Coffee Break NumPy

A Simple Road to Data Science Mastery That Fits Into Your Busy Life

MAYER, RIAZ, RIEGER

Coffee Break Numpy

A Simple Road to Data Science Mastery
That Fits Into Your Busy Life

Christian Mayer, Zohaib Riaz, and Lukas Rieger

November 2018

A puzzle a day to learn, code, and play.

Contents

— 1 —

Introduction

In the 21st century, a new skill penetrates every area of our lives. As you will see, it is one of the most powerful skills in the world. Harvard Business Review (HBR) labeled the profession that comes with this skill as the "sexiest job of the 21st century." You can use it for good (e.g., improving the health of society) or for bad (e.g., hacking democracy via massive-scale social network manipulation). It threatens, directly or indirectly, every single human profession: millions of professional drivers, factory workers, writers, medical doctors, researchers, teachers, coders, retailers, salespeople, and small business owners. Those people and many more may lose their job—only because of this one skill. Why? Because it has the power to create machines that surpass human-level performance by magnitudes.

This skill is a new way of coding: data science.

The CEO of Siemens, Joe Kaeser, believes that data is the new powerful asset class of the 21st century:

> *"Data is the oil, some say the gold, of the 21st century—the raw material that our economies, societies and democracies are increasingly being built on."* [1]

If data is the new asset class of the information society, data scientists are the new investment bankers.

What is data science?

> *"Data science is an interdisciplinary field that uses scientific methods, processes, algorithms, and systems to extract knowledge and insights from data in various forms, both structured and unstructured."* [2]

Data scientists use computers to gain insights from massive amounts of data. These insights have a profound influence on the products we see, the medicines we take, the education we enjoy, the movies we watch, the routes

[1]`https://www.linkedin.com/pulse/technology-society-digital-transformation-joe-kaeser/`

[2]`https://en.wikipedia.org/wiki/Data_science`

we drive, the holidays we choose, and the foods we produce and consume.

A powerful tool in the tool belts of data scientists is machine learning. Machine learning, at its core, is the discipline of teaching machines to detect patterns and perform tasks by presenting them training data. In general, the more training data, the better machines perform in various tasks such as medical analysis, financial analysis, biotechnology, research, games, fraud detection—with hundreds of new applications getting published every day in diverse research disciplines such as biology, mathematics, finance, engineering, history, and law theory. More training data leads to a higher degree of automation which, in turn, generates better results much quicker and cheaper than ever before.

So data is indeed the new asset class of the 21st century. As of 2019, six of the ten world's largest companies (by market capitalization) are tech companies (Microsoft, Google, Amazon, Apple, Facebook, Alibaba). Each of those companies expends significant effort to acquire a growing chunk of this valuable asset class. For example, those data sets consist of GPS location trajectories, customer behaviors, social network activities, web surfing behavior, health indicators, search interests—just to name a few.

It's a modern-day gold rush, and this book gives you the shovel to participate. This book aims to be a stepping stone on your path to becoming a master data scientist. It helps you learn faster by making use of proven principles of good teaching. It offers you ten to twenty hours of thorough training using a fun and effective training technique, called *practice testing*. Practice testing is scientifically proven to be one of the most efficient training techniques (see Chapter 3.8). More than 60,000 online students have successfully applied this learning system at my Python online learning platform `Finxter.com`. And practice testing will work for you, too.

This book focuses on one of the most popular programming languages for data science and machine learning: Python. A recent StackOverflow article *The Incredible Growth of Python*[3] shows that Python is one of the fastest growing major programming languages. But this book is not a general Python introduction like its predecessor *Coffee Break Python*[4]. Instead, this book teaches you the ins and outs of the NumPy library, which is used for numerical computations, for data science, and—more and more—for machine learning.

Let's be blunt: without understanding the concepts and ideas behind NumPy, you will not become a successful data scientist.

[3]`https://stackoverflow.blog/2017/09/06/incredible-growth-python/`

[4]`https://blog.finxter.com/coffee-break-python`

— 2 —

Why Learn NumPy?

To show you the importance of NumPy, let us clarify NumPy's distinguished position as a crucial Python library among other existing libraries such as Pandas.

First and foremost, NumPy adds strongly to the basic data structures in Python such as lists and dictionaries. It does so by providing the fundamental *array* data structure and supports it with a pack of powerful functions for data science. For example, you can easily compute basic statistics on NumPy arrays such as average, variance, standard deviation, and a lot of aggregator functions (e.g., summing over a subset of values). Coupled with these powerful functions, arrays can be used to process high dimensional data while reducing the coding effort to the minimum.

At the same time, NumPy offers you the flexibility of implementation which may be hard to achieve with other high-level libraries for Python. For instance, consider the Pandas library. Pandas is more specialized in handling tabular data, i.e., rows and columns of values, which is a popular format for many datasets. For this purpose, Pandas offers a popular DataFrame object type, which can read-in and manipulate data from standard file formats such as the comma-separated-values (CSV). DataFrames offer a variety of 'data-crunching' functions such as grouping, merging, and statistics. However, we may not always deal with tabular data in our data-science pipeline. On many occasions, we may need to compute special numerical functions over arrays or matrices. Here, NumPy is more specialized and offers the needed flexibility.

In our view, since Numpy adds to the basic capabilities of Python (with its arrays), programmers who wish to learn Python from the bottom up should learn NumPy first. This view is also substantiated by understanding how other popular data-processing tools work. For example, the well-known MATLAB environment offers NumPy like array/matrix handling capability as a basic feature. Then as a high-level data-structure, it also offers the handling of tabular data in the 'table' data-structure.

Hence, due to the powerful basic functionality offered by NumPy, it clearly distinguishes itself among other Python libraries. Moreover, it also forms a building block to support the functionality of high-level libraries such as Pandas.

— 3 —

A Case for Puzzle-based Learning

Definition: A *code puzzle* is an educative snippet of source code that teaches a single computer science concept by activating the learner's curiosity and involving them in the learning process.

Like the other books in the *Coffee Break Python* series, this book is based on the popular concept of puzzle-based learning to code—tried and tested by tens of thousands of online students and proven by educational science to be superior to most other learning techniques.

But before diving into practical puzzle solving, let us first study 10 reasons for puzzle-based learning—and why it helps you to learn NumPy faster and keep the basics longer. **Feel free to skip this bonus chapter if you**

already know about the benefits of puzzle-based learning from previous learning material. As you will see in this chapter, there is robust evidence in psychological science for each of these reasons. Yet, none of the existing coding books lift code puzzles to being first-class citizens. Instead, they are mostly focused on one-way teaching: the teacher speaks and you have to listen. This book attempts to change that. In brief, the 10 reasons for puzzle-based learning are the following.

1. Overcome the Knowledge Gap (Section 3.1)
2. Embrace the Eureka Moment (Section 3.2)
3. Divide and Conquer (Section 3.3)
4. Improve From Immediate Feedback (Section 3.4)
5. Measure Your Skills (Section 3.5)
6. Individualized Learning (Section 3.6)
7. Small is Beautiful (Section 3.7)
8. Active Beats Passive Learning (Section 3.8)
9. Make Source Code a First-class Citizen (Section 3.9)
10. What You See is All There is (Section 3.10)

3.1 Overcome the Knowledge Gap

The great teacher Socrates delivered complex knowledge by asking a sequence of questions. Each question built on answers to previous questions provided by the student. This more than 2400 year old teaching technique is still in widespread use today. A good teacher opens a gap between their knowledge and the learner's. The knowledge gap makes the learner realize that they do not know the answer to a burning question. This creates a tension in the learner's mind. To close this gap, the learner awaits the missing piece of knowledge from the teacher. Better yet, the learner starts developing their own answers. The learner *craves knowledge*.

Code puzzles open an immediate knowledge gap. When looking at the code, you first do not understand the meaning of the puzzle. The puzzle's semantics are hidden. But only you can transform the unsolved puzzle into a solved one. Look at this riddle: "What pulls you down and never lets go?" Can you feel the tension? Opening and closing a knowledge gap is a very powerful method for effective learning.[1]

Bad teachers open a knowledge gap that is too large. The learner feels frustrated because they cannot overcome the

[1]The answer is *Gravity*.

gap. Suppose you are standing before a river that you must cross. But you have not learned to swim yet. Now, consider two rivers. The first is the Colorado River that carved out the Grand Canyon—quite a gap. The second is Rattlesnake Creek. The fact that you have never heard of this river indicates that it is not too big of an obstacle. Most likely, you will not even attempt to swim through the big Colorado River. But you could swim over the Rattlesnake if you stretch your abilities just a little bit. You will focus, pep-talk yourself, and overcome the obstacle. As a result, your swimming skills and your confidence will grow a little bit.

Puzzles are like the Rattlesnake—they are not too great a challenge. You must stretch yourself to solve them, but you can do it, if you go all-out.

Constantly feeling a small but non-trivial knowledge gap creates a healthy learning environment. Stretch your limits, overcome the knowledge gap, and become better—one puzzle at a time.

3.2 Embrace the Eureka Moment

Humans are unique because of their ability to learn. Fast and thorough learning has always increased our chances of survival. Thus, evolution created a brilliant biological reaction to reinforce learning in your body. Your brain is wired to seek new information; it is wired to always process data, to always learn.

Did you ever feel the sudden burst of happiness after experiencing a eureka moment? Your brain releases endorphins, the moment you close a knowledge gap. The instant gratification from learning is highly addictive, but this addiction makes you smarter. Solving a puzzle gives your brain instant gratification. Easy puzzles lead to small, hard puzzles, which lead to large knowledge gaps. Overcome any of them and learn in the process.

3.3 Divide and Conquer

Learning to code is a complex task. You must learn a myriad of new concepts and language features. Many aspiring coders are overwhelmed by the complexity. They seek a clear path to mastery.

People tend to prioritize specific activities with clearly defined goals. If the path is not clear, we tend to drift away toward more specific paths. Most aspiring coders think they have a goal: becoming a better coder. Yet, this is not a specific goal at all. So what is a specific goal? *Watching Breaking Bad after dinner, Series 2 Episode 1* is as specific as it can be. Due to the specificity, watching Netflix is more powerful than the fuzzy path of learning to code. Hence, watching Netflix wins most of the time.

As any productivity expert will tell you: break a big task or goal into a series of smaller steps. Finishing each tiny step brings you one step closer to your big goal. *Divide and conquer* makes you feel in control, pushing you one step closer toward mastery. You want to become a master coder? Break the big coding skill into a list of sub-skills—understanding language features, designing algorithms, reading code—and then tackle each sub-skill one at a time.

But how can you do this if you don't know anything about the topic yet? You cannot really comprehend the important subtopics of the skill to be acquired—without a mentor who has already been there and done that, your learning speed will be slow. You don't have time to waste, do you?

Fortunately, code puzzles do this for you. They break up the huge task of learning to code into a series of smaller learning units. The student experiences laser focus on one learning task such as *matrix multiplication*, the *standard deviation*, or *slicing*. Don't worry if you do not understand these concepts yet—after working through this book, you will.

A good code puzzle delivers a single idea from the author's into the student's head. You can digest one puzzle at a time. Each puzzle is a step toward your bigger goal of mastering data science. Keep solving puzzles and you keep improving your skills.

3.4 Improve From Immediate Feedback

The right feedback is critical for your success. As a child, you learned to walk by trial and error—try, get feedback, adapt, and repeat. Unconsciously, you minimize negative and maximize positive feedback. You avoid falling because it hurts. You seek the approval of your parents. Feedback supervised your learning progress along the way.

But not only organic life benefits from the great learning technique of trial and error. In machine learning, algorithms learn by guessing an output and adapting their guesses based on their correctness. To learn anything, you need feedback such that you can adapt your actions.

However, an excellent learning environment provides you not only with feedback but with *immediate* feedback for your actions.

In contrast, poor learning environments do not provide any feedback at all or only with a large delay. Examples are activities with good short-term and bad long-term effects such as smoking, alcohol, or damaging the environment. People cannot control these activities because of the delayed feedback. If you were to slap your friend each time he lights a cigarette—a not overly drastic measure to safe his life—he would quickly stop smoking. If you want to learn fast, make sure that your environment provides immediate feedback. Your brain will find rules and patterns to maximize the reinforcement from the immediate feedback.

This book offers you an environment with immediate feedback to make learning to code NumPy easy and fast. Over time, your brain will absorb the meaning of a code snippet quicker and with higher precision this way. Learning this skill pushes you toward the top 10% of all coders. There are other environments with immediate feedback, like executing code and checking correctness, but puzzle-based learning is the most direct one: Each puzzle educates with immediate feedback.

3.5 Measure Your Skills

You need to have a definite goal to be successful. A definite goal is a powerful motivator and pushes you to stretch your skills constantly. The more definite and concrete it is, the stronger it becomes. Holding a definite goal in your mind is the first and foremost step toward its physical manifestation. Your beliefs bring your goal into reality.

Think about an experienced Python programmer you know, e.g., your nerdy colleague or class mate. How good are their Python skills compared to yours? On a scale from your grandmother to Bill Gates, where is your colleague and where are you? These questions are difficult to answer because there is no simple way to measure the skill level of a programmer. This creates a severe problem for your learning progress: the concept of being a good programmer becomes fuzzy and diluted. What you can't measure, you can't improve. Not being able to measure your coding skills diverts your focus from systematic improvement. Your goal becomes less definite.

So what should be your definite goal when learning a programming language? To answer this, let us travel briefly to the world of chess, which happens to provide an excellent learning environment for aspiring players. Every player has an Elo rating number that measures their skill level. You get an Elo rating when playing against other players—if you win, your Elo rating increases. Victories against stronger players lead to a greater increase of the Elo rating. Every ambitious chess player simply focuses on one thing: increasing their Elo rating. The ones that manage to push their Elo rating very high, earn grand master titles. They become respected among chess players and in the outside world.

Every chess player dreams of being a grandmaster. The goal is as definite as it can be: reaching an Elo of 2400 and master level (see Section 4). Thus, chess is a great learning environment—every player is always aware of their skill level. A player can measure how decisions and habits impact their Elo number. Do they improve when sleeping enough before important games? When training opening variants? When solving chess puzzles? What you can measure, you can improve.

The main idea of this book, and the associated learning app `Finxter.com`, is to transfer this method of measuring skills from the chess world to programming. Suppose you want to learn Python. The Finxter website assigns you a rating number that measures your coding skills. Every Python puzzle has a rating number as well, according to its difficulty level. You 'play' against a puzzle at your difficulty level: The puzzle and you will have more or less the same Elo rating so that you can enjoy personalized learning. If you solve the puzzle, your Elo increases and the puzzle's Elo decreases. Otherwise, your Elo decreases and the puzzle's Elo increases. Hence, the Elo ratings of the difficult puzzles increase over time. But only learners with high Elo ratings will see them. This self-organizing system ensures that you are always challenged but not overwhelmed, while you constantly receive feedback about how good your skills are in comparison with others. You always know exactly where you stand on your path to mastery.

3.6 Individualized Learning

The educational system today is built around the idea of classes and courses. In these environments, all students consume the same learning material from the same teacher applying the same teaching methods. This traditional idea of classes and courses has a strong foundation in our culture and social thinking patterns. Yet, science proves again and again the value of individualized learning. Individualized learning tailors the content, pace, style, and technology of teaching to the student's skills and interests. Of course, truly individualized learning has always required a lot of teachers. But paying a high number of teachers is expensive (at least in the short term) in a non-digital environment.

In the digital era, many fundamental limitations of our society begin to crack. Compute servers and intelligent machines can provide individualized learning with ease. But with changing limitations, we must adapt our thinking as well. Machines will enable truly individualized learning very soon; yet society needs time to adapt to this trend.

Puzzle-based learning is a perfect example of automated, individualized learning. The ideal puzzle stretches the student's abilities and is neither boring nor overwhelming. Finding the perfect learning material for each learner is an important and challenging problem. The Finxter learning system uses a simple but effective solution to solve this problem: the Elo rating system. The student solves puzzles at their individual skill level. This book with it's web backend Finxter pushes teaching toward individualized learning.

3.7 Small is Beautiful

The 21st century has seen a rise in microcontent. Microcontent is a short and accessible piece of valuable information such as the weather forecast, a news headline, or a cat video. Social media giants like Facebook and Twitter offer a stream of never-ending microcontent. Microcontent is powerful because it satisfies the desire for shallow entertainment. Microcontent has many benefits: the consumer stays engaged and interested, and it is easily digestible in a short time. Each piece of microcontent pushes your knowledge horizon a bit further. Today, millions of people are addicted to microcontent.

However, this addiction will also become a problem to these millions. The computer science professor Cal Newport shows in his book *Deep Work* that modern society values deep work more than shallow work. Deep work is a high-value activity that needs intense focus and skill. Examples of deep work are programming, writing, or researching. Contrarily, shallow work is every low-value activity that can be done by everybody (e.g., posting cat videos to social media). The demand for deep work grew with the rise of the information society; at the same time, the supply stayed constant or decreased, among other things because of the addictiveness of shallow social media. People that see and understand this trend can benefit tremendously. In a free market, the prices of scarce and demanded resources rise. Because of this, surgeons, lawyers, and software developers earn $100,000 per year and more. Their work cannot easily be replaced or outsourced to unskilled workers. If you are able to do deep work, to focus your attention on a challenging problem, society pays you generously.

What if we could marry the concepts of microcontent and deep work? This is the promise of puzzle-based learning. Finxter offers a stream of self-contained microcontent in the form of hundreds of small code puzzles. But instead of just being unrelated microcontent, each puzzle is a tiny stimulus that teaches a coding concept or language feature. Hence, each puzzle pushes your knowledge *in the same direction*.

Puzzle-based learning breaks the bold goal, i.e., *reach the mastery level in Python's NumPy library*, into tiny actionable steps: solve and understand one code puzzle per day. While solving the smaller tasks, you progress toward your larger goal. You take one step at a time to eventually reach the mastery level. A clear path to success.

3.8 Active Beats Passive Learning

Robust scientific evidence shows that active learning doubles students' learning performance. In a study on this topic, test scores of active learners improved by more than one grade compared to their passive learning fellow students.[2] Not using active learning techniques wastes your time and hinders you in reaching your full potential in any area of life. Switching to active learning is a simple tweak that will instantly improve your performance when learning any subject.

How does active learning work? Active learning requires the student to interact with the material, rather than simply consuming it. It is student- rather than teacher-centric. Great active learning techniques are asking and answering questions, self-testing, teaching, and summarizing. A popular study shows that one of the best learning techniques is *practice testing.*[3] In this learning technique, you test your knowledge even if you have not learned everything yet. Rather than *learning by doing*, it's *learning by testing.*

However, the study argues that students must feel safe during these tests. Therefore, the tests must be low-

[2] `https://en.wikipedia.org/wiki/Active_learning#Research_evidence`

[3] `http://journals.sagepub.com/doi/abs/10.1177/1529100612453266`

stake, i.e., students have little to lose. After the test, students get feedback about the correctness of the tests. The study shows that practice testing boosts long-term retention of the material by almost a factor of 10. As it turns out, solving a daily code puzzle is not just another learning technique—it is one of the best.

Although active learning is twice as effective, most books focus on passive learning. The author delivers information; the student passively consumes the information. Some programming books include active learning elements by adding tests or by asking the reader to try out the code examples. Yet, I always found this impracticable while reading on the train, on the bus, or in bed. But if these active elements drop out, learning becomes 100% passive again.

Fixing this mismatch between research and common practice drove me to write my *Coffee Break Python* book series about puzzle-based learning of Python and Python's libraries. In contrast to other books, this book makes active learning a first-class citizen. Solving code puzzles is an inherent active learning technique. You must develop the solution yourself, in every single puzzle. The teacher is as much in the background as possible—they only explain the correct solution if you couldn't work it out yourself. But before telling you the correct solution, your knowledge gap is already ripped wide open. Thus, you are mentally ready to digest new material.

Let me emphasize this argument again: puzzle-based learning is a variant of the active learning technique named practice testing. Practice testing is scientifically proven to teach you more in less time.

3.9 Make Code a First-class Citizen

Each grandmaster of chess has spent tens of thousands of hours looking into myriad chess positions. Over time, they develop a powerful skill: the intuition of the expert. When presented with a new position, they are able to name a small number of strong candidate moves within seconds. They operate on a higher level than normal people. For normal people, the position of a single chess piece is one chunk of information. Hence they can only memorize the position of about six chess pieces. But chess grand masters view a whole position or a sequence of moves as a single chunk of information. The extensive training and experience has burned strong patterns into their biological neural networks. Their brain is able to hold much more information—a result of the good learning environment they have put themselves in.

What are some principles of good learning? Let us dive into another example of a great learning environment—this time for machines. Google's artificial intelligence AlphaZero has proven to be the best chess playing entity in the world. AlphaZero uses artificial neural networks. An artificial neural network is the digital twin of the human brain with artificial neurons and synapses. It learns by example much like a grandmaster of chess. It presents itself a position, predicts a move, and adapts its prediction to the extent the prediction was incorrect.

Chess and machine learning exemplify principles of good learning that are valid in any field you want to master. First, transform the object to learn into a stimulus that you present to yourself over and over again. In chess, study as many chess positions as you can. In math, make reading mathematical papers with theorems and proofs a habit. In coding, expose yourself to lots of code. Second, seek feedback. Immediate feedback is better than delayed feedback. However, delayed feedback is still much better than no feedback at all. Third, take your time to learn and understand thoroughly. Although it is possible to learn on-the-go, you will cut corners. The person who prepares beforehand always has an edge. In the world of coding, some people recommend learning by coding practical projects and doing nothing more. Chess grandmasters, sports stars, and intelligent machines do not follow this advice. They learn by practicing isolated stimuli again and again until they have mastered them. Then they move on to more complex stimuli.

Puzzle-based learning is code-centric. You will find yourself staring at the code for a long time until the insight strikes. This creates new synapses in your brain that help you understand, write, and read code fast. Placing code at the center of the whole learning process creates an environment in which you will develop the powerful intuition of the expert. *Maximize the learning time you spend looking at code rather than at other stimuli.*

3.10 What You See is All There is

My professor of theoretical computer science used to tell us that if we only stare long enough at a proof, the meaning will transfer into our brains by osmosis. This fosters deep thinking, a state of mind where learning is more productive. In my experience, his staring method works—but only if the proof contains everything you need to know to solve it. It must be self-contained.

A good code puzzle beyond the most basic level is self-contained. You can solve it purely by staring at it until your mind follows your eyes—your mind develops a solution based on rational thinking. There is no need to look things up. If you are a great programmer, you will find the solution quickly. If not, it will take more time but you can still find the solution—it is just more challenging.

My gold standard was to design each puzzle such that it is mostly self-contained. However, to deliver on the book's promise of training your understanding of the Python basics, puzzles must introduce syntactical language elements as well. But even if the syntax in a puzzle challenges you, you should still develop your own solutions based on your imperfect knowledge. This probabilistic thinking opens the knowledge gap and prepares your brain to receive and digest the explained solution. After all, your goal is long-term retention of the material.

— 4 —

The Elo Rating for Python—and NumPy

Pick any sport you always loved to do. How good are you compared to others? The Elo rating answers this question with surprising accuracy. It assigns a number to each player that represents their skill in the sport. The higher the Elo number, the better the player.

Let us give a small example of how the Elo rating works in chess. Alice is a strong player with an Elo rating of 2000 while Bob is an intermediate player with Elo 1500. Say Alice and Bob play a chess game against each other. Who will win the game? As Alice is the stronger player, she should win the game. The Elo rating system rewards players for good and punishes for bad results: the better the result, the higher the reward. For Bob, a win, or even a draw, would be a very good outcome of the game.

For Alice, the only satisfying result is a win. Winning against a weaker player is less rewarding than winning against a stronger player. Thus, the Elo rating system rewards Alice with only +3 Elo points for a win. A loss costs her -37 Elo points, and even a draw costs her -17 points. Playing against a weaker player is risky for her because she has much to lose but little to win.

The idea of Finxter is to view your learning as a series of games between two players: you and the Python puzzle. Both players have an Elo rating. Your rating measures your current skills and the puzzle's rating reflects its difficulty. On our website `finxter.com`, a puzzle plays against hundreds of Finxter users. Over time, the puzzle's Elo rating converges to its true difficulty level—while your Elo rating converges to your true skill level. A compelling idea, isn't it?

Table 4.1 shows the ranks for each Elo rating level. The table is an opportunity for you to estimate your Python skill level. In the following, I describe how you can use this book to test your Python skills.

4.1 How to Use This Book

This book provides a series of 48 code puzzles plus explanations to test and train your NumPy skills. The puzzles start from an intermediate level and become gradu-

Elo rating	**Rank**
2500	World Class
2400-2500	Grandmaster
2300-2400	International Master
2200-2300	Master
2100-2200	National Master
2000-2100	Master Candidate
1900-2000	Authority
1800-1900	Professional
1700-1800	Expert
1600-1700	Experienced Intermediate
1500-1600	Intermediate
1400-1500	Experienced Learner
1300-1400	Learner
1200-1300	Scholar
1100-1200	Autodidact
1000-1100	Beginner
0-1000	Basic Knowledge

Table 4.1: Elo ratings and skill levels.

ally harder to reach advanced level. This book is perfect for users who have already reached intermediate Python coding level. Yet, even expert users can improve their speed of code understanding. No matter your current skill level, you will benefit from puzzle-based learning. It will deepen and accelerate your understanding of basic coding patterns. But even more importantly, you will take the first step towards your thorough data science education.

4.2 The Ideal Code Puzzle

The ideal code puzzle possesses each of the following six properties. The puzzle

1. has a surprising result;
2. provides new information;
3. is relevant and practical;
4. delivers one main idea;
5. can be solved by thinking alone; and
6. is challenging but not overwhelming.

This was the gold standard for all the puzzles created in this book. I did my best to adhere to this standard.

4.3 How to Exploit the Power of Habits?

You are what you repeatedly do. Your habits determine your success in life and in any specific area such as coding. Creating a powerful learning habit can take you a long way on your journey to becoming a code master. Charles Duhigg, a leading expert in the psychology of habits, shows that each habit follows a simple process called the *habit loop*. This process consists of three steps: trigger, routine, and reward.[1] First, the trigger starts the process. A trigger can be anything such as drinking your morning coffee. Second, the routine is an action you take when presented with the trigger. An example routine is to solve a code puzzle. Each routine is in anticipation of a reward. Third, the reward is anything that makes you feel good. When you overcome a knowledge gap, your brain releases endorphins—a powerful reward. Over time, your habit becomes stronger—you seek the reward.

Habits with strong manifestations in these three steps are life-changing. Invest 10% of your paycheck every month and you will be rich one day. Get used to the habit of solving one Python (or NumPy) puzzle a day as you drink your morning coffee—and enjoy the endorphin dose in your brain. Implementing this *Finxter loop* in your day

[1] Charles Duhigg, *The Power of Habit: Why We Do What We Do in Life and Business.*

sets up an automatic progress toward you becoming a better and better coder. As soon as you have established the Finxter loop as a strong habit, it will cost you neither a lot of time, nor energy. This is self-engineering at its finest level.

4.4 How to Test and Train Your Skills?

I recommend solving at least one or two code puzzles every day—e.g., as you drink your morning coffee. Then you spend the rest of your learning time on real projects that matter to you. The puzzles guarantee that your skills improve over time and the real project brings you results.

If you want to test your NumPy skills, use the following simple method.

1. Track your individual Elo rating as you read the book and solve the code puzzles. Simply write your current Elo rating into the book. Start with an initial rating of 1500 if you are a Python intermediate who is just starting out with NumPy. Otherwise, adapt this initial rating towards your estimated skill level in Python. Of course, if you already have an online rating on `finxter.com`, starting with this

rating would be the most precise option. Figure 4.4 shows five different examples of how your Elo will change while working through the book. Two factors impact the final rating: how you select your initial rating and how well you perform (the latter being more important).

2. If your solution is correct, add the Elo points according to the table given with each single puzzle. Otherwise, subtract the given Elo points from your current Elo number.

Solve the puzzles in a sequential manner because they build upon each other. Advanced readers can also solve puzzles in the sequence they wish—the Elo rating will still work. The Elo rating will become more accurate as you solve more and more puzzles. Although only an estimate, your Elo rating is an objective measure to compare your skills with the skills of others. Several Finxter users have reported that the rating is surprisingly accurate.

Use the following training plan to develop a strong learning habit with puzzle-based learning.

1. Select a daily trigger after which you solve code puzzles for 10 minutes. For example, decide on your *Coffee Break NumPy*, or even solve code puzzles as you brush your teeth or sit on the train to work, university, or school.

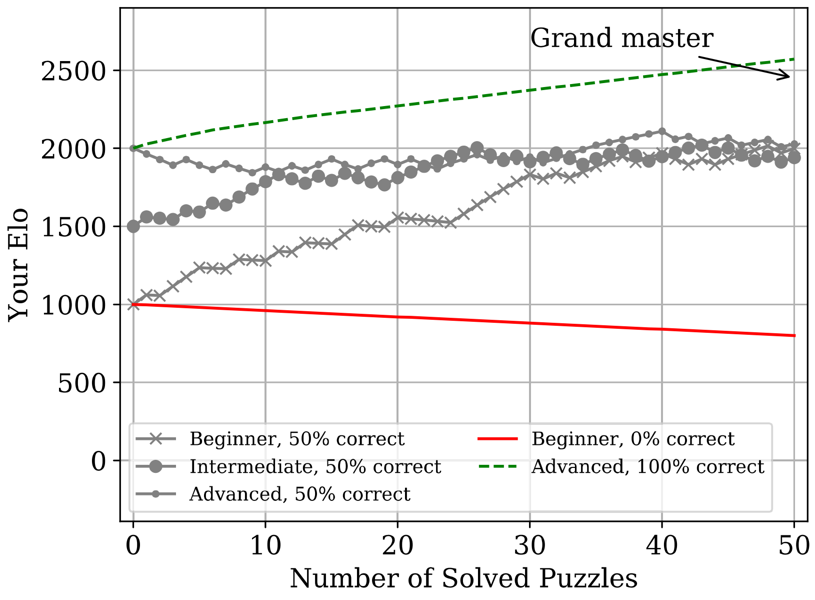


Figure 4.1: This plot exemplifies how your Elo rating may change while you work through the 50 code puzzles. No matter how you select your initial Elo, it will converge on your true skill level as you solve more puzzles. Note that you will lose Elo points faster when you have a higher Elo number. Your final Elo will be anywhere between 900 and 2500 after working through this book.

2. Scan over the puzzle in a first quick pass and ask yourself: what is the unique idea of this puzzle?

3. Dive deeply into the code. Try to understand the purpose of each symbol, even if it seems trivial at first. Avoid being shallow and lazy. Instead, solve each puzzle thoroughly and take your time. It's counterintuitive: To learn faster in less time, you must stay calm and take your time and allow yourself to dig deep. There is no shortcut.

4. Make sure you carry a pen with you and write your solution into the book. This ensures that you stay objective—we all have the tendency to fake ourselves. Active learning is a central idea of this book.

5. Look up the solution and read the explanation with care. Do you understand every aspect of the code? Write open questions down and look them up later, or send them to me (`info@finxter.com`). I will do everything I can to come up with a good explanation.

6. Only if your solution was 100% correct—including whitespaces, data types, and formatting of the output—do you get Elo points for this puzzle. Otherwise you should count it as a wrong solution and swallow the negative Elo points. The reason for this strict rule

is that this is the best way to train yourself to solve the puzzles thoroughly.

As you follow this simple training plan, your skill to understand source code quickly will improve. Over the long haul, this will have a huge impact on your career, income, and work satisfaction. You do not have to invest much time because the training plan requires only 10–20 minutes per day. But you must be persistent in your training effort. If you get off track, get right back on track the next day. When you run out of code puzzles, feel free to checkout `Finxter.com`, which has more than 300 handcrafted code puzzles. I regularly publish new code puzzles on the website as well.

4.5 What Can This Book Do For You?

Before we dive into puzzle solving, let me anticipate and address possible misconceptions about this book.

The puzzles are too easy/too hard. This book is for you if you already have some experience in coding. Your skill level in the Python programming language ranges from intermediate to expert. Even so, if you are already an advanced coder, this book is for you as well—if you read it in a different way. Measure the time you need to solve the puzzles and limit your solution time to only 10–20 seconds. This introduces an additional challenge for solving the puzzles: time pressure. Solving puzzles under time pressure sharpens your rapid code understanding skills even more. Eventually, you will feel that your coding intuition has improved. If the puzzles are too hard, great. Your knowledge gap must be open before you can effectively absorb information. Just take your time to thoroughly understand every bit of new information.

Learning to code is best done via coding on projects. This is only part of the truth. Yes, you can improve your skills to a certain level by diving into practical projects. But as in every other discipline, your skills will quickly hit your personal ceiling. Your ceiling is the maximum skill level you are able to reach, given your current limitations. These limitations come from a lack of thorough understanding of basic knowledge. You cannot understand higher-level knowledge properly without understanding the basic building blocks. Have you ever used machine learning techniques in your work? Without theoretical foundations, you are doomed. Theory pushes your ceiling upwards and gets rid of the limitations that hold you back.

Abraham Lincoln said: *"Give me six hours to chop down a tree and I will spend the first four sharpening the axe."* Do not fool yourself into the belief that *just doing it* is the most effective road to reach any goal. You must constantly sharpen the axe to be successful in any discipline. Learning to code is best done via practical coding *and* investing time into your personal growth. Millions of computer scientists enjoyed an academic education. They know that solving hundreds or thousands of toy examples in their studies built a strong and thorough foundation.

How am I supposed to solve this puzzle if I do not know the meaning of this specific NumPy function? Guess it! Python is an intuitive language, and NumPy has very intuitive naming of its functions. Think about potential meanings. Solve the puzzle for each of them—a good exercise for your brain. The more you work on the puzzle, even with imperfect knowledge, the better you prepare your brain to absorb the puzzle's explanation.

Why should I buy the book when puzzles are available for free at `Finxter. com`? My goal is to remove barriers to learning Python. Thus, all puzzles are available for free online. This book is based on the puzzles available at Finxter, but it extends them with more detailed and structured information. Nevertheless, if you don't like reading books, feel free to check out the website.

Anyway, why do some people thrive in their fields and become valued experts while others stagnate? They read books in their field. They increase their value to the marketplace by feeding themselves with valuable information. Over time, they have a huge advantage over their peers.

They get the opportunities to develop themselves even further. They enjoy their jobs and have much higher work satisfaction and life quality. Belonging to the top ten percent in your field yields hundreds of thousands of dollars during your career. However, there is a price you have to pay to unlock the gates to this world: you have to invest in books and your own personal development. The more time and money you spend on books, the more valuable you become to the marketplace!

The Elo-based rating is not accurate. Several finxters find the rating helpful, fair, and accurate in comparison to others. It provides a good indication of where one stands in the field of Python coders. If you feel the rating is not accurate, ask yourself whether you are objective. If you think you are, please let me know so that I have a chance to improve this book and the Finxter back-end.

— 5 —

A Quick Data Science Tutorial: The NumPy Library

This tutorial gives you a simple introduction, with many practical examples, to Python's NumPy library. You don't need any prerequisites to follow the tutorial. The idea of the tutorial is to give you everything you need to know to successfully solve the puzzles in the later parts of the book.

By working through the tutorial, you will gain a basic understanding of the most important NumPy functionality. Moreover, it will give you references to further reading as well as "next steps." Reading this tutorial takes 20–30 minutes and will be a fertile investment in your education and your coding efficiency. It's our belief that the purpose of any good learning material is to ultimately save your time.

So without further introduction, let's dive into the NumPy library in Python.

5.1 What is NumPy?

NumPy is a Python library that allows you to perform numerical calculations. Think about linear algebra in school or university—NumPy is the Python library for it. It's about matrices and vectors—and performing mathematical operations on them.

At the heart of NumPy is a basic data type, called the *NumPy array*. A NumPy array may have a number of dimensions, thus allowing it to represent quantities such as vectors (1D), matrices (2D), or higher dimensional arrays such as tensors. A NumPy array allows only one data type for all its elements. In this sense, NumPy arrays are different from Python lists that allow arbitrary data types. Therefore we say that NumPy requires homogeneous data values, so a NumPy array contains either integer or float values, but not both at the same time. These data type restrictions allow NumPy to specialize in providing efficient linear algebra operations.

Among those operations are maximum, minimum, average, standard deviation, variance, dot product, matrix product, and many more. NumPy implements these operations efficiently and in an easy-to-use manner. By learning NumPy, you equip yourself with a powerful tool for data analysis on numerical multi-dimensional data. But you may ask (and rightly so):

5.2 What can NumPy do for me?

Fear of missing out in machine learning and data science? Learning NumPy now is a great first step into the field of machine learning and data science. In machine learning, crucial algorithms and data structures rely on matrix computations, which are efficiently handled by NumPy. As said earlier, matrices in NumPy are nothing but arrays with homogenous data, for example, float values.

NumPy is among the most popular libraries in Python. Most machine learning experts agree that Python is the top programming language for machine learning. Within Python, NumPy is one of the most important libraries for data science and machine learning. For instance, searching for the keyword 'NumPy machine learning' reveals more than 3 million pages. Compare this to Python's scikit-learn library that directly addresses machine learning and results in approximately 3 million pages as well. So NumPy is as popular for machine learning as scikit-learn in Python! As you can see, NumPy produces as many results—even though it is not directly addressing machine learning (unlike scikit-learn).

No matter which library is more popular, NumPy is the 600-pound Gorilla in the machine learning and data science space. If you are serious about your career as a data scientist, you have to conquer NumPy now!

But NumPy is not only important for machine learning and data science. NumPy's diverse functionality and its computational efficiency leads to its use in various fields such as mathematics, electrical engineering, high-performance computing, and simulations.

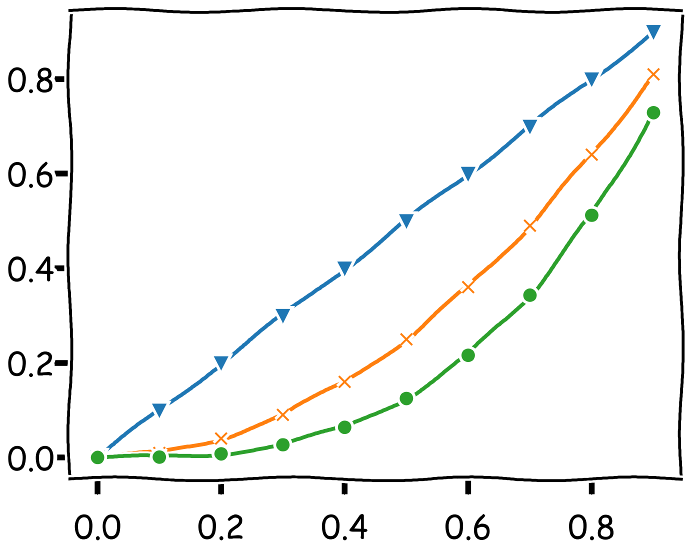
Also, if you need to visualize data, you are very reliant on the NumPy library. Here is an example from the official documentation of Python's plotting library matplotlib. You can see a small script that plots a linear, quadratic, and cubic function. It uses only two libraries: matplotlib and, obviously, NumPy!

```python
import numpy as np
import matplotlib.pyplot as plt

# evenly distributed data between 0 and 1
x = np.arange(0., 1., 0.1)

# xkcd-styled plot
plt.xkcd()

# linear, quadratic, and cubic plots
plt.plot(x, x, 'v-', x, x**2, 'x-', x, x**3, 'o-')
plt.savefig("functions.pdf")
plt.show()
```



Wherever you go in Python, NumPy is already there!

5.3 What are the Limitations of NumPy?

The focus of NumPy is working with numerical data. It's both powerful and low-level, thereby providing basic functionality for high-level algorithms. If you enter the machine learning and data science space, you want to master NumPy first. But eventually, you will use other libraries that operate on a higher level such as TensorFlow, Pandas, and scikit-learn. Those libraries contain out-of-the-box machine learning functions such as training and inference algorithms. Have a look at them after reading this tutorial.

Nevertheless, NumPy's handy functionality will definitely help you to use any off-the-shelf machine learning algorithms effectively. For example, it will typically allow you to pre-process your datasets and to post-process the predictions made by your trained machine-learning algorithms. If you do not understand the last sentence, don't worry. Simply stated, you will gradually find out that NumPy is often used alongside popular machine-learning and data-science libraries such as scikit-learn, TensorFlow, and Keras.

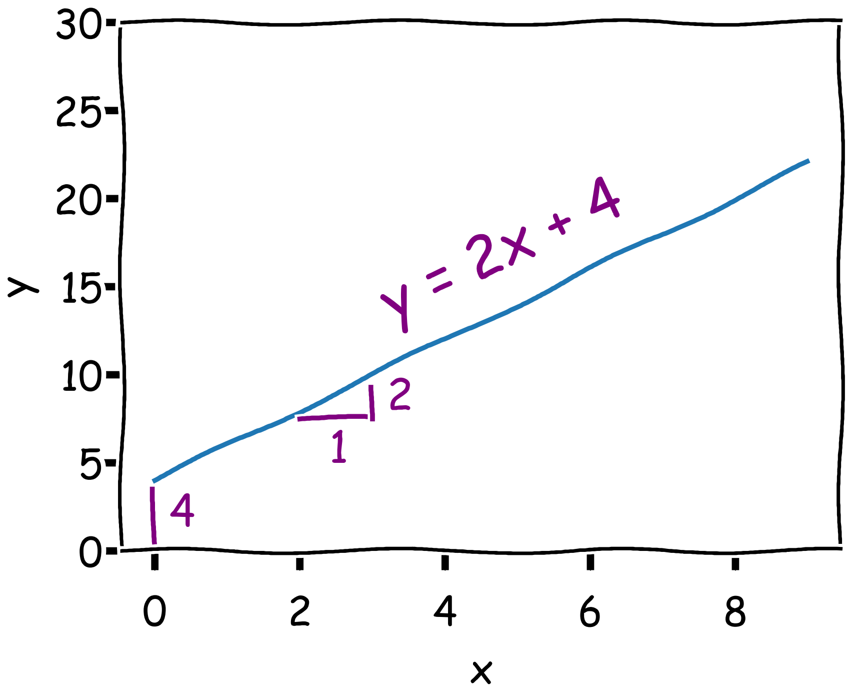
5.4 What are the Linear Algebra Basics You Need to Know?

NumPy is all about manipulating arrays. By learning NumPy, you will also learn and refresh your basic linear algebra skills from school. We will also repeat many concepts of linear algebra in this book. It's always better to learn the concepts first and the tools later. NumPy is only a specific tool that implements these concepts of linear algebra.

At the center of linear algebra stands the solution of linear equations. Here is one of those equations:

$$y = 2x + 4$$

If you plot this equation, you get the following output:



As you can see, the equation $y = 2x+4$ leads to a straight line on the space. This line defines a relationship between the values on the x-axis and the y-axis. Particularly, it allows the value of y to be determined for any given value of x.

Let me repeat this: You can determine the value of y for any given value of the input x.

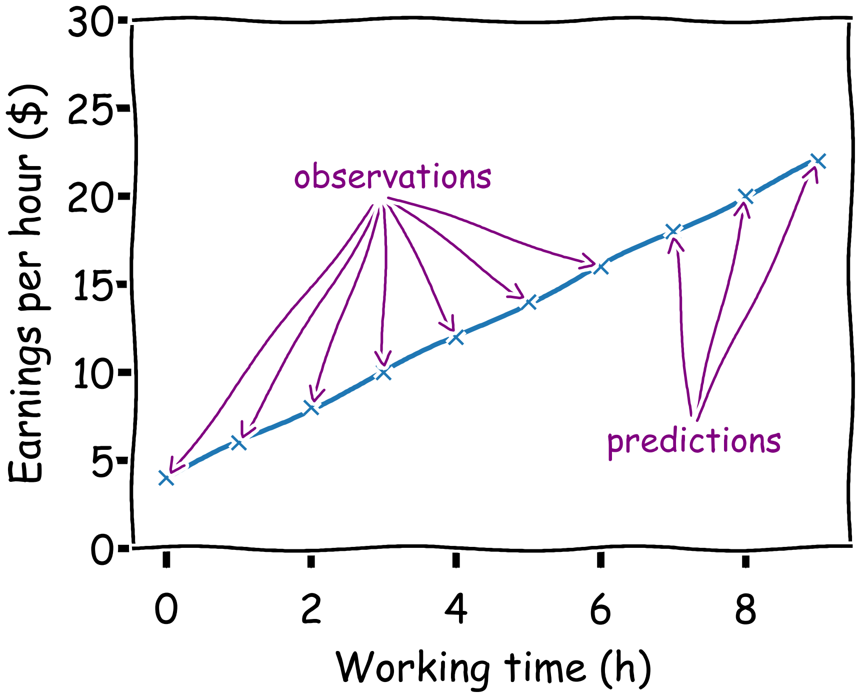
As it turns out, this is the very goal of any machine learning technique. From a bunch of data values belonging to certain variables (e.g, x and y), you want to find a function that describes the relationship between these variables. In machine learning this is called the learning phase. Subsequently you can use the learned function to "predict" the output value for any new input value. It works, even if you have never seen this input before. This second phase is called the inference phase.

Linear algebra helps you solve equations to do exactly that.

Here is an example with some fake data. Say, you have learned the relationship between the work ethics in number of hours worked per day and hourly wage in US dollars. Your learned relationship, also called a "model," is the equation $y = 2x + 4$. The input x is the number of hours worked per day and the output y is the hourly wage.

With this model, you can predict how much your boss earns based on the number of hours he or she invests in work. It works just like a machine: you put in x and get out y. This is what machine learning is all about:

learning a model representing the relationship between different variables using data from past observations and, later, using this model for making precise predictions.



Here is the script that does this plot for us. For simplicity, we do not include the code that labels the data points as observations and predictions.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0., 10., 1)
```

```
# [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]

y = 2 * x + 4
# [ 4.  6.  8. 10. 12. 14. 16. 18. 20. 22.]

print(x)
print(y)

# xkcd-styled plot
plt.xkcd()

plt.plot(x, y, 'x-')
plt.xlabel("Working time (h)")
plt.ylabel("Earnings per hour ($)")
plt.ylim((0,30))

plt.tight_layout()
plt.savefig("simple_linear_equation_example.pdf")
plt.show()
```

As you can see, before doing anything else in the script, we have to import the NumPy library. Use the statement `import NumPy as np` to do so. Each time you want to call a NumPy function, you have to use the short-name 'np' (e.g., `np.average(x)`). In theory, you can specify any other short-name, but it is better not to do this. The name 'np' has crystallized as a convention for the

NumPy library, so every experienced coder will expect adherence to this convention.

After the initial import, we create a series of floating point values between 0 and 9. These values serve as the x values that we want to map to their respective function values $y = f(x)$. The variable x holds a NumPy array of those floating point values.

The variable y holds a NumPy array of the same size. It's our output—one for each observed x value. Do you see the basic arithmetic of how to get the y values?

The equation $y = 2x + 4$ seems to do the same thing as discussed in the previous equation. But as it turns out, the meaning is very different: x is not a numerical value, it is a NumPy array!

When calculating $y = 2x + 4$, we are basically multiplying the NumPy array by 2 and adding the constant 4 to it. These are basic mathematical operations on multi-dimensional NumPy arrays, not just single numerical values.

Investigating these kinds of operations lies at the core of linear algebra. The NumPy array in the example is called a one-dimensional matrix or a vector of scalar values. The matrix x consists of 10 floating point values between 0 and 9 (inclusive): [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]. How do we know that the values in the NumPy array are of type float? We indicate this by writing a dot "." after