Algorithmic Analysis

Joe

Introduction

Session Objective

Please clone:
https://github.com/gSchool/DSI_Lectures
we'll be using python-intro/jGartner
(I stole most of this from Matt Drury)

1. Establish Style standards consistent with Python best practices

2. Familiarize yourself with the tools I will use, and make you aware of alternatives

3. Understand how to pair programing exercises will work

Introductions

Lead Instructor - Joseph Gartner

- Background:
  - Awarded Ph.D. in physics 2011 for my work at the Large Hadron Collider
  - Worked as a software engineer developing cloud deployment tools for an HR SAS company
  - Worked as a data scientist for Sotera Defense Systems. Worked on DARPAs XDATA and QCR programs
- Data science strengths - NLP, spark, mathematical methods
- Outside interests - rugby, jiu jitsu, music
- Thing I believe that others think is crazy - google is the first AI and is slowly taking control of everything

<>

Class Philosophy

THE OBSTACLE IS THE WAY

Embrace the challenge of what you have taken on. Being uncomfortable causes you to grow.

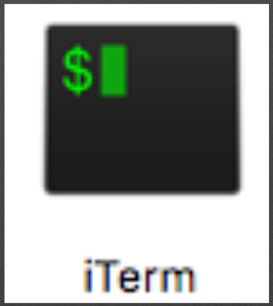
EGO IS THE ENEMY

There is no dishonor in making an error. Ask questions. Try to answer mine. Be kind to your classmates, they might be your coworkers.

Tools of the Trade

Terminal (iTerm)

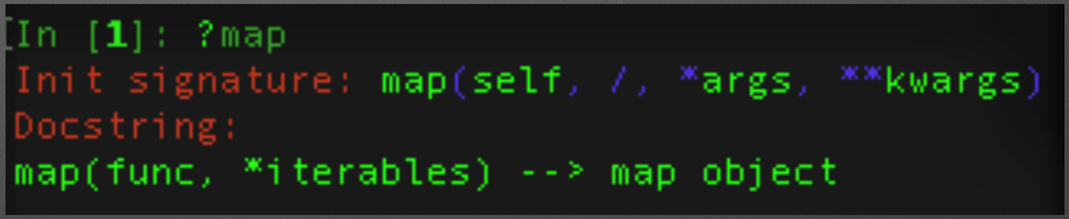
- We’ll be using the terminal as our primary way of interacting with the file system
    - Using git repos
    - Installing python packages
    - Running python scripts
    - Launching iPython
- If you’re not familiar with bash shell or linux, we’ll be going over fundamentals on Friday
- iTerm is a version of the terminal with some upgraded functionality



iPython

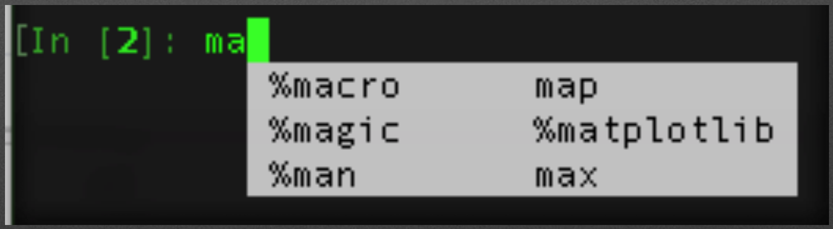
- iPython is an interpreter similar to an interactive python session. It has additional features that are very helpful:

View Docstring



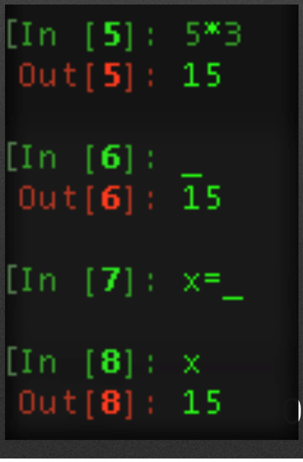
```
[In [1]: ?map
Init signature: map(self, /, *args, **kwargs)
Docstring:
map(func, *iterables) --> map object
```

Tab Completion



```
[In [2]: ma
          %macro      map
          %magic      %matplotlib
          %man        max
```

View Docstring



```
[In [5]: 5*3
Out[5]: 15

[In [6]: _
Out[6]: 15

[In [7]: x=_

[In [8]: x
Out[8]: 15
```

More at
https://ipython.org

Jupyter Notebooks

- Jupyter is a way of developing code that adheres to good scientific development.
- Let’s launch a notebook and walk through some features!

When to use Jupyter

- Jupyter is a great way of quickly developing models that do not fit into a larger pipe-line

- Jupyter is poor at complex interactions of large codebases instead, we need to use an IDE

Python IDE

- The IDE is an integrated development environment.
- The galvanize ‘official’ IDE is atom…I choose not to use atom
    - The reason for this is that atom does not come with a debugger by default, which in my opinion, is the single most important feature of any IDE.
    - My preference is pycharm: https://www.jetbrains.com/pycharm/
- Let’s look at a simple program to collect data from Twitter in an IDE

Best Practices

Python 3

- Unless noted otherwise, I all code presented will be Python 3
- There are subtle but important differences between python 3 and python 2
- If you don’t know what you’ve been working with up to this point, it has probably been python 2
- The majority of legacy code you will encounter is python 2
- So if python 2 is ‘default’, and most legacy code is python 2, and the differences are non-trivial, why python 3?
    - Python 2 is no longer supported. Developers are lazy and won’t switch till they have to, but the tipping point is quickly approaching.

2->3 Nuances

dict.iteritems() -> generator	dict.items() -> generator
filter, sorted, map -> creates lists	filter, sorted, map ->creates generators
print “blah”	print(“blah”)
range, xrange -> list, generator	range, xrange -> generator, DNE

Style

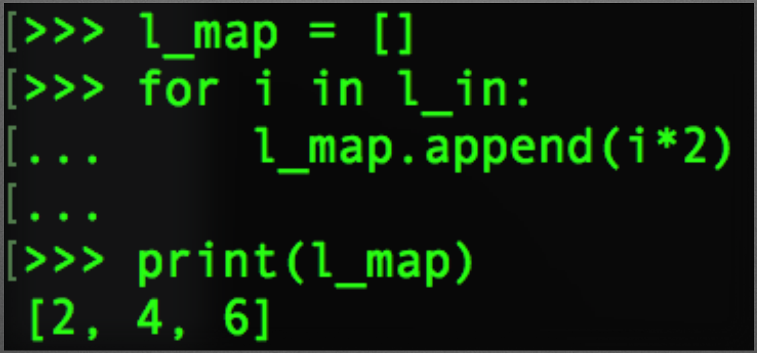
- Code is read more than it is written; in this way, style is substance

- Python is unique in that there are ‘pythonic’ ways of writing code, and deviating from these style selections are considered bad form

- If you come from a C/C++/Java background this can be difficult to make the adjustment, I suggest you put effort to making switches sooner than later!

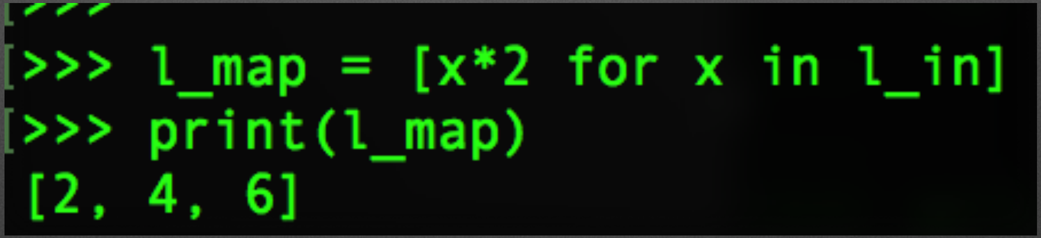
Example - Mapping a List

Bad



```
[>>> l_map = []
[>>> for i in l_in:
[...     l_map.append(i*2)
[...
[>>> print(l_map)
 [2, 4, 6]
```

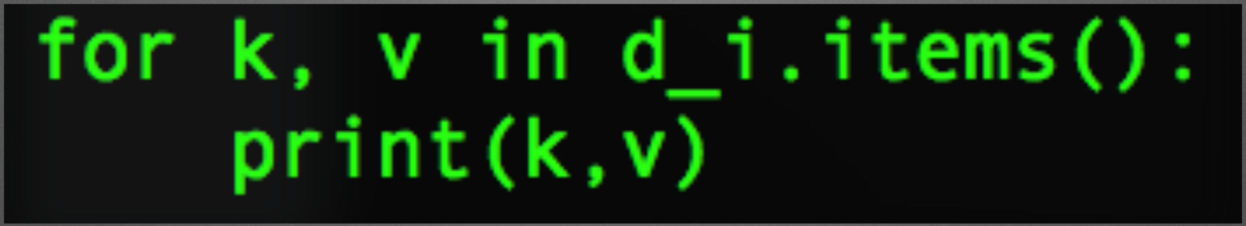
Good



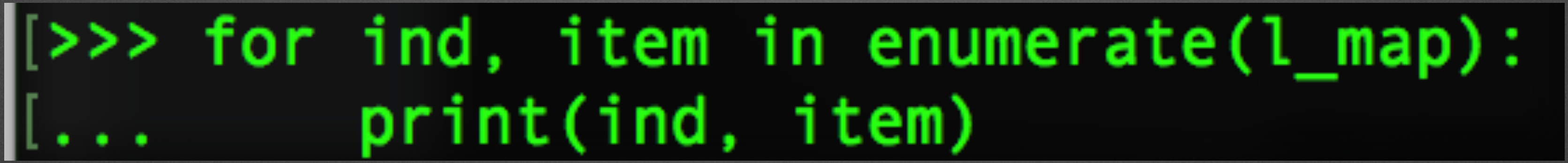
```
[>>> l_map = [x*2 for x in l_in]
[>>> print(l_map)
[2, 4, 6]
```

Complete style guide:
http://legacy.python.org/dev/peps/pep-0008/

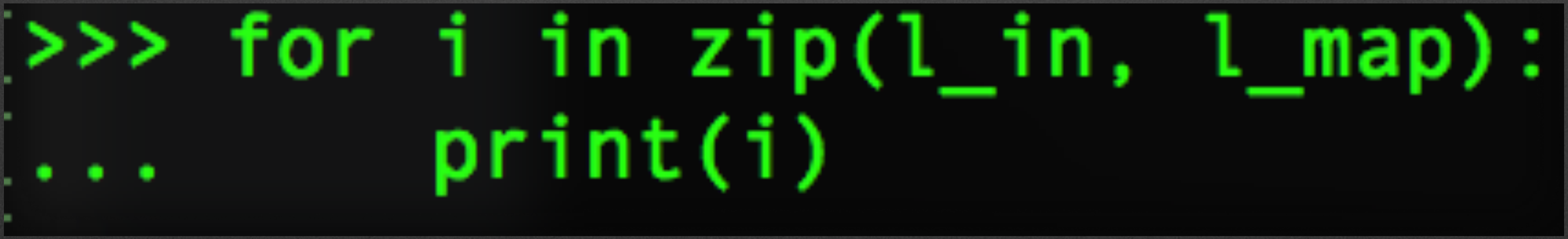
Looping



```
for k, v in d_i.items():
    print(k,v)
```

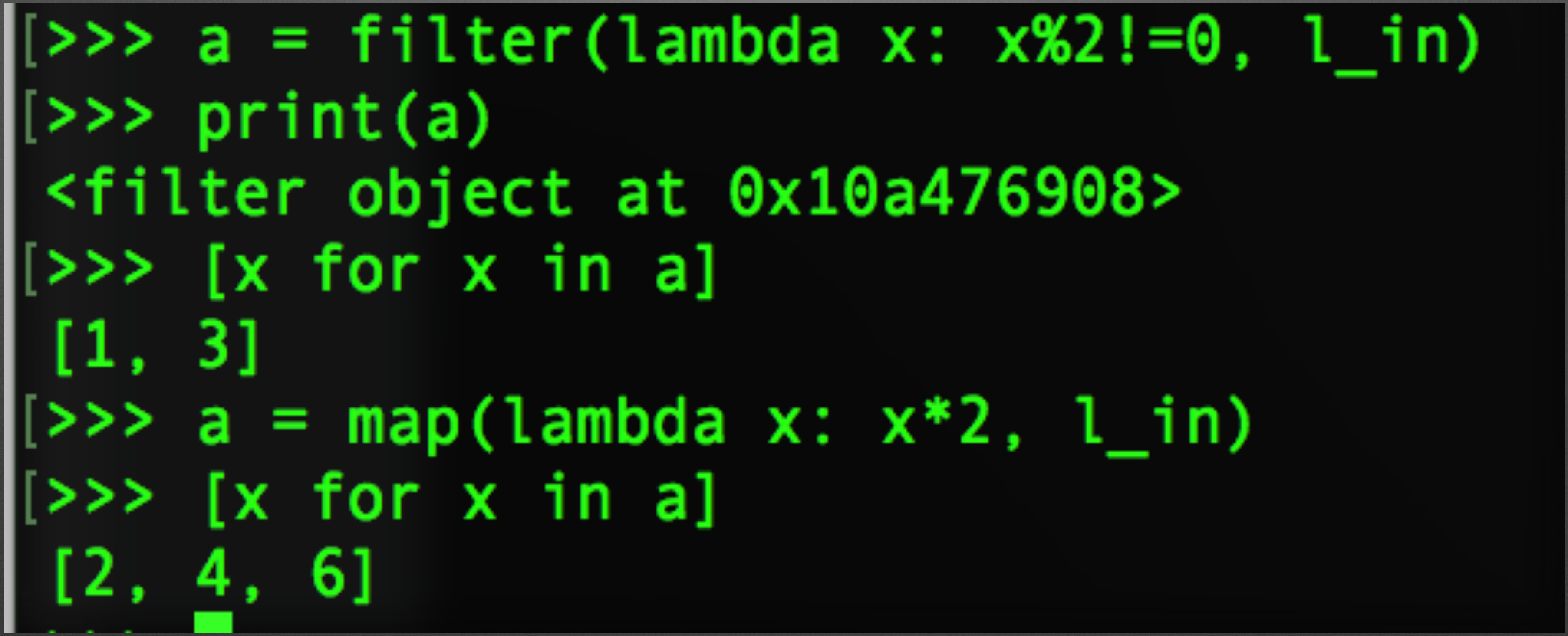


```
[>>> for ind, item in enumerate(l_map):
[...     print(ind, item)
```



```
>>> for i in zip(l_in, l_map):
...     print(i)
```

Lambdas



```
[>>> a = filter(lambda x: x%2!=0, l_in)
[>>> print(a)
<filter object at 0x10a476908>
[>>> [x for x in a]
[1, 3]
[>>> a = map(lambda x: x*2, l_in)
[>>> [x for x in a]
[2, 4, 6]
```

Runtime Efficiency

- The field of data science is a consequence of ‘big data’
  - Big data is a dataset that cannot fit on a single machine

- As such, data scientists must be mindful of the efficiency of their code
  - Judicious use of of generators helps avoid memory inefficiencies

- The other consideration is runtime

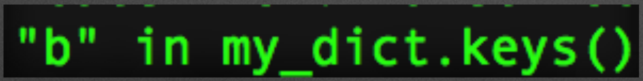
- The terminology to describe the runtime of code is called ‘big O’ notation

Big O Notation

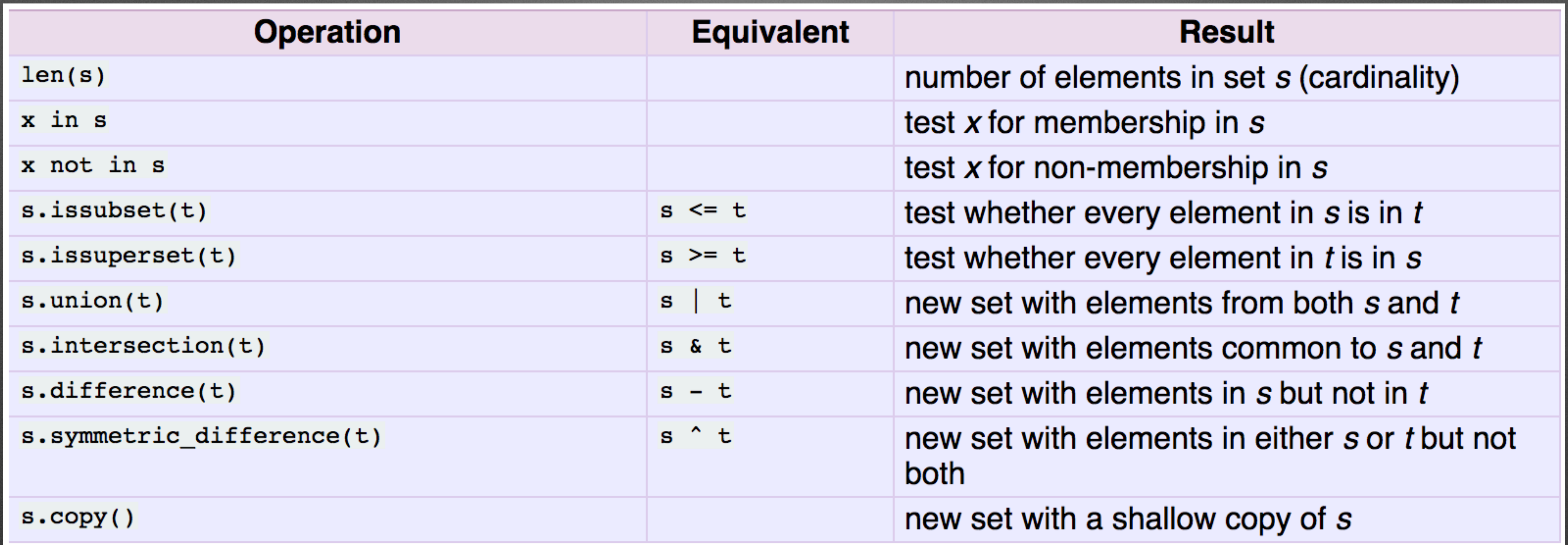
- Big O Notation - Used to describe how the runtime (and to a lesser extent, memory) of function increases as the size of the input array increases.
- This is an order of magnitude approximation, meaning we only worry about the leading term
    - Example: O(n) notebook

Dicts & Sets

- Dicts and sets are implemented as hash table, meaning they have O(1) lookup time

- This error:  `"b" in my_dict.keys()`
can add hours to the execution of code at scale!

Set Operations



Operation	Equivalent	Result
`len(s)`		number of elements in set *s* (cardinality)
`x in s`		test *x* for membership in *s*
`x not in s`		test *x* for non-membership in *s*
`s.issubset(t)`	`s <= t`	test whether every element in *s* is in *t*
`s.issuperset(t)`	`s >= t`	test whether every element in *t* is in *s*
`s.union(t)`	`s | t`	new set with elements from both *s* and *t*
`s.intersection(t)`	`s & t`	new set with elements common to *s* and *t*
`s.difference(t)`	`s - t`	new set with elements in *s* but not in *t*
`s.symmetric_difference(t)`	`s ^ t`	new set with elements in either *s* or *t* but not both
`s.copy()`		new set with a shallow copy of *s*

Set operations, as well as many other python base element operation are implemented in C, meaning they are *very* efficient

Pair Programing

Pair Programming

- Pair programming is a paradigm where two people work on the same computer, with a driver-navigator paradigm
  - Switch roles every 30 minutes

- Leads to:
  - Higher Quality output
  - Learn more
  - Forces you to explain your thought process

Best Practices

- Get to know your partner ("what did you think of lecture")
- Don't talk over each other, don't bogart the conversation
- Give your partner a chance to write code, don't "side seat" drive
- Disagree civilly
- *You are not a cop in an 80s movie "I work alone" is not an option*

Session Objective

Please clone:
https://github.com/gSchool/DSI_Lectures
we’ll be using python-intro/jGartner
(I stole most of this from Matt Drury)

1. Establish Style standards consistent with Python best practices

2. Familiarize yourself with the tools I will use, and make you aware of alternatives

3. Understand how to pair programing exercises will work