Les tuples

Doc. 3,1

Objectif

Regrouper plusieurs valeurs dans un seul objet, le tuple, pour qu'une fonction puisse renvoyer plusieurs résultats à la fois.

§1. Une fonction peut-elle renvoyer deux valeurs ?

Dans le Doc. 2,4, la fonction `plus_grand_plus_petit` retournait une chaîne de caractères comme `"max : 18.0, min : 7.5"`. Pour un humain, c'est très lisible. Mais imaginons qu'on veuille ensuite calculer l'écart entre le maximum et le minimum. Il faudrait fouiller dans le texte pour retrouver les chiffres, puis les convertir en nombres. On a fabriqué une phrase alors qu'on voulait deux nombres.

Le vrai problème est le suivant : une instruction `return` met fin à la fonction. On ne peut donc pas écrire deux `return` à la suite en espérant récupérer deux résultats. Faisons une expérience.

Q1. Recopier la fonction suivante dans la partie principale de l'éditeur Thonny :

```python
def min_max(a: float, b: float):
    """Retourne le plus petit puis le plus grand des nombres a et b."""
    if a < b:
        return a, b
    return b, a
```

1. Appeler `min_max(9, 4)` dans la console. Qu'affiche-t-elle ?
2. Saisir ensuite `resultat = min_max(9, 4)`, puis `type(resultat)`. Combien d'objets la fonction a-t-elle retournés ?

On n'a donc pas contourné la règle : une fonction renvoie toujours une seule valeur. On a simplement rangé les deux nombres dans une même boîte, et c'est la boîte qu'on renvoie.

À retenir

Un **tuple** (le programme dit aussi **p-uplet**) est une suite ordonnée de valeurs. On l'écrit en séparant les valeurs par des virgules, le plus souvent entre parenthèses : `(4, 9)`.

Le tuple est un **type construit** : il est fabriqué à partir de valeurs de types de base (`int`, `float`, `str`, `bool`).

Une fonction ne retourne qu'une seule valeur. Pour retourner plusieurs résultats, elle les regroupe dans un tuple : `return a, b`.

§2. Créer un tuple et lire ses éléments

§2.1. C'est la virgule qui fait le tuple

Q2. Pour chacune des expressions suivantes, prévoir le type de la valeur obtenue, puis vérifier avec la fonction `type` dans la console :

```python
>>> type((3, 7, 12))
```

```
>>> type(3, 7, 12)
>>> type((5))
>>> type((5,))
>>> type(())
```

Le deuxième essai provoque une erreur. Pourquoi, à votre avis ? Comment écrire un tuple qui ne contient qu'un seul élément ?

Les éléments d'un tuple ne sont pas forcément du même type. Une ville peut être décrite par son nom, sa latitude et sa longitude :

```python
ville = ("Paris", 48.86, 2.35)
```

Ce tuple n'a de sens que parce qu'on s'est mis d'accord sur l'ordre : d'abord le nom, ensuite la latitude, enfin la longitude. Si on écrivait `(48.86, "Paris", 2.35)`, Python ne protesterait pas, mais un programme qui attend le nom en premier serait trompé.

§2.2. Repérer un élément par son indice

Chaque élément d'un tuple occupe une position, appelée **indice**. En Python, les indices commencent à 0. Cela paraît étrange au début. Une façon de s'en souvenir : l'indice indique **combien de pas il faut faire depuis le début** pour atteindre l'élément. Le premier élément est à zéro pas du début.

indice	0	1	2
élément de `t`	3	7	12

On lit l'élément d'indice `i` du tuple `t` en écrivant `t[i]`. La fonction `len`, déjà rencontrée avec les chaînes de caractères, donne le nombre d'éléments du tuple.

Q3. Saisir `t = (3, 7, 12)` dans la console. Prévoir le résultat de chacune des instructions suivantes, puis vérifier :

```python
>>> t[0]
>>> t[2]
>>> len(t)
>>> t[3]
>>> t[len(t) - 1]
```

Quel est l'indice du dernier élément d'un tuple qui contient `n` éléments ?

Indices négatifs

Python accepte aussi des indices négatifs, qui comptent à partir de la fin : `t[-1]` est le dernier élément, `t[-2]` l'avant-dernier. Avec `t = (3, 7, 12)`, `t[-1]` vaut `12`, exactement comme `t[len(t) - 1]`.

Q4. Avec `ville = ("Paris", 48.86, 2.35)`, écrire une instruction qui affiche, à l'aide d'une f-string et des indices :

```
Paris : latitude 48.86, longitude 2.35
```

À retenir

Les éléments d'un tuple `t` sont repérés par leur **indice**, qui commence à 0. On lit l'élément d'indice `i` avec `t[i]`.

`len(t)` donne le nombre d'éléments de `t`. Les indices valides vont donc de `0` à `len(t) - 1`. Un indice en dehors de ces limites provoque une erreur `IndexError`.

Un tuple d'un seul élément s'écrit avec une virgule : `(5,)`. Le tuple vide s'écrit `()`.

§3. Un tuple ne peut pas être modifié

Q5. Toujours avec `t = (3, 7, 12)`, essayer de remplacer le premier élément par 4 :

```
>>> t[0] = 4
```
Python

Que se passe-t-il ?

Un tuple, une fois créé, ne change plus. On dit qu'il est **non mutable** (ou **immuable**). Est-ce gênant ? Pas vraiment, car on peut toujours fabriquer un **nouveau** tuple.

Q6. Prévoir, puis vérifier, le contenu de `t` et de `u` après chacune des instructions suivantes :

```
>>> t = (3, 7, 12)
>>> u = t + (20,)
>>> u
>>> t
>>> t = (4, t[1], t[2])
>>> t
```
Python

L'opérateur `+` a-t-il modifié `t` ? La dernière affectation a-t-elle modifié le tuple `(3, 7, 12)` ?

Q7. Prévoir puis vérifier le résultat de `7 in t` et de `8 in t`, avec `t = (4, 7, 12)`. De quel type est la valeur obtenue ?

Pourquoi Python propose-t-il un objet qu'on ne peut pas modifier ? Parce que c'est une garantie. Les coordonnées de Paris ne vont pas changer pendant l'exécution du programme. Si on les range dans un tuple et qu'on les confie à une fonction, on est certain de les retrouver intactes, quelle que soit la façon dont cette fonction a été écrite.

À retenir

Un tuple est **non mutable** : on ne peut pas modifier ses éléments. L'instruction `t[0] = 4` provoque une erreur `TypeError`.

On peut en revanche construire un nouveau tuple, par exemple par **concaténation** avec l'opérateur `+` : `(3, 7) + (12,)` vaut `(3, 7, 12)`.

`valeur in t` vaut `True` si `valeur` est un élément de `t`, `False` sinon.

§4. Parcourir un tuple

Un tuple est un objet **itérable**, au même titre que ce que produit `range` (voir Doc. 2,4). On peut donc le parcourir avec une boucle `for`. Il y a deux façons de le faire.

Q8. Recopier les deux fonctions suivantes dans l'éditeur :

```
def somme(valeurs: tuple[float, ...]) -> float:
    """Retourne la somme des éléments du tuple valeurs."""
    total = 0
    for v in valeurs:
```
Python

```
        total += v
    return total


def somme_2(valeurs: tuple[float, ...]) -> float:
    """Retourne la somme des éléments du tuple valeurs."""
    total = 0
    for i in range(len(valeurs)):
        total += valeurs[i]
    return total
```

1. Tester `somme((3, 7, 12))` et `somme_2((3, 7, 12))`.
2. Dans chaque fonction, quelles valeurs prend successivement la variable de boucle (`v` ou `i`) ?
3. L'annotation `tuple[float, ...]` signifie « tuple de nombres flottants, de longueur quelconque ». Pourquoi ne pas avoir écrit `tuple[float, float, float]` ?

La première écriture est plus courte. La seconde n'est utile que lorsqu'on a besoin de connaître la **position** des éléments, comme dans la question suivante.

Q9. Écrire la fonction `indice_max` dont la spécification est :

```python
def indice_max(valeurs: tuple[float, ...]) -> int:
    """Retourne l'indice du plus grand élément de valeurs.
    valeurs ne doit pas être vide.

    >>> indice_max((12, 5, 18, 7))
    2
    """
```

Indication. On suppose d'abord que le plus grand élément est celui d'indice 0, puis on compare cet élément à chacun des suivants.

À retenir

On parcourt un tuple `t` de deux façons :

— `for element in t:` donne directement les éléments, un par un ;
— `for i in range(len(t)):` donne les indices, et on lit l'élément avec `t[i]`. On utilise cette écriture lorsque la position de l'élément compte.

§5. Déballer un tuple

§5.1. L'affectation multiple

Quand un tuple contient peu d'éléments, il est souvent plus clair de donner un nom à chacun d'eux. Python permet de le faire en une seule ligne.

Q10. Saisir les instructions suivantes dans la console, en prévoyant le résultat avant chaque validation :

```python
>>> point = (2, 5)
>>> x, y = point
>>> x
```

```
>>> y
>>> a, b = (1, 2, 3)
```

Expliquer le message d'erreur obtenu à la dernière ligne.

Q11. Reprendre la fonction `min_max` de la première partie. Écrire dans la console une instruction qui range le résultat de `min_max(9, 4)` dans deux variables `petit` et `grand`, puis calculer l'écart entre ces deux nombres.

§5.2. Retour sur l'échange de deux variables

Dans le Doc. 2,2, on a vu que la séquence `a = b` puis `b = a` n'échange pas les valeurs de deux variables, et qu'il fallait passer par une variable `tmp`. Avec les tuples, Python propose une écriture bien plus courte.

Q12. Saisir les instructions suivantes dans la console :

```python
>>> a = 605
>>> b = 506
>>> a, b = b, a
>>> a, b
```

1. Les valeurs ont-elles été échangées ?
2. La séquence `a = b` puis `b = a` échouait parce que la valeur de `a` était écrasée avant d'avoir été recopiée. Expliquer, en décrivant ce que fait Python dans l'ordre, pourquoi ce problème ne se pose pas ici.

Q13. **Pour l'épreuve écrite.** Sans utiliser Python, compléter le tableau d'état de la séquence suivante, puis vérifier dans la console.

```python
a, b = 3, 8
a, b = b, a + b
a, b = b - a, a
```

Instruction	a	b
`a, b = 3, 8`		
`a, b = b, a + b`		
`a, b = b - a, a`		

Q14. Dans le Doc. 2,4, la fonction `fibo` utilisait une troisième variable `w` pour calculer le terme suivant de la suite de Fibonacci. Réécrire cette fonction sans la variable `w`, à l'aide d'une affectation multiple. Vérifier que `fibo(10)` retourne `'1 1 2 3 5 8 13 21 34 55 '`.

À retenir

L'**affectation multiple** `x, y = point` déballe un tuple : chaque variable de gauche reçoit l'élément de même position. Il doit y avoir autant de variables que d'éléments, sinon Python signale une erreur `ValueError`.

Dans `a, b = b, a`, Python évalue d'abord tout le membre de droite, qui forme un tuple, puis affecte les variables. C'est pourquoi cette instruction échange les valeurs de `a` et de `b`.

§6. Écrire une fonction qui renvoie un tuple

Pour annoncer qu'une fonction retourne un tuple, on indique dans sa signature le type de chaque élément, dans l'ordre : `-> tuple[int, int]` signifie « retourne un tuple de deux entiers ». Pour un tuple de longueur quelconque dont tous les éléments ont le même type, on écrit `tuple[float, ...]`.

⚠ Consigne

Écrire chaque fonction dans la partie principale de l'éditeur, avec sa spécification complète (signature et documentation). Tester ensuite chacun des exemples de la documentation dans la console.

Q15. **Division euclidienne.** Écrire la fonction `division_euclidienne` dont la spécification est :

```python
def division_euclidienne(a: int, b: int) -> tuple[int, int]:
    """Retourne le quotient et le reste de la division euclidienne
    de a par b (b non nul).

    >>> division_euclidienne(17, 5)
    (3, 2)
    >>> division_euclidienne(20, 4)
    (5, 0)
    """
```

Comparer avec le résultat de la fonction `divmod(17, 5)`, fournie par Python.

Q16. **Convertir une durée.** Une course de fond a duré 3725 s. On veut exprimer cette durée en heures, minutes et secondes.

1. Sans ordinateur, effectuer la conversion à la main, en détaillant le calcul.
2. Écrire la fonction `convertir_duree` dont la spécification est :

```python
def convertir_duree(duree: int) -> tuple[int, int, int]:
    """Convertit une durée donnée en secondes en un tuple
    (heures, minutes, secondes).

    >>> convertir_duree(3725)
    (1, 2, 5)
    >>> convertir_duree(59)
    (0, 0, 59)
    """
```

Q17. **Milieu d'un segment.** Dans un repère, un point est représenté par le tuple `(x, y)` de ses coordonnées. Écrire la fonction `milieu` dont la spécification est :

```python
def milieu(a: tuple[float, float], b: tuple[float, float]) -> tuple[float, float]:
    """Retourne les coordonnées du milieu du segment [AB].

    >>> milieu((0, 0), (4, 2))
    (2.0, 1.0)
    >>> milieu((-1, 3), (1, 3))
    (0.0, 3.0)
    """
```

Indication. Commencer par déballer chacun des deux tuples.

Q18. **Distance entre deux points.** Dans le Doc. 2,1, la fonction `distance` recevait quatre arguments `x1, y1, x2, y2`. Avec des tuples, deux arguments suffisent. On veut de plus que la fonction marche aussi dans l'espace, avec des points à trois coordonnées. Dans un repère orthonormé de l'espace, la distance entre $A(x_A, y_A, z_A)$ et $B(x_B, y_B, z_B)$ est :

$$AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2}$$

1. Écrire la fonction `distance` dont la spécification est :

```python
def distance(a: tuple[float, ...], b: tuple[float, ...]) -> float:
    """Retourne la distance entre les points a et b, qui doivent
    avoir le même nombre de coordonnées.

    >>> distance((0, 0), (3, 4))
    5.0
    >>> distance((1, 2, 3), (1, 2, 3))
    0.0
    >>> distance((0, 0, 0), (1, 2, 2))
    3.0
    """
```

2. Que devrait faire la fonction si on l'appelle avec `distance((0, 0), (1, 1, 1))` ? Ajouter au début du corps de la fonction l'instruction :

```python
assert len(a) == len(b), "les deux points doivent avoir la même dimension"
```

puis tester à nouveau cet appel.

À retenir

L'instruction `assert condition, "message"` vérifie qu'une condition est vraie. Si elle est fausse, le programme s'arrête avec une erreur `AssertionError` et affiche le message.

On s'en sert pour vérifier une **précondition**, c'est-à-dire une condition que les arguments doivent respecter pour que la fonction ait un sens. Mieux vaut un programme qui s'arrête en expliquant pourquoi qu'un programme qui retourne un résultat faux sans prévenir.

Q19. **Plus petit et plus grand élément.** Écrire la fonction `min_max_tuple` dont la spécification est :

```python
def min_max_tuple(valeurs: tuple[float, ...]) -> tuple[float, float]:
    """Retourne le plus petit et le plus grand élément de valeurs,
    qui ne doit pas être vide.

    >>> min_max_tuple((12, 5, 18, 7))
    (5, 18)
    >>> min_max_tuple((4,))
    (4, 4)
    """
```

Les fonctions `min` et `max` de Python ne doivent pas être utilisées. Vérifier la précondition par une assertion.

§7. Exercices de synthèse

Q20. **Équation du second degré.** On cherche les solutions réelles de l'équation $ax^2 + bx + c = 0$, avec $a \neq 0$. On note $\Delta = b^2 - 4ac$ le discriminant.

- Si $\Delta > 0$, il y a deux solutions : $x_1 = \frac{-b-\sqrt{\Delta}}{2a}$ et $x_2 = \frac{-b+\sqrt{\Delta}}{2a}$.
- Si $\Delta = 0$, il y a une seule solution : $x_0 = -\frac{b}{2a}$.
- Si $\Delta < 0$, il n'y a pas de solution réelle.

Le nombre de solutions change d'une équation à l'autre. Un tuple convient bien : il peut contenir deux, un ou zéro élément. Écrire la fonction `racines` dont la spécification est :

```python
def racines(a: float, b: float, c: float) -> tuple[float, ...]:
    """Retourne le tuple des solutions réelles de a x² + b x + c = 0,
    avec a non nul.

    >>> racines(1, -3, 2)
    (1.0, 2.0)
    >>> racines(1, 2, 1)
    (-1.0,)
    >>> racines(1, 0, 1)
    ()
    """
```

Q21. **Comparer deux dates.** Une date est représentée par un tuple `(jour, mois, annee)`. Écrire la fonction `est_avant` dont la spécification est :

```python
def est_avant(d1: tuple[int, int, int], d2: tuple[int, int, int]) -> bool:
    """Retourne True si la date d1 est strictement antérieure
    à la date d2, False sinon.

    >>> est_avant((14, 7, 1789), (11, 11, 1918))
    True
    >>> est_avant((25, 12, 2025), (1, 1, 2025))
    False
    >>> est_avant((3, 5, 2026), (3, 5, 2026))
    False
    >>> est_avant((3, 5, 2026), (4, 5, 2026))
    True
    """
```

Indication. Comparer d'abord les années. Si elles sont égales, comparer les mois, et ainsi de suite.

§8. Bilan

À l'issue de cette activité, je sais…

- créer un tuple, y compris un tuple d'un seul élément `(5,)` et le tuple vide `()` ;
- lire un élément d'un tuple grâce à son indice, en sachant que les indices vont de `0` à `len(t) - 1` ;

- expliquer pourquoi `t[0] = 4` provoque une erreur, et construire un nouveau tuple à la place ;
- tester la présence d'une valeur dans un tuple avec `in` ;
- parcourir un tuple par ses éléments ou par ses indices, et choisir l'écriture adaptée ;
- déballer un tuple par une affectation multiple, et expliquer pourquoi `a, b = b, a` échange deux valeurs ;
- écrire une fonction qui renvoie un tuple de valeurs, avec une signature du type `-> tuple[int, int]` ;
- vérifier une précondition à l'aide d'une assertion.