Introduction: Python language

https://www.learnpython.org/

- Python is a widely used general-purpose, high level programming language
- Free, open-source
- Python provides a clean, intuitive syntax
- Many built-in libraries and third-party extensions
- Simple, but extremely powerful

You can launch the Spyder IDE, which is included in Anaconda3
Type statements or expressions in console:

```
In [1]: print("Hello world")
Hello world
In [2]: x = 12**2
In [3]: x/2
72
In [4]: # this is a comment
```

To write a program, put commands in a file

```
# hello.py
print("Hello world")
x = 12**2
print(x)
```

Run file (F5) `hello.py`

```
Hello world
144
```

Introduction: Python language

Python Variables:

- You do not need to declare variables before using them, or declare their type
- Need to assign (initialize)
- Numbers: integers/floating point numbers; strings; lists; dictionaries,…
- INDEXING STARTS FROM 0!

Lists: declared by []

- Flexible arrays, *not* linked lists
  - `a = [99, "bottles of beer", ["on", "the", "wall"]]`
- Same operators as for strings
  - `a+b, a*3, a[0], a[-1], a[1:], len(a)`
- Item and slice assignment
  - `a[0] = 98`
  - `a[1:2] = ["bottles", "of", "beer"]`
    `# -> [98, "bottles", "of", "beer", ["on", "the", "wall"]]`
  - `del a[-1]`
    `# -> [98, "bottles", "of", "beer"]`

Introduction: Python language

Dictionaries, declared by {}

- Hash tables, "associative arrays"
  - `d = {"duck": "eend", "water": "water"}`
- Lookup:
  - `d["duck"]    # -> "eend"`
  - `d["back"]    # raises KeyError exception`
- Delete, insert, overwrite:
  - `del d["water"]     # {"duck": "eend", "back": "rug"}`
  - `d["back"] = "rug"  # {"duck": "eend", "back": "rug"}`
  - `d["duck"] = "duik" # {"duck": "duik", "back": "rug"}`
- Keys, values, items:
  - `d.keys()     -> ["duck", "back"]`
  - `d.values()   -> ["duik", "rug"]`
  - `d.items()    -> [("duck","duik"), ("back","rug")]`
- Presence check:
  - `d.has_key("duck")   # -> 1; d.has_key("spam") -> 0`
- Values of any type; keys almost any
  - ```
    { "name": "Guido",
      "age": 43,
      ("hello","world"): 1,
      42: "yes",
      "flag": ["red","white","blue"] }
    ```
    ```

Introduction: Python language

Control structures: (grouping indentation)

```
if condition:
    statements
[elif condition:
    statements] ...
else:
    statements
```

```
while condition:
    statements

for var in sequence:
    statements

break
continue
```

Note: *Spacing matters!*
Control structure scope dictated by indentation

Introduction: Python language

Functions, procedures

```
def name(arg1, arg2, ...):
    """documentation"""    # optional doc string
    statements

    return expression      # from function
    return                 # from procedure (returns None)
```

Example:

```
def gcd(a, b):
    """greatest common divisor"""
    while a != 0:
        a, b = b%a, a   # parallel assignment
    return b

>>> gcd.__doc__
'greatest common divisor'

>>> gcd(12, 20)
4
```

Introduction: Python language

Modules

- Collection of stuff in *foo*.py file
  - functions, classes, variables

- Importing modules:

```
import re
print( re.match("[a-z]+", s) )
from re import match
print( match("[a-z]+", s) )
```

- Import with rename:

```
import re as regex
from re import match as m
```

Major Python Packages

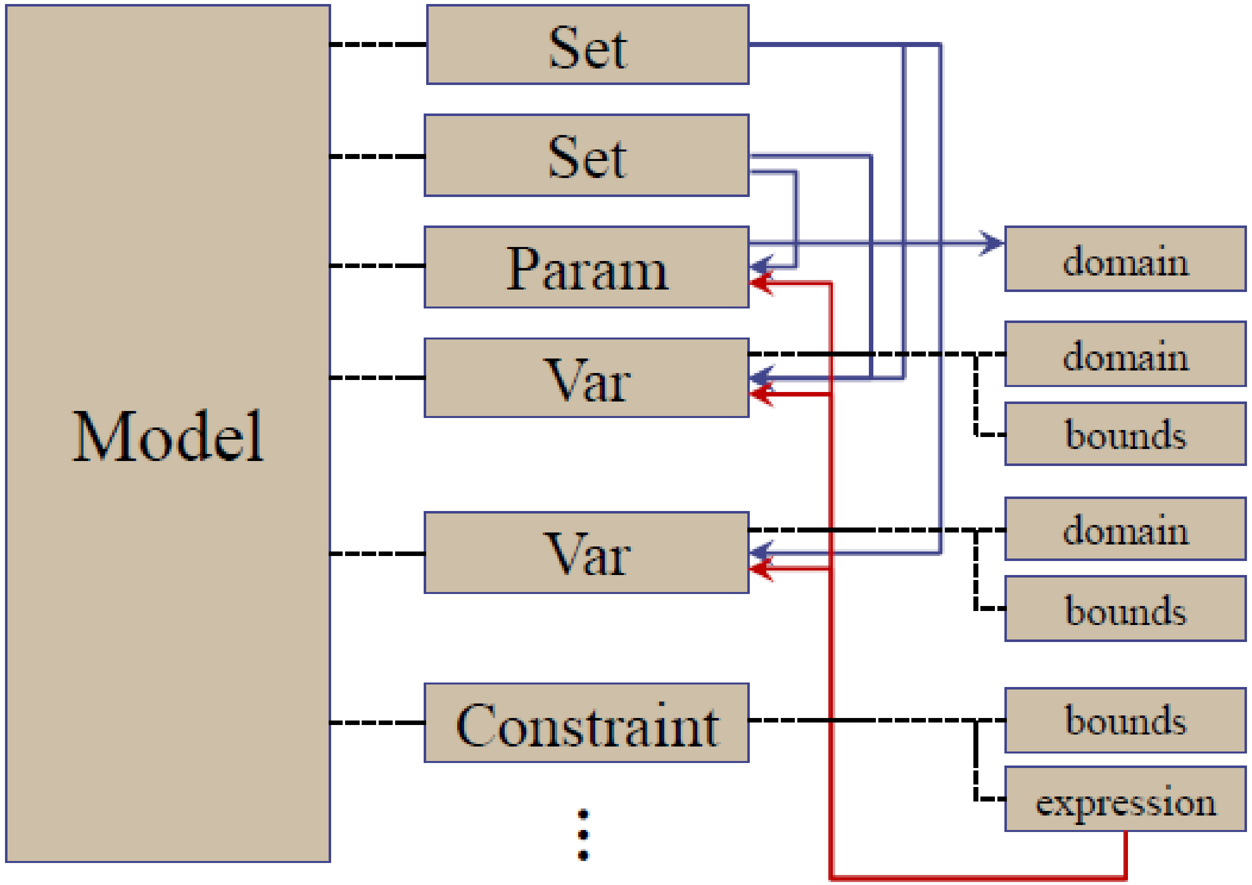
- SciPy
  - Scientific Python for mathematics and engineering
  - http://www.scipy.org
- Numpy
  - Numeric array package
  - http://www.numpy.org/
- Matplotlib
  - 2D plotting library
  - http://matplotlib.org/
- Pandas
  - Data structures and analysis
  - http://pandas.pydata.org/
- Ipython
  - Interactive Python shell
  - http://ipython.org/

Introduction: Pyomo fundamentals

https://jckantor.github.io/ND-Pyomo-Cookbook/

http://www.pyomo.org/documentation

- Pyomo is an *object model* for describing optimization problems
  - The fundamental objects used to build models are *Components*



Introduction: Pyomo fundamentals – the MODEL

```
from pyomo.environ import *

model = ConcreteModel()
```

Every Pyomo model starts with this; it tells Python to load the Pyomo Modeling Environment

Create an instance of a *Concrete* model

- Concrete models are immediately constructed
- Data must be present *at the time* components are defined

Local variable to hold the model we are about to construct

- While not required, by convention we use "`model`"
- If you choose to name your model something else, you will need to tell the Pyomo script the object name through the command line

Introduction: Pyomo fundamentals – the VARIABLES

```
model.a_variable = Var(within = NonNegativeReals)
```

The name you assign the object to becomes the object's name, and must be unique in any given model.

"`within`" is optional and sets the variable domain ("`domain`" is an alias for "`within`")

Several pre-defined domains, e.g., "`Binary`"

```
model.a_variable = Var(bounds = (0, None))
```

Same as above: "`domain`" is assumed to be `Reals` if missing

Introduction: Pyomo fundamentals – the OBJECTIVE FUNCTION

```
model.x = Var( initialize=-1.2, bounds=(-2, 2) )
model.y = Var( initialize= 1.0, bounds=(-2, 2) )

model.obj = Objective(
    expr= (1-model.x)**2 + 100*(model.y-model.x**2)**2,
    sense= minimize )
```

If "sense" is omitted, Pyomo assumes minimization

"expr" can be an expression, or any function-like object that returns an expression

Introduction: Pyomo fundamentals – the SETS to generate and manage indexes

- Any iterable object can be an index, e.g., lists:
  - `IDX_a = [1,2,5]`
  - `DATA = {1: 10, 2: 21, 5:42};`
    `IDX_b = DATA.keys()`

- Sets: objects for managing multidimensional indices
  - `model.IDX = Set( initialize = [1,2,5] )`

Note: capitalization matters:
Set = Pyomo class
set = native Python set

Like indices, Sets can be initialized from any iterable

- Sets of sequential integers are common
  - `model.IDX = Set( initialize=range(5) )`
  - `model.IDX = RangeSet( 5 )`

Note: **RangeSet** is 1-based.
This gives [1, 2, 3, 4, 5]

Note: Python **range** is 0-based.
This gives [0, 1, 2, 3, 4]

- You can provide lower and upper bounds to RangeSet
  - `model.IDX = RangeSet( 0, 4 )`

This gives [0, 1, 2, 3, 4]

Introduction: Pyomo fundamentals – the CONSTRAINTS

$$x_{w,c} \leq y_w \quad \forall\ w \in W, c \in C$$

```
W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)

def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)
```

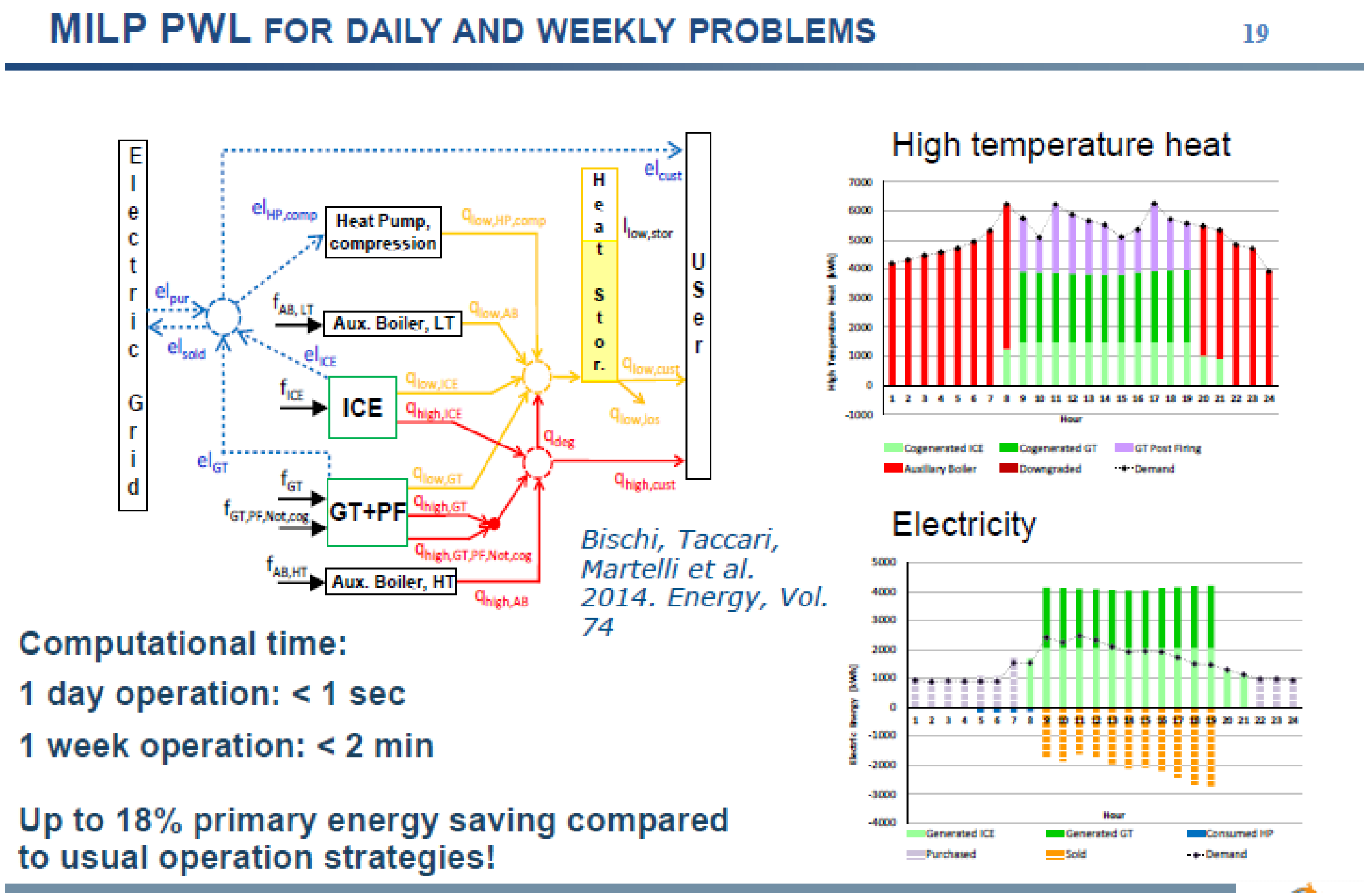
For indexed constraints, you provide a "`rule`" (function) that returns an expression (or tuple) for each index.

Each dimension of the indices is a separate argument to the rule

Rule is called once for every entry w in W crossed with every entry c in C!

The MES optimal operation MILP model:

The basic steps of a simple modeling process are as follows:
1. create an instance of a model using Pyomo modeling components
2. pass this instance to a solver to find a solution
3. report and analyze results from the solver



The MES optimal operation MILP model

Sets

I = set of all units

T = set of all time instants (24 hours in a day)

Subsets:
- Machines_fuel
- Machines_el
- Machines_th
- Machines_el_cons

Variables

- ‘MIMO’ units with 1 degree of freedom (ICE, ORC, boiler, heat pump,…)

$f_{i,t}$ = input fuel of unit i at time t

$q_{i,t}$ = output heat of unit i at time t

$p_{i,t}$ = output electricity of unit i at time t

$el\_in_{i,t}$ = input electricity of unit i at time t

$z_{i,t} \in \{0,1\}$ = 1 if unit i in on at time t, 0 if unit i in off at time t

$d_{SU\,i,t} \in \{0,1\}$ = 1 if unit i starts-up at time t, 0 otherwise

The MES optimal operation MILP model

Sets

I = set of all units

T = set of all time instants (24 hours in a day)

S = set of all storages

Subsets:
- Storages_el
- Storages_th

Variables

- Storages

$u_{s,t}$ = level of energy (heat or electricity) of storage s at time t

$Charge_{s,t}$ = energy charged to storage s at time t

$Discharge_{s,t}$ = energy discharged from storage s at time t

- Electricity sold/bought from the grid

${El_{buy}}_t$ = electricity bought from the grid at time t

${El_{sell}}_t$ = electricity sold to the grid at time t

ALL CONTINUOUS VARIABLES ARE POSITIVE REALS

The MES optimal operation MILP model

<u>Parameters</u>

d_{el_t}= demand of electricity at time t

d_{th_t}= demand of heat at time t

$El_{price_{sell,t}}$= electricity selling price to grid (constant) at time t

$El_{price_{buy,t}}$= electricity buying price from grid (constant) at time t

NG_{price}= natural gas price (constant)

$Biomass_{price}$= biomass price (constant)

The MES optimal operation MILP model

Parameters

$Fuel\_cost_i$ = fuel cost of unit i, for machines $i \in I\_f$

Linear performance maps:

${a_{th}}_i$ = coefficient of heat production map of unit i
${b_{th}}_i$ = constant term of heat production map of unit i

$$q_{i,t} = {a_{th}}_i f_{i,t} + {b_{th}}_i z_{i,t}$$

$$q_{i,t} = {a_{th}}_i el\_in_{i,t} + {b_{th}}_i z_{i,t}$$

${a_{el}}_i$ = coefficient of power production map of unit i
${b_{el}}_i$ = constant term of power production map of unit i

$$p_{i,t} = {a_{el}}_i f_{i,t} + {b_{el}}_i z_{i,t}$$

The MES optimal operation MILP model

Parameters

For machines $i \in I$:

$minIn_i$ = minimum input of unit i (if on)
$maxIn_i$= maximum input of unit i (if on)

$RUlim_i$ = ramp-up limit of unit i (only for ORC)

$\max\_n\_SU_i$ = maximum number of daily start-ups of unit i (only for ORC)

OM_i = fixed O&M costs of unit i
SU_i = fixed start-up cost of unit i

The MES optimal operation MILP model

Parameters

For storages

$maxC_s$ = maximum capacity (heat or electricity) of storage s
eta_{ch_s}, eta_{disch_s} = charge/discharge efficiency of electric storage s
eta_{diss_s} = thermal loss efficiency of thermal storage s

The MES optimal operation MILP model

Objective function (to be minimized):

$$OBJ = machines\_fuel\_cost \;+\; machines\_OM\_cost \;+\; machines\_SU\_cost \;+\; grid\_SellBuy$$

Where

$$machines\_fuel\_cost = \sum_{i \in I\_f} \sum_{t \in T} Fuel\_cost_i \cdot f_{i,t}$$

$$machines\_OM\_cost = \sum_{i \in I} \sum_{t \in T} OM_i \cdot z_{i,t}$$

$$machines\_SU\_cost = \sum_{i \in I} \sum_{t \in T} SU_i \cdot dSU_{i,t}$$

$$grid\_SellBuy = \sum_{t \in T} El_{price_{buy}} \cdot El_{buy_t} - \sum_{t \in T} El_{price_{sell}} \cdot El_{sell_t}$$

The MES optimal operation MILP model

Constraints:

Electricity balance

$$\sum_{i \in I_{el}} p_{i,t} - \sum_{i \in I_{el_{cons}}} el\_in_{i,t} + El_{buy_t} - El_{sell_t} + \sum_{s \in S_{el}} Discharge_{s,t} - \sum_{s \in S_{el}} Charge_{s,t} = d_{el_t}$$

$$\forall\, t \in T$$

The MES optimal operation MILP model

<u>Constraints:</u>

Heat balance

$$\sum_{i\in I_{th}} q_{i,t} + \sum_{s\in S_{th}} Discharge_{s,t} - \sum_{s\in S_{th}} Charge_{s,t} = d_{th_t}$$

$$\forall\, t \in T$$

The MES optimal operation MILP model

Constraints:

El. power production of units

$$p_{i,t} = a_{el_i} f_{i,t} + b_{el_i} z_{i,t} \qquad \forall\, i \in I\_el, \qquad \forall\, t \in T$$

Heat production ($i \in I\_th$) of units with fuel input or electricity input (heat pump)

$$q_{i,t} = a_{th_i} f_{i,t} + b_{th_i} z_{i,t} \qquad \forall\, i \in I\_f, \qquad \forall\, t \in T$$

$$q_{i,t} = a_{th_i} el\_in_{i,t} + b_{th_i} z_{i,t} \qquad \forall\, i \in I\_el\_cons, \qquad \forall\, t \in T$$

The MES optimal operation MILP model

Constraints:

Operating range of units – minimum input

$$f_{i,t} \geq minIn_i z_{i,t} \qquad \forall\, i \in I_f, \qquad \forall\, t \in T$$

$$el\_in_{i,t} \geq minIn_i z_{i,t} \qquad \forall\, i \in I_{el\_cons}, \qquad \forall\, t \in T$$

Operating range of units – maximum input

$$f_{i,t} \leq maxIn_i z_{i,t} \qquad \forall\, i \in I_f, \qquad \forall\, t \in T$$

$$el\_in_{i,t} \leq maxIn_i z_{i,t} \qquad \forall\, i \in I_{el\_cons}, \qquad \forall\, t \in T$$

The MES optimal operation MILP model

<u>Constraints:</u>

Logical constraints for start-up of units

$$z_{i,t} - z_{i,t-1} \leq dSU_{i,t}$$

$$\forall\, i \in I, \qquad \forall\, t \in T$$

NOTE: case when t == 0

Ramp-up constraints

$$f_{i,t} - f_{i,t-1} \leq RUlim_i z_{i,t} \qquad \forall\, i \in I_f, \qquad \forall\, t \in T$$

NOTE: case when t == 0

$$el\_in_{i,t} - el\_in_{i,t-1} \leq RUlim_i z_{i,t} \qquad \forall\, i \in I_{el\_cons}, \forall\, t \in T$$

The MES optimal operation MILP model

<u>Constraints:</u>

Maximum number of start-ups per day (only for ORC)

$$\sum_{t \in T} dSU_{i,t} \leq 1$$

The MES optimal operation MILP model

<u>Constraints:</u>

NOTE: case when t == 0

Thermal storage energy balance:

$$u_{s,t} - u_{s,t-1} \cdot eta_{diss,s} = Charge_{s,t} - Discharge_{s,t}$$

$$\forall\, s \in S\_th, \qquad \forall\, t \in T$$

Electric storage energy balance:

$$u_{s,t} - u_{s,t-1} = Charge_{s,t} \cdot eta_{ch,s} - Discharge_{s,t} / eta_{disch,s}$$

$$\forall\, s \in S\_el, \qquad \forall\, t \in T$$

Storage capacity

$$u_{s,t} \leq MaxC_s \qquad \forall\, s \in S, \qquad \forall\, t \in T$$

The MES optimal operation MILP model

Solve model

```
pyo.SolverFactory('CBC',mipgap = 0.005).solve(m).write()
```

Save results

```
.value
```

The MES optimal operation MILP model

Barplots

Day 1   Day 2