Corrigé – Python : variables, tests et fonctions

Mathématiques – Seconde

Algorithmique et programmation · exercices ★ à ★★★, pièges et QCM

Pour chaque erreur : relis la ligne du corrigé, trouve l'étape où tu t'es trompé, puis refais le calcul sans regarder.

Exercice 1 – Que contiennent les variables ? ★

Pour chaque programme, donne la valeur de chaque variable à la fin.

a) l. 1 : `a` reçoit `5`
l. 2 : `b = a + 3` : on calcule `5 + 3`, donc `b` vaut `8`
l. 3 : `a = 2 * b` : on calcule `2 * 8`, donc `a` vaut `16` ; l'ancienne valeur `5` est perdue
À la fin : `a` = `16`, `b` = `8`.

b) l. 1 : `x` reçoit `7`
l. 2 : `x = x - 10` : on calcule `7 - 10`, donc `x` vaut `-3` ; l'ancienne valeur `7` est perdue
l. 3 : `x = x * x` : on calcule `(-3) * (-3)`, donc `x` vaut `9` ; l'ancienne valeur `-3` est perdue
À la fin : `x` = `9`.

c) l. 1 : `a` reçoit `3`
l. 2 : `b` reçoit `8`
l. 3 : `a = b` : `a` prend la valeur de `b`, soit `8` ; l'ancienne valeur `3` est perdue
l. 4 : `b = a` : `b` prend la valeur de `a`, soit `8` (inchangé)
À la fin : `a` = `8`, `b` = `8`.
À la ligne 3, l'ancienne valeur de `a` est perdue : les deux variables valent 8, l'échange a échoué.

d) l. 1 : `a` reçoit `3`
l. 2 : `b` reçoit `8`
l. 3 : `c = a` : `c` prend la valeur de `a`, soit `3`
l. 4 : `a = b` : `a` prend la valeur de `b`, soit `8` ; l'ancienne valeur `3` est perdue
l. 5 : `b = c` : `b` prend la valeur de `c`, soit `3` ; l'ancienne valeur `8` est perdue
À la fin : `a` = `8`, `b` = `3`, `c` = `3`.
`c` a gardé l'ancienne valeur de `a` : l'échange réussit.

e) l. 1 : `n` reçoit `4`
l. 2 : `n = n + 1` : on calcule `4 + 1`, donc `n` vaut `5` ; l'ancienne valeur `4` est perdue
l. 3 : `m = n * 2` : on calcule `5 * 2`, donc `m` vaut `10`
l. 4 : `n = m - n` : on calcule `10 - 5`, donc `n` vaut `5` (inchangé)
À la fin : `n` = `5`, `m` = `10`.

f) l. 1 : `x` reçoit `2`
l. 2 : `y = x ** 3` : on calcule `2 ** 3`, donc `y` vaut `8` (puissance)
l. 3 : `x = y % 5` : on calcule `8 % 5`, donc `x` vaut `3` (reste de la division) ; l'ancienne valeur `2` est perdue
À la fin : `x` = `3`, `y` = `8`.

g) l. 1 : `a` reçoit `10`
l. 2 : `b = a / 4` : on calcule `10 / 4`, donc `b` vaut `2.5` (`/` donne un flottant)
l. 3 : `c = a // 4` : on calcule `10 // 4`, donc `c` vaut `2` (quotient entier)
À la fin : `a` = `10`, `b` = `2.5`, `c` = `2`.

h) l. 1 : `s` reçoit `'4'`
l. 2 : `t = s + s` : on calcule `'4' + '4'`, donc `t` vaut `'44'` (deux textes : `+` les colle)
l. 3 : `u = 2 * s` : on calcule `2 * '4'`, donc `u` vaut `'44'` (un texte multiplié par 2 est répété deux fois)
À la fin : `s` = `'4'`, `t` = `'44'`, `u` = `'44'`.

Exercice 2 – Opérations et types ★

Donne la valeur que Python calcule pour chaque expression.

a) `3` (quotient de 17 par 5)

b) `2` ($17 = 5 \times 3 + 2$: le reste)

c) `32` ($2^5 = 2 \times 2 \times 2 \times 2 \times 2$)

d) `3.5` (division décimale)

e) `4.0` (`/` donne toujours un flottant)

f) `0` (20 est un multiple de 4, le reste est nul)

g) `12` (la puissance d'abord : 3×4)

h) `14` (les parenthèses d'abord : 7×2)

i) `4` ($30 = 7 \times 4 + 2$: le quotient)

j) `True` (`==` compare les valeurs)

k) `True` (`7 % 2` vaut `1`, et `1 == 1` est vrai)

l) `'35'` (deux textes : `+` les colle)

Exercice 3 – Instructions conditionnelles ★

Qu'affiche chaque programme pour la valeur de `n` indiquée (placée en première ligne) ?

a) `15 >= 10` est vrai : `r = 15 - 10`.
Le programme affiche `5`.

b) `10 >= 10` est vrai : `r = 10 - 10`.
Le programme affiche `0`.

c) `4 >= 10` est faux, `4 >= 0` est vrai : `r = 2 * 4`.
Le programme affiche `8`.

d) `0 >= 10` est faux, `0 >= 0` est vrai : `r = 2 * 0`.
Le programme affiche `0`.

e) `-3 >= 10` et `-3 >= 0` sont faux : on passe au `else`, `r = -(-3)`.
Le programme affiche `3`.

f) `9.5 >= 10` est faux, `9.5 >= 0` est vrai : `r = 2 * 9.5`.
Le programme affiche `19.0`.

g) `12 % 2` et `12 % 3` valent `0` : la première condition est vraie.
Le programme affiche `A`.

h) `9 % 2` vaut `1` et `9 % 3` vaut `0` : la première condition est fausse ; la deuxième est vraie.
Le programme affiche `B`.

i) `8 % 2` vaut `0` et `8 % 3` vaut `2` : la première condition est fausse ; la deuxième aussi, on passe au `else`.
Le programme affiche `C`.

j) `0 % 2` et `0 % 3` valent `0` : la première condition est vraie.
Le programme affiche `A`.

Exercice 4 – Appeler une fonction ★★

Pour chaque fonction, donne ce que renvoient les appels.

a) On remplace le paramètre par la valeur de l'appel, puis on suit la fonction.
`f(4)` : $3 \times 4 - 5$, donc `f(4)` renvoie `7`.
`f(-1.5)` : $3 \times (-1{,}5) - 5 = -4{,}5 - 5$, donc `f(-1.5)` renvoie `-9.5`.

b) On remplace le paramètre par la valeur de l'appel, puis on suit la fonction.
`g(-3)` : $(-3)^2 - 2 \times (-3) = 9 + 6$, donc `g(-3)` renvoie `15`.
`g(0.5)` : $0{,}5^2 - 2 \times 0{,}5 = 0{,}25 - 1$, donc `g(0.5)` renvoie `-0.75`.

c) On remplace le paramètre par la valeur de l'appel, puis on suit la fonction.
`tarif(11)` : $11 < 12$ est vrai, donc `tarif(11)` renvoie `5`.
`tarif(12)` : $12 < 12$ est faux : on passe au `else`, donc `tarif(12)` renvoie `9`.

d) On remplace le paramètre par la valeur de l'appel, puis on suit la fonction.
`moyenne(7, 12)` : $19 \div 2$, un flottant, donc `moyenne(7, 12)` renvoie `9.5`.
`moyenne(moyenne(4, 8), 10)` : `moyenne(4, 8)` vaut `6.0`, puis $(6 + 10) \div 2$, donc `moyenne(moyenne(4, 8), 10)` renvoie `8.0`.

e) L'appel `double(5)` exécute `print(2 * 5)` : le programme affiche `10`.
La fonction n'a pas de `return` : l'appel vaut `None`, et c'est ce qui est rangé dans `y`.
La fonction affiche, elle ne renvoie rien.

Exercice 5 – Écrire une fonction ★★

Écris en Python la fonction décrite, avec `def` et `return`, puis donne le résultat du test.

a) $2x^2$ s'écrit `2 * x ** 2` : le signe $*$ ne s'oublie jamais.
Vérification : $2 \times 2^2 - 3 \times 2 + 1 = 8 - 6 + 1 = 3$.

```
def f(x):
    return 2 * x ** 2 - 3 * x + 1
```

Test : `f(2)` renvoie `3`.

b) Pour une évolution de t %, le coefficient multiplicateur est $1 + \dfrac{t}{100}$.
Baisse de 20 % : $1 + \dfrac{-20}{100} = 0{,}8$.

```
def coef(t):
    return 1 + t / 100
```

Test : `coef(-20)` renvoie `0.8`.

c) Taux d'évolution en % : $\dfrac{v_2 - v_1}{v_1} \times 100$; les parenthèses autour de `v2 - v1` sont obligatoires.
De 50 à 60 : $\dfrac{10}{50} \times 100 = 20$.

```
def taux(v1, v2):
    return (v2 - v1) / v1 * 100
```

Test : `taux(50, 60)` renvoie `20.0`.

d) $AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$; `sqrt` vient du module `math`.
Test : $\sqrt{3^2 + 4^2} = \sqrt{25} = 5$.

```
from math import sqrt
def distance(xA, yA, xB, yB):
    return sqrt((xB - xA) ** 2 + (yB - yA) ** 2)
```

Test : `distance(1, 2, 4, 6)` renvoie `5.0`.

e) Milieu : $\left(\dfrac{x_A + x_B}{2} ; \dfrac{y_A + y_B}{2}\right)$; la fonction renvoie les deux coordonnées, séparées par une virgule.
Test : $\dfrac{-1 + 4}{2} = 1{,}5$ et $\dfrac{3 + 5}{2} = 4$.

```
def milieu(xA, yA, xB, yB):
    return (xA + xB) / 2, (yA + yB) / 2
```

Test : `milieu(-1, 3, 4, 5)` renvoie `(1.5, 4.0)`.

f) n est pair quand le reste de sa division par 2 est nul : `n % 2 == 0` vaut déjà `True` ou `False`.
Test : $7 = 2 \times 3 + 1$, le reste vaut 1.

```
def est_pair(n):
    return n % 2 == 0
```

Test : `est_pair(7)` renvoie `False`.

Exercice 6 – Programmes à trous ★★

Complète chaque `...` pour que la fonction fasse ce qui est annoncé : `va` = valeur absolue, `maxi` = le plus grand des deux et `sur_droite` = le point est-il sur la droite d'équation $y = 2x + 1$.

a)

```
def va(x):
    if x < 0:
        return -x
    else:
        return x
```

Valeur absolue : si $x < 0$, on renvoie l'opposé `-x` (qui est positif) ; sinon on renvoie `x`.
Test : `va(-3)` renvoie `3` ; `va(2)` renvoie `2`.

b)

```
def maxi(a, b):
    if a > b:
        return a
    else:
        return b
```

Si $a > b$, le plus grand est `a` ; sinon, c'est `b`.
Test : `maxi(4, 9)` renvoie `9` ; `maxi(-2, -7)` renvoie `-2`.

c)

```
def sur_droite(x, y):
    if y == 2 * x + 1:
        return True
    return False
```

Le point $(x\,;y)$ est sur la droite quand $y = 2x + 1$: on compare `y` et `2 * x + 1`.
`==` *pour comparer, pas* `=`. Test : `sur_droite(3, 7)` renvoie `True` ; `sur_droite(-1, 1)` renvoie `False`.

Exercice 7 – La livraison ★★★

1. $30 < 50$: on paie la livraison, $30 + 4{,}9 = 34{,}9$ €.
   $50 \leqslant 60 < 100$: livraison gratuite, on paie 60 €.
   $120 \geqslant 100$: remise de 10 %, c'est multiplier par $1 - 0{,}1 = 0{,}9$: $120 \times 0{,}9 = 108$ €.

2. Trois cas, donc `if`, `elif`, `else`.

   ```
   def total(m):
       if m >= 100:
           return 0.9 * m
       elif m >= 50:
           return m
       else:
           return m + 4.9
   ```

   Le cas le plus restrictif ($m \geqslant 100$) est testé en premier ; le `elif` n'est lu que si $m < 100$, et le `else` que si $m < 50$.
   Test : `total(120)` renvoie `108.0`.

3. Elle renvoie `120`.
   $120 \geqslant 50$ est vrai, donc le premier `return` s'exécute : la fonction s'arrête.
   Le cas $m \geqslant 100$ n'est jamais atteint.
   Avec `elif`, seule la première condition vraie compte.

Exercice 8 – Le disque ★★★

1. $\Omega A = \sqrt{(x_A - x_\Omega)^2 + (y_A - y_\Omega)^2}$
   $\Omega A = \sqrt{(5-2)^2 + (3-(-1))^2} = \sqrt{9 + 16} = \sqrt{25} = 5$.
   $\Omega A \leqslant 5$: A est dans le disque (sur son bord).

2. On calcule la distance ΩM dans `d`, puis on la compare au rayon.

```
from math import sqrt
def dans_disque(x, y):
    d = sqrt((x - 2) ** 2 + (y + 1) ** 2)
    return d <= 5
```

`d <= 5` vaut `True` ou `False` : on peut le renvoyer directement.

Attention au signe : $y - (-1) = y + 1$.

3. Pour $M(7\,;2)$: $\Omega M = \sqrt{(7-2)^2 + (2+1)^2} = \sqrt{25+9} = \sqrt{34}$.
$\sqrt{34} > \sqrt{25} = 5$, donc `dans_disque(7, 2)` renvoie `False`.
Pour $O(0\,;0)$: $\Omega O = \sqrt{(0-2)^2 + (0+1)^2} = \sqrt{4+1} = \sqrt{5}$.
$\sqrt{5} \leqslant 5$, donc `dans_disque(0, 0)` renvoie `True`.

Exercice 9 – Trouve l'erreur ★★

Chaque programme fait autre chose que ce qui est annoncé en dessous. Trouve l'erreur, puis corrige.

a) Le programme s'arrête sur une erreur (`SyntaxError`).

Ligne fautive (l. 2) : `if n % 2 = 0:`

`=` *range une valeur ; pour comparer, il faut* `==` *(sinon erreur de syntaxe).*

Ligne corrigée : `if n % 2 == 0:`

```
n = 14
if n % 2 == 0:
    print("pair")
```

La version corrigée affiche `pair`.

b) Le programme s'arrête sur une erreur (`TypeError`).

Ligne fautive (l. 2) : `print(3 * r ** 2)`

La fonction affiche mais ne renvoie rien : `aire(10)` *vaut* `None`*, et* `None + 1` *plante.*

Ligne corrigée : `return 3 * r ** 2`

```
def aire(r):
    return 3 * r ** 2
A = aire(10) + 1
```

La version corrigée donne `A` $= 301$.

c) Le programme donne `p` $= 6$.

Lignes en cause : l. 2 à 5.

$8 < 18$ *est vrai : le premier cas gagne.*

Il faut tester le cas le plus restrictif (< 12*) d'abord.*

On échange les deux cas :

```
def prix(age):
    if age < 12:
        return 4
    elif age < 18:
        return 6
    else:
        return 10
p = prix(8)
```

La version corrigée donne `p` $= 4$.

d) Le programme donne `boites` $= 7.5$.

Ligne fautive (l. 2) : `boites = oeufs / 6`

`/` donne $7{,}5$ *; le nombre de boîtes pleines est le quotient entier `//`.*

Ligne corrigée : `boites = oeufs // 6`

```
oeufs = 45
boites = oeufs // 6
```

La version corrigée donne `boites` $= 7$.

e) Le programme donne `A` $= 7$.

Ligne fautive (l. 2) : `A = c ^ 2`

En Python, la puissance s'écrit `**` *(`^` donne un autre résultat :* 7*).*

Ligne corrigée : `A = c ** 2`

```
c = 5
A = c ** 2
```

La version corrigée donne `A` $= 25$.

f) Le programme donne `d` $= 58.59$ environ.

Ligne fautive (l. 2) : `d = (xB - xA) ** 2 + (yB - yA) ** 2 ** 0.5`

`** 0.5` *ne s'applique qu'à ce qui le précède (ici 2) : la racine ne porte pas sur la somme.*

Il faut des parenthèses autour de toute la somme.

Ligne corrigée : `d = ((xB - xA) ** 2 + (yB - yA) ** 2) ** 0.5`

```
xA, yA, xB, yB = 0, 1, 5, 13
d = ((xB - xA) ** 2 + (yB - yA) ** 2) ** 0.5
```

La version corrigée donne `d` $= 13.0$.

QCM – Six questions comme en contrôle

Une seule bonne réponse par question.

1. En Python, `19 % 4` vaut :

Réponse C : `3`

$19 = 4 \times 4 + 3$: `%` donne le reste.

2. Pour tester si `n` vaut 10, on écrit :

Réponse A : `if n == 10:`

`==` compare, et la ligne du `if` se termine par deux-points.

3. Après `a = 4`, `b = 7`, `a = a + b`, `b = a - b`, la variable `b` vaut :

Réponse D : `4`

`a = 4 + 7` donne `11`.

Puis `b = a - b` se calcule avec les valeurs du moment : $11 - 7 = 4$.

4. `2 + 3 ** 2` vaut :

Réponse B : `11`

La puissance d'abord : `3 ** 2` vaut 9, puis $2 + 9 = 11$.

5. Une fonction sans `return` qui se termine par `print(x)` renvoie :

Réponse D : `None`

Elle affiche `x` mais ne renvoie rien.

6. Avec `def f(x):` qui renvoie `2 * x` si `x > 0`, et `x - 1` sinon, `f(0)` vaut :

Réponse A : `-1`

$0 > 0$ est faux : on ne passe pas par le premier `return`.

On calcule $0 - 1 = -1$.

Ce document est la propriété intellectuelle de Ben, professeur particulier (benprofparticulier.fr), protégée par le Code de la propriété intellectuelle (art. L. 111-1 et L. 122-4). Il est mis à disposition pour un usage personnel de révision. Toute reproduction, diffusion, mise en ligne ou revente, totale ou partielle, sans autorisation écrite est interdite et constitue une contrefaçon (art. L. 335-2).