Advanced Data Management (CSCI 640/490)

Python

Dr. David Koop
Supporting Data Science

Studying how taxi demand varies over time is accentuated during rush hour on weekdays, when trips take much longer. This is a common practice when the destination is JFK. Note that while it is illegal for taxis to reject rides, they tend to accept more trips to the airport. The airport is a key transportation hub in NYC. By analyzing taxi movement to and from these locations, we can obtain insights into how people move around the city. To compare the number of trips originating at and from these locations, we can obtain insights into how people move around the city.

For example, by examining the average cost of trip per mile, we can see that it is higher within Manhattan. This provides an economic incentive for taxi companies to stay within Manhattan and avoid driving outside of the city. In order to study the movement patterns for airports and train stations, we can select the regions in their vicinity and examine trips starting at the two major airports in NYC: JFK and La Guardia, and group them using the time selection widgets (Fig. 5).

By hovering the mouse over a neighborhood, the system displays the exact number of trips ending in that neighborhood. We can also obtain insights into how people move into and out of the city. To compare the number of trips originating at the airports with trips starting at the train stations, Penn Station and Grand Central, we select the regions and maximize profits. To simplify the process of comparing multiple trips for the two years is very similar, including the significant drop in both volume and fare per mile is lower for Harlem, and thus, there is less economic incentive for drivers that go to Harlem.

Using the summary view, we can further explore features of the trips we found one surprising fact: the number of trips to/from Harlem compared to other more affluent neighborhoods (see Fig. 6). Further analysis also showed that the fare per mile is lower for Harlem, and thus, there is less economic incentive for drivers that go to Harlem. Note that while in this example we have focused on pickups, i.e., people arriving, it is easy to also study dropoffs. Starting from the map view shown in Fig. 7, we can group them (Fig. 8).

The user first selects the time slices of interest. This can be done using the time selection widgets (Fig. 9). The system then assigns to its time range. This is illustrated in Fig. 10. The list of time ranges is already expressed and generated by time intervals. Each map view and plot line is associated with a color parameter associated with the trips. The system displays the exact number of trips ending in that neighborhood. This one-week overview provides an accurate overview of city life, where people go and when. It also highlights social inequalities. People who live in Harlem have long complained about the lack of taxi service in their neighborhood. The plot clearly shows that their disconnection from the rest of the city is accentuated during rush hour on weekdays, when trips take much longer. This is a common practice when the destination is JFK.
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
field. They cited things in their own practice, how
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.

As data science becomes more commonplace and
simultaneously a bit demystified, we expect this
trend to continue as well. After all, last year's
respondents were just as excited about their
work (about 79% were “satisfied” or better).

How a Data Scientist Spends Their Day

Here’s where the popular view of data scientists diverges pretty significantly from reality. Generally,
we think of data scientists building algorithms, exploring data, and doing predictive analysis. That’s
actually not what they spend most of their time doing, however.

As you can see from the chart above, 3 out of every 5 data scientists we surveyed actually spend the
most time cleaning and organizing data. You may have heard this referred to as “data wrangling” or
compared to digital janitor work. Everything from list verification to removing commas to debugging
databases—-that time adds up and it adds up immensely. Messy data is by far the more time-
consuming aspect of the typical data scientist’s work flow. And nearly 60% said they simply spent too much
time doing it.

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]
Although domain-specific functions exist, they may not cover every type of transformation needed. We choose programs, sometimes with multiple functions, to exactly match the desired transformation. The front-end of Transform-Data-by-Example (TDE) is an extensible data transformation system called TDE: Extensible Data Transformation in Excel. In this work we focus on transform-... for date-time. (Left): input data is in column-C, user provides two desired output examples in columns-D. (Right): output is produced automatically.

**Unique Features**
- Head-domain Support.
- Extensibility.
- Search-by-Example.
- Automation.
- Transformations.
- Sharing.
- Business intelligence.
- Self-service.
- Program Synthesis.
- Users such as business analysts and data scientists today increasingly need to clean, standardize and transform diverse data sets, such as name, phone-number, us-address, url, unit-conversion, etc. Many of these transformations cannot be handled by any existing system. Therefore, we have built an instance of Transform-Data-by-Example (TDE) that can index rich transformations extracted from code, c source code, DLLs, web services, and mapping tables. We choose functions from GitHub that can already handle the desired transformation. However, those functions only produce the correctly transformed records and not the original data. Hence, we need to find a way to derive the original data from the transformed data. We observe that deriving the original data from a transformed data is essentially data recovery, and that these domain-specific functions can also handle data recovery.

As mentioned earlier, there are domain-specific functions that can already handle common transformations such as phone-number, name, url, unit-conversion, etc. In the next scenario, we demonstrate how Transform-Data-by-Example can be used to derive data from a transformed data set.

**First Scenario**

Suppose a user wants to obtain the names and addresses of customers from a set of transaction data. The user provides two examples of output, namely the first three values in column-D are provided as output examples. The desired target output, all within just a few seconds. Expert-users have the...
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': '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',

Carnegie Mellon University
Microsoft
Stanford University
Columbia University
UC Berkeley
Google
Microsoft
Carnegie Mellon University

[ICML, NIPS, EMNLP, NAACL, EACL, ACL]
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**

**NoSQL DATABASES**

- Column
- Graph
- Key-Value
- Document

---

V. Wilkinson
Scaling Dataframes

New Data Source → Prototyping → Exploring → Testing → Production

New spec
New requirements

Laptop/Workstation → Small Cluster → Large Cluster

MODIN

Feedback

[D. Petersohn]
Provenance and Reproducibility

Data Management

Visualization

Computation

Paper

DATA

DATA
Provenance and Reproducibility

Data Management

Data

Visualization

Provenance

Computation

Paper

DATA

DATA
About this course

- Course web page is authoritative:
  - faculty.cs.niu.edu/~dakoop/cs640-2024sp/
  - 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
Course Material

• Helpful Books:
  - Python for Data Analysis, W. McKinney
  - Effective Pandas, M. Harrison
  - Intro to Python, Deitel & Deitel
  - Python Data Science Handbook, J. VanderPlas

• Research papers
• Many websites
Syllabus Questions?
Class Roster Check
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_t = 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 the Lorenz differential equations."""
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1, projection='3d')
    ax.axis('off')
    ax.set_xlim(-25, 25)
    ax.set_ylim(-25, 25)
    ax.set_zlim(-25, 25)
    x, y, z = x_t
    return (sigma * (y - x), x * (rho - z) - y, x * y - beta * z)

def lorenz_deriv(x_y_z, t, sigma=sigma, beta=beta, rho=rho):
    """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]

# Choose random starting points, uniformly distributed from -15 to 15
np.random.seed(1)
x0 = -15 + 30 * np.random.random((N, 3))
```
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 3.12 version
- 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 runs the code: may see messages in a console/terminal window
  - Closing the browser window does not stop Jupyter
  - Use File → Shut Down to shut down everything
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 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
JupyterLab Outputs

- stdout: where print commands go
- stderr: where error messages go
- display: special output channel used to show rich outputs
- output: same as display but used to display the value of the last line of a cell
JupyterLab Output Types

```python
[2]: a = 12
    for i in range(3):
        print("Some output")
    plt.bar([1,2,3,4],[20,30,15,40])
    plt.show()
    a + 3
stdout
Some output
Some output
Some output
```

```
[2]:
```
```
output
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
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
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
  - Easy to wrap code from other languages like C/C++
- Open-source with a large, passionate community
Learning Python Resources

- Python for Programmers
- http://www.pythontutor.com
- https://www.python-course.eu
- https://software-carpentry.org/lessons/
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
    
    ```
    z = 20
    if x > 0:  
        if y > 0:  
            z = 100
        else:  
            z = 10
    ```
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
  
  ```
  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:**
  
  ```python
  a = "abcdef"
  a[0]
  ```

- **Slicing:** `a[1:3]`

- **Format:**
  
  ```python
  name = "Jane"
  print("Hello, {}".format(name))
  
  - or
  
  print(f"Hello, {name}")
  ```
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 (captialized) are defined values in Python
- \( v == 0 \) will evaluate to either True or False
Why do we create and use functions?
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!)
Defining Functions

- `def` keyword

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

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

- 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}\")
  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!
  • len function: number of items: len(l)
Indexing (Positive and Negative)

• Positive indices start at zero, negative at -1
• my_str = "abcde"; my_str[1] # "b"
• my_list = [1,2,3,4,5]; my_list[-3] # 3
• my_tuple = (1,2,3,4,5); my_tuple[-5] # 1
Slicing

• Positive or negative indices can be used at any step
• `my_str = "abcde"; my_str[1:3] # ["b", c]`
• `my_list = [1,2,3,4,5]; my_list[3:-1] # [4]`

• Implicit indices
  - `my_tuple = (1,2,3,4,5); my_tuple[-2:] # (4,5)`
  - `my_tuple[:3] # (1,2,3)`

```
[1:3]
[0 1 2 3 4]
```
```
[-4:-2]
[ a b c d e ]
```
```
[0 1 2 3 4]
```
```
[-5 -4 -3 -2 -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++:
  ```cpp
  for(int i = 0; i < 100; i++) {
    cout << i << endl;
  }
  ```

- In Python:
  ```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)`
  - `update` overwrites any key-value pairs in `d1` when the same key appears in `d2`
  - `d1 | d2`

• `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: 
  \[(k1, v1), (k2, v2), \ldots\]

- Unpacking a tuple or list
  - `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!
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
Exercises
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"
Exercise

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

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