Initiation à la programmation

Zhentao Li

29 février 2017

Les modules

Modules

- On peut ranger les définitions de fonctions se rapportant à une même application au sein d'un script commun baptisé red module.
- Un module est sauvegardé sous forme d'un fichier dont le nom a la forme `<nom du module>.py`.
- Pour utiliser un module, il faut se servir de l'instruction `import <nom du module>`.
- L'exécution de cette instruction consiste à exécuter le script définissant le module (ce script peut contenir des instructions autres que des définitions de fonctions).
- Pour importer un module, Python a besoin de connaître le chemin qui permet d'accéder au fichier correspondant. Ce chemin doit apparaître dans la liste des chemins possibles stockés dans la variable `path` du module `sys`.

Modules - Première méthode d'importation

```
>>> import random
>>> random.randint(0, 10)
9
```

Regardons de plus près cet exemple :

- L'instruction `import` vous permet d'importer toutes les fonctions du module `random`
- Ensuite, nous utilisons la fonction `randint(a, b)` du module `random`; attention cette fonction renvoie un nombre entier aléatoirement entre `a` inclus et `b` inclus.

Modules - Deuxième méthode d'importation

- Pour disposer d'une fonction du module:

```
>>> from random import randint
>>> random.randint(0, 10)
9
```

- Pour disposer de toutes les fonctions d'un module:

```
from math import *
racine = sqrt(49)
angle = pi/6
print sin(angle)
```

Normalement, `import *` est seulement utilisé dans l'interpréteur et déconseillé dans les scripts. C'est pour aider le lecteur à trouver la provenance des variables et fonctions du programme.

Modules courants

- `math` : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- `sys` : passage d'arguments, gestion de l'entrée/sortie standard

Modules courants

- `math` : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- `sys` : passage d'arguments, gestion de l'entrée/sortie standard
- `os` : dialogue avec le système d'exploitation.

Modules courants

- `math` : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- `sys` : passage d'arguments, gestion de l'entrée/sortie standard
- `os` : dialogue avec le système d'exploitation.
- `random` : génération de nombres aléatoires.

Modules courants

- `math` : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- `sys` : passage d'arguments, gestion de l'entrée/sortie standard
- `os` : dialogue avec le système d'exploitation.
- `random` : génération de nombres aléatoires.
- `time` : permet d'accéder aux fonctions gérant le temps.

Modules courants

- `math` : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- `sys` : passage d'arguments, gestion de l'entrée/sortie standard
- `os` : dialogue avec le système d'exploitation.
- `random` : génération de nombres aléatoires.
- `time` : permet d'accéder aux fonctions gérant le temps.
- `profile` : permet d'évaluer le temps d'exécution de chaque fonction dans un programme (profiling en anglais).

Modules courants

- `math` : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- `sys` : passage d'arguments, gestion de l'entrée/sortie standard
- `os` : dialogue avec le système d'exploitation.
- `random` : génération de nombres aléatoires.
- `time` : permet d'accéder aux fonctions gérant le temps.
- `profile` : permet d'évaluer le temps d'exécution de chaque fonction dans un programme (profiling en anglais).
- `urllib` : permet de récupérer des données sur internet depuis python.

Modules courants

- `math` : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- `sys` : passage d'arguments, gestion de l'entrée/sortie standard
- `os` : dialogue avec le système d'exploitation.
- `random` : génération de nombres aléatoires.
- `time` : permet d'accéder aux fonctions gérant le temps.
- `profile` : permet d'évaluer le temps d'exécution de chaque fonction dans un programme (profiling en anglais).
- `urllib` : permet de récupérer des données sur internet depuis python.
- `Tkinter` : interface python avec Tk (permet de créer des objets graphiques; nécessite d'installer Tk).

Modules courants

- `math` : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- `sys` : passage d'arguments, gestion de l'entrée/sortie standard
- `os` : dialogue avec le système d'exploitation.
- `random` : génération de nombres aléatoires.
- `time` : permet d'accéder aux fonctions gérant le temps.
- `profile` : permet d'évaluer le temps d'exécution de chaque fonction dans un programme (profiling en anglais).
- `urllib` : permet de récupérer des données sur internet depuis python.
- `Tkinter` : interface python avec Tk (permet de créer des objets graphiques; nécessite d'installer Tk).
- `re` : gestion des expressions régulières.

Modules courants

- `math` : fonctions et constantes mathématiques de base (sin, cos, exp, pi...).
- `sys` : passage d'arguments, gestion de l'entrée/sortie standard
- `os` : dialogue avec le système d'exploitation.
- `random` : génération de nombres aléatoires.
- `time` : permet d'accéder aux fonctions gérant le temps.
- `profile` : permet d'évaluer le temps d'exécution de chaque fonction dans un programme (profiling en anglais).
- `urllib` : permet de récupérer des données sur internet depuis python.
- `Tkinter` : interface python avec Tk (permet de créer des objets graphiques; nécessite d'installer Tk).
- `re` : gestion des expressions régulières.
- `pickle` : écriture et lecture de structures python.

Quelques fonctions des modules courants

```
>>> math.pow(2, 10)   #Exposant
1024.0
>>> code_retour = os.system("echo 123")  #Commande externe
123
>>> random.randint(0, 10)  #Choisis un entier au hasard
2
>>> random.choice(["a", 3, "d"])  #Choisis un element d'une liste
'd'
>>> time.time()  # Nombre de secondes depuis 1970-01-01
1393935277.116552
>>> time.sleep(3)  # Attendre 3 seconds
>>> profile.run("time.sleep(3)")  # Calcule le temps d'excution
         4 function calls in 0.000 seconds
   Ordered by: standard name
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 :0(setprofile)
        1    0.000    0.000    0.000    0.000 :0(sleep)
[...]
```

Quelques fonctions des modules courants

```
>>> handle = urllib.urlopen("http://www.google.com")
>>> html = handle.read()  # Lire le contenu d'une page comme un fichier
>>> print html
'<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage">
<head><meta itemprop="image" content="/images/google_favicon_128.png"><ti
tle>Google</title><script>(function(){\nwindow.google={kEI:"i8QVU8ejFqTt4
gTBvIDIAw",getEI:function(a){for(var b;a&&(!a.getAttribute||!(b=a.getAttr
[...]
>>> # Recherche des chaines de caractere dans un texte.
>>> # Dans cet exemple, on trouve les liens sur la page web'.
>>> re.findall('href="(.*?)"', html)
['/search?', 'http://www.google.fr/imghp?hl=fr&tab=wi', 'http://maps.
google.fr/maps?hl=fr&tab=wl', 'https://play.google.com/?hl=fr&tab=w8',
'http://www.youtube.com/?gl=FR&tab=w1', 'http://news.google.fr/nwshp?
[...]
>>> # Sauvegarde de variables dans des fichiers et lecture
>>> pickle.dump(liste, open("sauvegarde.pkl", "w"))
>>> listelue = pickle.load(open("sauvegarde.pkl"))
>>> listelue
['a', 'b', 'c']
```

Obtenir de l'aide

- Utiliser `help` pour obtenir de l'aide sur un module.

```
>>> import random
>>> help(random)
```

Obtenir de l'aide

- Utiliser `help` pour obtenir de l'aide sur un module.

```
>>> import random
>>> help(random)
```

- De même pour de l'aide sur une fonction.

```
>>> help(random.randint)
```

Obtenir de l'aide

- Utiliser `help` pour obtenir de l'aide sur un module.

```
>>> import random
>>> help(random)
```

- De même pour de l'aide sur une fonction.

```
>>> help(random.randint)
```

- Ça marche même sur des variables et des types.

```
>>> liste = [10, 20]
>>> help(liste)
>>> help(str)
```

Les fichiers

Fichiers: utilisation courante

```
$ cat exemple.txt
Ceci est la premiere ligne
Ceci est la deuxieme ligne
Ceci est la troisieme ligne
Ceci est la quatrieme et derniere ligne
```

```
>>> nom_fichier = "exemple.txt"
>>> contenu_fichier = open(nom_fichier).read()
>>> contenu_fichier
'Ceci est la premiere ligne\nCeci est la deuxieme ligne\nCe
ci est la
troisieme ligne\nCeci est la quatrieme et derniere ligne\n'
>>> contenu_a_ecrire = "hello world\n"
>>> open("exemple2.txt", "w").write(contenu_a_ecrire)
$ cat exemple2.txt
hello world
```

Autre exemple

Créez un fichier dans un éditeur de texte que vous sauverez dans votre répertoire avec le nom `exemple.txt`, par exemple :

```
Ceci est la premiere ligne
Ceci est la deuxieme ligne
Ceci est la troisieme ligne
Ceci est la quatrieme et derniere ligne
```

Exemple readlines

```
>>> file_in = open('exemple.txt','r')
>>> file_in
<open file 'exemple.txt', mode 'r' at 0x40155b20>
>>> lignes = file_in.readlines()
>>> lignes
['Ceci est la premiere ligne\n', 'Ceci est la deuxieme ligne\n',
'Ceci est la troisieme ligne\n', 'Ceci est la quatrieme et derniere ligne\n']
>>> for i in lignes:
...   print i
...
Ceci est la premiere ligne

Ceci est la deuxieme ligne

Ceci est la troisieme ligne

Ceci est la quatrieme et derniere ligne
>>> file_in.close()
>>> file_in
<closed file 'exemple.txt', mode 'r' at 0x40155b20>
>>> file_in.close()
```

Exemple read

```
Ceci est la premiere ligne
Ceci est la deuxieme ligne
Ceci est la troisieme ligne
Ceci est la quatrieme et derniere ligne
```

```
>>> file_in = open("exemple.txt", "r")
>>> file_in.read()
'Ceci est la premiere ligne\nCeci est la deuxieme ligne\nCe
ci est la troisieme ligne\nCeci est la quatrieme et dernier
e ligne\n'
>>> file_in.close()
```

Exemple `tell`

```
Ceci est la premiere ligne
Ceci est la deuxieme ligne
Ceci est la troisieme ligne
Ceci est la quatrieme et derniere ligne
```

```
>>> file_in = open("exemple.txt", "r")
>>> file_in.tell()
0L
>>> file_in.readline()
'Ceci est la premiere ligne\n'
>>> file_in.tell()
27L
>>> file_in.seek(0)
>>> file_in.tell()
0L
>>> file_in.readline()
'Ceci est la premiere ligne\n'
>>> file_in.close()
```

Exemple `write`

```
>>> animaux = ['girafe', 'hippopotame', 'singe', 'dahu',
'ornithorynque']
>>> file_out = open('exemple2.txt', 'w')
>>> for i in animaux:
...     file_out.write(i)
...
>>> file_out.close()
>>>
$ cat exemple2.txt
girafehippopotamesingedahuornithorynque
$
```

Utilisation de fichiers

- Il est important de dissocier les données des programmes qui les utilisent en rangeant ces données dans des fichiers séparés.
- Pour utiliser un fichier dans Python, il faut commencer par l'ouvrir ce qui retourne un objet de type `file`.
- Le paramètre facultatif `<mode>` indique le mode d'ouverture du fichier :
  - `r` : mode lecture (le fichier doit exister préalablement)
  - `w` : mode écriture (si le fichier existe, les données sont écrasées, sinon le fichier est créé)
  - `a` : mode ajout (si le fichier existe, les données écrites vont l'être après celles existantes, sinon le fichier est créé)
- Si le mode est omis, le mode par défaut est `r`.

Projets