Advanced Data Management (CSCI 490/680)

Python

Dr. David Koop
In this Notebook we explore the Lorenz system of differential equations:

\[
\begin{align*}
\dot{x} &= \sigma(y - x) \\
\dot{y} &= px - y - xz \\
\dot{z} &= -\rho z + xy
\end{align*}
\]

Let's call the function once to view the solutions. For this set of parameters, we see the trajectories swirling around two points, called attractors.

```python
In [4]: from Lorenz import solve_lorenz
t, x, y, z = solve_lorenz(N=10)
```

```python
def solve_lorenz(N=10, max_time=10.0, sigmas=10.0, betas=8.0/3, rhos=28.0):
    """Compute the time-derivative of a Lorenz system."""
    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
    ax.set_xlim([-25, 25])
    ax.set_ylim([-20, 20])
    ax.set_zlim([-25, 25])

    z = lorenz_deriv(x, y, z, sigmas, betas, rhos)
    return [sigmas * (y - x), x * (rhos - z) - y, x * y - betas * z]

# Choose random starting points, uniformly distributed from -15 to 15
np.random.seed(1)
x0 = -15 + 30 * np.random.random((N, 3))
```
JupyterLab Notebooks

• Can write code or plain text (can be styled Markdown)
  - Choose the type of cell using the dropdown menu
• Cells break up your code, but all data is **global**
  - Defining a variable `a` in one cell means it is available in **any** other cell
  - This includes cells **above** the cell `a` was defined in!
• Remember **Shift+Enter** to execute
• Enter just adds a new line
• Use `?<function_name>` for help
• Use Tab for **auto-complete** or suggestions
• Tab also indents, and Shift+Tab unindents
Exercise

• Given variables $x$ and $y$, print the long division answer of $x$ divided by $y$ with the remainder.

• Examples:
  - $x = 11, y = 4$ should print "2R3"
  - $x = 15, y = 2$ should print "7R1"
Quiz

• Suppose I want to write Python code to print the numbers from 1 to 100. What errors do you see?

```python
// print the numbers from 1 to 100
int counter = 1
while counter < 100 {
    print counter
    counter++
}
Quiz

• Suppose \( a = ['a', 'b', 'c', 'd'] \) and \( b = (1, 2, 3) \)

• What happens with?
  - \( a[0] \)
  - \( a[1:2] \)
  - \( b[:-2] \)
  - \( b.append(4) \)
  - \( a.extend(b) \)
  - \( a.pop(0) \)
  - \( b[0] = "100" \)
  - \( b + (4,) \)
• Suppose $a = ['a', 'b', 'c', 'd']$ and $b = (1, 2, 3)$

• What happens with?
  - $a[0]$ # 'a'
  - $a[1:2]$ # ['b']
  - $b[:-2]$ # (1,)
  - $b$.append(4) # error
  - $a$.extend($b$) # ['a', 'b', 'c', 'd', 1, 2, 3]
  - $a$.pop(0) # ['b', 'c', 'd']
  - $b[0]$ = "100" # error
  - $b + (4,)$ # (1,2,3,4)
Modifying Lists

• Add to a list `l`:
  - `l.append(v)`: add one value (v) to the end of the list
  - `l.extend(vlist)`: add multiple values (vlist) to the end of l
  - `l.insert(i, v)`: add one value (v) at index i

• Remove from a list `l`:
  - `del l[i]`: deletes the value at index i
  - `l.pop(i)`: removes the value at index i (and returns it)
  - `l.remove(v)`: removes the first occurrence of value v (careful!)

• Changing an entry:
  - `l[i] = v`: changes the value at index i to v (Watch out for IndexError!)
Why do we create and use functions?
Assignment 1

• To be released soon (planning on tomorrow)
• Using Python for data analysis
• Provided a1.ipynb file (right-click and download)
• Use basic python for now to demonstrate language knowledge
• Use Anaconda or hosted Python environment
• Turn .ipynb file in via Blackboard
Local Jupyter Environment

- [www.anaconda.com/download/](http://www.anaconda.com/download/)
- Anaconda has Jupyter Lab
- Use Python 3.8 version (**not** 2.7)
- Anaconda Navigator
  - GUI application for managing Python environment
  - Can install packages
  - Can start JupyterLab
- Can also use the shell to do this:
  - `$ jupyter lab`
  - `$ conda install <pkg_name>`
Hosted Jupyter Environments

• Nice to have ability to configure everything locally, but… you have to configure everything locally
• Solution: Cloud-hosted Jupyter (and Jupyter-like) environments
• Pros: No setup
• Cons: Limitations on resources: data and compute
• Options:
  - Google Colab (need a Google account)
  - Binder
  - Others…
Using Hosted Jupyter Environments

• Data:
  - Either point to a public URL or upload the data
  - Large datasets may not be supported, data may be deleted if uploaded (and isn't in Google Drive, etc.)

• Notebooks:
  - Can download the notebook locally (e.g. to use with a conda environment)
  - Currently, Python 3.8

• Differences:
  - Colab has tweaked much of the interface (e.g. different nomenclature)
Dictionaries

- One of the most useful features of Python
- Also known as associative arrays
- Exist in other languages but a core feature in Python
- Associate a key with a value
- When I want to find a value, I give the dictionary a key, and it returns the value
- Example: InspectionID (key) → InspectionRecord (value)
- Keys must be immutable (technically, hashable):
  - Normal types like numbers, strings are fine
  - Tuples work, but lists do not (TypeError: unhashable type: 'list')
- There is only one value per key!
Dictionaries

- Defining a dictionary: curly braces
  - `states = {'MA': 'Massachusetts', 'RI': 'Road Island', 'CT': 'Connecticut'}`

- Accessing a value: use brackets!
  - `states['MA']` or `states.get('MA')`

- Adding a value:
  - `states['NH'] = 'New Hampshire'`

- Checking for a key:
  - `'ME' in states` → **returns** `True` or `False`

- Removing a value: `states.pop('CT')` or `del states['CT']`

- Changing a value: `states['RI'] = 'Rhode Island'`
Dictionaries

• Combine dictionaries: \texttt{d1.update(d2)}
  - \texttt{update} overwrites any key-value pairs in \texttt{d1}
    when the same key appears in \texttt{d2}

• \texttt{len(d)} is the number of entries in \texttt{d}
Extracting Parts of a Dictionary

- `d.keys()`: the keys only
- `d.values()`: the values only
- `d.items()`: key-value pairs as a collection of tuples:
  
  ```python
  [(k1, v1), (k2, v2), ...]
  ```

- Unpacking a tuple or list
  ```python
  t = (1, 2)
  a, b = t
  ```

- Iterating through a dictionary:
  ```python
  for (k,v) in d.items():
      if k % 2 == 0:
          print(v)
  ```

- Important: keys, values, and items are in added order!
Example: Counting Letters

• Write code that takes a string \( s \) and creates a dictionary with that counts how often each letter appears in \( s \)

• \( \text{count\_letters}(\text{"Mississippi"}) \rightarrow \{ 's': 4, 'i': 4, 'p': 2, ... \} \)
Sets

- Just the keys from a dictionary
- Only one copy of each item
- Define like dictionaries without values
  - `s = {'a','b','c','e'}`
  - `'a' in s` # True

- Mutation
  - `s.add('f')`
  - `s.add('a')` # only one copy
  - `s.remove('c')`

- One gotcha:
  - `{}` is an empty dictionary not an empty set
Nesting Containers

• Can have lists inside of lists, tuples inside of tuples, dictionaries inside of dictionaries

• Can also have dictionaries inside of lists, tuples inside of dictionaries, …

• `d = {"Brady": [(2015, 4770, 36), (2014, 4109, 33)],
  "Luck": [(2015, 1881, 15), (2014, 4761, 40)],
  ...
  }

• JavaScript Object Notation (JSON) looks very similar for literal values; Python allows variables in these types of structures
Nesting Code

- Can have loops inside of loops, if statements inside of if statements.
- Careful with variable names:

```python
l = {0: 0, 1: 3, 4: 5, 9: 12}
for i in range(100):
square = i ** 2
max_val = l[square]
for i in range(max_val):
    print(i)
```

- Strange behavior, likely unintended, but Python won't complain!
None

• Like null in other languages, used as a placeholder when no value exists
• The value returned from a function that doesn't return a value

```python
def f(name):
    print("Hello,", name)
    v = f("Patricia")  # v will have the value None
```

• Also used when you need to create a new list or dictionary:

```python
def add_letters(s, d=None):
    if d is None:
        d = {}  
        d.update(count_letters(s))
```

• Looks like `d={} would make more sense, but that causes issues
• `None serves as a sentinel value in `add_letters`
is and ==

• == does a normal equality comparison
• is checks to see if the object is the exact same object
• Common style to write statements like if d is None: …

Weird behavior:
- a = 4 - 3
  a is 1 # True
- a = 10 ** 3
  a is 1000 # False
- a = 10 ** 3
  a == 1000 # True

• Generally, avoid is unless writing is None
is and ==

• == does a normal equality comparison
• is checks to see if the object is the exact same object
• Common style to write statements like if d is None: ...
• Weird behavior:
  - a = 4 - 3
    a is 1 # True
  - a = 10 ** 3
    a is 1000 # False
  - a = 10 ** 3
    a == 1000 # True

• Generally, avoid is unless writing is None
Objects

- `d = dict()` # construct an empty dictionary object
- `l = list()` # construct an empty list object
- `s = set()` # construct an empty set object
- `s = set([1, 2, 3, 4])` # construct a set with 4 numbers

**Calling methods:**
- `l.append('abc')`
- `d.update({"a": 'b'})`
- `s.add(3)`

- The method is tied to the object preceding the dot (e.g. `append` modifies `l` to add `'abc'`)
Python Modules

• Python module: a file containing definitions and statements

• Import statement: like Java, get a module that isn't a Python builtin

   ```python
   import collections
d = collections.defaultdict(list)
d[3].append(1)
   ```

• `import <name> as <shorter-name>`

   ```python
   import collections as c
   ```

• `from <module> import <name>` : don't need to refer to the module

   ```python
   from collections import defaultdict
d = defaultdict(list)
d[3].append(1)
   ```
Other Collections

- `collections.defaultdict`: specify a default value for any item in the dictionary (instead of `KeyError`)
- `collections.OrderedDict`: keep entries ordered according to when the key was inserted
  - `dict` objects are ordered in Python 3.7 but `OrderedDict` has some other features (equality comparison, reversed)
- `collections.Counter`: counts hashable objects, has a `most_common` method
Example: Counting Letters

• Write code that takes a string $s$ and creates a dictionary with that counts how often each letter appears in $s$

• `count_letters("Mississippi") → {'s': 4, 'i': 4, 'p': 2}, ...`
Solution using Counter

• Use an existing library made to count occurrences

```python
from collections import Counter
Counter("Mississippi")
```

• produces

```python
Counter({'M': 1, 'i': 4, 's': 4, 'p': 2})
```

• Improve: convert to lowercase first