EXERCICES

Exercices sur les boucles non bornées

S'entraîner avec six exercices sur les boucles while : prévoir l'exécution, tracer, corriger et écrire des programmes.

Créer un fichier `boucles_non_bornees.py` dans Thonny. Utiliser des boucles `while` pour les fonctions demandées.

Le cours sur les boucles non bornées présente la méthode et les rappels utiles.

Exercice 1 — Prévoir l'exécution

Exercice 1

Sans utiliser l'ordinateur, donner pour chaque programme les valeurs finales des variables et la valeur affichée à la fin de l'exécution.

Programme 1

PYTHON

```
a = 6
b = 0
while a > b:
    a = a - 1
    b = b + 1
print(a + b)
```

Programme 2

PYTHON

```
n = 5
while n < 5:
    n = n + 1
print(n)
```

Programme 3

PYTHON

```
n = 1
while n < 15:
    n = n * 2
print(n)
```

Exercice 1

Vérifier ensuite les résultats dans Thonny. Pour le programme 2, expliquer ce qui changerait si la condition était `n <= 5`.

☐ Exercice 1 terminé.

Exercice 2 — Tracer et représenter

Exercice 2

Considérer le programme suivant :

```python
n = 4
total = 0
while n > 0:
    total = total + n
    n = n - 1
print(total)
```

1. Dessiner son schéma de branchement, avec l'initialisation, le test, le corps et la suite du programme.
2. Recopier et compléter le tableau ci-dessous. Ajouter toutes les lignes nécessaires afin d'obtenir la trace d'exécution.
3. Donner le nombre d'itérations et la valeur affichée.
4. Prévoir le résultat si `n` vaut `0` au départ.

Test	n avant le test	total avant le test	Résultat de n > 0	n après le corps	total après le corps
1	4	0	…	…	…

☐ Exercice 2 terminé.

Exercice 3 — Corriger des boucles

Exercice 3

Analyser ces programmes sur le cahier **avant de les exécuter**. Pour chacun, expliquer l'erreur puis proposer une correction conforme à l'objectif.

Programme 1 : afficher les entiers de 1 à 4 inclus.

PYTHON
```
n = 1
while n <= 4:
    print(n)
n = n + 1
```

Programme 2 : soustraire 2 jusqu'à obtenir une valeur inférieure ou égale à 0.

PYTHON
```
n = 5
while n > 0:
    n = n + 2
print(n)
```

Programme 3 : calculer la somme des entiers de 1 à n, pour n positif ou nul.

PYTHON
```
def somme_incorrecte(n: int) -> int:
    i = 1
    total = 0
    while i <= n:
        total = total + i
        i = i + 1
        return total
```

□ Exercice 3 terminé.

Exercice 4 — Calculer une somme

Exercice 4

a) Écrire une fonction `suite(n : int)->int` qui prend en paramètre un entier *n* et calcule **1+2+3+ ... +n** en utilisant la boucle `while`.

Exercice 4

```
PYTHON
>>> suite(5)
15
```

b) Tester sa fonction `suite()` en l'appelant dans la console puis l'écrire dans le cahier.

☐ Exercice 4 terminé.

Exercice 5 — Construire une puissance

Exercice 5

Écrire une fonction `puissance_deux(n: int) -> int` qui prend un entier positif ou nul et renvoie 2^n.

Rappel : 2^n est le produit de `n` facteurs égaux à `2`. Par convention, $2^0 = 1$.

La fonction doit construire le résultat avec des multiplications successives. Ne pas utiliser `**`.

```
PYTHON
>>> puissance_deux(0)
1
>>> puissance_deux(1)
2
>>> puissance_deux(5)
32
```

Préciser les rôles du compteur et de l'accumulateur. À quelle valeur faut-il initialiser un résultat qui se construit par multiplications ?

☐ Exercice 5 terminé.

Exercice 6 — Compter les étapes (difficile)

Exercice 6

Rappel : Un entier est pair si le reste de la division entière par **2** vaut **0**.

a) Écrire une fonction `nb_etape(n : int)->int` qui prend en paramètre un entier **n**.

Tant que cet entier n'est pas égal à **1**, nous le divisons par **2** s'il est pair ou nous le multiplions par **3** et nous ajoutons **1** s'il est impair.

Exercice 6

La fonction renvoie comme résultat le nombre d'étapes nécessaires pour arriver à 1.

PYTHON

```
>>> nb_etape(734)
46
```

b) Tester sa fonction `nb_etape()` en l'appelant dans la console puis l'écrire dans le cahier.

□ Exercice 6 terminé.