```python
# — PRACTICE TEST CATALINA CISNEROS —

#0. import numpy and LT.box
import numpy as np
import LT.box as B

#1. Create array with given values
Es = np.array([0.6617, 1.1732, 1.3325])

#2. reate an empty array size 3
peaks = np.zeros(3)


#3. Read calibration file and save it in hcal
hcal=B.get_spectrum('calibration.Spe')

#4. show in a graph, with the automatic gaussian fit
hcal.plot()

#4. Select appropiate ranges
ranges = [(120, 144), (215, 243), (249, 272)]
i = 0
#6. Loop to repeat steps 4 and 5
for (xmin, xmax) in ranges:
    hcal.fit(xmin,xmax)
    hcal.plot()
    #5. obtain the mean value and store it in the array peaks
    peaks[i] = hcal.mean.value
    i += 1

# 7. Graph E vs peaks and label x and y axis

B.pl.figure()

B.plot_exp(Es, peaks)
B.pl.xlabel('Energy [MeV]')
B.pl.ylabel('peaks [Counts]')
B.pl.title('Energy vs Peaks')

# 8. Perform a linear fit on the graph
fitcal = B.linefit(peaks, Es)
fitcal.plot()


# 9. Verify calibration
B.pl.figure()
hcalnew = B.get_spectrum('calibration.Spe', calibration=fitcal)
hcalnew.plot()
# 10. set xaxis label to energies
B.pl.xlabel('Energies (MeV)')
```

```python
# 11. Create four member array
degrees = np.array([30, 40, 60, 80])
# 12. Convert to radians and store in thetas
thetas = np.deg2rad(degrees)

# 13. create an empty array for efs
efs = np.zeros(4)

# 14. create an empty array for efsErr
efsErr = np.zeros(4)

# 15. Apply calibration in 30_deg_target
htarget=B.get_spectrum('30_deg_target.Spe', calibration = fitcal)

# 16. Apply calibration in 30_deg_empty
hempty=B.get_spectrum('30_deg_empty.Spe', calibration = fitcal)

# 17. Graph both htarget and hempty
B.pl.figure()
htarget.plot()
hempty.plot()
B.pl.title('htarget and hempty')

# 18. hcompton substract target - empty
B.pl.figure()
hcompton = htarget - hempty
hcompton.plot()
B.pl.xlabel('Energies (MeV)')
B.pl.ylabel('counts')
B.pl.title('htarget-hempty')

# 19. Select appropiate ranges on htarget
hcompton.fit(0.45, 0.70)
hcompton.plot()
# 20. store peak pos mean in efs
efs[0] = np.array(hcompton.mean.value)
# 21. store err in efsErr
efsErr[0] = hcompton.mean.err

#22

deg = [40, 60, 80]
ranges = [(0.45, 0.70), (0.35, 0.60), (0.25, 0.50)]

i = [1, 2, 3]
for j, d in enumerate(deg, start=1):
    htarget=B.get_spectrum(f"{d}_deg_target.Spe", calibration = fitcal)
    hempty=B.get_spectrum(f"{d}_deg_empty.Spe", calibration = fitcal)
    B.pl.figure()
    htarget.plot()
```

```python
        hempty.plot()
        B.pl.figure()
        hcompton = htarget - hempty
        hcompton.plot()
        B.pl.xlabel('Energies (MeV)')
        B.pl.ylabel('counts')
        B.pl.title(f'{d} htarget-hempty')
        # 19. Select appropiate ranges on htarget
        xmin, xmax = ranges[j - 1]
        hcompton.fit(xmin, xmax)
        hcompton.plot()
        B.pl.title(f'{d} htarget-hempty')
        # 20. store peak pos mean in efs
        efs[j] = np.array(hcompton.mean.value)
        # 21. store err in efsErr
        efsErr[j] = hcompton.mean.err

# 23. compute e0/efs
e0= Es[0]
eratio = e0/efs
print(eratio)

# 24. graph eratio vs cos thetas
B.pl.figure()
cos_t = 1.0 - np.cos(thetas)
B.plot_exp(cos_t, eratio)
B.pl.ylabel('e-ratio')
B.pl.xlabel('cos(thetas)')
B.pl.title('1 - cos thetas vs e-ratio')

# 25. linefit to above
fit2 = B.linefit(cos_t, eratio)

# 26. slope from fit2
# 27. slope unc from fit2

# 28. offset unc from fit2
# 29. offset unc from fit2

eslope = fit2.slope
dslope = fit2.sigma_s
yoff   = fit2.offset
dyoff  = fit2.sigma_o

# 30. value of electron mass in MeV
me = e0 / eslope
dme = e0 * dslope / (eslope**2)


# 31. Print electron mass and uncertainty
print(f"Electron mass m_e = {me:.3f} ± {dme:.3f} MeV")
```

```python
# 32. Display electron mass result on the graph
B.pl.text(0.05, 0.90, f"m_e = {me:.3f} ± {dme:.3f} MeV", transform=B.pl.gca().tra

# 33. Print and show offset result on the graph
print(f"Offset = {yoff:.3f} ± {dyoff:.3f} (expected ≈ 1.000)")
B.pl.text(0.05, 0.84, f"Offset = {yoff:.3f} ± {dyoff:.3f} (expected ≈ 1.000)", tr
```