MATHS SECONDE FICHE DE COURS

Algorithmique et Python

cours et exercices corrigés de maths 2nde

Ce chapitre n'est pas un chapitre à côté des autres : c'est un OUTIL qui les sert tous. On programme pour simuler mille lancers de dé, pour tester une conjecture sur tous les nombres jusqu'à cent, pour calculer ce qu'on ne veut pas calculer à la main. Et pour cela, il suffit de savoir lire un programme ligne à ligne.

MOTS CLÉS	LE SECRET	OUTIL
Variable, affectation, condition, boucle, fonction	**Suivre le programme ligne à ligne, en notant les valeurs**	**Un ordinateur qui répète sans se tromper ni se fatiguer**

À quoi ça sert : Algorithmique et Python

C'est le chapitre qui sert tous les autres. En probabilités, on simule dix mille lancers pour voir la fréquence s'approcher de la théorie. En statistiques, on calcule une moyenne sur une série qu'aucun élève ne voudrait additionner à la main. En arithmétique, on teste une conjecture sur tous les entiers jusqu'à mille. ⭐ Et hors du lycée : un moteur de recherche, un GPS, une application de messagerie ne sont que des algorithmes — des suites d'instructions, écrites une fois, exécutées des milliards de fois.

Un peu d'histoire : Algorithmique et Python

Le mot « algorithme » vient du nom d'al-Khwârizmî, le mathématicien de Bagdad qui donna aussi son titre à l'algèbre au IX^{e} siècle : une suite de gestes à faire dans l'ordre, sans avoir à comprendre pourquoi ils marchent. Python, lui, est né en 1991 sous les doigts du Néerlandais Guido van Rossum, qui cherchait un langage lisible — et qui l'a baptisé non pas d'après le serpent, mais d'après la troupe comique des Monty Python.

Définition : Algorithmique et Python

Un ALGORITHME est une suite d'instructions qui mène d'une donnée à un résultat. Python est un langage qui permet de l'écrire pour qu'une machine l'exécute. Une VARIABLE est un nom qui retient une valeur ; l'AFFECTATION `x = 5` range la valeur 5 dans la boîte nommée x. ⛔ Ce signe `=` n'est pas l'égalité des mathématiques : il ne compare pas, il RANGE.

```
x = 5
y = x + 3
print(y)
```

Trois instructions, exécutées dans l'ordre. La ligne rouge lit x, calcule, et range le résultat dans y. Le programme affiche 8.

Un programme se lit de haut en bas, une ligne à la fois. ⭐ À chaque ligne, on peut noter ce que valent les variables : c'est tout le métier de ce chapitre.

Propriétés : Algorithmique et Python

⛔ La variable : `=` range, il ne compare pas

`x = x + 1` est une phrase FAUSSE en mathématiques et une instruction JUSTE en Python. On calcule d'abord la droite avec l'ancienne valeur, puis on range le résultat à gauche. Pour comparer, Python utilise `==`, avec deux signes.

x → 0 → 1 → 2 → 3

Après x = 0, l'instruction x = x + 1 répétée trois fois. À chaque tour, l'ancienne valeur sert à calculer la nouvelle.

Les types, et le piège de la division

Une valeur a un TYPE : entier (`int`), flottant (`float`), chaîne (`str`), booléen (`bool`). ⛔ En Python 3, l'opérateur `/` rend TOUJOURS un flottant, même quand la division tombe juste : `6 / 3` vaut `2.0`, pas `2`. Pour un entier, il faut la division entière `//`.

```
6 / 3   → 2.0    (float)
6 // 3  → 2      (int)
6 % 4   → 2      (int)
6 > 3   → True   (bool)
```

Quatre opérateurs, quatre types de résultat. La ligne rouge est celle qui surprend.

Les conditions : un seul bloc s'exécute

`if` teste, `elif` teste seulement si le précédent était faux, `else` ramasse le reste. On descend les tests dans l'ordre et on s'arrête au PREMIER qui est vrai. ⭐ `n % 2 == 0` est le test de parité : le reste de la division par 2 vaut 0.

```
if note >= 16:
    print("tres bien")
elif note >= 10:
    print("admis")
else:
    print("a revoir")
```

Avec note = 12 : le premier test échoue, la ligne rouge est vraie, et le programme s'arrête là.

Les boucles : on sait combien, ou on sait quand

`for` répète un nombre de fois CONNU d'avance — `range(5)` donne cinq tours, de 0 à 4. `while` répète TANT QUE sa condition reste vraie : on ne sait pas combien de tours, on sait seulement quand ça s'arrête.

for — bornée

```
s = 0
for i in range(4):
    s = s + i
```

while — non bornée

```
x = 1
while x < 50:
    x = x * 2
```

À gauche, quatre tours, décidés d'avance. À droite, on compte en traçant : 1, 2, 4, 8, 16, 32, 64 — six tours.

Les fonctions : `def` et `return`

Une fonction porte un nom, reçoit des arguments et RENVOIE un résultat avec `return`. On l'écrit une fois, on l'appelle autant qu'on veut. ⚠️ `return` renvoie une valeur au programme ; `print` l'affiche seulement à l'écran. Ce n'est pas la même chose.

```
def f(x):
    return 2 * x + 1
f(3)   → 7
f(10)  → 21
```

La ligne rouge est le calcul. Écrit une fois, il sert pour toutes les valeurs.

La simulation : compter ce qu'on ne peut pas prévoir

`randint(1, 6)` tire un entier au hasard entre 1 et 6, LES DEUX BORNES COMPRISES — donc six valeurs possibles. En répétant beaucoup et en comptant les succès, on ESTIME une probabilité : c'est la fréquence observée.

```
c = 0
for i in range(1000):
    if randint(1, 6) == 6:
        c = c + 1
print(c / 1000)
```

Le compteur ne monte que sous condition. À la fin, c / 1000 estime la probabilité d'obtenir un 6.

Méthode : Algorithmique et Python

1. Je trace, ligne à ligne

Je note la valeur de chaque variable APRÈS chaque instruction. C'est lent, c'est sûr, et c'est ce qu'on attend au contrôle : personne ne demande de deviner.

s → 0 → 0 → 1 → 3 → 6

s = 0, puis s = s + i pour i valant 0, 1, 2, 3. La dernière valeur est la réponse.

2. Je repère la structure avant le détail

Une indentation ouvre un bloc. Ce qui est décalé appartient au `if` ou à la boucle au-dessus ; ce qui revient à gauche s'exécute après. ⛔ En Python, l'alignement n'est pas de la mise en forme : c'est de la syntaxe.

```
for i in range(3):
    print(i)      # dans la boucle
print("fini")     # apres la boucle
```

La ligne rouge est décalée, donc répétée trois fois. La dernière ne l'est pas : elle s'exécute une seule fois.

3. Je vérifie sur un petit cas

Avant de faire confiance à un programme, je le fais tourner mentalement avec une valeur simple dont je connais la réponse. Si le programme se trompe là, inutile de l'essayer plus loin.

```
def carre(x):
    return x * x
carre(3)  → 9   ✓
```

Un cas dont on connaît la réponse suffit à démasquer la plupart des erreurs.

Selon ce que l'on cherche : Algorithmique et Python

Tracer un programme

« Que vaut x à la fin ? » — c'est LA question du contrôle. On suit les instructions dans l'ordre en notant les valeurs, sans chercher de raccourci.

x → 1 → 2 → 4 → 8 → 16

x = 1 puis x = x * 2, tant que x < 10.

Compter sous condition

Un compteur qui ne monte que si un test est vrai : c'est le motif de toutes les simulations, et de tout dénombrement.

```
c = 0
for t in [3, 6, 1, 6, 4]:
    if t == 6:
        c = c + 1
```

La ligne rouge décide. Ici c finit à 2.

Estimer une probabilité

On répète l'expérience un grand nombre de fois, on compte les succès, on divise. Plus on répète, plus l'estimation est fiable.

c / n → 0,20 → 0,17 → 0,167

La même simulation avec 10, 100 puis 10 000 lancers : l'estimation se rapproche de 1/6 ≈ 0,1667.

Exemples corrigés : Algorithmique et Python

Une affectation qui se mord la queue

`a = 2` puis `b = a` puis `a = 7`.

Que vaut `b` ?

`b` vaut 2. À la ligne `b = a`, Python lit la VALEUR d'a — c'est-à-dire 2 — et la range dans b. Changer a ensuite ne touche plus à b. ⛔ L'erreur est de croire que b « suit » a : une affectation copie une valeur, elle ne crée pas de lien.

⛔ Le type d'une division

`8 / 4` et `8 // 4`.

Ces deux expressions ont-elles le même type ?

```
8 / 4   → 2.0   (float)
8 // 4  → 2     (int)
```

Non. Les deux valent deux, mais `8 / 4` rend `2.0`, un FLOTTANT, tandis que `8 // 4` rend `2`, un ENTIER. ⭐ On ne regarde pas si le calcul tombe juste, on regarde l'OPÉRATEUR : en Python 3, `/` rend toujours un flottant.

Un test de parité

`n = 37`.

Qu'affiche `if n % 2 == 0: print("pair") else: print("impair")` ?

```
37 % 2  → 1
1 == 0  → False
→ else
```

Le programme affiche « impair ». `37 % 2` donne le reste de la division de 37 par 2, soit 1. La condition `1 == 0` est fausse, donc c'est le bloc `else` qui s'exécute. ⭐ Tester la parité, c'est tester si ce reste vaut zéro.

Combien de tours ?

`x = 1` et `c = 0`, puis `while x < 20: x = x * 2; c = c + 1`.

Que vaut `c` à la fin ?

`c` vaut 5. On trace : x vaut 1, 2, 4, 8, 16, puis 32 — et à ce moment la condition `x < 20` devient fausse, la boucle s'arrête. Il y a eu cinq passages, donc cinq incréments. ⛔ Ce nombre ne se devine pas : une boucle `while` se COMPTE.

Pièges à éviter : Algorithmique et Python

⛔ `=` n'est pas l'égalité. `x = x + 1` range dans x l'ancienne valeur augmentée de un. Pour comparer, on écrit `==`, avec deux signes.

⛔ `/` rend toujours un flottant en Python 3 : `6 / 3` vaut `2.0`. Pour un entier, c'est `//`.

⛔ `range(5)` donne $0, 1, 2, 3, 4$ — cinq valeurs, et la dernière n'est pas 5. De même `range(1, 4)` s'arrête à 3.

⛔ L'indentation est de la SYNTAXE, pas de la décoration. Une ligne décalée appartient au bloc au-dessus ; la même ligne non décalée s'exécute une seule fois.

⛔ Dans un `if / elif / else`, un SEUL bloc s'exécute. Dès qu'un test est vrai, les suivants ne sont même pas lus.

⛔ `return` renvoie une valeur au programme, `print` l'affiche à l'écran. Une fonction sans `return` ne renvoie rien, même si elle affiche quelque chose.

À retenir : Algorithmique et Python

`x = 5` range 5 dans x ; `x == 5` demande si x vaut 5.

Types : `int`, `float`, `str`, `bool`. Et `/` rend toujours un `float`.

`n % 2 == 0` teste si n est pair.

`for` : on sait combien de tours. `while` : on sait quand s'arrêter.

`range(n)` va de 0 à $n - 1$: n valeurs, pas n+1.

Pour répondre à « que vaut x à la fin ? », on TRACE ligne à ligne.

Exercices corrigés : Algorithmique et Python

1. Après `x = 4` puis `x = x + 6`, que vaut x ?

Correction :

10. Python calcule d'abord la droite avec l'ancienne valeur — $4 + 6 = 10$ — puis range le résultat dans x.

2. Après `a = 3`, `b = a`, `a = 9`, que vaut b ?

Correction :

3. La ligne `b = a` a copié la VALEUR d'a au moment où elle s'exécutait. Changer a ensuite ne modifie pas b.

3. Quel est le type de `10 / 5` en Python ?

Correction :

Un flottant (`float`) : `10 / 5` vaut `2.0`. L'opérateur `/` rend toujours un flottant, même quand la division tombe juste.

4. Quel est le type de `17 % 5` ?

Correction :

Un entier (`int`). `%` donne le reste d'une division euclidienne : ici 2. C'est `/` qui rend un flottant, pas `%` ni `//`.

5. Avec `n = 24`, qu'affiche `if n % 2 == 0: print("pair") else: print("impair")` ?

Correction :

« pair ». $24 \% 2 = 0$, donc la condition est vraie.

6. Avec `note = 8`, qu'affiche `if note >= 16: "tres bien" elif note >= 10: "admis" else: "a revoir"` ?

Correction :

« a revoir ». Les deux premiers tests échouent — $8 < 16$ et $8 < 10$ — donc c'est le bloc `else` qui s'exécute.

7. Que vaut s après `s = 0` puis `for i in range(4): s = s + i` ?

Correction :

6. i prend les valeurs $0, 1, 2, 3$ — et non 4 — donc $s = 0 + 1 + 2 + 3 = 6$.

8. Que vaut c après `x = 1`, `c = 0`, `while x < 100: x = x * 2; c = c + 1` ?

Correction :

7. On trace : $1, 2, 4, 8, 16, 32, 64, 128$. Il faut sept doublements pour dépasser 100.

9. Avec `def f(x): return 3 * x - 2`, que vaut `f(5)` ?

Correction :

13. On remplace x par 5 dans l'expression : $3 \times 5 - 2 = 13$.

10. Une simulation de 2000 lancers compte 340 succès. Quelle estimation de la probabilité ?

Correction :

$\frac{340}{2000} = 0{,}17$. Une estimation est une FRÉQUENCE observée : le nombre de succès divisé par le nombre d'essais.