

CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Announcements



- Grades have been updated on Blackboard. Let us know if you see anything missing, so we can fix it (we found duplicate accounts and typos in EmplID's).
- Each lecture includes a survey of computing research and tech in NYC.

*Today: Prof. Susan Epstein
Machine Learning*

Today's Topics



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

Today's Topics



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

In Pairs or Triples:

```
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)
```

In Pairs or Triples:

```
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)
```

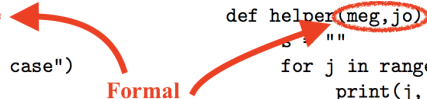
- What are the formal parameters for the functions?

In Pairs or Triples:

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

**Formal
Parameters**

```
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?

In Pairs or Triples:

```
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)
```

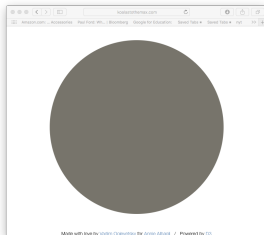

Python Tutor

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

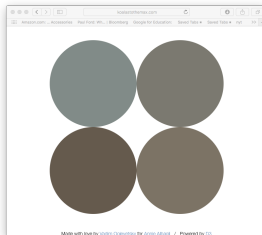
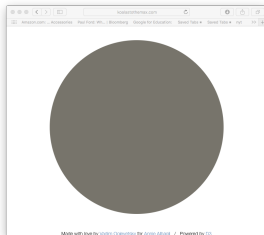
def helper(meg, jo):
    s = ""
    for j in range(meg):
        print(j, ": ", jo[j])
        if j % 2 == 0:
            s = s + jo[j]
        print("Building s:", s)
    return(s)
```

(Demo with pythonTutor)

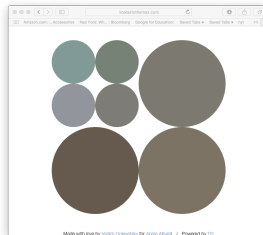
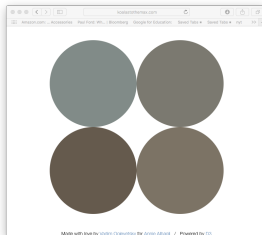
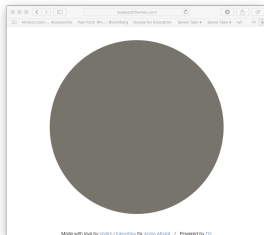
From Last Time: koalas



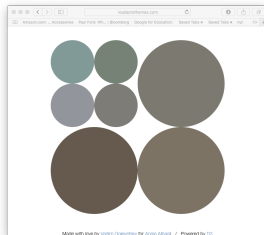
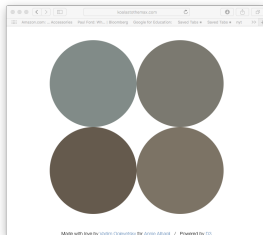
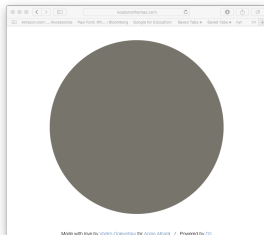
From Last Time: koalas



From Last Time: koalas

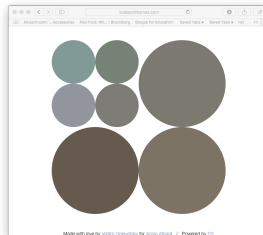
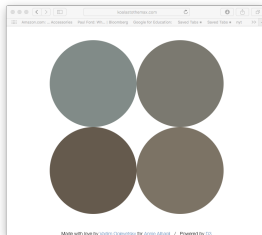
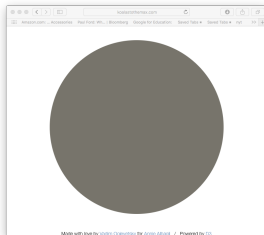


From Last Time: koalas



<http://koalastothemax.com>

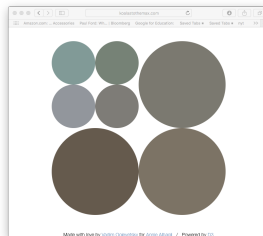
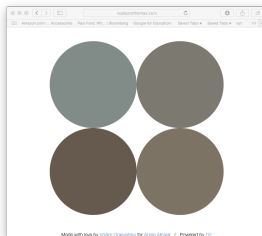
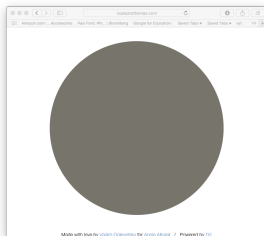
From Last Time: koalas



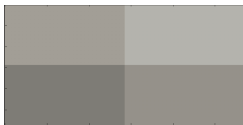
<http://koalastothemax.com>



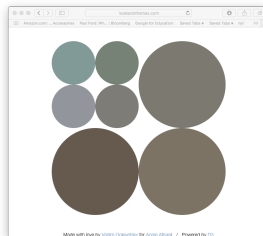
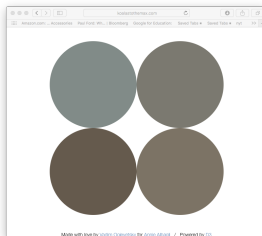
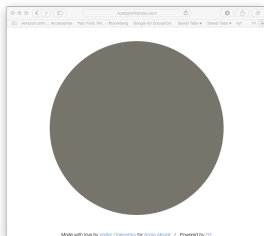
From Last Time: koalas



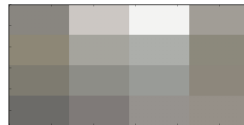
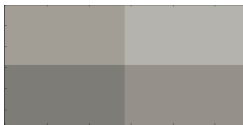
<http://koalastothemax.com>



From Last Time: koalas



<http://koalastothemax.com>



From Last Time: koalas

Process:



Get template
from github



Fill in missing
functions



Test locally
idle3/python3



Submit to
Gradescope

From Last Time: koalas



```
69 def main():
70     inFile = input('Enter image file name: ')
71     img = plt.imread(inFile)
72
73     #Divides the image in 1/2, 1/4, 1/8, ... 1/2^8, and displays each:
74     for i in range(8):
75         img2 = img.copy()    #Make a copy to average
76         quarter(img2,i)      #Split in half i times, and average regions
77
78         plt.imshow(img2)     #Load our new image into pyplot
79         plt.show()           #Show the image (waits until closed to continue)
80
81     #Shows the original image:
82     plt.imshow(img)          #Load image into pyplot
83     plt.show()               #Show the image (waits until closed to continue)
84
85
```

From Last Time: koalas



```
69 def main():
70     inFile = input('Enter image file name: ')
71     img = plt.imread(inFile)
72
73     #Divides the image in 1/2, 1/4, 1/8, ... 1/2^8, and displays each:
74     for i in range(8):
75         img2 = img.copy()    #Make a copy to average
76         quarter(img2,i)      #Split in half i times, and average regions
77
78         plt.imshow(img2)     #Load our new image into pyplot
79         plt.show()           #Show the image (waits until closed to continue)
80
81     #Shows the original image:
82     plt.imshow(img)          #Load image into pyplot
83     plt.show()               #Show the image (waits until closed to continue)
84
85
```

- The `main()` is written for you.

From Last Time: koalas



```
69 def main():
70     inFile = input('Enter image file name: ')
71     img = plt.imread(inFile)
72
73     #Divides the image in 1/2, 1/4, 1/8, ... 1/2^8, and displays each:
74     for i in range(8):
75         img2 = img.copy()    #Make a copy to average
76         quarter(img2,i)      #Split in half i times, and average regions
77
78         plt.imshow(img2)     #Load our new image into pyplot
79         plt.show()           #Show the image (waits until closed to continue)
80
81     #Shows the original image:
82     plt.imshow(img)          #Load image into pyplot
83     plt.show()               #Show the image (waits until closed to continue)
84
85
```

- The `main()` is written for you.
- Only fill in two functions: `average()` and `setRegion()`.

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.

In Pairs or Triples:

- Write the missing functions for the program:

```
def main():  
    tess = setUp()      #Returns a purple turtle with pen up.  
    for i in range(5):  
        x,y = getInput()    #Asks user for two numbers.  
        markLocation(tess,x,y) #Move tess to (x,y) and stamp.
```

In Pairs or Triples:

- Write the missing functions for the program:

```
def main():  
    tess = setUp()          #Returns a purple turtle with pen up.  
    for i in range(5):  
        x,y = getInput()      #Asks user for two numbers.  
        markLocation(tess,x,y) #Move tess to (x,y) and stamp.
```

Group Work: Fill in Missing Pieces

```
def main():  
    tess = setUp()      #Returns a purple turtle with pen up.  
    for i in range(5):  
        x,y = getInput()    #Asks user for two numbers.  
        markLocation(tess,x,y) #Move tess to (x,y) and stamp.
```

Group Work: Fill in Missing Pieces

- 1 Write import statements.

```
import turtle
```

```
def main():  
    tess = setUp()      #Returns a purple turtle with pen up.  
    for i in range(5):  
        x,y = getInput()    #Asks user for two numbers.  
        markLocation(tess,x,y) #Move tess to (x,y) and stamp.
```

Third Part: Fill in Missing Pieces

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

```
import turtle
def setUp():
    #FILL IN
def getInput():
    #FILL IN
def markLocation(t,x,y):
    #FILL IN

def main():
    tess = setUp()      #Returns a purple turtle with pen up.
    for i in range(5):
        x,y = getInput()      #Asks user for two numbers.
        markLocation(tess,x,y) #Move tess to (x,y) and stamp.
```

Third Part: Fill in Missing Pieces

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

```
import turtle

def setUp():
    #FILL IN
    return(newTurtle)

def getInput():
    #FILL IN
    return(x,y)

def markLocation(t,x,y):
    #FILL IN

def main():
    tess = setUp()      #Returns a purple turtle with pen up.
    for i in range(5):
        x,y = getInput()      #Asks user for two numbers.
        markLocation(tess,x,y) #Move tess to (x,y) and stamp.
```


Third Part: 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.

```
import turtle

def setUp():
    newTurtle = turtle.Turtle()
    newTurtle.penup()
    return(newTurtle)

def getInput():
    x = int(input('Enter x: '))
    y = int(input('Enter y: '))
    return(x,y)

def markLocation(t,x,y):
    t.goto(x,y)
    t.stamp()

def main():
    tess = setUp()      #Returns a purple turtle with pen up.
    for i in range(5):
        x,y = getInput()      #Asks user for two numbers.
```

In Pairs or Triples:

- Write a function that takes a number as an input and prints its corresponding name.

In Pairs or Triples:

- Write a function that takes a number as an input and prints its corresponding name.
- For example,

In Pairs or Triples:

- Write a function that takes a number as an input and prints its corresponding name.
- For example,
 - ▶ `num2string(0)` returns: zero

In Pairs or Triples:

- Write a function that takes a number as an input and prints its corresponding name.
- For example,
 - ▶ `num2string(0)` returns: zero
 - ▶ `num2string(1)` returns: one

In Pairs or Triples:

- Write a function that takes a number as an input and prints its corresponding name.
- For example,
 - ▶ `num2string(0)` returns: zero
 - ▶ `num2string(1)` returns: one
 - ▶ `num2string(2)` returns: two

In Pairs or Triples:

- Write a function that takes a number as an input and prints its corresponding name.
- 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.

Python Tutor



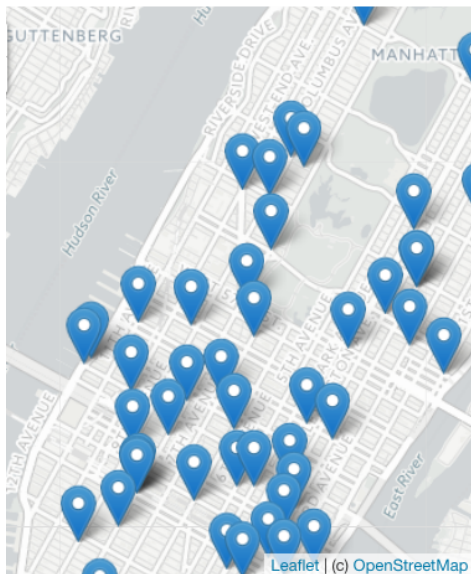
(On github)

Today's Topics



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

Folium



Folium

- A module for making HTML maps.

Folium



Folium

Folium



- A module for making HTML maps.
- It's a Python interface to the popular `leaflet.js`.

Folium

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.

Folium

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.
- An extra step:

Folium

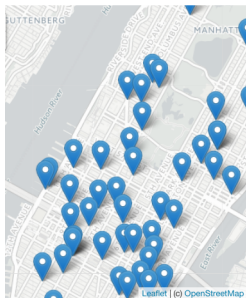
Folium



- 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.
- An extra step:

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

Demo



(Map created by Folium.)

Folium

- To use:
`import folium`

Folium



Folium

Folium



- To use:
`import folium`
- Create a map:
`myMap = folium.Map()`

Folium

Folium



- To use:
`import folium`
- Create a map:
`myMap = folium.Map()`
- Make markers:
`newMark = folium.Marker([lat,lon],popup=name)`

Folium

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)`

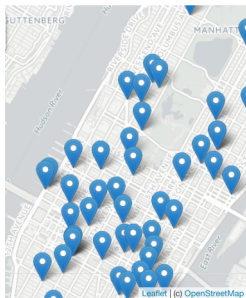
Folium

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)`
- Many options to customize background map ("tiles") and markers.

Demo



(Python program using Folium.)

In Pairs of Triples

- Predict which each line of code does:

```
m = folium.Map(  
    location=[45.372, -121.6972],  
    zoom_start=12,  
    tiles='Stamen Terrain'  
)  
  
folium.Marker(  
    location=[45.3288, -121.6625],  
    popup='Mt. Hood Meadows',  
    icon=folium.Icon(icon='cloud')  
) .add_to(m)  
  
folium.Marker(  
    location=[45.3311, -121.7113],  
    popup='Timberline Lodge',  
    icon=folium.Icon(color='green')  
) .add_to(m)  
  
folium.Marker(  
    location=[45.3300, -121.6823],  
    popup='Some Other Location',  
    icon=folium.Icon(color='red', icon='info-sign')  
) .add_to(m)
```



(example from Folium documentation)

Today's Topics



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

Python's random package

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

```
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's random package

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

```
import random
```

```
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's random package

- 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.

```
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's random package

- 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:

```
import turtle
import random

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

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

Python's random package

- 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 $[0.0, 1.0)$.

```
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's random package

- 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.

```
import turtle
import random

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

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

Trinket

```
import turtle
import random

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

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

(Demo turtle
random walk)

Today's Topics



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

In Pairs or Triples:

Predict what the code will do:

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

Python Tutor

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

(Demo with pythonTutor)

Indefinite Loops

- Indefinite loops repeat as long as the condition is true.

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

```
#Spring 2012 Final Exam, #8
nums = [1,4,0,6,5,2,9,8,12]
print(nums)
i=0
while i < len(nums)-1:
    if nums[i] < nums[i+1]:
        nums[i], nums[i+1] = nums[i+1], nums[i]
        i=i+1
print(nums)
```

Indefinite Loops

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

```
#Spring 2012 Final Exam, #8
nums = [1,4,0,6,5,2,9,8,12]
print(nums)
i=0
while i < len(nums)-1:
    if nums[i] < nums[i+1]:
        nums[i], nums[i+1] = nums[i+1], nums[i]
        i=i+1
print(nums)
```

- 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.

Indefinite Loops

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

```
#Spring 2012 Final Exam, #8
nums = [1,4,0,6,5,2,9,8,12]
print(nums)
i=0
while i < len(nums)-1:
    if nums[i] < nums[i+1]:
        nums[i], nums[i+1] = nums[i+1], nums[i]
        i=i+1
print(nums)
```

- 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.

Indefinite Loops

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

```
#Spring 2012 Final Exam, #8
nums = [1,4,0,6,5,2,9,8,12]
print(nums)
i=0
while i < len(nums)-1:
    if nums[i] < nums[i+1]:
        nums[i], nums[i+1] = nums[i+1], nums[i]
        i=i+1
print(nums)
```

- 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.

Indefinite Loops

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))
print('The distance entered is', dist)
```

```
#Spring 2012 Final Exam, #8
nums = [1,4,0,6,5,2,9,8,12]
print(nums)
i=0
while i < len(nums)-1:
    if nums[i] < nums[i+1]:
        nums[i], nums[i+1] = nums[i+1], nums[i]
        i=i+1
print(nums)
```

- 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...

Today's Topics



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

CS Survey Talk

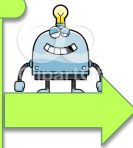


Prof. Susan Epstein
(Machine Learning)

Computational agents

- **Computational system** implements decisions and actions on a physical device
- A **computational agent** executes a perpetual **sense-decide-act loop**

Do forever
 Sense the world
 Select an action
 Execute that action



- How to **sense** the world: infrared sonar radar Kinect
microphone camera
- Given a set of possible actions, an **agent decides** by selecting one



Spring 2019

CSCI 127

1/6

Artificial intelligence (AI)

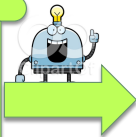
- An AI agent doesn't have to be a **robot** (embodied in the world)
- An AI agent doesn't have to be **autonomous** (make decisions entirely on its own)
- But it does have to be smart...
- That means it has to make smart decisions
- **Artificial intelligence** = **simulation** of intelligent (**human**) **behavior** by a computational agent

Do forever

Sense the world

Select an action

Execute that action



Nest

- Controlled by Wi-Fi from a smartphone
- Reprograms itself based on human behavior

Spring 2019

CSCI 127

2/6

What AI does

- Tackles hard, interesting problems
 - Does this image show cancer?
 - Should I move this car through the intersection?
 - How do I get to that concert?
- Builds models of perception, thinking, and action
- Uses these models to build smarter programs



Our autonomous robot navigators

- Despite uncertainty, noise, and constant changes in the world
- Learn models of their environment
- Make smart decisions with those models

Spring 2019

CSCI 127

3/6

How our robots navigate

- We built **SemaFORR**, a robot controller that makes decisions autonomously
- First the robots learn to travel by **building a model of the world** we put them in
- Then they prove they **can find both hard and easy targets** there
- Apollo has already done this on a small part of the 10th floor here
- And in **simulation** ROSie has traveled
 - Through much of Hunter, The Graduate Center, and MOMA
 - Through moving crowds of people
 - Without collision and without coming too close to people
 - And **explained her behavior** in natural language

Spring 2019

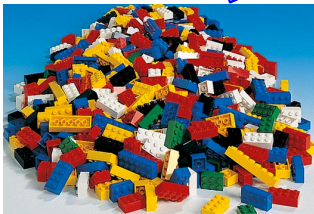
CSCI 127

4/6

How to build an intelligent agent

- Find good problems
- Start simple
- Run lots of experiments
- Analyze the results carefully
- ...and repeat

Fun problems
Good reasons
Learning algorithms



Spring 2019



CSCI 127



5/6

Want to know more?

- Fall 2019: SCI 111 Brains, Minds, and Machines = cognitive neuroscience + cognitive psychology + AI
- Fall 2019: CSCI 353 Machine Learning
- 2021: CSCI 350 Artificial Intelligence
- ...and then there's my lab, where workstations run 24/7, learning to be intelligent agents

Susan Epstein, Professor of Computer Science
1090C Hunter North
susan.epstein@hunter.cuny.edu
<http://www.cs.hunter.cuny.edu/~epstein/>

Spring 2019

CSCI 127

6/6

Recap

- On lecture slip, write down a topic you wish we had spent more time (and why).



Recap

- On lecture slip, write down a topic you wish we had spent more time (and why).
- Top-down design: breaking into subproblems, and implementing each part separately.



Recap

- On lecture slip, write down a topic you wish we had spent more time (and why).
- 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.



Recap



- On lecture slip, write down a topic you wish we had spent more time (and why).
- 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”).

Recap



- On lecture slip, write down a topic you wish we had spent more time (and why).
- 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.

Recap



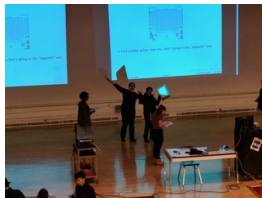
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Top-down design: breaking into subproblems, and implementing each part separately.
- Excellent approach: can then test each part separately before adding it to a large program.
- When possible, design so that your code is flexible to be reused (“code reuse”).
- Introduced a Python library, Folium for creating interactive HTML maps.
- Introduced `while` loops for repeating commands for an indefinite number of times.

Recap



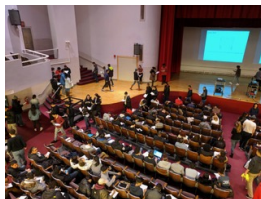
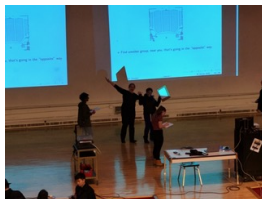
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Top-down design: breaking into subproblems, and implementing each part separately.
- Excellent approach: can then test each part separately before adding it to a large program.
- When possible, design so that your code is flexible to be reused (“code reuse”).
- Introduced a Python library, Folium for creating interactive HTML maps.
- Introduced `while` loops for repeating commands for an indefinite number of times.
- Pass your lecture slips to the aisles for the UTAs to collect.

Practice Quiz & Final Questions



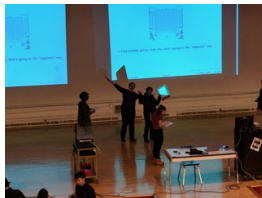
- Lightning rounds:

Practice Quiz & Final Questions



- Lightning rounds:
 - ▶ write as much you can for 60 seconds;

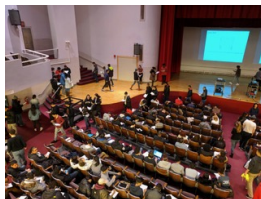
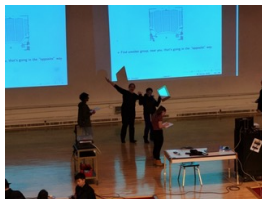
Practice Quiz & Final Questions



- Lightning rounds:

- ▶ write as much you can for 60 seconds;
- ▶ followed by answer; and

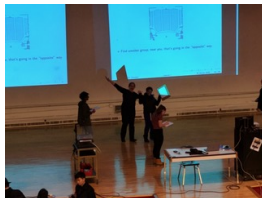
Practice Quiz & Final Questions



- Lightning rounds:

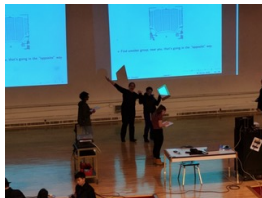
- ▶ write as much you can for 60 seconds;
- ▶ followed by answer; and
- ▶ repeat.

Practice Quiz & Final Questions



- 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](#)).

Practice Quiz & Final Questions



- 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 & Top-Down Design (Summer 18, #7 & #5).