□□ La Récursivité

Objectif : Comprendre le principe de récursivité et savoir écrire des fonctions récursives en Python.

□ Concept

Une fonction est **récursive** si elle **s'appelle elle-même**. Pour qu'elle s'arrête, elle doit toujours avoir :

- un **cas de base** — condition qui stoppe la récursion et retourne directement une valeur,
- un **appel récursif** — qui se rapproche du cas de base à chaque fois.

```python
def fonction(n) :
    if n == cas_de_base :      # ← condition d'arrêt
        return valeur_initiale
    return fonction(n - 1)     # ← appel récursif
```

□□ Sans cas de base, la fonction s'appelle à l'infini et le programme plante.
□ Chaque appel récursif doit **se rapprocher** du cas de base — sinon la récursion ne s'arrête jamais.

Opération	Code Python
Définir une fonction	`def f(n):`
Cas de base	`if n == 0: return valeur`
Appel récursif	`return f(n - 1)`
Retourner une valeur	`return expression`

□ Bloc 1 — Fonctions sur les entiers

□□ Exercice 1 — Factorielle

□ **Objectif :** Écrire une fonction récursive **fact(n)** qui calcule n! (factorielle de n). Rappel : 0! = 1, n! = n × (n-1)!

□ Trace d'exécution pour fact(4) :

Appel	Retourne
`fact(4)`	`4 × fact(3) = 24`
`fact(3)`	`3 × fact(2) = 6`
`fact(2)`	`2 × fact(1) = 2`
`fact(1)`	`1 × fact(0) = 1`
`fact(0)`	`1 ← cas de base`

```terminal
fact(5) = › 120
fact(0) = › 1
```

□□ Exercice 2 — Puissance (sans l'opérateur **)

□ **Objectif :** Écrire une fonction récursive **puissance(x, n)** qui calcule x^n sans utiliser **. Rappel : $x^0 = 1$, $x^n = x \times x^{n-1}$

```terminal
puissance(2, 10) = › 1024
puissance(3, 4) = › 81
puissance(5, 0) = › 1
```

□□ Exercice 3 — Somme des entiers de 1 à n

□ **Objectif :** Écrire une fonction récursive **somme(n)** qui calcule 1 + 2 + 3 + … + n. Rappel : somme(0) = 0, somme(n) = n + somme(n-1)

```terminal
somme(5) = › 15
somme(10) = › 55
somme(0) = › 0
```

□□ Exercice 4 — Multiplication par additions récursives

□ **Objectif :** Écrire une fonction récursive **multi(a, b)** qui calcule a × b en n'utilisant que l'addition. Rappel : a × 1 = a, a × b = a + multi(a, b-1)

```terminal
multi(3, 5) = › 15
multi(7, 4) = › 28
multi(6, 1) = › 6
```

□□ Exercice 5 — Lecture de code récursif

□ **Objectif :** Lire la fonction suivante et répondre aux questions sans l'exécuter.

```python
def f(a, b) :
    if b == 1 :
        return a
    return a + f(a, b - 1)
```

• Quel est le cas de base ?

• Que retourne **f(3, 5)** ? Déroulez à la main.

• Que calcule cette fonction en général ?

□ Bloc 2 — Suites définies récursivement

Certaines suites mathématiques se définissent à partir de leurs termes précédents — elles s'implémentent **naturellement** de façon récursive.

□□ Exercice 6 — Suite de Fibonacci

□ **Objectif :** Écrire une fonction récursive **fib(n)** qui retourne le n-ième terme de la suite de Fibonacci. Rappel : fib(0) = 0, fib(1) = 1, fib(n) = fib(n-1) + fib(n-2)

□ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 … — deux cas de base ici !

```terminal
fib(0) = › 0
fib(1) = › 1
fib(7) = › 13
fib(10) = › 55
```

□□ Exercice 7 — Suite géométrique récursive

□ **Objectif :** Écrire une fonction récursive **suite(n)** pour la suite définie par : $u_0 = 1$, $u_□ = 2 \times u_{□□_1}$

```terminal
suite(0) = › 1
suite(3) = › 8
suite(5) = › 32
```

□□ Exercice 8 — Suite à deux termes précédents

□ **Objectif :** Écrire une fonction récursive **suite2(n)** pour la suite : $u_0 = 1$, $u_1 = 1$, $u_□ = 3 \times u_{□□_1} + u_{□□_2}$

terminal

```
suite2(0) = › 1
suite2(1) = › 1
suite2(4) = › 41
```

□ Bloc 3 — Fonctions sur les chaînes

La récursivité s'applique aussi aux chaînes — on traite le **premier caractère** et on rappelle la fonction sur le **reste** de la chaîne.

python

```python
# Pattern général sur une chaîne ch :
def f(ch) :
    if len(ch) == 0 :          # cas de base — chaîne vide
        return valeur
    # traiter ch[0]
    return ... + f(ch[1:len(ch)])  # appel sur le reste
```

□ **ch[1:len(ch)]** donne la chaîne sans son premier caractère — c'est l'équivalent récursif de 'avancer d'un cran'.

□□ Exercice 9 — Longueur d'une chaîne (sans len)

□ **Objectif :** Écrire une fonction récursive **longueur(ch)** qui retourne le nombre de caractères d'une chaîne **sans utiliser len()**.

terminal

```
longueur("Bonjour") = › 7
longueur("") = › 0
longueur("a") = › 1
```

□□ Exercice 10 — Compter les occurrences d'une lettre

□ **Objectif :** Écrire une fonction récursive **compter(ch, lettre)** qui retourne le nombre de fois que **lettre** apparaît dans **ch**.

terminal

```
compter("banana", "a") = › 3
compter("python", "z") = › 0
```

□□ Exercice 11 — Palindrome

□ **Objectif :** Écrire une fonction récursive **palindrome(ch)** qui retourne **True** si la chaîne est un palindrome, **False** sinon. Un mot est palindrome si sa première et dernière lettres sont identiques ET si le mot intérieur est aussi un palindrome.

□ Cas de base : chaîne vide ou d'un seul caractère → True.
Appel récursif : **ch[0] == ch[len(ch)-1]** ET **palindrome(ch[1:len(ch)-1])**

terminal

```
palindrome("radar") = › True
palindrome("python") = › False
palindrome("a") = › True
```

□□ Exercice 12 — Somme des chiffres d'un entier

□ **Objectif :** Écrire une fonction récursive **somme_chiffres(n)** qui calcule la somme des chiffres d'un entier positif n. Utiliser **n % 10** pour le dernier chiffre et **n // 10** pour le reste.

□ Exemple : somme_chiffres(438) = 4+3+8 = 15
Cas de base : n < 10 → retourner n
Appel récursif : (n % 10) + somme_chiffres(n // 10)

terminal

```
somme_chiffres(438) = › 15
somme_chiffres(1234) = › 10
somme_chiffres(7) = › 7
```

□ Récapitulatif — Récursivité

□□ Structure d'une fonction récursive

Définir	`def f(n):`
Cas de base	`if n == 0: return valeur`
Appel récursif	`return f(n - 1)`
Deux cas de base	`if n == 0: ... if n == 1: ...`
Sur une chaîne	`if len(ch) == 0: return valeur`
Reste de chaîne	`ch[1:len(ch)]`
Dernier chiffre	`n % 10`
Sans dernier ch.	`n // 10`