

CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Two Dimensional Array Slicing

```
1 import numpy as np
2
3 numRows = 6
4 numCols = 6
5 a = np.zeros((numRows, numCols))
6 #create a table with 6 rows and 6 columns,
7 #each element is initialized to be zero.
8 #Do not forget parentheses around
9 #numRows, numCols.
```

Two Dimensional Array Slicing: II

```
8  for i in range(numRows):
9      for j in range(numCols):
10         a[i, j] = i*10 + j
11 #range(numRows) returns [0, 1, 2, 3, 4, 5],
12 #where outer loop variable i chooses from.
13 #When i is 0, run
14 #     for j in range(numCols):
15 #         a[i, j] = i*10 + j
16 #When i is 1, run
17 #     for j in range(numCols):
18 #         a[i, j] = i*10 + j
19 #The last round of i is 5.
```

Two Dimensional Array Slicing: III

```
20 for i in range(numRows):
21     for j in range(numCols):
22         print("%3i"%(a[i, j]), end="")
23         #"%3i"%(a[i, j]) prints a[i, j] --
24         #element of a at ith row and
25         #jth column -- as an 3-digit int.
26         #"%3i" is a place holder and is
27         #filled by a[i, j].
28         #If a[i, j] does not have 3 digits,
29         #pad space(s) to the left.
30         #end="" print w/o a new line.
31 print() #print a new line after each row
```

Two Dimensional Array Slicing: III

32

```
print(a[0, 3:5])
```

row \ col	0	1	2	3	4	5
0	0	1	2	3	4	5
1	10	11	12	13	14	15
2	20	21	22	23	24	25
3	30	31	32	33	34	35
4	40	41	42	43	44	45
5	50	51	52	53	54	55

Two Dimensional Array Slicing: III

32

```
print(a[0, 3:5])
```

row \ col	0	1	2	3	4	5
0	0	1	2	3	4	5
1	10	11	12	13	14	15
2	20	21	22	23	24	25
3	30	31	32	33	34	35
4	40	41	42	43	44	45
5	50	51	52	53	54	55

row \ col	0	1	2	3	4	5
0	0	1	2	3	4	5
1	10	11	12	13	14	15
2	20	21	22	23	24	25
3	30	31	32	33	34	35
4	40	41	42	43	44	45
5	50	51	52	53	54	55

print

```
[3. 4.]
```

Two Dimensional Array Slicing: IV

33

```
print(a[4:, 4:])
```

row \ col	0	1	2	3	4	5
0	0	1	2	3	4	5
1	10	11	12	13	14	15
2	20	21	22	23	24	25
3	30	31	32	33	34	35
4	40	41	42	43	44	45
5	50	51	52	53	54	55

Two Dimensional Array Slicing: IV

33

```
print(a[4:, 4:])
```

row \ col	0	1	2	3	4	5
0	0	1	2	3	4	5
1	10	11	12	13	14	15
2	20	21	22	23	24	25
3	30	31	32	33	34	35
4	40	41	42	43	44	45
5	50	51	52	53	54	55

row \ col	0	1	2	3	4	5
0	0	1	2	3	4	5
1	10	11	12	13	14	15
2	20	21	22	23	24	25
3	30	31	32	33	34	35
4	40	41	42	43	44	45
5	50	51	52	53	54	55

Print out

```
[[44. 45.]  
 [54. 55.]]
```


Two Dimensional Array Slicing: V

34

```
print(a[:, 2])
```

row \ col	0	1	2	3	4	5
0	0	1	2	3	4	5
1	10	11	12	13	14	15
2	20	21	22	23	24	25
3	30	31	32	33	34	35
4	40	41	42	43	44	45
5	50	51	52	53	54	55

Two Dimensional Array Slicing: V

34

```
print(a[:, 2])
```

row \ col	0	1	2	3	4	5
0	0	1	2	3	4	5
1	10	11	12	13	14	15
2	20	21	22	23	24	25
3	30	31	32	33	34	35
4	40	41	42	43	44	45
5	50	51	52	53	54	55

row \ col	0	1	2	3	4	5
0	0	1	2	3	4	5
1	10	11	12	13	14	15
2	20	21	22	23	24	25
3	30	31	32	33	34	35
4	40	41	42	43	44	45
5	50	51	52	53	54	55

Print out

```
[ 2. 12. 22. 32. 42. 52.]
```

Two Dimensional Array Slicing: VI

35

```
print(a[2::2, ::2])
```

	0	1	2	3	4	5
0	0	1	2	3	4	5
1	10	11	12	13	14	15
2	20	21	22	23	24	25
3	30	31	32	33	34	35
4	40	41	42	43	44	45
5	50	51	52	53	54	55

Two Dimensional Array Slicing: VI

35 `print(a[2::2, ::2])`

	0	1	2	3	4	5		0	1	2	3	4	5
0	0	1	2	3	4	5	0	0	1	2	3	4	5
1	10	11	12	13	14	15	1	10	11	12	13	14	15
2	20	21	22	23	24	25	2	20	21	22	23	24	25
3	30	31	32	33	34	35	3	30	31	32	33	34	35
4	40	41	42	43	44	45	4	40	41	42	43	44	45
5	50	51	52	53	54	55	5	50	51	52	53	54	55

print

```
[[20. 22. 24.]  
 [40. 42. 44.]]
```

Today's Topics



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

Today's Topics



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

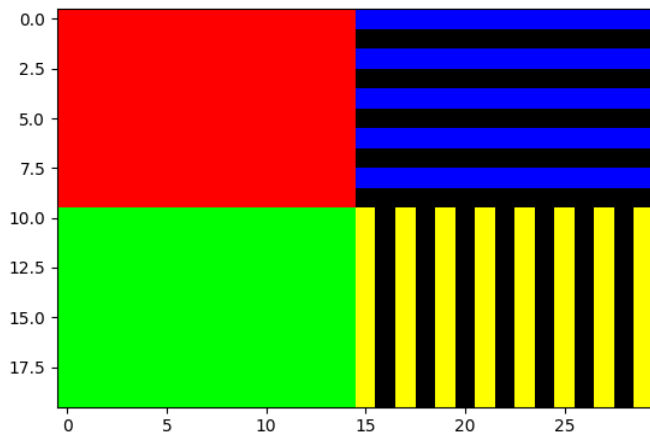
Image and Array

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 height= 20
5 width = 30
6
7 #An image is an array with height, width and
8 #depth 3 for r(ed) g(reen) b(lue)
9 img = np.zeros((height, width, 3))
10 img[:height//2, :width//2, 0] = 1
11 #which does this statement do? Same as
12 #img[:height//2, :width//2] = [1,0,0]
```

Image and Array: II

```
13 img[height//2:, :width//2, 1] = 1
14 #which does this statement do? Same as
15 #img[height//2:, :width//2] = [0,1,0]
16
17 img[:height//2:2, width//2:, 2] = 1
18 #What does this statement do?
19
20 img[height//2:, width//2::2] = [1, 1, 0]
21 #What does this statement do?
22
23 plt.imshow(img)
24 plt.show()
```


output for the above program



Challenge: Cropping Images

Crop an image to select the top quarter (upper left corner)



Challenge: Cropping Images

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 img = plt.imread("csBridge.png")
5 height = img.shape[0]
6 width = img.shape[1]
7 img2 = img[0:height//2, 0:width//2, :]
8 #img2 is top left of img. Same as
9 #img2 = img[:height//2, :width//2].
10 plt.imshow(img2)
11 plt.show()
12
13 plt.imsave("top_left_csBridge.png", img2)
```

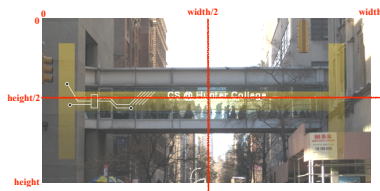
Challenge: Cropping Images

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



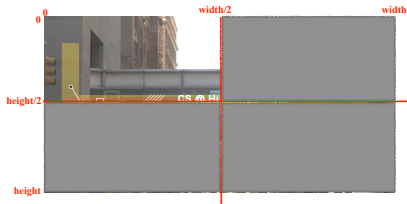
Challenge: Cropping Images

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



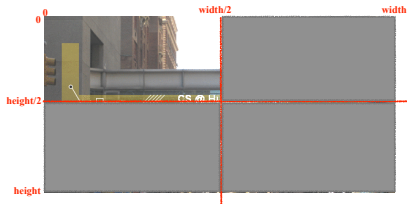
Challenge: Cropping Images

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



Challenge: Cropping Images

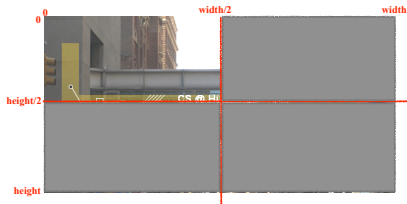
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



- How would you select the lower left corner?

Challenge: Cropping Images

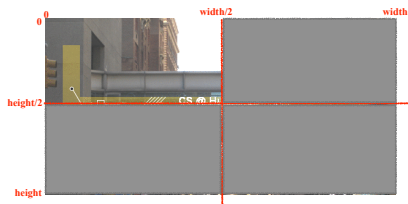
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



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

Challenge: Cropping Images

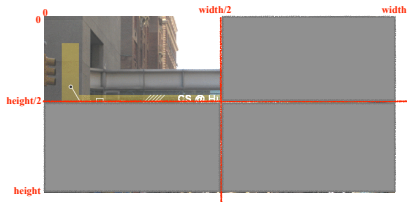
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



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

Challenge: Cropping Images

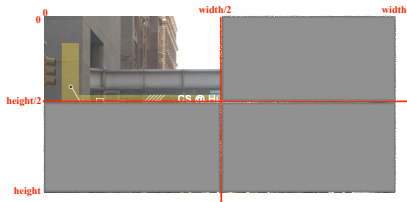
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



- 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: Cropping Images

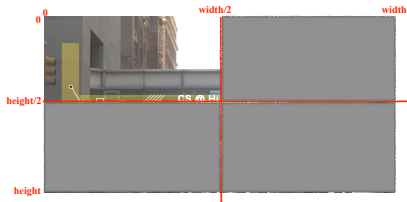
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



- 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: Cropping Images

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



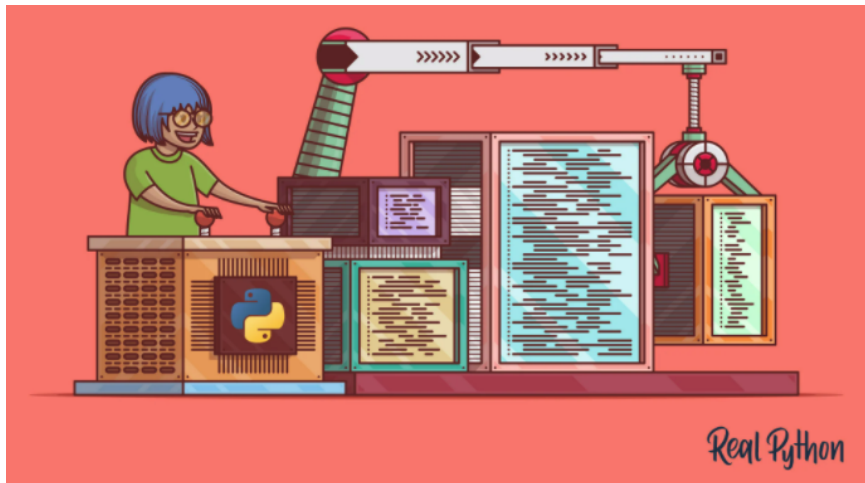
- How would you select the lower left corner?
`img2 = img[height//2:, :width//2]`
- How would you select the upper right corner?
`img2 = img[:height//2, width//2:]`
- How would you select the lower right corner?
`img2 = img[height//2:, width//2:]`

Today's Topics

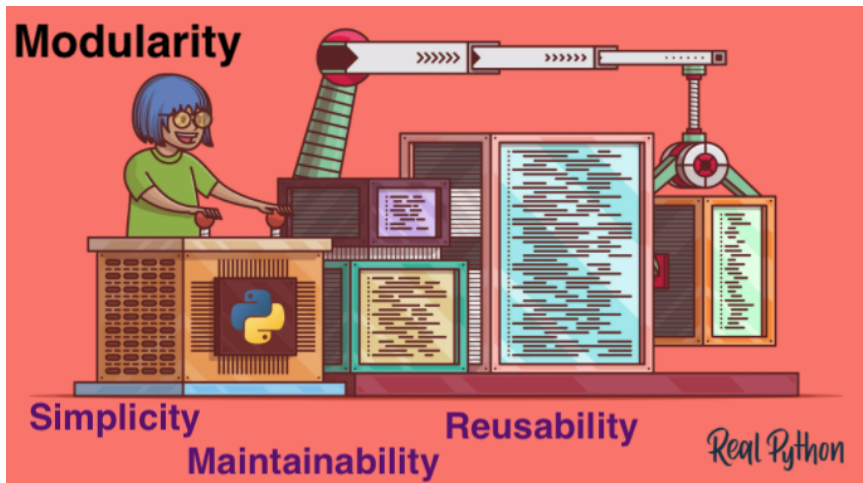


- Recap: Slicing & Images
- **Introduction to Functions**
- NYC Open Data

Modularity



Modularity



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()`
- Naming conventions same as variables

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()`
- Naming conventions same as variables
- 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()`
- Naming conventions same as variables
- 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()`
- Naming conventions same as variables
- 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()`
- Naming conventions same as variables
- 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()
```

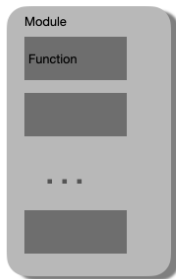

Python Tutor

```
#Name: your name here  
#Date: October 2017  
#This program, uses functions,  
#    says hello to the world!
```

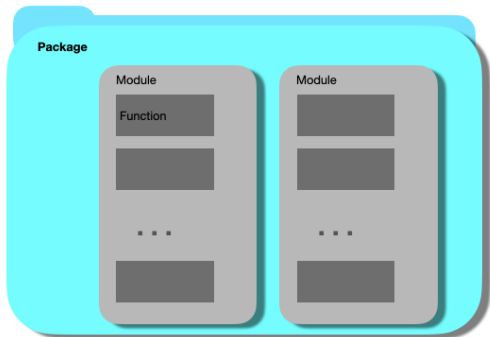
```
def main():  
    print("Hello, World!")  
  
if __name__ == "__main__":  
    main()
```

(Demo with pythonTutor)

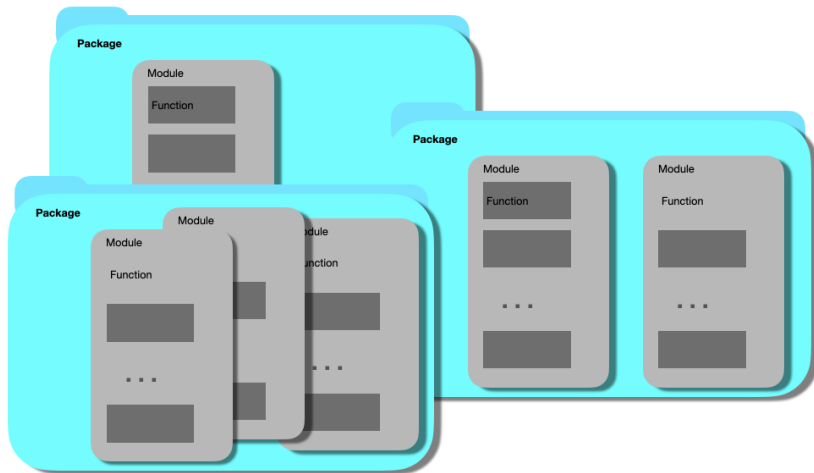
functions - modules - packages



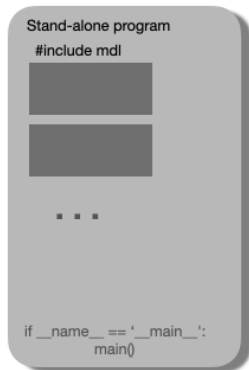
functions - modules - packages



functions - modules - packages



Stand-alone program



Challenge:

Predict what the code will do:

```
1 def totalWithTax(food,tip):  
2     total = 0  
3     tax = 0.1  
4     total = food + food * tax  
5     total = total + tip  
6     return(total)  
7  
8 lunch = float(input('Enter lunch total: '))  
9 lTip = float(input('Enter lunch tip: ' ))  
10 lTotal = totalWithTax(lunch, lTip)  
11 print('Lunch total is', lTotal)
```

totalWithTax function: continued

```
1 def totalWithTax(food,tip):  
2     total = 0  
3     tax = 0.1  
4     total = food + food * tax  
5     total = total + tip  
6     return(total)
```

Omit code to calculate lunch total...

```
12 dinner= float(input('Enter dinner total: '))  
13 dTip = float(input('Enter dinner tip: ' ))  
14 dTotal = totalWithTax(dinner, dTip)  
15 print('Dinner total is', dTotal)
```

Scope

```
def eight():  
    x = 5+3  
    print(x)  
  
def nine():  
    x = "nine"  
    print(x)
```

- You can have multiple functions.

Scope

```
def eight():  
    x = 5+3  
    print(x)  
  
def nine():  
    x = "nine"  
    print(x)
```

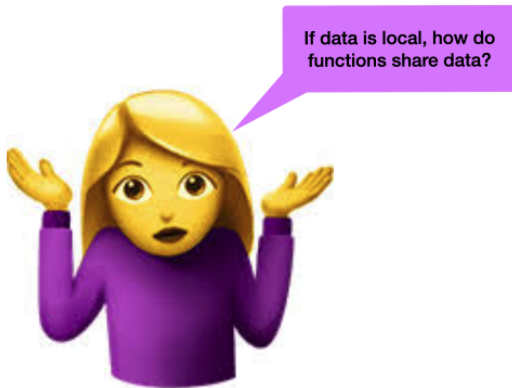
- You can have multiple functions.
- Each function defines the **scope** of its local variables

Scope

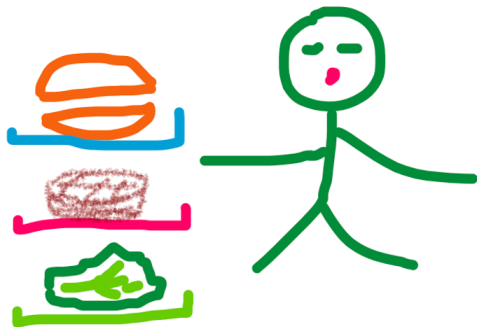
```
def eight():  
    x = 5+3  
    print(x)  
  
def nine():  
    x = "nine"  
    print(x)
```

- You can have multiple functions.
- Each function defines the **scope** of its local variables
- A variable defined inside a function is **local**, i.e. defined only inside that function.

Local Data?



Function Example: burger



Function name: `burger` (like a variable name, no space is allowed)

Input:

- `bread`: representing for bread layer
- `meat`: representing for meat layer
- `vegetable`: representing for vegetable layer

Return: a hamburger

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)
```

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**.

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.

Challenge:

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()
```

Challenge:

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 circled values to underlined parameter names. Red arrows, labeled "Formal Parameters", point from underlined parameter names to the function call.

- Actual Parameters (circled):** `verse` (in `mystery`), `verse, c` (in `enigma`), and `"jam"` (in `mystery`).
- Formal Parameters (underlined):** `prob4()`, `v` (in `mystery`), and `v, c` (in `enigma`).

Challenge:

Predict what the code will do:

```
1 def prob4():
2     verse = "jam tomorrow and jam yesterday,
3         "
4     print("The rule is,")
5     c = mystery(verse)
6     w = enigma(verse,c)
7     print(c,w)
8 def mystery(v):
9     print(v)
10    c = v.count("jam")
11    return(c)
12 def enigma(v,c):
```

Challenge:

Predict what the code will do:

```
1 def prob4():  
2     verse = "jam tomorrow and jam yesterday,"  
3     print("The rule is ,")  
4     c = mystery(verse)  
5     w = enigma(verse,c)  
6     print(c,w)
```

Omit code of function mystery.

```
11 def enigma(v,c):  
12     print("but never", v[-1])  
13     for i in range(c):  
14         print("jam")  
15     return("day. ")  
16 prob4()
```

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()
```

(Demo with pythonTutor)

Challenge:

Predict what the code will do:

```
1 # From "Teaching with Python" by John Zelle
2 def happy():
3     print("Happy Birthday to you!")
4
5 def sing(P):
6     happy()
7     happy()
8     print("Happy Birthday dear " + P + "!")
9     happy()
10
11 sing("Fred")
12 sing("Thomas")
```


Challenge:

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)

Github

- Used to collaborate on and share code, documents, etc.



Octocat

Github

- Used to collaborate on and share code, documents, etc.
- Supporting Open-Source Software: original source code is made freely available and may be redistributed and modified.



Octocat

Github



Octocat

- Used to collaborate on and share code, documents, etc.
- Supporting Open-Source Software: original source code is made freely available and may be redistributed and modified.
- More formally: `git` is a version control protocol for tracking changes and versions of documents.

Github



Octocat

- Used to collaborate on and share code, documents, etc.
- Supporting Open-Source Software: original source code is made freely available and may be redistributed and modified.
- More formally: `git` is a version control protocol for tracking changes and versions of documents.
- Github provides hosting for repositories (**'repos'**) of code.

Github



Octocat

- Used to collaborate on and share code, documents, etc.
- Supporting Open-Source Software: original source code is made freely available and may be redistributed and modified.
- More formally: `git` is a version control protocol for tracking changes and versions of documents.
- Github provides hosting for repositories (**'repos'**) of code.
- Also convenient place to host websites (i.e. `huntercsci127.github.io`).

Github



Octocat

- Used to collaborate on and share code, documents, etc.
- Supporting Open-Source Software: original source code is made freely available and may be redistributed and modified.
- More formally: `git` is a version control protocol for tracking changes and versions of documents.
- Github provides hosting for repositories (**'repos'**) of code.
- Also convenient place to host websites (i.e. `huntercsci127.github.io`).
- In Lab6 you set up github accounts to copy (**'clone'**) documents from the class repo. (More in future courses.)

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**

Design Challenge - Solution

Stars						
Temperature (K)	Luminosity(L/L _o)	Radius(R/R _o)	Absolute magnitude(M _v)	Star type	Star color	Spectral Class
3068	0.0024	0.17	16.12	Brown Dwarf	Red	M
25000	0.056	0.0084	10.58	White Dwarf	Blue White	B
2650	0.00069	0.11	17.45	Brown Dwarf	Red	M
11790	0.00015	0.011	12.59	White Dwarf	Yellowish White	F
15276	1136	7.2	-1.97	Main Sequence	Blue-white	B
5800	0.81	0.9	5.05	Main Sequence	yellow-white	F
16500	0.013	0.014	11.89	White Dwarf	Blue White	B
3192	0.00362	0.1967	13.53	Red Dwarf	Red	M
6380	1.35	0.98	2.93	Main Sequence	yellow-white	F
3834	272000	1183	-9.2	Hypergiant	Red	M

- **Libraries:** pandas

Design Challenge - Solution

Stars						
Temperature (K)	Luminosity(L/L _o)	Radius(R/R _o)	Absolute magnitude(M _v)	Star type	Star color	Spectral Class
3068	0.0024	0.17	16.12	Brown Dwarf	Red	M
25000	0.056	0.0084	10.58	White Dwarf	Blue White	B
2650	0.00069	0.11	17.45	Brown Dwarf	Red	M
11790	0.00015	0.011	12.59	White Dwarf	Yellowish White	F
15276	1136	7.2	-1.97	Main Sequence	Blue-white	B
5800	0.81	0.9	5.05	Main Sequence	yellow-white	F
16500	0.013	0.014	11.89	White Dwarf	Blue White	B
3192	0.00362	0.1967	13.53	Red Dwarf	Red	M
6380	1.35	0.98	2.93	Main Sequence	yellow-white	F
3834	272000	1183	-9.2	Hypergiant	Red	M

- **Libraries:** pandas
- **Process:**
 - ▶ Print **max** of '**Luminosity**' column

Design Challenge - Solution

Stars						
Temperature (K)	Luminosity(L/L _o)	Radius(R/R _o)	Absolute magnitude(M _v)	Star type	Star color	Spectral Class
3068	0.0024	0.17	16.12	Brown Dwarf	Red	M
25000	0.056	0.0084	10.58	White Dwarf	Blue White	B
2650	0.00069	0.11	17.45	Brown Dwarf	Red	M
11790	0.00015	0.011	12.59	White Dwarf	Yellowish White	F
15276	1136	7.2	-1.97	Main Sequence	Blue-white	B
5800	0.81	0.9	5.05	Main Sequence	yellow-white	F
16500	0.013	0.014	11.89	White Dwarf	Blue White	B
3192	0.00362	0.1967	13.53	Red Dwarf	Red	M
6380	1.35	0.98	2.93	Main Sequence	yellow-white	F
3834	272000	1183	-9.2	Hypergiant	Red	M

- **Libraries:** pandas
- **Process:**
 - ▶ Print **max** of '**Luminosity**' column
 - ▶ Print **min** of '**Temperature**' column

Design Challenge - Solution

Stars						
Temperature (K)	Luminosity(L/L _o)	Radius(R/R _o)	Absolute magnitude(M _v)	Star type	Star color	Spectral Class
3068	0.0024	0.17	16.12	Brown Dwarf	Red	M
25000	0.056	0.0084	10.58	White Dwarf	Blue White	B
2650	0.00069	0.11	17.45	Brown Dwarf	Red	M
11790	0.00015	0.011	12.59	White Dwarf	Yellowish White	F
15276	1136	7.2	-1.97	Main Sequence	Blue-white	B
5800	0.81	0.9	5.05	Main Sequence	yellow-white	F
16500	0.013	0.014	11.89	White Dwarf	Blue White	B
3192	0.00362	0.1967	13.53	Red Dwarf	Red	M
6380	1.35	0.98	2.93	Main Sequence	yellow-white	F
3834	272000	1183	-9.2	Hypergiant	Red	M

- **Libraries:** pandas
- **Process:**
 - ▶ Print **max** of '**Luminosity**' column
 - ▶ Print **min** of '**Temperature**' column
 - ▶ **groupby** '**Star Type**' and **get group** '**Hypergiant**' to print **average** '**Radius**'

Design Challenge - Code

- **Libraries:** pandas

```
import pandas as pd  
stars = pd.read_csv('Stars.csv')
```

Design Challenge - Code

- **Libraries:** pandas

```
import pandas as pd  
stars = pd.read_csv('Stars.csv')
```

- **Process:**

- ▶ Print **max** of '**Luminosity**' column

```
1 print(stars['Luminosity(L/Lo)'].max())
```

Design Challenge - Code

- **Libraries:** pandas

```
import pandas as pd  
stars = pd.read_csv('Stars.csv')
```

- **Process:**

- ▶ Print **max** of '**Luminosity**' column

```
1 print(stars['Luminosity(L/Lo)'].max())
```

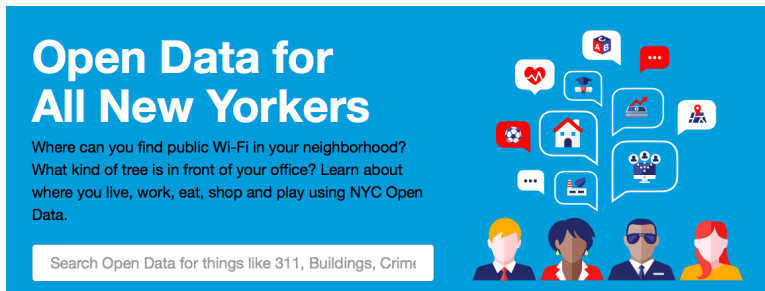
- ▶ Prints **min** of '**Temperature**' column and store it in temp variable

```
1 print(stars['Temperature(K)'].min())
```

- **groupby** 'Star Type' and get a group of **Hypergiant**, then print **average of 'Radius'** column for this group.

```
1 grouped = stars.groupby('Star type')
2 hypergiant = grouped.get_group('
  Hypergiant')
3 print("Hypergiant average radius:",
  hypergiant['Radius(R/Ro)'].mean())
```

Accessing Structured Data: NYC Open Data



Open Data for All New Yorkers

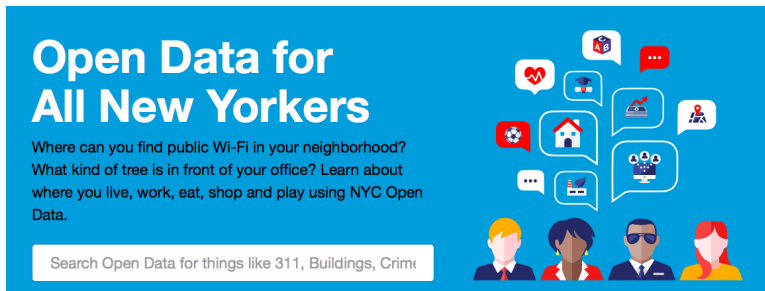
Where can you find public Wi-Fi in your neighborhood? What kind of tree is in front of your office? Learn about where you live, work, eat, shop and play using NYC Open Data.

Search Open Data for things like 311, Buildings, Crime

The banner features a blue background with white text. On the right, there are several speech bubbles containing icons for various city services: a heart with a pulse line, a graduation cap, a house, a soccer ball, a factory, a bar chart, a location pin, and a group of people. Below the speech bubbles are four stylized avatars of diverse people.

- Freely available source of data.

Accessing Structured Data: NYC Open Data



Open Data for All New Yorkers

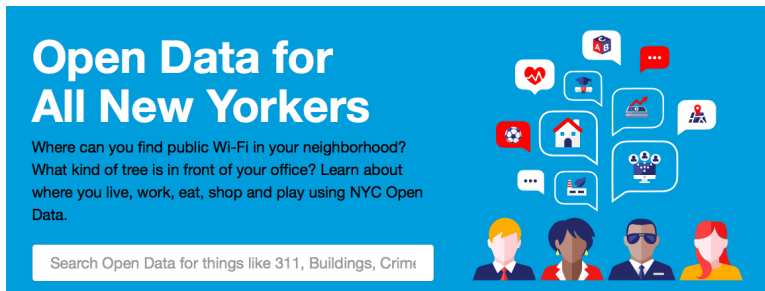
Where can you find public Wi-Fi in your neighborhood?
What kind of tree is in front of your office? Learn about where you live, work, eat, shop and play using NYC Open Data.

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 speech bubbles containing icons representing various data categories: 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 diverse New Yorkers.

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

Accessing Structured Data: NYC Open Data



Open Data for All New Yorkers

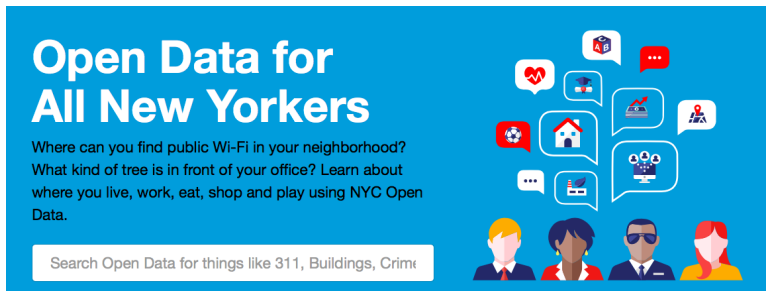
Where can you find public Wi-Fi in your neighborhood?
What kind of tree is in front of your office? Learn about where you live, work, eat, shop and play using NYC Open Data.

Search Open Data for things like 311, Buildings, Crime

The graphic features a blue background with white text. Below the title, there are two lines of text asking questions about finding public Wi-Fi and trees. Below that is a search bar with the text "Search Open Data for things like 311, Buildings, Crime". To the right of the text, there are several 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 person with a magnifying glass, and a group of people. At the bottom right, there are four stylized human figures in different colors (yellow, blue, black, and red) wearing various outfits.

- 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

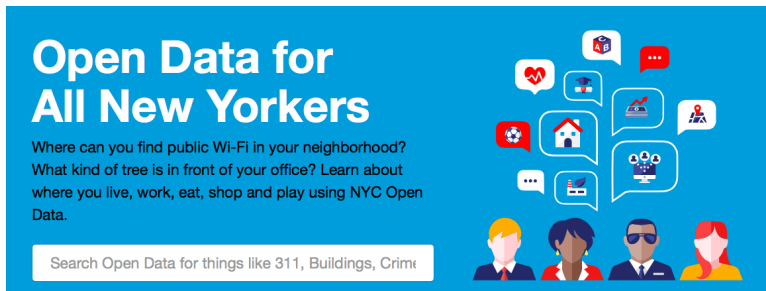
Where can you find public Wi-Fi in your neighborhood?
What kind of tree is in front of your office? Learn about where you live, work, eat, shop and play using NYC Open Data.

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 and styles.

- 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, 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.
- 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	StartDateT...	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> 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?

Example: OpenData Film Permits



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>

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/nycopendata/when-permit-required.page>

EventID	EventType	StartDateT...	EndDateTime	EnteredOn	EventAg...	ParkingInf...	Borne...	Cent...	Police...	Categ...	SubC...	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 09:44...	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 betwe...	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 betwe...	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 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...	ParkingInf...	Borne...	Cent...	Police...	Categ...	SubC...	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...	12/06/2018 09:11...	Mayor's Offi...	EAGLE STREET bet...	Brooklyn	1	84	Television	Episodic S...	United Sta...	11222
45481	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/06/2018 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 11:00...	12/06/2018 03:28...	Mayor's Offi...	13 AVENUE betwe...	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 09:05...	Mayor's Offi...	ELDERST STREET b...	Brooklyn	4, 6	104, 76, 83	Television	Episodic 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	Episodic S...	United Sta...	11227
454865	Shooting Permit	12/06/2018 07:00...	12/06/2018 10:00...	12/06/2018 02:17...	Mayor's Offi...	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
```

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	EnteredAt...	EventAg...	ParkingInf...	Borne...	Cent...	Police...	Categ...	SubC...	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	Epicodic S...	United Sta...	11101
45467	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/06/2018 09:11...	Mayor's Offi...	EAGLE STREET bet...	Brooklyn	1	84	Television	Epicodic S...	United Sta...	11222
45491	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/06/2018 09:44...	Mayor's Offi...	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...	12/06/2018 03:28...	Mayor's Offi...	13 AVENUE betwe...	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 09:05...	Mayor's Offi...	ELBERT STREET b...	Brooklyn	4, 6	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...	ELBERT 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 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
```

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	EnteredAt...	EventAg...	ParkingHeld	Borne...	Cent...	Police...	Camp...	SubC...	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 bet...	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 09:44...	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...	12/04/2018 03:28...	Mayor's Offi...	13 AVENUE betwe...	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 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 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)
```

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	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, 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 Office	13 AVENUE betwe...	Queens	1, 3, 7	108, 7, 96	Film	Feature	United Sta...	10002, 11...
454914	Shooting Permit	12/06/2018 08:00	12/06/2018 11:00	12/04/2018 09:05	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 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
```

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	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 bet...	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, 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 Office	13 AVENUE betwe...	Queens	1, 3, 7	108, 7, 96	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 |

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?

Recap

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



Recap



- **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



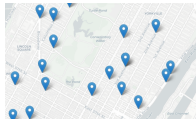
- **Functions** are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
Example: `print("Hello", "World")`

Recap



- **Functions** 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** 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



- **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

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.

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
 - ▶ 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(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
 - ▶ repeat.
 - Past exams are on the webpage (under [Final Exam Information](#)).
 - Theme: Functions!
- Starting with Spring 19 V3, #4(b).

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments (**programs 31-35**)

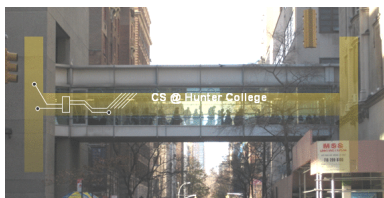
Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments (**programs 31-35**)
- If you need help, schedule an appointment for Tutoring in lab 1001G 11:30am-5:30pm

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments (**programs 31-35**)
- If you need help, schedule an appointment for Tutoring in lab 1001G 11:30am-5:30pm
- Take the Lecture Preview on Blackboard on Monday (or no later than 10:15am on Tuesday)

Lecture Slips & Writing Boards



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