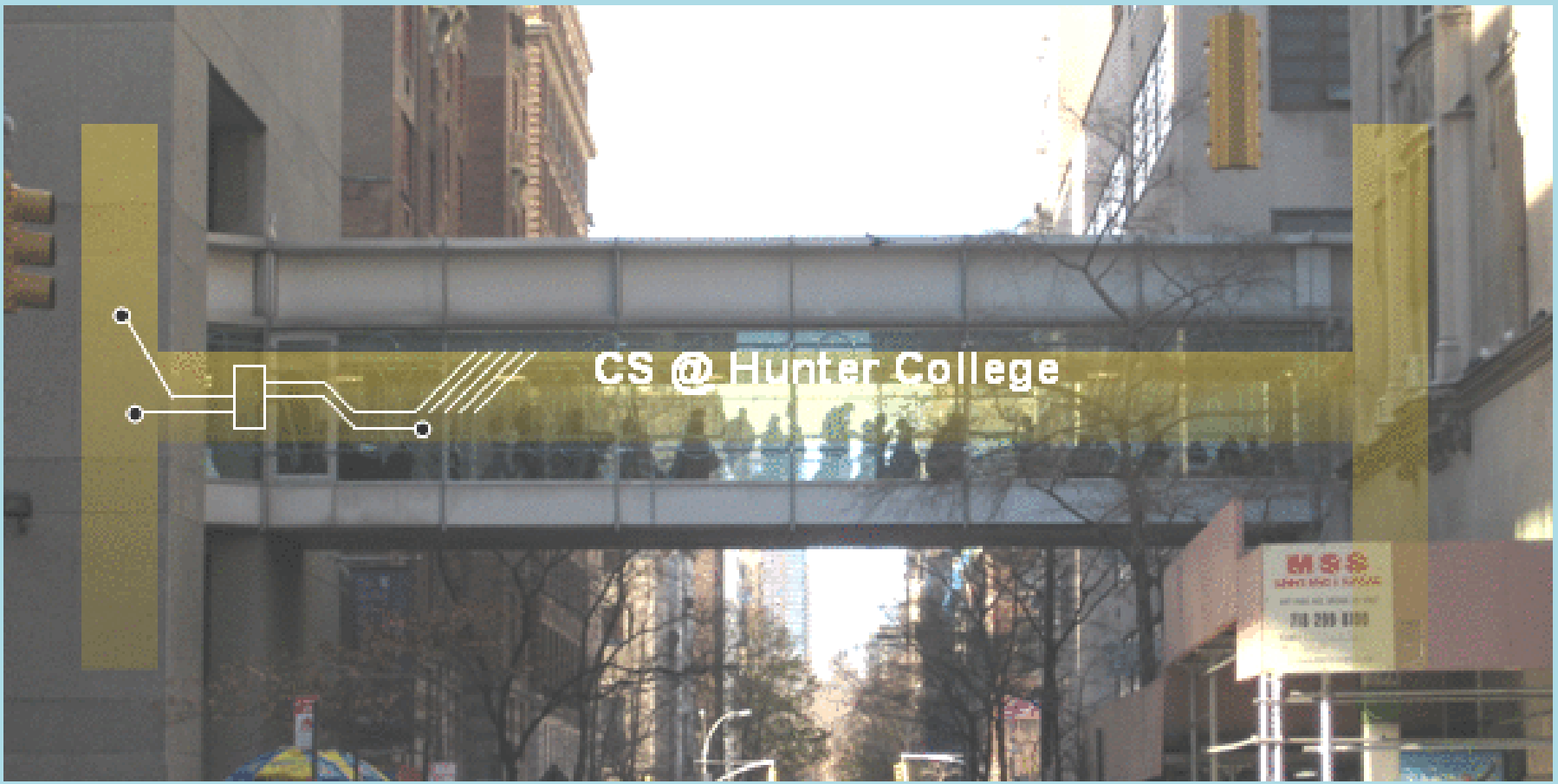
CSCI 127: Introduction to Computer Science



hunter.cuny.edu/csci

Today's Topics

- Recap: Functions & Top Down Design
- Mapping GIS Data
- Random Numbers
- Indefinite Loops

Today's Topics

- **Recap: Functions & Top Down Design**
- Mapping GIS Data
- Random Numbers
- Indefinite Loops

Challenge:

```
def prob4(amy, beth):
    if amy > 4:
        print("Easy case")
        kate = -1
    else:
        print("Complex case")
        kate = helper(amy,beth)
    return(kate)
```

```
def helper(meg,jo):
    s = ""
    for j in range(meg):
        print(j, ": ", jo[j])
        if j % 2 == 0:
            s = s + jo[j]
            print("Building s:", s)
    return(s)
```

- What are the formal parameters for the functions?
- What is the output of:

```
r = prob4(4,"city")
print("Return:  ", r)
```

- What is the output of:

```
r = prob4(2,"university")
print("Return:  ", r)
```

Challenge:

```
def prob4(amy, beth):
    if amy > 4:
        print("Easy case")
        kate = -1
    else:
        print("Complex case")
        kate = helper(amy,beth)
    return(kate)
```

```
def helper(meg,jo):
    s = ""
    for j in range(meg):
        print(j, ": ", jo[j])
        if j % 2 == 0:
            s = s + jo[j]
            print("Building s:", s)
    return(s)
```

- What are the formal parameters for the functions?

Challenge:

```
def prob4(amy, beth):
    if amy > 4:
        print("Easy case")
        kate = -1
    else:
        print("Complex case")
        kate = helper(amy,beth)
    return(kate)
```

```
def helper(meg,jo):
    s = ""
    for j in range(meg):
        print(j, ": ", jo[j])
        if j % 2 == 0:
            s = s + jo[j]
            print("Building s:", s)
    return(s)
```

- What are the formal parameters for the functions?

Challenge:

```
def prob4(amy, beth):
    if amy > 4:
        print("Easy case")
        kate = -1
    else:
        print("Complex case")
        kate = helper(amy,beth)
    return(kate)
```

```
def helper(meg,jo):
    s = ""
    for j in range(meg):
        print(j, ": ", jo[j])
        if j % 2 == 0:
            s = s + jo[j]
            print("Building s:", s)
    return(s)
```

- What is the output of:

```
r = prob4(4,"city")
print("Return:  ", r)
```

- What is the output of:

```
r = prob4(2,"university")
print("Return:  ", r)
```

Python Tutor

```
def prob4(amy, beth):
    if amy > 4:
        print("Easy case")
        kate = -1
    else:
        print("Complex case")
        kate = helper(amy,beth)
    return(kate)
```

```
def helper(meg,jo):
    s = ""
    for j in range(meg):
        print(j, ": ", jo[j])
        if j % 2 == 0:
            s = s + jo[j]
            print("Building s:", s)
    return(s)
```

- Demo with `pythonTutor`
- "Sisters Example" under week 9 handouts (on course page)

Top-Down Design

- **Top-down design** is the process of breaking the task 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:

Write the missing functions for the program:

```
1  def main():
2      #setUp: returns a purple turtle with pen up
3      tess = setUp()
4      for i in range(5):
5          #getInput: returns two numbers from user input
6          x,y = getInput()
7          #markLocation: moves tess to (x,y) and stamps
8          markLocation(tess,x,y)
```

Group Work: Fill in Missing Pieces

1. Write import statements.

```
1 import turtle
```

Fill in Missing Pieces

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

```
1 def setUp():
2     #FILL IN
3 def getInput():
4     #FILL IN
5 def markLocation(t,x,y):
6     #FILL IN
```

Fill in Missing Pieces

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

```
1 def setUp():
2     #FILL IN
3     return newTurtle
4 def getInput():
5     #FILL IN
6     return x,y
7 def markLocation(t,x,y):
8     #FILL IN
9     #does not return a value
```

Fill in Missing Pieces

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

```
1 def setUp():
2     #Create a new turtle
3     newTurtle = turtle.Turtle()
4     #Set the turtle so the pen is up
5     newTurtle.penup()
6     #Set the turtle so that the color is purple
7     newTurtle.color("purple")
8     #return the turtle with the setup
9     return newTurtle
```

Fill in Missing Pieces

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

```
1 def getInput():
2     #Ask the user for a value, convert it to
3     #an int and store it in x
4     x = int(input("Enter x: "))
5     #Ask the user for another value, convert it to
6     #an int and store it in y
7     y = int(input("Enter y: "))
8     #we can return two items in python
9     return x, y
```

Fill in Missing Pieces

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

```
1 def markLocation(t, x, y):
2     #t is the turtle given to the function
3     #x and y are locations given to the function
4     t.goto(x, y)
5     t.stamp()
6     #does not return a value
```

Complete Code (1/2)

```
1  import turtle
2
3  def main():
4      tess = setUp()
5      for i in range(5):
6          x,y = getInput()
7          markLocation(tess,x,y)
8
9  def setUp():
10     newTurtle = turtle.Turtle()
11     newTurtle.color("purple")
12     newTurtle.penup()
13     return(newTurtle)
```

Complete Code (2/2)

```
1  def getInput():
2      x = int(input("Enter x: "))
3      y = int(input("Enter y: "))
4      return(x,y)
5
6  def markLocation(t,x,y):
7      t.goto(x,y)
8      t.stamp()
9
10 if __name__ == "__main__":
11     main()
```

Challenge:

- Write a function that takes a number as an input and prints its corresponding name as a string.
- For example,
  - `num2string(0)` returns: `"zero"`
  - `num2string(1)` returns: `"one"`
  - `num2string(2)` returns: `"two"`
- You may assume that only single digits, 0,1,...,9, are given as input.

PythonTutor

- Starter code can be found on GitHub as numsConvert.py
- The `pythonTutor` link is under week 9 handouts (on course page) titled "num2string example"

Today's Topics

- Recap: Functions & Top Down Design
- **Mapping GIS Data**
- Random Numbers
- Indefinite Loops

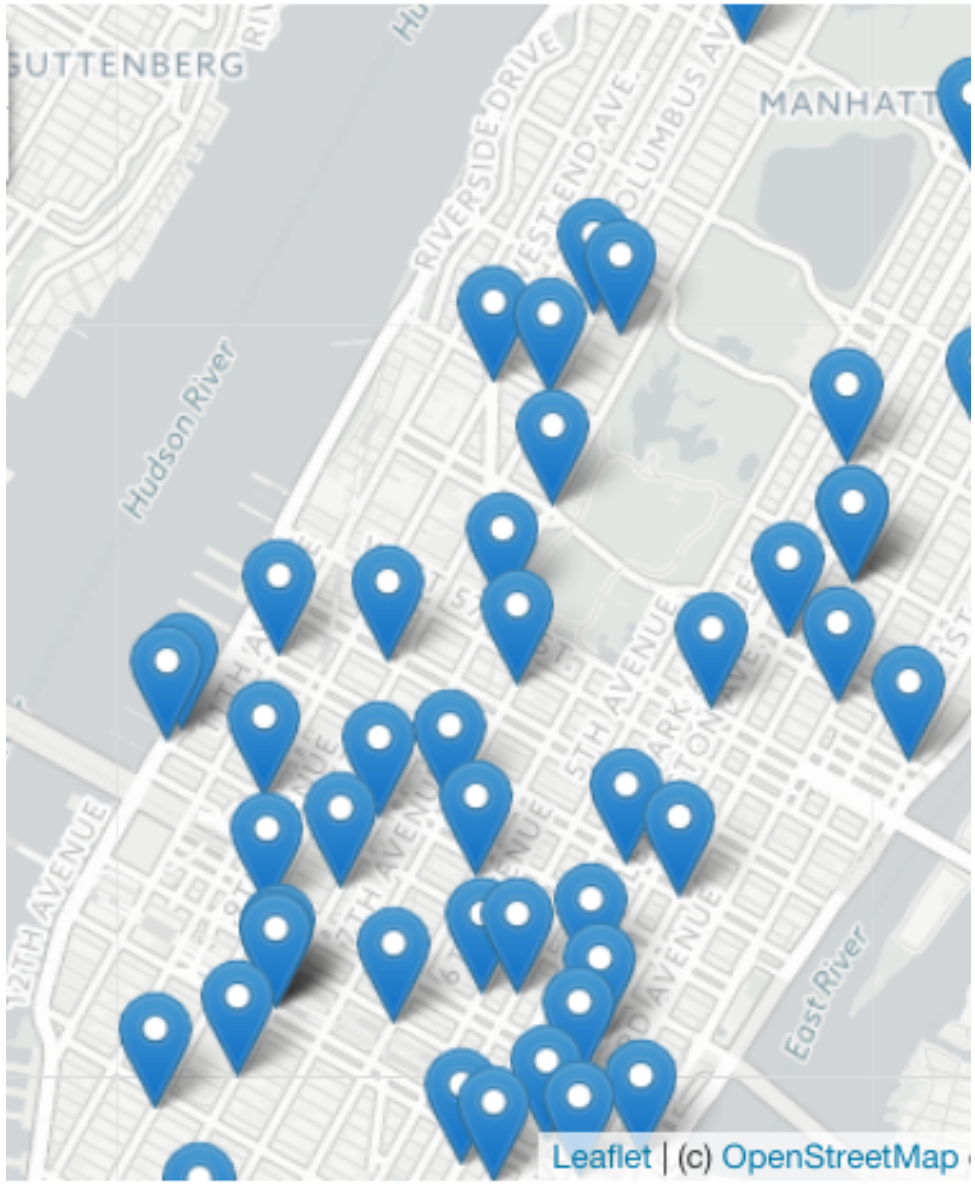
GIS Data

What is GIS data?

- A geographic information system (GIS) consists of integrated computer hardware and software that store, manage, analyze, edit, output, and visualize geographic data.

We can use a python library called `Folium` to access this kind of data and generate HTML files that display interactive maps when opened in a browser window.

Folium

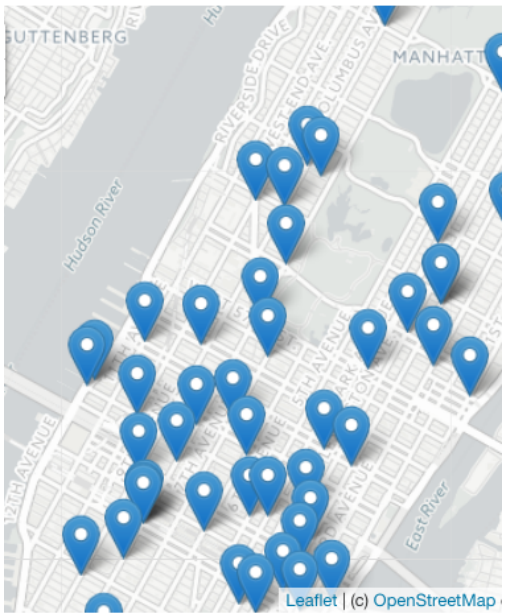

Hudson River
RIVERSIDE DRIVE
WEST END AVE
COLUMBUS AVE
5TH AVENUE
12TH AVENUE
East River
Leaflet | (c) OpenStreetMap

Folium

- A module for making HTML maps.
- It's a Python interface to the popular `leaflet.js`.
- Outputs `.html` files which you can open in a browser.
- The generated `.html` files will appear in the same folder as the program
- Process:

Write code. → *Run program.* → *Open .html in browser.*

Demo



Map created by `Folium`

- **Link** to interactive map

Folium

- To use:

  ```
  import folium
  ```

- Create a map:

  ```
  myMap = folium.Map()
  ```

- Make markers:

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

- Add to the map:

  ```
  newMark.add_to(myMap)
  ```

- Save the map to an HTML file:

  ```
  myMap.save(outfile=filename)
  ```

Folium

Example program using Folium:

```
1  import folium
2
3  #the location parameter is optional
4  #when supplied, the map will open to the given lat,lon
5  myMap = folium.Map(location=[40.71, -74.01])
6
7  #create a new marker that displays "NYC" at lat,lon
8  nycMarker = folium.Marker([40.71, -74.01],popup="NYC")
9
10 #add the marker to the map
11 nycMarker.add_to(myMap)
12
13 #save the map to an HTML file
14 myMap.save(outfile="nycMap.html")
```

Today's Topics

- Recap: Functions & Top Down Design
- Mapping GIS Data
- **Random Numbers**
- Indefinite Loops

Python's random package

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
  trey.forward(10)
  a = random.randrange(0,360,90)
  trey.right(a)
```

- Python has a built-in package for generating pseudo-random numbers.
- To use:

  `import random`

- Useful command to generate whole numbers:

  `random.randrange(start,stop,step)`

  which gives a number chosen randomly from the specified range.
- Useful command to generate real numbers:

  `random.random()`

  which gives a number chosen (uniformly) at random from [0.0,1.0).
- Very useful for simulations, games, and testing.

Python's random package

Link to example

```
1  import turtle
2  import random
3
4  trey = turtle.Turtle()
5  trey.speed(10)
6  for i in range(100):
7      trey.forward(10)
8      #Possible values for a: [0, 90, 180, 270]
9      a = random.randrange(0, 360, 90)
10     trey.right(a)
```

Today's Topics

- Recap: Functions & Top Down Design
- Mapping GIS Data
- Random Numbers
- **Indefinite Loops**

Challenge:

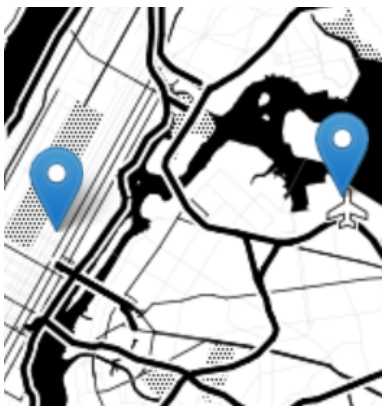
Predict what the code will do:

```
1 dist = int(input("Enter distance: "))
2 while dist < 0:
3     print("Distances cannot be negative.")
4     dist = int(input("Enter distance: "))
5
6 print("The distance entered is", dist)
```

Indefinite Loops

- Indefinite loops repeat as long as the condition is true.
- Could execute the body of the loop zero times, 10 times, infinite number of times.
- The condition determines how many times.
- Very useful for checking input, simulations, and games.
- More details next lecture...

Recap



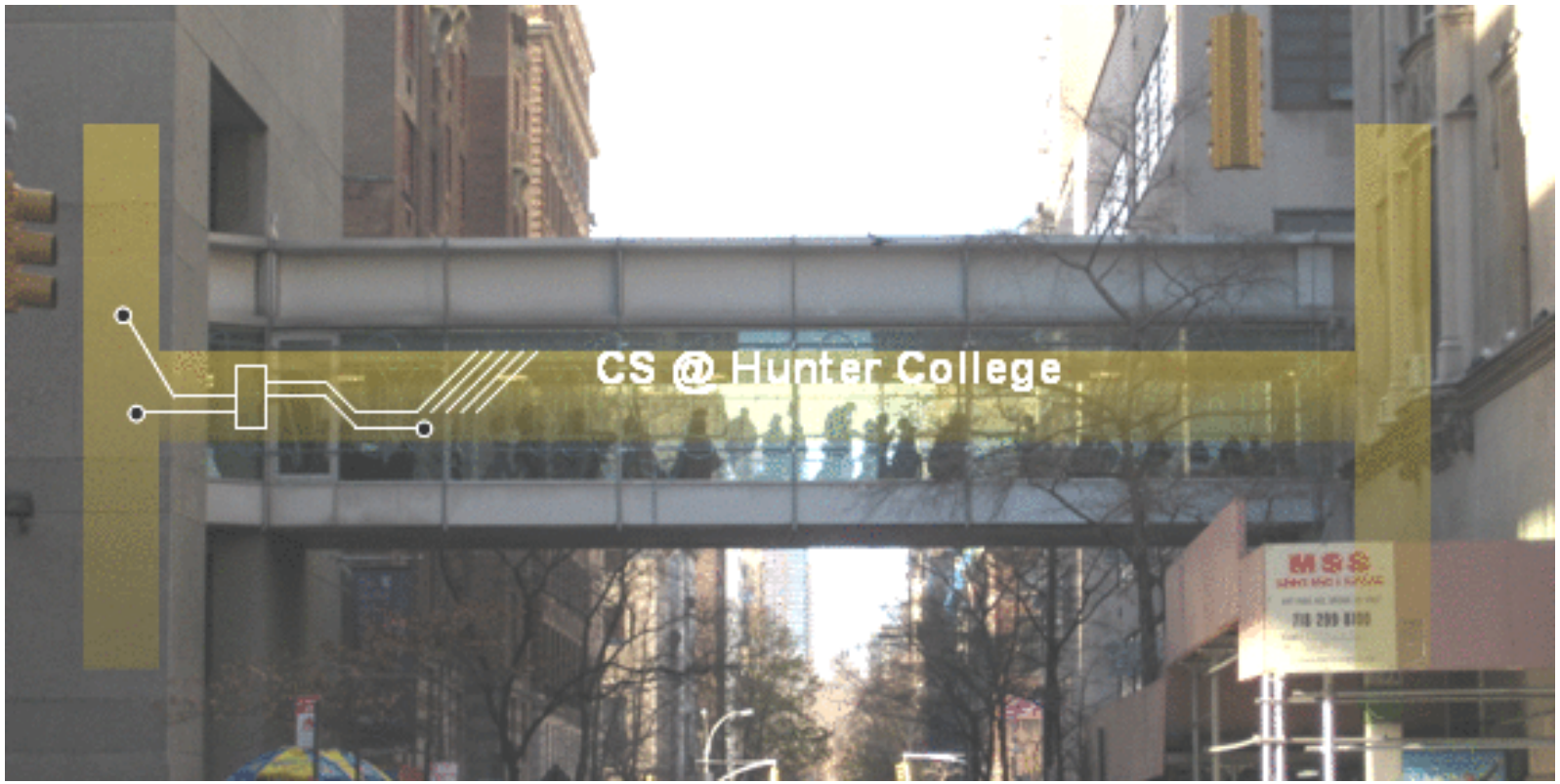
- Top-down design: breaking into subproblems, and implementing each part separately.
- Excellent approach: can then test each part separately before adding it to a large program.
- When possible, design so that your code is flexible to be reused ("code reuse").
- Introduced a Python library, `Folium` for creating interactive HTML maps.
- Introduced generating random numbers as well as using `while` loops for repeating commands for an indefinite number of times.

Halloween Challenge

This program demonstrates the use of while loops and random numbers!

Trick or Treat

Lecture Slips & Writing Boards



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