In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.constants
```

In [2]:

```
class Ising:
    def __init__(self,x,y,B=1.0,J=1.0,M=1.0,kT=1.0,xr=2,yr=2):
        ''' The initialization of the Ising class object, the enteries in order are x-len
gth, y-length, Magnetic Field, Coupling Factor, Mu, kT, Periodic Boundary conditions x an
d y '''
        self.xlen = x
        self.ylen = y
        #Default Config
        self.config = np.ones((x,y),dtype=int)
        #Characteristic properties --> Optional (Defaults already present)
        self.mag = B
        self.coup = J
        self.mu = M
        self.kT = kT
        self.xr = xr
        self.yr = yr
        self.prob = 0

    def __str__(self):
        ''' Returns the configuration when the object is printed'''
        return str(self.config)

    def reset(self):
        ''' Reset the configuration to all ones'''
        self.config = np.ones((self.xlen,self.ylen),dtype=int)
        returno11111

    def random(self):
        ''' Randomizes the configurations'''
        for i in range(self.xlen):
            for j in range(self.ylen):
                if np.random.randint(2) == 0 :
                    self.config[i][j] = -1
                else:
                    self.config[i][j] = 1

    def magnetization(self):
        ''' Returns the magnetization of the current configuration'''
        return self.mu*np.sum(self.config)/(self.xlen*self.ylen)

    def energy(self):
        ''' Calculates the energy of the current configuration od the given Ising model''
'
        # Potential only fron magnetic field
        V = -1 * self.mag * np.sum(self.config)
        V *= self.xr + self.yr - 1

        # Potential from interactions
        # For the interactions coupling with only the element below it and after it is co
nsidered
        # so as to not have any duplications
        temp = 0
        temp1=0
        temp2=0
        for i in range(self.xlen):
            for j in range(self.ylen):
                if i != self.xlen - 1:
                    temp -= self.config[i][j] * self.config[i+1][j]
                if j != self.ylen - 1:
                    temp -= self.config[i][j] * self.config[i][j+1]
```

```
        temp *= (self.xr + self.yr - 1)

        #Applying PBC conditions
        for j in range(self.ylen):
            temp1 -= self.config[-1][j] * self.config[0][j]
        temp1 *= self.xr

        for j in range(self.xlen):
            temp2 -= self.config[j][-1] * self.config[j][0]
        temp2 *= self.yr

        temp += temp1 + temp2
        #Complete energy due to Coupling
        temp*=self.coup

        #Total Energy
        V += temp
        return V


    def MC(self):
        nx = np.random.randint(0,self.xlen)
        ny = np.random.randint(0,self.ylen)

        SumSpin=0

        if nx!=self.xlen-1:
            SumSpin+=self.config[nx+1][ny]
        else:
            SumSpin+=self.config[0][ny]
        if ny!=self.ylen-1:
            SumSpin+=self.config[nx][ny+1]
        else:
            SumSpin+=self.config[nx][0]
        if nx!=0:
            SumSpin+=self.config[nx-1][ny]
        else:
            SumSpin+=self.config[-1][ny]
        if ny!=0:
            SumSpin+=self.config[nx][ny-1]
        else:
            SumSpin+=self.config[nx][-1]

        V = self.config[nx,ny]*(self.mag*self.mu*2 + self.coup*SumSpin)
        #Due to the PBC, multiple changes happen at once
        #Hence Potential =
        V *= self.xr + self.yr -1
        p_acc = np.exp(-1.0/self.kT*V)
        if V < 0 or np.random.random() < p_acc:
            self.config[nx,ny]*=-1
            accept=True
        else:
            accept=False
        self.prob = p_acc
        return accept

    def Store(self,stepSize,totStep,filename):
        f = open(filename, "w")
        fa = open("Config.dle", "w")
        f.write("Step No:Magnetization:Temp:Energy\n")
        fa.write("Config according to step\n")
        Step=0
        while Step < totStep:
            flag = self.MC()
            if flag == True:
                Step+=1
                if Step%stepSize == 0:
                    rem_temp = str(self.config) + "\n\n"
                    fa.write(rem_temp)
                    temp_str = str(Step) + ":" + str(self.magnetization()) + ":" + str(s
elf.kT/scipy.constants.k) + ":" + str(self.energy()) + "\n"
                    f.write(temp_str)
```

```
        f.close()
        fa.close()

    def Heat(self,stepSize,totStep):
        Step=0
        E=0
        U=0
        while Step < totStep:
            flag = self.MC()
            if flag == True:
                Step+=1
                if Step%stepSize == 0:
                    temp=self.energy()
                    E+=temp
                    U+=temp*temp
        totEnergy= U - E*E
        totEnergy/=Step
        C = totEnergy/(self.kT*self.kT)
        return C

    def Mag_Step(self,stepSize,totStep):
        Step=0
        E=0
        U=0
        while Step < totStep:
            flag = self.MC()
            Step+=1
            if Step%stepSize == 0:
                E = self.magnetization()
        E/=Step
        return E

    def equiibration(self,steps):
        x = np.array([])
        y = np.array([])
        i = 0
        while i < steps * self.xlen * self.ylen * (self.xr + self.yr - 1):
            flag = modObj.MC()
            val = modObj.energy()
            x=np.append(x,val)
            y = np.append(y,i)
            i+=1
        return [x,y]
```

In [3]:

```
modObj = Ising(50,50,0)
print(modObj)
modObj.mag = 0
modObj.energy()
```

```
[[1 1 1 ... 1 1 1]
 [1 1 1 ... 1 1 1]
 [1 1 1 ... 1 1 1]
 ...
 [1 1 1 ... 1 1 1]
 [1 1 1 ... 1 1 1]
 [1 1 1 ... 1 1 1]]
```

Out[3]:

```
-14900.0
```

Plot of Energy vs Temperature

In [6]:

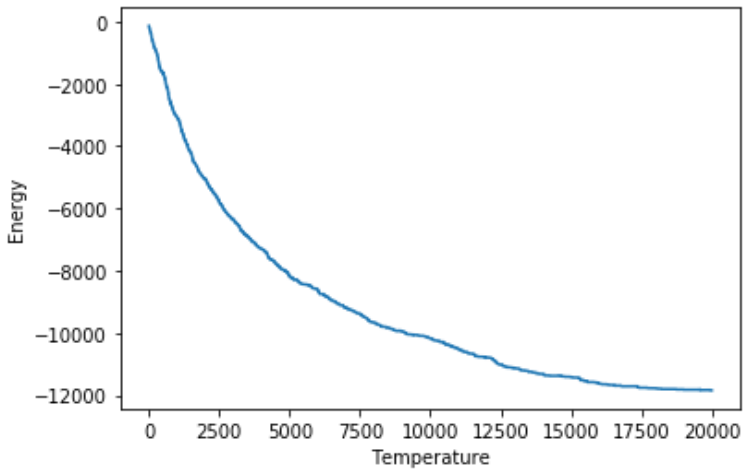
```
step = 100
totStep = 2000
modObj.kT = 1
modObj.random()
```

```
x = np.array([])
y = np.array([])
i = 0
while i < 20000:
    flag = modObj.MC()
    val = modObj.energy()
    x=np.append(x,val)
    y = np.append(y,i)
    i+=1


plt.xlabel('Temperature')
plt.ylabel('Energy')
plt.plot(y,x)
```

Out[6]:

```
[<matplotlib.lines.Line2D at 0x7f3247ac7a10>]
```



Plot of Magnetization vs Temperature

in the following code we shall plot m vs k_bTT is plotted

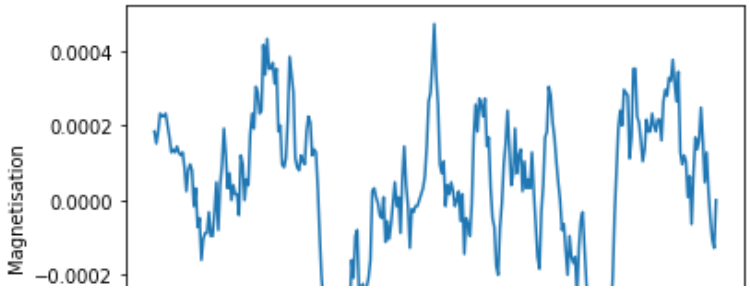
In [7]:

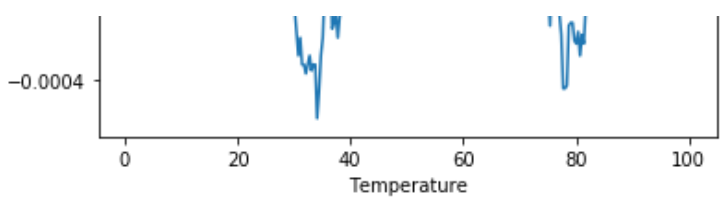
```
modObj.mag = 0
step = 100
totStep = 100
finTemp = 100
minTemp = 0.5
x = np.array([])
y = np.array([])
for i in np.linspace(minTemp,finTemp,300):
    modObj.kT = i
    val = modObj.Mag_Step(step,totStep)
    x=np.append(x,val)
    y = np.append(y,i)
plt.xlabel('Temperature')
plt.ylabel('Magnetisation')

plt.plot(y,x)
```

Out[7]:

```
[<matplotlib.lines.Line2D at 0x7f3247a3f610>]
```





In [ ]: