cadCAD Edu Cheat Sheet: Standard Notebook Layout (SNL)

① State Variables

```python
initial_state = {
  # State Variable key-value pair:
  # Key can be any text/string
  # Value can be any Python type
  'a': 0,
  'b': 1,
  'c': 2,
  ...
}
```

② System Parameters

```python
system_params = {
  # System Parameter key-value pair:
  # Key can be any text/string
  # Value can be a list of any Python type
  'a': [0],
  'b': [1],
  'c': [2],
  ...
}
```

③ Policy Functions

```python
def p_policy_name(params, substep, state_history, previous_state):
  '''
  Args:
    params (dict): Python dictionary containing the system parameters
    substep (int): integer value representing a step within a single timestep
    state_history (list): Python list of all previous states
    previous_state (dict): Python dictionary that defines what the v
      state of the system was at the previous timestep or substep
  Returns:
    dict: key as signal name, and value as any Python type
  '''
  # Logic to generate value to be passed to State Update Function(s)
  # as policy_input
  signal_value = ...
  return {'signal_name', signal_value}
```

④ State Update Functions

```python
def s_state_variable_name(params, substep, state_history,
    previous_state, policy_input):
  '''
  Args:
    Same arguments as Policy Function, as well as:
    policy_input (dict): Python dictionary of signals or actions
      from Policy Functions
  Returns:
    tuple: key as State Variable name, and value as any Python type
  '''
  # The updated value of the State Variable
  state_variable_value = ...
  return 'state_variable_name', state_variable_value
```

⑤ Partial State Update Blocks (PSUBs)

```python
# This entire structure represents one timestep
partial_state_update_blocks = [
  # First Partial State Update Block, and one substep
  {
    # Policy Functions
    'policies': {
      # Any text/string as the key,
      # any valid Policy Function as the value
      'policy_name': p_policy_name,
      ...
    },
    # State Update Functions
    'variables': {
      # State Variable and State Update Function key-value pair:
      # Any valid State Variable name as the key,
      # any valid State Update Function as the value
      'state_variable_name': s_state_variable_name,
      ...
    }
  },
  # Additional PSUBs / substeps
  ...
]
```

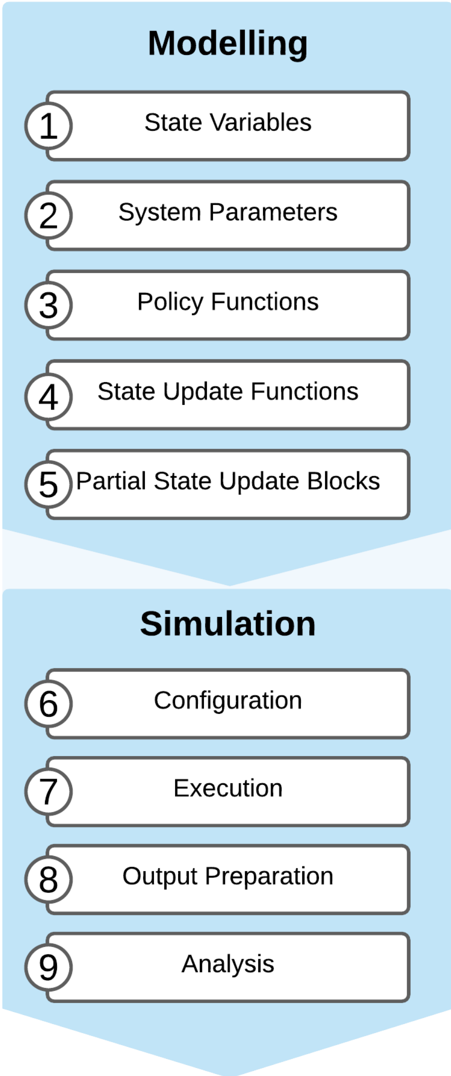
Standard Dependencies

```python
# cadCAD configuration modules
from cadCAD.configuration.utils import config_sim
from cadCAD.configuration import Experiment

# cadCAD simulation engine modules
from cadCAD.engine import ExecutionMode, ExecutionContext
from cadCAD.engine import Executor

# cadCAD global simulation configuration list
from cadCAD import configs

# Included with cadCAD
import pandas as pd
```



⑥ Configuration

```python
del configs[:] # Clears any prior configs

sim_config = config_sim({
  'N': 1, # Number of Monte Carlo Runs
  'T': range(100), # Number of timesteps
  'M': system_params # System Parameters
})

experiment.append_configs(
  # Model initial state
  initial_state=initial_state,
  # Model Partial State Update Blocks
  partial_state_update_blocks=partial_state_update_blocks,
  # Simulation configuration
  sim_configs=sim_config
)
```

⑦ Execution

```python
# ExecutionContext instance (used for more advanced cadCAD config)
exec_context = ExecutionContext()

# Creates a simulation Executor instance
simulation = Executor(
  exec_context=exec_context,
  # cadCAD configuration list
  configs=configs
)

# Executes the simulation, and returns the raw results
raw_result, tensor_field, sessions = simulation.execute()
```

⑧ Output Preparation

```python
# Convert cadCAD raw results from list of dictionaries
# to Pandas DataFrame format
simulation_result = pd.DataFrame(raw_result)
# Display first 5 rows (head) of DataFrame
simulation_result.head()
```

Illustrative simulation result with 1 **timestep**, 1 Monte Carlo **run**, 2 Partial State Update Blocks **(substep)**, and a Parameter Sweep of 2 parameters **(subset)**:

	state_variable_name_1	state_variable_name_2	simulation	subset	run	substep	timestep
0	0	0	0	0	1	0	0
1	1	0	0	0	1	1	1
2	1	1	0	0	1	2	1
3	0	0	0	1	1	0	0
4	1	0	0	1	1	1	1
5	1	1	0	1	1	2	1

Note that **timestep 0** and **substep 0** correspond to the “initial state”, and **simulation** is incremented for an A/B test of more than one model configuration.

```python
# Selects the first simulation and subset
simulation_result.query('simulation == 0 and subset == 0')
# Selects the rows where state is greater than zero
simulation_result.query('state_variable_name_1 > 0')
# Selects the state 'state_variable_name' column
simulation_result['state_variable_name_1']
```

⑨ Analysis

```python
# Sets the Pandas plotting backend to use Plotly
pd.options.plotting.backend = "plotly"

# Plot state on the y-axis and the timesteps on the x-axis
simulation_result.plot(
  kind='line',
  x='timestep',
  y=['state_variable_name']
)
```

Version 1.0