

CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Announcements



- Guest Lecturer: Dr. Tiziana Ligorio

Today's Topics



- Recap: Slicing & Images
- Introduction to Functions
- NYC Open Data

Today's Topics



- **Recap: Slicing & Images**
- Introduction to Functions
- NYC Open Data

In Pairs or Triples:

Review: predict what the code will do:

```
motto = "Mihi Cura Futuri"  
l = len(motto)  
for i in range(l):  
    print(motto[i])  
for j in range(l-1,-1,-1):  
    print(motto[j])
```

```
import matplotlib.pyplot as plt  
import numpy as np  
img = plt.imread('csBridge.png')  
plt.imshow(img2)  
plt.show()  
height = img.shape[0]  
width = img.shape[1]  
img2 = img[height//2:, width//2:]  
plt.imshow(img2)  
plt.show()
```

Python Tutor

```
motto = "Mihi Cura Futuri"  
l = len(motto)  
for i in range(l):  
    print(motto[i])  
for j in range(l-1,-1,-1):  
    print(motto[j])
```

(Demo with pythonTutor)

Images

```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge.png')
plt.imshow(img2)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[height//2:, width//2:]
plt.imshow(img2)
plt.show()
```

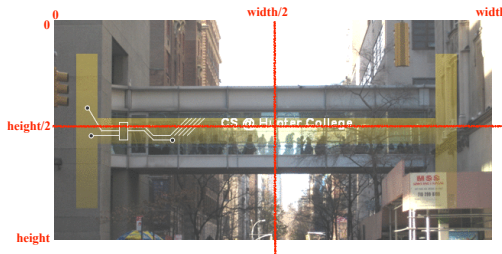
Challenge: Image

```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge.png')
plt.imshow(img2)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[height//2:, width//2:]
plt.imshow(img2)
plt.show()
```



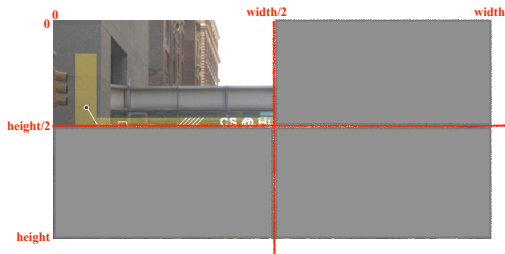
Challenge: Image

```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge.png')
plt.imshow(img2)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[height//2:, width//2:]
plt.imshow(img2)
plt.show()
```



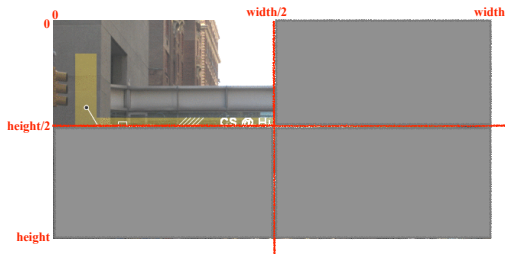
Challenge: Image

```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge.png')
plt.imshow(img2)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[height//2:, width//2:]
plt.imshow(img2)
plt.show()
```



Challenge: Image

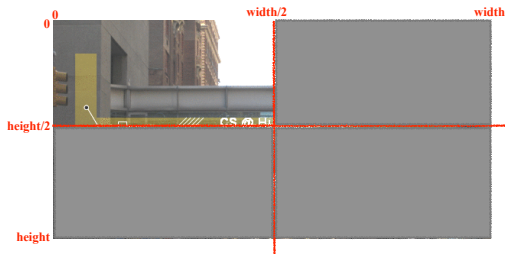
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge.png')
plt.imshow(img2)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[height//2:, width//2:]
plt.imshow(img2)
plt.show()
```



- How would you select the lower left corner?

Challenge: Image

```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge.png')
plt.imshow(img2)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[height//2:, width//2:]
plt.imshow(img2)
plt.show()
```

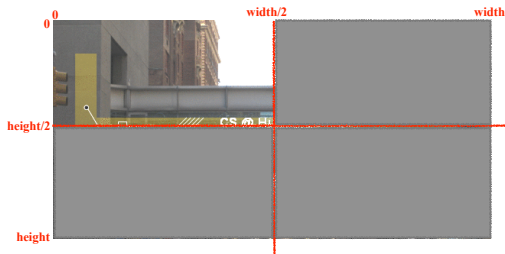


- How would you select the lower left corner?

```
img2 = img[height//2:, :width//2]
```

Challenge: Image

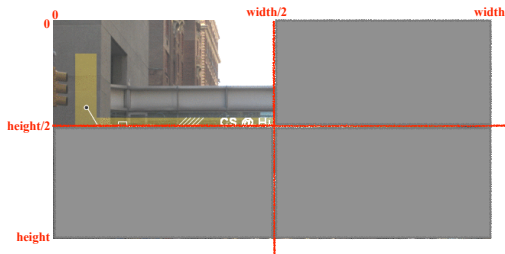
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge.png')
plt.imshow(img2)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[height//2:, width//2:]
plt.imshow(img2)
plt.show()
```



- How would you select the lower left corner?
`img2 = img[height//2:, :width//2]`
- How would you select the upper right corner?

Challenge: Image

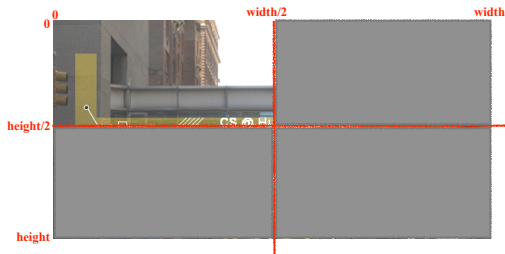
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge.png')
plt.imshow(img2)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[height//2:, width//2:]
plt.imshow(img2)
plt.show()
```



- How would you select the lower left corner?
`img2 = img[height//2:, :width//2]`
- How would you select the upper right corner?
`img2 = img[height//2:, width//2:]`

Challenge: Image

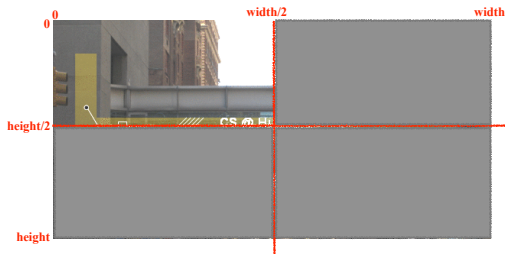
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge.png')
plt.imshow(img2)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[height//2:, width//2:]
plt.imshow(img2)
plt.show()
```



- How would you select the lower left corner?
`img2 = img[height//2:, :width//2]`
- How would you select the upper right corner?
`img2 = img[height//2:, width//2:]`
- How would you select the lower right corner?

Challenge: Image

```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge.png')
plt.imshow(img2)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[height//2:, width//2:]
plt.imshow(img2)
plt.show()
```



- How would you select the lower left corner?
`img2 = img[height//2:, :width//2]`
- How would you select the upper right corner?
`img2 = img[height//2:, width//2:]`
- How would you select the lower right corner?
`img2 = img[:height//2, :width//2]`

Today's Topics



- Recap: Slicing & Images
- **Introduction to Functions**
- NYC Open Data

Functions

- Functions are a way to break code into pieces, that can be easily reused.

```
#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

```
#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

- Functions can have **input parameters**.
- Surrounded by parentheses, both in the function definition, and in the function call (invocation).

```
def totalWithTax(food,tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)  
  
lunch = float(input('Enter lunch total: '))  
lTip = float(input('Enter lunch tip: ' ))  
lTotal = totalWithTax(lunch, lTip)  
print('Lunch total is', lTotal)  
  
dinner= float(input('Enter dinner total: '))  
dTip = float(input('Enter dinner tip: ' ))  
dTotal = totalWithTax(dinner, dTip)  
print('Dinner total is', dTotal)
```

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()
```

The diagram illustrates the flow of parameters between functions. Purple arrows, labeled "Actual Parameters", point from the arguments in function calls to the parameters in function definitions. Red arrows, labeled "Formal Parameters", point from the parameter names in function definitions to the function call. Specifically, purple arrows point from `verse` in `mystery(verse)` to `v` in `mystery(v)`, and from `verse` and `c` in `enigma(verse, c)` to `v` and `c` in `enigma(v, c)`. Red arrows point from `v` in `mystery(v)` and `v, c` in `enigma(v, c)` to the `mystery` and `enigma` calls within `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: '))
    nString = monthString(n)
    print('The month is', nString)
```

(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()
```

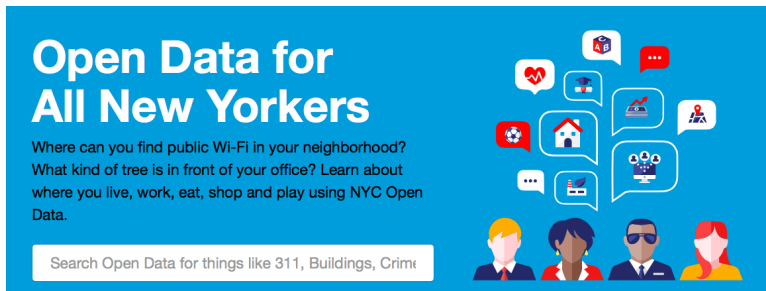
- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.

Today's Topics



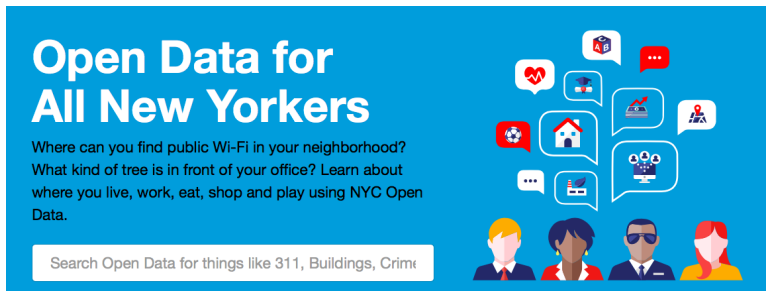
- Recap: Slicing & Images
- Introduction to Functions
- **NYC Open Data**

Accessing Structured Data: NYC Open Data

A blue banner for NYC Open Data. On the left, the text "Open Data for All New Yorkers" is in large white font. Below it, a paragraph asks: "Where can you find public Wi-Fi in your neighborhood? What kind of tree is in front of your office? Learn about where you live, work, eat, shop and play using NYC Open Data." At the bottom left is a white search bar with the placeholder text "Search Open Data for things like 311, Buildings, Crime". On the right, there are several white speech bubbles containing icons: a heart with a pulse line, a graduation cap, a house, a soccer ball, a bar chart with an upward arrow, a location pin, a group of people, and a Wi-Fi symbol. Below the speech bubbles are four stylized human avatars with different skin tones and hairstyles.

- Freely available source of data.

Accessing Structured Data: NYC Open Data

A blue banner for 'Open Data for All New Yorkers'. The title is in large white font. Below it, a paragraph asks questions about finding public Wi-Fi, trees, and local amenities using NYC Open Data. A search bar contains the text 'Search Open Data for things like 311, Buildings, Crime'. On the right, there are several speech bubbles containing icons for a heart with a pulse line, a graduation cap, a bar chart, a location pin, a house, a soccer ball, a factory, and a group of people. At the bottom right are four stylized avatars of diverse people.

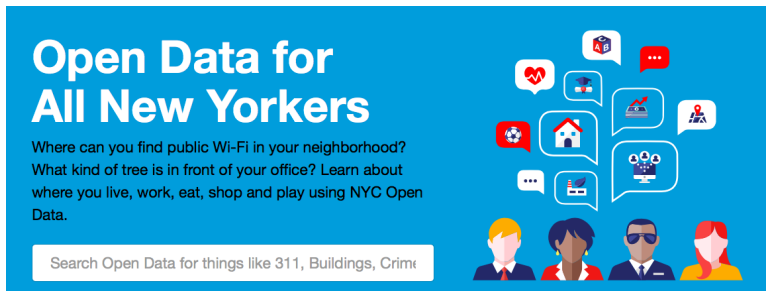
Open Data for All New Yorkers

Where can you find public Wi-Fi in your neighborhood? What kind of tree is in front of your office? Learn about where you live, work, eat, shop and play using NYC Open Data.

Search Open Data for things like 311, Buildings, Crime

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

Accessing Structured Data: NYC Open Data



Open Data for All New Yorkers

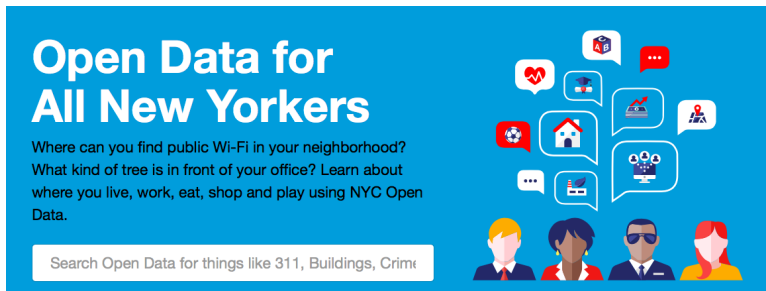
Where can you find public Wi-Fi in your neighborhood?
What kind of tree is in front of your office? Learn about where you live, work, eat, shop and play using NYC Open Data.

Search Open Data for things like 311, Buildings, Crime

The banner features a blue background with white text. On the right side, there are several white speech bubbles containing icons: a heart with a pulse line, a graduation cap, a bar chart with an upward arrow, a location pin, a house, a soccer ball, a factory, and a group of people. Below the speech bubbles are four stylized human figures with different skin tones and hairstyles.

- 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



Open Data for All New Yorkers

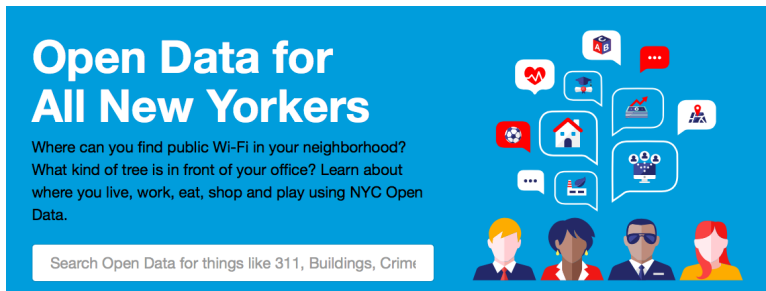
Where can you find public Wi-Fi in your neighborhood?
What kind of tree is in front of your office? Learn about where you live, work, eat, shop and play using NYC Open Data.

Search Open Data for things like 311, Buildings, Crime

The banner features a blue background with white text. On the right side, there are several white speech bubbles containing various icons: a heart with a pulse line, a graduation cap, a bar chart with an upward arrow, a location pin, a house, a soccer ball, a factory, and a group of people. Below the speech bubbles are four stylized human figures with different hair colors (yellow, dark blue, dark blue with sunglasses, and red) and clothing.

- 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



Open Data for All New Yorkers

Where can you find public Wi-Fi in your neighborhood?
What kind of tree is in front of your office? Learn about where you live, work, eat, shop and play using NYC Open Data.

Search Open Data for things like 311, Buildings, Crime

The banner features a blue background with white text. On the right side, there are several white speech bubbles containing various icons: a heart with a pulse line, a graduation cap, a bar chart with an upward arrow, a location pin, a house, a soccer ball, a factory, and a group of people. Below the speech bubbles are four stylized human figures with different skin tones and hairstyles, representing a diverse community.

- 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: OpenData Film Permits



Home Data About ▾ Learn

Film Permits

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/mome/permits/when-permit-required.page>

EventID	EventType	StartDateTL	EndDateTime	EnteredOn	EventAg	ParkingHeld	Borou
455063	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:36...	Mayor's Offic...	STARR AVENUE b...	Queens
454967	Shooting Permit	12/06/2018 07:00...	12/06/2018 05:00...	12/04/2018 09:11...	Mayor's Offic...	EAGLE STREET be...	Brooklyn
454941	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/04/2018 05:44...	Mayor's Offic...	SOUTH OXFORD ...	Brooklyn
454920	Shooting Permit	12/06/2018 10:00...	12/06/2018 11:59...	12/04/2018 03:28...	Mayor's Offic...	13 AVENUE betw...	Queens
454914	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/04/2018 03:05...	Mayor's Offic...	ELDERT STREET b...	Brooklyn
454909	Shooting Permit	12/05/2018 08:00...	12/05/2018 06:00...	12/04/2018 02:45...	Mayor's Offic...	ELDERT STREET b...	Brooklyn
454905	Shooting Permit	12/06/2018 07:00...	12/06/2018 10:00...	12/04/2018 02:17...	Mayor's Offic...	35 STREET betwe...	Queens

Example: OpenData Film Permits

NYC OpenData Home Data About ▾ Learn ▾ Alerts Contact Us Blog

Film Permits 📄 📱 📧

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/mome/permits/when-permit-required.page> More Views Filter Visualize Export Discuss Embed About

EventID	EventType	StartDateTL	EndDateTime	EnteredOn	EventAg	ParkingHeld	Borou	Com	Police	Categ	SubC	Count	ZipCo
455063	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:36...	Mayor's Offic...	STARR AVENUE b...	Queens	2	108	Television	Episodic s...	United Sta...	11101
454967	Shooting Permit	12/06/2018 07:00...	12/06/2018 05:00...	12/04/2018 09:11...	Mayor's Offic...	EAGLE STREET be...	Brooklyn	1	94	Television	Episodic s...	United Sta...	11222
454941	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/04/2018 05:44...	Mayor's Offic...	SOUTH OXFORD ...	Brooklyn	2, 6	76, 88	Still Photo...	Not Applic...	United Sta...	11217, 11...
454920	Shooting Permit	12/06/2018 10:00...	12/06/2018 11:59...	12/04/2018 03:28...	Mayor's Offic...	13 AVENUE betw...	Queens	1, 3, 7	109, 7, 90	Film	Feature	United Sta...	10002, 11...
454914	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/04/2018 03:05...	Mayor's Offic...	ELBERT STREET b...	Brooklyn	4, 5	104, 75, 83	Television	Episodic s...	United Sta...	11207, 11...
454909	Shooting Permit	12/05/2018 08:00...	12/05/2018 06:00...	12/04/2018 02:45...	Mayor's Offic...	ELBERT STREET b...	Brooklyn	4	83	Television	Episodic s...	United Sta...	11237
454905	Shooting Permit	12/06/2018 07:00...	12/06/2018 10:00...	12/04/2018 02:17...	Mayor's Offic...	35 STREET betwe...	Queens	1	114	Television	Cable-epis...	United Sta...	11101, 11...

- What's the most popular street for filming?

Example: OpenData Film Permits

NYC OpenData

Home Data About ▾ Learn ▾ Alerts Contact Us Blog |

Film Permits

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/mome/permits/when-permit-required.page>

Find in this Dataset

EventID	EventType	StartDateTL	EndDateTime	EnteredOn	EventAg	ParkingHeld	Borou	Com	Police	Categ	SubC	Count	ZipCo
455063	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:36...	Mayor's Offic...	STARR AVENUE b...	Queens	2	108	Television	Episodic s...	United Sta...	11101
454967	Shooting Permit	12/06/2018 07:00...	12/06/2018 05:00...	12/04/2018 09:11...	Mayor's Offic...	EAGLE STREET be...	Brooklyn	1	94	Television	Episodic s...	United Sta...	11222
454941	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/04/2018 05:44...	Mayor's Offic...	SOUTH OXFORD ...	Brooklyn	2, 6	76, 88	Still Photo...	Not Applic...	United Sta...	11217, 11...
454920	Shooting Permit	12/06/2018 10:00...	12/06/2018 11:59...	12/04/2018 03:28...	Mayor's Offic...	13 AVENUE betw...	Queens	1, 3, 7	109, 7, 90	Film	Feature	United Sta...	10002, 11...
454914	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/04/2018 03:05...	Mayor's Offic...	ELBERT STREET b...	Brooklyn	4, 5	104, 75, 83	Television	Episodic s...	United Sta...	11207, 11...
454909	Shooting Permit	12/05/2018 08:00...	12/05/2018 06:00...	12/04/2018 02:45...	Mayor's Offic...	ELBERT STREET b...	Brooklyn	4	83	Television	Episodic s...	United Sta...	11237
454905	Shooting Permit	12/06/2018 07:00...	12/06/2018 10:00...	12/04/2018 02:17...	Mayor's Offic...	35 STREET betwe...	Queens	1	114	Television	Cable-epis...	United Sta...	11101, 11...

- What's the most popular street for filming?
- What's the most popular borough?

Example: OpenData Film Permits



Home Data About ▾ Learn ▾ Alerts Contact Us Blog

Sign In

Film Permits

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/home/permits/when-permit-required.page>

Find in this Dataset

More Views Filter Visualize Export Discuss Embed About

EventID	EventType	StartDateTL	EndDateTime	EnteredOn	EventAg...	ParkingHeld	Borou...	Com...	Police...	Categ...	SubC...	Count...	ZipCo...
455063	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:36...	Mayor's Offic...	STARR AVENUE b...	Queens	2	108	Television	Episodic s...	United Sta...	11101
454967	Shooting Permit	12/06/2018 07:00...	12/06/2018 05:00...	12/04/2018 09:11...	Mayor's Offic...	EAGLE STREET be...	Brooklyn	1	94	Television	Episodic s...	United Sta...	11222
454941	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/04/2018 05:44...	Mayor's Offic...	SOUTH OXFORD ...	Brooklyn	2, 6	76, 88	Still Photo...	Not Applic...	United Sta...	11217, 11...
454920	Shooting Permit	12/06/2018 10:00...	12/06/2018 11:59...	12/04/2018 03:28...	Mayor's Offic...	13 AVENUE betw...	Queens	1, 3, 7	109, 7, 90	Film	Feature	United Sta...	10002, 11...
454914	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/04/2018 03:05...	Mayor's Offic...	ELBERT STREET b...	Brooklyn	4, 5	104, 75, 83	Television	Episodic s...	United Sta...	11207, 11...
454909	Shooting Permit	12/05/2018 08:00...	12/05/2018 06:00...	12/04/2018 02:45...	Mayor's Offic...	ELBERT STREET b...	Brooklyn	4	83	Television	Episodic s...	United Sta...	11237
454905	Shooting Permit	12/06/2018 07:00...	12/06/2018 10:00...	12/04/2018 02:17...	Mayor's Offic...	35 STREET betwe...	Queens	1	114	Television	Cable-epis...	United Sta...	11101, 11...

- What's the most popular street for filming?
- What's the most popular borough?
- How many TV episodes were filmed?

Example: OpenData Film Permits

NYC OpenData Home Data About Learn Alerts Contact Us Blog

Film Permits

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/nycopen/permits/when-permit-required.page>

EventID	EventType	StartDateT...	EndDateTime	EnteredOn	EventAg...	ParkingInf...	Borne...	Cent...	Police...	Categ...	SubCat...	Count...	ZipCo...
455063	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:35...	Mayor's Offi...	STARKE AVENUE b...	Queens	2	108	Television	Epicodic S...	United Sta...	11101
454967	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/06/2018 09:11...	Mayor's Offi...	EAGLE STREET b...	Brooklyn	1	84	Television	Epicodic S...	United Sta...	11222
454941	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/06/2018 05:44...	Mayor's Offi...	SOUTH OXFORD ...	Brooklyn	2, 6	76, 88	3/8 Photo...	Not Applica...	United Sta...	11215, 11...
454920	Shooting Permit	12/06/2018 13:00...	12/06/2018 11:00...	12/06/2018 03:28...	Mayor's Offi...	13 AVENUE betw...	Queens	1, 3, 7	108, 7, 98	Film	Feature	United Sta...	10052, 11...
454914	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/06/2018 03:05...	Mayor's Offi...	ELDERST STREET b...	Brooklyn	4, 9	104, 76, 83	Television	Epicodic S...	United Sta...	11200, 11...
454909	Shooting Permit	12/05/2018 08:00...	12/05/2018 09:00...	12/06/2018 02:45...	Mayor's Offi...	ELDERST STREET b...	Brooklyn	4	83	Television	Epicodic S...	United Sta...	11227
454905	Shooting Permit	12/06/2018 07:00...	12/06/2018 10:00...	12/06/2018 02:17...	Mayor's Offi...	35 STREET betw...	Queens	1	114	Television	Cable-epic...	United Sta...	11101, 11...

- Download the data as a CSV file and store on your computer.

Example: OpenData Film Permits

NYC OpenData

Home Data About Learn Alerts Contact Us Blog Sign In

Film Permits

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/nycopendata/permits/when-permit-required.page>

Find in this Dataset

More Views Filter Visualize Export Discuss Embed About

EventID	EventType	StartDateT...	EndDateTime	EnteredOn	EventAg...	ParkingInf...	Borne...	Cent...	Police...	Categ...	SubCat...	Count...	ZipCo...
45503	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:35...	Mayor's Offi...	STARKE AVENUE b...	Queens	2	108	Television	Episodic S...	United Sta...	11101
45467	Shooting Permit	12/06/2018 07:00...	12/06/2018 05:00...	12/04/2018 09:11...	Mayor's Offi...	EAGLE STREET b...	Brooklyn	1	84	Television	Episodic S...	United Sta...	11222
45481	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/04/2018 05:44...	Mayor's Offi...	SOUTH OXFORD ...	Brooklyn	2, 6	76, 88	3/8 Photo...	Not Applica...	United Sta...	11215, 11...
45400	Shooting Permit	12/06/2018 13:00...	12/06/2018 11:00...	12/04/2018 03:28...	Mayor's Offi...	13 AVENUE betw...	Queens	1, 3, 7	108, 7, 98	Film	Feature	United Sta...	10002, 11...
454914	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/04/2018 03:05...	Mayor's Offi...	ELDERST STREET b...	Brooklyn	4, 9	104, 76, 83	Television	Episodic S...	United Sta...	11200, 11...
454909	Shooting Permit	12/05/2018 08:00...	12/05/2018 06:00...	12/04/2018 02:45...	Mayor's Offi...	ELDERST STREET b...	Brooklyn	4	83	Television	Episodic S...	United Sta...	11227
45485	Shooting Permit	12/06/2018 07:00...	12/06/2018 10:00...	12/04/2018 02:17...	Mayor's Offi...	35 STREET betw...	Queens	1	114	Television	Cable-epi...	United Sta...	11101, 11...

- Download the data as a CSV file and store on your computer.
- Python program:

```
#CSci 127 Teaching Staff
#March 2019
#OpenData Film Permits
```

```
#Import pandas for reading and analyzing CSV data:
import pandas as pd
csvFile = "filmPermits.csv" #Name of the CSV file
tickets = pd.read_csv(csvFile)#Read in the file to a dataframe
```

Example: OpenData Film Permits

NYC OpenData

Home Data About Learn Alerts Contact Us Blog Sign In

Film Permits

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/nycopendata/permits/when-permit-required.page>

Find in this Dataset

More Views Filter Visualize Export Discuss Embed About

EventId	EventType	StartDate	EndDate	EventDate	EventName	ParkingInfo	Borough	Cent.	Police	Catg.	SubCat.	County	ZipCode
45503	Shooting Permit	12/06/2018 07:00	12/06/2018 09:00	12/05/2018 12:35	Mayor's Office	STARKE AVENUE B...	Queens	2	108	Television	Episodic S...	United Sta...	11101
45467	Shooting Permit	12/06/2018 07:00	12/06/2018 09:00	12/04/2018 09:11	Mayor's Office	EAGLE STREET B...	Brooklyn	1	84	Television	Episodic S...	United Sta...	11222
45491	Shooting Permit	12/06/2018 07:00	12/06/2018 07:00	12/04/2018 05:44	Mayor's Office	SOUTH OXFORD ...	Brooklyn	2, 6	76, 88	S&B Photo...	Not Applica...	United Sta...	11215, 11...
45400	Shooting Permit	12/06/2018 13:00	12/06/2018 11:00	12/04/2018 03:28	Mayor's Office	13 AVENUE betw...	Queens	1, 3, 7	108, 7, 98	Film	Feature	United Sta...	10002, 11...
454914	Shooting Permit	12/06/2018 08:00	12/06/2018 11:00	12/04/2018 03:05	Mayor's Office	ELBERT STREET B...	Brooklyn	4, 6	104, 76, 83	Television	Episodic S...	United Sta...	11200, 11...
454809	Shooting Permit	12/05/2018 08:00	12/05/2018 09:00	12/04/2018 02:45	Mayor's Office	ELBERT STREET B...	Brooklyn	4	83	Television	Episodic S...	United Sta...	11227
454865	Shooting Permit	12/06/2018 07:00	12/06/2018 10:00	12/04/2018 02:17	Mayor's Office	35 STREET betw...	Queens	1	114	Television	Cable-epic...	United Sta...	11101, 11...

- Download the data as a CSV file and store on your computer.

- Python program:

```
#CSci 127 Teaching Staff
```

```
#March 2019
```

```
#OpenData Film Permits
```

```
#Import pandas for reading and analyzing CSV data:
```

```
import pandas as pd
```

```
csvFile = "filmPermits.csv" #Name of the CSV file
```

```
tickets = pd.read_csv(csvFile)#Read in the file to a dataframe
```

```
print(tickets) #Print out the dataframe
```

Example: OpenData Film Permits

NYC OpenData

Home Data About Learn Alerts Contact Us Blog Q Sign In

Film Permits

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/nycopendata/permits/when-permit-required.page>

Find in this Dataset

More Views Filter Visualize Export Discuss Embed About

EventID	EventType	StartDateT...	EndDateTime	EnteredOn	EventAg...	ParkingHeld	Borne...	Cent...	Police...	Categ...	SubC...	Count...	ZipCo...
45503	Shooting Permit	12/05/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:35...	Mayor's Offi...	STARKE AVENUE b...	Queens	2	108	Television	Episodic S...	United Sta...	11101
45467	Shooting Permit	12/06/2018 07:00...	12/06/2018 05:00...	12/06/2018 09:11...	Mayor's Offi...	EAGLE STREET b...	Brooklyn	1	84	Television	Episodic S...	United Sta...	11222
45491	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/06/2018 05:44...	Mayor's Offi...	SOUTH OXFORD ...	Brooklyn	2, 6	76, 88	SUB Photo...	Not Applica...	United Sta...	11215, 11...
45400	Shooting Permit	12/06/2018 13:00...	12/06/2018 11:00...	12/06/2018 03:28...	Mayor's Offi...	13 AVENUE betw...	Queens	1, 3, 7	108, 7, 98	Film	Feature	United Sta...	10002, 11...
454914	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/06/2018 03:05...	Mayor's Offi...	ELBERT STREET b...	Brooklyn	4, 6	104, 76, 83	Television	Episodic S...	United Sta...	11202, 11...
454809	Shooting Permit	12/05/2018 08:00...	12/05/2018 05:00...	12/06/2018 02:45...	Mayor's Offi...	ELBERT STREET b...	Brooklyn	4	83	Television	Episodic S...	United Sta...	11227
45485	Shooting Permit	12/06/2018 07:00...	12/06/2018 10:00...	12/06/2018 02:17...	Mayor's Offi...	35 STREET betw...	Queens	1	114	Television	Cable-epic...	United Sta...	11101, 11...

- Download the data as a CSV file and store on your computer.
- Python program:

```
#CSci 127 Teaching Staff
#March 2019
#OpenData Film Permits
```

```
#Import pandas for reading and analyzing CSV data:
import pandas as pd
csvFile = "filmPermits.csv" #Name of the CSV file
tickets = pd.read_csv(csvFile)#Read in the file to a dataframe
print(tickets) #Print out the dataframe
print(tickets["ParkingHeld"]) #Print out streets (multiple times)
```

Example: OpenData Film Permits

NYC OpenData

Home Data About Learn Alerts Contact Us Blog Search Sign In

Film Permits

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/nycopendata/permits/when-permit-required.page>

Find in this Dataset

More Views Filter Visualize Export Discuss Embed About

EventID	EventType	StartDateT...	EndDateTime	EnteredOn	EventAg...	ParkingHeld	Borne...	Cent...	Police...	Categ...	SubCat...	Count...	ZipCo...
45503	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:35...	Mayor's Offi...	STARKE AVENUE b...	Queens	2	108	Television	Episodic S...	United Sta...	11101
45467	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/04/2018 09:11...	Mayor's Offi...	EAGLE STREET b...	Brooklyn	1	84	Television	Episodic S...	United Sta...	11222
45461	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/04/2018 09:44...	Mayor's Offi...	SOUTH OXFORD ...	Brooklyn	2, 6	76, 88	3/8 Photo...	Not Applica...	United Sta...	11215, 11...
45400	Shooting Permit	12/06/2018 13:00...	12/06/2018 11:55...	12/04/2018 03:28...	Mayor's Offi...	13 AVENUE betw...	Queens	1, 3, 7	108, 7, 98	Film	Feature	United Sta...	10052, 11...
454914	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/04/2018 09:05...	Mayor's Offi...	ELBERT STREET b...	Brooklyn	4, 6	104, 76, 83	Television	Episodic S...	United Sta...	11200, 11...
454809	Shooting Permit	12/05/2018 08:00...	12/05/2018 09:00...	12/04/2018 02:45...	Mayor's Offi...	ELBERT STREET b...	Brooklyn	4	83	Television	Episodic S...	United Sta...	11227
454865	Shooting Permit	12/06/2018 07:00...	12/06/2018 10:00...	12/04/2018 02:17...	Mayor's Offi...	35 STREET betw...	Queens	1	114	Television	Cable-epic...	United Sta...	11101, 11...

- Download the data as a CSV file and store on your computer.
- Python program:

```
#CSci 127 Teaching Staff
#March 2019
#OpenData Film Permits
```

```
#Import pandas for reading and analyzing CSV data:
```

```
import pandas as pd
csvFile = "filmPermits.csv" #Name of the CSV file
tickets = pd.read_csv(csvFile)#Read in the file to a dataframe
print(tickets) #Print out the dataframe
print(tickets["ParkingHeld"]) #Print out streets (multiple times)
print(tickets["ParkingHeld"].value_counts()) #Print out streets & number of times used
```


Example: OpenData Film Permits

NYC OpenData

Home Data About Learn Alerts Contact Us Blog Sign In

Film Permits

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/nycopendata/permits/when-permit-required.page>

Find in this Dataset

More Views Filter Visualize Export Discuss Embed About

EventID	EventType	StartDate	EndDate	EnterDate	EventAge	ParkingHeld	Borne	Cent.	Police	Comp.	SubC.	Count	ZipCo.
45503	Shooting Permit	12/06/2018 07:00	12/06/2018 09:00	12/05/2018 12:35	Mayor's Office	STARKE AVENUE b...	Queens	2	108	Television	Episodic s...	United Sta...	11101
45467	Shooting Permit	12/06/2018 07:00	12/06/2018 09:00	12/04/2018 09:11	Mayor's Office	EAGLE STREET b...	Brooklyn	1	84	Television	Episodic s...	United Sta...	11222
45461	Shooting Permit	12/06/2018 07:00	12/06/2018 07:00	12/04/2018 09:44	Mayor's Office	SOUTH OXFORD	Brooklyn	2	76, 88	SUB Photo...	Not Applica...	United Sta...	11215, 11...
45400	Shooting Permit	12/06/2018 13:00	12/06/2018 11:00	12/04/2018 03:28	Mayor's Office	13 AVENUE betwe...	Queens	1, 3, 7	108, 7, 98	Film	Feature	United Sta...	10002, 11...
45414	Shooting Permit	12/06/2018 08:00	12/06/2018 11:00	12/04/2018 03:05	Mayor's Office	ELBERT STREET b...	Brooklyn	4, 6	104, 76, 83	Television	Episodic s...	United Sta...	11200, 11...
45409	Shooting Permit	12/05/2018 08:00	12/05/2018 09:00	12/04/2018 02:45	Mayor's Office	ELBERT STREET b...	Brooklyn	4	83	Television	Episodic s...	United Sta...	11227
45405	Shooting Permit	12/06/2018 07:00	12/06/2018 10:00	12/04/2018 02:17	Mayor's Office	35 STREET betwe...	Queens	1	114	Television	Cable-epic...	United Sta...	11101, 11...

- Download the data as a CSV file and store on your computer.
- Python program:

```
#CSci 127 Teaching Staff
#March 2019
#OpenData Film Permits
```

```
#Import pandas for reading and analyzing CSV data:
```

```
import pandas as pd
csvFile = "filmPermits.csv" #Name of the CSV file
tickets = pd.read_csv(csvFile)#Read in the file to a dataframe
print(tickets) #Print out the dataframe
print(tickets["ParkingHeld"]) #Print out streets (multiple times)
print(tickets["ParkingHeld"].value_counts()) #Print out streets & number of times used
print(tickets["ParkingHeld"].value_counts()[:10]) #Print 10 most popular
```

Example: OpenData Film Permits

NYC OpenData

Home Data About Learn Alerts Contact Us Blog Sign In

Film Permits

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/mome/permits/when-permit-required.page>

Find in this Dataset

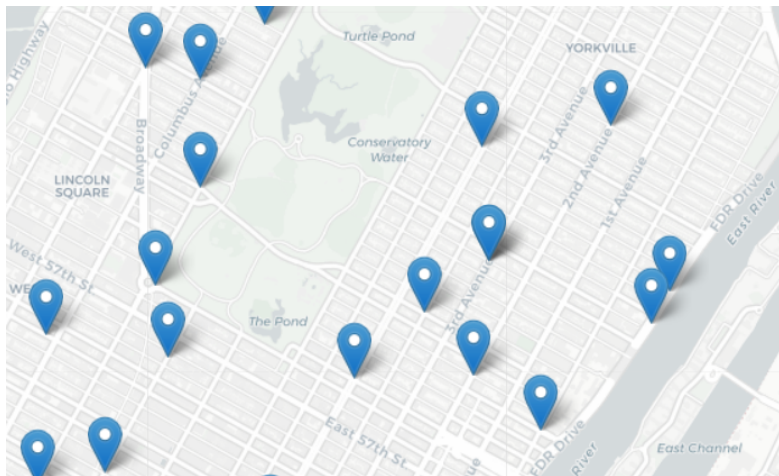
More Views Filter Visualize Export Discuss Embed About

EventID	EventType	StartDateTL	EndDateTime	EnteredOn	EventAg	ParkingHeld	Borou	Com	Police	Categ	SubC	Count	ZipCo
455063	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:36...	Mayor's Offic...	STARR AVENUE b...	Queens	2	108	Television	Episodic s...	United Sta...	11101
454967	Shooting Permit	12/06/2018 07:00...	12/06/2018 05:00...	12/04/2018 09:11...	Mayor's Offic...	EAGLE STREET be...	Brooklyn	1	94	Television	Episodic s...	United Sta...	11222
454941	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/04/2018 05:44...	Mayor's Offic...	SOUTH OXFORD ...	Brooklyn	2, 6	76, 88	Still Photo...	Not Applic...	United Sta...	11217, 11...
454920	Shooting Permit	12/06/2018 10:00...	12/06/2018 11:59...	12/04/2018 03:28...	Mayor's Offic...	13 AVENUE betw...	Queens	1, 3, 7	109, 7, 90	Film	Feature	United Sta...	10002, 11...
454914	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/04/2018 03:05...	Mayor's Offic...	ELBERT STREET b...	Brooklyn	4, 5	104, 75, 83	Television	Episodic s...	United Sta...	11207, 11...
454909	Shooting Permit	12/05/2018 08:00...	12/05/2018 06:00...	12/04/2018 02:45...	Mayor's Offic...	ELBERT STREET b...	Brooklyn	4	83	Television	Episodic s...	United Sta...	11237
454905	Shooting Permit	12/06/2018 07:00...	12/06/2018 10:00...	12/04/2018 02:17...	Mayor's Offic...	35 STREET betwe...	Queens	1	114	Television	Cable-epis...	United Sta...	11101, 11...

Can approach the other questions in the same way:

- What's the most popular street for filming?
- What's the most popular borough?
- How many TV episodes were filmed?

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.

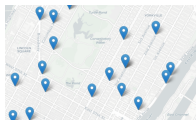
Recap

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

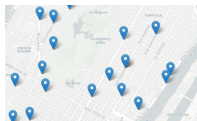


Recap

- 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.

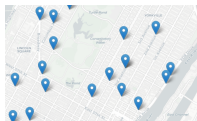


Recap



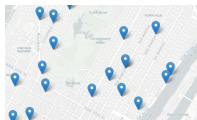
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:

Recap



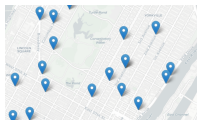
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
Example: `print("Hello", "World")`

Recap



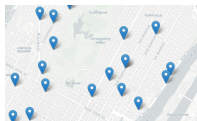
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
Example: `print("Hello", "World")`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

Recap



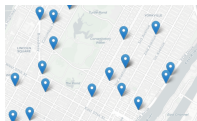
- On lecture slip, write down a topic you wish we had spent more time (and why).
- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
Example: `print("Hello", "World")`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

```
def prob4(l):
    verse = "jam tomorrow and jam yesterday."
    print("The rule is.")
    c = mystery(verse)
    w = enigma(verse, c)
    print(c, w)
def mystery(s):
    print(w)
    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()
```

- 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)
```

```
def prob4(l):
    verse = "jam tomorrow and jam yesterday."
    print("The rule is.")
    c = mystery(verse)
    w = enigma(verse, c)
    print(c, w)
def mystery(s):
    print(w)
    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()
```

- 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!")
```

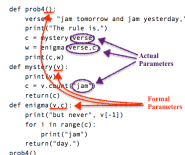
```
if __name__ == "__main__":
    main()
```

```
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)
```

```
def prob4(l):
    verse = "jam tomorrow and jam yesterday."
    print("The rule is.")
    c = mystery(verse)
    w = enigma(verse, c)
    print(c, w)
def mystery(s):
    print(w)
    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()
```



- 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)
```

```
def prob4(l):
    verse = "jam tomorrow and jam yesterday."
    print("The rule is.")
    c = mystery(verse)
    w = enigma(verse, c)
    print(c, w)
def mystery(s):
    print(w)
    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()
```

- 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)
```

```
def prob4(l):
    verse = "jam tomorrow and jam yesterday."
    print("The rule is.")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(s):
    print(s)
    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()
```

- 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)
```

```
def prob4(l):
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is.")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(s):
    print(w)
    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()
```

- 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)
```

```
def prob4(l):
    verse = "jam tomorrow and jam yesterday."
    print("The rule is.")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(s):
    print(w)
    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()
```

- 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)
```

```
def prob4(l):
    verse = "jam tomorrow and jam yesterday."
    print("The rule is.")
    c = mystery(verse,c)
    w = enigma(verse,c)
    print(c,w)
def mystery(v,c):
    print(v,c)
    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()
```

- 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!

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)
```

```
def prob4(l):
    verse = "jam tomorrow and jam yesterday,"
    print("The rule is.")
    c = mystery(verse)
    w = enigma(verse, c)
    print(c, w)
def mystery(s):
    print(w)
    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()
```

- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:
 - ▶ write as much you can for 60 seconds;
 - ▶ followed by answer; and
 - ▶ repeat.
- Past exams are on the webpage (under [Final Exam Information](#)).
- Theme: Functions!
Starting with F18 Version 2, #4.

Writing Boards



- Return writing boards as you leave...