À faire individuellement ou par petits groupes de deux ou trois.

Exercice 1. I/O sur la console et conditions

I/O (ou IO) signifie *Input/Output*, soit «entrées/sorties». Pour cet exercice, nous allons lire et écrire des lignes de texte sur la console, c'est-à-dire la «zone des résultats» où s'affiche le résultat des **`print()`**.

Ouvrez VS Code, puis ouvrez votre workspace s'il ne se rouvre pas tout seul. Pour rappel, il s'agit toujours du dossier dans lequel vous avez stocké vos fichiers Python. Si vous travaillez sur les machines virtuelles de l'EPFL, il est dans `~/Desktop/myfiles/ICCProgrammation_xyz`. Créez-y ensuite des nouveaux fichiers Python pour ces exercices, en prenant garde à ce que leurs noms se terminent bien toujours par **`.py`**.

(a) Ajoutez ce code dans votre nouveau fichier (tel quel — sans écrire votre prénom ou nom directement dans le code Python).

Examinez ce code. Repérez la fonction **`input()`**: ce qu'elle va faire, c'est arrêter l'exécution du code et attendre que l'utilisateur de votre programme tape quelque chose sur le terminal, suivi de la touche Retour ou Enter. C'est le texte tapé à ce moment-là qui sera retourné par la fonction. Ici, donc, ce texte tapé sera stocké dans la variable **`full_name`**.

Faites tourner votre programme. Vous verrez qu'il s'arrête après la première ligne, car il attend ensuite ce que vous tapez dans le terminal pour continuer. Tapez donc votre prénom et nom sur le terminal dans la zone de résultats en bas de la fenêtre. Une fois que vous avez tapé Enter ou Retour, les deux dernières lignes s'exécutent.

```
print("Bonjour, veuillez entrer votre prénom et nom:")
full_name: str = input()
print(f"Le nom que vous avez tapé est {full_name}.")
```

(b) En partant du principe que le prénom est séparé du nom par une[1] espace (`" "`), créez deux nouvelles variables **`first_name`** et **`last_name`**, et donnez-leur comme valeur le nom et le prénom en les extrayant avec du code de votre variable **`full_name`**. Utilisez pour cela la méthode **`index()`** pour repérer la position de l'espace, puis le slicing pour extraire des sous-strings en fonction de la position de l'espace.

(c) Essayez de taper uniquement votre prénom (sans y inclure d'espace) et lancez votre programme. Que se passe-t-il? Modifiez votre programme pour faire en sorte que si l'utilisateur ne tape que son prénom (c'est-à-dire, si le string **`full_name`** ne contient pas d'espace), on affiche un message d'erreur, en utilisant une condition — un `if`.

Exercice 2. Structures conditionnelles et opérateur «modulo»

Une année n est bissextile si elle respecte l'un ou l'autre des deux critères suivants :

- n est divisible par 4 sans être divisible par 100 ;
- n est divisible par 400.

a) Observez le code suivant, recopiez-le puis exécutez-le.

```
mod_1: int = 5 % 2
mod_2: int = 18 % 7
mod_3: int = 24  % 8
print(mod_1)
print(mod_2)
print(mod_3)
```

À votre avis, quel est le rôle de l'opérateur `%` (qui se lit «modulo») ? En cas de doute, demandez à l'enseignant ou à un·e assistant·e.

[1] Le mot «espace» est bel et bien féminin dans ce contexte.

b) Reformulez la définition d'une année bissextile en mettant en évidence les mots SI, ALORS, ainsi que d'éventuelles opérations conduisant à une valeur booléenne.

c) En utilisant l'opérateur modulo, écrivez un programme qui détermine puis affiche si les années 2024 et 2025 sont bissextiles ou non. Vous pourrez utiliser une seule variable et exécuter plusieurs fois le programme avec des valeurs différentes pour cette variable.

Exercice 3. Boucles dans les strings

(a) Créez un nouveau fichier Python et insérez-y cette ligne telle quelle sans la modifier:

```
line: str = input("Veuillez taper quelque chose: ")
```

Complétez ensuite le code: à l'aide d'une boucle **`for-in`** et en utilisant les fonctions **`len()`** et **`range()`**, affichez chaque caractère du string **`line`** individuellement avec la fonction **`print()`**. Utilisez pour cela le fait que, si **`my_string`** est un string, alors **`my_string[i]`** est le caractère à la position i de ce string.

(b) Modifiez votre code en simplifiant la boucle: utilisez pour cela le fait que le **`for-in`** peut aussi marcher directement sur une valeur de type string plutôt que sur une valeur retournée par la fonction **`range()`**. Par exemple: **`for c in my_string`** va itérer sur tous les caractères du string **`my_string`**.

(c) Écrivez une boucle qui compte le nombre de caractères en majuscule et en minuscule du string **`line`** et qui affiche ces deux nombres.

Pour cela, vous pouvez utiliser les méthodes **`isupper()`** et **`islower()`**. Elles vous renvoient un `bool` qui indique si le string sur lequel elles sont appelées est en majuscule ou minuscule, respectivement.

(d) Modifiez votre programme pour qu'il affiche aussi le nombre de caractères qui ne sont ni en majuscule, ni en minuscule (par exemple, des chiffres ou autres symboles).

Exercice 4. Boucles et manipulation de chaînes de caractères, suite

Considérez le programme suivant :

```
course_title: str = "information, calcul, communication"
for idx, character in enumerate(course_title):
    print(f"course_title[{idx}] = {character}")
```

(a) Anticipez le résultat de ce programme sans l'exécuter. Puis vérifiez votre réponse en exécutant ce programme.

(b) Créez une variable **`capitalized_course_title`** qui contient, pour le moment, la chaîne de caractères vide `""`. Nous souhaitons que cette nouvelle variable contienne, à la fin de l'exécution du programme, la chaîne de caractères `"Information, Calcul, Communication"`.

(i) Comment construire, lettre après lettre, le contenu de la variable **`capitalized_course_title`** pour que celui-ci soit identique au contenu de la variable **`course_title`** ?

(ii) Quelles conditions permettraient de ne capitaliser que les premières lettres de chaque mot ? Vous pouvez prendre en compte l'indice des lettres (en particulier pour la première lettre) ou les caractères précédant ceux devant être écrits en majuscule.

(iii) Implémentez ces conditions pour construire, lettre après lettre, le contenu souhaité de la variable **`capitalized_course_title`**. Vous pourrez utiliser la *méthode* **`.upper()`** vue dans la Série 1.

(c) Affichez, après la dernière itération de la boucle, le contenu de la variable **`capitalized_course_title`**.

Exercice 5. Un petit interpréteur interactif

(a) Depuis la page Moodle du cours, copiez et collez ce code dans un nouveau fichier Python.

Que fait chaque ligne de ce programme?

Comment est-ce que la boucle fonctionne? Quelle est la condition? Quand et comment la variable de boucle est-elle modifiée?

```python
should_continue: bool = True

while should_continue:
    print("Je vais évaluer un calcul pour vous.")

    number1_string: str = input("Tapez le premier nombre: ")
    number1: float = float(number1_string)

    number2_string: str = input("Tapez le second nombre: ")
    number2: float = float(number2_string)

    operation: str = input("Tapez l'opération: ")
    if operation == "+":
        result = number1 + number2
        print(f"{number1} + {number2} = {result}")
    else:
        print(f"Désolé, je ne connais pas 'lopération '{operation}'")

    maybe_quit: str = input("Tapez 'q' pour quitter, ou autre chose pour recommencer: ")
    if maybe_quit == "q":
        print("Bye!")
        should_continue = False
```

(b) Modifiez le programme pour qu'il effectue aussi une addition si on tape `"plus"` plutôt que `"+"` comme opération.

(c) Modifiez le programme pour qu'il puisse aussi évaluer des soustractions, multiplications et divisions.