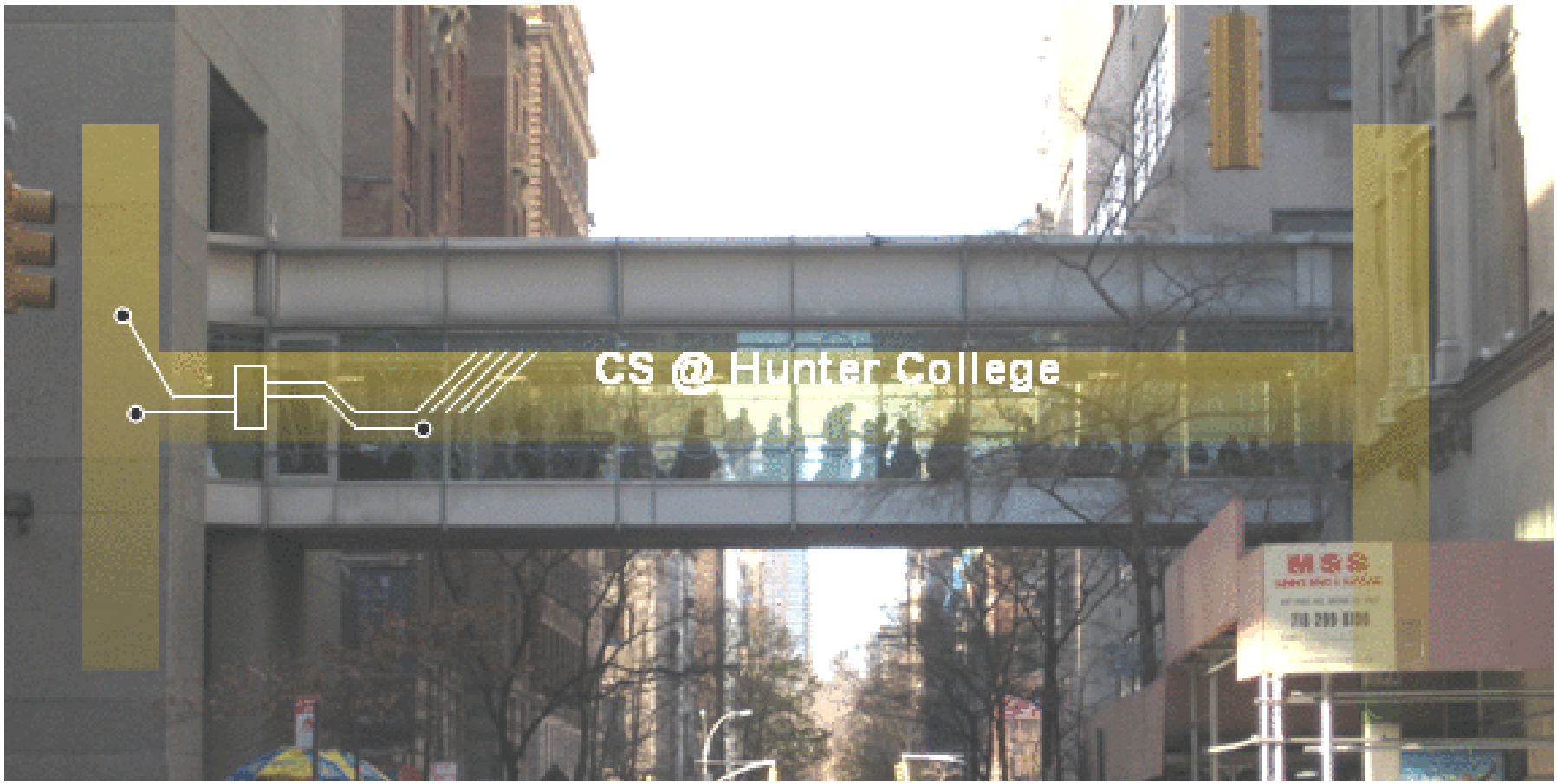
CSCI 127: Introduction to Computer Science



hunter.cuny.edu/csci

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

```
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?

dessert.py

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

dessert.py

```
9  def foo(w):
10     r = ""
11     for i in range(len(w)-1,-1,-1):
12         r = r + w[i]
13     return(r)
14
15 enigma(7,"caramel","dulce de leche")
16 enigma(3,"cupcake","vanilla")
17 enigma(10,"pie","nomel")
```

Demo: dessert.py

Link to code

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.
- 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)

Today's Topics

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

Design Question

- Design an algorithm that uses NYC Open Data collision data and computes the closest collision to the location the user provides
- See the dataset: Motor Vehicle Collisions - Crashes

CRASH D... *crash_date*	CRASH TIME *crash_time*	BOROUGH *borough*	ZIP CODE *zip_code*	LATITUDE *latitude*	LONGITUDE *longitude*
10/03/2023	0:29	MANHATTAN	10027	40.81722	-73.95228
10/03/2023	0:54	QUEENS	11434	40.68775	-73.79039
10/03/2023	1:14			40.73243	-73.83512
10/03/2023	3:15			40.666546	-73.78808
10/03/2023	5:15	QUEENS	11420	40.67484	-73.82344
10/03/2023	0:00			40.752663	-73.746025
10/03/2023	5:30			40.755203	-73.74191
10/03/2023	7:10				

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)

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

2. Ask the user for location

```
inLat = input("Enter latitude:")
inLon = input("Enter longitude:")
```

3. Open the CSV file with the crash data

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

4. Calculate the closet collision to the user

```
crashLat, crashLon = findClosest(collisions, inLat, inLon)
locationStr = "(" + crashLat + "," + crashLon + ")"
```

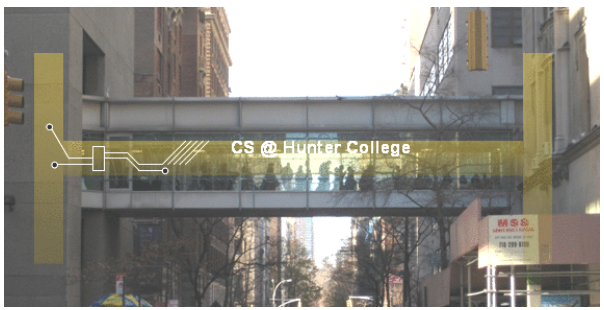
5. Print the location

```
print("Closest collision at (LAT,LON):", locationStr)
```

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.
  - 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.

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**

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()
```

triangle.py

```
1 import turtle
2
3 def setUp(t, dist, col):
4     t.penup()
5     t.forward(dist)
6     t.pendown()
7     t.color(col)
```

triangle.py: II

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

triangle.py: III

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

triangle.py: IV

```
1  def main():
2      side = int(input("Enter side length: "))
3      nessa = turtle.Turtle()
4      setUp(nessa, 100, "violet")
5      nestedTriangle(nessa, side)
6
7      frank = turtle.Turtle()
8      setUp(frank, -100, "red")
9      fractalTriangle(frank, side)
10
11 if __name__ == "__main__":
12     main()
```

Demo

```
#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 Think CS: 16. Recursion

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()
```

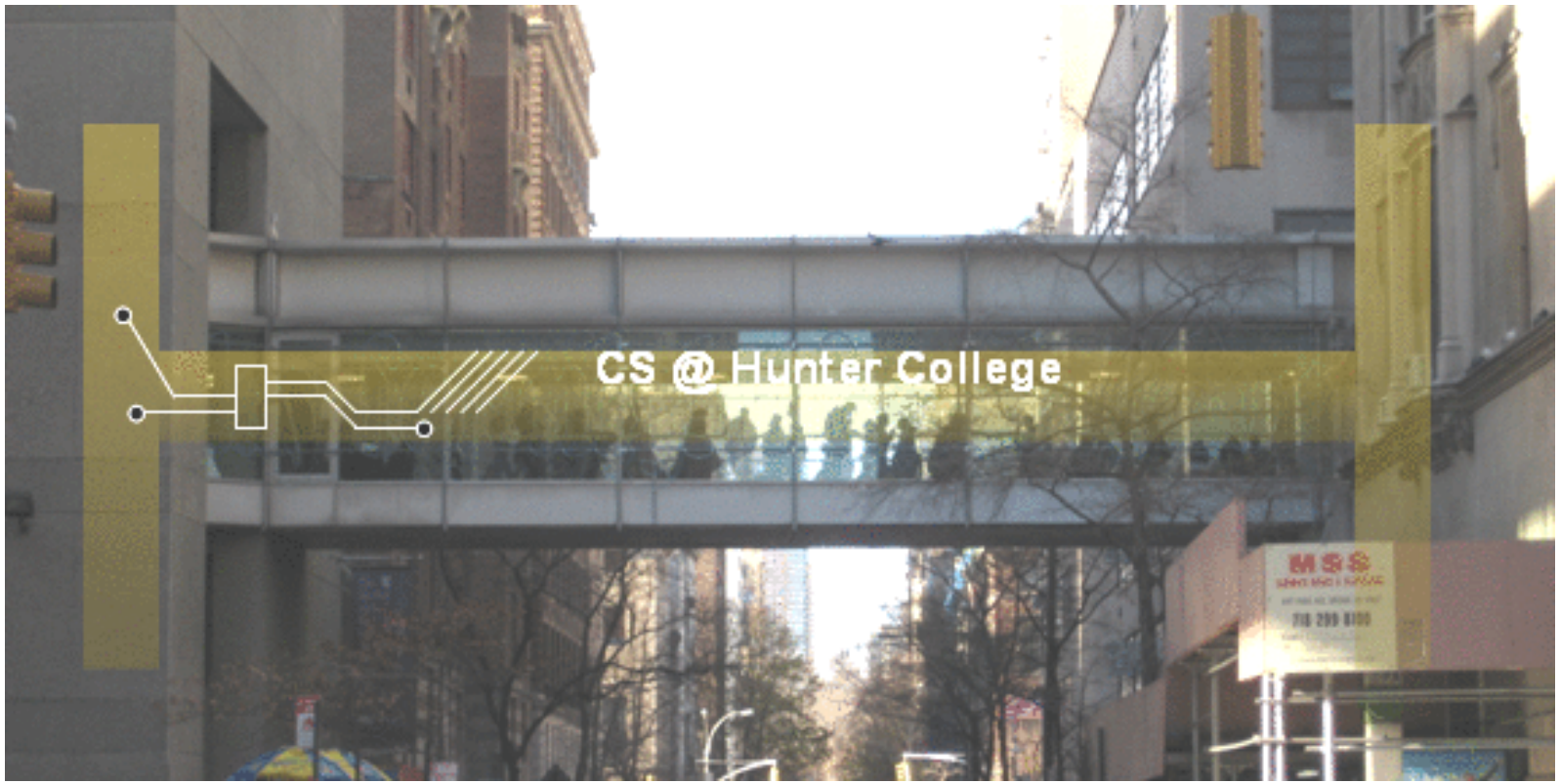
- Functions are a way to break code into pieces, that can be easily reused.
- Functions can have **input parameters** that bring information into the function,
- And **return values** that send information back.
- Top-down design: breaking into subproblems, and implementing each part separately.
- Excellent approach: can then test each part separately before adding it to a large program.

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
- 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.