# CSci 127: Introduction to Computer Science
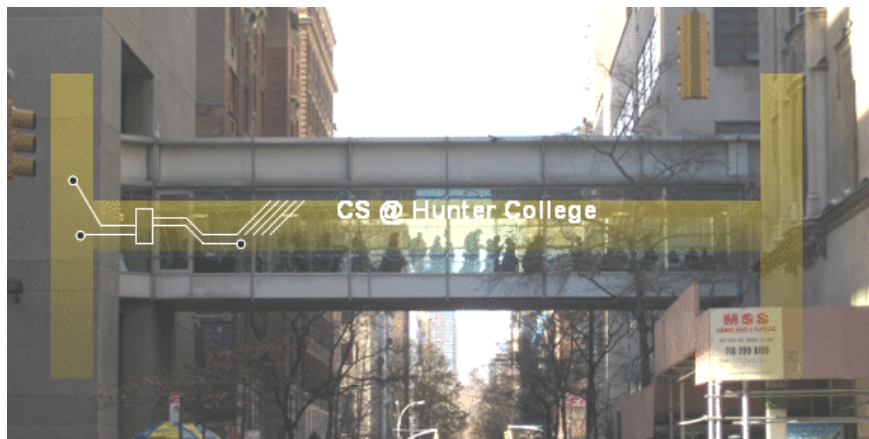
# Frequently Asked Questions

- **Do I have to take the final?**
  *Yes, you must to pass the final (60 out of 100 points) to the pass the class.*
  *Please review the grading policy on the course syllabus:*
  *https://huntercsci127.github.io/f22/syl.html*

# Frequently Asked Questions

From email

- ○ **Do I have to take the final?**
  *Yes, you must to pass the final (60 out of 100 points) to the pass the class. Please review the grading policy on the course syllabus: https://huntercsci127.github.io/f22/syl.html*

- ○ **Can I take the course Pass/No Credit?**
  *Yes, but check with your advisor that it is possible with your major and standing. Learn more about it here: https://hunter.cuny.edu/students/registration/register-for-classes/credit-no-credit/*

# Today's Topics



- More on Functions
- Recap: Open Data
- Top Down Design
- Design Challenge

# Today's Topics

- **More on Functions**
- Recap: Open Data
- Top Down Design
- Design Challenge

# Recap: Input Parameters & Return Values

- Functions can have **input parameters**.

```python
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)
```

# Recap: Input Parameters & Return Values

```python
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)
```

- Functions can have **input parameters**.
- Surrounded by parentheses, both in the function definition, and in the function call (invocation).

# Recap: Input Parameters & Return Values

```python
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)
```

- Functions can have **input parameters**.

- Surrounded by parentheses, both in the function definition, and in the function call (invocation).

- The "placeholders" in the function definition: **formal parameters**.

# Recap: Input Parameters & Return Values

```python
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)
```

- Functions can have **input parameters**.
- Surrounded by parentheses, both in the function definition, and in the function call (invocation).
- The "placeholders" in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**

# Recap: Input Parameters & Return Values

```python
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)
```

- Functions can have **input parameters**.
- Surrounded by parentheses, both in the function definition, and in the function call (invocation).
- The "placeholders" in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**
- Functions can also **return values** to where it was called.

# Recap: Input Parameters & Return Values

```python
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**

- Functions can have **input parameters**.

- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).

- The "placeholders" in the function definition: **formal parameters**.

- The ones in the function call: **actual parameters**.

- Functions can also **return values** to where it was called.

# Challenge:

- What are the formal parameters? What is returned?

```python
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)
```

```python
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: [                    ]

dessert.py

```
1  def enigma1(x,y,z):
2      if x == len(y):
3          return(z)
4      elif x < len(y):
5          return(y[0:x])
6      else:
7          s = cont1(z)
8          return(s+y)
```

dessert.py: II

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

enigma1(7,"caramel","dulce de leche")
enigma1(3,"cupcake","vanilla")
enigma1(10,"pie","nomel")
```

# 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)

(a) enigma1(7,"caramel","dulce de leche")

(b) enigma1(3,"cupcake","vanilla")

(c) enigma1(10,"pie","nutmel")
```

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

Return: [          ]

Return: [          ]

Return: [          ]

(Demo with pythonTutor)

# Input Parameters

```python
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.

# Input Parameters

```python
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.

# Input Parameters

```python
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.
- The actual parameters do not change.

# Input Parameters

```python
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.
- The actual parameters do not change.
- The copies are discarded when the function is done.

# Input Parameters

```python
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.
- The actual parameters do not change.
- The copies are discarded when the function is done.
- The time a variable exists is called its **scope**.

# Input Parameters: What about Lists?

```
#Fall 2013 Final Exam, 5

def kuwae( inLst ):
    tot = 1
    for item in inLst:
        tot = tot * item
    return tot

def foo( inLst ):
    if ( inLst[-1] > inLst[0] ):
        return kuwae( inLst )
    else:
        return -1

foo( [2, 4, 6, 8] )

foo( [4002, 328, 457, 1] )
```

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

# Input Parameters: What about Lists?

```
#Fall 2013 Final Exam, 5

def kuwae( inLst ):
    tot = 1
    for item in inLst:
        tot = tot * item
    return tot

def foo( inLst ):
    if ( inLst[-1] > inLst[0] ):
        return kuwae( inLst )
    else:
        return -1

foo( [2, 4, 6, 8] )

foo( [4002, 328, 457, 1] )
```

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

- What is copied with a list?

# Input Parameters: What about Lists?

```
#Fall 2013 Final Exam, 5

def kuwae( inLst ):
    tot = 1
    for item in inLst:
        tot = tot * item
    return tot

def foo( inLst ):
    if ( inLst[-1] > inLst[0] ):
        return kuwae( inLst )
    else:
        return -1

foo( [2, 4, 6, 8] )

foo( [4002, 328, 457, 1] )
```

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

- What is copied with a list?

- The address of the list, but not the individual elements.

# Input Parameters: What about Lists?

```
#Fall 2013 Final Exam, 5

def kuwae( inLst ):
    tot = 1
    for item in inLst:
        tot = tot * item
    return tot

def foo( inLst ):
    if ( inLst[-1] > inLst[0] ):
        return kuwae( inLst )
    else:
        return -1

foo( [2, 4, 6, 8] )

foo( [4002, 328, 457, 1] )
```

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

- What is copied with a list?

- The address of the list, but not the individual elements.

- The actual parameters do not change, but the inside elements might.

# Input Parameters: What about Lists?

```
#Fall 2013 Final Exam, 5

def kuwae( inLst ):
    tot = 1
    for item in inLst:
        tot = tot * item
    return tot

def foo( inLst ):
    if ( inLst[-1] > inLst[0] ):
        return kuwae( inLst )
    else:
        return -1

foo( [2, 4, 6, 8] )

foo( [4002, 328, 457, 1] )
```

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

- What is copied with a list?

- The address of the list, but not the individual elements.

- The actual parameters do not change, but the inside elements might.

- Easier to see with a demo.

# Python Tutor

```
#Fall 2013 Final Exam, 5

def kuwae( inLst ):
    tot = 1
    for item in inLst:
        tot = tot * item
    return tot

def foo( inLst ):
    if ( inLst[-1] > inLst[0] ):
        return kuwae( inLst )
    else:
        return -1

foo( [2, 4, 6, 8] )

foo( [4002, 328, 457, 1] )
```

(Demo with `pythonTutor`)

# Challenge:

*Predict what the code will do:*

```python
#CSci 127 Teaching Staff
#Triangles two ways...
import turtle

def setUp(t, dist, col):
    t.penup()
    t.forward(dist)
    t.pendown()
    t.color(col)


def nestedTriangle(t, side):
    if side > 10:
        for i in range(3):
            t.forward(side)
            t.left(120)
        nestedTriangle(t, side/2)


def fractalTriangle(t, side):
    if side > 10:
        for i in range(3):
            t.forward(side)
            t.left(120)
            fractalTriangle(t, side/2)
```

```python
def main():
    nessa = turtle.Turtle()
    setUp(nessa, 100, "violet")
    nestedTriangle(nessa, 160)

    frank = turtle.Turtle()
    setUp(frank, -100, "red")
    fractalTriangle(frank, 160)

if __name__ == "__main__":
    main()
```

triangle.py

```python
1  import turtle
2  def setUp(t, dist, col):
3      t.penup()
4      t.forward(dist)
5      t.pendown()
6      t.color(col)
7
8  def nestedTriangle(t, side):
9      if side > 10:
10         for i in range(3):
11             t.forward(side)
12             t.left(120)
13         nestedTriangle(t, side/2)
```

triangle.py: II

```
14  def fractalTriangle(t, side):
15      if side > 10:
16          for i in range(3):
17              t.forward(side)
18              t.left(120)
19              fractalTriangle(t, side/2)
```

triangle.py: III

```python
20   def main():
21       side = int(input("Enter side of a
             triangle: "))
22       nessa = turtle.Turtle()
23       setUp(nessa, 100, "violet")
24       nestedTriangle(nessa, side)
25
26       frank = turtle.Turtle()
27       setUp(frank, -100, "red")
28       fractalTriangle(frank, side)
29
30   if __name__ == "__main__":
31       main()
```

# IDLE

```
#CSci 127 Teaching Staff
#Triangles two ways...
import turtle

def setUp(t, dist, col):
    t.penup()
    t.forward(dist)
    t.pendown()
    t.color(col)

def nestedTriangle(t, side):
    if side > 10:
        for i in range(3):
            t.forward(side)
            t.left(120)
        nestedTriangle(t, side/2)

def fractalTriangle(t, side):
    if side > 10:
        for i in range(3):
            t.forward(side)
            t.left(120)
            fractalTriangle(t, side/2)
```

(Demo with `IDLE`)

# Today's Topics



- More on Functions
- **Recap: Open Data**
- Top Down Design
- Design Challenge

# Design Question
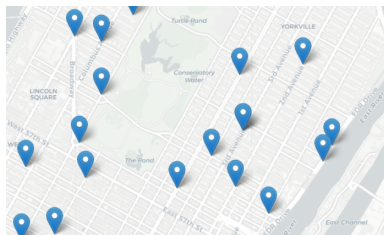


Design an algorithm that finds the collision that is closest to input location.

| DATE | TIME | BOROUGH | ZIP CODE | LATITUDE | LONGITUDE | LOCATION | ON STREET N | CROSS STREE | OFF STREET | NUMBER OF |
|------|------|---------|----------|----------|-----------|----------|-------------|-------------|------------|-----------|
| 12/31/16 | 9:56 | | | | | | 2 AVENUE | | | 0 |
| 12/31/16 | 9:55 | BRONX | 10462 | 40.83521 | -73.85497 | (40.8352098 | UNIONPORT | OLMSTEAD AVENUE | | 0 |
| 12/31/16 | 9:50 | | | | | | JESUP AVENUE | | | 0 |
| 12/31/16 | 9:40 | BROOKLYN | 11225 | 40.66911 | -73.95335 | (40.6691137 | ROGERS AVE | UNION STREET | | 0 |
| 12/31/16 | 20:23 | BROOKLYN | 11209 | 40.62578 | -74.02415 | (40.6257805 | 80 STREET | 5 AVENUE | | 0 |
| 12/31/16 | 20:20 | QUEENS | 11375 | 40.71958 | -73.83977 | (40.719584, | ASCAN AVEN | QUEENS BOULEVARD | | 0 |
| 12/31/16 | 20:15 | BROOKLYN | 11204 | | | | 60 STREET | BAY PARKWAY | | 0 |
| 12/31/16 | 20:10 | | | 40.66479 | -73.82047 | (40.6647944, | -73.8204653) | | | 0 |
| 12/31/16 | 20:10 | | | | | | 69 STREET | 37 AVENUE | | 0 |
| 12/31/16 | 20:05 | BRONX | 10457 | 40.85429 | -73.90026 | (40.8542925 | RYER AVENU | EAST 181 STREET | | 0 |

CSci 127 (Hunter)　　　　　　Lecture 8　　　　　　Nov 1, 2022　　20 / 44

# OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

How to approach this:

- Create a "To Do" list of what your program has to accomplish.
- Read through the problem, and break it into "To Do" items.
- Don't worry if you don't know how to do all the items you write down.

# OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

How to approach this:

- Create a "To Do" list of what your program has to accomplish.
- Read through the problem, and break it into "To Do" items.
- Don't worry if you don't know how to do all the items you write down.
- Example:
    1. Find data set (great place to look: NYC OpenData).
    2. Ask user for current location.
    3. Open up the CSV file.
    4. Check distance to each to user's location.
    5. Print the location with the smallest distance.

# OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

How to approach this:

- Create a "To Do" list of what your program has to accomplish.
- Read through the problem, and break it into "To Do" items.
- Don't worry if you don't know how to do all the items you write down.
- Example:
  1. Find data set (great place to look: NYC OpenData).
  2. Ask user for current location.
  3. Open up the CSV file.
  4. Check distance to each to user's location.
  5. Print the location with the smallest distance.
- Let's use function names as placeholders for the ones we're unsure...

# OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

1. Find data set (great place to look: NYC OpenData).

## OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

1. Find data set (great place to look: NYC OpenData).

   ```
   import pandas as pd
   inF = input('Enter CSV file name:')
   ```

## OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

1. Find data set (great place to look: NYC OpenData).

   ```
   import pandas as pd
   inF = input('Enter CSV file name:')
   ```

2. Ask user for current location.

## OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

1. Find data set (great place to look: NYC OpenData).
   ```
   import pandas as pd
   inF = input('Enter CSV file name:')
   ```

2. Ask user for current location.
   ```
   lat = float(input('Enter latitude:'))
   lon = float(input('Enter longitude:'))
   ```

## OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

1. Find data set (great place to look: NYC OpenData).
   ```
   import pandas as pd
   inF = input('Enter CSV file name:')
   ```

2. Ask user for current location.
   ```
   lat = float(input('Enter latitude:'))
   lon = float(input('Enter longitude:'))
   ```

3. Open up the CSV file.

## OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

1. Find data set (great place to look: NYC OpenData).
   ```
   import pandas as pd
   inF = input('Enter CSV file name:')
   ```

2. Ask user for current location.
   ```
   lat = float(input('Enter latitude:'))
   lon = float(input('Enter longitude:'))
   ```

3. Open up the CSV file.
   ```
   collisions = pd.read_csv(inF)
   ```

## OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

1. Find data set (great place to look: NYC OpenData).
   ```
   import pandas as pd
   inF = input('Enter CSV file name:')
   ```

2. Ask user for current location.
   ```
   lat = float(input('Enter latitude:'))
   lon = float(input('Enter longitude:'))
   ```

3. Open up the CSV file.
   ```
   collisions = pd.read_csv(inF)
   ```

4. Check distance to each to user's location.

## OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

1. Find data set (great place to look: NYC OpenData).
   ```
   import pandas as pd
   inF = input('Enter CSV file name:')
   ```

2. Ask user for current location.
   ```
   lat = float(input('Enter latitude:'))
   lon = float(input('Enter longitude:'))
   ```

3. Open up the CSV file.
   ```
   collisions = pd.read_csv(inF)
   ```

4. Check distance to each to user's location.
   ```
   closestLat, closestLon = findClosest(collisions, lat, lon)
   ```

## OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

1. Find data set (great place to look: NYC OpenData).

   ```
   import pandas as pd
   inF = input('Enter CSV file name:')
   ```

2. Ask user for current location.

   ```
   lat = float(input('Enter latitude:'))
   lon = float(input('Enter longitude:'))
   ```

3. Open up the CSV file.

   ```
   collisions = pd.read_csv(inF)
   ```

4. Check distance to each to user's location.

   ```
   closestLat, closestLon = findClosest(collisions, lat, lon)
   ```

5. Print the location with the smallest distance.

## OpenData Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

1. Find data set (great place to look: NYC OpenData).

   ```
   import pandas as pd
   inF = input('Enter CSV file name:')
   ```

2. Ask user for current location.

   ```
   lat = float(input('Enter latitude:'))
   lon = float(input('Enter longitude:'))
   ```

3. Open up the CSV file.

   ```
   collisions = pd.read_csv(inF)
   ```

4. Check distance to each to user's location.

   ```
   closestLat, closestLon = findClosest(collisions, lat, lon)
   ```

5. Print the location with the smallest distance.

   ```
   print("The closest is at lat:", closestLat, "and lon:",
   closestLon)
   ```

# Today's Topics

- More on Functions
- Recap: Open Data
- **Top Down Design**
- Design Challenge

# 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:
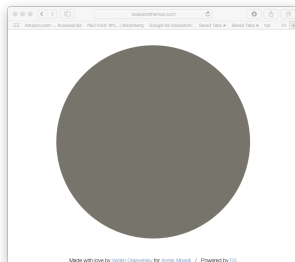


http://koalastothemax.com

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

# Demo
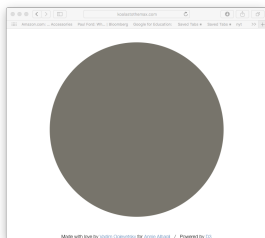
# Demo
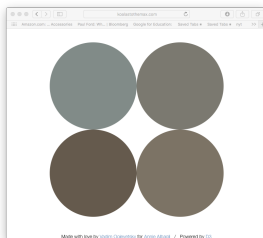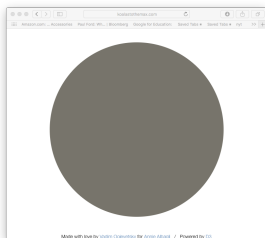
# Demo
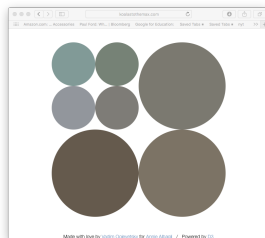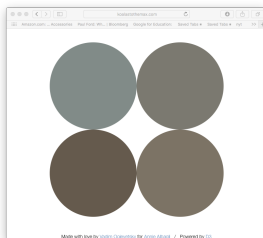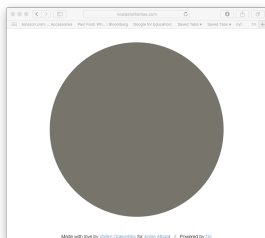
# Demo
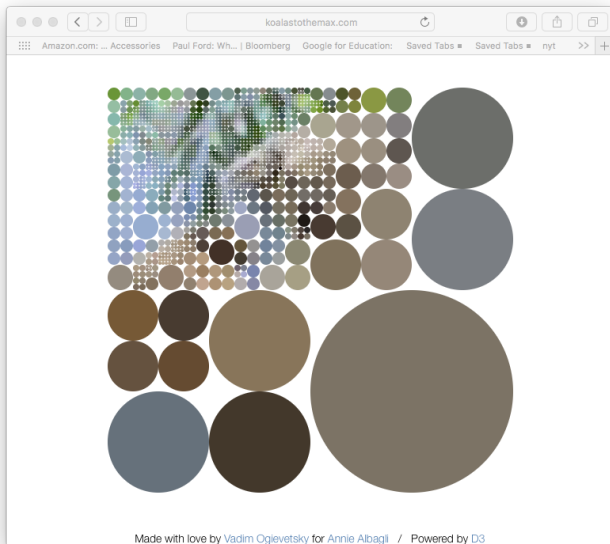
# Design: Koalas to the Max
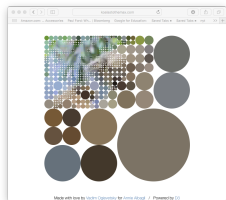


- **Input:** Image & mouse movements

# Design: Koalas to the Max
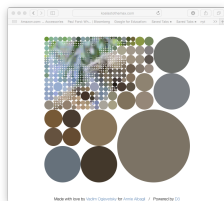


- **Input:** Image & mouse movements
- **Output:** Completed image

# Design: Koalas to the Max
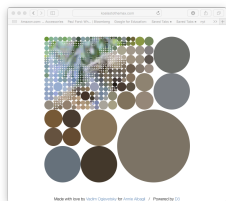


- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**

# Design: Koalas to the Max
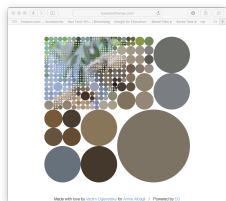


- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**
    - Every mouse movement,

# Design: Koalas to the Max
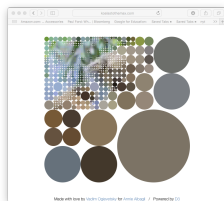


- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**
  - ► Every mouse movement,
  - ► Divide the region into 4 quarters.

# Design: Koalas to the Max
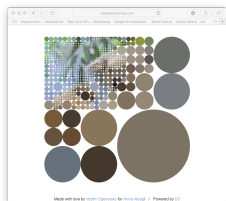


- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**
  - ▶ Every mouse movement,
  - ▶  Divide the region into 4 quarters.
  - ▶  Average the color of each quarter.

# Design: Koalas to the Max
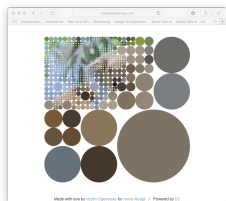


- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**
  - ▶ Every mouse movement,
  - ▶ Divide the region into 4 quarters.
  - ▶ Average the color of each quarter.
  - ▶ Set each quarter to its average.

What does the following code do?

```python
import random
import numpy as np
import matplotlib.pyplot as plt

h = 320 #height of image, make it divided by
     power of 2
w = 640 #width of image
img = np.zeros((h, w, 3))

level = 2
#hReg is height of region, needs to be int
hReg = h//2**level
wReg = w//2**level #wReg is width of region
```

What does the following code do?

```
14   for i in range(2**level):
15       for j in range(2**level):
16           img[i*hReg:(i+1)*hReg, j*wReg:(j+1)*
                 wReg] = [random.uniform(0,1),
                 random.uniform(0,1), random.
                 uniform(0,1)]
17           #random.uniform() returns a random
                 floating point number in [0, 1]
```

# What does the code do? III

```
18 plt.imshow(img)
19 plt.show()
20 #suppose level is 2, then there are 2**level
     = 4 cuts in horizontal and vertical
     direction,
21 #a total of 16 grids
22 plt.imsave("color_grid_" + str(2**level*2**
     level) + ".png", img)
```

# Calculate and set average color for region of an image

```python
def quarter(img2, levels):
    """
    Takes an image and the number of splits
        to make.
    Splits the image into regions (2**
        levels x 2**levels pieces)
    and averages each of these regions
        separately.
    Calls average() and setRegion() to
        average and set colors for the
        regions.
    """
```

## Calculate and set average color for region of an image

```
 8      h = img2.shape[0]
 9      w = img2.shape[1]
10      hReg = h//2**levels
11      wReg = w//2**levels
12      for i in range(2**levels):
13          for j in range(2**levels):
14              #Average the region:
15              r,g,b = average(img2[i*hReg:(i
                    +1)*hReg,j*wReg:(j+1)*wReg
                    ])
16              setRegion(img2[i*hReg:(i+1)*
                    hReg,j*wReg:(j+1)*wReg],r,g
                    ,b)
```

# Averaging `numpy` arrays

- Average each color channel of the image:

# Averaging `numpy` arrays

- Average each color channel of the image:
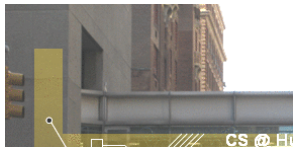
# Averaging numpy arrays

- Average each color channel of the image:
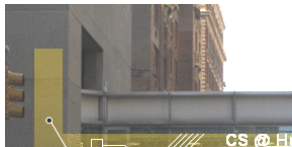


```
redAve = np.average(region[:,:,0])
```

# Averaging `numpy` arrays

- Average each color channel of the image:
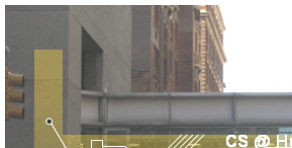


```
redAve = np.average(region[:,:,0])
greenAve = np.average(region[:,:,1])
```

# Averaging numpy arrays

- Average each color channel of the image:
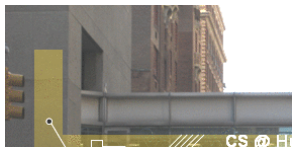


```
redAve = np.average(region[:,:,0])
greenAve = np.average(region[:,:,1])
blueAve = np.average(region[:,:,2])
```

# Averaging numpy arrays
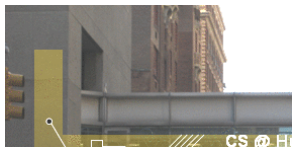
- Average each color channel of the image:



```
redAve = np.average(region[:,:,0])
greenAve = np.average(region[:,:,1])
blueAve = np.average(region[:,:,2])
```

- Set each pixel to the average value:

# Averaging numpy arrays
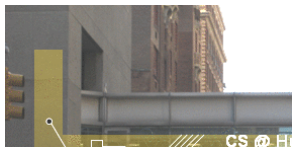
- Average each color channel of the image:



```python
redAve = np.average(region[:,:,0])
greenAve = np.average(region[:,:,1])
blueAve = np.average(region[:,:,2])
```

- Set each pixel to the average value:

```python
region[:,:,0] = redAve
```

# Averaging numpy arrays
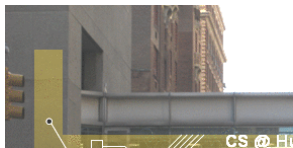
- Average each color channel of the image:



```
redAve = np.average(region[:,:,0])
greenAve = np.average(region[:,:,1])
blueAve = np.average(region[:,:,2])
```

- Set each pixel to the average value:

```
region[:,:,0] = redAve
region[:,:,1] = greenAve
```

# Averaging `numpy` arrays
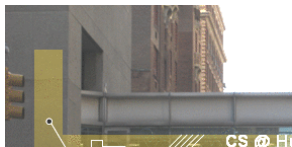
- Average each color channel of the image:



```python
redAve = np.average(region[:,:,0])
greenAve = np.average(region[:,:,1])
blueAve = np.average(region[:,:,2])
```

- Set each pixel to the average value:

```python
region[:,:,0] = redAve
region[:,:,1] = greenAve
region[:,:,2] = blueAve
```

# Averaging `numpy` arrays
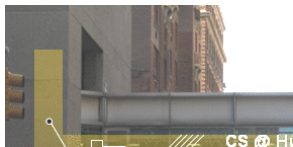
- Average each color channel of the image:



```
redAve = np.average(region[:,:,0])
greenAve = np.average(region[:,:,1])
blueAve = np.average(region[:,:,2])
```

- Set each pixel to the average value:

```
region[:,:,0] = redAve
region[:,:,1] = greenAve
region[:,:,2] = blueAve
```

# Today's Topics

- More on Functions
- Recap: Open Data
- Top Down Design
- **Design Challenge**

# Design Challenge

| Job ID | Agency | Posting T | # O | Business Title | Civil Service | Title Cod | Level | Job Category | Full- | Sala |
|--------|--------|-----------|-----|----------------|---------------|-----------|-------|--------------|-------|------|
| 246814 | DEPT OF INFO | External | 1 | Senior Architect Cloud Infrastructure D | SENIOR IT AF | 6800 | 0 | Information | F | |
| 246814 | DEPT OF INFO | Internal | 1 | Senior Architect Cloud Infrastructure D | SENIOR IT AF | 6800 | 0 | Information | F | |
| 247320 | DEPT OF ENVI | Internal | 2 | MECHANICAL ENGINEERING INTERN | MECHANICA | 20403 | 0 | Engineering, | F | |
| 247320 | DEPT OF ENVI | External | 2 | MECHANICAL ENGINEERING INTERN | MECHANICA | 20403 | 0 | Engineering, | F | |
| 269885 | DEPT OF ENVI | External | 1 | MECHANICAL ENGINEERING INTERN | MECHANICA | 20403 | 0 | Engineering, | F | |
| 269885 | DEPT OF ENVI | Internal | 1 | MECHANICAL ENGINEERING INTERN | MECHANICA | 20403 | 0 | Engineering, | F | |
| 285120 | NYC HOUSING | External | 1 | Deputy Director for Engineering | ADMINISTRA | 10015 | M3 | Engineering, | P | |
| 285120 | NYC HOUSING | Internal | 1 | Deputy Director for Engineering | ADMINISTRA | 10015 | M3 | Engineering, | P | |
| 287202 | DEPT OF ENVI | External | 4 | MECHANICAL ENGINEERING INTERN | MECHANICA | 20403 | 0 | Engineering, | F | |
| 287202 | DEPT OF ENVI | Internal | 4 | MECHANICAL ENGINEERING INTERN | MECHANICA | 20403 | 0 | Engineering, | F | |

(data.cityofnewyork.us/City-Government/NYC-Jobs/kpav-sd4t)

Find all current city job postings for internship positions.

# Design Challenge



(data.cityofnewyork.us/City-Government/NYC-Jobs/kpav-sd4t)

- **Input:** CSV file from NYC OpenData.

# Design Challenge



(data.cityofnewyork.us/City-Government/NYC-Jobs/kpav-sd4t)

- **Input:** CSV file from NYC OpenData.
- **Output:** A list of internships offered by the city.

# Design Challenge



(data.cityofnewyork.us/City-Government/NYC-Jobs/kpav-sd4t)

- **Input:** CSV file from NYC OpenData.
- **Output:** A list of internships offered by the city.
- **Process:**

# Design Challenge



(data.cityofnewyork.us/City-Government/NYC-Jobs/kpav-sd4t)

- **Input:** CSV file from NYC OpenData.
- **Output:** A list of internships offered by the city.
- **Process:**
  1. Open the file.

# Design Challenge



(data.cityofnewyork.us/City-Government/NYC-Jobs/kpav-sd4t)

- **Input:** CSV file from NYC OpenData.
- **Output:** A list of internships offered by the city.
- **Process:**
  1. Open the file.
  2. Select the rows that have "intern" in the business title.

# Design Challenge



(data.cityofnewyork.us/City-Government/NYC-Jobs/kpav-sd4t)

- **Input:** CSV file from NYC OpenData.
- **Output:** A list of internships offered by the city.
- **Process:**
    1. Open the file.
    2. Select the rows that have "intern" in the business title.
    3. Print out those rows.

# Recap

- Functions are a way to break code into pieces, that can be easily reused.

```
#Name:   your name here
#Date:   October 2017
#This program, uses functions,
#     says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

# Recap

```
#Name:  your name here
#Date:  October 2017
#This program, uses functions,
#     says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.

- Functions can have **input parameters** that bring information into the function,

# Recap

```
#Name:  your name here
#Date:  October 2017
#This program, uses functions,
#     says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.

- Functions can have **input parameters** that bring information into the function,

- And **return values** that send information back.

# Recap

```
#Name:  your name here
#Date:  October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.

- Functions can have **input parameters** that bring information into the function,

- And **return values** that send information back.

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

# Recap

```
#Name:   your name here
#Date:   October 2017
#This program, uses functions,
#    says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.

- Functions can have **input parameters** that bring information into the function,

- And **return values** that send information back.

- 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.

# Practice Quiz & Final Questions
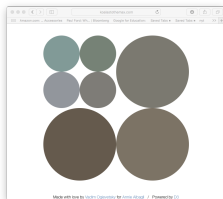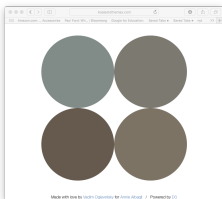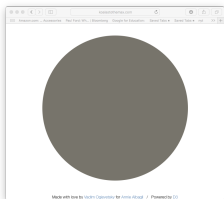


- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and
  - repeat.
- Past exams are on the webpage (under Final Exam Information).
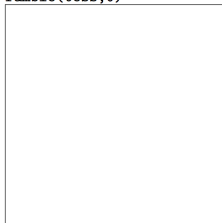- Theme: Functions! Starting with S18, V1, #4a and #4b.

# Final Exam: Spring 2018, Version 1, #4a

4. (a) Draw the output for the function calls:
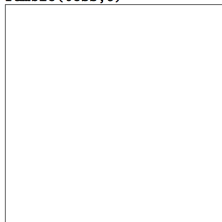
```
import turtle
tess = turtle.Turtle()
tess.shape("turtle")

def ramble(t,side):
    if side == 0:
        t.stamp()
    else:
        for i in range(side):
            t.forward(50)
            t.left(360/side)
```

i. `ramble(tess,0)`

ii. `ramble(tess,6)`

# Final Exam: Spring 2018, Version 1, #4a

4. (a) Draw the output for the function calls:

```
import turtle
tess = turtle.Turtle()
tess.shape("turtle")

def ramble(t,side):
    if side == 0:
        t.stamp()
    else:
        for i in range(side):
            t.forward(50)
            t.left(360/side)
```

i. ramble(tess,0)

ii. ramble(tess,6)

(Demo with `trinket`)

# Final Exam: Spring 2018, Version 1, #4b

(b) For the following code:

```
def v1(vincent, munem):
    if vincent + munem > 0:
        return vincent
    else:
        return -1
```

```
def start():
    panda = 20
    minh = -30
    qiuqun = v1(panda,minh)
    return qiuqun
```

   i. What are the formal parameters for `v1()`:

   ii. What are the formal parameters for `start()`:
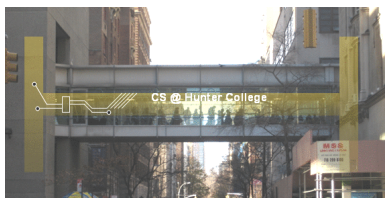
   iii. What does `start()` return:

# Weekly Reminders!



Before next lecture, don't forget to:
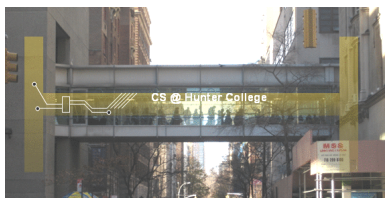
- Work on this week's Online Lab

# 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 (**once a week**) in lab 1001G Hunter North
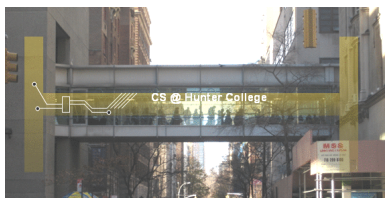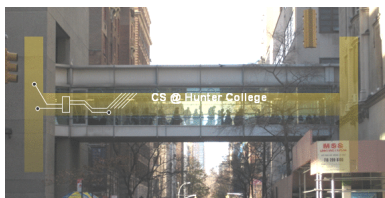
# Weekly Reminders!



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 (**once a week**) in lab 1001G Hunter North
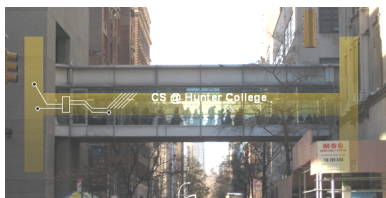- Submit this week's 5 programming assignments **(programs 36-40)**

# 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 (**once a week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments **(programs 36-40)**
- If you need help, schedule an appointment for Tutoring in lab 1001G 11:30am-5:30pm

# 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 (**once a week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments **(programs 36-40)**
- If you need help, schedule an appointment for Tutoring in lab 1001G 11:30am-5:30pm
- Take the Lecture Preview on Blackboard on Monday (or no later than 10am on Tuesday)

# Lecture Slips & Writing Boards



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