Advanced Data Management (CSCI 490/680)

Python

Dr. David Koop
Data to Knowledge and Beyond

[D. Somerville, based on H. McLeod's original]
Data to Knowledge and Beyond

[D. Somerville, based on H. McLeod’s original]
Data to Knowledge and Beyond

[D. Somerville, based on H. McLeod’s original]

Require People
Data to Knowledge

Can computers do this for us?

[D. Somerville, based on H. McLeod's original]
FINDINGS

we got about the future of the data science, the most salient takeaway was how excited our respondents were about the evolution of the job they saw their jobs getting more interesting and less repetitive, all while expressing a real and broad enthusiasm about the value of the work in their organization.

How do data scientists spend their time?

What data scientists spend the most time doing

- Building training sets: 3%
- Cleaning and organizing data: 60%
- Collecting data sets: 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%

[CrowdFlower Data Science Report, 2016]
Finding & Discovering Data (even data you already have!)

DATA WAREHOUSE

VS

DATA LAKE

Data is processed and organized into a single schema before being put into the warehouse

Raw and unstructured data goes into a data lake

The analysis is done on the cleansed data in the warehouse

Data is selected and organized as and when needed

[S. Dewan]
Data Wrangling

1 INTRODUCTION

1.1 Motivation

Data wrangling is a labor-intensive and time-consuming process. In a recent crawl, we extracted over 1.8M functions from GitHub. (Right): Transformations for addresses. The system has the following features that can leverage rich transformation logic in source code, DLLs, web services, and mapping tables, so that end-users can already support many important domains out of the box, there will be diverse application domains.

1.2 Unique Features

- Extensibility: users can simply point to their code, DLLs, web services, and mapping tables, and execute a SQL query or a pivot table using this data, as day-of-the-week is missing from the input. However, deriving day-of-the-week from phone-numbers, or zip-code from addresses, both of which missing from the input. However, deriving day-of-the-week from phone-numbers, or zip-code from addresses, both of which is therefore called Search-by-Example. TDE works is just like a search engine “indexing” a new document.
- Head-domain Support: TDE automatically synthesizes new programs whose output are consistent with the given examples. Within a few seconds, a ranked list of programs are returned as the top result.
- Self-service data transformation: (More details of the system can be found in a full research article [11]). We develop novel algorithms to make this possible at an interactive speed, with just a few (typically three) input/output examples.
- APIs: are returned as the top result.
- University/Institutional Copyright Notice: Permission to make copies of this material for classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

1.3 Outline

This data set is obviously not ready for analysis yet – an analyst needs to figure out which day-of-the-week (Mon, Tue, etc.) has been filled in. In this example, date values in the first column are completely inconsistent with the format of the date in the fourth column. An analyst may want to use this column to perform analysis. Suppose she wants to derive day-of-the-week from phone-numbers, or zip-code from addresses, both of which is missing from the input. However, deriving day-of-the-week from phone-numbers, or zip-code from addresses, both of which is therefore called Search-by-Example. TDE works is just like a search engine “indexing” a new document.

2 System Overview

TDE synthesizes programs whose output are consistent with the given examples. Within a few seconds, a ranked list of programs are returned as the top result.

2.1 Dataset

We have built a production-quality data-transformation engine. (Left): Demonstration for date-time. (Right): After clicking on the “Get Transformation” button, the transformed data is returned with the output examples.

2.2 Demonstration

We have built a production-quality data-transformation engine. In a recent crawl, we extracted over 1.8M functions from GitHub. Transform-Data-by-Example (TDE) system has the following features that can index rich transformation logic in source code, DLLs, web services and mapping tables, so that end-users can already support many important domains out of the box, there will be diverse application domains.

3 Search-by-Example

TDE is therefore called Search-by-Example. TDE works is just like a search engine “indexing” a new document.

3.1 Search Engine

3.1.1 Query Formulation

Query formulation is crucial in search engines. In this section, we will discuss how we formulate queries for TDE.

3.1.2 Query Execution

Query execution is the process of searching through the index to find relevant documents.

4 Evaluation

We evaluate the effectiveness of TDE against existing PBE systems such as FlashFill that compose results (shaded in green).

5 Conclusion

In this paper, we have presented TDE, a novel search engine for data transformation. TDE allows end-users to search transformation for date-time. (Left): input data is in column-C, user provides two desired output examples in columns-D and E, and TDE automatically synthesizes new programs whose output are consistent with the given examples. Within a few seconds, a ranked list of programs are returned as the top result.

6 References

[1] Y. He, Kukjin Lee, Kris Ganjam, Yue Wang, Twitter Inc., San Francisco, USA. 2018. Transform-Data-by-Example (TDE) system has the following features that can index rich transformation logic in source code, DLLs, web services and mapping tables, so that end-users can already support many important domains out of the box, there will be diverse application domains.

7 Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments and suggestions.

8 Appendix

8.1 Additional Experimental Results

8.1.1 Additional Experimental Results for TDE

8.1.2 Additional Experimental Results for Existing PBE Systems

8.2 Additional Data Wrangling Examples

8.2.1 Additional Data Wrangling Examples for TDE

8.2.2 Additional Data Wrangling Examples for Existing PBE Systems

9 Appendix H

9.1 Additional Technical Details

9.1.1 Additional Technical Details for TDE

9.1.2 Additional Technical Details for Existing PBE Systems

9.2 Additional Technical Details for Data Wrangling Examples

9.3 Additional Technical Details for Additional Data Wrangling Examples

10 Appendix I

10.1 Additional Experimental Results

10.1.1 Additional Experimental Results for TDE

10.1.2 Additional Experimental Results for Existing PBE Systems

10.2 Additional Data Wrangling Examples

10.2.1 Additional Data Wrangling Examples for TDE

10.2.2 Additional Data Wrangling Examples for Existing PBE Systems

11 Appendix J

11.1 Additional Technical Details

11.1.1 Additional Technical Details for TDE

11.1.2 Additional Technical Details for Existing PBE Systems

11.2 Additional Technical Details for Data Wrangling Examples

12 Appendix K

12.1 Additional Experimental Results

12.1.1 Additional Experimental Results for TDE

12.1.2 Additional Experimental Results for Existing PBE Systems

12.2 Additional Data Wrangling Examples

12.2.1 Additional Data Wrangling Examples for TDE

12.2.2 Additional Data Wrangling Examples for Existing PBE Systems

13 Appendix L

13.1 Additional Technical Details

13.1.1 Additional Technical Details for TDE

13.1.2 Additional Technical Details for Existing PBE Systems

13.2 Additional Technical Details for Data Wrangling Examples

14 Appendix M

14.1 Additional Experimental Results

14.1.1 Additional Experimental Results for TDE

14.1.2 Additional Experimental Results for Existing PBE Systems

14.2 Additional Data Wrangling Examples

14.2.1 Additional Data Wrangling Examples for TDE

14.2.2 Additional Data Wrangling Examples for Existing PBE Systems

15 Appendix N

15.1 Additional Technical Details

15.1.1 Additional Technical Details for TDE

15.1.2 Additional Technical Details for Existing PBE Systems

15.2 Additional Technical Details for Data Wrangling Examples

16 Appendix O

16.1 Additional Experimental Results

16.1.1 Additional Experimental Results for TDE

16.1.2 Additional Experimental Results for Existing PBE Systems

16.2 Additional Data Wrangling Examples

16.2.1 Additional Data Wrangling Examples for TDE

16.2.2 Additional Data Wrangling Examples for Existing PBE Systems

17 Appendix P

17.1 Additional Technical Details

17.1.1 Additional Technical Details for TDE

17.1.2 Additional Technical Details for Existing PBE Systems

17.2 Additional Technical Details for Data Wrangling Examples

18 Appendix Q

18.1 Additional Experimental Results

18.1.1 Additional Experimental Results for TDE

18.1.2 Additional Experimental Results for Existing PBE Systems

18.2 Additional Data Wrangling Examples

18.2.1 Additional Data Wrangling Examples for TDE

18.2.2 Additional Data Wrangling Examples for Existing PBE Systems

19 Appendix R

19.1 Additional Technical Details

19.1.1 Additional Technical Details for TDE

19.1.2 Additional Technical Details for Existing PBE Systems

19.2 Additional Technical Details for Data Wrangling Examples

20 Appendix S

20.1 Additional Experimental Results

20.1.1 Additional Experimental Results for TDE

20.1.2 Additional Experimental Results for Existing PBE Systems

20.2 Additional Data Wrangling Examples

20.2.1 Additional Data Wrangling Examples for TDE

20.2.2 Additional Data Wrangling Examples for Existing PBE Systems

21 Appendix T

21.1 Additional Technical Details

21.1.1 Additional Technical Details for TDE

21.1.2 Additional Technical Details for Existing PBE Systems

21.2 Additional Technical Details for Data Wrangling Examples

22 Appendix U

22.1 Additional Experimental Results

22.1.1 Additional Experimental Results for TDE

22.1.2 Additional Experimental Results for Existing PBE Systems

22.2 Additional Data Wrangling Examples

22.2.1 Additional Data Wrangling Examples for TDE

22.2.2 Additional Data Wrangling Examples for Existing PBE Systems

23 Appendix V

23.1 Additional Technical Details

23.1.1 Additional Technical Details for TDE

23.1.2 Additional Technical Details for Existing PBE Systems

23.2 Additional Technical Details for Data Wrangling Examples

24 Appendix W

24.1 Additional Experimental Results

24.1.1 Additional Experimental Results for TDE

24.1.2 Additional Experimental Results for Existing PBE Systems

24.2 Additional Data Wrangling Examples

24.2.1 Additional Data Wrangling Examples for TDE

24.2.2 Additional Data Wrangling Examples for Existing PBE Systems

25 Appendix X

25.1 Additional Technical Details

25.1.1 Additional Technical Details for TDE

25.1.2 Additional Technical Details for Existing PBE Systems

25.2 Additional Technical Details for Data Wrangling Examples

26 Appendix Y

26.1 Additional Experimental Results

26.1.1 Additional Experimental Results for TDE

26.1.2 Additional Experimental Results for Existing PBE Systems

26.2 Additional Data Wrangling Examples

26.2.1 Additional Data Wrangling Examples for TDE

26.2.2 Additional Data Wrangling Examples for Existing PBE Systems

27 Appendix Z

27.1 Additional Technical Details

27.1.1 Additional Technical Details for TDE

27.1.2 Additional Technical Details for Existing PBE Systems

27.2 Additional Technical Details for Data Wrangling Examples

D. Koop, CSCI 490/680, Spring 2020

Image 1786 to 2279

[Image 534x522 to 2279x1064]
Data Cleaning/Standardization (Aliases)

- 'google brain resident': 'google',
- 'google brain': 'google',
- 'google inc': 'google',
- 'google inc.': 'google',
- 'google research nyc': 'google',
- 'google research': 'google',
- 'google, inc.': 'google',
- 'deepmind @ google': 'deepmind',
- 'deepmind technologies': 'deepmind',
- 'google deepmind': 'deepmind',

- 'ibm research - china': 'ibm',
- 'ibm research': 'ibm',
- 'ibm research, ny': 'ibm',
- 'ibm research, usa': 'ibm',
- 'ibm t.j. watson research center': 'ibm',
- 'ibm t.j. watson research': 'ibm',
- 'ibm t.j watson research center': 'ibm',
- 'ibm t.j. watson research center': 'ibm',
- 'ibm thomas j. watson research center': 'ibm',
- 'ibm tj watson research center': 'ibm',

- 'microsoft research cambridge': 'microsoft',
- 'microsoft research india': 'microsoft',
- 'microsoft research maluuba': 'microsoft',
- 'microsoft research new england': 'microsoft',
- 'microsoft research': 'microsoft',
- 'microsoft research, redmond, w': 'microsoft',
- 'microsoft research, redmond, wa': 'microsoft',
- 'microsoft research': 'microsoft',

[NLP Publishing Stats, M. Rei & R. Allen]
Data Integration

• Google Thinks I’m Dead (I know otherwise.) [R. Abrams, NYTimes, 2017]

• Not only Google, but also Alexa:
  - "Alexa replies that Rachel Abrams is a sprinter from the Northern Mariana Islands (which is true of someone else)."
  - "He asks if Rachel Abrams is deceased, and Alexa responds yes, citing information in the Knowledge Graph panel."
Data Storage

SQL DATABASES
- Relational

NoSQL DATABASES
- Column
- Graph
- Key-Value
- Document
Provenance and Reproducibility

Data Management

Visualization

Computation

DATA

DATA

Visualization

Paper

Data Management
Provenance and Reproducibility

Data Management

Computation

Visualization

Provenance

Paper
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)
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
In [5]: def solve_lorenz(N=10, max_time=4.0, sigma=10.0, beta=8.0/3, rho=28.0):
    ...:     
   ...:     
```
JupyterLab

- An interactive, configurable programming environment
- Supports many activities including notebooks
- Runs in your web browser
- Notebooks:
  - Originally designed for Python
  - Supports other languages, too
  - Displays results (even interactive maps) inline
  - You decide how to divide code into executable cells
  - Shift+Enter to execute a cell
Installing Python & JupyterLab

- www.anaconda.com/download/
- Anaconda has Jupyter Lab
- Use Python 3.7 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>
JupyterLab Notebook Tips

• Starts with a directory view
• Create new notebooks using the Launcher (+ icon on the left)
  - New notebooks have the name "Untitled"
  - File → Rename Notebook… (or right-click) to change the name
• Save a notebook using the command under the File menu
• Shutting down the notebook requires quitting the kernel
  - Web browser is interface to display code and results
  - Kernel actually runs the code: may see messages in a console/terminal window
  - Closing the browser window does not stop Jupyter
JupyterLab Notebooks

• Open a notebook using the left panel like you would in a desktop view
• Past results are displayed—does not mean they are loaded in memory
• Use "Run All" or "Run All Above" to re-execute past work
  - If you shut down the kernel, all of the data and variables you defined need to be redefined (so you need to re-run all)
  - **Watch Out—Order Matters:** If you went back and re-executed cells in a different order than they are shown, doing "Run All" may not produce the same results!
• Edit mode (green) versus Command mode (blue == Be Careful)
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
Other JupyterLab Features

• Terminal
  - Similar to what you see on turing/hopper but for your local machine

• File Viewers
  - CSV
  - Plugins available

• Console
  - Can be linked to notebooks
JupyterLab Documentation

• JupyterLab Tutorial Video
• JupyterLab Documentation
Jupyter Notebook

- Original Notebook Interface
- Just notebooks
- Same rich representations and text
- Same cell structure
- Same notebook files .ipynb
- Web-based
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
Learning Python Resources

- https://software-carpentry.org/lessons.html
- http://www.pythontutor.com
- http://www.python-course.eu
- http://thepythonguru.com
- https://en.wikibooks.org/wiki/A_Beginner%27s_Python_Tutorial
- https://learnpythonthehardway.org
- learnpython.org
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
    ```
Advanced: Python 2 and 3

- [https://docs.python.org/3/whatsnew/3.0.html](https://docs.python.org/3/whatsnew/3.0.html)
- **Key Differences:**
  - `print` as a function: `print "Hello"` vs. `print("Hello")`
  - Views and iterators instead of lists
  - Integer division: `5/2 = 2.5, 5//2 = 2`
  - Unicode as standard
  - String formatting:
    - Py2: "Hello %s. You are %d years old" % (name, age)
    - Py3: "Hello {}. You are {} years old".format(name, age)
    - Py3.6: f"Hello {name}. You are {age} years old"
In-Class Notebook

• Try out the examples from the following slides:
  - Download
  - View
Print function

- `print("Hello World")`
- Can also print variables:
  ```python
  name = "Jane"
  print("Hello," , name)
  ```
Python Variables and Types

- No type declaration necessary
- Variables are names, not memory locations
  
  ```python
  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)
Python Math and String "Math"

- Standard Operators: +, -, *, /, %
- Division "does what you want" (new in v3)
  - 5 / 2 = 2.5
  - 5 // 2 = 2  # use // for integer division
- Shortcuts: +=, -=, *=
- No ++, --
- Exponentiation (Power): **
- Order of operations and parentheses: (4 - 3 - 1 vs. 4 - (3 - 1))
- "abc" + "def"
- "abc" * 3
Python Strings

• Strings can be delimited by single or double quotes
  - "abc" and 'abc' are exactly the same thing
  - Easier use of quotes in strings: "Joe's" or 'He said "Stop!"'
• String concatenation: "abc" + "def"
• Repetition: "abc" * 3
• Special characters: \n \t like Java/C++
Python Strings

• Indexing:
  
  ```
  a = "abcdef"
  a[0]
  ```

• Slicing: ```a[1:3]```

• Format:
  
  ```
  name = "Jane"
  print("Hello, {}".format(name))
  ```
  
  - or

  ```
  print(f"Hello, {name}")
  ```
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)
Conditionals

• if, else
  - Again, indentation is required
• elif
  - Shorthand for else: if:
• Same type of boolean expressions (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

```python
• 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)
```
True and False

• True and False (capsitalized) are defined values in Python
• v == 0 will evaluate to either True or False
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++
}
```
Functions

- Calling functions is as expected:
  - `mul(2,3) # computes 2*3 (mul from operator package)`
  - Values passed to the function are parameters
  - May be variables!
    - `a = 5`
    - `b = 7`
    - `mul(a,b)`

- `print` is a function
  - `print("This line doesn't end.", end=" ")`
  - `print("See it continues")`
  - `end` is also a parameter, but this has a different syntax (keyword argument!)
Why do we create and use functions?
Defining Functions

- **def** keyword

- Arguments have names but **no types**
  
  ```python
def hello(name):
    print(f"Hello {name}\n")
  ```

- Can have defaults:
  ```python
def hello(name="Jane Doe"):  
    print(f"Hello {name}\n")
  ```

- With defaults, we can skip the parameter: `hello()` or `hello("John")`

- Also can pick and choose arguments:
  ```python
def hello(name1="Joe", name2="Jane"):  
    print(f"Hello {name1} and {name2}\n")
  hello(name2="Mary")
  ```
Return statement

• Return statement gives back a value:

```python
def mul(a,b):
    return a * b
```

• Variables changed in the function won't be updated:

```python
def increment(a):
    a += 1
    return a
b = 12
c = increment(b)
print(b,c)
```
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!
  • \texttt{len} function: number of items: \texttt{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:
    ```python
    [<start>:<end>]
    ```
  - Returns a new list (`b = a[:]`)  
  - Don't need to specify the beginning or end
Examples

• Suppose \( a = ['a', 'b', 'c', 'd'] \)
• What are?
  - \( a[0] \)
  - \( a[1:2] \)
  - \( a[3:] \)
  - \( a[:-2] \)
  - \( a[:-1] \)
Tuples

- months = ('January','February','March','April', 'May','June','July','August','September','October', 'November','December')

- Useful when you know you're not going to change the contents or add or delete values

- Can index and slice

- Also, can create new tuples from existing ones:
  - t = (1,2,3)
    u = (4,5,6)
  - v = t + u # v points to a new object
  - t += u # t is a new object
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!)
Modifying a list

- \( v = [1, 2, 3] \)
  \( w = [4, 5, 6] \)

- \( x = v + w \) # \( x \) is a new list \([1, 2, 3, 4, 5, 6]\)

- \( v.extend(w) \) # \( v \) is mutated to \([1, 2, 3, 4, 5, 6]\)

- \( v += w \) # \( v \) is mutated to \([1, 2, 3, 4, 5, 6]\)

- \( v.append(w) \) # \( v \) is mutated to \([1, 2, 3, [4, 5, 6]]\)

- \( x = v + 4 \) # error

- \( v += 4 \) # error

- \( v += [4] \) # \( v \) is mutated to \([1, 2, 3, 4]\)
in: Checking for a value

• The `in` operator:
  - 'a' in l
  - 'a' not in l

• Not very fast for lists
For loops

• Used much more frequently than while loops
• Is actually a "for-each" type of loop
• In Java, this is:
  - for (String item : someList) {
    System.out.println(item);
  }
• In Python, this is:
  - 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))
What about counting?

- In C++:
  ```
  for(int i = 0; i < 100; i++) {
    cout << i << endl;
  }
  ```

- In Python:
  ```
  for i in range(0,100): # or range(100)
    print(i)
  ```

- `range(100)` vs. `list(range(100))`

- What about only even integers?
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: \(d1.update(d2)\)
  - \texttt{update} overwrites any key-value pairs in \(d1\) when the same key appears in \(d2\)

• \texttt{len(d)} is the number of entries in \(d\)
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:
  
  ```
  [(k1, v1), (k2, v2), ...]
  ```
- Unpacking a tuple or list
  ```
  t = (1,2)
  a, b = t
  ```
- Iterating through a dictionary:
  ```
  for (k,v) in d.items():
      if k % 2 == 0:
          print(v)
  ```
- Important: keys, values, and items are not in any specific order!
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