Chapter 1

Introducing Python

In This Chapter

- The history of Python
- What people use Python for
- Useful concepts for Python programming

Welcome to Python! If you're the type of person who wants to know what you're getting into, this chapter is for you. We give you a quick history of Python and its community of developers. You find out what Python is and isn't good for (the "is" section is much longer than the "isn't" section) and the most important principles of good Python programming. If you're new to programming, you'll see how it's very similar to a task you're probably familiar with.

The Right Tool for the Job

Python is a general-purpose, high-level language that can be extended and embedded (included in applications as a tool for writing macros). That makes Python a smart choice for many programming problems, both small and large, and not so good for a couple of computing tasks.

Good uses of Python

Python is ideal for projects that require quick development. It supports multiple programming philosophies, so it's good for programs that require flexibility. The many packages and modules already written for Python provide versatility and save you time.

The story of Python

Guido van Rossum created Python and is affectionately bestowed with the title "Benevolent Dictator For Life" by the Python community. In the late 1980s, Guido liked features of several programming languages, but none of them had all the features he wanted. Specifically, he wanted a language that had the following features:

- **Scripting language:** A *script* is a program that controls other programs. Scripting languages are good for quick development and prototyping because they're good at passing messages from one component to another and at handling fiddly stuff like memory management so that the programmer doesn't have to. Python has grown beyond scripting languages, which are used mostly for small applications. The Python community prefers to call Python a *dynamic programming language*.
- **Indentation for statement grouping:** Python specifies that several statements are part of a single group by indenting them. The indented group is called a *code block*. Other languages use different syntax or punctuation for statement grouping. For example, the C programming language uses `{` to begin an instruction and `}` to end it. Indentation is considered good practice in other languages also, but Python was one of the first to *enforce* indentation. Indentation makes code easier to read, and code blocks set off with indentation have fewer begin/end words and punctuation to accidentally leave out (which means fewer bugs).
- **High-level data types:** Computers store everything in 1s and 0s, but humans need to work with data in more complex forms, such as text. A language that supports such complex data is said to have *high-level data types.* A high-level data type is easy to manipulate. For example, Python strings can be searched, sliced, joined, split, set to upper- or lowercase, or have white space removed. High-level data types in Python, such as lists and dicts (which can store other data types), encompass much more functionality than in other languages.
- **Extensibility:** An extensible programming language can be added to. These languages are very powerful because additions make them suitable for multiple applications and operating systems. Extensions can add data types or concepts, modules, and plug-ins. Python is extensible in several ways. A core group of programmers works on modifying and improving the language, while hundreds of other programmers write modules for specific purposes.
- **Interpreted:** Interpreted languages run directly from source code that humans generate (whereas programs written in *compiled languages,* like C++, must be translated to machine code before they can run). Interpreted languages run more slowly because the translation takes place on the fly, but development and debugging is faster because you don't have to wait for the compiler. Interpreted languages are easier to run on multiple operating systems. In the case of Python, it's easy to write code that works on multiple operating systems — with no need to make modifications.

  People argue over whether Python is an interpreted or compiled language. Although Python works like an interpreted language in many ways, its code is compiled before execution (like Java), and many of its capabilities run at full machine speed because they're written in C — leaving you free to focus on making your application work.

Guido began writing Python during his Christmas vacation in 1989, and over the next year, he added to the program based on feedback from colleagues. He released it to the public in February 1991 by posting to the Usenet system of newsgroups. In Guido's words: "The rest is in the Misc/HISTORY file."

Fast development

High-level features make Python a wise alternative for prototyping and fast development of complex applications:

- Python is interpreted, so writing working programs and fixing mistakes in programs is fast.

  Programs written in interpreted languages can be tested as soon as they're written, without waiting for the code to compile.
- Python takes care of such fiddly details as memory management behind the scenes.
- Python has debugging features built in.

All these features make Python a good language for

- Off-the-cuff, quick programming
- Prototyping (sketching the design basics of complex programs, or testing particular solutions)
- Applications that change, build on themselves, and add new features frequently

Programming styles

Python is a *multi-paradigm* language (meaning it supports more than one style or philosophy of programming). This makes it good for applications that benefit from a flexible approach to programming. Python includes tools for the following paradigms:

- Object-oriented programming (OOP for short) is one of the popular programming styles that Python supports. OOP breaks up code into individual units that pass messages back and forth.

  Object-oriented programming is good for applications that have multiple parts that need to communicate with each other.
- Python has features in common with the following languages. If you know these languages, you'll find features in Python that you are familiar with, making Python easier to learn:

- **Java:** An object-oriented language especially for applications used over networks
- **Perl:** A procedural language used for text manipulation, system administration, Web development, and network programming
- **Tcl:** Used for rapid prototyping, scripting, GUIs, and testing
- **Scheme:** A functional programming language (a language that focuses on performing actions and calculations by using functions.

  For more about functions, see Chapter 11, and for an intro to functional programming, see Chapter 16.)

Python For Dummies includes a brief introduction to object-oriented programming (Chapter 13), an overview of using Python for Web development (Chapter 20), and tips for scripting and testing.

Versatility

Python *modules* (collections of features for performing tasks) let Python work with

- **Multiple operating systems and user interfaces**

  With *Python For Dummies*, you can write and run programs on Windows, Mac, and Unix (including Linux). Python programmers have also written code for other operating systems, from cell phones to supercomputers.
- **Special kinds of data (such as images and sound)**

Python comes with dozens of built-in modules. New modules can be written in either Python or C/C++.

Companies that use Python

The main portal to Python and the Python community is `www.python.org`. This portal contains a page that lists some companies that use Python, including

- Yahoo! (for Yahoo! Maps)
- Google (for its spider and search engine)
- *Linux Weekly News* (published by using a Web application written in Python)
- Industrial Light & Magic (used in the production of special effects for such movies as *The Phantom Menace* and *The Mummy Returns*).

Other commercial uses include financial applications, educational software, games, and business software.

Convenience

Most programming languages offer convenience features, but none boast the combination of convenience and power that Python offers:

- **Python can be embedded in other applications and used for creating macros.** For example, Python is embedded in Paint Shop Pro 8 and later versions as a scripting language.
- **Python is free for anyone to use and distribute (commercially or non-commercially),** so any individual or company can use it without paying license fees.
- **Python has powerful text manipulation and search features for applications that process a lot of text information.**
- **You can build large applications with Python, even though it doesn't check programs before they run.** In technical terms, Python doesn't have *compile-time checking*. Python supports large programs by connecting multiple modules together and bundling them into packages. Each module can be built and tested separately.
- **Python includes support for testing and error-checking both of individual modules and of whole programs.**

Sometimes, Python isn't so hot

Python by itself isn't best for applications that need to interface closely with the computer's hardware because

- **Python is an *interpreted language*.**

  Interpreted languages are slower than compiled languages.
- **Python is a *high-level language*** that uses many layers to communicate with the computer's hardware and operating system.

Python might not be the best choice for building the following types of applications and systems:

- **Graphics-intensive applications, such as action games**

  But some games use Python because specialized modules can be written to interface with hardware. The `pygame` module is one such package. (Modern computers are extremely fast, which means it's more important to be able to write clean code quickly than to get maximum speed out of the software, except for the most graphics-intensive games.)
- **The foundations of an operating system**

The Python developer community

Python has attracted many users who collectively make up a community that

- Promotes Python
- Discusses and implements improvements to the language
- Supports newcomers
- Encourages standards and conventions that improve Python's usability and readability
- Values *simplicity* and *fun* (after all, Python was named after Monty Python, the British comedy troupe)

The Python community has created words to describe its philosophy:

Pythonic identifies code that meets the following criteria:

It includes interfaces or features that work well with Python.

It makes good use of Python *idioms* (standard ways of performing tasks) and shows understanding of the language.

Unpythonic code is roughly translated from other languages instead of following Python's philosophy.

Pythonistas are knowledgeable users of Python (especially users who promote the language).

Cooking Up Programs

Writing programs is a little bit like working with recipes. For example, you can

- **Write a recipe to make bread from scratch.**

  In Python, you can build a program from scratch, writing all your own code and using only Python's basic built-in functions.

- **Use the product of one recipe in another recipe (for example, a recipe for turkey stuffing uses bread as an ingredient).**

  After you write program that performs a basic task, you can insert it into other programs the same way you add any ingredient to a recipe.

- **Buy premade bread.**

  Python comes with many *modules,* which are sets of programs other people have written that you can plug into your program, just like you can buy bread at the store without baking it yourself.

Python's even better than bread because most Python modules are *free!*

When you write a program, you are telling the computer to do something. *Python For Dummies* gives you step-by-step instructions that help you understand how to write the way a computer "thinks."

Unlike you, computers are pretty stupid. They can do only a few things. All the actions that humans make them do are the result of the computer's doing those few things over and over, in different combinations, very quickly.

Training your assistant

Imagine that you're a baker, and you have taken on an apprentice baker who is as stupid as a computer. If you want to show your baker how to make bread from scratch, you need to start with very basic steps. You've already started by putting warm water and sugar in a small bowl. Then you and the apprentice have this conversation:

You: "Add a package of yeast."

Apprentice: "I can't find a package of yeast."

You: "The refrigerator is over there. Inside the refrigerator is a little package labeled *Yeast.* Go get it."

The apprentice gets the package and says, "Now what?"

You: "Put the package in the bowl."

The apprentice puts the package in the bowl.

You: "Hey! Open the package first!"

By now you might doubt the wisdom of hiring an apprentice baker who needs to be told things that seem completely obvious to you. But if you persevere, you'll come out ahead. If this apprentice is like a computer, then after finally figuring out how to bake bread in your kitchen, your new baker will be able to prepare 100 loaves a minute!

Combining ingredients

When your apprentice baker knows all the procedures involved in baking bread, such as finding the ingredients on the shelves, finding the pots and pans, mixing ingredients, and operating the oven, you can assign other tasks that use those same procedures. Baking bread involves combining ingredients in a bowl, so if you need to combine ingredients for another recipe, the

apprentice already knows how to do that. So when you want to explain how to make cookies, you can now say "combine sugar, flour, and butter in a bowl" without explaining where to find the bowls or the sugar.

In Python, after you've written a program to do something, you can import it into another program. So the more you work with Python, the faster you'll be able to write programs.