Intro to Python Development

Taryn Heilman / Jon Courtney
DSI January 21, 2018

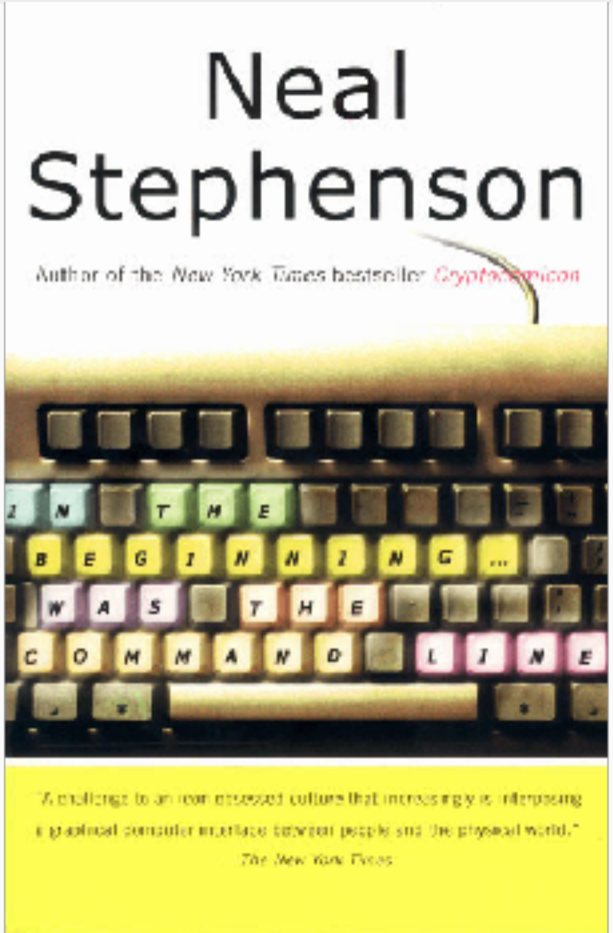
galvanize

Development Tools

galvanize

Unix Command Line / Bash

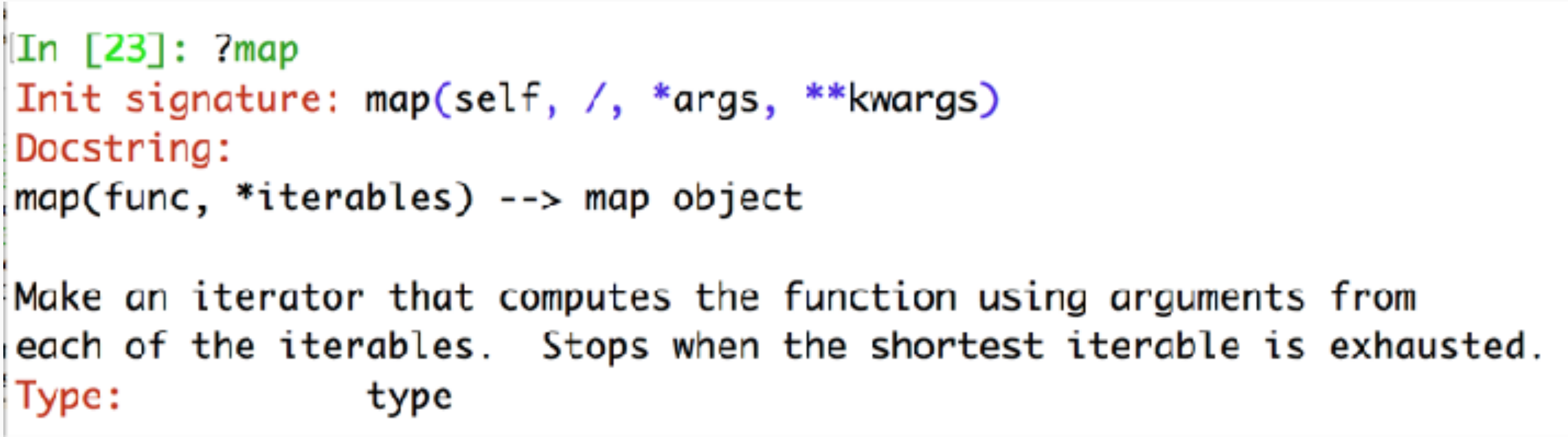
- Accessed via OSX/Ubuntu Terminal app
  - iTerm has some nice improvements
- Useful for:
  - Git commands
  - Installing python packages
  - Running python scripts
  - Launching iPython
  - Accessing AWS
  - Getting stuck in vi


Neal Stephenson
Author of the New York Times bestseller Cryptonomicon
IN THE BEGINNING... WAS THE COMMAND LINE
"A challenge to an icon-obsessed culture that increasingly is interposing a graphical computer interface between people and the physical world."
The New York Times

iPython

- Interactive Read-Evaluate-Print-Loop (REPL)
- Feature-rich
- Useful for:
  - Viewing docstrings / help
  - Tab completion
  - Accessing previous results

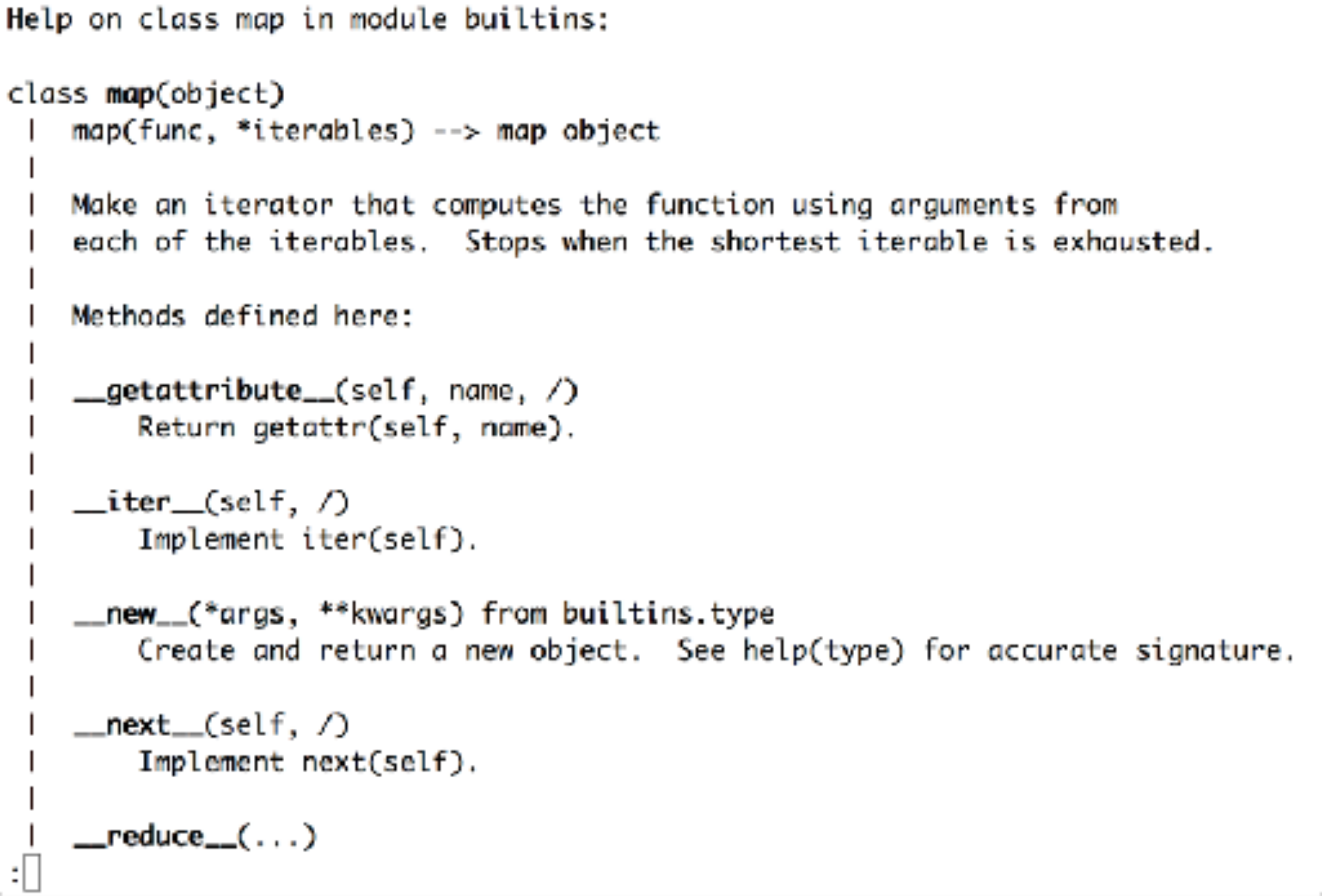
Viewing docstrings



```
In [23]: ?map
Init signature: map(self, /, *args, **kwargs)
Docstring:
map(func, *iterables) --> map object

Make an iterator that computes the function using arguments from
each of the iterables.  Stops when the shortest iterable is exhausted.
Type:           type
```

help() command



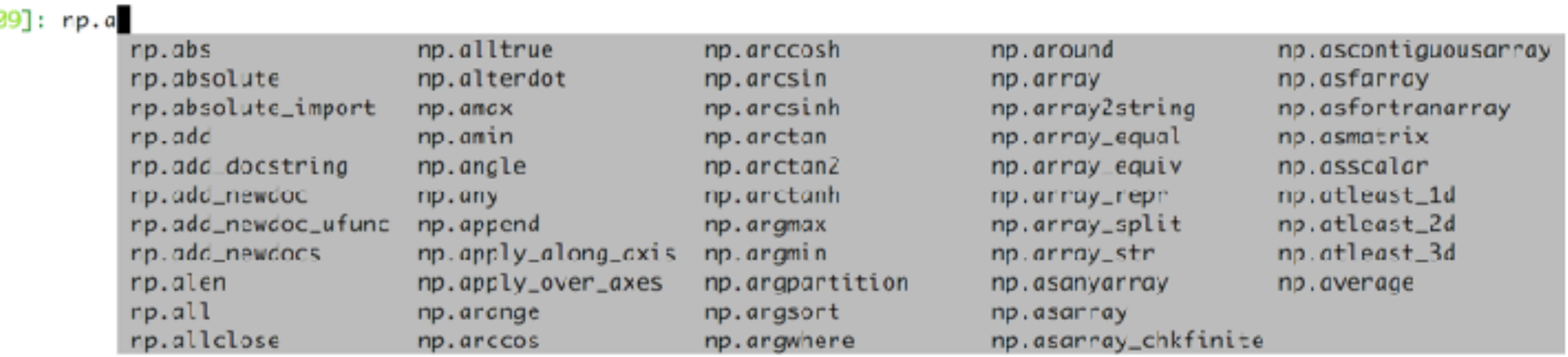
```
Help on class map in module builtins:

class map(object)
 |  map(func, *iterables) --> map object
 |
 |  Make an iterator that computes the function using arguments from
 |  each of the iterables.  Stops when the shortest iterable is exhausted.
 |
 |  Methods defined here:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  __next__(self, /)
 |      Implement next(self).
 |
 |  __reduce__(...)
:
```

Tab completion

Search through available methods/attributes

No need to memorize every method available… just know where to look!



```
39]: np.a
      np.abs                 np.alltrue             np.arccosh             np.around                 np.ascontiguousarray
      np.absolute            np.alterdot            np.arcsin              np.array                  np.asfarray
      np.absolute_import     np.amax                np.arcsinh             np.array2string           np.asfortranarray
      np.add                 np.amin                np.arctan              np.array_equal            np.asmatrix
      np.add_docstring       np.angle               np.arctan2             np.array_equiv            np.asscalar
      np.add_newdoc          np.any                 np.arctanh             np.array_repr             np.atleast_1d
      np.add_newdoc_ufunc    np.append              np.argmax              np.array_split            np.atleast_2d
      np.add_newdocs         np.apply_along_axis    np.argmin              np.array_str              np.atleast_3d
      np.alen                np.apply_over_axes     np.argpartition        np.asanyarray             np.average
      np.all                 np.arange              np.argsort             np.asarray
      np.allclose            np.arccos              np.argwhere            np.asarray_chkfinite
```

Access previous results

- The ‘_’ variable contains the result of the last executed command
- Use up arrow to access previous inputs
- `%hist` prints command history

iPython module auto-reload

`~/.ipython/profile_default/startup/autoreload_startup.ipy`

```
%load_ext autoreload
%autoreload 2

# Exclude autoimports
%aimport -np
%aimport -pd
%aimport -sp
%aimport -sklearn
%aimport -skimage
%aimport -mpl
%aimport -plt
%aimport -logging
```

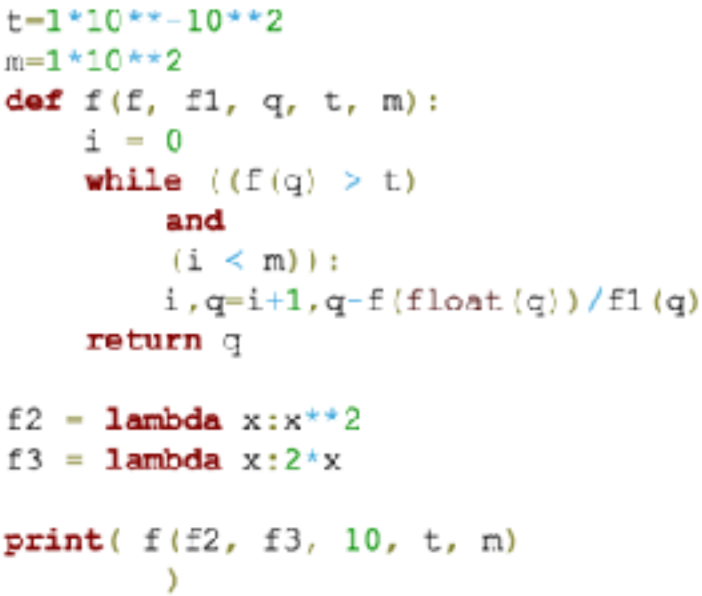
Jupyter (formerly iPython) Notebooks

- Interactive notebook for python based on iPython
    - % jupyter-notebook *notebook_name.ipynb*
- Great for:
    - Exploratory Data Analysis (EDA)
    - Demonstrations
    - Visualization with cloud computing
- **Terrible** for:
    - Writing robustly engineered code
    - Keeping track of execution order / program state

Writing Clean Code

galvanize

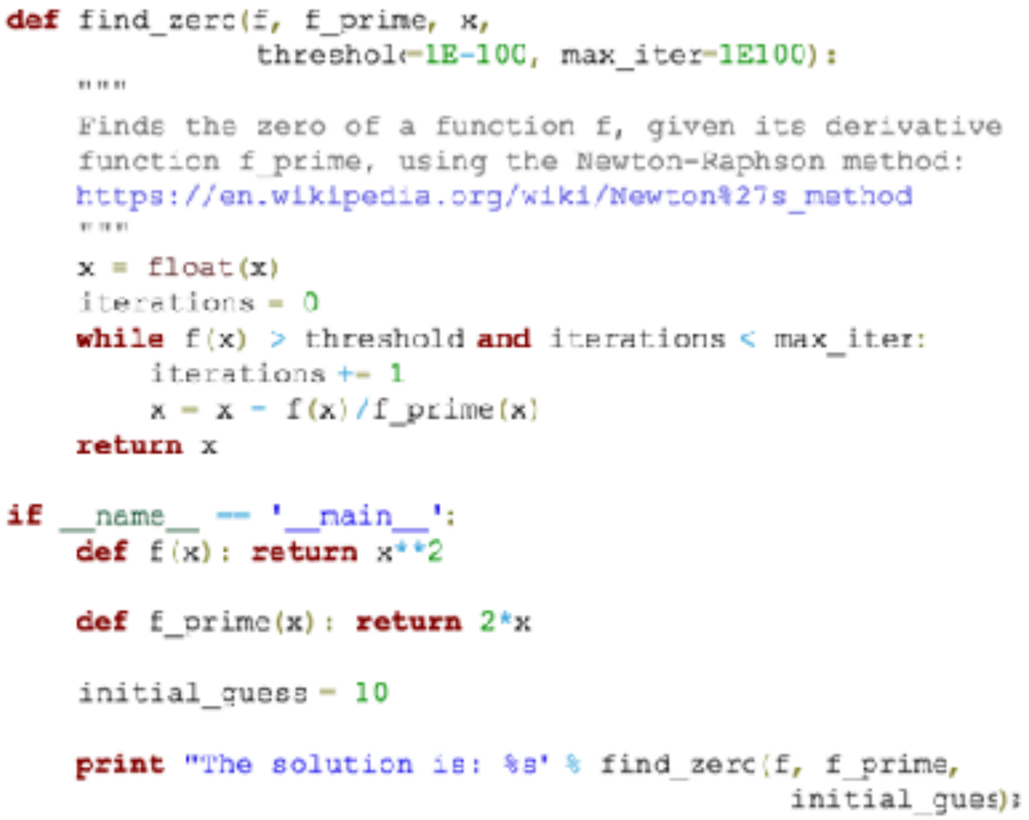
Style and Structure



```
t=1*10**-10**2
m=1*10**2
def f(f, f1, q, t, m):
    i = 0
    while ((f(q) > t)
        and
        (i < m)):
        i,q=i+1,q-f(float(q))/f1(q)
    return q

f2 = lambda x:x**2
f3 = lambda x:2*x

print( f(f2, f3, 10, t, m)
        )
```



```
def find_zero(f, f_prime, x,
              threshold=1E-100, max_iter=1E100):
    """
    Finds the zero of a function f, given its derivative
    function f_prime, using the Newton-Raphson method:
    https://en.wikipedia.org/wiki/Newton%27s_method
    """
    x = float(x)
    iterations = 0
    while f(x) > threshold and iterations < max_iter:
        iterations += 1
        x = x - f(x)/f_prime(x)
    return x

if __name__ == '__main__':
    def f(x): return x**2

    def f_prime(x): return 2*x

    initial_guess = 10

    print "The solution is: %s" % find_zero(f, f_prime,
                                            initial_gues)
```

Style and Structure

- Code is read more than it is written; style is substance
- Structure your code into functions, modules, classes
- Follow the DRY principle:
    - DRY - “Don’t Repeat Yourself”
    - In contrast to WET = “We Enjoy Typing”
- Use pep8 style guide
    - https://www.python.org/dev/peps/pep-0008

Writing Efficient Code

galvanize

Writing Efficient Code

- Code that analyzes a lot of data can run out of memory or take forever to complete

- Optimizing your code can be the difference between code that takes a few minutes to run and code that will effectively never finish running

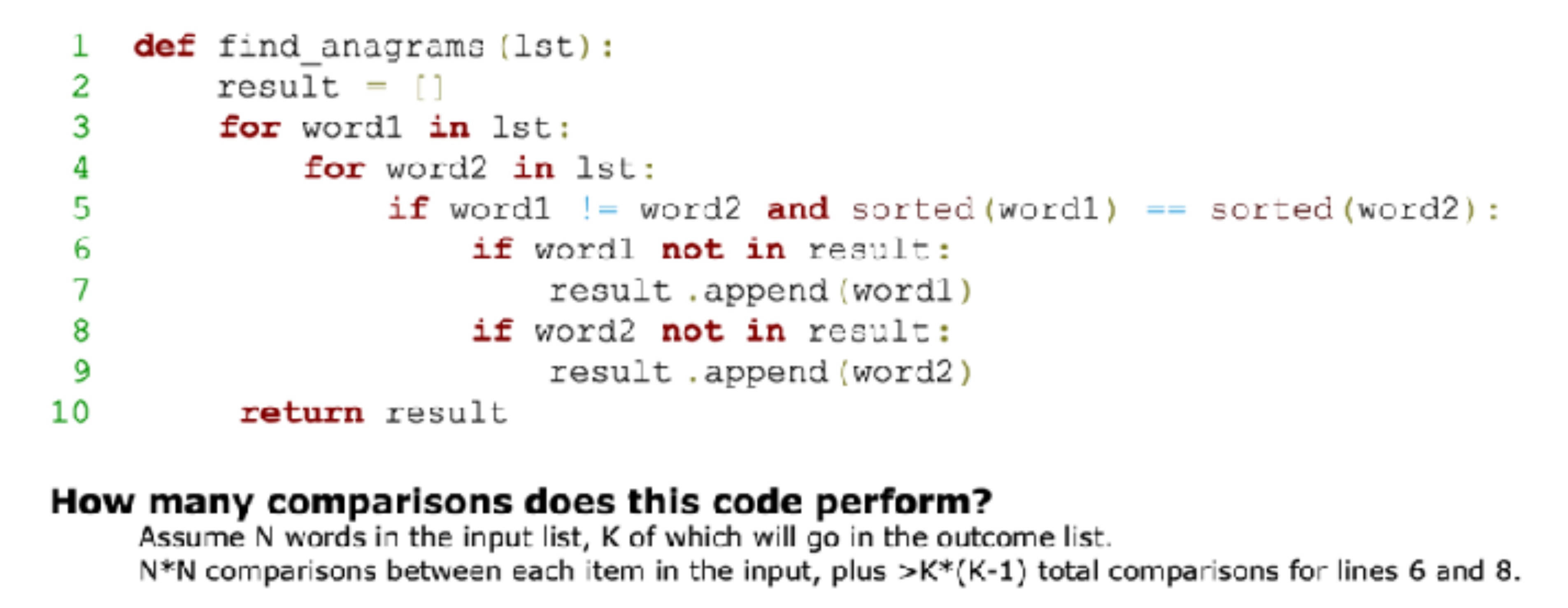
- *Runtime complexity* (aka “big-O” notation) is a very popular interview topic

    - $O(n)$, $O(n^2)$, $O(n^3)$, $O(n!)$…

Saving memory with generators

- Too large a memory footprint will slow your program down, due to *swapping*
- Generators are automatic in python 3 for many things
  - e.g., range returns a generator in python 3, returned a list in python2
  - (Many other examples…zip, .items, etc.)
  - Lists store the entire thing in memory, generators only give you the next item when needed. (Generators save memory, not runtime.)

Writing efficient code - number of comparisons



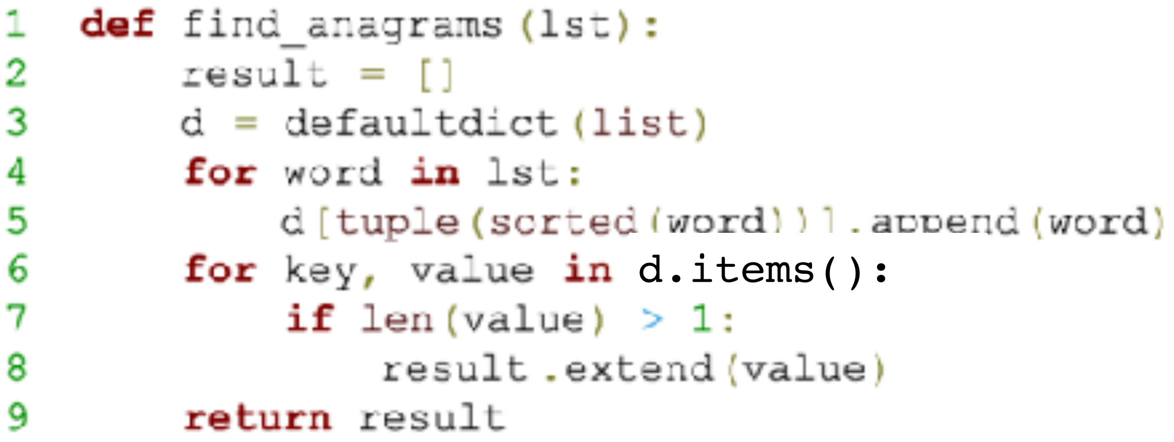
```
1  def find_anagrams(lst):
2      result = []
3      for word1 in lst:
4          for word2 in lst:
5              if word1 != word2 and sorted(word1) == sorted(word2):
6                  if word1 not in result:
7                      result.append(word1)
8                  if word2 not in result:
9                      result.append(word2)
10      return result
```

How many comparisons does this code perform?

Assume N words in the input list, K of which will go in the outcome list.

N*N comparisons between each item in the input, plus >K*(K-1) total comparisons for lines 6 and 8.

Writing efficient code - number of comparisons



```python
def find_anagrams(lst):
    result = []
    d = defaultdict(list)
    for word in lst:
        d[tuple(sorted(word))].append(word)
    for key, value in d.items():
        if len(value) > 1:
            result.extend(value)
    return result
```

How many comparisons does this code perform?

Assuming N words in the input list, then there are N lookups in the dictionary to fill it with the input, and a loop through <N items to determine the output.

How?

Hashing!

Dictionaries

- Dictionaries are “associative arrays”: unordered collections of *key:value* pairs

```
i.e. homestate = {“frank”:“oregon”, “adam”:“new york”,
                  “taryn”:“north carolina”}
```

- Python dictionaries are implemented using hash tables for efficiency
    - Keys must be of *immutable* types
- Instead of iterating through a list of tuples with (name, home_state) pair, you can access a key’s value directly :

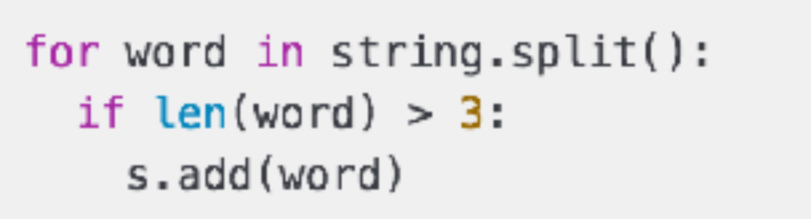
```
i.e. homestate[‘taryn’]
```

- Looping:

```
for key, value in dict.items():
```

Sets

- Unordered collection of unique elements
    - A set is like a dictionary with only keys and no values
- Sets are useful for checking membership and de-duplication. For example:
    - n in my_list takes len(my_list) steps
    - n in my_set takes 1 step
- Example: get all the unique words in a string that are longer than 3 characters:

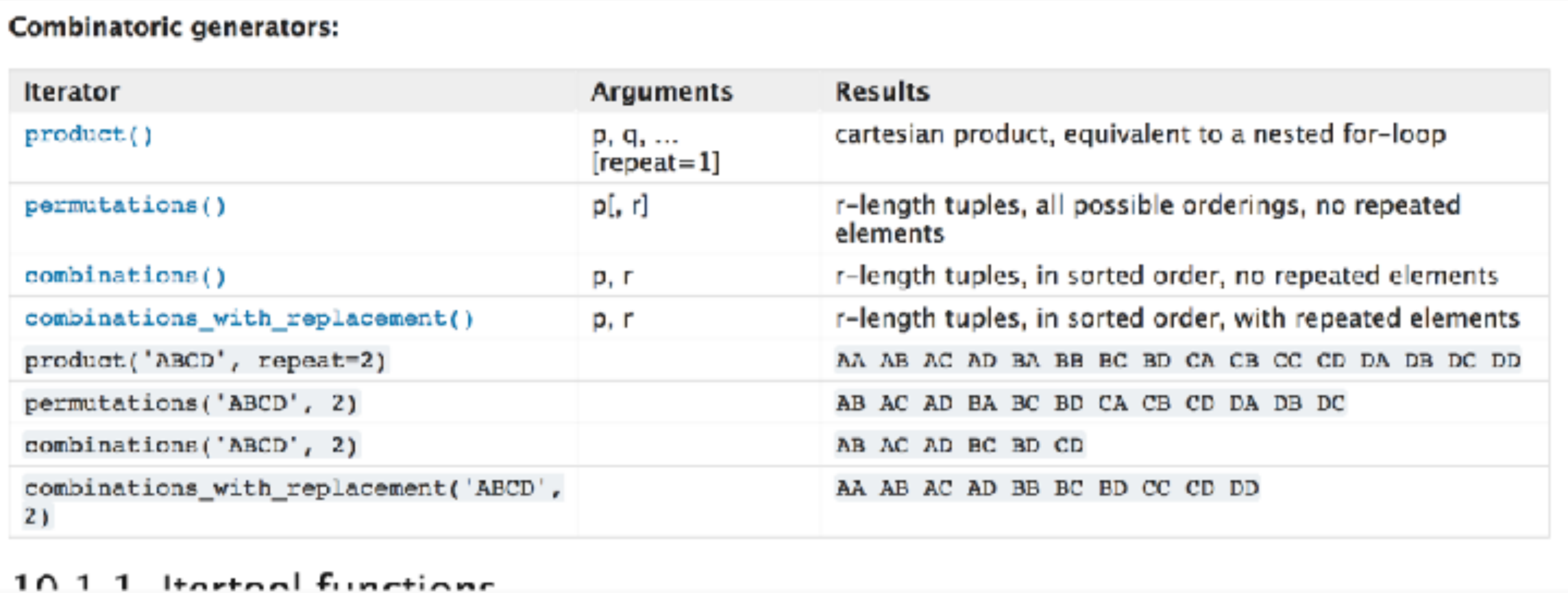


```
for word in string.split():
  if len(word) > 3:
    s.add(word)
```

As a comprehension?

- Sets are also for removing duplicates in a list if you don’t care abut order

Itertools



Combinatoric generators:

Iterator	Arguments	Results
`product()`	p, q, … [repeat=1]	cartesian product, equivalent to a nested for-loop
`permutations()`	p[, r]	r-length tuples, all possible orderings, no repeated elements
`combinations()`	p, r	r-length tuples, in sorted order, no repeated elements
`combinations_with_replacement()`	p, r	r-length tuples, in sorted order, with repeated elements
`product('ABCD', repeat=2)`		`AA AB AC AD BA BB BC BD CA CB CC CD DA DB DC DD`
`permutations('ABCD', 2)`		`AB AC AD BA BC BD CA CB CD DA DB DC`
`combinations('ABCD', 2)`		`AB AC AD BC BD CD`
`combinations_with_replacement('ABCD', 2)`		`AA AB AC AD BB BC BD CC CD DD`

10.1.1. Itertool functions

https://docs.python.org/3/library/itertools.html