2/11/2020 Week 5 Module 1
Interactive Exercise for Molecular Dynamics

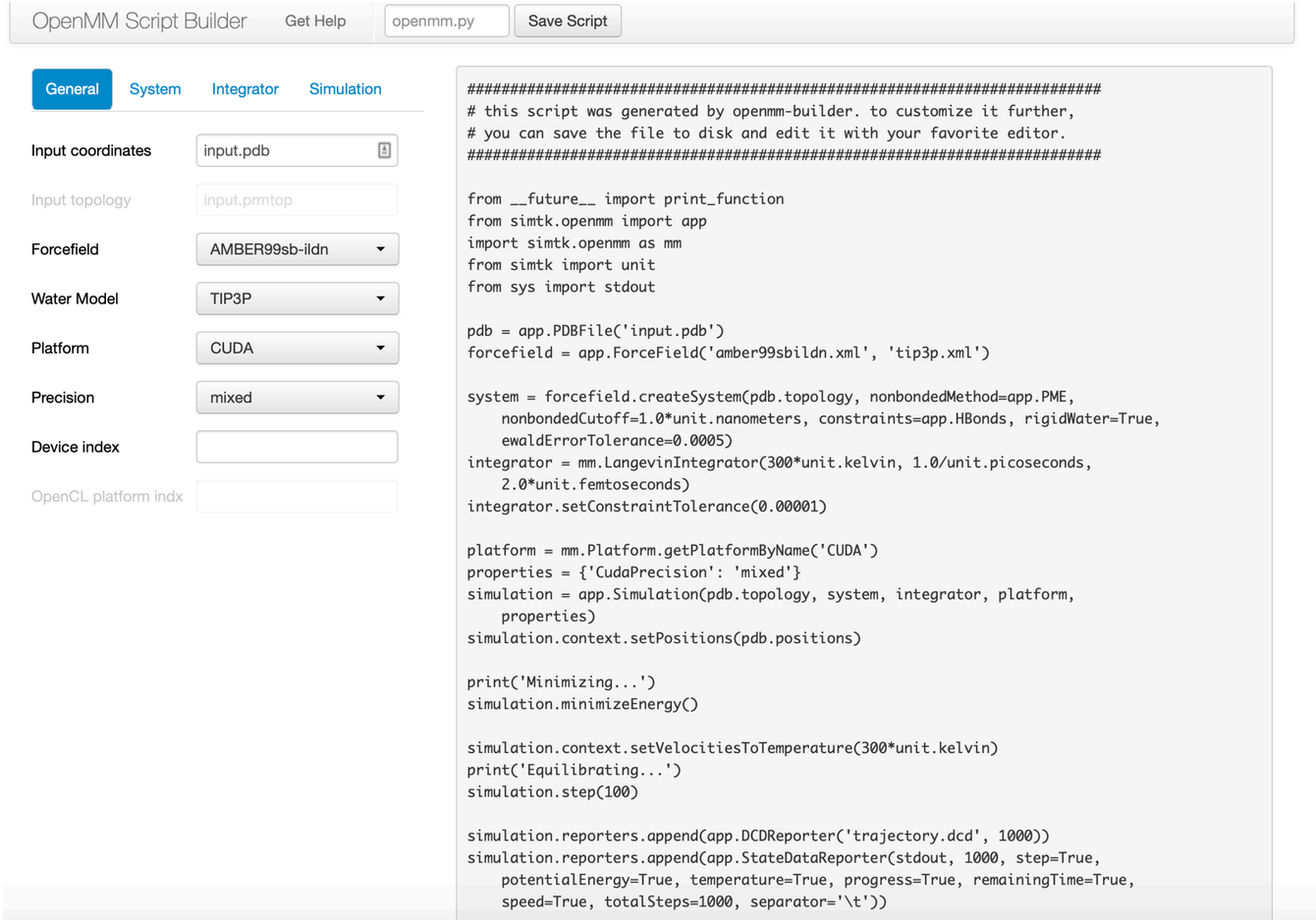
- In this module, you will learn how to prepare a python script to run OpenMM using the OpenMM script builder (http://builder.openmm.org)
  - navigate the UNIX terminal
- In the next class session, you will learn how to
  - execute programs with a command line interface (CLI)
  - run a short MD simulation on your desktop machine
  - visualize the simulation with VMD

Preparing a script to run MD

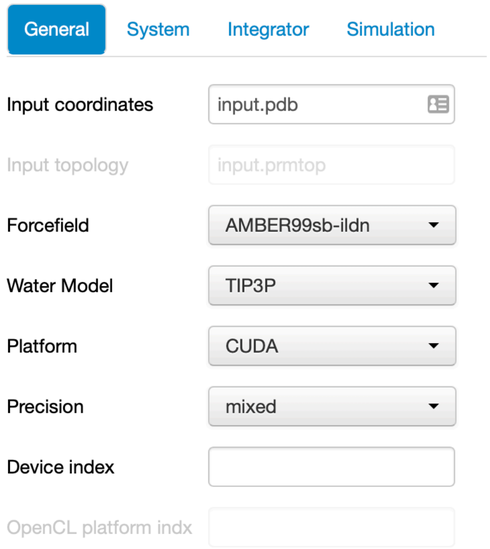
Why write a script?

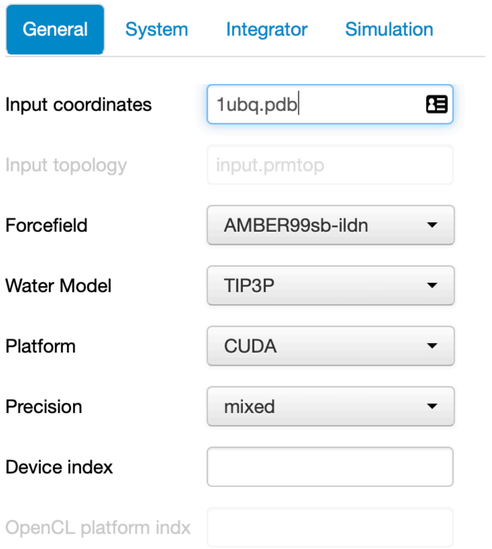
- Molecular dynamics simulations require a lot of information about
  - Input data
    - coordinates
    - topology
      - which atoms are included in energy terms
      - parameters for functions in energy terms
  - System description
    - periodicity
    - constraints
  - Integrators
    - algorithms to propagate forward in time
    - adjust box size
    - adjust kinetic energy (temperature)
  - Simulation
    - how long to run
    - how much output data to store
- OpenMM can be run from widely-used computer programming languages, python and C++, facilitating extension and combination with other code

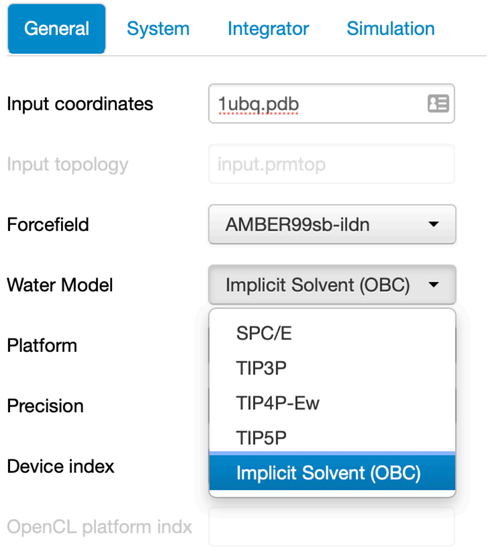
- This is the default script in the OpenMM script builder: http://builder.openmm.org
- Let's go through what the parameters mean and I'll give you values for a simple simulation of ubiquitin, a protein that we looked at during the VMD tutorial
- If you hover your mouse over the parameter names, it'll also give a brief description
- A more detailed description of simulation parameters is available at http://docs.openmm.org/latest/userguide/application.html#simulation-parameters

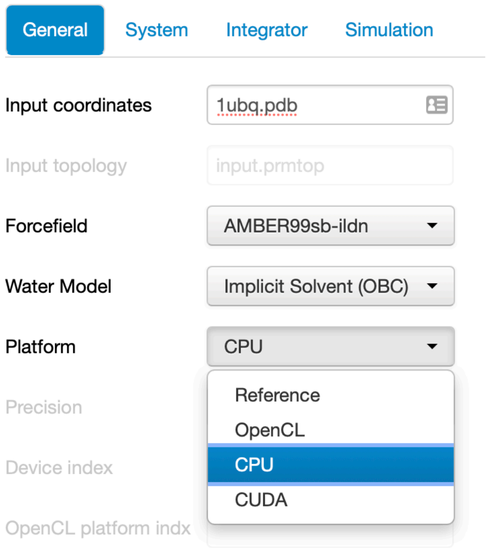


- For “Input coordinates”, enter 1ubq.pdb
- “Forcefield” is the set of parameters and functions that describes the energy of a system. Let’s keep “AMBER99sb-ildn”
- “Water Model” includes
  - descriptions that vary in the number of point charges
  - a description that doesn’t actually model water at all, but its effect on the electrostatic energy and a penalty for forming surfaces. For computational speed, let’s pick this “Implicit Solvent” version
- “Platform” describes the version of the code and the hardware it will run on
  - “Reference” is meant to be the most readable code
  - “CUDA” and “OpenCL” are meant for GPUs, which make MD simulations much faster. “CUDA” only works with Nvidia GPUs and “OpenCL” on others
  - “CPU” is a faster version of Reference.
  - Since the laboratory machines don’t have GPUs, let’s use “CPU”
- The other options are GPU-specific

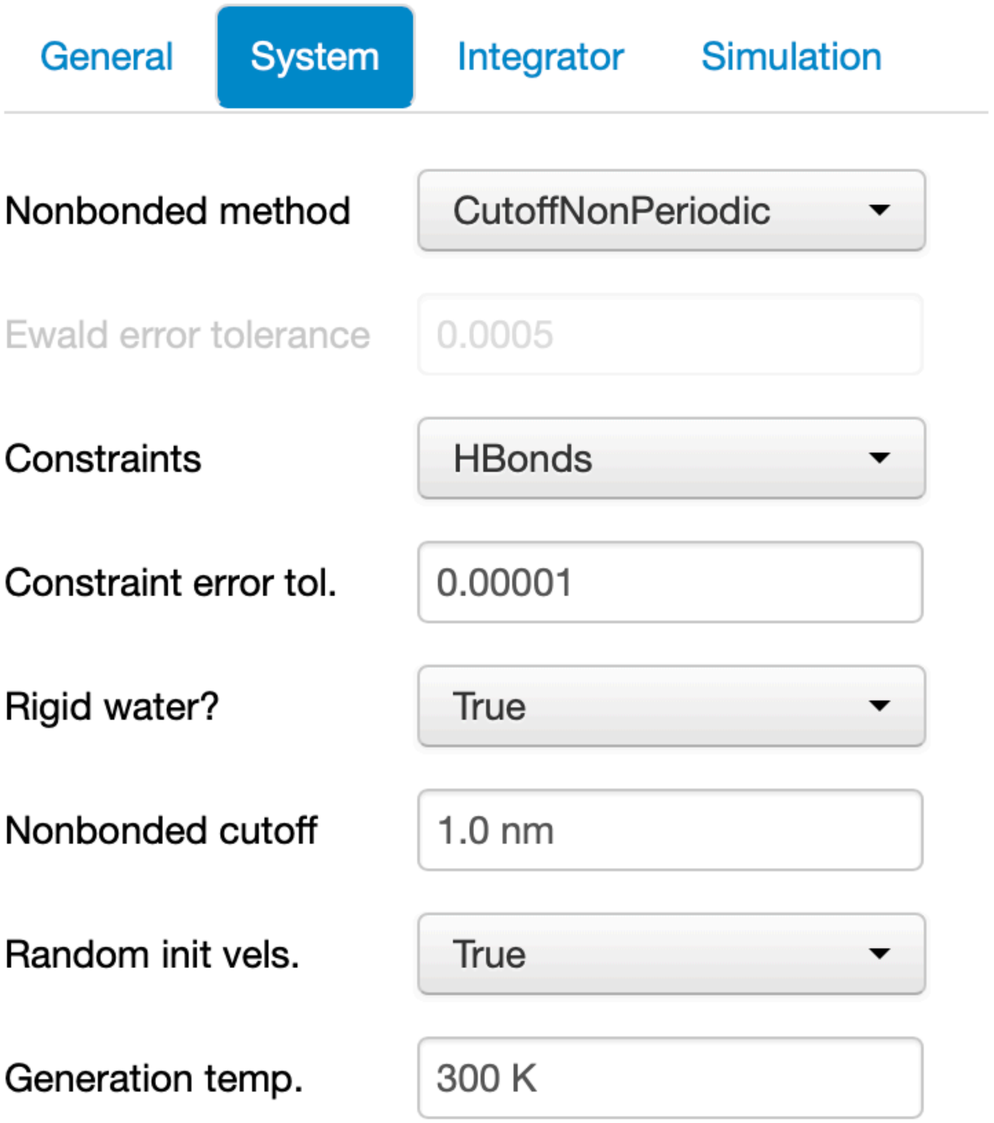




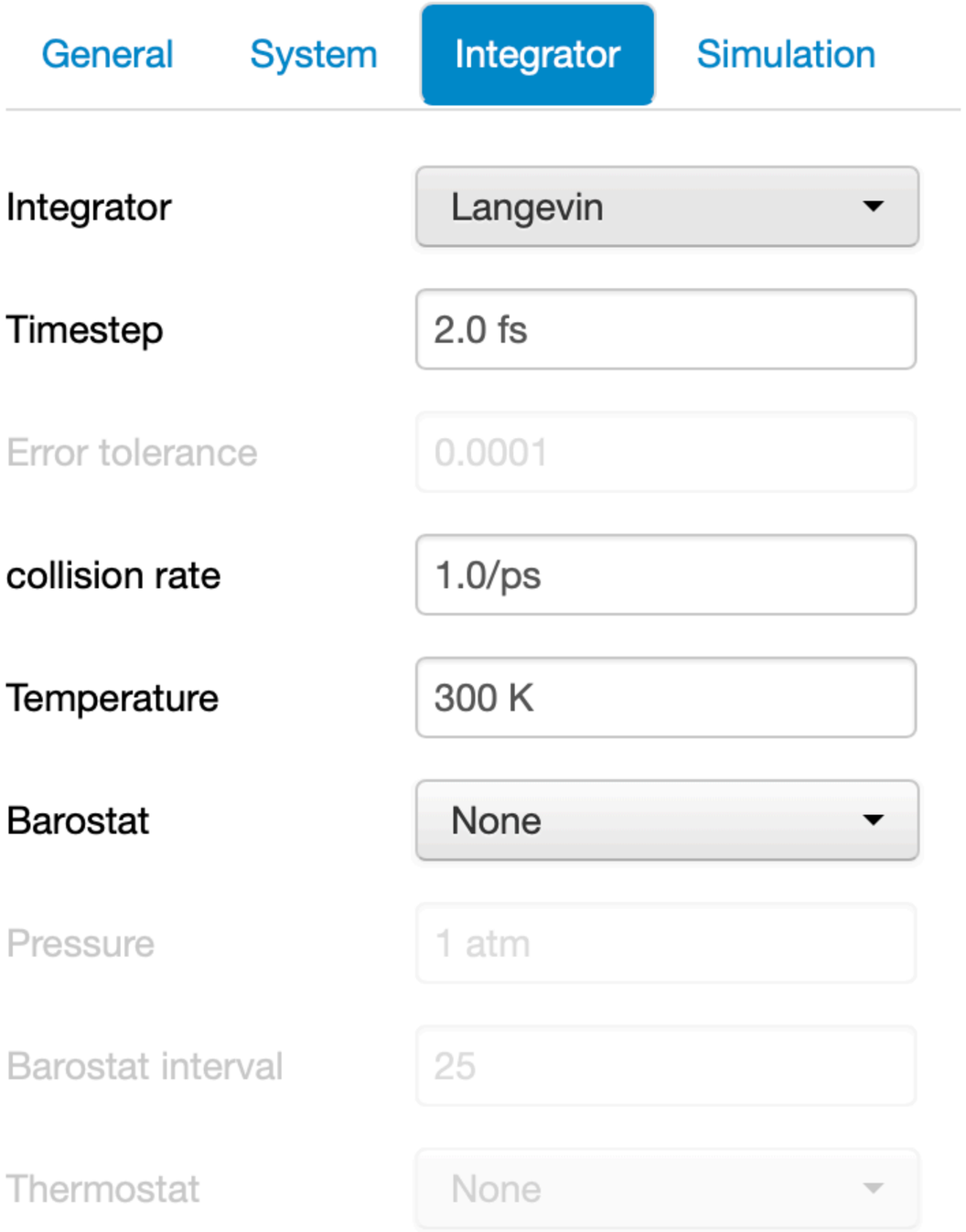




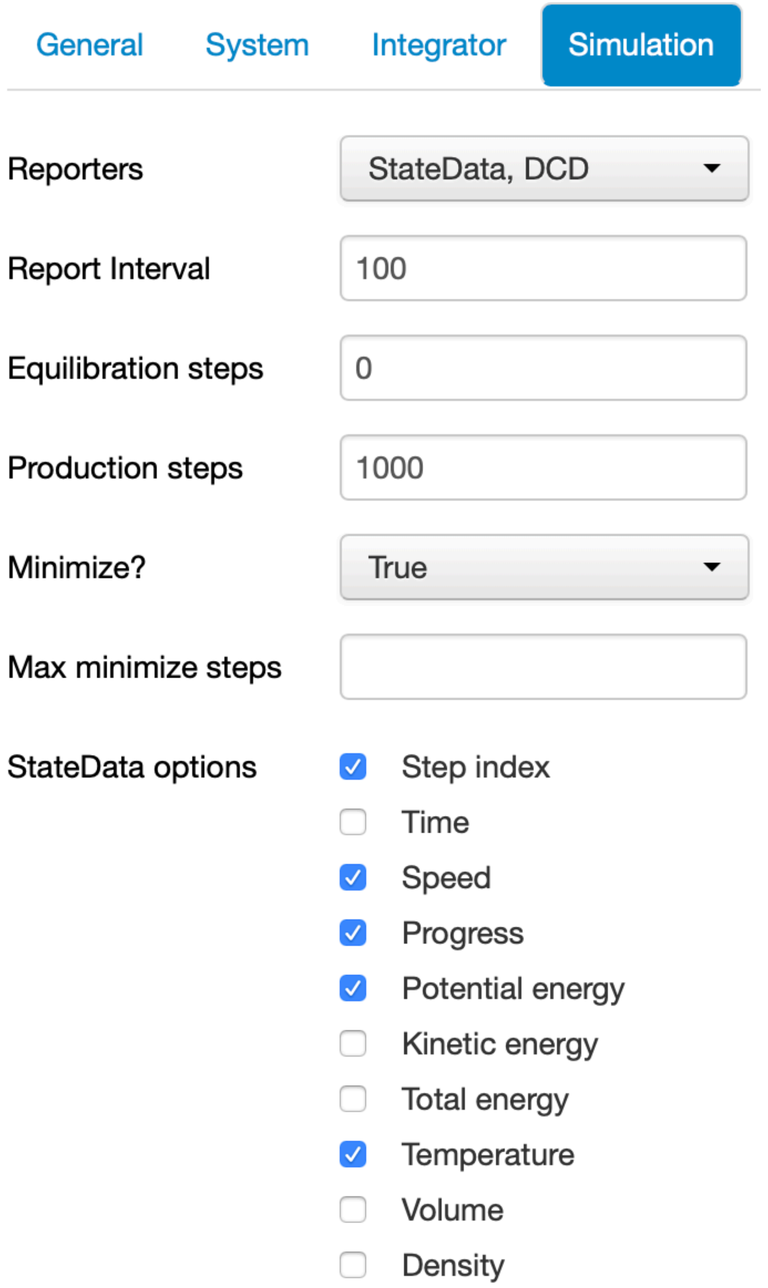
- “Nonbonded method” describes how long-range interactions are treated. The more interactions that need to be computed, the slower an MD simulation will be.
  - Cutoffs don’t perform calculations if two particles are beyond a certain distance apart.
  - Ewald methods calculate long-range interaction energies between a system and its periodic images. It assumes the system repeats indefinitely.
  - Since we are using implicit solvent, we don’t need periodicity.
  - Let’s use “CutoffNonPeriodic”.
- “Constraints”
  - force a degree of freedom to be a certain value
  - allow a larger time step, giving you more bang (simulation time) for the buck (compute time)
- Let’s keep the other “System” parameters as is



- “Integrator” is the algorithm that goes from one configuration to the next
  - Verlet is completely deterministic
  - Langevin adds some random noise to the motion. The level of noise maintains the system at a certain temperature.
  - Brownian is so random that there is no momentum
  - Variable methods use different time steps and depend on an error tolerance
  - Let’s use Langevin and keep default values of other parameters
- “Barostat”
  - allows the volume of the system to change
  - keeps the system at a certain pressure
  - Since we are using implicit water, let’s not use a barostat

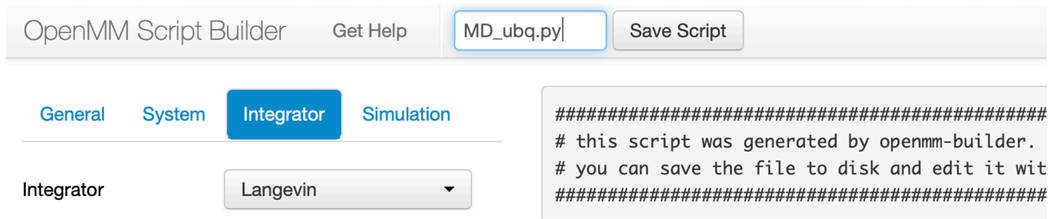


- “Reporters” store data about the simulation
  - “StateData” gives various options listed in the check boxes
  - “DCD” is a binary file format for molecular dynamics trajectories
- “Report Interval” is how often the data are stored
- “Equilibration” is the number of steps before data is stored
- “Production” is the number of steps the simulation is run
- “Minimize” will minimize the energy before running the simulation.
- Let’s set the options as shown on the right



- Your script from the OpenMM script builder should look like the window on the right
- Create a directory called “OpenMM” on the Desktop.
- Save the script as “MD_ubq.py” and move it into the “OpenMM” directory.
- Copy “1ubq.pdb” from the VMD tutorial into the same “OpenMM” directory. The file can also be found on github.

```
#############################################################################
# this script was generated by openmm-builder. to customize it further,
# you can save the file to disk and edit it with your favorite editor.
#############################################################################

from __future__ import print_function
from simtk.openmm import app
import simtk.openmm as mm
from simtk import unit
from sys import stdout

pdb = app.PDBFile('1ubq.pdb')
forcefield = app.ForceField('amber99sbildn.xml', 'amber99_obc.xml')

system = forcefield.createSystem(pdb.topology,
    nonbondedMethod=app.CutoffNonPeriodic, nonbondedCutoff=1.0*unit.nanometers,
    constraints=app.HBonds, rigidWater=True)
integrator = mm.LangevinIntegrator(300*unit.kelvin, 1.0/unit.picoseconds,
    2.0*unit.femtoseconds)
integrator.setConstraintTolerance(0.00001)

platform = mm.Platform.getPlatformByName('CPU')
simulation = app.Simulation(pdb.topology, system, integrator, platform)
simulation.context.setPositions(pdb.positions)

print('Minimizing...')
simulation.minimizeEnergy()

simulation.context.setVelocitiesToTemperature(300*unit.kelvin)
simulation.reporters.append(app.DCDReporter('trajectory.dcd', 100))
simulation.reporters.append(app.StateDataReporter(stdout, 100, step=True,
    time=True, potentialEnergy=True, temperature=True, progress=True,
    remainingTime=True, speed=True, totalSteps=1000, separator='\t'))

print('Running Production...')
simulation.step(1000)
print('Done!')
```



- We are not completely ready for the simulation because 1ubq.pdb
  - does not have hydrogen molecules
  - includes water, which does not help when we have an implicit solvent model
- OpenMM provides a package that does some simple modeling, including these required tasks.
- To make our script use this capability, open a text editor and add the five lines shown in the box

```
pdb = app.PDBFile('1ubq.pdb')
forcefield = app.ForceField('amber99sbildn.xml', 'amber99_obc.xml')

modeller = app.Modeller(pdb.topology, pdb.positions)
modeller.deleteWater()
modeller.addHydrogens(forcefield)
pdb = modeller
app.PDBFile.writeFile(pdb.topology, pdb.positions, open('ubq_mod.pdb', 'w'))

system = forcefield.createSystem(pdb.topology,
    nonbondedMethod=app.CutoffNonPeriodic, nonbondedCutoff=1.0*unit.nanometers,
    constraints=app.HBonds, rigidWater=True)
integrator = mm.LangevinIntegrator(300*unit.kelvin, 1.0/unit.picoseconds,
```

The complete script can also be found on github.

Additional Resources

- In “Introduction to OpenMM” on http://openmm.org/documentation.html, Peter Eastman describes an OpenMM script from 5:00 to 15:03