Cours 1 : Exemples corrigés

I Données et variables

Affectation des variables

Exemple 1. *Affectation simultanée*

```
x = y = 7
```

`>>> x, y, id(y) == id(x)` *renvoie* `(7, 7, True)`.

Exemple 2. *Affectation parallèle*

```
a, b, c, d = 4, 4, 8.33, 8.33
```

`>>> a == b, a is b, c == d, c is d` *renvoie* `(True, True, True, True)`.

Exemple 3. *Incrémentation*
`>>> a = a + 1; a += 1; a + 1 = g` *renvoie*
`File "<console>", line 1` *et* `SyntaxError: can't assign to operator`.

Types de données

Exemple 4. *Données numériques*

```
a = 5; b = float(a)
```

`>>> a, b, a is b` *renvoie* `(5, 5.0, False)`.

Exemple 5. *Données alphanumériques*

```
phrase1 = '''C'est un beau roman'''
phrase2 = "C'est une belle histoire"
phrase3 = 'C'est une romance d'aujourd'hui'

couplet = phrase1 + \n + phrase2 + \n + phrase3
tableau = 'a \t b \t c'
```

Corrigé des lignes :

```
phrase3 = 'C\'est une romance d\'aujourd\'hui'
couplet = phrase1 + '\n' + phrase2 + '\n' + phrase3
```

- `print(phrase1[1])` *renvoie* `'`.
- `print(phrase2[-1])` *renvoie* `e`.
- `print(phrase3[2:5])` *renvoie* `est`.
- `print(couplet)` *renvoie les trois lignes suivantes :* `C'est un beau roman`, `C'est une belle histoire` *et* `C'est une romance d'aujourd'hui`.
- `print(tableau)` *renvoie* `a  b  c`.

Exemple 6. *Listes*

```
liste_exemple = ['lundi', 4, 8.33, ['mot', 3]]
```

- `print(liste_exemple[1])` *renvoie* `4`.
- `print(liste_exemple[3][0])` *renvoie* `mot`.

Propriétés d'une séquence

Exemple 7.
```
nom = 'Robert'
for lettre in nom:
    print(lettre)
for element in liste_exemple:
    print(element, 'de type', type(element))
3 in liste_exemple, 3 in liste_exemple[3]
```

La console renvoie les 6 caractères de `nom` *sur 6 lignes, puis les 4 lignes suivantes :*

```
lundi de type <class 'str'>
4 de type <class 'int'>
8.33 de type <class 'float'>
['mot', 3] de type <class 'list'>
```

Et enfin, elle renvoie le tuple `(False, True)`.

II Notion de classe et d'objet

Exemple 8. *Méthodes à associer à la classe Point :*
- `format_tuple(self)` *qui retourne le point sous la forme d'un tuple.*
- `format_list(self)` *qui retourne le point sous la forme d'une liste.*

```
    def format_tuple(self) :
        '''Retourne le Point sous la forme d'un tuple.'''
        return((self.abscisse, self.ordonnee, self.cote))

    def format_list(self) :
        '''Retourne le Point sous la forme d'une liste.'''
        return([self.abscisse, self.ordonnee, self.cote])
```

Exemple 9. *Définition d'une nouvelle classe* `Vecteur` *avec les méthodes suivantes :*
- *une méthode constructeur* `__init__` *qui définit le vecteur nul par défaut.*
- `affichage(self)` *qui affiche le vecteur sous forme d'une chaîne de caractères.*
- `format_tuple(self)` *qui retourne le vecteur sous la forme d'un tuple.*
- `format_list(self)` *qui retourne le vecteur sous la forme d'une liste.*

```
class Vecteur :
    '''Définition d'un vecteur en coordonnées cartésiennes 3D.'''
    def __init__(self, vx = 0, vy = 0, vz = 0):
        '''Constructeur du Vecteur.'''
        self.abscisse = vx
        self.ordonnee = vy
        self.cote = vz

    def affichage(self) :
        '''Affiche le Vecteur sous forme d'une chaîne de caractères.'''
        print ('(', str(self.abscisse), ',', str(self.ordonnee), ',', \
               str(self.cote), ')')

    def format_tuple(self) :
        '''Retourne le Vecteur sous la forme d'un tuple.'''
        return((self.abscisse, self.ordonnee, self.cote))

    def format_list(self) :
        '''Retourne le Vecteur sous la forme d'une liste.'''
        return([self.abscisse, self.ordonnee, self.cote])
```

Ajout de la méthode suivante à la classe Point :
- `bipoint(self, p)` *qui retourne le vecteur (Point, p).*

```
    def bipoint(self, p) :
        '''Retourne le vecteur (Point, p).'''
        return(Vecteur(p.abscisse - self.abscisse, p.ordonnee - self.ordonnee,\
                       p.cote - self.cote))
```

Redéfinition des opérateurs dans une classe

Exemple 10. *Redéfinition de l'opérateur* `*` *pour qu'il retourne un* `Vecteur` *qui est le Vecteur initial multiplié par un réel k. Remarquer le défaut de commutativité de l'opérateur défini.*

```
def __mul__(self, k):
    '''Retourne un Vecteur qui est le Vecteur initial multiplié par k.'''
    return(Vecteur(self.abscisse * k, self.ordonnee * k, self.cote * k))
```

III Compléments sur les chaînes de caractères

Comparaison de deux chaînes

Exemple 11. `>>> 'Robert' < 'Jean', 'Robert' < 'jean'` *renvoie* `(False, True)`.

Méthodes associées aux objets chaînes

Exemple 12.
— `couplet.split('\n')` *renvoie* `["C'est un beau roman", "C'est une belle histoire", "C'est une romance d'aujourd'hui"]`.
— `phrase1.split()` *renvoie* `["C'est", 'un', 'beau', 'roman']`.
— `couplet.find("'")` *renvoie* `1`.
— `couplet.find('beau')` *renvoie* `9`.
— `couplet.index('y')` *renvoie* `Traceback (most recent call last): File "<console>", line 1, in <module> ValueError: substring not found`.
— `couplet.find('y')` *renvoie* `-1`.
— `couplet.count("'")` *renvoie* `5`.
— `couplet.count("C'est")` *renvoie* `3`.
— `nom.lower()` *renvoie* `'robert'`.
— `nom.upper()` *renvoie* `'ROBERT'`.
— `nom.swapcase()` *renvoie* `'rOBERT'`.
— `'jean'.capitalize()` *renvoie* `'Jean'`.
— `'il fait beau. tout va bien.'.capitalize()` *renvoie* `'Il fait beau. tout va bien.'`.
— `'  Mot à  mot \n  '.strip()` *renvoie* `'Mot à  mot'`.

IV Compléments sur les listes

Méthodes associées aux objets listes

Exemple 13. `noms = ['Robert', 'Jean', 'Lucie']`
— `noms.sort()` *modifie* `noms` *qui devient* `['Jean', 'Lucie', 'Robert']`.
— `noms.append('Arthur')` *modifie* `noms` *qui devient* `['Jean', 'Lucie', 'Robert', 'Arthur']`.
— `noms.reverse()` *modifie* `noms` *qui devient* `['Arthur', 'Robert', 'Lucie', 'Jean']`.
— `noms.index('Jean')` *renvoie* `3`.
— `noms.remove('Robert')` *modifie* `noms` *qui devient* `['Arthur', 'Lucie', 'Jean']`. *del(noms[1])/ modifie* `noms` *qui devient* `['Arthur', 'Jean']`.

Le 'slicing' ou découpage en tranche

Exemple 14. — `lettres = [chr(code) for code in range(ord('a'), ord('a') + 5)]` *construit la liste* `['a', 'b', 'c', 'd', 'e']`.
- `lettres[3:3] = ['\\']` *modifie* `lettres` *qui devient* `['a', 'b', 'c', '\\', 'd', 'e']`.
- `lettres[6:6] = ['f', 'g', 'h']` *modifie* `lettres` *qui devient* `['a', 'b', 'c', '\\', 'd', 'e', 'f', 'g', 'h']`.
- `lettres[1:2] = []` *modifie* `lettres` *qui devient* `['a', 'c', '\\', 'd', 'e', 'f', 'g', 'h']`.
- `lettres[4:6] = ['i']` *modifie* `lettres` *qui devient* `['a', 'c', '\\', 'd', 'i', 'g', 'h']`.
- `lettres[4:] = ['c']` *modifie* `lettres` *qui devient* `['a', 'c', '\\', 'd', 'c']`.

Enfin,

```
groupe = ''
for element in lettres :
    groupe += element.upper()
print(groupe)
```

renvoie `AC\DC`.

Copie de listes

Exemple 15. *Indiquer ce que renvoie la console pour les lignes de commandes suivantes.*
- `liste_1 = list(range(1, 4))` *construit la liste* `[1, 2, 3]`.
- `liste_2 = liste_1` *crée un simple alias de* `liste_1` *et non une vraie copie.*
- `liste_1[0] = 'a'` *modifie* `liste_1` *qui devient* `['a', 2, 3]`, `liste_2` *est elle aussi modifiée car elle pointe sur la même adresse que* `liste_1`.
- `liste_2[2] = 'c'` *modifie* `liste_2` *qui devient* `['a', 2, 'c']`.
- `liste_1, liste_2` *renvoie donc les deux mêmes listes :* `['a', 2, 'c']`.