Introduction / Notebooks / Fundament Data Types

Week 1: Computer Programming for Data Scientists (7CCSMCMP)

26 Sept 2016

Event: Wed 28 Sept. 15:00 to 17:00

https://www.eventbrite.co.uk/e/special-guest-speaker-dr-cathy-oneil-weapons-of-math-destruction-tickets-27673653643

Special Guest Speaker: Dr Cathy O'Neil, Weapons of Math Destruction

by King's College London

Free

REGISTER

DESCRIPTION

We live in the age of the algorithm. Increasingly, companies use mathematical models to make decisions for everything from sorting CVs to granting mortgages, targeting voters to monitoring health. Most people think mathematical modelling is objective, because they are intimidated by math and because they trust numbers. In fact, Cathy O'Neil argues, algorithms are increasing inequality and threatening democracy.

Cathy O'Neil is the first person to explore how Big Data is being used to discriminate against people based on income, race and class. In this urgent, revealing and terrifying book, O'Neil shows that the algorithms we use are opaque and unregulated, even when they're wrong. As more and more of our lives are handled by computers, what can we do to stop ourselves falling victim to these unaccountable and under regulated systems?

DATE AND TIME

Wed 28 September 2016
15:00 – 17:00
Add to Calendar

LOCATION

S-2.08
King's College London
Strand
London
View Map

FRIENDS WHO ARE GOING

Lecturers

Chipp Jansen

chipp.jansen@kcl.ac.uk

Office Hours (**S6.03**)
Thursday 12pm to 2pm

Background Computer Science, Fine Arts, and Geography

4 years Industry Experience at agent-based modeling company, Data Visualisation, Platform Dev

Isabel Sassoon

isabel.sassoon@kcl.ac.uk

Office Hours (**S6.33**)
Thursday 10am to 12pm

Background PhD Research in Data Science, Medical Data Focus

12 year Industry Experience as Data Scientist at SAS (statistics and data mining)

Teaching Assistants (Labs and Tutorials)

Tsvetan Zhivkov tsvetan.zhivkov@kcl.ac.uk

Jasper Schulz jasper.schulz@kcl.ac.uk

Sundararam Rathod sundararam.rathod@kcl.ac.uk

Today

- Module Overview / Schedule
- Module Resources
- Data Science Workflow
- Topics from Last Week's Mini-Course
- Example from Mini-Course Lab Reviewing Topics
- Fundamental Python Data Types
    - lists
    - dictionaries
    - tuple
    - sets
- Development Environments: Script-based vs Notebook
- Package Management
- Class Exercise

Module Activities

Each Lecture Topic: **Two** 1 hour lectures

Friday, 2-3 pm (STRAND S-1.27) ***Strand Floor -1***

Monday, 4-5 pm (WAT/F-WB1.16)

Practical Lab Sessions

Either Thursdays, 2pm to 4pm (S4.01)

Or Fridays, 11am to 1pm (K4U.13/14)

Practicals start **THIS** week

Tutorial Sessions

Thursdays, 5-6 pm (WAT/F-WB1.16)

No Tutorial THIS week

Topics Schedule

Topic	Week	Start	Who
Introduction / Fundamentals	1	26-Sept	Chipp
Intro to Handling Data	1/2	30-Sept	Chipp
Abstract Data Structures Objects	2/3	7-Oct	Chipp
Numerical Computing	3/4	14-Oct	Isabel
Graphics and Interactivity with Python	4/5	21-Oct	Both
Structured Data with Pandas 1	5	28-Oct	Isabel
READING WEEK	6	31-Oct	
Structured Data with Pandas 2	7	7-Nov	Isabel
Spatial and Geographical Processing	7/8	11-Nov	Chipp
Introductions to Graphs	8/9	18-Nov	Isabel
Natural Language Processing	9/10	15-Nov	Isabel
High Performance Computing	10/11	2-Dec	Chipp
End-to-End Overview w/ Dataset	11	9-Dec	Isabel
EXAM REVISION	12	12-Dec	Both

Module Assessment

January Exam (80%)

Coursework / Quiz (20%)

Coursework Programming Assignments, turned in on KEATs

Quiz is Multiple Choice, conducted during Week 6 Tutorial

What	Worth	Released	Due
Coursework 1	5%	Friday, 14-Oct	Sunday, 5-Nov
Coursework 2	5%	Friday, 11-Nov	Sunday, 27-Nov
Coursework 3	5%	Friday, 2-Dec	Sunday, 18-Dec
Quiz	5%	*During Week 6 Tutorial (10-Nov)*	

Feedback on Coursework / Quizzes within 3 Weeks

Resources

- **KEATs Page** "*Computer Programming - 7CCSMCMP (2016-17)*"

  https://keats.kcl.ac.uk/course/view.php?id=38066

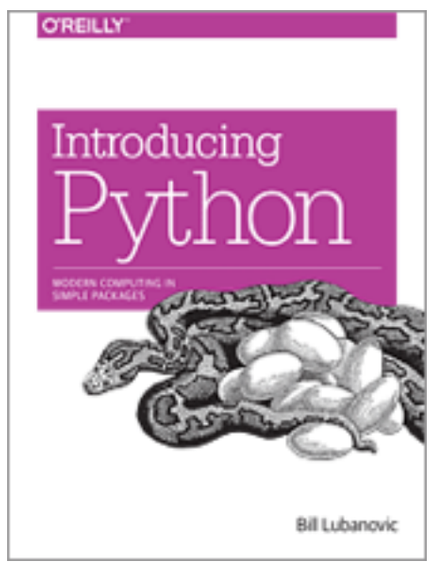
  Discussion Group for Questions ⟶ 

  Lecture Capture ⟶ 

- **Books (optional)**

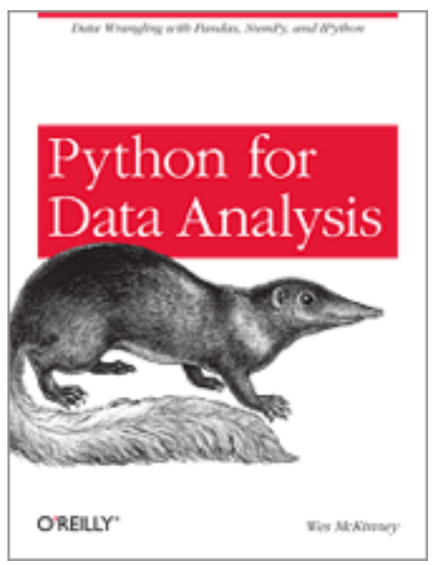
Introducing Python, 1st Ed.
by: Bill Lubanovic
Publisher: O'Reilly Media (2014)

http://shop.oreilly.com/product/0636920028659.do



Python for Data Analysis, 1st Ed.
by: Wes McKinney
Publisher: O'Reilly Media (2012)

http://shop.oreilly.com/product/0636920023784.do



- **Online Resources**, see end of Lecture Slides

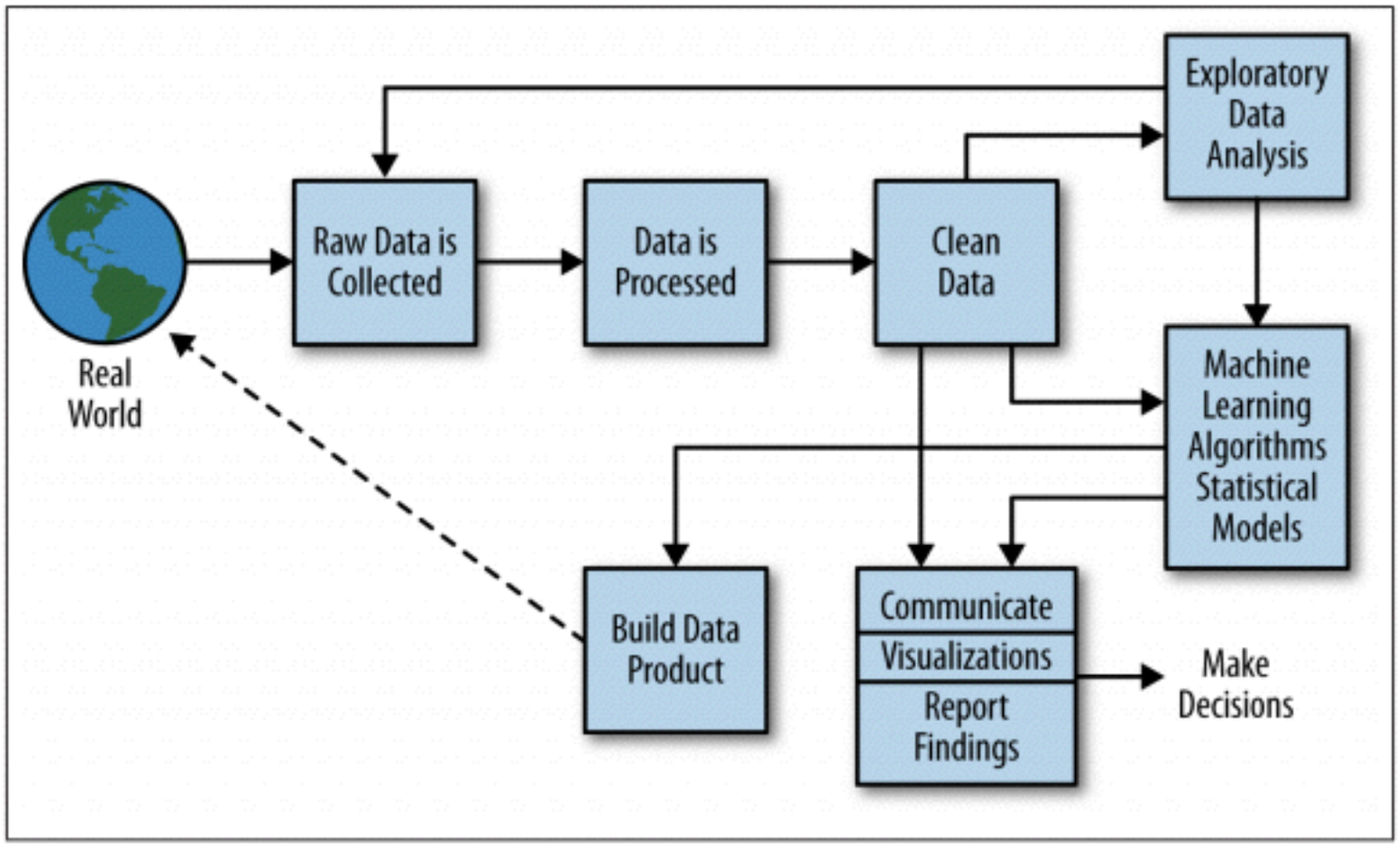
Why Python for Data Science?

- Python is an *interpreted* language (as opposed to *compiled* like C/C++/FORTRAN)
- Programmers love it because, it is:
  - *dynamic* (i.e. single line of code will run in the Python interpreter)

```
print("hello world!")
```

  - used as *glue* language, existing legacy scientific computing software (written in C/C++/FORTRAN) can be combined as *software libraries* in python
- has many software libraries useful in data analysis (i.e. *numpy*, *pandas*, *matplotlib*, *scipy*)
- used for both *exploratory data analysis* as well as building *production systems*
- *object-oriented* (structured around *objects* that contain *variables* and *methods,* where methods act on variables)

Data Science Workflow

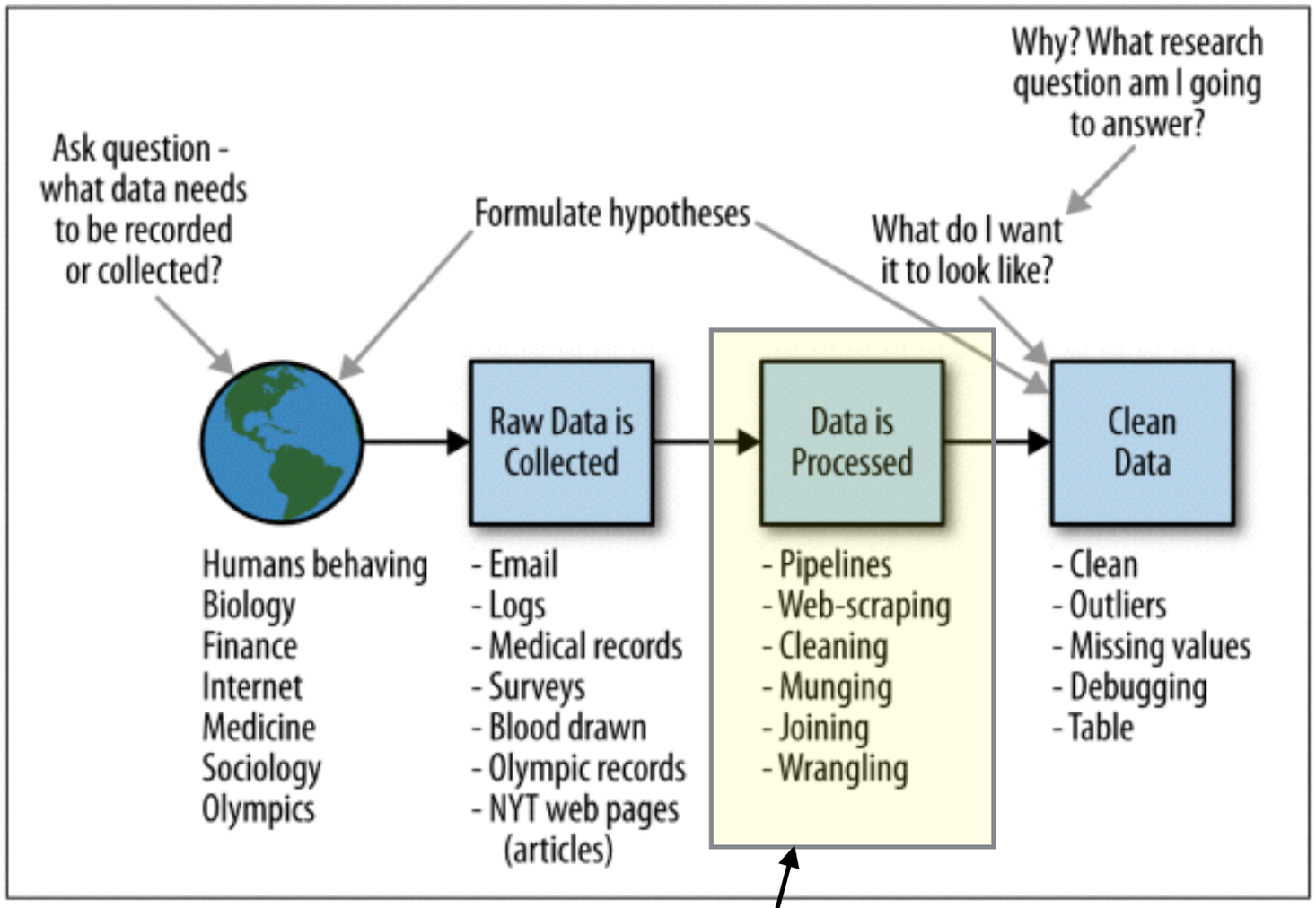


- Data Science iterates through this process

- Python as a programming language can be used in each of these steps

From Schutt, R and O'Neil, C (2013). *Doing Data Science*. O'reilly Publishing. pp 43-44

Data Science Workflow



- In particular **Data Processing**
- In this module, we will focus on getting you familiar with python programming

From Schutt, R and O'Neil, C (2013). *Doing Data Science*. O'reilly Publishing. pp 43-44

Data Science Induction Mini-Course

https://keats.kcl.ac.uk/course/view.php?id=42370

Introduction to Python

Part 1	Output (i.e. `print("hello world")`) Data types (integers, float, boolean) Strings Operators (assignment, mathematical, Boolean, relational)
Part 2	Branching (using `if` statements) Looping (using a `while` loop and a `for` loop) Random numbers (using the `random` module)
Part 3	Functions Writing to a file with Python
Part 4	Fundamental Data Structures in Python - lists - dictionaries Classes

Example from Lab 4 Part 1 from Data Science Mini-Course:

Country Metals from the Paralympics

http://www.bbc.co.uk/sport/paralympics/rio-2016/medals/countries

Write a Python program to store and total the number of medals for some countries

Medal Table

By Country | By Sport

#	Country				Total	
1	China	107	81	51	**239**	v
2	Great Britain & N. Ireland	64	39	44	**147**	v
3	Ukraine	41	37	39	**117**	v
4	United States	40	44	31	**115**	v
5	Australia	22	30	29	**81**	v

Example [1] from Lab 4 Part 1 from Data Science Mini-Course:

*Use an integer **variable** and output medals as a **concatenated** string*

```
# UK Gold Medals
uk = 64

print("Rio 2016 Paralympics")
print("uk: " + str(uk) + " gold medals")
```

```
Rio 2016 Paralympics
uk: 64 gold medals
```

By Country | By Sport

#	Country				Total	
1	China	107	81	51	**239**	∨
2	Great Britain & N. Ireland	64	39	44	**147**	∨
3	Ukraine	41	37	39	**117**	∨
4	United States	40	44	31	**115**	∨
5	Australia	22	30	29	**81**	∨
6	Germany	18	25	14	**57**	∨
7	Netherlands	17	19	26	**62**	∨

Example [2] from Lab 4 Part 1 from Data Science Mini-Course:

Multiple variables store UK's medals

```
# Medals for the UK
uk_gold = 64
uk_silver = 39
uk_bronze = 44

print("Rio 2016 Paralympics")
print("uk: " + str(uk_gold) + " gold medals")
print("uk: " + str(uk_silver) + " silver medals")
print("uk: " + str(uk_bronze) + " bronze medals")
```

```
Rio 2016 Paralympics
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
```

2		Great Britain & N. Ireland	64	39	44	147
3		Ukraine	41	37	39	117
4		United States	40	44	31	115
5		Australia	22	30	29	81
6		Germany	18	25	14	57
7		Netherlands	17	19	26	62

Example [3] from Lab 4 Part 1 from Data Science Mini-Course**:**

```
# Medals for the UK using a Dictionary
uk = {}
uk["gold"] = 64
uk["silver"] = 39
uk["bronze"] = 44

print("Rio 2016 Paralympics")
print("uk: " + str(uk["gold"]) + " gold medals")
print("uk: " + str(uk["silver"]) + " silver medals")
print("uk: " + str(uk["bronze"]) + " bronze medals")
```

```
Rio 2016 Paralympics
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
```

*Store medals numbers in a **dictionary {}**, which is **key-value map***

Dictionaries can store mixed data-types: ints, strings, other dictionaries!

Example [4] from Lab 4 Part 1 from Data Science Mini-Course:

```
# Medals for the UK using a Dictionary
uk = {}
uk["gold"] = 64
uk["silver"] = 39
uk["bronze"] = 44

print("Rio 2016 Paralympics")
print("uk: " + str(uk["gold"]) + " gold medals")
print("uk: " + str(uk["silver"]) + " silver medals")
print("uk: " + str(uk["bronze"]) + " bronze medals")
```

```
Rio 2016 Paralympics
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
```

```
# You can print Dictionaries
print(uk)
```

```
{'bronze': 44, 'silver': 39, 'gold': 64}
```

key *value*

Strings can use *single-quotes* as well

Example [5] from Lab 4 Part 1 from Data Science Mini-Course:

```
# Medals for the UK using a Dictionary
uk = {}
uk["gold"] = 64
uk["silver"] = 39
uk["bronze"] = 44

# Medal colours stored in a list
medal_colours = ["gold", "silver", "bronze"]

print("Rio 2016 Paralympics")

# Use a for loop to print out UK's medals
for m in medal_colours:
    print("uk: " + str(uk[m]) + " " + m + " medals")
```

```
Rio 2016 Paralympics
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
```

*Use a **list** to store types of medals, and a **for loop** to print out medals*

Example [5] from Lab 4 Part 1 from Data Science Mini-Course:

```
# Medals for the UK using a Dictionary
uk = {}
uk["gold"] = 64
uk["silver"] = 39
uk["bronze"] = 44

# Medal colours stored in a list
medal_colours = ["gold", "silver", "bronze"]

print("Rio 2016 Paralympics")

# Use a for loop to print out UK's medals
for m in medal_colours:
    print("uk: " + str(uk[m]) + " " + m + " medals")
```

```
Rio 2016 Paralympics
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
```

*Python uses **indentation** (tabs or spaces) to structure **code blocks***

Example [6] from Lab 4 Part 1 from Data Science Mini-Course:

```
# What happens when I change the list?
uk = {}
uk["gold"] = 64
uk["silver"] = 39
uk["bronze"] = 44

# Medal colours stored in a list
medal_colours = ["bronze", "gold", "silver"]

print("Rio 2016 Paralympics")

# Use a for loop to print out UK's medals
for m in medal_colours:
    print("uk: " + str(uk[m]) + " " + m + " medals")
```

```
Rio 2016 Paralympics
uk: 44 bronze medals
uk: 64 gold medals
uk: 39 silver medals
```

*Lists are **ordered**, notice the order changed in the **for-loop***

Example [7] from Lab 4 Part 1 from Data Science Mini-Course:

```
# What happens when I change the list?
uk = {}
uk["gold"] = 64
uk["silver"] = 39
uk["bronze"] = 44

# Medal colours stored in a list
medal_colours = ["bronze", "lead", "gold", "silver"]

print("Rio 2016 Paralympics")

# Use a for loop to print out UK's medals
for m in medal_colours:
    print("uk: " + str(uk[m]) + " " + m + " medals")
```

```
Rio 2016 Paralympics
uk: 44 bronze medals
---------------------------------------------------------------------
--
KeyError                                  Traceback (most recent call las
t)
<ipython-input-21-641d7b5cfb11> in <module>()
     12 # Use a for loop to print out UK's medals
     13 for m in medals:
---> 14     print("uk: " + str(uk[m]) + " " + m + " medals")

KeyError: 'lead'
```

Example [8] from Lab 4 Part 1 from Data Science Mini-Course:

```
# Add some more countries
uk = {"bronze": 44, "gold": 64, "silver": 39}
cub = {"gold": 8, "silver": 1, "bronze": 6}
kr = {"silver": 11, "bronze": 17, "gold": 7}

# Medal colours stored in a list
medal_colours = ["gold", "silver", "bronze"]

print("Rio 2016 Paralympics")

# Use a for loop to print out UK's medals
for m in medal_colours:
    print("uk: " + str(uk[m]) + " " + m + " medals")

# Cuba
for m in medal_colours:
    print("cub: " + str(cub[m]) + " " + m + " medals")

# South Korea
for m in medal_colours:
    print("kr: " + str(kr[m]) + " " + m + " medals")
```

```
Rio 2016 Paralympics
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
cub: 8 gold medals
cub: 1 silver medals
cub: 6 bronze medals
kr: 7 gold medals
kr: 11 silver medals
kr: 17 bronze medals
```

Let's add some more countries, and loops

Example [9] from Lab 4 Part 1 from Data Science Mini-Course:

```
# Use a function to re-use code
uk = {"bronze": 44, "gold": 64, "silver": 39}
cub = {"gold": 8, "silver": 1, "bronze": 6}
kr = {"silver": 11, "bronze": 17, "gold": 7}

# function to print out a country's medals
def print_medals(country):
    # Medal colours stored in a list
    medal_colours = ["gold", "silver", "bronze"]

    # Use a for loop to print out UK's medals
    for m in medal_colours:
        print("uk: " + str(country[m]) + " " + m + " medals")


print("Rio 2016 Paralympics")

print_medals(uk)
print_medals(cub)
print_medals(kr)
```

```
Rio 2016 Paralympics
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
uk: 8 gold medals
uk: 1 silver medals
uk: 6 bronze medals
uk: 7 gold medals
uk: 11 silver medals
uk: 17 bronze medals
```

*Generalize the process of printing medals with a **function***

What's wrong here?

Example [9] from Lab 4 Part 1 from Data Science Mini-Course:

```
# Use a function to re-use code
uk = {"bronze": 44, "gold": 64, "silver": 39}
cub = {"gold": 8, "silver": 1, "bronze": 6}
kr = {"silver": 11, "bronze": 17, "gold": 7}

# function to print out a country's medals
def print_medals(country):
    # Medal colours stored in a list
    medal_colours = ["gold", "silver", "bronze"]

    # Use a for loop to print out UK's medals
    for m in medal_colours:
        print("uk: " + str(country[m]) + " " + m + " medals")


print("Rio 2016 Paralympics")

print_medals(uk)
print_medals(cub)
print_medals(kr)
```

```
Rio 2016 Paralympics
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
uk: 8 gold medals
uk: 1 silver medals
uk: 6 bronze medals
uk: 7 gold medals
uk: 11 silver medals
uk: 17 bronze medals
```

*"**`uk:`**" is a static string, and is printing for any country*

Example [10] from Lab 4 Part 1 from Data Science Mini-Course:

```
# Include the country name as a value in the dictionary
uk = {"bronze": 44, "gold": 64, "silver": 39, "name": "uk"}
cub = {"gold": 8, "silver": 1, "bronze": 6, "name": "cuba"}
kr = {"silver": 11, "bronze": 17, "gold": 7, "name": "south korea"}

# function to print out a country's medals
def print_medals(country):
    # Medal colours stored in a list
    medal_colours = ["gold", "silver", "bronze"]

    # Use a for loop to print out a country's medals
    for m in medal_colours:
        print(country["name"] + ": " + str(country[m]) + " " + m + " medals")

print("Rio 2016 Paralympics")

print_medals(uk)
print_medals(cub)
print_medals(kr)
```

```
Rio 2016 Paralympics
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
cuba: 8 gold medals
cuba: 1 silver medals
cuba: 6 bronze medals
south korea: 7 gold medals
south korea: 11 silver medals
south korea: 17 bronze medals
```

Add the country name to the dictionary to print correctly

Example [10] from Lab 4 Part 1 from Data Science Mini-Course:

```
# Include the country name as a value in the dictionary
uk = {"bronze": 44, "gold": 64, "silver": 39, "name": "uk"}
cub = {"gold": 8, "silver": 1, "bronze": 6, "name": "cuba"}
kr = {"silver": 11, "bronze": 17, "gold": 7, "name": "south korea"}

# function to print out a country's medals
def print_medals(country):
    # Medal colours stored in a list
    medal_colours = ["gold", "silver", "bronze"]

    # Use a for loop to print out a country's medals
    for m in medal_colours:
        print(country["name"] + ": " + str(country[m]) + " " + m + " medals")

print("Rio 2016 Paralympics")

print_medals(uk)
print_medals(cub)
print_medals(kr)
```

```
Rio 2016 Paralympics
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
cuba: 8 gold medals
cuba: 1 silver medals
cuba: 6 bronze medals
south korea: 7 gold medals
south korea: 11 silver medals
south korea: 17 bronze medals
```

How shall we account for the singular/plural grammar?

Example [11] from Lab 4 Part 1 from Data Science Mini-Course:

```
# Use branching to print out a singular string
uk = {"bronze": 44, "gold": 64, "silver": 39, "name": "uk"}
cub = {"gold": 8, "silver": 1, "bronze": 6, "name": "cuba"}
kr = {"silver": 11, "bronze": 17, "gold": 7, "name": "south korea"}

# function to print out a country's medals
def print_medals(country):
    # Medal colours stored in a list
    medal_colours = ["gold", "silver", "bronze"]

    # Use a for loop to print out a country's medals
    for m in medal_colours:
        if(country[m] == 1): # single medal
            print(country["name"] + ": " + str(country[m]) + " " + m + " medal")
        else: # plural medals
            print(country["name"] + ": " + str(country[m]) + " " + m + " medals")
```

```
print("Rio 2016 Paralympics")

print_medals(uk)
print_medals(cub)
print_medals(kr)
```

```
Rio 2016 Paralympics
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
cuba: 8 gold medals
cuba: 1 silver medal
cuba: 6 bronze medals
south korea: 7 gold medals
south korea: 11 silver medals
south korea: 17 bronze medals
```

*Use **branching**, an **if** statement, to handle the singular case*

Example [12] from Lab 4 Part 1 from Data Science Mini-Course:

```
# Alternativly, create a string, and add an 's'
uk = {"bronze": 44, "gold": 64, "silver": 39, "name": "uk"}
cub = {"gold": 8, "silver": 1, "bronze": 6, "name": "cuba"}
kr = {"silver": 11, "bronze": 17, "gold": 7, "name": "south korea"}

# function to print out a country's medals
def print_medals(country):
    # Medal colours stored in a list
    medal_colours = ["gold", "silver", "bronze"]

    # Use a for loop to print out a country's medals
    for m in medal_colours:
        medal_string = country["name"] + ": " + str(country[m]) + " " + m + " medal"
        if(country[m] > 1): # plural medals, add an 's'
            medal_string = medal_string + "s"
        print(medal_string)

print("Rio 2016 Paralympics")

print_medals(uk)
print_medals(cub)
print_medals(kr)
```

```
Rio 2016 Paralympics
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
cuba: 8 gold medals
cuba: 1 silver medal
cuba: 6 bronze medals
south korea: 7 gold medals
south korea: 11 silver medals
south korea: 17 bronze medals
```

*Alternatively, use a **string variable,** and add an 's' if plural*

Example [13] from Lab 4 Part 1 from Data Science Mini-Course**:**

```
# Use a for loop to print out all of the countries
uk = {"bronze": 44, "gold": 64, "silver": 39, "name": "uk"}
cub = {"gold": 8, "silver": 1, "bronze": 6, "name": "cuba"}
kr = {"silver": 11, "bronze": 17, "gold": 7, "name": "south korea"}

# Store the Country dictionaries in a list()
countries = [cub, kr, uk]

# function to print out a country's medals
def print_medals(country):
    # Medal colours stored in a list
    medal_colours = ["gold", "silver", "bronze"]

    # Use a for loop to print out a country's medals
    for m in medal_colours:
        medal_string = country["name"] + ": " + str(country[m]) + " " + m + " medal"
        if(country[m] > 1): # plural medals, add an 's'
            medal_string = medal_string + "s"
        print(medal_string)

print("Rio 2016 Paralympics")

# Use a loop to print out all of the countries
for cntry in countries:
    print_medals(cntry)
```

*Put the countries in a **list**, use another **for-loop** to print out the medals*

```
Rio 2016 Paralympics
cuba: 8 gold medals
cuba: 1 silver medal
cuba: 6 bronze medals
south korea: 7 gold medals
south korea: 11 silver medals
south korea: 17 bronze medals
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
```

Example [14] from Lab 4 Part 1 from Data Science Mini-Course:

```
# Finally print out the total number of medals
uk = {"bronze": 44, "gold": 64, "silver": 39, "name": "uk"}
cub = {"gold": 8, "silver": 1, "bronze": 6, "name": "cuba"}
kr = {"silver": 11, "bronze": 17, "gold": 7, "name": "south korea"}

# Store the Country dictionaries in a list()
countries = [cub, kr, uk]

# function to print out a country's medals
def print_medals(country):
    # Medal colours stored in a list
    medal_colours = ["gold", "silver", "bronze"]

    # Use a for loop to print out a country's medals
    for m in medal_colours:
        medal_string = country["name"] + ": " + str(country[m]) + " " + m + " medal"
        if(country[m] > 1): # plural medals, add an 's'
            medal_string = medal_string + "s"
        print(medal_string)

# function to return the total number of medals for a country
def total_medals(country):
    total = 0

    # Medal colours stored in a list
    medal_colours = ["gold", "silver", "bronze"]

    # Use a for loop to total the number of metals
    for m in medal_colours:
        total = total + country[m]

    # return the total number of medals as function value
    return total

print("Rio 2016 Paralympics")

# Use a loop to print out all of the countries
```

example continues

Example [14] continues…

```
# function to return the total number of medals for a country
def total_medals(country):
    total = 0

    # Medal colours stored in a list
    medal_colours = ["gold", "silver", "bronze"]

    # Use a for loop to total the number of metals
    for m in medal_colours:
        total = total + country[m]

    # return the total number of medals as function value
    return total

print("Rio 2016 Paralympics")

# Use a loop to print out all of the countries
for cntry in countries:
    print_medals(cntry)
    print("total: " + str(total_medals(cntry)))
```

```
Rio 2016 Paralympics
cuba: 8 gold medals
cuba: 1 silver medal
cuba: 6 bronze medals
total: 15
south korea: 7 gold medals
south korea: 11 silver medals
south korea: 17 bronze medals
total: 35
uk: 64 gold medals
uk: 39 silver medals
uk: 44 bronze medals
total: 147
```

*Add a new function to calculate the **total** number of medals*

*The function **returns** the total value, and the return value is printed out*

Fundamental Data Structures

Lists - `["gold", "silver", "bronze"]`

Dictionaries - `{"gold":7, "silver":11, "bronze": 17}`

Tuples - `("gold", 7)`

Sets - `set(["gold", "silver", "bronze"])`

Fundamental Data Structures - tuples

- a **tuple** is like a list, but it is *immutable*, meaning that its contents cannot be changed (like a constant)

```
# example comparing list and tuple
mytup = ("hello", "and", "goodbye")
mylist = ["hello", "and", "goodbye"]
print("mytup =")
for i in mytup:
    print(i)
print("mylist =")
for i in mylist:
    print(i)
mylist[1] = "or"
mytup[1] = "or"
print("done")
```

- Your Program will CRASH here, because you cannot change values in **`mytup`**!!!

Fundamental Data Structures - tuples

- output:

```
mytup =
hello
and
goodbye
mylist =
hello
and
goodbye
```

```
---------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
<ipython-input-44-937fa2e40b85> in <module>()
      9     print(i)
     10 mylist[1] = "or"
---> 11 mytup[1] = "or"
     12 print("done")

TypeError: 'tuple' object does not support item assignment
```

Fundamental Data Structures - sets

- a **set** is an *unordered* collection of items, with *no duplicates*
- the `set()` function is used to create a set
- the `in` operator is used to indicate whether an item is in a set or not
- the `enumerate` function can be used to enumerate the items in a set

```
# Example of working with a set
myset = set( ["hello", "aloha", "bonjour"] )
j = "hello" in myset
k = "goodbye" in myset
print "Is hello in the set? " + str(j) + " and goodbye? " + str(k)
for i in enumerate(myset):
    print i
print "done"
```

```
Is hello in the set? True and goodbye? False
(0, 'bonjour')
(1, 'hello')
(2, 'aloha')
done
```

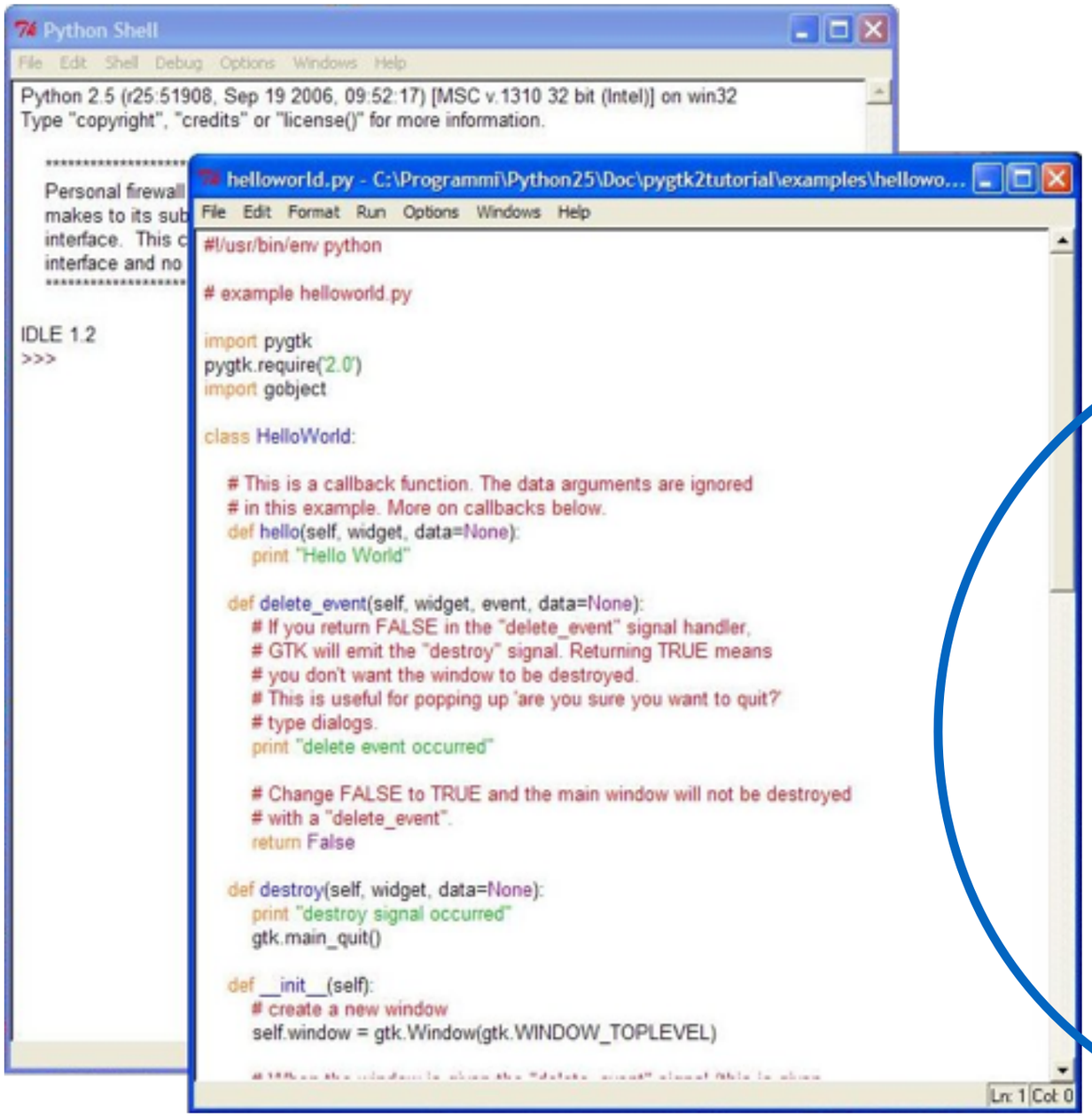
Fundamental Data Structures - sets

- you can check whether an item is in a set, but you cannot index items in the set; e.g., saying `set[1]` will crash your programme

- you can use *set operators* on sets:

  difference: –
  union: |
  intersection: `&`
  union minus intersection: ^

Development Environment - Text Editor / Command Prompt



(1) Edit python script (.py) file (i.e. text file)

(2) Run the python script on command-line

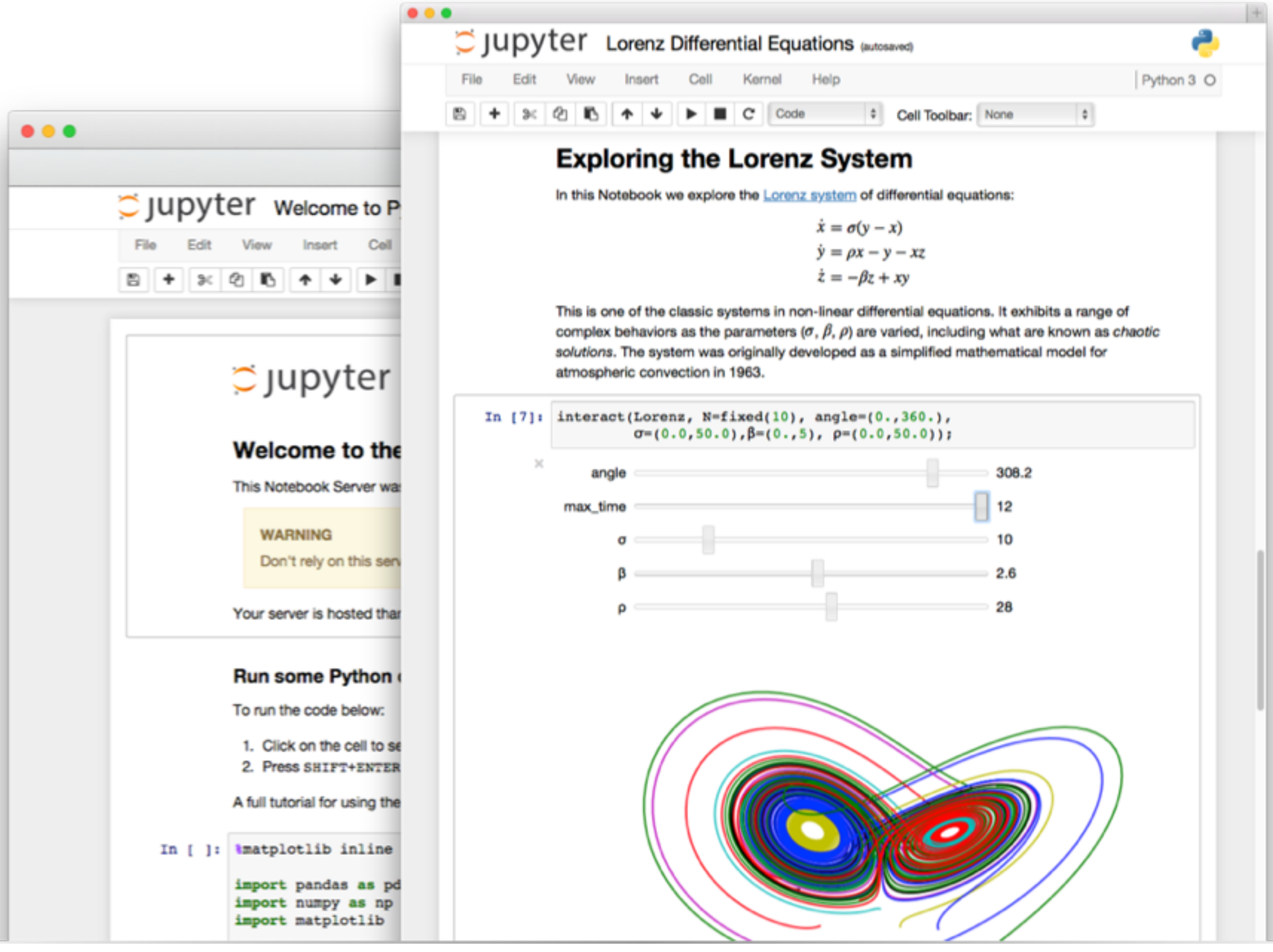
(3) Evaluate results of program (printed to text terminal or other output)

Repeat

IDLE Editor and Shell

https://docs.python.org/2/library/idle.html

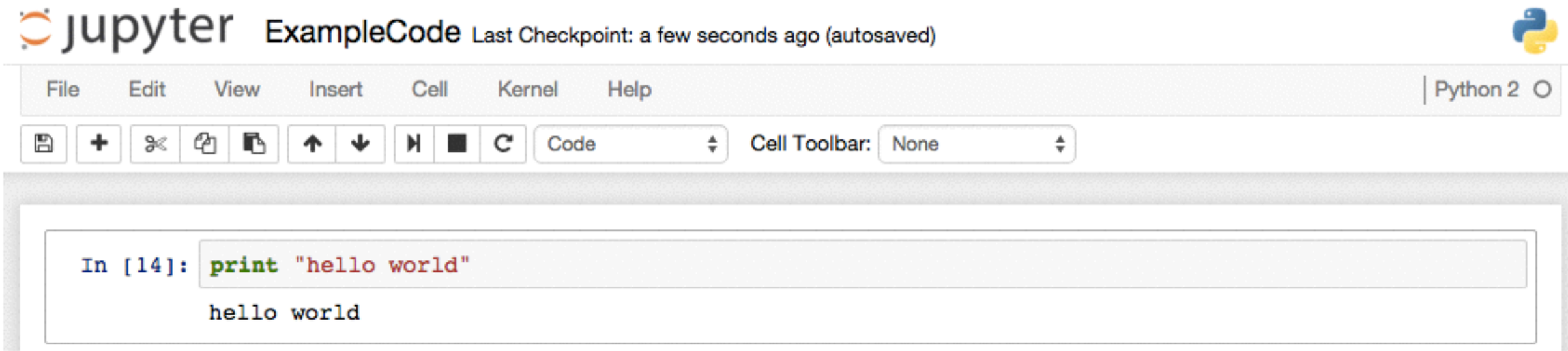
Development Environment - Jupyter Notebook



Source: https://jupyter.org/

Development Environment - Jupyter Notebook

Details and Overview of a Notebook



Details of the Notebook:

- cells contain code.
- execute cells one by one, but not necessarily in a linear order
- cells can be split / merged /moved /deleted
- all cells share a global variable context
- code executes **not** in the browser, but in a kernel (a dedicated python session)
- output results in the notebook are cached, but kernel states is not

Development Environment - Jupyter Notebook

Notebook Features: Markdown Cell-type

Markdown is a simple text mark-up language common to content management systems.

https://help.github.com/articles/markdown-basics/

http://daringfireball.net/projects/markdown/

Add rich document formatting with Markdown Cells.

```
# Markdown ( The largest heading )
## The second largest heading
...
###### The 6th largest heading

A paragraph starts simply with a new line.

Read [Markdown Basics](https://help.github.com/articles/markdown-basics/
) on github.

Check out this neat program I wrote:

```
x = 0
x = 2 + 2
what is x
```
```

Markdown (The largest heading)

The second largest heading

...

The 6th largest heading

A paragraph starts simply with a new line.

Read Markdown Basics on github.

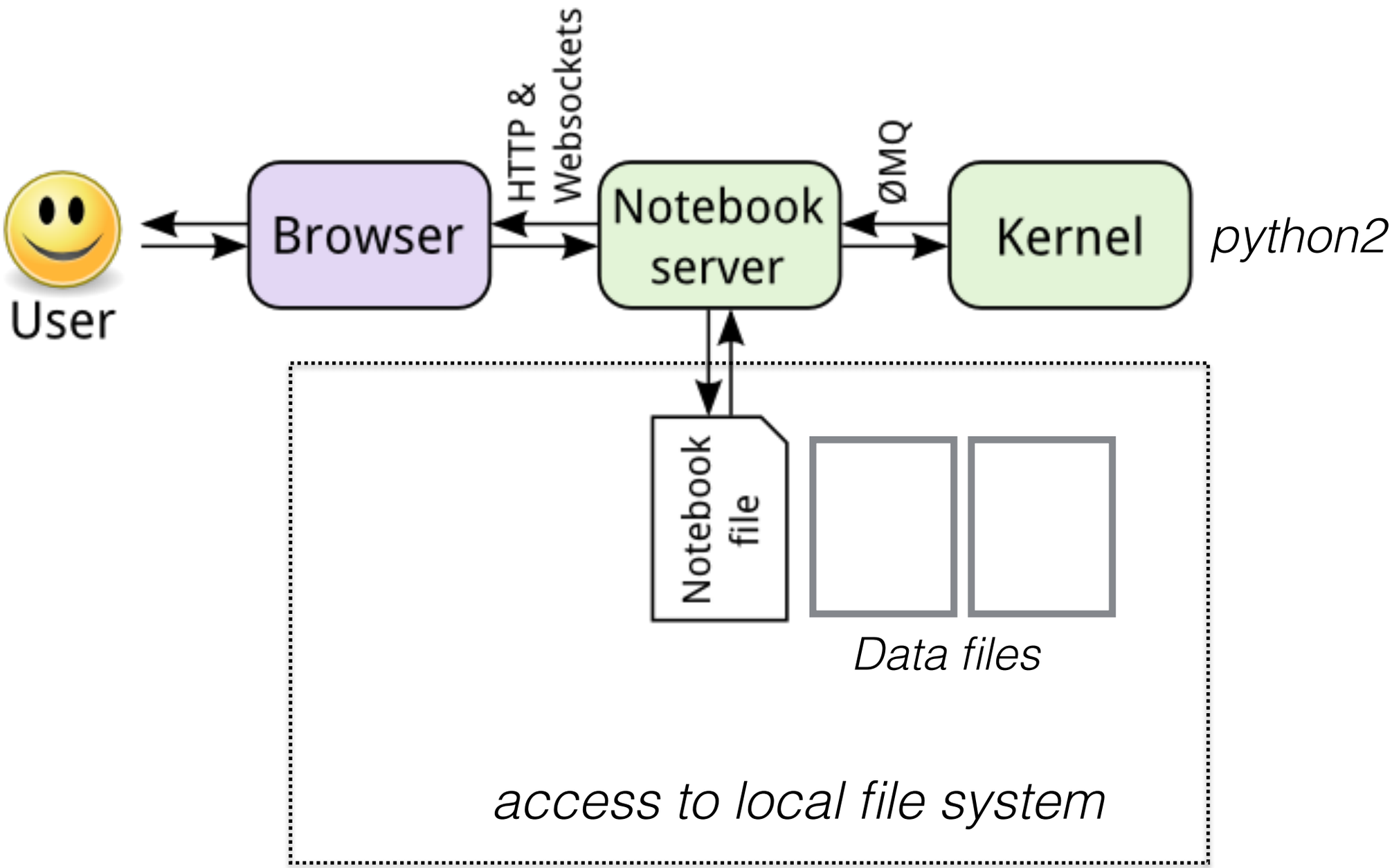
Check out this neat program I wrote:

```
x = 0
x = 2 + 2
what is x
```

Development Environment - Jupyter Notebook

Jupyter Notebook as a System

Notebook runs as a **client-server** architecture: stand-alone on your **local machine.**



Source: http://ipython-minrk.readthedocs.org/en/latest/development/how_ipython_works.html

Development Environment - Jupyter Notebook

Notebook Features: Multiple kernels Available

- Use different languages in Jupyter (**JU**lia, **PYT**hon, **R**)
- How many different kernels exist on Jupyter?

IPython/Jupyter kernels:

The Kernel Zero, is of course IPython, which you can get though ipykernel, and still comes (for now) as a dependency of jupyter. The IPython kernel can be thought as a reference implementation, here are other available kernels:

Name	Link	Jupyter/IPython Version	Language(s) Version
ICSharp	https://github.com/zabirauf/icsharp	Jupyter 4.0	C# 4.0+
IRKernel	http://irkernel.github.io/	IPython 3.0	R 3.2
SageMath	http://www.sagemath.org/	IPython 3.2	Any

Add your project to the table above:

1. IJulia
2. IHaskell
3. IFSharp
4. IRuby
5. IGo
6. IScala
7. IMathics
8. IAldor
9. Calico Project - kernels implemented in Mono, including Java, IronPython, Boo, Logo, BASIC, and many others
10. LuaJIT/Torch
11. Lua Kernel abandoned, forked as IPyLua

https://github.com/ipython/ipython/wiki/IPython-kernels-for-other-languages

Development Environment - Jupyter Notebook

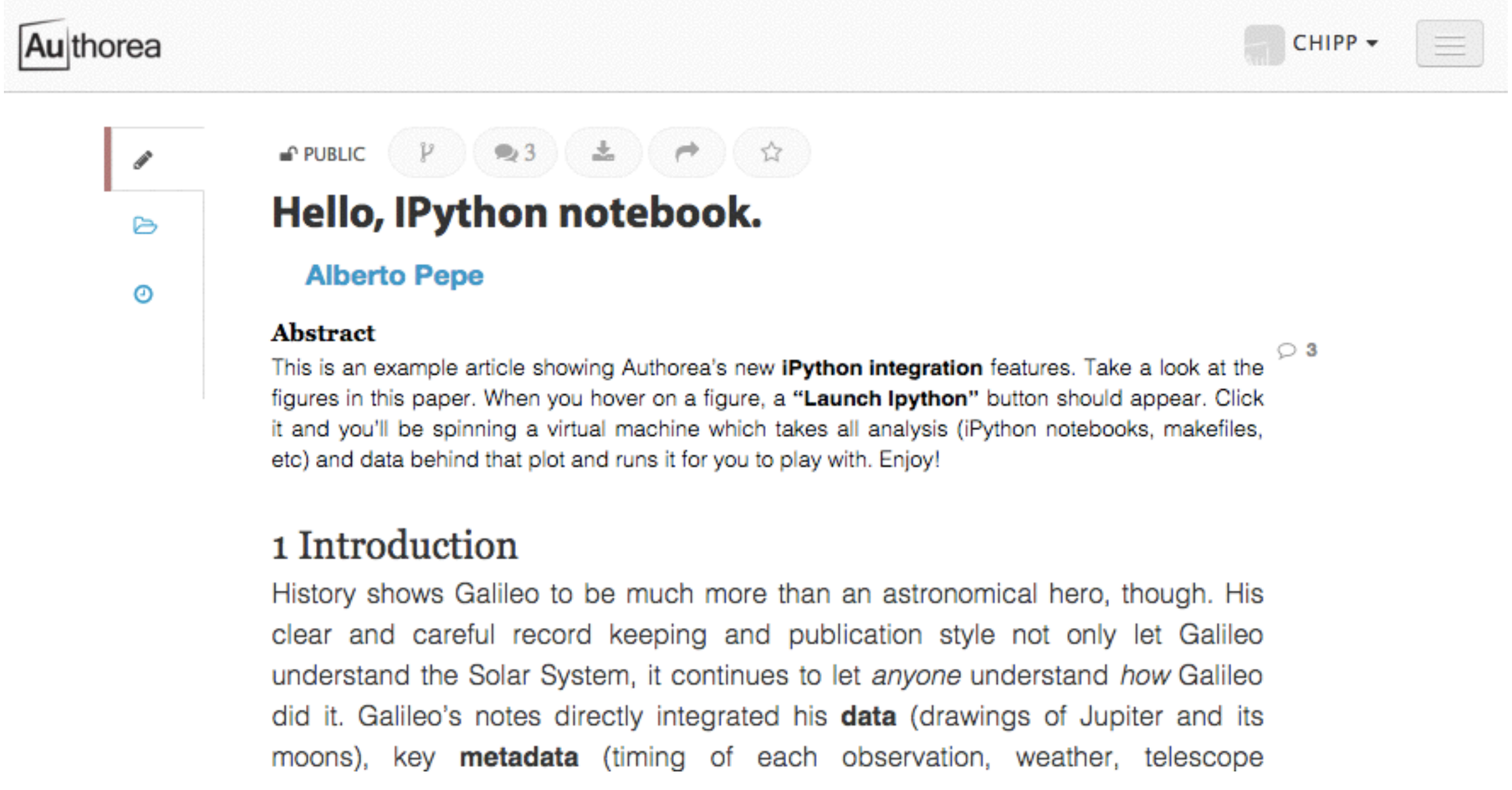
Notebook Features: Notebook as a Service

Notebooks can also run on hosted servers (i.e. kernels not running on local machine)

[Example: https://try.jupyter.org/](https://try.jupyter.org/)

Example of a Service: Embedding a Notebook as part of a Written Paper

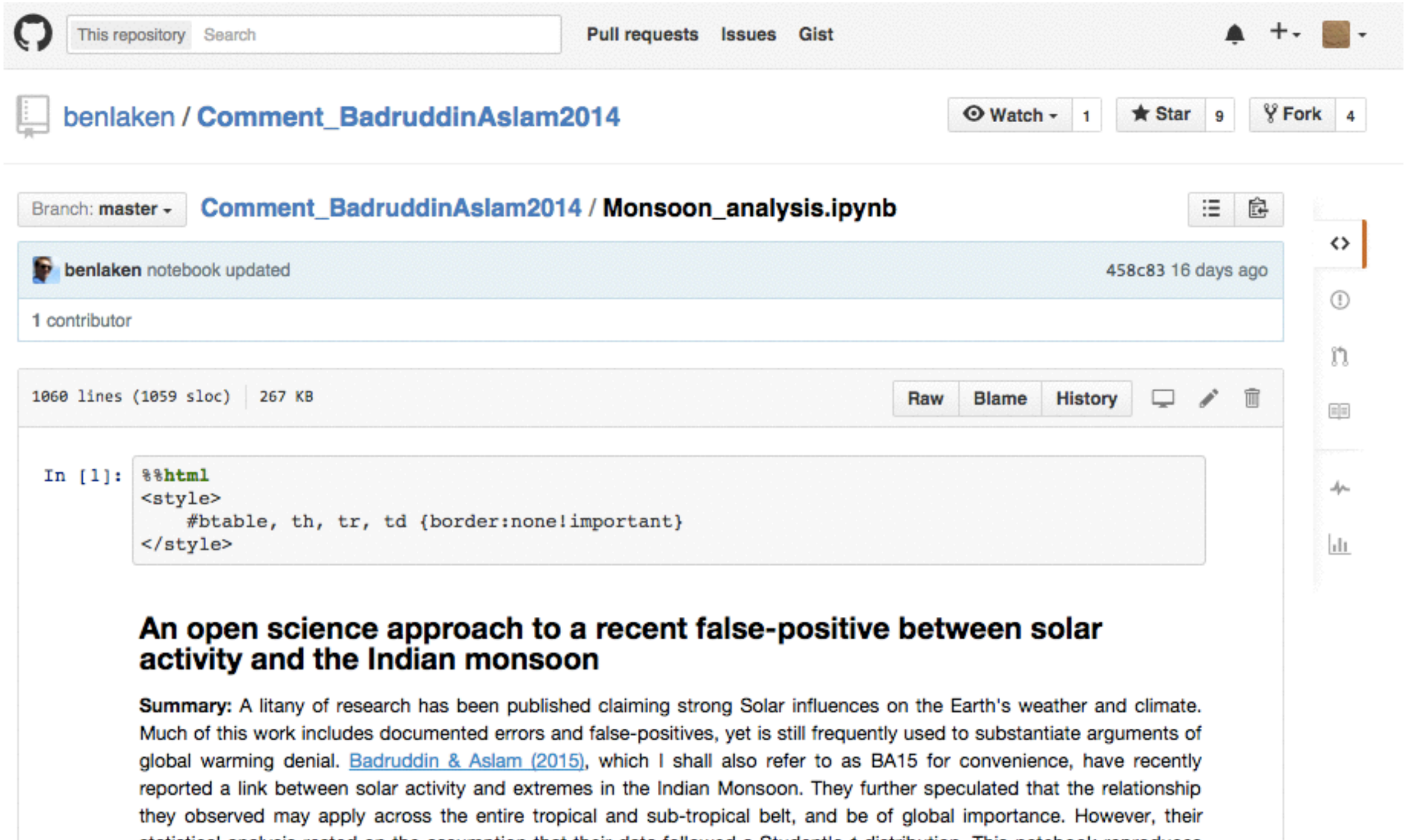
- https://www.authorea.com/users/3/articles/3434



Development Environment - Jupyter Notebook

Notebook Features: Github (code repo) include viewers

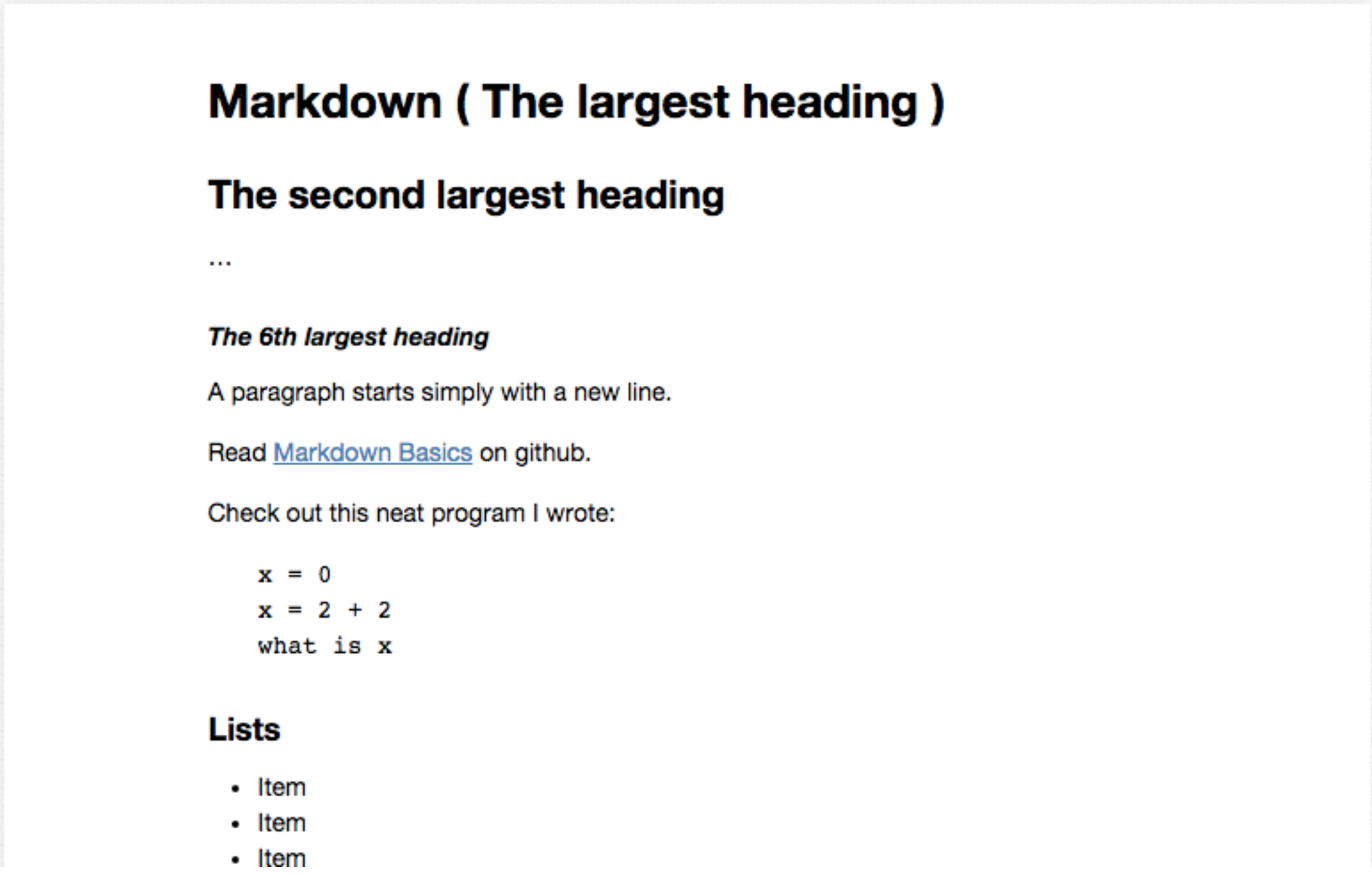
- Github supports viewing notebook in a rendered format:
  - Cameron Davidson-Pilon's Probabilistic Programming and Bayesian Methods for Hackers.
  - Lorena Barba's Aerodynamics-Hydrodynamics with Python.
  - Benjamin Laken's Monsoon Analysis.



Development Environment - Jupyter Notebook

Notebook Features: Exporting a Static Version

- Export as HTML for a static document (Save As -> HTML)



Markdown (The largest heading)

The second largest heading

...

The 6th largest heading

A paragraph starts simply with a new line.

Read Markdown Basics on github.

Check out this neat program I wrote:

```
x = 0
x = 2 + 2
what is x
```

Lists

- Item
- Item
- Item

Development Environment - Jupyter Notebook

Export the Notebook as a Runnable .py script

- Export Notebook as .py script, which can be run from the command-line
- Can use a notebook can be the first step in prototyping a data science workflow.

jupyter ExampleCode (autosaved)

File Edit View Insert Cell Kernel Help Python 2

Code Cell Toolbar: None

```
In [ ]: print "hello world"

In [ ]: #--
        # x5a.py
        # @author: sklar
        # @created: 9-oct-2015
        #
        # this script demonstrate use of the json module to read and parse a
        # data file stored in json format.
        #
        # more documentation on json file reading and writing:
        #  https://docs.python.org/2/library/json.html?highlight=json#module-json
        #
        # sample data downloade from:
        #  http://www.omdbapi.com
        #--

        import sys
        import json

In [ ]: try:
        # data file in json format
            f = open( 'data/movies.json', 'r' )
            mdb = json.load( f )
        # handle exceptions:
        except IOError as iox:
            print 'there was an I/O error trying to open the data file: ' + str( io
```

ExampleCode.py

```
1
2   # coding: utf-8
3
4   # In[ ]:
5
6   print "hello world"
7
8
9   # In[ ]:
10
11  #--
12  # x5a.py
13  # @author: sklar
14  # @created: 9-oct-2015
15  #
16  # this script demonstrate use of the json module to read and parse a
17  # data file stored in json format.
18  #
19  # more documentation on json file reading and writing:
20  #  https://docs.python.org/2/library/json.html?highlight=json#module-json
21  #
22  # sample data downloade from:
23  #  http://www.omdbapi.com
24  #--
25
26  import sys
27  import json
28
29
30  # In[ ]:
```

Development Environment - Jupyter Notebook

Advantages of using a Notebook Format

- highly *interactive*, and useful for *exploratory data analysis*
- no need to re-run the script every time to generate more results
- *building a document*: including markdown, and other components for a presentation
- *multiple languages* through different kernels
- easy to share results, and results are more *transparent* (code and results are in the same document).
- embedded *interactive widgets* (will see more in a bit)

Development Environment - Jupyter Notebook

Disadvantages of using a Notebook Format

- *non-linearity* of cells, state of the variables can get confusing
- *kernel can crash* and require restarting (losing data results that was stored in variables)
- no *text based input* (i.e. `raw_input`), need to use widgets
- not a *traditional IDE* to build multiple modules and a linearly running program with command line arguments
- promote *lazy practice* when building data workflows
  - need to move the notebook into a script at some point

Development Environment - Jupyter Notebook

Many Examples of Notebooks for sharing results

https://github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks

1. Entire books or other large collections of notebooks on a topic
    - Introductory Tutorials
    - Programming and Computer Science
    - Statistics, Machine Learning and Data Science
    - Mathematics, Physics, Chemistry, Biology
    - Earth Science and Geo-Spatial data
    - Linguistics and Text Mining
    - Signal Processing
    - Engineering Education
2. Scientific computing and data analysis with the SciPy Stack
    - General topics in scientific computing
    - Social data
    - Psychology and Neuroscience
    - Machine Learning, Statistics and Probability
    - Physics, Chemistry and Biology
    - Economics
    - Earth science and geo-spatial data
    - Data visualization and plotting
    - Mathematics
    - Signal and Sound Processing
    - Natural Language Processing
    - Pandas for data analysis
3. General Python Programming
4. Notebooks in languages other than Python
    - Julia
    - Haskell

How do you get new Python Libraries?

Use Python Package Management

(1) **Anaconda -** open source (but commercial), *standardized* out-of-the-box environment

https://www.continuum.io/anaconda-overview

```
% conda install notebook
```

(2) **pip** - Install package from the Python Package Index (PyPI)

Recommended for those managing their own systems

https://pypi.python.org/pypi

https://packaging.python.org/current/

Online Resources

Python: https://www.python.org

Python documentation (we are using Python 2.7 in this course):

- Python version v2.x: https://docs.python.org/2/
- Python version v3.x: https://docs.python.org/3/

on the differences between Python v2.x and v3.x:

http://stromberg.dnsalias.org/~dstromberg/Intro-to-Python/Python%202%20and%203.pdf

Tutorials

- Python version 2.x: https://docs.python.org/2/tutorial/
- Python version 3.x: https://docs.python.org/3/tutorial/
- Python in 10 minutes, for programmers: http://www.stavros.io/tutorials/python/

Notebooks

- Jupyter Notebook: jupyter.org
- Docs: http://jupyter.readthedocs.io/en/latest/index.html
- Try it in your browser: https://try.jupyter.org/

Class Exercise

Helps us understand the introductory programming level of the class

On a blank sheet of paper (raise your hand if you need one):

- Your **Name** and **Email**
- Write python code (or *psudo-code*) to convert a list of numbers from integers to english words.
  - Assume integers are in a range from 1 to 5
  - Store the converted words in a new list **without** modifying the input list
  - Start with this list as your input:

```
nums = [4, 2, 3, 2, 1, 5]
```

  - Your resulting list will be:

```
["four", "two", "three", "two", "one", "five"]
```

are you done early? add python code to reverse the order of your new word list

Class Exercise - Lecture 1- 7CCSMCMP

- **Name**
- **Email**

- Write python code (or *psudo-code*) to convert a list of numbers from integers to english words.
  - Assume integers are in a range from 1 to 5
  - Store the converted words in a new list **without** modifying the input list
  - Start with this list as your input:

```
nums = [4, 2, 3, 2, 1, 5]
```

  - Your resulting list will be:

```
[“four”, “two”, “three”, “two”, “one”, “five”]
```

are you done early? add python code to reverse the order of your new word list