

# CSci 127: Introduction to Computer Science



[hunter.cuny.edu/csci](https://hunter.cuny.edu/csci)

# 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/s22/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/s22/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**.

```
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

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

```
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

- 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**.

```
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

```
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

```
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

```
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?

```
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","nomel")`

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

Return:

Return:

Return:

# Python Tutor

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

(a) enigma1(7,"caramel","d'œufs de lait")
(b) enigma1(3,"cupcake","vanille")
(c) enigma1(10,"pie","tome1")
```

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

Return:

Return:

Return:

(Demo with pythonTutor)

# Input Parameters

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

```
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

# Input Parameters

```
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

```
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

```
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

```
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?

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

```
#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] )
```

# 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:*

```
#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)
```

```
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()
```

```
#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)
```

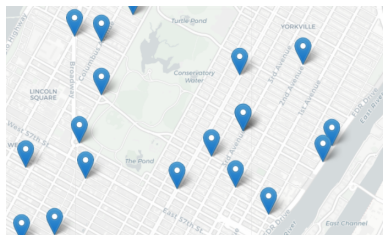
(Demo with IDLE)

# 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	CROSS STREET	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 AVENUE	EAST 181 STREET		0

# 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.

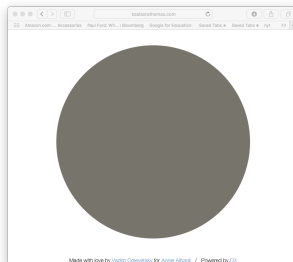
# Top-Down Design



- 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.



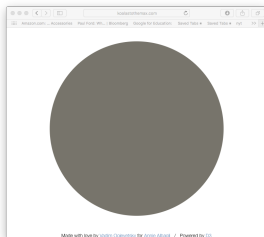
# Challenge:



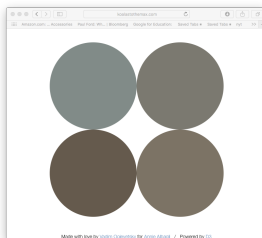
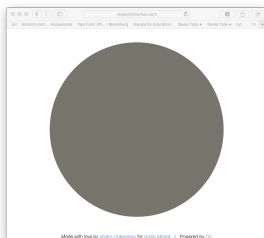
`http://koalastothemax.com`

- 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.

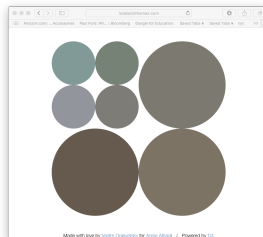
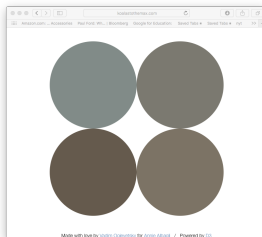
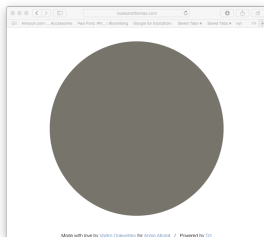
# Demo



# Demo



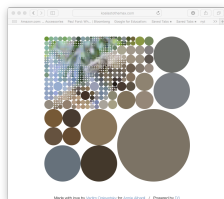
# Demo



# Demo

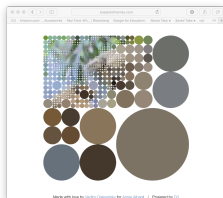


# Design: Koalas to the Max



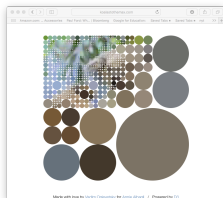
- **Input:** Image & mouse movements

# Design: Koalas to the Max



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

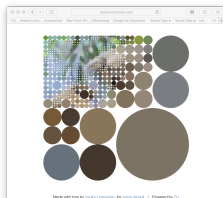
# Design: Koalas to the Max



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

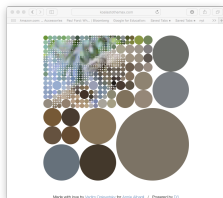


# Design: Koalas to the Max



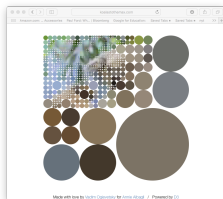
- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**
  - ▶ Every mouse movement,

# Design: Koalas to the Max



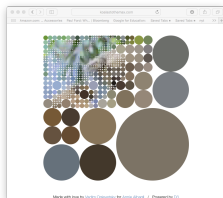
- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**
  - ▶ Every mouse movement,
  - ▶ Divide the region into 4 quarters.

# 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



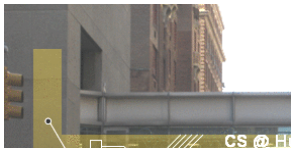
- **Input:** Image & mouse movements
- **Output:** Completed image
- **Design:**
  - ▶ Every mouse movement,
  - ▶ Divide the region into 4 quarters.
  - ▶ Average the color of each quarter.
  - ▶ Set each quarter to its average.

# Averaging numpy arrays

- Average each color channel of the image:

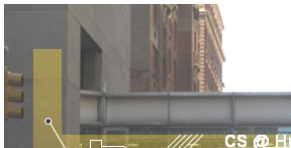
# Averaging numpy arrays

- Average each color channel of the image:



# Averaging numpy arrays

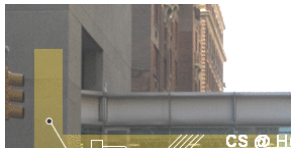
- Average each color channel of the image:



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

# Averaging numpy arrays

- Average each color channel of the image:

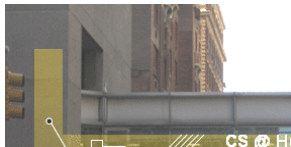


```
redAve = np.average(region[:, :, 0])  
greenAve = np.average(region[:, :, 1])
```



# Averaging numpy arrays

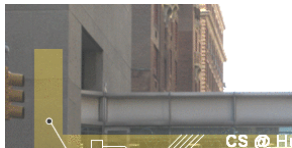
- Average each color channel of the image:



```
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:

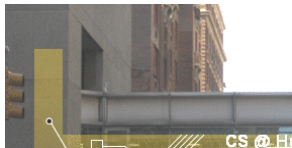


```
redAve = np.average(region[:, :, 0])  
greenAve = np.average(region[:, :, 1])  
blueAve = np.average(region[:, :, 2])
```

- Set each pixel to the average value:

# Averaging numpy arrays

- Average each color channel of the image:



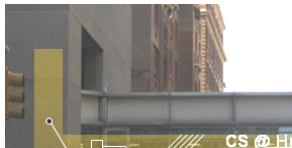
```
redAve = np.average(region[:, :, 0])  
greenAve = np.average(region[:, :, 1])  
blueAve = np.average(region[:, :, 2])
```

- Set each pixel to the average value:

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

# Averaging numpy arrays

- Average each color channel of the image:



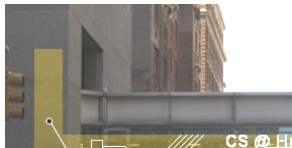
```
redAve = np.average(region[:, :, 0])  
greenAve = np.average(region[:, :, 1])  
blueAve = np.average(region[:, :, 2])
```

- Set each pixel to the average value:

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

# Averaging numpy arrays

- Average each color channel of the image:



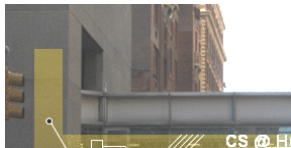
```
redAve = np.average(region[:, :, 0])  
greenAve = np.average(region[:, :, 1])  
blueAve = np.average(region[:, :, 2])
```

- Set each pixel to the average value:

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

# 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-	Sal
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

Job ID	Agency	Posting T #	O Business Title	Civil Service	Title Code	Level	Job Category	Full-	Salary Range	Salary Range
246814	DEPT OF INFO External	1	Senior Architect Cloud Infrastructure Di	SENIOR IT AR	6800	0	Information	F	100000	130000
246814	DEPT OF INFO Internal	1	Senior Architect Cloud Infrastructure Di	SENIOR IT AR	6800	0	Information	F	100000	130000
247320	DEPT OF ENVI Internal	2	MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
247320	DEPT OF ENVI External	2	MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
269885	DEPT OF ENVI External	1	MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
269885	DEPT OF ENVI Internal	1	MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
285120	NYC HOUSING External	1	Deputy Director for Engineering	ADMINISTRA	10015	M3	Engineering	P	115000	130000
285120	NYC HOUSING Internal	1	Deputy Director for Engineering	ADMINISTRA	10015	M3	Engineering	P	115000	130000
287202	DEPT OF ENVI External	4	MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
287202	DEPT OF ENVI Internal	4	MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000

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

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

# Design Challenge

Job ID	Agency	Posting T #	O Business Title	Civil Service	Title Code	Level	Job Category	Full-	Salary Range	Salary Range
246814	DEPT OF INFO	External	1 Senior Architect Cloud Infrastructure Di	SENIOR IT AR	6800	0	Information	F	100000	130000
246814	DEPT OF INFO	Internal	1 Senior Architect Cloud Infrastructure Di	SENIOR IT AR	6800	0	Information	F	100000	130000
247320	DEPT OF ENVI	Internal	2 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
247320	DEPT OF ENVI	External	2 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
269885	DEPT OF ENVI	External	1 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
269885	DEPT OF ENVI	Internal	1 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
285120	NYC HOUSING	External	1 Deputy Director for Engineering	ADMINISTRA	10015	M3	Engineering	P	115000	130000
285120	NYC HOUSING	Internal	1 Deputy Director for Engineering	ADMINISTRA	10015	M3	Engineering	P	115000	130000
287202	DEPT OF ENVI	External	4 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
287202	DEPT OF ENVI	Internal	4 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000

(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

Job ID	Agency	Posting T	# O	Business Title	Civil Service	Title Code	Level	Job Category	Full-	Salary Range	Salary Range
246814	DEPT OF INFO	External	1	Senior Architect Cloud Infrastructure Di	SENIOR IT AR	6800	0	Information	F	100000	130000
246814	DEPT OF INFO	Internal	1	Senior Architect Cloud Infrastructure Di	SENIOR IT AR	6800	0	Information	F	100000	130000
247320	DEPT OF ENVI	Internal	2	MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
247320	DEPT OF ENVI	External	2	MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
269885	DEPT OF ENVI	External	1	MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
269885	DEPT OF ENVI	Internal	1	MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
285120	NYC HOUSING	External	1	Deputy Director for Engineering	ADMINISTRA	10015	M3	Engineering	P	115000	130000
285120	NYC HOUSING	Internal	1	Deputy Director for Engineering	ADMINISTRA	10015	M3	Engineering	P	115000	130000
287202	DEPT OF ENVI	External	4	MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
287202	DEPT OF ENVI	Internal	4	MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000

(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

Job ID	Agency	Posting T #	O Business Title	Civil Service	Title Code	Level	Job Category	Full-	Salary Range	Salary Range
246814	DEPT OF INFO	External	1 Senior Architect Cloud Infrastructure Di	SENIOR IT AR	6800	0	Information	F	100000	130000
246814	DEPT OF INFO	Internal	1 Senior Architect Cloud Infrastructure Di	SENIOR IT AR	6800	0	Information	F	100000	130000
247320	DEPT OF ENVI	Internal	2 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
247320	DEPT OF ENVI	External	2 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
269885	DEPT OF ENVI	External	1 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
269885	DEPT OF ENVI	Internal	1 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
285120	NYC HOUSING	External	1 Deputy Director for Engineering	ADMINISTRA	10015	M3	Engineering	P	115000	130000
285120	NYC HOUSING	Internal	1 Deputy Director for Engineering	ADMINISTRA	10015	M3	Engineering	P	115000	130000
287202	DEPT OF ENVI	External	4 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
287202	DEPT OF ENVI	Internal	4 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000

(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

Job ID	Agency	Posting T #	O Business Title	Civil Service	Title Code	Level	Job Category	Full-	Salary Range	Salary Range
246814	DEPT OF INFO	External	1 Senior Architect Cloud Infrastructure	DI SENIOR IT AR	6800	0	Information	F	100000	130000
246814	DEPT OF INFO	Internal	1 Senior Architect Cloud Infrastructure	DI SENIOR IT AR	6800	0	Information	F	100000	130000
247320	DEPT OF ENVI	Internal	2 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
247320	DEPT OF ENVI	External	2 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
269885	DEPT OF ENVI	External	1 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
269885	DEPT OF ENVI	Internal	1 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
285120	NYC HOUSING	External	1 Deputy Director for Engineering	ADMINISTRA	10015	M3	Engineering	P	115000	130000
285120	NYC HOUSING	Internal	1 Deputy Director for Engineering	ADMINISTRA	10015	M3	Engineering	P	115000	130000
287202	DEPT OF ENVI	External	4 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
287202	DEPT OF ENVI	Internal	4 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000

(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

Job ID	Agency	Posting T #	O Business Title	Civil Service	Title Code	Level	Job Category	Full-	Salary Range	Salary Range
246814	DEPT OF INFO	External	1 Senior Architect Cloud Infrastructure	Di SENIOR IT AR	6800	0	Information	F	100000	130000
246814	DEPT OF INFO	Internal	1 Senior Architect Cloud Infrastructure	Di SENIOR IT AR	6800	0	Information	F	100000	130000
247320	DEPT OF ENVI	Internal	2 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
247320	DEPT OF ENVI	External	2 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
269885	DEPT OF ENVI	Internal	1 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
269885	DEPT OF ENVI	Internal	1 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
285120	NYC HOUSING	External	1 Deputy Director for Engineering	ADMINISTRA	10015	M3	Engineering	P	115000	130000
285120	NYC HOUSING	Internal	1 Deputy Director for Engineering	ADMINISTRA	10015	M3	Engineering	P	115000	130000
287202	DEPT OF ENVI	External	4 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000
287202	DEPT OF ENVI	Internal	4 MECHANICAL ENGINEERING INTERN	MECHANICA	20403	0	Engineering	F	52000	52000

(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

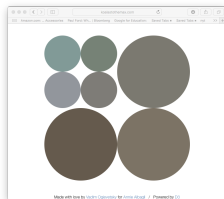
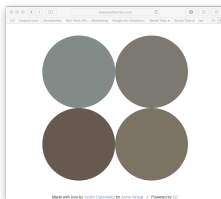
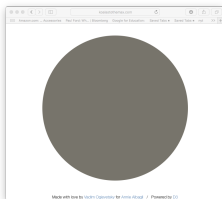
```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#    says hello to the world!

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

Name:

EmpID:

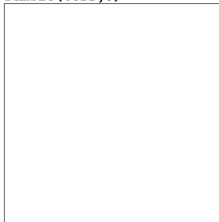
CSci 127 Final, S18, V1

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

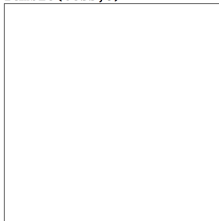
```
import turtle
tess = turtle.Turtle()
tess.shape("turtle")

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

Names: \_\_\_\_\_ ExamID: \_\_\_\_\_ CSci 127 Final, S18, V1

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

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

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

i. random\_sides(0)



ii. random\_sides(4)



(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 1001E Hunter North

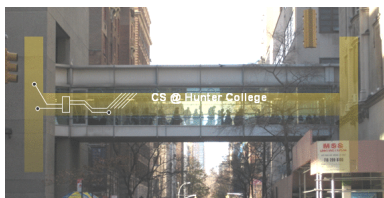
# Weekly Reminders!



Before next lecture, don't forget to:

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

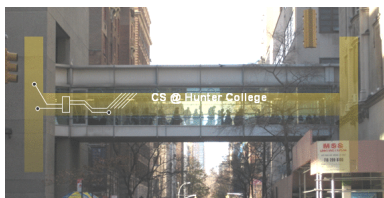
# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001E Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every two weeks**) in lab 1001E Hunter North
- Submit this week's 5 programming assignments (**programs 36-40**)

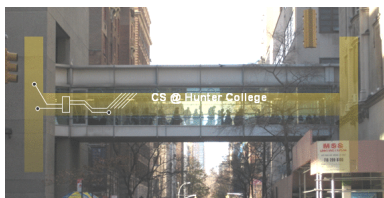
# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001E Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every two weeks**) in lab 1001E Hunter North
- Submit this week's 5 programming assignments (**programs 36-40**)
- If you need help, schedule an appointment for Tutoring in lab 1001E 11am-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 1001E Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every two weeks**) in lab 1001E Hunter North
- Submit this week's 5 programming assignments (**programs 36-40**)
- If you need help, schedule an appointment for Tutoring in lab 1001E 11am-5pm
- 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.