Initiation à la programmation
Seconde partie du cours
Exercices

Exercice 1a : QCM (Exercice d'examen)

A. Que produira l'instruction <print> du morceau de code suivant :

```
01: liste_A = [9,5,8,7,6,2,1,3,4]
02: liste_B = []
03: liste_B.append(liste_A[4])
04: liste_B.append(liste_A[6])
05: liste_A.sort()
06: liste_B.append(liste_A[6])
07: liste_B.reverse()
08: print liste_B[1]+liste_B[2]
```

a. 7
b. 8
c. 9
d. erreur !

B. Quel est le résultat du fragment de code suivant :

```
01: compteur = 2
02: a,b = 1,1
03: while True:
04:     if a > 5 and b > 5:
05:        break
06:     for i in range(1,3):
07:         b+=1
08:         for j in range(3):
09:             a+=1
10: print a+b
```

a. 10
b. 16
c. 20
d. 26

C. Que produira l'instruction <print> du morceau de code suivant :

```
01: chaine_A = "bonjour"
02: chaine_B = "salut"
03: chaine_A = chaine_A[1:-3]
04: chaine_B = chaine_B[3::-1]
05: chaine_C = chaine_A + str(len(chaine_A)) + chaine_B
06: print chaine_C[-6:6]
```

a. o4ul
b. j3ul
c. nj4l
d. jo5

D. Que produira l'instruction <print> du morceau de code suivant :

```
01: stock = {'Banane':17, 'Poire':17, 'Pomme':25}
02: stock['Poire'] = stock['Poire'] + 5
03: print 'Kiwi :', stock.get('Kiwi', 'Pas en stock')
04: print 'Poire :', stock['Poire']
```

a. Kiwi : Pas en stock
   Poire : 17
b. Kiwi : 0
   Poire : 22
c. Kiwi : Pas en stock
   Poire : 22
d. Kiwi : Pas en stock
   Poire : Erreur

Exercice 1b : QCM

E. Que produira l'instruction <print> du morceau de code suivant :

```
a = [1, 2, 3]
b = a
c = "cou"
a.reverse()
b.reverse()
c.upper()
print b + [c]
```

a. [1, 2, 3, 'cou']
b. [1, 2, 3, 'COU']
c. [3, 2, 1, 'cou']
d. [3, 2, 1, 'COU']

F. Que produira l'instruction <print> du morceau de code suivant :

```
li1 = [1, "a"]
li2 = [2, "b"]
a = (li1, li2)
b = tuple(a)
b[0][0] = 0
print a
b[0] = 0
print a
```

e. ERREUR !

f. ([1, 'a'], [2, 'b'])
([1, 'a'], [2, 'b'])

g. ([0, 'a'], [2, 'b'])
(0, [2, 'b'])

h. ([0, 'a'], [2, 'b'])
ERREUR !

G. Que produira l'instruction <print> du morceau de code suivant :

```
li = ["z", "m", "a"]
tu = tuple(li)
print li.sort()
print sorted(li) == li
print sorted(tu)
```

a. ERREUR !

b. None
True
['a', 'm', 'z']

c. ['a', 'm', 'z']
False
ERREUR !

d. ['a', 'm', 'z']
False
['a', 'm', 'z']

H. Que produira l'instruction <print> du morceau de code suivant :

```
dico = {}
dico[2] = 3
dico[1] = 4
for entree in sorted(dico.items()):
    print "%i-%i" % (entree[1], dico.get(entree[0] + 1, 5))
```

a. 3-5
4-3

b. 1-4
2-3

c. 4-2
3-3

d. 4-3
3-5

Exercice 2 : Théorie (Exercice d'examen)

Les questions suivantes parcourent la théorie :

a) **Donner la ou les différences (maximum 2, les plus importantes) qu'il y a entre les caractéristiques des couples de type suivants :**
   a. **List & Tuple :**

   b. **List & String :**

   c. **Tuple & String :**

   d. **Dict & (List, Tuple et String) :**

b) **Est-il possible d'accéder aux éléments présents dans un dictionnaire à l'aide de la technique du slicing ? Si oui, montrez un petit exemple / Si non, dites pourquoi ?**

c) **Soit deux variables** `<a et b>`**, contenant respectivement une liste** `[1,2]` **et un entier** `5`**. Comment créer (de la manière la plus concise) deux nouvelles variables** `<montuple1 et montuple2>` **en se basant uniquement sur les variables d'origine, de telle sorte que ces variables contiennent respectivement :**

   `montuple1` ← **un tuple avec tous les éléments suivants dans l'ordre : 1, 2, 1, 2, 1, 2, 1, 2, 1, 2**

   `montuple2` ← **un tuple avec le seul élément 5**

Exercice 4 : Etoiles (Presque exercice d'examen)

Ecrivez un script Python en utilisant <u>des boucles "*while*" ou "*for*" imbriquées</u> qui affichera la figure suivante composée uniquement d'étoiles et d'espaces (les commentaires à droite ne sont que des indications pour vous aider):

```
01: *          ****          # 1 étoile - 1 espace - 9 espaces - 4 étoiles
02: *       ***              # 1 étoile - 1 espace - 6 espaces - 3 étoiles
03: *    **                  # 1 étoile - 1 espace - 3 espaces - 2 étoiles
04: * *                      # 1 étoile - 1 espace - 0 espace - 1 étoile
05: *    **                  # 1 étoile - 1 espace - 3 espaces - 2 étoiles
06: *       ***              # 1 étoile - 1 espace - 6 espaces - 3 étoiles
07: *          ****          # 1 étoile - 1 espace - 9 espaces - 4 étoiles
```

Exercice 5 : Compréhension (Exercice d'examen)

Commentez, décrivez, analysez le fragment de code qui suit. C'est-à-dire que vous devez donner ce que va imprimer ce code, mais vous devez également préciser comment il arrive à ce résultat.

```
01: dico_points = {'H30001':[12,15,8,9,17],
                   'H30003':[12,15,'disp',9,16],
                   'H30002':['abs',15,'disp',9,'abs']}
02: clefs = dico_points.keys()
03: clefs.sort()
04: for clef in clefs:
05:     somme_points = 0.0
06:     nb_points = 0
07:     for point in dico_points[clef]:
08:          if str(point).isdigit():
09:              somme_points += point
10:              nb_points += 1
11:          elif point == "abs":
12:              somme_points += 0
13:              nb_points += 1
14:     moyenne = somme_points / nb_points
15:     print clef + " : " + str(moyenne)
```

Exercice 6 : Traduction d'un algorithme (Exercice d'examen)

Soit une variable qui contient une chaîne de caractères, prenons comme hypothèse qu'elle ne contient que des chiffres de « 0 » à « 9 » (sans espaces, ponctuations ou tout autre caractère). En voici un exemple ;

```
01: txt = '235498135792130546'
```

Il est demandé d'écrire un petit programme qui va détecter la plus longue suite consécutive d'augmentation entre deux chiffres consécutifs et afficher sur la sortie standard la taille de cette suite. Le résultat sur la variable donnée en exemple serait :

```
02: print nb_up_conseq → 4
                         (qui provient des 4 augmentations successives
                          entre les 5 chiffres 13579)
```

Attention, le programme doit fonctionner quelques soit la suite de chiffres. Afin de vous guider dans la conception de votre programme, voici un algorithme qui fonctionne :

> **Deux variables (**`nb_up_conseq` **et** `nb_serie_en_cours`**) sont créées et initialisées à 0. Ensuite, on parcourt à l'aide d'une boucle <for> et d'une variable nommée** `<ind>` **une liste qui va de l'indice** `0` **à l'indice de l'avant-dernier élément de la chaîne de caractère. Le corps de cette boucle contient un test, si le caractère qui se trouve en position** `« ind »` **est plus grand ou égal que celui qui se trouve en position** `« ind+1 »`**, alors la variable** `nb_serie_en_cours` **est remise à** `0`**, sinon** `nb_serie_en_cours` **est incrémentée de** `1`**. Finalement le corps de la boucle contient un deuxième test, si** `nb_serie_en_cours` **est supérieur à** `nb_up_conseq` **alors cette deuxième variable prend la valeur actuelle de la variable** `nb_serie_en_cours`**. Le corps de la boucle se termine et il ne reste plus qu'à afficher le résultat attendu qui se trouve de la variable** `nb_up_conseq`**.**

Implémentez l'algorithme proposé :

Exercice 7 : Traduction d'un algorithme (Exercice d'examen)

Soit une variable qui contient une chaîne de caractères, prenons comme hypothèse qu'elle ne contient pas de ponctuation, mais que certains espaces ont été répliqués plusieurs fois entre les mots. Cette chaîne de caractères pourrait être la suivante :

```
01: txt = 'lmdk    qfmskl  mslkdf lm  fsjq flmqkjf    mslfk qmls   '
```

Afin de « nettoyer » une telle chaîne de caractères, il vous est demandé de créer 2 scripts différents. Ces scripts auront tous deux exactement le même effet, ils vont créer une nouvelle chaîne de caractères. Celle-ci n'aura plus d'espace dupliqué, il n'en restera plus qu'un entre chaque mot. De plus, cette chaîne de caractères n'aura plus d'espace à la fin. Le résultat sur la chaîne donnée en exemple serait :

```
01: print txt_new → 'lmdk qfmskl mslkdf lm fsjq flmqkjf mslfk qmls'
```

Il vous est interdit d'utiliser la fonction prédéfinie `<split()>`, sans paramètre sur la chaîne txt. Par contre, vous pouvez utiliser cette fonction en lui donnant comme paramètre <' '> qui est une chaîne de caractère ne contenant qu'un espace. Pour vous aider, voici 3 algorithmes qui fonctionnent :

a. **Commencez en séparer votre chaîne à l'aide de la fonction `<txt.split(' ')>`, ceci vous donnera une liste contenant les mots et des chaînes de caractères vides (ces derniers proviennent de la séparation entre deux espaces consécutifs). Créer une nouvelle liste vide et remplissez là en parcourant la liste précédente et en ne gardant que les chaînes différentes d'une chaîne vide. Réassemblez votre nouvelle liste dans une nouvelle chaîne de caractères à l'aide de la fonction `<' '.join()>`, appliquée sur une chaîne ne contenant qu'un espace et qui prend comme paramètre votre nouvelle liste.**
b. **Créez une chaîne ne contenant que le premier caractère de txt. Remplissez cette nouvelle chaîne en parcourant la chaîne txt caractère par caractère en partant du second caractère. Les caractères différents du caractère espace seront intégrés à la nouvelle chaîne, par contre, les caractères espace ne seront intégré que si le dernier caractère intégré n'était pas lui-même un espace (pour tester ceci, vous devrez retenir le dernier caractère intégré dans une variable). Après ce parcours, vous devrez encore éliminer le caractère espace qui sera encore présent à la fin de votre nouvelle chaîne (par exemple en utilisant la fonction `<strip()>` sur votre nouvelle chaîne de caractères).**
c. **Créez une chaîne égale à txt. Ensuite, vous allez utiliser la fonction `<replace(old_string,new_string)>` sur cette nouvelle chaîne. Cette fonction prend deux paramètres, le premier (old_string) est la chaîne de caractères à remplacer (mettez une chaîne ne contenant que deux espaces), le deuxième (new_string) est la chaîne de caractère qui va être utilisé pour remplacer le premier paramètre (mettez une chaîne ne contenant qu'un espace). Cette fonction ne change pas la chaîne sur laquelle elle est utilisée, mais elle renvoie une chaîne qui intègre la modification souhaitée. Pour utiliser cette fonction, vous allez donc l'appliquer à la variable contenant votre nouvelle chaîne (copie de la chaîne initiale) et introduire le résultat de la fonction dans la variable sur laquelle vous venez de l'appliquer (vous allez donc écraser l'ancienne valeur de votre variable par la nouvelle). Faites ce mécanisme tant que vous trouverez des doubles espaces par la méthode `<find('  ')>` qui renvoie -1 si elle ne trouve pas la chaîne cherchée (une chaîne ne contenant que 2 espaces ici). Lorsque vous aurez éliminé toutes les duplication d'espace, vous devrez encore éliminer le caractère espace qui sera encore présent à la fin de votre nouvelle chaîne (par exemple en utilisant la fonction `<strip()>` sur votre nouvelle chaîne de caractères).**