ICC: Programmation

MX, Cours 5, 18 octobre 2019

Jean-Philippe Pellet

Previously, on Programmation…

- Types de base en Python: int, float, str, bool
- Méthodes, fonctions et slicing pour calculer des valeurs dérivées
- Conditions pour exécuter du code selon la valeur d'une expression booléenne: `if <condition>: ... else: …` et ses variantes
- Boucles pour exécuter du code plusieurs fois:
  - Boucle `while <condition>: ...`
  - Boucle `for i in range(...): …`
- Déclaration de fonctions avec type de retour et paramètres:
  - `def calculate_area(r: float) -> float: return ...`
- Utilisation de listes:
  - `values: List[int] = [1, 4, 2, 7, 3]`
  - `squares = [x * x for x in values]`

Répétition — Listes

Déclarez une liste avec les int de 0 (incl.) à 5 (excl.)

```
from typing import List
values: List[int] = [0, 1, 2, 3, 4]
values: List[int] = list(range(5)) # idem!
```

Ajoutez 1 à la valeur de la première case

```
values[0] += 1              # [1, 1, 2, 3, 4]
```

Supprimez les deux premières cases

```
values[0:2] = []            # [2, 3, 4]
```

Ajoutez les deux valeurs –2 et –3 entre 2 et 3

```
values[1:1] = [-2, -3]  # [2, -2, -3, 3, 4]
```

Ajoutez 42 à la fin de la liste

```
values.append(42)           # [2, -2, -3, 3, 4, 42]
```

Effacez toute la liste

```
values.clear()              # []
```

Cours de cette semaine

Objets immuables ou modifiables

Imports

Sets & dictionaries

Motivation: question

```python
def modify(ws: List[str]) -> None:
    ws[1] = "beau"
    ws[2] = "il"


words = ["fort", "belle", "elle", "dort"]
print(words)
modify(words)
print(words)
```

Qu'affiche le second print()?

```
['fort', 'beau', 'il', 'dort']
```

Motivation: question

```python
def modify(v: int) -> None:
    v = v + 4


value = 42
print(value)
modify(value)
print(value)
```

Qu'affiche le second print()?

42

But why??

Objet immuable, objet variable

- En Python, on peut classifier les valeurs qu'on donne aux variables en deux

— Les objets immuables ne changent pas intrinsèquement

➡ int, float, str

— Les objets variables peuvent changer via des appels de méthodes, slicing, opérateurs, etc.l

➡ List, Set, Dict

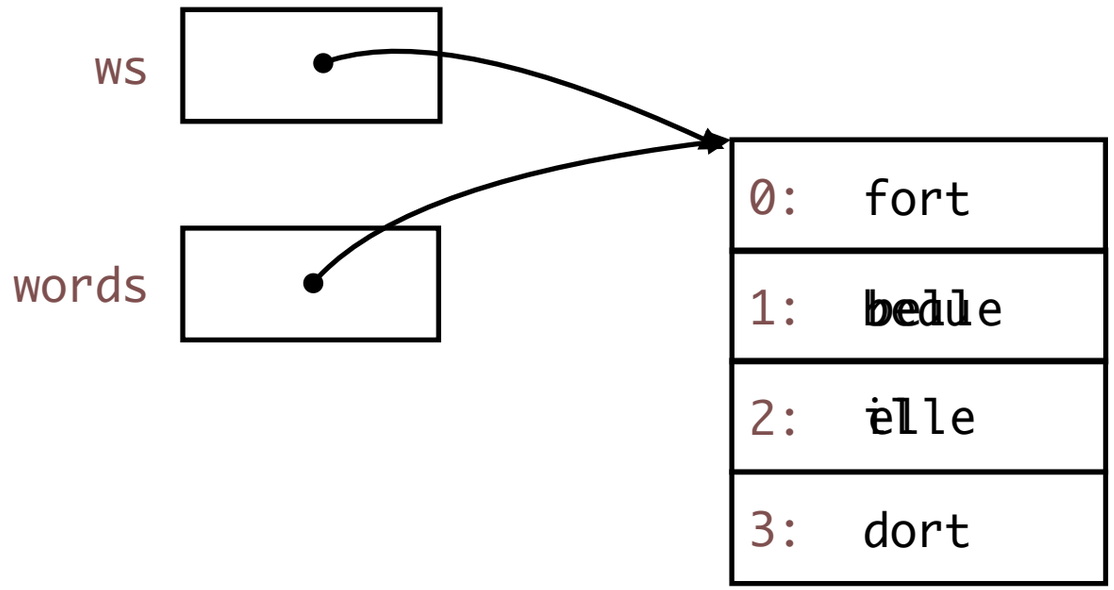
Avec un objet immuable

```python
def modify(v: int) -> None:
    v = v + 4

value = 42
print(value)  # 42
modify(value)
print(value)  # toujours 42
```

`int` est un type dont les objets sont `immuables`. On ne les modifie pas directement, mais on crée de «nouveaux ints» à chaque opération. Réassigner une variable locale comme `v` *ne change pas* `value`.

Avec un objet modifiable

```
def modify(ws: List[str]) -> None:
    ws[1] = "beau"
    ws[2] = "il"


words = ["fort", "belle", "elle", "dort"]
print(words)
modify(words)
print(words)    # ['fort', 'beau', 'il', 'dort']
```



List est un type dont les objets sont modifiable. On peut manipuler directement leur contenu. Les modifications sont vues par toutes les références au même objet.

Modifier l'immuable?

Comment modifier une variable d'un type immuable?

On peut le faire seulement indirectement en Python.

1. Il faut retourner la valeur modifiée depuis la méthode

```
def modify2(value: int) -> int:
    return value + 4


value = 42
print(value)   # 42
value = modify2(value)
print(value)   # 46
```

2. Il faut réassigner la valeur de retour à la variable à modifier lors de l'appel

Cours de cette semaine

Objets immuables ou modifiables

Imports

Sets & dictionaries

Import

- Pour utiliser du code défini ailleurs, on utilise un import

```
import math
print(math.cos(math.pi))
```

Tout ce qui est défini dans math.py est accessible avec «math.xxx»

```
import math as m
print(m.cos(m.pi))
```

Tout ce qui est défini dans math.py est accessible avec «**m**.xxx»

```
from math import *
print(cos(pi))
```

Tout ce qui est défini dans math.py est accessible directement

```
from math import pi, cos as cosinus
print(cosinus(pi))
```

Seulement *pi* et *cos*, définis dans math.py, sont importés; *cos* est renommé *cosinus*

```
from typing import List
```

Aussi valable pour les types

Partager du code entre plusieurs fichiers

- Chaque fichier *.py* est un module
- Vous pouvez donc créer et importer vos propres modules

Dans mytools.py

```
from typing import List

def double(values: List[int]) -> List[int]:
    return [2 * x for x in values]

def make_string(values: List[int], separator: str = ", ") -> str:
    return separator.join([str(x) for x in values])
```

Dans un autre fichier du même dossier

```
from mytools import *
print(double([1, 2, 3]))
print(make_string([1, 2, 3], separator=" -> "))
```

Variante:

```
from mytools import double as dbl
print(dbl([1, 2, 3]))
```

on renomme une fonction

Cours de cette semaine

Objets immuables ou modifiables

Imports

Sets & dictionaries

Set

- Un set (ensemble) est similaire à une liste, mais
- … n'a pas d'ordre intrinsèque
  - Pas possible d'utiliser l'indexation [i] ou le slicing [x:y]
- … contient un élément au plus une fois
  - et permet de tester rapidement s'il contient un élément ou non
- Objets modifiables (non immuables) comme les listes

Comparaison List/Set

```python
from typing import List, Set


my_list: List[str] = ["bonjour", "hello", "bonjour"]
print(len(my_list))    # 3
print(my_list[2])      # 'bonjour'
my_list = []           # liste vide


my_set: Set[str] = {"bonjour", "hello", "bonjour"}
print(len(my_set))     # 2
#print(my_set[2])      # pas possible, le set n'a pas d'ordre
my_set = set()         # set vide - pas {}


my_set = set(my_list)  # conversion de liste en set
my_list = list(my_set) # conversion de set en liste
```

Listes: avec la notation [ ]

Sets: avec la notation { }

Éléments dupliqués ne sont pas ajoutés une seconde fois

Autres méthodes utiles sur les sets

- `add(x)` — ajouter un élément *(listes: `append(x)`)*
- `clear()` — tout effacer
- `remove(x)` — supprime `x`
- `x in my_set` — teste si my_set contient
  - `if x in my_set: ...`
  - `if x not in my_set: ...`
- Méthodes pour l'union, l'intersection ou encore la différence de plusieurs sets
  - Serait aussi possible avec les listes, mais plus lent qu'avec les sets
    - ➡ Représentation interne différente

Dictionnaire

- Un dict (dictionnaire) est une structure qui relie des clés à des valeurs
- Conceptuellement, peut-être vu comme un ensemble de paires
- Objets modifiable (non immuables) comme les listes et les sets

Démo

Dictionnaire: exemple

```
from typing import Dict

ages: Dict[str, int] = {"Alex": 5, "Emelyne": 4, "Victor": 1}
print(ages)              # {'Alex': 5, 'Emelyne': 4, 'Victor': 1}

print(ages["Alex"])      # 5
# print(ages[“Sandra"])  # erreur d'exécution

if "Sandra" in ages:     # seulement si la clé est présente
    print(ages["Sandra"])
else:
    print("n/a")

→   ages["Nathan"] = 4

    print(ages)
    # {'Alex': 5, 'Emelyne': 4,
    #  'Victor': 1, 'Nathan': 4}

    del ages["Nathan"]
```

Clé		Valeur associée
"Alex"	→	5
"Emelyne"	→	4
"Victor"	→	1
"Nathan"	→	4

Exemple plus intéressant: *MyLittleFacebook*

```
from typing import *

# MyLittleFacebook
friendships: Dict[str, Set[str]] = {}

def add_friends(name1: str, name2: str) -> None:
    if name1 in friendships:
        friendships[name1].add(name2)
    else:
        friendships[name1] = {name2}
    if name2 in friendships:
        friendships[name2].add(name1)
    else:
        friendships[name2] = {name1}


add_friends("Alex", "Victor")
add_friends("Alex", "Emelyne")
add_friends("Alex", "Emelyne")
add_friends("Emelyne", "Rose")
```

Clé		Valeur associée
`"Alex"`	→	`{"Victor"}` `"Emelyne"}`
`"Victor"`	→	`{"Alex"}`
`"Emelyne"`	→	`{"Rose"}` `"Alex"}`
`"Rose"`	→	`{"Emelyne"}`

Résumé Cours 5

- Les objets immuables ne peuvent pas être modifiés; les objets modifiables peuvent subir des modifications (*Yay*…)
- On utilise import ou from… import pour réutiliser du code défini dans un autre fichier
- Un set (de type `Set[T]`) est un peu comme une liste, mais assure l'unicité des éléments
- Un dictionnaire (de type `Dict[K, V]`) fait correspondre des clés à des valeurs