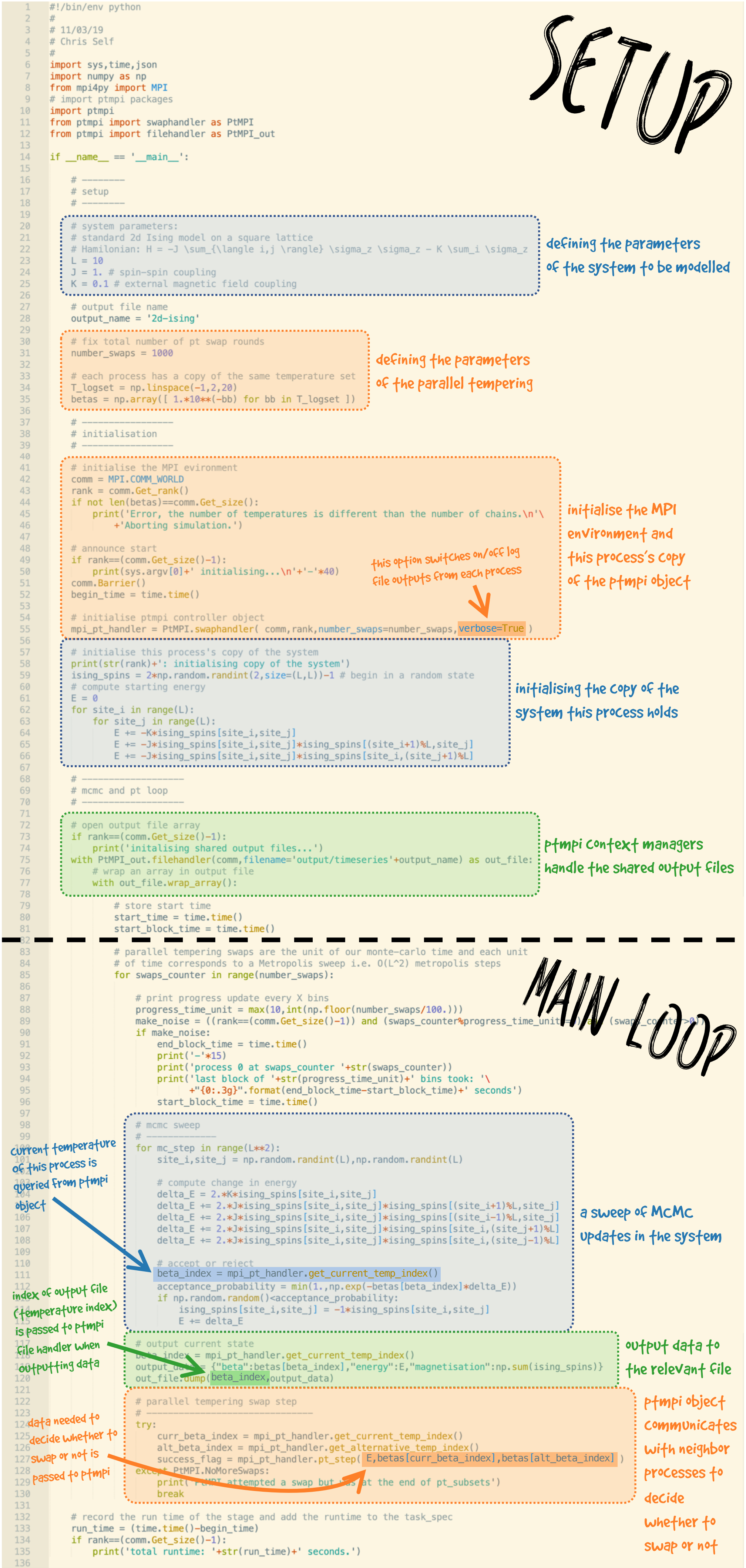

SETUP

```
#!/bin/env python
#
# 11/03/19
# Chris Self
#
import sys,time,json
import numpy as np
from mpi4py import MPI
# import ptmpi packages
import ptmpi
from ptmpi import swaphandler as PtMPI
from ptmpi import filehandler as PtMPI_out

if __name__ == '__main__':

    # --------
    # setup
    # --------

    # system parameters:
    # standard 2d Ising model on a square lattice
    # Hamilonian: H = -J \sum_{\langle i,j \rangle} \sigma_z \sigma_z - K \sum_i \sigma_z
    L = 10
    J = 1. # spin-spin coupling
    K = 0.1 # external magnetic field coupling

    # output file name
    output_name = '2d-ising'

    # fix total number of pt swap rounds
    number_swaps = 1000

    # each process has a copy of the same temperature set
    T_logset = np.linspace(-1,2,20)
    betas = np.array([ 1.*10**(-bb) for bb in T_logset ])

    # ----------------
    # initialisation
    # ----------------

    # initialise the MPI evironment
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    if not len(betas)==comm.Get_size():
        print('Error, the number of temperatures is different than the number of chains.\n'\
            +'Aborting simulation.')

    # announce start
    if rank==(comm.Get_size()-1):
        print(sys.argv[0]+' initialising...\n'+'-'*40)
    comm.Barrier()
    begin_time = time.time()

    # initialise ptmpi controller object
    mpi_pt_handler = PtMPI.swaphandler( comm,rank,number_swaps=number_swaps,verbose=True )

    # initialise this process's copy of the system
    print(str(rank)+': initialising copy of the system')
    ising_spins = 2*np.random.randint(2,size=(L,L))-1 # begin in a random state
    # compute starting energy
    E = 0
    for site_i in range(L):
        for site_j in range(L):
            E += -K*ising_spins[site_i,site_j]
            E += -J*ising_spins[site_i,site_j]*ising_spins[(site_i+1)%L,site_j]
            E += -J*ising_spins[site_i,site_j]*ising_spins[site_i,(site_j+1)%L]

    # ------------------
    # mcmc and pt loop
    # ------------------

    # open output file array
    if rank==(comm.Get_size()-1):
        print('initalising shared output files...')
    with PtMPI_out.filehandler(comm,filename='output/timeseries'+output_name) as out_file:
        # wrap an array in output file
        with out_file.wrap_array():

            # store start time
            start_time = time.time()
            start_block_time = time.time()
```

defining the parameters of the system to be modelled

defining the parameters of the parallel tempering

initialise the MPI environment and this process's copy of the ptmpi object

this option switches on/off log file outputs from each process

initialising the copy of the system this process holds

ptmpi context managers handle the shared output files

MAIN LOOP

```
            # parallel tempering swaps are the unit of our monte-carlo time and each unit
            # of time corresponds to a Metropolis sweep i.e. O(L^2) metropolis steps
            for swaps_counter in range(number_swaps):

                # print progress update every X bins
                progress_time_unit = max(10,int(np.floor(number_swaps/100.)))
                make_noise = ((rank==(comm.Get_size()-1)) and (swaps_counter%progress_time_unit==0) and (swaps_counter>0))
                if make_noise:
                    end_block_time = time.time()
                    print('-'*15)
                    print('process 0 at swaps_counter '+str(swaps_counter))
                    print('last block of '+str(progress_time_unit)+' bins took: '\
                          +"{0:.3g}".format(end_block_time-start_block_time)+' seconds')
                    start_block_time = time.time()

                # mcmc sweep
                # -----------
                for mc_step in range(L**2):
                    site_i,site_j = np.random.randint(L),np.random.randint(L)

                    # compute change in energy
                    delta_E = 2.*K*ising_spins[site_i,site_j]
                    delta_E += 2.*J*ising_spins[site_i,site_j]*ising_spins[(site_i+1)%L,site_j]
                    delta_E += 2.*J*ising_spins[site_i,site_j]*ising_spins[(site_i-1)%L,site_j]
                    delta_E += 2.*J*ising_spins[site_i,site_j]*ising_spins[site_i,(site_j+1)%L]
                    delta_E += 2.*J*ising_spins[site_i,site_j]*ising_spins[site_i,(site_j-1)%L]

                    # accept or reject
                    beta_index = mpi_pt_handler.get_current_temp_index()
                    acceptance_probability = min(1.,np.exp(-betas[beta_index]*delta_E))
                    if np.random.random()<acceptance_probability:
                        ising_spins[site_i,site_j] = -1*ising_spins[site_i,site_j]
                        E += delta_E

                # output current state
                beta_index = mpi_pt_handler.get_current_temp_index()
                output_data = {"beta":betas[beta_index],"energy":E,"magnetisation":np.sum(ising_spins)}
                out_file.dump(beta_index,output_data)

                # parallel tempering swap step
                # ------------------------------
                try:
                    curr_beta_index = mpi_pt_handler.get_current_temp_index()
                    alt_beta_index = mpi_pt_handler.get_alternative_temp_index()
                    success_flag = mpi_pt_handler.pt_step( E,betas[curr_beta_index],betas[alt_beta_index] )
                except PtMPI.NoMoreSwaps:
                    print('PtMPI attempted a swap but was at the end of pt_subsets')
                    break

    # record the run time of the stage and add the runtime to the task_spec
    run_time = (time.time()-begin_time)
    if rank==(comm.Get_size()-1):
        print('total runtime: '+str(run_time)+' seconds.')
```

current temperature of this process is queried from ptmpi object

a sweep of McMC updates in the system

index of output file (temperature index) is passed to ptmpi file handler when outputting data

output data to the relevant file

data needed to decide whether to swap or not is passed to ptmpi

ptmpi object communicates with neighbor processes to decide whether to swap or not