INTRODUCTION AU LANGAGE PYTHON

C. Schlick

schlick@u-bordeaux.fr

CHAPITRE 5

Classes & Instances

Rappel : Structure du code Python

- **Code Python =** Ensemble de paquetages
  - Concrètement : *paquetage = dossier (disque)*
  - Symboliquement : *paquetage <=> livre*
- **Paquetage / Package =** Ensemble de modules
  - Concrètement : *module = fichier (disque)*
  - Symboliquement : *module <=> chapitre*
- **Module / Module =** Ensemble de blocs
  - Concrètement : *bloc = séquence de lignes*
  - Symboliquement : *bloc <=> paragraphe*

Rappel : Structure du code Python

- **Bloc / Block =** Ensemble d'instructions
  - Conc. : *instruction = séquence de caractères*
  - Symb. : *instruction <=> phrase grammaticale*
- **Instruction / Statement =** Entité élémentaire du langage, obtenue en combinant cinq types de symboles spécifiques :
  - Symboles rigides : ***mots réservés / keywords, délimiteurs / delimiters, opérateurs / operators***
  - Symboles flexibles : ***litéraux / literals, identificateurs / identifiers***

Expression ≠ Instruction

Instruction = Action à effectuer

- *Chaque instruction figurant dans le code va modifier l'état courant de l'automate (transition)*
  - Symb. : *instruction <=> phrase grammaticale*

Expression = Donnée à construire

- *Chaque expression figurant dans le code va rajouter une nouvelle donnée pour l'automate*
- *Une expression est une combinaison algébrique mêlant opérateurs, délimiteurs et identificateurs*
  - Symb. : *expression <=> groupe nominal*

Mots réservés / Keywords

Six familles de mots réservés : *chacune reliée à une catégorie spécifique d'instructions*

1. **Affectation :** `=  +=  -=  *=  /=  ...`
2. **Définition :** `class/def/lambda/global/nonlocal/[local]/with ... as/from ... import ... as`
3. **Répétition :** `while/for`
4. **Condition :** `if/elif/else/assert/try/except`
5. **Rupture :** `break/continue/return/yield/raise`
6. **Inaction :** `pass`

Délimiteurs / Delimiters

Huit familles de délimiteurs :

1. **Bloc :** `<head>':' <indented body> '\n'`
2. **Instruction :** `<statement> <';'|'\n'>`
3. **Commentaire :** `'#' <comment> '\n'`
4. **Séquence :** `<expression','> [expression',' ...]`
5. **Tuple :** `'(' <sequence> ')'`
6. **Liste :** `'[' <sequence> ']'`
7. **Ensemble & Dictionnaire :** `'{' <sequence> '}'`
8. **Chaîne :** `'...' | "..." | '''...''' | """..."""`

Opérateurs / Operators

Neuf familles d'opérateurs (par priorité) :

1. **Groupes :** `(...)  [...]  {...}`
2. **Attributs :** `x(y...)  x[y...]  x.y`
3. **Exposant :** `**`
4. **Opérateurs unaires :** `-x  +x  ~x`
5. **Opé. multiplicatifs :** `*  /  //  %  @`
6. **Opé. additifs :** `+  -`
7. **Opé. binaires :** `<<  >>  :  &  :  ^  :  |`
8. **Comparaisons :** `==  !=  <  <=  >  >=  is  is␣not  in  not␣in`
9. **Opé. booléens :** `not  :  and  :  or`

Identificateurs / Identifiers

- Un identificateur est un symbole arbitraire créé par un développeur pour nommer chacune des entités définies dans le code : *module, fonction, constante, variable, classe, objet, attribut…*
- ***Certaines contraintes tout de même :***
  - Un identificateur est une suite arbitraire de caractères pris parmi les 63 caractères suivants : `a b` … `z A B` … `Z 0` … `9 _`
  - Le 1er caractère ne peut pas être un chiffre, afin de ne pas confondre les identificateurs avec les symboles numériques

Espaces de noms / Namespaces

- Les espaces de noms sont des dictionnaires qui associent à chaque identificateur, la zone mémoire où est stockée l'entité associée, ainsi que le codage binaire utilisé pour cette entité
- Ces espaces sont définis hiérarchiquement :
  - au niveau de chaque fonction (***local level***)
  - au niveau de chaque objet (***object level***)
  - au niveau de chaque classe (***class level***)
  - au niveau de chaque module (***global level***)
  - au niveau de l'interpréteur (***builtin level***)

Donnée = Type + Valeur

Donnée = Séquence binaire + Codage

- *Toute donnée traitée par l'automate est constituée d'une séquence binaire (0 ou 1) de taille variable, complétée par la description du codage utilisé*

Codage <=> Type

- *Les données de même nature sont codées de manière identique, ce codage est appelé "* **Type** *" de la donnée*

Séquence binaire <=> Valeur

- *Une fois le codage spécifié, la séquence binaire associée à une donnée est définie de manière unique, et est appelée "* **Valeur** *" de la donnée*

Objet = Classe + Instance

- *Le vocabulaire change lorsqu'on se place dans le paradigme de la programmation par objets :*

Donnée <=> Objet

Type <=> Classe

Valeur <=> Instance

Pourquoi une terminologie différente ?

Procédural ≠ Objet

Paradigme procédural :

- *Chaque espace de noms va regrouper trois catégories d'identificateurs : les **constantes**, les **variables** et les **fonctions***
- *Les données sont stockées dans les constantes et les variables, alors que les actions sur ces données sont stockées dans les fonctions*
- ***Le paradigme procédural va privilégier les actions par rapport aux données** :*

  Code = ensemble de fonctions qui s'échangent des variables et des constantes

Procédural ≠ Objet

Paradigme objet :

- *Il n'y a que deux catégories d'identificateurs : les **instances** et les **classes***
- *Les données sont stockées dans des instances, alors que les actions à réaliser sur ces données vont être regroupées à l'intérieur des classes*
- ***Le paradigme objet va privilégier les données par rapport aux actions** :*

  Code = ensemble d'objets qui s'envoient des messages pour identifier les actions à réaliser

Procédural ≠ Objet

Paradigme procédural :

- *Le traitement des données peut être effectué :*
  - *Avec les opérateurs standards du langage*
  - *Avec un ensemble de fonctions prédéfinies*
  - *Avec des fonctions définies par l'utilisateur*

Paradigme objet :

- *Le traitement des données peut être effectué uniquement par des **méthodes** (i.e. fonctions de traitement spécifiques à la nature de la donnée concernée)*

Procédural ≠ Objet

Avantages du paradigme objet :

- *Regroupement en un seul endroit de l'ensemble des traitements possibles pour une donnée*
- *Possibilité de réutiliser un objet dans plusieurs applications sans réécriture du code associé*
- *Possibilité d'étendre les fonctionnalités d'un objet par un mécanisme de dérivation (cf. UML)*

Inconvénient du paradigme objet :

- *Une certaine lourdeur dans la mise en œuvre avec une phase de conception plus longue*

Introduction au langage Python

CHAPITRE 6

Mise en œuvre des objets en Python

Définition d'une classe

Classe : `class`

- *La définition de classe permet d'associer un identificateur avec un nouveau **type** de données*
- *Syntaxe à utiliser :*

```
class child(parent…):
  block
```

- *Les différentes classes sont réparties dans une structure arborescente ayant la classe minimale* `object` *comme racine*

Exemple d'utilisation

```
# classe de niveau 1
class A(object):
  x = 1
  def f(): return A.x + 1
# classe de niveau 2
class B(A):
  y = 2
  def g(): return B.x+B.y
# notation pointée
A.x, A.f() # --> 1, 2
B.x, B.f() # --> 1, 2
B.y, B.g() # --> 2, 3
```


Définition d'une classe

Classe : `class`

- *Chaque classe définit un espace de noms qui va regrouper **attributs** (= constantes & variables) et **méthodes** (= fonctions)*
- *L'ensemble des attributs associé à une classe est appelé **état** (**state**)*
- *L'ensemble des méthodes associé à une classe est appelé **comportement** (**behavior**)*

Exemple d'utilisation

```
# classe de niveau 1
class A(object):
  x = 1
  def f(): return A.x + 1
# classe de niveau 2
class B(A):
  y = 2
  def g(): return B.x+B.y
# notation pointée
A.x, A.f() # --> 1, 2
B.x, B.f() # --> 1, 2
B.y, B.g() # --> 2, 3
```


Définition d'une classe

Classe : `class`

- *Il existe de nombreux comportements standards pour une classe, qui sont activés simplement en définissant des méthodes dites **spéciales** avec un prototype prédéfini*
- *Ces méthodes spéciales se différencient par leur identificateur spécifique :*

  __name__

Méthodes spéciales :

- `__init__` : *initialisation*
- `__repr__` : *représentation*
- `__str__` : *conversion en* `str`
- `__len__` : *taille*
- `__contains__` : *inclusion*
- `__eq__` : *test* `==` *(equal)*
- `__le__` : *test* `<=` *(less or equal)*
- `__lt__` : *test* `<` *(less than)*
- `__add__` : *opérateur* `+`
- `__sub__` : *opérateur* `-`
- `__call__` : *opérateur* `()`
- `__getitem__` : *opérateur* `[]`

… et pleins d'autres !

Définition d'une classe

Classe : `class`

- *Les méthodes spéciales représentent la base du fonctionnement de Python*
- *La quasi totalité des instructions mettent en œuvre un **mécanisme de traduction automatique de code** par l'interpréteur*
- *Ce fonctionnement sera mis en œuvre pour toutes les classes créées par les développeurs*

Code utilisateur :

```
A = 1
B = A+1
C = str(A/B)
```

Code interpréteur :

```
int.__init__(A, 1)
int.__init__(B,
  int.__add__(A, 1))
str.__init__(C,
  float.__str__(
  int.__div__(A, B)))
```


Définition d'une instance

`name = class(…)`

- *La définition d'une instance (**instanciation**) permet d'associer un identificateur avec une nouvelle **valeur***
- *Comme pour les classes, les instances vont définir un espace de noms qui pourra soit contenir des attributs ou des méthodes*
- *L'accès à cet espace de noms se fera en utilisant la **notation pointée***

Exemple d'utilisation

```
class A(object):
  x = 1
  def __init__(self, y):
    self.y = y
# instanciation
a = A(0) ; b = A(1)
# lecture des attributs
a.x, a.y # --> 1, 0
b.x, b.y # --> 1, 1
# édition d'un attribut
a.y = 2  # d'instance
a.y, b.y # --> 2, 1
# édition d'un attribut
A.x = 3  # de classe
a.x, b.x # --> 3, 3
```


Définition d'une instance

`name = class(…)`

- *Un attribut défini au niveau de la classe, est commun à toutes les instances, et est appelé **attribut de classe** (son nom est alors préfixé par le nom de la classe)*
- *Un attribut défini au niveau d'une instance, est appelé **attribut d'instance** et n'est pas partagé par les autres instances de la classe (son nom est alors préfixé par le nom de l'instance)*

Exemple d'utilisation

```
class A(object):
  x = 1
  def __init__(self, y):
    self.y = y
# instanciation
a = A(0) ; b = A(1)
# lecture des attributs
a.x, a.y # --> 1, 0
b.x, b.y # --> 1, 1
# édition d'un attribut
a.y = 2  # d'instance
a.y, b.y # --> 2, 1
# édition d'un attribut
A.x = 3  # de classe
a.x, b.x # --> 3, 3
```


Définition d'une instance

`name = class(…)`

- *Toutes les méthodes sont communes à toutes les instances d'une classe*
- *Ce qui différencie une **méthode de classe** d'une **méthode d'instance**, est le fait que ces dernières ont systématiquement un 1er paramètre appelé `self`*
- *Dans le bloc d'instructions d'une classe, `self` correspond à une référence sur l'instance en cours*

Exemple d'utilisation

```
class A(object):
  x = 1
  def __init__(self, y):
    self.y = y
  def f() : return -A.x
  def g(self, x):
    return self.y + x
# instanciation et accès
a = A(2)
a.x, a.y # --> 1, 2
A.f() # --> -1
a.f() # --> Error
a.g(1) # --> 3
A.g(1) # --> Error
```