Modules et notions d'import

La bibliothèque standard
Utilisation de modules
Conversion de type
Opérateurs et comparaison

Alexandre Gramfort

Programmation Python 2 : Modules et notions d'import

Alexandre Gramfort

La bibliothèque standard et ses modules

- Les fonctions Python sont organisées par *modules*
- Bibliothèque standard Python (*Python Standard Library*) : collection de modules donnant accès à des fonctionnalités de bases : appels au système d'exploitation, gestion des fichiers, gestion des chaînes de caractères, interface réseau, etc.

Références

- The Python Language Reference:

http://docs.python.org/2/reference/index.html

- The Python Standard Library:

http://docs.python.org/2/library/

Utilisation des modules

- Un module doit être *importé* avant de pouvoir être utilisé, exemple :

```
In [21]: import math
```

Alexandre Gramfort

Modules et notions d'import

La bibliothèque standard
Utilisation de modules
Conversion de type
Opérateurs et comparaison

Le module math peut maintenant être utilisé :

In [22]:
```
import math

x = math.cos(2 * math.pi)

print(x)
```

```
1.0
```

Ou bien en important que les fonctions dont on a besoin:

In [23]:
```
from math import cos, pi

x = cos(2 * pi)

print(x)
```

```
1.0
```

Alexandre Gramfort

Modules et notions d'import

La bibliothèque standard
Utilisation de modules
Conversion de type
Opérateurs et comparaison

Ou bien en important tout: (non recommandé)

In [24]:
```
from math import *
tanh(1)
```

Out[24]: 0.7615941559557649

Ou bien en important avec un autre nom (par exemple abréviation)

In [25]:
```
import math as m
print(m.cos(1.))
```

```
0.540302305868
```

Connaitre le contenu d'un module

- Une fois un module importé on peut lister les symboles disponibles avec la fonction `dir`:

In [26]:
```
import math

print(dir(math))
```

```
['__doc__', '__file__', '__name__', '__package__', 'acos',
 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil
', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc
', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', '
frexp', 'fsum', 'gamma', 'hypot', 'isinf', 'isnan', 'ldexp
', 'lgamma', 'log', 'log10', 'log1p', 'modf', 'pi', 'pow',
 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Modules et notions d'import

La bibliothèque standard
Utilisation de modules
Conversion de type
Opérateurs et comparaison

- Pour accéder à l'aide : `help`

```
In [27]: help(math.log)
```

```
Help on built-in function log in module math:

log(...)
    log(x[, base])

    Return the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.
```

```
In [28]: math.log(10, 2)
```

```
Out[28]: 3.3219280948873626
```

Alexandre Gramfort

Modules et notions d'import

La bibliothèque standard
Utilisation de modules
Conversion de type
Opérateurs et comparaison

Alexandre Gramfort

`help` peut être aussi utilisée sur des modules :

In [29]:
```
help(math)
```

```
Help on module math:

NAME
    math

FILE
    /Users/alex/anaconda/python.app/Contents/lib/python2.7/
lib-dynload/math.so

DESCRIPTION
    This module is always available.  It provides access to
 the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.
```

Fractions

In [30]:
```
import fractions
a = fractions.Fraction(2, 3)
b = fractions.Fraction(1, 2)
print(a + b)
```

```
7/6
```

Utiliser `isinstance` pour tester les types des variables :

In [31]:
```
print(type(a))
print(isinstance(a, fractions.Fraction))
```

```
<class 'fractions.Fraction'>
True
```

In [32]:
```
a = fractions.Fraction(1, 1)
print(isinstance(a, int))
```

```
False
```

Modules et notions d'import

La bibliothèque standard
Utilisation de modules
Conversion de type
Opérateurs et comparaison

Conversion de type (cast en Anglais)

In [33]:
```
x = 1.5
print(x, type(x))
```

```
(1.5, <type 'float'>)
```

In [34]:
```
x = int(x)
print(x, type(x))
```

```
(1, <type 'int'>)
```

In [35]:
```
z = complex(x)
print(z, type(z))
```

```
((1+0j), <type 'complex'>)
```

Alexandre Gramfort

Modules et notions d'import

La bibliothèque standard
Utilisation de modules
Conversion de type
Opérateurs et comparaison

```
In [36]: x = float(z)
         print(x, type(x))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-36-7002fbe22da4> in <module>()
----> 1 x = float(z)
      2 print(x, type(x))

TypeError: can't convert complex to float
```

Alexandre Gramfort

Modules et notions d'import

La bibliothèque standard
Utilisation de modules
Conversion de type
Opérateurs et comparaison

Operateurs et comparaisons

Opérations booléennes en anglais `and`, `not`, `or`.

```
In [37]: True and False
Out[37]: False

In [38]: not False
Out[38]: True

In [39]: True or False
Out[39]: True
```

Alexandre Gramfort

Modules et notions d'import

La bibliothèque standard
Utilisation de modules
Conversion de type
Opérateurs et comparaison

Alexandre Gramfort

Comparaisons >, <, >= (plus grand ou égal), <= (inférieur ou égal), == égalité et != inégalité

```
In [40]: 2 > 1
Out[40]: True

In [41]: 2 > 2
Out[41]: False

In [42]: 2 >= 2
Out[42]: True

In [43]: 2 == 2
Out[43]: True

In [44]: 2 != 3
Out[44]: True
```