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



def myDFT1(inputx, l, x, y):
    n = 2*l - 1
    acoff = np.arange(0, l+1).reshape(1, -1)
    bcoff = np.arange(1, l).reshape(1, -1)
    amid = (np.cos(x*acoff)).T
    bmid = (np.sin(x*bcoff)).T

    ak = 2 * (amid @ y) / (n + 1)
    bk = 2 * (bmid @ y) / (n + 1)

    akSum = np.array(ak[1:-1])
    coff = np.arange(1, l).reshape(-1, 1)
    inside = coff * inputx
    ainside = np.cos(inside).T
    binside = np.sin(inside).T
    asum = (ainside @ akSum).reshape(1, -1)
    print("asum1: \n", asum)

    bsum = (binside @ bk).reshape(1,-1)

    return asum + bsum + 0.5 * ak[0] + 0.5 * np.cos(l * inputx) * ak[-1]

def myDFT(inputx: np.array, l: int, x: np.array, y:np.array, endPoint=2*np.pi):
    # len(inputx) = m, len(x) = n+1, len(y) = n+1
    n = 2*l - 1
    DFTScalingFactor = 2 * np.pi / endPoint
    # assert len(x) == n + 1 and len(y) == n + 1
    acoffk = np.arange(0, l+1).reshape(-1, 1)  # shape of (l+1, 1)
    bcoffk = np.arange(1, l).reshape(-1, 1)    # shape of (l-1, 1)
    acoffk = acoffk * DFTScalingFactor
    bcoffk = bcoffk * DFTScalingFactor
    amid = np.cos(acoffk * x)                  # shape of (l+1, n+1)
    bmid = np.sin(bcoffk * x)                  # shape of (l-1, n+1)
    ak = 2* (amid @ y) / (n + 1)               # shape of (l+1)
    bk = 2* (bmid @ y) / (n + 1)               # shape of (l-1)
    # assert len(ak) == l+1 and len(bk) == l-1
    coffab = np.arange(1, l)
    coffab = coffab * DFTScalingFactor
    reshapeInputx = inputx.reshape(-1, 1)
    akSummation = np.array(ak[1:-1])
    bkSummation = bk
    # assert len(akSummation) == len(bk)
    asum = (np.cos(reshapeInputx * coffab) @ akSummation)
    bsum = (np.sin(reshapeInputx * coffab) @ bkSummation)
    # assert asum.shape == inputx.shape and bsum.shape == inputx.shape
    outputy = asum + bsum + 0.5 * (ak[0] + ak[-1] * np.cos(ak[l-1] * inputx))
    # assert len(outputy) == len(inputx)
    return outputy

def question1(l, endPoint= 2*np.pi, printB=False):
    n = 2*l - 1
    x = (endPoint * (np.arange(0, n + 1)) / ((n + 1)))
    y = np.log(x+1)
    print("question1 x: \n", x)

    inputx = (2 * np.pi * (np.arange(0, 32 + 1)) / (32 + 1))
    groundTruth = (np.log(inputx+1))

    myY = myDFT(inputx, l, x, y, endPoint)
    print("question1 myY: \n", myY)
```

```python
    error = myY - groundTruth
    errorNorm = np.linalg.norm(error)
    # print("myY: ", myY)
    # print("groundTruth: ", groundTruth)
    # print("error: ", error)
    plt.plot(inputx, myY, color="blue", label="DFT approximation")
    plt.plot(inputx, groundTruth, color="red", label="log(x+1)")
    plt.title(f"l: {l}, error: {errorNorm}")
    plt.legend()
    if printB:
        plt.savefig(f"question1_l={l}.jpg")
    plt.show()

# question1(16, True)

def f(x):
    return np.log(x + 1)

def g(x):
    return np.log(4 * np.pi - x + 1)

def getQuestion2Y(x):
    topX = x[x < 2 * np.pi]
    lowX = x[x >= 2 * np.pi]
    topY = f(topX)
    lowY = g(lowX)
    y = np.concatenate((topY, lowY))
    return y

def question2(l,endPoint= 2*np.pi, printB=False):
    n = 2*l - 1
    interpolatex = endPoint * np.arange(0, n + 1) / ((n + 1))
    interpolatey = getQuestion2Y(interpolatex)

    inputx = 2 * np.pi * np.arange(0, n + 1) / (n + 1)
    outputyTrue = getQuestion2Y(inputx)
    outputy = myDFT(inputx, l, interpolatex, interpolatey, endPoint)

    error = outputy - outputyTrue
    errorNorm = np.linalg.norm(error)
    plt.plot(inputx, outputy, color="blue", label="approximation")
    plt.plot(inputx, outputyTrue, color="red", label="groundTruth")
    plt.title(f"l: {l}, error: {errorNorm}")
    plt.legend()
    if printB:
        plt.savefig(f"question2_l={l}.jpg")
    plt.show()

question1(32, 4*np.pi, True)
```