## Exercice 1 – Distinction liste / tuple

1. Parmi les déclarations suivantes, lesquelles créent un **tuple** et lesquelles créent une **liste** ?

   a. `x = [1, 2, 3]` b. `y = (1, 2, 3)` c. `z = 1, 2, 3` d. `t = (42,)` e. `u = [42]` f. `v = ("a")`

2. On exécute le code suivant :

   ```
   notes = (12, 15, 18)
   notes[0] = 10
   ```

   a. Que se passe-t-il ? b. Explique en une phrase pourquoi c'est un comportement **important**.

3. Transforme la liste `L = [4, 5, 6]` en tuple, puis le tuple `(7, 8, 9)` en liste (écrire les deux lignes de code).

---

## Exercice 2 – Dépaquetage de tuples

1. Que contiennent `a` et `b` après l'exécution du code suivant ?

   ```
   coord = (10, 20)
   a, b = coord
   ```

2. Que contient `x`, `y` et `z` ?

   ```
   x, y, z = (1, 2, 3)
   ```

3. On veut échanger les valeurs de deux variables `x` et `y` **sans utiliser de variable temporaire**. Complète le code :

   ```
   x = 5
   y = 9

   # échange de x et y en une seule ligne grâce aux tuples
   ____________________________

   print(x, y)  # doit afficher: 9 5
   ```

4. Avec l'opérateur * (dépaquetage étendu) :

   Que contiennent `premier`, `milieu` et `dernier` ?

```
données = (10, 20, 30, 40, 50)
premier, *milieu, dernier = données
```

---

## Exercice 3 – Tuples comme retour de fonction

On souhaite écrire une fonction qui prend un prix TTC (toutes taxes comprises) et renvoie **deux valeurs** :

- le prix HT (hors taxe)
- le montant de la TVA

On suppose une TVA de 20 %.

1. Complète la fonction :

```
def decomposer_prix(prix_ttc):
    """
    Renvoie un tuple (prix_ht, tva)
    avec une TVA de 20%
    """
    # écrire le calcul ici
    ____________________________
    ____________________________
    return (prix_ht, tva)
```

2. Utilise cette fonction pour afficher un message clair pour un prix TTC de 120 :

```
prix_ttc = 120
# appeler la fonction ici

_____________________________________
print("Prix HT :", prix_ht)
print("TVA :", tva)
```

3. Explique en une ou deux phrases pourquoi il est **pratique** que la fonction renvoie un tuple plutôt que deux valeurs séparées dans deux variables globales.

---

## Exercice 4 – Manipulation de tuples en pratique

On a la liste suivante de points 2D (x, y) :

```
points = [(1, 2), (3, 4), (-1, 5), (0, 0)]
```

1. Écrire une boucle qui affiche chaque point sur une ligne, sous la forme :

   ```
   Point : x = ..., y = ...
   ```

2. Calcule la **distance au carré** de chaque point à l'origine (0, 0), c'est-à-dire `x*x + y*y`, et affiche :

   ```
   Point (1, 2) : distance² = 5
   ...
   ```

3. Écris une fonction `point_le_plus_lointain(points)` qui prend une liste de tuples `(x, y)` et renvoie **le tuple du point le plus éloigné** de l'origine (toujours avec la distance au carré, pas besoin de racine carrée).

   Signature :

   ```
   def point_le_plus_lointain(points):
       # à compléter
   ```

---

## Exercice 5 – Création de tuples imbriqués (niveau ++)

On modélise un agenda simplifié : chaque rendez-vous est un tuple :

```
(nom, (heure_debut, heure_fin))
```

Par exemple :

```
rdv1 = ("Dentiste", (9, 10))
rdv2 = ("Cours de sport", (18, 19))
```

1. Crée trois rendez-vous différents de ton choix, puis mets-les dans une liste `agenda`.

2. Écris une boucle qui affiche chaque rendez-vous sous la forme :

```
Dentiste : de 9h à 10h
```

en utilisant le dépaquetage de tuples.

3. Écris une fonction `est_en_conflit(rdv_a, rdv_b)` qui renvoie `True` si deux rendez-vous **se chevauchent** (même heure ou intersectent), `False` sinon.

   Aide :

   - `rdv_a` et `rdv_b` sont des tuples comme ci-dessus.
   - Deux intervalles `(a_debut, a_fin)` et `(b_debut, b_fin)` se chevauchent si :

     ```
     a_debut < b_fin and b_debut < a_fin
     ```