**Question 1.** En approchant la dérivée $y'(t_k)$ par le taux d'accroissement

$$\frac{y(t_{k+1})-y(t_k)}{t_{k+1}-t_k} = \frac{y(t_{k+1})-y(t_k)}{h},$$

on trouve que, pour tout $k \in [\![0, n-1]\!]$, $\dfrac{y_{k+1}-y_k}{h} = \sin(t_k)\sin(y_k)$, i.e. :

$$\forall k \in [\![0, n-1]\!],\ y_{k+1} = y_k + h\sin(t_k)\sin(y_k).$$

```
a, b = 0, 50
y_0, n = 1, 255
t, y = [a], [y_0]
h = (b-a)/n
for k in range(n):
    y.append(y[k]+h*np.sin(t[k])*np.sin(y[k]))
    t.append(t[k]+h)
plt.plot(t,y)
plt.show()
```

**Question 2.** Par le même raisonnement qu'à la question précédente, on trouve

$$\forall k \in [\![0, n-1]\!],\ y_{k+1} = y_k + hF(t_k, y_k).$$

```
def euler(F,a,b,y_0,n):
    t, y = [a], [y_0]
    h = (b-a)/n
    for k in range(n):
        y.append(y[k]+h*F(t[k],y[k]))
        t.append(t[k]+h)
    return t, y
```

**Question 3.** On exploite une propriété de la fonction `exp` (et plus généralement de toute fonction) du module `numpy` : la fonction est vectorielle. Ainsi, si `t` est une liste ou un array de nombres, `np.exp(t)` est array dont les éléments sont les images des éléments de `t` par la fonction `exp`.
On identifie enfin la fonction $F$ : l'équation différentielle $y' = y$ est de la forme $y'(t) = F(t, y(t))$ où $F : (t, y) \mapsto y$.

```
def F(t,y):
    return y
a, b, y_0 = 0, 1, 1
for n in [10,100,1000]:
    t, y = euler(F, a, b, y_0, n)
    legende = "n = " + str(n)
    plt.plot(t,y,label = legende)
plt.plot(t, np.exp(t), label= "exp")
plt.legend(loc = "best")
plt.show()
```

**Question 4.** On s'inspire directement du code de la fonction `euler`.

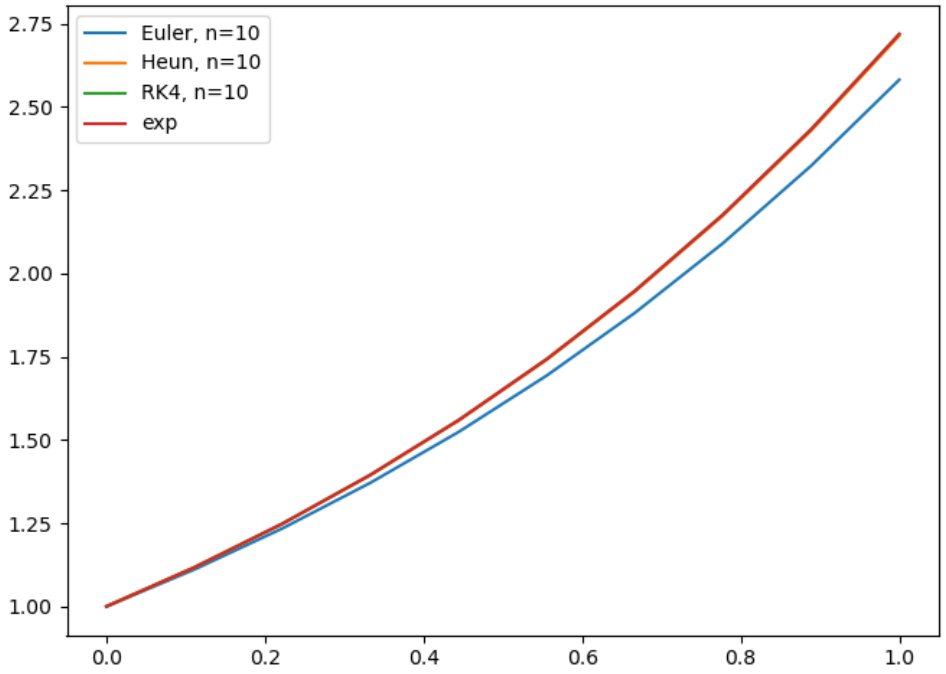
```
def heun(F,a,b,y_0,n):
    t, y = [a], [y_0]
    h =  (b-a)/n
    for k in range(n):
        p1 = F(t[k],y[k])
        p2 = F(t[k]+h,y[k]+h*p1)
        y.append(y[k]+h*(p1+p2)/2)
        t.append(t[k]+h)
    return t, y

def rk4(F,a,b,y_0,n):
    t, y = [a], [y_0]
    h =  (b-a)/n
    for k in range(n):
        p1 = F(t[k],y[k])
        p2 = F(t[k]+h/2,y[k]+h*p1/2)
        p3 = F(t[k]+h/2,y[k]+h*p2/2)
        p4 = F(t[k]+h,y[k]+h*p3)
        y.append(y[k]+h*(p1+2*p2+2*p3+p4)/6)
        t.append(t[k]+h)
    return t, y
```

*On fera attention à bien faire des calculs intermédiaires en Python ; pas parce que cela est nécessaire, mais pour limiter le risque d'erreur, pour faciliter la relecture et la compréhension du code par le jury.*

**Question 5.** Le script ci-dessous permet l'obtention du graphique ci-après.

```
a, b, y_0, n = 0, 1, 1, 10
t_euler, y_euler = euler(F, a, b, y_0, n)
plt.plot(t_euler, y_euler, label = "Euler, n=10")
t_heun, y_heun = heun(F, a, b, y_0, n)
plt.plot(t_heun, y_heun, label="Heun, n=10")
t_rk4, y_rk4 = rk4(F, a, b, y_0, n)
plt.plot(t_rk4, y_rk4, label="RK4, n=10")
plt.plot(t_rk4, np.exp(t_rk4), label="exp")
plt.legend(loc = "best")
plt.show()
```




**Question 6.** Lorsque $\beta = 0$, tout se passe comme s'il n'y avait pas de prédateurs. La première équation du système devient alors $x' = \alpha x$. La résolution de cette équation différentielle montre que la population de proies croit exponentiellement.

Lorsque $\delta = 0$, tout se passe comme s'il n'y avait pas de proies. La seconde équation du système devient alors $y' = -\gamma y$. La résolution de cette équation différentielle montre que la population de prédateurs décroit exponentiellement.

**Question 7.** La quantité $\beta x(t)y(t)$ représente le nombre de proies mangés par les prédateurs par unité de temps à l'instant $t$.

La quantité $\delta x(t)y(t)$ représente le nombre de naissances des prédateurs par unité de temps à l'instant $t$.

**Question 8.** En approchant les dérivées $x'(t_k)$ et $y'(t_k)$ par les taux d'accroissement $\dfrac{x_{k+1}-x_k}{t_{k+1}-t_k}$ et $\dfrac{y_{k+1}-y_k}{t_{k+1}-t_k}$, on obtient :

$$\forall k \in [\![0, n-1]\!],\ \frac{x_{k+1}-x_k}{h} = x_k(\alpha - \beta y_k) \quad \text{et} \quad \frac{y_{k+1}-y_k}{h} = y_k(\delta x_k - \gamma),$$

ou encore :

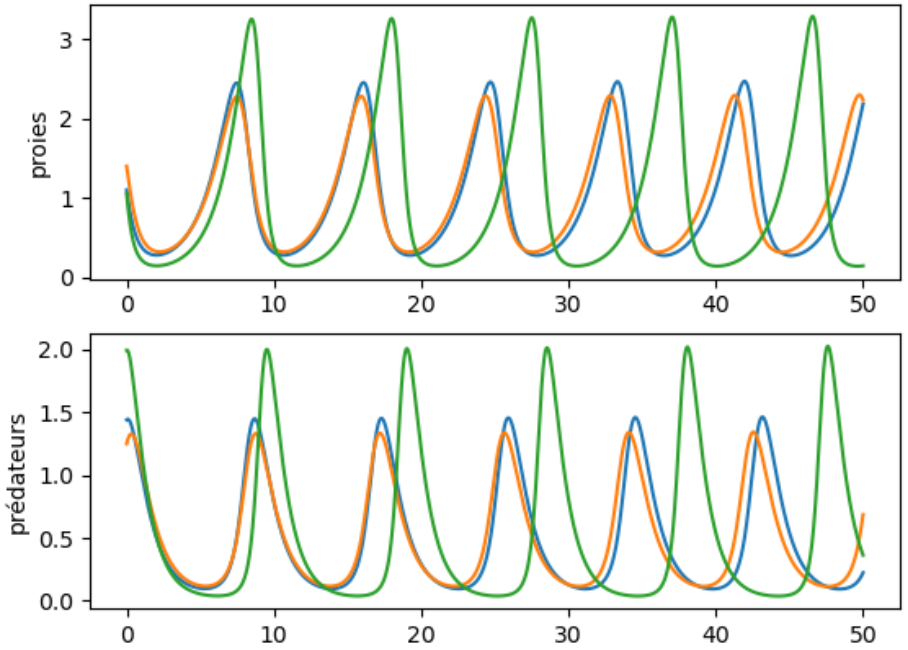
$$\forall k \in [\![0, n-1]\!],\ x_{k+1} = x_k + hx_k(\alpha - \beta y_k) \quad \text{et} \quad y_{k+1} = y_k + hy_k(\delta x_k - \gamma).$$

On en déduit le code de la fonction demandée :

```
def lotkaVolterra(x0,y0,a,b,n,alpha,beta,delta,gamma):
    t, x, y = [a], [x0], [y0]
    h = (b-a)/n
    for k in range(n):
        x.append(x[k] + h*x[k]*(alpha-beta*y[k]))
        y.append(y[k] + h*y[k]*(delta*x[k]-gamma))
        t.append(t[k]+h)
    return t, x, y
```

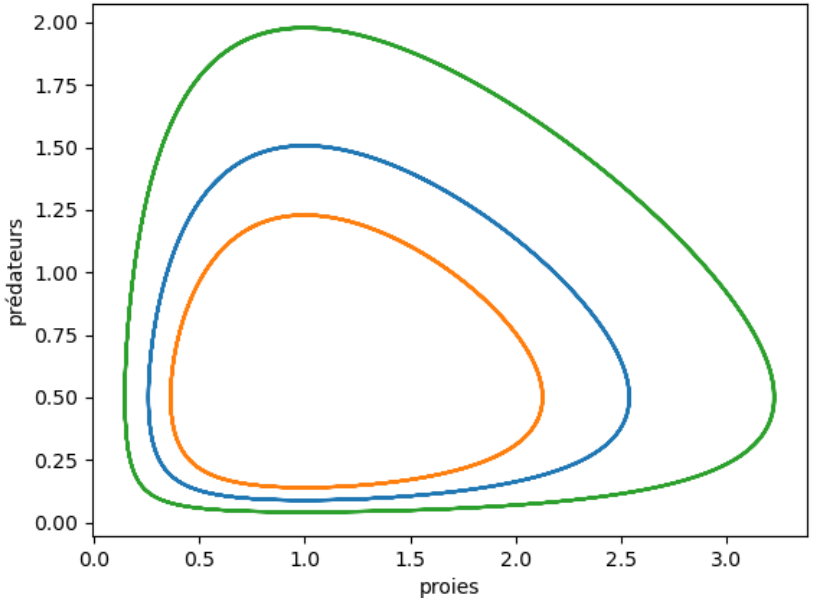
**Question 9.** Le script ci-dessous permet d'afficher l'évolution de trois populations de proies/prédateurs, en choisissant au hasard les effectifs initiaux.

```
a, b= 0, 50
alpha, beta = 2/3, 4/3
delta, gamma = 1, 1
n = 500000
fig, (ax1, ax2) = plt.subplots(2)
for u in range(3):
    x0, y0 = 1+rd.random(), 1+rd.random()
    t, x, y = lotkaVolterra(x0,y0,a,b,n,alpha,beta,delta,gamma)
    ax1.plot(t,x)
    ax2.plot(t,y)
ax1.set_ylabel("proies")
ax2.set_ylabel("prédateurs")
plt.show()
```




**Question 10.** Le script ci-dessous permet d'afficher ci-après l'évolution de trois populations de prédateurs en fonction de trois populations de proies.

```
a, b= 0, 50
alpha, beta = 2/3, 4/3
delta, gamma = 1, 1
n = 500000
for u in range(3):
    x0 = 1+rd.random()
    y0 = 1+rd.random()
    t, x, y = lotkaVolterra(x0,y0,a,b,n,alpha,beta,delta,gamma)
    plt.plot(x,y)
plt.xlabel("proies")
plt.ylabel("prédateurs")
plt.show()
```




* *

*