Learning\_Python\_3E\_Mark\_C01

vii

Table of Contents

Preface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . xxix

Part I. Getting Started

1. A Python Q&A Session . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

Why Do People Use Python? 3

Software Quality 5

Developer Productivity 5

Is Python a “Scripting Language”? 6

OK, but What’s the Downside? 7

Who Uses Python Today? 8

What Can I Do with Python? 9

Systems Programming 9

GUIs 9

Internet Scripting 10

Component Integration 10

Database Programming 11

Rapid Prototyping 11

Numeric and Scientific Programming 11

Gaming, Images, AI, XML, Robots, and More 12

What Are Python’s Technical Strengths? 12

It’s Object Oriented 12

It’s Free 13

It’s Portable 13

It’s Powerful 14

It’s Mixable

viii | Table of Contents

It’s Easy to Use 15

It’s Easy to Learn 17

It’s Named After Monty Python 17

How Does Python Stack Up to Language X? 18

Chapter Summary 19

Brain Builder 20

Chapter Quiz 20

Quiz Answers 20

2. How Python Runs Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

Introducing the Python Interpreter 22

Program Execution 24

The Programmer’s View 24

Python’s View 25

Byte code compilation 25

The Python Virtual Machine (PVM) 26

Performance implications 26

Development implications 27

Execution Model Variations 27

Python Implementation Alternatives 28

CPython 28

Jython 28

IronPython 29

Execution Optimization Tools 29

The Psyco just-in-time compiler 29

The Shedskin C++ translator 30

Frozen Binaries 31

Future Possibilities? 32

Chapter Summary 32

Brain Builder 33

Chapter Quiz 33

Quiz Answers

Table of Contents | ix

3. How You Run Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

Interactive Coding 34

Using the Interactive Prompt 37

System Command Lines and Files 37

Using Command Lines and Files 40

Unix Executable Scripts (#!) 41

Clicking File Icons 42

Clicking Icons on Windows 42

The raw\_input Trick 44

Other Icon-Click Limitations 45

Module Imports and Reloads 45

The Grander Module Story: Attributes 47

Modules and namespaces 49

import and reload Usage Notes 49

The IDLE User Interface 50

IDLE Basics 51

Using IDLE 52

Advanced IDLE Tools 54

Other IDEs 54

Embedding Calls 56

Frozen Binary Executables 56

Text Editor Launch Options 57

Other Launch Options 57

Future Possibilities? 57

Which Option Should I Use? 58

Chapter Summary 58

Brain Builder 59

Chapter Quiz 59

Quiz Answers 59

Brain Builder: Part I Exercises

x | Table of Contents

Part II. Types and Operations

4. Introducing Python Object Types . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65

Why Use Built-in Types? 66

Python’s Core Data Types 67

Numbers 68

Strings 69

Sequence Operations 70

Immutability 71

Type-Specific Methods 72

Getting Help 73

Other Ways to Code Strings 74

Pattern Matching 75

Lists 75

Sequence Operations 76

Type-Specific Operations 76

Bounds Checking 77

Nesting 77

List Comprehensions 78

Dictionaries 79

Mapping Operations 79

Nesting Revisited 80

Sorting Keys: for Loops 81

Iteration and Optimization 83

Missing Keys: if Tests 84

Tuples 85

Why Tuples? 85

Files 85

Other File-Like Tools 86

Other Core Types 87

How to Break Your Code’s Flexibility 88

User-Defined Classes 88

And Everything Else 89

Chapter Summary 90

Brain Builder 91

Chapter Quiz 91

Quiz Answers

Table of Contents | xi

5. Numbers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93

Python Numeric Types 93

Numeric Literals 94

Built-in Numeric Tools and Extensions 95

Python Expression Operators 96

Mixed Operators Follow Operator Precedence 97

Parentheses Group Subexpressions 97

Mixed Types Are Converted Up 97

Preview: Operator Overloading 98

Numbers in Action 99

Variables and Basic Expressions 99

Numeric Display Formats 100

Division: Classic, Floor, and True 102

Bitwise Operations 103

Long Integers 103

Complex Numbers 104

Hexadecimal and Octal Notation 105

Other Built-in Numeric Tools 106

Other Numeric Types 107

Decimal Numbers 107

Sets 108

Booleans 109

Third-Party Extensions 110

Chapter Summary 110

Brain Builder 111

Chapter Quiz 111

Quiz Answers 111

6. The Dynamic Typing Interlude . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112

The Case of the Missing Declaration Statements 112

Variables, Objects, and References 112

Types Live with Objects, Not Variables 114

Objects Are Garbage-Collected 115

Shared References 116

Shared References and In-Place Changes 118

Shared References and Equality 119

Dynamic Typing Is Everywhere

xii | Table of Contents

Chapter Summary 121

Brain Builder 122

Chapter Quiz 122

Quiz Answers 122

7. Strings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 123

String Literals 124

Single- and Double-Quoted Strings Are the Same 125

Escape Sequences Represent Special Bytes 125

Raw Strings Suppress Escapes 127

Triple Quotes Code Multiline Block Strings 129

Unicode Strings Encode Larger Character Sets 130

Strings in Action 132

Basic Operations 132

Indexing and Slicing 133

Extended slicing: the third limit 135

String Conversion Tools 136

Character code conversions 138

Changing Strings 139

String Formatting 140

Advanced String Formatting 141

Dictionary-Based String Formatting 142

String Methods 143

String Method Examples: Changing Strings 144

String Method Examples: Parsing Text 146

Other Common String Methods in Action 147

The Original string Module 148

General Type Categories 149

Types Share Operation Sets by Categories 149

Mutable Types Can Be Changed In-Place 150

Chapter Summary 150

Brain Builder 151

Chapter Quiz 151

Quiz Answers

Table of Contents | xiii

8. Lists and Dictionaries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 152

Lists 152

Lists in Action 154

Basic List Operations 154

Indexing, Slicing, and Matrixes 155

Changing Lists In-Place 156

Index and slice assignments 156

List method calls 157

Other common list operations 159

Dictionaries 160

Dictionaries in Action 161

Basic Dictionary Operations 162

Changing Dictionaries In-Place 163

More Dictionary Methods 163

A Languages Table 165

Dictionary Usage Notes 166

Using dictionaries to simulate flexible lists 166

Using dictionaries for sparse data structures 167

Avoiding missing-key errors 167

Using dictionaries as “records” 168

Other ways to make dictionaries 169

Chapter Summary 170

Brain Builder 171

Chapter Quiz 171

Quiz Answers 171

9. Tuples, Files, and Everything Else . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 172

Tuples 172

Tuples in Action 173

Tuple syntax peculiarities: commas and parentheses 174

Conversions and immutability 174

Why Lists and Tuples? 175

Files 176

Opening Files 176

Using Files

xiv | Table of Contents

Files in Action 178

Storing and parsing Python objects in files 178

Storing native Python objects with pickle 180

Storing and parsing packed binary data in files 181

Other File Tools 182

Type Categories Revisited 182

Object Flexibility 183

References Versus Copies 184

Comparisons, Equality, and Truth 186

The Meaning of True and False in Python 188

Python’s Type Hierarchies 189

Other Types in Python 191

Built-in Type Gotchas 191

Assignment Creates References, Not Copies 191

Repetition Adds One Level Deep 192

Beware of Cyclic Data Structures 193

Immutable Types Can’t Be Changed In-Place 193

Chapter Summary 193

Brain Builder 195

Chapter Quiz 195

Quiz Answers 195

Brain Builder: Part II Exercises 196

Part III. Statements and Syntax

10. Introducing Python Statements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 201

Python Program Structure Revisited 201

Python’s Statements 202

A Tale of Two ifs 203

What Python Adds 204

What Python Removes 204

Parentheses are optional 204

End of line is end of statement 204

End of indentation is end of block 205

Why Indentation Syntax? 206

A Few Special Cases

Table of Contents | xv

Statement rule special cases 208

Block rule special case 209

A Quick Example: Interactive Loops 210

A Simple Interactive Loop 210

Doing Math on User Inputs 211

Handling Errors by Testing Inputs 212

Handling Errors with try Statements 213

Nesting Code Three Levels Deep 214

Chapter Summary 215

Brain Builder 216

Chapter Quiz 216

Quiz Answers 216

11. Assignment, Expressions, and print . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 217

Assignment Statements 217

Assignment Statement Forms 218

Sequence Assignments 219

Advanced sequence assignment patterns 220

Multiple-Target Assignments 222

Multiple-target assignment and shared references 222

Augmented Assignments 223

Augmented assignment and shared references 225

Variable Name Rules 225

Naming conventions 227

Names have no type, but objects do 227

Expression Statements 228

Expression Statements and In-Place Changes 229

print Statements 229

The Python “Hello World” Program 230

Redirecting the Output Stream 231

The print >> file Extension 232

Chapter Summary 234

Brain Builder 235

Chapter Quiz 235

Quiz Answers

xvi | Table of Contents

12. if Tests . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 236

if Statements 236

General Format 236

Basic Examples 237

Multiway Branching 237

Python Syntax Rules 239

Block Delimiters 240

Statement Delimiters 241

A Few Special Cases 242

Truth Tests 243

The if/else Ternary Expression 244

Chapter Summary 246

Brain Builder 247

Chapter Quiz 247

Quiz Answers 247

13. while and for Loops . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 248

while Loops 248

General Format 249

Examples 249

break, continue, pass, and the Loop else 250

General Loop Format 250

Examples 251

pass 251

continue 251

break 252

else 252

More on the loop else clause 253

for Loops 254

General Format 254

Examples 256

Basic usage 256

Other data types 256

Tuple assignment in for 257

Nested for loops

Table of Contents | xvii

Iterators: A First Look 258

File Iterators 260

Other Built-in Type Iterators 262

Other Iteration Contexts 263

User-Defined Iterators 264

Loop Coding Techniques 265

Counter Loops: while and range 265

Nonexhaustive Traversals: range 266

Changing Lists: range 267

Parallel Traversals: zip and map 268

Dictionary construction with zip 270

Generating Both Offsets and Items: enumerate 271

List Comprehensions: A First Look 272

List Comprehension Basics 272

Using List Comprehensions on Files 273

Extended List Comprehension Syntax 274

Chapter Summary 275

Brain Builder 276

Chapter Quiz 276

Quiz Answers 276

14. The Documentation Interlude . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 278

Python Documentation Sources 278

# Comments 279

The dir Function 279

Docstrings: \_ \_doc\_ \_ 280

User-defined docstrings 281

Docstring standards 282

Built-in docstrings 282

PyDoc: The help Function 283

PyDoc: HTML Reports 285

Standard Manual Set 289

Web Resources 289

Published Books 290

Common Coding Gotchas

xviii | Table of Contents

Chapter Summary 293

Brain Builder 294

Chapter Quiz 294

Quiz Answers 294

Brain Builder: Part III Exercises 295

Part IV. Functions

15. Function Basics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 299

Why Use Functions? 300

Coding Functions 300

def Statements 302

def Executes at Runtime 303

A First Example: Definitions and Calls 303

Definition 304

Calls 304

Polymorphism in Python 305

A Second Example: Intersecting Sequences 306

Definition 306

Calls 306

Polymorphism Revisited 307

Local Variables 308

Chapter Summary 308

Brain Builder 309

Chapter Quiz 309

Quiz Answers 309

16. Scopes and Arguments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 310

Scope Rules 310

Python Scope Basics 311

Name Resolution: The LEGB Rule 312

Scope Example 314

The Built-in Scope 314

The global Statement 316

Minimize Global Variables 317

Minimize Cross-File Changes 318

Other Ways to Access Globals

Table of Contents | xix

Scopes and Nested Functions 320

Nested Scope Details 320

Nested Scope Examples 321

Factory functions 321

Retaining enclosing scopes’ state with defaults 323

Nested scopes and lambdas 324

Scopes versus defaults with loop variables 324

Arbitrary scope nesting 326

Passing Arguments 326

Arguments and Shared References 327

Avoiding Mutable Argument Changes 329

Simulating Output Parameters 329

Special Argument-Matching Modes 330

Keyword and Default Examples 332

Keywords 332

Defaults 333

Arbitrary Arguments Examples 333

Collecting arguments 334

Unpacking arguments 334

Combining Keywords and Defaults 335

The min Wakeup Call 336

Full credit 336

Bonus points 337

The punch line 338

A More Useful Example: General Set Functions 338

Argument Matching: The Gritty Details 339

Chapter Summary 340

Brain Builder 342

Chapter Quiz 342

Quiz Answers 343

17. Advanced Function Topics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 344

Anonymous Functions: lambda 344

lambda Expressions 344

Why Use lambda? 346

How (Not) to Obfuscate Your Python Code 347

Nested lambdas and Scopes

xx | Table of Contents

Applying Functions to Arguments 350

The apply Built-in 350

Passing keyword arguments 351

apply-Like Call Syntax 351

Mapping Functions over Sequences: map 352

Functional Programming Tools: filter and reduce 353

List Comprehensions Revisited: Mappings 355

List Comprehension Basics 355

Adding Tests and Nested Loops 356

List Comprehensions and Matrixes 358

Comprehending List Comprehensions 360

Iterators Revisited: Generators 360

Generator Function Example 362

Extended Generator Function Protocol: send Versus next 364

Iterators and Built-in Types 364

Generator Expressions: Iterators Meet List Comprehensions 365

Timing Iteration Alternatives 366

Function Design Concepts 369

Functions Are Objects: Indirect Calls 370

Function Gotchas 371

Local Names Are Detected Statically 372

Defaults and Mutable Objects 373

Functions Without returns 375

Enclosing Scope Loop Variables 375

Chapter Summary 375

Brain Builder 377

Chapter Quiz 377

Quiz Answers 377

Brain Builder: Part IV Exercises 379

Part V. Modules

18. Modules: The Big Picture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 385

Why Use Modules? 385

Python Program Architecture 386

How to Structure a Program 387

Imports and Attributes 387

Standard Library Modules

Table of Contents | xxi

How Imports Work 389

1. Find It 390

The module search path 390

The sys.path list 392

Module file selection 393

Advanced module selection concepts 393

2. Compile It (Maybe) 394

3. Run It 394

Chapter Summary 395

Brain Builder 397

Chapter Quiz 397

Quiz Answers 397

19. Module Coding Basics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 398

Module Creation 398

Module Usage 399

The import Statement 399

The from statement 400

The from \* Statement 400

Imports Happen Only Once 400

import and from Are Assignments 401

Cross-File Name Changes 402

import and from Equivalence 402

Potential Pitfalls of the from Statement 403

When import is required 404

Module Namespaces 404

Files Generate Namespaces 405

Attribute Name Qualification 406

Imports Versus Scopes 407

Namespace Nesting 408

Reloading Modules 409

reload Basics 410

reload Example 411

Chapter Summary 412

Brain Builder 414

Chapter Quiz 414

Quiz Answers

xxii | Table of Contents

20. Module Packages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 415

Package Import Basics 415

Packages and Search Path Settings 416

Package \_ \_init\_ \_.py Files 416

Package Import Example 418

from Versus import with Packages 419

Why Use Package Imports? 420

A Tale of Three Systems 421

Chapter Summary 424

Brain Builder 425

Chapter Quiz 425

Quiz Answers 425

21. Advanced Module Topics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 426

Data Hiding in Modules 426

Minimizing from \* Damage: \_X and \_ \_all\_ \_ 426

Enabling Future Language Features 427

Mixed Usage Modes: \_ \_name\_ \_ and \_ \_main\_ \_ 428

Unit Tests with \_ \_name\_ \_ 429

Changing the Module Search Path 430

The import as Extension 431

Relative Import Syntax 431

Why Relative Imports? 432

Module Design Concepts 434

Modules Are Objects: Metaprograms 435

Module Gotchas 437

Statement Order Matters in Top-Level Code 437

Importing Modules by Name String 438

from Copies Names but Doesn’t Link 439

from \* Can Obscure the Meaning of Variables 440

reload May Not Impact from Imports 440

reload, from, and Interactive Testing 441

reload Isn’t Applied Transitively 442

Recursive from Imports May Not Work 443

Chapter Summary

Table of Contents | xxiii

Brain Builder 445

Chapter Quiz 445

Quiz Answers 445

Brain Builder: Part V Exercises 446

Part VI. Classes and OOP

22. OOP: The Big Picture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 451

Why Use Classes? 452

OOP from 30,000 Feet 453

Attribute Inheritance Search 453

Classes and Instances 455

Class Method Calls 456

Coding Class Trees 456

OOP Is About Code Reuse 459

Chapter Summary 462

Brain Builder 463

Chapter Quiz 463

Quiz Answers 463

23. Class Coding Basics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 465

Classes Generate Multiple Instance Objects 465

Class Objects Provide Default Behavior 466

Instance Objects Are Concrete Items 466

A First Example 467

Classes Are Customized by Inheritance 469

A Second Example 470

Classes Are Attributes in Modules 471

Classes Can Intercept Python Operators 472

A Third Example 474

Why Use Operator Overloading? 475

The World’s Simplest Python Class 476

Chapter Summary 478

Brain Builder 479

Chapter Quiz 479

Quiz Answers

xxiv | Table of Contents

24. Class Coding Details . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 481

The class Statement 481

General Form 481

Example 482

Methods 484

Example 485

Calling Superclass Constructors 486

Other Method Call Possibilities 486

Inheritance 486

Attribute Tree Construction 487

Specializing Inherited Methods 488

Class Interface Techniques 489

Abstract Superclasses 490

Operator Overloading 491

Common Operator Overloading Methods 492

\_ \_getitem\_ \_ Intercepts Index References 493

\_ \_getitem\_ \_ and \_ \_iter\_ \_ Implement Iteration 493

User-Defined Iterators 494

Multiple iterators on one object 496

\_ \_getattr\_ \_ and \_ \_setattr\_ \_ Catch Attribute References 498

Emulating Privacy for Instance Attributes 499

\_ \_repr\_ \_ and \_ \_str\_ \_ Return String Representations 500

\_ \_radd\_ \_ Handles Right-Side Addition 502

\_ \_call\_ \_ Intercepts Calls 502

Function Interfaces and Callback-Based Code 503

\_ \_del\_ \_ Is a Destructor 505

Namespaces: The Whole Story 506

Simple Names: Global Unless Assigned 506

Attribute Names: Object Namespaces 506

The “Zen” of Python Namespaces: Assignments Classify Names 506

Namespace Dictionaries 508

Namespace Links 511

A More Realistic Example 512

Chapter Summary 515

Brain Builder 516

Chapter Quiz 516

Quiz Answer

Table of Contents | xxv

25. Designing with Classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 518

Python and OOP 518

Overloading by Call Signatures (or Not) 519

Classes As Records 519

OOP and Inheritance: “Is-a” Relationships 521

OOP and Composition: “Has-a” Relationships 523

Stream Processors Revisited 524

OOP and Delegation 527

Multiple Inheritance 529

Classes Are Objects: Generic Object Factories 532

Why Factories? 533

Methods Are Objects: Bound or Unbound 534

Documentation Strings Revisited 535

Classes Versus Modules 537

Chapter Summary 537

Brain Builder 538

Chapter Quiz 538

Quiz Answers 538

26. Advanced Class Topics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 539

Extending Built-in Types 539

Extending Types by Embedding 540

Extending Types by Subclassing 540

Pseudoprivate Class Attributes 543

Name Mangling Overview 543

Why Use Pseudoprivate Attributes? 544

New-Style Classes 545

Diamond Inheritance Change 546

Diamond inheritance example 547

Explicit conflict resolution 547

Other New-Style Class Extensions 549

Static and class methods 549

Instance slots 549

Class properties 550

New \_ \_getattribute\_ \_ overloading method 552

Static and Class Methods 552

Using Static and Class Methods

xxvi | Table of Contents

Function Decorators 556

Decorator Example 558

Class Gotchas 559

Changing Class Attributes Can Have Side Effects 559

Multiple Inheritance: Order Matters 560

Methods, Classes, and Nested Scopes 561

“Overwrapping-itis” 563

Chapter Summary 564

Brain Builder 565

Chapter Quiz 565

Quiz Answers 565

Brain Builder: Part VI Exercises 566

Part VII. Exceptions and Tools

27. Exception Basics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 575

Why Use Exceptions? 576

Exception Roles 576

Exception Handling: The Short Story 577

The try/except/else Statement 581

try Statement Clauses 582

The try/else Clause 585

Example: Default Behavior 585

Example: Catching Built-in Exceptions 586

The try/finally Statement 587

Example: Coding Termination Actions with try/finally 588

Unified try/except/finally 589

Combining finally and except by Nesting 590

Unified try Example 591

The raise Statement 592

Example: Raising and Catching User-Defined Exceptions 593

Example: Passing Extra Data with raise 593

Example: Propagating Exceptions with raise 594

The assert Statement 595

Example: Trapping Constraints (but Not Errors)

Table of Contents | xxvii

with/as Context Managers 596

Basic Usage 596

The Context Management Protocol 598

Chapter Summary 600

Brain Builder 601

Chapter Quiz 601

Quiz Answers 601

28. Exception Objects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 602

String-Based Exceptions 603

String Exceptions Are Right Out! 603

Class-Based Exceptions 604

Class Exception Example 604

Why Class Exceptions? 606

Built-in Exception Classes 609

Specifying Exception Text 610

Sending Extra Data and Behavior in Instances 611

Example: Extra data with classes and strings 611

General raise Statement Forms 613

Chapter Summary 615

Brain Builder 616

Chapter Quiz 616

Quiz Answers 616

29. Designing with Exceptions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 617

Nesting Exception Handlers 617

Example: Control-Flow Nesting 619

Example: Syntactic Nesting 619

Exception Idioms 621

Exceptions Aren’t Always Errors 621

Functions Signal Conditions with raise 622

Debugging with Outer try Statements 622

Running In-Process Tests 623

More on sys.exc\_info 624

Exception Design Tips 624

What Should Be Wrapped 624

Catching Too Much: Avoid Empty excepts 625

Catching Too Little: Use Class-Based Categories

xxviii | Table of Contents

Exception Gotchas 627

String Exceptions Match by Identity, Not by Value 628

Catching the Wrong Thing 629

Core Language Summary 629

The Python Toolset 630

Development Tools for Larger Projects 631

Chapter Summary 634

Brain Builder 635

Chapter Quiz 635

Quiz Answers 635

Brain Builder: Part VII Exercises 636

Part VIII. Appendixes

A. Installation and Configuration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 639

B. Solutions to End-of-Part Exercises . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 646

Index . . .

3

**Chapter 1 CHAPTER 1 A Python Q&A Session1**

If you’ve bought this book, you may already know what Python is, and why it’s an important tool to learn. If you don’t, you probably won’t be sold on Python until you’ve learned the language by reading the rest of this book and have done a project or two. But before we jump into details, the first few pages of this book will briefly introduce some of the main reasons behind Python’s popularity. To begin sculpting a definition of Python, this chapter takes the form of a question-and-answer session, which poses some of the most common questions asked by beginners.

Si compró este libro, es posible que ya sepa qué es Python y por qué es una herramienta importante para aprender. Si no lo hace, es probable que no se convenza de Python hasta que haya aprendido el lenguaje leyendo el resto de este libro y haya realizado uno o dos proyectos. Pero antes de entrar en detalles, las primeras páginas de este libro presentarán brevemente algunas de las principales razones detrás de la popularidad de Python. Para comenzar a esculpir una definición de Python, este capítulo toma la forma de una sesión de preguntas y respuestas, que plantea algunas de las preguntas más comunes que hacen los principiantes.

Why Do People Use Python?

Because there are many programming languages available today, this is the usual first question of newcomers. Given that there are roughly 1 million Python users out there at the moment, there really is no way to answer this question with complete accuracy. The choice of development tools is sometimes based on unique constraints or personal preference. But after teaching Python to roughly 200 groups and 3,000 students during the last 10 years, some common themes have emerged. The primary factors cited by Python users seem to be these:

Debido a que hay muchos lenguajes de programación disponibles en la actualidad, esta es la primera pregunta habitual de los recién llegados. Dado que hay aproximadamente 1 millón de usuarios de Python en este momento, realmente no hay forma de responder a esta pregunta con total precisión. La elección de las herramientas de desarrollo a veces se basa en restricciones únicas o preferencias personales. Pero después de enseñar Python a aproximadamente 200 grupos y 3000 estudiantes durante los últimos 10 años, han surgido algunos temas comunes. Los principales factores citados por los usuarios de Python parecen ser estos:

**Software quality**

For many, Python’s focus on readability, coherence, and software quality in general sets it apart from other tools in the scripting world. 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 tounderstand, 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).

Para muchos, el enfoque de Python en la legibilidad, la coherencia y la calidad del software en general lo distingue de otras herramientas en el mundo de las secuencias de comandos. El código de Python está diseñado para ser legible y, por lo tanto, reutilizable y mantenible, mucho más que los lenguajes de secuencias de comandos tradicionales. La uniformidad del código de Python hace que sea fácil de entender, incluso si no lo escribiste. Además, Python tiene un gran soporte para mecanismos de reutilización de software más avanzados, como la programación orientada a objetos (POO).

**Developer productivity**

Python boosts developer productivity many times beyond compiled or statically typed languages such as C, C++, and Java. Python code is typically one-third to one-fifth the size of equivalent C++ or Java code. That means there is less to type, less to debug, and less to maintain after the fact. Python programs also run immediately, without the lengthy compile and link steps of some other tools, further boosting programmer speed.

Python aumenta la productividad de los desarrolladores muchas veces más allá de los lenguajes compilados o tipificados estáticamente, como C, C++ y Java. El código de Python suele tener entre un tercio y una quinta parte del tamaño del código C++ o Java equivalente. Eso significa que hay menos para escribir, menos para depurar y menos para mantener después del hecho. Los programas de Python también se ejecutan de inmediato, sin los largos pasos de compilación y vinculación de otras herramientas, lo que aumenta aún más la velocidad del programador.

**Program portability**

Most Python programs run unchanged on all major computer 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, database access programs, web-based systems, and more. Even operating system interfaces, including program launches and directory processing, are as portable in Python as they can possibly be.

La mayoría de los programas de Python se ejecutan sin cambios en las principales plataformas informáticas. Portar el código de Python entre Linux y Windows, por ejemplo, suele ser solo una cuestión de copiar el código de un script entre máquinas. Además, Python ofrece múltiples opciones para codificar interfaces gráficas de usuario portátiles, programas de acceso a bases de datos, sistemas basados en web y más. Incluso las interfaces del sistema operativo, incluidos los inicios de programas y el procesamiento de directorios, son tan portátiles en Python como es posible.

**Support libraries**

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. In addition, Python can be extended with both homegrown libraries and a vast collection of third-party application support software. Python’s third-party domain offers tools for web site 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 pro- gramming system.

Python viene con una gran colección de funcionalidades preconstruidas y portátiles, conocida como la biblioteca estándar. Esta biblioteca admite una variedad de tareas de programación a nivel de aplicación, desde coincidencia de patrones de texto hasta secuencias de comandos de red. Además, Python se puede ampliar con bibliotecas propias y una amplia colección de software de soporte de aplicaciones de terceros. El dominio de terceros de Python ofrece herramientas para la construcción de sitios web, programación numérica, acceso a puertos serie, desarrollo de juegos y mucho más. La extensión NumPy, por ejemplo, ha sido descrita como un equivalente gratuito y más potente del sistema de programación numérica de Matlab.

**Component integration**

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. Today, Python code can invoke C and C++ libraries, can be called from C and C++ programs, can integrate with Java components, can communicate over frameworks such as COM and .NET, and can interact over networks with interfaces like SOAP, XML-RPC, and CORBA. It is not a standalone tool.

Los scripts de Python pueden comunicarse fácilmente con otras partes de una aplicación mediante una variedad de mecanismos de integración. Dichas integraciones permiten que Python se use como una herramienta de personalización y extensión de productos. Hoy en día, el código Python puede invocar bibliotecas C y C++, puede llamarse desde programas C y C++, puede integrarse con componentes Java, puede comunicarse a través de marcos como COM y .NET, y puede interactuar a través de redes con interfaces como SOAP, XML-RPC y CORBA. No es una herramienta independiente.

**Enjoyment**

Because of Python’s ease of use and built-in toolset, it can make the act of programming more pleasure than chore. Although this may be an intangible benefit, its effect on productivity is an important asset. Of these factors, the first two (quality and productivity) are probably the most compelling benefits to most Python users

Debido a la facilidad de uso de Python y al conjunto de herramientas integrado, puede hacer que el acto de programar sea más un placer que una tarea. Aunque esto puede ser un beneficio intangible, su efecto sobre la productividad es un activo importante. De estos factores, los dos primeros (calidad y productividad) son probablemente los beneficios más atractivos para la mayoría de los usuarios de Python.

**Software Quality**

By design, Python implements a deliberately simple and readable syntax, and a highly coherent programming model. As a slogan at a recent Python conference attests, the net result is that Python seems to “fit your brain”—that is, features of the language interact in consistent and limited ways, and follow naturally from a small set of core concepts. This 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 regular-looking code.

Por diseño, Python implementa una sintaxis deliberadamente simple y legible, y un modelo de programación altamente coherente. Como lo atestigua un eslogan en una conferencia reciente de Python, el resultado neto es que Python parece "ajustarse a su cerebro", es decir, las características del lenguaje interactúan de manera consistente y limitada, y se derivan naturalmente de un pequeño conjunto de conceptos básicos. Esto hace que el idioma sea más fácil de aprender, comprender y recordar. En la práctica, los programadores de Python no necesitan consultar constantemente los manuales cuando leen o escriben código; es un sistema diseñado consistentemente que muchos encuentran que produce un código sorprendentemente regular.

By philosophy, Python adopts a somewhat minimalist 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, and a small set of coherent interactions everywhere in the language. 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. \*

Por filosofía, Python adopta un enfoque algo minimalista. Esto significa que, aunque normalmente hay varias formas de realizar una tarea de codificación, normalmente hay una sola forma obvia, algunas alternativas menos obvias y un pequeño conjunto de interacciones coherentes en todo el lenguaje. Además, Python no toma decisiones arbitrarias por ti; cuando las interacciones son ambiguas, se prefiere la intervención explícita a la “magia”. En la forma de pensar de Python, lo explícito es mejor que lo implícito, y lo simple es mejor que lo complejo.

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.

Más allá de estos temas de diseño, Python incluye herramientas como módulos y programación orientada a objetos que naturalmente promueven la reutilización del código. Y debido a que Python se enfoca en la calidad, también lo son, naturalmente, los programadores de Python.

**Developer Productivity**

During the great Internet boom of the mid-to-late 1990s, it was difficult to find enough programmers to implement software projects; developers were asked to implement systems as fast as the Internet evolved. Now, in the post-boom era of layoffs and economic recession, the picture has shifted. Today, programming staffs are often asked to accomplish the same tasks with even fewer people. In both of these scenarios, Python has shined as a tool that allows programmers to get more done with less effort. It is deliberately optimized for speed of development—its simple syntax, dynamic typing, lack of compile steps, and built-in toolset allow programmers to develop programs in a fraction of the time needed when using some other tools. The net effect is that Python typically boosts developer productivity many times beyond the levels supported by traditional languages. That’s good news in both boom and bust times, and everywhere the software industry goes in between.

Durante el gran auge de Internet de mediados a fines de la década de 1990, era difícil encontrar suficientes programadores para implementar proyectos de software; Se pidió a los desarrolladores que implementaran sistemas tan rápido como evolucionó Internet. Ahora, en la era posterior al auge de los despidos y la recesión económica, la imagen ha cambiado. Hoy en día, al personal de programación se le pide a menudo que realice las mismas tareas con incluso menos personas. En ambos escenarios, Python ha brillado como una herramienta que permite a los programadores hacer más con menos esfuerzo. Está deliberadamente optimizado para la velocidad de desarrollo: su sintaxis simple, escritura dinámica, falta de pasos de compilación y conjunto de herramientas integrado permite a los programadores desarrollar programas en una fracción del tiempo necesario cuando se utilizan otras herramientas. El efecto neto es que Python normalmente aumenta la productividad de los desarrolladores muchas veces más allá de los niveles admitidos por los lenguajes tradicionales. Esas son buenas noticias tanto en tiempos de auge como de caída, y en todos los lugares intermedios de la industria del software.

**Is Python a “Scripting Language”?**

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. In fact, people often use the word “script” instead of “program” to describe a Python code file. In this book, the terms “script” and “program” are used interchangeably, with a slight preference for “script” to describe a simpler top-level file, and “program” to refer to a more sophisticated multifile application. Because the term “scripting language” has so many different meanings to different observers, 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:

Python es un lenguaje de programación de propósito general que a menudo se aplica en roles de secuencias de comandos. Comúnmente se define como un lenguaje de secuencias de comandos orientado a objetos, una definición que combina el soporte para OOP con una orientación general hacia los roles de secuencias de comandos. De hecho, las personas a menudo usan la palabra "script" en lugar de "programa" para describir un archivo de código de Python. En este libro, los términos "secuencia de comandos" y "programa" se usan indistintamente, con una ligera preferencia por "secuencia de comandos" para describir un archivo de nivel superior más simple y "programa" para referirse a una aplicación de varios archivos más sofisticada. Debido a que el término "lenguaje de secuencias de comandos" tiene tantos significados diferentes para diferentes observadores, algunos preferirían que no se aplicara a Python en absoluto. De hecho, la gente tiende a hacer tres asociaciones muy diferentes, algunas de las cuales son más útiles que otras, cuando oyen etiquetar a Python como tal:

**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.

A veces, cuando las personas escuchan que Python se describe como un lenguaje de secuencias de comandos, piensan que Python es una herramienta para codificar secuencias de comandos orientadas al sistema operativo. Dichos programas a menudo se inician desde las líneas de comando de la consola y realizan tareas como procesar archivos de texto e iniciar otros programas. Los programas de Python pueden y cumplen tales roles, pero este es solo uno de las docenas de dominios de aplicaciones comunes de Python. No es solo un mejor lenguaje de shell-script.

**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 having 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 Python programmers code standalone scripts without ever using or knowing about any integrated components. It is not just a control language.

Para otros, la secuencia de comandos se refiere a una capa de "pegamento" utilizada para controlar y dirigir (es decir, secuencias de comandos) otros componentes de la aplicación. De hecho, los programas de Python a menudo se implementan en el contexto de aplicaciones más grandes. Por ejemplo, para probar dispositivos de hardware, los programas de Python pueden llamar a componentes que brindan acceso de bajo nivel a un dispositivo. De manera similar, los programas pueden ejecutar bits de código Python en puntos estratégicos para admitir la personalización del producto del usuario final sin tener que enviar y volver a compilar el código fuente de todo el sistema. La simplicidad de Python lo convierte en una herramienta de control naturalmente flexible. Sin embargo, técnicamente, esto también es solo un rol común de Python; muchos programadores de Python codifican secuencias de comandos independientes sin usar ni conocer ningún componente integrado. No es solo un lenguaje de control.

**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.

Probablemente la mejor manera de pensar en el término "lenguaje de secuencias de comandos" es que se refiere a un lenguaje simple que se usa para tareas de codificación rápidas. Esto es especialmente cierto cuando el término se aplica a Python, que permite un desarrollo de programas mucho más rápido que los lenguajes compilados como C++. Su rápido ciclo de desarrollo fomenta un modo de programación exploratorio e incremental que debe experimentarse para apreciarse. Sin embargo, no se deje engañar: Python no es solo para tareas simples. Más bien, simplifica las tareas por su facilidad de uso y flexibilidad. Python tiene un conjunto de funciones simple, pero permite que los programas aumenten su sofisticación según sea necesario. Por eso, se usa comúnmente para tareas tácticas rápidas y desarrollo estratégico a largo plazo.

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.

Entonces, ¿Python es un lenguaje de secuencias de comandos o no? Depende de a quién le preguntes. En general, el término "secuencias de comandos" probablemente se utilice mejor para describir el modo de desarrollo rápido y flexible que admite Python, en lugar de un dominio de aplicación en particular.

**OK, but What’s the Downside?**

After using Python for 15 years and teaching it for 10, the only downside to Python I’ve found is that, as currently implemented, its execution speed may not always be as fast as compiled languages such as C and C++. We’ll talk about implementation concepts in detail later in this book. In short, the standard implementations of Python today compile (i.e., translate) source code statements to an intermediate format known as byte code, and then interpret the byte code. Byte code provides portability, as it is a platform-independent format. However, because Python is not compiled all the way down to binary machine code (e.g., instructions for an Intel chip), some programs will run more slowly in Python than in a fully compiled language like C.

Después de usar Python durante 15 años y enseñarlo durante 10, el único inconveniente de Python que he encontrado es que, tal como se implementa actualmente, su velocidad de ejecución puede no ser siempre tan rápida como la de los lenguajes compilados como C y C++. Hablaremos sobre los conceptos de implementación en detalle más adelante en este libro. En resumen, las implementaciones estándar de Python hoy en día compilan (es decir, traducen) declaraciones de código fuente a un formato intermedio conocido como código de bytes y luego interpretan el código de bytes. El código de bytes proporciona portabilidad, ya que es un formato independiente de la plataforma. Sin embargo, debido a que Python no está compilado completamente hasta el código de máquina binario (por ejemplo, instrucciones para un chip Intel), algunos programas se ejecutarán más lentamente en Python que en un lenguaje completamente compilado como C.

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 process a file or construct a GUI, your program is actually running 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.

Si alguna vez le importará la diferencia de velocidad de ejecución depende de qué tipo de programas escriba. Python se ha optimizado varias veces y el código de Python se ejecuta lo suficientemente rápido por sí mismo en la mayoría de los dominios de aplicación. Además, cada vez que hace algo "real" en un script de Python, como procesar un archivo o construir una GUI, su programa en realidad se ejecuta a velocidad C, ya que tales tareas se envían inmediatamente al código C compilado dentro del intérprete de Python. Más fundamentalmente, la ganancia de velocidad de desarrollo de Python suele ser mucho más importante que cualquier pérdida de velocidad de ejecución, especialmente dadas las velocidades de las computadoras modernas.

Even at today’s CPU speeds, though, there still are some domains that do require optimal execution speeds. Numeric programming and animation, for example, often need at least their core number-crunching components to run at C speed (or better). If you work in such a domain, you can still use Python—simply split off the parts of the application that require optimal speed into compiled extensions, and link those into your system for use in Python scripts. We won’t talk about extensions much in this text, but this is really just an instance of the Python-as-control-language role that 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 efficient and easy to use. You may never need to code such extensions in your own Python work, but they provide a powerful optimization mechanism if you ever do.

Sin embargo, incluso a las velocidades de CPU actuales, todavía hay algunos dominios que requieren velocidades de ejecución óptimas. La programación numérica y la animación, por ejemplo, a menudo necesitan al menos sus componentes básicos de procesamiento de números para ejecutarse a velocidad C (o mejor). Si trabaja en un dominio de este tipo, aún puede usar Python: simplemente divida las partes de la aplicación que requieren una velocidad óptima en extensiones compiladas y vincúlelas a su sistema para usarlas en scripts de Python. No hablaremos mucho sobre las extensiones en este texto, pero esto es solo una instancia del rol de Python como lenguaje de control que discutimos anteriormente. Un excelente ejemplo de esta estrategia de lenguaje dual es la extensión de programación numérica NumPy para Python; al combinar bibliotecas de extensión numérica compiladas y optimizadas con el lenguaje Python, NumPy convierte a Python en una herramienta de programación numérica que es eficiente y fácil de usar. Es posible que nunca necesite codificar tales extensiones en su propio trabajo de Python, pero proporcionan un poderoso mecanismo de optimización si alguna vez lo necesita.

**Who Uses Python Today?**

At this writing, in 2007, the best estimate anyone can seem to make of the size of the Python user base is that there are roughly 1 million Python users around the world today (plus or minus a few). This estimate is based on various statistics, like download rates and developer surveys. Because Python is open source, a more exact count is difficult—there are no license registrations to tally. Moreover, Python is automatically included with Linux distributions, Macintosh computers, and some products and hardware, further clouding the user-base picture.

Al momento de escribir esto, en 2007, la mejor estimación que cualquiera puede hacer sobre el tamaño de la base de usuarios de Python es que hay aproximadamente 1 millón de usuarios de Python en todo el mundo hoy (más o menos unos pocos). Esta estimación se basa en varias estadísticas, como tasas de descarga y encuestas de desarrolladores. Debido a que Python es de código abierto, es difícil hacer un recuento más exacto: no hay registros de licencias para contar. Además, Python se incluye automáticamente con las distribuciones de Linux, las computadoras Macintosh y algunos productos y hardware, lo que enturbia aún más la imagen de la base de usuarios.

In general, though, Python enjoys a large user base, and a very active developer community. Because Python has been around for more than 15 years and has been widely used, it is also very stable and robust. Besides being employed by individual users, Python is also being applied in real revenue-generating products by real companies. For instance:

Sin embargo, en general, Python disfruta de una gran base de usuarios y una comunidad de desarrolladores muy activa. Debido a que Python existe desde hace más de 15 años y ha sido ampliamente utilizado, también es muy estable y robusto. Además de ser empleado por usuarios individuales, Python también se está aplicando en productos reales que generan ingresos por parte de empresas reales. Por ejemplo:

• Google makes extensive use of Python in its web search system, and employs

Python’s creator.

• The YouTube video sharing service is largely written in Python.

• The popular BitTorrent peer-to-peer file sharing system is a Python program.

• Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM use Python for

hardware testing.

• Industrial Light & Magic, Pixar, and others use Python in the production of

movie animation.

• JPMorgan Chase, UBS, Getco, and Citadel apply Python for financial market

forecasting.

• NASA, Los Alamos, Fermilab, JPL, and others use Python for scientific program-

ming tasks.

• iRobot uses Python to develop commercial robotic vacuum cleaners.

• ESRI uses Python as an end-user customization tool for its popular GIS mapping

products.

• The NSA uses Python for cryptography and intelligence analysis.

• The IronPort email server product uses more than 1 million lines of Python code

to do its job.

• The One Laptop Per Child (OLPC) project builds its user interface and activity

model in Python.

And so on. Probably the only common thread amongst 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 Python is being used by virtually every substantial organization writing software, 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. For more details on companies using Python today, see Python’s web site at http:// [www.python.org](http://www.python.org).

Etcétera. Probablemente, el único hilo común entre las empresas que usan Python hoy en día es que Python se usa en todo el mapa, en términos de dominios de aplicación. Su naturaleza de propósito general lo hace aplicable a casi todos los campos, no solo a uno. De hecho, es seguro decir que Python está siendo utilizado por prácticamente todas las organizaciones importantes que escriben software, ya sea para tareas tácticas a corto plazo, como pruebas y administración, o para el desarrollo de productos estratégicos a largo plazo. Python ha demostrado funcionar bien en ambos modos. Para obtener más detalles sobre las empresas que utilizan Python en la actualidad, consulte el sitio web de Python en <http://www.python.org>.

**What Can I Do with Python?**

Besides being a well-designed programming language, Python is also useful for accomplishing real-world tasks—the sorts of things developers do day in and day out. It’s commonly used in a variety of domains, as a tool for scripting other components and implementing standalone programs. In fact, as a general-purpose language, Python’s roles are virtually unlimited: you can use it for everything from web site 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 web site, or other resources for more details.

Además de ser un lenguaje de programación bien diseñado, Python también es útil para realizar tareas del mundo real, el tipo de cosas que los desarrolladores hacen día tras día. Se usa comúnmente en una variedad de dominios, como una herramienta para crear scripts de otros componentes e implementar programas independientes. De hecho, como lenguaje de propósito general, las funciones de Python son prácticamente ilimitadas: puede usarlo para todo, desde el desarrollo de sitios web y juegos hasta la robótica y el control de naves espaciales. Sin embargo, los roles de Python más comunes actualmente parecen caer en algunas categorías amplias. Las siguientes secciones describen algunas de las aplicaciones más comunes de Python en la actualidad, así como las herramientas utilizadas en cada dominio. No podremos explorar las herramientas mencionadas aquí en profundidad; si está interesado en alguno de estos temas, consulte el sitio web de Python u otros recursos para obtener más detalles.

**Systems Programming**

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). Python programs can search files and directory trees, launch other programs, do parallel processing with processes and threads, and so on. 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, 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.

Las interfaces integradas de Python para los servicios del sistema operativo lo hacen ideal para escribir herramientas y utilidades de administración de sistemas portátiles y mantenibles (a veces llamadas herramientas de shell). Los programas de Python pueden buscar archivos y árboles de directorios, iniciar otros programas, realizar procesamiento paralelo con procesos e hilos, etc. La biblioteca estándar de Python viene con enlaces POSIX y soporte para todas las herramientas habituales del sistema operativo: variables de entorno, archivos, sockets, conductos, procesos, subprocesos múltiples, coincidencia de patrones de expresiones regulares, argumentos de línea de comandos, interfaces de flujo estándar, lanzadores de comandos de shell, nombre de archivo expansión, y más. Además, la mayor parte de las interfaces del sistema de Python están diseñadas para ser portátiles; por ejemplo, una secuencia de comandos que copia árboles de directorios normalmente se ejecuta sin cambios en todas las principales plataformas de Python.

**GUIs**

Python’s simplicity and rapid turnaround also make it a good match for graphical user interface (GUI) programming. Python comes with a standard object-oriented interface to the Tk GUI API called Tkinter, which allows Python programs to implement portable GUIs with a native look and feel. Python/Tkinter GUIs run unchanged on MS 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 PythonCard and Dabo are built on top of base APIs such as wxPython and Tkinter. With the proper library, you can also use other GUI toolkits in Python, such as Qt, GTK, MFC, and Swing. For applications that run in web browsers, or have simple interface requirements, both Jython (the Java version of Python, described in Chapter 2) and Python server-side CGI scripts provide additional user interface options.

La simplicidad y la rapidez de respuesta de Python también lo convierten en una buena opción para la programación de la interfaz gráfica de usuario (GUI). Python viene con una interfaz estándar orientada a objetos para la API Tk GUI llamada Tkinter, que permite que los programas de Python implementen GUI portátiles con una apariencia nativa. Las GUI de Python/Tkinter se ejecutan sin cambios en MS Windows, X Windows (en Unix y Linux) y Mac OS (tanto Classic como OS X). Un paquete de extensión gratuito, PMW, agrega widgets avanzados al conjunto de herramientas de Tkinter. Además, la API de GUI de wxPython, basada en una biblioteca de C++, ofrece un conjunto de herramientas alternativo para construir GUI portátiles en Python. Los kits de herramientas de nivel superior, como PythonCard y Dabo, se construyen sobre las API básicas, como wxPython y Tkinter. Con la biblioteca adecuada, también puede usar otros juegos de herramientas GUI en Python, como Qt, GTK, MFC y Swing. Para las aplicaciones que se ejecutan en navegadores web o tienen requisitos de interfaz simples, tanto Jython (la versión Java de Python, descrita en el Capítulo 2) como los scripts CGI del lado del servidor de Python brindan opciones de interfaz de usuario adicionales.

**Internet Scripting**

Python comes with standard Internet modules that allow Python programs to perform a wide variety of networking tasks, in client and server modes. Scripts can communicate over sockets; extract form information sent to server-side CGI scripts; transfer files by FTP; process XML files; send, receive, compose, and parse email; fetch web pages by URLs; parse the HTML and XML of fetched web pages; communicate over XML-RPC, SOAP, and Telnet; and more. Python’s libraries make these tasks remarkably simple.

Python viene con módulos de Internet estándar que permiten que los programas de Python realicen una amplia variedad de tareas de red, en modo cliente y servidor. Los scripts pueden comunicarse a través de sockets; extraer información de formulario enviada a scripts CGI del lado del servidor; transferir archivos por FTP; procesar archivos XML; enviar, recibir, redactar y analizar correos electrónicos; buscar páginas web por URL; analizar el HTML y XML de las páginas web obtenidas; comunicarse a través de XML-RPC, SOAP y Telnet; y más. Las bibliotecas de Python hacen que estas tareas sean notablemente simples.

In addition, there is a large collection of third-party tools available on the Web for doing Internet programming in Python. For instance, the HTMLGen system generates HTML files from Python class-based descriptions, the mod\_python package runs Python efficiently within the Apache web server and supports server-side templating with its Python Server Pages, and the Jython system provides for seamless Python/Java integration, and supports coding of server-side applets that run on clients. In addition, full-blown web development packages for Python, such as Django, Turbo-Gears, Pylons, Zope, and WebWare, support quick construction of full-featured and production-quality web sites with Python.

Además, hay una gran colección de herramientas de terceros disponibles en la Web para realizar programación de Internet en Python. Por ejemplo, el sistema HTMLGen genera archivos HTML a partir de descripciones basadas en clases de Python, el paquete mod\_python ejecuta Python de manera eficiente dentro del servidor web Apache y admite plantillas del lado del servidor con sus páginas de servidor de Python, y el sistema Jython proporciona una integración perfecta de Python/Java. y admite la codificación de subprogramas del lado del servidor que se ejecutan en los clientes. Además, los paquetes completos de desarrollo web para Python, como Django, Turbo-Gears, Pylons, Zope y WebWare, admiten la construcción rápida de sitios web con todas las funciones y calidad de producción con Python.

**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 larger frameworks, such as Python’s COM support on MS Windows, the Jython Java-based implementation, the IronPython .NET-based implementation, and various CORBA toolkits for Python, provide alternative ways to script components. On Windows, for example, Python scripts can use frameworks to script MS Word and Excel.

Discutimos el rol de integración de componentes anteriormente cuando describimos Python como un lenguaje de control. La capacidad de Python para ser ampliado e integrado en los sistemas C y C++ lo hace útil como un lenguaje de unión flexible para crear secuencias de comandos del comportamiento de otros sistemas y componentes. Por ejemplo, la integración de una biblioteca C en Python permite que Python pruebe y ejecute los componentes de la biblioteca, y la incorporación de Python en un producto permite codificar las personalizaciones en el sitio sin tener que volver a compilar todo el producto o enviar su código fuente. Las herramientas como los generadores de código SWIG y SIP pueden automatizar gran parte del trabajo necesario para vincular componentes compilados en Python para su uso en secuencias de comandos. Y los marcos más grandes, como la compatibilidad con COM de Python en MS Windows, la implementación basada en Java de Jython, la implementación basada en .NET de IronPython y varios kits de herramientas CORBA para Python, brindan formas alternativas de crear componentes de secuencias de comandos. En Windows, por ejemplo, los scripts de Python pueden usar marcos para escribir MS Word y Excel.

**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 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 have to do is replace the underlying vendor interface.

Para las demandas de bases de datos tradicionales, existen interfaces de Python para todos los sistemas de bases de datos relacionales de uso común: Sybase, Oracle, Informix, ODBC, MySQL, PostgreSQL, SQLite y más. El mundo de Python también ha definido una API de base de datos portátil para acceder a sistemas de bases de datos SQL desde scripts de Python, que se ve igual en una variedad de sistemas de bases de datos subyacentes. Por ejemplo, debido a que las interfaces de los proveedores implementan la API portátil, un script escrito para funcionar con el sistema gratuito MySQL funcionará prácticamente sin cambios en otros sistemas (como Oracle); todo lo que tiene que hacer es reemplazar la interfaz de proveedor subyacente.

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 a third-party system named ZODB that provides a complete object-oriented database system for Python scripts, and another called SQLObject that maps relational tables onto Python’s class model. And, as of Python 2.5, SQLite is a standard part of Python itself.

El módulo pickle estándar de Python proporciona un sistema de persistencia de objetos simple: permite que los programas guarden y restauren fácilmente objetos completos de Python en archivos y objetos similares a archivos. En la Web, también encontrará un sistema de terceros llamado ZODB que proporciona un sistema completo de base de datos orientado a objetos para scripts de Python, y otro llamado SQLObject que asigna tablas relacionales al modelo de clases de Python. Y, a partir de Python 2.5, SQLite es una parte estándar de Python.

**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.

Para los programas de Python, los componentes escritos en Python y C tienen el mismo aspecto. Debido a esto, es posible crear prototipos de sistemas en Python inicialmente y luego mover los componentes seleccionados a un lenguaje compilado como C o C++ para su entrega. A diferencia de algunas herramientas de creación de prototipos, Python no requiere una reescritura completa una vez que el prototipo se ha solidificado. Las partes del sistema que no requieren la eficiencia de un lenguaje como C++ pueden permanecer codificadas en Python para facilitar el mantenimiento y el uso.

**Numeric and Scientific Programming**

The NumPy 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, which 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.

La extensión de programación numérica NumPy para Python mencionada anteriormente incluye herramientas avanzadas como un objeto de matriz, interfaces para bibliotecas matemáticas estándar y mucho más. Al integrar Python con rutinas numéricas codificadas en un lenguaje compilado para aumentar la velocidad, NumPy convierte a Python en una herramienta de programación numérica sofisticada pero fácil de usar, que a menudo puede reemplazar el código existente escrito en lenguajes compilados tradicionales como FORTRAN o C++. Las herramientas numéricas adicionales para Python admiten animación, visualización 3D, procesamiento paralelo, etc.

**Gaming, Images, AI, XML, Robots, and More**

Python is commonly applied in more domains than can be mentioned here. For example, you can do graphics and game programming in Python with the pygame system; image processing with the PIL package and others; robot control programming with the PyRo toolkit; XML parsing with the xml library package, the xmlrpclib module, and third-party extensions; AI programming with neural network simulators and expert system shells; and natural language analysis with the NLTK package. You can even play solitaire with the PySol program. You’ll find support for many such fields at the Vaults of Parnassus, and the newer PyPI web sites (search Google or http://www.python.org for links).

Python se aplica comúnmente en más dominios de los que se pueden mencionar aquí. Por ejemplo, puedes hacer gráficos y programación de juegos en Python con el sistema pygame; procesamiento de imágenes con el paquete PIL y otros; programación de control de robots con el kit de herramientas PyRo; Análisis XML con el paquete de la biblioteca xml, el módulo xmlrpclib y extensiones de terceros; Programación de IA con simuladores de redes neuronales y shells de sistemas expertos; y análisis de lenguaje natural con el paquete NLTK. Incluso puedes jugar al solitario con el programa PySol. Encontrará soporte para muchos de estos campos en Vaults of Parnassus y en los sitios web más nuevos de PyPI (busque enlaces en Google o http://www.python.org).

In general, many of these specific domains are largely just instances of Python’s component integration role in action again. Adding Python 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.

En general, muchos de estos dominios específicos son en gran medida solo instancias del rol de integración de componentes de Python en acción nuevamente. Agregar Python como interfaz para bibliotecas de componentes escritos en un lenguaje compilado, como C, hace que Python sea útil para crear secuencias de comandos en una amplia variedad de dominios. Como lenguaje de propósito general que admite la integración, Python es ampliamente aplicable.

**What Are Python’s Technical Strengths?**

Naturally, this is a developer’s question. 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 developers, though, here is a quick introduction to some of Python’s top technical features.

Naturalmente, esta es una pregunta de desarrollador. Si aún no tiene experiencia en programación, el lenguaje en las próximas secciones puede ser un poco desconcertante; no se preocupe, exploraremos todos estos términos con más detalle a medida que avancemos en este libro. Sin embargo, para los desarrolladores, aquí hay una introducción rápida a algunas de las principales funciones técnicas de Python.

**It’s Object Oriented**

Python is an object-oriented language, from the ground up. Its class model supports advanced notions such as polymorphism, operator overloading, and multiple inheritance; yet, in the context of Python’s 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 than with just about any other OOP language available. Besides serving as a powerful code structuring and reuse device, Python’s OOP nature makes it ideal as a scripting tool for object-oriented systems languages such as C++ and Java. For example, with the appropriate glue code, Python programs can subclass (specialize) classes implemented in C++, Java, and C#.

Python es un lenguaje orientado a objetos, desde cero. Su modelo de clase admite nociones avanzadas como polimorfismo, sobrecarga de operadores y herencia múltiple; sin embargo, en el contexto de la sintaxis y escritura simples de Python, la programación orientada a objetos es notablemente fácil de aplicar. De hecho, si no comprende estos términos, encontrará que son mucho más fáciles de aprender con Python que con cualquier otro lenguaje OOP disponible. Además de servir como un poderoso dispositivo de estructuración y reutilización de código, la naturaleza OOP de Python lo hace ideal como herramienta de secuencias de comandos para lenguajes de sistemas orientados a objetos como C++ y Java. Por ejemplo, con el código de conexión adecuado, los programas de Python pueden subclasificar (especializar) las clases implementadas en C++, Java y C#.

Of equal significance, OOP is an option in Python; you can go far without having to become an object guru all at once. Much like C++, Python supports both procedural and object-oriented programming modes. Its object-oriented tools can be applied if and when constraints allow. This is especially useful in tactical development modes, which preclude design phases

De igual importancia, OOP es una opción en Python; puedes llegar lejos sin tener que convertirte en un gurú de objetos de una sola vez. Al igual que C ++, Python admite los modos de programación orientados a objetos y de procedimiento. Sus herramientas orientadas a objetos se pueden aplicar siempre y cuando las restricciones lo permitan. Esto es especialmente útil en los modos de desarrollo táctico, que excluyen las fases de diseño.

**It’s Free**

Python is completely free to use and distribute. As with other open source software, such as Tcl, 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. In fact, you can even sell Python’s source code, if you are so inclined. But 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 vendors would do well to notice. 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 it’s available as a final resort and ultimate documentation source. You’re not dependent on the whims of a commercial vendor.

Python es completamente libre de usar y distribuir. Al igual que con otro software de código abierto, como Tcl, Perl, Linux y Apache, puede obtener el código fuente completo del sistema Python de forma gratuita en Internet. No hay restricciones para copiarlo, integrarlo en sus sistemas o enviarlo con sus productos. De hecho, incluso puede vender el código fuente de Python, si así lo desea. Pero no te hagas una idea equivocada: "gratis" no significa "sin soporte". Por el contrario, la comunidad en línea de Python responde a las consultas de los usuarios con una velocidad que la mayoría de los proveedores de software comercial harían bien en notar. Además, debido a que Python viene con el código fuente completo, empodera a los desarrolladores, lo que lleva a la creación de un gran equipo de expertos en implementación. Aunque estudiar o cambiar la implementación de un lenguaje de programación no es la idea divertida de todos, es reconfortante saber que está disponible como último recurso y fuente de documentación definitiva. No depende de los caprichos de un vendedor comercial.

Python development is performed by a community, which largely coordinates its efforts over the Internet. It consists of Python’s creator—Guido van Rossum, the officially anointed Benevolent Dictator for Life (BDFL) of Python—plus a cast of thousands. Language changes must follow a formal enhancement procedure (known as the PEP process), and be scrutinized by a formal testing system and the BDFL. Happily, this tends to make Python more conservative with changes than some other languages.

El desarrollo de Python lo realiza una comunidad, que coordina en gran medida sus esfuerzos a través de Internet. Está formado por el creador de Python, Guido van Rossum, el dictador benévolo de por vida oficialmente ungido (BDFL) de Python, además de un elenco de miles. Los cambios de idioma deben seguir un procedimiento de mejora formal (conocido como el proceso PEP) y ser examinados por un sistema de prueba formal y el BDFL. Afortunadamente, esto tiende a hacer que Python sea más conservador con los cambios que otros lenguajes.

**It’s 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:

La implementación estándar de Python está escrita en ANSI C portátil y compila y se ejecuta en prácticamente todas las plataformas principales actualmente en uso. Por ejemplo, los programas de Python se ejecutan hoy en todo, desde PDA hasta supercomputadoras. Como lista parcial, Python está disponible en:

• Linux and Unix systems

• Microsoft Windows and DOS (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

• And more

Besides the language interpreter itself, the standard library modules that ship with Python are also 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, which allows Python programs to implement full-featured graphical user interfaces that run on all major GUI platforms without program changes.

Además del propio intérprete de lenguaje, los módulos de la biblioteca estándar que se envían con Python también se implementan para que sean lo más portátiles posible a través de los límites de la plataforma. Además, los programas de Python se compilan automáticamente en código de bytes portátil, que se ejecuta de la misma manera en cualquier plataforma con una versión compatible de Python instalada (más sobre esto en el próximo capítulo). Lo que eso significa es que los programas de Python que usan el lenguaje central y las bibliotecas estándar se ejecutan de la misma manera en Linux, Windows y la mayoría de los demás sistemas con un intérprete de Python. La mayoría de los puertos de Python también contienen extensiones específicas de la plataforma (por ejemplo, compatibilidad con COM en Windows), pero el lenguaje y las bibliotecas centrales de Python funcionan igual en todas partes. Como se mencionó anteriormente, Python también incluye una interfaz para el kit de herramientas Tk GUI llamado Tkinter, que permite que los programas de Python implementen interfaces gráficas de usuario con todas las funciones que se ejecutan en todas las principales plataformas GUI sin cambios en el programa.

**It’s 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:

Desde la perspectiva de las características, Python es algo así como un híbrido. Su conjunto de herramientas lo sitúa entre los lenguajes de secuencias de comandos tradicionales (como Tcl, Scheme y Perl) y los lenguajes de desarrollo de sistemas (como C, C++ y Java). Python proporciona toda la simplicidad y facilidad de uso de un lenguaje de secuencias de comandos, junto con herramientas de ingeniería de software más avanzadas que normalmente se encuentran en lenguajes compilados. A diferencia de algunos lenguajes de secuencias de comandos, esta combinación hace que Python sea útil para proyectos de desarrollo a gran escala. Como vista previa, estas son algunas de las cosas principales que encontrará en la caja de herramientas de Python:

**Dynamic typing**

Python keeps track of the kinds of objects your program uses when it runs; itdoesn’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.

Python realiza un seguimiento de los tipos de objetos que usa su programa cuando se ejecuta; no requiere declaraciones complicadas de tipo y tamaño en su código. De hecho, como verá en el Capítulo 6, no existe tal cosa como una declaración de tipo o variable en ninguna parte de Python. Debido a que el código de Python no restringe los tipos de datos, también suele aplicarse automáticamente a una amplia gama de objetos.

**Automatic memory management**

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

Python asigna objetos automáticamente y los reclama ("recolección de basura") cuando ya no se usan, y la mayoría crece y se reduce a pedido. Como aprenderá, Python realiza un seguimiento de los detalles de la memoria de bajo nivel para que usted no tenga que hacerlo.

**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.

Para construir sistemas más grandes, Python incluye herramientas como módulos, clases y excepciones. Estas herramientas le permiten organizar sistemas en componentes, usar programación orientada a objetos para reutilizar y personalizar código y manejar eventos y errores con elegancia.

**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.

Python proporciona estructuras de datos de uso común, como listas, diccionarios y cadenas, como partes intrínsecas del lenguaje; como verá, ambos son flexibles y fáciles de usar. Por ejemplo, los objetos incorporados pueden crecer y reducirse a pedido, pueden anidarse arbitrariamente para representar información compleja y más.

**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.

Para procesar todos esos tipos de objetos, Python viene con operaciones potentes y estándar, que incluyen concatenación (unión de colecciones), división (extracción de secciones), clasificación, mapeo y más.

**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. Python’s library tools are where much of the application-level action occurs.

Para tareas más específicas, Python también viene con una gran colección de herramientas de biblioteca precodificadas que admiten todo, desde la coincidencia de expresiones regulares hasta la creación de redes. Las herramientas de la biblioteca de Python son donde ocurre gran parte de la acción a nivel de aplicación.

**Third-party utilities**

Because Python is open source, developers are encouraged to contribute pre- coded tools that support tasks beyond those supported by its built-ins; on the Web, you’ll find free support for COM, imaging, CORBA ORBs, 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.

Debido a que Python es de código abierto, se alienta a los desarrolladores a que contribuyan con herramientas precodificadas que admitan tareas más allá de las admitidas por sus funciones integradas; en la Web, encontrará soporte gratuito para COM, imágenes, ORB de CORBA, XML, acceso a bases de datos y mucho más. A pesar de la variedad de herramientas en Python, conserva una sintaxis y un diseño notablemente simples. El resultado es una poderosa herramienta de programación con toda la facilidad de uso de un lenguaje de programación.

**It’s Mixable**

Python programs can easily be “glued” to components written in other languages in a variety of ways. For example, Python’s C API lets C programs call and be called by Python programs flexibly. That means you can add functionality to the Python system as needed, and use Python programs within other environments or systems. Mixing Python with libraries coded in languages such as C or C++, for instance, makes it an easy-to-use frontend language and customization tool. As mentioned earlier, this also makes Python good at rapid prototyping; systems may be implemented in Python first, to leverage its speed of development, and later, moved to C for delivery, one piece at a time, according to performance demands.

Los programas de Python se pueden "pegar" fácilmente a componentes escritos en otros lenguajes en una variedad de formas. Por ejemplo, la API de C de Python permite que los programas de C llamen y sean llamados por los programas de Python de manera flexible. Eso significa que puede agregar funcionalidad al sistema de Python según sea necesario y usar programas de Python dentro de otros entornos o sistemas. Mezclar Python con bibliotecas codificadas en lenguajes como C o C++, por ejemplo, lo convierte en una herramienta de personalización y lenguaje de interfaz fácil de usar. Como se mencionó anteriormente, esto también hace que Python sea bueno para la creación rápida de prototipos; Los sistemas pueden implementarse en Python primero, para aprovechar su velocidad de desarrollo, y luego, trasladarse a C para su entrega, una pieza a la vez, de acuerdo con las demandas de rendimiento.

**It’s Easy to Use**

To run a Python program, you simply type it and run it. There are no intermediate compile and link steps, like there are for languages such as C or C++. Python executes programs immediately, which makes for an interactive programming experience and rapid turnaround after program changes—in many cases, you can witness the effect of a program change as fast as you can type it.

Para ejecutar un programa de Python, simplemente escríbalo y ejecútelo. No hay pasos intermedios de compilación y enlace, como los hay para lenguajes como C o C++. Python ejecuta los programas de inmediato, lo que lo convierte en una experiencia de programación interactiva y un cambio rápido después de los cambios de programa; en muchos casos, puede presenciar el efecto de un cambio de programa tan rápido como puede escribirlo.

Of course, development cycle turnaround is only one aspect of Python’s ease of use. It also provides a deliberately simple syntax and powerful built-in tools. In fact, some have gone so far as to call Python “executable pseudocode.” Because it eliminates much of the complexity in other tools, Python programs are simpler, smaller, and more flexible than equivalent programs in languages like C, C++, and Java!

Por supuesto, la rotación del ciclo de desarrollo es solo un aspecto de la facilidad de uso de Python. También proporciona una sintaxis deliberadamente simple y potentes herramientas integradas. De hecho, algunos han ido tan lejos como para llamar a Python "pseudocódigo ejecutable". Debido a que elimina gran parte de la complejidad de otras herramientas, los programas de Python son más simples, pequeños y flexibles que los programas equivalentes en lenguajes como C, C++ y Java.

**Python Is Engineering, Not Art**

When Python first emerged on the software scene in the early 1990s, it spawned what is now something of a classic conflict between its proponents and those of another popular scripting language, Perl. Personally, I think the debate is tired and unwarranted today—developers are smart enough to draw their own conclusions. Still, this is one of the most common topics I’m asked about on the training road, so it seems fitting to say a few words on the topic here.

Cuando Python apareció por primera vez en la escena del software a principios de la década de 1990, generó lo que ahora es una especie de conflicto clásico entre sus defensores y los de otro lenguaje de secuencias de comandos popular, Perl. Personalmente, creo que el debate está cansado e injustificado hoy en día: los desarrolladores son lo suficientemente inteligentes como para sacar sus propias conclusiones. Aún así, este es uno de los temas más comunes sobre los que me preguntan en el camino de la capacitación, por lo que parece apropiado decir algunas palabras sobre el tema aquí.

The short story is this: you can do everything in Python that you can in Perl, but you can read your code after you do it. That’s it—their domains largely overlap, but Python is more focused on producing readable code. For many, the enhanced readability of Python translates to code reusability and maintainability, making Python a better choice for programs that will not be written once and thrown away. Perl code is easy to write, but difficult to read. Given that most software has a lifespan much longer than its initial creation, many see Python as a more effective tool. The somewhat longer story reflects the backgrounds of the designers of the two languages, and underscores some of the main reasons people choose to use Python. Python’s creator is a mathematician by training; as such, he produced a language with a high degree of uniformity—its syntax and toolset are remarkably coherent. Moreover, like math, its design is orthogonal—most of the language follows from a small set of core concepts. For instance, once one grasps Python’s flavor of polymorphism, the rest is largely just details.

La historia corta es esta: puedes hacer todo lo que puedes hacer en Python en Perl, pero puedes leer tu código después de hacerlo. Eso es todo: sus dominios se superponen en gran medida, pero Python está más enfocado en producir código legible. Para muchos, la legibilidad mejorada de Python se traduce en la reutilización y el mantenimiento del código, lo que convierte a Python en una mejor opción para los programas que no se escribirán una vez y se desecharán. El código Perl es fácil de escribir, pero difícil de leer. Dado que la mayoría del software tiene una vida útil mucho más larga que su creación inicial, muchos ven a Python como una herramienta más efectiva. La historia algo más larga refleja los antecedentes de los diseñadores de los dos lenguajes y subraya algunas de las razones principales por las que las personas eligen usar Python. El creador de Python es matemático de formación; como tal, produjo un lenguaje con un alto grado de uniformidad: su sintaxis y conjunto de herramientas son notablemente coherentes. Además, al igual que las matemáticas, su diseño es ortogonal: la mayor parte del lenguaje se deriva de un pequeño conjunto de conceptos básicos. Por ejemplo, una vez que uno capta el sabor del polimorfismo de Python, el resto son en gran medida solo detalles.

By contrast, the creator of the Perl language is a linguist, and its design reflects this heritage. There are many ways to accomplish the same tasks in Perl, and language constructs interact in context-sensitive and sometimes quite subtle ways—much like natural language. As the well-known Perl motto states, “There’s more than one way to do it.” Given this design, both the Perl language and its user community have historically encouraged freedom of expression when writing code. One person’s Perl code can be radically different from another’s. In fact, writing unique, tricky code is often a source of pride among Perl users.

Por el contrario, el creador del lenguaje Perl es un lingüista y su diseño refleja esta herencia. Hay muchas formas de realizar las mismas tareas en Perl, y las construcciones del lenguaje interactúan de forma sensible al contexto y, a veces, de formas muy sutiles, como el lenguaje natural. Como dice el conocido lema de Perl: "Hay más de una forma de hacerlo". Dado este diseño, tanto el lenguaje Perl como su comunidad de usuarios han fomentado históricamente la libertad de expresión al escribir código. El código Perl de una persona puede ser radicalmente diferente al de otra. De hecho, escribir código único y complicado suele ser motivo de orgullo entre los usuarios de Perl.

But as anyone who has done any substantial code maintenance should be able to attest, freedom of expression is great for art, but lousy for engineering. In engineering, we need a minimal feature set and predictability. In engineering, freedom of expression can lead to maintenance nightmares. As more than one Perl user has confided to me, the result of too much freedom is often code that is much easier to rewrite from scratch

than to modify.

Pero como cualquiera que haya realizado un mantenimiento sustancial del código debería poder atestiguar, la libertad de expresión es excelente para el arte, pero pésima para la ingeniería. En ingeniería, necesitamos un conjunto mínimo de características y previsibilidad. En ingeniería, la libertad de expresión puede conducir a pesadillas de mantenimiento. Como me ha confiado más de un usuario de Perl, el resultado de demasiada libertad suele ser un código que es mucho más fácil de reescribir desde cero que de modificar.

Consider this: when people create a painting or a sculpture, they do so for themselves for purely aesthetic purposes. The possibility of someone else having to change that painting or sculpture later does not enter into it. This is a critical difference between art and engineering. When people write software, they are not writing it for themselves. In fact, they are not even writing primarily for the computer. Rather, good programmers know that code is written for the next human being who has to read it in order to maintain or reuse it. If that person cannot understand the code, it’s all but useless in a realistic development scenario. This is where many people find that Python most clearly differentiates itself from scripting languages like Perl. Because Python’s syntax model almost forces users to write readable code, Python programs lend themselves more directly to the full software development cycle. And because Python emphasizes ideas such as limited interactions, uniformity, regularity, and consistency, it more directly fosters code that can be used long after it is first written.

Considere esto: cuando las personas crean una pintura o una escultura, lo hacen para sí mismos con fines puramente estéticos. No entra en ella la posibilidad de que otro tenga que cambiar después ese cuadro o esa escultura. Esta es una diferencia crítica entre el arte y la ingeniería. Cuando las personas escriben software, no lo hacen para sí mismas. De hecho, ni siquiera están escribiendo principalmente para la computadora. Más bien, los buenos programadores saben que el código está escrito para el siguiente ser humano que tiene que leerlo para mantenerlo o reutilizarlo. Si esa persona no puede entender el código, es casi inútil en un escenario de desarrollo realista. Aquí es donde muchas personas encuentran que Python se diferencia más claramente de los lenguajes de secuencias de comandos como Perl. Debido a que el modelo de sintaxis de Python casi obliga a los usuarios a escribir código legible, los programas de Python se prestan más directamente al ciclo completo de desarrollo de software. Y debido a que Python enfatiza ideas tales como interacciones limitadas, uniformidad, regularidad y consistencia, fomenta de manera más directa el código que se puede usar mucho después de que se escribe por primera vez.

In the long run, Python’s focus on code quality in itself boosts programmer productivity, as well as programmer satisfaction. Python programmers can be creative, too, and as we’ll see, the language does offer multiple solutions for some tasks. At its core, though, Python encourages good engineering in ways that other scripting languages often do not. At least, that’s the common consensus among many people who have adopted Python. You should always judge such claims for yourself, of course, by learning what Python has to offer. To help you get started, let’s move on to the next chapter.

A la larga, el enfoque de Python en la calidad del código en sí mismo aumenta la productividad del programador, así como la satisfacción del programador. Los programadores de Python también pueden ser creativos y, como veremos, el lenguaje ofrece múltiples soluciones para algunas tareas. Sin embargo, en su esencia, Python fomenta la buena ingeniería de una manera que otros lenguajes de secuencias de comandos a menudo no lo hacen. Al menos, ese es el consenso común entre muchas personas que han adoptado Python. Siempre debe juzgar tales afirmaciones por sí mismo, por supuesto, aprendiendo lo que Python tiene para ofrecer. Para ayudarlo a comenzar, pasemos al siguiente capítulo.

**It’s Easy to Learn**

This brings us to a key point of this book: compared to other programming languages, the core Python language is remarkably easy to learn. In fact, you can expect to be coding significant Python programs in a matter of days (or perhaps in just hours, if you’re already an experienced programmer). That’s good news for professional developers seeking to learn the language to use on the job, as well as for end users of systems that expose a Python layer for customization or control. Today, many systems rely on the fact that end users can quickly learn enough Python to tailor their Python customizations’ code onsite, with little or no support. Although Python does have advanced programming tools, its core language is still simple for beginners and gurus alike.

Esto nos lleva a un punto clave de este libro: en comparación con otros lenguajes de programación, el lenguaje central de Python es notablemente fácil de aprender. De hecho, puede esperar estar codificando importantes programas de Python en cuestión de días (o quizás en solo unas horas, si ya es un programador experimentado). Esas son buenas noticias para los desarrolladores profesionales que buscan aprender el lenguaje para usar en el trabajo, así como para los usuarios finales de sistemas que exponen una capa de Python para personalización o control. Hoy en día, muchos sistemas se basan en el hecho de que los usuarios finales pueden aprender rápidamente suficiente Python para adaptar el código de sus personalizaciones de Python en el sitio, con poco o ningún soporte. Aunque Python tiene herramientas de programación avanzadas, su lenguaje central sigue siendo simple tanto para principiantes como para gurús.

**It’s Named After Monty Python**

OK, this isn’t quite a technical strength, but it does seem to be a surprisingly well-kept secret that I wish to expose up front. Despite all the reptile icons in the Python world, the truth is that Python creator Guido van Rossum named it after the BBC comedy series Monty Python’s Flying Circus. He is a big fan of Monty Python, as are many software developers (indeed, there seems to almost be a 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: talks at Python conferences are regularly billed as “The Spanish Inquisition.”

De acuerdo, esto no es una fortaleza técnica, pero parece ser un secreto sorprendentemente bien guardado que deseo exponer por adelantado. A pesar de todos los íconos de reptiles en el mundo de Python, la verdad es que el creador de Python, Guido van Rossum, lo nombró así por la serie de comedia de la BBC Monty Python's Flying Circus. Es un gran admirador de Monty Python, al igual que muchos desarrolladores de software (de hecho, parece haber casi una simetría entre los dos campos). Este legado inevitablemente agrega una calidad humorística a los ejemplos de código de Python. Por ejemplo, los tradicionales "foo" y "bar" para los nombres de variables genéricas se convierten en "spam" y "eggs" en el mundo de Python. Los ocasionales "Brian", "ni" y "shrubbery" también deben su aparición a este homónimo. Incluso afecta a la comunidad de Python en general: las charlas en las conferencias de Python se anuncian regularmente como "La Inquisición española".

All of this is, of course, very funny if you are familiar with the show, but less so otherwise. You don’t need to be familiar with the series to make sense of examples that borrow references to Monty Python (including many you will see in this book), but at least you now know their root.

Todo esto es, por supuesto, muy divertido si está familiarizado con el programa, pero menos de lo contrario. No necesita estar familiarizado con la serie para entender los ejemplos que toman prestadas referencias a Monty Python (incluidos muchos que verá en este libro), pero al menos ahora conoce su raíz.

**How Does Python Stack Up to Language X?**

Finally, to place it in the context of what you may already know, people sometimes compare Python to languages such as Perl, Tcl, and Java. We talked about performance earlier, so here we’ll focus on functionality. While other languages are also useful tools to know and use, many people find that Python:

Finalmente, para ubicarlo en el contexto de lo que quizás ya sepa, las personas a veces comparan Python con lenguajes como Perl, Tcl y Java. Hablamos sobre el rendimiento anteriormente, por lo que aquí nos centraremos en la funcionalidad. Mientras que otros lenguajes también son herramientas útiles para conocer y usar, muchas personas encuentran que Python:

• Is more powerful than Tcl. Python’s support for “programming in the large”

makes it applicable to the development of larger systems.

Es más potente que Tcl. El soporte de Python para la "programación en general" lo hace aplicable al desarrollo de sistemas más grandes.

• Has a cleaner syntax and simpler design than Perl, which makes it more read-

able and maintainable, and helps reduce program bugs.

Tiene una sintaxis más limpia y un diseño más simple que Perl, lo que lo hace más legible y fácil de mantener, y ayuda a reducir los errores del programa.

• Is simpler and easier to use than Java. Python is a scripting language, but Java

inherits much of the complexity and syntax of systems languages such as C++.

Es más simple y fácil de usar que Java. Python es un lenguaje de secuencias de comandos, pero Java hereda gran parte de la complejidad y la sintaxis de los lenguajes de sistemas como C++.

• Is simpler and easier to use than C++, but often doesn’t compete with C++, either; as a scripting language, Python often serves different roles.

Es más simple y fácil de usar que C++, pero a menudo tampoco compite con C++; Como lenguaje de secuencias de comandos, Python a menudo cumple diferentes funciones.

• Is both more powerful and more cross-platform than Visual Basic. Its open source nature also means it is not controlled by a single company.

Es más poderoso y más multiplataforma que Visual Basic. Su naturaleza de código abierto también significa que no está controlado por una sola empresa.

• Is more mature and has a more readable syntax than Ruby. Unlike Ruby and Java, OOP is an option in Python—Python does not impose OOP on users or projects to which it may not apply.

Es más maduro y tiene una sintaxis más legible que Ruby. A diferencia de Ruby y Java, OOP es una opción en Python: Python no impone OOP a los usuarios o proyectos a los que no se puede aplicar.

• Has the dynamic flavor of languages like SmallTalk and Lisp, but also has a simple, traditional syntax accessible to developers as well as end users of customizable systems

Tiene el sabor dinámico de lenguajes como SmallTalk y Lisp, pero también tiene una sintaxis simple y tradicional accesible tanto para desarrolladores como para usuarios finales de sistemas personalizables.

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 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. They do, however, reflect the common experience of many developers who have taken time to explore what Python has to offer.

Especialmente para los programas que hacen más que escanear archivos de texto, y que podrían tener que ser leídos en el futuro por otros (¡o por usted!), muchas personas encuentran que Python encaja mejor que cualquier otro lenguaje de secuencias de comandos o programación disponible en la actualidad. Además, a menos que su aplicación requiera el máximo rendimiento, Python suele ser una alternativa viable a los lenguajes de desarrollo de sistemas como C, C++ y Java: el código de Python será mucho menos difícil de escribir, depurar y mantener. Por supuesto, su autor ha sido un evangelista de Python desde 1992, así que tome estos comentarios como pueda. Sin embargo, reflejan la experiencia común de muchos desarrolladores que se han tomado el tiempo de explorar lo que Python tiene para ofrecer.

**Chapter Summary**

And that concludes the hype portion of this book. In this chapter, we’ve explored some of the reasons that people pick Python for their programming tasks. We’ve also seen how it is applied, and a representative sample of who is using it today. My goal is to teach Python, though, not to sell it. The best way to judge a language is to see it in action, so the rest of this book focuses entirely on the language details we’ve glossed over here. To get started, the next two chapters begin our technical introduction to the language. There, we explore ways to run Python programs, peek at Python’s byte code execution model, and introduce the basics of module files for saving code. The goal will be to give you just enough information to run the examples and exercises in the rest of the book. You won’t really start programming until Chapter 4, but make sure you have a handle on the startup details before moving on.

Y eso concluye la parte exagerada de este libro. En este capítulo, hemos explorado algunas de las razones por las que las personas eligen Python para sus tareas de programación. También hemos visto cómo se aplica y una muestra representativa de quién lo está usando hoy. Sin embargo, mi objetivo es enseñar Python, no venderlo. La mejor manera de juzgar un idioma es verlo en acción, por lo que el resto de este libro se enfoca completamente en los detalles del idioma que hemos mencionado aquí. Para comenzar, los siguientes dos capítulos comienzan nuestra introducción técnica al lenguaje. Allí, exploramos formas de ejecutar programas de Python, echamos un vistazo al modelo de ejecución de código de bytes de Python e introducimos los conceptos básicos de los archivos de módulo para guardar código. El objetivo será brindarle la información suficiente para ejecutar los ejemplos y ejercicios en el resto del libro. Realmente no comenzará a programar hasta el Capítulo 4, pero asegúrese de tener un control sobre los detalles de inicio antes de continuar.

**Chapter Quiz**

In this edition of the book, we will be closing each chapter with a quick pop quiz about the material presented to help you review the key concepts. The answers for these quizzes appear immediately after the questions, and you are encouraged to read the answers once you’ve taken a crack at the questions yourself. In addition to these end-of-chapter quizzes, you’ll find lab exercises at the end of each part of the book, designed to help you start coding Python on your own. For now, here’s your first test. Good luck!

En esta edición del libro, cerraremos cada capítulo con un breve cuestionario sorpresa sobre el material presentado para ayudarlo a repasar los conceptos clave. Las respuestas de estos cuestionarios aparecen inmediatamente después de las preguntas, y se le anima a leer las respuestas una vez que haya respondido a las preguntas usted mismo. Además de estos cuestionarios al final del capítulo, encontrará ejercicios de laboratorio al final de cada parte del libro, diseñados para ayudarlo a comenzar a codificar Python por su cuenta. Por ahora, aquí está tu primera prueba. ¡Buena suerte!

1. What are the six main reasons that people choose to use Python?

2. Name four notable companies or organizations using Python today.

3. Why might you not want to use Python in an application?

4. What can you do with Python?

5. What’s the significance of the Python import this statement?

6. Why does “spam” show up in so many Python examples in books and on the

Web?

7. What is your favorite color?

**Quiz Answers**

How did you do? Here are the answers I came up with, though there may be multiple solutions to some quiz questions. Again, even if you’re sure you got a question right, I encourage you to look at these answers for additional context. See the chapter’s text for more details if any of these responses don’t make sense to you.

1. Software quality, developer productivity, program portability, support libraries, component integration, and enjoyment. Of these, the quality and productivity themes seem to be the main reasons that people choose to use Python.

2. Google, Industrial Light & Magic, Jet Propulsion Labs, ESRI, and many more. Almost every organization doing software development is using Python in some fashion, whether for long-term strategic product development, or for short-term tactical tasks such as testing and system administration.

3. Python’s downside is performance: it won’t run as quickly as fully compiled languages like C and C++. On the other hand, it’s quick enough for most applications, and typical Python code runs at close to C speed anyhow because it invokes linked-in C code in the interpreter. If speed is critical, compiled extensions are available for number-crunching parts of an application.

4. You can use Python for nearly anything you can do with a computer—from web site development and gaming, to robotics and spacecraft control.

5. import this triggers an Easter egg inside Python that displays some of the design philosophies underlying the language. You’ll learn how to run this statement in the next chapter.

6. “Spam” is a reference from a famous Monty Python skit in which people trying to order food in a cafeteria are drowned out by a chorus of Vikings singing about spam. Oh, and it’s also a common variable name in Python scripts....

7. Blue. No, yellow!