Exercices 11 à 18

Exercice 11 (arguments optionnels I). On considère une fonction `f` définie avec l'en-tête suivant:

```
def f(x,y,z=3,L=[],t=True):
    ...
```

Parmi les lignes ci-dessous, lesquelles ne constituent pas des appels valides à la fonction `f` ? (expliquer alors pourquoi)

```
1  f(1)
2  f(1,3)
3  f(1,3,4)
4  f(y=2,x=1)
5  f(t=False,1,2)
6  f(4,5,L=[1])
7  f(1,1,3,False)
8  f(1,1,L=[3],False)
9  d = {'x':1,'y':2,'t':False}; f(**d)
10 d = dict(x=1,y=2,t=False); f(**d)
```

Exercice 12 (arguments optionnels II). On appelle une fonction `f` avec la syntaxe `f(3,[],z=5)`. Parmi les en-têtes de définition de la fonction `f` considérés ci-dessous, lesquels conduiront à une erreur ? (expliquer alors pourquoi)

```
1  def f(x,L,t=True,z=1):
2  def f(x,L,t=True,z):
3  def f(x,z=5,L=[]):
4  def f(x,L,t,z):
```

Exercice 13 (arguments optionnels et fonctions lambda).
Déterminer ce qu'affichent les programmes ci-dessous:

```
def f(x,y=3,z=True):
    if z:
        return y-x

print(f(3))
print(f(5,z=False))
print(f(y=1,x=3))
```

```
g = lambda x,y=1 : x//y+x%y
print(g(17))
print(g(17,3))
```

```
def F(g,x,y=3,z=2):
    return x+g(y)*z

print(F(lambda t:t**2,2,4))
```

Exercice 14 (méthode des trapèzes).

La méthode des trapèzes consiste à estimer numériquement la valeur de $\int_a^b f$ en subdivisant l'intervalle $[a,b]$ en n sous-intervalles de même longueur, et en estimant sur chaque sous-intervalle $[\alpha,\beta]$ l'intégrale $\int_\alpha^\beta f$ par l'aire du trapèze de sommets $(\alpha,0);(\alpha,f(\alpha));(\beta,f(\beta));(\beta,0)$.

(1) Écrire l'expression mathématique de l'estimation de $\int_a^b f$ par la méthode des trapèzes (on pourra s'inspirer de la formule vue en cours pour la méthode des rectangles).

(2) Écrire la fonction Python `trapezes(f,a,b,n)` associée.

(3) **(TP)** Calculer, à 10^{-6} près (précision estimée), la constante de Wilbraham-Gibbs $C = \displaystyle\int_0^\pi \frac{\sin x}{x}\,dx$ (expliquer la démarche utilisée pour atteindre cette précision).

Exercice 15 (appels récursifs). Indiquer le résultat affiché par chaque programme ci-dessous.

```
def f(a,b):
    if b==0:
        return a
    return f(a+1,b-1)

print(f(23,41))
print(f(10,-1))
```

```
def g(x):
    if x==0:
        return 1
    return 10*g(x-1)

print(g(8))
print(g(7.99))
```

```
def h(x):
    assert type(x) is int
    if x==0:
        return "0b"
    return h(x//2)+str(x%2)

print(h(19))
print(h(1.3))
```

Exercice 16 (suite récurrente d'ordre 1). Soit $(u_n)_{n\in N}$ la suite définie par $u_0 \in \mathbb{R}$ et la récurrence

$$\forall n \in \mathbb{N}, \quad u_{n+1} = \begin{cases} \sin(u_n) & \text{si} \quad u_n > \frac{1}{2}, \\ \cos(u_n) & \text{sinon.} \end{cases}$$

(1) Écrire une fonction `suite1(u0,n)` non récursive qui renvoie le terme d'indice `n` associé à `u0`.
(2) Écrire une fonction `suite2(u0,n)` **récursive** qui renvoie le même résultat que `suite1(u0,n)`.
(3) Écrire une fonction `suite3(u0,n)` non récursive qui renvoie la liste des termes d'indices 0 à `n`.
(4) Écrire une fonction `suite4(u0,n)` **récursive** qui renvoie le même résultat que `suite3(u0,n)`.

Exercice 17 (retour sur la suite de Syracuse). On rappelle que la suite de Syracuse est définie par $u_0 \in \mathbb{N}^*$ et la relation de récurrence

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

(1) Écrire une fonction `temps_de_vol(u0)`, qui renvoie le premier indice n pour lequel $u_n = 1$.
(2) (**TP**) Calculer (dans une liste `y`) les *temps de vol* pour tous les entiers `u0` compris entre 1 et 100 (stockés dans une liste `x`), puis tracer le graphe de la fonction `temps_de_vol` à l'aide des instructions Python

```
import matplotlib.pyplot as plt
plt.plot(x,y); plt.show()
```

(3) Écrire une fonction `altitude_max(u0)`, qui renvoie le maximum de la suite (u_n) (on supposera que la conjecture de Syracuse est vraie pour ne calculer qu'un nombre fini de termes)
(4) (**TP**) Représenter, en suivant le même schéma qu'à la question 2, le graphe de la fonction `altitude_max` pour `u0` compris entre 1 et 100.

Exercice 18 (complexité de l'algorithme d'Euclide).

(1) Rappeler (sans regarder le cours !) l'algorithme d'Euclide pour calculer le PGCD de deux entiers positifs a et b, et proposer une implémentation en Python sous la forme d'une fonction **récursive** `pgcd(a,b)`.
(2) On souhaite estimer la complexité dans le cas le pire de cet algorithme, en comptant, pour un réel positif A donné, le nombre maximal $c(A)$ de divisions euclidiennes calculées lors du calcul de `pgcd(a,b)` pour $0 \leqslant b \leqslant a \leqslant A$.
   (a) On suppose que $0 \leqslant b \leqslant a \leqslant A$ et on note r le reste dans la division de a par b. Montrer que $r \leqslant \frac{a}{2}$ (on pourra distinguer deux cas, selon que $b \leqslant \frac{a}{2}$ ou non).
   (b) En déduire que le calcul de $\text{pgcd}(a,b)$ aboutit, deux appels récursifs plus tard (si l'algorithme ne termine pas avant), au calcul de $\text{pgcd}(a',b')$ avec $a' \geqslant b'$ et $a' \leqslant \frac{a}{2}$.
   (c) En déduire que $c(A) \leqslant c\left(\frac{A}{2}\right) + 2$, puis que que $c(A) \leqslant 1 + 2\log_2(A)$, puis que $c(A) = O(\log A)$.
(3) **(TP*)** Écrire une fonction non récursive `complexite_pgcd(a,b)` qui renvoie le nombre de divisions effectuées pour calculer le PGCD de a et b par l'algorithme d'Euclide, puis former une liste double `L` telle que `L[b][a]=complexite_pgcd(a,b)` pour tous les entiers $0 \leqslant a, b \leqslant 1000$. Visualiser ensuite le résultat à l'aide des instructions Python ci-dessous et estimer la pente $(\Delta a/\Delta b)$ de la droite remarquable observée (celle qui correspond aux pires cas). Ce nombre vous rappelle-t-il quelque chose ?

```
import matplotlib.pyplot as plt
plt.imshow(L); plt.show()
```