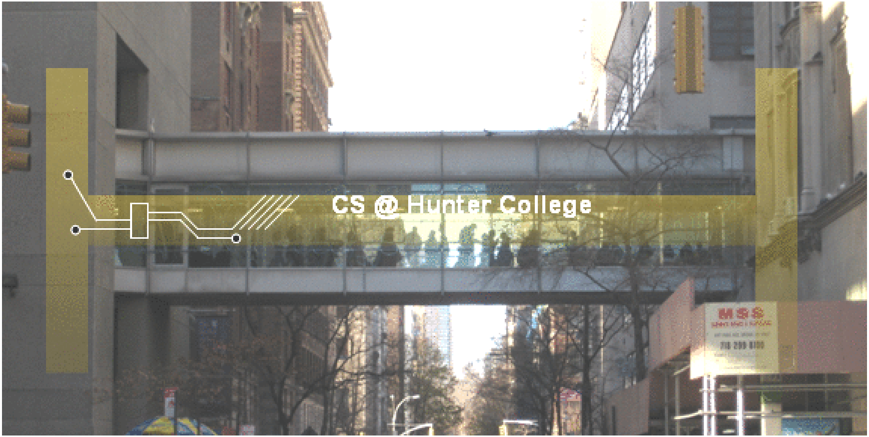
CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Announcements

- Welcome back!

Announcements

- Welcome back!
- Classes on Wednesday, 11 April 2018 follows Friday schedule.

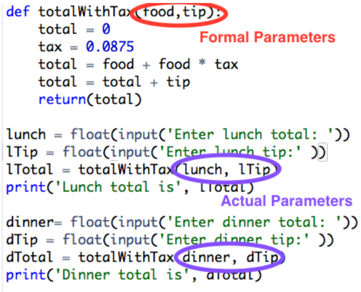
Announcements

- Welcome back!
- Classes on Wednesday, 11 April 2018 follows Friday schedule.
- End of lecture: quiz/final exam review.

Today's Topics

- Recap: Parameters & Functions
- Top-down Design
- Mapping GIS Data
- Code Reuse
- Final Exam Overview

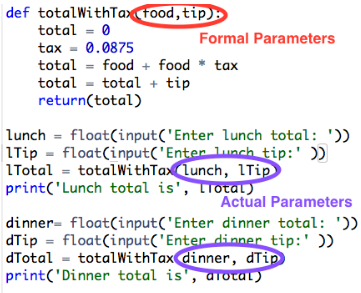
Recap: Input Parameters & Return Values



```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip:' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip:' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- When called, the actual parameter values are copied to the formal parameters.

Recap: Input Parameters & Return Values



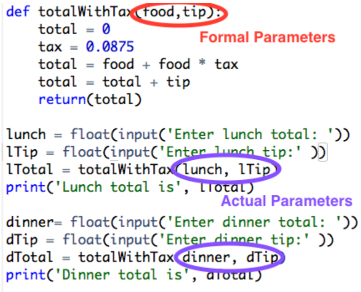
```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip:' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip:' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

Formal Parameters

Actual Parameters

- When called, the actual parameter values are copied to the formal parameters.
- All the commands inside the function are performed on the copies.

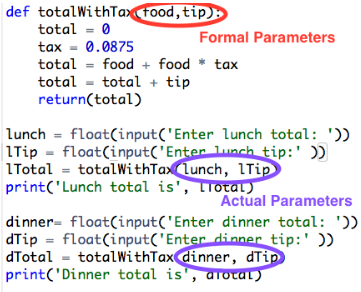
Recap: Input Parameters & Return Values



```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip:' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip:' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- When called, the actual parameter values are copied to the formal parameters.
- All the commands inside the function are performed on the copies.
- The actual parameters do not change.

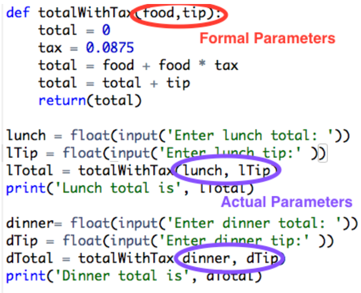
Recap: Input Parameters & Return Values



```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip:' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip:' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- When called, the actual parameter values are copied to the formal parameters.
- All the commands inside the function are performed on the copies.
- The actual parameters do not change.
- The copies are discarded when the function is done.

Recap: Input Parameters & Return Values

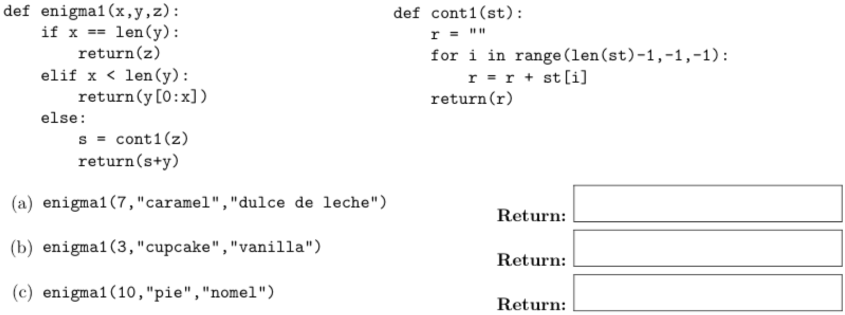


```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip:' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip:' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- When called, the actual parameter values are copied to the formal parameters.
- All the commands inside the function are performed on the copies.
- The actual parameters do not change.
- The copies are discarded when the function is done.
- The time a variable exists is called its **scope**.

In Pairs or Triples:

- What are the formal parameters? What is returned?



```
def enigma1(x,y,z):
    if x == len(y):
        return(z)
    elif x < len(y):
        return(y[0:x])
    else:
        s = cont1(z)
        return(s+y)
```

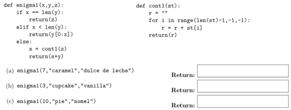
```
def cont1(st):
    r = ""
    for i in range(len(st)-1,-1,-1):
        r = r + st[i]
    return(r)
```

(a) `enigma1(7,"caramel","dulce de leche")` **Return:** □

(b) `enigma1(3,"cupcake","vanilla")` **Return:** □

(c) `enigma1(10,"pie","nomel")` **Return:** □

Python Tutor



```
def enigma1(x,y,z):
    if x == len(y):
        return(z)
    elif x < len(y):
        return(y[0:x])
    else:
        s = cont1(z)
        return(s+y)
```

```
def cont1(st):
    r = ""
    for i in range(len(st)-1,-1,-1):
        r = r + st[i]
    return(r)
```

(a) `enigma1(7,"caramel","dulce de leche")` **Return:** □

(b) `enigma1(3,"cupcake","vanilla")` **Return:** □

(c) `enigma1(10,"pie","nomel")` **Return:** □

(Demo with `pythonTutor`)

In Pairs or Triples:

- Write the missing functions for the program:

```
def main():
    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.
```

Group Work: Fill in Missing Pieces

```
def main():
    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.
```

Group Work: Fill in Missing Pieces

1. Write import statements.

```
import turtle
```

```
def main():
    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.
```

Third Part: Fill in Missing Pieces

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

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




def main():
    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.
```

Third Part: Fill in Missing Pieces

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

```
import turtle
def setUp():
    #FILL IN
    return(newTurtle)
def getInput():
    #FILL IN
    return(x,y)
def markLocation(t,x,y):
    #FILL IN



def main():
    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.
```

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.

```
import turtle
def setUp():
    newTurtle = turtle.Turtle()
    newTurtle.penup()
    return(newTurtle)
def getInput():
    x = int(input('Enter x: '))
    y = int(input('Enter y: '))
    return(x,y)
def markLocation(t,x,y):
    t.goto(x,y)
    t.stamp()
def main():
    tess = setUp()    #Returns a purple turtle with pen up.
    for i in range(5):
        x,y = getInput()       #Asks user for two numbers.
```

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.

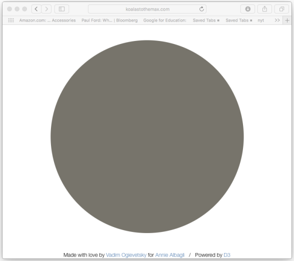
Top-Down Design

- 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.

In Pairs or Triples:

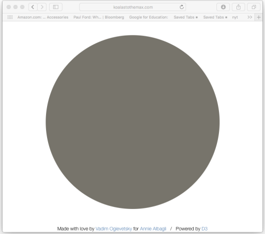


Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

`http://koalastothemax.com`

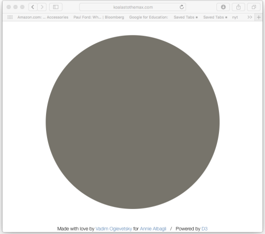
- Top-down design puzzle:
  - What does `koalastomax` do?
  - What does each circle represent?
- Write a high-level design for it.
- Translate into a `main()` with function calls.

Demo

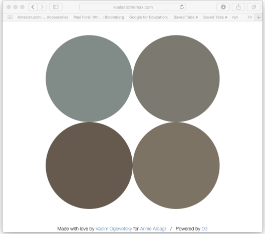


Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

Demo

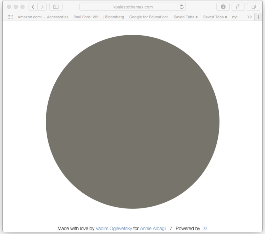


Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

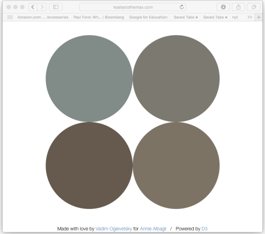


Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

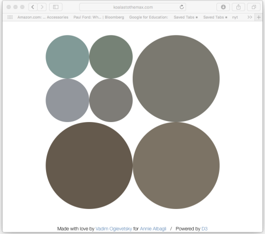
Demo



Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

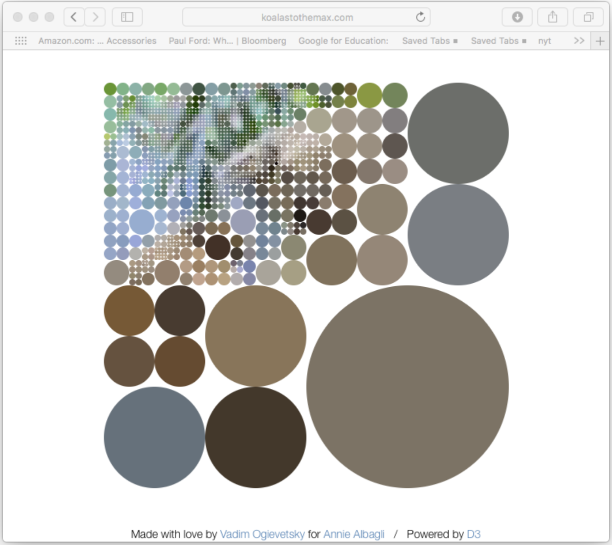


Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3



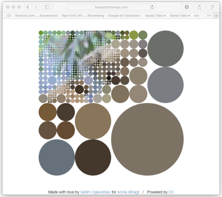
Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

Demo



Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

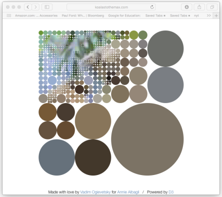
Design: Koalas to the Max



Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

- **Input:** Image & mouse movements

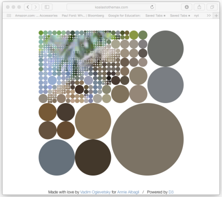
Design: Koalas to the Max



Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

- **Input:** Image & mouse movements
- **Output:** Completed image

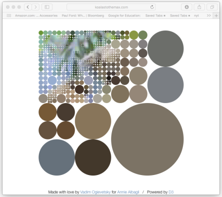
Design: Koalas to the Max



Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**

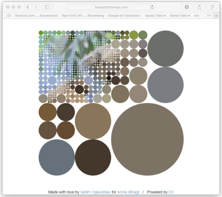
Design: Koalas to the Max



Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

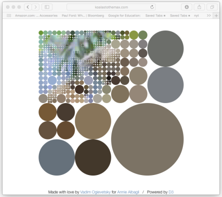
- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**
  - Every mouse movement,

Design: Koalas to the Max


koalastothemax.com
Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**
  - Every mouse movement,
  - Divide the region into 4 quarters.

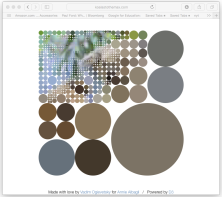
Design: Koalas to the Max



Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**
  - Every mouse movement,
  - Divide the region into 4 quarters.
  - Average the color of each region.

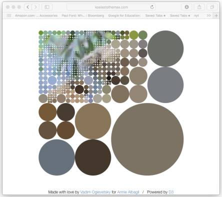
Design: Koalas to the Max



Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**
  - Every mouse movement,
  - Divide the region into 4 quarters.
  - Average the color of each region.
  - Set each region to its average.

Design: Koalas to the Max

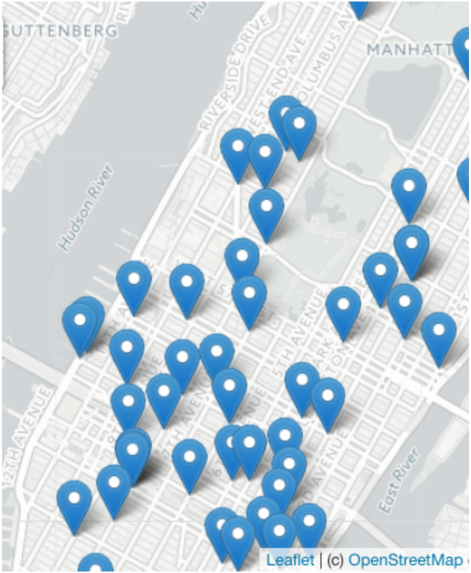


Made with love by Vadim Ogievetsky for Annie Albagli / Powered by D3

- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**
  - Every mouse movement,
  - Divide the region into 4 quarters.
  - Average the color of each region.
  - Set each region to its average.

(Demo program from github.)

Folium


GUTTENBERG
MANHATTAN
RIVERSIDE DRIVE
WEST END AVE.
COLUMBUS AVE
Hudson River
East River
12TH AVENUE
5TH AVENUE
PARK
Leaflet | (c) OpenStreetMap

Folium

- A module for making HTML maps.

Folium

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

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

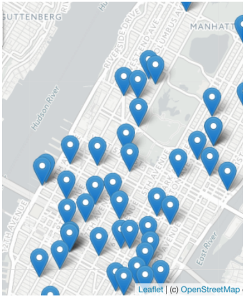
- A module for making HTML maps.
- It's a Python interface to the popular `leaflet.js`.
- Outputs `.html` files which you can open in a browser.
- An extra step:

Folium

- 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. → *Run program.* → *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:

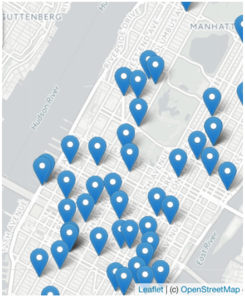
  `newMark = folium.Marker([lat,lon],popup=name)`

- Add to the map:

  `newMark.add_to(myMap)`

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

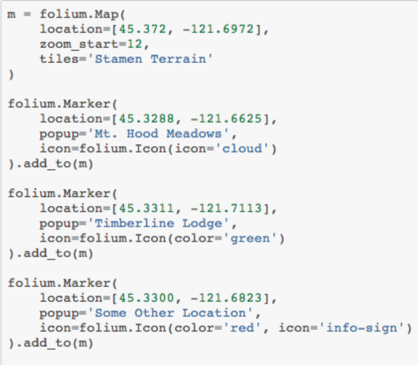
Demo



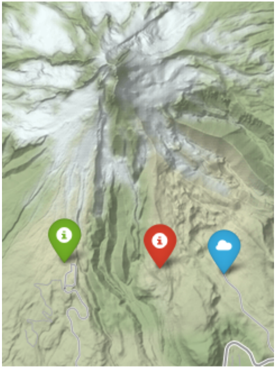
(Python program using Folium.)

In Pairs of Triples

- Predict which each line of code does:

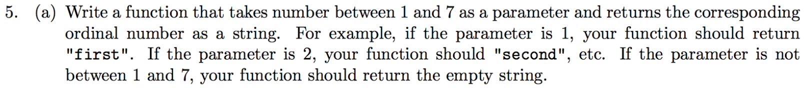


```
m = folium.Map(
    location=[45.372, -121.6972],
    zoom_start=12,
    tiles='Stamen Terrain'
)

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)
```



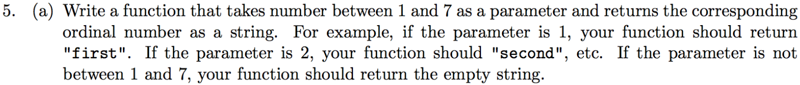
(example from Folium documentation)

In Pairs or Triples:



5. (a) Write a function that takes number between 1 and 7 as a parameter and returns the corresponding ordinal number as a string. For example, if the parameter is 1, your function should return `"first"`. If the parameter is 2, your function should `"second"`, etc. If the parameter is not between 1 and 7, your function should return the empty string.

In Pairs or Triples:



5. (a) Write a function that takes number between 1 and 7 as a parameter and returns the corresponding ordinal number as a string. For example, if the parameter is 1, your function should return `"first"`. If the parameter is 2, your function should `"second"`, etc. If the parameter is not between 1 and 7, your function should return the empty string.

(Python Tutor)

Code Reuse

+
−
Hudson River
EEHAWKEN
Hell Gate
East River
BROADWAY

- Goal: design your code to be reused.

Code Reuse

+
−
Hudson River
EHAWKEN
Hell Gate
East River
BROADWAY

- Goal: design your code to be reused.
- Example: code to make maps of CUNY locations from CSV files.

Code Reuse

+
−
N CITY
EEHAWKEN
Hudson River
Hell Gate
East River
BROADWAY

- Goal: design your code to be reused.
- Example: code to make maps of CUNY locations from CSV files.
  - Same idea can be used for mapping traffic collisions data.

Code Reuse

N CITY
+
−
EEHAWKEN
Hudson River
Hell Gate
East River
BROADWAY

- Goal: design your code to be reused.
- Example: code to make maps of CUNY locations from CSV files.
    - Same idea can be used for mapping traffic collisions data.
    - Or recycling bins, or wifi locations, or 311 calls,...

Code Reuse

+
−
EEHAWKEN
Hudson River
Hell Gate
East River
BROADWAY

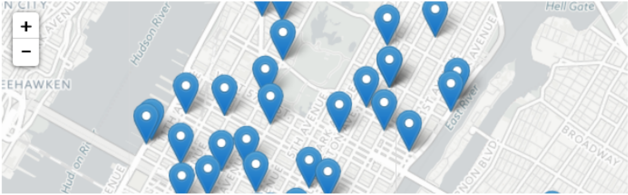
- Goal: design your code to be reused.
- Example: code to make maps of CUNY locations from CSV files.
  - Same idea can be used for mapping traffic collisions data.
  - Or recycling bins, or wifi locations, or 311 calls,...
  - Small wrinkle: some call the columns “Latitude”, while others use “LATITUDE”, “latitude”, or “lat”.

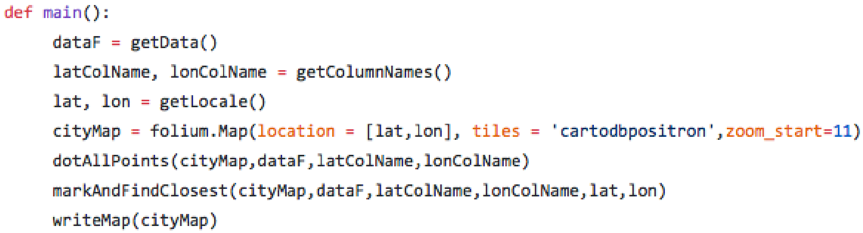
Code Reuse

+
−
Hudson River
EEHAWKEN
Hell Gate
East River
BROADWAY

- Goal: design your code to be reused.
- Example: code to make maps of CUNY locations from CSV files.
  - Same idea can be used for mapping traffic collisions data.
  - Or recycling bins, or wifi locations, or 311 calls,...
  - Small wrinkle: some call the columns “Latitude”, while others use “LATITUDE”, “latitude”, or “lat”.
  - Solution: ask user for column names and pass as parameters.

Code Reuse

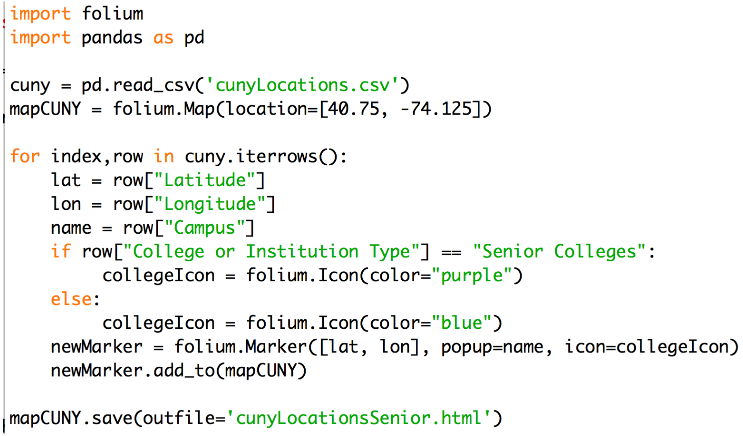




```
def main():
      dataF = getData()
      latColName, lonColName = getColumnNames()
      lat, lon = getLocale()
      cityMap = folium.Map(location = [lat,lon], tiles = 'cartodbpositron',zoom_start=11)
      dotAllPoints(cityMap,dataF,latColName,lonColName)
      markAndFindClosest(cityMap,dataF,latColName,lonColName,lat,lon)
      writeMap(cityMap)
```

In Pairs or Triples:

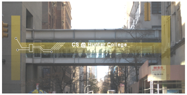
What does this code do?



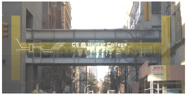
```
import folium
import pandas as pd

cuny = pd.read_csv('cunyLocations.csv')
mapCUNY = folium.Map(location=[40.75, -74.125])

for index,row in cuny.iterrows():
    lat = row["Latitude"]
    lon = row["Longitude"]
    name = row["Campus"]
    if row["College or Institution Type"] == "Senior Colleges":
         collegeIcon = folium.Icon(color="purple")
    else:
         collegeIcon = folium.Icon(color="blue")
    newMarker = folium.Marker([lat, lon], popup=name, icon=collegeIcon)
    newMarker.add_to(mapCUNY)

mapCUNY.save(outfile='cunyLocationsSenior.html')
```

Recap: Top-down Design & `Folium`

- On lecture slip, write down a topic you wish we had spent more time (and why).

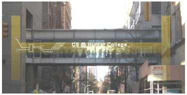


Recap: Top-down Design & `Folium`



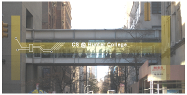
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Top-down design: breaking into subproblems, and implementing each part separately.

Recap: Top-down Design & `Folium`



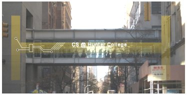
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Top-down design: breaking into subproblems, and implementing each part separately.
- Excellent approach: can then test each part separately before adding it to a large program.

Recap: Top-down Design & `Folium`



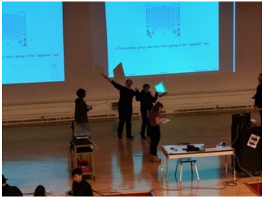
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Top-down design: breaking into subproblems, and implementing each part separately.
- Excellent approach: can then test each part separately before adding it to a large program.
- When possible, design so that your code is flexible to be reused ("code reuse").

Recap: Top-down Design & Folium



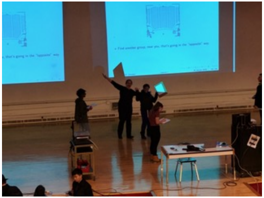
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Top-down design: breaking into subproblems, and implementing each part separately.
- Excellent approach: can then test each part separately before adding it to a large program.
- When possible, design so that your code is flexible to be reused ("code reuse").
- Introduced a Python library, `Folium` for creating interactive HTML maps.

Practice Quiz & Final Questions



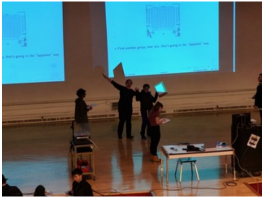
- Lightning rounds:

Practice Quiz & Final Questions



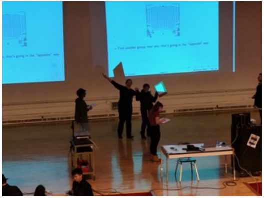
- Lightning rounds:
  - write as much you can for 60 seconds;

Practice Quiz & Final Questions



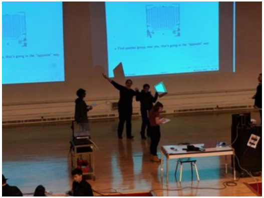
- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and

Practice Quiz & Final Questions



- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and
  - repeat.

Practice Quiz & Final Questions



- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and
  - repeat.
- Continue from last time on the mock exam (on web page).