Exercice 1. ★☆☆ **Années bissextiles**
Soit `A` un entier naturel. Écrire un programme qui permet de savoir si l'année `A` est bissextile ou non.
On rappelle qu'une année est bissextile si elle est soit divisible par 4 mais non divisible par 100, soit divisible par 400.

Exercice 2. ★☆☆
Soient a et b deux entiers naturels.

1. Écrire une boucle affichant et incrémentant (+1) la valeur de a tant qu'elle reste inférieure ou égale à celle de b.
2. Écrire une boucle décrémentant (-1) la valeur de b, affichant sa valeur si elle est impaire, jusqu'à ce que b soit nul.

Exercice 3. ☆☆☆ **Somme des puissances cinquièmes d'entiers naturels.**
Soit m un entier naturel non nul donné. Écrire un programme qui affiche la valeur de la somme

$$1^5 + 2^5 + \cdots + m^5.$$

Exercice 4. ★☆☆
Former la liste des nombres pairs entre $-n$ et n, où n est un entier relatif donné.

Exercice 5. ☆☆☆ **Table de multiplication.**

Affecter à `c` un chiffre et écrire un programme qui affiche la table de multiplication de `c`. Par exemple, si `c` est le chiffre 2, Python affichera les lignes ci-contre :

```
0*2=0
1*2=2
⋮
9*2=18
```

Exercice 6. ☆☆☆ **Premier terme de l'écriture binaire d'un entier naturel**
Écrire un programme qui, étant donné un entier naturel non nul n, affiche la plus grande valeur de l'entier naturel m tel que 2^m divise n.

Exercice 7. ☆☆☆ **Départements**
Écrire un programme qui, étant donné un numéro de départements de la région Auvergne-Rhône-Alpes, affiche le nom du département associé.
On prendra soin d'afficher un message d'erreur si le numéro n'est pas un numéro de département valide.

Exercice 8. ★☆☆ **Somme et produit des éléments d'une liste de réels**
Soit `L` une liste de réels. Écrire un programme qui calcule la somme des éléments de `L`. Faire de même avec le produit.

Exercice 9. ★☆☆ **Algorithme de Viete**
Traduire en langage Python l'algorithme (de Viète) suivant :

$a \leftarrow 2$
$n \leftarrow 1$
$c \leftarrow 0$
`Boucle :`
    $c \leftarrow \sqrt{2+c}$
    $b \leftarrow 2/c$
    $a \leftarrow ab$
`Fin Boucle dès que` a `est une valeur approchée à` 10^{-3} `près de` π `ou dès que` $n > 500$.

Exercice 10. ★★☆ **Algorithme de Brent-Salamin**

En 1975, Richard Brent et Eugene Salamin ont élaboré l'algorithme suivant, qui permet d'obtenir une suite qui converge rapidement vers π (cet algorithme a permis de réaliser plusieurs records de calcul des décimales de π).

On pose $a_0 = 1$, $b_0 = \frac{1}{\sqrt{2}}$, $t_0 = \frac{1}{4}$, $p_0 = 1$ *puis, pour tout* $n \in \mathbb{N}$,

$$a_{n+1} = \frac{a_n + b_n}{2},\ b_{n+1} = \sqrt{a_n b_n},\ t_{n+1} = t_n - p_n(a_n - a_{n+1})^2 \text{ et } p_{n+1} = 2p_n.$$

Lorsque a_n *et* b_n *sont « proches », la valeur de* $\frac{(a_n+b_n)^2}{4t_n}$ *est une valeur approchée de* π.

Implémenter cet algorithme en langage Python tant que $|a_n - b_n| > 10^{-10}$.
Indication : l'instruction `abs(a)` *renvoie la valeur absolue de* `a`*; après avoir écrit* `import math`*, on peut calculer la racine carrée d'un réel positif ou nul a grâce à l'instruction* `math.sqrt(a)`.

Solutions

Solution de l'ex 1. Voici une proposition :

```
if A%4==0 and A%100!=0:
    print('l'année',A,'est bissextile')
elif A%400==0:
    print('l'année',A,'est bissextile')
else:
    print('l'année',A,'n'est pas bissextile')
```

Solution de l'ex 2.

1. Une boucle affichant et incrémentant (+1) la valeur de a tant qu'elle reste inférieure ou égale à celle de b :

```
while a<=b:
    print(a)
    a+=1
```

2. Une boucle décrémentant (-1) la valeur de b, affichant sa valeur si elle est impaire, jusqu'à ce que b soit nul :

```
while b>=0:
    if b%2==1:
        print(b)
    b-=1
```

Solution de l'ex 3. Voici une possibilité d'algorithme.

```
S=0 # initialisation de la somme
for k in range(1,m+1):
    S+=k**5 # on incrémente de la somme
print(S)
```

Solution de l'ex 4. On imagine facilement le cas où `n` est positif ou nul ; on s'y ramène en posant `n=-n` dans le cas où `n` est strictement négatif (on aurait également pu utiliser la fonction `abs` prédéfinie dans Python, qui renvoie la valeur absolue d'un réel). On propose alors une première version avec la boucle `for` :

```
if n<0:
    n=-n
L=[] # Initialisation de la liste
for k in range(-n,n+1):
    if k%2==0:
        L+=[k] # Ajout de k dans L si k est pair
```

que l'on peut réécrire de manière raccourcie avec l'écriture en compréhension d'une liste :

```
if n<0:
    n=-n
L=[k for k in range(-n,n+1) if k%2==0]
```

Solution de l'ex 5. Voici un exemple d'algorithme :

```
c=4
for k in range(1,10):
    print(k,'*',c,'=',k*c)
```

Solution de l'ex 6. Ici, il paraît naturel d'utiliser la boucle `while` :

```
m=0
while n%2**m==0:
    m+=1
print(m-1)
```

Solution de l'ex 7. Du conditionnement, en veux-tu ? En voilà !

```
num=12 # Un exemple...
if num==1:
    print("Ain")
elif num==3:
    print("Allier")
elif num==7:
    print("Ardèche")
elif num==15:
    print("Cantal")
elif num==26:
    print("Drôme")
elif num==38:
    print("Isère")
elif num==42:
    print("Loire")
elif num==43:
    print("Haute-Loire")
elif num==69:
    print("Rhône")
elif num==73:
    print("Savoie")
elif num==74:
    print("Haute-Savoie")
else:
    print("Ce numéro n'est pas un numéro d'un département de la région
    Auvergne-Rhône-Alpes")
```

Solution de l'ex 8. On construit les deux algorithmes sur le même modèle :

```
S=0
for elem in L:
    S+=elem
```

```
P=1
for elem in L:
    P*=elem
```

Solution de l'ex 9. Traduction simple, en n'oubliant pas d'incrémenter le compteur `n` :

```
import math
a=2
n=1
c=0
while abs(math.pi-a)>10**(-3) or n<=500:
    c=math.sqrt(2+c)
    b=2/c
    a=a*b
    n+=1
```

Solution de l'ex 10. En utilisant l'affectation parallèle de Python, cet algorithme est en fait assez facile à écrire :

```
import math
a,b,t,p=1,1/math.sqrt(2),1/4,1
cpt=1
while abs(a-b)>10**(-10):
    aa=(a+b)/2
    a,b,t,p=aa,math.sqrt(a*b),t-p*(a-aa)**2,2*p
    cpt+=1
print(cpt,((a+b)**2)/(4*t))
```

Exercice 1. ☆☆☆ ❤ **Volume d'une boule**

1. Écrire une fonction `cube` qui renvoie le cube de son argument d'entrée (flottant).
2. Écrire une fonction, utilisant la fonction `cube`, qui renvoie le volume d'une boule de rayon donné.

Exercice 2. ★☆☆ **Norme d'un vecteur**

Écrire une fonction qui calcule la norme d'un vecteur ; on prendra en compte dans cette seule fonction les cas du plan et de l'espace.

Exercice 3. ☆☆☆ ❤ **Somme des éléments d'une liste de réels**

Écrire une fonction qui calcule la somme des éléments d'une liste de réels (entiers ou flottants) donnée. Faire de même pour le produit.

Exercice 4. ★☆☆ ❤ **Somme des premiers entiers naturels**

1. Écrire une fonction `somme_entiers` qui calcule à l'aide d'une boucle la somme des entiers de 1 à n, où n est un entier donné.
2. Écrire une fonction `somme_entiers2` qui fait de même au regard du cours de mathématiques.
3. Vérifier que les deux fonctions sont égales sur les 100 premiers entiers naturels non nuls.

Exercice 5. ★☆☆ ❤ **Binôme de Newton**

Dans tout l'exercice, on s'interdira d'utiliser la fonction puissance (**) de Python.

1. Écrire une fonction qui calcule la factorielle d'un entier naturel donné.
2. Écrire une fonction qui calcule $\binom{n}{k}$ étant données deux variables d'entrée n et k, entières naturelles.
3. Écrire une fonction qui renvoie le nombre a^n pour un flottant a et un entier naturel n donné.
4. Écrire une fonction `Newton1` qui, étant donnés deux flottants a et b et un entier naturel n, renvoie le nombre $\sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$.
5. Écrire une fonction `Newton2` qui renvoie $(a+b)^n$.
6. Écrire un script qui vérifie que les deux fonctions précédentes sont égales pour trois variables d'entrée données.
7. Faire afficher le triangle de Pascal jusqu'à une ligne donnée.

Exercice 6. ★★☆ ❤ **Conjecture de Syracuse**

On se donne un entier naturel non nul a. On pose alors $u_0 = a$ et

$$\forall n \in \mathbb{N}, \quad u_{n+1} = \begin{cases} 3u_n + 1 & \text{si } u_n \text{ est impair} \\ \frac{u_n}{2} & \text{sinon} \end{cases}$$

S'il existe un entier n tel que $u_n = 1$, alors la suite est périodique à partir de ce n (1421421421...). La conjecture, dite de Syracuse, prétend que, pour n'importe quelle valeur de a, il existe un entier N tel que $u_N = 1$.

1. Écrire une fonction `syracuse` qui, étant donné un entier `a`, renvoie l'entier `a/2` si `a` est pair, et l'entier `3a+1` sinon.
2. Écrire une fonction `test_syracuse` qui teste la conjecture de Syracuse pour un entier naturel donné.
3. Améliorer le dernier algorithme de manière à obtenir la valeur minimale de N telle que $u_N = 1$ (si elle existe...) pour une valeur de a donnée.

Exercice 7. ☆☆☆ **Liste des diviseurs positifs d'un entier naturel**
Écrire une fonction qui, étant donné un entier naturel n, renvoie la liste de ses diviseurs positifs.

Exercice 8. ★★★ ☠ **Liste des sous-listes d'une liste**
Écrire une fonction qui, étant donné une liste d'objets, renvoie la liste de toutes ses sous-listes.
Par exemple, appliquée à $[0, 2, 1]$, cette fonction renverra $[[\,], [0], [2], [1], [0, 2], [2, 1], [0, 2, 1]]$.

Exercice 9. ☆☆☆ ❤
Que fait cette fonction ?

```
def mystere(a,b):
    a=a+b
    b=a-b
    a=a-b
    return a,b
```

Exercice 10. ★★☆
Que fait cette fonction ?

```
def mystere(a,b):
    if a>=b>=0 or a<=b<=0:
        while abs(a)>=abs(b):
            a=a-b
    else:
        while a*b<0:
            a=a+b
    return a
```

Exercice 11. ★☆☆ ❤
Que fait cette fonction ?

```
def mystere(L):
    M=[]
    n=len(L)
    for k in range(n-1,-1,-1):
        M=M+[L[k]]
    return M
```

Exercice 12. ☆☆☆ 🚲 **Affichage de lettres en séquence**
Écrire une fonction qui affiche, dans l'ordre et les unes au-dessous des autres, les lettres d'une chaîne de caractères donnée.

Exercice 13. ★☆☆ ❤ **Nombre de voyelles**
Écrire une fonction `nombre_voyelles` qui renvoie le nombre de voyelles d'une chaîne de caractères donnée.

Exercice 14. ★☆☆ **Affichage d'un triangle d'étoiles**
Que fait l'instruction `3*'bla'` ?
En déduire une fonction qui affiche `n` lignes analogues aux quatre suivantes :

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

Exercice 15. ★★☆ ❤ **Palindrômes**

1. Écrire une fonction qui renvoie la chaîne de caractères obtenue comme miroir d'une chaîne de caractères donnée.
   Par exemple, la fonction appliquée à `"Hugo grandit"` renverra `"tidnarg oguH"`.
2. Écrire une fonction qui teste si une chaîne de caractères est un palindrôme.

Exercice 16. ★★★ **The look and say sequence**
On observe la suite d'entiers naturels suivantes :

1 11 21 1211 111221 312211 ...

1. Conjecturer les septième et le huitième termes de cette suite.
2. Écrire une fonction `conway` qui reçoit en argument un entier `n` et affiche les `n` premiers termes de la suite.
   Indication : on pourra utiliser la fonction `str` *qui convertit un entier en une chaîne de caractères.*

Exercice 17. ★★★ **Anagrammes**
Écrire une fonction qui teste si deux chaînes de caractères sont des anagrammes, c'est-à-dire si elles sont formées des mêmes lettres mais dans un ordre différent.
Les espaces et les accents ne sont pas pris en compte. Ainsi, sont des anagrammes :

- `"aube"` *et* `"beau"`
- `"parisien"` *et* `"aspirine"`
- `"traces écrites"` *et* `"écarts et crise"`

Exercice 18. ☆☆☆ ❤ **Suite arithmético-géométrique**
Écrire une fonction qui, étant donnés quatre nombres `a` (flottant différent de 1), `b` (flottant non nul), `u0` (flottant) et `n` (entier naturel), renvoie le $n^{\text{ème}}$ terme de la suite arithmético-géométrique de raisons `a` et `b` et de premier terme `u0`.

Exercice 19. ★☆☆ ❤ **Suite récurrente double**
On considère la suite (u_n) définie par $u_0 = u_1 = 1$ réel fixé quelconque et

$$\forall n \in \mathbb{N}, \quad u_{n+2} = u_{n+1} + \frac{2}{n+2} u_n.$$

Écrire une fonction qui renvoie la liste des valeurs de u_k pour k entre 0 et un entier naturel donné n.

Exercice 20. ★★☆ **Suite itérative**
Écrire une fonction qui, étant donné un entier naturel n, renvoie

$$\sqrt{2+\sqrt{2+\sqrt{2+\cdots+\sqrt{2+\sqrt{2}}}}}$$

où le symbole racine est utilisé n fois.

Solutions

Solution de l'ex 1. 1. Voici la simplissime fonction `cube` :

```
def cube(x):
    return x**3
```

2. On utilise la fonction `cube` pour définir la fonction suivante :

```
import math
def volume_sphere(r):
    return 4*math.pi*cube(r)/3
```

Solution de l'ex 2. Voici ladite fonction :

```
import math
def norme(v):
    s=0
    for i in range(len(v)):
        s+=v[i]**2
    return math.sqrt(s)
```

Solution de l'ex 3. On reprend les deux algorithmes écrits dans le chapitre précédent :

```
def somme(L):
    S=0
    for elem in L:
        S+=elem
    return S
```

```
def produit(P):
    P=1
    for elem in L:
        P*=elem
    return P
```

Solution de l'ex 4. 1. On calque une solution à cette question sur l'exercice précédent :

```
def somme_entiers(n):
    S=0
    for k in range(1,n+1):
        S+=k
    return S
```

2. Simplement, on propose :

```
def somme_entiers2(n):
    return n*(n+1)/2
```

3. On propose l'idée suivante : on initialise un booléen (`test`) à `True` ; ensuite, on effectue les 100 tests concernés et dès qu'un test se révèle faux, on affecte à `test` la valeur `False` (dans ce cas, on rajoute l'instruction `break` : son effet est d'arrêter la boucle `for`). À la fin de ce procédé, il y a deux possibilités : soit l'un (au moins) des tests s'est avéré faux, auquel cas `test` contient la valeur `False` ; soit tous les tests étaient vrais, et alors `test` n'a pas été modifié et contient donc la valeur `True`.

```
test=True
for k in range(100):
    if somme_entiers(k)!=somme_entiers2(k):
        test=False
        break # facultatif ; plus « efficace »
print(test)
```

Solution de l'ex 5. 1. Voici la fonction factorielle avec une boucle `for` :

```
def factorielle(n):
    f=1
    for k in range(1,n+1):
        f*=k
    return f
```

2. Voici une proposition de fonction, construite à partir de la fonction précédente. On sait que $\binom{n}{k}$ est un entier naturel, dès que k et n le sont ; de plus on sait que le calcul sur les entiers en Python est axact, alors que le calcul sur les flottants est approché. En conséquence, on utilise l'instruction `//` au lieu de `/` pour effectuer la division apparaissant dans la définition du coefficient binomial.

```
def coeff_binome(n,k):
    if 0<=k<=n:
        return factorielle(n)//(factorielle(k)*factorielle(n-k))
    else:
        return 0
```

3. Voici la fonction `puissance` :

```
def puissance(a,n):
    if n==0:
        return 1
    else:
        p=1
        for k in range(n):
            p*=a
        return p
```

4. Voici enfin la fonction `Newton1` :

```
def Newton1(a,b,n):
    S=0
    for k in range(n+1):
        S+=coeff_binome(n,k)*puissance(a,k)*puissance(b,n-k)
    return S
```

5. Voici la fonction `Newton2` :

```
def Newton2(a,b,n):
    return puissance(a+b,n)
```

6. Voici un script pour tester l'égalité des deux fonctions précédentes, en attribuant à `a`, `b` et `n` deux valeurs arbitraires :

```
a,b,n=2,3,5
if Newton1(a,b,n)==Newton2(a,b,n):
    print("les deux fonctions retournent le même résultat")
else:
    print("les deux fonctions ne retournent pas le même résultat")
```

7. Affichage du triangle de Pascal, les lignes au dessous des autres, affichées sous forme de lignes :

```
N=5
for n in range(N+1):
    L=[]
    for k in range(n+1):
        L+=[coeff_binome(n,k)]
    print(L)
```

Solution de l'ex 6. 1. Voici la fonction `syracuse` :

```
def syracuse(a):
    if a%2==0:
        return a//2
    else:
        return 3*a+1
```

2.3. Voici la version directement modifiée :

```
def test_syracuse(n):
    N=0
    while n!=1:
        n=syracuse(n)
        N+=1
    return N
```

Solution de l'ex 7. Voici une première version :

```
def diviseurs_positifs(n):
    L=[]
    for k in range(1,n+1):
        if n%k==0:
            L+=[k]
    return L
```

On observe que l'on aurait pu se passer d'un grand nombre de tests, très facilement : en effet, entre $\sqrt{n}+1$ et $n-1$, aucun entier ne peut être diviseur de n... On propose alors l'amélioration suivante :

```
from math import sqrt
def diviseurs_positifs2(n):
    L=[1]
    for k in range(2,int(sqrt(n))+1):
        if n%k==0:
            L+=[k]
    return L+[n]
```

Cette amélioration n'est pas anecdotique. Avec la bibliothèque `time`, on peut observer une différence temporelle pour obtenir un résultat de l'un ou l'autre algorithme et, par exemple, il aura fallu plus de 252 secondes à Python pour calculer `diviseurs_positifs(1 000 000 000)` alors que le calcul de `diviseurs_positifs2(1 000 000 000)` a été presque immédiat...

Solution de l'ex 8. Voici une solution bien décortiquée, mais sans commentaire et donc illisible :

```
def suivant(L,n):
    der=-1
    m=n
    while L[der]==m-1:
        der-=1
        m-=1
    L[der]+=1
    if L[0]==n-len(L):
        return L,False
    else:
        return L,True
def conversion(L,Lp):
    M=[]
    for place in Lp:
        M+=[L[place]]
    return M
def sous_listes_taille(L,k):
    Lp=list(range(k))
    SLk=[conversion(L,Lp)]
    test=True
    while test:
        Lp,test=suivant(Lp,len(L))
        SLk+=[conversion(L,Lp)]
    return SLk
def sous_listes(L):
    SL=[[]]
    for k in range(1,len(L)):
        SL+=[sous_listes_taille(L,k)]
    return SL+[L]
```

Solution de l'ex 9. Cette fonction s'applique à deux flottants `a` et `b` et renvoie le couple `(b,a)`.

Solution de l'ex 10. Cette fonction s'applique à deux entiers `a` et `b` et renvoie `a%b`.

Solution de l'ex 11. Cette fonction renvoie la liste « miroir » d'une liste donnée (c'est-à-dire la liste elle-même, mais dont les éléments ont été écrits dans l'ordre inverse).

Solution de l'ex 12. Voici une fonction :

```
def affiche(ch):
    for lettre in ch:
        print(lettre)
```

Solution de l'ex 13. Voici une fonction :

```
def nombre_voyelles(ch):
    cpt=0
    for lettre in ch:
        if lettre in 'aeiouyAEIOUY':
            cpt+=1
    return cpt
```

Solution de l'ex 14. `3*'bla'` renvoie `'blablabla'`. D'où la fonction :

```
def affiche(n):
    for k in range(1,n+1):
        print(k*'*')
```

Solution de l'ex 15. 1. Cf. exercice 11.

2. En utilisant une fonction `miroir` écrite en question 1, on propose :

```
def palindrome(ch):
    return ch==miroir(ch)
```

Solution de l'ex 18. Il s'agit de calculer u_n où $\forall k \in \mathbb{N}$, $u_{k+1} = au_k + b$.

```
def suite_arithm_geom(a,b,u0,n):
    u=u0
    for k in range(n):
        u=a*u+b
    return u
```

Solution de l'ex 19. On gère la relation de récurrence en utilisant deux variables que l'on fait évoluer en même temps. On stocke le tout dans une liste.

```
def suite_rec_double(n):
    u,v=1,1
    L=[u,v]
    for k in range(n-1):
        u,v=v,v+2*u/(k+2)
        L+=[v]
    return L
```

C'est en fait plus simple de gérer la relation de récurrence en cherchant les termes u_{n+1} et u_n dans la suite des termes de u :

```
def suite_rec_double2(n):
    L=[1,1]
    for k in range(n-1):
        L+=[L[-1]+2*L[-2]/(k+2)]
    return L
```

Solution de l'ex 20. La première difficulté de cet exercice est de traduire l'énoncé : le nombre cherché peut s'écrire comme le terme général de la suite récurrente définie par $u_1 = \sqrt{2}$ et $\forall n \in \mathbb{N}^*$, $u_{n+1} = \sqrt{2+u_n}$. La seconde difficulté est de traiter le cas où $n = 0$ à part : dans ce cas, le nombre vaut 2...

```
import math # bibliothèque contenant la fonction racine carrée
n=2 # valeur de l'entier n pour tester
if n==0: # le cas où n vaut 0 est très particulier...
    u=2
else: # les autres cas
    u=math.sqrt(2) # initialisation de la valeur de u (u_1)
    for k in range(2,n+1): # calcul de u_n avec n >= 2
        u=math.sqrt(2+u)
print(u)
```