Exploration du module Scene – 1ère partie

Merci à Diamond Editions pour son aimable autorisation pour la mise en ligne de cet article, initialement publié dans Linux Magazine N°82

Olivier Saraja - olivier.saraja@linuxgraphic.org

Dans cet article, nous allons approfondir notre connaissance du module Scene. Nous en avons déjà fait usage dans le passé, notamment pour lier des objets à la scène courante, mais ce module nous réserve de nombreux autres usages, comme nous allons le découvrir.

Jusqu'à présent, nous avons fait appel au module **`Scene`** à chaque fois que nous avons souhaité insérer un objet dans notre scène au-travers d'un script python. La démarche était alors la suivante, par exemple pour la création d'une nouvelle caméra.

(1) création d'un nouveau bloc de données:

```
cam = Camera.New('persp')
```

(2) récupération de la scène courante:

```
scn = Scene.GetCurrent()
```

(3) création d'un nouvel objet:

```
camera = Object.New('Camera')
```

(4) établissement de la liaison entre le bloc de données de caméra et l'objet caméra:

```
camera.link(cam)
```

(5) établissement d'une liaison entre l'objet caméra et la scène courante:

```
scn.link(camera)
```

> **ATTENTION**
>
> Dans cet exemple, nous avons respecté une certaine logique hiérarchique dans la structure des données, et en particulier, nous noterons que la liaison de l'objet à la scène est établie en dernier lieu. En effet, en supposant que nous aurions d'abord établi une liaison entre un objet et la scène, nous aurions immédiatement généré un bloc de données vide pour garnir cet objet. Au moment de l'établissement de la liaison entre bloc de données et objet, le bloc spécifié va écraser le bloc vide, mais celui-ci restera quelque part dans la mémoire de Blender, l'encombrant légèrement. Si cela n'a certainement aucune incidence lors de la création et de l'usage de scripts simples, cela peut avoir une forte incidence sur la mémoire utilisée par Blender lors de la génération automatique d'un grand nombre d'objets (un script de génération automatique de gazon, par exemple?). Respecter cet ordre de liaison garantit donc une réservation optimale de la mémoire.

La procédure est la même pour l'insertion d'un objet de type lampe, d'un maillage ou de n'importe quel autre type d'objet: elle fait systématiquement appel au module **`Scene`**, qui en devient pratiquement incontournable. Mais ce module couvre d'autres applications, très différentes, de la duplication de scène à la gestion des calques actifs, en passant par la possibilité de piloter le rendu de vos scènes, ou les réglages du moteur de radiosité. Mais nous découvrirons tout cela très progressivement.

1. Le Module Scene

Comme tout autre module, le module **Scene** doit être importé, généralement dans la deuxième ligne de votre script:

```
01:    import Blender
02:    from Blender import Scene
03:    ...
```

Il s'agit du premier module dont nous approfondissons l'étude qui propose également des sous-modules: **Radio** (pour la simulation d'un illumination par solution de radiosité) et **Render** (pour contrôler les paramètres de rendu d'une scène). Optionnellement, nous importerons ces deux sous-modules grâce à une ligne de type:

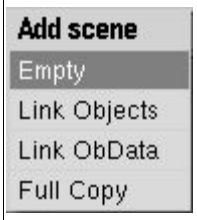
```
03:    from Blender.Scene import Radio, Render
04:    ...
```

Le module **Scene** propose quatre fonctions essentielles, permettant de récupérer les données relatives à une scène existante, à récupérer la scène courante, à créer une nouvelle scène, et enfin à supprimer une scène.

NOTES

Un fichier Blender peut contenir plusieurs scènes; chaque scène peut être strictement indépendantes les unes des autres, ou bien partager certains éléments comme des blocs de données, ou des objets.

Dans Blender, la gestion des scènes est simple et très limitée. Dans la barre de menu principale se trouve un bouton menu **SCE** portant le nom de la scène courante (par défaut: **Scene**). Sur sa gauche se trouve un petit ascenseur permettant de naviguer entre les différentes scènes existantes, ou d'en créer de nouvelles grâce à l'option **ADD NEW**. Dans ce dernier cas, Blender propose différents types de scène: **Empty** crée simplement une nouvelle scène, totalement vierge; les autres options seront détaillées lors de la description de la méthode **copy()** en 2.1.



Les options d'ajout d'une scène dans Blender

1.1 La fonction Get():

Cette fonction permet de récupérer une scène particulière en utilisant son nom, ou bien de récupérer la liste de toutes les scènes. Par exemple:

```
scn = Scene.Get()
print scn
```

permet d'imprimer dans la console la liste de toutes les scènes disponibles:

```
[[Scene "Scene"], [Scene "Scene.001"]]
```

et d'en récupérer ensuite une en particulier. Si vous connaissez le nom de la scène qui vous intéresse, vous pouvez également récupérer celle-ci simplement en la nommant:

```
scn = Scene.Get('Scene.001')
```

1.2 La foncion GetCurrent():

Très similaire à la fonction précédente, celle-ci permet de récupérer la scène couramment active. Par exemple:

```
scn = Scene.GetCurrent()
```

1.3 La fonction New([nom]):

Cette fonction permet de créer une nouvelle scène et de lui donner un nom. Par exemple:

```
scn = Scene.New('NouvelleScene')
```

créera une nouvelle scène sobrement nommée... **`NouvelleScene`**!

1.4 La fonction Unlink([nom]):

Celle-ci agit à l'opposé de la précédente, car au lieu de créer une scène nommée, elle l'efface! Attention à sa syntaxe particulière, car elle n'admet pas comme argument un nom de scène, il vous faudra donc passer par une variable ou une fonction pour spécifier la scène à effacer. Par exemple:

```
scn_del = Scene.Get('NouvelleScene')
scn = Scene.Unlink(scn_del)
```

Une formulation plus économe et strictement équivalente serait:

```
scn = Scene.Unlink(Scene.Get('NouvelleScene'))
```

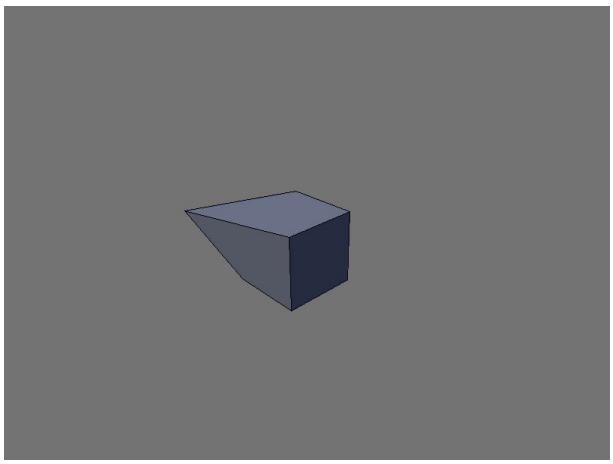
2. Les méthodes propres au module Scene

2.1 La méthode copy():

Cette méthode permet de produire une copie d'une scène. L'argument passé à la méthode **`copy()`** permet de déterminer la façon dont les objets « enfants » de la scène sont dupliqués. L'argument **`0`** lie les objets aux objets originaux; l'argument **`1`** lie les blocs de données aux blocs de données des objets originaux; enfin, l'argument **`2`** crée une copie intégrale et indépendante de la scène. Par exemple, supposons et exécutons le script suivant:

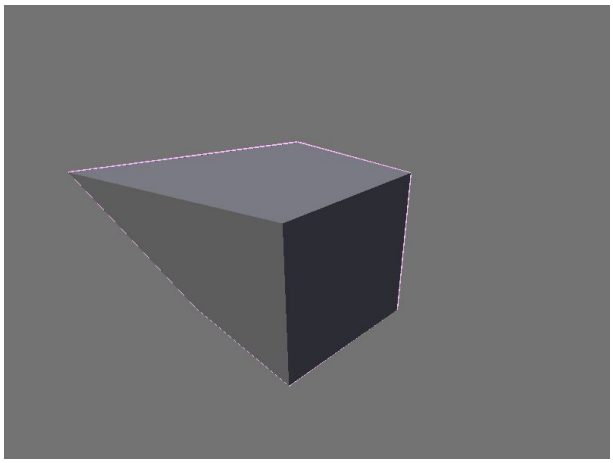
```
01:   import Blender
02:   from Blender import Scene
03:   scn = Scene.GetCurrent()
04:   print scn
05:   scn0 = scn.copy(0)
06:   scn0.setName('LinkObjects')
07:   scn1 = scn.copy(1)
08:   scn1.setName('LinkObjectData')
09:   scn2 = scn.copy(2)
10:   scn2.setName('FullCopy')
11:   scn = Scn.Get()
12:   print scn
```

Nous avons maintenant quatre scènes: **`Scene`**, **`LinkObjects`**, **`LinkObjectData`** et **`FullCopy`**. Pour comprendre la différence entre ces différents modes de copie, faites l'expérience suivante: dans Blender, assurez-vous d'être dans la scène d'origine nommée **`Scene`**. Editez ensuite le cube (touche ***[TAB]***) et déplacez un ou plusieurs de ses points de contrôle (par exemple grâce à la touche ***[G]***).



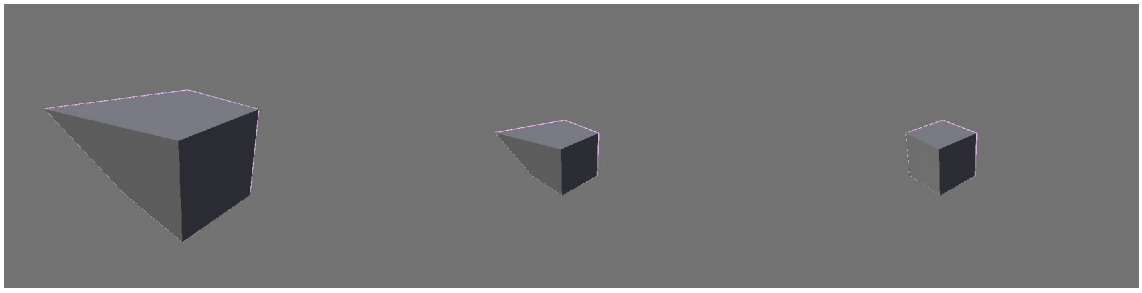
En mode édition, nous modifions le maillage du cube de la scène d'origine

Une fois ceci fait, sortez du mode édition, et redimensionnez (touche ***[S]***) le cube d'un facteur 2.



Hors du mode édition, nous redimensionnons le « cube » ainsi retouché

Basculez successivement de scène en scène. Vous constaterez que dans la scène **`LinkObject`**, le cube a suivi à la fois le changement de forme et le changement de dimension que celui de la scène d'origine: l'objet de la nouvelle scène est directement lié à la celui de la scène d'origine. Dans la cène **`LinkObjectData`**, seul le changement de forme a suivi celui de la scène d'origine: le cube de la nouvelle scène et celui de la scène d'origine partagent le même bloc de données, et donc le même maillage, mais les transformations géométriques (déplacement, rotation, dimensions) appliquées à l'objet (hors du mode édition, donc!) ne sont pas transmises. Enfin, dans la scène **`FullCopy`**, le cube n'a suivi ni la modification du maillage, ni les transformations géométriques; cette scène est strictement indépendante de la scène d'origine, aucune modification ou transformation n'a été transmise de scène à scène.



Respectivement, la scène LinkObjects (strictement identique à la scène Scene), la scène LinkObjectData (le maillage est identique, mais pas les dimensions de l'objet) et enfin la scène FullCopy (qui ne partage ni bloc de données, ni objet, avec scène d'origine)

2.2 La méthode makeCurrent():

Cette méthode permet de déterminer la scène courante, par exemple dans le cas où, par l'usage d'autres méthodes ou fonctions, vous créez ou récupérez de multiple scène. Par exemple, nous pourrions continuer le petit script précédent par les lignes:

```
13:    scn = Scene.Get('LinkObject')
14:    scn.makeCurrent()
```

pour faire de la scène **`LinkObject`** la scène couramment active de Blender.

2.3 La méthode link():

Cette méthode permet d'insérer dans une scène un objet en provenance d'une autre scène; l'objet importé est lié à l'objet d'origine: les modifications de maillage comme les transformations géométriques seront répercutées d'une scène à l'autre. Après avoir créé de multiples scènes comme dans le paragraphe 2.1 au sujet de la méthode **`copy()`**, dans Blender, retournons dans la scène d'origine **`Scene`** et ajoutons une **`UVSphere`**. Créons ensuite un nouveau script (dans la fenêtre d'édition de texte: ***File > New***) et saisissons le code suivant:

```
01:    import Blender
02:    from Blender import Scene, Object
03:    scn = Scene.Get('Scene')
04:    scn.makeCurrent()
05:    ob = Object.Get('Sphere')
06:    scn = Scene.Get('FullCopy')
07:    scn.makeCurrent()
08:    scn.link(ob)
```

Lorsque nous l'exécuterons, le script basculera automatiquement dans la scène d'origine **`Scene`** si nous n'y étions pas déjà, récupérera l'objet nommé **`Sphere`** sous le nom de variable **`ob`**, rebasculera dans la scène **`FullCopy`**, et grâce à la méthode **`link()`**, liera l'objet **`ob`** à la scène.

2.4 La méthode unlink():

A l'opposé de la méthode précédente, celle-ci supprime un objet désigné de la scène courante. Pour poursuivre sur notre exemple précédent, ajoutons les lignes suivantes au script précédent:

```
09:    scn = Scene.Get('Scene')
10:    scn.makeCurrent()
11:    ob = Object.Get('Cube')
12:    scn.unlink(ob)
```

Lorsque nous l'exécutons, ce script agit à l'identique du précédent, à l'exception qu'il retourne dans la scène d'origine **`Scene`**, qu'il sélectionne l'objet nommé **`Cube`**, et le supprime de la scène d'origine. En conclusion, nous n'avons plus que la sphère dans la scène d'origine, tandis que dans la scène **`FullCopy`**, nous avons le cube et la sphère, récupérée de la première scène.

2.5 La méthode getActiveObject():

Cette méthode permet de récupérer l'objet actif de la scène. Il est important de comprendre ici la différence entre objet *actif* et objet *sélectionné*. Même quand aucun objet n'est sélectionné, il y a toujours un objet actif; généralement, il s'agit du dernier objet sélectionné, ou du dernier maillage ou objet pour lequel vous seriez entré en mode édition.

REMARQUE

La méthode **Object.GetSelected()[0]** permet de récupérer l'objet actif si celui-ci est sélectionné: l'objet actif apparaît en tête de liste. Il y a donc une différence fonctionnelle fondamentale entre les méthodes **getActiveObject()** du module **Scene** et **getSelected()** du module **Object**.

Par exemple:

```
01:    import Blender
02:    from Blender import Scene
03:    scn = Scene.getCurrent()
04:    active = scn.getActiveObject()
```

vous montre comment récupérer simplement le dernier objet actif et en stocker la référence dans la variable nommée **active**. Il s'agira bien sûr de l'objet actuellement sélectionné, s'il existe une telle sélection, ou du dernier objet sélectionné (ou édité) si aucune sélection n'est disponible.

2.6 La méthode getChildren()

Grâce à cette méthode, il est possible de récupérer la liste de tous les objets d'une scène particulière.

REMARQUE

Le module **Object** propose une méthode très similaire, **Object.Get()**, qui récupère toutefois la liste de tous les objets existants de Blender, toutes scènes confondues. La méthode **Scene.getChildren()** se révèle plus pratique dans la mesure où elle ne retourne que la liste des objets de la scène courante, ce qui est pratique dans le cadre de scripts d'export, par exemple.

Essayons le script suivant avec la scène par défaut de Blender:

```
01:    import Blender
02:    from Blender import Scene
03:    scn = Scene.getCurrent()
04:    children = scn.getChildren()
05:    print children
```

on obtient dans la console la liste des objets de la scène:

```
[[Object "Cube"], [Object "Lamp"], [Object "Camera"]]
```

2.7 La méthode getCurrentCamera():

Cette méthode permet de récupérer la caméra active de la scène. A noter que dans Blender, n'importe quel objet peut être une caméra, et que cette méthode peut donc récupérer un objet en lieu et place d'une vraie caméra! Par exemple, sélectionnez la lampe de la scène par défaut de Blender, et appuyez sur la combinaison de touches **[Ctrl]**+**[0]** du pavé numérique. Lancez maintenant le petit script suivant:

```
01:    import Blender
02:    from Blender import Scene
03:    scn = Scene.getCurrent()
04:    cam = scn.getCurrentCamera()
```

le résultat, dans la console, sera alors le suivant:

```
[Object "Lamp"]
```

Vous pouvez également utiliser la méthode **setCurrentCamera()** afin de définir, au-travers de Python, la caméra active. Par exemple, le script:

```
01:    import Blender
02:    from Blender import Scene, Object
03:    scn = Scene.getCurrent()
04:    cam = Object.Get('Camera')
05:    scn.setCurrentCamera(cam)
```

récupère l'objet nommé **Camera** dans l'objet **cam**, puis la méthode **setCurrentCamera()** fait de cet objet **cam** la caméra active de la scène **scn**.

2.8 La méthode getName():

Sans surprise, cette méthode permet de récupérer le nom de la scène courante. Par exemple:

```
04:    print scn.getName()
```

renverra **Scene** dans la console pour la scène par défaut de Blender. Bien sûr, vous pouvez également utiliser la méthode **setName()** pour changer ce nom si cela ne vous convient pas:

```
05:    scn.setName('SceneAlternative')
```

2.9 La méthode getLayers():

Supposons que les calques actifs soient les suivants, dans votre scène Blender:

Les calques 1, 2 et 5 sont activés

Nous pouvons utiliser la méthode **getLayers()** pour afficher les numéros de calque actifs dans la console grâce au petit script suivant:

```
01:    import Blender
02:    from Blender import Scene
03:    scn = Scene.getCurrent()
04:    liste_calques = scn.getLayers()
05:    print liste_calques
```

Ce qui se traduit par l'affichage effectif suivant, dans la console:

```
[1, 2, 5]
```

Bien sûr, nous pouvons également spécifier grâce à la méthode **setLayers()** les numéros de calques à activer. Par exemple:

```
04:    calques_actifs = scn.setLayers([1, 5, 9])
```

activera les calques suivants de Blender:

Désormais, ce sont les calques 1, 5 et 9 qui sont actifs

ASTUCE

La méthode **setLayers()** sert seulement à déterminer les calques actifs de la scène courante, mais aucunement à déterminer sur quel calque se trouve un objet particulier. Cette dernière action est du ressort du module **Object**, et de la variable **layers**. Par exemple, reprenons la scène par défaut de Blender. Elle contient une caméra, un cube et une lampe dans le calque numéro un. Exécutons maintenant le script suivant:

```
01:    import Blender
02:    from Blender import Object
03:    ob_list = Object.Get()
04:    item = 0
05:    for obj in ob_list:
06:           obj = ob_list[item]
07:           obj.layers = [5]
08:           item = item + 1
```

Il récupère les noms de chaque objet (toutes scènes et tous calques confondus) et les stocke dans

une liste nommé **ob_list**. On récupère ensuite chaque objet de la liste et on le stocke dans une variable temporaire **obj**. On définit alors le cinquième calque comme étant le calque de destination de l'objet en question grâce à la ligne **obj.layers = [5]**.

2.10 Les méthodes getRadiosityContext() et getRenderingContext():

Ces deux méthodes permettent de récupérer les réglages propres aux sous-modules **Radio** et **Render**. Par exemple:

```
Rndr = scn.getRenderingContext()
Rndr.render()
```

permet d'ordonner à Blender d'effectuer un rendu de la scène courante. Nous verrons dans de prochains articles les possibilités et les méthodes propres aux sous-modules **Radio** et **Render**.

3. Exemples de scripts

Vous trouverez ci-après deux exemples de scripts faisant appel au menu **Scene**. Ils n'ont pas de grande prétention, à part d'automatiser certaines tâches qui peuvent être longues et répétitives lorsque vos scènes contiennent un très grand nombre d'objets, et de remettre en pratique les fonctions élémentaires **For** et **if**.

3.1 Fusion des calques actifs

Le petit script qui suit permet, pour la scène courante et celle-ci seulement, de déplacer vers un même calque tous les objets figurant sur les calques actifs au moment où il est lancé. Les calques dont le contenu a été déplacé deviennent donc vides.

Ce script illustre la différence importante entre la méthode **setLayers()** du module **Scene** et la variable **Layers** du module **Object**: la première définit le calque actif affiché dans la fenêtre 3D, tandis que le second indique le ou les calques sur lesquels un objet est visible.

```
01:    import Blender
02:    from Blender import Scene, Object
03:    print '==============='
04:    print 'Debut du script'
05:    scn = Scene.GetCurrent()
06:    print 'Nom de la scene: ', scn
07:    children = scn.getChildren()
08:    print 'Liste des objets de la scene: ', children
09:    liste_calques = scn.getLayers()
10:    print 'Liste des calques actifs: ', liste_calques
11:    for lc in liste_calques:
12:            premier_calque = lc
13:            break
14:    print 'Calque de fusion: ', premier_calque
15:    item=0
16:    for obj in children:
17:            obj = children[item]
18:            for ln in obj.layers:
19:                    for lc in liste_calques:
20:                            if (ln == lc):
21:                                    fusion = 1
22:                                    obj.layers = [premier_calque]
23:                                    break
24:                            else:
25:                                    fusion=0
26:            print obj, 'fusion = ', fusion
27:            item = item + 1
28:    print 'Fin du script'
```

3.2 Fusion vers le bas

Ce script est très inspiré du précédent, mais au lieu de fusionner tous les calques actifs ensemble (dans le premier calque), il fusionne la calque actif avec le calque immédiatement situé sous lui. Par exemple, le cinquième calque est fusionné vers le bas, avec le quatrième calque. Toutefois, rien n'est prévus dans le cas d'un objet visible sur plusieurs calques à la fois: après une fusion, il n'apparaît que sur le calque (inférieur) de fusion. Il mériterait donc d'être amélioré mais cela n'a pratiquement plus rien à voir avec le module **Scene**.

Ici, les points intéressants sont d'une part le changement de calque, après la fusion, pour montrer le calque immédiatement inférieur présentant les objets « fusionnés » grâce à la méthode **setLayers()**, et l'usage de la fonction **Redraw()** pour réactualiser le contenu de la fenêtre 3D.

```
01:    import Blender
02:    from Blender import Scene, Object
03:    print '==============='
04:    print 'Debut du script'
05:    scn = Scene.GetCurrent()
06:    print 'Nom de la scene: ', scn
07:    children = scn.getChildren()
08:    print 'Liste des objets de la scene: ', children
09:    active_layer = scn.getLayers()[0]
10:    print 'Le calque actif porte le numero: ', active_layer
11:    item=0
12:    mergeddown=0
13:    for obj in children:
14:            obj = children[item]
15:            ln = obj.layers[-1]
16:            print obj, 'layer is: ', ln
17:            if (ln == active_layer):
18:                    mergeddown = 1
19:                    obj.layers = [active_layer - 1]
20:            print obj, 'Fusion vers le bas = ', mergeddown
21:            item = item + 1
22:    scn.setLayers([active_layer-1])
23:    Blender.Redraw()
24:    print 'Fin du script'
```

4. Conclusion

Nous venons d'explorer les fonctions essentielles du module **Scene**. Ce sont les plus basiques, mais elles peuvent jouer un rôle essentiel dans vos scripts dès lors que vous souhaitez insérer de nouveaux objets, mais aussi réaliser l'export des calques visibles d'une scène vers une autre scène, par exemple. C'est toutefois un module à tiroirs, puisqu'il propose également deux sous-modules; le premier, **Radio**, permet de contrôler le contexte de radiosité de vos scènes. Pour sa part, le second, **Render**, permet de contrôler le contexte de rendu de vos scènes (à ce sujet, il est à noter que chaque scène d'un même fichier **blend** peut présenter des contextes de rendu différents). Mais ces deux sous-modules ne seront étudiés que dans les prochains articles, et vous noterez alors qu'ils ne manquent pas non plus d'intérêt pratique.

ANNEXE: Blender 2.41

Au moment où vous lirez ces lignes, Blender 2.41 sera sorti depuis déjà quelques semaines. Par rapport à la 2.40, peu de nouvelles fonctionnalités du point de vue de l'utilisateur. Cette nouvelle version a en revanche vu les efforts des développeurs s'axer sur le moteur de jeu, mais aussi sur les modules Python qui peuvent faciliter l'écriture de scripts d'import ou d'export d'animations. En particulier, cette version voit le grand retour du module Pose, qui avait été mis entre parenthèse le temps de la ré-écriture du système d'animation, qui hisse désormais Blender un cran plus haut dans la liste des applications incontournables d'animation 3D.

5. Liens

La page de Blender (version actuelle: v2.41): **www.blender.org**

La documentation officielle de python pour Blender v2.40:
http://www.blender.org/documentation/240PythonDoc/index.html

La documentation de python: **http://docs.python.org/download.html**