Algorithmique et Programmation

Modules de fonctions

Elise Bonzon

elise.bonzon@mi.parisdescartes.fr

LIPADE - Université Paris Descartes
http://www.math-info.univ-paris5.fr/∽bonzon/

Modules de fonctions

Modules de fonctions

Les modules en Python

Modules

Un module de fonctions est un fichier qui regroupe des ensembles de fonctions.

Un module peut également regrouper d'autres outils (classes, données...). On utilise souvent le terme de *bibliothèque*.

Les modules en Python

Modules

Un module de fonctions est un fichier qui regroupe des ensembles de fonctions.

Un module peut également regrouper d'autres outils (classes, données...). On utilise souvent le terme de *bibliothèque*.

- L'utilisation des modules est très fréquente. Elle permet de :
  - ré-utiliser du code
  - isoler, dans un espace identifié, des fonctionnalités particulières
- Il existe un grand nombre de modules fournis d'office avec Python
- Il est également possible de définir ses propres modules

Modules de fonction

- Le **nom d'un fichier** en Python se termine par `.py` et ne contient que des lettres minuscules, des chiffres et des soulignés. Aucun espace !

Modules de fonction

- Le **nom d'un fichier** en Python se termine par `.py` et ne contient que des lettres minuscules, des chiffres et des soulignés. Aucun espace !
  - `essai2.py` ; `essai_tortue.py` ; ~~`Essai Tortue.py`~~

Modules de fonction

- Le **nom d'un fichier** en Python se termine par `.py` et ne contient que des lettres minuscules, des chiffres et des soulignés. Aucun espace !
  - `essai2.py` ; `essai_tortue.py` ; ~~`Essai Tortue.py`~~
- Un module est un fichier `nom_fichier.py` écrit en Python et contenant :

Modules de fonction

- Le **nom d'un fichier** en Python se termine par `.py` et ne contient que des lettres minuscules, des chiffres et des soulignés. Aucun espace !
  - `essai2.py` ; `essai_tortue.py` ; ~~`Essai Tortue.py`~~
- Un module est un fichier `nom_fichier.py` écrit en Python et contenant :
  - des définitions

Modules de fonction

- Le **nom d'un fichier** en Python se termine par `.py` et ne contient que des lettres minuscules, des chiffres et des soulignés. Aucun espace !
  - `essai2.py` ; `essai_tortue.py` ; ~~`Essai Tortue.py`~~
- Un module est un fichier `nom_fichier.py` écrit en Python et contenant :
  - des définitions
  - des instructions (par exemple d'affichage)

Modules de fonction

- Le **nom d'un fichier** en Python se termine par `.py` et ne contient que des lettres minuscules, des chiffres et des soulignés. Aucun espace !
  - `essai2.py` ; `essai_tortue.py` ; ~~`Essai Tortue.py`~~
- Un module est un fichier `nom_fichier.py` écrit en Python et contenant :
  - des définitions
  - des instructions (par exemple d'affichage)
- Un module peut être destiné :

Modules de fonction

- Le **nom d'un fichier** en Python se termine par `.py` et ne contient que des lettres minuscules, des chiffres et des soulignés. Aucun espace !
  - `essai2.py` ; `essai_tortue.py` ; ~~`Essai Tortue.py`~~
- Un module est un fichier `nom_fichier.py` écrit en Python et contenant :
  - des définitions
  - des instructions (par exemple d'affichage)
- Un module peut être destiné :
  - à être directement exécuté. Lorsqu'il est court et effectue une action ré-utilisable, on parle souvent d'un script

Modules de fonction

- Le **nom d'un fichier** en Python se termine par `.py` et ne contient que des lettres minuscules, des chiffres et des soulignés. Aucun espace !
  - `essai2.py` ; `essai_tortue.py` ; ~~`Essai Tortue.py`~~
- Un module est un fichier `nom_fichier.py` écrit en Python et contenant :
  - des définitions
  - des instructions (par exemple d'affichage)
- Un module peut être destiné :
  - à être directement exécuté. Lorsqu'il est court et effectue une action ré-utilisable, on parle souvent d'un script
  - à être utilisé par un autre module. Il exporte alors un certain nombre de fonctionnalités.

Syntaxes d'import d'un module

3 syntaxes possibles pour importer un module, ou les fonctions d'un module, dans un autre fichier.

```
import nom_module

from nom_module import nom1, nom2...

from nom_module import *
```

Un exemple de module : calcul de la TVA

```
#Fichier tva.py
TVA = 19.6
COEFF = 1 + TVA / 100
def prix_ttc(p):
    """ float x float --> float
    Retourne le prix TTC en fonction du COEFF défini par TVA"""
    return p * COEFF
```

Un exemple de module : calcul de la TVA

```
#Fichier tva.py
TVA = 19.6
COEFF = 1 + TVA / 100
def prix_ttc(p):
    """ float x float --> float
    Retourne le prix TTC en fonction du COEFF défini par TVA"""
    return p * COEFF
```

```
#Fichier use_tva.py
import tva
```

Un exemple de module : calcul de la TVA

```
#Fichier tva.py
TVA = 19.6
COEFF = 1 + TVA / 100
def prix_ttc(p):
    """ float x float --> float
    Retourne le prix TTC en fonction du COEFF défini par TVA"""
    return p * COEFF
```

```
#Fichier use_tva.py
import tva
def bilan(x):
    """float --> None
    Affiche le prix TTC du prix HT donné en entrée"""
    #Utilisation prix_ttc définie dans le module tva
    prix = tva.prix_ttc(x)
    print('Avec un taux de', tva.TVA, end = '%, ')
    print('un produit à', x, 'euros HT vaut', prix, 'euros TTC.')

bilan(26)
```

import nom_module.py

```
import tva
```

- Le module `tva.py` définit les variables `TVA`, `COEFF` et la fonction `prix_ttc`.
- Utilisée dans le module `use-tva`, l'instruction `import tva` rend les définitions de `tva` accessibles depuis `use-tva`
- Pour accéder aux variables et fonctions définies dans le module `tva`, il faut préfixer leur nom par le nom de leur module :
  - `tva.COEFF` ; `tva.TVA` ; `tva.prix_ttc`

import nom_module.py as

```
import nom_module as nm
```

- Il est possible que le module importé ait un nom un peu long, et qu'il soit fastidieux de le préfixer à chaque utilisation
- Il est possible de le renommer, avec la commande `import nom_module as nm`
- Pour accéder aux variables et fonctions définies dans le module `nom_module`, il faut préfixer leur nom par le nom choisi :
  - `nm.fonction1` ...

from nom_module.py import

```
from tva import TVA
```

from nom_module.py import

```
from tva import TVA
```

- Permet de n'importer que quelques fonctions ou variables du fichier `tva.py`

from nom_module.py import

```
from tva import TVA
```

- Permet de n'importer que quelques fonctions ou variables du fichier `tva.py`
- Il ne faut plus préfixer le nom des variables par le nom du module

from nom_module.py import

```
from tva import TVA
```

- Permet de n'importer que quelques fonctions ou variables du fichier `tva.py`
- Il ne faut plus préfixer le nom des variables par le nom du module

```
>>> from tva import TVA, prix_ttc
```

from nom_module.py import

```
from tva import TVA
```

- Permet de n'importer que quelques fonctions ou variables du fichier `tva.py`
- Il ne faut plus préfixer le nom des variables par le nom du module

```
>>> from tva import TVA, prix_ttc
>>> prix_ttc(26)
31.096
```

from nom_module.py import

```
from tva import TVA
```

- Permet de n'importer que quelques fonctions ou variables du fichier `tva.py`
- Il ne faut plus préfixer le nom des variables par le nom du module

```
>>> from tva import TVA, prix_ttc
>>> prix_ttc(26)
31.096
>>> TVA
19.6
```

from nom_module.py import

```
from tva import TVA
```

- Permet de n'importer que quelques fonctions ou variables du fichier tva.py
- Il ne faut plus préfixer le nom des variables par le nom du module

```
>>> from tva import TVA, prix_ttc
>>> prix_ttc(26)
31.096
>>> TVA
19.6
>>> tva.TVA
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'tva' is not defined
```

Import de fonctions

```
#Fichier tva.py
TVA = 19.6
COEFF = 1 + TVA / 100
```

Import de fonctions

```
#Fichier tva.py
TVA = 19.6
COEFF = 1 + TVA / 100
```

```
#Fichier tva_55.py
TVA = 5.5
COEFF = 1 + TVA / 100
```

Import de fonctions

```
#Fichier tva.py
TVA = 19.6
COEFF = 1 + TVA / 100
```

```
#Fichier tva_55.py
TVA = 5.5
COEFF = 1 + TVA / 100
```

```
#Fichier use_tva.py
from tva import TVA
from tva_55 import TVA
```

Import de fonctions

```
#Fichier tva.py
TVA = 19.6
COEFF = 1 + TVA / 100
```

```
#Fichier tva_55.py
TVA = 5.5
COEFF = 1 + TVA / 100
```

```
#Fichier use_tva.py
from tva import TVA
from tva_55 import TVA
```

$\Rightarrow$ conflit sur la variable `TVA` !

Import de fonctions

```
#Fichier tva.py
TVA = 19.6
COEFF = 1 + TVA / 100
```

```
#Fichier tva_55.py
TVA = 5.5
COEFF = 1 + TVA / 100
```

```
#Fichier use_tva.py
from tva import TVA
from tva_55 import TVA
```

$\Rightarrow$ conflit sur la variable `TVA` !

Deux possibilités :

Import de fonctions

```
#Fichier tva.py
TVA = 19.6
COEFF = 1 + TVA / 100
```

```
#Fichier tva_55.py
TVA = 5.5
COEFF = 1 + TVA / 100
```

```
#Fichier use_tva.py
from tva import TVA
from tva_55 import TVA
```

$\Rightarrow$ conflit sur la variable `TVA` !

Deux possibilités :

- importer les modules complets
  - $\rightarrow$ `tva.TVA` et `tva_55.TVA`

Import de fonctions

```
#Fichier tva.py
TVA = 19.6
COEFF = 1 + TVA / 100
```

```
#Fichier tva_55.py
TVA = 5.5
COEFF = 1 + TVA / 100
```

```
#Fichier use_tva.py
from tva import TVA
from tva_55 import TVA
```

$\Rightarrow$ conflit sur la variable `TVA` !

Deux possibilités :

- importer les modules complets

  $\rightarrow$ `tva.TVA` et `tva_55.TVA`

- Renommer les variables

from nom_module.py import X as Y

```
from tva import TVA as TVA19
```

from nom_module.py import X as Y

```
from tva import TVA as TVA19
```

- Permet d'importer une fonction ou une variable du fichier `tva.py`, et de les renommer en `TVA19`

from nom_module.py import X as Y

```
from tva import TVA as TVA19
```

- Permet d'importer une fonction ou une variable du fichier `tva.py`, et de les renommer en `TVA19`
- Le mot `TVA19` devient un nom de variable du module courant, remplaçant le nom `TVA`

from nom_module.py import X as Y

```
from tva import TVA as TVA19
```

- Permet d'importer une fonction ou une variable du fichier `tva.py`, et de les renommer en `TVA19`
- Le mot `TVA19` devient un nom de variable du module courant, remplaçant le nom `TVA`
- Permet d'éviter des conflits de noms

from nom_module.py import *

```
from tva import *
```

from nom_module.py import *

```
from tva import *
```

- Permet d'importer **tout** ce que contient la module `tva.py`, sans avoir besoin d'utiliser le préfixe `tva`

from nom_module.py import *

```
from tva import *
```

- Permet d'importer **tout** ce que contient la module `tva.py`, sans avoir besoin d'utiliser le préfixe `tva`
- MAIS usage souvent déconseillé :

from nom_module.py import *

```
from tva import *
```

- Permet d'importer **tout** ce que contient la module `tva.py`, sans avoir besoin d'utiliser le préfixe `tva`
- MAIS usage souvent déconseillé :
  - On ne sait pas quels noms sont importés, donc risque de conflit sur les noms

from nom_module.py import *

```
from tva import *
```

- Permet d'importer **tout** ce que contient la module `tva.py`, sans avoir besoin d'utiliser le préfixe `tva`
- MAIS usage souvent déconseillé :
  - On ne sait pas quels noms sont importés, donc risque de conflit sur les noms
  - Un grand espace de noms est utilisé, souvent inutilement

Obtenir de l'aide sur les modules importés

```
>>> import tva
>>> help(tva)
Help on module tva:
NAME
tva - #Fichier tva.py
FUNCTIONS
prix_ttc(p)
DATA
COEFF = 1.196
TVA = 19.6
FILE
/PATH/tva.py
```

- Pour vous **déplacer dans l'aide**, utilisez les flèches du haut et du bas, ou les touches page-up et page-down
- Pour **quitter l'aide**, appuyez sur la touche Q.
- Pour **chercher du texte**, tapez / puis le texte que vous cherchez puis la touche Entrée.
- Pour obtenir de l'aide sur une fonction : `help(tva.prix_ttc)`

Obtenir la liste des noms définis

```
>>> dir(tva)
['COEFF', 'TVA', '__builtins__', '__cached__', '__doc__',
'__file__', '__loader__', '__name__', '__package__',
'__spec__', 'prix_ttc']
>>> dir()
['__annotations__', '__builtins__', '__doc__',
 '__loader__', '__name__', '__package__', '__spec__', 'tva']
```

- La fonction interne `dir()` est utilisée pour trouver quels noms sont définis par un module. Elle donne une liste de chaînes classées par ordre lexicographique
- Sans paramètre, `dir()` liste les noms actuellement définis
- La fonction `dir()` ne liste pas les noms des fonctions et variables natives. Si vous en voulez la liste, ils sont définis dans le module standard `__builtin__` : `dir(__builtin__)`

Abstraction des données

Un nouveau type de données

- De nombreux types existent en Pyhton : `int`, `float`, `string`, `list`, `set`...
- Mais le type *matrice* n'existe pas

$\Rightarrow$ Définir un nouveau type (abstrait) de données *matrice*, en utilisant des types de données existants

- Utiliser ce type, sans prendre en compte la façon dont il est construit

$\Rightarrow$ Concevoir une "boite à outils" *matrice*

Type Matrice 2×2

- Une matrice 2×2 est la donnée de 4 nombres, organisés en ligne et en colonne

$$\begin{pmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{pmatrix}$$

Type Matrice 2×2

- Une matrice 2×2 est la donnée de 4 nombres, organisés en ligne et en colonne

$$\begin{pmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{pmatrix}$$

- Il faut savoir :

Type Matrice 2×2

- Une matrice 2×2 est la donnée de 4 nombres, organisés en ligne et en colonne

$$\begin{pmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{pmatrix}$$

- Il faut savoir :
  - Construire une matrice

Type Matrice 2×2

- Une matrice 2×2 est la donnée de 4 nombres, organisés en ligne et en colonne

$$\begin{pmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{pmatrix}$$

- Il faut savoir :
  - Construire une matrice
  - Accéder à ses éléments

Type Matrice 2×2

- Une matrice 2×2 est la donnée de 4 nombres, organisés en ligne et en colonne

$$\begin{pmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{pmatrix}$$

- Il faut savoir :
  - Construire une matrice
  - Accéder à ses éléments
- 2 méthodes pour cela :

Type Matrice 2×2

- Une matrice 2×2 est la donnée de 4 nombres, organisés en ligne et en colonne

$$\begin{pmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{pmatrix}$$

- Il faut savoir :
  - Construire une matrice
  - Accéder à ses éléments
- 2 méthodes pour cela :
  - Implémentation par une liste

Type Matrice 2×2

- Une matrice 2×2 est la donnée de 4 nombres, organisés en ligne et en colonne

$$\begin{pmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{pmatrix}$$

- Il faut savoir :
  - Construire une matrice
  - Accéder à ses éléments
- 2 méthodes pour cela :
  - Implémentation par une liste
  - Implémentation par une liste de listes

Type Matrice 2×2

```
#En utilisant une liste
```

Type Matrice 2×2

```
#En utilisant une liste
def matrice(a,b,c,d) :
    """Int x Int x Int x Int --> List
    Retourne une matrice 2x2 formée des 4 entiers en entrée"""
    return [a, b, c, d]
```

Type Matrice 2×2

```
#En utilisant une liste
def matrice(a,b,c,d) :
    """Int x Int x Int x Int --> List
    Retourne une matrice 2x2 formée des 4 entiers en entrée"""
    return [a, b, c, d]

def matrice_ref(M, li, co):
    """List x Int x Int --> Int. Retourne l'entier M(li, co)"""
    return M[2 * li + co]
```

Type Matrice 2×2

```
#En utilisant une liste
def matrice(a,b,c,d) :
    """Int x Int x Int x Int --> List
    Retourne une matrice 2x2 formée des 4 entiers en entrée"""
    return [a, b, c, d]

def matrice_ref(M, li, co):
    """List x Int x Int --> Int. Retourne l'entier M(li, co)"""
    return M[2 * li + co]
```

```
#En utilisant une liste de listes
```

Type Matrice 2×2

```python
#En utilisant une liste
def matrice(a,b,c,d) :
    """Int x Int x Int x Int --> List
    Retourne une matrice 2x2 formée des 4 entiers en entrée"""
    return [a, b, c, d]

def matrice_ref(M, li, co):
    """List x Int x Int --> Int. Retourne l'entier M(li, co)"""
    return M[2 * li + co]
```

```python
#En utilisant une liste de listes
def matrice(a,b,c,d) :
    """Int x Int x Int x Int --> List
    Retourne une matrice 2x2 formée des 4 entiers en entrée"""
    return [[a, b], [c, d]]
```

Type Matrice 2×2

```python
#En utilisant une liste
def matrice(a,b,c,d) :
    """Int x Int x Int x Int --> List
    Retourne une matrice 2x2 formée des 4 entiers en entrée"""
    return [a, b, c, d]

def matrice_ref(M, li, co):
    """List x Int x Int --> Int. Retourne l'entier M(li, co)"""
    return M[2 * li + co]
```

```python
#En utilisant une liste de listes
def matrice(a,b,c,d) :
    """Int x Int x Int x Int --> List
    Retourne une matrice 2x2 formée des 4 entiers en entrée"""
    return [[a, b], [c, d]]

def matrice_ref(M, li, co):
    """List x Int x Int --> Int. Retourne l'entier M(li, co)"""
    return M[li][co]
```

Type Matrice 2×2

- Il n'y a pas unicité de la représentation concrète d'une matrice
- L'important, c'est la matrice, pas sa représentation !
- En dehors du module, on n'a pas besoin de connaitre la représentation concrète des matrices

```
import matrice as mat #quelle que soit la représentation choisie

def determinant(M):
    """Matrice --> int
    Retourne le déterminant d'une matrice
    M_{0,0}*M_{1,1} - M_{0,1}*M_{1,0}"""

    d1 = mat.matrice_ref(M,0,0) * mat.matrice_ref(M,1,1)
    d2 = mat.matrice_ref(M,0,1) * mat.matrice_ref(M,1,0)
    return(d1 - d2)

M = mat.matrice(1, 2, 3, 4)
print("Le déterminant de la matrice", M, "est :", determinant(M))
```

Modules courants

Modules courants

- Il existe de nombreux modules de base en Python
- La liste complète est présente à l'adresse :
  https://docs.python.org/fr/3/py-modindex.html
- N'hésitez pas à la parcourir !

Quelques modules qui vous seront utiles

- `math` : fonctions et constantes mathématiques de base (`sin, cos, exp, pi`...).
- `cmath` : fonctions et constantes mathématiques avec des nombres complexes
- `sys` : interaction avec l'interpréteur Python, passage d'arguments
- `random` : génération de nombres aléatoires.
- `fractions` : fournit un support de l'arithmétique des nombres rationnels.

Et bien d'autres !

Quelques exemples : `math`

```
>>> import math
```

Quelques exemples : math

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
 'erf','erfc', 'exp',  'expm1', 'fabs', 'factorial', 'floor',
 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma',
 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow',
 'radians',  'remainder', 'sin', 'sinh', 'sqrt', 'tan',
 'tanh', 'tau', 'trunc']
```

Quelques exemples : math

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
 'erf','erfc', 'exp',  'expm1', 'fabs', 'factorial', 'floor',
 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma',
 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow',
 'radians',  'remainder', 'sin', 'sinh', 'sqrt', 'tan',
 'tanh', 'tau', 'trunc']
>>> math.pi
3.141592653589793
```

Quelques exemples : math

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
 'erf','erfc', 'exp',  'expm1', 'fabs', 'factorial', 'floor',
 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma',
 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow',
 'radians',  'remainder', 'sin', 'sinh', 'sqrt', 'tan',
 'tanh', 'tau', 'trunc']
>>> math.pi
3.141592653589793
>>> math.cos(math.pi/4)
0.7071067811865476
```

Quelques exemples : math

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
 'erf','erfc', 'exp',  'expm1', 'fabs', 'factorial', 'floor',
 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma',
 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow',
 'radians',  'remainder', 'sin', 'sinh', 'sqrt', 'tan',
 'tanh', 'tau', 'trunc']
>>> math.pi
3.141592653589793
>>> math.cos(math.pi/4)
0.7071067811865476
>>> math.factorial(5)
120
```

Quelques exemples : `random`

```
>>> import random
```

Quelques exemples : random

```
>>> import random
>>> random.random() #Random float: 0.0 <= x < 1.0
0.37444887175646646
```

Quelques exemples : random

```
>>> import random
>>> random.random() #Random float: 0.0 <= x < 1.0
0.37444887175646646
>>> random.uniform(2.5, 10.0) #Random float: 2.5<=x<10.0
3.180014607311752 3
```

Quelques exemples : random

```
>>> import random
>>> random.random() #Random float: 0.0 <= x < 1.0
0.37444887175646646
>>> random.uniform(2.5, 10.0) #Random float: 2.5<=x<10.0
3.180014607311752 3
>>> random.randrange(10)  #Int entre 0 et 9
7
```

Quelques exemples : random

```
>>> import random
>>> random.random() #Random float: 0.0 <= x < 1.0
0.37444887175646646
>>> random.uniform(2.5, 10.0) #Random float: 2.5<=x<10.0
3.180014607311752 3
>>> random.randrange(10)  #Int entre 0 et 9
7
>>> random.randrange(0, 101, 2) #Entier pair entre 0 et 100
26
```

Quelques exemples : random

```
>>> import random
>>> random.random() #Random float: 0.0 <= x < 1.0
0.37444887175646646
>>> random.uniform(2.5, 10.0) #Random float: 2.5<=x<10.0
3.180014607311752 3
>>> random.randrange(10)  #Int entre 0 et 9
7
>>> random.randrange(0, 101, 2) #Entier pair entre 0 et 100
26
#Choix aléatoire d'un élément dans une séquence
>>> random.choice(['win', 'lose', 'draw'])
'draw'
```

Quelques exemples : random

```
>>> import random
>>> random.random() #Random float: 0.0 <= x < 1.0
0.37444887175646646
>>> random.uniform(2.5, 10.0) #Random float: 2.5<=x<10.0
3.180014607311752 3
>>> random.randrange(10)  #Int entre 0 et 9
7
>>> random.randrange(0, 101, 2) #Entier pair entre 0 et 100
26
#Choix aléatoire d'un élément dans une séquence
>>> random.choice(['win', 'lose', 'draw'])
'draw'
#4 éléments d'une séquence
>>> random.sample([10, 20, 30, 40, 50], 4)
[40, 10, 50, 30]
```

Quelques exemples : random

```
>>> import random
>>> random.random() #Random float: 0.0 <= x < 1.0
0.37444887175646646
>>> random.uniform(2.5, 10.0) #Random float: 2.5<=x<10.0
3.180014607311752 3
>>> random.randrange(10)  #Int entre 0 et 9
7
>>> random.randrange(0, 101, 2) #Entier pair entre 0 et 100
26
#Choix aléatoire d'un élément dans une séquence
>>> random.choice(['win', 'lose', 'draw'])
'draw'
#4 éléments d'une séquence
>>> random.sample([10, 20, 30, 40, 50], 4)
[40, 10, 50, 30]
>>> l = [1, 2, 3, 4]
>>> random.shuffle(l) #Mélange d'une liste
>>> l
[2, 3, 1, 4]
```

Quelques exemples : fractions

```
>>> from fractions import Fraction
```

Quelques exemples : fractions

```
>>> from fractions import Fraction
>>> Fraction(16, -10)
Fraction(-8, 5)
```

Quelques exemples : `fractions`

```
>>> from fractions import Fraction
>>> Fraction(16, -10)
Fraction(-8, 5)
>>> Fraction(123)
Fraction(123, 1)
```

Quelques exemples : fractions

```
>>> from fractions import Fraction
>>> Fraction(16, -10)
Fraction(-8, 5)
>>> Fraction(123)
Fraction(123, 1)
>>> Fraction(2.25)
Fraction(9, 4)
```

Quelques exemples : fractions

```
>>> from fractions import Fraction
>>> Fraction(16, -10)
Fraction(-8, 5)
>>> Fraction(123)
Fraction(123, 1)
>>> Fraction(2.25)
Fraction(9, 4)
>>> Fraction('7e-6')
Fraction(7, 1000000)
```

Quelques exemples : fractions

```
>>> from fractions import Fraction
>>> Fraction(16, -10)
Fraction(-8, 5)
>>> Fraction(123)
Fraction(123, 1)
>>> Fraction(2.25)
Fraction(9, 4)
>>> Fraction('7e-6')
Fraction(7, 1000000)
>>> Fraction('-.125')
Fraction(-1, 8)
```

Quelques exemples : fractions

```
>>> from fractions import Fraction
>>> Fraction(16, -10)
Fraction(-8, 5)
>>> Fraction(123)
Fraction(123, 1)
>>> Fraction(2.25)
Fraction(9, 4)
>>> Fraction('7e-6')
Fraction(7, 1000000)
>>> Fraction('-.125')
Fraction(-1, 8)
>>> Fraction(-.125)
Fraction(-1, 8)
```

Exemple : aire et périmètre d'un triangle

- Ecrire une fonction qui calcule le périmètre et l'aire d'un triangle quelconque dont l'utilisateur fournit les 3 côtés. Rappel, avec d la longueur du demi-périmètre et a, b et c celles des 3 côtés :

$$S = \sqrt{d(d-a)(d-b)(d-c)}$$

Exemple : aire et périmètre d'un triangle

```
from math import sqrt

def perimetre(a, b, c) :
    """Float x Float x Float --> Float
    Calcule le périmètre d'un triangle défini par a, b, et c"""
    return a + b + c

def aire(a, b, c):
    """Float x Float x Float --> Float
    Calcule l'aire d'un triangle défini par a, b, et c"""
    d = perimetre(a, b, c)/2
    return sqrt(d * (d-a) * (d-b) * (d-c))

a = float(input("Longueur du premier côté : "))
b = float(input("Longueur du deuxième côté : "))
c = float(input("Longueur du troisième côté : "))

print("Le triangle de longueur " + str(a) + ",", end = " ")
print(str(b) + " et " + str(c) + " a pour périmètre", end = " ")
print(perimetre(a, b, c), "et pour aire", aire(a,b,c))
```

Dessiner avec `turtle`

Module `turtle`

Module `turtle`

Le module `turtle` permet de réaliser très simplement des dessins géométriques.

Module turtle

Module turtle

Le module turtle permet de réaliser très simplement des dessins géométriques.

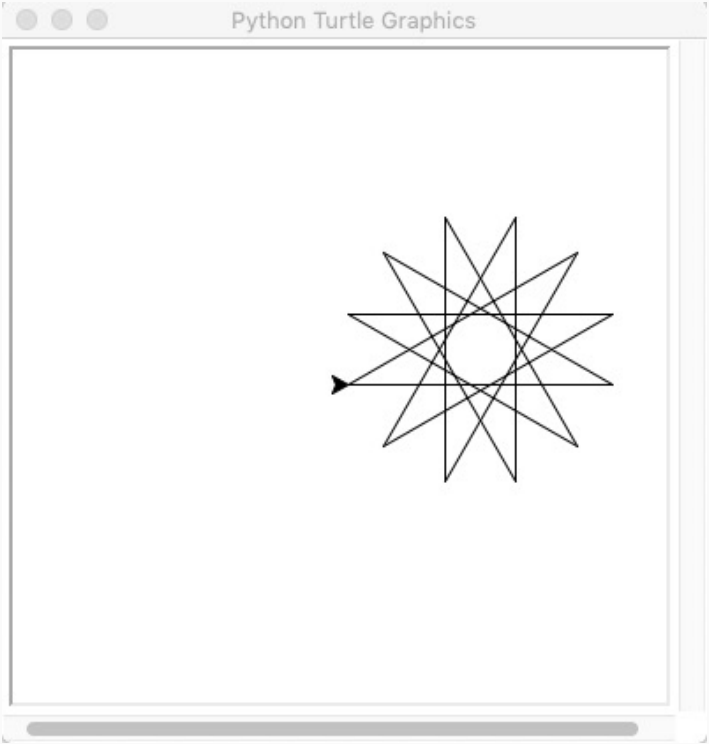
```
>>> import turtle
>>> a = 0
>>> while a<12 :
...     a = a+1
...     turtle.forward(150)
...     turtle.left(150)
...
```

Module `turtle`

Module `turtle`

Le module `turtle` permet de réaliser très simplement des dessins géométriques.

```
>>> import turtle
>>> a = 0
>>> while a<12 :
...     a = a+1
...     turtle.forward(150)
...     turtle.left(150)
...
```



Quelques fonctions du module `turtle`

- `reset()` : on efface, et on recommence
- `goto(x, y)` : aller à l'endroit des coordonnées `x, y`
- `forward(distance)` : avancer d'une distance donnée
- `backward(distance)` : reculer d'une distance donnée
- `left(angle)` : tourner à gauche d'un angle donné (en degrés)
- `right(angle)` : tourner à droite d'un angle donné (en degrés)
- `up()` : relever le crayon (se déplacer sans dessiner)
- `down()` : abaisser le crayon
- `pensize(épaisseur)` : épaisseur du trait de crayon
- `color(couleur)` : couleur du trait
- `write(texte)` : `texte` est une chaine de caractères

Flocon de neige avec `turtle` (1)

```
import turtle as tu
```

Flocon de neige avec `turtle` (1)

```
import turtle as tu

def formev():
    """None --> None. Procédure affichant une forme en V"""
    tu.right(25)
    tu.forward(50)
    tu.backward(50)
    tu.left(50)
    tu.forward(50)
    tu.backward(50)
    tu.right(25)
```

Flocon de neige avec `turtle` (1)

```
import turtle as tu

def formev():
    """None --> None. Procédure affichant une forme en V"""
    tu.right(25)
    tu.forward(50)
    tu.backward(50)
    tu.left(50)
    tu.forward(50)
    tu.backward(50)
    tu.right(25)

def brancheFlocon():
    """None --> None. Dessine la branche d'un flocon"""
    for x in range(4):
        tu.forward(30)
        formev()
    tu.backward(120)
```

Flocon de neige avec `turtle` (2)

```
def flocon():
    """None --> None. Dessine un flocon"""
    for x in range(6):
        brancheFlocon()
        tu.right(60)
```

Flocon de neige avec `turtle` (2)

```
def flocon():
    """None --> None. Dessine un flocon"""
    for x in range(6):
        brancheFlocon()
        tu.right(60)

tu.Screen().bgcolor("brown") #Couleur de fond de la fenêtre
```

Flocon de neige avec `turtle` (2)

```
def flocon():
    """None --> None. Dessine un flocon"""
    for x in range(6):
        brancheFlocon()
        tu.right(60)

tu.Screen().bgcolor("brown") #Couleur de fond de la fenêtre
tu.speed(5) #Vitesse de déplacement de la tortue
```

Flocon de neige avec `turtle` (2)

```
def flocon():
    """None --> None. Dessine un flocon"""
    for x in range(6):
        brancheFlocon()
        tu.right(60)

tu.Screen().bgcolor("brown") #Couleur de fond de la fenêtre
tu.speed(5) #Vitesse de déplacement de la tortue
tu.pencolor("white") #Couleur du trait
```

Flocon de neige avec `turtle` (2)

```
def flocon():
    """None --> None. Dessine un flocon"""
    for x in range(6):
        brancheFlocon()
        tu.right(60)

tu.Screen().bgcolor("brown") #Couleur de fond de la fenêtre
tu.speed(5) #Vitesse de déplacement de la tortue
tu.pencolor("white") #Couleur du trait
tu.pensize(6)#Epaisseur du trait
```

Flocon de neige avec `turtle` (2)

```
def flocon():
    """None --> None. Dessine un flocon"""
    for x in range(6):
        brancheFlocon()
        tu.right(60)

tu.Screen().bgcolor("brown") #Couleur de fond de la fenêtre
tu.speed(5) #Vitesse de déplacement de la tortue
tu.pencolor("white") #Couleur du trait
tu.pensize(6)#Epaisseur du trait

flocon()
```

Flocon de neige avec `turtle` (2)

```
def flocon():
    """None --> None. Dessine un flocon"""
    for x in range(6):
        brancheFlocon()
        tu.right(60)

tu.Screen().bgcolor("brown") #Couleur de fond de la fenêtre
tu.speed(5) #Vitesse de déplacement de la tortue
tu.pencolor("white") #Couleur du trait
tu.pensize(6)#Epaisseur du trait

flocon()
tu.exitonclick() #Pour que la fenetre ne se ferme pas seule
```

Pour conclure

Résumé du cours

Aujourd'hui, on a vu

- Ce qu'est un module de fonction
- Comment en créer un, et comment l'importer
- Comment utiliser un module de fonction pour abstraire des données
- Quelques modules courants
- Comment dessiner avec `turtle`