Python
Les décorateurs

Nicolas Delestre

INSA
ROUEN NORMANDIE

Précisions sur les fonctions 1 / 3

Objets retournés

- Une fonction retourne `None` (par exemple si pas de `return`) ou un objet
- Une fonction possède des variables locales qui référencent des objets
- Des invocations d'une même fonction peuvent retourner des objets différents (au sens de l'identité)

```
1 def foo1(a):
2     interne = [a]
3     return interne
```

```
>>> a1=foo1(1)
>>> a2=foo1(1)
>>> a3=foo1(2)
>>> a1
[1]
>>> a2
[1]
>>> a3
[2]
```

```
>>> a1 is a2
False
>>> a1 is a3
False
>>> a2 is a3
False
```

Précisions sur les fonctions 2 / 3

Retourner une fonction (lambda)

- Une fonction est un objet
- Une fonction (principale) peut très bien retourner une fonction
- Chaque appel de le fonction (principale) peut retourner une fonction différente (au sens de l'identité)

```
1 def foo2(a):
2     interne = lambda : a
3     return interne
```

```
>>> b1=foo2(1)
>>> b2=foo2(1)
>>> b3=foo2(2)
>>> b1()
1
>>> b2()
1
>>> b3()
2
```

```
>>> b1 is b2
False
>>> b1 is b3
False
>>> b2 is b3
False
```

Précisions sur les fonctions 3 / 3

Retourner une fonction (interne)

- Une fonction interne est une variable locale
- Elle peut donc être retournée
- Chaque appel à la fonction principale retournera une fonction différente (au sens de l'identité)

```
1 def foo3(a):
2     def interne():
3         return a
4     return interne
```

```
>>> c1=foo3(1)
>>> c2=foo3(1)
>>> c3=foo3(2)
>>> c1()
1
>>> c2()
1
>>> c3()
2
```

```
>>> c1 is c2
False
>>> c1 is c3
False
>>> c2 is c3
False
```

Les décorateurs

Rappels

- Des instructions, commençant par un `@`, se trouvant juste avant une définition d'une fonction qui modifie le comportement de cette fonction
- Exemples : `@property`, `@staticmethod`, `@fixture`, etc.

Comment ?

- Développer une fonction qui prend une fonction en paramètre et qui retourne une fonction, par exemple :

```
@dec2                                    ⇒    def func(arg1, arg2, ...):
@dec1                                             pass
def func(arg1, arg2, ...):                 func = dec2(dec1(func))
    pass
```

Exemple issue de `https://www.python.org/dev/peps/pep-0318/`

Le décorateur « cache » 1 / 4

Objectif

Sauvegarder les résultats d'une fonction : lorsque l'on appelle une deuxième fois cette fonction avec les même paramètres, il n'y a pas de nouveau le calcul du résultat

Principe

- Attacher à la fonction un dictionnaire dont les clés sont les paramètres effectifs de la fonction et les valeurs les valeurs calculées
- À chaque appel de fonction, on vérifie si les paramètres effectifs sont une clé du dictionnaire, et dans ce cas on retourne la valeur associée. Sinon on calcule la valeur et on la sauvegarde dans le dictionnaire avant de la retourner

INSA ROUEN NORMANDIE

Code du décorateur cache

```
1  def cache(fnt_a_cacher):
2      def fnt_avec_cache(*args):
3          try:
4              fnt_avec_cache.le_cache
5          except AttributeError:
6              fnt_avec_cache.le_cache = {}
7          le_cache = fnt_avec_cache.le_cache
8          if args in le_cache.keys():
9              return le_cache[args]
10         else:
11             val = fnt_a_cacher(*args)
12             le_cache[args] = val
13             return val
14     return fnt_avec_cache
```

INSA ROUEN NORMANDIE

Le décorateur « cache » 3 / 4

cnp sans utilisation du décorateur cache

```
def cnp(n,p):
    if p == 1 or n == p:
        return 1
    else:
        return cnp(n-1, p-1) + cnp(n-1, p)
```

Sous ipython3

```
In [11]: %time cnp(30,20)
CPU times: user 4.59 s, sys: 4 ms, total: 4.59 s
Wall time: 4.59 s
Out[11]: 20030010

In [12]: %time cnp(31,20)
CPU times: user 12.1 s, sys: 0 ns, total: 12.1 s
Wall time: 12.1 s
Out[12]: 54627300
```

Le décorateur « cache » 4 / 4

cnp avec utilisation du décorateur cache

```
@cache
def cnp(n,p):
    if p == 1 or n == p:
        return 1
    else:
        return cnp(n-1, p-1) + cnp(n-1, p)
```

Sous ipython3

```
In [2]: %time cnp(30,20)
CPU times: user 870 us, sys: 0 ns, total: 870 us
Wall time: 978 us
Out[2]: 20030010

In [3]: %time cnp(300,200)
CPU times: user 33.5 ms, sys: 30 us, total: 33.6 ms
Wall time: 33.3 ms
Out[3]: 2772167642172376496522255684217603720186977337162432472442022438203170
885549339080
```