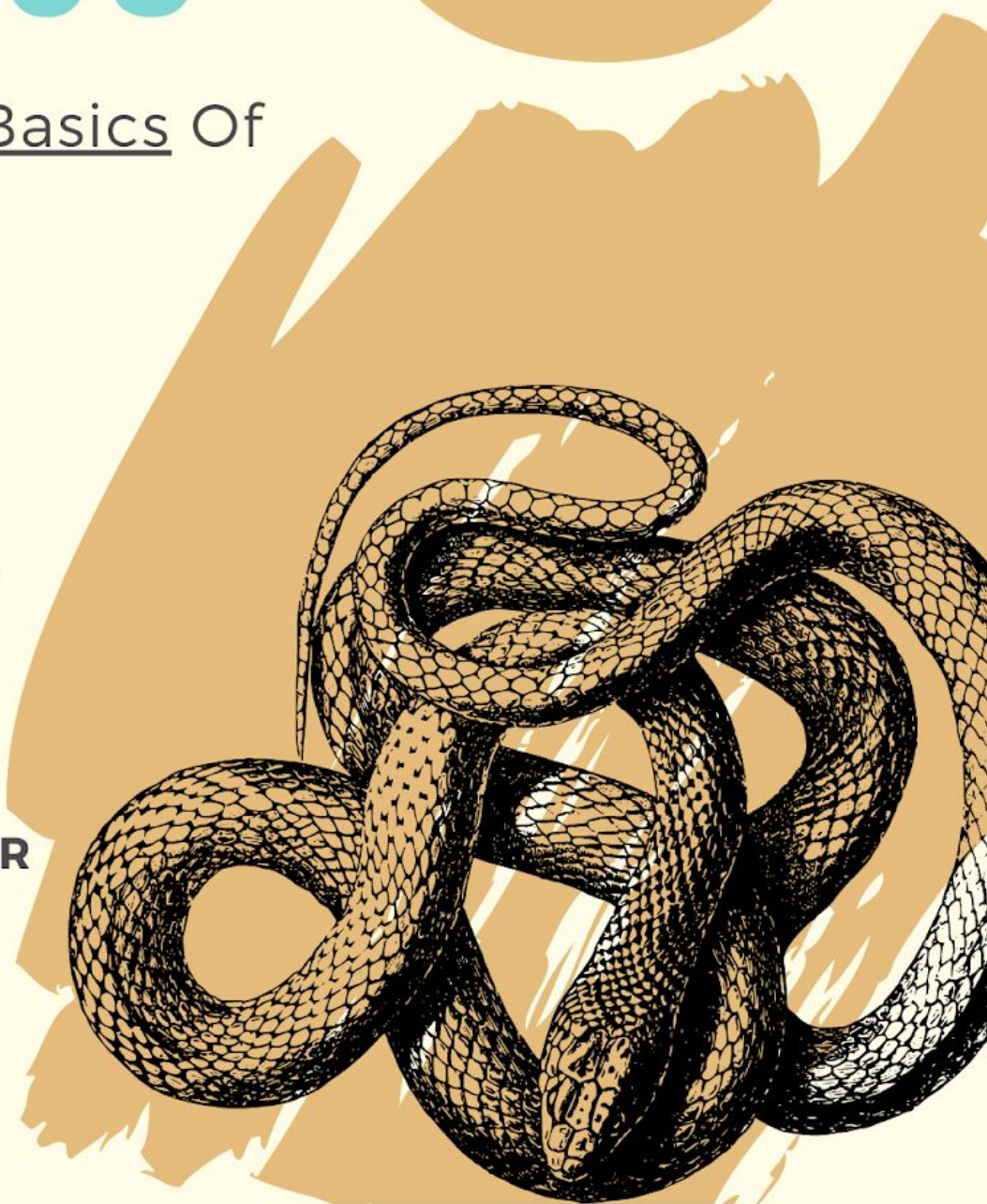
THE GREAT COMPANY BRINGS YOU

PYTHON BASICS

Master The Basics Of *Python*



VERSION:
10.06.2020

MADE WITH

BY
HISHAM EL-AMIR

Python Basics

Master The Basics Of Python

By
Hisham El-Amir

Python Basics, First Edition 10.10.2020
by Hisham El-Amir

Published @ leanpub.

Table Of Content

Preface

Just as Python 3 is about the future, this edition of the Python Basics represents a major change over this edition. First and foremost, this is meant to be a very forward looking book. All of the recipes have been written and tested with Python 3.3 without regard to past Python versions or the “old way” of doing things. In fact, many of the recipes will only work with Python 3.3 and above. Doing so may be a calculated risk, but the ultimate goal is to write a book of recipes based on the most modern tools and idioms possible. It is hoped that the recipes can serve as a guide for people writing new code in Python 3 or those who hope to modernize existing code.
Needless to say, writing a book of recipes in this style presents a certain editorial challenge. An online search for Python recipes returns literally thousands of useful recipes on sites such as ActiveState’s Python recipes or Stack Overflow. However, most of these recipes are steeped in history and the past. Moreover, they often use outdated techniques that have simply become a built-in feature of Python 3.3. Finding recipes exclusively focused on Python 3 can be a bit more difficult.
Rather than attempting to seek out Python 3-specific recipes, the topics of this book are merely inspired by existing code and techniques. Using these ideas as a springboard, the writing is an original work that has been deliberately written with the most modern Python programming techniques possible. Thus, it can serve as a reference for anyone who wants to write their code in a modern style.

In choosing which recipes to include, there is a certain realization that it is simply impossible to write a book that covers every possible thing that someone might do with Python. Thus, a priority has been given to topics that focus on the core Python language as well as tasks that are common to a wide variety of application domains. In addition, many of the recipes aim to illustrate features that are new to Python 3 and more likely to be unknown to even experienced programmers using older versions. There is also a certain preference to recipes that illustrate a generally applicable programming technique (i.e., programming patterns) as opposed to those that narrowly try to address a very specific practical problem. Although certain third-party packages get coverage, a majority of the recipes focus on the core language and standard library.

Who This Book Is For

This book is aimed at more experienced Python programmers who are looking to deepen their understanding of the language and modern programming idioms. Much of the material focuses on some of the more advanced techniques used by libraries, frameworks, and applications. Throughout the book, the recipes generally assume that the reader already has the necessary background to understand the topic at hand (e.g., general knowledge of computer science, data structures, complexity, systems programming, concurrency, C programming, etc.). Moreover, the recipes are often just skeletons that aim to provide essential information for getting started, but which require the reader to do more research to

fill in the details. As such, it is assumed that the reader knows how to use search engines and Python's excellent online documentation.
Many of the more advanced recipes will reward the reader's patience with a much greater insight into how Python actually works under the covers. You will learn new tricks and techniques that can be applied to your own code.

Who This Book Is Not For

This is not a book designed for beginners trying to learn Python for the first time. In fact, it already assumes that you know the basics that might be taught in a Python tutorial or more introductory book. This book is also not designed to serve as a quick reference manual (e.g., quickly looking up the functions in a specific module). Instead, the book aims to focus on specific programming topics, show possible solutions, and serve as a springboard for jumping into more advanced material you might find online or in a reference.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic. Indicates new terms, URLs, email addresses, filenames, and file extensions.
`Constant width`. Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.
Oswald. Shows all important indexes for instance the table info, figure info, etc...

Online Code Examples

Almost all of the code examples in this book are available online at [github link soon]. The authors welcome bug fixes, improvements, and comments.

How to Contact Us

Please address comments and questions concerning this book to the publisher:
Hisham El-Amir EMail: hishamelamir001@gmail.com

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at [dataisutopia webpage].

Chapter 1: Introduction to Python

If you've bought this book, you may already know what Python is and why it's an important tool to learn. And even if you don't know anything about Python, I guarantee that by finishing this book, you will get to know many things about this beautiful language.

But before we jump into details, this first chapter of this book will briefly introduce some of the main reasons behind Python's popularity. And before getting to the code, we need to convince you why particularly choosing Python to learn, and what we will do in this chapter.

Reason to Use Python.

Because there are many programming languages available today, this is the usual first question of newcomers.
And for the newcomers, I will state a fact. In 2018~2020, there are about *8.2* million developers in the world who code using Python and that population is now larger than those who build in Java, who number 7.6 million.
But, it is always optional when it comes to the choice of development tools which is sometimes based on unique constraints or personal preference.
But after teaching Python to many students as I can count during the last few years, I have seen some common themes emerge. I can say that there are some factor are common to Python users, and if I collected them, they will be:

Software perspective. After you learn Python, you will see that Python compared to any other programming language, it will win in terms of readability, coherence.
So from the software perspective, a programming language that increases code reusability and maintainability in general will be considered a perfect one. Python code is designed to be readable, and hence reusable and maintainable—much more so than traditional scripting languages.

The uniformity of Python code makes it easy to understand, which makes it look like pseudo-code, even if you did not write it. In addition, Python has deep support for more advanced software reuse mechanisms, such as *object-oriented programming* (OOP) and *functional programming*.

Productivity. Compared to compiled languages such as C, C++ and Java, Python boosts developer productivity way beyond your imagination. And the main reason is that Python code is typically less in size compared to C++ or Java code. That means there is less to type, less to debug, and less to maintain.

As scripting language, Python programs also run immediately, without the lengthy compile and link steps required by some other tools, further boosting programmer speed.

Portability. You can write one Python code, and run it on many different platforms, porting Python code between *Linux* and *Windows*, for example, is usually just a matter of copying a script's code between machines.

Moreover, Python offers multiple options for coding portable *graphical user interfaces* (GUI), *database* access programs, *web-based* systems, and more.

Support. Python comes with a large collection of prebuilt and portable functionality, known as the standard library. This library supports an array of application-level programming tasks, from text pattern matching to network scripting.

Python's third-party domain offers tools for website construction, numeric programming, serial port access, game development, and much more. The NumPy extension, for instance, has been described as a free and more powerful equivalent to the Matlab numeric programming system.

Integrations. Python scripts can easily communicate with other parts of an application, using a variety of integration mechanisms. Such integrations allow Python to be used as a product customization and extension tool.

Python code can invoke C and C++ libraries, can be called from C and C++ programs, can integrate with Java and .NET components, can communicate over frameworks too.

Software Perspective

Python is designed to be simple and readable as a coding syntax and a highly coherent programming model. Which makes the language easier to learn, understand, and remember. In practice, Python programmers do not need to constantly refer to manuals when reading or writing code; it's a consistently designed system that many find yields surprisingly uniform code.

If you have the chance to compare Python to Java for instance, and the both codes are doing the same job, you will notice that Python—by philosophy—adopts a somewhat minimalism approach.
This means that although there are usually multiple ways to accomplish a coding task, there is usually just one obvious way, a few less obvious alternatives.

Moreover, Python doesn't make arbitrary decisions for you; when interactions are ambiguous, explicit intervention is preferred over "*magic*".

> In the Python way of thinking, explicit is better than implicit, and simple is better than complex.

Beyond such design themes, Python includes tools such as modules and OOP that naturally promote code reusability. And because Python is focused on quality, so too, naturally, are Python programmers.

Python as a Scripting Language.

If you wonder if Python is a scripting language or not? The simple answer is "**yes**" and here is why. Python is a general-purpose programming language that is often applied in scripting roles. It is commonly defined as an object-oriented scripting language—a definition that blends support for OOP with an overall orientation toward scripting roles.

You may hear some developers often use the word "script" instead of "program" to describe a Python code file. In keeping with this tradition, this book uses the terms "script" and "program" interchangeably, with a slight preference for "script" to describe a simpler top-level file and "program" to refer to a more sophisticated multi-file application.

Because the term "scripting language" has so many different meanings to different observers, though, some would prefer that it not be applied to Python at all. In fact, people tend to make three very different associations, some of which are more useful than others, when they hear Python labeled as such:

- ***Shell tools.*** Sometimes when people hear Python described as a scripting language, they think it means that Python is a tool for coding operating-system-oriented scripts. Such programs are often launched from console command lines and perform tasks such as processing text files and launching other programs.

  Python programs can and do serve such roles, but this is just one of dozens of common Python application domains. It is not just a better shell-script language.

- ***Control language.*** To others, scripting refers to a "glue" layer used to control and direct (i.e., script) other application components. Python programs are indeed often deployed in the context of larger applications. For instance, to test hardware devices, Python programs may call out to components that give low-level access to a device. Similarly, programs may run bits of Python code at strategic points to support end-user product customization without the need to ship and recompile the entire system's source code.

  Python's simplicity makes it a naturally flexible control tool. Technically, though, this is also just a common Python role; many (perhaps most) Python programmers code standalone scripts without ever using or knowing about any integrated components. It is not just a control language.

- ***Ease of use.*** Probably the best way to think of the term "scripting language" is that it refers to a simple language used for quickly coding tasks. This is especially true when the term is applied to Python, which allows much faster program development than compiled languages like C++. Its rapid development cycle fosters an

exploratory, incremental mode of programming that has to be experienced to be appreciated.
Don't be fooled, though—Python is not just for simple tasks. Rather, it makes tasks simple by its ease of use and flexibility. Python has a simple feature set, but it allows programs to scale up in sophistication as needed. Because of that, it is commonly used for quick tactical tasks and longer-term strategic development.

So, is Python a scripting language or not? It depends on whom you ask. In general, the term "scripting" is probably best used to describe the rapid and flexible mode of development that Python supports, rather than a particular application domain.

Weakness of Python.

It is not regular to mention the weakness of any programming language before it's strengths. But as it doesn't matter what are Python weaknesses, it doesn't hurt any developer who uses it.

If you did use Python for a long time, you'll find that the only significant universal downside to Python is that, as currently implemented, its execution speed may not always be as fast as that of fully compiled and lower-level languages such as C and C++.
Though relatively rare today, you find some tasks that require a speed that you can not reach with programming languages other than lower-level languages, such as those that are more directly mapped to the underlying hardware architecture.

Whether you will ever care about the execution speed difference depends on what kinds of programs you write. Python has been optimized numerous times, and Python code runs fast enough by itself in most application domains.
Furthermore, whenever you do something "real" in a Python script, like processing a file or constructing a graphical user interface (GUI), your program will actually run at C speed, since such tasks are immediately dispatched to compiled C code inside the Python interpreter.
More fundamentally, Python's speed-of-development gain is often far more important than any speed-of-execution loss, especially given modern computer speeds. Even at today's CPU speeds, though, there still are some domains that do require optimal execution speeds.

We won't talk about extensions much in this text, but this is really just an instance of the Python-as-control-language role we discussed earlier. A prime example of this dual language strategy is the NumPy numeric programming extension for Python; by combining compiled and optimized numeric extension libraries with the Python language, NumPy turns Python into a numeric programming tool that is simultaneously efficient and easy to use. When needed, such extensions provide a powerful optimization tool.

Other Python Tradeoffs: The Intangible Bits we did mention that execution speed is the only major downside to Python. That's indeed the case for most Python users, and especially for newcomers. Most people find Python to be easy to learn and fun to use, especially when

compared with its contemporaries like Java, C#, and C++. In the interest of full disclosure, though, I should also note up front some more abstract tradeoffs I've observed in my two decades in the Python world—both as an educator and developer.

As a developer, I also at times question the tradeoffs inherent in Python's "batteries included" approach to development. Its emphasis on prebuilt tools can add dependencies (what if a battery you use is changed, broken, or deprecated?), and encourage special-case solutions over general principles that may serve users better in the long run (how can you evaluate or use a tool well if you don't understand its purpose?). We'll see examples of both of these concerns in this book.
For typical users, and especially for hobbyists and beginners, Python's toolset approach is a major asset. But you shouldn't be surprised when you outgrow precoded tools, and can benefit from the sorts of skills this book aims to impart. Or, to paraphrase a proverb: give people a tool, and they'll code for a day; teach them how to build tools, and they'll code for a lifetime. This book's job is more the latter than the former.
As mentioned elsewhere in this chapter, both Python and its toolbox model are also susceptible to downsides common to open source projects in general—the potential triumph of the personal preference of the few over common usage of the many, and the occasional appearance of anarchy and even elitism—though these tend to be most grievous on the leading edge of new releases.

Python Users Today.

While writing this book, the best estimate anyone can seem to make of the size of the Python user base is that there are roughly 9 million Python users around the world today (maybe plus a few).
And because Python is open source, and offers a set of good features and mainly used in some fields, this count increases each month as some companies acquire this programming language to be used in it's real and complex applications.

In general, though, Python enjoys a large user base and a very active developer community. Again while I am writing this book, I checked pypl and found that Python generally considered to be in the top 5 (It was the top 1) most widely used programming languages in the world today (its exact ranking varies per source and date).
Because Python has been around for over two decades and has been widely used, it is also very stable and robust.
Besides being leveraged by individual users, Python is also being applied in real revenue-generating products by real companies. For instance, among the generally known Python user base:

- Google makes extensive use of Python in its web search systems.
- Google's App Engine web development framework uses Python as an application language.
- The popular YouTube video sharing service is largely written in Python.
- Netflix and Yelp have both documented the role of Python in their software infrastructures.

- The Dropbox storage service codes both its server and desktop client software primarily in Python.
- The Raspberry Pi single-board computer promotes Python as its educational language.
- EVE Online, a massively multiplayer online game (MMOG) by CCP Games, uses Python broadly.
- The Civilization IV game's customizable scripted events are written entirely in Python.
- The widespread BitTorrent peer-to-peer file sharing system began its life as a Python program.
- Industrial Light & Magic, Pixar, and others use Python in the production of animated movies.
- ESRI uses Python as an end-user customization tool for its popular GIS mapping products.
- The IronPort email server product uses more than 1 million lines of Python code to do its job.
- Maya, a powerful integrated 3D modeling and animation system, provides a Python scripting API.
- The NSA uses Python for cryptography and intelligence analysis.
- iRobot uses Python to develop commercial and military robotic devices.
- The One Laptop Per Child (OLPC) project built its user interface and activity model in Python.
- Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM use Python for hardware testing.
- JPMorgan Chase, UBS, Getco, and Citadel apply Python to financial market forecasting.
- NASA, Los Alamos, Fermilab, JPL, and others use Python for scientific programming tasks.

And so on—though this list is representative, a full accounting is beyond this book's scope, and is almost guaranteed to change over time. For an up-to-date sampling of additional Python users, applications, and software, try the following pages currently at Python's site and Wikipedia, as well as a search in your favorite web browser:

- Success stories.
- Application domains.
- User quotes.
- Wikipedia page.

Probably the only common thread among the companies using Python today is that Python is used all over the map, in terms of application domains. Its general-purpose nature makes it applicable to almost all fields, not just one.

In fact, it's safe to say that virtually every substantial organization writing software is using Python, whether for short-term tactical tasks, such as testing and administration, or for long-term strategic product development. Python has proven to work well in both modes.

To Do with Python.

The most frequent question is asked, is what I can do with Python? The answer to that question will be in this section. So, in addition that Python is a well-designed programming language, it is useful for accomplishing real-world tasks.
It's commonly used in a variety of domains, as a tool for scripting other components and implementing standalone programs. And as Python is considered as a general-purpose language, it's roles are virtually unlimited: you can use it for everything from website development and gaming to robotics and spacecraft control.
However, the most common Python roles currently seem to fall into a few broad categories. The next few sections describe some of Python's most common applications today, as well as tools used in each domain. We won't be able to explore the tools mentioned here in any depth—if you are interested in any of these topics, see the Python website or other resources for more details.

Systems Programming

It's intuitive to know that Python's built-in interfaces to operating-system services make it ideal for writing portable, maintainable system-administration tools and utilities (sometimes called shell tools). It is even used and packed with every linux distribution now, also you should know that Python programs can do from file searching and directory trees, to launching other programs and parallel processing with processes, threads.

Python's standard library comes with POSIX bindings and support for all the usual OS tools: environment variables, files, sockets, pipes, processes, multiple threads, regular expression pattern matching, command-line arguments, standard stream interfaces, shell-command launchers, filename expansion, zip file utilities, XML and JSON parsers, CSV file handlers, and more.
In addition, the bulk of Python's system interfaces are designed to be portable; for example, a script that copies directory trees typically runs unchanged on all major Python platforms.

GUIs

If you want to know if Python supports GUI's or not. Python does support graphical user interfaces and with simplicity and rapid turnaround Python makes a good match for programming on the desktop.
Python comes with a standard object-oriented interface to the Tk GUI API called tkinter (Tkinter) that allows Python programs to implement portable GUIs with a native look and feel. Python/tkinter GUIs run unchanged on Microsoft Windows, X Windows (on Unix and Linux), and the Mac OS (both Classic and OS X).
A free extension package, PMW, adds advanced widgets to the tkinter toolkit. In addition, the wxPython GUI API, based on a C++ library, offers an alternative toolkit for constructing portable GUIs in Python.

Higher-level toolkits such as Dabo are built on top of base APIs such as wxPython and tkinter. With the proper library, you can also use GUI support in other toolkits in Python, such as Qt with PyQt for example.
For applications that run in web browsers or have simple interface requirements, both Jython and Python web frameworks and server-side CGI scripts, described in the next section, provide additional user interface options.

Web Development

Python comes with standard Internet modules that allow Python programs to perform a wide variety of networking tasks, and develop both client and server.
Scripts can communicate over *sockets*; extract form information sent to server-side CGI scripts; transfer files by FTP; parse and generate XML and JSON documents; send, receive, compose, and parse email; fetch web pages by URLs; and more. Python's libraries make these tasks remarkably simple.

In addition, full-blown web development framework packages for Python, such as Django, Flask, TurboGears, web2py, support quick construction of full-featured and production-quality websites with Python. Many of these include features such as object-relational mappers, a Model/View/Controller architecture, server-side scripting and templating, and AJAX support, to provide complete and enterprise-level web development solutions.
More recently, Python has expanded into rich Internet applications (RIAs), with tools such as Silverlight in IronPython, and pyjs (a.k.a. pyjamas) and its Python-to-JavaScript compiler, AJAX framework, and widget set. Python also has moved into cloud computing, with App Engine, and others described in the database section ahead. Where the Web leads, Python quickly follows.

Component Integration

We discussed the component integration role earlier when describing Python as a control language. Python's ability to be extended by and embedded in C and C++ systems makes it useful as a flexible glue language for scripting the behavior of other systems and components. For instance, integrating a C library into Python enables Python to test and launch the library's components, and embedding Python in a product enables onsite customizations to be coded without having to recompile the entire product (or ship its source code at all).
Tools such as the SWIG and SIP code generators can automate much of the work needed to link compiled components into Python for use in scripts, and the Cython system allows coders to mix Python and C-like code. Larger frameworks, such as Python's COM support on Windows, the Jython Java-based implementation, and the IronPython .NET-based implementation provide alternative ways to script components. On Windows, for example, Python scripts can use frameworks to script Word and Excel, access Silverlight, and much more.

Database Programming

For traditional database demands, there are Python interfaces to all commonly used relational database systems—Sybase, Oracle, Informix, ODBC, MySQL, PostgreSQL, SQLite, and more. The Python world has also defined a portable database API for accessing SQL database systems from Python scripts, which looks the same on a variety of underlying database systems. For instance, because the vendor interfaces implement the portable API, a script written to work with the free MySQL system will work largely unchanged on other systems (such as Oracle); all you generally have to do is replace the underlying vendor interface. The in-process SQLite embedded SQL database engine is a standard part of Python itself since 2.5, supporting both prototyping and basic program storage needs.

In the noSQL department, Python's standard pickle module provides a simple object persistence system—it allows programs to easily save and restore entire Python objects to files and file-like objects. On the Web, you'll also find third-party open source systems named ZODB and Durus that provide complete object-oriented database systems for Python scripts; others, such as SQLObject and SQLAlchemy, that implement object relational mappers (ORMs), which graft Python's class model into relational tables; and PyMongo, an interface to MongoDB, a high-performance, noSQL, open source JSON-style document database, which stores data in structures very similar to Python's own lists and dictionaries, and whose text may be parsed and created with Python's own standard library json module.

Still other systems offer more specialized ways to store data, including the datastore in Google's App Engine, which models data with Python classes and provides extensive scalability, as well as additional emerging cloud storage options such as Azure, Pi-Cloud, OpenStack, and Stackato.

Rapid Prototyping

To Python programs, components written in Python and C look the same. Because of this, it's possible to prototype systems in Python initially, and then move selected components to a compiled language such as C or C++ for delivery. Unlike some prototyping tools, Python doesn't require a complete rewrite once the prototype has solidified. Parts of the system that don't require the efficiency of a language such as C++ can remain coded in Python for ease of maintenance and use.

Numeric and Scientific Programming

Python is also heavily used in numeric programming—a domain that would not traditionally have been considered to be in the scope of scripting languages, but has grown to become one of Python's most compelling use cases. Prominent here, the NumPy high-performance numeric programming extension for Python mentioned earlier includes such advanced tools as an array object, interfaces to standard mathematical libraries, and much more. By integrating Python with numeric routines coded in a compiled language for speed, NumPy

turns Python into a sophisticated yet easy-to-use numeric programming tool that can often replace existing code written in traditional compiled languages such as FORTRAN or C++.

Additional numeric tools for Python support animation, 3D visualization, parallel processing, and so on. The popular SciPy and ScientificPython extensions, for example, provide additional libraries of scientific programming tools and use NumPy as a core component.

And More: Gaming, Images, Data Mining, Robots, Excel...

Python is commonly applied in more domains than can be covered here. For example, you'll find tools that allow you to use Python to do:

- Game programming and multimedia with pygame, cgkit, pyglet, PySoy, Panda3D, and others.
- Serial port communication on Windows, Linux, and more with the PySerial extension.
- Image processing with PIL and its newer Pillow fork, PyOpenGL, Blender, Maya, and more.
- Robot control programming with the PyRo toolkit.
- Natural language analysis with the NLTK package.
- Instrumentation on the Raspberry Pi and Arduino boards.
- Mobile computing with ports of Python to the Google Android and Apple iOS platforms.
- Excel spreadsheet function and macro programming with the PyXLL or DataNitro add-ins.
- Media file content and metadata tag processing with PyMedia, ID3, PIL/Pillow, and more.
- Artificial intelligence with the PyBrain neural net library and the Milk machine learning toolkit.
- Expert system programming with PyCLIPS, Pyke, Pyrolog, and pyDatalog.
- Network monitoring with zenoss, written in and customized with Python.
- Python-scripted design and modeling with PythonCAD, PythonOCC, FreeCAD, and others.
- Document processing and generation with ReportLab, Sphinx, Cheetah, PyPDF, and so on.
- Data visualization with Mayavi, matplotlib, VTK, VPython, and more.
- XML parsing with the xml library package, the xmlrpclib module, and third-party extensions.
- JSON and CSV file processing with the json and csv modules.
- Data mining with the Orange framework, the Pattern bundle, Scrapy, and custom code.

You can even play solitaire with the PySolFC program. And of course, you can always code custom Python scripts in less buzzword-laden domains to perform day-to-day system administration, process your email, manage your document and media libraries, and so on. You'll find links to the support in many fields at the PyPI website, and via web searches (search Google or http://www.python.org for links).

Though of broad practical use, many of these specific domains are largely just instances of Python's component integration role in action again. Adding it as a frontend to libraries of components written in a compiled language such as C makes Python useful for scripting in a wide variety of domains. As a general-purpose language that supports integration, Python is widely applicable.

Python Development and Support.

As a popular open source system, Python enjoys a large and active development community that responds to issues and develops enhancements with a speed that many commercial software developers might find remarkable. Python developers coordinate work online with a source-control system. Changes are developed per a formal protocol, which includes writing a PEP (Python Enhancement Proposal) or other document, and extensions to Python's regression testing system. In fact, modifying Python today is roughly as involved as changing commercial software—a far cry from Python's early days, when an email to its creator would suffice, but a good thing given its large user base today.

The PSF (Python Software Foundation), a formal nonprofit group, organizes conferences and deals with intellectual property issues. Numerous Python conferences are held around the world; O'Reilly's OSCON and the PSF's PyCon are the largest. The former of these addresses multiple open source projects, and the latter is a Python-only event that has experienced strong growth in recent years.

Open Source Tradeoffs

Having said that, it's important to note that while Python enjoys a vigorous development community, this comes with inherent tradeoffs. Open source software can also appear chaotic and even resemble anarchy at times, and may not always be as smoothly implemented as the prior paragraphs might imply. Some changes may still manage to defy official protocols, and as in all human endeavors, mistakes still happen despite the process controls.

Moreover, open source projects exchange commercial interests for the personal preferences of a current set of developers, which may or may not be the same as yours—you are not held hostage by a company, but you are at the mercy of those with spare time to change the system. The net effect is that open source software evolution is often driven by the few, but imposed on the many.
In practice, though, these tradeoffs impact those on the "bleeding" edge of new releases much more than those using established versions of the system, including prior releases in both Python 3.X and 2.X.

Strengths of Python.

Most Python users always ask about why they should use Python, mainly to answer such types of questions, you need to know the strengths of Python. If you don't already have a programming background, the language in the next few sections may be a bit baffling—don't worry, we'll explore all of these terms in more detail as we proceed through this book. For users and developers, though, here is a quick introduction to some of Python's top technical features.

Object-Oriented and Functional Programming

Python itself is considered as an object-oriented language, from the ground up. Its class model supports advanced notions such as:

- Polymorphism.
- Operator overloading.
- Multiple inheritance.

Yet, as Python has simple syntax and typing, OOP is remarkably easy to apply. In fact, if you don't understand these terms, you'll find they are much easier to learn with Python compared with any OOP language available. Also you should know that not all scripting languages contain that much OOP that Python has.

Besides serving as a powerful code structuring and reuse device, Python's OOP nature makes it ideal as a scripting tool for other object-oriented systems languages, which makes it very powerful in the terms of usability and mixability. For example, with the appropriate snippet of code, Python programs can subclass (specialize) classes implemented in C++, Java, and C#.
In addition to its original procedural (statement-based) and object-oriented (class-based) paradigms, Python in recent years has acquired built-in support for functional programming—a set that by most measures includes generators, comprehensions, closures, maps, decorators, anonymous function lambdas, and first-class function objects.
These can serve as both complement and alternative to its OOP tools.

Free

Python is completely free to use, edit and distribute. As with other open source software, such as Perl, Linux, and Apache, you can fetch the entire Python system's source code for free on the Internet. There are no restrictions on copying it, embedding it in your systems, or shipping it with your products.

Most of us when we hear the word "free" they understand it as "unsupported". Don't get the wrong idea "free" doesn't mean "unsupported".

On the contrary, the Python online community responds to user queries with a speed that most commercial software help desks would do well to try to emulate. Moreover, because Python comes with complete source code, it empowers developers, leading to the creation of a large team of implementation experts.
Although studying or changing a programming language's implementation isn't everyone's idea of fun, it's comforting to know that you can do so if you need to. You're not dependent on the whims of a commercial vendor, because the ultimate documentation—source code—is at your disposal as a last resort.

You should know that Python also has a multiple of free IDE to use, compared to R. For instance, one of those IDEs is very similar to R Studio (spyder), and one is made for experimentation (Jupyter). Both of IDE's are made specially for Python to use its powerful features and accelerate the development process.

As mentioned earlier, Python development is performed by a community that largely coordinates its efforts over the Internet. It consists of Python's original creator—Guido van Rossum—plus a supporting cast of thousands. Language changes must follow a formal enhancement procedure and be scrutinized by other developers. This tends to make Python more conservative with changes than some other languages and systems.

Portable

The standard implementation of Python is written in portable ANSI C, and it compiles and runs on virtually every major platform currently in use. For example, Python programs run today on everything from PDAs to supercomputers. As a partial list, Python is available on:

- Linux and Unix systems
- Microsoft Windows (all modern flavors)
- Mac OS (both OS X and Classic)
- BeOS, OS/2, VMS, and QNX
- Real-time systems such as VxWorks
- Cray supercomputers and IBM mainframes
- PDAs running Palm OS, PocketPC, and Linux
- Cell phones running Symbian OS, and Windows Mobile
- Gaming consoles and iPods
- Tablets and smartphones running Google's Android and Apple's iOS
- And more

Like the language interpreter itself, the standard library modules that ship with Python are implemented to be as portable across platform boundaries as possible. Further, Python programs are automatically compiled to portable byte code, which runs the same on any platform with a compatible version of Python installed (more on this in the next chapter).

What that means is that Python programs using the core language and standard libraries run the same on Linux, Windows, and most other systems with a Python interpreter.
Most Python ports also contain platform-specific extensions (e.g., COM support on Windows), but the core Python language and libraries work the same everywhere. As

mentioned earlier, Python also includes an interface to the Tk GUI toolkit called tkinter (Tkinter in 2.X), which allows Python programs to implement full-featured graphical user interfaces that run on all major GUI desktop platforms without program changes.

Powerful

From a features perspective, Python is something of a hybrid. Its toolset places it between traditional scripting languages (such as Tcl, Scheme, and Perl) and systems development languages (such as C, C++, and Java). Python provides all the simplicity and ease of use of a scripting language, along with more advanced software-engineering tools typically found in compiled languages. Unlike some scripting languages, this combination makes Python useful for large-scale development projects. As a preview, here are some of the main things you'll find in Python's toolbox:

Dynamic typing. Python keeps track of the kinds of objects your program uses when it runs; it doesn't require complicated type and size declarations in your code. In fact, as you'll see in Chapter 6, there is no such thing as a type or variable declaration anywhere in Python. Because Python code does not constrain data types, it is also usually automatically applicable to a whole range of objects.

Automatic memory management. Python automatically allocates objects and reclaims ("garbage collects") them when they are no longer used, and most can grow and shrink on demand. As you'll learn, Python keeps track of low-level memory details so you don't have to.

Programming-in-the-large support. For building larger systems, Python includes tools such as modules, classes, and exceptions. These tools allow you to organize systems into components, use OOP to reuse and customize code, and handle events and errors gracefully. Python's functional programming tools, described earlier, provide additional ways to meet many of the same goals.

Built-in object types. Python provides commonly used data structures such as lists, dictionaries, and strings as intrinsic parts of the language; as you'll see, they're both flexible and easy to use. For instance, built-in objects can grow and shrink on demand, can be arbitrarily nested to represent complex information, and more.

Built-in tools. To process all those object types, Python comes with powerful and standard operations, including concatenation (joining collections), slicing (extracting sections), sorting, mapping, and more.

Library utilities. For more specific tasks, Python also comes with a large collection of precoded library tools that support everything from regular expression matching to networking. Once you learn the language itself, Python's library tools are where much of the application-level action occurs.

Third-party utilities. Because Python is open source, developers are encouraged to contribute precoded tools that support tasks beyond those supported by its built-ins; on the Web, you'll find free support for COM, imaging, numeric programming, XML, database access, and much more.

Despite the array of tools in Python, it retains a remarkably simple syntax and design. The result is a powerful programming tool with all the usability of a scripting language.

Relatively Easy to Learn

This brings us to the point of this book: especially when compared to other widely used programming languages, the core Python language is remarkably easy to learn. In fact, if you're an experienced programmer, you can expect to be coding small-scale Python programs in a matter of days, and may be able to pick up some limited portions of the language in just hours—though you shouldn't expect to become an expert quite that fast (despite what you may have heard from marketing departments!).

Naturally, mastering any topic as substantial as today's Python is not trivial, and we'll devote the rest of this book to this task. But the true investment required to master Python is worthwhile—in the end, you'll gain programming skills that apply to nearly every computer application domain. Moreover, most find Python's learning curve to be much gentler than that of other programming tools.

Named After Monty Python

OK, this isn't quite a technical strength, but it does seem to be a surprisingly well-kept secret in the Python world that I wish to expose up front. Despite all the reptiles on Python books and icons, the truth is that Python is named after the British comedy group Monty Python—makers of the 1970s BBC comedy series Monty Python's Flying Circus and a handful of later full-length films, including Monty Python and the Holy Grail, that are still widely popular today. Python's original creator was a fan of Monty Python, as are many software developers (indeed, there seems to be a sort of symmetry between the two fields...).

This legacy inevitably adds a humorous quality to Python code examples. For instance, the traditional "foo" and "bar" for generic variable names become "spam" and "eggs" in the Python world. The occasional "Brian," "ni," and "shrubbery" likewise owe their appearances to this namesake. It even impacts the Python community at large: some events at Python conferences are regularly billed as "The Spanish Inquisition."

All of this is, of course, very funny if you are familiar with the shows, but less so otherwise. You don't need to be familiar with Monty Python's work to make sense of examples that borrow references from it, including many you will see in this book, but at least you now know their root. (Hey—I've warned you.)

Comparing Python to Other Languages.

Finally, we will place Python programming language in the it's position compared to other programming languages you may already know. People sometimes compare Python to languages such as C#, Perl and Java. This section summarizes common consensus in this department.

Small note here, you should know that it isn't about what is the best programming language, because no one will win eventually, it doesn't work that way. Moreover, this is not a zero sum game—most programmers will use many languages over their careers. Nevertheless, programming tools present choices and tradeoffs that merit consideration. After all, if Python didn't offer something over its alternatives, it would never have been used in the first place.

We talked about performance tradeoffs earlier, so here we'll focus on functionality. While other languages are also useful tools to know and use, many people find that Python:

- Is more *readable* than Perl. It's obvious that Python has a clear syntax and a simple, coherent design compared to Perl. And as discussed before this advantage makes Python more reusable and maintainable, and helps reduce program bugs.
- Is *simpler and easier* to use compared to Java and C#. Python is a scripting language, while Java and C# are compiled languages and both inherit much of the complexity and syntax of larger OOP systems languages like C++. Also it is simpler and easier to use than C++ too. Python code is simpler than the equivalent C++ and often 1/3 to 1/5 as large, and Python serves different roles compared to C++.
  And if you compared Python to C, of course it's simpler and higher-level than C. Python's detachment from underlying hardware architecture makes code less complex, better structured, and more approachable than C, C++'s progenitor.
- Is more *readable* and *general-purpose* than PHP. Python is used to construct websites too, but it is also applied to nearly every other computer domain, from robotics to movie animation and gaming.
- Is more *powerful* and *general-purpose* than JavaScript. Python has a larger toolset, and is not as tightly bound to web development. It's also used for scientific modeling, instrumentation, and more.
- Is more *readable* and established than Ruby. Python syntax is less cluttered, especially in nontrivial code, and its OOP is fully optional for users and projects to which it may not apply.
- Is more *mature* and *broadly focused* than Lua. Python's larger feature set and more extensive library support give it a wider scope than Lua, an embedded "glue" language like Tcl.
- Is *less esoteric* than Smalltalk, Lisp, and Prolog. Python has the dynamic flavor of languages like these, but also has a traditional syntax accessible to both developers and end users of customizable systems.

Especially for programs that do more than scan text files, and that might have to be read in the future by others (or by you!), many people find that Python fits the bill better than any other scripting or programming language available today. Furthermore, unless your application requires peak performance, Python is often a viable alternative to systems

development languages such as C, C++, and Java: Python code can often achieve the same goals, but will be much less difficult to write, debug, and maintain.
Of course, your author has been a card-carrying Python evangelist since 1992, so take these comments as you may (and other languages' advocates' mileage may vary arbitrarily). They do, however, reflect the common experience of many developers who have taken time to explore what Python has to offer.

Summary.

And by reaching here, we conclude the introductory portion of the book. In this chapter, we've explored some of the reasons that people pick Python for their programming tasks. Also We've seen how it is applied and looked at a representative sample of who is using it today.
From this book perspective, we are trying to see the Python programming language in action, so the rest of this book focuses entirely on the language details.
The next chapters in this part will begin the technical introduction to the language. In them, we'll explore ways to run Python programs, peek at Python's bytecode execution model, and introduce the basics of module files for saving code.