1.2. Algorithms

An **algorithm** is a step by step list of instructions that if followed exactly will solve the problem under consideration.

Our goal in programming is to take a problem and develop an algorithm that can serve as a general solution. Once we have such a solution, we can express it as a program and use our computer to automate the execution. These programs are written in **programming languages**.

**Check your understanding**

intro-2-1: An algorithm is:

Top of Form

A. A solution to a problem that can be solved by a computer.  
B. A step by step sequence of instructions that if followed exactly will solve the problem under consideration.  
C. A series of instructions implemented in a programming language.  
D. A special kind of notation used by programmers.

1.3. The Python Programming Language

The programming language you will be learning is Python. Python is an example of a **high-level language**; other high-level languages you might have heard of are C++, PHP, and Java.

As you might infer from the name high-level language, there are also **low-level languages**, sometimes referred to as machine languages or assembly languages. Loosely speaking, computers can only execute programs written in low-level languages. Thus, programs written in a high-level language have to be processed before they can run. This extra processing takes some time, which is a small disadvantage of high-level languages. However, the advantages to high-level languages are enormous.

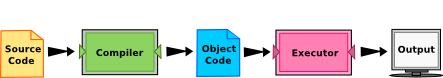
First, it is much easier to program in a high-level language. Programs written in a high-level language take less time to write, they are shorter and easier to read, and they are more likely to be correct. Second, high-level languages are **portable**, meaning that they can run on different kinds of computers with few or no modifications. Low-level programs can run on only one kind of computer and have to be rewritten to run on another.

Due to these advantages, almost all programs are written in high-level languages. Low-level languages are used only for a few specialized applications.

Two kinds of programs process high-level languages into low-level languages: **interpreters** and **compilers**. An interpreter reads a high-level program and executes it, meaning that it does what the program says. It processes the program a little at a time, alternately reading lines and performing computations.



A compiler reads the program and translates it completely before the program starts running. In this case, the high-level program is called the **source code**, and the translated program is called the **object code** or the **executable**. Once a program is compiled, you can execute it repeatedly without further translation.



Many modern languages use both processes. They are first compiled into a lower level language, called **byte code**, and then interpreted by a program called a **virtual machine**. Python uses both processes, but because of the way programmers interact with it, it is usually considered an interpreted language.

For the core material in this book, you will not need to install or run python natively on your computer. Instead, you’ll be writing simple programs and executing them right in your browser.

At some point, you will find it useful to have a complete python environment, rather than the limited environment available in this online textbook. To do that, you will either install python on your computer so that it can run natively, or use a remote server that provides either a command line shell or a jupyter notebook environment.

**Check your understanding**

intro-3-1: Source code is another name for:

Top of Form

A. the instructions in a program, written in a high-level language.  
B. the language that you are programming in (e.g., Python).  
C. the environment/tool in which you are programming.  
D. the number (or “code”) that you must input at the top of each program to tell the computer how to execute your program.

Bottom of Form

intro-3-2: What is the difference between a high-level programming language and a low-level programming language?

Top of Form

A. It is high-level if you are standing and low-level if you are sitting.  
B. It is high-level if you are programming for a computer and low-level if you are programming for a phone or mobile device.  
C. It is high-level if the program must be processed before it can run, and low-level if the computer can execute it without additional processing.  
D. It is high-level if it easy to program in and is very short; it is low-level if it is really hard to program in and the programs are really long.

Bottom of Form

intro-3-3: Pick the best replacements for 1 and 2 in the following sentence: When comparing compilers and interpreters, a compiler is like 1 while an interpreter is like 2.

Top of Form

A. 1 = a process, 2 = a function  
B. 1 = translating an entire book, 2 = translating a line at a time  
C. 1 = software, 2 = hardware  
D. 1 = object code, 2 = byte code

Bottom of Form

Assignment:

1. Print your name
2. Print 10 + 10

1.4. Comments

As programs get bigger and more complicated, they get more difficult to read. Formal languages are dense, and it is often difficult to look at a piece of code and figure out what it is doing, or why. For this reason, it is a good idea to add notes to your programs to explain in natural language what the program is doing. These notes are called comments.

A **comment** in a computer program is text that is intended only for the human reader - it is completely ignored by the interpreter. In Python, the # token starts a comment. The rest of the line is ignored. Here is a new version of *Hello, World!*.

#---------------------------------------------------

# This demo program shows off how elegant Python is!

# Written by Joe Soap, December 2010.

# Anyone may freely copy or modify this program.

#---------------------------------------------------

​

print("Hello, World!") # Isn't this easy!

​

Notice that when you run this program, it still only prints the phrase Hello, World! None of the comments appear. You’ll also notice that we’ve left a blank line in the program. Blank lines are also ignored by the interpreter, but comments and blank lines can make your programs much easier for humans to parse. Use them liberally!

**Check your understanding**

intro-10-1: What are comments for?

Top of Form

A. To tell the computer what you mean in your program.  
B. For the people who are reading your code to know, in natural language, what the program is doing.  
C. Nothing, they are extraneous information that is not needed.  
D. Nothing in a short program. They are only needed for really large programs.

# 1.5. Python Convention

See [PEP 8 – Style Guide for Python Code](https://www.python.org/dev/peps/pep-0008/)

Some of them are:

* Import statements go at the top, and each have their own line
* Indent code using spaces instead of tabs
* Use four spaces for each indentation level
* Limit lines to 79 characters(72 for docstrings/comments)
* Separate functions and classes by two blank lines.
* Within classes, separate methods by one blank line
* No spaces around function calls, indexes, keyword arguments

|  |  |
| --- | --- |
| Do | Don’t |
| spam(ham[1], {eggs: 2}) | spam( ham[ 1 ], {egg: 2 }) |
| fn(arg) | fn (arg) |
| dct[‘key’] = lst[index] | dct [‘key’] = lst [index] |
| x = 1  y = 1  long\_variable = 3 | x = 1  y = 1  long\_variable = 3 |
| hypot = x\*x + y\*y | hypot = x \* x + y \* y |
| i = i + 1 | i=i+1 |