Ruby Is For Fun

Self-Study Course, From Absolute Beginner to Advanced

Roman Pushkin

Ruby Is For Fun

Self-Study Course, From Absolute Beginner to Advanced

Roman Pushkin

This book is for sale at http://leanpub.com/rubyisforfun

This version was published on 2018-10-06

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Front Matter

In lieu of an introduction

In the 21st century, programming has become one of the most important sciences in any economy. Processes which used to take place without the aid of computers have been partly or completely optimized. Businesses and private individuals have realized how useful electronic machines are, and the age of a flourishing IT industry has begin.

Certain specific trends have formed within all this variety of technologies. The most convenient tools for carrying out particular tasks have been determined. Programming languages have undergone significant changes. It is not as easy for the ordinary reader to understand all these languages and technologies as it might appear at first glance.

It became obvious at a certain point that “programmer” is one of the professions of the 21st century. But how do you become a programmer? In which direction should you apply your efforts? What needs to be studied, and what does not? What is the most efficient use of your time in mastering any technology?

Before answering these questions, the most important question of all must be answered. Why is it necessary to become a programmer? What’s the sense of it?

Some will want to become a programmer to develop micro-programs for inter-continental ballistic missiles and the space industry. Some will want to become a programmer to create their own games. Some will want to learn programming in electronic tables to calculate taxes more efficiently.

But the purpose of this book is more mundane. The author assumes that to the question “Why do I need to become a programmer?” will give the answer “to earn money as a programmer.” Such an answer is usually given by people who have already tried some profession and want to use their time more efficiently to earn good money.

They could also be young people forced to keep up with the times and to learn technologies, and the quickest way of getting a result from such knowledge, as quickly as possible. Furthermore, this means a result not only in the form of the knowledge of how to write this or that program, but a result in terms of money.

Knowledge of any field in programming assumes a knowledge of the basics of the language, along with elementary theory (which is different for each field), basic concepts and definitions, and also a knowledge of tools other than the basic ones (such as an operating system, utilities and auxiliary programs).

There is a vast number of fields. They include game development, scientific research of various kinds, data processing and analysis, web programming, programming for mobile devices, and so on. It is not possible to be a specialist in all these at the same time.

So the person starting, or wishing to start, on programming faces a choice. Where to apply? What to study?

If you are a scientist at a research institute, your choice will most likely fall on the language “python” or “C++”, since a large number of libraries for data processing and analysis exists for these languages.

If for example, you work as a watchman and are quite contented with your work, you could study some exotic programming language not much required in the market, simply to avoid being bored.

If you live in a society in which ever-increasing accounts are paid every month, where you need to think not only of today, but of tomorrow, your choice will be different.

You will need to study something in high demand very quickly, to find work as soon as possible.

The language “Ruby”, used in web programming, is something between “finding work quickly”, “studying something simple and interesting” and “will come in useful in the future”. Ruby not only enables you to compile boring programs while working in an office, but can also be useful at home, in everyday life. (One of my recent programs was about teaching how to play the guitar).

Furthermore, the philosophy of the language itself assumes that the study and use of programs will not be boring. For example, one of the principles of the language is the Principle of Least Surprise, which is explained as follows: “whatever you do, you will most likely succeed”. That is inspiring, you must agree!

Other programming languages also exist. I am not saying they are bad, not at all. Each language is good for a certain purpose. But let us remember our own task and make the comparison with some other languages.

Ruby vs. Ybur

The language “Ybur” is Ruby in reverse. It is an exotic programming language which no-one knows except me. I’ve only just thought it up and I don’t know what it does. Let us compare Ybur and Ruby using the three parameters described above.

Finding work quickly

Ruby is a very popular language, it is easy to find work where it is used. Ybur? No-one’s ever heard of it, finding work using it is impossible.

There is no need to compare the other parameters. In other words, if what is important to you is not programming in itself (though that’s no bad thing either), but

the possibility of earning money in the foreseeable future, Ruby is not a bad choice. It is quite a popular language. Of course, other popular programming languages exist too. We might say that JavaScript is the most popular. But let us compare JavaScript and Ruby.

Learning something simple and interesting

Ruby incorporates the Principle of Least Surprise, and that is not at all bad. JavaScript was not initially created in accordance with this principle. It is more complicated than Ruby, because it is completely asynchronous (you'll have to take my word for that for the time being).

We can show that JavaScript is not as simple as it looks at first glance. Let us consider a Ruby program for sorting numbers:

Example: Simple program to sort four numbers in Ruby

```
[11, 3, 2, 1].sort()
```

The above program has to sort the numbers 11, 3, 2 and 1 into rising order (it is not important for now if you don't understand this syntax, we'll deal with that subject later). The result of the Ruby program's work is: 1, 2, 3, 11. No surprise there! But let us write the same program in JavaScript:

Example: Incorrect program to sort four numbers in JavaScript

```
[11, 3, 2, 1].sort();
```

In this case, the syntax is very similar, and differs only in the semicolon at the end. But what will the result be? Even experienced JavaScript programmers cannot always give the correct answer. The program's results are quite unexpected: 1, 11, 2, 3. Why this is so is a question of history. But to sort numbers in JavaScript, you have to write:

Example: Correct program to sort four numbers in JavaScript

```
[11, 3, 2, 1].sort((a, b) => a - b);
```

Once you understand it, it isn't difficult. But the question is something else. Do you want to waste time on such fine points in the initial stage? JavaScript is much in demand, and every Ruby programmer must know it at a minimal level. But I must say I would want a *very great deal of money* to be a full-time JavaScript developer.

Could come in useful in the future

JavaScript is developing very dynamically. Knowledge gained ten years ago is not always up-to-date (in this case I am speaking of popular frameworks - sets of tools). In Ruby's case, the Rails framework has existed for more than ten years. Knowledge gained ten years ago is still applicable.

Incidentally, it is worth making a separate comment about the applicability of knowledge. Knowledge of shell-scripting languages is still applicable. Little has changed in more than 30 years. Knowledge of the basics of Computer Science is still applicable in interviews and at work (this knowledge has hardly aged at all). So it is definitly something you want to learn at some point of time.

But no-one can make precise predictions about the applicability of a particular *programming language* in the future. However, one may look as the statistics for recent years. At the time this book was being written, Microsoft bought GitHub, written in Ruby, for 7.5 billion dollars. In other words, the language is in fine form today and widely used. Updates are being issued, the speed and syntax are being improved. And the number of available libraries makes a rapid solution of virtually any problem possible (within the framework of the field called web programming).

Ruby is for fun

In our opinion, programming language should not only solve certain business problems, but should also be easy enough to use every day without problems.

For example, Java is a fine tool for solving business problems. But it has to be treated with respect. The language is statically-typed (we'll come back to this topic). The type of data on which different operations are carried out must be specified. This takes time, and is fully justified in the business field, where it is better to spend several times as long on development, rather than having to pay for mistakes later.

In the case of Ruby, the program can be written quickly, in a simple way. It is not all that reliable (which can be a problem sometimes), but many companies, particularly startups, have come to the conclusion that it is *reliable enough*, and the relatively slow speed of program execution is not a problem either. After all, in today's world we often have to do something quickly to get quick investments, to attract the first customers, take advantage of momentum while competitors are still trying to find their way.

From the author's personal point of view, Ruby is a good tool for doing something of my own - some project of mine, a program which can be shared with others, attract attention or earn money.

In other words, Ruby is an efficient and interesting language, not only for work but for its own sake too.

What we will study

As stated earlier, there are many trends in software development. Each trend is unique and requires its own know-how. The author believes that there are currently at least two "tried and tested" trends in programming which give the maximum result

in the minimum time. “Results” here means both cash compensation and knowing how to do something with your own hands.

The first is mobile development – programs for mobile phones (Android, iPhone), tablets (iPad) and other devices.

The second is web programming (or web development).

However, mobile development itself often means optimizing the code for mobile devices in all sorts of ways. The programming language and SDK (software development kit) is very often bound up with a certain style of development. And this style is very different from the *classic* object-oriented programming, it is more procedural. With procedural programming, you can’t always make full use of the language’s capabilities, though this is not always important, particularly if your aim is to earn a salary.

A second aspect of program development for mobile devices is that there are two main mobile platforms at the present time. One belongs to Apple Corporation, the other to Google. How these platforms will develop in the future depends entirely on the policy of these companies.

In the case of web programming in Ruby, it all looks somewhat different. The language itself is being developed and supported by the programmers themselves. The web framework Rails – about which more later – is also supported exclusively by the community. This enables programmers from all over the world to create a convenient tool just as they want, without having to look over their shoulder at the policy of any company.

Furthermore, programming in Ruby is rarely used on mobile devices, therefore in practice, they hardly ever have to be “specially” optimized. But the main difference between Ruby and the mobile development languages is that Ruby is a dynamic language – not in the sense that it is developing dynamically (though it is), but that it includes what is called *dynamic typing* of data as mentioned earlier.

The main advantage of dynamic typing compared with static is that there are fewer rules and less strictness, which gives a higher rate of development of apps by the programmer (admittedly at the cost of slower performance of the written programs, and of "sufficient" reliability. But performance rate does not particularly interest us, since Ruby is not used for developing mobile apps, although it is more often than not a key link on the server and facilitates the functioning of mobile apps for iOS, Android, etc.).

No doubt other programming trends not checked by the authors of this book do exist. For example, the development of computer games. A lifetime would probably not be enough to "check" all the trends, so we will leave this task to more inquisitive minds, and restrict ourselves to what is actually in demand in the market, permits "rapid input" and is more or less interesting rather than boring.

Web programming or something else?

The book "Ruby is for fun" is divided into two parts. In Part One (which you are reading now), we consider the basis of the Ruby language and its use from the so-called command line. Part Two (upcoming) will include web programming and the Rails framework.

"*Wait!*" the observant reader will say. "*Surely we've just been talking about web programming? Yet it's going to be in Part Two as well?*"

Quite true. The point is that Ruby is quite a powerful tool in itself. Students of the online Ruby school have found work even without knowledge of web programming. The basics of the language and the ability to find and use the required libraries already make possible the creation of quite useful apps which can be used for data processing (for example, so-called "web-scraping") to create script configurations and to control an operating system (which will definitely come in handy for any system administrator), for working with files of different formats etc.

The ability to use a language for jobs of various kinds not connected with web programming gives an indisputable advantage before you start web programming. Web programming itself involves the knowledge of certain generally accepted concepts. And we'll now be able to solve these problems using a tool we are already learning how to handle.

How much do programmers earn?

This is a very important question for those who know nothing about programming. But before answering it, I want to go back.

Since Ruby is mainly a language for web programming, it is Ruby on which programmers base remote work. The idea of working remotely on one project is particularly prevalent in web programming.

This is understandable. To create software for aircraft, for example, it is more useful to be in a science center working hand in hand with colleagues there. But in the case of web projects, it is often unimportant where the developer is actually located. The team "*37 signals*", the developers of which are in various places around the world and even in different time zones, has also contributed to the remote development concept. It was in "*37 signals*" that the first version of Rails, probably the most popular framework for web development, appeared.

It has been proved over the last ten years that remote development is possible. You don't always need to keep a team of programmers all in the same office. This is a huge plus for Ruby programmers: it means that they are not tied to any specific location, and can work for a company in the USA from a small town in Greece or Spain, for example. And that way they will receive a salary far in excess of what they could get locally.

If you look at the statistics of remote working[1], Ruby holds second place for the

[1] https://remoteok.io/stats.php

number of vacancies available. First place goes to JavaScript, but only because a basic knowledge of JavaScript is essential, and is required along with other languages: Java, PHP, Ruby etc. But "pure JavaScript" (Node.js) for full-stack programming is only in third place.

It is worth noting that the number of jobs in a certain language is not the most important indicator, and it can be of no importance at all in making the right choice for the rest of your life. We are only speaking of what we know now. It's very hard to predict several years ahead in the IT industry. But undoubtedly, the good news is that you don't need thousands of jobs, you only need to find one. Nor is it all that important how much time you spend looking for a job – a week, two weeks or two months, you'll certainly find one.

Here we come to statistics collected by Ruby School students. How much do Ruby programmers earn? Before answering, we will stipulate that we are only talking about remote working. The remote salary market is more stable. It was balanced out by programmers from different countries, and a certain value was arrived at for it. There is no sense in comparing salaries "on the spot": a Ruby programmer can (perhaps even must) work remotely, in most cases this is more profitable. It is also assumed that programmers have a basic knowledge of English, so that they can communicate with clients from other countries.

Salary categories can be divided into three parts. They may differ depending on type of work (remote or in an office) and on geographical location (immaterial if you are working remotely). At the present time the hourly rate for a programmer with one year's experience is from 15 to 30 US dollars an hour. From one to three years, roughly from 30 to 50. From three years upward, roughly from 50 dollars an hour. On reaching the 50-dollar level, it all becomes very individual. By the way, the average number of hours a month is 160.

From our own experience, it is quite possible, with no special talents, to master Ruby programming in one year and find your first remote-working job. Many Ruby School

students have done so, and you can find confirmation of these words on our website.

Your advantage

Before we start creating our first program, it is important to remind ourselves what is not related to programming. Everyone has different life experience. Some may have come to programming from music, some from finance. Any musician will find it vastly simpler to write a program for teaching people to read music. It will be simpler for a financier to write a program for a commercial balance sheet. What does your advantage consist of?

As you study Ruby, the question of creating your own program or series of programs based on your ideas will keep cropping up. This is necessary for the following reasons.

Firstly, any program usually concerns some business problem. Programmers are paid money to optimize business processes, simplify real life and save the time that people spend on all sorts of actions. For example, imagine a queue in some state institution in 1986. Many people have gathered in the waiting room and are waiting their turn. And now imagine that there is a programmer who has written an "electronic queue" program. Anyone can sign up to be seen, and come precisely at the appointed time. He or she could use the time that would have been wasted in the queue, e.g. teaching a math lesson to school-kids.

The economic benefit is obvious. Time that would have been spent in the queue can now be put to good use. And all because a useful website has been created. It is the same with your knowledge. Knowledge of any subject field is a valuable asset in itself. Try to see your advantage, think in what way you can improve the world. It's a good thing to have several ideas written down on paper. As you work with this book, you can come back to it and ask yourself: Could I do this with Ruby?

Secondly, by using your advantage in a particular field, you can create a program simply to demonstrate your knowledge. Even the simplest program written by a pro-

fessional musician will delight programmers of great experience, which musicians are not.

Don't throw your programs away, even the most naïve of them can be improved later. They will also come in handy when you are looking for work. It is far better to have some code sample available than not to have one. Your programs may seem insignificant, but in getting a job it is not one isolated program that matters, but the combination of everything you have demonstrated: a knowledge of programming, programs you have written, your resume, domain knowledge, an active GitHub account and an active blog about programming.

Thirdly, if you are not working on your project, your success will depend on chance, It is hard to predict in just what group you will end up or what software quality standards there will be in your company. People are naturally hopeful about the future, but experience shows that in real life, everything is somewhat different, and success often depends on chance.

It's a pity to find yourself in a company with bureaucratic complexities or working in a group of people with low technical qualifications. Furthermore, the newbie programmer may not even recognize these signs, and consequently can suffer depression and disappointment in his or her chosen profession. But in fact programming should give you satisfaction. And your own project is your landmark, an indicator of your level of improvement and insurance against bad luck.

In any difficult situation in your new work, you can say to yourself: "*Yes, maybe I'm not very productive in this job, but here is my project and here is a demonstration of my technical qualifications. So the problem is most likely not in me but in something else.*" Furthermore, this argument can always be used for a dialog with your manager, and the project itself can be added to your resume. Your project depends only on you, and there is a better-than-zero probability that one day your own project will start bringing in money for you.

Do this

Keep an ideas notebook. Write in it absolutely all the ideas that come into your head. You may return to them a week, a month or a year later.

Part I. First steps

Runtime environment

Runtime environment is an important concept. The concept of "environment" itself will be introduced later, but it's not the same thing. Runtime environment is where and "by whom" your programs will be launched in Ruby. Let's say a qualified chemist conducts an experiment in a test tube, a big glass jar and even his or her own bath. It can be used by any "interpreter" (program-launching program) in Windows, Mac and Linux.

When the author first started with a computer, there was only one runtime environment, because there was no choice. When the computer was switched on, a cursor and the letters "OK" came on, meaning a program could be loaded. Now computers have become more intelligent, and a newbie also has to understand how to launch a program, where to enter the program text, "with what" to launch the written program and which runtime environment is best.

In fact. it is not particularly important to us which actual program the system is launched in. These days a program written in any of the popular programming languages can be launched in three OS: Windows, MacOS and Linux. Usually no changes are required in the program itself. If there are, they are minimal.

Statistics on the use of operating systems shows that the most popular one today is Windows. So we will begin with Windows, although it is not the best choice. The reason we decided to do this is to get us as quickly as possible into the run of things, so that any starting programmer can write a required program as quickly as possible.

After all, setting up the runtime environment is not usually a very simple matter for beginners, and its apparent complexity may scare off a student at the first stage.

In spite of the fact that we shall begin by launching our programs in the Windows OS, we advise strongly against using the Windows for *launching programs* in Ruby, However, if you like, this OS can be used for *writing programs*. In any case, the authors recommend installing Linux as soon as possible (the *Linux Mint Cinnamon* edition, as the simplest installation medium), and using it. If you use Mac, there is no need to install Linux.