Stage LIESSE Python/BD

Structures de données et algorithmique

Ahmed El Rheddane (Ahmed.El-Rheddane@imag.fr)

Ensimag, Grenoble INP

Octobre 2013

Sommaire

Petit échauffement

- Combien y a-t-il de zéros dans 2013 factorielle ?
- Faut-il encore recoder la fonction factorielle ?
  - Non, on peut la réutiliser si on l'a déjà dans un fichier .py !
- S'assurer que le fichier à importer est bien dans le dossier courant
- Deux façons de faire :
  - Importer un élément spécifique :
    - `from` *`nom-du-fichier-sans-py`* `import` *`factorielle`*
    - Faire directement appel à `factorielle(2013)`
  - Importer tous les éléments d'un *module*
    - `import` *`nom-du-module`*
    - Pour chaque appel à un élément : *`nom-du-module.élément`*
- Outre nos fichiers, Python inclut de base des modules qu'on peut importer
  - Entre autres, un module `math` contenant une fonction `factorial`

Revenons à l'exercice

- Indice supplémentaire :
  - Utiliser la fonction `count` des chaînes de caractères
- Réponse :

```
>>> from math import factorial
>>> str(factorial(2013)).count("0")
1032
```

- De `int` à `long`, des entiers infinis !

```
>>> import sys
>>> sys.maxint
9223372036854775807
>>> sys.maxint + 1
9223372036854775808L
```

- Depuis Python 3, il n'y a plus de long, que des int.

Toujours autour des imports

- Créer un fichier *salutations.py* et définissez-y deux fonctions, l'une affichant "Bonjour", l'autre "Bonsoir"
- Créer un deuxième fichier, importer le module *salutations* et appeler ses deux fonctions.

Types en Python

- Toute expression Python a un type :

```
>>> a = [0, 1, 1.]
>>> type(a)
<type 'list'>
>>> type(a[1] + a[2])
<type 'float'>
```

- Même une fonction qui ne retourne pas de valeur !

```
>>> def affichage():
...     print "Bonjour"
...
>>> type(affichage())
Bonjour
<type 'NoneType'>
```

Références en Python (1)

- Considérons les deux listes :

```
>>> a = [0, 1, 1]
>>> b = [0, 1, 1]
```

- a et b sont-elles égales ?

```
>>> a == b
True
```

- Pourtant, `a` et `b` *référencent* deux *objets* différents.

```
>>> a[0] = -1
>>> b
[0, 1, 1]
```

Références en Python (2)

- Pour vérifier l'égalité des références, on utilise `is`.

```
>>> a = [0, 1, 1]
>>> b = [0, 1, 1]
>>> a is b
False
>>> a = b
>>> a is b
True
```

- Maintenant que `a` et `b` désignent le même objet :

```
>>> a[0] = -1
>>> b
[-1, 1, 1]
```

Références en Python (3)

- Considérons cet exemple :

```
>>> m = [[0]*2]*2
>>> m
[[0, 0], [0, 0]]
```

- Si, maintenant, on assignait la case (0,0) :

```
>>> m[0][0] = 1
>>> m
[[1, 0], [1, 0]]
```

  - Comment ça se fait ?
- Python va d'abord évaluer l'expression [[0]*2], et comme c'est une liste va utiliser la référence du résultat pour l'opération suivante

Structures de données (2)

- Les ensembles :
  - Des listes non ordonnées.
  - Ne contiennent pas de doublons.

```
>>> set(["zero", 1, 1, "deux"])
set([1, 'zero', 'deux'])
```

- Les dictionnaires
  - Associent des valeurs à des clés.
  - les clés doivent être de type non modifiable.

```
>>> d = {}
>>> d["cle1"] = "valeur1"
>>> d["cle1"]
'valeur1'
>>> "cle2" in d
False
```

Ex. 1 : Fibonacci itérative

- Implémentez une fonction `fib1(n)` itérative qui retourne pour chaque n, F_n tel que :
  - $F_0 = 0$.
  - $F_1 = 1$.
  - $F_n = F_{n-1} + F_{n-2}$ pour tout $n > 1$.
- Indices :
  - Traitez les cas particuliers à part.
  - Calculez les F_2, F_3, ..., F_n successivement.
- On dit que le coût de cette fonction est *linéaire*.

Ex. 3 : Récursivité améliorée

- Il se trouve que :
  - $F_{2n-1} = F_n^2 + F_{n-1}^2$.
  - $F_{2n} = (2F_{n-1} + F_n)F_n$.
- Utilisez ces formules pour implémenter `fib3(n)`, une version récursive améliorée de Fibonacci.
- Indices :
  - L'opérateur modulo en Python est `%`.
- En quoi cette version est-elle meilleure que `fib2(n)` ?

Structures de données (1)

- Les listes :
  - Peuvent contenir des éléments de types différents.
  - Peuvent être modifiées.

```
>>> l = [0, 1, 1., "deux"]
>>> l.append(True)
>>> l
[0, 1, 1., 'deux', True]
```

- Les tuples :
  - Ne peuvent être modifiés.
  - Utiles pour les affectations.

```
>>> (a, b, c) = (0, 1, 1)
```

Fonctions récursives

- Fonction récursive : fonction qui se rappelle elle-même.
- Attention à la condition d'arrêt !

```
def fact(n):
    if n <= 1:
        # condition d'arret
        return 1
    else:
        # appel recursif
        return n * fact(n - 1)
```

Ex. 2 : Fibonacci récursive

- Implémentez maintenant la fonction `fib2(n)`, la version récursive de `fib1(n)`.
- Indice :
  - N'oubliez pas les conditions d'arrêts !
- Combien de fois appelle-t-on *fib2*(0) pour calculer *fib2*(n) ?
  - On dit que le coût de cette fonction est *exponentiel*.

Ex. 4 : Fibonacci ultime !

- Au tour de la mémoïsation :
  - sauvegarder les résultats (dans un dictionnaire), afin d'éviter les calculs redondants.
- Implémentez la fonction `fib4(n, dico)`, sur la base de `fib3(n)` et du principe de mémoïsation.
- Indices :
  - Avant de retourner un résultat, pensez à l'insérer dans le dictionnaire.
  - Avant de calculer un résultat, vérifiez s'il n'est pas déjà dans le dictionnaire.
- Cette fonction est de coût *logarithmique*, coût optimal pour le calcul de Fibonacci.

Ex. 5 : Période de Pisano

- La suite de Fibonacci modulo n est périodique, on appelle période de Pisano sa période.
  - Vous pouvez, au choix, admettre ce résultat ou le démontrer.
- Implémentez une fonction `pi(n)` qui, retourne pour chaque n, la période de Pisano correspondante.
- Indices :
  - Calculer les nombre de Fibonacci à la suite.
  - Quelle est la version la plus appropriée ?
    - ⋆ Itérative !
  - Comment détecter que m est une période ?
    - ⋆ $F_m = 0$ et $F_{m+1} = 1$.
- Pensez au coût de votre programme.

Ex. Autour des nombres premiers

- Pour un `n` donné, écrivez une fonction `estpremier(n)` qui vérifie la primalité de `n`.
  - Indice : utilisez la fonction `sqrt` du module `math`.
- Écrivez une fonction qui retourne les `n` premiers nombres premiers.
  - Indice : vous pouvez sauvegarder les nombres premiers trouvés dans une liste, et les utilister pour vérifier la primalité des suivants.