Suivre l'évolution des variables

Doc. 2,6

Objectif

Faire tourner des programmes à la main, en notant dans un tableau l'évolution du contenu des variables, pour prévoir ce que font un appel de fonction, une alternative et une boucle.

Chaque exercice se fait d'abord sur papier, sans ordinateur. Le programme est ensuite recopié dans Thonny et exécuté en mode débogage (touche F7) pour vérifier le tableau.

Méthode : remplir un tableau d'état

- Une ligne du tableau correspond à une instruction exécutée, dans l'ordre où Python l'exécute (ou à une itération, quand le tableau le précise).
- Sur chaque ligne, on écrit la valeur de chaque variable juste après l'exécution de l'instruction. Une case reste vide tant que la variable n'existe pas.
- Pour une condition (`if`, `elif`, `while`), on la recopie en remplaçant les variables par leurs valeurs, puis on écrit sa valeur : `True` ou `False`.
- Pour calculer une expression, on utilise les valeurs que les variables ont à ce moment-là, et non leurs valeurs du début.

§1. Exercice 1 — Suivre un appel de fonction

On considère le programme suivant.

```python
def transforme(x: int) -> int:
    """Fonction à étudier."""
    y = 2 * x
    x = y + 1
    return x * y

a = 3
b = transforme(a)
c = transforme(b - 40)
```

Q1. Sans utiliser Python, compléter le tableau d'état ci-dessous. Les variables `x` et `y` n'existent que pendant l'exécution de la fonction : leurs cases restent vides en dehors d'un appel.

Ligne	a	b	c	x	y
7					
8 : appel de `transforme`					
3					
4					

5 : valeur retournée					
8 : affectation de `b`					
9 : appel de `transforme`					
3					
4					
5 : valeur retournée					
9 : affectation de `c`					

Q2. À la ligne 4, la valeur de `x` change. Pourquoi a vaut-elle toujours `3` à la fin du programme ?

Q3. Sans utiliser Python, déterminer la valeur de `transforme(transforme(1))` en détaillant les étapes. Vérifier ensuite dans la console.

§2. Exercice 2 — Une fonction qui en appelle une autre

Dans le programme suivant, trois variables différentes portent le même nom `n` : celle du programme principal, le paramètre de `f` et le paramètre de `double`.

```python
def double(n: int) -> int:
    """Retourne le double de n."""
    n = 2 * n
    return n

def f(n: int) -> int:
    """Fonction à étudier."""
    m = double(n) + 1
    n = double(m)
    return n - m

n = 3
r = f(n)
s = f(r) + n
```

Q4. Sans utiliser Python, compléter le tableau d'état ci-dessous pour les lignes 12 et 13 du programme. La colonne « En cours » indique où se trouve l'exécution : programme principal, `f` ou `double`.

Ligne	**En cours**	**n** (programme)	**r**	**n** (dans `f`)	**m**	**n** (dans `double`)
12	programme	3				

Q5. Sans construire de nouveau tableau, déterminer la valeur de `s` après la ligne 14. Quelle valeur de `n` est utilisée dans l'expression `f(r) + n` ?

Q6. Un élève remplace les lignes 3 et 4 par la seule instruction `print(2 * n)`. Prévoir ce qui se passe lors de l'exécution de la ligne 13, puis vérifier.

§3. Exercice 3 — Quelle branche est exécutée ?

Un cinéma calcule le prix d'une place avec la fonction suivante.

```python
AGE_ENFANT = 12
AGE_SENIOR = 65
PRIX_ENFANT = 5
PRIX_REDUIT = 7
PRIX_NORMAL = 10

def tarif(age: int, etudiant: bool) -> int:
    """Retourne le prix de la place, en euros."""
    if age < AGE_ENFANT:
        prix = PRIX_ENFANT
    elif age >= AGE_SENIOR or etudiant:
        prix = PRIX_REDUIT
    else:
        prix = PRIX_NORMAL
    return prix
```

Q7. Sans utiliser Python, compléter le tableau ci-dessous. Dans la colonne du milieu, écrire chaque condition évaluée, dans l'ordre, avec les valeurs des variables et le résultat de l'évaluation. Une condition qui n'est pas évaluée ne doit pas apparaître.

Appel	Conditions évaluées	Valeur retournée
`tarif(8, True)`		
`tarif(70, False)`		
`tarif(17, True)`		
`tarif(30, False)`		
`tarif(12, False)`		

Q8. Une personne de 8 ans est aussi étudiante dans une école de musique : `tarif(8, True)` retourne `5` et non `7`. Expliquer pourquoi à partir de la structure `if` / `elif` / `else`.

Q9. Un élève remplace `elif` par `if` à la ligne 11. Le programme reste correct pour Python. Sans utiliser Python, déterminer ce que retournent alors `tarif(8, True)` et `tarif(8, False)`. Vérifier ensuite.

§4. Exercice 4 — Des if successifs ou un elif ?

On compare deux versions d'une fonction de calcul de score. En dehors de leur nom, elles ne diffèrent que par les mots-clés des lignes 5 et 7.

```python
def ajuste_a(score: int) -> int:
    """Calcule un nouveau score."""
    if score < 10:
        score = score + 3
    if score < 15:
        score = score + 2
    if score > 14:
        score = score + 1
    return score
```

```python
def ajuste_b(score: int) -> int:
    """Calcule un nouveau score."""
    if score < 10:
        score = score + 3
    elif score < 15:
        score = score + 2
    elif score > 14:
        score = score + 1
    return score
```

Q10. Sans utiliser Python, suivre l'appel `ajuste_a(9)` en complétant le tableau ci-dessous. Chaque ligne du tableau correspond à une ligne exécutée de la fonction. Toutes les lignes du tableau ne seront pas forcément utilisées.

Ligne	Condition testée	score
1		9

Q11. Faire de même pour `ajuste_b(9)`, puis compléter le tableau des valeurs retournées ci-dessous.

score de départ	`ajuste_a`	`ajuste_b`
9	14	
13		
16		

Q12. **Plus difficile.** Pour quelles valeurs entières de `score`, comprises entre 0 et 20, les deux versions retournent-elles des valeurs différentes ? Justifier, puis vérifier avec une boucle dans la console.

§5. Exercice 5 — Faire tourner une boucle à la main

On considère les deux fonctions suivantes.

```python
def mystere(n: int) -> int:
    """Fonction à étudier."""
    s = 0
    for i in range(1, n + 1, 2):
        s = s + i
    return s
```

```python
def mystere_bis(n: int) -> int:
    """Fonction à étudier."""
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 2
    return s
```

Q13. Quels entiers sont produits par `range(1, 8, 2)` ? Combien d'itérations la boucle de `mystere(7)` effectue-t-elle ?

Q14. Sans utiliser Python, compléter le tableau d'état de l'appel `mystere(7)`. Chaque ligne donne les valeurs à la fin d'une itération.

Étape	i	s
avant la boucle		0
itération 1		
itération 2		
itération 3		
itération 4		

Q15. Sans utiliser Python, compléter le tableau d'état de l'appel `mystere_bis(7)`. La condition de la boucle est évaluée au début de chaque itération : la dernière ligne correspond au test qui arrête la boucle.

Étape	Condition testée	s	i
avant la boucle		0	1
itération 1	`1 <= 7` : `True`		
itération 2			
itération 3			
itération 4			
fin			

Q16. À la fin de chaque appel, quelle est la dernière valeur prise par `i` : dans `mystere(7)` ? dans `mystere_bis(7)` ? Expliquer la différence.

Q17. Sans utiliser Python, prévoir les valeurs de `mystere(8)` et de `mystere_bis(0)`. Combien d'itérations sont effectuées dans le second appel ?

Q18. Quel calcul réalisent ces deux fonctions ?

§6. Exercice 6 — Fonctions, alternative et boucles réunies

§6.1. Une boucle non bornée qui appelle une fonction

On part d'un entier strictement positif `n`. S'il est pair, on le divise par 2 ; sinon, on le multiplie par 3 et on ajoute 1. On recommence jusqu'à obtenir 1. La fonction `syracuse` compte le nombre d'étapes et retient la plus grande valeur atteinte.

```python
1  def est_pair(n: int) -> bool:
```

```
2       """Retourne True si n est pair, False sinon."""
3       return n % 2 == 0
4
5   def syracuse(n: int) -> str:
6       """Retourne le nombre d'étapes pour atteindre 1
7       et la plus grande valeur atteinte.
8       """
9       etapes = 0
10      maxi = n
11      while n != 1:
12          if est_pair(n):
13              n = n // 2
14          else:
15              n = 3 * n + 1
16          if n > maxi:
17              maxi = n
18          etapes += 1
19      return f"{etapes} étapes, maximum {maxi}"
```

Q19. Sans utiliser Python, compléter le tableau d'état de l'appel `syracuse(6)`. Chaque ligne donne les valeurs à la fin d'une itération ; la colonne `est_pair(n)` contient la valeur retournée par cet appel, au début de l'itération. Toutes les lignes ne seront pas forcément utilisées. Quelle chaîne de caractères est retournée ?

Étape	`est_pair(n)`	n	maxi	etapes
avant la boucle		6		
itération 1				
itération 2				
itération 3				
itération 4				
itération 5				
itération 6				
itération 7				
itération 8				
itération 9				
itération 10				

Q20. Sans utiliser Python, prévoir la valeur retournée par `syracuse(1)`. Combien de fois le bloc de la boucle est-il exécuté ?

Q21. Un élève appelle `syracuse(0)`, alors que la fonction attend un entier strictement positif. Faire tourner à la main les deux premières itérations. Que se passe-t-il ? Quelle modification de la ligne 11 éviterait ce problème ?

§6.2. Deux boucles imbriquées

```
def motif(n: int) -> str:
```
Python

```
    """Fonction à étudier."""
    texte = ""
    for i in range(1, n + 1):
        for j in range(i):
            texte += "*"
        texte += "-"
    return texte
```

Q22. Sans utiliser Python, compléter le tableau d'état de l'appel `motif(3)`. Une ligne correspond à une instruction `texte += ...` exécutée ; la colonne `j` reste vide pour l'instruction `texte += "-"`. Quelle chaîne est retournée ?

i	j	texte

Q23. Sans utiliser Python, déterminer le nombre de caractères de la chaîne retournée par `motif(4)`. Vérifier avec `len(motif(4))`.

§7. Bilan

À l'issue de cette activité, je sais...

- remplir un tableau d'état en utilisant, pour chaque instruction, les valeurs des variables juste avant son exécution ;
- suivre un appel de fonction : le paramètre reçoit la valeur de l'argument, les variables de la fonction disparaissent après le `return` ;
- expliquer pourquoi modifier un paramètre ne modifie pas la variable passée en argument ;
- distinguer une fonction qui retourne une valeur d'une fonction qui l'affiche ;
- repérer le seul bloc exécuté dans une structure `if` / `elif` / `else`, et la différence avec des `if` successifs ;
- compter les itérations d'une boucle `for` avec `range` et d'une boucle `while`, y compris lorsqu'il n'y en a aucune ;
- repérer une boucle infinie en faisant tourner une boucle à la main ;
- suivre deux boucles imbriquées.