
Aide-mémoires Québec-Python Documentation

Version 0.0.1

Bernard Chhun

27 September 2017

1	Table des matières	3
1.1	Installation de Python	3
1.2	Environnements virtuels	4
1.3	Python	4

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

Table des matières

Installation de Python

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 installé 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.

```
1 $ conda info
2 $ conda install fabric
3 $ 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 troisième 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 !

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 :

```
1 # Installer virtualenv si ce n'est pas déjà fait
2 sudo pip install virtualenv
3
4 # Créer un virtualenv
5 virtualenv monvenv
6
7 # Activer l'environnement virtuel
8     ./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 :

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

Pour recréer votre environnement :

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

Python

Création de variables

```
1 # Peut être True ou False
2
3
4     1000
5
6         "bacon ipsum"
7
8     1 2 3 4
9
10        1 2 3 4 # Aussi nommée tuple
11
12
13     "une_cle" "une_valeur"
14     "une_2e_cle" "une_2e_valeur"
15
16
17 # Ne contiendra qu'une seule fois pomme
18
19     "pomme"
20     "pomme"
21     "orange"
22     "poire"
23     "banane"
24
```

Structures de contrôle

Si logique

```
1 if
2     print "Oui"
3 elif
4     print "Non"
5 else
6     print "Aucune condition dans le if ou le elif n'est Vraie"
```

Boucle For

```
1         "rouge" "bleu" "vert"
2
3 for
4     print
5
6
7 for
8     if
9         # Arrête l'exécution et sort de la boucle
10        break
11 else
12     print "Affiche ceci s'il n'y a pas eu de break"
```

Opérations

Arithmétique

```
1     3     4
2
3 # Addition
4
5 # Soustraction
6
7 # Multiplication
8
9 # Division
10
11 # Division entière
12
13 # Modulo (reste de la division)
14
15 # Puissance
16
```

Comparaison

```
1      3      4      5      1 2 3 4 5
2
3 # Égalité
4
5 # Inégalité
6
7 # Chainage de comparaisons
8
9 # Opérateur "dans"
10
11 # Opérateur "est"
12
13
14 # Tout élément non nul ou non vide est évalué à vrai
15 if
16     print "Sera affiché"
```

Les fonctions

```
1 def bien_le_bonjour
2     """
3     Cette fonction souhaite une bonne journée au prénom
4     en paramètre
5     """
6     print "Bonjour {}"
7
8         "Bernard"
```

Programmation orienté objet

Les classes

```
1 class
2     ""
3     ""
4         0
5         0
6
7     def __init__
8         """
9         Un constructeur
10        """
11
12
13
14     def roule
15         """
16         Roule ma boule !
17         """
18
```

L'héritage

```

1  # Animal est un Objet
2  class
3      def __init__
4          pass
5
6  # Cheval est un Animal
7  class
8      """
9      Une classe qui hérite d'une autre contient tous les attributs et
10     méthodes de son parent. Ici, Cheval hérite d'Animal.
11     """
12     def galoper
13         """
14         Définition d'une fonction propre à Cheval.
15         """
16         print "Je galope!"
17
18 class
19     def __init__
20         print "Init de Humain"
21
22     def parler
23         print "Je parle!"
24
25 # Centaure est un Cheval et un Humain
26 class
27     def __init__
28         # super() utilise le MRO pour trouver l'objet parent
29         # Dans ce cas-ci, tente d'appeler le __init__ de Cheval, sinon repli
30         # sur celui de Humain. Puisque Cheval n'a pas de __init__ qui lui
31         # est propre, celui de Humain() est appelé.
32
33
34         # Appel explicite utilisant le polymorphisme
35
36
37         # Cet appel exécutera le __init__ de Animal
38
39
40         # Appel des méthodes héritées
41
42

```

Les modules

```

1  # Importation absolue
2  from          import
3  import
4
5  print          1 2 3
6
7  # Importation relative
8  from          import
9  from          import

```

Les exceptions

```
1 # Les blocs else et finally sont optionnels
2 try
3     raise          "Mon Exception"
4 except           as
5     print "Erreur survenue: "
6 else
7     print "Si aucune erreur n'est survenue, afficher ceci"
8 finally
9     print "Toujours affiché"
```

Compréhensions

Liste

```
1     1 2 3 4
2
3         2 for
4     for     if     2
```

Ensemble

```
1     1 2 3 4
2
3         2 for
4     for     if     2
```

Dictionnaire

```
1         2 for     2 4 6
```