Information, Calcul et Communication

CS-119(k) ICC – Programmation
Semaine 5

Rafael Pires
rafael.pires@epfl.ch

Lausanne, 17.03.2025

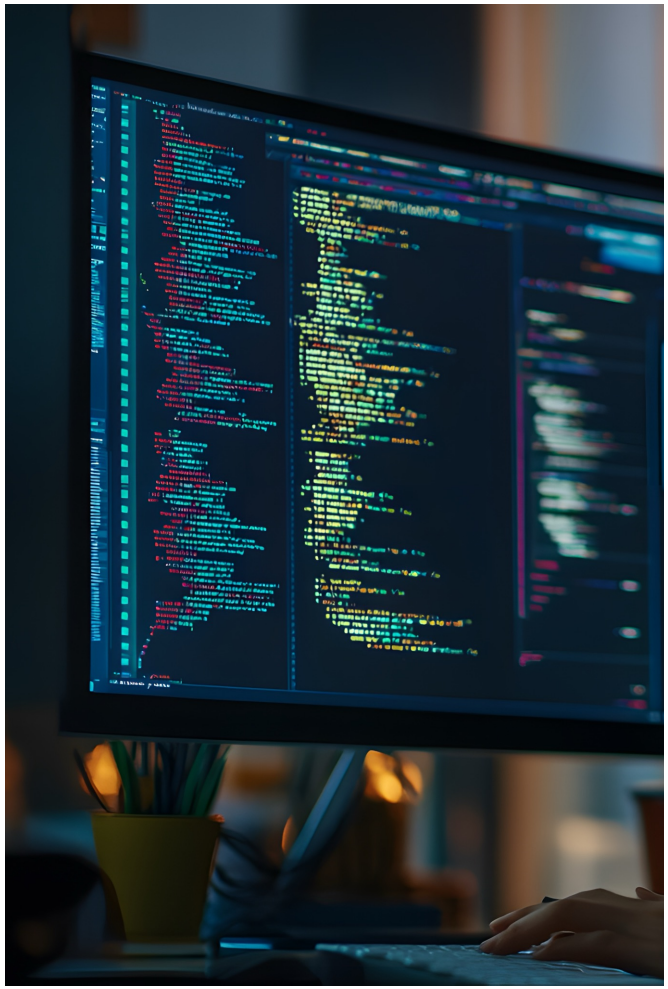
Evaluation du corps estudiantin

- **Évaluation indicative sur l'enseignement** (dès aujourd'hui jusqu'au dimanche 23.03)
  - Une question sur le déroulement du cours
  - Commentaire
- Évaluation approfondie à la fin du semestre

- Donnez un retour sur le cours sur IS-Academia
  - Feedback anonyme
  - Dites ce qui vous a plu, ce qui vous a déplu;

    ce qui vous convient, ce qui ne vous convient pas
  - Soyez constructifs : c'est votre opportunité d'aider à améliorer le cours durant les 9 semaines à venir. Je lirai tous vos retours et en tiendrai compte.

Précédemment, dans… ICC-P



Données

- Types de base en Python : `int`, `float`, `str`, `bool`
- Listes

```
values: list[int] = [1, 4, 2, 7, 3]
```

Traitement

- Méthodes, fonctions et slicing pour calculer des valeurs dérivées
- Branchements pour exécuter du code selon la valeur d'une expression booléenne
- Boucles pour exécuter du code plusieurs fois
  - Interruptions et saut dans les boucles avec `break` et `continue`
- Fonctions

Listes et slicing

- Accès aux éléments via slicing

```
my_ints = [10, 20, 30, 40, 50, 60]
my_ints[0]          # 10
my_ints[0:2]        # [10, 20]
my_ints[:4]         # [10, 20, 30, 40]
my_ints[4:]         # [50, 60]
```

- Modification des éléments via slicing

```
my_ints[4:] = [55, 65]          # [10, 20, 30, 40, 55, 65]
my_ints[4:] = []                # [10, 20, 30, 40]
my_ints[0:0] = [0, 1, 2]        # [0, 1, 2, 10, 20, 30, 40]
my_ints[0] = [0, 1, 2]          # [[0, 1, 2], 1, 2, 10, 20, 30, 40]
```

`import` et `from`

```
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 pi, cos as cosine
print(cosine(pi))
```

Seulement **`pi`** et **`cos`** défini dans `math.py` sont importés; **`cos`** est renommé **`cosine`**

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

Tout ce qui est défini dans `math.py` est accessible directement.
Non recommandé !

Partager du code entre plusieurs fichiers

tri_insertion.py

```python
def permuter(L, i, j):
    L[i], L[j] = L[j], L[i]

def bonne_place(L, i):
    while i > 0 and L[i] < L[i-1]:
        permuter(L, i, i-1)
        i = i - 1

def tri_insertion(L):
    for i in range(1, len(L)):
        bonne_place(L, i)
```

tri_temps.py

```python
import tri_fusion
import tri_insertion
import random
import time

ma_liste = list(range(10000))
random.shuffle(ma_liste)

start_fusion = time.time()
tri_fusion.tri_fusion(ma_liste)
end_fusion = time.time()
print("tri_fusion", end_fusion - start_fusion)

start_insertion = time.time()
tri_insertion.tri_insertion(ma_liste)
end_insertion = time.time()
print("tri_insertion", end_insertion - start_insertion)
```

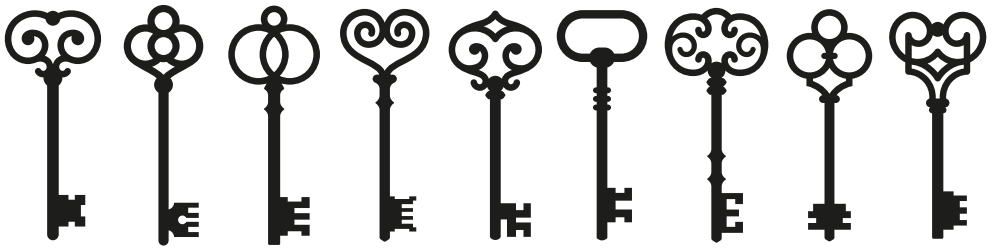
tri_fusion.py

```python
def fusion(L1 : list[int], L2 : list[int]) -> list[int]:
    i, j = 0, 0
    L = []
    while i < len(L1) and j < len(L2):
        if L1[i] < L2[j]:
            L.append(L1[i])
            i += 1
        else:
            L.append(L2[j])
            j += 1
    L.extend(L1[i:])
    L.extend(L2[j:])
    return L

def tri_fusion(L : list[int]) -> list[int]:
    if len(L) <= 1:
        return L
    else:
        m = len(L) // 2
        L1 = tri_fusion(L[:m])
        L2 = tri_fusion(L[m:])
        return fusion(L1, L2)
```

Test case : Problème des 100 prisonniers

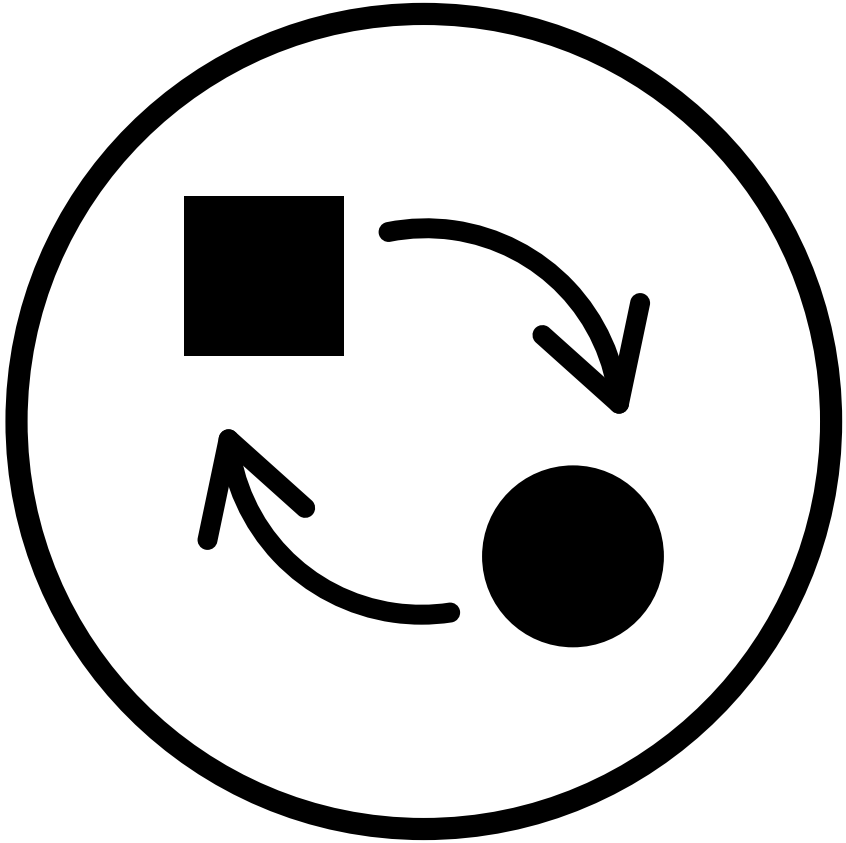
- 100 prisonniers, chacun numéroté de 0 à 99, et 100 boîtes, également numérotées de 0 à 99.
  Chaque boîte contient une clé différente, correspondant à l'un des prisonniers.
- Chaque prisonnier peut ouvrir 50 boîtes, sans savoir ce que les autres ont fait,
  et doit tout remettre en place.
- Si tous les prisonniers trouvent leur clé, ils sont libérés. Si un seul échoue,
  ils sont tous exécutés.
- La question est donc : **Quelle est la meilleure stratégie et quelle est la probabilité de réussite ?**
  - Approche naïve : $1/2^{100}$
  - Approche optimale : environ 31%
- https://www.youtube.com/watch?v=iSNsgj1OCLA

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

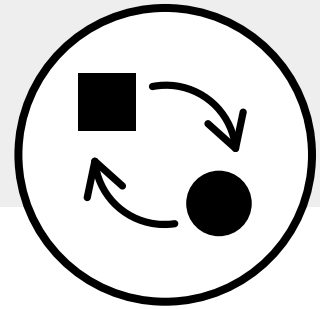


Simulons ce problème !

Objets immuables ou modifiables



Objets immuables ou modifiables



Paul de Rességuier, Épitaphe d'une jeune fille

```
number = 3
other_number = number
other_number += 1


print(number)
print(other_number)
```

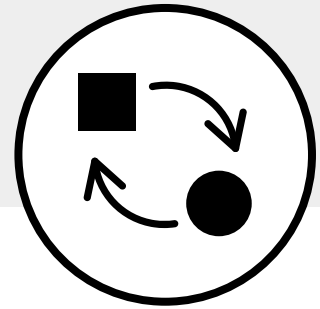
```
words = ["fort", "belle", "elle", "dort"]
other_words = words
other_words[1] = "beau"
other_words[2] = "il"

print(words)
print(other_words)
```

- **Qu'affichent les `print()` ?**

```
3
4
```

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

Objets immuables ou modifiables



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


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

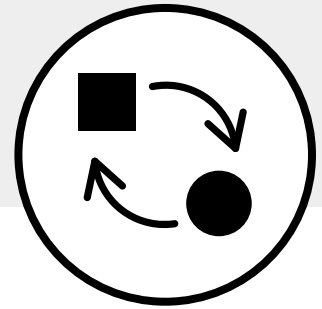
```python
def modify(ws: list[str]) -> None:
    ws[1] = "beau"
    ws[2] = "il"

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

- **Qu'affichent les `print()` ?**

```
42
```

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

Objets immuables ou modifiables

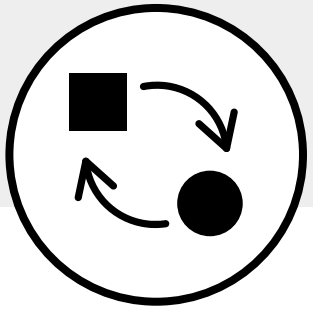


- En Python, on peut classifier les valeurs qu'on donne aux variables en deux

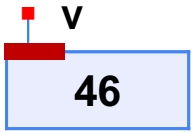
- Les objets immuables ne changent pas intrinsèquement
  - `int`, `float`, `str`, `bool`

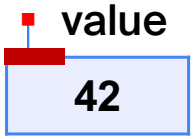
- Les objets modifiables peuvent changer via des appels de méthodes, slicing, opérateurs, etc.
  - `list`, `set`, `dict`

Objets immuables



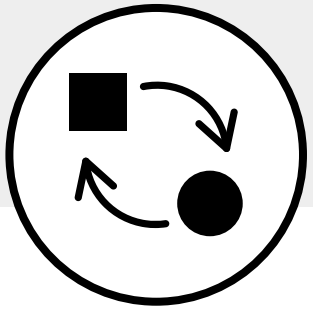
```
def modify(v: int) -> None:
    v = v + 4

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



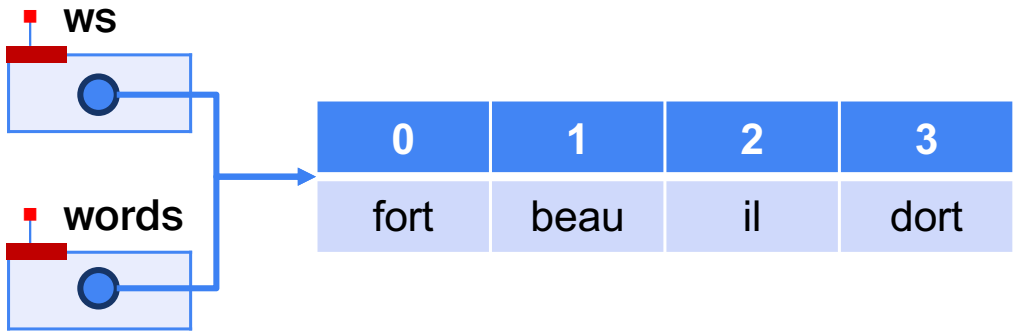


- `int` est un type dont les objets sont immuables. On ne les modifie pas directement, mais on crée de « nouveaux ints » à chaque opération.
  Dans l'exemple, réaffecter une variable locale comme `v` ne change pas `value`.

Objets immuables ou modifiables

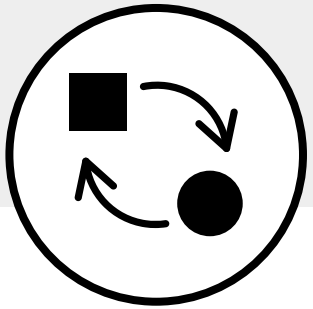


```
def modify(ws: list[str]) -> None:
    ws[1] = "beau"
    ws[2] = "il"

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



- `list` est un type dont les objets sont modifiables.
  On peut manipuler directement leur contenu.
  Les modifications sont vues par toutes les références au même objet.

Modifier l'immuable ?

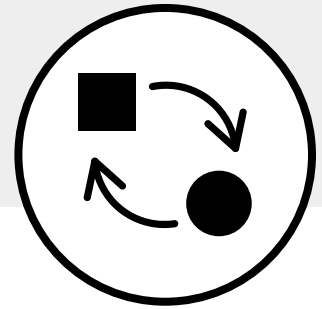


- Comment modifier une variable d'un type immuable?
  - On peut le faire seulement indirectement en Python.

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

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

1. Il faut retourner la valeur modifiée depuis la fonction.
2. Il faut réassigner la valeur de retour à la variable à modifier lors de l'appel.

Empêcher de modifier le modifiable ?

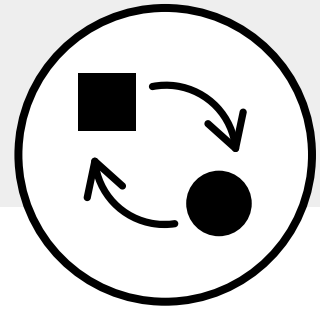


- Comment être sûr qu'un appel de fonction ne va pas modifier une structure modifiable comme une liste?
  - On peut passer une copie de la structure.

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

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

1. Même fonction qu'avant
2. C'est une copie de la liste qui est fournie. L'original reste intact.

Immuable vs. modifiable



- Types dont les instances sont immuables

```
int
float
bool
str
datetime
tuple
frozenset
```

- Types dont les instances sont modifiables

```
list
set
dict
classes
```

Sets

Sets

- 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
- Les sets sont modifiables (non immuables) comme les listes
  - mais ils doivent contenir des éléments immuables
    (donc, par exemple: pas de listes dans des sets)

Comparaison `list`/`set`

```python
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
```

- **`list`** avec la notation **`[]`**
- **`set`** avec la notation **`{}`**
- Éléments dupliqués ne sont pas ajoutés une seconde fois

Autres méthodes utiles sur les sets

- `my_set.add(x)`– ajouter un élément (listes `append(x)`)
- `my_set.clear()` – tout effacer
- `my_set.remove(x)` – supprime x
- `x in my_set` – teste si `my_set` contient `x` (en temps constant: O(1))
  - `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

Résumé Cours 5 – ICC-P

- On utilise `import` ou `from` pour réutiliser du code défini dans un autre fichier
- Les objets immuables ne peuvent pas être modifiés; les objets modifiables peuvent subir des modifications
- Un set (de type `set[T]`) est un peu comme une liste, mais assure l'unicité des éléments
  - Pas d'ordre intrinsèque; on ne peut pas récupérer l'élément i
  - On peut convertir entre des listes et des sets facilement

rafael.pires@epfl.ch

Merci