Python avancé

Gaël LE MIGNOT – Pilot Systems

Décembre 2006

Plan

Paramètres avancés

Généralités

- Passage de paramètres nommé et non nommé
- Les opérateurs * et **

Exemple

```
>>> def func(*args, **kwargs):
...     print args, kwargs
...
>>> func(1, "bleu", reponse=42)
(1, 'bleu') {'reponse': 42}
```

Les classes Kangourou

Exemple

```
>>> from kangourou import *
>>> k = Kangourou()
>>> k.jump()
Sansnom, un kangourou roux, saute (1)
>>> albert = KangourouBleu("Albert")
>>> albert.jump()
Albert, un kangourou bleu, saute (1)
>>> albert.jump()
Albert, un kangourou bleu, saute (2)
```

Les classes Kangourou (2)

Exemple

```
>>> from kangourou import *
>>> bebe = BebeKangourou()
>>> bebe.jump(5)
Sansnom, un kangourou roux, saute (1)
Sansnom, un kangourou roux, saute (2)
Sansnom, un kangourou roux, saute (3)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "kangourou.py", line 42, in jump
    raise RuntimeError, "%s est fatigué" % (self.na
RuntimeError: Sansnom est fatigué
```

Proxyfication manuelle

Exemple

```
>>> class LoggingKangourou(Kangourou):
...     def jump(self, *args, **kwargs):
...         print "-> jump,", args, kwargs
...         Kangourou.jump(self, *args, **kwargs)
...         print "<- jump"
>>> l = LoggingKangourou()
>>> l.jump(2)
-> jump, (2,) {}
Sansnom, un kangourou roux, saute (1)
Sansnom, un kangourou roux, saute (2)
<- jump
```

Map, filter et reduce

Généralités

- `map` applique une fonction sur tous les éléments d'une liste
- `filter` permet de filtrer une liste
- `reduce` permet de convertir une liste en un objet

Exemple

```
>>> l = [ "le", "kangourou", "vert" ]
>>> map(len, l)
[2, 9, 4]
>>> filter(lambda s: "e" in s, l)
['le', 'vert']
```

Exemple de reduce

Reduce

```
>>> def mul(a, b):
...     print "Multiply %d with %d" % (a, b)
...     return a * b
...
>>> reduce(mul, range(1,6))
Multiply 1 with 2
Multiply 2 with 3
Multiply 6 with 4
Multiply 24 with 5
120
```

Les listes comprehensions

Généralités

- permet de réaliser `map` et `filter`
- syntaxe: `[ expr for var in iterable if condition ]`

Exemples

```
>>> [ 2*i for i in range(5) ]
[0, 2, 4, 6, 8]
>>> l = [ "le", "kangourou", "vert" ]
>>> [ len(s) for s in l if "e" in s ]
[2, 4]
```

Boucles et modifications

Exemple

```
>>> l = range(10)
>>> for i in l:
...     if i % 3 == 0:
...         l.append(i)
...
```

Solutions

- utiliser une list comprehension
- utiliser `[:]`

Break, else

Solutions

- `break` et `continue` comme en C
- `else:` pour si aucun `break` n'a été fait

Exemple

```
>>> for i in l:
...     if i % 3 == 0:
...         print i
...         break
... else:
...     print "Non trouvé"
```

Python, un langage dynamique

Généralités

- Tout comportement, en Python, peut être reprogrammé
- On utilise pour celà des méthodes et attributs « magiques »
- Exemples: `__init__` et `__del__`

Les protocoles

- Un groupe de méthodes de customisation suivant un modèle de comportements
- Exemple: l'opérateur [], le comportement d'entier, ...
- Toujours respecter les protocoles, autant que possible

Accès aux attributs (1)

Version usuelle

- `__getattr__(self, key)`
- `__setattr__(self, key, value)`
- `__delattr__(self, key)`

Version avancées

- `__getattribute__(self, key)`
- Précautions d'usage

Accès aux attributs (2)

Exemple

```
>>> from kangourou import *
>>> from getattr import *
>>> k = Kangourou()
>>> k.jump()
Sansnom, un kangourou roux, saute (1)
>>> l = Logging(k)
>>> l.jump(1)
-> jump, (1,) {}
Sansnom, un kangourou roux, saute (2)
<- jump
```

Accès aux attributs (3)

Exemple (suite)

```
>>> l = Logging(BebeKangourou())
>>> l.jump(5)
-> jump, (5,) {}
Sansnom, un kangourou roux, saute (1)
Sansnom, un kangourou roux, saute (2)
Sansnom, un kangourou roux, saute (3)
<- jump
Traceback (most recent call last):
  [...]
RuntimeError: Sansnom est fatigué
```

Les propriétés

Principe

- Permet d'utiliser un accesseur de manière transparente
- Syntaxe: `attribute(getter, setter)`

Exemple

```
>>> c = Circle()
>>> c.radius = 2 ; c.area
12.566370614359172
>>> c.area = 4 ; c.radius
1.1283791670955126
```

Émuler des types de base

Les conteneurs

- `__getitem__`, `IndexError`, `KeyError`
- `__setitem__`, `__delitem__`
- `__contains__`, `__len__`
- Les méthodes normales: `insert`, ...
- La classe `DictMixin`

Les fonctions

- `__call__`

Les fonctions diverses

Comparaisons

- `__cmp__`
- Les comparaisons riches: `__lt__`, `__ne__`, ...
- `__nonzero__`

Conversion en chaîne

- `__str__`
- `__repr__`
- `__unicode__`

Les monkey patchs

Principe

- On peut modifier dynamiquement les méthodes des classes
- Usages: hotfix, ajout d'un hook, ...
- Il est aussi possible de modifier un seul objet

Exemple

```
>>> from kangourou import Kangourou
>>> k = Kangourou()
>>> Kangourou.jump = lambda self, n=1: n
>>> k.jump(2)
 2
```

Les itérateurs (1)

Principe

- La méthode `__iter__`
- Permet de ne pas générer de liste immense
- Permet les itérations infinies
- Permet une itération réentrante

Les objets itérateurs

- La méthode `__next__`
- `StopIteration`

Les itérateurs (2)

Exemple

```
>>> from fibo import Fibonacci
>>> f = Fibonacci(maxval = 6)
>>> for i in f:
...     print i
1
2
3
5
8
13
```

Les itérateurs (3)

Exemple

```
>>> f = Fibonacci()
>>> for i, j in zip(f, range(4)):
...     print i, j
...     if i == 3:
...         for i, j in zip(f, range(2)):
...             print " >", i, j
1 0
2 1
3 2
 > 1 0
 > 2 1
5 3
```

Les générateurs

Principe

- Une méthode plus simple d'avoir des itérateurs
- Appelé *continuations* en fonctionnel
- Réalisés via `yield`

Exemple

```
>>> from fibogen import Fibonacci
>>> f = Fibonacci()
>>> iter(f)
<generator object at 0xb7dd694c>
```

Les sous-fonctions (1)

Généralités

- Des fonctions peuvent contenir des fonctions
- Structure appelée *inner functions*

Les closures

- Principe des closures
- Les closures en Python
- Contourner via `dict`

Les sous-fonctions (2)

Exemple

```
>>> def func(a):
...     def inner(b):
...         return a + b
...     return inner
...
>>> f = func(40)
>>> f(2)
42
```

Les décorateurs (1)

Généralités

- Un décorateur permet d'enrober une fonction
- Utilités: log, protection, ...
- Un décorateur peut être n'importe quel *callable*
- Exemples: `staticmethod`, `classmethod`

Syntaxe

```
@decorator
def function(params):
    code
```

Les décorateurs (2)

Exemple

```
>>> def verbose(func):
...     def inner(*args, **kwargs):
...         print func.__name__
...         return func(*args, **kwargs)
...     return inner
>>> @verbose
... def func(a):
...     print a
>>> func(42)
func
42
```

Les décorateurs avancés

Généralités

- Un décorateur peut aussi prendre des paramètres
- Dans ce cas, on a un niveau d'appel en plus
- Les derniers kangourous...

Syntaxe

```
@decorator(params)
def function(params):
    code
```

Conclusion

URLs

- Python: `http://www.python.org`
- Slides: `http://www.pilotsystems.net`