



Tr
e
o
x
E

Scripting Python sous Linux

Développez vos outils système

En téléchargement



scripts des exemples



+ QUIZ

Version en ligne

OFFERTE !

pendant 1 an

Christophe BONNET





Les éléments à télécharger sont disponibles à l'adresse suivante :

<http://www.editions-eni.fr>

Saisissez la référence ENI de l'ouvrage **EISCRPYT** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Avant-propos

Partie 1 : Découvrir

Chapitre 1-1

L'environnement de travail

1. Python2 ou Python3 ? 15
2. L'environnement de travail 16
3. Un terminal et l'interpréteur Python 16
4. La configuration de l'environnement Python. 17
 - 4.1 Étape 1 : repérez le binaire Python qui vous intéresse 18
 - 4.2 Étape 2 : vérifiez la présence
de la commande pip qui va avec 18
 - 4.3 Étape 3 : vérifiez le module virtualenv 19
 - 4.4 Étape 4 : installez virtualenv wrapper. 19
5. Les autres outils nécessaires. 22

Chapitre 1-2

Le côté fonctionnel classique de Python

1. Introduction 23
2. Premiers pas 23
 - 2.1 La commande python. 24
 - 2.2 L'indentation comme syntaxe. 26

2 _____ Scripting Python sous Linux

Développez vos outils système

Chapitre 1-3

Quelques instructions de base

1. Introduction	33
2. Les opérateurs	33
3. Les variables	35
4. Quelques instructions de base	36
5. Résumé	38

Chapitre 1-4

Les types de données en Python

1. Introduction	39
2. Les booléens	40
2.1 Les opérateurs booléens	41
2.2 Les comparaisons logiques	42
3. Les numériques	43
3.1 Les entiers	43
3.2 Les flottants	44
3.3 Les opérations	45
4. Les alphanumériques	47
4.1 Les opérations applicables aux chaînes de caractères	49
4.2 Les méthodes applicables aux chaînes de caractères	50
4.3 Les modificateurs de chaînes ou "string modifiers"	57
5. Les conteneurs ou séquentiels	59
5.1 Les listes	59
5.2 Les dictionnaires	69
5.3 Les tuples	75
5.4 Les sets	76
5.5 Les frozensets	81
6. Autres types de données	81
7. Résumé	83

Chapitre 1-5 Le langage Python

1. Introduction	85
2. La fonction print()	85
2.1 Print() formatage C-STYLE	86
2.2 Print() formatage chaîne.format()	89
2.3 Print() les autres options	94
3. Les structures conditionnelles	95
4. Les boucles.	97
4.1 Quelques exemples simples	98
4.2 La fonction range()	99
4.3 Un "else" dans les boucles.	100
4.4 sorted() et sort()	102
4.5 enumerate()	102
4.6 L'affectation parallèle dans les boucles	103
4.7 Les dictionnaires et la fonction items()	104
4.8 Les listes en compréhension.	106
5. Les fonctions	108
5.1 Les arguments de fonctions avec Python	110
5.2 Les fonctions et la portée des variables	114
5.3 La notion de passage de paramètres par référence	118
6. Les modules et les paquets	121
6.1 Les espaces de noms	123
6.2 Les paquets ou packages.	125
6.3 La recherche des modules et paquets.	129
6.4 Le fichier __main__.py	129
6.5 Exemple avec la gestion d'un restaurant.	129
7. Les exceptions et la gestion des erreurs	140
7.1 Les instructions try ... except ... finally	141
7.2 L'instruction assert	143
7.3 Déclencher des exceptions	144

4 _____ Scripting Python sous Linux

Développez vos outils système

8. Les entrées/sorties (fichier et autres)	146
9. L'instruction With	150
10. Exemple de script : hdump	152
11. Résumé	155

Chapitre 1-6

Le côté Objet de Python

1. Introduction	157
2. La POO	158
3. L'objet	158
4. La classe	159
5. Une classe simple	159
6. Ajoutons des attributs	165
7. Un premier script « objet »	167
8. La surcharge de fonction	170
9. L'héritage	171
10. Exemples de scripts	173
10.1 Ajoutons le format HTML	180
10.2 Ajoutons la balise <table>	183
10.3 Ajoutons le format CSV	185
11. Résumé	189

Chapitre 1-7

Librairie standard et quelques autres

1. Introduction	191
2. La commande pip	192
3. Les modules sys et os	194
3.1 Le module sys	194

3.2	Le module os	195
4.	Les options de la ligne de commandes.	197
5.	L'interception des signaux	200
6.	Les fichiers temporaires	203
7.	Les modules pour les opérations sur les fichiers et les répertoires.	204
7.1	os.path.	204
7.2	shutil	206
7.2.1	filecmp.	206
7.3	path.py	209
7.4	pathlib.	209
8.	La gestion des processus et sous-processus.	211
8.1	Subprocess.run()	212
8.2	subprocess.Popen()	214
8.3	Envoyer une commande plus complexe et récupérer la sortie.	216
8.4	Multiprocessing - Le parallélisme par processus.	218
9.	Mathplotlib.	224
10.	Les expressions régulières (it's a kind of magic)	227
10.1	Exemple : récupérer des informations sur l'état du système avec sar (system activity report)	235
10.2	Exemple : récupérer des informations sur la mémoire des processus.	237
11.	Les dates et le temps (back to the future).	241
11.1	stdlib : calendar, datetime, dateutil, time	242
11.2	Arrow	244
12.	Le module logging.	245
13.	L'accès aux fichiers en mode binaire et le module struct	251
14.	La génération de données aléatoires	253
15.	L'accès aux bases de données	256
15.1	Les bases de données "SQL"	258
15.2	Les bases de données "NoSQL".	269

6 _____ Scripting Python sous Linux

Développez vos outils système

16. Les ORM ou Object Relationnal Mapping	277
16.1 SQLAlchemy	278
16.2 Les autres ORM	287
17. Réseau	289
17.1 Un serveur web en une ligne de commande	289
17.2 Envoyer des mails	290
17.3 Python et ssh	292
17.4 Le transfert de fichier avec ftplib	294
17.5 telnet lib	297
18. Python et le réseau des réseaux : Internet	302
18.1 Urllib : requests	303
18.2 BeautifulSoup	308
19. Outils	314
19.1 Pexpect	314
19.2 Cmd	319
19.3 shlex - Analyse lexicale simple	326
19.4 Le module humanfriendly	328
20. Résumé	329

Chapitre 1-8

Aller plus loin avec le langage Python et la POO

1. Introduction	331
2. Quelques concepts objet essentiels	331
2.1 Le polymorphisme	331
2.2 L'héritage multiple	332
2.3 Le singleton	335
2.4 La fabrique d'objets	337
2.5 La fermeture, ou closure en anglais	339
3. Les méthodes spéciales d'instances	341
3.1 Les fonctions spéciales classiques	342
3.2 La surcharge des opérateurs	347

4. Le gestionnaire de contexte (with) <code>__enter__</code> , <code>__exit__</code>	350
5. Les objets mutables et non mutables	352
5.1 Les mutables	353
5.2 Les non mutables	354
6. Quelques informations supplémentaires sur les classes en Python.	358
6.1 Les attributs implicites.	358
6.2 Les fonctions et classes incluses.	359
6.3 Les attributs statiques et méthodes statiques	361
6.4 Les <code>__slots__</code> pour les performances	363
7. Les docstrings - chaînes de documentation.	365
7.1 Définition	365
7.2 Usage	366
7.3 Génération de documentation.	369
7.4 Tests	371
8. Les décorateurs	372
9. Les itérateurs, générateurs et autres expressions génératrices	375
9.1 Les itérateurs.	375
9.2 Les générateurs	378
10. Gérer ses propres exceptions	381
11. Les fonctions natives	383
11.1 Les fonctions natives inclassables	383
11.2 Les fonctions natives binaires	385
11.3 Les fonctions natives de conversion ou de création de type	385
11.4 Les fonctions natives sur les itérables	386
11.5 Les fonctions natives sur les numériques	387
11.6 Les fonctions natives sur les objets	388
12. Résumé	391

8 _____ Scripting Python sous Linux

Développez vos outils système

Partie 2 : Pratiquer

Chapitre 2-1

Récupérer des infos sur le système

1. Introduction	393
2. psutil : récupérer des informations sur le système	394
3. Des informations sur les composants	396
3.1 Les processeurs	396
3.2 La mémoire	398
3.3 Les périphériques de stockage	401
3.4 Le réseau	402
4. Capteurs et autres informations	406
4.1 Les capteurs	406
4.2 Autres informations	407
4.2.1 Boot time	407
4.2.2 Les utilisateurs	408
5. Des informations sur les processus	408
5.1 Classe Objet et méthodes fournies par psutil	408
5.2 Principes d'utilisation	412
5.3 Exemples d'utilisation	414
6. Résumé	418

Chapitre 2-2

Les formats de fichiers populaires

1. Introduction	419
2. Le format de fichier "INI"	419
3. Comma Separated Values : CSV	424
4. MS Office	427

5. Le module odfpv	429
5.1 Document Texte	430
5.2 Document Feuille de calcul	433
6. Le module multi-format pyexcel	436
7. Le format de fichier JSON	439
8. Le format de fichier XML	441
9. Le module tarfile pour archives tar	447
10. Le format zip	450
11. Résumé	452

Chapitre 2-3

Manipulation de données

1. Introduction	453
2. SQLite en mémoire	454
2.1 La mission	454
2.2 La récupération des données	455
2.3 La définition de la base de données	457
2.4 Le script principal	459
3. SMS vers HTML (ou autre)	462
3.1 Extraction des SMS	463
3.2 Transformation des SMS	463
3.3 Conversion	469
3.4 Script	470
4. D'une base à l'autre	472
4.1 Le contexte	472
4.2 Les schémas	473
4.3 Le script principal	476
5. Résumé	482

10 _____ Scripting Python sous Linux

Développez vos outils système

Chapitre 2-4

La génération de rapports

1. Introduction	485
2. La génération de PDF : Reportlab	485
2.1 Hello World en PDF et Reportlab	486
2.2 Une table avec Reportlab	487
2.3 Encore une table, mais en mieux	492
3. Le moteur de patrons Jinja2	497
3.1 Jinja et le HTML	498
3.2 Ajoutons un peu de CSS (Cascading Style Sheet)	503
3.3 Autre cas d'utilisation	506
4. Un autre moteur de patron : Pug/Jade	511
4.1 Prérequis : installation de Pug	513
4.2 Le début du projet	513
4.3 Bootstrap, c'est bien	514
4.4 Make, c'est bien	519
5. Résumé	522

Chapitre 2-5

Simulation d'activité

1. Introduction	523
2. Description	524
3. La structure des données	525
4. L'initialisation de la base de données	528
4.1 definitions.py	528
4.2 populate.py	533
5. La connexion à la base de données	544
6. Les compteurs	545
7. Les commandes clients	547

8. La livraison des commandes clients.	550
9. La facturation des commandes livrées.	552
10. Le réapprovisionnement du stock	554
11. La réception des commandes fournisseurs	558
12. Utilisation	560
13. Résumé	563

Partie 3 : Progresser

Chapitre 3-1

Trucs et astuces

1. Introduction	565
2. Adapter le copier/coller d'un tableur pour un wiki	566
3. Déballage avec Python (unpacking)	568
4. L'underscore et Python	570
4.1 Dans l'interpréteur	570
4.2 Pour ignorer des valeurs	570
4.3 Dans les boucles	571
4.4 Pour la séparation des milliers	571
4.5 Pour le nommage des variables	572
5. Agacer les CPU et mesurer le temps de son code	572
6. Créer un décorateur (logger interne)	576
7. Bien écrire du code Python (PEP8)	577
7.1 Espaces	579
7.2 Lignes	579
7.3 Tabs et encoding	580
7.4 Encoding : UTF8	580
7.5 Docstring	580
7.6 Nom de variables	580
8. Résumé	581

12 _____ Scripting Python sous Linux

Développez vos outils système

Chapitre 3-2 Exemples

1. Introduction	583
2. Python Bataille	584
3. Python, vérification d'une main au poker	587
4. Génération de fichiers et répertoires aléatoire	592
5. Graphique d'utilisation d'un serveur sur un mois	595
6. psutil analyse des informations des processus	601
7. Améliorer Apache Index Of	609
7.1 WSGI	610
7.2 Analyse des besoins	612
7.3 Bottle	613
7.4 Uwsgi - Comment se faire un environnement de test & dev ?	614
7.5 Le script startup.py	614
7.6 Le script page_html.py	616
7.7 Les templates	620
7.8 Les fichiers de paramètres	623
7.9 La mise en exploitation	626
7.10 Le résultat	627
8. Résumé	628

Chapitre 3-3 Aller plus loin avec Python

1. Introduction	629
2. Bottle & Flask	630
2.1 Bottle	630
2.2 Flask	631

3. Outils	631
3.1 Watchdog	631
3.2 paramiko	632
3.3 Supervisor	633
4. Interfaces utilisateurs	635
4.1 Interface graphique : Tkinter	635
4.2 Autres interfaces graphiques	637
4.3 Interface console: curses	638
4.4 Interface console : Urwid	638
5. Résumé	639

Chapitre 3-4

Pour aller encore plus loin

1. Introduction	641
2. Twisted	641
3. Brython	643
4. Fuse	644
5. Ipython et Jupyter	645
6. Sphinx	648
7. Ansible	652
8. Le framework Django	655
9. Réseau SCAPY	657
10. Apache Airflow	658
11. Résumé	662

14 _____ Scripting Python sous Linux

Développez vos outils système

Annexes

1. Ressources Python	663
2. Memento GIT	665
3. Déboguer en Python	671
4. Compiler Python depuis les sources	674
5. Pourquoi être fan des expressions régulières ?	675
6. Les outils utilisés pour cet ouvrage	679
7. Création d'une machine virtuelle Debian	684
8. Quelques conseils et petites choses à méditer	718
9. La dernière section	720

Index	723
-------------	-----



Partie 2 : Pratiquer

Chapitre 2-1 Récupérer des infos sur le système

1. Introduction

La découverte du langage Python est terminée. Déjà dans votre cerveau de nombreuses nouvelles connexions neuronales se sont créées. Mais celles-ci n'ont pas encore un poids suffisant pour produire du code Python sans forcer. Il est nécessaire de passer en phase d'apprentissage et de pratiquer ce langage, et en premier lieu de regarder et comprendre comment font les autres scripteurs Python.

C'est le but de la deuxième partie de cet ouvrage : pratiquer.

Maintenant, il est possible d'utiliser Python et quelques-uns de ses modules pour en faire des outils du quotidien.

Voici ce que nous allons aborder dans cette deuxième partie :

- Acquérir des informations sur le système
- Utiliser des formats populaires de fichiers
- Manipuler des données
- Générer des rapports et des sites statiques
- Simuler de l'activité sur une base de données

2. psutil : récupérer des informations sur le système

Les premières versions du langage Python ont été inspirées par d'autres langages, tels ABC ou Modula-3, mais surtout le langage C et les outils Unix. Python n'a donc jamais été très loin du système d'exploitation ; de plus, il offre toutes les possibilités du langage C et du système d'exploitation Unix, à portée de script.

Mais au fil du temps, Python est devenu un composant essentiel de nombreuses distributions Linux, a été porté et est compilable sur de très nombreux systèmes d'exploitation. Cela a permis la réalisation de bibliothèques multiplateformes comme `psutil`.

`psutil` (*Python System and process UTILities*) permet de récupérer des informations sur l'utilisation du système et sur la gestion des processus en cours d'exécution.

Ce module regroupe les informations initialement données par de nombreuses commandes Unix telles que : `ps`, `top`, `lsof`, `netstat`, `ifconfig`, `who`, `df`, `kill`, `free`, `nice`, `ionice`, `iostat`, `iotop`, `uptime`, `pidof`, `tty`, `taskset`, `pmap`.

Multiplateforme, il est utilisable sur les plateformes suivantes : Linux, Windows, MacOS, FreeBSD, OpenBSD, NetBSD, Sun Solaris et AIX.

Ce module est essentiel pour superviser, monitorer ou analyser l'utilisation de vos systèmes. Il permet de récupérer des informations sur les processeurs, la mémoire, les disques, les interfaces réseau, les différents capteurs présents et surtout sur les processus en cours d'exécution.

psutil a permis la réalisation d'un outil comme glances, dont voici un exemple de session :

```
XXXXXXXXXX (LinuxMint 18.3 64bit / Linux 4.4.0-170-generic)                                Uptime: 0:32:51

CPU      1.9% nice:    0.0%  LOAD   4-core  MEM    10.6%  SWAP    0.0%
user:    1.0% irq:    0.0%  1 min:  0.24  total: 15.4G total: 15.7G
system:  0.7% iowait: 0.2%  5 min:  0.25  used:  1.63G used:    0
idle:    98.1% steal: 0.0% 15 min:  0.19  free:  13.7G free:  15.7G

NETWORK  Rx/s  Tx/s  TASKS 231 (657 thr), 1 run, 230 slp, 0 oth sorted automatically
eth0      0b    0b
lo       520b  520b
wlan0      0b    0b

DISK I/O  R/s    W/s
loop0      0      0
loop1      0      0
loop2      0      0
loop3      0      0
loop4      0      0
loop5      0      0
sda1       0      0
sda2       0      0
sda3       0      0
sda4       0      0
sda5       0      0
sda6       0      0

FILE SYS  Used  Total
/ (sda5)  215G  240G
ore/8213  89.1M  89.1M
ore/8268  89.1M  89.1M
shell/39  56.5M  56.5M
shell/77  56.5M  56.5M
2020-01-04 08:53:52      No warning or critical alert detected
```

glances

Comme top sur Linux ou topas sur AIX, il se rafraîchit à intervalles réguliers. De nombreux exemples se trouvent dans le répertoire scripts sur GitHub à l'adresse suivante : <https://github.com/giampaolo/psutil>

Même si quelques exemples sont donnés dans cet ouvrage, le site est bien fourni en scripts utiles et significatifs, suffisamment pour que cela soit notifié.

Normalement, psutil est livré avec les versions de Python 3.4 et supérieures, sinon faites 'pip install psutil'.

3. Des informations sur les composants

Un ordinateur, finalement, n'est pas quelque chose de complexe, il n'y a que 4 composants principaux.

Et le module `psutil` permet de tous les examiner en détail.

3.1 Les processeurs

Très simplement, `psutil` permet d'obtenir des informations sur les processeurs avec les méthodes suivantes :

`cpu_times` Retourne un tuple nommé avec le temps passé en secondes par CPU en mode user / system / idle.

Le détail dépend du système.

`percpu=True` permet d'avoir le détail par CPU.

`cpu_percent` Retourne l'utilisation du CPU en pourcentage.

`Interval=1` temporise la première utilisation en secondes.

`percpu=True` permet d'avoir le détail par CPU.

`cpu_times_percent` Retourne les pourcentages de l'utilisation des CPU comme le fait la commande `sar -u (%user %system ...%iowait ...%idle)`.

`cpu_count` Retourne le nombre de CPU logiques.

Si l'argument `logical = False`, retourne le nombre de CPU physiques.

`cpu_stats` Retourne différentes informations comme le nombre de changements de contexte depuis le boot, le nombre d'interruptions...

`cpu_freq` Retourne la fréquence du/des CPU.

`percpu=True` permet d'avoir le détail par CPU.

`getloadavg` Retourne la charge du système il y a 1, 5 et 15 minutes.

Voici un petit exemple pour vérifier si cela se tient avec la commande `sar -u` :

```
# fichier : psutil/cpul.py

import psutil
import datetime
import time

temps = 1
max_temps=5

count=1

while count < max_temps:
    print(datetime.datetime.now().time(),
          psutil.cpu_times_percent(interval=1))
    time.sleep(temps)
    count += 1
```

On met tout cela dans un script shell :

```
python cpul.py > cpul.txt &
sar -u 1 5 >sar.txt &
```

On le lance et il est ensuite possible de consulter les fichiers.

Le résultat de la commande `sar` :

```
Linux 4.4.0-170-generic (W550-BR-007)      04/01/2020      _x86_64_      (4 CPU)

11:51:38      CPU      %user      %nice      %system      %iowait      %steal      %idle
11:51:39      all       3,99       0,00       0,50       0,75       0,00      94,76
11:51:40      all       1,01       0,00       0,75       0,00       0,00      98,24
11:51:41      all       1,74       0,00       0,75       1,00       0,00      96,52
11:51:42      all       2,49       0,00       0,75       1,00       0,00      95,76
11:51:43      all       0,76       0,00       0,25       0,25       0,00      98,74
Moyenne :      all       2,00       0,00       0,60       0,60       0,00      96,80
```

Le résultat de la commande Python `cpu1.py` :

```
11:51:38.720886  scputimes(user=1.0, nice=0.0, system=0.5, idle=98.3,
11:51:40.723420  scputimes(user=1.5, nice=0.0, system=0.7, idle=97.0,
11:51:42.726065  scputimes(user=0.8, nice=0.0, system=0.3, idle=98.7,
11:51:44.728709  scputimes(user=0.5, nice=0.0, system=0.5, idle=98.7,
```

Les valeurs sont quand même assez différentes, sans être non plus foncièrement très éloignées. Mais finalement, une précision absolue n'est pas nécessaire, surtout dans les cas qui nous intéressent.

Généralement, dans notre cas, l'utilisation du CPU est à 10 % près.

Si, sur de grandes périodes de plus de 10-15 minutes en exploitation normale, les valeurs sont :

entre 1 et 75 %	Ok.
à partir de 75 % et au-delà de 80 %	Il faut mettre en œuvre une solution de remplacement ou d'extension.
à partir de 90 %	Il faut planifier l'évolution.

Mais pour être tout à fait honnête, cela n'arrive quasiment plus, au regard de la puissance des processeurs.

3.2 La mémoire

Le module propose deux méthodes pour interroger l'état de la mémoire :

`virtual_memory` Retourne un tuple nommé avec des informations dépendantes du système.

`total` : mémoire totale (sauf la mémoire swap)

`available` : mémoire disponible (sans la mémoire swap)

Pour Linux :

`active` : en cours d'utilisation

`inactive` : marquée comme inutilisée

`buffers` : mémoire cache

`cached` : mémoire cache aussi

`shared` : mémoire partagée

`slab` : cache du noyau

`swap_memory` Retourne des statistiques sur la mémoire swap.

`total` : mémoire swap totale

`used` : mémoire swap utilisée

`free` : mémoire swap libre

`percent` : usage en pourcentage

`sin` : cumul des octets écrits dans la swap

`sout` : cumul des octets sortis de la swap

Voici un petit script pour illustrer ces informations.

Celui-ci crée deux listes (`mem`, `swp`) à partir des deux méthodes fournies par `psutil`. Ensuite, il comble les listes avec des blancs pour faciliter l'affichage en colonne, vu que les deux listes n'auront pas le même nombre d'éléments. Il suffit ensuite de faire un `print ()` par rang.

Ce script est une réécriture du script fourni en exemple `meminfo.py` sur le site de `psutil`.

```
#fichier : psutil/ps_meminfo.py

import psutil
from psutil._common import bytes2human

def fmt_ntuple( nt ):
    r = []
    for name in nt._fields:
        valeur = getattr(nt, name)
        if name != 'percent':
            valeur = bytes2human(valeur)
        else:
            valeur = "{:5.2f}%".format(valeur)
        r.append( "{:10s} : {:>7s}".
            format( name.capitalize(), valeur ))
    return r

def main():
    mem = fmt_ntuple( psutil.virtual_memory())
```