Quiz Section Week 6
May 2, 2017

Exams

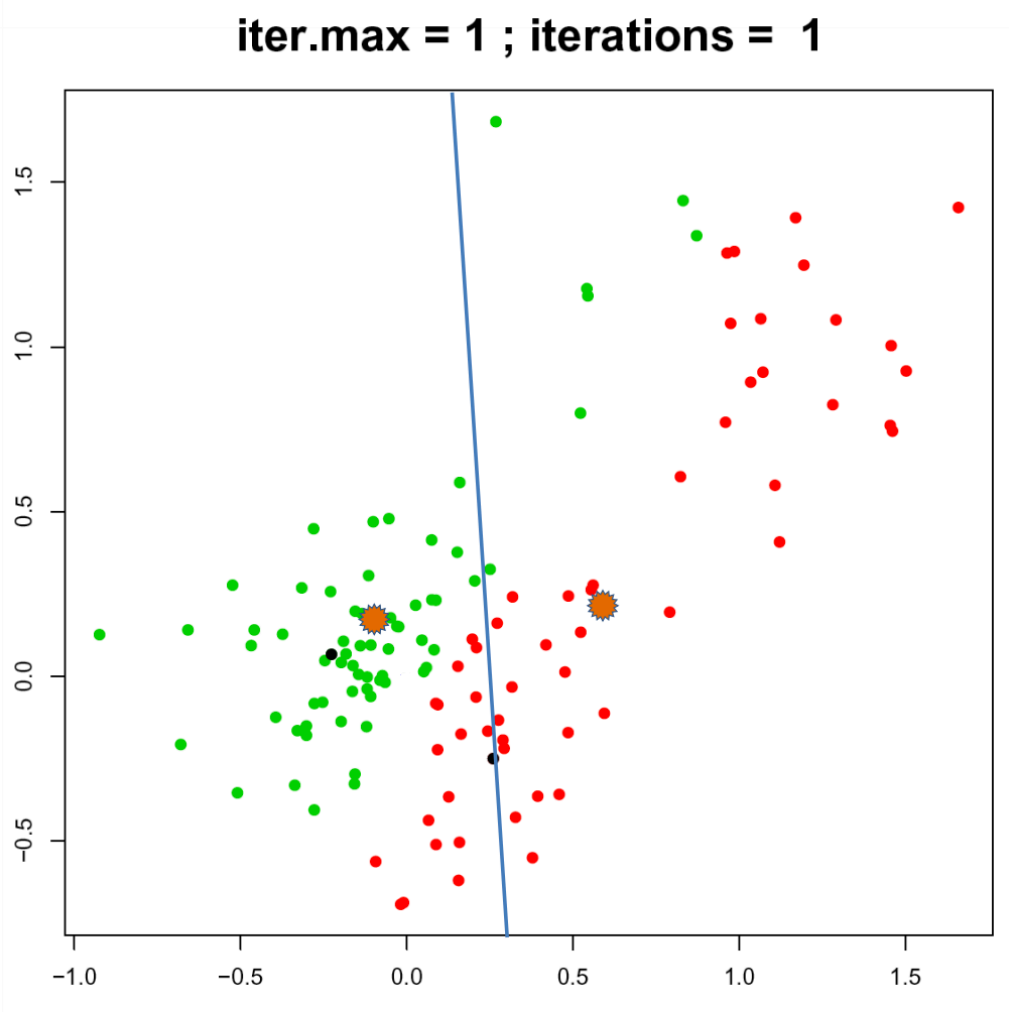
More on functions

A bit more on input/output, lists

K-means **termination criteria**

The K-means algorithm finds clusters by iteratively optimizing and re-calculating clusters…when to stop?

- When we've found a good solution
  - Clusters don't change
  - Centers don't move
- When it becomes clear there is no good solution
  - Reach an arbitrary max # of iterations



Remember: A programming language has different elements that you can combine in infinite ways

Variables in different flavors/structures

Simple data types

- integer
- float (numeric)
- character
- Boolean

Collection data types

- string (list of characters)
- list of integers
- dictionaries
- list of dictionaries
- etc

Functions and Operators

sum

len

and

...

Control statements

- If/elif/else
- for
- while
- def
- return

Sometimes more than 1 option for how to store and access some data

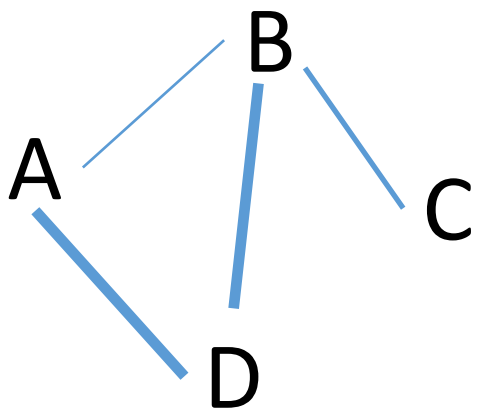
```
distances =
[[0,1,2,4],[1,0,2,5],[2,2,0,5],[4,5,5,0]]
```

```
distances = {}
distances['A'] = {'B':1, 'C':2, 'D':4}
distances['B'] = {'A':1, 'C':2, 'D':5}
distances['C'] = {'A':2, 'B':2, 'D':5}
distances['D'] = {'A':4, 'B':5, 'C':5}
```

	A	B	C	D
A	0	1	2	4
B	1	0	2	5
C	2	2	0	5
D	4	5	5	0

You could define a network the same way!

```
network = {}
network[('A','B')] = 1
network[('A','C')] = 2
network[('A','D')] = 4
network[('B','D')] = 4
# etc.
```



For directed networks?

Example: print the pair of points that have the minimum distance

```
minimum_distance = float('inf') #Infinity
closest_pair = [] #empty list
```

	A	B	C	D
A	0	1	2	4
B	1	0	2	5
C	2	2	0	5
D	4	5	5	0

Example: print the pair of points that have the minimum distance

	A	B	C	D
A	0	1	2	4
B	1	0	2	5
C	2	2	0	5
D	4	5	5	0

```
minimum_distance = float('inf') #Infinity
closest_pair = [] #empty list

for x in distances:
    for y in distances[x]:
        if distances[x][y] < minimum_distance:
            minimum_distance = distances[x][y]
            closest_pair = [x,y]
print closest_pair, minimum_distance
```

Programming question

What condition should we check?

If it's found in list2

How many and what for loops do we need to check each item?

Check whether every element in list1 is also found in list2

```
def my_intersection(list1, list2):
```

```
list_num = my_intersection([1,3,5,21],[5,4,19,21])
print list_num # should print [5,21]
```

```
list_mix =
my_intersection([1,"r","hello",7],[4,"hello"])
print list_mix # should print ["hello"]
```

Programming question

What condition should we check?

If it's found in list2

How many and what for loops do we need to check each item?

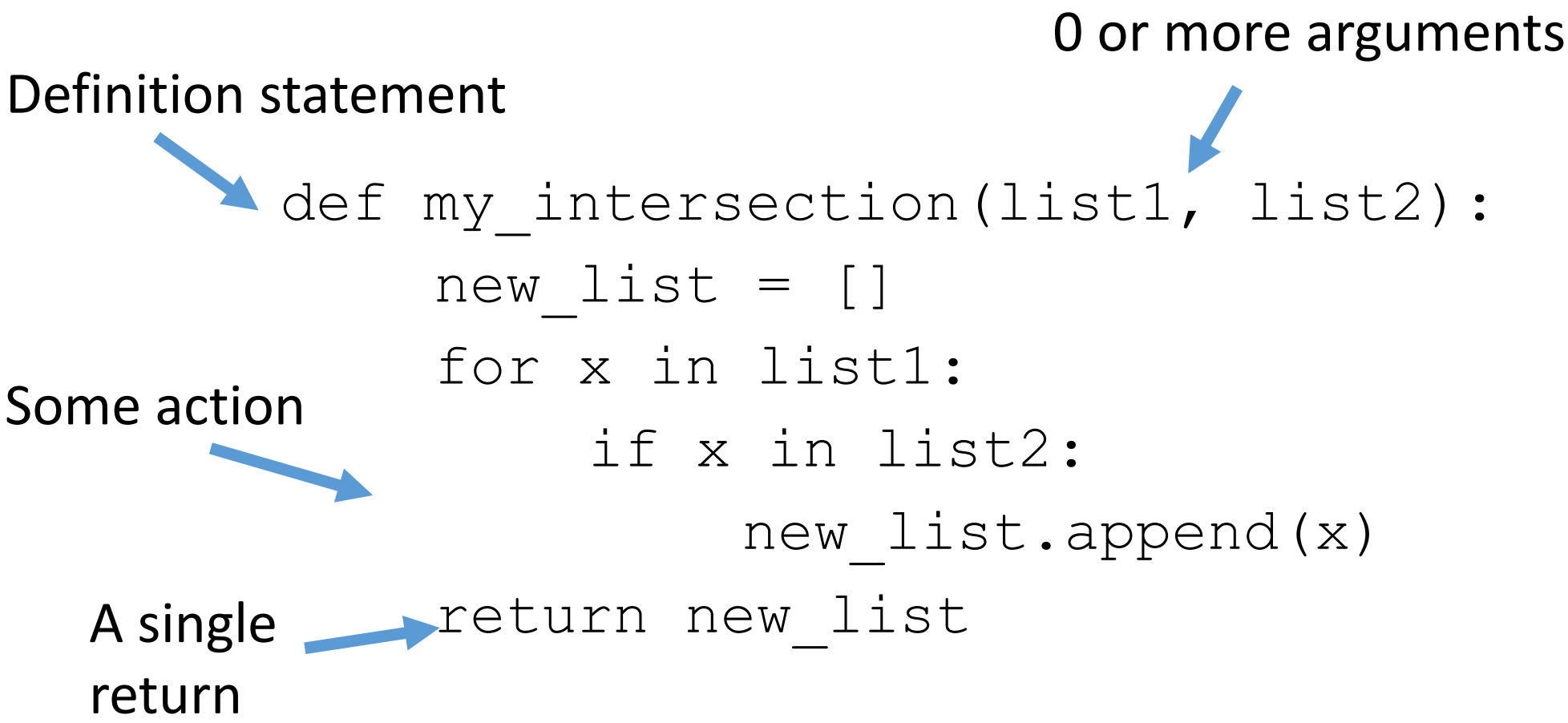
Check whether every element in list1 is also found in list2

```
def my_intersection(list1, list2):
        new_list = []
        for x in list1:
                if x in list2:
                        new_list.append(x)
        return new_list

list_num = my_intersection([1,3,5,21],[5,4,19,21])
print list_num # should print [5,21]

list_mix =
my_intersection([1,"r","hello",7],[4,"hello"])
print list_mix # should print ["hello"]
```

Reminder: Anatomy of a function



Once you've defined a function, you can use it again and again in many different ways!

```
list_num =
my_intersection([1,3,5,21],[5,4,19,21])
print list_num # should print [5,21]

list_mix =
my_intersection([1,"r","hello",7],[4,"hello"])
print list_mix # should print ["hello"]
```

You can use a function in another function! E.g. Homework 4

```
def average_distance(point1, point_list, euclidean):
      #1) Calculate distance from point1 to each point in point list
      sum_dists = 0
      for j in range(len(point_list)):
            if(euclidean):
                  #What goes here?
            else:
                  #What goes here?
      #2) Calculate the average of the resulting distances and
return this value
      avg_dist = float(sum_dists)/len(point_list)
      return avg_dist
```

You just have to define or import the definition of a function before you can use it

You can use a function in another function! E.g. Homework 4

```
def average_distance(point1, point_list, euclidean):
      #1) Calculate distance from point1 to each point in point list
      sum_dists = 0
      for j in range(len(point_list)):
            if(euclidean):
                  sum_dists = sum_dists + euclidean_distance(point1,
point_list[j])
            else:
                  sum_dists = sum_dists + manhattan_distance(point1,
point_list[j])
      #2) Calculate the average of the resulting distances and return this
value
      avg_dist = float(sum_dists)/len(point_list)
      return avg_dist
```

You just have to define or import the definition of a function before you can use it

You can even use a function from a different file!

intersection_function.py:

```
def my_intersection(list1,
list2):
      new_list = []
      for x in list1:
        if x in list2:
            new_list.append(x)
      return new_list
```

calc_intersections.py:

```
#This line imports all function
#definitions from the file
#intersection_function.py
from intersection_function import *

list_num =
my_intersection([1,3,5,21],[5,4,19,21])
print list_num

list_mix =
my_intersection([1,"r","hello",7],[4,"hel
lo"])
print list_mix
```

You can provide default values for function arguments

```
def less_than(myList, num=4):
    new_list = []
    for x in myList:
        if x < num:

    new_list.append(x)
    return new_list
```

```
>>> less_than([12,3,7]) # will use default value for num
```

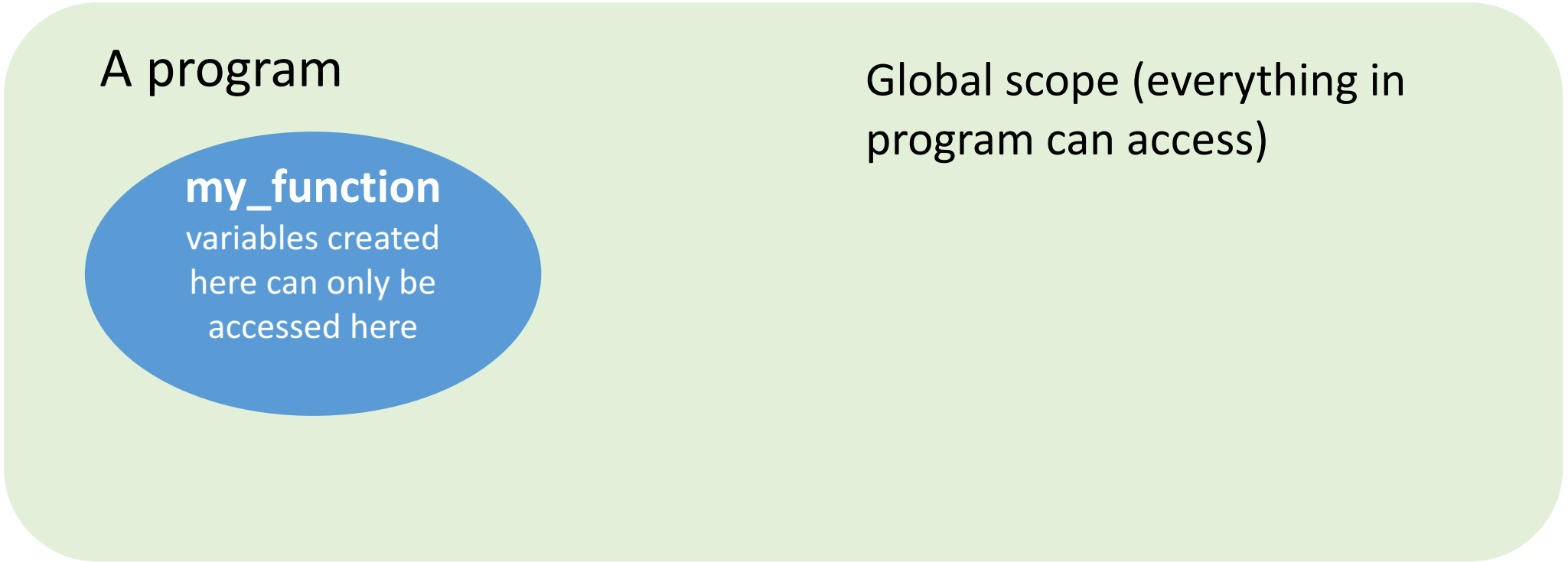
[3]

```
>>> less_than([12,3,7], num = 8)
```

[3,7]

Scope of a variable

- Variables created in the main part of your program can be accessed anywhere (**global** scope)
- Variables created within functions are only accessible within that function (**local** scope)



Scope of a variable

```
new_list = [0,1,2]

def less_than(myList, num = 4):
     new_list = []
     for x in myList:
          if x < num:
               new_list.append(x)
     return new_list

print new_list
anotherList = [3,7,12]
print less_than(anotherList)
```

Scope of a variable

```
new_list = [0,1,2]

def less_than(myList, num = 4):
     #new_list = []
     for x in myList:
          if x < num:
               new_list.append(x)
     return new_list

print new_list
anotherList = [3,7,12]
print less_than(anotherList)
```

Don't do this!! You'll confuse yourself

Define all your functions at the beginning of your program or in another file

Example program structure with input/output

python analyze_sequence_pairs.py inputfile.txt outputfile.txt

Setup

```
#import needed modules and functions
#
```

Input

```
#Read in data from file
```

```
#Do a calculation
```

Output

```
#Write output to file
```

Example program structure with input/output

python analyze_sequence_pairs.py inputfile.txt outputfile.txt

Setup

```
import sys
from qs6 import * #import the definition of calculate_jukes_cantor
```

Input

```
fin = open(sys.argv[1],'r')
seqs = []
for line in fin:
     seqs.append(line.rstrip()) # gets rid of \n at the end of the
line
print seqs
fin.close()
```

```
answer = calculate_jukes_cantor(seqs[0], seqs[1])
```

Output

```
fout = open(sys.argv[2],'w')
fout.write( seqs[0] + ' ' + seqs[1] + ' ')
fout.write( str(answer) + '\n')
fout.close()
```

Lists (and strings): Some helpful ways to access and modify

```
>>> my_list = [1,2,3]
>>> my_list.append(4)
>>> my_list.remove(4)
>>> my_list.pop()
>>> my_list.extend([4,5,6])
#compare with .append([4,5,6]
>>> my_list[2]
>>> my_list[2:4]
>>> my_list[1:]
>>> my_list[-1]
>>> my_list.sort() #Doesn't output anything!
>>> print my_list
>>> my_list.sort(reverse = True)
>>> print my_list
```

Exercise: modify the Jukes-Cantor program to instead calculate and write to a file the # of times a start codon occurs in each sequence

Use this function:

```
def count_start_codons(seq):
    num_starts = seq.count("ATG")
    return num_starts
```

```
python count_starts.py sequences.txt output_file.txt
```

output file:

ATGGGGGATG	2
CAGTTATGCCT	1

Reminder: Tons of resources online for extra programming practice

- I still recommend this one:
  - http://interactivepython.org/runestone/static/thinkcspy/index.html

- You can use the help() function to learn about what other functions do:

```
>>> help(len)
>>> help(open)
>>> my_list = []
>>> help(my_list.append)
```