Algorithms

If problem solving is a central part of computer science, then the solutions that you create through the problem solving process are also important. In computer science, we refer to these solutions as **algorithms**. An algorithm is a step by step list of instructions that if followed exactly will solve the problem under consideration.

Our goal in computer science is to take a problem and develop an algorithm that can serve as a general solution. Once we have such a solution, we can use our computer to automate the execution. As noted above, programming is a skill that allows a computer scientist to take an algorithm and represent it in a notation (a program) that can be followed by a computer. These programs are written in **programming languages**.

Check your understanding

intr-1: What is the most important skill for a computer scientist?

- a) To think like a computer.
- b) To be able to write code really well.
- (•) c) To be able to solve problems.
- d) To be really good at math.

Check Me | Compare Me

intr-2: An algorithm is:

- a) A solution to a problem that can be solved by a computer.
- (•) b) A step by step list 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 computer scientists.

Check Me | Compare Me

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.

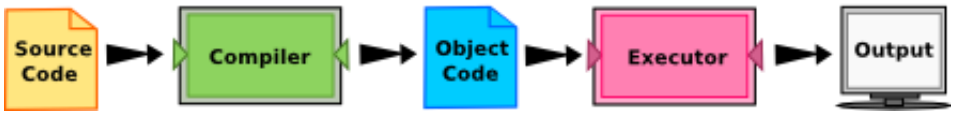
As you might infer from the name high-level language, there are also **low-level languages**, sometimes referred to as machine languages or assembly languages. Loosely speaking, computers can only execute programs written in low-level languages. Thus, programs written in a high-level language have to 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.

There are two ways to use the Python interpreter: *shell mode* and *program mode*. In shell mode, you type Python expressions into the **Python shell**, and the interpreter immediately shows the result. The example below shows the Python shell at work.

```
$ python3
Python 3.2 (r32:88445, Mar 25 2011, 19:28:28)
[GCC 4.5.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 3
5
>>>
```

The `>>>` is called the **Python prompt**. The interpreter uses the prompt to indicate that it is ready for instructions. We typed `2 + 3`. The interpreter evaluated our expression and replied `5`. On the next line it gave a new prompt indicating that it is ready for more input.

Working directly in the interpreter is convenient for testing short bits of code because you get immediate feedback. Think of it as scratch paper used to help you work out problems.

Alternatively, you can write an entire program by placing lines of Python instructions in a file and then use the interpreter to execute the contents of the file as a whole. Such a file is often referred to as **source code**. For example, we used a text editor to create a source code file named `firstprogram.py` with the following contents:

```
print("My first program adds two numbers, 2 and 3:")
print(2 + 3)
```

By convention, files that contain Python programs have names that end with `.py`. Following this convention will help your operating system and other programs identify a file as containing python code.

```
$ python firstprogram.py
My first program adds two numbers, 2 and 3:
5
```

These examples show Python being run from a Unix command line. In other development environments, the details of executing programs may differ. Also, most programs are more interesting than this one.

Want to learn more about Python?

If you would like to learn more about installing and using Python, here are some video links. Installing Python for Windows (http://youtu.be/9EfGpN1Pnsg) shows you how to install the Python environment under Windows Vista, Installing Python for Mac (http://youtu.be/MEmEJCLLI2k) shows you how to install under Mac OS/X, and Installing Python for Linux (http://youtu.be/RLPYBxfAud4) shows you how to install from the Linux command line. Using Python (http://youtu.be/kXbpB5_ywDw) shows you some details about the Python shell and source code.

Check your understanding

intr-3: Source code is another name for:

- (•) a) the instructions in a program, stored in a file.
- ( ) 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.

Check Me | Compare Me

intr-4: What is the difference between a high-level programming language and a low-level programming language?

- ( ) 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.

Check Me | Compare Me

intr-5: 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.

- ( ) 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

Check Me | Compare Me

More About Programs

A **program** is a sequence of instructions that specifies how to perform a computation. The computation might be something as complex as rendering an html page in a web browser or encoding a video and streaming it across the network. It can also be a symbolic computation, such as searching for and replacing text in a document or (strangely enough) compiling a program.

The details look different in different languages, but a few basic instructions appear in just about every language.

input
Get data from the keyboard, a file, or some other device.

output
Display data on the screen or send data to a file or other device.

math and logic
Perform basic mathematical operations like addition and multiplication and logical operations like `and`, `or`, and `not`.

conditional execution
Check for certain conditions and execute the appropriate sequence of statements.

repetition
Perform some action repeatedly, usually with some variation.

Believe it or not, that's pretty much all there is to it. Every program you've ever used, no matter how complicated, is made up of instructions that look more or less like these. Thus, we can describe programming as the process of breaking a large, complex task into smaller and smaller subtasks until the subtasks are simple enough to be performed with sequences of these basic instructions.

Check your understanding

intr-8: A program is:

- (•) a) a sequence of instructions that specifies how to perform a computation.
- ( ) b) something you follow along at a play or concert.
- ( ) c) a computation, even a symbolic computation.
- ( ) d) the same thing as an algorithm.

Check Me | Compare Me

Special Ways to Execute Python

There are two additional ways to execute Python programs. Both techniques are designed to assist you as you learn the Python programming language. They will help you increase your understanding of how Python programs work.

First, you can write, modify, and execute programs using a unique **activecode** interpreter that allows you to execute Python code right in the text itself (right from the web browser). Although this is certainly not the way real programs are written, it provides an excellent environment for learning a programming language like Python since you can experiment with the language as you are reading.

Take a look at the activecode interpreter in action. If we use the Python code from the previous example and make it active, you will see that it can be executed directly by pressing the *run* button. Try pressing the *run* button below.

```
1 print("My first program multiplies two numbers, 2 and 3:")
2 print(2 * 3)
3
```

ActiveCode: 1 (ch01_1)

Run

```
My first program multiplies two numbers, 2 and 3:
6
```

Now try modifying the activecode program shown above. First, modify the string in the first print statement by changing the word *adds* to the word *multiplies*. Now press *run*. You can see that the result of the program has changed. However, it still prints “5” as the answer. Modify the second print statement by changing the addition symbol, the “+”, to the multiplication symbol, “*”. Press *run* to see the new results.

In addition to activecode, you can also execute Python code with the assistance of a unique visualization tool. This tool, known as **codelens**, allows you to control the step by step execution of a program. It also lets you see the values of all variables as they are created and modified. The following example shows codelens in action on the same program as we saw above. Note that in activecode, the source code executes from

beginning to end and you can see the final result. In codelens you can see and control the step by step progress. Note that the red arrow always points to the next line of code that is going to be executed. The light green arrow points to the line that was just executed.

```
1  print("My first program adds two numbers, 2 and 3:"
2  print(2 + 3)
```

<< First | < Back | Step 1 of 2 | Forward > | Last >>

line that has just executed

next line to execute

Frames Objects

CodeLens: 1 (firstexample)

The examples in this course use a mixture of the standard Python interpreter, source code, activecode, and codelens. You will be able to tell which is which by looking for either the Python prompt in the case of a shell mode program, the *run* button for the activecode, or the *forward/backward* buttons for codelens.

Check your understanding

intr-6: The activecode interpreter allows you to (select all that apply):

- [x] a) save programs and reload saved programs.
- [x] b) type in Python source code.
- [x] c) execute Python code right in the text itself within the web browser.
- [ ] d) receive a yes/no answer about whether your code is correct or not.

Check Me | Compare Me

Correct!
a: You can (and should) save the contents of the activecode window.
b: You are not limited to running the examples that are already there. Try adding to them and creating your own.
c: The activecode interpreter will allow you type Python code into the textbox and then you can see it execute as the interpreter interprets and executes the source code.

intr-7: Codelens allows you to (select all that apply):

- [ ] a) measure the speed of a program's execution.
- [x] b) control the step by step execution of a program.
- [ ] c) write and execute your own Python code.
- [x] d) execute the Python code that is in codelens.

Check Me | Compare Me

Correct!
b: By using codelens, you can control the execution of a program step by step. You can even go backwards!
d: By stepping forward through the Python code in codelens, you are executing the Python program.

© Copyright 2013 Brad Miller, David Ranum, Created using Runestone Interactive.