Labo d'introduction à l'informatique

pour les mathématiques

Yann Thorimbert

UNIVERSITÉ DE GENÈVE
CENTRE UNIVERSITAIRE D'INFORMATIQUE

Semaine 5

Déballages de séquences
Séquences en intension

Partie 1 - Déballage de séquences

Accès aux éléments d'une séquence (méthode classique)

Méthode déjà connue :

```
texte = "Jean Dupont A8x4592"
infos = texte.split()
prenom = infos[0]
nom = infos[1]
identifiant = infos[2]
```

Accès aux éléments d'une séquence (déballage)

Raccourci syntaxique:

```
texte = "Jean Dupont A8x4592"
infos = texte.split()
prenom, nom, identifiant = infos

# Fonctionne à condition que le nombre de variables corresponde !
# Permet de simplifier en : prenom, nom, identifiant = texte.split( )
```

Déballage de tuples

Fonctionne sur les listes aussi bien que les tuples :

```
infos_geneve = ("Suisse", 530_000, 400, "français")
pays, population, altitude, langue = infos_geneve
```

Déballage et boucles

Pratique très courante : déballer un tuple au sein d'une variable de boucle :

```
coords = [(2,-5),  (-14, 8),  (4,-3),  (8,8)]

#chaque élément de notre liste est un tuple :
for x,y in coords:
    distance_eucl_carre = x**2 + y**2
    if distance_eucl_carre > 100 :
        print(f"{(x,y)} est à une distance > 10 de l'origine")
```

Note sur les variables inutilisées

Il est courant de rencontrer la notation suivante pour signifier qu'une des variables déballées ne nous intéresse pas :

```
coords = [(2,-5),  (-14, 8),  (4,-3),  (8,8)]
for _, y in coords:
    # logique de code où la coordonnée x n'est pas utilisée

# Par ailleurs :
for _ in range(10):
    print("Ceci est répété 10 fois")
```

Partie 2 - Séquences en intension

Extension vs intension en mathématiques

Un ensemble se définit explicitement :

$E = \{0, 1, 4, 9, 16, 25, 36, 49, 64\}$

Ou en donnant la règle qui permet de l'obtenir :

$E = \{x^2 \mid x < 9, x \in \mathbb{N}\}$

Certains langages de programmation permettent de procéder de la même façon.

Extension vs intension en Python

Première méthode en Python :

```
E = [0, 1, 4, 9, 16, 25, 36, 49, 64]
```

Seconde méthode en Python :

```
E = []
for i in range(9):
    E.append(i**2)
```

Ou encore (nouveau pour nous) :

```
E = [i**2 for i in range(9)]
```

Listes en intension (*list comprehension*)

Certains types de traitement sont si courants qu'on leur a donné une syntaxe raccourcie :

```
noms = ["James", "Jennifer", "Michael", "Linda", "David", "Mary"]
noms_de_5_lettres = []
for n in noms:
    if len(n) == 5:
        noms_de_5_lettres.append(n)
```

Peut se formuler en intension (*comprehension* en anglais) comme :

```
noms = ["James", "Jennifer", "Michael", "Linda", "David", "Mary"]
noms_de_5_lettres = [n for n in noms if len(n) == 5]
```

Listes en intension (*list comprehension*) avec déballage

```
coords = [(2,-5),  (-14, 8),  (4,-3),  (8,8)]

mes_coords_x = [x for x,y in coords if y < 0]
mes_coords_xyz = [(x, y, (x+y)/2) for x,y in coords]
```

À noter que la condition n'est pas obligatoire :

```
mes_coords_y = [y for x,y in coords]
```

Autres cas d'utilisation

```
mes_nombres = [2, 3, 4, 5, 7, 12, 34, 231, 234]

#l'utilisation conjointe avec des fonctions permet de simplifier:
somme_pairs = sum([n for n in mes_nombres if n%2 == 0])

#si je définis une fonction mon_filtre qui retourne un booléen :
mes_nombres_filtres = [n for n in mes_nombres if mon_filtre(n)]

#liste des nombres premiers plus petits que 100 :
premiers_petits = [n for n in range(100) if est_premier(n)]
```

Autres séquences en intension

```
a = tuple(i for i in range(14, 50, 3))

noms = ["James", "Jennifer", "Michael", "Linda", "David", "Mary"]
b = {nom : nom.count("e") for nom in noms}

c = [[k for k in range(i)] for i in range(3,6)]
# c = [[0, 1, 2], [0, 1, 2, 3], [0, 1, 2, 3, 4]]
```