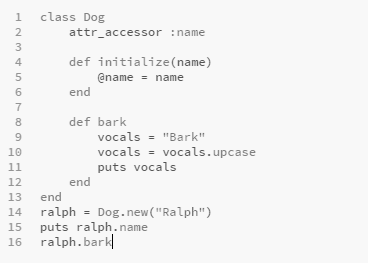
Whew, that was a lot of information. Now let’s put it to use. Does a dog know his own name? Probably. Also, can an owner of a dog change the dog’s name? That sounds reasonable. So it looks like ‘attr\_accessor’ fits our dog the best. To finish off, write the code to display Ralph’s name below the dog class.

Now let’s make our dog come to life. First we will create a method called bark. Then create a variable called ‘vocals’ and set it equal to “Bark”. We don’t need the ‘@’ here because this variable will not be called or changed outside the class once the dog is first created. Below your ‘vocals’ variable, write the code to display our vocals. Down below your Dog class, write ‘ralph.bark’. Here we are calling our method which will execute the code inside. You should see Ralph bark when you run the program.



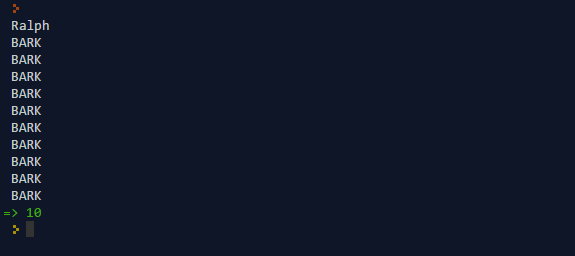
(insert dogbark.png)

What if the mailman just drove by the house and Ralph is *really* excited? We want to give him a super loud bark. Below your vocals variable, let’s use our first Ruby method to make our string be all uppercase. Try running the program again and see what happens. This is one of the many Ruby methods that you can use.

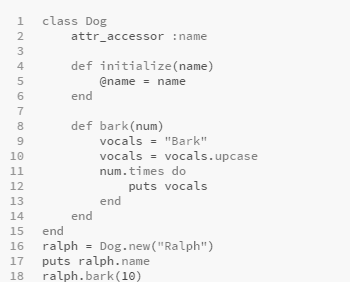


(Insert dog2.rb)

Let’s say one day your friend comes over and Ralph barks 15 times. With our current code, we’ll have to write ‘ralph.bark’ 15 times and keep adding more if he continues to bark. Instead, let’s use something pretty cool in Ruby to make our bark method flexible. In front of the bark we put parentheses and the variable ‘num’ inside. This is how to pass variables into a method with Ruby. In place of our ‘puts vocals’, we are going to *iterate* through the number we passed in. All this means is that we are going to do something the number of times equal to the number we passed in. We simply use the ‘.times do’ method which will make our dog bark the number of times we would like him to bark. We just have one last thing to do, which is to pass the number into Ralph when we tell him to bark. Try out the code one more time. It should look like this.

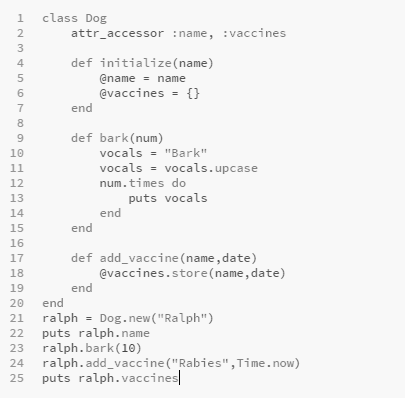


Now if I could only control my dog’s barking like that. … There are many ways to loop such as the ‘while’ loop, the ‘until’ loop, the ‘for next’ loop, along with others. You can look these up on the Ruby docs.



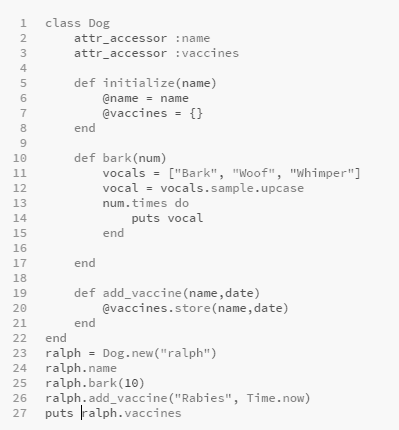
(Insert dog3.rb)

One day, Ralph goes to the vet and realizes he needs kept track of his medical history. Ralph needs a way to store all the vaccines he gets. If we use a string, there is no possible way to store this type of information. Instead we’ll use a *hash*. A hash is similar to a dictionary where we each key corresponds with one value. For example, a dictionary has a word as the key and the definition as the value. For our vaccine variable, we’ll use the name of the vaccine as the key and the date as the value. Let’s make our vaccine variable, an instance variable this time because we need to read and write to it outside of our dog, and allow ourselves to read and write to it. All have to do to add a vaccine is write an ‘add\_vaccine’ method, pass in our name and date, and then use the store method to save our new vaccine. Try running the program and see how it turns out.



(Insert dog4.rb)

Ralph isn’t that smart, but he has a few different things he can say. We did him a horrible disservice by only allowing him to bark. To solve this, we will create an array of things he can say. An array simply stores a set of data (strings, numbers, hashes, or even arrays). Let’s add an array of the different things Ralph can say into the bark method. To rotate through them, all we have to do is set vocals equal to a random thing to say by using the sample method. Try running the program twice to make sure it changes each time. Notice how I wrote “vocals.sample.upcase”. Ruby methods can be compounded and used right after one another which makes Ruby a very powerful language.



(Insert dog\_final.rb)

Now that our dog model is complete, there are a few things to clear up. I referred to Ralph the whole tutorial as if he was the only dog. However, each time the program runs, a different dog is created. It just so happened we named each one Ralph. The other is that you need to remember there is an infinite amount of ways to create a dog class. Feel free to make it your own and explore.

**The Next Steps:**

If this was your first time seeing Ruby, I would advise learning Ruby some more before going into its applications. A good place to go would be <http://tryruby.org>. For those of you who are intermediate or experts, try and see if you can do any of these problems: <https://projecteuler.net/archives>.



Whenever you feel comfortable with the language of Ruby, the most logical next step would be to learn Ruby on Rails. It is a framework for making interactive websites and was used to make websites like Twitter. A great place to learn Rails is our Ruby on Rails Beginner course right here on Learn to Program <https://learntoprogram.tv/courses/ruby-on-rails>.