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, z = solve_Lorenz(N=10)
```

```python
def solve_Lorenz(N=10, max_time=4.0, sigma=10.0, beta=8.0/3, rho=28.0):
    """Solve a set of the Lorenz differential equations."""
    fig, plt.figure()
    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
    ax.axis('off')
    ax.set_zlim(-25, 25)
    ax.set_xlim(-25, 25)
    ax.set_ylim(-25, 25)

    x0, y0, z0 = x, y, z
    return [sigma * (y0 - x0), x0 * (rho - z0) - y0, x0 * y0 - beta * z0]

def lorenz_deriv(x, y, z, sigma, beta, rho=28.0):
    """Compute the time-derivative of a Lorenz system."""
    x, y, z = x, y, z
    return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]
```

```python
# Choose random starting points, uniformly distributed from -15 to 15
x0 = -15 + 30 * np.random.randn(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 that variable is accessible 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
Python

- Started in December 1989 by Guido van Rossum
- “Python has surpassed Java as the top language used to introduce U.S. students to programming…” (ComputerWorld, 2014)
- Python and R are the two top languages for data science
- High-level, interpreted language
- Supports multiple paradigms (OOP, procedural, imperative)
- Help programmers write readable code
- Use less code to do more
- Lots of libraries for python
  - Designed to be extensible
Python Compared to C++ and Java

• Dynamic Typing
  - A variable does not have a fixed type
  - Example: `a = 1; a = "abc"

• Indentation
  - Braces define blocks in Java, good style is to indent but not required
  - Indentation is **critical** in Python

```python
z = 20
if x > 0:
    if y > 0:
        z = 100
else:
    z = 10
```
Python Variables and Types

• No type declaration necessary
• Variables are names, not memory locations
  
  \[
  a = 0
  \]
  
  \[
  a = "abc"
  \]
  
  \[
  a = 3.14159
  \]
• Don't worry about types, but think about types
• Strings are a type
• Integers are as big as you want them
• Floats can hold large numbers, too (double-precision)
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"
Loops

• while <condition>:
  <indented block>
  # end of while block (indentation done)

• Remember the colon!

• a = 5
  while a > 0:
    print(a)
    a -= 2

• a > 0 is the condition

• Python has standard boolean operators (<, >, <=, >=, ==, !=)
  - What does a boolean operation return?
  - Linking boolean comparisons (and, or)
break and continue

- **break** stops the execution of the loop
- **continue** skips the rest of the loop and goes to the next iteration

```
• a = 7
  while a > 0:
    a -= 2
    if a < 4:
      break
    print(a)
```

```
• a = 7
  while a > 0:
    a -= 2
    if a < 4 and a > 2:
      continue
    print(a)
```
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++
}
```
Python Containers

• Container: store more than one value
• Mutable versus immutable: Can we update the container?
  - Yes → mutable
  - No → immutable
  - Lists are mutable, tuples are immutable
• Lists and tuples may contain values of different types:
  • List: [1,"abc",12.34]
  • Tuple: (1, "abc", 12.34)
• You can also put functions in containers!
• len function: number of items: len(l)
Indexing and Slicing

- Just like with strings
- Indexing:
  - Where do we start counting?
  - Use brackets [] to retrieve one value
  - Can use negative values (count from the end)
- Slicing:
  - Use brackets plus a colon to retrieve multiple values:
    \[ [<\text{start}>:]<\text{end}> \]
  - Returns a new list \( b = a[::] \)
  - Don't need to specify the beginning or end
Quiz

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

• What happens with?
  - \( a[0] \)
  - \( b[:-2] \)
  - \( b.append(4) \)
  - \( a.extend(b) \)
  - \( a.pop(0) \)
  - \( b[0] = "100" \)
  - \( b + (4,) \)
Quiz

- Suppose \( a = ['a', 'b', 'c', 'd'] \) and \( b = (1, 2, 3) \)
- What happens with?
  - \( a[0] \) # 'a'
  - \( b[:-2] \) # (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!)
For loops

- Used much more frequently than while loops
- Is actually a "for-each" type of loop
- In Java, this is:
  ```java
  for (String item : someList) {
      System.out.println(item);
  }
  ```
- In Python, this is:
  ```python
  for item in someList:
      print(item)
  ```
- Grabs each element of `someList` in order and puts it into `item`
- Be careful modifying container in a for loop! (e.g. `someList.append(new_item)`)

D. Koop, CSCI 490/680, Spring 2020
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!
Sets

- Sets are like dictionaries but without any values:
  - `s = {'MA', 'RI', 'CT', 'NH'}; t = {'MA', 'NY', 'NH'}`
  - `{}` is an empty dictionary, `set()` is an empty set
  - Adding values: `s.add('ME')`
  - Removing values: `s.discard('CT')`
  - Exists: "CT" in `s`
  - Union: `s | t => {'MA', 'RI', 'CT', 'NH', 'NY'}`
  - Intersection: `s & t => {'MA', 'NH'}`
  - Exclusive-or (xor): `s ^ t => {'RI', 'CT', 'NY'}`
  - Difference: `s - t => {'RI', 'CT'}`
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
About this course

• Course web page is authoritative:
  - http://faculty.cs.niu.edu/~dakoop/cs680-2020sp
  - Schedule, Readings, Assignments will be posted online
  - Check the web site before emailing me
• Course is meant to be more "cutting edge"
  - Still focus on building skills related to data management
  - Tune into current research and tools
• Requires student participation: readings and discussions
• Exam Dates: Feb. 18, March 26, May 5 (final)
Assignment 1

- Using Python for data analysis
- Analyze hurricane data (through 2018)
- Provided a1.ipynb file (right-click and download)
- Use basic python (+ collections module) for now to demonstrate language knowledge
- Use Anaconda or hosted Python environment
- Due next Wednesday
- Turn .ipynb file in via Blackboard
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:
  - Azure Notebooks (can use your NIU account)
  - 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 3.6

• Differences:
  - Colab has tweaked much of the interface (e.g. different nomenclature)
  - Azure is, for the most part, running Jupyter
  - Azure is more of a preview than Colab
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:
  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

- Python caches common integer objects

- Generally, avoid is unless writing is 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
Iterators

• Remember range, values, keys, items?
• They return **iterators**: objects that traverse containers, only need to know how to get the next element
• Given iterator `it`, `next(it)` gives the next element
• `StopIteration` exception if there isn't another element
• Generally, we don't worry about this as the for loop handles everything automatically…but you cannot index or slice an iterator
• `d.values()[0]` will not work!
• If you need to index or slice, construct a list from an iterator
• `list(d.values())[0]` or `list(range(100))[-1]`
List Comprehensions

• Shorthand for transformative or filtering for loops

  squares = []
  for i in range(10):
      squares.append(i**2)

  squares = [i**2 for i in range(10)]

• Filtering:

  squares = []
  for i in range(10):
      if i % 3 != 1:
          squares.append(i ** 2)

  squares = [i**2 for i in range(10) if i % 3 != 1]

• if clause follows the for clause
Dictionary Comprehensions

• Similar idea, but allow dictionary construction
• Could use lists:
  - names = dict([(k, v) for k,v in ... if ...])
• Native comprehension:
  - names = {"Al": ["Smith", "Brown"], "Beth": ["Jones"]}
    first_counts = {k: len(v) for k,v in names.items()}
• Could do this with a for loop as well