

# Programmation

## GC/MX, Cours 12

9 décembre 2022

(Enregistrement de l'année passée)

Jean-Philippe Pellet

```
class ProgramView(Canvas):
    def __init__(self, parent, view) -> None:
        Canvas.__init__(self, parent, height=2 * TOP_MARGIN + 4 * LINE_HEIGHT, highlightthickness=0, view)

    def redraw(
        self,
        program: Program,
        current_subprogram: List[Instruction],
        current_instruction_index: int,
    ) -> None:
        height = self.winfo_height()
        width = self.winfo_width()

        self.delete(ALL)
        self.create_rectangle(0, 0, width, height, fill=window_background_color, width=0)

        # boucle pour les 4 sous-programmes P1 à P4
        for i, subprogram in enumerate(
            [program.P1, program.P2, program.P3, program.P4]
        ):
            # dessin du titre
            instruction_center_y = TOP_MARGIN + 1 * LINE_HEIGHT + LINE_HEIGHT // 2
            self.create_text(LEFT_MARGIN // 2, instruction_center_y, text=f"P{i + 1}")

            # dessin de chaque instruction
            for j, instr in enumerate(subprogram):
                instruction_center_x = (
                    LEFT_MARGIN
                    + j * (INSTRUCTION_BOX_SPACING + INSTRUCTION_BOX_WIDTH)
                    + INSTRUCTION_BOX_WIDTH // 2
                )
                instruction_x = instruction_center_x - INSTRUCTION_BOX_WIDTH // 2
                instruction_y = instruction_center_y - INSTRUCTION_BOX_HEIGHT // 2
                instruction_width = INSTRUCTION_BOX_WIDTH
                instruction_height = INSTRUCTION_BOX_HEIGHT
```

# 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**, **sets** et **dictionnaires**
- Déclaration de **classes**: `@dataclass class` `Rectangle`: ...
- Création, chargement, manipulation et sauvegarde d'**images**
- **Programmation dynamique** pour trouver des seams
- **Fonctions** comme valeurs, paramètres; fonctions d'**ordre supérieur**

# Cours de cette semaine

*Lecture/écriture de fichiers texte*  
*Opérations sur des chaînes de caractères*

# 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!



cat.jpg - Data			
Len:	Type/Creator:	Sel:	
\$000039AB	/	\$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 60 45 78 69 66 00 00 49 49		.....`Exif..ll
00000020:	2A 00 08 00 00 00 02 00 31 01 02 00 07 00 00 00		*.....1.....
00000030:	26 00 00 00 69 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 DB 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 0C		.....
000000D0:	0D 0C 0C 0D 0C 0C 0D 0D 0C 0C 0C 0C 0D 0C 0C 0C		.....
000000E0:	0C		.....
000000F0:	0C FF C0 00 11		.....
00001000:	08 01 86 01 86 03 01 22 00 02 11 01 03 11 01 FF		....."
00001100:	C4 00 1D 00 00 01 04 03 01 01 00 00 00 00 00 00		.....
00001200:	00 00 00 00 00 04 05 06 07 01 02 03 08 09 FF C4		.....
00001300:	00 38 10 00 01 03 03 04 01 03 04 01 03 02 04 07		.8.....
00001400:	01 00 00 01 00 02 11 03 04 05 06 12 21 31 41 07		.....!1A.
00001500:	13 51 22 61 71 81 32 14 91 A1 08 E1 15 16 42 F0		.Q"aq.2.....B.
00001600:	23 52 62 B1 C1 D1 F1 17 FF C4 00 1B 01 00 02 03		#Rb.....
00001700:	01 01 01 00 00 00 00 00 00 00 00 00 00 00 04 03		.....
00001800:	05 06 02 01 07 FF C4 00 28 11 00 02 02 02 02 02		.....(.....
00001900:	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**

# Les fichiers texte

une *séquence de bytes* (1 byte = 8 bits)

0	1	2	3	4	5	6	7	8	9	10	11	index du byte
01001000	01100101	01101100	01101100	01101111	00100000	01010111	11011111	01110010	01101100	01100100	00100001	notation binaire
48	65	6C	6C	6F	20	57	6F	72	6C	64	21	notation hexadécimale
<b>H</b>	<b>e</b>	<b>l</b>	<b>l</b>	<b>o</b>		<b>W</b>	<b>o</b>	<b>r</b>	<b>l</b>	<b>d</b>	<b>!</b>	<i>inter- prétation texte</i>

+ quelques *informations supplémentaires*:

*nom, extension/format, taille, droits d'accès, date de création,  
date de la dernière modification, ..., 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* (Charset)

# ASCII Charset

---

American Standard Code for Information Interchange (1963)

0	Null character	65	“A”
1	Start of header	66	“B”
...		67	“C”
9	Tabulator (“\t”)	...	...
10	Line feed (“\n”)	90	“Z”
...		...	
32	Space	97	“a”
...		...	...
48	“0”	122	“z”
...	...	...	
57	“9”	127	Delete

Manquent: à, é, ä, €, ...

# Unicode, UTF-8 Charset

---

<http://fr.wikipedia.org/wiki/Unicode>

<http://fr.wikipedia.org/wiki/UTF-8>

## Unicode:

Liste avec tous les caractères qui existent,  
y compris les caractères chinois, japonais, ...  
(env. 1 million théoriquement, env. 120'000 utilisés)

## UTF-8:

Format de codage des signes Unicode;  
utilise **entre 1 et 4 bytes** pour un caractère

# Tableau des caractères Unicode

The screenshot shows the 'Characters' application window. On the left is a sidebar with various character categories. The main area displays a table of characters with columns for Unicode, Title, and Category. Below this is a grid of characters, with the 'Cyrillic' category selected. The grid shows characters from U+0440 to U+04E0, including Cyrillic letters with diacritics and special characters.

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	с	Т	т	У	у	Ф	ф	Х	х	Ц	ц	Ч	ч	Ш	ш
0450	Щ	щ	Ъ	ъ	Ы	ы	Ь	ь	Э	э	Ю	ю	Я	я		
0460	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ
0470	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ
0480	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ
0490	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ
04A0	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ
04B0	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ
04C0	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ
04D0	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ
04E0	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ

# Lire un fichier texte

---

## Démo

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

En fait, on n'écrit pas ceci!

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

On ouvre ce fichier en lecture (*read*)

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

```
print(contents)
```

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

Lecture!

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

# Écrire un fichier texte

---

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

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(...)`

```
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?**
  - Méthode `split()`, qui sépare une chaîne de caractère en une liste de sous-chaînes
  - **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*

Démo

```
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
1978;94;Days of Heaven;Drama;Gere, Richard;Adams, Brooke;Malick, Terrence;14;No;NicholasCage.png
1983;140;Octopussy;Action;Moore, Roger;Adams, Maud;Glen, John;68;No;NicholasCage.png
1984;101;Target Eagle;Action;Connors, Chuck;Adams, Maud;Loma, José Antonio de la;14;No;NicholasCage.png
1989;99;American Angels: Baptism of Blood, The;Drama;Bergen, Robert D.;Adams, Trudy;Sebastian;14;No;NicholasCage.png
1985;104;Subway;Drama;Lambert, Christopher;Adjani, Isabelle;Besson, Luc;6;No;NicholasCage.png
1990;149;Camille Claudel;Drama;Depardieu, Gérard;Adjani, Isabelle;Nuytten, Bruno;32;No;NicholasCage.png
1982;188;Fanny and Alexander;Drama;Ahlstedt, Börje;Adolphson, Kristina;Bergman, Ingmar;81;No;NicholasCage.png
```

# Lecture et modélisation

---

```
from dataclasses import dataclass
from typing import List
```

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

Lecture de tout le fichier d'un coup

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

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

```
@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

# Création des objets depuis le texte lu

```
movies: List[Movie] = []  
substitutions = {  
    "&": "and",  
    "'": "'",  
    "vs.": "vs",  
}  
for line in lines[2:]:  
    parts = line.split(";")  
  
    duration_str = parts[1]  
    if duration_str == "":  
        duration = 0  
    else:  
        duration = int(duration_str)  
  
    title = parts[2]  
    if ", " in title:  
        title = " ".join(title.split(", ")[:-1])  
    for key, value in substitutions.items():  
        title = title.replace(key, value)  
  
    movie = Movie(title, int(parts[0]), duration, parts[8] == "Yes")  
    movies.append(movie)
```

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

Les substitutions de caractères qu'on voudra faire dans le titre des films

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

Chaque ligne est séparée selon les points-virgules qu'elle contient

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

On «nettoie» le titre, on applique les substitutions

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

# Traitement des données et écriture

---

```
def criterion(m: Movie) -> int:  
    return m.duration
```

```
movies.sort(key=criterion)
```

On trie les films selon un critère de tri donné par ce que cette fonction retourne pour chaque film; ici, sa durée

On prépare un fichier de sortie pour y écrire des résultats

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

```
for movie in movies:  
    if movie.has_awards:
```

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

```
        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")
```

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 casse
- Test de préfixe, suffixe, contenance

- Split

- Sépare un string en une liste selon un délimiteur

- Join

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

- Replace

- Rechercher-remplacer

```
text = "..."
```

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

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

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

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

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

*Pas discuté: les expressions régulières, un moyen très puissant de manipuler du texte*

# Résumé Cours 12

---

- 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 **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