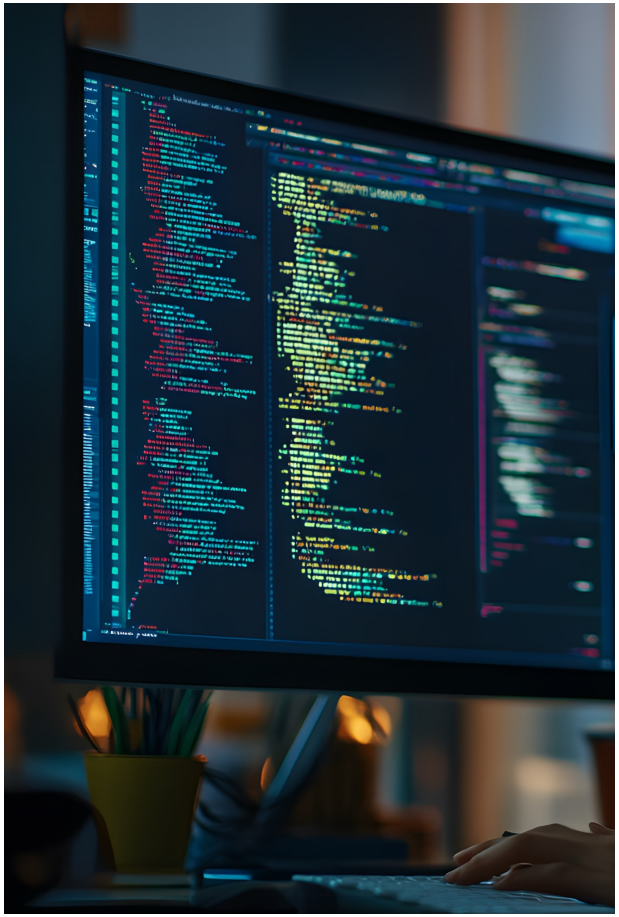
Information, Calcul et Communication

CS-119(k) ICC – Programmation
Semaine 3

Rafael Pires
rafael.pires@epfl.ch

Lausanne, 03.03.2025

Précédemment, dans… ICC-P



- Types de base en Python: `int`, `float`, `str`, `bool`
- Méthodes, fonctions et slicing
  pour calculer des valeurs dérivées
- Branchements pour exécuter du code selon la valeur d'une expression booléenne

```
if <condition>:
    …
```

```
if <condition>:
    …
else:
    …
```

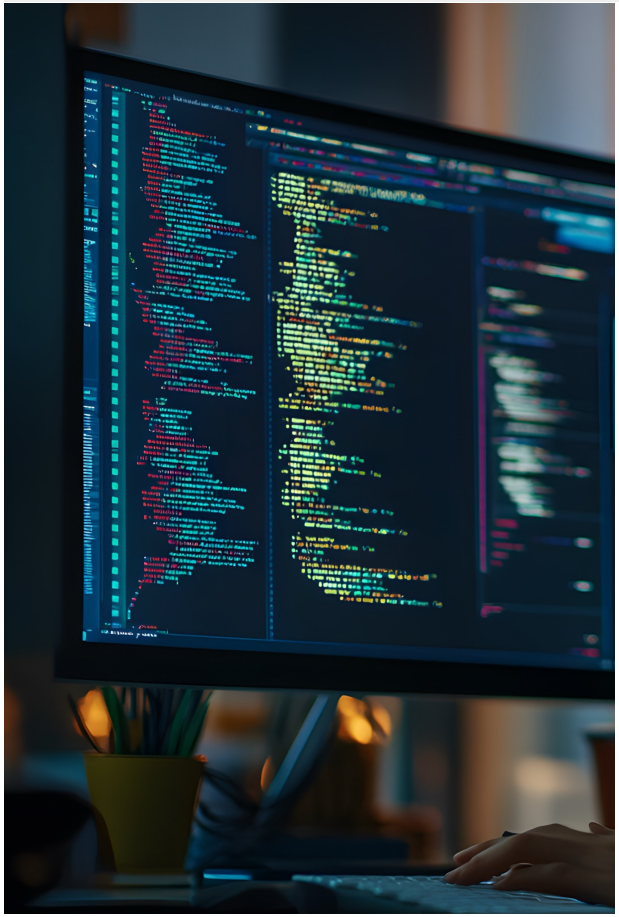
```
if <condition1>:
    …
elif <condition2>:
    …
else:
    …
```

- Boucles pour exécuter du code plusieurs fois

```
while <condition>:
    …
```

```
for i in range(…):
    …
```

Précédemment, dans... ICC-P (exercices)



- **input**

```
number1_str: str = input("Tapez le premier nombre: ")
number1: float = float(number1_str)
```

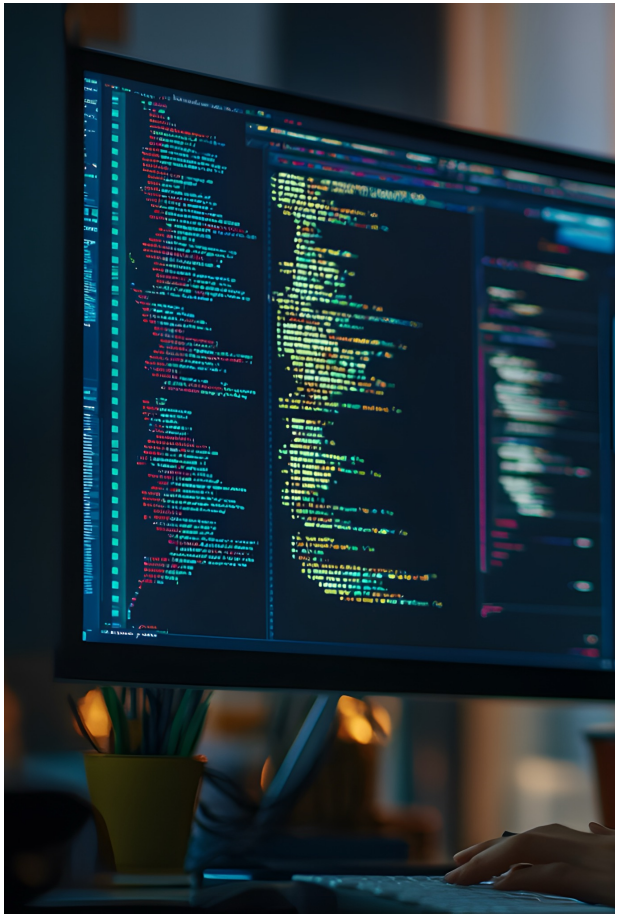
- **index**

```
space_pos: int = full_name.index(" ")
```

- **enumerate**

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

Précédemment, dans... ICC-P



- **Boucle while**

```
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

- **Boucle for**

Variable de boucle

- Première valeur
- Borne supérieure non incluse
- Incrément

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

- Que faire à chaque itération

Attention aux indices !

Indices et comparateur d'inégalité

ICC-T : vendredi ≠ **ICC-P : lundi**

Itération		Boucle conditionnelle	Itération		Boucle conditionnelle
pour i allant de 1 à 10, (répeter …)	≡	i ← 1 tant que $i \leq 10$ (répeter …) i ← i + 1	for i in range(0,10): (répeter …)	≡	i ← 0 tant que $i < 10$ (répeter …) i ← i + 1

Nombre de itérations

pour i allant de 1 à *n*,
(répeter …)

n – 1 + 1 = n

=

for i in range(0,*n*):
(répeter …)

n – 0 = n

Attention aux indices !

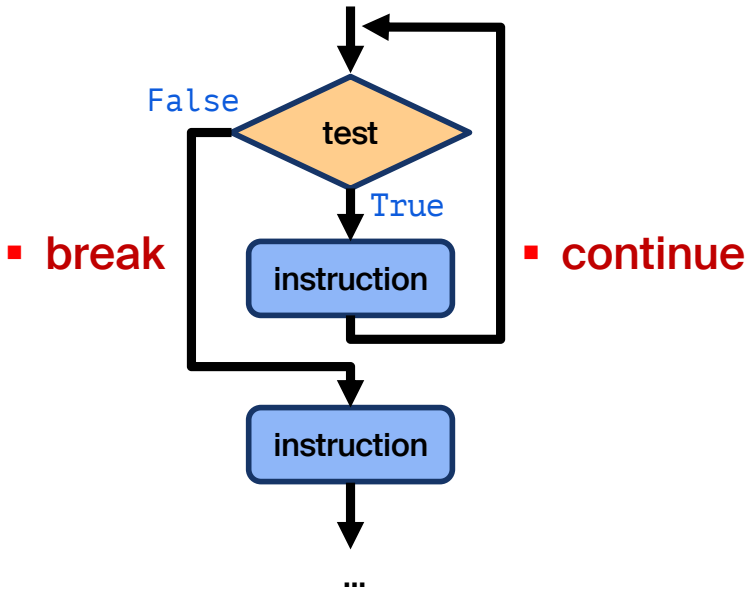
Valeur minimale

entrée : liste ***L*** de nombres entiers, de taille ***n***
sortie : la (ou une des) valeur(s) minimale(s) de la liste

```
min ← L(1)
Pour i allant de 2 à n
        Si L(i) < min
                min ← L(i)
Sortir : min
```

```
ma_liste = [13, 47, 18, 15, 11, 19, 46, 18, 15]
min = ma_liste[0]
for i in range(2, len(ma_liste)):
    if ma_liste[i] < min:
        min = ma_liste[i]
print(min)
```

Interruption et saut dans les boucles



- break
- continue

Interruption et saut dans les boucles

- **break**

```
for i in range(10):
    if i == 5:
        break # Stoppe la boucle dès que i vaut 5
    print(i, end=" ")
print()

# Sortie attendue: 0 1 2 3 4
```

- **continue**

```
for i in range(10):
    if i == 5:
        continue # Ignore le reste du bloc et passe à l'itération suivante
    print(i, end=" ")
print()

# Sortie attendue: 0 1 2 3 4 6 7 8 9
```

Quiz

- Quelle valeur doit remplacer '?' ?

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

≡

```
for i in range(2, ?, 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)
```

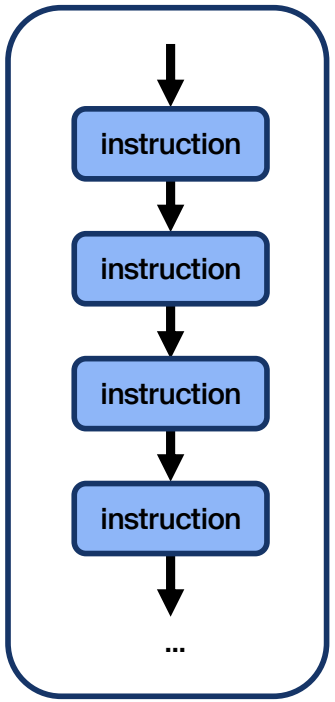
Fonctions

Fonctions : motivation

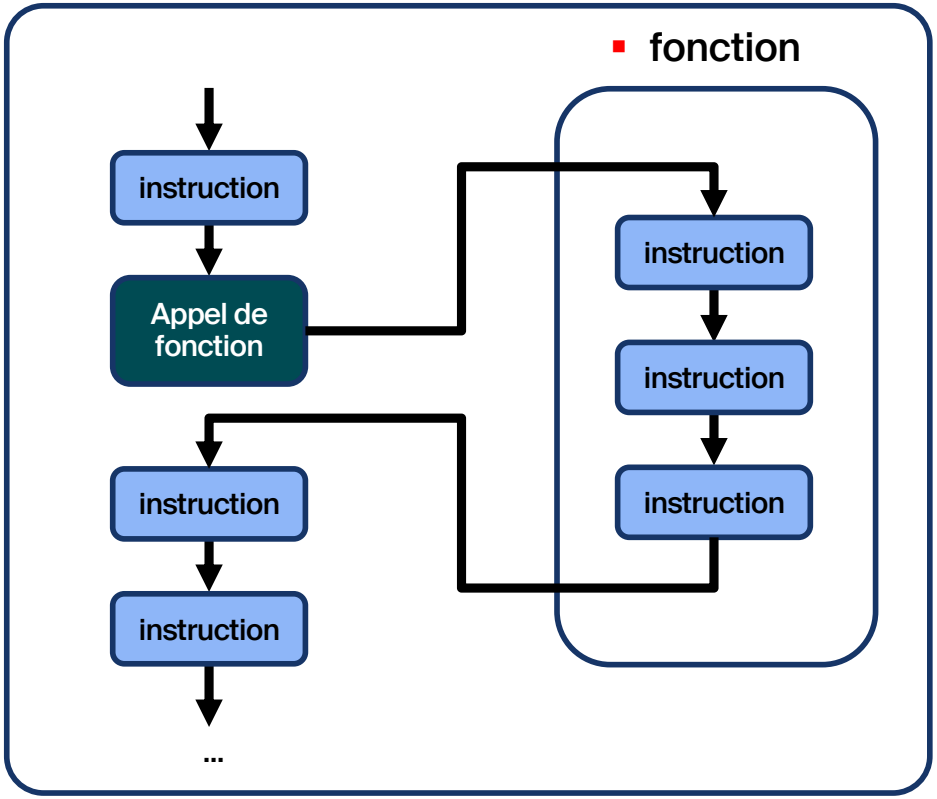
- J’ai besoin de répéter un calcul sans écrire plusieurs fois le même code.
- Ce bloc de code est utile ailleurs, autant lui donner un nom et l’isoler.
- Si je dois modifier ce calcul plus tard, mieux vaut ne le changer qu’à un seul endroit.
- Mon code devient plus clair et plus réutilisable en extrayant cette partie dans une fonction.

Fonctions

- Flux linéaire



- Avec appel de fonction



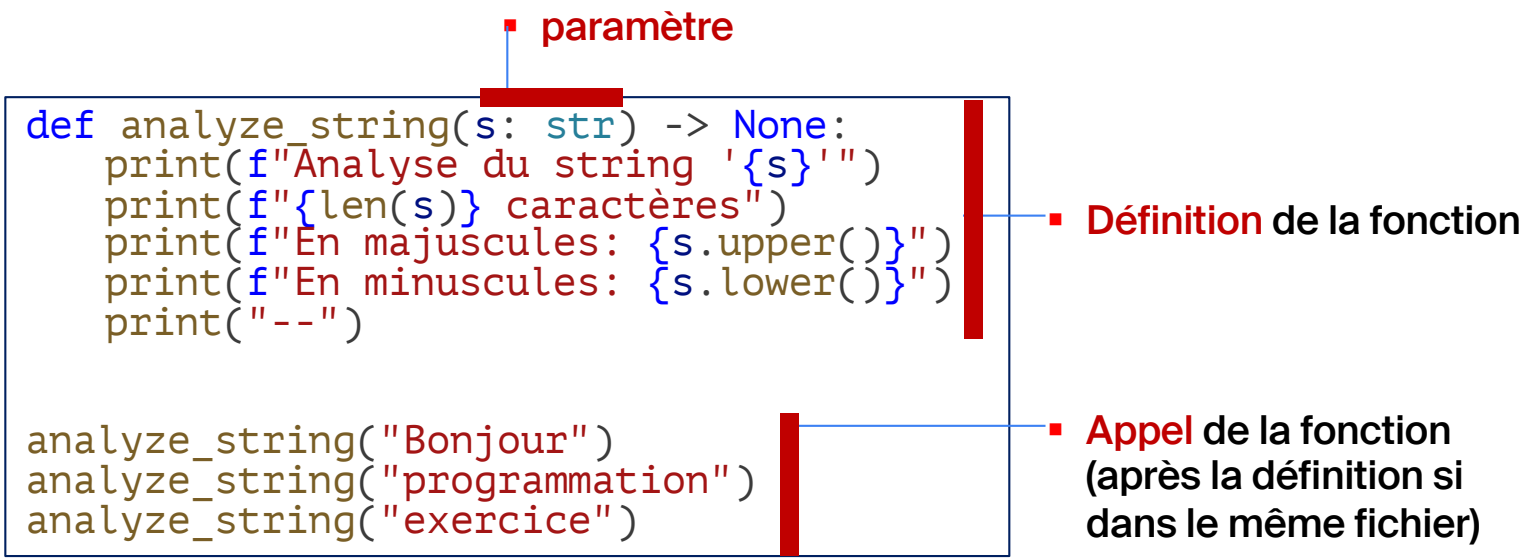
Fonctions connues

```
print("texte")

len("contenu")

range(2, 100, 2)

math.floor(3.1416)

"paul".upper()
```

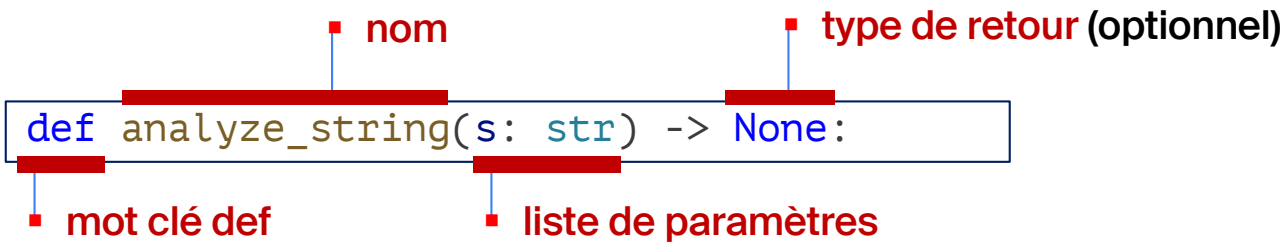
- fonction dans un autre module
- méthode (fonction associée à un type)

Fonctions : exemple

- Les noms des paramètres sont indépendants des variables passées à la fonction)
- Les paramètres sont des variables locales initialisées automatiquement lors de l'appel de la fonction.



Anatomie d'une fonction



- Une fonction a un **nom**
- Une fonction prend une liste de **paramètres** et peut **retourner** une valeur d'un certain type.
- Chaque paramètre a un nom et un type (plusieurs paramètres sont séparés par une virgule). Il agit comme une **variable** dans la fonction.
- `None` indique l'absence de valeur de retour. Pour en renvoyer une, on précise son type avant les deux-points et on utilise `return`.

Modèle

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

Calculer l'aire d'un cercle

- J'utilise ce module
- 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.
- En dehors de la fonction, peu importe que le paramètre s'appelle `r`.

ICC-T 02 : Tri par insertion

Tri par insertion

entrée : liste *L* de taille *n*
sortie : liste *L* triée dans l'ordre croissant

```
Pour i allant de 2 à n :
        Si L(i) < L(i-1), alors
                L ← insérer(L, i)
Sortir : L
```

insérer

entrée : liste *L*, indice *i*
sortie : liste *L* avec l'élément L(i) bien placé

```
Tant que i > 1 et L(i) < L(i-1) :
        L ← permuter(L, i, i-1)
        i ← i-1
Sortir : L
```

permuter

entrée : liste *L*, indices *j* et *k*
sortie : liste *L* avec les éléments L(j) et L(k) permutés

```
temp ← L(j)
L(j) ← L(k)
L(k) ← temp
Sortir : L
```

Valeurs par défaut des paramètres

- Si les paramètres `to` et `n` ne sont pas précisés lors de l'appel, on utilise les valeurs standards

```
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 réordonner les paramètres nommés
```

Conseils

Conseil : Utilisez des noms clairs et des types

```python
def f(a, b):
    return a * b / 2

x = 5
y = 10
z = f(x, y)
print(z)
```

```python
def aire_triangle(base: float, hauteur: float) -> float:
    return base * hauteur / 2

base_triangle: float = 5
hauteur_triangle: float = 10
aire: float = aire_triangle(base_triangle, hauteur_triangle)
print(aire)
```

En programmation, les noms, c'est comme les blagues :
si tu dois expliquer, c'est qu'ils sont mauvais !

Résumé Cours 3 – ICC-P

- Les mots clés `break` et `continue` permettent de changer le flux d'une boucle n'importe quand
- 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

rafael.pires@epfl.ch

Merci