DATA 520
Lecture 15

Storing Data

Exam

9. How many times will the code block be run in this example?

```
velocities = [0.03, 3.54, 9.81, 19.62, 29.43]
for speed in velocities: # note: speed IS defined
    velocities = velocities * 1.61/sqrt(2)
```

-will not run (sqrt is not defined), BUT THIS code will run:

```
    velocities = velocities * 2
print(velocities)
[0.0, 9.81, 19.62, 29.43, 0.0, 9.81, 19.62, 29.43, 0.0, 9.81, 19.62, 29.43, 0.0, 9.81, 19.62, 29.43, 0.0,
   9.81, 19.62, 29.43, 0.0, 9.81, 19.62, 29.43, 0.0, 9.81, 19.62, 29.43, 0.0, 9.81, 19.62, 29.43, 0.0, 9.81,
   19.62, 29.43, 0.0, 9.81, 19.62, 29.43, 0.0, 9.81, 19.62, 29.43, 0.0, 9.81, 19.62, 29.43, 0.0, 9.81, 19.62,
   29.43, 0.0, 9.81, 19.62, 29.43, 0.0, 9.81, 19.62, 29.43, 0.0, 9.81, 19.62, 29.43]
```

- it was duplicated 16 (2^4) times!

THIS code will run too:

```
    velocities = velocities + [5]
print(velocities)
[0.0, 9.81, 19.62, 29.43, 5, 5, 5, 5]
```

Exam

10. Will these statements run? If not, why not? (assume any variables have values)

c. s10 == 93.3 + 4

- returns boolean

12. What is wrong with the following function code? I want to print the number of rabbits with each loop, increasing the rabbit number each time by 2, up to and including 200 rabbits. But it won't run. Fix ALL errors so it will run and follow best practices by writing or drawing corrections. Indents are already either 4 spaces or none..

```
def multrabbits
    rabbits = 0  # I am setting it at one first.
while rabbits >= 200
    rabbits = rabbits + 2  *** Here it increases one at a time
print(rabbits)
```

```
def multrabbits(*args):
    rabbits = 0 # I am setting it at zero first.
    while rabbits <= 200:  # indented + 4
        print(rabbits)  # indented + 4
        rabbits = rabbits + 2 # Here it increases two at a time  # indented + 8

multrabbits(0) # or multrabbits()
```

Exam

14. How many times will the print command in the block be run?

```
country = 'United Arab Emirates' # length = 20
for ch in country:
   if not ch.islower():
       print(ch)
```

I accepted 3, but it is 5 times (spaces are not lower case)

```
   if ch.isupper():  - will run 3 times (U,A,E)
```

Exam

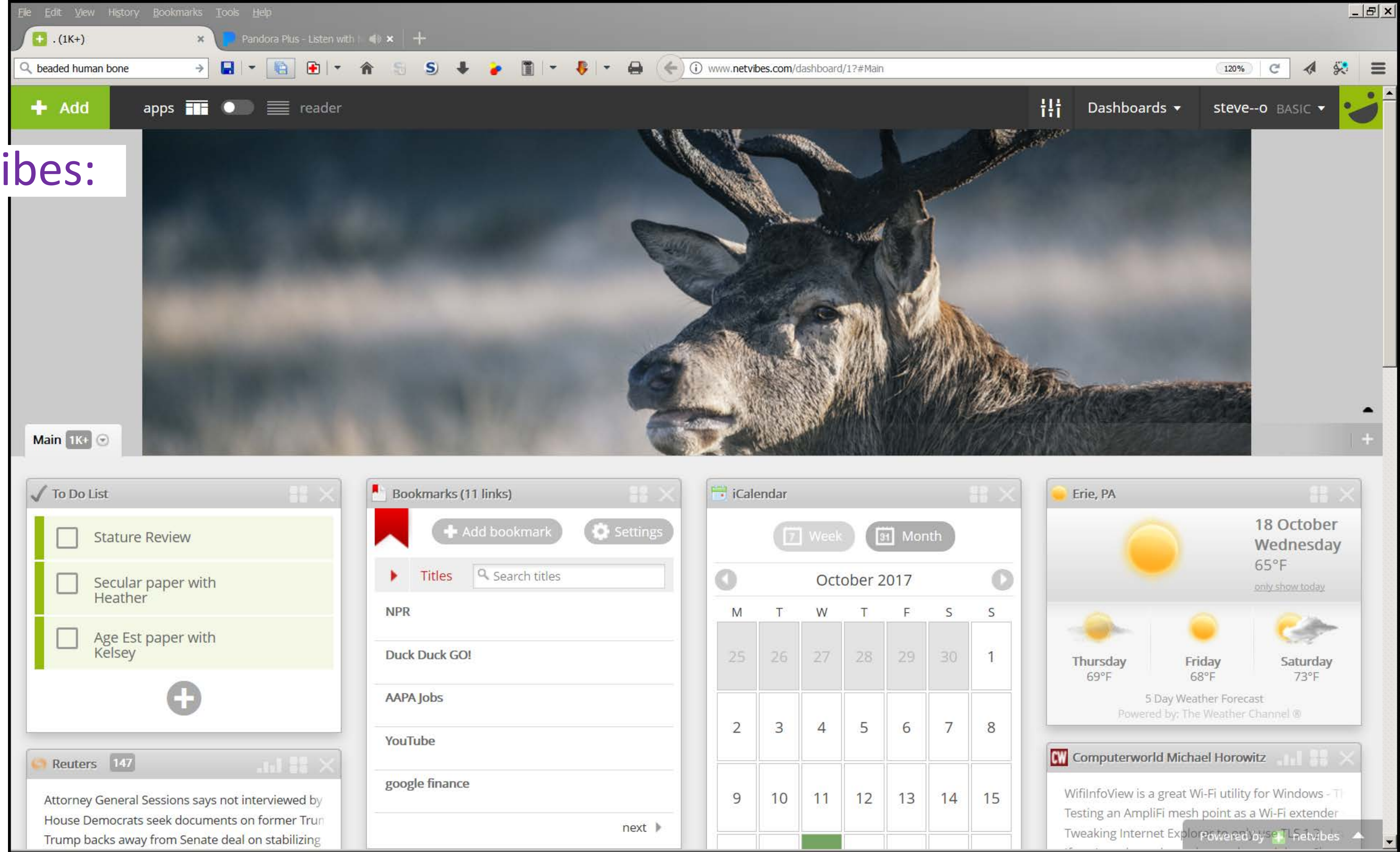
15. Write out a function that will calculate the mean of all numbers in a list called ScoresD that is passed to it. All items will be numbers. You do not need to follow the function design recipe.

```
def automean(list1):
    print(sum(list1)/len(list1))
```

16. Write code that will produce a list of all leap years from 1800 up to and including 2050. 1800 was a leap year.

```
list(range(1800,2051,4))
or
list(range(1800,2050,4)) # because 2050 is not a leap year
```

NetVibes:



File Edit View History Bookmarks Tools Help
. (1K+)
Pandora Plus - Listen with
beaded human bone
www.netvibes.com/dashboard/1?#Main
120%
Add
apps
reader
Dashboards
steve--o BASIC
Main 1K+

To Do List

- Stature Review
- Secular paper with Heather
- Age Est paper with Kelsey

Bookmarks (11 links)

Add bookmark | Settings

Titles | Search titles

- NPR
- Duck Duck GO!
- AAPA Jobs
- YouTube
- google finance

next

iCalendar

Week | Month

October 2017

M	T	W	T	F	S	S
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15

Erie, PA

18 October Wednesday 65°F

only show today

Thursday	Friday	Saturday
69°F	68°F	73°F

5 Day Weather Forecast

Powered by: The Weather Channel ®

Reuters 147

Attorney General Sessions says not interviewed by

House Democrats seek documents on former Trur

Trump backs away from Senate deal on stabilizing

Computerworld Michael Horowitz

WifiInfoView is a great Wi-Fi utility for Windows - T

Testing an AmpliFi mesh point as a Wi-Fi extender

Tweaking Internet Explo

Powered by netvibes

Storing Data: Sets

We have mentioned lists :

```
list2 = [1,2,"red", 'dfgg']  # square brackets
```

We also have **sets**

sets do not have a certain order and have no duplicates (similar to a shopping list)

```
vowelset = {'a', 'e', 'i', 'o', 'u'} # braces / curly brackets
{'e', 'o', 'a', 'u', 'i'}
```

- they are either there, or not there

```
vowelset2 = {'i', 'e', 'a', 'o', 'u', 'u', 'u', 'u', 'u'} # curly brackets
{'e', 'o', 'a', 'u', 'i'}
 vowelset == vowelset2
True
```

- so we are testing if ALL elements in one set are in another set

Storing Data: Sets

Sets are like checklists - or a catalog - things in a set

To create an empty set, use the set function and parentheses (NOT curly brackets)

```
tobeaddedto = set() # but it is displayed with curly brackets
>>> type(tobeaddedto)
<class set>
newset = {1,2,3,4,2,5,6,4,5,3,4}
newlist = [2,3,4,5,6,7]
```

To create a set from a list, use parens around brackets:

(BUT sets are displayed with curly brackets)

```
newset = set([1,2,3,4,2,5,6,4,5,3,4])  # or newest = set(newlist)
# NOT newset = set(1,2,3,4,2,5,6,4,5,3,4) - will produce error
newset
{1, 2, 3, 4, 5, 6}
```

Set Methods

We can use range to fill in values

```
set(range(1,11))
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Sets have operations like sets in math - union, intersection, add, subtract

- they are MUCH faster than looping through lists to find things in common

```
vowelset.add('y')
{'o', 'e', 'u', 'i', 'a', 'y'} # in random position

vowelset.remove('y')
{'o', 'e', 'u', 'i', 'a'}
```

Set Methods

```
ten = set(range(10))
lows = {0, 1, 2, 3, 4}
odds = {1, 3, 5, 7, 9}

lows.intersection(odds) # what items are in common in both?
{1, 3}
lows.union(odds) # merge list of both (list has no duplicates)
{0, 1, 2, 3, 4, 5, 7, 9}

lows.clear() # empty the set
lows
set()
```

Set Methods

```
ten = set(range(10))    lows = {0, 1, 2, 3, 4}   odds = {1, 3, 5, 7, 9}

lows.difference(odds) # what items are in lows but NOT in odds
{0, 2, 4}
odds.difference(lows) # what items are in odds but NOT in lows
{9, 5, 7}
lows.symmetric_difference(odds) # what items are unique to each
{0, 2, 4, 5, 7, 9}

lows.issubset(ten)  # are ALL items in lows items in ten?
True
ten.issuperset(lows) # does ten have all items in lows? (are ALL items in lows items in ten?)
True
lows.issuperset(odds)
False
```

Set Methods

These methods can also be done using operators

Method Call	Operator	Example	Result
set1.difference(set2)	set1 - set2	`lows - odds`	`{0, 2, 4}`
set1.intersection(set2)	set1 & set2	`lows & odds`	`{1,3}`
set1.issubset(set2)	set1 <= set2	`lows <= odds`	`False`
set1.issuperset(set2)	set1 >= set2	`lows >= odds`	`False`
set1.union(set2)	set1 | set2	`lows | odds`	`{0, 1, 2, 3, 4, 5, 7, 9}`
set1.symmetric_difference(set2)	set1 ^ set2	`lows ^ odds`	`{0, 2, 4, 5, 7, 9}`

Storing Data: Sets

These methods work VERY WELL with word lists

save as birdlist.txt

```
canada goose
canada goose
long-tailed jaeger
canada goose
snow goose
canada goose
long-tailed jaeger
canada goose
northern fulmar
```

Storing Data: Sets

These methods work VERY WELL with word lists

save as read_bird_list.py

```
# read_bird_list.py
observations_file = open('birdlist.txt')
birds_observed = set()
for line in observations_file:
    bird = line.strip()
    birds_observed.add(bird)
```

and run

```
>>> birds_observed
{'northern fulmar', 'canada goose', 'long-tailed jaeger', 'snow goose'}

>>> for species in birds_observed:  # loop to print sets, like lists
    print(species)

# no duplicates!
```

Storing Data: Tuples

There are also tuples, like lists but can't be modified after assignment ("immutable")

- tuples use parentheses

```
>>> bases = ('A', 'C', 'G', 'T')
>>> type(bases)
<class 'tuple'>

>>> for base in bases:
    print(base)

fillin_tuple = () # empty tuple
single_item_tuple = (8,) # tuple with one item
single_item_tuple = (8) # tuple with one item
```

Storing Data: Tuples

"immutability": resistant to changing

```
>>> life = (['Canada', 76.5], ['United States', 75.5], ['Mexico', 72.0])
>>> life[0] = life[1]
Traceback (most recent call last):
  File "<pyshell#52>", line 1, in <module>
    life[0] = life[1]
TypeError: 'tuple' object does not support item assignment
```

But we can assign elements:

```
life[0][1] = 80.0
life
(['Canada', 80.0], ['United States', 75.5], ['Mexico', 72.0])
```

Storing Data: Tuples

Multiple values can be assigned to tuples

```
>>> 10,20
(10,20)
>>> x,y = 10,20  # right side is evaluated first
>>> x
10
>>> y
20
```

and using tuples makes it easy to switch values of variables

```
>>> t1 = 'first'
>>> t2 = 'second'

>>> t1
'first'

>>> t1, t2 = t2, t1
>>> t1
'second'
```

Storing Data: Dictionaries

What if we want to count things and put into a list [] /set {} / tuple ()?

```
# count_bird_list.py
observations_file = open('birdlist.txt')
bird_counts = []
for line in observations_file:
    bird = line.strip()
    found = False
    # Find bird in the list of bird counts.
    for entry in bird_counts:
        if entry[0] == bird:
            entry[1] = entry[1] + 1
            found = True

    if not found:
        bird_counts.append([bird, 1]) # first time

observations_file.close()
for entry in bird_counts:
    print(entry[0], entry[1])
```

This is fine for a short list - it works -
but it looks through **every** item to see if it matches
- what if there are thousands?

Storing Data: Dictionaries

We can look up values in a set, but they might not be there (add/increment)

Dictionaries store items with a string index rather than a numerical index

A dictionary is best for this: it is a key - value pair (a key is unique)

(similar to a key field in a data table)

The keys are immutable (can't be changed, though can be deleted)

We define a dictionary by putting key/value pairs

inside curly braces, separated by a colon (:)

```
bird_to_observations = {'canada goose': 3, 'northern fulmar': 1}
```

Storing Data: Dictionaries

We can look up values in a dictionary:

```
bird_to_observations['northern fulmar']
1

# try to find something not there
bird_to_observations['long-tailed jaeger']
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 'long-tailed jaeger'

# empty dictionary - paired brackets
bird_to_observations = {}

# add single record, one at a time
bird_to_observations['snow goose'] = 33
bird_to_observations['eagle'] = 999
```

Storing Data: Dictionaries

```
# change a value in a single record
bird_to_observations['eagle'] = 9  # was 999 before

#remove a dictionary entry
del bird_to_observations['snow goose']

# you can't remove a key if it is not there

# we can use if to see if an entry is in the dictionary:
>>> bird_to_observations = {'eagle': 999, 'snow goose': 33}
>>> if 'eagle' in bird_to_observations:
    print('eagles have been seen')

# remove eagles and it will not print
del bird_to_observations['eagle']
```

Storing Data: Dictionaries

```
# loop through a dictionary
```

for *«variable»* in *«dictionary»*:

«block»

```
# set some up
>>> bird_to_observations = {'canada goose': 183, 'long-tailed jaeger': 71,
    'snow goose': 63, 'northern fulmar': 1}

# print bird, numbers. notice that order is arbitrary
>>> for bird in bird_to_observations:
    print(bird, bird_to_observations[bird])
long-tailed jaeger 71
canada goose 183
northern fulmar 1
snow goose 63
```

This is using the bird as the key (index) to return the value linked to it.

Storing Data: Dictionaries

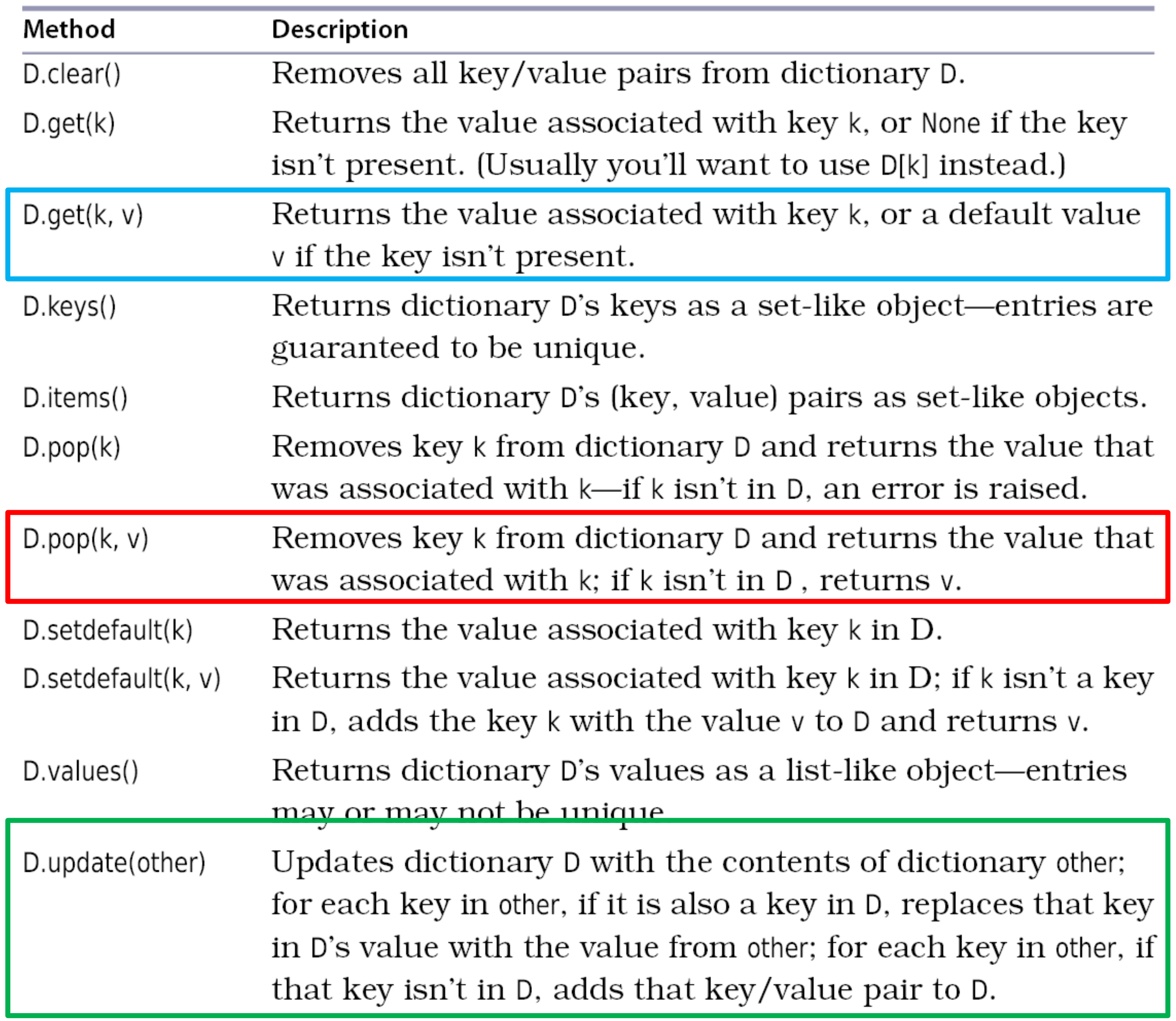
```
# let's look further at dictionaries
>>> scientist_to_birthdate = {'Newton' : 1642, 'Darwin' : 1809,
    'Turing' : 1912}

>>> scientist_to_birthdate.keys()
dict_keys(['Darwin', 'Newton', 'Turing'])
>>> scientist_to_birthdate.values()
dict_values([1809, 1642, 1912])
>>> scientist_to_birthdate.items()
dict_items([('Darwin', 1809), ('Newton', 1642), ('Turing', 1912)])
>>> scientist_to_birthdate.get('Newton')
1642
```

Storing Data: Dictionaries

```
# create another birthday set
>>> researcher_to_birthdate = {'Curie' : 1867, 'Hopper' : 1906,
    'Franklin' : 1920}

# merge dictionaries
>>> scientist_to_birthdate.update(researcher_to_birthdate)

>>> scientist_to_birthdate
{'Hopper': 1906, 'Darwin': 1809, 'Turing': 1912, 'Newton': 1642,
'Franklin': 1920, 'Curie': 1867}

# researcher unchanged
>>> researcher_to_birthdate
{'Franklin': 1920, 'Hopper': 1906, 'Curie': 1867}

# empty a dictionary
researcher_to_birthdate.clear()
```

Grace Hopper

Dictionary methods



Method	Description
D.clear()	Removes all key/value pairs from dictionary D.
D.get(k)	Returns the value associated with key k, or None if the key isn't present. (Usually you'll want to use D[k] instead.)
D.get(k, v)	Returns the value associated with key k, or a default value v if the key isn't present.
D.keys()	Returns dictionary D's keys as a set-like object—entries are guaranteed to be unique.
D.items()	Returns dictionary D's (key, value) pairs as set-like objects.
D.pop(k)	Removes key k from dictionary D and returns the value that was associated with k—if k isn't in D, an error is raised.
D.pop(k, v)	Removes key k from dictionary D and returns the value that was associated with k; if k isn't in D , returns v.
D.setdefault(k)	Returns the value associated with key k in D.
D.setdefault(k, v)	Returns the value associated with key k in D; if k isn't a key in D, adds the key k with the value v to D and returns v.
D.values()	Returns dictionary D's values as a list-like object—entries may or may not be unique.
D.update(other)	Updates dictionary D with the contents of dictionary other; for each key in other, if it is also a key in D, replaces that key in D's value with the value from other; for each key in other, if that key isn't in D, adds that key/value pair to D.

all use parentheses

Dictionaries

Dictionaries remind me of a simple data table (select, merge, delete, update)

```
# values - linked to keys
scientist_to_birthdate.values()
dict_values([1867, 1912, 1906, 1920, 1809, 1642])

# we can loop over keys and values in a dictionary
>>> for scientist, birthdate in scientist_to_birthdate.items():
    print(scientist, 'was born in', birthdate)

Curie was born in 1867
Turing was born in 1912
Hopper was born in 1906
Franklin was born in 1920
Darwin was born in 1809
Newton was born in 1642
```

Dictionaries

A better way to count things in a set/list

```
# count_bird_list2.py
observations_file = open('birdlist.txt')
bird_to_observations = {}
for line in observations_file:
    bird = line.strip()
    if bird in bird_to_observations:
        bird_to_observations[bird] = bird_to_observations[bird] + 1
    else:
        bird_to_observations[bird] = 1

observations_file.close()

# Print each bird and the number of times it was seen.
for bird, observations in bird_to_observations.items():
    print(bird, observations)
```

Dictionaries

A better way to count things in a set/list

```
# count_bird_list3.py
observations_file = open('birdlist.txt')
bird_to_observations = {}
for line in observations_file:
    bird = line.strip()
    bird_to_observations[bird] = bird_to_observations.get(bird, 0) + 1

observations_file.close()

# Print each bird and the number of times it was seen.
for bird, observations in bird_to_observations.items():
    print(bird, observations)
```

Dictionaries

A better way to count things in a set/list

Sort your dictionary

```
>>> sorted_birds = sorted(bird_to_observations.keys())
>>> for bird in sorted_birds:
    print(bird, bird_to_observations[bird])
canada goose 5
long-tailed jaeger 2
northern fulmar 1
snow goose 1

these are sorted alphabetically. what about sorting by counts?
>>> sorted_birds = sorted(bird_to_observations.values())
>>> for bird in sorted_birds:
    print(bird, bird_to_observations[bird])

Traceback (most recent call last):
  File "<pyshell#32>", line 2, in <module>
    print(bird, bird_to_observations[bird])
KeyError: 1
```

Dictionaries

We want an inverted dictionary

```
# bird_count_ID.py
# Invert the dictionary
# bird, count
bird_to_observations = {'canada goose': 183, 'long-tailed jaeger': 71,
    'snow goose': 63, 'northern fulmar': 1, 'snow goose': 1}

observations_to_birds_list = {}
for bird, observations in bird_to_observations.items():
    if observations in observations_to_birds_list:  # count is the index
        observations_to_birds_list[observations].append(bird)
    else:
        observations_to_birds_list[observations] = [bird]

print(observations_to_birds_list)
```

```
{1: ['northern fulmar', 'snow goose'], 71: ['long-tailed jaeger'], 183: ['canada
   goose']}
```

Dictionaries

Print the inverted dictionary

```
# bird_count_ID.py
# Invert the dictionary
# bird, count
bird_to_observations = {'canada goose': 183, 'long-tailed jaeger': 71,
    'snow goose': 63, 'northern fulmar': 1, 'snow goose': 1}

observations_to_birds_list = {}
for bird, observations in bird_to_observations.items():
    if observations in observations_to_birds_list:  # count is the index
        observations_to_birds_list[observations].append(bird)
    else:
        observations_to_birds_list[observations] = [bird]

# print(observations_to_birds_list)
# Print the inverted dictionary
observations_sorted = sorted(observations_to_birds_list.keys())

for observations in observations_sorted:
    print(observations, ':', end=" ")

    for bird in observations_to_birds_list[observations]:
        print(' ', bird, end=" ")
    print()
```

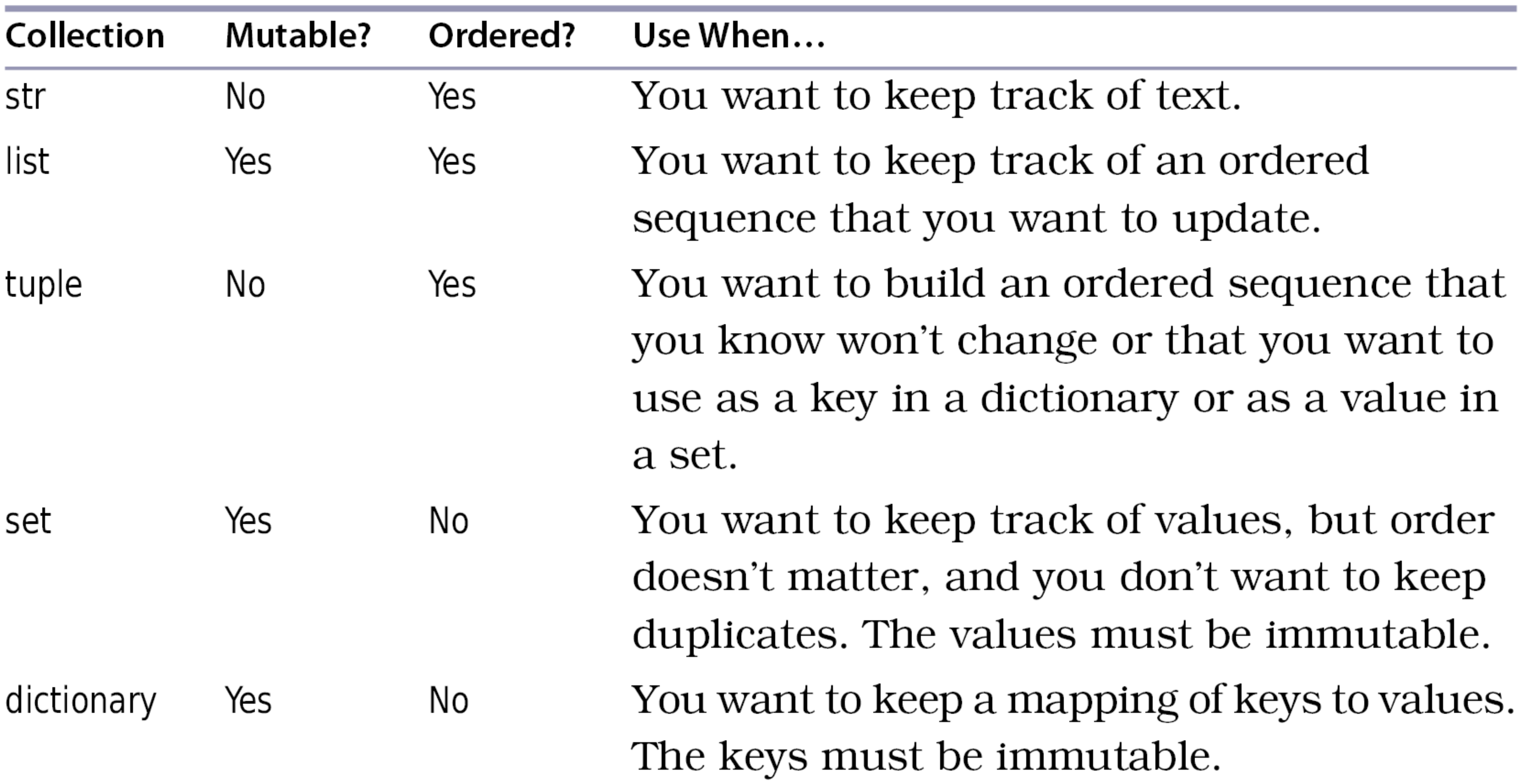
Dictionaries

```
1 :    snow goose   northern fulmar
71 :    long-tailed jaeger
183 :    canada goose
```

Boolean tests of collections:

```
>>> odds = set([1, 3, 5, 7, 9])
>>> 9 in odds
True
>>> 8 in odds
False

>>> bird_to_observations = {'canada goose': 183, 'long-tailed jaeger': 71,
    'snow goose': 63, 'northern fulmar': 1}
>>> 'snow goose' in bird_to_observations
True
```

Data "Collections"



Collection	Mutable?	Ordered?	Use When…
str	No	Yes	You want to keep track of text.
list	Yes	Yes	You want to keep track of an ordered sequence that you want to update.
tuple	No	Yes	You want to build an ordered sequence that you know won't change or that you want to use as a key in a dictionary or as a value in a set.
set	Yes	No	You want to keep track of values, but order doesn't matter, and you don't want to keep duplicates. The values must be immutable.
dictionary	Yes	No	You want to keep a mapping of keys to values. The keys must be immutable.

Homework 12

Due before class Monday

Gries 11.8 on page 219

1 2 4 6 8

NOTE for all homeworks: Gries web solutions may be incorrect; at a minimum double-check the recipe, use better variable names, test your code, and provide what the question asks for.

For Next Time

From The Programming Historian: (http://programminghistorian.org)

and https://programminghistorian.org/lessons/

visit

http://programminghistorian.org/lessons/working-with-web-pages

https://programminghistorian.org/lessons/from-html-to-list-of-words-1

https://programminghistorian.org/lessons/from-html-to-list-of-words-2

https://programminghistorian.org/lessons/counting-frequencies

and read through them.