

TP Noté Python – M1

Suite de nombres entiers

November 15, 2010

Pour la petite histoire ... *L'encyclopédie en ligne des suites de nombres entiers* (originellement en anglais *On-Line Encyclopedia of Integer Sequences*, couramment abrégé sous le sigle OEIS) est un site web permettant d'effectuer gratuitement des recherches parmi une base de données de suites d'entiers présentant un intérêt mathématique ou parfois simplement ludique. Dans cette forme et cette présentation, c'est la plus grande du monde (en 2008). Elle est consultée des milliers de fois chaque jour. L'OEIS est probablement la principale référence dans le domaine des suites d'entiers, pour les mathématiciens professionnels et amateurs, pour lesquels elle représente une ressource d'une très grande richesse. L'OEIS est une base de données qui contient plus de 178 436 suites au 24 août 2010, chacune se voyant attribuée un numéro de série. Elle est entièrement accessible par moteur de recherche : on peut rechercher une suite par sous-suite, par mot clé, ou par numéro de série. Chaque entrée propose les premiers termes de chaque suite, une ou des définitions, des références à des suites liées ou analogues, les motivations mathématiques, des liens vers la littérature, etc. Une capture du site en ligne est présentée Fig 1.

Le but de cet exercice est de développer un module python élémentaire (`intseqdb`) permettant de créer des séquences d'entiers et de les stocker dans une base de données simple, cette base de données pouvant être interrogée (*"Quelles sont les séquences de la base de données qui commencent par telle ou telle suite d'entiers ?"*). Il est important de noter dès à présent que les séquences seront définies par générateur, *i.e.*, chaque séquence utilisera un générateur pour produire ses valeurs (elle pourra donc générer un nombre arbitraire de valeurs) et il sera donc à la charge de l'appelant de déterminer combien d'éléments de la séquence doivent être générés.

Pour illustrer notre propos, utilisons notre module `intseqsb` pour définir quelques séquences d'entiers (les puissances de 1, de 2, de 4 et les nombres de Catalan). Ces séquences sont ensuite ajoutées à notre base de données qui peut être interrogée pour chercher toutes les séquences commençant par une suite d'entiers donnée.

```
import intseqdb

# a function that return a function defining a generator
# for the Catalan numbers: 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, ...
def catalan_gen():
    def _catalan_gen():
        i = 0
        val = 1
        while True:
            yield val
            val = val * (2 * (2*i + 1)) / (i + 2)
            i = i + 1
    # only return the function defining the generator
    return _catalan_gen
```

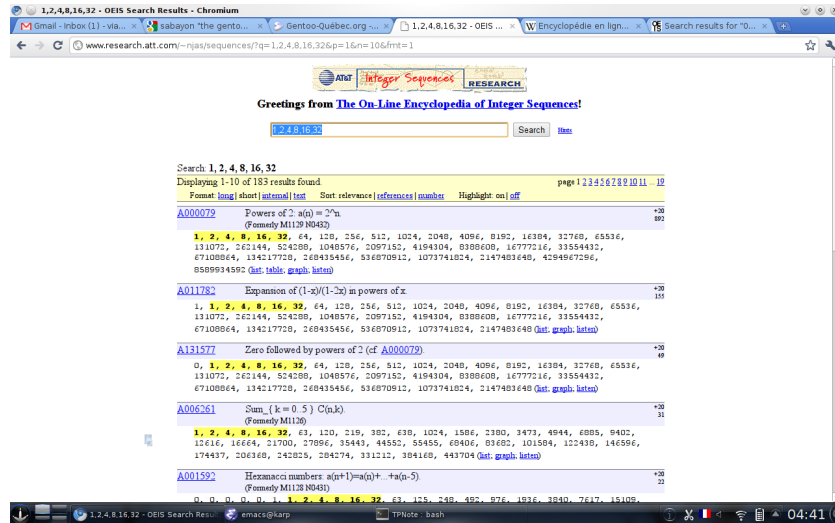


FIGURE 1 – La page d’accueil de l’encyclopédie en ligne des suites de nombres entiers (OEIS) (<http://www.research.att.com/njas/sequences/>).

```
# a function that return a function defining a generator
# for the powers of n: n^0, n^1, n^2, n^3, ...
def power_gen(n):
    def _power_gen():
        val = 1
        while True:
            yield val
            val = val * n
    # only return the function defining the generator
    return _power_gen

# create a database (only print the 10 first elements of each sequence)
db = intseqdb.Database(10)
# add some integer sequences into the database
db.add_sequence(intseqdb.Sequence("Powers of 1", power_gen(1)))
db.add_sequence(intseqdb.Sequence("Powers of 2", power_gen(2)))
db.add_sequence(intseqdb.Sequence("Powers of 4", power_gen(4)))
db.add_sequence(intseqdb.Sequence("Catalan numbers", catalan_gen()))
# print everything, i.e., all sequences stored in the database
print '*****'
print "number of sequences: %d" % len(db)
for sequence in db:
    print sequence
# search and print all sequences that begin as 1, 1
print '*****'
print 'all sequences that begin as 1, 1'
for sequence in db.search([1, 1]):
    print sequence
# search and print all sequences that begin as 1, 1, 2
print '*****'
```

```

print 'all sequences that begin as 1, 1, 2'
for sequence in db.search([1, 1, 2]):
    print sequence
# search and print all sequences that begin as 1, 1, 2, 3
print '*****'
print 'all sequences that begin as 1, 1, 2, 3'
for sequence in db.search([1, 1, 2, 3]):
    print sequence
print '*****'

```

La sortie de ce programme est la suivante (vous remarquerez que les 10 premières valeurs de chaque séquence sont affichées, c'est le rôle de l'argument dans `db = Database(10)`).

```

krtek@karp $ python test_intseqdb.py
*****
number of sequences: 4
Powers of 1 : 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
Powers of 2 : 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, ...
Powers of 4 : 1, 4, 16, 64, 256, 1024, 4096, 16384, 65536, 262144, ...
Catalan numbers : 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, ...
*****
all sequences that begin as 1, 1
Powers of 1 : 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
Catalan numbers : 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, ...
*****
all sequences that begin as 1, 1, 2
Catalan numbers : 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, ...
*****
all sequences that begin as 1, 1, 2, 3
*****
krtek@karp $

```

Question 1

Complétez le squelette de la classe `Sequence` du module `intseqdb`.

```

class Sequence(object):
    "A integer sequence"
    def __init__(self, description, generator, rep_len = 5):
        """create an integer sequence from a string description,
        a generator describing its elements, and an optional integer
        specifying how many elements are printed in a string representation
        (default is 5)"""
        # A completer
    def set_rep_len(self, rep_len):
        """setup how many elements of this integer sequence are printed in
        a string representation?"""
        # A completer
    def __str__(self):
        """string representation of an integer sequence: description +
        the first rep_len elements"""
        # A completer

```

```
def __iter__(self):
    """return a generator to iterate over the elements of this
    integer sequence"""
    # A completer
```

La fonction take

```
import itertools as it
def take(n, iterable):
    "Return first n items of the iterable as a list"
    return list(it.islice(iterable, n))
```

est fournie pour vous aider, il n'est cependant pas obligatoire de l'utiliser. Il est par contre **impératif** que votre implémentation produise le même résultat que celui présenté plus haut.

Question 2

Complétez le squelette de la classe Database du module intseqdb.

```
class Database(object):
    "A database to store and query integer sequences"
    def __init__(self, rep_len):
        """create an empty database (for each integer sequence stored in the
        database, only the first rep_len elements will be printed on request)"""
        # A completer
    def __len__(self):
        "return the number of integer sequences in the database"
        # A completer
    def __iter__(self):
        """return a generator to iterate over this database, i.e., a generator
        for all the integer sequences stored in the database"""
        # A completer
    def add_sequence(self, sequence):
        "add an integer sequence (object Sequence) to the database"
        # A completer
    def search(self, ints):
        """return a generator to iterate over all integer sequences that
        begin as ints. ints must be an iterable of fixed length (not an
        infinite generator)"""
        # A completer
```

Il est **impératif** que votre implémentation produise le même résultat que celui présenté plus haut, et que la méthode Database.search retourne un générateur.

Question 3

Proposer une implémentation d'une méthode Database.search_seq(self, sequence, cmp_len) qui prend en argument une instance de la classe Sequence et un entier. Cette méthode retourne un générateur sur toutes les séquences présentes dans la base de données pour lesquelles il y a concordance avec les cmp_len premiers entiers de la séquence sequence passée en argument (notez qu'il est nécessaire de préciser cmp_len puisque les séquences génèrent un nombre arbitraire de valeurs).