Resources to learn programming

Felix Dietrich*

September 13, 2019

Abstract

Here is a list of resources to get you started to learn programming. The list is neither complete nor unbiased, and reflects my set of skills and the resources I used to build them up more than it covers the entire field. Still, I tried to include a rather broad set of entry points into many different areas.

1 History

Early computing machines date back to Charles Babbage (see `https://plato.stanford.edu/entries/computing-history/`). He was working with the mathematician Ada Lovelace who published the first algorithm computable by their machine, the "Analytical Engine". In the 20th century, Alan Turing was an anchor point in computer science, writing on computable numbers [5] and tackling the famous Entscheidungsproblem with the introduction of the computational tool later called the "Turing machine". Around the same time, Wilhelm Ackermann introduced a function that is not "computable", see [1]. On a more practical level, Donald Knuth wrote his four volumes on The Art of Computer Programming [3], and introduced the computer type setting system TEX (later refined to LATEX by Leslie Lamport).

2 Basics

There is a whole realm on different hardware that I will not go into here, as well as the direct interfaces between hardware and software (called firmware and drivers). On top of this basis, there exist the operating systems, most famously Microsoft Windows, Linux with all its variants, and Apple OS, which are mostly used on personal computers, and the Windows Phone, Android, and iOS variants for mobile devices.

Programming languages typically provide high-level interfaces between human-readable text and their compiled versions, which are executable by the processing units. It is challenging to define "the best programming language", as their use very much depends on the task (learning programming the first time, learning a new language with some background, quickly developing prototypes, developing robust software...). The TIOBE index provides a first glimpse into which languages are popular, based on a broad and systematic search engine rating: `https://www.tiobe.com/tiobe-index`.

Be aware that there is a crucial difference between software development and programming. It is a somewhat dangerous difference, since it is not at all apparent to beginners. You may want to compare learning *programming* to learning how to use a hammer to construct a small wooden hut, and learning *software development* to studying architecture to construct skyscrapers. Both are used to provide fundamentally the same (walls and a roof), but are very different in their implementation. For example, an architect normally does not do the actual construction of a building.

3 Starting points

There is no objective reason to select a specific operating system to start learning programming. At this stage, it is much more important that you select the system you are most comfortable with, and start from there. After mastering a certain range of languages and algorithms, it might be helpful to get accustomed to an operating system with a Linux kernel (and its command line interface), since many software developers work with it and thus it is usually easy to install the latest version of their software using Linux. On

*felix.dietrich@tum.de

Microsoft Windows 10, you can install the Ubuntu Linux system with its command line terminal version as an integrated software, to access a large part of the Linux environment from the surrounding Windows operating system (see `https://tutorials.ubuntu.com/tutorial/tutorial-ubuntu-on-windows#0`).

For quick results and beginners, especially for plotting results from numerical computations, the Python language is ideal. It is typically used as a "glue language", meaning it takes software written in other languages and wraps them into easily accessible interfaces. A good starting point is the website `https://www.learnpython.org/`, where you can learn python without even installing anything (the website has an integrated python environment).

The Jupyter environment provides a convenient way to start and develop python programs. It is an environment that also supports many other programming languages, so getting familiar with it opens up more possibilities beyond Python. You can try it out in the browser, without installing anything here: `https://jupyter.org/try`. Once you are familiar with the environment, I recommend installing it by going through this tutorial: `https://jupyter.org/install`.

Numerical computations heavily using the glue language property of python can be done with the `numpy` and `scipy` packages, which you can read about here (including install options):

- NumPy `https://numpy.org/`
- SciPy `https://www.scipy.org/`

Another, very robust and managed programming environment is provided by Microsoft, called .NET (including an open source version and development environment for all major operating systems), here: `https://dotnet.microsoft.com/`. The main language in this environment is called C#, though I recommend also looking into functional programming with F#. You can learn more here: `fsharp.org`, here `https://fsharpforfunandprofit.com/`, and try it out in the browser here `https://try.fsharp.org/`.

4 Code organization

There are numerous ways to organize your development environment. A proven standard is the so-called *version control*, which essentially is keeping all changes to a source code, not just the latest version. This is extremely helpful when searching for errors in a team project, but even for projects with only a single developer, version control can help save many hours of work. A good version control system used in the industrial context is `git`, developed by Linus Torvalds (who is also the initial author of the Linux kernel for the corresponding operating system). Many free online services to store code repositories exist, with free and secure online storage up to a certain size limit. Examples are

- GitLab (GitLab Inc.) `https://about.gitlab.com/`
- GitHub (Microsoft) `https://github.com/`
- Bitbucket (Atlassian) `https://bitbucket.org/product/`

Note that you do not have to use an online service to store your data, as `git` supports setting up your repository anywhere (e.g. your local machine). The online services only store a copy of this local repository so you can access it from any computer. A good introduction to the Git environment, as well as a client software for all major operating systems can be found here: `https://git-scm.com/book/en/v1/Getting-Started-Git-Basics`. On Microsoft Windows, the Tortoise Git Shell Integration is very convenient, see `https://tortoisegit.org/`.

A version control system with a slightly different philosophy and implementation is `subversion` (Apache), see `https://subversion.apache.org/`.

5 Algorithms

There is a whole world of software underlying the high-level languages mentioned above. The most critical and well-developed concerns kernels (the base for operating systems) and code for linear algebra. When using higher-level programming languages, these algorithms are well hidden in the background and do not need to be installed or accessed directly.

NetLib provides an overview here: `http://www.netlib.org/`, with LAPACK/BLAS (here: `http://www.netlib.org/lapack/#_presentation`) the most commonly used, also by NumPy/SciPy cited above.

For statistical computations, the R language is the standard (see https://www.r-project.org/). It may not be the most convenient language for software development, but is used a lot by statisticians and thus has a lot of additional packages, even for very advanced and specific algorithms. You can learn R here, and also download the RStudio development environment https://www.rstudio.com/online-learning/#r-programming.

When you want to learn and develop your skills in developing these basic algorithms, I can recommend looking into the ACM programming challenges (ACM stands for the Association for Computing Machinery).

The challenges are computing problems posted online, that can be solved with a short piece of code in many languages. You can submit your solutions online, and see if the automated testing system accepts them. One example system is the UvA online judge with thousands of problems:

https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8.

Be aware that many of the challenges are extremely difficult, and require you to combine multiple, very advanced algorithms. Still, there are also a lot of beginner problems that do not require a lot of knowledge.

To learn about algorithms in general, I can recommend the book by Thomas Cormen [2] which you can find online.

For symbolic computations, the Wolfram Language is very well developed and can be accessed online, through https://www.wolframalpha.com/. The Mathematica Software is the backbone, and can be purchased (many universities provide it as a download for students and employees). An example for the power of this tool can be found here, where PDE are solved analytically with a few lines of code. https://reference.wolfram.com/language/howto/SolveAPartialDifferentialEquation.html

For numerical computations, the MATLAB software is widely used in industry, see https://www.mathworks.com/products/matlab.html. The PDE example can be found here: https://www.mathworks.com/help/matlab/math/partial-differential-equations.html

Free analogues of MATLAB are the Octave software, see https://www.gnu.org/software/octave/ and the SageMath software, see http://www.sagemath.org/.

6 Machine learning

There is no clear definition of machine learning, so I will just define it as "software that improves when more data is available".

This includes manifold learning such as Diffusion Maps here:

https://github.com/jmbr/diffusion-maps.

The scikit learn package offers many different algorithms for machine learning, including the basics for linear regression. https://scikit-learn.org/stable/

There are many computationally efficient software packages for neural networks, examples are https://pytorch.org/ (Facebook) and https://www.tensorflow.org/ (Alphabet/Google).

Gaussian Process Regression can be done with the packages for neural networks, but specialized software is available, too: https://sheffieldml.github.io/GPy/, with the standard literature by Rasmussen [4].

For a more machine learning oriented problem set, you can take a look at the Kaggle website (Alphabet/Google, https://www.kaggle.com/), with thousands of datasets, problems, and contests to work on.

7 Game development

It is extremely easy to start game development, even though the same rigor and perseverance that applies to software development also applies when you want to actually publish a game.

The development environments Unity and Unreal are two of the largest, and are both free for hobby developers. They can be found here

https://unity.com/, with a large number of beginner tutorials here https://learn.unity.com/tutorials.

https://www.unrealengine.com/en-US/

Good resources for basic algorithms are provided by Amit Patel and Sebastian Lague here:

https://www.redblobgames.com/

https://www.youtube.com/user/Cercopithecan/playlists

A cute example of what is possible with procedural generation is the blog here: `https://heredragonsabound.blogspot.com/`.

References

[1] Wilhelm Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen*, 99(1):118–133, dec 1928.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2005.

[3] Donald E. Knuth. *The Art of Computer Programming, Volumes 1-4A Boxed Set*. Addison-Wesley Professional, 2011.

[4] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation And Machine Learning)*. The MIT Press, 2005.

[5] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.