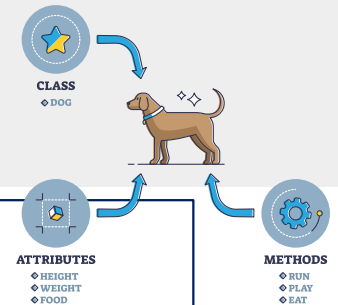


Information, Calcul et Communication

CS-119(k) ICC – Programmation Semaine 7

Rafael Pires
rafael.pires@epfl.ch

Les paramètres aux méthodes



```
from dataclasses import dataclass
```

```
@dataclass
```

```
class Rectangle:
```

```
    width: float
```

```
    height: float
```

```
    def area(self):
```

```
        return self.width * self.height
```

■ Pas besoin de paramètre : dépend uniquement des attributs internes !

```
    def resize(self, factor: float):
```

```
        self.width *= factor
```

```
        self.height *= factor
```

■ Besoin d'un paramètre : le facteur change à chaque appel

```
# Exemples d'utilisation
```

```
r1 = Rectangle(10, 5)
```

```
print("Aire de r1 :", r1.area()) # → 50, pas besoin de paramètre
```

```
r1.resize(2) # redimensionne avec un facteur donné
```

```
print("Nouvelle aire de r1 :", r1.area()) # → 200, changement dû au facteur
```

La concaténation des listes



```
a = "ha"  
b = a + "ha" # Concaténation → "haha"  
c = a * 3     # Répétition → "hahaha"
```

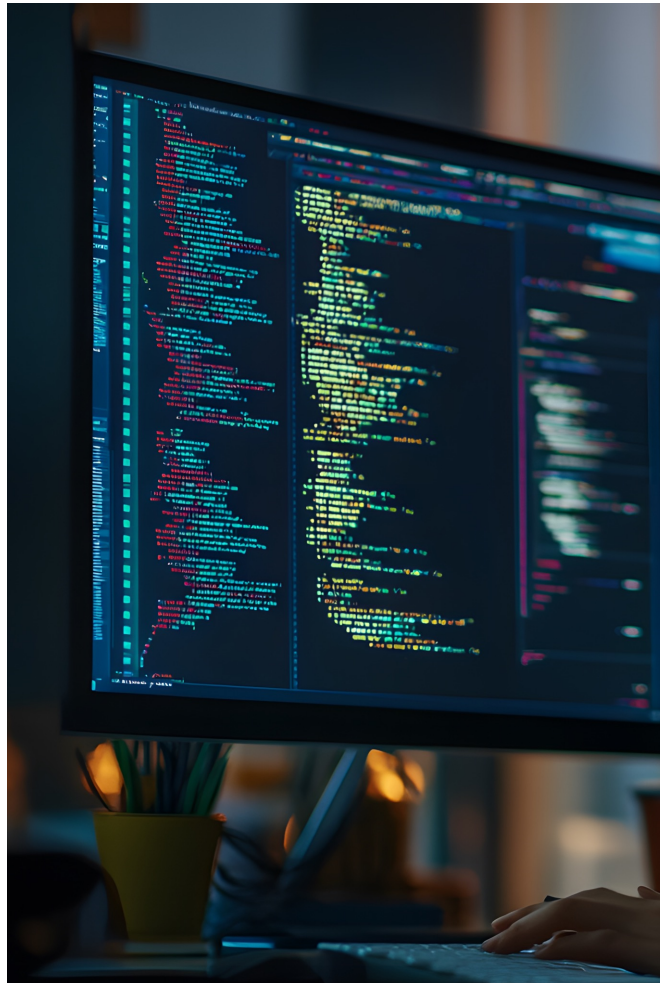
```
x = [1, 2]  
y = x + [3, 4] # Concaténation → [1, 2, 3, 4]  
z = x * 3      # Répétition → [1, 2, 1, 2, 1, 2]
```

- À retenir :
 - + colle deux listes ou deux chaînes → même type des deux côtés.
 - * n répète le contenu n fois.
 - Les opérations sont **non destructives** (elles créent une **nouvelle** liste ou chaîne).

Comment savoir ?

→ **011** **0,375** $(0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3})$
ou **3** $(0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) ?$

Précédemment, dans... ICC-P



Données

- Types de base en Python (immuables) : **int**, **float**, **str**, **bool**
- Types modifiables
 - **Listes**
 - **Sets**
 - **Dictionnaires**
 - **Dataclasses**

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

```
values: set[int] = {1, 3, 4, 2, 7, 2, 3}
```

```
values: dict[str,int] = {"A": 5, "B": 3}
```

```
@dataclass  
class Cylinder:  
    radius: float  
    height: float
```

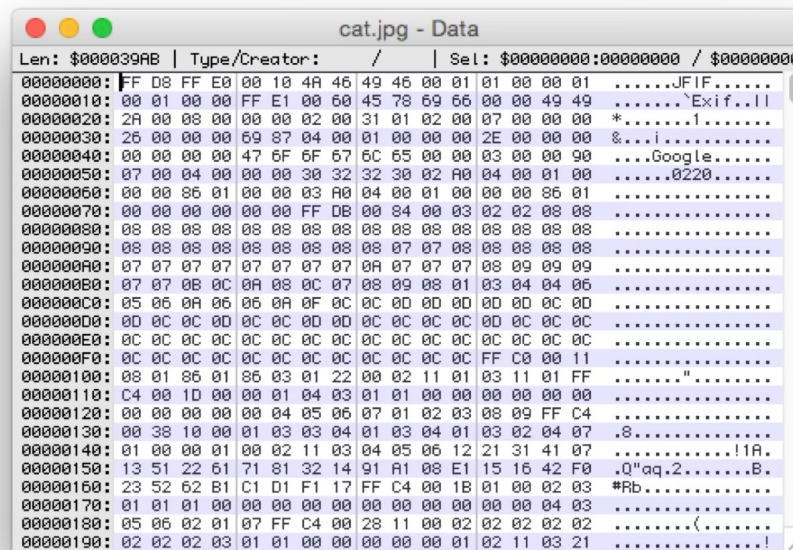
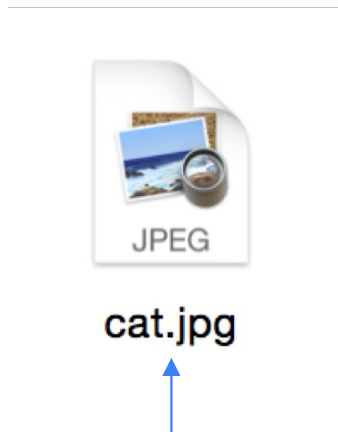
Traitement

- **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
- **Fonctions**

Fichiers



Fichiers et bytes

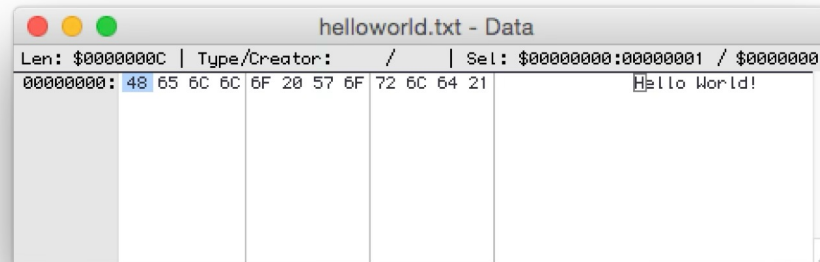


Un fichier est une séquence de bytes qui, interprétés correctement, ont du sens.

Fichiers et bytes



helloworld.txt



0	1	2	3	4	5	6	7	8	9	10	11
01001000	01100101	01101100	01101100	01101111	00100000	01010111	1101111	01110010	01101100	01100100	00100001
48	65	6C	6C	6F	20	57	6F	72	6C	64	21
H	e	l	l	o		W	o	r	l	d	!

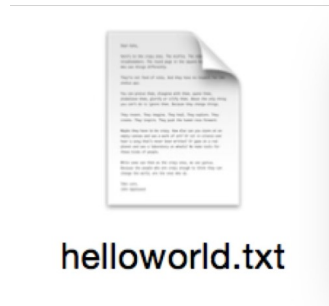
■ Indice de l'octet

■ binaire

■ hexadécimal

■ Interprétation texte

Fichiers et bytes



helloworld.txt - Data				
Len: \$0000000C	Type/Creator: /	Set: \$00000000:00000001 / \$00000001		
00000000:	48	65	6C	6C 6F 20 57 6F 72 6C 64 21 Hello World!



En plus des **données**, il y a des **métadonnées**, gérées par le **système d'exploitation** : nom, extension ou format, taille, droits d'accès, date de création, date de dernière modification, la position physique sur le disque.

Le système qui **interprète** chaque byte (ou séquence de bytes) pour retrouver le caractère correspondant repose sur un certain **encodage des caractères**.

Fichiers et bytes



- American Standard Code for Information Interchange (**ASCII**, 1963)

0	Null character
1	Start of header
...	
9	Tabulator (“\t”)
10	Line feed (“\n”)
...	
32	Space
...	
48	“0”
...	...
57	“9”

65	“A”
66	“B”
67	“C”
...	...
90	“Z”
...	
97	“a”
...	...
122	“z”
...	
127	Delete

Fichiers et bytes



- Unicode :
 - Standard qui attribue un **numéro unique à chaque caractère**, couvrant pratiquement tous les systèmes d'écriture du monde (latin, chinois, japonais, arabe, etc.).
 - Espace d'adressage théorique : ~1.1 million points de code
 - Caractères réellement attribués : un peu plus de 149 mil.

- UTF-8 : Format de codage des caractères Unicode.
 - Chaque caractère est encodé **sur 1 à 4 octets**, selon sa position dans la table Unicode :
 - UTF-8 est **compatible avec ASCII** (très pratique).
 - Plus le caractère est « exotique », plus il prend de place.

Tableaux de caractères Unicode



Characters

Recently Used

Favorites

Unicode

Math Symbols

Arrows

Currency Symbols

Pictographs

Bullets/Stars

Emoji

Latin

Divination Symbols

Digits - All

Enclosed Characters

Geometrical Shapes

Ideographic Desc. Characters

Letterlike Symbols - All

Box Drawing

Musical Symbols

Parentheses - All

Phonetic Alphabet

Braille Patterns

Punctuation - All

Sign/Standard Symbols

Technical Symbols

Armenian

Coptic

Cyrillic

Georgian

Glagolitic

Greek

Dingbats

Arabic

Unicode	Title	Category
00000180	Latin Extended-B	Latin
00000250	IPA Extensions	Latin
00000280	Spacing Modifier Letters	Modifier Letters
00000300	Combining Diacritical Marks	Combining Marks
00000370	Greek and Coptic	Greek
00000400	Cyrillic	Cyrillic
00000500	Cyrillic Supplement	Cyrillic
00000530	Armenian	Armenian
00000590	Hebrew	Hebrew
00000600	Arabic	Arabic
00000700	Syriac	Syriac
00000750	Arabic Supplemant	Arabic

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0440	0440	0441	0442	0443	0444	0445	0446	0447	0448	0449	044A	044B	044C	044D	044E	044F
0450	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	045A	045B	045C	045D	045E	045F
0460	0460	0461	0462	0463	0464	0465	0466	0467	0468	0469	046A	046B	046C	046D	046E	046F
0470	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479	047A	047B	047C	047D	047E	047F
0480	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	048A	048B	048C	048D	048E	048F
0490	0490	0491	0492	0493	0494	0495	0496	0497	0498	0499	049A	049B	049C	049D	049E	049F
04A0	04A0	04A1	04A2	04A3	04A4	04A5	04A6	04A7	04A8	04A9	04AA	04AB	04AC	04AD	04AE	04AF
04B0	04B0	04B1	04B2	04B3	04B4	04B5	04B6	04B7	04B8	04B9	04BA	04BB	04BC	04BD	04BE	04BF
04C0	04C0	04C1	04C2	04C3	04C4	04C5	04C6	04C7	04C8	04C9	04CA	04CB	04CC	04CD	04CE	04CF
04D0	04D0	04D1	04D2	04D3	04D4	04D5	04D6	04D7	04D8	04D9	04DA	04DB	04DC	04DD	04DE	04DF
04E0	04E0	04E1	04E2	04E3	04E4	04E5	04E6	04E7	04E8	04E9	04EA	04EB	04EC	04ED	04EE	04EF
											</					

Lire un fichier texte



```
file = open("movies.txt", "r", encoding="utf-8")
contents = file.read()
file.close()
```

■ En fait, on n'écrit pas ceci!

■ On ouvre ce fichier en lecture (read)

■ On donne l'encodage à utiliser pour ne pas avoir de surprise

```
with open("movies.txt", "r", encoding="utf-8") as file:
    contents = file.read()
print(contents)
```

■ Lecture!

■ Pour être sûr de ne pas oublier le `close()`, on utilise un **with... as**

■ Contient, sous forme de grand string unique, l'intégralité du fichier *movies.txt*

Écrire un fichier texte



■ On ouvre ce fichier en écriture (write)

```
with open("results.txt", "w", encoding="utf-8") as file:  
    file.write(...)
```

■ Le travail est fait par la méthode **write(...)**

■ Comme pour la lecture, on utilise un **with... as**, qui fait un **close()** automatique

```
file.write("une ligne\nune autre ligne")
```

■ Pour écrire plusieurs lignes, on doit indiquer le caractère de nouvelle ligne **\n**

```
for ...:
```

```
    file.write(" ... \n")
```

■ Si les résultats à écrire sont générés au fur et à mesure, on peut les écrire petit à petit (ici, ligne par ligne) avec des appels répétés de **write()**

Traiter les données lues



- Pour de petits fichiers, **on peut tout lire d'un coup...**
- Mais comment traiter le contenu **ligne par ligne**?
 - **Conversions** de sous-chaînes en **int**, en **datetime**, en **bool**, etc.
 - **Transformations** de chaînes avec `tests`, `split()`, `replace()`
 - **Modélisation** des données lues avec des classes
- Exemple : lecture du fichier *movies.txt*

```
Year;Length;Title;Subject;Actor;Actress;Director;Popularity;Awards;*Image
INT;INT;STRING;CAT;CAT;CAT;CAT;INT;BOOL;STRING
1990;111;Tie Me Up! Tie Me Down!;Comedy;Banderas, Antonio;Abril, Victoria;Almodóvar, Pedro;68;No;NicholasCage.png
1991;113;High Heels;Comedy;Bosé, Miguel;Abril, Victoria;Almodóvar, Pedro;68;No;NicholasCage.png
1983;104;Dead Zone, The;Horror;Walken, Christopher;Adams, Brooke;Cronenberg, David;79;No;NicholasCage.png
1979;122;Cuba;Action;Connery, Sean;Adams, Brooke;Lester, Richard;6;No;seanConnery.png
```

...

Lecture et modélisation



```
from dataclasses import dataclass
```

```
@dataclass
```

```
class Movie:
    title: str
    year: int
    duration: int
    has_awards: bool
```

■ On décide d'une représentation sous de forme de classe pour nos données à traiter

```
with open("movies.txt", "r", encoding="utf-8") as file:
    contents = file.read()
```

```
lines = contents.split("\n")
```

■ Séparation du contenu de chaque ligne, à chaque apparition du caractère \n

Lecture et modélisation



```
movies: list[Movie] = []  
  
for line in lines[2:]:  
    parts = line.split(";")  
  
    duration_str = parts[1]  
    if duration_str == "":  
        duration = 0  
    else:  
        duration = int(duration_str)  
  
    movie = Movie(title, int(parts[0]), duration, parts[8] == "Yes")  
    movies.append(movie)
```

■ On prépare une liste de **Movies**, pour l'instant vide

■ On saute les deux premières lignes (cf. exemple de contenu)

■ De temps en temps, la durée est vide... on la met alors à 0. Sinon, on convertit de **str** vers **int**

■ On crée un nouvel objet de type **Movie** et on l'ajoute à la liste

Traitement de données et écriture



```
with open("results.txt", "w", encoding="utf-8") as file:
    total_duration = 0

    for movie in movies:
        if movie.has_awards:
            file.write(f"{movie.duration:3} min: \"{movie.title}\" ({movie.year})\n")
            total_duration += movie.duration

    file.write(f"\nTotal duration: about {total_duration // 60} hours")
```

■ On prépare un fichier de sortie pour y écrire des données.

■ On y écrit ce qu'on veut : ici, la durée, le titre et l'année des films ayant reçu un prix, chacun sur une ligne.

■ Dans la boucle, on en a profité pour calculer la durée totale de ces films, qu'on écrit comme ligne finale de ce fichier

Résumé : opérations sur les strings

- Tests

- Test de **longueur**, de **case** ■

```
text = "..."  
  
if len(text) > 20:  
if text == text.lower():
```

- Test de **préfixe**, **suffixe**, **contenance** ■

```
if text.startswith("..."):  
if text.endswith("..."):  
if "..." in text:
```

- Split

- **Sépare** un string en une liste **selon un délimiteur**

```
text = "a,b,c"  
parts = text.split(",")
```

- Join

- **Crée** un string à partir d'une liste **en insérant un délimiteur**

```
", ".join(parts)
```

- Replace

- **Rechercher-remplacer**

```
new_text = text.replace("a", "A")
```

Résumé Cours 7 – ICC-P

- Un caractère est représenté par une série de bytes selon un **encodage** précis
 - Aujourd'hui, **UTF-8** est le plus utilisé
- On peut facilement **lire et écrire des fichiers texte** en Python
 - Nouvelle structure : le **with ... as** pour un **close()** automatique
- On peut traiter les données lues avec :
 - Des **transformations** de strings
 - Des **conversions** vers d'autres types
 - La **construction d'objets** via une modélisation par des classes

rafael.pires@epfl.ch

EPFL



Merci