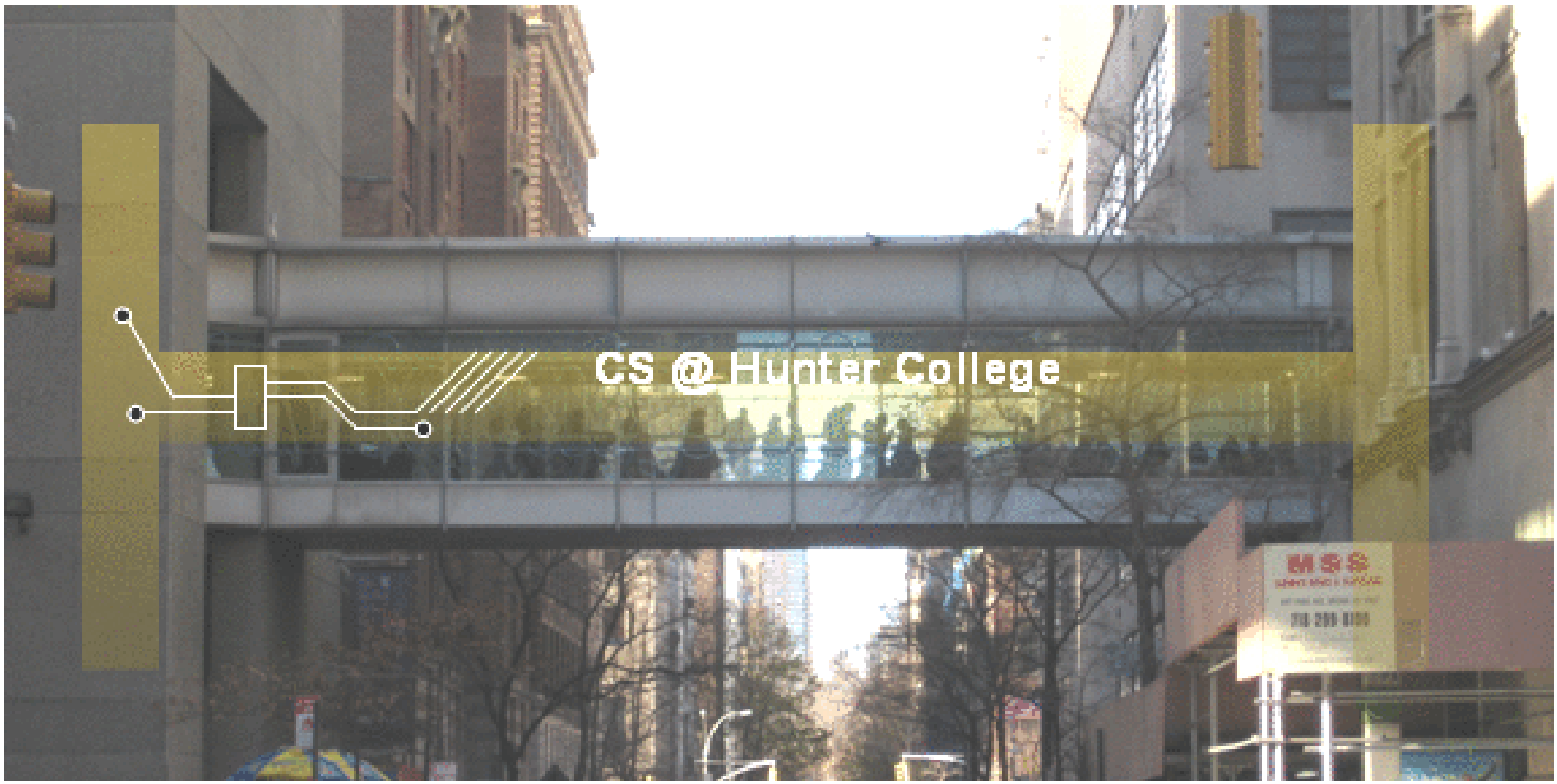
CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Announcements

- Each lecture includes a survey of computing research and tech in NYC.

Announcements

- Each lecture includes a survey of computing research and tech in NYC.

  Today: Adrienne Schmoeker & Albert Webber
  NYC OpenData Initiative
  Mayor's Office

Announcements

- Each lecture includes a survey of computing research and tech in NYC.

  Today: Adrienne Schmoeker & Albert Webber
  NYC OpenData Initiative
  Mayor's Office

- 11:10am: Informal Q&A with Adrienne & Albert in 1203 Hunter East.

Frequently Asked Questions

From lecture slips & recitation sections.

Frequently Asked Questions

From lecture slips & recitation sections.

- I didn't get the torus-based islands & pools from last time!

Frequently Asked Questions

From lecture slips & recitation sections.

- I didn't get the torus-based islands & pools from last time!
  No worries– we'll talk about it first.

Frequently Asked Questions

From lecture slips & recitation sections.

- I didn't get the torus-based islands & pools from last time!
  No worries– we'll talk about it first.
- Why do we have design questions (like the torus-land)?

Frequently Asked Questions

From lecture slips & recitation sections.

- I didn't get the torus-based islands & pools from last time!
  No worries– we'll talk about it first.
- Why do we have design questions (like the torus-land)?
  The design questions cover two of the course's learning objectives: exposure to advanced computer science topics & problem solving.

Frequently Asked Questions

From lecture slips & recitation sections.

- I didn't get the torus-based islands & pools from last time!
  No worries– we'll talk about it first.
- Why do we have design questions (like the torus-land)?
  The design questions cover two of the course's learning objectives: exposure to advanced computer science topics & problem solving.
- Please, more time on circuits/logical expressions/truth tables/decisions!

Frequently Asked Questions

From lecture slips & recitation sections.

- I didn't get the torus-based islands & pools from last time!
  No worries– we'll talk about it first.
- Why do we have design questions (like the torus-land)?
  The design questions cover two of the course's learning objectives: exposure to advanced computer science topics & problem solving.
- Please, more time on circuits/logical expressions/truth tables/decisions!
  We will do a bit today, but much more in the following weeks.

Frequently Asked Questions

From lecture slips & recitation sections.

- I didn't get the torus-based islands & pools from last time!
  No worries– we'll talk about it first.
- Why do we have design questions (like the torus-land)?
  The design questions cover two of the course's learning objectives: exposure to advanced computer science topics & problem solving.
- Please, more time on circuits/logical expressions/truth tables/decisions!
  We will do a bit today, but much more in the following weeks.
- I turned in a lecture slip, but it's not showing up on Gradescope.

Frequently Asked Questions

From lecture slips & recitation sections.

- I didn't get the torus-based islands & pools from last time!
  No worries– we'll talk about it first.
- Why do we have design questions (like the torus-land)?
  The design questions cover two of the course's learning objectives: exposure to advanced computer science topics & problem solving.
- Please, more time on circuits/logical expressions/truth tables/decisions!
  We will do a bit today, but much more in the following weeks.
- I turned in a lecture slip, but it's not showing up on Gradescope.
  We always have a few with no name, only the first name, or are hard to read.

Frequently Asked Questions

From lecture slips & recitation sections.

- I didn't get the torus-based islands & pools from last time!
  No worries– we'll talk about it first.
- Why do we have design questions (like the torus-land)?
  The design questions cover two of the course's learning objectives: exposure to advanced computer science topics & problem solving.
- Please, more time on circuits/logical expressions/truth tables/decisions!
  We will do a bit today, but much more in the following weeks.
- I turned in a lecture slip, but it's not showing up on Gradescope.
  We always have a few with no name, only the first name, or are hard to read. Send us email with the lecture number and your name, and we'll search for it.

Frequently Asked Questions

From lecture slips & recitation sections.

- I didn't get the torus-based islands & pools from last time!
  No worries– we'll talk about it first.
- Why do we have design questions (like the torus-land)?
  The design questions cover two of the course's learning objectives: exposure to advanced computer science topics & problem solving.
- Please, more time on circuits/logical expressions/truth tables/decisions!
  We will do a bit today, but much more in the following weeks.
- I turned in a lecture slip, but it's not showing up on Gradescope.
  We always have a few with no name, only the first name, or are hard to read. Send us email with the lecture number and your name, and we'll search for it. p.s. Including your EmplID (& printing your name) makes it much easier.

Frequently Asked Questions

From lecture slips & recitation sections.

- I didn't get the torus-based islands & pools from last time!
  No worries– we'll talk about it first.
- Why do we have design questions (like the torus-land)?
  The design questions cover two of the course's learning objectives: exposure to advanced computer science topics & problem solving.
- Please, more time on circuits/logical expressions/truth tables/decisions!
  We will do a bit today, but much more in the following weeks.
- I turned in a lecture slip, but it's not showing up on Gradescope.
  We always have a few with no name, only the first name, or are hard to read. Send us email with the lecture number and your name, and we'll search for it. p.s. Including your EmplID (& printing your name) makes it much easier.
- For the intrepid few that keep asking: When are you covering recursion?

Frequently Asked Questions

From lecture slips & recitation sections.

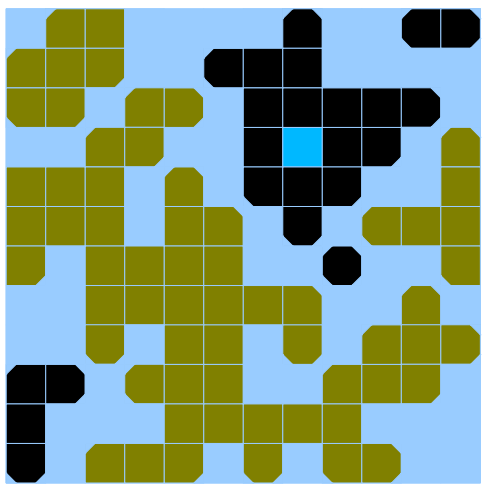
- I didn't get the torus-based islands & pools from last time!
  No worries– we'll talk about it first.
- Why do we have design questions (like the torus-land)?
  The design questions cover two of the course's learning objectives: exposure to advanced computer science topics & problem solving.
- Please, more time on circuits/logical expressions/truth tables/decisions!
  We will do a bit today, but much more in the following weeks.
- I turned in a lecture slip, but it's not showing up on Gradescope.
  We always have a few with no name, only the first name, or are hard to read. Send us email with the lecture number and your name, and we'll search for it. p.s. Including your EmplID (& printing your name) makes it much easier.
- For the intrepid few that keep asking: When are you covering recursion?
  When we cover functions. See today and next week's lecture, and Program #40.

Today's Topics

- Recap: Prof. Saad's torus-land
- Introduction to Functions
- NYC Open Data

Recap: Living on a Torus

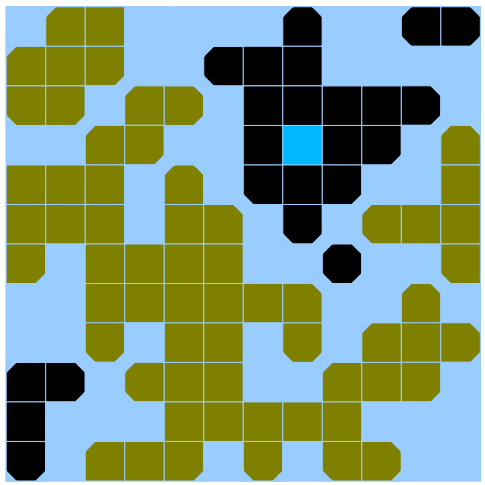
A random torus with
5 islands and 2 pools



- How many pools and how many islands does each version have? (Collect all 5!)

Recap: Living on a Torus

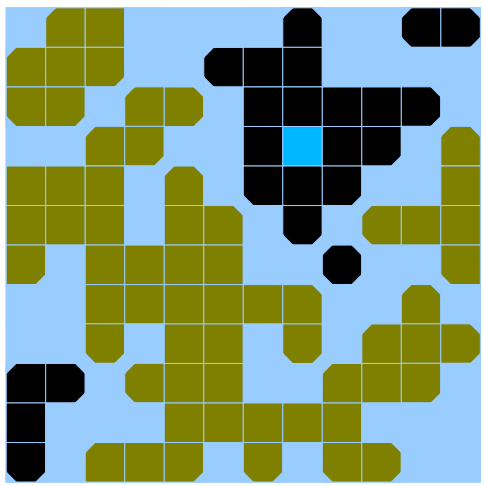
A random torus with
5 islands and 2 pools



- How many pools and how many islands does each version have? (Collect all 5!)
- Design an algorithm that will count the number of islands.

Recap: Count the number of islands.

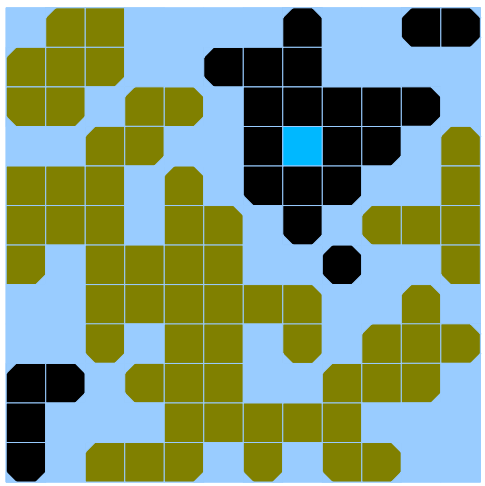
A random torus with
5 islands and 2 pools



Recap: Count the number of islands.

- **Input:**

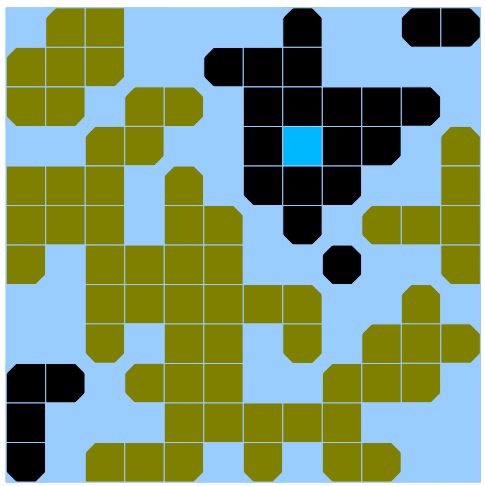
A random torus with
5 islands and 2 pools



Recap: Count the number of islands.

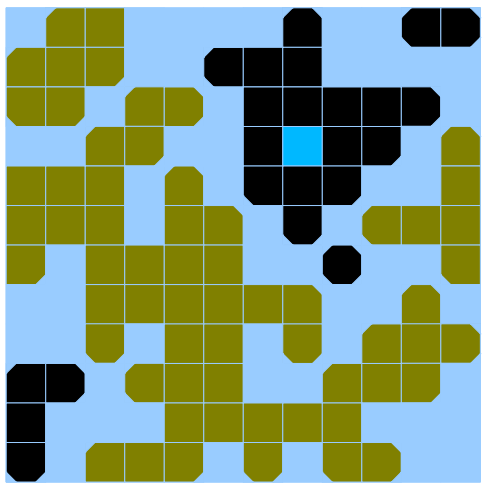
- **Input:** A 2D grid (array) filled with shaded and blank squares.

A random torus with
5 islands and 2 pools



Recap: Count the number of islands.

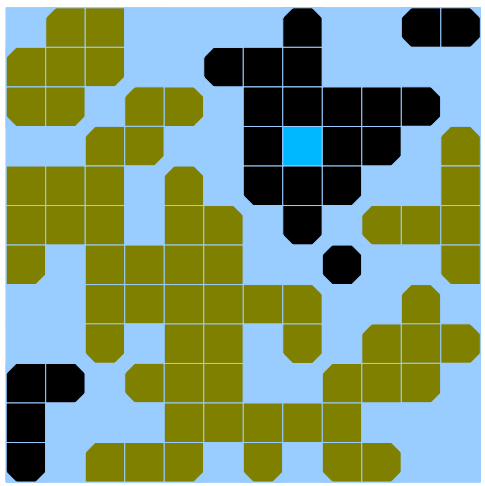
A random torus with
5 islands and 2 pools



- **Input:** A 2D grid (array) filled with shaded and blank squares.
- **Output:**

Recap: Count the number of islands.

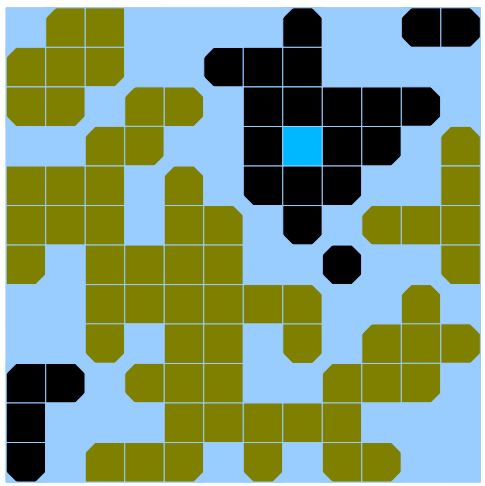
A random torus with
5 islands and 2 pools



- **Input:** A 2D grid (array) filled with shaded and blank squares.
- **Output:** The number of islands.

Recap: Count the number of islands.

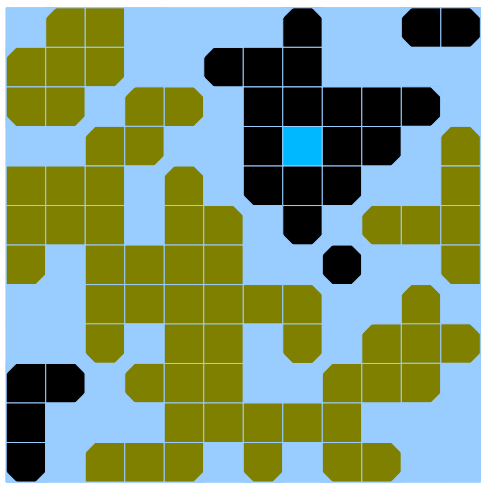
A random torus with
5 islands and 2 pools



- **Input:** A 2D grid (array) filled with shaded and blank squares.
- **Output:** The number of islands.
- **Design:**

Recap: Count the number of islands.

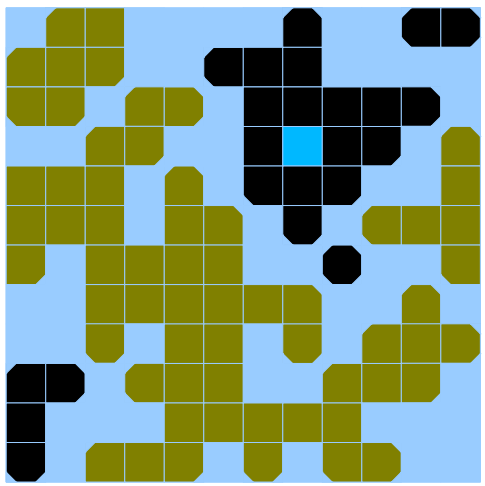
A random torus with
5 islands and 2 pools



- **Input:** A 2D grid (array) filled with shaded and blank squares.
- **Output:** The number of islands.
- **Design:** Lots of ways to do this. Here's one:

Recap: Count the number of islands.

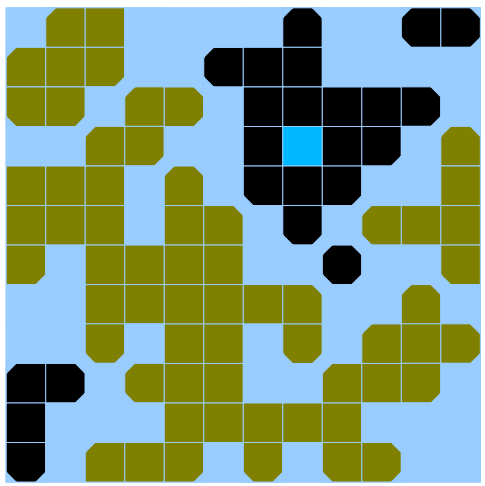
A random torus with
5 islands and 2 pools



- **Input:** A 2D grid (array) filled with shaded and blank squares.
- **Output:** The number of islands.
- **Design:** Lots of ways to do this. Here's one:
  - Set up a variable `count` to hold the number of islands (set to 0).

Recap: Count the number of islands.

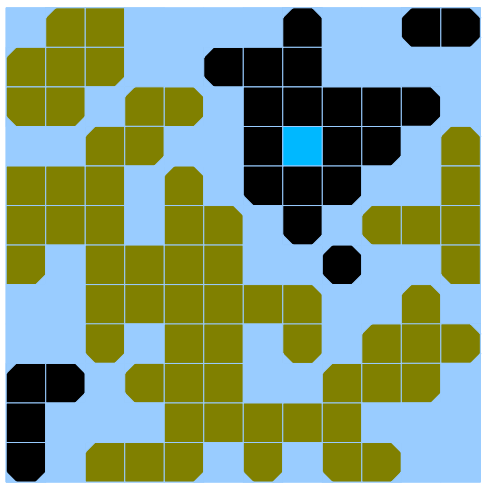
A random torus with
5 islands and 2 pools



- **Input:** A 2D grid (array) filled with shaded and blank squares.
- **Output:** The number of islands.
- **Design:** Lots of ways to do this. Here's one:
  - Set up a variable `count` to hold the number of islands (set to 0).
  - For each square (start in upper left corner). If it's shaded & unvisited:

Recap: Count the number of islands.

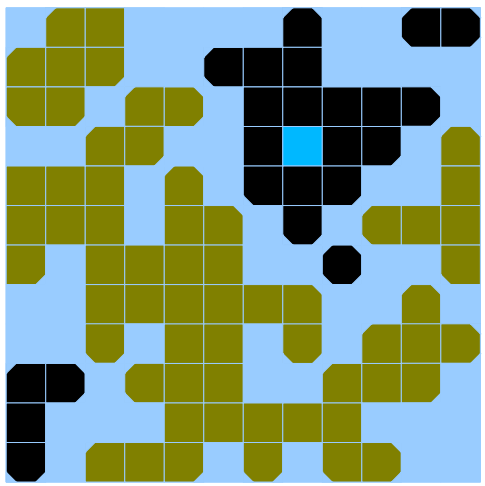
A random torus with
5 islands and 2 pools



- **Input:** A 2D grid (array) filled with shaded and blank squares.
- **Output:** The number of islands.
- **Design:** Lots of ways to do this. Here's one:
  - Set up a variable `count` to hold the number of islands (set to 0).
  - For each square (start in upper left corner). If it's shaded & unvisited:
    - ★ Add one to the `count` of islands.

Recap: Count the number of islands.

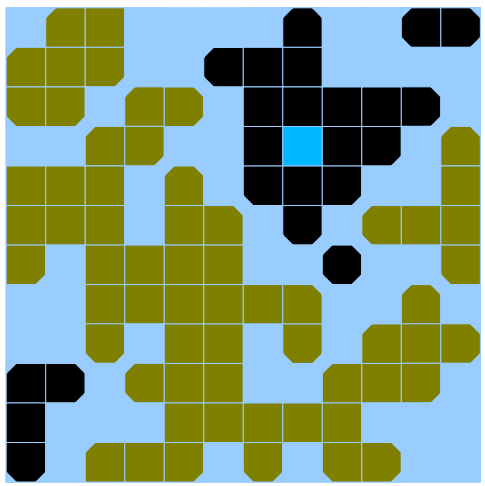
A random torus with
5 islands and 2 pools



- **Input:** A 2D grid (array) filled with shaded and blank squares.
- **Output:** The number of islands.
- **Design:** Lots of ways to do this. Here's one:
  - Set up a variable `count` to hold the number of islands (set to 0).
  - For each square (start in upper left corner). If it's shaded & unvisited:
    - ★ Add one to the `count` of islands.
    - ★ Mark it as visited (color it purple).

Recap: Count the number of islands.

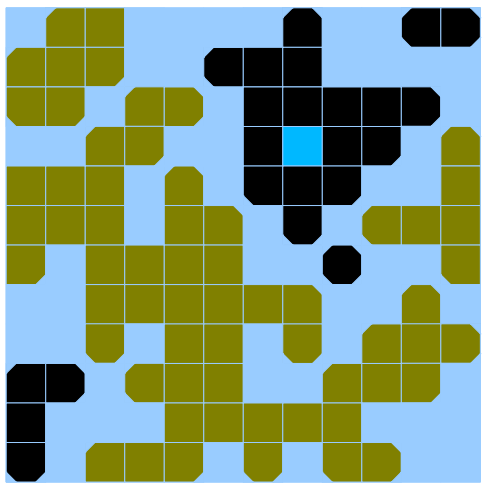
A random torus with
5 islands and 2 pools



- **Input:** A 2D grid (array) filled with shaded and blank squares.
- **Output:** The number of islands.
- **Design:** Lots of ways to do this. Here's one:
  - Set up a variable `count` to hold the number of islands (set to 0).
  - For each square (start in upper left corner). If it's shaded & unvisited:
    - ★ Add one to the `count` of islands.
    - ★ Mark it as visited (color it purple).
    - ★ If it has any neighbors that are land, mark those as purple.

Recap: Count the number of islands.

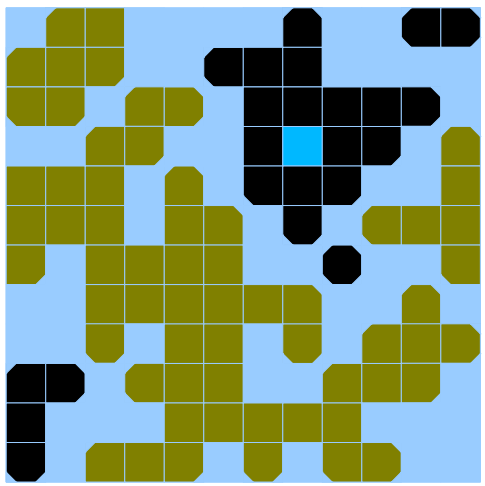
A random torus with
5 islands and 2 pools



- **Input:** A 2D grid (array) filled with shaded and blank squares.
- **Output:** The number of islands.
- **Design:** Lots of ways to do this. Here's one:
  - Set up a variable `count` to hold the number of islands (set to 0).
  - For each square (start in upper left corner). If it's shaded & unvisited:
    - ★ Add one to the `count` of islands.
    - ★ Mark it as visited (color it purple).
    - ★ If it has any neighbors that are land, mark those as purple.
    - ★ Keep checking neighbors, until all are marked.

Recap: Count the number of islands.

A random torus with
5 islands and 2 pools



- **Input:** A 2D grid (array) filled with shaded and blank squares.
- **Output:** The number of islands.
- **Design:** Lots of ways to do this. Here's one:
  - Set up a variable `count` to hold the number of islands (set to 0).
  - For each square (start in upper left corner). If it's shaded & unvisited:
    - ★ Add one to the `count` of islands.
    - ★ Mark it as visited (color it purple).
    - ★ If it has any neighbors that are land, mark those as purple.
    - ★ Keep checking neighbors, until all are marked.
  - Return `count`.

Functions

```
#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

```
#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.
- Many languages require that all code must be organized with functions.

Functions

```
#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.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`

Functions

```
#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.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:

Functions

```
#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.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis: Example: `print("Hello", "World")`

Functions

```
#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.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis: Example: `print("Hello", "World")`
- Can write, or **define** your own functions,

Functions

```
#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.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis: Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.

"Hello, World!" with Functions

```
#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()
```

Python Tutor

```
#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()
```

(Demo with pythonTutor)

In Pairs or Triples:

Predict what the code will do:

```
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)
```

Python Tutor

```
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)
```

(Demo with `pythonTutor`)

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)
```

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).

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**.

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**

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.

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 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.

In Pairs or Triples:

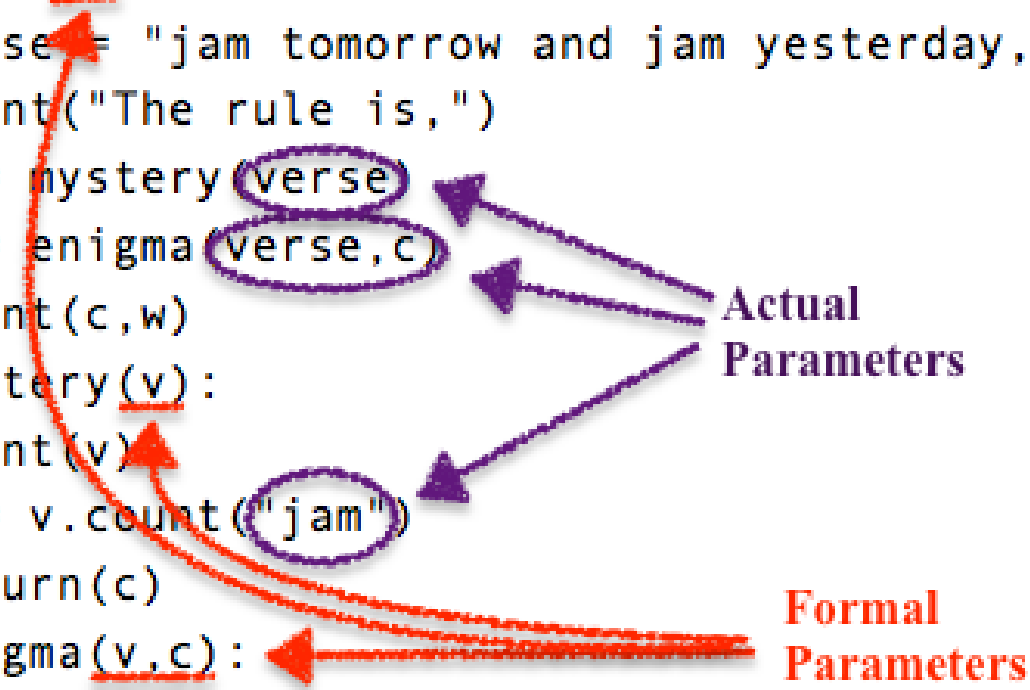
Circle the actual parameters and underline the formal parameters:

```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

In Pairs or Triples:

Circle the actual parameters and underline the formal parameters:

```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```



In Pairs or Triples:

Predict what the code will do:

```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

```
#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] )
```

Python Tutor

```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

```
#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`)

In Pairs or Triples:

Predict what the code will do:

```
#Greet loop example

def greetLoop(person):
    print("Greetings")
    for i in range(5):
        print("Hello", person)

greetLoop("Thomas")
```

```
# From "Teaching with Python" by John Zelle

def happy():
    print("Happy Birthday to you!")

def sing(P):
    happy()
    happy()
    print("Happy Birthday dear " + P + "!")
    happy()


sing("Fred")
sing("Thomas")
sing("Hunter")
```

Python Tutor

```
#Greet loop example

def greetLoop(person):
    print("Greetings")
    for i in range(5):
        print("Hello", person)

greetLoop("Thomas")
```

```
# From "Teaching with Python" by John Zelle

def happy():
    print("Happy Birthday to you!")

def sing(P):
    happy()
    happy()
    print("Happy Birthday dear " + P + "!")
    happy()


sing("Fred")
sing("Thomas")
sing("Hunter")
```

(Demo with `pythonTutor`)

In Pairs or Triples:

Fill in the missing code:

```
def monthString(monthNum):
    """
    Takes as input a number, monthNum, and
    returns the corresponding month name as a string.
    Example: monthString(1) returns "January".
    Assumes that input is an integer ranging from 1 to 12
    """

    monthString = ""

    ###################################
    ### FILL IN YOUR CODE HERE      ###
    ### Other than your name above, ###
    ### this is the only section    ###
    ### you change in this program. ###
    ###################################

    return(monthString)

def main():
    n = int(input('Enter the number of the month: '))
    mString = monthString(n)
    print('The month is', mString)
```

IDLE

```
def monthString(monthNum):
    """
    Takes as input a number, monthNum, and
    returns the corresponding month name as a string.
    Example: monthString(1) returns "January".
    Assumes that input is an integer ranging from 1 to 12
    """

    monthString = ""

    ###################################
    ### FILL IN YOUR CODE HERE      ###
    ### Other than your name above, ###
    ### this is the only section    ###
    ### you change in this program. ###
    ###################################

    return(monthString)

def main():
    n = int(input('Enter the number of the month: '))
    mString = monthString(n)
    print('The month is', mString)
```

(Demo with IDLE)

In Pairs or Triples:

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)

Recap: Functions

```
#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.

Recap: Functions

```
#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.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:

Recap: Functions

```
#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.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis: Example: `print("Hello", "World")`

Recap: Functions

```
#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.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis: Example: `print("Hello", "World")`
- Can write, or **define** your own functions,

Recap: Functions

```
#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.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis: Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.

Accessing Structured Data: NYC Open Data

- Freely available source of data.

Accessing Structured Data: NYC Open Data

- Freely available source of data.
- Maintained by the NYC data analytics team.

Accessing Structured Data: NYC Open Data

- Freely available source of data.
- Maintained by the NYC data analytics team.
- We will use several different ones for this class.

Accessing Structured Data: NYC Open Data

- Freely available source of data.
- Maintained by the NYC data analytics team.
- We will use several different ones for this class.
- Will use pandas, pyplot & folium libraries to analyze, visualize and map the data.

Accessing Structured Data: NYC Open Data

- Freely available source of data.
- Maintained by the NYC data analytics team.
- We will use several different ones for this class.
- Will use pandas, pyplot & folium libraries to analyze, visualize and map the data.
- Lab 7 covers accessing and downloading NYC OpenData datasets.

Example: Reading in CSV Files

```
Source:  https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,,
All population figures are consistent with present-day boundaries.,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,,
,,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,5346,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813669,279122,32903,23593,25492,1174779
1870,942292,419921,45468,37393,33029,1478103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,152999,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018356,469042,732016,116531,5620048
1930,1867312,2560401,1079129,1265258,158346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550849,1451277,191555,7891957
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1585873,2504700,2230722,1385108,468730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

`nycHistPop.csv`

In Lab 6

Example: Reading in CSV Files

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
Source:  https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,,
All population figures are consistent with present-day boundaries.,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,,
,,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,5346,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813669,279122,32903,23593,25492,1174779
1870,942292,419921,45468,37393,33029,1478103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,152999,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018356,469042,732016,116531,5620048
1930,1867312,2560401,1079129,1265258,158346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550849,1451277,191555,7891957
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1585873,2504700,2230722,1385108,468730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

`nycHistPop.csv`

In Lab 6

Example: Reading in CSV Files

```
import matplotlib.pyplot as plt
import pandas as pd

pop = pd.read_csv('nycHistPop.csv',skiprows=5)
```

```
Source:  https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,,
All population figures are consistent with present-day boundaries.,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,,
,,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,5346,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813669,279122,32903,23593,25492,1174779
1870,942292,419921,45468,37393,33029,1478103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,152999,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018356,469042,732016,116531,5620048
1930,1867312,2560401,1079129,1265258,158346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550849,1451277,191555,7891957
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1585873,2504700,2230722,1385108,468730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

nycHistPop.csv

In Lab 6

Example: Reading in CSV Files

```
import matplotlib.pyplot as plt
import pandas as pd

pop = pd.read_csv('nycHistPop.csv',skiprows=5)

pop.plot(x="Year")
plt.show()
```

```
Source:  https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,,
All population figures are consistent with present-day boundaries.,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,,
,,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,5346,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813669,279122,32903,23593,25492,1174779
1870,942292,419921,45468,37393,33029,1478103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,152999,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018356,469042,732016,116531,5620048
1930,1867312,2560401,1079129,1265258,158346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550849,1451277,191555,7891957
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1585873,2504700,2230722,1385108,468730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

nycHistPop.csv

In Lab 6

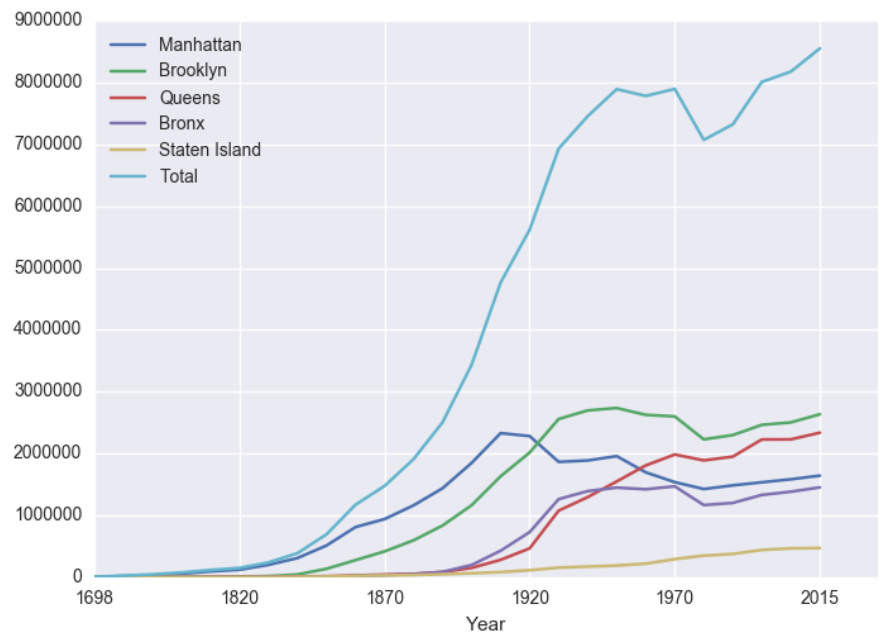
Example: Reading in CSV Files

```
import matplotlib.pyplot as plt
import pandas as pd

pop = pd.read_csv('nycHistPop.csv',skiprows=5)

pop.plot(x="Year")
plt.show()
```

```
Source:  https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,,
All population figures are consistent with present-day boundaries.,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,,
,,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,5346,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813669,279122,32903,23593,25492,1174779
1870,942292,419921,45468,37393,33029,1478103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,152999,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018356,469042,732016,116531,5620048
1930,1867312,2560401,1079129,1265258,158346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550849,1451277,191555,7891957
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1585873,2504700,2230722,1385108,468730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

nycHistPop.csv

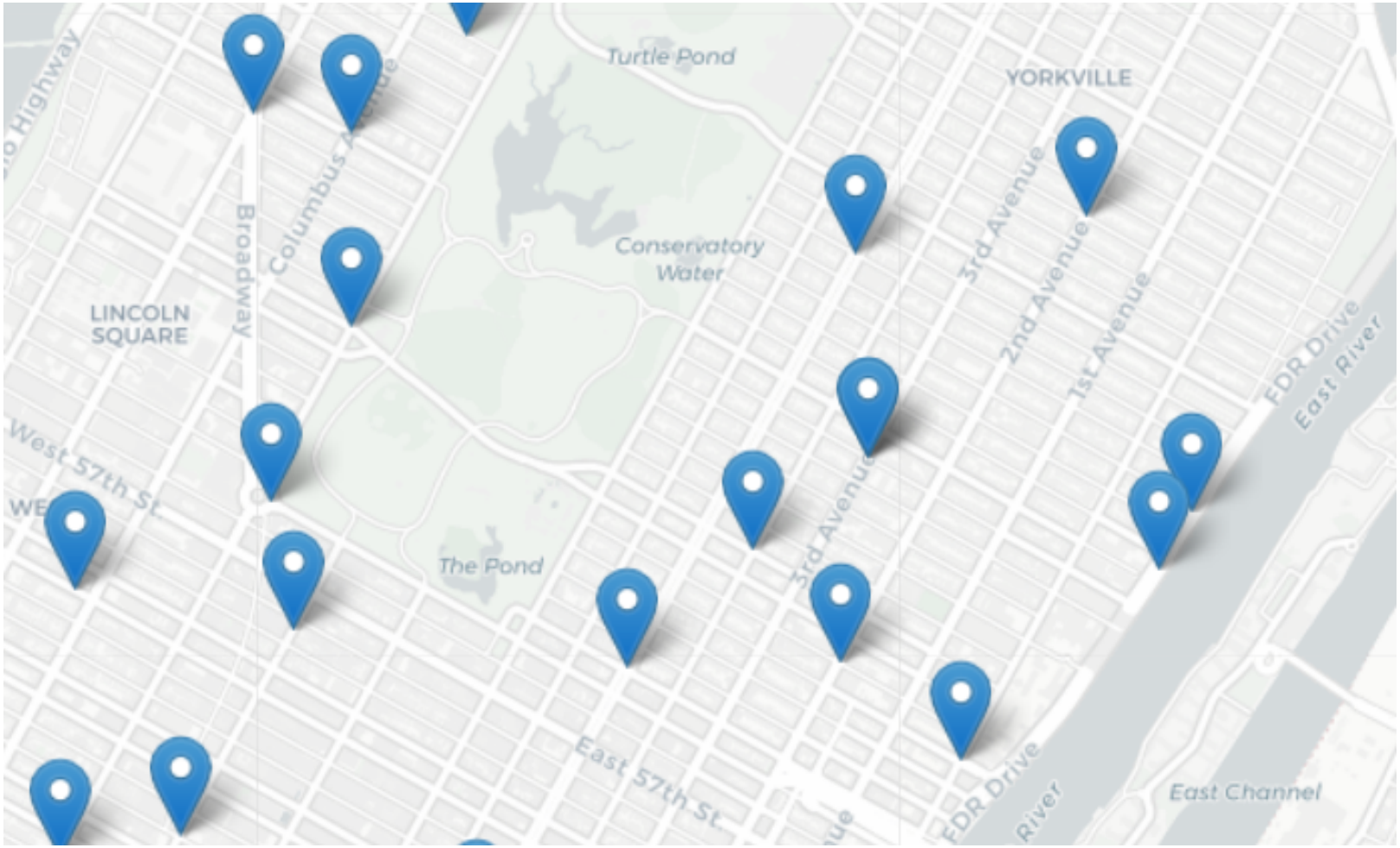
In Lab 6



CS Survey: Adrienne Schmoeker & Albert Webber

- Adrienne Schmoeker & Albert Webber
  NYC Mayor's Office of Data Analytics

Design Question



Design an algorithm that finds the closest collision.

(Sample NYC OpenData collision data file on back of lecture slip.)

Design Question

Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

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.

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.

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.

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:

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).

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.

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.

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.

Design Question

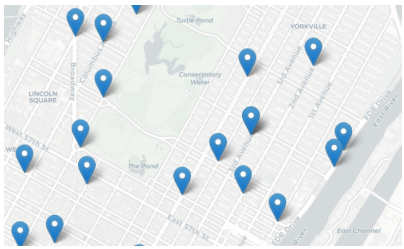
Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

How to approach this:

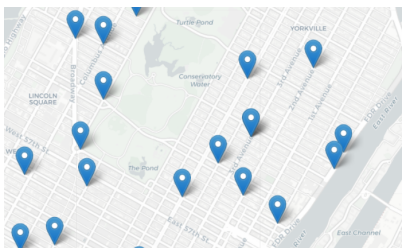
- Create a "To Do" list of what your program has to accomplish.
- Read through the problem, and break it into "To Do" items.
- Don't worry if you don't know how to do all the items you write down.
- Example:
  1. Find data set (great place to look: NYC OpenData).
  2. Ask user for current location.
  3. Open up the CSV file.
  4. Check distance to each to user's location.
  5. Save the location with the smallest distance.

Recap

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

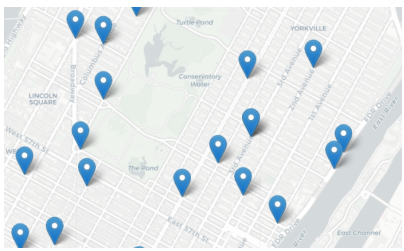


Recap



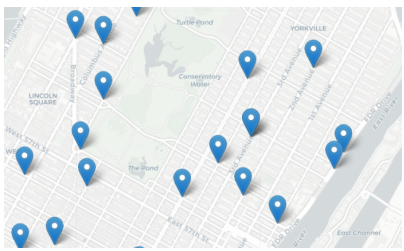
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Functions are a way to break code into pieces, that can be easily reused.

Recap



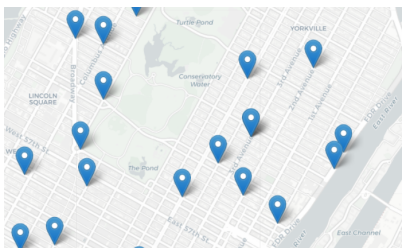
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:

Recap



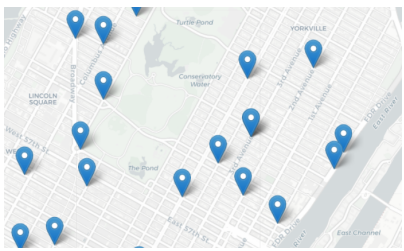
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
  Example: `print("Hello", "World")`

Recap



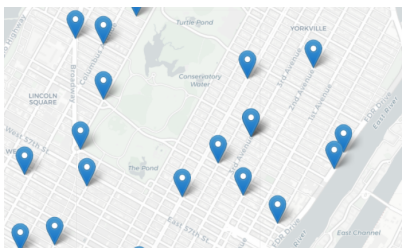
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
  Example: `print("Hello", "World")`
- Can write, or **define** your own functions,

Recap



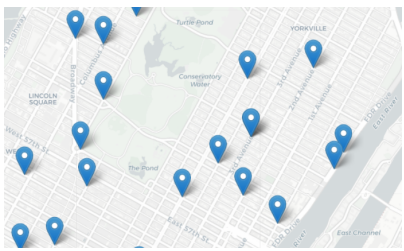
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
  Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.

Recap



- On lecture slip, write down a topic you wish we had spent more time (and why).
- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
  Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.
- Accessing Formatted Data: NYC OpenData

Recap



- On lecture slip, write down a topic you wish we had spent more time (and why).
- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
  Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.
- Accessing Formatted Data: NYC OpenData
- Pass your lecture slips to the aisles for the UTAs to collect.

Practice Quiz & Final Questions

```
#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()
```

```
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

```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

Actual Parameters

Formal Parameters

- Since you must pass the final exam to pass the course, we end every lecture with final exam review.

Practice Quiz & Final Questions

```
#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()
```

```
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

```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

Actual Parameters

Formal Parameters

- 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).

Practice Quiz & Final Questions

```
#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()
```

```
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

```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

Actual Parameters

Formal Parameters

- 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:

Practice Quiz & Final Questions

```
#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()
```

```
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

```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

Actual Parameters

Formal Parameters

- 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;

Practice Quiz & Final Questions

```
#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()
```

```
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

```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

Actual Parameters

Formal Parameters

- 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

Practice Quiz & Final Questions

```
#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()
```

```
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

```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

Actual Parameters

Formal Parameters

- 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.

Practice Quiz & Final Questions

```
#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()
```

```
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

```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

Actual Parameters

Formal Parameters

- 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).

Practice Quiz & Final Questions

```
#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()
```

```
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

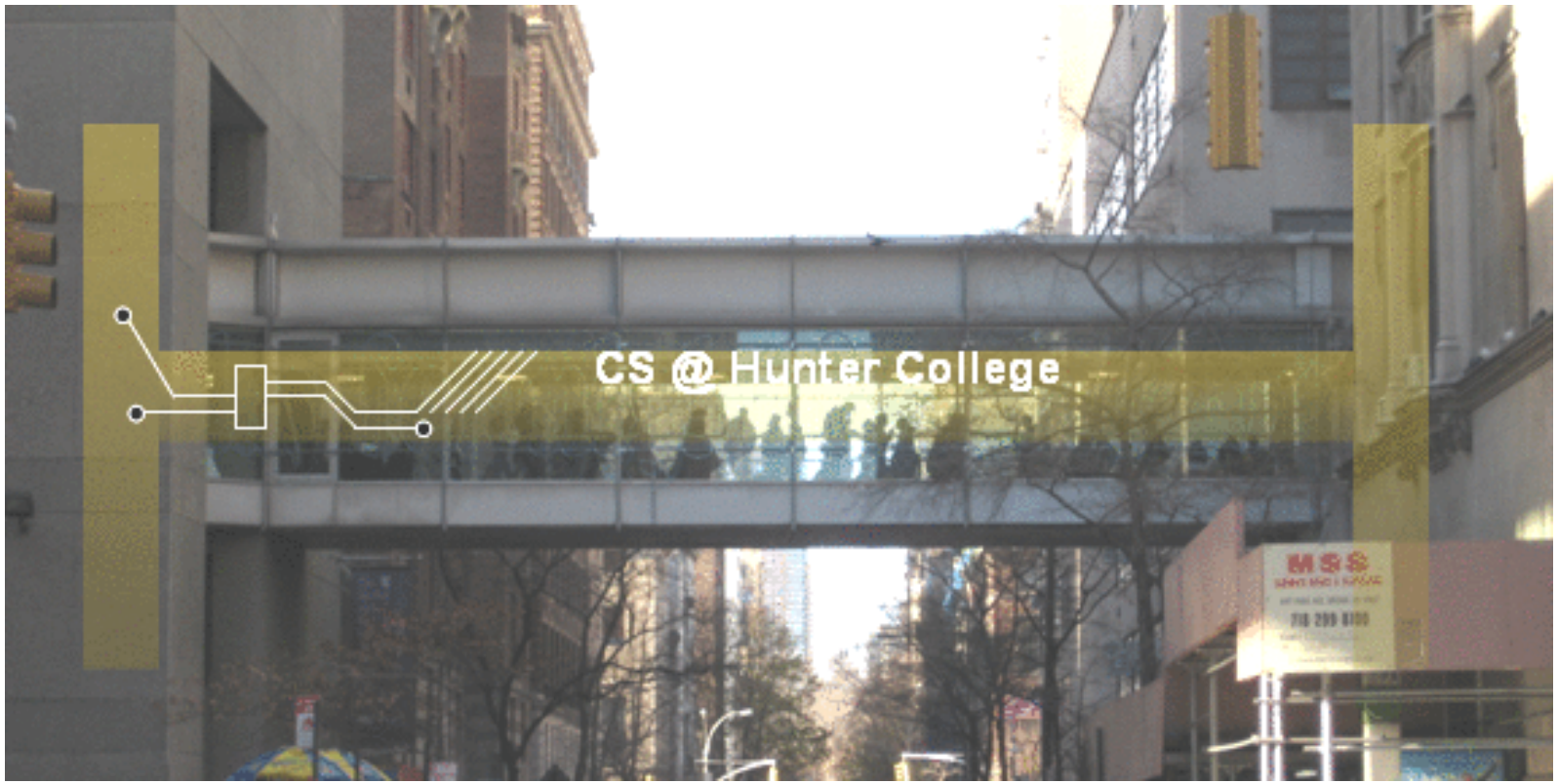
```
def prob4():
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is,")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(v):
    print(v)
    c = v.count("jam")
    return(c)
def enigma(v,c):
    print("but never", v[-1])
    for i in range(c):
        print("jam")
    return("day.")
prob4()
```

Actual Parameters

Formal Parameters

- 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!

Writing Boards



- Return writing boards as you leave...