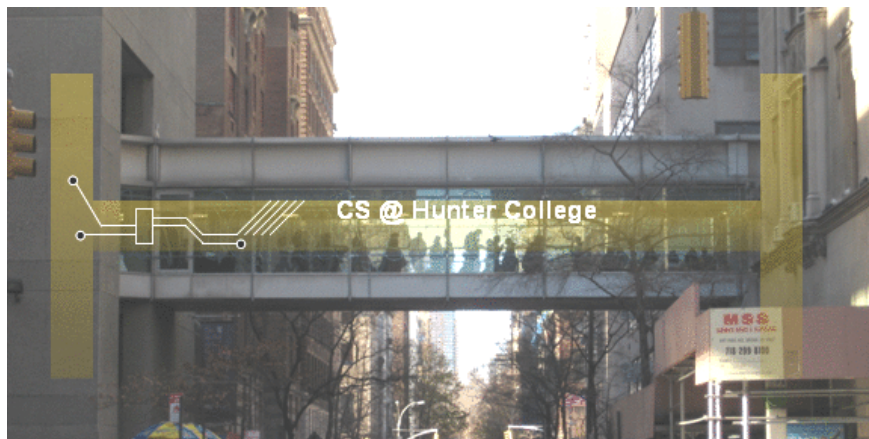


CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Frequently Asked Questions

From email

- Can you go through the OpenData challenge from last week?

Frequently Asked Questions

From email

- **Can you go through the OpenData challenge from last week?**
Yes, we'll start with functions, and then go on to the OpenData challenge.

Frequently Asked Questions

From email

- **Can you go through the OpenData challenge from last week?**
Yes, we'll start with functions, and then go on to the OpenData challenge.
- **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/summer21/syl.html>*

Frequently Asked Questions

From email

- **Can you go through the OpenData challenge from last week?**
Yes, we'll start with functions, and then go on to the OpenData challenge.
- **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/summer21/syl.html>*
- **Can I take the course Credit/No Credit?**
*Yes, but check with your advisor that it is possible with your major and standing.
Learn more about it here:
<http://catalog.hunter.cuny.edu/content.php?catoid=43&navoid=13649>*

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

```
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

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

(a) enigma(7,"coronel","dalon de leche")
(b) enigma(3,"cupcake","vanilla")
(c) enigma(10,"pie","towel")
```

```
def cont1(s):
    r = ""
    for i in range(len(s)-1,-1,-1):
        r = s + str(i)
    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?

```
#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:

```
#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()
```

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)
```

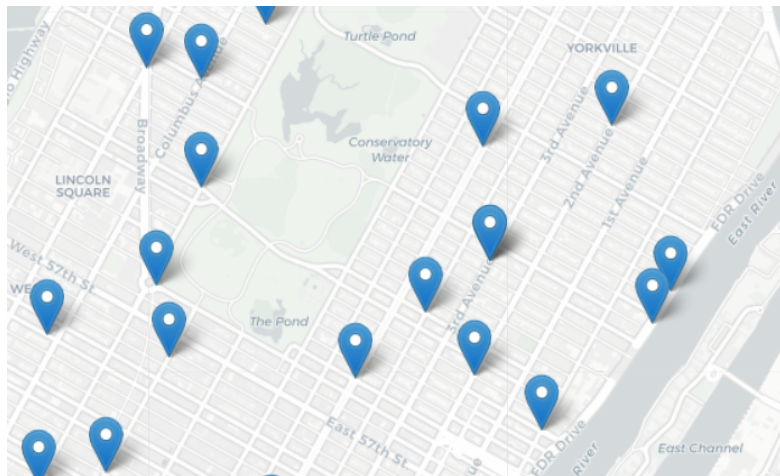
(Demo with IDLE)

Today's Topics



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

OpenData Design Question



Design an algorithm that finds the closest collision.

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  
i n F = input('Enter CSV file name:')
```

- 2 Ask user for current location.

```
l a t = float(input('Enter latitude:'))  
l o n = float(input('Enter longitude:'))
```

- 3 Open up the CSV file.

```
col l i s i o n s = pd.read_csv(i n F)
```

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  
i n F = input('Enter CSV file name:')
```

- 2 Ask user for current location.

```
l a t = float(input('Enter latitude:'))  
l o n = float(input('Enter longitude:'))
```

- 3 Open up the CSV file.

```
col l i s i o n s = pd.read_csv(i n F)
```

- 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  
i n F = input('Enter CSV file name:')
```

- 2 Ask user for current location.

```
l a t = float(input('Enter latitude:'))  
l o n = float(input('Enter longitude:'))
```

- 3 Open up the CSV file.

```
col l i s i o n s = pd.read_csv(i n F)
```

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

```
cl o s e s t L a t , cl o s e s t L o n = fi n d C l o s e s t ( col l i s i o n s , l a t , l o n )
```


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  
i n F = i n p u t (' E n t e r C S V f i l e n a m e : ' )
```

- 2 Ask user for current location.

```
l a t = f l o a t ( i n p u t (' E n t e r l a t i t u d e : ' ) )  
l o n = f l o a t ( i n p u t (' E n t e r l o n g i t u d e : ' ) )
```

- 3 Open up the CSV file.

```
c o l l i s i o n s = p d . r e a d _ c s v ( i n F )
```

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

```
c l o s e s t L a t , c l o s e s t L o n = f i n d C l o s e s t ( c o l l i s i o n s , l a t , l o n )
```

- 5 Print the location with the smallest distance.

```
p r i n t (" T h e c l o s e s t i s a t l a t : " , l a t , " a n d l o n : " , l o n )
```

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

(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.

Recap

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

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

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

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

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

(Demo with trinket)

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

Reminders!

Before next lecture, don't forget to:
Review this class's Lecture and Lab

Reminders!

Before next lecture, don't forget to:

Review this class's Lecture and Lab

Take the Lab Quiz on Gradescope by 9pm on today

Reminders!

Before next lecture, don't forget to:

Review this class's Lecture and Lab

Take the Lab Quiz on Gradescope by 9pm on today

Submit this class's 5 programming assignments (programs 36-40)

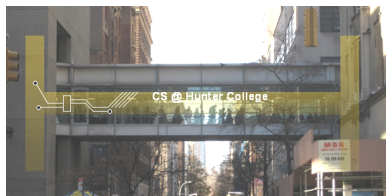
Reminders!



Before next lecture, don't forget to:

- Review this class's Lecture and Lab
- Take the Lab Quiz on Gradescope by 9pm on today
- Submit this class's 5 programming assignments (programs 36-40)
- Attend tutoring on Tues, Wed, or Thu for help!!!

Reminders!



Before next lecture, don't forget to:

- Review this class's Lecture and Lab
- Take the Lab Quiz on Gradescope by 9pm on today
- Submit this class's 5 programming assignments (programs 36-40)
- Attend tutoring on Tues, Wed, or Thu for help!!!