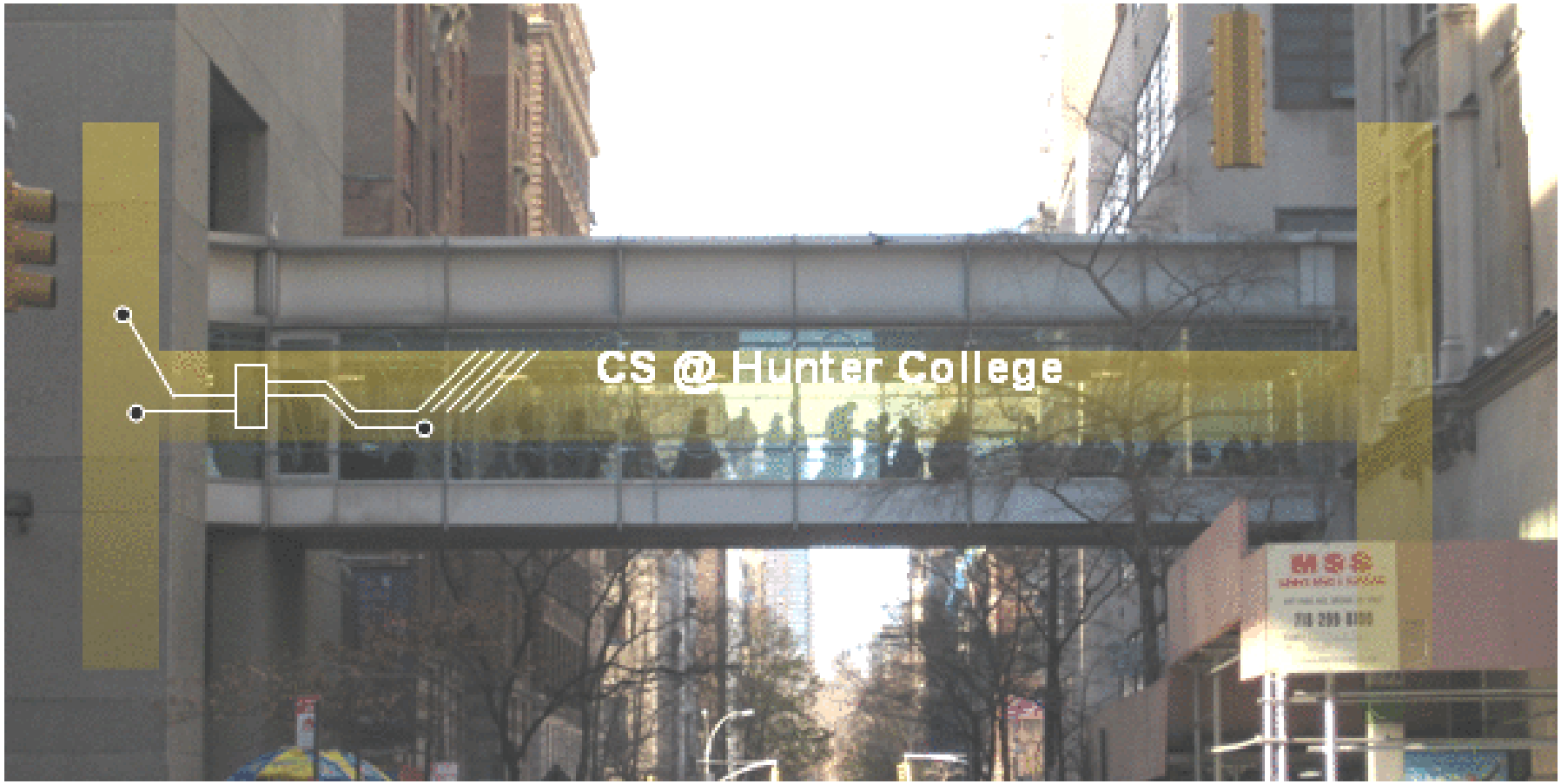
CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Reminder

- This is a large course

Reminder

- This is a large course
- We have college-mandated regulations to keep the lab at capacity

Reminder

- This is a large course
- We have college-mandated regulations to keep the lab at capacity
- We have a limited number of UTAs

Reminder

- This is a large course
- We have college-mandated regulations to keep the lab at capacity
- We have a limited number of UTAs
- **You must make an appointment to visit the lab**

Reminder

- This is a large course
- We have college-mandated regulations to keep the lab at capacity
- We have a limited number of UTAs
- **You must make an appointment to visit the lab**
- **You will be admitted in the lab at the time of your appointment**

Frequently Asked Questions

From email and tutoring.

- **How do I prepare for the final exam?**

Frequently Asked Questions

From email and tutoring.

- **How do I prepare for the final exam?**
  Assuming you are already attending lecture meetings and reading the Online Lab each week,

Frequently Asked Questions

From email and tutoring.

- **How do I prepare for the final exam?**
  Assuming you are already attending lecture meetings and reading the Online Lab each week,
  - *Take the quizzes, if you get a wrong answer, review it and make sure you understand.*

Frequently Asked Questions

From email and tutoring.

- **How do I prepare for the final exam?**
  Assuming you are already attending lecture meetings and reading the Online Lab each week,
  - *Take the quizzes, if you get a wrong answer, review it and make sure you understand.*
  - *Work-on and* **understand** *the programming assignments.*

Frequently Asked Questions

From email and tutoring.

- **How do I prepare for the final exam?**
  Assuming you are already attending lecture meetings and reading the Online Lab each week,
  - *Take the quizzes, if you get a wrong answer, review it and make sure you understand.*
  - *Work-on and* **understand** *the programming assignments.*
  - *Take past exams available on the course webpage. Take it without looking at the answers (give yourself 1.5 hours) then compare with answer key.*

Frequently Asked Questions

From email and tutoring.

- **How do I prepare for the final exam?**
  Assuming you are already attending lecture meetings and reading the Online Lab each week,
  - *Take the quizzes, if you get a wrong answer, review it and make sure you understand.*
  - *Work-on and* **understand** *the programming assignments.*
  - *Take past exams available on the course webpage. Take it without looking at the answers (give yourself 1.5 hours) then compare with answer key.*
  - *Condense the skeletal notes we provide for each lab into a smaller set of notes for quick reference.*

Frequently Asked Questions

From email and tutoring.

- **How do I prepare for the final exam?**
  Assuming you are already attending lecture meetings and reading the Online Lab each week,
  - *Take the quizzes, if you get a wrong answer, review it and make sure you understand.*
  - *Work-on and* **understand** *the programming assignments.*
  - *Take past exams available on the course webpage. Take it without looking at the answers (give yourself 1.5 hours) then compare with answer key.*
  - *Condense the skeletal notes we provide for each lab into a smaller set of notes for quick reference.*
  - *As you practice, keep refining you reference sheet that you can keep handy during the exam (write down anything you wished you could quickly look up while taking the practice exam)*

Frequently Asked Questions

From email and tutoring.

- **How do I prepare for the final exam?**
  Assuming you are already attending lecture meetings and reading the Online Lab each week,
  - *Take the quizzes, if you get a wrong answer, review it and make sure you understand.*
  - *Work-on and* **understand** *the programming assignments.*
  - *Take past exams available on the course webpage. Take it without looking at the answers (give yourself 1.5 hours) then compare with answer key.*
  - *Condense the skeletal notes we provide for each lab into a smaller set of notes for quick reference.*
  - *As you practice, keep refining you reference sheet that you can keep handy during the exam (write down anything you wished you could quickly look up while taking the practice exam)*
  - *If you don't understand a question (from quiz or past exam) or a programming assignment, go to drop-in tutoring and ask a TA to explain.*

Frequently Asked Questions

From email and tutoring.

- **How do I prepare for the final exam?**
  Assuming you are already attending lecture meetings and reading the Online Lab each week,
  - *Take the quizzes, if you get a wrong answer, review it and make sure you understand.*
  - *Work-on and* **understand** *the programming assignments.*
  - *Take past exams available on the course webpage. Take it without looking at the answers (give yourself 1.5 hours) then compare with answer key.*
  - *Condense the skeletal notes we provide for each lab into a smaller set of notes for quick reference.*
  - *As you practice, keep refining you reference sheet that you can keep handy during the exam (write down anything you wished you could quickly look up while taking the practice exam)*
  - *If you don't understand a question (from quiz or past exam) or a programming assignment, go to drop-in tutoring and ask a TA to explain.*
  - *More practice opportunities will be provided closer to the exam.*

Today's Topics

- Recap: Functions & Top Down Design
- Mapping GIS Data
- Random Numbers
- Indefinite Loops

Today's Topics

- **Recap: Functions & Top Down Design**
- Mapping GIS Data
- Random Numbers
- Indefinite Loops

Challenge:

```
def prob4(amy, beth):
    if amy > 4:
        print("Easy case")
        kate = -1
    else:
        print("Complex case")
        kate = helper(amy,beth)
    return(kate)
```

```
def helper(meg,jo):
    s = ""
    for j in range(meg):
        print(j, ": ", jo[j])
        if j % 2 == 0:
            s = s + jo[j]
            print("Building s:", s)
    return(s)
```

- What are the formal parameters for the functions?
- What is the output of:

```
r = prob4(4,"city")
print("Return:  ", r)
```

- What is the output of:

```
r = prob4(2,"university")
print("Return:  ", r)
```

Challenge:

```
def prob4(amy, beth):
    if amy > 4:
        print("Easy case")
        kate = -1
    else:
        print("Complex case")
        kate = helper(amy,beth)
    return(kate)
```

```
def helper(meg,jo):
    s = ""
    for j in range(meg):
        print(j, ": ", jo[j])
        if j % 2 == 0:
            s = s + jo[j]
            print("Building s:", s)
    return(s)
```

- What are the formal parameters for the functions?

Challenge:

```
def prob4(amy, beth):
    if amy > 4:
        print("Easy case")
        kate = -1
    else:
        print("Complex case")
        kate = helper(amy,beth)
    return(kate)
```

Formal Parameters

```
def helper(meg,jo):
    s = ""
    for j in range(meg):
        print(j, ": ", jo[j])
        if j % 2 == 0:
            s = s + jo[j]
            print("Building s:", s)
    return(s)
```

- What are the formal parameters for the functions?

Challenge:

```
def prob4(amy, beth):
    if amy > 4:
        print("Easy case")
        kate = -1
    else:
        print("Complex case")
        kate = helper(amy,beth)
    return(kate)
```

```
def helper(meg,jo):
    s = ""
    for j in range(meg):
        print(j, ": ", jo[j])
        if j % 2 == 0:
            s = s + jo[j]
            print("Building s:", s)
    return(s)
```

- What is the output of:

```
r = prob4(4,"city")
print("Return:  ", r)
```

- What is the output of:

```
r = prob4(2,"university")
print("Return:  ", r)
```

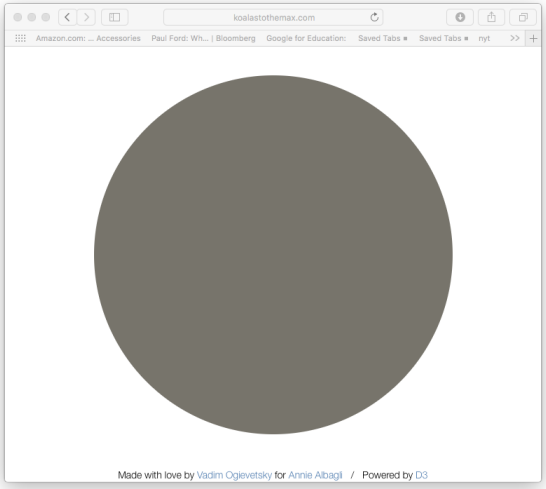
Python Tutor

```
def prob4(amy, beth):
     if amy > 4:
          print("Easy case")
          kate = -1
     else:
          print("Complex case")
          kate = helper(amy,beth)
     return(kate)
```

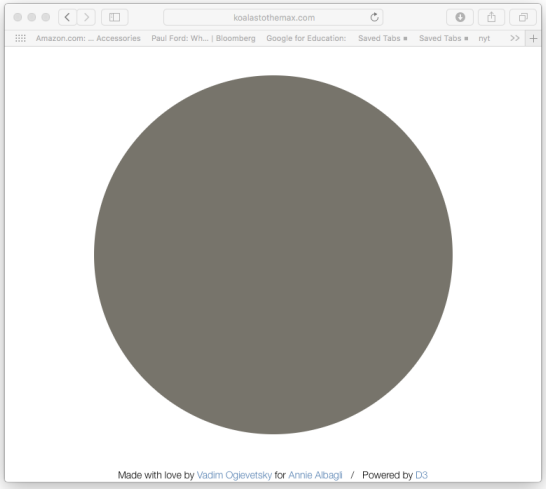
```
def helper(meg,jo):
     s = ""
     for j in range(meg):
          print(j, ": ", jo[j])
          if j % 2 == 0:
               s = s + jo[j]
               print("Building s:", s)
     return(s)
```

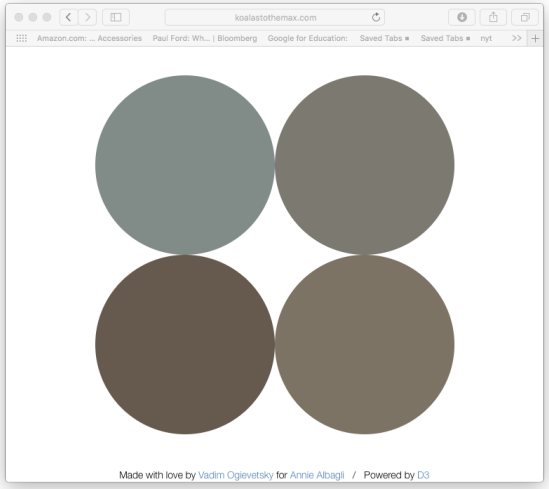
(Demo with pythonTutor)

From Last Time: koalas

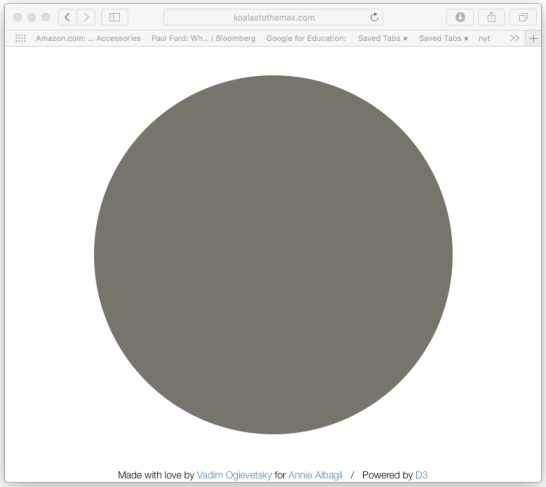


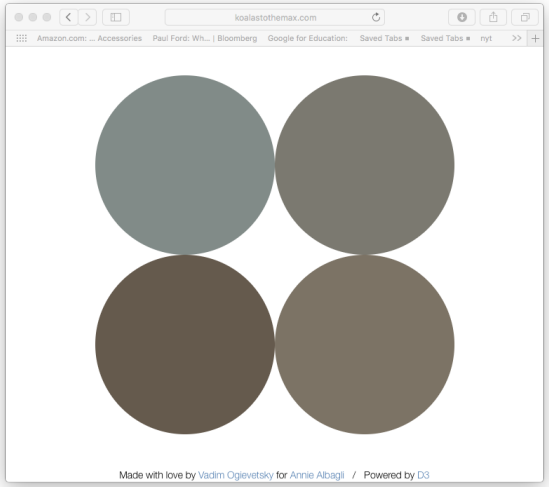
From Last Time: koalas

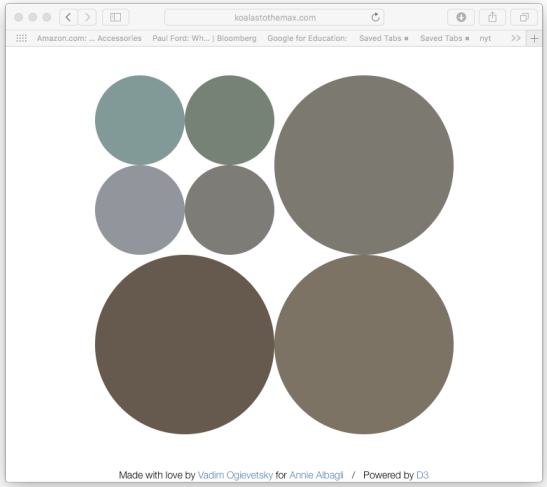




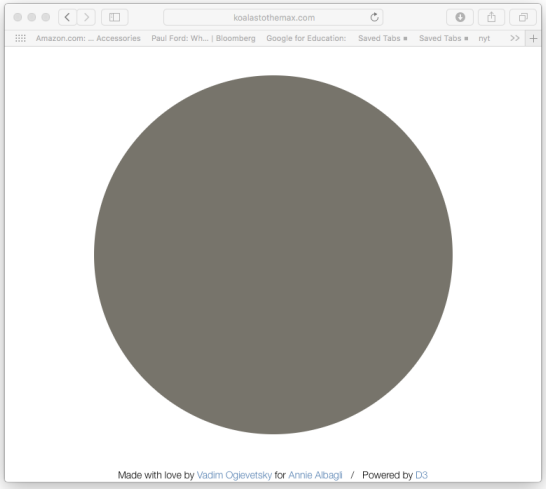
From Last Time: koalas

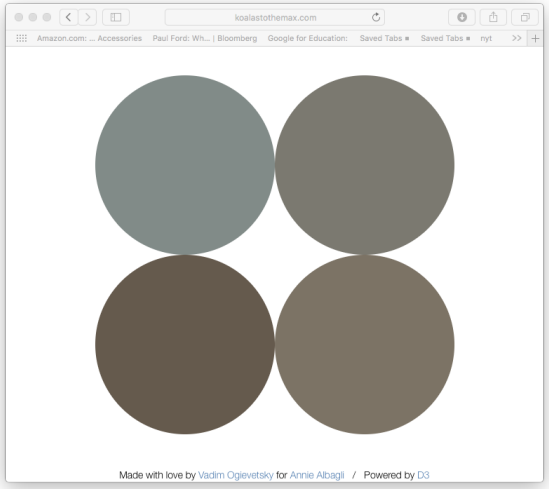


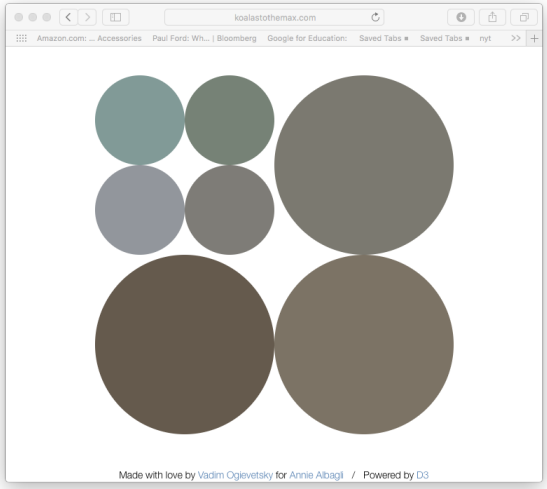




From Last Time: koalas

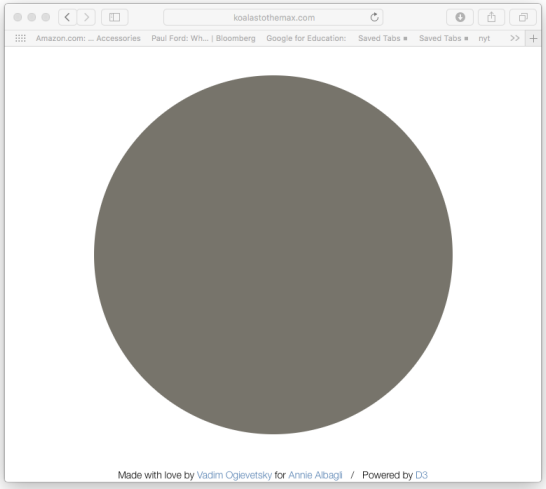


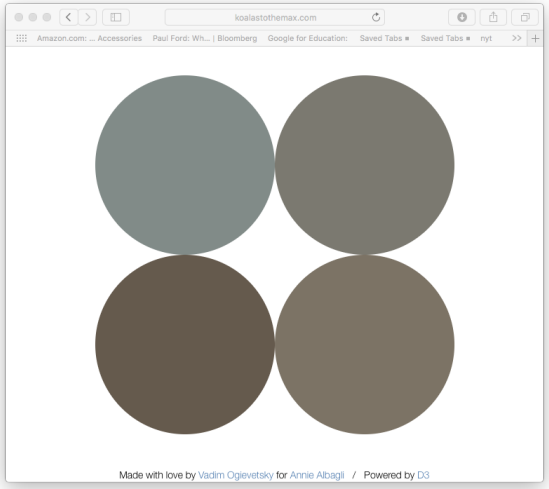


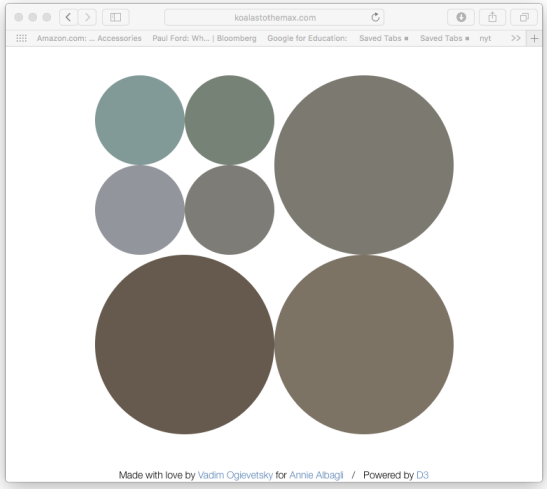


`http://koalastothemax.com`

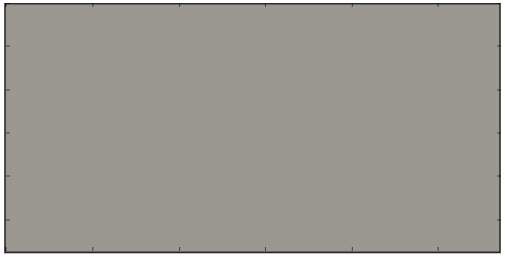
From Last Time: koalas



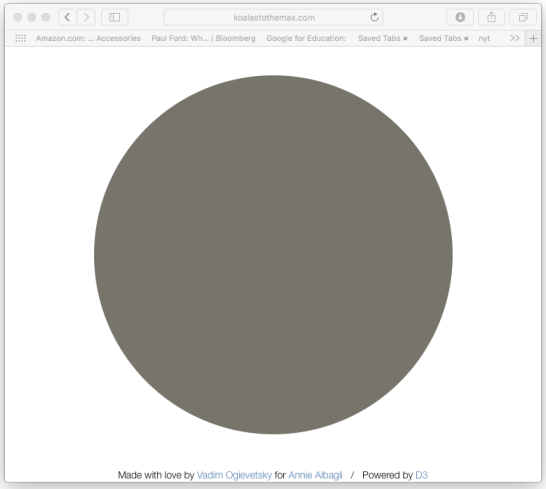


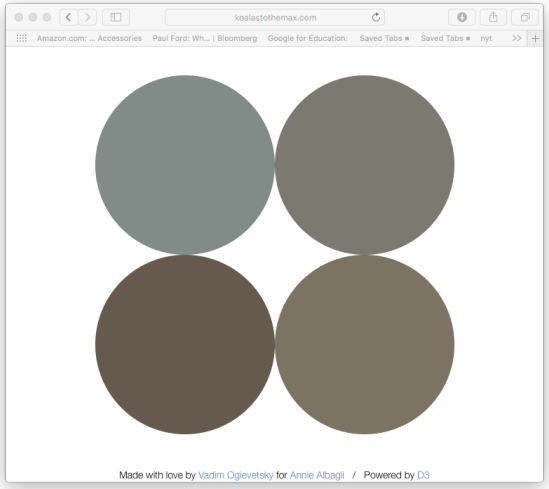


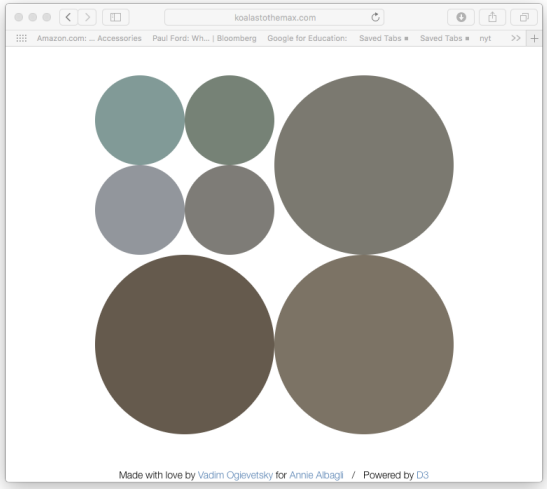
`http://koalastothemax.com`



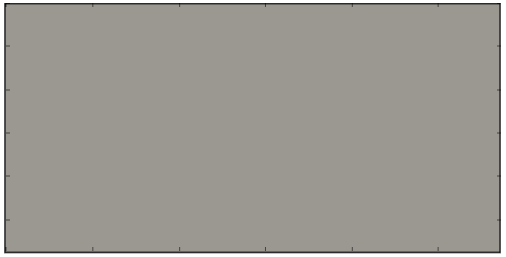
From Last Time: koalas

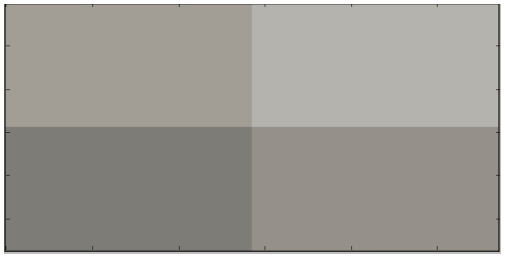




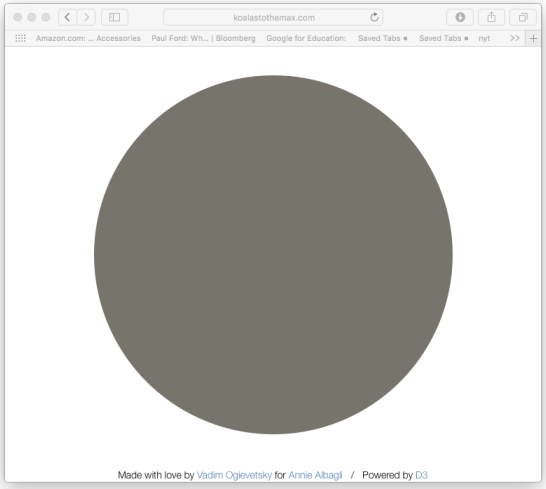


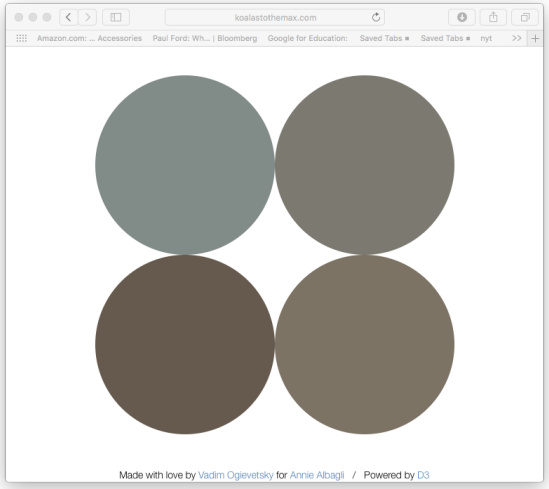
`http://koalastothemax.com`

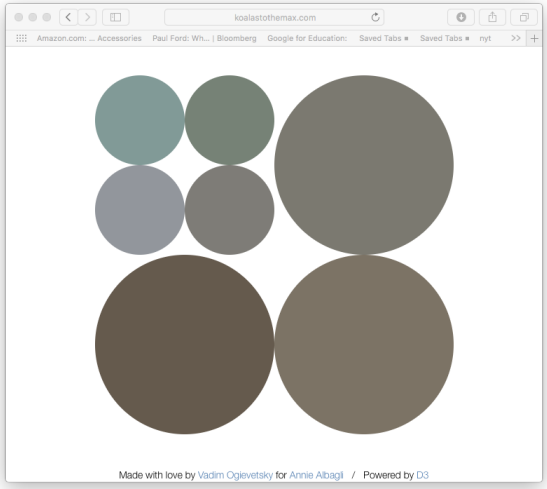




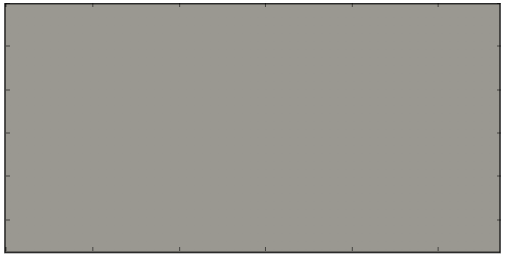
From Last Time: koalas

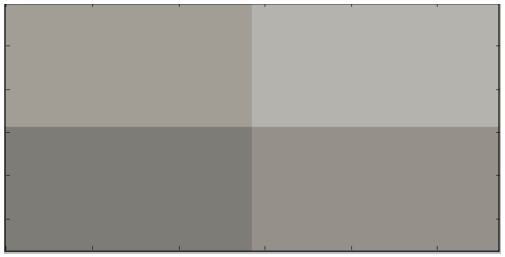


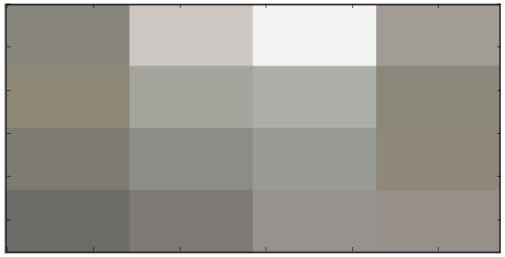




`http://koalastothemax.com`

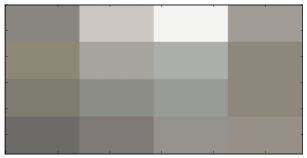






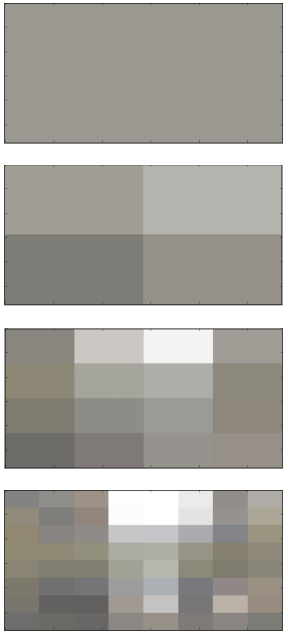
From Last Time: koalas

Process:



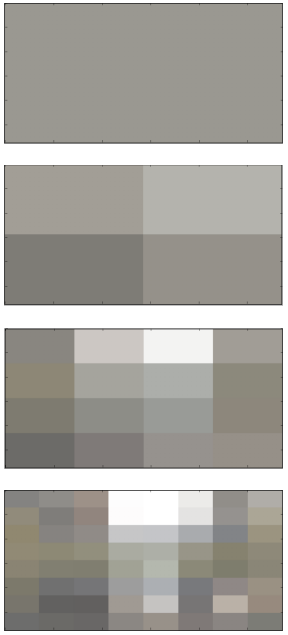
Get template from `github` → Fill in missing functions → Test locally idle3/python3 → Submit to Gradescope

From Last Time: koalas



```python
def main():
    inFile = input('Enter image file name: ')
    img = plt.imread(inFile)

    #Divides the image in 1/2, 1/4, 1/8, ... 1/2^8, and displays each:
    for i in range(8):
        img2 = img.copy()   #Make a copy to average
        quarter(img2,i)     #Split in half i times, and average regions

        plt.imshow(img2)    #Load our new image into pyplot
        plt.show()          #Show the image (waits until closed to continue)

    #Shows the original image:
    plt.imshow(img)         #Load image into pyplot
    plt.show()              #Show the image (waits until closed to continue)
```

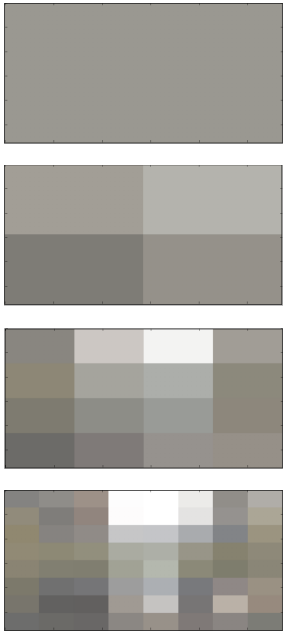
From Last Time: koalas



```python
def main():
    inFile = input('Enter image file name: ')
    img = plt.imread(inFile)

    #Divides the image in 1/2, 1/4, 1/8, ... 1/2^8, and displays each:
    for i in range(8):
        img2 = img.copy()   #Make a copy to average
        quarter(img2,i)     #Split in half i times, and average regions

        plt.imshow(img2)    #Load our new image into pyplot
        plt.show()          #Show the image (waits until closed to continue)

    #Shows the original image:
    plt.imshow(img)         #Load image into pyplot
    plt.show()              #Show the image (waits until closed to continue)
```

- The `main()` is written for you.

From Last Time: koalas



```python
def main():
    inFile = input('Enter image file name: ')
    img = plt.imread(inFile)

    #Divides the image in 1/2, 1/4, 1/8, ... 1/2^8, and displays each:
    for i in range(8):
        img2 = img.copy()   #Make a copy to average
        quarter(img2,i)     #Split in half i times, and average regions

        plt.imshow(img2)    #Load our new image into pyplot
        plt.show()          #Show the image (waits until closed to continue)

    #Shows the original image:
    plt.imshow(img)         #Load image into pyplot
    plt.show()              #Show the image (waits until closed to continue)
```

- The `main()` is written for you.
- Only fill in two functions: `average()` and `setRegion()`.

Top-Down Design

- The last example demonstrates **top-down design**: breaking into subproblems, and implementing each part separately.

Top-Down Design

- The last example demonstrates **top-down design**: breaking into subproblems, and implementing each part separately.
  - Break the problem into tasks for a "To Do" list.

Top-Down Design

- The last example demonstrates **top-down design**: breaking into subproblems, and implementing each part separately.
  - Break the problem into tasks for a "To Do" list.
  - Translate list into function names & inputs/returns.

Top-Down Design

- The last example demonstrates **top-down design**: breaking into subproblems, and implementing each part separately.
  - Break the problem into tasks for a "To Do" list.
  - Translate list into function names & inputs/returns.
  - Implement the functions, one-by-one.

Top-Down Design

- The last example demonstrates **top-down design**: breaking into subproblems, and implementing each part separately.
  - Break the problem into tasks for a "To Do" list.
  - Translate list into function names & inputs/returns.
  - Implement the functions, one-by-one.
- Excellent approach since you can then test each part separately before adding it to a large program.

Top-Down Design

- The last example demonstrates **top-down design**: breaking into subproblems, and implementing each part separately.
  - Break the problem into tasks for a "To Do" list.
  - Translate list into function names & inputs/returns.
  - Implement the functions, one-by-one.
- Excellent approach since you can then test each part separately before adding it to a large program.
- Very common when working with a team: each has their own functions to implement and maintain.

Challenge:

Write the missing functions for the program:

```
def main():
    turtle.setworldcoordinates(-100, -100, 100, 100)
    tess = setUp() #Returns a purple turtle with pen up.
    for i in range(5):
        x,y = getInput() #Asks user for two numbers.
        markLocation(tess,x,y) #Move tess to (x,y) and stamp.
    turtle.done() #need a user to click close window button to exit.
```

Challenge:

Write the missing functions for the program:

```
def main():
    turtle.setworldcoordinates(-100, -100, 100, 100)
    tess = setUp() #Returns a purple turtle with pen up.
    for i in range(5):
        x,y = getInput() #Asks user for two numbers.
        markLocation(tess,x,y) #Move tess to (x,y) and stamp.
    turtle.done() #need a user to click close window button to exit.
```

Group Work: Fill in Missing Pieces

1. Write import statements.

```
import turtle
def main():
    turtle.setworldcoordinates(-100, -100,
        100, 100)
    tess = setUp() #Returns a purple turtle
        with pen up.
    for i in range(5):
        x,y = getInput() #Asks user for two
            numbers.
        markLocation(tess,x,y) #Move tess to
            (x,y) and stamp.
    turtle.done()
```

Third Part: Fill in Missing Pieces

1. Write import statements.
2. Write down new function names and inputs.

```
def setUp():
    #FILL IN
def getInput():
    #FILL IN
def markLocation(t,x,y):
    #FILL IN

def main():
    #code omitted
```

Third Part: Fill in Missing Pieces

1. Write import statements.
2. Write down new function names and inputs.
3. Fill in return values.

```
def setUp():
    #FILL IN
    return newTurtle
def getInput():
    #FILL IN
    return x,y
def markLocation(t,x,y):
    #FILL IN
def main():
    #code omitted
```

Third Part: Fill in Missing Pieces

1. Write import statements.
2. Write down new function names and inputs.
3. Fill in return values.
4. Fill in body of functions.

```
def setUp():
    newTurtle = turtle.Turtle()
    newTurtle.penup()
    newTurtle.color("purple")
    return newTurtle
```

Third Part: Fill in Missing Pieces

1. Write import statements.
2. Write down new function names and inputs.
3. Fill in return values.
4. Fill in body of functions.

```
def getInput():
    x = int(input("Enter x: ")) #input("Enter x: ") take input after prompt "Enter x: " and returns a string, int(input("Enter x: ")) converts that string to an int
    y = int(input("Enter y: "))
    return x, y #we can return two items in python
```

Third Part: Fill in Missing Pieces

1. Write import statements.
2. Write down new function names and inputs.
3. Fill in return values.
4. Fill in body of functions.

```
def markLocation(tess, x, y):
    tess.goto(x, y)
    tess.stamp()
```

Complete Code

```
import turtle

def setUp():
    newTurtle = turtle.Turtle()
    newTurtle.penup()
    newTurtle.color("purple")
    return newTurtle
```

Complete Code: II

```
def getInput():
    x = int(input("Enter x: ")) #input("Enter x: ") take input after prompt "Enter x: " and returns a string, int(input("Enter x: ")) converts that string to an int
    y = int(input("Enter y: "))
    return x, y #we can return two items in python
```

Complete Code: III

```
def markLocation(tess, x, y):
    tess.goto(x, y)
    tess.stamp()
```

Complete Code: IV

```
def main():
    turtle.setworldcoordinates(-100, -100, 100, 100)
    tess = setUp() #Returns a purple turtle with pen up.
    for i in range(5):
        x,y = getInput() #Asks user for two numbers.
        markLocation(tess,x,y) #Move tess to (x,y) and stamp.

    turtle.done() #add so that the screen does not exit without a user click
```

Complete Code: IV

```
25 if __name__ == '__main__':
26     main()
```

Challenge:

- Write a function that takes a number as an input and prints its corresponding name.

Challenge:

- Write a function that takes a number as an input and prints its corresponding name.
- For example,

Challenge:

- Write a function that takes a number as an input and prints its corresponding name.
- For example,
  - `num2string(0)` returns: `zero`

Challenge:

- Write a function that takes a number as an input and prints its corresponding name.
- For example,
  - `num2string(0)` returns: `zero`
  - `num2string(1)` returns: `one`

Challenge:

- Write a function that takes a number as an input and prints its corresponding name.
- For example,
  - `num2string(0)` returns: `zero`
  - `num2string(1)` returns: `one`
  - `num2string(2)` returns: `two`

Challenge:

- Write a function that takes a number as an input and prints its corresponding name.
- For example,
  - `num2string(0)` returns: `zero`
  - `num2string(1)` returns: `one`
  - `num2string(2)` returns: `two`
- You may assume that only single digits, 0,1,...,9, are given as input.

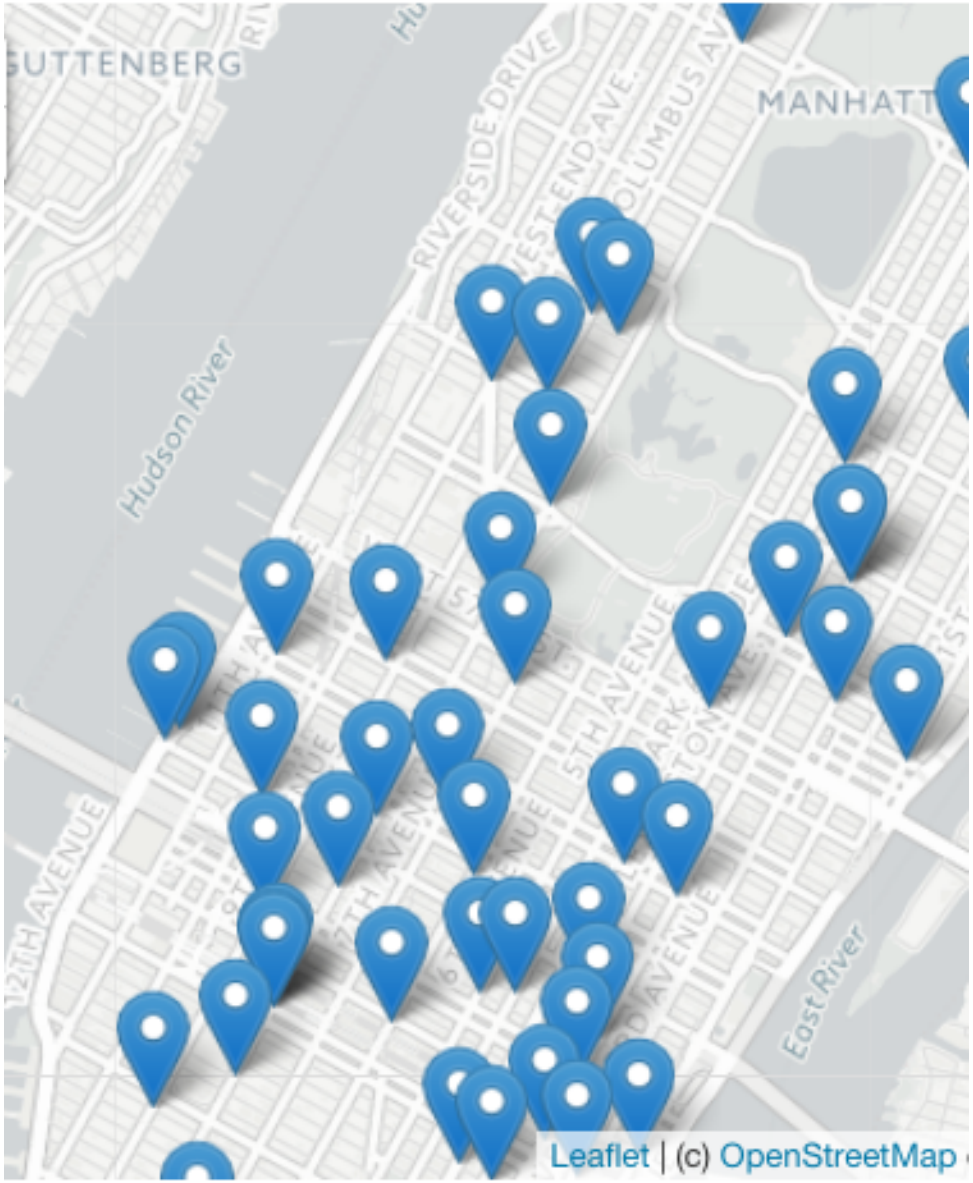
Python Tutor

(numsConvert.py on On github)

Today's Topics

- Recap: Functions & Top Down Design
- **Mapping GIS Data**
- Random Numbers
- Indefinite Loops

Folium


SUTTENBERG
MANHATT
RIVERSIDE DRIVE
WEST END AVE
COLUMBUS AV
Hudson River
5TH AVENUE
12TH AVENUE
East River
Leaflet | (c) OpenStreetMap

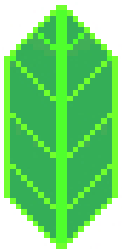
Folium

- A module for making HTML maps.

Folium

Folium

Folium



- A module for making HTML maps.
- It's a Python interface to the popular `leaflet.js`.

Folium

Folium

- A module for making HTML maps.
- It's a Python interface to the popular `leaflet.js`.
- Outputs `.html` files which you can open in a browser.

Folium

Folium

- A module for making HTML maps.
- It's a Python interface to the popular `leaflet.js`.
- Outputs `.html` files which you can open in a browser.
- An extra step:

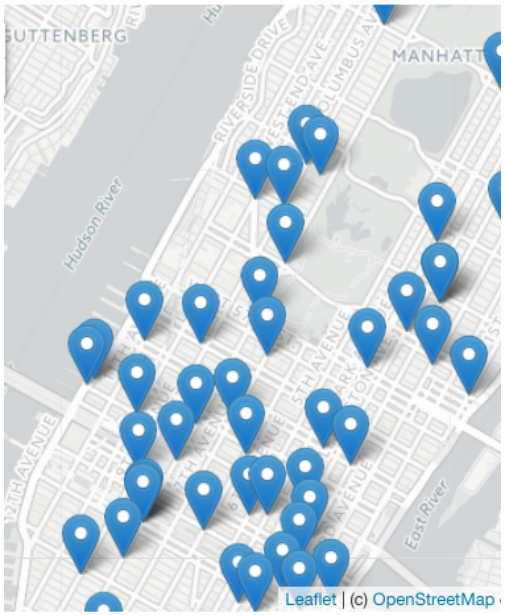
Folium

Folium

- A module for making HTML maps.
- It's a Python interface to the popular `leaflet.js`.
- Outputs `.html` files which you can open in a browser.
- An extra step:

Write code.	$\rightarrow$	*Run program.*	$\rightarrow$	*Open .html in browser.*

Demo



(Map created by Folium.)

Folium

- To use:

  ```
  import folium
  ```

Folium

Folium

- To use:

  ```
  import folium
  ```

- Create a map:

  ```
  myMap = folium.Map()
  ```

Folium

- To use:

  ```
  import folium
  ```

- Create a map:

  ```
  myMap = folium.Map()
  ```

- Make markers:

  ```
  newMark = folium.Marker([lat,lon],popup=name)
  ```

Folium

- To use:

  ```
  import folium
  ```

- Create a map:

  ```
  myMap = folium.Map()
  ```

- Make markers:

  ```
  newMark = folium.Marker([lat,lon],popup=name)
  ```

- Add to the map:

  ```
  newMark.add_to(myMap)
  ```

Folium

- To use:

  ```
  import folium
  ```

- Create a map:

  ```
  myMap = folium.Map()
  ```

- Make markers:

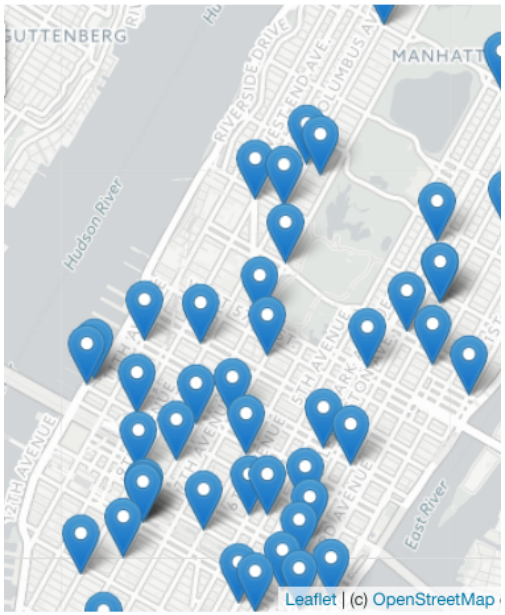
  ```
  newMark = folium.Marker([lat,lon],popup=name)
  ```

- Add to the map:

  ```
  newMark.add_to(myMap)
  ```

- Many options to customize background map ("tiles") and markers.

Demo



(Python program using Folium.)

In Pairs of Triples

```
import folium
import webbrowser
import os #use to find directory

m = folium.Map(location = [45.372,
   -121.6972],
               zoom_start = 12,
               tiles = 'Stamen Terrain')
#stamen Terrain highlights hill shading and
#natural vegetation colors
```

In Pairs of Triples

```
folium.Marker(
        location = [45.3288, -121.6625],
        popup = 'Mt. Hood Meadows',
        icon = folium.Icon(icon = 'cloud')
).add_to(m)
```

In Pairs of Triples

```
folium.Marker(
        location = [45.3311, -121.7113],
        popup = 'Timberline Lodge',
        icon = folium.Icon(color = 'green')
).add_to(m)
```

In Pairs of Triples

```
folium.Marker(
        location = [45.3300, -121.6823],
        popup = 'some other location',
        icon = folium.Icon(color = 'red',
        icon = 'info_sign')
).add_to(m)
```

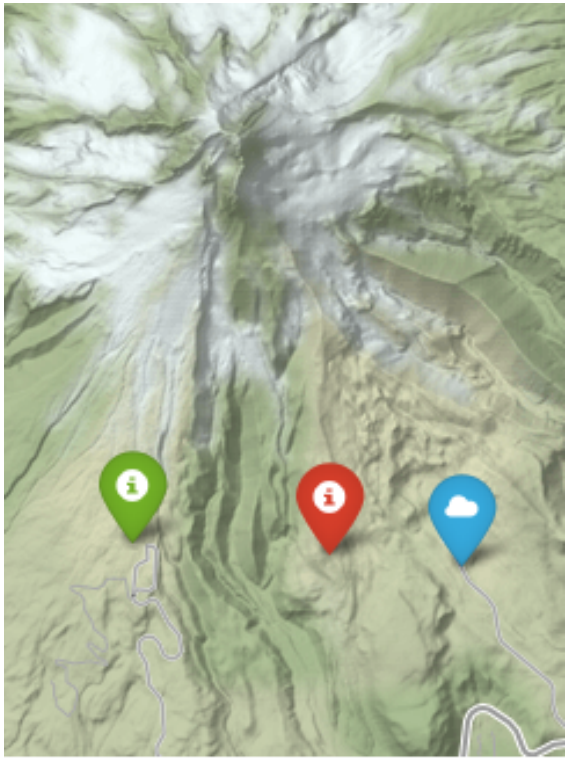
In Pairs of Triples

```
#google "display html page python"
#https://stackoverflow.com/questions/40905703/how-to-open-an-html-file-in-the-browser-from-python
filename = 'three_marks.html'
m.save(outfile = filename)
webbrowser.open('file://' + os.path.realpath(filename))
```

In Pairs of Triples

- Predict which each line of code does:

```
m = folium.Map(
    location=[45.372, -121.6972],
    zoom_start=12,
    tiles='Stamen Terrain'
)

folium.Marker(
    location=[45.3288, -121.6625],
    popup='Mt. Hood Meadows',
    icon=folium.Icon(icon='cloud')
).add_to(m)

folium.Marker(
    location=[45.3311, -121.7113],
    popup='Timberline Lodge',
    icon=folium.Icon(color='green')
).add_to(m)

folium.Marker(
    location=[45.3300, -121.6823],
    popup='Some Other Location',
    icon=folium.Icon(color='red', icon='info-sign')
).add_to(m)
```



(example from Folium documentation)

Today's Topics

- Recap: Functions & Top Down Design
- Mapping GIS Data
- **Random Numbers**
- Indefinite Loops

Python's random package

- Python has a built-in package for generating pseudo-random numbers.

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
  trey.forward(10)
  a = random.randrange(0,360,90)
  trey.right(a)
```

Python's random package

- Python has a built-in package for generating pseudo-random numbers.
- To use:

```
import random
```

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
  trey.forward(10)
  a = random.randrange(0,360,90)
  trey.right(a)
```

Python's random package

- Python has a built-in package for generating pseudo-random numbers.
- To use:

  `import random`

- Useful command to generate whole numbers:

  `random.randrange(start,stop,step)`

  which gives a number chosen randomly from the specified range.

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
  trey.forward(10)
  a = random.randrange(0,360,90)
  trey.right(a)
```

Python's random package

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
  trey.forward(10)
  a = random.randrange(0,360,90)
  trey.right(a)
```

- Python has a built-in package for generating pseudo-random numbers.
- To use:

  `import random`

- Useful command to generate whole numbers:

  `random.randrange(start,stop,step)`

  which gives a number chosen randomly from the specified range.
- Useful command to generate real numbers:

Python's random package

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
  trey.forward(10)
  a = random.randrange(0,360,90)
  trey.right(a)
```

- Python has a built-in package for generating pseudo-random numbers.
- To use:

  `import random`

- Useful command to generate whole numbers:

  `random.randrange(start,stop,step)`

  which gives a number chosen randomly from the specified range.
- Useful command to generate real numbers:

  `random.random()`

  which gives a number chosen (uniformly) at random from [0.0,1.0).

Python's random package

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
  trey.forward(10)
  a = random.randrange(0,360,90)
  trey.right(a)
```

- Python has a built-in package for generating pseudo-random numbers.
- To use:

  `import random`

- Useful command to generate whole numbers:

  `random.randrange(start,stop,step)`

  which gives a number chosen randomly from the specified range.
- Useful command to generate real numbers:

  `random.random()`

  which gives a number chosen (uniformly) at random from [0.0,1.0).
- Very useful for simulations, games, and testing.

Trinket

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)
for i in range(100):
    trey.forward(10)
    a = random.randrange(0, 360, 90) # generate a random int in [0, 90, 180, 270]
    trey.right(a)
turtle.done() #wait user to click x (window close button) in the top left of window.
```

Today's Topics

- Recap: Functions & Top Down Design
- Mapping GIS Data
- Random Numbers
- **Indefinite Loops**

Challenge:

Predict what the code will do:

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```

Challenge:

Predict what the code will do:

```
nums = [1, 4, 0, 6, 5, 2, 9, 8, 12]
print(nums)
i = 0
while i < len(nums)-1:
    if nums[i] < nums[i+1]:
        nums[i], nums[i+1] = nums[i+1], nums[i]
    i = i+1
print(nums)
```

Indefinite Loops

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```

```
#Spring 2012 Final Exam, #8

nums = [1,4,0,6,5,2,9,8,12]
print(nums)
i=0
while i < len(nums)-1:
    if nums[i] < nums[i+1]:
        nums[i], nums[i+1] = nums[i+1], nums[i]
    i=i+1

print(nums)
```

- Indefinite loops repeat as long as the condition is true.

Indefinite Loops

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```

```
#Spring 2012 Final Exam, #8

nums = [1,4,0,6,5,2,9,8,12]
print(nums)
i=0
while i < len(nums)-1:
    if nums[i] < nums[i+1]:
        nums[i], nums[i+1] = nums[i+1], nums[i]
    i=i+1

print(nums)
```

- Indefinite loops repeat as long as the condition is true.
- Could execute the body of the loop zero times, 10 times, infinite number of times.

Indefinite Loops

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```

```
#Spring 2012 Final Exam, #8

nums = [1,4,0,6,5,2,9,8,12]
print(nums)
i=0
while i < len(nums)-1:
    if nums[i] < nums[i+1]:
        nums[i], nums[i+1] = nums[i+1], nums[i]
    i=i+1

print(nums)
```

- Indefinite loops repeat as long as the condition is true.
- Could execute the body of the loop zero times, 10 times, infinite number of times.
- The condition determines how many times.

Indefinite Loops

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```

```
#Spring 2012 Final Exam, #8

nums = [1,4,0,6,5,2,9,8,12]
print(nums)
i=0
while i < len(nums)-1:
    if nums[i] < nums[i+1]:
        nums[i], nums[i+1] = nums[i+1], nums[i]
    i=i+1

print(nums)
```

- Indefinite loops repeat as long as the condition is true.
- Could execute the body of the loop zero times, 10 times, infinite number of times.
- The condition determines how many times.
- Very useful for checking input, simulations, and games.

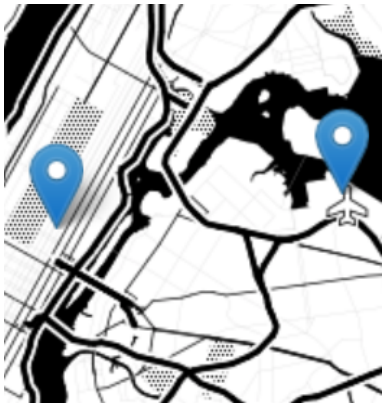
Indefinite Loops

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```

```
#Spring 2012 Final Exam, #8

nums = [1,4,0,6,5,2,9,8,12]
print(nums)
i=0
while i < len(nums)-1:
    if nums[i] < nums[i+1]:
        nums[i], nums[i+1] = nums[i+1], nums[i]
    i=i+1

print(nums)
```

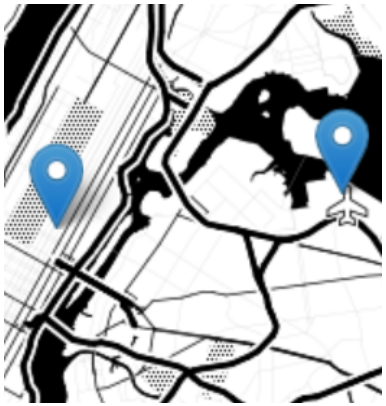
- Indefinite loops repeat as long as the condition is true.
- Could execute the body of the loop zero times, 10 times, infinite number of times.
- The condition determines how many times.
- Very useful for checking input, simulations, and games.
- More details next lecture...

Recap



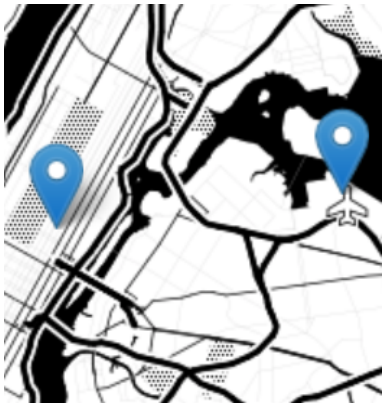
- Top-down design: breaking into subproblems, and implementing each part separately.

Recap



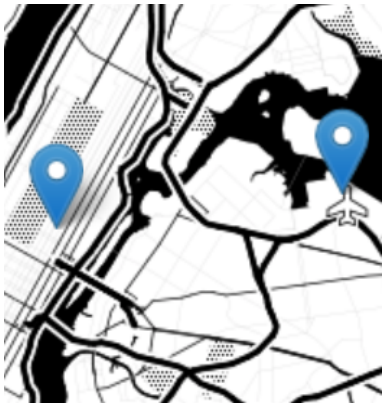
- Top-down design: breaking into subproblems, and implementing each part separately.
- Excellent approach: can then test each part separately before adding it to a large program.

Recap



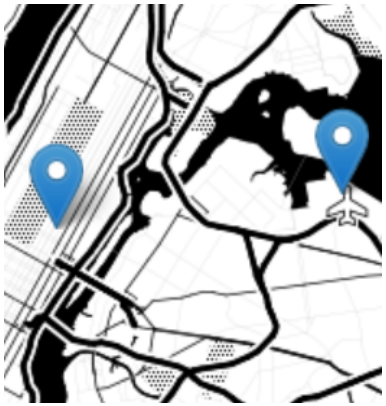
- Top-down design: breaking into subproblems, and implementing each part separately.
- Excellent approach: can then test each part separately before adding it to a large program.
- When possible, design so that your code is flexible to be reused ("code reuse").

Recap



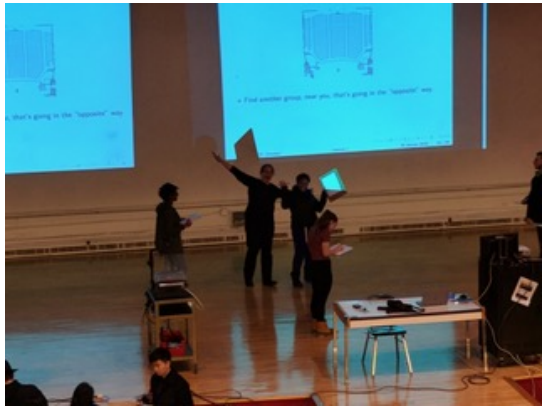
- Top-down design: breaking into subproblems, and implementing each part separately.
- Excellent approach: can then test each part separately before adding it to a large program.
- When possible, design so that your code is flexible to be reused ("code reuse").
- Introduced a Python library, `Folium` for creating interactive HTML maps.

Recap



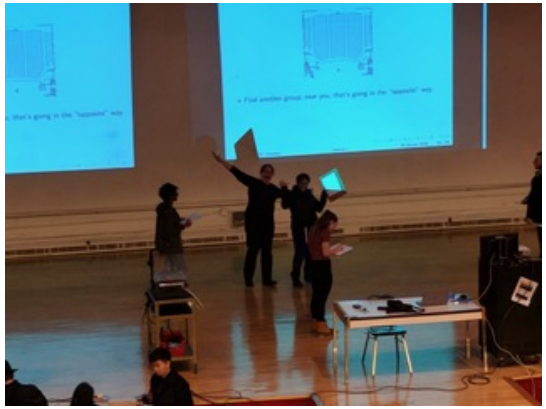
- Top-down design: breaking into subproblems, and implementing each part separately.
- Excellent approach: can then test each part separately before adding it to a large program.
- When possible, design so that your code is flexible to be reused ("code reuse").
- Introduced a Python library, `Folium` for creating interactive HTML maps.
- Introduced `while` loops for repeating commands for an indefinite number of times.

Practice Quiz & Final Questions



- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and
  - repeat.
- Past exams are on the webpage (under Final Exam Information).

Practice Quiz & Final Questions



- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and
  - repeat.
- Past exams are on the webpage (under Final Exam Information).
- Theme: Functions & Top-Down Design (Summer 18, #7).

Weekly Reminders!

Before next lecture, don't forget to:

- Work on this week's Online Lab

Weekly Reminders!

Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North

Weekly Reminders!

Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North

Weekly Reminders!

Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments **(programs 41-45)**

Weekly Reminders!

Before next lecture, don't forget to:

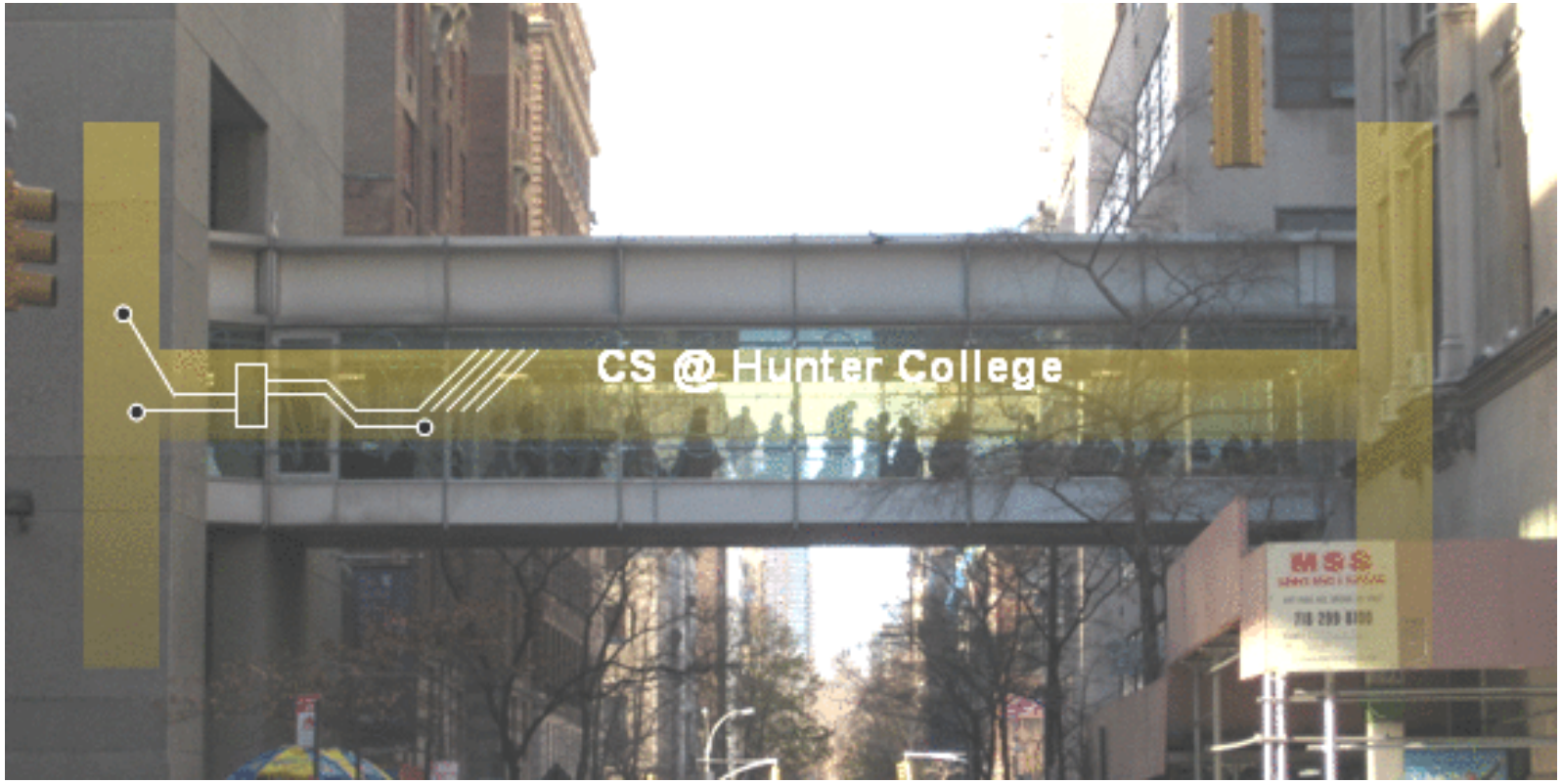
- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments **(programs 41-45)**
- If you need help, schedule an appointment for Tutoring in lab 1001G 11:30am-5pm

Weekly Reminders!

Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments **(programs 41-45)**
- If you need help, schedule an appointment for Tutoring in lab 1001G 11:30am-5pm
- Take the Lecture Preview on Blackboard on Monday (or no later than 10:15am on Tuesday)

Lecture Slips & Writing Boards



- Hand your lecture slip to a UTA.
- Return writing boards as you leave.