Python
Map, filter et *comprehension*

Nicolas Delestre

INSA
ROUEN NORMANDIE

Contexte

Créer une nouvelle liste à partir d'une séquence (*source*)

- assez courant
- algorithme fondé sur l'initialisation d'une liste résultat (à vide) et sur le parcours de la séquence donnée avec le complètement de la liste résultat en appliquant une formule ou en sélectionnant des éléments

Pour appliquer une formule

```
def mettre_au_carre_v0(l):
    res = []
    for n in l:
        res.append(n ** 2)
    return res
```

Pour filtrer des éléments

```
def selectionner_pairs_v0(l):
    res = []
    for n in l:
        if n % 2 == 0:
            res.append(n)
    return res
```

Généralisation

Pour appliquer une formule

```
def appliquer(l,
              calcul=lambda x: x):
    res = []
    for n in l:
        res.append(calcul(n))
    return res


def mettre_au_carre_v1(l):
    return appliquer(l,
                     lambda n: n**2)
```

Pour filtrer des éléments

```
def selectionner(l,
                 a_garder=lambda x: True):
    res = []
    for n in l:
        if a_garder(n):
            res.append(n)
    return res


def selectionner_pairs_v1(l):
    return selectionner(l,
                        lambda n: n % 2 == 0)
```

INSA ROUEN NORMANDIE

map et filter 1 / 2

map

- permet d'appliquer une fonction (souvent une expression lambda) à chaque élément d'une séquence
- syntaxe : `map(fonction, séquence)`
- on obtient un objet itérable tels que les éléments sont calculés à la demande (*yielding iterator*)

```
def mettre_au_carre_v2(l):
    return list(map(lambda n: n**2, l))
```

INSA ROUEN NORMANDIE

map et filter 2 / 2

filter

- permet de sélectionner les éléments d'une séquence , ceux pour lesquels une condition est vrai
- syntaxe : `filter(fonction, liste)`
- comme `map`, `filter` retourne un objet, un *yielding iterator*

```
def selectionner_pairs_v2(l):
    return list(filter(lambda n: n % 2, l))
```

INSA ROUEN NORMANDIE

≪ Compréhension ≫ de séquence

- syntaxe introduite avec python 2 :
  - `[foo(e) for e in sequence]`
  - `(foo(e) for e in sequence)`
  - `{foo(e) for e in sequence}`
  - `{e:foo(e) for e in sequence}`
- équivalente (en terme de fonctionnalité) aux fonctions `map` et `filter`
- plus *pythonic*

```
def mettre_au_carre_v3(l):
    return [n**2 for n in l]

def selectionner_pairs_v3(l):
    return [n for n in l if n % 2 == 0]
```

Performances 1 / 2

```
def benchmark_fonctions_un_parametre_naturel(nom_module, nom_fonctions, taille=10000000):
    for nom_fonction in nom_fonctions:
        code = f"from {nom_module} import {nom_fonction} as m; l=range({taille}); list(m(l))"
        print(f"{nom_fonction}, {timeit.timeit(code, number=1)}")

def benchmark_mettre_au_carre(taille=10000000):
    benchmark_fonctions_un_parametre_naturel("mapFilterComprehension", ["mettre_au_carre_v%d" % i for
        i in range(4)], taille)

def benchmark_selectionner_pairs(taille=10000000):
    benchmark_fonctions_un_parametre_naturel("mapFilterComprehension", ["selectionner_pairs_v%d" % i
        for i in range(4)], taille)

if __name__ == "__main__":
    benchmark_mettre_au_carre()
    benchmark_selectionner_pairs()
```

ROUEN NORMANDIE

Performances 2 / 2

```
$ python mapFilterComprehension.py
mettre_au_carre_v0, 3.116361833992414
mettre_au_carre_v1, 3.7307979429897387
mettre_au_carre_v2, 3.257178820000263
mettre_au_carre_v3, 2.8759335440117866
selectionner_pairs_v0, 0.714096306997817
selectionner_pairs_v1, 1.249474090989679
selectionner_pairs_v2, 0.8969143869762775
selectionner_pairs_v3, 0.5892258970125113
```

INSA ROUEN NORMANDIE

Conclusion

- Quatre façons d'appliquer un traitement ou de sélectionner des éléments d'une séquence :
  - algorithmes classiques
  - algorithmes classiques généralisés : `appliquer`, `selectionner`
  - fonctions de base de python : `map` et `filter`
  - compréhension
- Favoriser les compréhensions car :
  - plus lisibles, plus *pythonic*
  - elles peuvent produire différents types de séquences
  - plus rapides

INSA ROUEN NORMANDIE