Brian Reference

Units		

Names	`volt, amp, siemens, ...`
	`Mohm, mvolt, umetre, ...`
	`mV, mS, um, cm2, ...`	

Arithmetic	`(2*amp)*(3*ohm) == 6*volt` `(2*amp)+(3*ohm)` raises `DimensionMismatchError`	
IPython		
Run script	`%run example.py`	
Shell	`cd examples` `ls`	
Help	`NeuronGroup?` `help(Connection)`	
Source	`NeuronGroup??`	
Equations		

Equations object	`eqs = Equations('''`
	`dV/dt=-V/(10*ms) : volt`
	`''')`	

Syntax	`dV/dt = -(V-V0)/(10*ms) : volt`
	`I = I0+I1 : amp`
	`u = V`
	`V0 : volt`
		Differential eqn.
		Equation
		Alias
		Parameter

Noise	`dV/dt = xi*sigma/tau**0.5 : volt`	
NeuronGroup		
Creation	`G = NeuronGroup(100, model=eqns, threshold=10*mV, reset=0*mV)` `G = NeuronGroup(100, model=eqns, threshold='v>=vt', reset='v=vr', refractory=5*ms)`	
State variables	`G.V = Vr+(Vt-Vr)*rand(len(G))`	
Subgroups	`Ge = G[0:50]`	
Connection		
Creation	`C = Connection(P, Q, 'V')`	Connection from group P to state variable V of group Q.

Weights	`C.connect_full(weight=2*mV)`
	`C.connect_random(sparseness=0.1, weight=lambda i,j: exp(-abs(i-j)*.1)*3*mV)`
	`C.connect(W=eye(len(P),len(Q)))`
	`C.connect_full(Pe, Q, weight=2*mV)`
	`C[0,3]=4*mV`	

Construction by block	`Pe = P[0:50]`
	`Pi = P[50:100]`
	`C.connect_full(Pe, Pi, weight=1*mV)`	

One line definition	`C = Connection(P, Q, 'V', weight=1*mV, sparseness=0.1)`	

Delays	`C = Connection(P, Q, 'V', delay=2*ms)`
	`C = Connection(P, Q, 'V', delay=True)`
	`C.delay[1,3]=3*ms`
		Homogeneous delays
		Heterogeneous delays

Spike-timing dependent plasticity		

General STDP	`C = Connection(P, Q, 'ge')`
	`eqs_stdp='''`
	`dA_pre/dt=-A_pre/tau_pre : 1`
	`dA_post/dt=-A_post/tau_post : 1`
	`'''`
	`stdp=STDP(C, eqs=eqs, pre='A_pre+=dA_pre;w+=A_post', post='A_post+=dA_post;w+=A_pre',wmax=gmax)`
		Dynamics of synaptic variables
		
		`pre` (`post`) is executed for each presynaptic (postsynaptic) spike.

Exponential STDP	`stdp=ExponentialSTDP(synapses, taup, taum, Ap, Am, wmax=gmax, interactions='all', update='additive')`	Interactions can be 'all', 'nearest', 'nearest_post', 'nearest_pre'; update can be 'additive', 'multiplicative', 'mixed'.
	Short-term plasticity	
Creation	`C = Connection(P, Q, 'ge')` `stp = STP(C, taud=100*ms, tauf=5*ms, U=.6)`	
	SpikeMonitor	
Creation	`M = SpikeMonitor(G)`	
Attributes	`M.nspikes` `M.spikes`	M.spikes is a list of pairs (i,t) for neuron i at time t.

	`def f(spikes):`
	`  print spikes`
	`M = SpikeMonitor(G, function=f)`
		Custom monitoring.

	StateMonitor	
Creation	`M = StateMonitor(G, 'V')`	Just record summary stats
	`M = StateMonitor(G, 'V', record=5)`	Record neuron 5
	`M = StateMonitor(G, 'V', record=[5,6,7])`	Record neurons 5, 6, 7
	`M = StateMonitor(G, 'V', record=True)`	Record all neurons
Attributes	`M.times`	Recording times (array of length nsteps)
	`M.mean, M.var, M.std`	Summary statistics (array of length num neurons)
	`M[i]`	Recorded values of neuron i (length nsteps)
	PopulationRateMonitor	
Creation	`M = PopulationRateMonitor(G)` `M = PopulationRateMonitor(G, bin=5*ms)`	
Attributes	`M.times` `M.rates`	
Methods	`M.smooth_rate(width=10*ms)` `M.smooth_rate(width=10*ms, filter='flat')`	
	Network	
Running	`run(10*second)`	Uses all defined objects from the current file or function, for finer control use the Network object

Operations	`@network_operation`
	`def f():`
	`  ...`
		Calls the function f every update step

Network object	`net = Network(G, C, f)` `net.run(10*second)`	To specify exactly which objects should be used.
	Plotting	

Raster plot	`raster_plot()`
	`raster_plot(M)`
	`raster_plot(M1, M2)`	

Graphs	`plot(M.times/ms, M[i]/mV)`	See pylab docs for details
Connection strengths	`imshow(C.W.todense(), interpolation='nearest', origin='lower')`	C.W.todense() is the connection matrix
Showing	`show()`	

Built-in NeuronGroup types		

Poisson group	`P=PoissonGroup(N, rates)`
	`P=PoissonGroup(100, 10*Hz)`
	`P=PoissonGroup(3, array([1,2,3])*Hz)`
	`P=PoissonGroup(1, lambda t : (1+sin(t))*50*Hz)`
		rates can be a value, array or function of time returning a value or array, all in Hz

Pulse packet	`P=PulsePacket(t, n, sigma)` `P=PulsePacket(50*ms, 100, 5*ms)`	
Spike generator	`P=SpikeGeneratorGroup(N, spiketimes)` `P=SpikeGeneratorGroup(2,[(0,1*ms), (1,3*ms), (0,5*ms)])`	spiketimes should be a list of pairs (i,t) for neuron i firing at time t

For more information, see the online manual:

http://www.briansimulator.org/docs