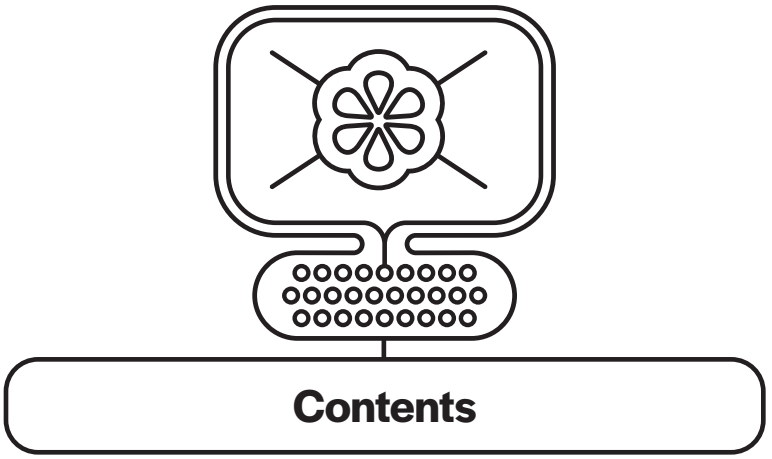
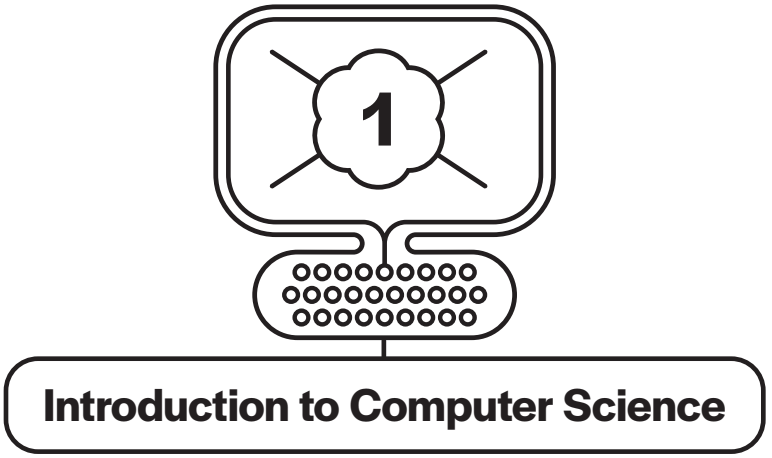


Contents



Congratulations on taking your first step into CS!

In this first unit, you'll learn what computer science (CS) is and where it came from, and start to learn about how it works. You'll get set up with the tools that we'll be using in the following chapters to help you become a proficient computer scientist by the end of the book.

CS topics in this chapter:

- Algorithms
- Comments
- Output

1.1 WELCOME TO CS!

CS IS PROBLEM SOLVING

If you like solving problems, you have come to the right place! In CS, we solve problems by designing and developing two components:

- **Algorithms**, which are a way of thinking
- **Code**, which is a way of communicating

In this book, we will be designing our algorithms in English, and translating them into the Python programming language. This will allow us to communicate with computers to solve our problems.

WHAT ARE ALGORITHMS?

An algorithm is simply a list of steps to follow in order to complete a task. For example, cookie recipes are algorithms. They have ingredients as input, and a list of steps to produce a very tasty output: cookies! Another example is IKEA furniture instructions. If you can write clear, step-by-step instructions (e.g., how to build a chair), you've got great potential as a computer scientist!

Algorithms can also be **optimized** for different things. You may want to design instructions for how to do the task as fast as possible, or in a simple way, or to minimize the space needed. For instance, in our cookie recipe example, some recipe titles might be "Chocolate chip cookies in 15 minutes" (fast) or "Easy chocolate chip cookies" (simple) or "One-bowl chocolate chip cookies" (minimizing space), optimizing for different needs. There are often multiple recipes, or algorithms, to achieve a similar result.

WHAT ARE PROGRAMMING LANGUAGES?

The computer is our fast, number crunching tool to solve problems for us. In order to communicate with the computer, we need to speak a language it understands. Programming languages have been developed to be a common language that both humans and the computer can interpret and understand.

Python, Java, and C++ are names of some programming languages. Just like English, Japanese, French, and other natural languages, they are used to communicate meaning, and each has different grammar, syntax, and vocabulary.

We say that some programming languages, such as Python and JavaScript, are **interpreted** languages, while C, C++, and Java are **compiled** languages. What does compiled mean? When you run a program, your instructions to the computer are translated from the human-readable language such as Python or C++ into 0s and 1s, which are digital signals that the computer can understand.

Translation can in happen two ways: line-by-line or in a batch. To understand this better, imagine being at the United Nations. There are professional translators who sit next to each diplomat and translate everything being said in real time. These translators are called **interpreters**. At the same meeting, there may be thick piles of documents on each desk that have been **compiled** into big translated reports before the meeting.

In the same way, Python is an **interpreted** language because the Python interpreter on your computer translates each line of code into machine code in real time, on the fly, as the program runs on your computer. With C++, the entire program is completely **compiled** into machine code before anything runs.

Now that you know the difference between interpreted and compiled programming languages, can you list two examples of programming languages for each type?

WHAT IS PROGRAMMING?

Once we have a general algorithm design in mind, and have selected a programming language, we can begin to program! Programming itself is the process where you break a large, complex task down into smaller subtasks and constructs that are recognizable to a computer.

Common programming constructs include:

- Input
- Output
- Conditions
- Repetition
- Math or logic

We will explore all of these in the coming chapters!

DID YOU KNOW? The first programmer was an English mathematician named Lady Ada Lovelace. In 1842, she wrote the first computer program for Charles Babbage's Analytical Engine (1837). Her first program calculated Bernoulli numbers, but she also envisioned a future where the numbers she worked with could represent more than numbers. She believed that any data represented by numbers could be processed by a machine like the Analytical Engine, to compose music, create images, or do science. Today, with our modern computers, we know that she was right!

1.1.1 Learning Outcomes

At the end of this unit, you will be able to …

- write Python comments
- explain what pseudocode is
- describe the main characteristics of an algorithm
- give an example of problem solving by subdividing tasks in to subtasks
- write a program header block (i.e., initial comments with information about the author, date, and purpose of the code)
- output `Hello World` using `print()`

1.1.2 Alien Explanations

Do you like science fiction? In this section, we'll deepen our understanding of algorithms by inventing an example that may seem strange, even alien, at first. All will be clear, soon!

Imagine that one day the Earth is visited by aliens from another planet. You have been chosen to communicate with the alien, which knows nothing about the human race. You meet the alien, which comically looks exactly like a pea-green octopus with a space helmet. Luckily, the alien happens to speak English. The first thing it says is, "I would like to learn about the human species. Please, tell me something about your daily life as a human."

You reply, "Well, we do this thing every day. We brush our teeth!"

The alien replies, "Please, human, explain 'brush our teeth.'"

EXERCISE

Explain to the alien the procedure to "brush one's teeth." You can either explain to another person nearby who takes the role of the alien (be as curious as possible!), or write out the explanation.

DEBRIEF

First off, it's not easy to explain something that comes so simply to us! You may have included the tools or input involved: a hand, a toothbrush … what else? Toothpaste? Mouth? Many components are involved. For an alien that does not have teeth or even hands, this requires patience and explanation in detail.

Secondly, what did your explanation look like? Was it a list of numbered steps? How detailed was it? Did you include aspects such as how long the brushing should last, or how much toothpaste you need? The clearer your steps were, the closer to algorithmic thinking they will be.

Spoiler alert: Your computer is just like this alien! It has no idea about the human world. We therefore need to be patient with it and explain our algorithms very clearly. Luckily, to communicate with our computer, we have the common language of Python, which we will learn in this book. Once we learn how to communicate clearly in Python, we can ask our computer to do what we want!

1.1.3 Writing and Running Your Programs

Throughout this book you'll see a lot of the following:

```
1  print("Hello World!")
```

These are code samples that you can copy and paste to a Python interpreter to try them out yourself or experiment with by making changes to it (the number on the left is the line number; you do not have to write it in your code). There are two main kinds of coding environments with Python interpreters you can use.

ONLINE CODING ENVIRONMENTS

Online coding environments are websites that provide a workspace for you to write and run your programs in an Internet browser window. Depending on the service provider, some offer saving your code under a user account and/or directly to your computer. Some also offer subscription plans with more features. Here are a few that we find quite useful and that are available on some of the most common operating systems (i.e., Windows, macOS, Linux).

Online Python (https://www.online-python.com) is an in-browser Integrated Development Environment (IDE) for Python. You can download your work as a `.py` or `.zip` file and can upload text files, too. You do not have to create an account to use its services, but it has some limitations, such as not being able to upload images. One great feature is that once you create a program, you can share a link to it with your friends and family. Online Python is a simple but good IDE to use.

Trinket (https://trinket.io) is another in-browser IDE that lets you write, run, and share your programs on its online platform. It also includes some lessons for users to learn about coding and provides some other ways to program (via Blocks). It has fewer features than the offline coding environments described below but still works well. You have to sign up for a free account with some limitations to use its services.

Programiz (https://www.programiz.com/) is a learning platform offering free and paid courses for various programming languages, including Python. Similarly to Trinket, it includes a free online compiler and lets you write, run, and share your code without signing up. Besides programming languages, it also offers courses on data structures and algorithms, which are important next steps if you are interested in CS.

OFFLINE CODING ENVIRONMENTS

One obvious shortcoming of the online coding environments is that you have to stay connected to the Internet to use them. If you want to write and run your programs on your

computer without worrying about access to the Internet, installing an offline application is the way to go. Note that we use Python 3 in this book.

Python IDLE (https://www.python.org/downloads) is from the official creator of the Python language. It is simple but has all the things you need to develop your programs in Python. When you download and install a Python version, it will install both the interpreter and an application called IDLE (Integrated Development and Learning Environment). Once installed, you can start IDLE and use the top menu to write and run your programs.

Mu (https://codewith.mu) is a Python IDE created for beginner programmers. It includes a few useful modules (e.g., `pygame`) that other IDEs might have to install separately, so one can start quickly without worrying about downloading extra modules. The idea is to minimize setup and downloads so that beginners can start as soon as they install the application. The application also has a few useful features for beginners to customize their editor and check their code.

PyCharm (https://www.jetbrains.com/pycharm) is a fully-fledged IDE for Python development. It has many development tools such as on-the-fly code analysis, a graphical debugger, a unit tester, and version control (it's completely okay if you don't know what these are), which are great for projects of bigger scale but might be excessive for beginners. There are two versions for download: a paid Professional version with other language support, and a free Community version, which is more than sufficient in most cases.

SUMMARY

There are also other coding environments and tools out there that we haven't covered, and those that we have covered may change over time. So it is up to you to decide which one you want to use. However, no matter which coding environment and tools you choose to use, the idea is the same: plan your algorithm, write the code for it, run it, and keep improving it—that's the way to learn how to code.

1.1.4 Motivational Quote Generator

Hooray! You have made it to the first programming example in this book! In many books, the first programming example you'll see is the "Hello World" program. But let's do something different here.

> "Hello World!" is an in-joke within computer scientists and programmers because this is typically the very first program they write when they learn a programming language. A Hello World program illustrates the basic syntax of the language by properly displaying a simple greeting message.

We all need a few words of wisdom or encouragement from time to time—this is why people like opening fortune cookies and putting up posters with motivational phrases. In this section, let's write a Motivational Quote Generator program that says something motivating, wise, or funny, if you like, to the user.

WHO IS THE USER? Programmers build applications for people to use it. The people who use your application are called users.

```
1  # Motivational Quote Generator
2  # Author:
3  # Date:
4
5  print("If you can dream it, you can do it.")
```

In the code sample above, let's identify several important parts of a program:

- Lines 1–3 are **comments** that are not interpreted or run by the computer. Instead, their purpose is to provide useful information about the code to the person reading or writing the code. In Python, each line of comment begins with the `#` symbol. In this book, we call the comments at the top of your file (typically containing information such as title, author, and date) the **header**.
- Line 4 is an empty line, which will be ignored by the computer. It is there to visually partition the code so that the code is easier for humans to read. You can have as many empty lines as you want, but it's best to use them with a purpose, and consistently.
- Line 5 is the code executed by the computer. It represents a single command: **print** a sentence to the screen. Try replacing the sentence in the sample code with other words (or even a combination of strings and numbers separated by commas). The sentence must be surrounded by quotation marks. If you want to print the quotation mark itself, you must precede it with a backslash, like this: `print("As they say, \" Practice makes perfect!\"")`.

Comments are important parts of the code. In addition to the header, we use comments to explain the code, leave notes, and provide documentation. If you want to have more than one line of comments, you can use **multiline docstrings** by surrounding the lines with three double quotes: `"""`.

1.1.5 Review Questions

Time to test how much you understand the content in this chapter! If you need to, feel free to go back and review.

THEORY AND UNDERSTANDING

- What are the two components of CS we'll study in this workbook?
- What is an example of an algorithm?
- What is the name of the programming language we'll use in this book? Is it interpreted or compiled?
- Where can we practice coding online? Offline?
- What goes into the header of a program?
- What does learning a programming language have in common with learning a natural language?

SYNTAX SELF-CHECK

What do the following functions and keywords mean?

- `print("Hello, World")`
- `print("I have", 1)`
- `#`
- `"a string"`
- ```
  """
  a multiline string
  another line
  """
  ```

1.1.6 Practice Exercises

CODING

Now it's time for you to practice by writing some code. When you are finished, you can go to the Solutions section on our companion website to compare your answers with ours. Note that there can be many answers to the same question, so don't worry if your code is not exactly the same as ours. For now, the important thing is that your code produce the results you expect.

DISPLAYING A SENTENCE

Write a line of code that outputs the following sentence to the screen:

```
Dr. Evil: Back in the 60's, we developed a sophisticated heat beam we called a "laser".
```

MOTIVATIONAL QUOTE GENERATOR

Write your own Python program displaying a motivation quote. Begin with a header that matches your desired title, your name, and the date. Then replace the quote with one of your choices based on the sample code we showed you earlier. Use the Run button to make sure it prints on the screen as expected.

1.1.7 Glossary

- **code (aka source code)**, the vocabulary used when writing a computer program. There are different kinds/levels of code, including pseudocode, which is easily readable by humans, programming code, which follows the rules of a specific language, and machine code, which is understood by the computer but typically not by a human.
- **pseudocode**, code that is written in plain language so that it can be read by humans. Pseudocode typically follows some structural conventions, such as line breaks and indentations, to indicate order of execution. It uses some standard keywords, such as `repeat` and `if-then`, and symbols, such as `+` and `:=`, to represent programming logic, but omits language-specific details such as variable declarations. Many algorithms are written in this form for computer scientists to quickly communicate their ideas without fixating on a particular programming language.
- **programming**, the process of (a) breaking a large, complex task into smaller and smaller tasks until they are simple enough to be performed with sequences of basic constructs, including input/output, repetition, conditionals and logic and (b) writing them using code.
```

- **algorithm**, a step-by-step list of instructions to complete a task or solve a problem. There are usually many different ways to complete the same task or solve the same problem, and the clearer each step is, the better (there are different ways to make an algorithm better, e.g., faster, or requiring less space).
- **header**, a block of comments providing some information about the program preceding the code. It includes the name(s) of the author(s), the creation and modification dates, and a high-level description of the program.
- **comment**, a special section adjacent to the code and written in plain English explaining what that code does. It is for human readers to understand the code and is indicated by a special symbol (`#` in Python) so the computer will ignore it when executing the code. It can be a single sentence or a short paragraph.
- **interpretation**, the process used by the computer to execute the source code. For an interpreted programming language, the source code is translated directly into machine code during execution. Python and JavaScript are examples of this.
- **compilation**, the other process used by the computer to execute the source code, where the source code is first translated into machine/object (aka compiled) code and then executed. C/C++ and Java are examples of compiled programming languages.

Index

Pages in italics indicate figures