Bibliotecas

In [ ]:
```
import sympy as sp
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation
from matplotlib.animation import PillowWriter
from sympy.printing import latex
```

Simbolos do sympy

In [ ]:
```
t, g, m, l = sp.symbols('t g m l')

theta = sp.symbols(r'theta', cls=sp.Function)
theta = theta(t)
theta_dot = sp.diff(theta, t)
theta_ddot = sp.diff(theta_dot, t)
```

Equações da posição em 'x' e 'y' da massa

In [ ]:
```
x = l*sp.sin(theta)
y = -l*sp.cos(theta)
```

Equação energia cinética

In [ ]:
```
T1 = sp.Rational(1, 2)*m*sp.diff(x, t)**2

T = T1

T
```

Out[ ]:
$$\frac{l^2 m \cos^2\left(\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2}{2}$$

Equação energia potencial gravitacional

In [ ]:
```
U1 = y*m*g

U = U1

U
```

Out[ ]: $-glm\cos\left(\theta(t)\right)$

Equação de Lagrange

In [ ]:
```
L = T - U

L
```

Out[ ]: $$glm\cos\left(\theta(t)\right) + \frac{l^2 m\cos^2\left(\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2}{2}$$

EDO θ(t)

In [ ]:
```
eq = sp.diff(L, theta) - sp.diff(sp.diff(L, theta_dot), t)
EDOθ = sp.simplify(eq)

EDOθ
```

Out[ ]: $$lm\left(-g\sin\left(\theta(t)\right) + \frac{l\sin\left(2\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2}{2} - l\cos^2\left(\theta(t)\right)\frac{d^2}{dt^2}\theta(t)\right)$$

Solução da EDO

In [ ]:
```
sols = sp.solve(EDOθ, theta_ddot)[0]

sols
```

Out[ ]:
$$-\frac{g\sin\left(\theta(t)\right)}{l\cos^2\left(\theta(t)\right)}+\tan\left(\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2$$

Transforma as equações simbólicas em equações solucionáveis

In [ ]:
```
dz1dt_f = sp.lambdify((theta, theta_dot, g, l, m), sols)
dthetadt_f = sp.lambdify(theta_dot, theta_dot)
Ep = sp.lambdify((theta, g, l, m), U)
Ec = sp.lambdify((theta, theta_dot, m, l), T)
```

Função que será usada para retornar a posição e velocidade no intervalo de tempo proposto pela solução no método ODEINT

In [ ]:
```
def dSdt(S, t, g, l, m):
    theta, thetad = S
    return [dthetadt_f(thetad),
            dz1dt_f(theta, thetad, g, l, m)]
```

Define as condições iniciais e calcula a solução da EDO

In [ ]:
```
tempo_simulacao = 10  # 10 s
passo = 1000  # 0.001 s
t = np.linspace(0, tempo_simulacao, passo+1)
g = 9.81
l = 1
m = 1
deg = 30
theta0 = deg*np.pi/180
dtheta0 = 0

sol = odeint(dSdt, y0=[theta0, dtheta0], t=t, args=(g, l, m))
```

Posição e Velocidade da massa

In [ ]:
```
thepos = sol.T[0]
thedot = sol.T[1]
```

Calcula a posição real com base no comprimento original da mola, como não há movimento na direção 'y' sua posição é sempre zero

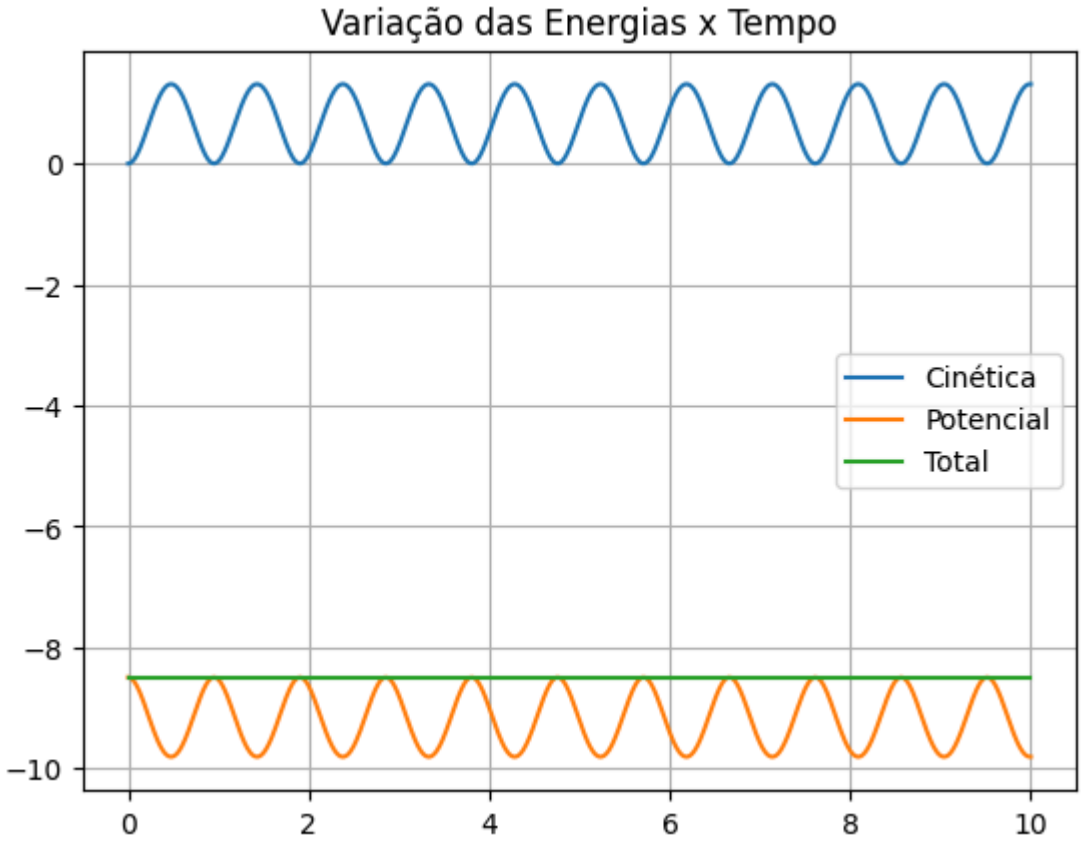
In [ ]:
```
def pos(t, the, l):
    x1 = l*np.sin(the)
    y1 = -l*np.cos(the)
    return [
        x1, y1
    ]


xpos, ypos = pos(t, thepos, l)
```

Cálcula as energias do sistema

In [ ]:
```
Cine = Ec(thepos, thedot, m, l)
Pot = Ep(thepos, g, l, m)
Etotal = Cine + Pot
```

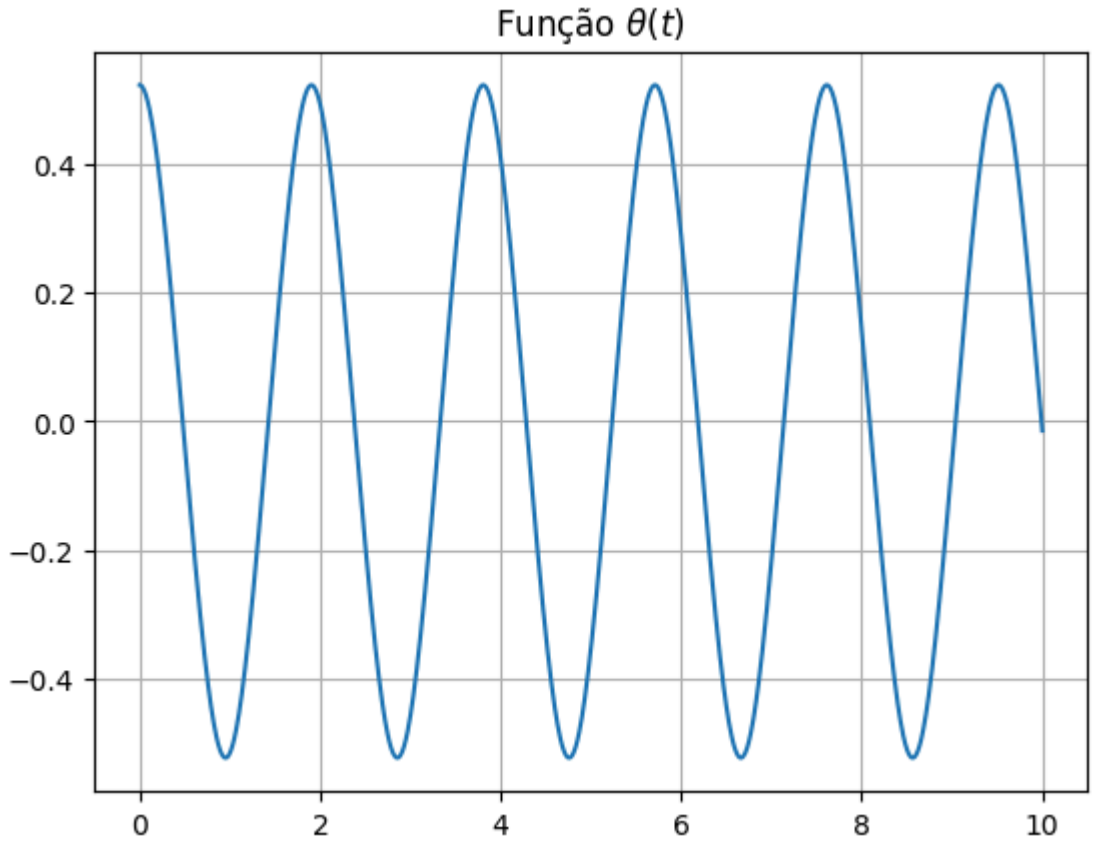
Plota a variação das energias

In [ ]:
```
plt.title('Variação das Energias x Tempo')
plt.plot(t, Cine, label='Cinética')
plt.plot(t, Pot, label='Potencial')
plt.plot(t, Etotal, label='Total')
plt.legend()
plt.grid()
plt.show()
```



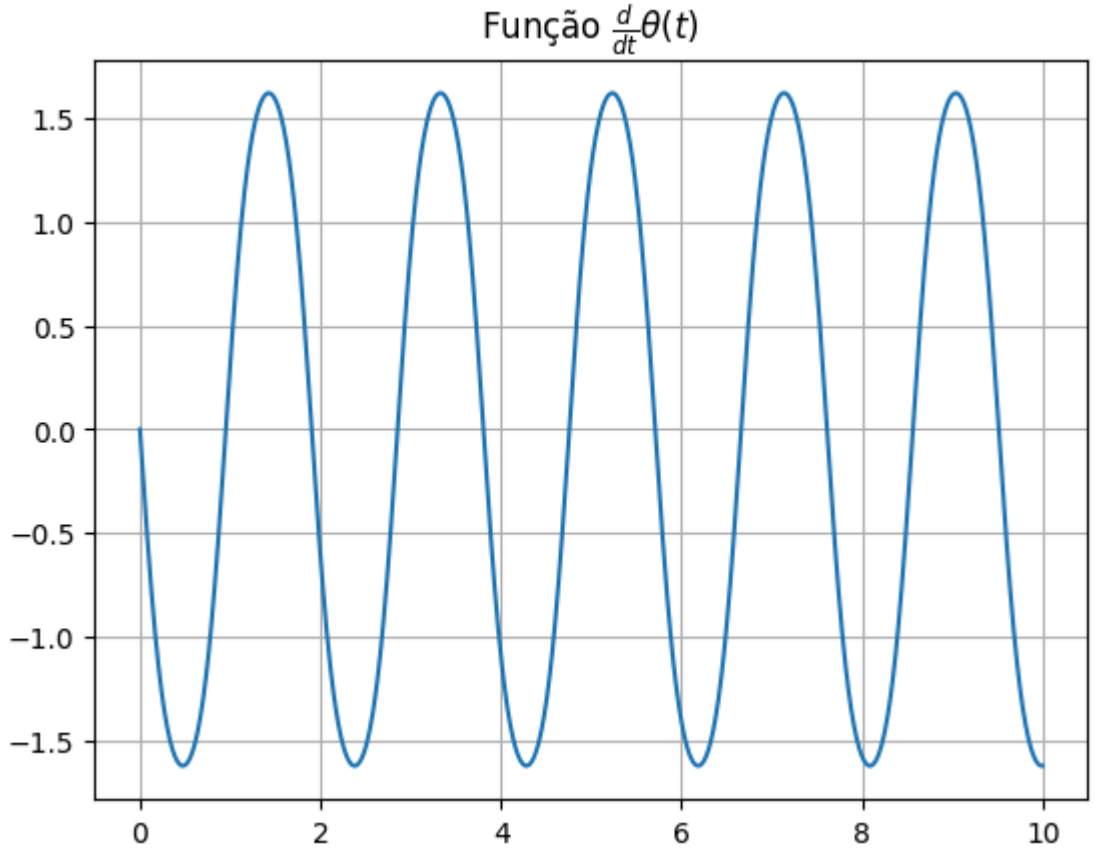
Plota a função θ(t)

In [ ]:
```
plt.title(f'Função ${latex(theta)}$')
plt.plot(t, thepos)
plt.grid()
plt.show()
```



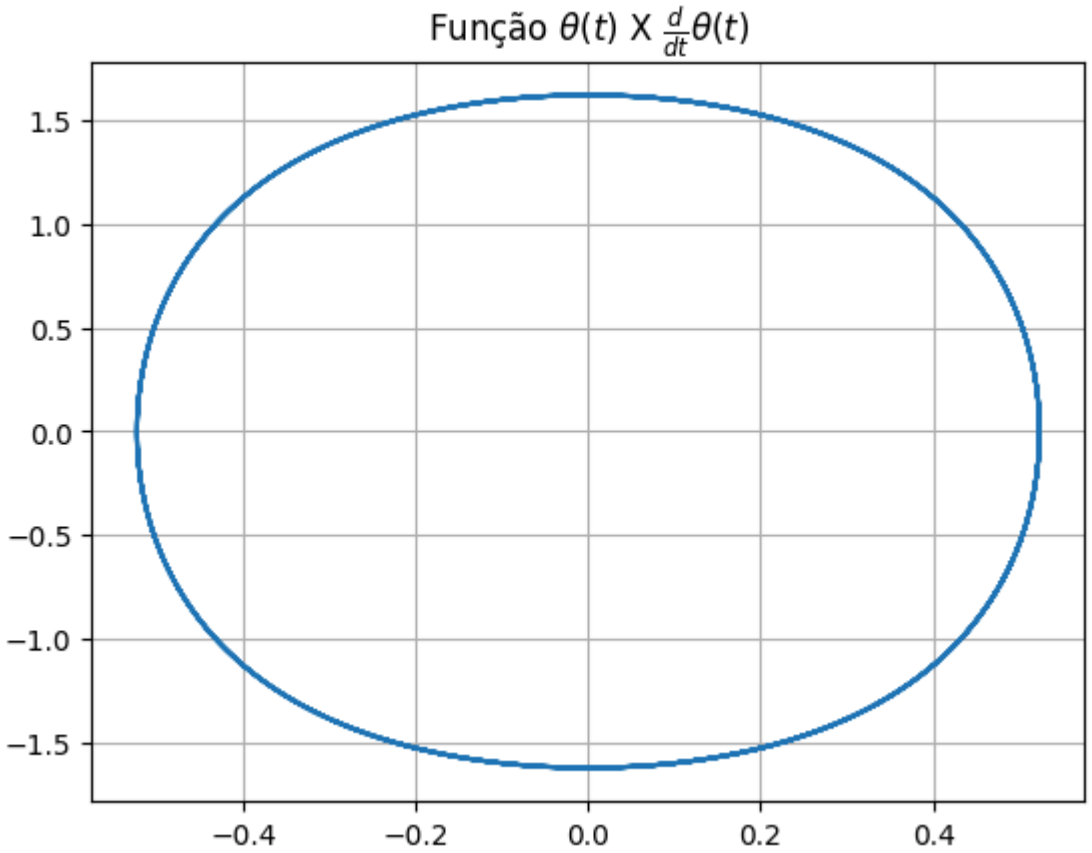
Plota a função dθ(t)/dt

In [ ]:
```
plt.title(f'Função ${latex(theta_dot)}$')
plt.plot(t, thedot)
plt.grid()
plt.show()
```



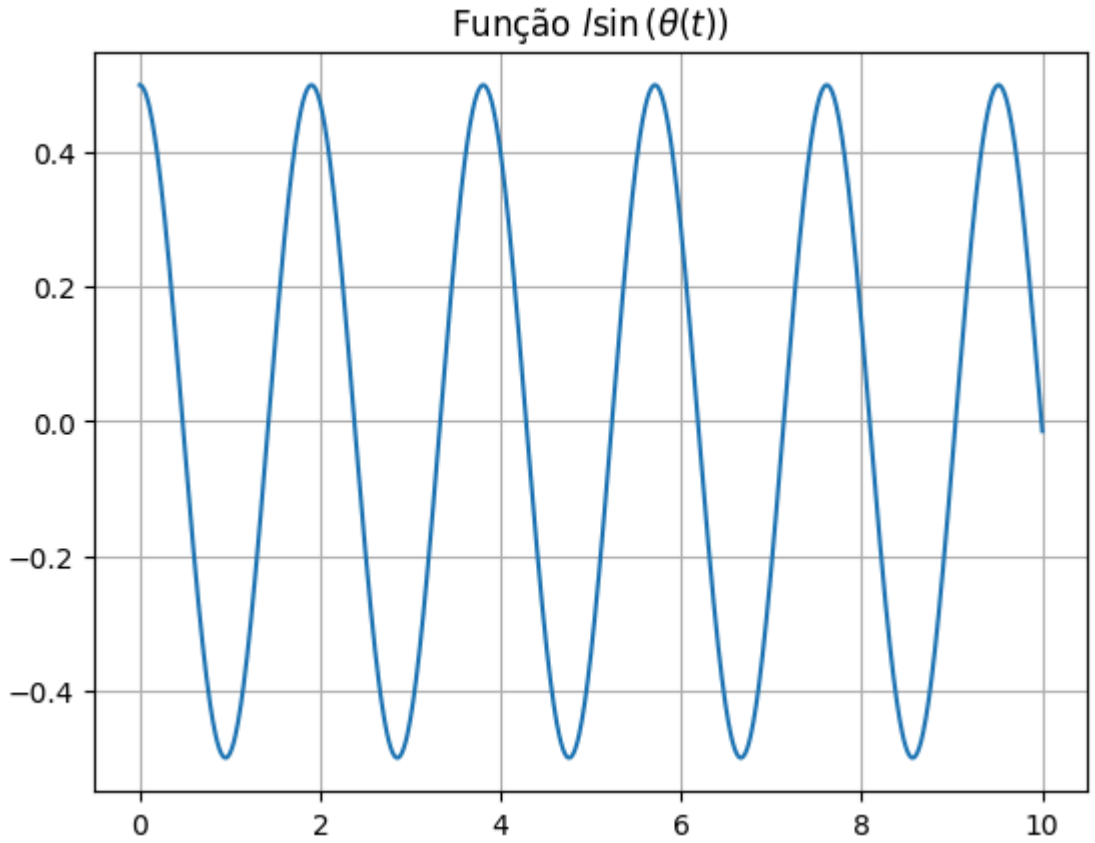
Plota a fase θ(t)xdθ(t)/dt

In [ ]:
```
plt.title(f'Função ${latex(theta)}$ X ${latex(theta_dot)}$')
plt.plot(thepos, thedot)
plt.grid()
plt.show()
```



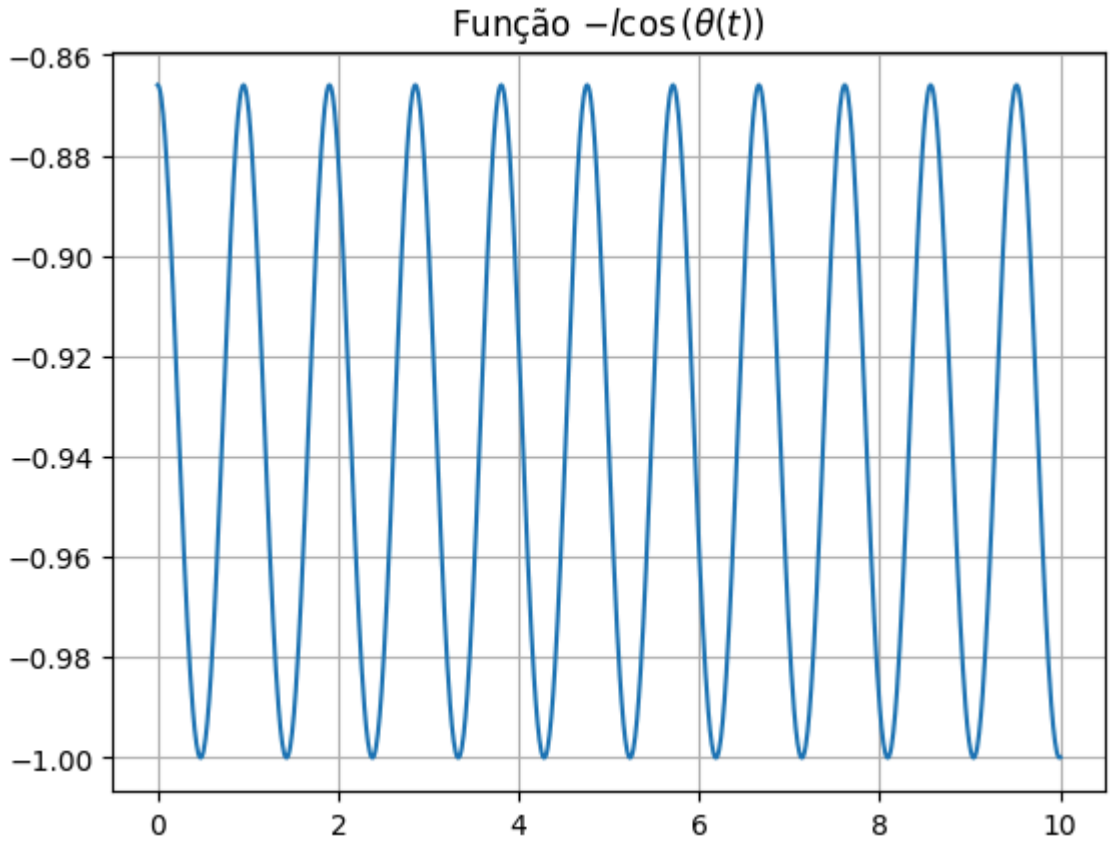
Plota a posição da massa em 'x'

In [ ]:
```
plt.title(f'Função ${latex(x)}$')
plt.plot(t, xpos)
plt.grid()
plt.show()
```



Plota a posição da massa em 'y'

In [ ]:
```
plt.title(f'Função ${latex(y)}$')
plt.plot(t, ypos)
plt.grid()
plt.show()
```



Cria a animação do sistema

In [ ]:
```
def animate(i):
    ln.set_data([0, xpos[i]], [0, ypos[i]])
    cur.set_data(xpos[:i+1], ypos[:i+1])


fig, ax = plt.subplots(1, 1, figsize=(5, 5))
ax.set_xlim((min(xpos)-0.5), (max(xpos)+0.5))
ax.set_ylim((min(ypos)-0.5), 0.5)
ax.grid()
ln, = ax.plot([], [], 'bo--', lw=2, markersize=8)
cur, = ax.plot(xpos[0], ypos[0], 'black', lw=1)
```

```
ani = animation.FuncAnimation(fig, animate, frames=passo, interval=10)
ani.save('pen_simples.gif', writer='pillow', fps=25)
```