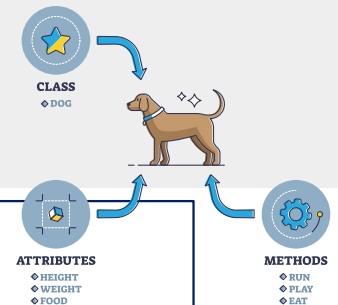


Information, Calcul et Communication

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

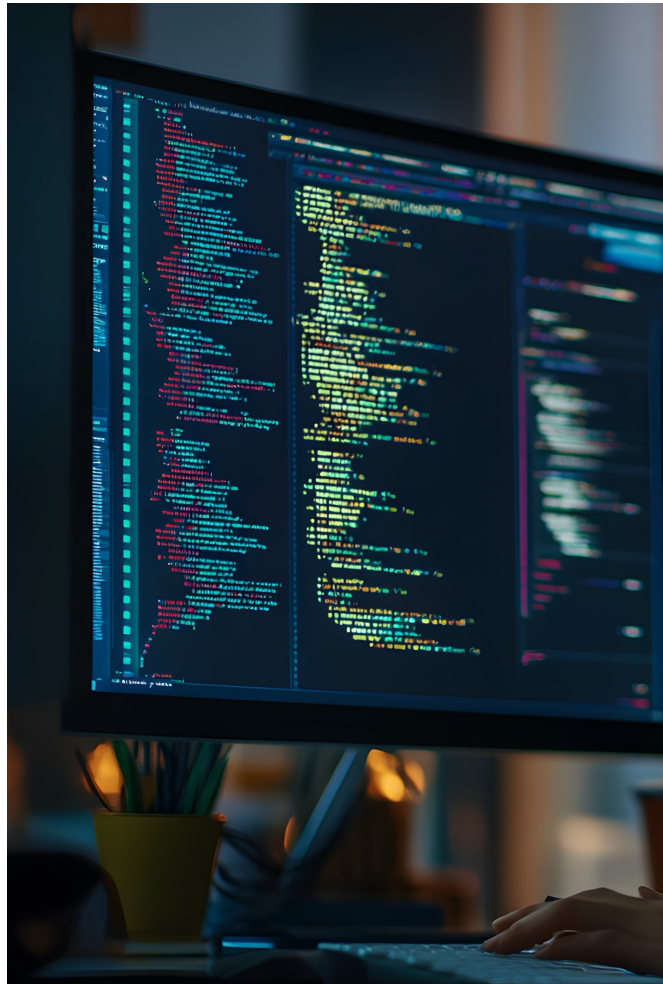


```
a = "ha"  
b = a + "ha" # Concaténation → "haha"  
c = a * 3    # Répétition → "hahaha"
```

```
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).

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
```

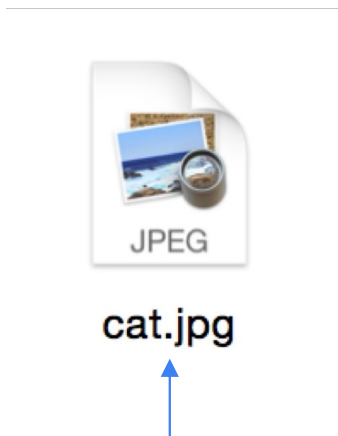
Traitement

- **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

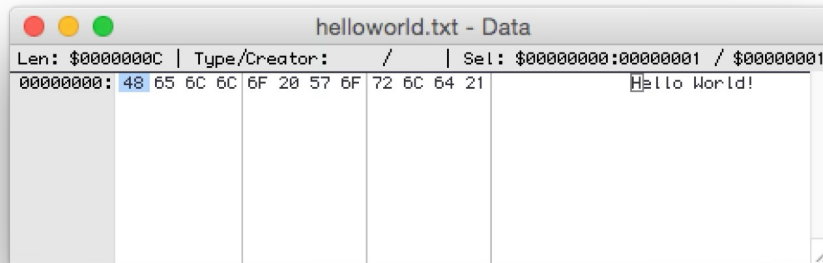
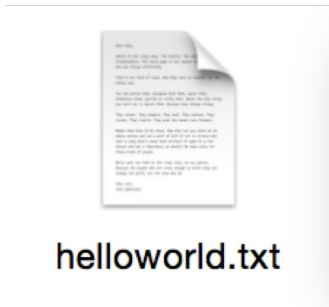


```
cat.jpg - Data
Len: $000039AB | Type/Creator: / | Sel: $00000000:00000000 / $00000000
00000000: FF D8 FF E0 00 10 4A 46 49 46 00 01 01 00 00 01 .....JFIF.....
00000010: 00 01 00 00 FF E1 00 50 45 78 59 56 00 00 49 49 .....`Exif..ll
00000020: 2A 00 08 00 00 00 02 00 31 01 02 00 07 00 00 00 *.....!.....
00000030: 26 00 00 00 59 87 04 00 01 00 00 00 2E 00 00 00 &...i.....
00000040: 00 00 00 00 47 6F 6F 67 6C 65 00 00 03 00 00 90 ....Google.....
00000050: 07 00 04 00 00 00 30 32 32 30 02 A0 04 00 01 00 ....0220.....
00000060: 00 00 86 01 00 00 03 A0 04 00 01 00 00 00 86 01 .....
00000070: 00 00 00 00 00 00 FF D8 00 84 00 03 02 02 08 08 .....
00000080: 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 .....
00000090: 08 08 08 08 08 08 08 08 08 07 07 08 08 08 08 08 .....
000000A0: 07 07 07 07 07 07 07 07 0A 07 07 07 08 09 09 09 .....
000000B0: 07 07 08 0C 0A 08 0C 07 08 09 08 01 03 04 04 06 .....
000000C0: 05 06 0A 06 06 0A 0F 0C 0C 0D 0D 0D 0D 0C 0C 0D .....
000000D0: 0D 0C 0C 0D 0C 0C 0D 0D 0C 0C 0C 0C 0D 0C 0C 0C .....
000000E0: 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C .....
000000F0: 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C FF C0 00 11 .....
00000100: 08 01 86 01 86 03 01 22 00 02 11 01 03 11 01 FF .....".
00000110: C4 00 1D 00 00 01 04 03 01 01 00 00 00 00 00 00 .....
00000120: 00 00 00 00 00 04 05 06 07 01 02 03 08 09 FF C4 .....
00000130: 00 38 10 00 01 03 03 04 01 03 04 01 03 02 04 07 .8.....
00000140: 01 00 00 01 00 02 11 03 04 05 06 12 21 31 41 07 .....!1A.
00000150: 13 51 22 61 71 81 32 14 91 A1 08 E1 15 16 42 F0 .Q"aq.2.....B.
00000160: 23 52 62 B1 C1 D1 F1 17 FF C4 00 1B 01 00 02 03 #Rb.....
00000170: 01 01 01 00 00 00 00 00 00 00 00 00 00 00 04 03 .....
00000180: 05 06 02 01 07 FF C4 00 28 11 00 02 02 02 02 02 .....(.....
00000190: 02 02 02 03 01 01 00 00 00 00 00 01 02 11 03 21 .....!
```



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

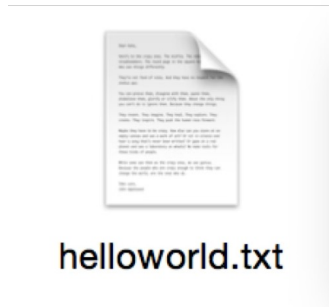
Fichiers et bytes



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:	Type/Creator:			Sel:
\$0000000C	/			\$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



The screenshot shows the 'Characters' application window with the 'Unicode' section selected. The main area displays a grid of Cyrillic characters, including Latin Extended-B, IPA Extensions, Spacing Modifier Letters, Combining Diacritical Marks, Greek and Coptic, Cyrillic, and Cyrillic Supplement. The grid is organized by Unicode range (0440-04E0) and columns (0-16). A search bar is visible at the top right, and an 'Add to Favorites' button is at the bottom right.

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 Supplement	Arabic

Unicode	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0440	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	
0450	è	ë	ĥ	í	ε	s	í	ï	j	љ	њ	ћ	ќ	й	ў	ц
0460	Ω	w	Ѡ	ѡ	Є	є	А	а	Ӓ	ӓ	Ӕ	ӕ	Ӧ	ӧ	Ө	ө
0470	Ψ	ψ	Θ	θ	V	v	ÿ	ÿ	Ϸ	ϸ	Ϲ	Ϻ	ϻ	ϼ	Ͻ	Ͼ
0480	С	с	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ	Ѫ	ѫ	Ѭ	ѭ
0490	Г	г	Ғ	ғ	Б	б	Ж	ж	З	з	Қ	қ	К	к	К	к
04A0	К	к	Ң	ң	Н	н	П	п	Q	q	Ç	ç	Т	т	У	у
04B0	Ү	ү	Х	х	Ц	ц	Ч	ч	Ч	ч	Һ	һ	Е	е	Е	е
04C0	І	Ӓ	ӓ	Ӕ	ӕ	Ӧ	ӧ	Ө	ө	Ӫ	ӫ	Ӭ	ӭ	Ӯ	ӯ	Ӱ
04D0	Ӓ	ӓ	Ӕ	ӕ	Ӧ	ӧ	Ө	ө	Ӫ	ӫ	Ӭ	ӭ	Ӯ	ӯ	Ӱ	ӱ
04E0	Ӓ	ӓ	Ӕ	ӕ	Ӧ	ӧ	Ө	ө	Ӫ	ӫ	Ӭ	ӭ	Ӯ	ӯ	Ӱ	ӱ

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