```python
import numpy as np
import matplotlib.pyplot as plt
import math


# Q1: compute the taylor series approximation of f(x) = cos(x)

#finds percent error between two values
def error(val1, val2, abs=False):
    if abs:
        return np.abs(100 * (val1 - val2)/val2)
    else:
        return 100 * (val1 - val2)/val2

#returns value for cosine derivatives
def derivativeCos(x, level):
    level = level % 4
    if(level == 0):
        return math.cos(x)
    elif( level == 1):
        return -math.sin(x)
    elif(level == 2):
        return -math.cos(x)
    elif( level == 3):
        return math.sin(x)


#use taylor series to estimate value
def taylor_approx(x, x1, n ):
    sum = 0
    h = x1 - x
    for i in range(0, n + 1 ):
        sum += (derivativeCos(x,i) * ((h)**i)) / math.factorial(i)
    return sum

#error from taylor approx method
def taylor_error(x, x1, n):
    return error( taylor_approx(x, x1, n), math.cos(x1))

#number of terms to compute
nVals = [i for i in range(0,6)]
trueErrors = []
values = []
#computing values at x values
for n in nVals:
    trueErrors.append(taylor_error(.25, 1, n))
    values.append(taylor_approx(.25, 1, n))

#plotting
print(math.cos(1))
print(values)

plt.figure(figsize=(10,8))
plt.plot(nVals,trueErrors)
plt.title("Values by n terms")
plt.ylabel("Error percentage")
plt.xlabel("Number of terms")
plt.grid("on")
plt.show()



#The error rapidly approches zero. This approches the true value of cos(x) is a much faster method than euler's
```