Module EA4 – Éléments d'Algorithmique II

micro-cours de Python

Dominique Poulalhon
dominique.poulalhon@liafa.univ-paris-diderot.fr

Université Paris Diderot
L2 Informatique
Année universitaire 2014-2015

un langage interprété

- lancement de l'interpréteur interactif, pratique pour faire de petits tests rapides : exécuter `python` ou `python3` (la version courante est la version 3)

```
>>> a = 2
>>> a *= 2
>>> a
4
>>> a *= 2
>>> a
8
```

- scripts dans des fichiers avec comme première ligne la référence de l'interpréteur, par exemple :

```
#!/usr/bin/python3
```

Présentation rapide

langage conçu pour être *lisible* et *épuré* :

- peu de constructions syntaxiques
- blocs marqués uniquement par l'indentation (qui devient donc cruciale)

langage de haut-niveau permettant l'écriture rapide de prototypes :

- pas de déclaration de variables
- typage dynamique
- types de base très souples et puissants
- nombreux modules spécialisés

langage multiparadigme : à la fois impératif, objet et fonctionnel

typage dynamique : déterminé à l'exécution

```
>>> a = 8
>>> type(a)
<class 'int'>
>>> a += 3.5
>>> type(a)
<class 'float'>
```

typage dynamique : déterminé à l'exécution

```
>>> a = 8
>>> type(a)
<class 'int'>
>>> a += 3.5
>>> type(a)
<class 'float'>
```

typage fort : on ne peut pas mélanger les types n'importe comment

```
>>> a = 8
>>> print("a vaut"+ a) ## provoque une erreur de type
```

Les types principaux

- `int` : entiers *de longueur non bornée*

```
>>> p = 2
>>> p **= p  # mettre p à la puissance p ==> p = 4
>>> p **= p  # ==> p = 256
>>> p **= p ; p
```

```
32317006071311007300714876688669951960441026697154840321303454275246551388678908931972014115229134636887179609218980194941195591504909210950881523864482831206308773673009960917501977503896521067960576383840675682767922186426197561618380943384761704705816458520363050428875758915410658086075523991239303855219143333896683424206849747865645694948561760353263220580778056593310261927084603141502585928641771167259436037184618573575983511523016459044036976132332872312271256847108202097251571017269313234696785425806566979350459972683529986382155251663894373355436021354332296046453184786049521481935558536110595962306 56
```

Les types principaux

- `int` : entiers *de longueur non bornée*
- `float` : flottants sur 64 bits
- `bool` : booléens `True` et `False`
- `str` : chaînes de caractères
- `list` : listes, non nécessairement homogènes
- `tuple` : k-uplets
- `set` : ensembles
- `dict` : dictionnaires (ou tables d'association)

Un type très utile : les listes

manipulables comme des tableaux et comme des listes chaînées

```
>>> l = [ 1, 3, 7, 'coucou', 4]
>>> l[2]
7
>>> l[1:4]
[3, 7, 'coucou']
>>> l.insert(2, "salut")
>>> l
[1, 3, 'salut', 7, 'coucou', 4]
>>> l.append(3)
>>> l
[1, 3, 'salut', 7, 'coucou', 4, 3]
>>> l.count(3)
2
```

en Python, les blocs sont définis uniquement par l'indentation

```
if test1 :
  # bloc d'instructions
elif test2 :
  # bloc d'instructions
elif test3 :
  # bloc d'instructions
...
else :
  # bloc d'instructions
```

en Python, les blocs sont définis uniquement par l'indentation

```
while test :
  # bloc d'instructions
```

```
for elt in sequence_iterable : # liste, tuple, chaîne...
  # bloc d'instructions
```

en Python, les blocs sont définis uniquement par l'indentation

```
while test :
  # bloc d'instructions
```

```
for elt in sequence_iterable : # liste, tuple, chaîne...
  # bloc d'instructions
```

En particulier, pour itérer sur une plage d'entiers « comme en Java » :

```
>>> for i in range(1, 4) :
...     print('cou' * i)
...
cou
coucou
coucoucou
```

en Python, les blocs sont définis uniquement par l'indentation

```
def ma_fonction (liste_des_parametres) :
  # bloc d'instructions
```

```
>>> def truc(x) : # pas de déclaration de type
...     return x + x # ou 2 * x, ou x * 2
...
>>> truc(3)
6
>>> truc('cou') # magique!
'coucou'
```