EPFL

ICC

Programmation

CGC/SIE, **Cours 3**

4 mars 2023

Jean-Philippe Pellet

Previously, on Programmation…

- Types de base en Python: int, float, str, bool
- Déclaration d'une variable avec valeur initiale
- Conversion entre ces types; comparaisons pour obtenir des bool
- Méthodes, fonctions et slicing pour calculer des valeurs dérivées
- Google/StackOverflow… pour rechercher comment faire quelque chose
- Conditions pour exécuter du code selon la valeur d'une expression booléenne:
  - `if <condition>: ...`
  - `if <condition>: ... else: ...`
  - `if <condition1>: ... elif <condition2>: ... else: ...`
- Boucles pour exécuter du code plusieurs fois:
  - Boucle `while <condition>: ...`
  - *Aujourd'hui:* Boucle `for i in range(...): ...`

Répétition — *while* et *for*

```
i = 7
while i < 100:
    print(i)
    i = i + 7
```

Initialisation

Test

Que faire à chaque itération

Mise à jour de la variable de boucle

```
for i in range(7, 100, 7):
    print(i)
```

Pour chaque i dans ce *range*

Première valeur

Borne supérieure non incluse

Incrément

Que faire à chaque itération

1. Initialisation
2. Test. Si test true: continuer à 3. Sinon, sauter à 5.
3. Exécuter le corps de la boucle
4. Mise à jour de la variable de boucle; continuer à 2.
5. Sortir de la boucle et continuer l'exécution du code après la fin de boucle

Répétition — *if ... else*

- Deux floats. Comment afficher une phrase si le premier est plus petit que le second?

```
value1: float = 0.1
value2: float = 1.0
if value1 < value2:
    print(f"{value1} est plus petit que {value2}")
else:
    print(f"{value1} n'est pas plus petit que {value2}")
```

- Qu'est-ce que ce code affiche?

```
print("début")
if True:
    print("A")
if False:
    print("B")
if not True:
    print("C")
if True and False:
    print("D")
if True or False:
    print("E")
print("fin")
```

Répétition — Boucle *while*

```
while <bool_condition>:
    # code à exécuter tant que la condition est True

    # éventuellement, si on veut tout de suite
    # arrêter la boucle sans se préoccuper de la
    # condition, il y cette instruction:
    if <bool_condition>:
        break
```

Optionnel! Beaucoup de boucles ne l'ont pas.
Un *break* ne peut être que dans une boucle, et
n'a a priori de sens que s'il est dans un *if*

Répétition — *while* et *for-in*

- Comment afficher tous les nombres pairs de 2 à 100 avec une boucle *while*?

```
i = 2
while i <= 100:
    print(i)
    i += 2
```

- Et avec une boucle *for-in*?

```
for i in range(2, 101, 2):
    print(i)
```

- Qu'affiche ce code?

```
target = 2
while target < 100:
    target = target * 2
print(target)
```

- Qu'affiche ce code?

```
title = "code"
for _ in title:
    print(title)
```

Cours de cette semaine

Fonctions

Motivation

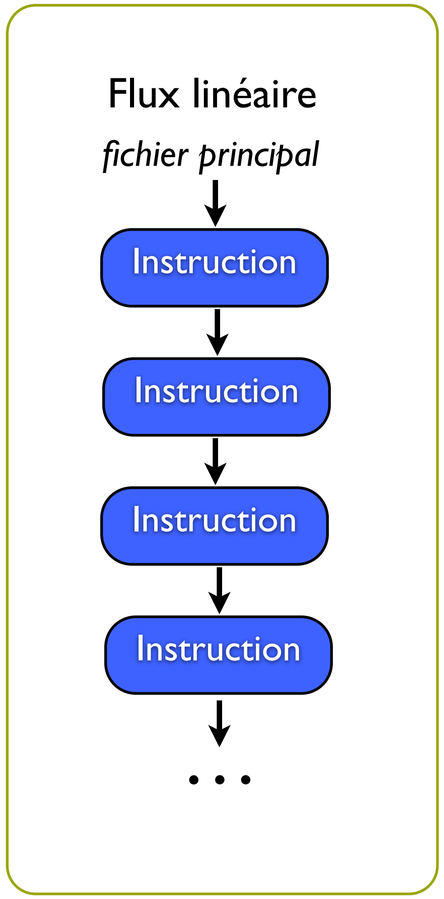
- Fonctions:

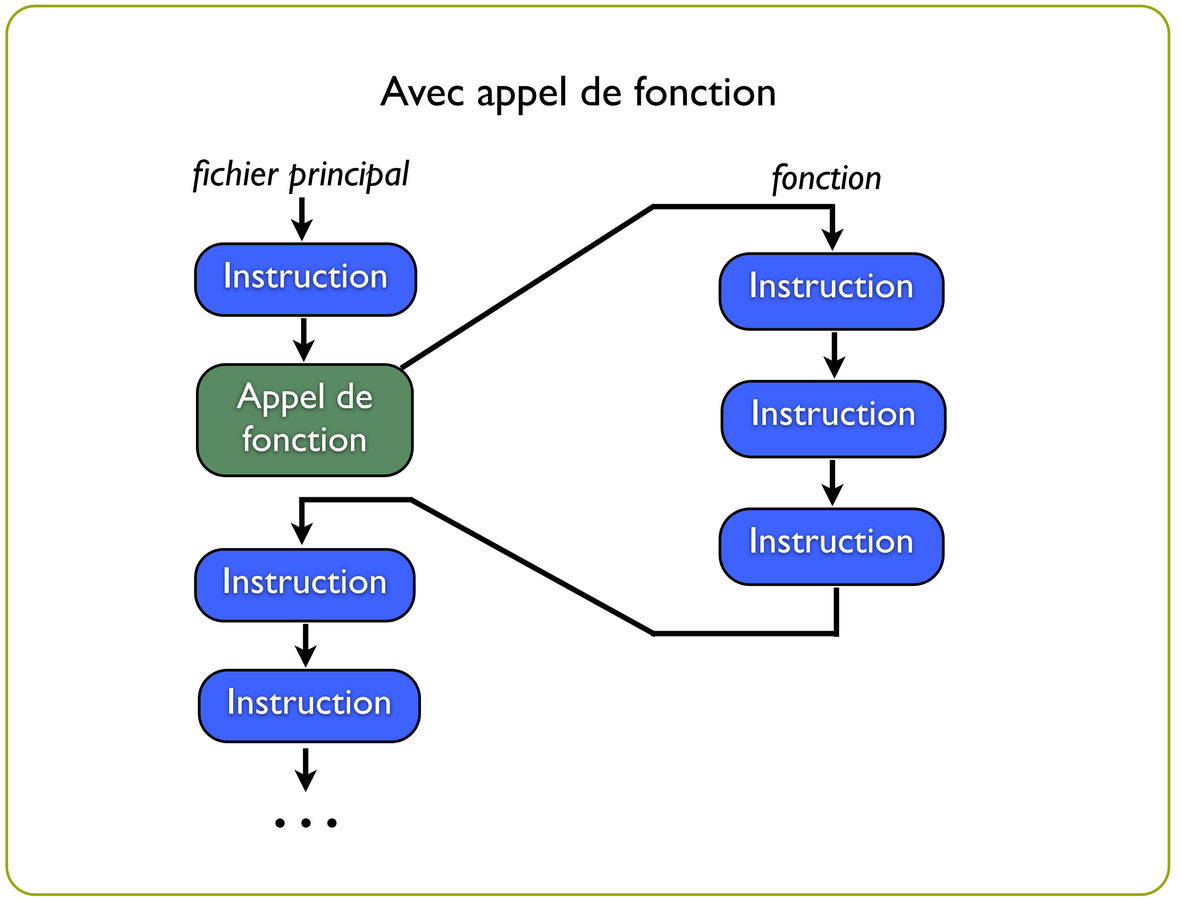
  «Je fais plusieurs fois le même calcul, mais je ne veux pas écrire mon code plusieurs fois»

  ou:

  «Ce morceau de code est important et doit être "factorisé" à part avec son propre nom»

Fonctions





Fonction ≈ sous-algorithme

Fonctions connues

```
print("texte")

len("contenu")

range(2, 100, 2)

math.floor(3.1416)

"paul".upper()
```

Fonction dans un autre module

Fonction appelée sur une variable: on parle plutôt de **méthode**

Fonctions: exemple

```
def analyze_string(s: str) -> None:
    print(f"Analyse du string '{s}'")
    print(f"{len(s)} caractères")
    print(f"En majuscules: {s.upper()}")
    print(f"En minuscules: {s.lower()}")
    print("--")
```

Définition de la fonction

Paramètre

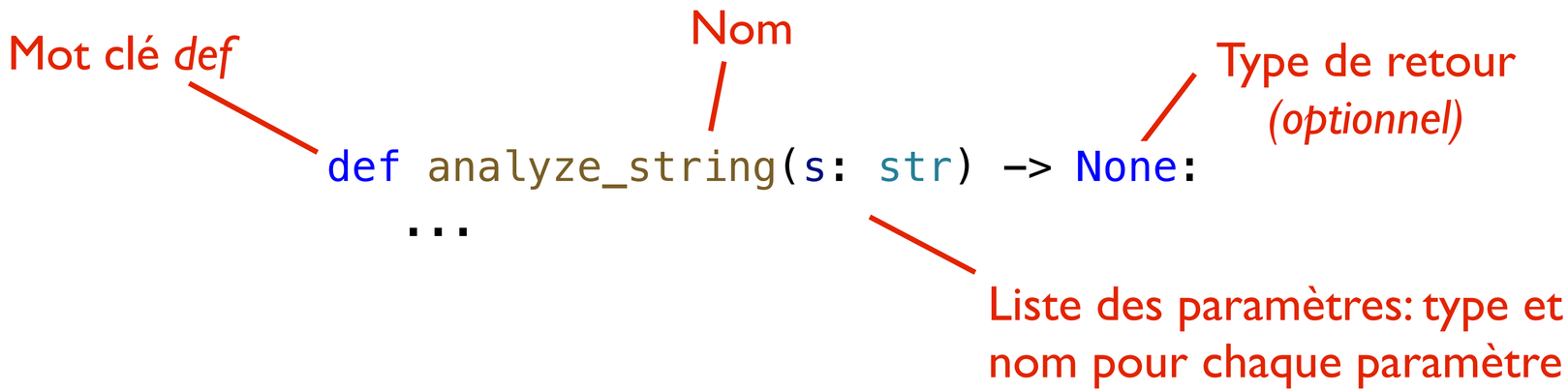
```
analyze_string("Bonjour")
analyze_string("programmation")
analyze_string("exercice")
```

Appel de la fonction (après la définition si dans le même fichier)

Les noms des paramètres sont libres et sont indépendants des noms de variables utilisés, le cas échéant, lors de l'appel de la fonction.

On peut considérer les paramètres comme des variables locales normales dont la valeur est automatiquement attribuée lors de l'appel de fonction.

Anatomie d'une fonction



- Une fonction a un nom
- Une fonction peut calculer une valeur de retour (qui a un type), et a une liste de paramètres
- Chaque paramètre a aussi un nom et un type; il se comporte comme une variable dans la fonction. Plusieurs paramètres sont séparés par une virgule.
- `None` signifie qu'on ne renvoie aucune valeur. Si on désire renvoyer une valeur, on spécifie un type *avant* les deux-points et on utilise le mot clé `return` pour signaler la valeur de retour.

Calculer l'aire d'un cercle

Démo

Calculer l'aire d'un cercle

«J'utilise ce module Python»

«J'ai besoin d'un paramètre de type float, que je vais appeler *r*»

«Je retourne un float»

```python
import math

def calculate_circle_area(r: float) -> float:
    area = math.pi * r * r
    return area

area1: float = calculate_circle_area(2.0)
print(area1)

area2: float = calculate_circle_area(5.0)
print(area2)

area3: float = calculate_circle_area(10.5)
print(area3)
```

«Mon résultat est la valeur qui suit»

Le reste du code, à part *dans* la fonction, s'en fiche que le paramètre soit appelé *r*

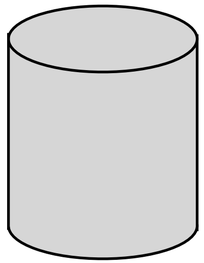
Modèle

```
def nom_fonction(param1: type1, param2: type2, ...) -> type_de_retour:
    <instructions>
    return valeur_de_retour
```

(souligné = à remplacer par votre propre code)

Exercice en 5 minutes

Écrivez une fonction qui retourne le volume d'un cylindre de rayon *r* et hauteur *h*.



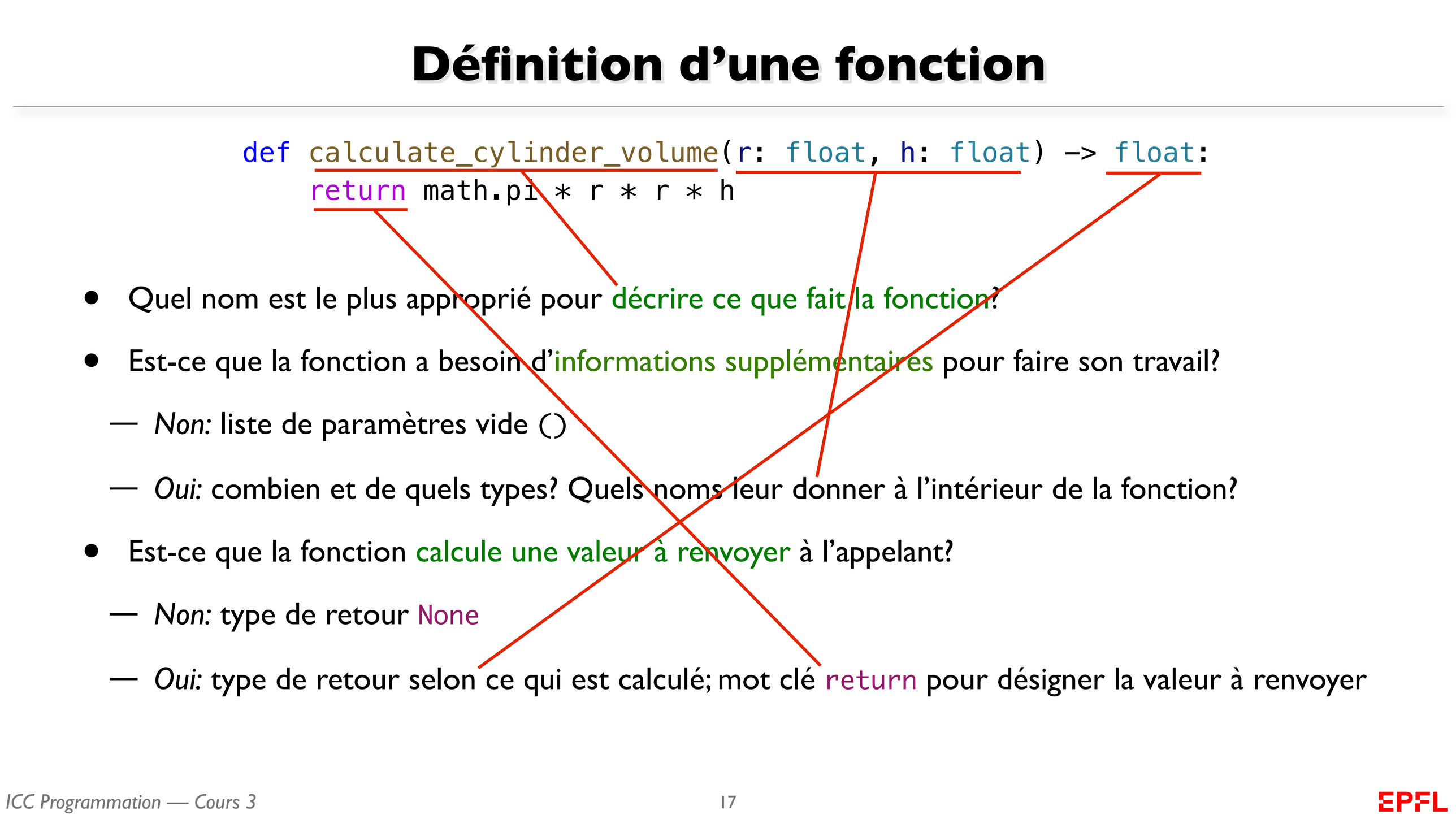
$$V = \pi r^2 h$$

Indice: de combien de paramètres avez-vous besoin?
Quel type retournez-vous?

```
def calculate_cylinder_volume(r: float, h: float) -> float:
    return math.pi * r * r * h
```

Définition d'une fonction

```
def calculate_cylinder_volume(r: float, h: float) -> float:
    return math.pi * r * r * h
```



- Quel nom est le plus approprié pour décrire ce que fait la fonction?
- Est-ce que la fonction a besoin d'informations supplémentaires pour faire son travail?
  - *Non:* liste de paramètres vide ()
  - *Oui:* combien et de quels types? Quels noms leur donner à l'intérieur de la fonction?
- Est-ce que la fonction calcule une valeur à renvoyer à l'appelant?
  - *Non:* type de retour None
  - *Oui:* type de retour selon ce qui est calculé; mot clé `return` pour désigner la valeur à renvoyer

Autre exemple I

Écrivez une méthode `is_all_upper` *qui indique si un string est tout en majuscules.*

```
def is_all_upper(some_string: str) -> bool:
    if some_string.upper() == some_string:
        return True
    else:
        return False

test_string = "ALLUPPERCASE"
if is_all_upper(test_string):
    print("ce string est tout en majuscules")
else:
    print("ce string n'est pas tout en majuscules")
```

Variante plus concise:

```
def is_all_upper(some_string: str) -> bool:
    return some_string.upper() == some_string
```

Autre exemple II

Écrivez une méthode `repeat_string` *qui crée un string formé de n répétitions d'un string donné.*

```
def repeat_string(string: str, n: int) -> str:
    result = ""
    for _ in range(n):
        result += string
    return result
```

```
verse = repeat_string("ta ", 3) + "taaaa\n"
fifth = repeat_string(verse, 2)
print(fifth)
```

Variante plus concise:

```
verse = ("ta " * 3) + "taaaa\n"
print(verse * 2)
```

«Multiplier» un string = le répéter ce nombre de fois

Valeurs par défaut des paramètres

«Si ces paramètres ne sont pas précisés lors de l'appel, utilise ces valeurs»

```
def say_hello(to: str = "John", n: int = 1) -> None:
    hello = " hello" * n
    print(f"Oh,{hello}, {to}!")
```

```
say_hello()                      # valeurs pas précisée: on utilise to="John" et n=1

say_hello("John")                # le premier paramètre est précisé, et n=1

say_hello(to = "John")           # même effet ici

# say_hello(3)                   # impossible, le premier paramètre est de type str

say_hello(n = 2)                 # OK, car le paramètre est nommé

say_hello(n = 3, to = "James")   # on peut réordrer les paramètres nommés
```

Résumé Cours 3

- Les fonctions permettent d'isoler une série d'instructions du reste du code
- Chaque fonction a un nom, un type de retour, et une liste de paramètres (chacun avec un nom et un type)
  - Chaque paramètre peut avoir une valeur par défaut si non spécifié lors de l'appel
- Si le type de retour n'est pas None, on renvoie une valeur avec le mot clé return
- return cause la fin de l'exécution du reste du code de la fonction