

Table des matières

Avant-propos	XXXIII
---------------------------	---------------

PREMIÈRE PARTIE

Découverte de Python	1
-----------------------------------	----------

CHAPITRE 1

Introduction	3
---------------------------	----------

Python ?	3
De qualité	4
Orienté objet	4
Portable	4
Facile à intégrer	5
Hautement productif	5
Dynamique	6
Python et les autres langages	6
Python et Perl	6
Python et Java	7
Python et C++/C#	8
Python et PHP 5	8
Python et Ruby	9

CHAPITRE 2

Python pour quels usages ?.....	11
--	-----------

Administration système	12
Des API simples et efficaces	12
<i>Manipuler des fichiers et des dossiers</i>	12
<i>Manipuler des programmes</i>	13
<i>Envoyer et recevoir des courriers électroniques</i>	14
<i>Échanger des informations avec d'autres systèmes</i>	16
Le match Perl-Python	18
<i>Syntaxe</i>	19

<i>Structures de données</i>	20
<i>Manipulation de texte</i>	21
<i>Conclusion</i>	21
Prototypage rapide d'applications	22
Objectif d'une maquette	22
Maquette d'interfaces	22
Maquette de bibliothèque ou Fake	24
<i>Exemple de prototype de bibliothèque</i>	24
Recherche et calcul scientifique	25
Facilité de prise en main	25
Création ou utilisation d'outils spécialisés	26
Applications de gestion	26
Interface utilisateur	27
Stockage de données	27
<i>Sérialisation des objets</i>	28
<i>Les bases de données relationnelles</i>	29
Applications web	30
En un mot...	30

CHAPITRE 3

Environnement de développement 31

Installation sous Linux	31
Installation par distribution	32
<i>Packages Debian</i>	32
<i>Packages RedHat</i>	33
<i>Distributions Mandrake et Fedora Core</i>	33
Compilation des sources	33
<i>Étapes d'installation</i>	34
<i>Options de compilation</i>	34
<i>Compilation et installation de Python</i>	36
Gérer plusieurs versions de Python	37
Installation sous MS-Windows	38
Installation sous Mac OS X	39
Premiers tests de Python en mode interactif	39
Script de démarrage du mode interactif	40
Le choix d'un éditeur	42
La coloration syntaxique	43
La standardisation automatique	43
Les raccourcis clavier et les macros	43
L'édition multiple	43

Le repliement de code et la recherche	43
L'autocomplétion	44
L'interpréteur et le débogueur embarqués	44
La licence	44
Les plates-formes reconnues	44
En un mot...	47

DEUXIÈME PARTIE

Éléments du langage49

CHAPITRE 4

Syntaxe du langage..... 51

L'instruction <code>print</code>	52
Les commentaires	52
Modèle de données	53
Les littéraux	55
Littéraux alphanumériques	55
Littéraux numériques	57
<i>Littéraux pour les entiers</i>	57
<i>Littéraux pour les valeurs à virgule flottante</i>	59
<i>Littéraux pour les nombres complexes</i>	59
Les types standards	60
Les types à valeur unique	60
<i>None</i>	60
<i>NotImplemented</i>	61
<i>Ellipsis</i>	61
Les nombres	61
<i>Les nombres entiers</i>	61
<i>Les nombres à virgule flottante</i>	62
<i>Les nombres complexes</i>	62
<i>Les décimaux</i>	63
Les séquences	63
<i>Les séquences immuables</i>	64
<i>Les séquences modifiables</i>	68
Les mappings	70
Les opérateurs	74
Opérateurs de base	74
Autres opérateurs	76
<i>Modulo</i>	76
<i>Négation</i>	76

<i>Inversion</i>	77
<i>Puissance</i>	77
<i>Appartenance</i>	77
<i>Opérateurs binaires</i>	78
Opérateurs de comparaison	79
<i>Principes de la comparaison</i>	79
Ordre de traitement des opérations	80
Construction de comparaisons complexes	80
L'indentation	81
Les structures conditionnelles	82
L'instruction if	82
L'instruction for..in	83
L'instruction while	85
En un mot...	86

CHAPITRE 5

Structuration du code **87**

Fonctions	87
Contexte d'exécution et directive global	88
Directive return	89
Paramètres d'une fonction	90
<i>Paramètres explicites et valeurs par défaut</i>	90
<i>Les paramètres non explicites</i>	92
<i>Les paramètres arbitraires</i>	93
<i>Collisions de paramètres</i>	94
<i>Signatures multiples de fonctions</i>	95
Directive lambda	95
Documentation strings (docstrings)	96
Decorators	96
Classes	101
Définition	102
Espace de noms	102
Héritage	104
<i>Héritage multiple</i>	105
<i>Surcharge des attributs</i>	106
<i>Constructeur et destructeur</i>	107
Attributs privés	108
Méthodes spéciales	109
<i>Représentation et comparaison de l'objet</i>	110
<i>Utilisation de l'objet comme fonction</i>	111
<i>Accès aux attributs de l'objet</i>	112

<i>Utilisation de l'objet comme conteneur</i>	113
<i>Utilisation de l'objet comme type numérique</i>	115
New-style classes	116
<i>Constructeur statique</i>	117
<i>Surcharge de type() par metaclass</i>	118
<i>Descriptors</i>	119
<i>Properties</i>	121
<i>Slots</i>	122
<i>Decorators pour les classes</i>	123
Modules	123
Directive import	124
Primitive reload	125
Directives from et as	126
Paquets	127
Organisation d'un paquet	127
Import * et __all__	128
Références relatives	129
Exceptions	129
Exceptions du langage	130
<i>Classes d'exception de base</i>	131
<i>Classes concrètes</i>	132
try..except..else	132
try..finally	134
Les list comprehensions	134
Generators et iterators	136
Iterators	136
Generators	138
Generator expression (genexp)	139
En un mot...	139

CHAPITRE 6

Les primitives 141

Primitives du langage	142
abs : abs(nombre)->nombre	142
apply : apply(object[, args[, kwargs]])->value	142
bool : bool([x])->bool	142
buffer : buffer(object [, offset[, size]])	143
callable : callable(object)->bool	143
chr : chr(i)->character	144
classmethod : classmethod(function)->method	144
cmp : cmp(x, y)->integer	145

<code>coerce : coerce(x, y)->(x1, y1)</code>	146
<code>compile : compile(source, filename, mode[, flags[, dont_inherit]])-></code>	
<code>code object</code>	146
<code>complex : complex(real[, imag])->complex number</code>	147
<code>copyright</code>	148
<code>credits</code>	148
<code>delattr : delattr(object, name)</code>	149
<code>dict : dict()->new empty dictionary</code>	149
<code>dict : dict(mapping)->new dictionary</code>	150
<code>dict : dict(seq)->new dictionary</code>	150
<code>dict : dict(**kwargs)->new dictionary</code>	150
<code>dir : dir([object])->list of strings</code>	150
<code>divmod : divmod(x, y)->(div, mod)</code>	152
<code>enumerate : enumerate(iterable)->iterator for index, value of iterable</code>	152
<code>eval : eval(source[, globals[, locals]])->value</code>	152
<code>execfile : execfile(filename[, globals[, locals]])</code>	153
<code>exit : exit->string</code>	153
<code>file : file(name[, mode[, buffering]])->file object</code>	153
<code>filter : filter(function or None, sequence)->list, tuple, or string</code>	155
<code>float : float(x)->floating point number</code>	156
<code>frozenset : frozenset(iterable)->frozenset object</code>	156
<code>getattr : getattr(object, name[, default])->value</code>	156
<code>globals : globals()->dictionary</code>	157
<code>hasattr : hasattr(object, name)->bool</code>	157
<code>hash : hash(object)->integer</code>	158
<code>help : Define the built-in 'help'</code>	158
<code>hex : hex(number)->string</code>	158
<code>id : id(object)->integer</code>	159
<code>input : input([prompt])->value</code>	159
<code>int : int(x[, base])->integer</code>	159
<code>intern : intern(string)->string</code>	160
<code>isinstance : isinstance(object, class-or-type-or-tuple)->bool</code>	160
<code>issubclass : issubclass(C, B)->bool</code>	161
<code>iter : iter(collection)->iterator ou iter(callable, sentinel)->iterator</code>	161
<code>len : len(object)->integer</code>	162
<code>licence : licence()->interactive prompt</code>	162
<code>list : list()->new list et list(sequence)->new list</code>	163
<code>locals : locals()->dictionary</code>	164
<code>long : long(x[, base])->integer</code>	164
<code>map : map(function, sequence[, sequence...])->list</code>	164
<code>min : min(sequence)->value</code>	165

object : The most base type	166
oct : oct(number)->string	166
open : file(name[, mode[, buffering]])->file object	166
ord : ord(c)->integer	166
pow : pow(x, y[, z])->number	166
property : property(fget=None, fset=None, fdel=None, doc=None)-> property attribute	166
quit : quit->string	167
range : range([start,] stop[, step])->list of integers	168
raw_input : raw_input([prompt])->string	168
reduce : reduce(function, sequence[, initial])->value	169
reload : reload(module)->module	170
repr : repr(object)->string	170
round : round(number[, ndigits])->floating point number	170
set : set(iterable)->set object	171
setattr : setattr(object, name, value)	171
slice : slice([start,] stop[, step])	171
sorted : sorted(iterable, cmp=None, key=None, reverse=False)-> new sorted list	172
staticmethod : staticmethod(function)->method	173
str : str(object)->string	174
sum : sum(sequence, start=0)->value	175
super : super(type, obj)->bound super object	175
tuple : tuple([sequence])->tuple	176
type : type(object)->the object's type	176
type : type(name, bases, dict)->a new type	177
unichr : unichr(i)->Unicode character	177
unicode : unicode(string [, encoding[, errors]])->object	177
vars : vars([object])->dictionary	178
xrange : xrange([start,] stop[, step])->xrange object	178
zip : zip(seq1 [, seq2 [...]])->[(seq1[0], seq2[0]...), (...)]	178
Exceptions du langage	179
Erreurs	179
<i>AssertionError</i>	179
<i>AttributeError</i>	180
<i>EOFError</i>	180
<i>FloatingPointError</i>	180
<i>IOError</i>	181
<i>ImportError</i>	181
<i>IndentationError</i>	181
<i>IndexError</i>	181

<i>KeyError</i>	181
<i>KeyboardInterrupt</i>	182
<i>MemoryError</i>	182
<i>NameError</i>	182
<i>NotImplementedError</i>	182
<i>OSError</i>	183
<i>OverflowError</i>	183
<i>ReferenceError</i>	183
<i>RuntimeError</i>	183
<i>StopIteration</i>	184
<i>SyntaxError</i>	184
<i>SystemError</i>	184
<i>SystemExit</i>	184
<i>TabError</i>	184
<i>TypeError</i>	185
<i>UnboundLocalError</i>	185
<i>UnicodeEncodeError</i>	185
<i>UnicodeDecodeError</i>	186
<i>UnicodeTranslateError</i>	186
<i>ValueError</i>	186
<i>WindowsError</i>	187
<i>ZeroDivisionError</i>	187
Avertissements	187
<i>UserWarning</i>	187
<i>DeprecationWarning</i>	187
<i>FutureWarning</i>	188
<i>OverflowWarning</i>	188
<i>PendingDeprecationWarning</i>	188
<i>RuntimeWarning</i>	188
<i>SyntaxWarning</i>	188
En un mot...	188

CHAPITRE 7

Conventions de codage..... 189

Mise en page du code	190
Indentation	190
Taille maximum d'une ligne	190
Commentaires	191
<i>Commentaires simples</i>	192
<i>Commentaires en fin de ligne</i>	192
<i>Blocs de commentaires</i>	192

<i>Documentation strings ou docstrings</i>	193
Espaceur du code	194
Espaces dans les expressions et définitions	195
Conventions de nommage	196
Modules	197
Classes	198
Fonctions et variables globales d'un module, méthodes et attributs d'une classe ..	199
Constantes	199
Structure d'un module	199
En-tête	199
<i>Interpréteur</i>	199
<i>Encodage</i>	200
<i>Copyright et licence</i>	200
<i>Tags</i>	200
Docstring de module	201
Variables globales spécifiques	201
Clauses d'importations	201
<i>Les jokers</i>	202
<i>Organisation des clauses</i>	202
Variables globales	203
Fonctions et classes, le corps du module	203
<i>Structuration d'une classe</i>	204
Conseils pour le choix des noms	204
Règles générales	205
<i>Du sens</i>	205
<i>Choix de la langue</i>	205
<i>Unicité des noms</i>	205
<i>La bonne longueur</i>	205
<i>Éviter le mélange domaine/technique</i>	205
Règles pour chaque type	206
<i>Modules</i>	206
<i>Classes</i>	206
<i>Méthodes et fonctions</i>	206
<i>Variables</i>	207
En un mot...	207

TROISIÈME PARTIE

La bibliothèque standard 209

CHAPITRE 8

Principaux modules, partie 1 211

Interaction avec l'interpréteur	212
sys	212
<i>argv</i>	212
<i>executable</i>	212
<i>exc_info()</i> -> <i>infos</i>	212
<i>exit()</i>	212
<i>modules</i>	213
<i>last_type, last_value, last_traceback</i>	213
<i>path</i>	213
<i>platform</i>	213
<i>stdin, stdout et stderr</i>	214
Accès au système	214
os	214
<i>Opérations sur les descripteurs de fichiers</i>	215
<i>Manipulation des fichiers et répertoires</i>	217
<i>Manipulation des processus</i>	225
<i>Informations sur le système</i>	231
subprocess	233
<i>call(*args, **kwargs)</i> -> <i>returncode</i>	233
<i>class Popen</i>	233
os.path	235
platform	236
Utilitaires fichiers	238
shutil	238
<i>copy(src, dst)</i>	238
<i>copy2(src, dst)</i>	239
<i>copytree(src, dst[, symlinks])</i>	239
<i>rmtree(path, [ignore_errors[, onerror]])</i>	239
<i>move(src, dst)</i>	239
dircache	239
filecmp	240
<i>cmp(f1, f2[, shallow=True[, use_statcache]])</i> -> <i>bool</i>	240
<i>class dircmp(a, b[, ignore[, hide]])</i> -> <i>instance</i>	240
Outils de compression	242
gzip	242

<i>class GzipFile([filename[, mode[, compresslevel[, fileobj]]])</i>	242
<i>open(filename[, mode[, compresslevel]])</i>	243
zipfile	245
<i>class ZipFile(file[, mode[, compression]])</i>	245
<i>class ZipInfo([filename[, date_time]])</i>	246
<i>is_zipfile(filename)-> bool</i>	247
Programmation réseau	247
urllib2	248
ftplib	251
En un mot...	254

CHAPITRE 9

Principaux modules, partie 2..... 255

Persistence	255
cPickle	256
<i>dump(object, file[, protocol])</i>	256
<i>load(file)->object</i>	256
<i>dumps(object[, protocol]->string</i>	257
<i>loads(string[, protocol]->object</i>	257
shelve	258
<i>open(filename[, flag[, protocol[, writeback]])</i>	259
Conversion, transformation de données	260
base64	260
<i>b64encode(s[, altchars]->string</i>	260
<i>b64decode(s[, altchars]->string</i>	260
md5	261
<i>new([s])</i>	261
<i>class md5([s])</i>	262
sha	262
Calculs numériques	263
math	263
<i>fonctions de conversion</i>	263
<i>fonctions trigonométriques</i>	264
<i>constantes</i>	265
Structures de données	266
array	266
<i>array(typecode[, initializer]->array</i>	266
collections	268
<i>Le type deque</i>	268
decimal	269
<i>class Decimal([value [, context]])</i>	269

cStringIO	270
<i>class StringIO([buffer])</i>	270
Utilitaires divers	271
atexit	271
pdb	272
<i>Le mode pas à pas</i>	272
<i>Alias et fichier .pdbrc</i>	277
<i>Le mode post mortem</i>	278
getpass	280
copy	280
difflib	281
<i>Affichage des différences</i>	281
<i>Restauration</i>	284
time	285
<i>Epoch</i>	285
<i>UTC/GMT</i>	285
<i>Fonctions de manipulation</i>	286
<i>Formatage des dates</i>	287
datetime	289
<i>class timedelta</i>	289
<i>class date</i>	290
<i>class time</i>	292
<i>class datetime</i>	294
<i>random</i>	294
En un mot...	296

CHAPITRE 10

Principaux modules, partie 3..... 297

Le module itertools	297
chain(*iterables)->iterator	297
count([firstval])->iterator	298
cycle(iterable)->iterator	298
dropwhile(predicate, iterable)->iterator	298
groupby(iterable[, keyfunc])->iterator	299
ifilter(predicate, iterable)->iterator	300
ifilterfalse(predicate, iterable)->iterator	300
imap(function, *iterables)->iterator	300
islice(iterable, [start,] stop [, step])->iterator	301
izip(*iterables)->iterator	301
repeat(element, times)->iterator	301
starmap(function, sequence)->iterator	302
takewhile(predicate, iterable)->iterator	302
tee(iterable[, n=2])->tuple of iterators	302

Le module re	303
Expressions régulières ?	303
Notation pour les expressions régulières	304
Syntaxe des expressions régulières	305
<i>Symboles simples</i>	305
<i>Symboles de répétition</i>	307
<i>Symboles de regroupement</i>	308
<i>Exemples plus complets</i>	310
<i>Fonctions et objets de re</i>	310
Le module Tkinter	313
Programmation événementielle	314
La classe Tk	314
Les widgets de base de Tkinter	315
<i>Positionnement d'un widget</i>	316
<i>Options et méthodes d'un widget</i>	317
Binding d'événements	326
Application type avec Tkinter	328
Extensions pour Tkinter	330
En un mot...	330

CHAPITRE 11

Exercices corrigés	331
Mode d'emploi du chapitre	331
Programme	332
Exercice 1 : programme paramétrable	332
Texte	335
Exercice 2 : le chiffrement de César	335
Exercice 3 : transformer les adresses e-mails et les URL d'un texte en liens . . .	337
Exercice 4 : trier des phrases suivant le nombre de mots	339
Fichiers	341
Exercice 5 : recherche et remplacement de texte	341
Exercice 6 : ajout d'un fichier dans une archive zip	343
Threads	346
Exercice 7 : Tkinter, recherche d'un texte dans des fichiers en tâche de fond .	346
Exercice 8 : le design pattern producteur-consommateur	351
Persistence	354
Exercice 9 : rendre persistants tous les objets d'un programme	354
Web et réseau	357
Exercice 10 : vérificateur de liens	357
Exercice 11 : aspirateur de page web	358

Exercice 12 : récupération d'un résumé des nouveaux e-mails reçus	362
Divers	365
Exercice 13 : système de documentation en ligne des modules	365
En un mot...	368

QUATRIÈME PARTIE

Techniques avancées 369

CHAPITRE 12

Programmation dirigée par les tests 371

À quoi servent les tests ?	372
Barrière culturelle	372
Principes	373
Tests unitaires	373
<i>Construction d'un test unitaire</i>	374
<i>Évolution des use cases</i>	375
<i>Non-régression</i>	376
<i>Regroupement des tests</i>	379
<i>Tests plus complexes : raconter une histoire</i>	379
<i>Les bouchons</i>	380
<i>Test coverage</i>	385
<i>Qualité des tests</i>	385
Tests fonctionnels	385
<i>Tests de l'interface</i>	386
<i>Tests de l'ergonomie</i>	386
<i>Dépendance forte à l'outil utilisé et au type d'interface</i>	387
Outils	387
unittest	388
<i>Définition des test cases</i>	388
<i>Organisation d'une campagne de tests</i>	392
doctests	395
<i>Exécution des doctests</i>	396
<i>Syntaxe des doctests</i>	397
<i>Environnement et options d'exécution</i>	400
<i>doctests dans un fichier texte séparé</i>	405
<i>Script de test</i>	408
Coverage	410
Intégration dans l'environnement d'un projet	413
En un mot...	414

CHAPITRE 13

Bonnes pratiques et optimisation du code 415

Quand optimiser ?	416
Profiling	416
Méthodes de profiling	417
Outils de profiling	417
<i>Le module profile</i>	417
<i>Le module hotshot</i>	418
<i>Le module pstats</i>	419
<i>hotshot et pstats</i>	421
<i>timeit</i>	422
Amélioration des performances	424
Code patterns	425
<i>Quel type de conteneur choisir ?</i>	425
<i>Trier des valeurs</i>	425
<i>Concaténer des chaînes</i>	428
<i>Remplacer certains tests par une gestion d'exception</i>	429
<i>Minimiser les appels et rapprocher le code</i>	430
<i>Utiliser les list comprehensions</i>	432
<i>Utiliser les generators et les genexp</i>	433
<i>Préférer les fonctions d'itertools</i>	433
Caching	434
Multithreading	436
<i>Ressources partagées : difficultés de programmation</i>	437
<i>Le module threading</i>	438
<i>Le module Queue</i>	446
Le côté obscur de la force : extension du langage	447
<i>Environnement de compilation</i>	447
<i>Binding de bibliothèque</i>	448
<i>Création d'un module d'extension</i>	452
Optimisation de l'utilisation de mémoire vive	458
<i>Économie de mémoire</i>	459
Optimisation du bytecode	460
Psyco et Pyrex	460
<i>Psyco</i>	460
<i>Pyrex</i>	462
Les tests de performance continus	464
Rapport sur les performances	464
Tests de performance ciblés	465
decorator timed	465
En un mot...	468

CHAPITRE 14

Programmation orientée objet 469

Principes généraux	469
Typage, classification et encapsulation	470
<i>Typage de Liskov</i>	470
<i>Encapsulation</i>	471
Héritage et polymorphisme	473
<i>Héritage</i>	473
<i>Polymorphisme</i>	474
<i>Duck typing et interfaces</i>	476
Relations entre objets	478
<i>Relation simple</i>	478
<i>Relation multiple</i>	479
Héritage multiple	480
Classes abstraites	481
Métaclasses	481
Garbage collecting	481
Design patterns orientés objet	482
Patterns de génération d'objets	482
<i>Singleton et Borg</i>	483
<i>Factory</i>	486
Patterns fonctionnels	487
<i>Visitor</i>	487
<i>Observer</i>	489
<i>Memento</i>	493
<i>Chain of responsibility</i>	495
<i>State</i>	499
Patterns structurels	501
<i>Adapter</i>	501
<i>Facade</i>	503
<i>Proxy</i>	504
En un mot...	506

ANNEXE A

L'histoire de Python 507

Le langage ABC	507
Le projet Amoeba	509
Le CNRI	510
PythonLabs et BeOpen.com	510
Python Software Foundation et Digital Creations	511
Python et Zope	511

ANNEXE B

Bibliothèques tierces	513
Installer une bibliothèque externe	514
Bases de données	515
Gadfly	516
pysqlite	516
mysql-python	516
psycopg	516
ODBC	516
python-ldap	517
SQLObject	517
Traitement de texte	517
ElementTree	517
lxml	518
Beautiful Soup	518
Packaging, distribution	518
setuptools	518
Tests fonctionnels et contrôle qualité	519
Twill	519
FunkLoad	519
guitest	520
PyLint	520
Pyflakes	520
MS-Windows	520
Win32 Extensions	520
win32com	521
Interfaces graphiques	521
wxPython	521
PyQT	521
PyGTK	522
Reporting et conversion	522
ReportLab	522
RML2PDF	522
reStructuredText	523
rest2web	523
Jeux et 3D	523
Pygame	523
Soya 3D	524
vpython	524
PyOpenGL	524

Audio et Vidéo	524
PyMedia	524
PyAlsa	525
Bibliothèques scientifiques	525
Numerical Python	525
SciPy	525
Biopython	525
Web	526
CheeseShop	526
ANNEXE C	
Sites, flux RSS, blogs et autres friandises...	527
Flux RSS	527
Blogs	529
Sites	530
Index	531