Pyomo Style Guide

This style guide supports the development of consistent, readable, and maintainable Pyomo models. These guidelines supplement standard Python style guides conventions, such as PEP 8, with specific recommendations for Pyomo. Comments and suggestions are welcome.

Pyomo Coding

Use `pyo` for the Pyomo namespace

For consistency with Pyomo documentation and the Pyomo book, the preferred namespace convention for Pyomo is `pyo`

```
import pyomo.environ as pyo
```

Use `ConcreteModel` instead of `AbstractModel`

Pyomo provides two methods for creating model instances, `AbstractModel` or `ConcreteModel` as described in the documentation. `AbstractModel` specifies a model with symbolic parameters which can be specified later to define a specific model instance. A `ConcreteModel` requires parameter values to be specified when the model is specified.

Because Pyomo is embedded within Python, the preferred to method for creating model instances is a Python function or class that accepts parameter values and returns a `ConcreteModel` model instance.

Prefer short model and block names

Model and block names should be consistent with PEP 8 naming standards. For readability and to avoid excessively long lines, short model names are preferred. A single lower case `m` is acceptable in most instances of a model with a single block.

Variable and Parameter names

Models derived directly from a mathematical formulations in documentation accompanying the Pyomo model can use the same variable and parameter name. For example, a mathematical model written as

$$f = \max_{x,y} \quad 3x + 4y$$

$$\text{subject to}$$

$$2x + y \leq 10$$
$$x + 2y \leq 15$$

which can be encoded

```
import pyomo.environ as pyo

# create model instance
m = pyo.ConcreteModel()

# decision variables
m.x = pyo.Var(domain=pyo.NonNegativeReals)
m.y = pyo.Var(domain=pyo.NonNegativeReals)

# objective
m.f = pyo.Objective(expr = 40*m.x + 30*m.y, sense=pyo.maximize)

# declare constraints
m.a = pyo.Constraint(expr = 2*m.x + m.y <= 10)
m.b = pyo.Constraint(expr = m.x + 2*m.y <= 15)

m.pprint()
```

Pyomo models that are not accompanied by mathematical documentation defining the variables, parameters, constraints, and objectives should use standard Python naming conventions. Following PEP 8, these names should be lower case with words separated by underscores as necessary to improve readability.

Use Pyomo Sets for indexing

Pyomo modeling elements including `Param`, `Var`, `Constraint` should be indexed by Pyomo Sets rather than iterable Python objects. For example, given a Python dictionary

```
bounds = {"a": 12, "b": 23, "c": 14}
```

The following

```
m.B = pyo.Set(initialize=bounds.keys())
m.x = pyo.Var(m.B)
```

Is preferred, rather than

```
m.x = pyo.Var(bounds.keys())
```

Use of upper-case letters to denote sets is an acceptable deviation from PEP style guidelines. Lower case letters can be used to denote elements of the set. For example,

$$f = \min \sum_{b \in B} x_b$$

may be written

```
m.f = pyo.Objective(expr=sum(m.x[b] for b in m.B))
```

mimicking the mathematical notation.

Use lambda functions to improve readability

Indexed constraints normally require a rule to generate the constraint from problem data. The rule is a Python function that returns a Pyomo equality or inequality expression, or a Python 3-tuple of the form (lb, expr, ub). The function requires a model instance as the first argument, and one additional argument for each index used to specify the constraint.

In some cases rules are simple enough to express in a single line. For these cases a Python lambda expression may be used to improve readability. For example, the constraint

```
def c_rule(m, s):
  return m.x[s] <= m.ub[s]
m.c = pyo.Constraint(m.S, rule=c_rule)
```

may be expressed as

```
m.c = pyo.Constraint(m.S, rule=lambda m, s: m.x[s] <= m.ub[s])
```

for a more succinct and readable specification of an indexed constraint.

Working with Data

Use Tidy Data

Tidy data is a semantic model for of organizing data sets. The core principle of Tidy data is that each data set is organized by rows and columns where each entry is a single value. Each column contains all data associated with single variable. Each row contains all values for a single observation.

Tidy data may be read in multiple ways. Pandas `DataFrame` objects are particularly well suited to Tidy Data and recommended for reading, writing, visualizing, and displaying Tidy Data.

When using doubly nested Python dictionaries for Tidy Data, the primary keys will provide unique identifiers for each observation. Each observation is a dictionary where secondary keys label the variables and each entry will consist of a single value.

Alternative structures may include nested lists, lists of dictionaries, or numpy arrays. In each case a single data will be referenced as `data[obs][var]` where `obs` identifies a particular observation or slice of observations, and `var` identifies a variable.

Higher dimensional data structures should encode values as `data[ds][obs][var]` where `ds` identifies a data set composed of Tidy Data.

Use Pandas for display and visualization

The Pandas library provides an extensive array of functions for the manipulation, display, and visualization of data sets.