Aide-mémoires Québec-Python Documentation

Version 0.0.1

Bernard Chhun

21 August 2016

Vous y retrouverez tous les aide-mémoires que nous rédigeons pour nos ateliers et nos présentations.

CHAPITRE 1

Table des matières

1.1 Installation de Python

1.1.1 Sous Windows et Mac OS

La problématique principale sous Windows et Mac OS est la nécessité de compiler certains modules sur votre poste.

En effet, si vous n'avez pas installer explicitement un compilateur, vos tentatives d'installation de module externe échoueront.

Il existe heureusement des distributions qui nous rendent la vie plus facile.

En voici 3 :

- ActivePython
- PythonXY
- Anaconda

Les instructions suivantes s'appliquent pour l'utilisation d'Anaconda, mais sentez-vous bien à l'aise d'utiliser *ActivePython* ou *PythonXY*.

Installer *Anaconda*

Téléchargez la distribution anaconda et installez le sur votre poste

Tester votre installation

Tester le tout en tapant les commandes suivantes dans votre terminal.

```
$ conda info
$ conda install fabric
$ fab --help
```

La première commande nous affiche des détails sur *Anaconda* et *conda*.

La deuxième commande installe le module *Fabric* qui nécessite une compilation du module *PyCrypto*.

La troisème commande affiche l'aide de la commande *fab* qui provient du module *Fabric*.

Si tout se déroule bien pour ces étapes, vous êtes en bateau !

1.2 Environnements virtuels

Installer des paquets en Python à l'aide de pip ou easy_install demande par défaut les accès administrateurs. Leur installation terminent également dans votre installation système, ce qui peut causer des ennuis.

Pour remédier à ce problème, Vous pouvez créer un environnement virtuel :

```
# Installer virtualenv si ce n'est pas déjà fait
sudo pip install virtualenv

# Créer un virtualenv
virtualenv monvenv

# Activer l'environnement virtuel
        ./monvenv/bin/activate
```

Si vous publiez votre code par github, bitbucket ou similaire, il est recommandé de fournir un fichier *requirements.txt* qui permet de recréer l'environnement virtuel, soit les dépendances de votre projet :

```
pip freeze > ./requirements.py
```

Pour recréer votre environnement :

```
pip install -r ./requirements.txt
```

1.3 Python

1.3.1 Création de variables

```
# Peut être True ou False


             1000

                       "bacon ipsum"

                     1  2  3  4

                        1  2  3  4  # Aussi nommée tuple


    "une_cle"  "une_valeur"
    "une_2e_cle"  "une_2e_valeur"


# Ne contiendra qu'une seule fois pomme

    "pomme"
    "pomme"
    "orange"
    "poire"
    "banane"

```

1.3.2 Structures de contrôle

Si logique

```
if
    print "Oui"
elif
    print "Non"
else
    print "Aucune condition dans le if ou le elif n'est Vraie"
```

Boucle *For*

```
             "rouge"   "bleu"   "vert"

for
    print


for
    if
        # Arrête l'exécution et sort de la boucle
        break
else
    print "Affiche ceci s'il n'y a pas eu de break"
```

1.3.3 Opérations

Arithmétique

```
    3       4

# Addition

# Soustraction

# Multiplication

# Division

# Division entière

# Modulo (reste de la division)

# Puissance

```

Comparaison

```
    3       4       5       1   2   3   4   5

# Égalité

# Inégalité

# Chaînage de comparaisons

# Opérateur "dans"

# Opérateur "est"


# Tout élément non nul ou non vide est évalué à vrai
if
    print "Sera affiché"
```

1.3.4 Les fonctions

```
def bien_le_bonjour
    """
    Cette fonction souhaite une bonne journée au prénom
    en paramètre
    """
    print "Bonjour {}"

                "Bernard"
```

1.3.5 Programmation orienté objet

Les classes

```
class
              ""
             ""
                 0
                 0

    def __init__
        """
        Un constructeur
        """



    def roule
        """
        Roule ma boule !
        """

```

L'héritage

```
# Animal est un Objet
class
    def __init__
        pass

# Cheval est un Animal
class
    """
    Une classe qui hérite d'une autre contient tous les attributs et
    méthodes de son parent. Ici, Cheval hérite d'Animal.
    """
    def galoper
        """
        Définition d'une fonction propre à Cheval.
        """
        print "Je galope!"

class
    def __init__
        print "Init de Humain"

    def parler
        print "Je parle!"

# Centaure est un Cheval et un Humain
class
    def __init__
        # super() utilise le MRO pour trouver l'objet parent
        # Dans ce cas-ci, tente d'appeler le __init__ de Cheval, sinon repli
        # sur celui de Humain. Puisque Cheval n'a pas de __init__ qui lui
        # est propre, celui de Humain() est appelé.


        # Appel explicite utilisant le polymorphisme


        # Cet appel exécutera le __init__ de Animal


        # Appel des méthodes héritées


```

1.3.6 Les modules

```
# Importation absolue
from        import
import

print                  1   2   3

# Importation relative
from                                    import
from              import
```

1.3.7 Les exceptions

```
# Les blocs else et finally sont optionnels
try
    raise           "Mon Exception"
except                          as
    print "Erreur survenue: "
else
    print "Si aucune erreur n'est survenue, afficher ceci"
finally
    print "Toujours affiché"
```

1.3.8 Compréhensions

Liste

```
     1  2  3  4

             2 for
           for        if         2
```

Ensemble

```
     1  2  3  4

             2 for
           for        if         2
```

Dictionnaire

```
                2 for       2  4  6
```