Les tuples (ou p-uplets)

Les tuples sont des séquences, c'est à dire des regroupement de **plusieurs valeurs/éléments sous un seul nom**. Les termes p-uplet ou tuple sont utilisés indifféremment pour désigner cette généralisation des couple, triplet, quadruplet, etc. Ces informations peuvent être de différents types au sein d'un même tuple.

Pour définir un tuple, on utilise des parenthèses pour contenir les différentes valeurs et des virgules pour les séparer.

```
coord = (2, -3)
print(coord)          # affiche (2, -3)
print(type(coord))    # affiche <class 'tuple'>
```

On accède aux éléments d'un tuple un par un. Pour cela on utilise des crochets (brackets en anglais) contenant l'indice (index en anglais) de l'élément. Le premier élément est contenu à l'indice 0.

```
coord = (2, -3)
print(coord[0])       # affiche 2
print(coord[1])       # affiche -3
print(coord[2])       # affiche IndexError: tuple index out of range

personne = ('Guido', 'van Rossum', '31/01/1956')
print("Le créateur de Python est né le", personne[2])
# affiche Le créateur de Python est né le 31/01/1956
```

Pour connaître le nombre d'élément d'un tuple, aussi appelé sa longueur ou sa taille, on utilise la fonction python `len` .

```
coord = (2, -3)
personne = ('Guido', 'van Rossum', '31/01/1956')
print(len(coord), len(personne))                  # Affiche 2 3
```

Connaître la longueur d'un tuple permet d'en parcourir tous les éléments avec une boucle. Il existe deux parcours :

Le parcours par indice : la variable de boucle prend successivement tous les indices valides du tuple. On utilise `len` dans le range, ainsi la variable de boucle va prendre les valeur de `0` à `taille_du_tuple - 1`.

```
personne = ('Guido', 'van Rossum', '31/01/1956')
for i in range( len(personne) ) :
    print(personne[i])

# Affiche successivement 'Guido', 'van Rossum' et '31/01/1956'
```

Le parcours par valeur ou parcours par élément : la variable de boucle prend successivement toutes les valeurs du tuple. On n'utilise cette fois pas de range, mais directement le tuple.

```
coord = (2, -3)
for ele in coord :
    print(ele)

# affiche successivement 2 et -3
```

On ne peut pas modifier les valeurs d'un tuple, on dit qu'il est non mutable. Les tuples sont aussi de taille statique : on ne peut pas ajouter ou redimensionner un tuple existant.

Exercice 1 : Predict, run, investigate.

Sans exécuter, qu'affichent les codes suivants ? Vérifier en exécutant le code.

`# Code 1`
 	
`courses = ('pain', 'BBQ', 'lait', 'gel')`
`print(courses[1])`
`print(courses[4])`
	`# Code 2`
	
	`notes = (20, 14.2, 14.1, 16)`
	`print(len(notes))`

`# Code 3`
 	
`notes = (20, 15, 15, 17)`
`m = 0`
`for n in notes:`
`    m = m + n`
`print(m/len(notes))`
	`# Code 4`
	
	`coord = (12.1, 15.2)`
	`def myst(c):`
	`    return c[1]`
	
	`print(myst(coord))`

Exercice 2 : Modify.

1) Modifier le code 1 de l'exercice 1 pour afficher les éléments `'pain'` et `'lait'` du tuple course.

2) Modifier le code 3 de l'exercice 1 en utilisant un parcours par indice. Le résultat doit être le même.

3) En modifiant le code 3 de l'exercice 1, écrire un programme qui calcule une moyenne mais cette fois pondérée, en ajoutant et en utilisant le tuple de coefficients suivant : `coef = (1, 2, 3, 1)`. Le résultat doit être égal à 16.

4) En modifiant le code 4 de l'exercice 1, écrire une fonction `somme` prenant en paramètre deux tuples de coordonnées (x, y) et qui renvoie un nouveau tuple contenant $(x_1 + x_2, y_1 + y_2)$. `print(somme((1,2), (3, 4))) #(4, 6)`

Exercice 3 : Make.

1) Écrire une fonction `temps_vers_secondes(s)` qui prend en paramètre un tuple de la forme (heures, minutes, secondes) et renvoie ce temps converti en seconde.

```
t = (1, 23, 45) # 1H, 23 mins, 45 secs
print(temps_vers_secondes(t)) # affiche 5025
```

2) Écrire une fonction `seconde_vers_temps(s)` qui prend en paramètre une durée en secondes et qui renvoie le résultat dans un tuple de la forme (heures, minutes, secondes).

```
print(seconde_vers_temps(5025)) # affiche (1, 23, 45)

print(seconde_vers_temps(temps_vers_secondes((1, 23, 45))) == (1, 23, 45)) # True
print(temps_vers_secondes(seconde_vers_temps(5025)) == 5025)               # True
```

3) Écrire la fonction `somme(t1, t2)` qui prend en paramètre deux tuples de la forme (heures, minutes, secondes) et renvoie la somme de ces temps.

```
print(somme((1, 23, 45), (5, 43, 21))) # affiche (7, 7, 6)
```

4) Écrire la fonction `decimal(t)` qui prend en paramètre un tuple contenant des chiffres de la forme (... , centaines, dizaines ,unités) et qui retourne le nombre correspondant.

```
print(decimal( (1, 2, 3) ))   # 1*100 + 2*10 + 3*1 | affiche 123
print(decimal( (0, 0, 1, 0, 1) ))    # affiche 101
print(decimal( (5, 4, 3, 2, 1) ) + decimal( (4, 5, 6, 7, 9) ) ) # affiche 100000
```

5) Morpion

On décide de représenter une grille de morpion de taille 3 x 3 par un tuple contenant 9 valeurs :

1	2	3
4	5	6
7	8	9

Grille = (1, 2, 3, 4, 5, 6, 7, 8, 9)

Les valeurs du tuple peuvent être prise parmi les valeurs suivantes :

- '_' pour indiquer que la case n'a pas été jouée.
- 'x' pour indiquer que le joueur 1 a joué sur la case.
- 'o' pour indiquer que le joueur 2 a joué sur la case.

Voici un exemple de tuple représentant une grille gagnante (horizontalement) pour le joueur 1 :

```
g0 = ('x', 'x', 'x', 'o', '_', 'o', 'o', '_', '_')
```

On se limite par la suite aux grilles gagnantes verticales.

1. Donner 2 tuples g1 et g2 représentant des grilles gagnantes pour le joueurs 1.

2. Donner 2 tuples g3 et g4 représentant des grilles gagnantes pour le joueurs 2.

3. Donner 2 tuples g5 et g6 représentant des grilles sans gagnant (sans victoire ni verticale, ni horizontale, ni diagonale).

4. Écrire la fonction `victoire_verticale(g)` qui prend en paramètre un tuple représentant une grille et qui retourne 'x', 'o' ou '_' si la grille est gagnante pour le joueur 1, le joueur 2 ou aucun des joueurs respectivement.
   Tester votre fonction avec les grilles des questions 1, 2 et 3.

```
g7 = ('x', '_', 'o', '_', '_', 'o', 'x', '_', '_')
v = victoire_verticale(g7)
print(v)   # Affiche _
```

5. Écrire une fonction `initialisation_grille()` qui retourne une grille vide.

```
print(initialisation_grille()) #affiche ('_', '_', '_', '_', '_', '_', '_', '_', '_')
```

6. Écrire une fonction `jouable(g, n)` qui retourne `True` si la case `n` (en suivant la numérotation vue précédemment) de la grille `g` n'a pas encore été jouée et `False` sinon.

```
g7 = ('x', '_', 'o', '_', '_', 'o', 'x', '_', '_')
print(jouable(g6, 1))      # affiche False
print(jouable(g6, 4))      # affiche True
```

7. Écrire une fonction jouer(g, n, j) qui retourne une copie de la grille g en modifiant la case n avec la valeur j.
   Aucune vérification n'est à faire pour savoir si la case a déjà été jouée ou non.

```
g7 = ('x', '_', 'o', '_', '_', 'o', 'x', '_', '_')
g8 = jouer(g7, 4, 'o')
print(g8) # affiche ('x', '_', 'o', 'o', '_', 'o', 'x', '_', '_')
```