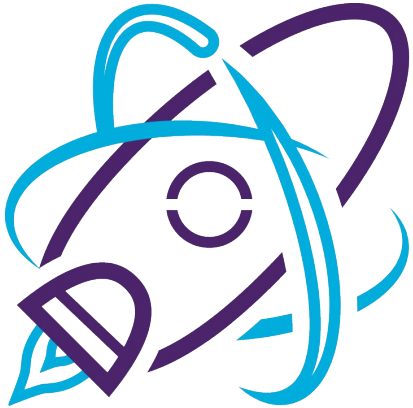


ReactIT

Formation & Recrutement
www.react-it.fr

Python <> Devops

ReactIT

Python et ses applications

Python est un langage utilisé dans de nombreux domaines dont le développement d'applications Web et la datascience.

Il dispose d'un arsenal d'outils et de bibliothèques dont la connaissance exhaustive nécessiterait un effort très important.

Il est possible d'exploiter la puissance de Python pour le devops en se concentrant sur un sous ensemble de fonctionnalités.

Python et ses applications

Utilisez ipython pour bénéficier d'un shell interactif:

```
!cat .envrc

!ls

!touch test.txt
```

iPython

```
In [4]: contents = !ls

In [5]: print(contents)
['myproject.txt']

In [6]: directory = !pwd

In [7]: print(directory)
['/Users/jakevdp/notebooks/tmp/myproject']
```

Script Python

Pour créer un script Python, commencez votre fichier par:

```
#!/usr/bin/env python
```

`/usr/bin/env` assurera que l'interpréteur utilisé sera le premier qui sera trouvé dans le `$PATH`

Pensez à donner les droits en exécution:

```
chmod +x script.py
```

Script Python

Il est également possible de mettre le numéro de version en dur:

```
#!/usr/bin/env python3.6

import sys

print(sys.version)
```

Expressions régulières

```
import re

str = 'purple alice-b@google.com monkey dishwasher'
match = re.search(r'[\w.-]+@[\w.-]+', str)

if match:

  print match.group()  ## 'alice-b@google.com'
```

Affichage des dates

```
import datetime

x = datetime.datetime.now()

print(x)
```

Avec ipython:

```
!date
```

Système de fichier

Dans une approche devops, on est constamment entrain de chercher, parser et modifier des fichiers qu'il s'agisse de fichiers de logs ou de configuration.

Toutes ces opérations sont possibles avec Python.

Système de fichier

```
file_path = “hello.txt”

text = open_file.read() # la lecture se fait en une ligne

len(text) # le contenu est directement accessible

text[4] # le résultat est une chaîne de caractères

open_file.close() # on ferme le fichier après utilisation
```

Système de fichier

```
text = '''export stage=PROD'''

outF = open('.envrc', 'w')

outF.write(text)

open_file.close()

!cat .envrc
```

Système de fichier

`pathlib` offre une surcouche qui simplifie encore l'utilisation des fichiers

```
from pathlib import Path

p = Path('.')

p.write_text('Text file contents')
```

Système de fichier

`pathlib` propose toutes les opérations Unix sur les fichiers

```
p = Path('setup.py')

p.exists()

p.stat().st_mode

p.chmod(0o444)

p.owner()
```

Charger un JSON

```
import json

with open('policy.json') as f:

  policy = json.load(f)
```

Afficher des objets Python

Le module pprint va automatiquement mettre en forme les objets Python:

```
from pprint import pprint

pprint(policy)
```

Dumper un objet en JSON

```
import json

json.dumps(['foo', {'bar': ('baz', None, 1.0, 2)}])
```

Manipuler le YAML

Le format YML est un format très répandue parmi les outils devops dont Ansible fait parti.

```
pip install PyYAML
```

Manipuler le YAML

```
import yaml

yml_content = yaml.safe_load(opened_file)

pprint(yml)

yaml.dump(yml_content, opened_file)
```

Format XML

XML est un format très répandue supporté par Python

```
import xml.etree.ElementTree as ET

tree = ET.parse('country_data.xml')

root = tree.getroot()
```

Format XML

Il est facile de parcourir un fichier XML:

```
for child in root:
  print(child.tag, child.attrib)
```

Format CSV

En ce qui concerne les données, qu'il s'agisse de logs ou d'exports de bases de données, le format CSV est le format le plus répandu:

```
import csv
with open('eggs.csv', newline='') as csvfile:
  spamreader = csv.reader(csvfile, delimiter=' ',
quotechar='|')
  for row in spamreader:
    print(', '.join(row))
```

Format CSV

La bibliothèque pandas est surtout utilisé dans le domaine de la datascience et offre des possibilités très intérssante de manipulation de la donnée (filtrage, transformation…)

```
import pandas as pd
df = pd.read_csv('data.csv')
type(df)
df.head()
df.describe
```

Format CSV

```
df['servers']

df['servers'][0]

df[df['size']>10]
```

Chiffrement

Il peut arriver d'avoir le besoin de chiffrer ou déchiffrer des fichiers.
hashlib permet des hash SHA1, SHA224, SHA384,SHA512, MD5.

```
import hashlib

secret = "mot magique"

bsecret = secret.encode() # encode en UTF-8
m = hashlib.md5()
m.update(bsecret)
```

Chiffrement

Les hashs ne permette pas de transmettre un message chiffré que l'on peut déchiffrer. Ils sont juste utilise pour éviter la transmission d'une information en clair (exemple: stockage de mot de passe).

Chiffrement

```
rom Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import binascii
keyPair = RSA.generate(3072)
pubKey = keyPair.publickey()
print(f"Public key: (n={hex(pubKey.n)}, e={hex(pubKey.e)})")
pubKeyPEM = pubKey.exportKey()
print(pubKeyPEM.decode('ascii'))
print(f"Private key: (n={hex(pubKey.n)}, d={hex(keyPair.d)})")
privKeyPEM = keyPair.exportKey()
print(privKeyPEM.decode('ascii'))
```

Module OS

Le module OS offre les fonctionnalités bas niveau disponible sur le système d'exploitation (Operating System). On retrouve notamment toutes les fonctionnalités Unix: chmod, mkdir, rename...

```
os.mkdir('logs')
os.rename('data.txt', 'archive_data.txt')
os.getcwd()
```

Fire

Fire est un module Python qui permet de faciliter le développement d'outils en ligne de commande (CLI).

```
pip install fire
```

Fire

```
import fire

def hello(name="World"):
  return "Hello %s!" % name

if __name__ == '__main__':
  fire.Fire(hello)
```

Fire

```
python hello.py  # Hello World!
python hello.py --name=David  # Hello David!
python hello.py --help  # Shows usage information.
```

Fire

```
import fire

class Calculator(object):
  """A simple calculator class."""

  def double(self, number):
    return 2 * number

if __name__ == '__main__':
  fire.Fire(Calculator)
```

Module sys

Le module sys permet une manipulation bas niveau de l'interpréteur Python.

```
print(sys.platform)
print(sys.version_info)
```

Module sys

Il permet de lancer des sous commandes:

```
cp = subprocess.run(['ls', '-l'], capture_output=True,
universal_newlines=True)
cp.stdout
cp = subprocess.run(['ls', '/toto'], capture_output=True,
universal_newlines=True)
cp.stderr
```

Module sys

On peut récupérer les paramètres:

```
sys.argv[0]
sys.argv[1]
sys.argv[2]
sys.argv[3]

$ ./script.py --a-flag some-value 13
```

Module sys

argparse offre une simplicité d'utilisation des paramètres

```
import argparse
parser = argparse.ArgumentParser()
parser.parse_args()
```

Module sys

```
$ ./script.py --help
usage: script.py [-h]

optional arguments:
  -h, --help  show this help message and exit
```

Module sys

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("echo")
args = parser.parse_args()
print(args.echo)
```

Module sys

```
$ ./script.py --help
usage: script.py [-h] echo

positional arguments:
  echo

optional arguments:
  -h, --help  show this help message and exit

$ ./script.py foo
foo
```

Automatisation et système de fichier

TP

Après avoir télécharger le jeu de données (https://www.data.gouv.fr/en/datasets/departements-de-france/), vous réaliser un outil CLI qui effectue les actions suivantes:

- Dans le dossier fourni (appelé dossier cible) en paramètre: création de l'arborescence de dossiers des régions (un par région). Si le dossier en paramètre n'existe pas vous devrez le créer
- Si le dossier cible contient déjà des fichiers, vous affichez un message d'erreur et vous interroger l'utilisateur sur son choix: abandonner ou vider le dossier
- Dans chaque dossier des régions, vous créerez un dossier par département

Nous allons maintenant enrichir le script précédent afin qu'il crée des fiches de monuments historiques pour chaque département. Le jeu de donnée est ici https://www.data.gouv.fr/fr/datasets/immeubles-proteges-au-titre-des-monuments-historiques-2/

Vous importerez uniquement les bâtiments qui font références aux rois, vous construirez donc une expression régulière qui matchera sur un nom de roi. On autorise des approximations à ce niveau. Révisez votre histoire ici : https://fr.geneawiki.com/index.php/Liste_des_rois_de_France

Seules les capétiens nous intéresse.

Chaque monument aura donc sa fiche (un fichier YML) dans le dossier correspondant à son département.

Le nom sera extrait de la première colonne :

```
Calvaire (cad. C 624)  : classement par arrêté du 18 décembre 1963
```

```yaml
nom: xxx
description: yyy
commune: zzz
localisation:
  latitude: lat
  longitude: long
```

Export

TP

Vous réaliser un script CLI qui permettra la recherche de bâtiment. Celui ci prendra en paramètre un mot clef de recherche (exemple: château) avec un paramètre optionnel de région.

Sans paramètres, l'outil affiche uniquement le nombre de résultats trouvés.

Avec le paramètre `--export`, l'outil permettra d'effectuer un export XML des fiches monuments (un seul fichier XML).

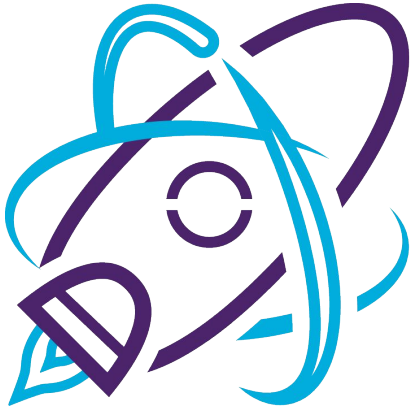
Bonus...

Complétez le script précédent pour ajouter le chiffrage du fichier et son envoi par email. Vous pourrez utiliser un service smtp gratuit comme mailtrap (https://mailtrap.io/).

Le fichier sera trop lourd, vous ne pourrez pas le transmettre en pièce jointe. Vous allez créer un bucket S3 (par exemple chez clever cloud, c'est gratuit pour des petits fichiers) et uploader votre fichier puis créer un lien public à l'aide de l'outil s3cmd ou autre solution de votre choix.

Vous serez peut être amener à compresser votre fichier XML (format zip ou gunzip), dans ce cas pensez à le faire **AVANT** le chiffrement pour plus d'efficacité.

Des questions ?



ReactIT

Formation & Recrutement
www.react-it.fr

Python <> Devops