```
            Psi2[k]=list(item)
            Psi2[k][l-1]='1'
            Psi2[k]="".join(Psi2[k])
        if item[l-1]=='1':
            Psi2[k]=list(item)
            Psi2[k][l-1]='0'
            Psi2[k]="".join(Psi2[k])
    return Psi2

def construct_MPDW(Psi,L,W):
    if args.W != 0.0:
        a = 2*W * np.random.random_sample(L) - W #mu in [-W,W]
    else:
        a = np.zeros(L)
    C = np.zeros(len(Psi))
    for i in range(len(Psi)):
        item=Psi[i]
        for j in range(len(item)):
            if item[j]=='1':
                C[i]+=a[j]
    A = np.diag(C)
    return A

def findMagnetizationStates(length,particlenumber):
    """
    constructs a table with the integer representations of all binaries
    with a given number of 1s
    """
    s = np.arange(2**length)
    bitcount = np.array([bin(x).count("1") for x in s])
    return s.compress(bitcount==particlenumber)

def bit(state,j,length):
    """return value of bit j"""
    return state >> (length-1-j) & 1
def bitFlip(state,j,k,length):
    """flip bits j and k of state if they are unequal"""
    return state ^ (2**(length-1-j)+2**(length-1-k))

# construct many-particle Hamiltonian for clean SSH model
def construct_MPH(V,length,table):
    """construct clean Hamiltonian"""
    nos = len(table)
    h = np.zeros((nos,nos),np.float)
    for b,t in enumerate(table): # loop over eigenstates
        for j in range(length-args.openbc): # loop over sites
            k = (j+1)%length # right neighboring site
            # Heisenberg interaction for equal nearest neighbors
            if bit(t,j,length)==bit(t,k,length):
                h[b,b]+=0.25*V
            # Heisenberg interaction + tunneling for unequal nearest neighbors
            else:
                h[b,b]-=0.25*V
                bp = findIndex(table, bitFlip(t, j, k, length))
                #bp = findIndex(table, bitFlip(t, j, k, length))-1
                h[b,bp]=0.5
    return h
```