Python

I. Instructions élémentaires 1

Les exercices de cette section sont à effectuer sans utiliser Python, seulement sur feuille.

Exercice 1

1. Pour chacune des expressions données plus bas, indiquer la valeur du résultat obtenu ainsi que son type (`int` (entier), `float` (flottant), `str` (chaîne de caractère) ou `bool` (booléen)).

   (a) `2*5`
   (b) `2.0*4`
   (c) `4/3`
   (d) `4//3`
   (e) `2 == 5`
   (f) `"3"*2`
   (g) `str(4) == 4`
   (h) `15%4`
   (i) `2**3`

2. On considère ici qu'initialement, `A=10, B=5, C=2` et `D=4`.
   Indiquer pour chaque cas ci-dessous s'il s'agit d'une instruction ou d'une expression, voire d'un code non valide. Expliquer ce qui se produit lorsque le code est valide.

   (a) `A=10, B=5`
   (b) `A,B=10,5`
   (c) `3 * A + 5 * B`
   (d) `3 A - 2 B`
   (e) `A / B`
   (f) `A // B`
   (g) `A += 2`
   (h) `A = 10`
   (i) `A == 12`
   (j) `B == A // 2`
   (k) `C ** B`
   (l) `B % D`

Exercice 2 (Vrai/Faux)
Pour chacune des affirmations suivantes, dire si elle est vraie ou fausse. Justifier si possible la réponse.

1. Après les instructions :

```
x=3
y=5
x=y
y=x
```

   la valeur de x est 5 et celle de y est 3.

2. Après les instructions :

```
x=3
y=5
y==x
x=y
```

   la valeur de x est 5 et celle de y est 5.

3. Une instruction de la forme `if ...` est une boucle conditionnelle.

4. Avec `for i in range(10):`, la variable `i` prend 9 valeurs puisque la dernière est 9.

Exercice 3 (QCM)

Pour chaque question, une seule réponse parmi celles proposées est exacte.

1. Parmi les propositions suivantes, laquelle n'est pas une expression ?

   (a) a<b (b) a !=b (c) a=b (d) a>=b

2. On considère les instructions suivantes :

```
a=8
b=5
a==b+1
b=b+1
a==b+1
b=b+1
print(a==b+1)
```

   Quel est le résultat affiché ?

   (a) 8 (b) une erreur (c) False (d) True

3. Combien de fois la fonction `print` est-elle appelée dans le code en Python suivant ?

```
n=4
for i in range(2,n):
    print(i)
```

   (a) Jamais (b) une fois (c) deux fois (d) trois fois

4. Quelle est la valeur finale de x après l'exécution du code Python suivant ?

```
x=1
for i in range(4):
    x=x+i
```

   (a) 6 (b) 7 (c) 10 (d) 11

5. Quelles sont les valeurs finales de x et y après l'exécution de ce code ?

```
x=4
while x>0:
    y=0
    while y<x:
        y=y+1
        x=x-1
```

   (a) x=-1, y=0 (b) x=0, y=0 (c) x=0, y=1 (d) boucle infinie

Exercice 4

Dans chacun des deux cas, quelle est la valeur de `x` à la fin de l'exécution du code ?

```
x=1
n=1
while n>1:
    x=x+n
    n=n-1
```

```
x=0
for i in range(2):
    x=x*i
    for j in range(3):
        x=x+j
```

Exercice 5
Dans chacun des cas suivants, indiquer l'affichage obtenu lors de l'exécution du code.

1.
```
for n in range (10):
  print(n)
```

2.
```
for n in range(2,7):
  print(n)
```

3.
```
for n in range(100,110,2):
  print(n)
```

Exercice 6
Que vaut `f` à la fin de l'exécution des d'instructions suivantes ?

```
n=5
f=0
i=1
while i<n+1:
  f=f+i
  i=i+1
```

Exercice 7
Que fait cette suite d'instructions ?

```
s=0
for i in range(5) :
  x=int(input('Entrez un nombre : '))
  s=s+x
```

II. Instructions élémentaires 2

À partir d'ici, les codes sont à écrire et tester en Python, sauf indication contraire.

Exercice 8
Pour chacun des cas suivants, donner une expression Python qui calcule le résultat souhaité.

1. Quel nombre de secondes y a-t-il dans un jour (24 heures) ?
2. Amel, Benoît et Célia ont acheté 4^3 biscuits. On souhaite les répartir équitablement.
   (a) Combien chacun en aura ?
   (b) Combien en reste-t-il ?
3. Écrire une expression booléenne, utilisant la variable `x`, les comparaisons et les opérations booléennes permettant de vérifier que :
   (a) $x \in [1;9]$
   (b) $x \in]-\infty;0] \cup]2;3]$
4. Soit `n` un entier. On souhaite vérifier si `n` est divisible par `3` sans qu'il ne vaille `0`.
   Écrire une expression booléenne correspondant à cette vérification.

Exercice 9
Écrire du code Python qui affecte à `nb_pommes` un entier demandé à l'utilisateur en utilisant la fonction `input`. Écrire ensuite le code qui, si le nombre de pommes est pair, affiche `"divisible par 2"`. Si ce n'est pas le cas, le code doit afficher `"non divisible par 2"`. Il s'agit ici d'utiliser les instructions `if` et `else`.

Exercice 10
Écrire du code Python qui demande un nombre (flottant) `moyenne` puis, en fonction de sa valeur, affiche la mention au bac. Il s'agit ici d'utiliser les instructions `if`, `elif` et `else`.

Exercice 11
Une séance de cinéma est interdite aux moins de 16 ans. Le prix du billet varie avec l'âge : les seniors (à partir de 65 ans) et les mineurs (moins de 18 ans) paient un tarif réduit, les autres un tarif plein. Écrire un programme qui demande à un utilisateur de saisir son âge et qui lui donne une réponse quant à son autorisation de voir le film ainsi que le tarif du billet.
Exemple : si l'utilisateur saisit 22 , le programme doit afficher « Vous pouvez-voir ce film, le billet est au tarif plein. »

Exercice 12
Une année est bissextile si elle est divisible par 4 mais non divisible par 100. Les années divisibles par 400 sont également bissextiles.
Écrire un programme qui demande à l'utilisateur de saisir une année et qui affiche un message pour préciser si cette année est bissextile ou non.

Exercice 13
Écrire un programme qui fait faire 10 tours de manège en affichant un message à chaque tour :

```
C'est le tour n°1.
C'est le tour n°2.
...
C'est le tour n°10.
```

Aide : on peut utiliser l'argument `sep=""` dans la fonction `print` si on l'utilise avec plusieurs arguments pour ne pas avoir d'espace autour du numéro du tour.
Il y a plusieurs manières de faire.

Exercice 14
Écrire un programme qui affiche tous les nombres entre 1 et 10, et indique pour chacun si celui-ci est pair ou impair :

```
1 est impair
2 est pair
...
10 est pair
```

Exercice 15
Robert ajoute 50€ à sa cagnotte tous les mois jusqu'à atteindre 1200€. Écrire une boucle Python `while` qui calcule le nombre de mois nécessaires pour qu'il obtienne assez d'argent. Penser à définir une variable `cagnote`, initialisée avant la boucle, puis qui évolue à chaque itération comme chaque mois, de même qu'une variable `compteur` qui compte le nombre de mois passés.

Exercice 16
Le haricot magique de Jack double de hauteur tous les jours. Il mesure 1 cm le premier jour. Écrire une boucle `while` qui calcule le nombre de jours nécessaires pour qu'il atteigne 1 km de hauteur.

Exercice 17
Écrire un programme qui demande à l'utilisateur (par le biais d'un `input`) un mot de passe et l'enregistre dans une variable `mdp`.
Le programme doit ensuite demander de fournir à nouveau le même mot de passe, et le redemander tant que celui qui est donné n'est pas le bon (il faut donc comparer le mot de passé donné au début, `mdp`, avec la nouvelle chaîne de caractère fournie par l'utilisateur).
Une fois le bon mot de passe donné, le programme affiche « Vous êtes connecté. » et s'arrête.

Exercice 18
Écrire un code qui affiche, à l'aide d'une boucle `for`, uniquement les nombres pairs allant de `0` à `50`.
Proposer deux codes différents, un premier utilisant une instruction `if`, le second utilisant astucieusement la fonction `range`.

Exercice 19
Écrire une boucle `for` qui calcule la somme des entiers impairs plus petits que `100`, puis afficher le résultat.
Aide : définir une variable `somme` (bien choisir sa valeur initiale) à laquelle ajouter chaque entier impair obtenu dans une boucle `for`.

Exercice 20
Écrire une boucle `for` qui calcule le produit des entiers dont le reste est `2` dans la division par `3` et qui sont inférieurs à `200`.

Exercice 21
Le programme suivant affiche la table de multiplication par `5` :

```
for x in range(3):
    print('5 x', x, '=', 5 * x)
```

1. Modifier le code pour qu'il affiche la table complète (de 5×0 à 5×10).
2. Modifier le code pour qu'il demande un nombre entier puis affiche la table complète de multiplication de ce nombre (et pas forcément de 5).
3. Écrire un nouveau programme qui affiche toutes les tables, successivement, de celle de 1 à celle de 10.

   Penser à ajouter une ligne de séparation entre les tables (un `print()` sans paramètre permet d'aller à la ligne).

Exercice 22
Écrire un code qui affiche, à l'aide d'une boucle `for`, les nombres de `0` à `99`, à raison de `10` nombres par ligne :

```
0 1 2 3 4 5 6 7 8 9
10 11 12 13...
...
90 91 92 93 94 95 96 97 98 99
```

Aide : penser à utiliser éventuellement les paramètres `sep` et/ou `end` de la fonction `print`.

III. Fonctions

Exercice 23 (QCM – sans utiliser Python)
Voici ci-contre une fonction définie en Python.
que retourne la fonction f si le paramètre x vaut 15 ?

```
def f(x):
    for d in range(2,x):
        if x%d == 0:
            return d
```

1. 3
2. 5
3. 3 et 5
4. 3, 5 et 15

 Pour tous les exercices qui suivent, penser à tester les fonctions définies en les exécutant et en affichant les résultats obtenus.

Exercice 24
Écrire une fonction qui prend en paramètre deux nombres et retourne le plus grand des deux.

Exercice 25
Écrire une fonction qui prend en paramètre trois nombres et retourne le plus grand des trois nombres.

Exercice 26

1. Écrire une fonction `SommeEntiers` qui prend en paramètre un entier `n` et qui retourne la somme des `n` premier entiers naturels non nuls $(1 + 2 + 3 + \cdots + n)$.
2. Écrire une fonction `SommeCarres` qui prend en paramètre un entier strictement positif `n` et qui retourne la somme des `n` premier carrés non nuls $(1^2 + 2^2 + 3^2 + \cdots + n^2)$.
3. Définir une fonction `SommeInverses` qui prend comme argument un entier n et qui retourne la somme des inverses des entiers entre 1 et n.
4. Écrire une fonction `SommeDiviseurs` qui prend en paramètre un entier naturel non nul et retourne la somme de ses diviseurs.

Exercice 27

1. Écrire une fonction `Prime` qui prend en paramètre un entier naturel non nul et retourne `True` si ce nombre est premier et `False` sinon.
2. Écrire une fonction `sumPrime(n)` qui retourne la somme des nombres premiers compris entre 1 et n (inclus).

Exercice 28
En utilisant la fonction `randint` du module `random`, écrire une fonction `Jeu` qui prend en paramètre un nombre entier `n` strictement positif, simule `n` fois le tirage d'un dé cubique équilibré (donc une valeur au hasard parmi les nombres 1,2,3,4,5 et 6), puis retourne le pourcentage de 6 obtenus.

IV. Fonctions – autres exercices

Exercice 29
Définir une fonction qui prend en argument un nombre entier `n` et qui :

- si `n` est pair, le divise par 2 ;
- si `n` est impair, le multiplie par 3 et lui ajoute 1.

puis qui retourne la nouvelle valeur de `n`.

Exercice 30
On considère une fonction f dont l'expression est $f(x) = -2x$ si $x < 0$ et $f(x) = x^2$ sinon.
Définir une fonction qui prend en argument un nombre flottant x et qui retourne l'image $f(x)$.

Exercice 31
Même chose que l'exercice précédent, mais avec une fonction g définie en trois morceaux :

$$g(x) = \begin{cases} x^3 + 2 \text{ si } x < 0 \\ x^2 + 2 \text{ si } 0 \leqslant x < 3 \\ 14 - x \text{ si } x \geqslant 3 \end{cases}$$

Exercice 32
Définir une fonction `factorielle` qui prend comme argument un entier `n` et qui retourne le produit des entiers de `1` à `n`.

Exercice 33
Définir une fonction qui prend comme un argument un entier `n` et qui retourne le nombre d'entiers i entre `-n` et `n` (compris) pour lesquels $f(i) \leqslant g(i)$ (fonctions définies dans les exercices précédents).

Exercice 34
Définir une fonction qui prend comme argument un entier n, qui demande n nombres (flottants) et retourne la moyenne de ces nombres.

Exercice 35
Définir une fonction qui prend pour argument un nombre flottant $A > 0$ et qui retourne le plus petit entier n tel que $n^3 + 2n + 1 \geqslant A$.

Exercice 36
Définir une fonction `SommeDuree` qui prend pour argument quatre nombres entiers $m1$,$s1$,$m2$,$s2$ représentant deux durées en minutes et secondes, et qui retourne la somme des deux durées en minutes et secondes sous forme de chaîne de caractères.

Exercice 37
Définir une fonction sans argument qui demande deux nombres (flottants) et un symbole d'opération (`"+"`, `"-"`, `"/"` ou `"*"`) puis qui effectue et retourne le résultat du calcul à effectuer.

Exercice 38
Un nombre parfait est un nombre n dont la somme des diviseurs propres (c'est à dire strictement inférieurs à n) est égal à n. Définir une fonction qui affiche les entiers n parfaits situés entre 2 et 1000.
Pour cela on pourra réutiliser la fonction `SommeDiviseurs` vue dans un exercice précédent.

Exercice 39
Deux nombres entiers M et N sont amicaux si la somme des diviseurs propres de M est égal à N et si la somme des diviseurs propres de N est égale à M.
Définir une fonction qui prend en argument un nombre entier `maxi` et qui affiche les couples de nombres amicaux inférieurs à `maxi`.
On pourra tester la fonction avec `maxi=3000`.

Exercice 40
Écrire une fonction qui convertit un nombre binaire en nombre décimal.
On pourra considérer deux manière différentes de donner le nombre binaire :

1. Sous forme de chaîne de caractère
2. Sous forme d'un entier écrit seulement avec des 0 et des 1.

Exercice 41
Écrire une fonction qui prend en paramètre un entier n et affiche, pour i allant de 1 à n, i étoiles sur la ligne numéro i.
Par exemple, pour n = 5, afficher :

```
*
* *
* * *
* * * *
* * * * *
```

Exercice 42

1. Écrire une fonction qui prend en paramètre un entier positif n et affiche un carré d'étoiles plein de côté n.
   Par exemple, si n vaut 4, votre fonction affichera :

   ```
   ****
   ****
   ****
   ****
   ```

2. Modifier votre fonction pour qu'elle affiche un carré creux.
   Par exemple pour n valant 4,

   ```
   ****
   *  *
   *  *
   ****
   ```

Exercice 43
Écrire une fonction qui prend en entrée un entier n et affiche une pyramide d'étoiles de hauteur n. Si n est négatif, la pyramide devra être inversée.
Par exemple, sur entrée n = 4, afficher :

```
   *
  * *
 * * *
* * * *
```

Sur entrée n = -3, afficher :

```
* * *
 * *
  *
```

Exercice 44
Écrire une fonction `power(x, n)` qui retourne la valeur de x^n, bien sûr sans utiliser l'opérateur `**`.

Exercice 45
définir une fonction qui prend pour paramètres trois entiers `n`, `inf` et `sup`, et qui affiche la table de multiplication de `n` entre `inf` et `sup`. Par exemple, l'exécution de la fonction avec les paramètres valant respectivement `2`, `3` et `5` affichera :

```
2*3=6
2*4=8
2*5=10
```

Penser à vérifier que `inf` est bien inférieur à `sup` et, si ce n'est pas le cas, échanger les valeurs.

Exercice 46
Soit f la fonction mathématique définie par $f(x) = x^3 - 3x + 2$.
Définir une fonction `tabulation` qui prend en paramètre trois nombres `inf`, `sup` et `pas` de type `float` et qui affiche les images des valeurs de `inf` à `sup` avec un pas de `pas`, sous une forme compréhensible (comme `f(1)=0`).
Même chose que pour l'exercice précédent pour ce qui est de `inf` et `sup`.

Exercice 47
On donne ci-dessous deux manières d'obtenir le PGCD (plus grand diviseur commun) de deux nombres entiers positifs A et B. Définir, suivant chacune de ces manières, une fonction qui prend en argument les valeurs A et B et qui retourne leur PGCD.

1. L'algorithme d'Euclide utilise le reste de la division entière de A par B :
   - soit Q le quotient de la division de A par B et R le reste.
   - si R=0 alors le PGCD est B
   - sinon on remplace A par B , B par R et on recommence.
2. On a propriété suivante :
   - PGCD(A,B)=PGCD(A-B,B) si A>B
   - PGCD(A,B)=PGCD(A,B-A) si A<B
   - PGCD(A,B)=A si A=B.

V. Listes

Exercice 48
Partie 1
Écrire les instructions Python permettant de réaliser successivement les actions suivantes :

1. Définir la liste `duck` qui contient uniquement la chaîne de caractère `"Donald"`.
2. Ajouter à la liste `duck` l'élément `"Donna"` en utilisant la méthode `append`.
3. Définir la liste `neveux` contenant les chaînes de caractères `"Riri"`, `"Fifi"` et `"Loulou"`.
4. Ajouter à la liste `duck` les éléments de la liste `neveux` en utilisant la **concaténation**.
5. Afficher le premier éléments de la liste `duck` (on doit obtenir `"Donald"`).
6. Afficher l'avant-dernier éléments de la liste `duck` (on doit obtenir `"Fifi"`).
   Obtenir cet affichage deux fois, en utilisant deux valeurs d'indices différentes.
7. Modifier l'élément d'indice `1` de la liste `duck` pour qu'il devienne `"Daisy"`.
8. Afficher la liste `duck`. On doit obtenir `["Donald","Daisy","Riri","Fifi","Loulou"]`
9. Afficher la longueur de la liste (on doit obtenir `5`)
10. Parcourir la liste `duck` **par valeurs** et afficher ses éléments.
11. Parcourir la liste `duck` **par indices** et afficher ses éléments (l'affichage doit donc être le même qu'à la question précédente).
12. Parcourir la liste `neveux` **par couples indices, valeurs** et afficher des chaînes de caractères de la forme :

```
Neveu 1 : Riri
Neveu 2 : Fifi
Neveu 3 : Loulou
```

Partie 2
Exécuter les instructions suivantes ligne après ligne dans un **interpréteur** Python. Observer à chaque étape ce qui est éventuellement retourné. Dans les cas où rien n'est retourné, ou si ce qui est retourné n'est pas une liste, refaire afficher le contenu de la liste (en entrant simplement `s`). Indiquer alors ce que font les instructions.
À la fin, la liste doit contenir l'ensemble des jours de la semaine, mais triés par ordre alphabétique :

```
['dimanche', 'jeudi', 'lundi', 'mardi', 'mercredi', 'samedi', 'vendredi']
```

 En cas d'erreur, il faut recommencer à zéro car la liste sera modifiée au fur et à mesure.

```
s=['Lune',2.0,'mardi',230,12,True,'samedi','a']
s
s[0]
s[-3]
s[1:]
s[:2]
s[1:3]
s[0:4:2]
s[:]
len(s)
del(s[1])
s+['jeudi','vendredi']
s
```

```
s*2
s[2]='mercredi'
s.insert(4,None)
s[3:5]=['jeudi','vendredi']
s.pop(-3)
s.pop()
s.append('dimanche')
s.remove('Lune')
s=['lundi']+s
s.index('jeudi')
s.reverse()
s.sort()
s
```

Partie 3

Écrire des lignes de code dans un fichier Python qui effectuent les actions suivantes successivement. Pour les fonctions et méthodes spécifiques, chercher dans la librairie Python sur les listes.

1. Définir la liste `l` contenant les éléments suivants dans l'ordre :
   45,17,89,38,10 et 74.
2. Trier `l` (utilisation de la méthode `sort` : `l.sort()`) puis l'afficher.
3. Ajouter l'élément « `12` » puis afficher la liste.
4. Renverser la liste puis l'afficher.
5. Afficher l'indice de l'élément « `10` ».
6. Enlever l'élément « `38` » et afficher la liste.
7. Afficher la sous-liste du 2e au 3e élément.
8. Afficher la sous-liste du début au 2e élément.
9. Afficher la sous-liste du 3e élément au dernier.
10. Afficher le deuxième élément en partant de la fin.

Exercice 49

Créer dans chaque cas la liste respectant la définition donnée, si possible de plusieurs manières différentes (avec la méthode `append`, ou par compréhension, ou encore autrement, comme par exemple en utilisant la syntaxe `list(range(...))`.

1. La liste contenant 100 zéros (sans écrire ces 100 zéros, y compris en copiant/collant).
2. La liste contenant les nombres pairs de 0 à 50.
   Rappel : un nombre est pair s'il est divisible par 2, autrement dit si le reste de la division de ce nombre par 2 vaut 0.
3. La liste contenant les multiples de 3 (divisibles par 3) parmi les nombres allant de 1 à 100.
   Aide : le principe est similaire à la question précédente, si ce n'est que l'on divise par 3 au lieu de 2.
4. Une liste de 100 nombres tirés aléatoirement entre 1 et 6.
   Pour cela, utiliser la fonction `randint(a,b)` du module `random` qui permet d'obtenir des valeurs aléatoires entières situées entre `a` et `b` inclus.
5. La liste contenant les 15 premiers carrés parfaits (`[1,4,9,16,...]`).
6. La liste contenant les carrés des nombres de 20 à 40.
7. La liste de 20 nombres alternant entre 0 et 1 (`[1,0,1,0,...]`).

Exercice 50

1. Indiquer l'affichage obtenu lors de l'exécution du code suivant :

```
voyelles=['a','e','i','o','u','y']
for voyelle in voyelles:
  print(voyelle)
```

2. Faire un autre code qui effectue la même chose mais à l'aide d'une boucle de cette forme :

```
for i in range(len(voyelles)):
  ...
```

Exercice 51

On peut obtenir la liste des lettres en majuscule de l'alphabet avec la définition suivante :
`majuscules=[chr(ord('A')+i) for i in range(26)]`
La fonction `chr` permet d'obtenir un caractère à partir de son codage numérique.
La fonction `ord` permet d'obtenir le codage numérique d'un caractère.

1. De quelle manière a été définie la liste `majuscules` ?
2. Définir d'une autre manière la liste `minuscules` des lettres minuscules de l'alphabet.

3. En utilisant une boucle `for` et les deux listes `majuscules` et `minuscules`, écrire la comptine suivante :

```
La majuscule de a est A
La majuscule de b est B
La majuscule de c est C
...
La majuscule de z est Z
```

Exercice 52
Définir une liste contenant les valeurs 2, 3, 5, 7, 11, 13, 17 et 19, puis écrire un code Python qui permette d'afficher les 15 premiers termes des tables de multiplication des nombres de cette liste.
On doit obtenir quelque chose comme :

```
2  4  6  8  10  12  14  16  18  20  22  24  26  28  30
3  6  9  12  15  18  21  24  27  30  33  36  39  42  45
5  10  15  20  25  30  35  40  45  50  55  60  65  70  75
...
19                    ...                   285
```

Exercice 53
Soit la liste suivante :
`['Yun-Sung','Mark','Charline','Aissatou','Adam','Jérôme','Bouchra']`.
Écrire un code Python qui affiche chacun de ces prénoms avec le nombre de caractères correspondant.
Indication : la fonction `len` s'applique aussi aux chaînes de caractères.

Exercice 54
Soit la liste d'entiers `maliste` définie de la manière suivante :

```
maliste = list(range(1,29,3))
```

1. Quelles sont les valeurs contenues dans `maliste` ? Répondre avant de vérifier en l'affichant.
2. Écrire un code Python qui modifie les éléments pairs de `maliste` en les divisant par 2.
   Aide : parcourir la liste par indices (éventuellement par indices et valeurs) ; pour chaque élément d'indice `i`, tester s'il est pair, et si c'est le cas, remplacer l'élément d'indice `i` par sa valeur actuelle divisée par 2.

Exercice 55
Écrire un code Python capable d'afficher la liste de tous les jours d'une année imaginaire, laquelle commencerait un jeudi. Le code utilisera lui-même trois listes : une liste des noms de jours de la semaine, une liste des noms des mois, et une liste des nombres de jours que comportent chacun des mois (ne pas tenir compte des années bissextiles). Exemple de sortie :

```
jeudi 1 janvier
vendredi 2 janvier
samedi 3 janvier
dimanche 4 janvier
# etc., jusqu'au jeudi 31 décembre.
```

Exercice 56
Écrire un code Python qui crée automatiquement la liste valeurs des sinus des angles de 0° à 90°, par pas de 5°.
Pour utiliser la fonction `sin()`, il faut importer le module `math` en ajoutant la ligne de code :

```
from math import sin
```

⚠ la fonction `sin()` considère que les angles fournis en argument sont en radians (en sachant que $360° = 2\pi$ radians), et que les deux unités sont proportionnelles.

⚠ Noter que l'on ne veut pas afficher les valeurs, mais créer la liste contenant ces valeurs.

Exercice 57
Écrire un code Python qui recherche le mot le plus long dans une phrase donnée.
À l'exécution, l'utilisateur doit pouvoir entrer une phrase de son choix, dont on supposera qu'elle ne contient pas de caractères spéciaux, en particulier pas de point ni de virgule (seulement des mots séparés par des espaces).
Indication : l'instruction `chaine.split(" ")`, et même plus simplement `chaine.split()`, transforme une chaîne de caractères `chaine` en liste en prenant comme séparateur la chaîne `" "` (espace). Autrement dit, avec une telle instruction on obtient une liste de mots à partir d'une chaîne de caractères contenant des mots séparés par des espaces.

VI. Listes et fonctions

Les exercices marqués d'une étoile (*) sont des exercices à savoir faire et refaire car ils peuvent être demandés lors d'une épreuve de bac en terminale, sous cette forme ou sous une forme modifiée.
Si il en existe, il n'est pas autorisé d'utiliser de fonction Python prédéfinie permettant de répondre directement aux problèmes.
Penser à exécuter au minimum les exemples donnés, et éventuellement d'autres, pour vérifier les résultats retournés par les fonctions.

Exercice 58 (*)
Écrire une fonction `appartient(liste,x)` qui prend pour argument une liste et une valeur `x` et qui retourne `True` si la liste contient la valeur `x`, et qui retourne `False` si ce n'est pas le cas.

```
>>> appartient([3,1,7,5,9],5)
True
>>> appartient([3,1,7,5,9],2)
False
```

Exercice 59 (*)
Écrire une fonction `occurrences(liste,x)` qui prend pour argument une liste et une valeur `x` et qui retourne le nombre d'occurrences (autrement dit d'apparitions) de `x` dans la liste.

```
>>> occurrences([3,5,7,5,9],5)
2
>>> occurrences([3,5,7,5,9],2)
0
```

Aide : définir une variable servant de compteur, qui augmentera à chaque valeur lue dans la liste parcourue.

Exercice 60 (*)
Écrire une fonction `moyenne_simple` qui prend en argument une liste de nombres et qui retourne la moyenne de cette liste.

```
>>> moyenne_simple([3,5,7,5,9])
5.8
>>> (3+5+7+5+9)/5
5.8
```

Aide : définir une variable à laquelle on ajoute successivement chaque valeur de la liste parcourue.

Exercice 61 (*)
Une série statistique est une liste de couples de valeurs `(x,n)` représentant une valeur et un effectif. Écrire une fonction `moyenne_ponderee(serie)` qui prend en argument une série statistique et qui retourne la moyenne pondérée de cette série.

```
>>> moyenne_ponderee([(12.5,1),(13,2),(7,4)])
9.5
>>> (12.5*1+13*2+7*4)/(1+2+4)
9.5
```

Aide : si les éléments d'une liste sont tous sous la forme d'un couple, on peut la parcourir en donnant un nom de variable à chacun des éléments de ces couples, de la manière suivante (similaire à ce que l'on fait lors d'un parcours par couples indices, valeurs) :

```
for x,n in serie:
    ...
```

D'autre part, il faut ici deux variables : une qui contient la somme des produits des valeurs par les effectifs, et une autre qui contient la somme des effectifs.

Exercice 62 (*)
Écrire une fonction `maximum` qui prend en argument une liste de nombres (considérée non vide) et retourne le maximum de la liste.

```
>>> maximum([3,5,7,5,9,1])
9
```

 la liste peut contenir des valeurs quelconques, y compris (seulement) négatives.
Aide : l'idée est d'utiliser une variable prenant pour valeur le maximum trouvé lors du parcours de la liste et qui change à chaque fois que l'on trouve une valeur supérieure.

Exercice 63 (*)
Écrire une fonction `liste_indices(liste,x)` qui prend en argument une liste de nombres et un nombre `x` et retourne la liste des indices de tous les éléments égaux à `x`, éventuellement vide si il n'y a aucune occurrence de `x` dans `liste`.

```
>>> liste_indices([3,5,9,5,9,1],5)
[1, 3]
>>> liste_indices([3,5,9,5,9,1],7)
[]
```

Exercice 64 (*)
Écrire une fonction `maxi_indices` qui prend en argument une liste de nombres et retourne le couple formé par le maximum de la liste et la liste des indices des éléments de la liste égaux à ce maximum. Si la liste est vide, le couple retourné doit être `(None,[])`.

```
>>> maxi_indices([3,5,9,5,9,1])
(9, [2, 4])
>>> maxi_indices([])
(None, [])
```

Exercice 65 (*)
Écrire une fonction `est_croissante` qui prend en argument une liste de nombres quelconques et qui retourne `True` si les valeurs de la liste sont rangées par ordre croissant, `False` sinon.
On considérera qu'une liste ayant au plus un élément est croissante.

```
>>> est_croissante([1, 2, 3])
True
>>> est_croissante([1, 3, 2])
False
```

Exercice 66
Écrire une fonction `trie_sans_doublon` qui prend en argument une liste de nombres quelconques, certains d'entre eux étant éventuellement présents en plusieurs exemplaires, et qui retourne une copie triée de cette liste dans laquelle il n'y a pas de doublon (on pourra utiliser la fonction `sorted(l)` qui permet d'obtenir une liste triée à partir d'une liste `l`).

```
>>> trie_sans_doublon([3,5,9,5,9,1])
[1, 3, 5, 9]
```

Exercice 67
Écrire une fonction `progression(a, b, n)`, où `a`, `b` et `n` sont trois entiers, et qui retourne une liste de taille `n` contenant les entiers `a`, `a + b`, `a + 2b`, `a + 3b`, ..., `a + (n-1)b`.
On pourra donner au moins deux manières différentes de définir cette fonction.

```
>>> progression(2, 7, 5)
[2, 9, 16, 23, 30]
```

Exercice 68
Définir une fonction `mediane` qui prend en argument une liste (non triée) et qui retourne la médiane de cette liste.
Rappel : pour obtenir la médiane d'une liste de valeurs, on doit trier la liste (on peut obtenir une liste triée en Python avec la fonction `sorted`), puis :

- Si le nombre d'éléments de la liste est impair, la médiane est la valeur située au milieu de la liste triée ;
- Sinon, la médiane est la moyenne des deux valeurs situées au milieu.

```
>>> mediane([3,1,7,5,9])
5
>>> mediane([3,1,7,5])
4.0
```

Aide : l'indice du milieu d'une liste de longueur impaire `n` est `n//2`. Prendre au moins un exemple pour voir ce que cela donne lorsque `n` est pair.

Exercice 69
Écrire une fonction `interlace(l1, l2)` qui prend en entrée deux listes `l1` et `l2` de même longueur et retourne une liste de longueur double qui contient les valeurs des listes de façon entrelacée, c'est-à-dire :
`[l1[0], l2[0], l1[1], l2[1], ..., l1[n-1], l2[n-1]]` (où `n` est la longueur des deux listes).

```
>>> interlace([0, 1, 6],[2, 4, 7])
[0, 2, 1, 4, 6, 7]
```

Il y a beaucoup de manières différentes de réaliser cela.

Exercice 70

1. Définir une fonction Python prenant comme argument un entier naturel n et qui renvoie une liste de la forme : `[1,1,1,2,1,3,1,4,1,5,1,6,...,1,n]`.
2. Définir une fonction Python prenant comme argument un entier naturel n et qui construit une liste de la forme : `[1,1,2,1,2,3,1,2,3,4,...,1,2,3,..., n]`.

Exercice 71

Créer une fonction `decale_droite` qui prend en argument une liste et renvoie une nouvelle liste contenant les mêmes éléments mais à des positions décalées d'un rang vers la droite (la dernière devenant la première), comme dans l'exemple suivant :

```
>>> decale_droite([1,2,3,4])
[4, 1, 2, 3]
```

Exercice 72

Écrire une fonction `valeurs_sup` qui retourne les éléments d'une liste qui sont supérieurs ou égaux à une valeur donnée, à l'image de l'exemple suivant :

```
>>> valeurs_sup([2,1,3],1.5)
[2, 3]
```

Exercice 73

Écrire une fonction `permute(liste,i,j)` prend en argument une liste et deux valeurs d'indices `i` et `j` et qui retourne une liste dont les éléments sont les mêmes, mais avec les éléments d'indices `i` et `j` permutés, à l'image de l'exemple suivant :

```
>>> permute([1,2,3,4,5],1,3)
[1, 4, 3, 2, 5]
```

Exercice 74

Écrire une fonction `renverse` qui retourne une liste dont les éléments sont dans l'ordre inverse de la liste donnée en argument, à l'image de l'exemple suivant :

```
>>> renverse([2,1,3])
[3, 1, 2]
```

Remarque : le code `liste[::-1]` permet d'obtenir cela, mais le but est de l'obtenir avec quelques lignes de code tout de même. Plusieurs manières sont possibles.

Exercice 75 (Facultatif)

Définir une fonction `valeurs_communes(l1,l2)` qui prend en argument deux listes et qui retourne la liste des valeurs communes de `l1` et `l2`.
Les valeurs de cette liste devront être uniques et dans le même ordre que dans `l1`.

```
>>> valeurs_communes([1, 7, 9, 5, 6, 7, 5],[7, 3, 2, 6, 2, 4])
[7, 6]
```

Exercice 76 (Facultatif)

Écrire une fonction `plus_longue_suite_croissante` qui prend en paramètre une liste de nombres et renvoie la plus longue suite croissante de nombre consécutifs qu'elle contient. S'il existe plusieurs sous suites croissantes de longueur maximale, la première sera renvoyée.
Exemples :

```
>>> plus_longue_suite_croissante([1, 5, 2, 3, 4, 0, 6])
[2, 3, 4]
>>> plus_longue_suite_croissante([1, 5, 7, 2, 3, 4, 0])
[1, 5, 7]
>>> plus_longue_suite_croissante([5, 4, 3, 2, 1])
[]
```

Exercice 77 (Facultatif – avec des chaînes de caractère)
Les chaînes de caractère ne sont pas exactement des listes, mais on peut accéder à leurs éléments (les caractères) de la même manière que pour les listes, avec les indices.
Il est également possible de concaténer des chaînes de caractères comme on le fait pour des listes.

1. Écrire une fonction `DebutFin` qui prend en argument un mot (de type `str`) et retourne `True` si le mot commence et se termine par la même lettre et `False` sinon.

```
>>> DebutFin("ABBA")
True
>>> DebutFin("Fin")
False
```

2. Écrire une fonction qui prend en argument deux mots (de type `str`) et retourne `True` si les deux mots commencent par la même lettre et se terminent par la même lettre et `False` sinon.

```
>>> MemeDebutFin("infinis","imitais")
True
>>> MemeDebutFin("Carreau","Coeur")
False
```

3. Écrire une fonction `Double` qui prend en argument un mot (de type `str`) et retourne le mot obtenu en doublant chaque lettre.

```
>>> Double("bon")
'bboonn'
```