AGRO CAMPUS OUEST

Python 3
–
Chaines de caractères et commandes systèmes

AGRO CAMPUS OUEST

Plan

1) Importation de modules
2) Exceptions
3) Modules os et sys
4) Manipulation de chaines de caractères

Extensions au langage de base : importation de modules

- Python est organisé en modules
  - Un module contient des fonctionnalités spécifiques qui étendent les possibilités du langage
    - Un module est simplement un fichier contenant des définitions et des instructions Python. Le nom du fichier est le nom du module auquel est ajouté le suffixe .py.
  - Avantages:
    - permet de relancer un même programme sans tout réécrire (en créant éventuellement un script)
    - d'utiliser dans plusieurs programmes une même fonction sans la réécrire dans chaque programme.
  - Les modules sont structurées hiérarchiquement
    - Modules, sous-modules, etc

Extensions au langage de base : importation de modules

- L'instruction "`import <module>`" importe tout le module `<module>`, en exécutant le total du module.
  - Le module est exécuté en premier avant de pouvoir travailler sur les objets.
  - Pour utiliser ensuite le module, nous écrirons "`<module>.nom`".
- L'instruction "`from`" s'utilise avec "`import`" de la manière "`from <module> import <nom>`".
  - Seul l'objet `<nom>` du module `<module>` sera importé, le code restant du module sera aussi exécuté, mais les objets ne seront pas mémorisés et les éventuelles variables ne seront pas affectées.
  - Si `<nom>` est remplacé par `*`, on obtient alors une copie de tous les noms définis a la racine du module
  - Pour utiliser ensuite nom, nous écrirons juste "`<nom>`".

Extensions au langage de base : importation de modules

- Exemple
  - Exemples d'importation d'une fonctionnalité du module `os`
  - Trois manières d'accéder à la même fonction

```
import os

os.path.isdir('toto')
```

```
from os import path

path.isdir('toto')
```

```
from os.path import isdir

isdir('toto')
```

Extensions au langage de base : importation de modules

- Renommage des modules
  - Le mot clé « as » permet de renommer un module
  - Facilite son utilisation (avec des noms raccourcis)
- Exemple

```
import numpy
A = numpy.zeros(10)
```

```
import numpy as np
A = np.zeros(10)
```

Extensions au langage de base : importation de modules

- Comment connaître les modules intéressants
  - Certains modules sont très usuels : un peu de pratique vous permettra de les connaître
  - Anaconda propose un liste de modules « standards » qui peuvent être facilement installé
  - Google peut vous aider à identifier les modules spécifiques dont vous avez besoin
  - Vos cours introduiront les modules spécifiques aux données spatiales (plutôt en M2)

AGRO CAMPUS OUEST

Plan

1) Importation de modules
2) Exceptions
3) Modules os et sys
4) Manipulation de chaines de caractères

Gestion des exceptions

- Qu'est ce qu'une exception ?
  - Une « exception » désigne une erreur d'exécution qui est survenue lors de l'exécution d'un programme.
    - Correspond à un comportement « exceptionnel » qu'il faut traiter à part
  - Dans certains cas, on peut anticiper la survenue telles erreurs, par exemple
    - Erreur de saisie (ce n'est pas un entier)
    - Opération impossible (division par 0
  - Plutôt que d'éviter les erreurs … on peut préférer gérer (a posteriori) les exceptions

```
X = int(input())
B = 45/X
```

Gestion des exceptions

- Exemples des messages sur les deux erreurs
  - Deux types d'exceptions différents
  - Le message d'erreur donne le « type » des exceptions

```
Traceback (most recent call last):

  File "<ipython-input-13-3745c47b4620>", line 1, in <module>
    X = int(input())

ValueError: invalid literal for int() with base 10: 'toto'
```

```
Traceback (most recent call last):

  File "<ipython-input-15-7ddb3339545c>", line 1, in <module>
    B = 45/X

ZeroDivisionError: division by zero
```

Gestion des exceptions

- La gestion des exceptions se fait grâce à l'utilisation d'une expression `try` / `except` [/`else` / `finally`].

```
try :
   X = int(input())
except ValueError :
   print('erreur de saisie')

B = 45/X
```

Gestion des exceptions

- La gestion des exceptions se fait grâce à l'utilisation d'une expression `try` / `except` [/`else` / `finally`].

```
try :
   X = int(input())
    B = 45/X
except ValueError :
   print('erreur de saisie')
except ZeroDivisionError :
   Print('division par zero')
```

Gestion des exceptions

- Exemple pour une saisie correcte de nombre

```
while True:
    print("saisir un entier")
    try:
        x = int(input())
    except ValueError:
        print("un entier abrutis!!")
    else:
        break

print("Voilà ...")
```

AGRO CAMPUS OUEST

Plan

1) Importation de modules
2) Exceptions
3) Modules os et sys
4) Manipulation de chaines de caractères

Modules de base: sys

- Très souvent utilisé
- Utile pour les utilisations “linux”
  - sys.argv donne accès aux arguments de la commande en ligne
  - sys.stdin, sys.stdout, sys.stderr donne accès aux entrées et sorties standard/erreur
- sys.exit() met fin au programme (instantannément)

Modules de base: sys

```
import sys
data = sys.stdin.read()

if len(sys.argv) < 2:
    print("There is a problem!", file=sys.stderr)
    sys.exit()

filename = sys.argv[1]

more_data = open(filename,'r').read()
results = compute(data,more_data)

print(results, file=sys.stdout)
```

c:\> test.py cmd-line-arg1 < stdin.txt > stdout.txt

Modules de base : os, os.path

- Les chemins de fichier
  - Les fichiers sont identifiés par des noms
  - Plus généralement … le nom d'un fichier est un chemin
    - Une partie de description d'un dossier
    - Le nom du fichier
  - Deux types de nommage
    - Chemins relatifs
    - Chemins absolus
  - En fonction des systèmes les conventions peuvent changer (Windows/Linux)
- Dans un programme
  - Un **nom de fichier** est une chaine de caractères dans un programme

Module de base : os, os.path

- `os.getcwd()` : chemin absolue du répertoire courant
- `os.chdir(path)` : définie le répertoire de travail
- `os.mkdir(path)` : création d'un répertoire
- `os.path.abspath(filename)`
  - Chemin absolu vers le fichier nommé « filename »
- `os.path.exists(filename)`
  - Teste si le fichier "filename" existe
- `os.path.join(path1,path2,path3)`
  - Joindre des bouts de chemin (avec les bons séparateurs)
- `os.path.split(path)`
  - Sépare simplement le nom de fichier du chemin du dossier.

Basic modules: os, os.path

```
# Import important modules
import os
import os.path
import sys

# Check for command-line arguments
if len(sys.argv) < 2:
    print("There is a problem: No enough arguments!")
    sys.exit()

# Get the filename
filename = sys.argv[1]

# Get the current working directory
cwd = os.getcwd()
print cwd

# Turn a filename into a full path
abspath = os.path.abspath(filename)
print abspath
```

Basic modules: os, os.path

```
# make the home directory path
homedir = '/home/student'
print( homedir )

# Check if the file is there
if os.path.exists(filename):
    print( filename + " is there")
else:
    print( filename + " does not exist")

# Check if the file is in the current working directory
new_filename = os.path.join(cwd,filename)
if os.path.exists(new_filename):
    print( new_filename + "is there" )
else:
    print( new_filename + "does not exist")

# Check if the file is in home directory
new_filename = os.path.join(homedir,filename)
if os.path.exists(new_filename):
    print( new_filename + "is there" )
else:
    print( new_filename + "does not exist")
```

Module de base : os

- Instruction : `os.system`
  - Prend en paramètre une **chaîne de caractères** qui sera une « commande système »
  - Permet de demander à l'ordinateur d'exécuter n'importe quel commande
  - Donne accès à des fonctionnalités extérieures
    - e.g. commandes GDAL
      - `gdalwrap`
      - `gdal_transform`
      - `gdal_merge`
      - ...
  - Exemple
    - `gdalwarp -t_srs '+proj=utm +zone=11 +datum=WGS84' raw_spot.tif utm11.tif`

Généralisable à toutes les URI

- URI : unified resource identifier
  - Convention de nommage des ressources accessibles via interêt
- Même principe général pour la programmation
  - Une URI est une chaîne de caractères dans un programme
  - Utilisation d'une librairie permettant d'exploiter cette description de ressources
    - Librairie urllib2
      - Ouverture d'une URI
      - Récupération directe d'une image

```
import urllib.request
urllib.request.urlretrieve("http://modis-atmos.gsfc.nasa.gov/IMAGES/MYD02/GRANULE/2013_07_07/188.1305.rgb143.jpg", "Image.jpg")
```

Problèmes communs

- Chaînes de caractères utilisés pour encoder
  - Des commandes
  - Des noms de fichiers
  - Des URI
-

AGRO CAMPUS OUEST

Plan

1) Importation de modules
2) Exceptions
3) Modules os et sys
4) Manipulation de chaines de caractères

Manipulations des chaînes de caractères

- Découpage - assemblage : `split` et `join`
  - Les méthodes split et join permettent de découper une chaine selon un séparateur pour obtenir une liste, et à l'inverse de reconstruire une chaîne à partir d'une liste.
  - split permet donc de découper
    - `'abc=:=def=:=ghi=:=jkl'.split('=:=')`
  - Et à l'inverse
    - `"=:=".join(['abc', 'def', 'ghi', 'jkl'])`
  - Attention toutefois si le séparateur est un terminateur, la liste résultat contient alors une dernière chaîne vide. En pratique, on utilisera la méthode strip, que nous allons voir ci-dessous, avant la méthode split pour éviter ce problème.
    - `'abc;def;ghi;jkl;'.split(';')`
  - Qui s'inverse correctement cependant
    - `";".join(['abc', 'def', 'ghi', 'jkl', ''])`

Manipulations des chaînes de caractères

- Remplacements : replace
  - `replace` est très pratique pour remplacer une sous-chaîne par une autre, avec une limite éventuelle sur le nombre de remplacements
    - `"abcdefabcdefabcdef".replace("abc", "zoo")`
    - `"abcdefabcdefabcdef".replace("abc", "zoo", 2)`
  - Plusieurs appels à replace peuvent être chaînés comme ceci
    - `"les [x] qui disent [y]".replace("[x]","chevaliers").replace("[y]", "Ni")`

Manipulations des chaînes de caractères

- Nettoyage : `strip`
  - On pourrait par exemple utiliser replace pour enlever les espaces dans une chaîne, ce qui peut être utile pour "nettoyer" comme ceci
    - `" abc:def:ghi ".replace(" ", "")`
  - Toutefois bien souvent on préfère utiliser strip qui ne s'occupe que du début et de la fin de la chaîne, et gère aussi les tabulations et autres retour à la ligne
    - `" \tune chaine avec des trucs qui dépassent \n".strip()`
  - On peut appliquer strip avant split pour éviter le problème du dernier élément vide.
    - `'abc;def;ghi;jkl;'.strip(';').split(';')`

Manipulations des chaînes de caractères

- Rechercher une sous-chaîne
  - Plusieurs outils permettent de chercher une sous-chaîne. Il existe `find` qui renvoie le plus petit index où on trouve la sous-chaîne

```
"abcdefcdefghefghijk".find("def")
"abcdefcdefghefghijk".find("zoo")
```

    - Que retourne la fonction dans le second cas ?
    - De manière très Pythoniste :

```
"def" in "abcdefcdefghefghijk"
```

  - `rfind` fonctionne comme `find` mais en partant de la fin de la chaîne

```
"abcdefcdefghefghijk".rfind("fgh")
"abcdefcdefghefghijk"[13]
```

Manipulations des chaînes de caractères

- La méthode `index` se comporte comme `find`, mais en cas d'absence elle lève une exception plutôt que de renvoyer -1

```
"abcdefcdefghefghijk".index("def")
try:
    "abcdefcdefghefghijk".index("zoo")
except Exception as e:
    print("OOPS", type(e), e)
```

- La méthode `count` compte le nombre d'occurrences d'une sous-chaîne

```
"abcdefcdefghefghijk".count("ef")
```

- Signalons enfin les méthodes de commodité suivantes

```
"abcdefcdefghefghijk".startswith("abcd")
"abcdefcdefghefghijk".endswith("ghijk")
```

Manipulations des chaînes de caractères

- Sous-chaînes :
  - Il est possible d'extraire des sous-chaines de caractères en utilisant la notation crocheté
    - On donne des positions entre les crochets
    - Obligatoirement des entiers (c'est une forme d'opérateur)
    - Les positions commencent à 0

```
>>> Toto = 'ma chaine'
>>> Toto[4]
h
>>> Toto[4:6]
hai
```

Formatage des chaînes de caractères

- Le formatage d'une chaine de caractères désigne la construction d'une chaine de caractères à partir d'informations contenues dans des variables
- Il existe plusieurs manières de formater une chaîne de caractères
  - La concaténation simple
  - L'utilisation des {}
  - L'utilisation des %

Formattage par l'exemple

Créez `s` initialisé à `"abcdef"`, `i` initialisé à 12 et `f` initialisé à 3.14159

Étudiez et comprenez les formattages suivants :

```
format(i,"d")
format(i,"12d")
format(i,">12d")
format(i,".<12d")
format(i,"*^12d")
format(f,"f")
format(f,"12f")
format(f,"0>12.2f")
format(f,"^12.5f")
format(f,".3f")
```

```
format(s,"s")
format(s,"12s")
format(s,">12s")
format(s,".^12s")
format(s,"*<12s")
```

Composition d'une chaîne de caractères

```
print( "Je m'appelle {0} {1} ({3} {0} pour
l'administration) et j'ai {2} ".format(prenom, nom,
age, nom.upper()))
```

- On peut utiliser simplement l'ordre plutôt que de numéroter les accolades

```
>>> date = "Dimanche 24 juillet 2011"
>>> heure = "17:00"
>>> print("Cela s'est produit le {}, à
{}.".format(date, heure))
Cela s'est produit le Dimanche 24 juillet
2011, à 17:00.
```

Chaine de caractères « raw »

- On trouve parfois des expressions de la forme

  `r'ceci est une chaîne de caractère'`

- Le r avant le simple quote indique que la chaîne de caractère doit être prise comme une « chaine brute »
  - Dans une chaine classique, il y a des caractères spéciaux tels que `'\n'`, `'\t'` … en gros le `'\'` sert à coder un caractère spécial … pour écrire un `'\'` il faut donc écrire `'\\'`
  - Dans une chaîne brute, on inhibe la transformation des caractères spéciaux
- Un exemple

```
print( 'ceci es\t u\n chai\ne')
print(r'ceci es\t u\n chai\ne')
```