CM3: dictionnaires et récursion

Programmation 2: L2 mathématique

Florian Bridoux

October 18, 2019

Récursion

```
>>> def ma_fonction(n):
...     print("niveau:",n)
...     if n == 0:
...         print("fin")
...     else:
...         ma_fonction(n-1)
...
>>> ma_fonction(5)
niveau: 5
niveau: 4
niveau: 3
niveau: 2
niveau: 1
niveau: 0
fin
```

Récursion

```
>>> def recursive():
...     recursive()
...
>>> recursive()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in recursive
  File "<stdin>", line 2, in recursive
  File "<stdin>", line 2, in recursive
  [Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
```

Récursion

```
>>> def factorielle (n) :
...         if n <= 1 :
...             return 1                        # cas terminal
...         return n * factorielle (n-1)
...
>>> factorielle(5)
120
```

Récursion

Exercice: Observez la fonction suivant:

```
def a(b):
    if b == 0:
        return 0
    elif b % 5 == 0:
        return a(b-1) + 1
    else:
        return a(b-1)
```

1. Que vaut $a(0)$? $a(1)$? $a(4)$? $a(5)$? $a(11)$?
2. Que fait cette fonction?

La suite de Fibonacci

Exercice:

1. $F_1 = 1$, $F_2 = 1$ et $\forall i > 2,\ F_i = F_{i-1} + F_{i-2}$
2. Écrire une fonction récursive $fibo(n)$ qui renvoit le n-ieme terme de la suite de fibonacci.

La suite de Fibonacci

Exercice:

1. $F_1 = 1$, $F_2 = 1$ et $\forall i > 2,\ F_i = F_{i-1} + F_{i-2}$
2. Écrire une fonction récursive *fibo*(n) qui renvoit le n-ieme terme de la suite de fibonacci.

Correction:

```
def fibo(n):
    if n <= 2:
        return 1
    else :
        return fibo(n-1) + fibo(n-2)
```

Dictionnaire

Un dictionnaire est une sorte de liste, qui associe des mots clés à des valeurs.

```
>>> notes = { "Pierre" : 12, "Sophie" : 15, "Manon":6 }
>>> type(notes)
<class 'dict'>
>>> notes["Pierre"]
12
>>> notes["Sophie"]
15
>>> notes["Manon"]
6
>>> notes["Pierre"] = 5
>>> notes["Pierre"]
5
```

Dictionnaire

```
>>> notes = { "Pierre" : 12, "Sophie" : 15, "Manon":6 }
>>> notes["Léon"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Léon'
>>> notes["Léon"] = 3
>>> notes["Léon"]
3
>>> notes
{'Pierre': 5, 'Sophie': 15, 'Manon': 6, 'Léon': 3}
```

Dictionnaire

```
>>> notes
{'Pierre': 5, 'Sophie': 15, 'Manon': 6, 'Léon': 3}
>>> print(notes.get("Max"))
None
>>> notes.get("Sophie")
15
>>> notes.update( {"Max":3,"Léon":5} )
>>> notes
{'Pierre': 5, 'Sophie': 15, 'Manon': 6, 'Léon': 5,
 'Max': 3}
```

Dictionnaire

```
>>> notes
{'Pierre': 5, 'Sophie': 15, 'Manon': 6, 'Léon': 5,
 'Max': 3}
>>> del notes["Max"]
>>> notes
{'Pierre': 5, 'Sophie': 15, 'Manon': 6, 'Léon': 5}
>>> len(notes)
4
>>> "Sophie" in notes
True
>>> "Max" in notes
False
>>> 5 in notes
False
```

Dictionnaire

```
>>> for nom in notes:
...     print(nom, notes[nom])
Pierre 5
Sophie 15
Manon 6
Léon 5
>>> for nom,note in notes.items():
...     print(nom,note)
...
Pierre 5
Sophie 15
Manon 6
Léon 5
```

Dictionnaire

```
>>> list(notes)
['Pierre', 'Sophie', 'Manon', 'Léon']
>>> tuple(notes)
('Pierre', 'Sophie', 'Manon', 'Léon')
>>> notes.keys()
dict_keys(['Pierre', 'Sophie', 'Manon', 'Léon'])
>>> notes.values()
dict_values([5, 15, 6, 5])
>>> list(notes.values())
[5, 15, 6, 5]
```

Dictionnaire

On peut mettre autre chose que des strings en clé :

```
>>> pipo = { 10 : "dix", 3.14 : "pi", True : "vrai", (1,2,3)
>>> pipo[10]
dix
>>> pipo[3.14]
pi
>>> pipo[True]
vrai
>>> pipo[(1,2,3)]
suite
>>> lst = [1,2,3]
```

Mais l'objet doit être imutable.

```
>>> d = {lst:3}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Dictionnaire

```
>>> d = {3:lst}
>>> d[3]
[1, 2, 3]
>>> d[3].append(4)
>>> lst
[1, 2, 3, 4]
>>> d[3]
[1, 2, 3, 4]
```

Dictionnaire

Exercice:

1. Écrire une fonction *histogramme* qui prend en paramètre une string *text* et renvoi un dictionnaire associant à chaque lettre son nombre d'occurrence dans *text*.

Dictionnaire

Exercice:

1. Écrire une fonction *histogramme* qui prend en paramètre une string *text* et renvoi un dictionnaire associant à chaque lettre son nombre d'occurrence dans *text*.

Correction:

```
def histogramme(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] += 1
    return d
```

La suite de Fibonacci

Exercice: Optimisez la fonction suivante à l'aide d'un dictionnaire:

```
def fibo(n):
    if n <= 2:
        return 1
    else :
        return fibo(n-1) + fibo(n-2)
```

La suite de Fibonacci

Exercice: Optimisez la fonction suivante à l'aide d'un dictionnaire:

```
def fibo(n):
    if n <= 2:
        return 1
    else :
        return fibo(n-1) + fibo(n-2)
```

Correction:

```
mon_dict = {}
def fibo(n):
    if n not in mon_dict:
        if n <= 1:
            mon_dict[n] = 1
        else :
            mon_dict[n] = fibo(n-1) + fibo(n-2)
    return mon_dict[n]
```