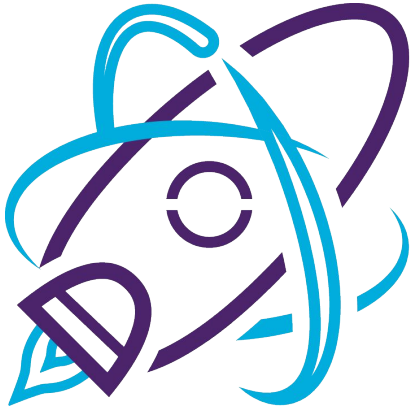


ReactIT

Formation & Recrutement
www.react-it.fr

Formation : Python

ReactIT

Votre formateur

- **Loïc Guillois**
- Développeur Web full stack depuis plus de 10 ans
- Expérience en environnement
  - SSII
  - Grand comptes Banque / Assurance / La Poste
  - Startup (Gamific.TV, La Fourchette.com, Akeneo)
  - Indépendant / freelance
- Enseignant / formateur depuis 5 ans
  - Développement
  - No SQL
  - Devops
  - Objets connectés

Faisons connaissance

Objectifs pédagogiques

- A l'issue de cette formation, vous serez en mesure de :
  - Connaître les possibilités du langage Python
  - Réaliser une application en s'appuyant sur des fonctionnalités avancées
  - Apprendre à écrire des programmes ou scripts grâce au langage Python

Prérequis du module:

  - Connaître un langage de programmation objet

Programme détaillé

Formation sur 3 jours:

- Découverte du langage
- La programmation orientée objet avec Python
- Les bibliothèques de Python
- Fonctionnalités avancées

Théorie et pratique.

Bibliothèques externes

Python en standard est fourni avec beaucoup de chose dans sa bibliothèque standard.

Cependant comme tous les langages, il est possible d'utiliser des **bibliothèques et frameworks externes**.

Bibliothèques externes

Pour chercher une bibliothèques tierce, vous pouvez utiliser le site suivant https://pypi.python.org/pypi ou utiliser directement votre terminal en utilisant la commande :

```
pip search bibliothèque
```

Bibliothèques externes

Exemple

```
pip install pdf
```

Les dépendances seront automatiquements téléchargées.

Bibliothèques externes

La bonne approche pour vos projets est d'utiliser un fichier `requirements.txt` avec toutes vos dépendances.

Vous pouvez exporter vos dépendances:

```
pip freeze > requirements.txt
```

Exemple:

```
appdirs==1.4.3
astroid==1.5.2
```

Tests logiciel

Pour assurer la meilleur qualité logiciel, il est nécessaire de mettre en place des tests. Il en existe de très nombreux type:

- Tests unitaires
- Tests d'intégration
- Tests fonctionnels / d'interface

Mais aussi: tests de performance, tests de sécurité, tests de conformité, tests d'accessibilité...

Tests logiciel

Comme il est très coûteux de tout tester, on parle de pyramide des tests. L'idée est d'implémenter les tests qui ont le meilleur retour sur investissement.

Le niveau de tests exigé dépend du métier. Un logiciel embarqué dans le système de freinage d'une voiture ou une centrale nucléaire n'a pas le même niveau d'exigence qu'un jeu vidéo.

Tests logiciel

La recommandation est la suivante pour une application Web:

- 70% de tests unitaires
- 20% de tests d'intégration
- 10% de tests d'interface.

Au besoin, des tests de sécurité, de performance ou d’accessibilité peuvent compléter ce panel.

Tests d'interface / fonctionnel

Permettent de **tester une application comme un vrai utilisateur**. Ces tests ont l'avantage d'être extrêmement fidèles à la réalité, mais ce sont les plus longs et les plus compliqués à écrire.

Exemple: simuler une connexion utilisateur.

Tests d'intégration

Permettent de tester plusieurs composants. Certains sont simulés (mock), d'autres sont réels. Ils peuvent s'exécuter sur l'émulateur ou sur un équipement réel. Ils sont plus longs à écrire que des tests unitaires, mais sont beaucoup plus fidèles à la réalité.

Tests unitaires

Petits, rapides et très ciblés. Ils permettent de détecter les erreurs au plus tôt.

La majorité des composants sont simulés, ce qui implique qu'ils ne représentent pas forcément la réalité. Ils ont l'avantage de pouvoir s'exécuter sur l'ordinateur du développeur, ce qui permet de les passer très rapidement.

Tests unitaires

Pour pouvoir mettre en place des tests unitaires, il faut que le code soit correctement organisés avec des fonctions et des objets testable.

Un test unitaire va vérifier la réponse obtenu pour des paramètres donnés.

Exemple: une fonction qui calcul un montant TTC. Le test vérifiera le résultat obtenu pour une réponse connue.

Tests unitaires

l y a eu historiquement **plusieurs frameworks** de développement de tests unitaires sous python:

- unittest
- nose
- doctest
- pytest

unittest

unittest s'inspire des framework de tests unitaires des autres langages, comme JUnit en Java.

unittest est intégré de base à Python.

doctest

doctest est également intégré de base à Python.

Son avantage est d'intégrer les tests unitaires dans la documentation. En revanche, cela peut alourdir celle ci.

pytest est une bibliothèque externe très populaire.

Facile à mettre en oeuvre, il est plus complet qu'unittest, notamment au niveau de l'introspection. Il offre un meilleur niveau d'outillage (recherche des tests, configuration...)

unittest: Structure de base d'un test unitaire

Basic_Test.py

```
import unittest
class Testing(unittest.TestCase):
    def test_string(self):
        a = 'some'
        b = 'some'
        self.assertEqual(a, b)

if __name__ == '__main__':
    unittest.main()
```

unittest: exécution

```
python -m unittest Basic_Test.Testing
```

unittest: les assertions

```
assertEqual
assertNotEqual
assertTrue
assertFalse
assertIs
assertIsNot
assertIsNone
...
```

Manipuler les fichiers en Python

Lecture de fichiers texte:

```
mon_fichier = open("fichier.txt", "r")
contenu = mon_fichier.read()
print(contenu)
mon_fichier.close()
```

Les erreurs (droits d'accès, espace disque, fichier introuvable…) lèveront des exceptions. Vous devrez les gérer avec un bloc `try / except`

Manipuler les fichiers en Python

Ecrire un fichier texte:

```
mon_fichier = open("fichier.txt", "w")
contenu = mon_fichier.write("hello world")
mon_fichier.close()
```

Il est important de penser à fermer le fichier.

Manipuler les fichiers en Python

Il est possible de parcourir un fichier ligne par ligne:

```
f=open("myfile.txt","r")
while True:
        try:
            line=next(f)
            print (line)
        except StopIteration:
            break
f.close()
```

Manipuler les fichiers en Python

L'écriture de fichier binaire passe par un buffer:

```
f=open("binfile.bin","wb")
num=[5, 10, 15, 20, 25]
arr=bytearray(num)
f.write(arr)
f.close()
```

Manipuler les fichiers en Python

La lecture de fichier binaire est simple en Python:

```
f=open("binfile.bin","rb")
num=list(f.read())
print (num)
f.close()
```

Manipuler les fichiers en Python

Le module `os` contient beaucoup de fonctions intéressantes pour créer et supprimer des fichiers et des répertoires. Quelques exemples:

```
import os
path = os.getcwd()
list = os.listdir('c:/python24')
os.mkdir('c:/test')
```

Interface graphique

Lorsque que l'on développement un logiciel (client lourd). Il y a en (simplifiant) deux approches:

- Utiliser le système de fenêtre
- Utiliser une bibliothèque de rendu 3D

Ces deux approches sont complètement différentes.

Interface graphique

En ce qui nous concerne, nous utiliserons les bibliothèques graphiques s'appuyant sur le système de fenêtrage:

- Tkinter
- wxPython
- PyGTK
- PyQt

Toutes ces bibliothèques s'appuient sur des SDK qui existent depuis des décennies en C / C++.

Interface graphique: Tkinter

Tkinter (Tk interface) est un module intégré à la bibliothèque standard de Python, bien qu'il ne soit pas maintenu directement par les développeurs de Python. Il offre un moyen de créer des interfaces graphiques via Python.

Tkinter est disponible sur Windows et la plupart des systèmes Unix.

Il permet donc de créer des logiciels multi-plateformes.

Tkinter

```
from tkinter import *

fenetre = Tk()

# on ajoute un label
champ_label = Label(fenetre, text="Hello World !")

# on affiche le label
champ_label.pack()

fenetre.mainloop()
```

Tkinter

Les événements:

```
bouton_quitter = Button(fenetre, text="Quitter", command=self.cliquer)

def cliquer(self):
    self.nb_clic += 1
    self.message["text"] = "Vous avez cliqué {} fois.".format(self.nb_clic)
```

Bases de données

Principe: pour se connecter à une base de donnée, on utilise un connecteur. Ce connecteur est spécifique à la base de donnée (MySQL, MongoDB, Redis...)

Bases de données: MySQL

Connexion et création de base

```
import mysql.connector as mysql
db = mysql.connect(
    host = "localhost",
    user = "root",
    passwd = "dbms"
)
cursor = db.cursor()
cursor.execute("CREATE DATABASE datacamp")
cursor.execute("SHOW DATABASES")
```

Bases de données: MySQL

Connexion et création de table

```
import mysql.connector as mysql
db = mysql.connect(
    host = "localhost",
    user = "root",
    passwd = "dbms",
    database = "datacamp"
)
cursor = db.cursor()
cursor.execute("CREATE TABLE users (name VARCHAR(255), user_name VARCHAR(255))")
cursor.execute("SHOW TABLES")
```

Bases de données: MySQL

Insérer des données

```
query = "INSERT INTO users (name, user_name) VALUES (%s, %s)"
values = ("Loïc GUILLOIS", "lgu")
cursor.execute(query, values)
db.commit()
print(cursor.rowcount, "record inserted")
```

Bases de données: MySQL

Sélectionner des données

```
query = "SELECT * FROM users"
cursor.execute(query)
records = cursor.fetchall()
for record in records:
    print(record)
```

Programmation réseau

Créer un serveur Web simple

```
import http.server
import socketserver

PORT = 8080
Handler = http.server.SimpleHTTPRequestHandler

with socketserver.TCPServer(("", PORT), Handler) as httpd:
    print("serving at port", PORT)
    httpd.serve_forever()
```

Programmation réseau

Envoyer un email

```
import smtplib
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText

msg = MIMEMultipart()
msg['From'] = 'XXX@gmail.com'
msg['To'] = 'YYY@gmail.com'
msg['Subject'] = 'Le sujet de mon mail'
message = 'Bonjour !'
msg.attach(MIMEText(message))
```

Programmation réseau

Envoyer un email (suite)

```
mailserver = smtplib.SMTP('smtp.gmail.com', 587)
mailserver.ehlo()
mailserver.starttls()
mailserver.ehlo()
mailserver.login('XXX@gmail.com', 'PASSWORD')
mailserver.sendmail('XXX@gmail.com', 'XXX@gmail.com', msg.as_string())
mailserver.quit()
```

Programmation réseau

Créer un Bot IRC

```
import socket, string, time, thread
SERVER = ''
PORT = 6667
NICKNAME = 'loic'
CHANNEL = 'gamerz'

def main():
    global IRC
    IRC = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    IRC.connect((SERVER, PORT))
    thread.start_new_thread(Listener(),("Thread No:1",2))
```

Programmation réseau

Créer un Bot IRC (suite)

```
def send_data(command):
     IRC.send(command + '\n')

def Listener():
    send_data('USER Blah')
    send_data('NICK Blah')
    while (1):
      buffer = IRC.recv(1024)
      msg = string.split(buffer)
      if msg[0] == "PING":
           print 'Pinged!'
           IRC.send("PONG %s" % msg[1] + '\n')

main()
```

Outils d’analyse statique

Il y a plusieurs outils d'analyse statique en Python, qui permette de s’assurer de la qualité du code écrit:

- pylint
- pychecker
- pyflakes
- pep8
- flake8

Une fois intégré dans vos outils de développement, Ils apportent des gains de productivité.

Débuggeur

Le module `pdb` définit un débogueur de code source interactif pour les programmes Python.

Il supporte le paramétrage (conditionnel) de **points d'arrêt** et l'exécution du code source ligne par ligne, **l'inspection** des frames de la pile, la liste du code source, et l'évaluation arbitraire de code Python dans le contexte de n'importe quelle frame de la pile.

Des questions ?

Restons en contact

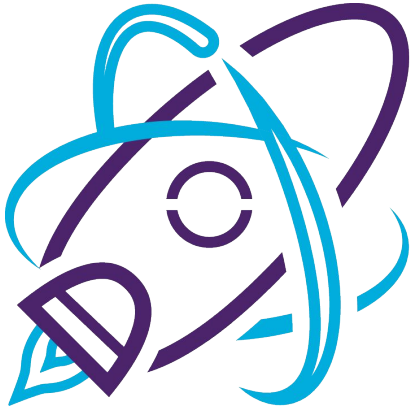
Votre contact:

- React IT
  - Loïc Guillois, président
  - hello@react-it.fr

www.react-it.fr

JUNIOR JOBS

Découvrez aussi http://junior.jobs



ReactIT

Formation & Recrutement
www.react-it.fr

Formation : Python