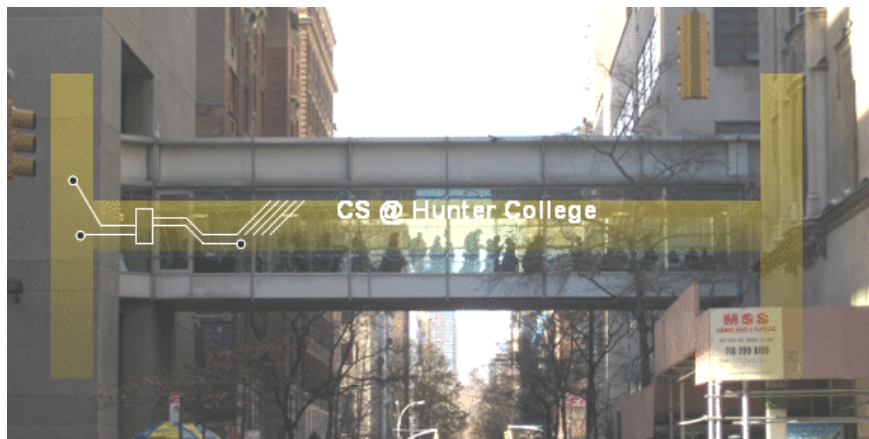


CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Today's Topics



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

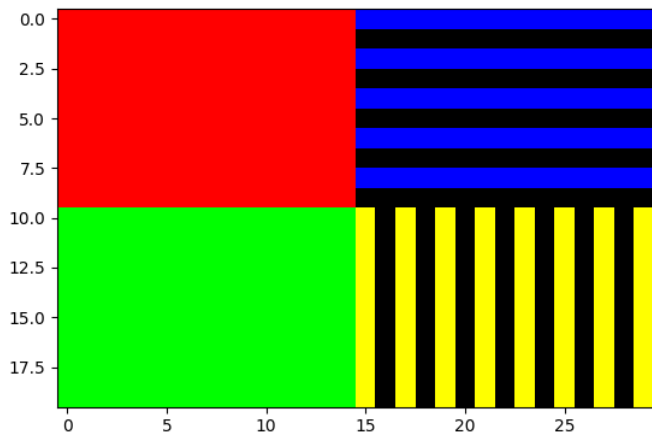
Image and Array

```
1 import matplotlib.pyplot as plt
2 import numpy as np #code link in trinket
3 #code link in replit
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 #link to replit
3 #In replit, if do not see cropped image,
   click Tools in left pane, choose Output.
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 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()
```



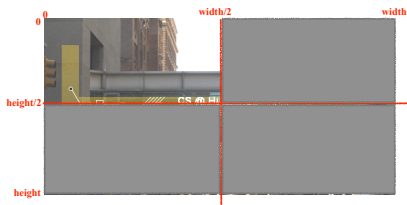
Challenge: Cropping Images

```
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()
```



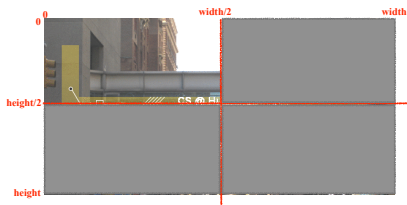
Challenge: Cropping Images

```
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

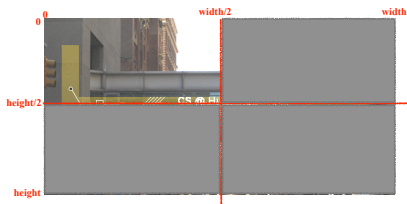
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



- 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()
```

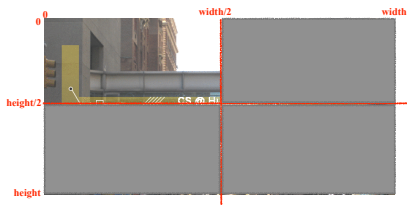


- How would you select the lower left corner?

```
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()
```



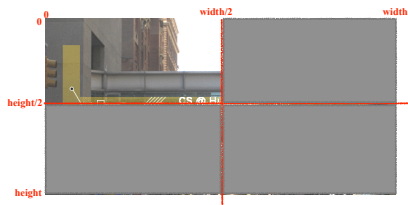
- How would you select the lower left corner?

```
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?

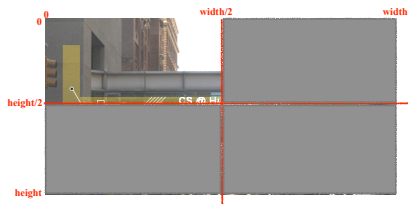
```
img2 = img[height//2:, :width//2]
```

- 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]
```

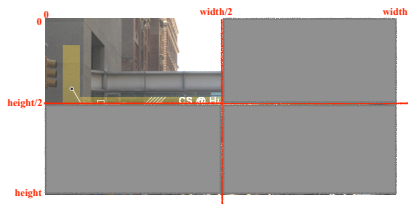
- How would you select the upper right corner?

```
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?

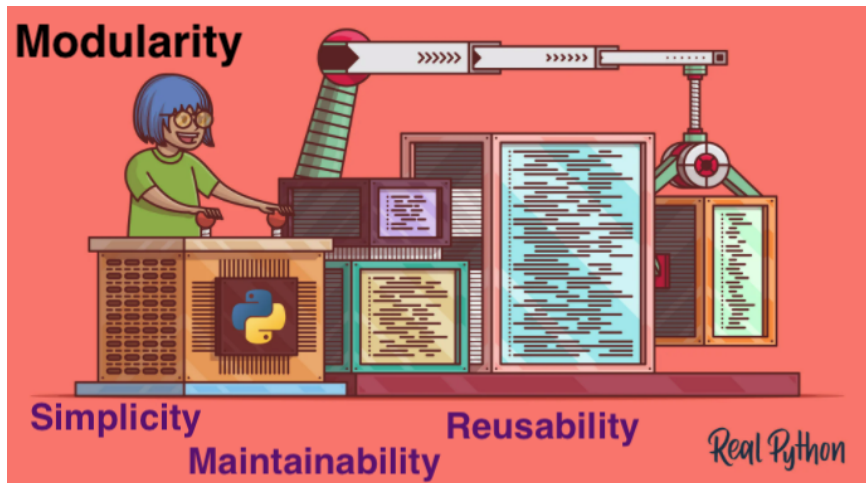
```
img2 = img[height//2:, width//2:]
```


Today's Topics



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

Modularity



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()
```

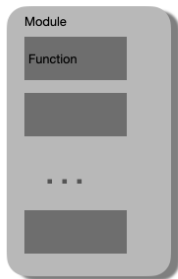
- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- Naming conventions same as variables
- You **call** or **invoke** a function by typing its name, followed 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

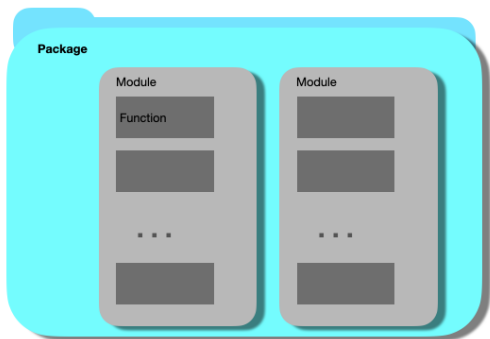
[link in PythonTutor](#)

```
1 #Name: your name here
2 #Date: March 2017
3 #This program, uses functions,
4 #   says hello to the world!
5
6 def main():
7     print("Hello, World!")
8
9 if __name__ == "__main__":
10    main()
```

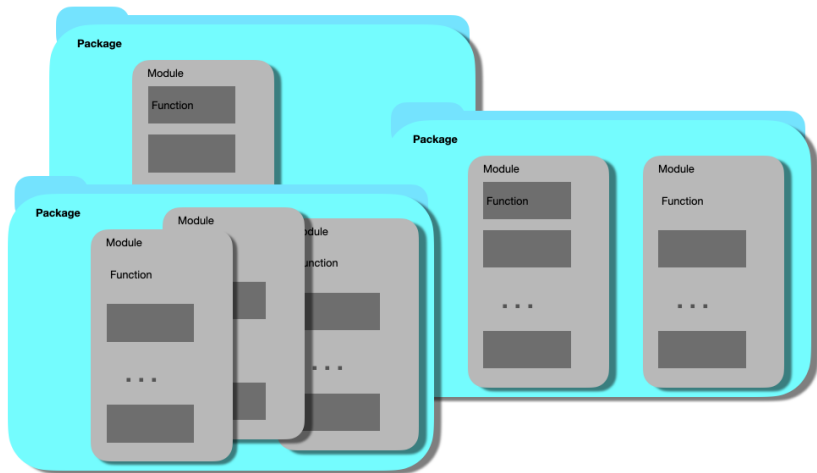
functions - modules - packages



functions - modules - packages



functions - modules - packages



Stand-alone program

```
Stand-alone program
#include mdl
[Redacted]
[Redacted]
...

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


Challenge: *Predict what the code will do:*

```
1 def totalWithTax(food,tip): #link in PythonTutor
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): #link in PythonTutor
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?



If data is local, how do functions share 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

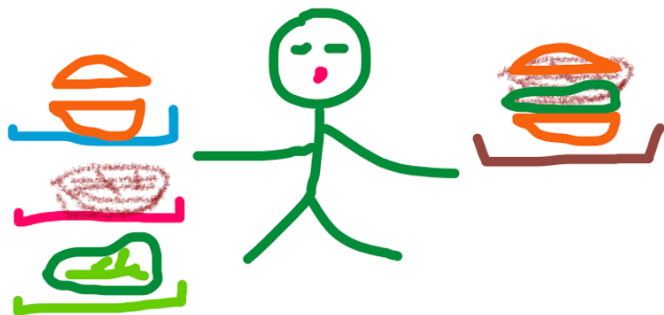
Return: a hamburger

Burger function definition

Pseudocode of burger function.

```
1 def burger(bread, meat, veg):  
2     pick a bread, put on top  
3     put meat  
4     put vegetable  
5     put a bread at the bottom  
6  
7     return the burger made
```


Pseudocode to call burger function



```
1 def main():  
2     myBurger = burger("wheat bread", "beef",  
3     "lettuce")  
     eat myBurger
```

Input Parameters & Return Values

- Functions can have **input parameters**.

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

Input Parameters & Return Values

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

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

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

Input Parameters & Return Values

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

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

- Functions can have **input parameters**.
- Surrounded by parentheses, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.

Input Parameters & Return Values

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

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

- Functions can have **input parameters**.
- Surrounded by parentheses, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**

Input Parameters & Return Values

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

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()
```

Actual Parameters

Formal Parameters

Challenge: *Predict what the code will do:*

```
1 def prob4(): #link in PythonTutor
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)
7 def mystery(v):
8     print(v)
9     c = v.count("jam")
10    return(c)
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()
```

Challenge: *Predict what the code will do:*

```
1 def prob4(): #link in PythonTutor
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()
```

Challenge: *Predict what the code will do:*

```
1 # From "Teaching with Python" by John Zelle
2 def happy(): #link to PythonTutor
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")
13 sing("Hunter")
```

Challenge: *Fill in the missing code:*

```
1 def monthString(monthNum): #link in PythonTutor
2     """
3     Takes as input a number, monthNum, and
4     returns the corresponding month name as a string.
5     Example: monthString(1) returns "January".
6     Assumes that input is an integer ranging from 1 to 12
7     """
8
9     monthString = ""
10
11     #####
12     ### FILL IN YOUR CODE HERE ###
13     ### Other than your name above, ###
14     ### this is the only section    ###
15     ### you change in this program. ###
16     #####
17
18     return(monthString)
19
20
21 def main():
22     n = int(input('Enter the number of the month: '))
23     mString = monthString(n)
24     print('The month is', mString)
```

Define monthString

```
1 def monthString(monthNum): #link in PythonTutor
2     monthString = ""
3
4     if monthNum == 1:
5         monthString = "January"
6     elif monthNum == 2:
7         monthString = "February"
8     elif monthNum == 3:
9         monthString = "March"
10    #... Omit code when monthNum in [4,11]
11    elif monthNum == 12:
12        monthString = "December"
13
14    return(monthString)
```

Another solution to define monthString

```
1 def monthString(month): #link in PythonTutor
2     monthNames = ['January', 'February', 'March', 'April', \
3     'May', 'June', 'July', 'August', 'September', \
4     'October', 'November', 'December']
5     #\ means connect the next line
6     #if you have codes spread more than one line ,
7     #you can use \ to connect these lines .
8
9     if month < 1 or month > 12:
10        return ""
11    else:
12        return monthNames[month-1]
13        #if month == 1, return monthName[0],
14        ##if month == 2, return monthName[1],
15        #...
16        #if month == 12, return monthName[11].
```

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

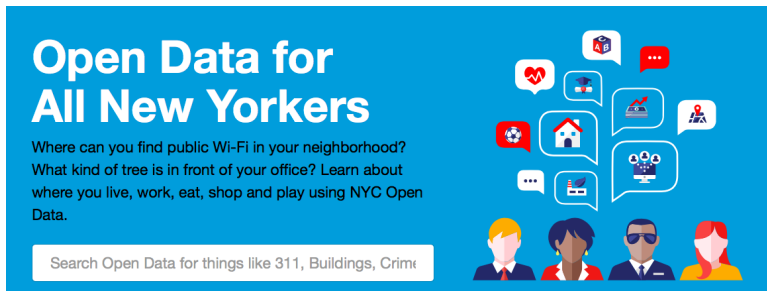
```
1 print(stars['Temperature(K)'].min())
```

- **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())
```

[Link in replit](#)

Accessing Structured Data: NYC Open Data



Open Data for All New Yorkers

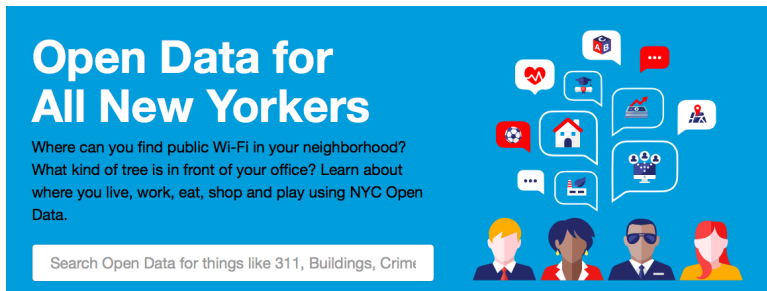
Where can you find public Wi-Fi in your neighborhood?
What kind of tree is in front of your office? Learn about where you live, work, eat, shop and play using NYC Open Data.

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 for various services: 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 Wi-Fi symbol, and a group of people. Below the speech bubbles are four stylized human figures representing diverse individuals: a man with blonde hair, a woman with dark skin and hair, a man with sunglasses, and a woman with red hair.

- Freely available source of data.

Accessing Structured Data: NYC Open Data



Open Data for All New Yorkers

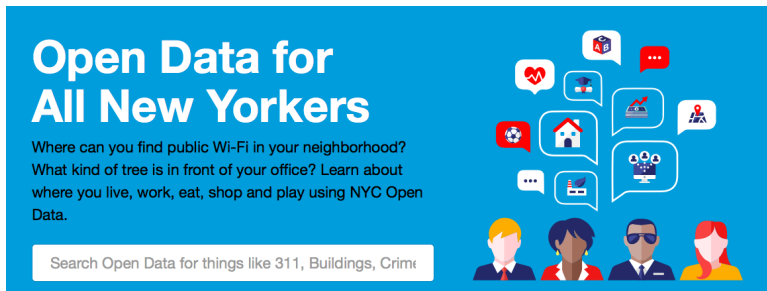
Where can you find public Wi-Fi in your neighborhood?
What kind of tree is in front of your office? Learn about where you live, work, eat, shop and play using NYC Open Data.

Search Open Data for things like 311, Buildings, Crim

The banner features a blue background with white text. On the right side, there are several white speech bubbles containing icons for a heart, a graduation cap, a house, a soccer ball, a bar chart, a location pin, a Wi-Fi symbol, and a group of people. Below the speech bubbles are four stylized human figures representing diverse individuals.

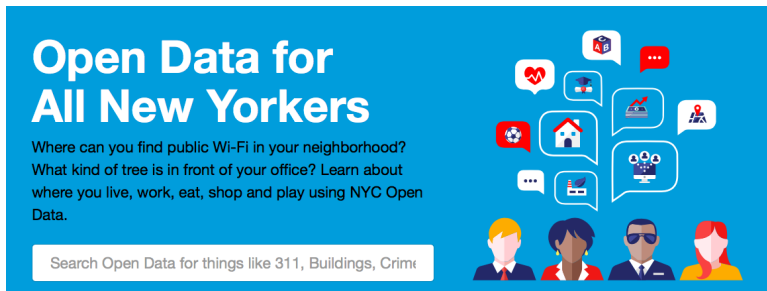
- Freely available source of data.
- Maintained by the NYC data analytics team.

Accessing Structured Data: NYC Open Data

A promotional graphic for NYC Open Data. It features a blue background with the title "Open Data for All New Yorkers" in large white text. Below the title, there are two lines of text: "Where can you find public Wi-Fi in your neighborhood?" and "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, there is a white search bar with the text "Search Open Data for things like 311, Buildings, Crim". On the right side, there are several speech bubbles containing icons for various services: 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 gear, and a person with a magnifying glass. At the bottom right, there are four stylized human figures representing diverse people: a man with blonde hair, a woman with dark skin and hair, a man with sunglasses, and a woman with red hair.

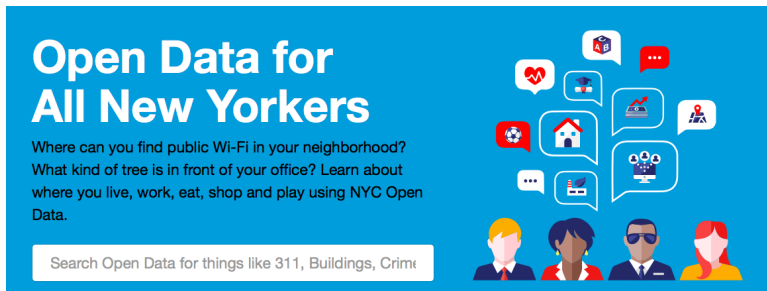
- Freely available source of data.
- Maintained by the NYC data analytics team.
- We will use several different ones for this class.

Accessing Structured Data: NYC Open Data

A blue banner for NYC Open Data. On the left, the text reads "Open Data for All New Yorkers" in large white font. Below it, a smaller white font 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 of the banner is a white search bar with the text "Search Open Data for things like 311, Buildings, Crim". On the right side of the banner, there are several white speech bubbles containing various 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 figures representing diverse people: a man with blonde hair, a woman with dark skin and hair, a man with sunglasses, and a woman with red hair.

- 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

A promotional graphic for NYC Open Data. It features a blue background with the title "Open Data for All New Yorkers" in large white text. Below the title is a paragraph of text: "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 text "Search Open Data for things like 311, Buildings, Crim". On the right side, there are several colorful icons representing different data categories: 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 person with a magnifying glass. At the bottom right, there are four stylized human figures representing diverse people.

- 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

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

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	StartDateT...	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...	ELDERT 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...	ELDERT 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

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...	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...	ELDERT 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...	ELDERT 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

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...	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...	ELDERT 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...	ELDERT 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 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/when-permits-required.page>

Find in this Dataset

More Views Filter Visualize Export Discuss Embed About

EventID	Event Type	Start Date/Time	End Date/Time	EventID	EventID	Parking/land	Boro.	Cont.	Permis.	Comp.	SubC.	Event.	Zip Co.
45363	Shooting Permit	12/06/2018 07:00.	12/06/2018 09:00.	12/05/2018 12:35.	Mayor's Offi...	STARBUCKS b...	Queens	2	108	Television	Episode 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...	GALLE STREET b...	Brooklyn	1	84	Television	Episode S...	United Sta...	11222
45481	Shooting Permit	12/06/2018 07:00.	12/06/2018 07:00.	12/06/2018 05:44.	Mayor's Offi...	SOUTH OXFORD ...	Brooklyn	2	76, 88	Still Photo...	Not Applic...	United Sta...	11217, 11...
45400	Shooting Permit	12/06/2018 13:00.	12/06/2018 11:00.	12/06/2018 03:20.	Mayor's Offi...	13 AVENUE betw...	Queens	1, 2, 7	108, 7, 96	Film	Feature	United Sta...	10002, 11...
45494	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, 5	104, 76, 89	Television	Episode S...	United Sta...	11207, 11...
45489	Shooting Permit	12/05/2018 08:00.	12/05/2018 06:00.	12/06/2018 02:45.	Mayor's Offi...	ELBERT STREET b...	Brooklyn	4	83	Television	Episode 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.

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/nycopen/permits/when-permits-required.page>

More Views Filter Visualize Export Discuss Embed About

EventID	EventType	StartDate	EndDate	EventDate	EventTag	ParkingInfo	Boro	Cont.	Permis.	Comp.	SubC.	Exam.	ZipCo.
45363	Shooting Permit	12/06/2018 07:00	12/06/2018 09:00	12/05/2018 12:35	Mayor's Offi...	STARBUCKS AVE...	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...	GALLE 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	Still Photo...	Not Applic...	United Sta...	11217, 11...
45400	Shooting Permit	12/06/2018 13:00	12/06/2018 11:00	12/06/2018 03:20	Mayor's Offi...	13 AVENUE betw...	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	12/06/2018 09:05	Mayor's Offi...	ELBERT STREET b...	Brooklyn	4, 5	104, 76, 89	Television	Episodic S...	United Sta...	11207, 11...
45489	Shooting Permit	12/05/2018 08:00	12/05/2018 06:00	12/06/2018 02:45	Mayor's Offi...	ELBERT STREET b...	Brooklyn	4	83	Television	Episodic S...	United Sta...	11207
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
```

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/nycopen/permits/when-permits-required.page>

More Views Filter Visualize Export Discuss Embed About

EventID	EventType	StartDate	EndDate	EventDate	EventOrg	ParkingInfo	Boro	Com.	Permis.	Comp.	SubC.	Exam.	ZipCo.			
45363	Shooting Permit	12/06/2018 07:00	12/06/2018 09:00	12/05/2018 12:35	Mayor's Offi...	STARBUCK AVENUE S...	Queens	2	108	Television	Episode 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...	GALLE STREET bet...	Brooklyn	1	84	Television	Episode s...	United Sta...	11222			
45481	Shooting Permit	12/06/2018 07:00	12/06/2018 07:00	12/06/2018 05:44	Mayor's Offi...	SOUTH OXFORD ...	Brooklyn	2	76	88	Still Photo...	Not Applic...	United Sta...	11217, 11...		
45400	Shooting Permit	12/06/2018 13:00	12/06/2018 11:00	12/06/2018 03:20	Mayor's Offi...	13 AVENUE betwe...	Queens	1	17	108	7	96	Film	Feature	United Sta...	10002, 11...
45414	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	8	108	76	83	Television	Episode s...	United Sta...	11207, 11...
45489	Shooting Permit	12/05/2018 08:00	12/05/2018 06:00	12/06/2018 02:45	Mayor's Offi...	ELBERT STREET b...	Brooklyn	4	83	Television	Episode 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 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: [link in replit](#)

```
#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

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-permits-required.page>

More Views | Filter | Visualize | Export | Discuss | Embed | About

EventID	EventType	StartDate	EndDate	EventDate	EventName	ParkingHeld	Block	Comm.	Permits	Comp.	SubC.	Event	ZipCode
45363	Shooting Permit	12/06/2018 07:00	12/06/2018 09:00	12/05/2018 12:35	Mayor's Offi...	STARBUCKS AVENUE S...	Queens	2	108	Television	Episodic L...	United Sta...	11101
45467	Shooting Permit	12/06/2018 07:00	12/06/2018 09:00	12/06/2018 09:11	Mayor's Offi...	GALLIE STREET B...	Brooklyn	1	84	Television	Episodic L...	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	76, 88	TV Photo...	Not Applica...	United Sta...	11217, 11...
45400	Shooting Permit	12/06/2018 12:00	12/06/2018 11:00	12/06/2018 03:20	Mayor's Offi...	13 AVENUE betwe...	Queens	1, 2, 7	108, 7, 96	Film	Feature	United Sta...	10002, 11...
45414	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, 5	104, 76, 89	Television	Episodic L...	United Sta...	11207, 11...
45489	Shooting Permit	12/05/2018 08:00	12/05/2018 06:00	12/06/2018 02:45	Mayor's Offi...	ELBERT STREET b...	Brooklyn	4	83	Television	Episodic L...	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 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 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/nycopen/permits/when-permits-required.page>

More Views Filter Visualize Export Discuss Embed About

EventID	EventType	StartEventTime	EndEventTime	EnteredAt	EventAg.	ParkingHeld	Block	Com.	Permis.	Comp.	SubC.	Exam.	ZipCo.
45363	Shooting Permit	12/06/2018 07:00	12/06/2018 09:00	12/05/2018 12:35	Mayor's Offi...	STARBUCKS AVE...	Queens	2	108	Television	Episodic L...	United Sta...	11101
45467	Shooting Permit	12/06/2018 07:00	12/06/2018 09:00	12/06/2018 09:11	Mayor's Offi...	GALLE STREET b...	Brooklyn	1	84	Television	Episodic L...	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	76, 88	Still Photo...	Not Applic...	United Sta...	11217, 11...
45400	Shooting Permit	12/06/2018 13:00	12/06/2018 11:00	12/06/2018 03:20	Mayor's Offi...	13 AVENUE betw...	Queens	1, 3, 7	108, 7, 96	Film	Feature	United Sta...	10002, 11...
45494	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, 5	108, 76, 88	Television	Episodic L...	United Sta...	11207, 11...
45489	Shooting Permit	12/05/2018 08:00	12/05/2018 06:00	12/06/2018 02:45	Mayor's Offi...	ELBERT STREET b...	Brooklyn	4	83	Television	Episodic L...	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 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

Film Permits

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a park, or a park. See <http://www.nyc.gov/open-data/permits-when-permits-are-required>

EventID	EventType	StartDateTime	EndDateTime	EnteredAt	EventAg.	ParkingHeld	Borne.	Com.	Permis.	Comp.	SubC.	Exam.	ZipCo.
45363	Shooting Permit	12/05/2018 07:00	12/05/2018 09:00	12/05/2018 12:35	Mayor's Offi...	STARBUCKS AVE. S...	Queens	2	108	Television	Episode 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...	GALLE STREET B...	Brooklyn	1	84	Television	Episode S...	United Sta...	11222
45481	Shooting Permit	12/06/2018 07:00	12/06/2018 07:00	12/06/2018 05:44	Mayor's Offi...	SOUTH OXFORD ...	Brooklyn	2	76, 88	Still Photo...	Not Applic...	United Sta...	11217, 11...
45400	Shooting Permit	12/06/2018 13:00	12/06/2018 11:00	12/06/2018 03:20	Mayor's Offi...	13 AVENUE BENE...	Queens	1, 2, 7	108, 7, 96	Film	Feature	United Sta...	10002, 11...
45414	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, 5	108, 76, 83	Television	Episode S...	United Sta...	11207, 11...
45489	Shooting Permit	12/05/2018 08:00	12/05/2018 06:00	12/06/2018 02:45	Mayor's Offi...	ELBERT STREET B...	Brooklyn	4	83	Television	Episode 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 BENE...	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

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	StartDateT...	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 bew...	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 betw...	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 Challenge

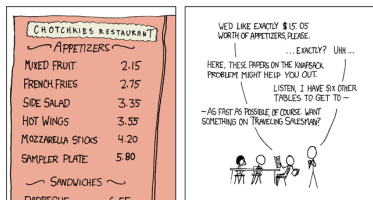
MY HOBBY: EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS

CHOTCHKIES RESTAURANT	
~ APPETIZERS ~	
MIXED FRUIT	2.15
FRENCH FRIES	2.75
SIDE SALAD	3.35
HOT WINGS	3.55
MOZZARELLA STICKS	4.20
SAMPLER PLATE	5.80
~ SANDWICHES ~	
BARBECUE	6.55



Design Challenge

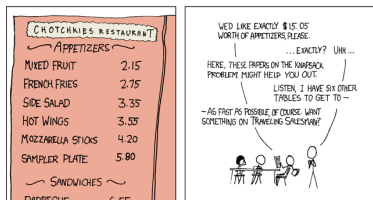
MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS



- Possible solutions:

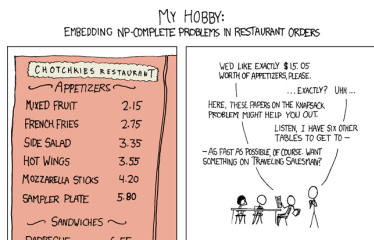
Design Challenge

MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS



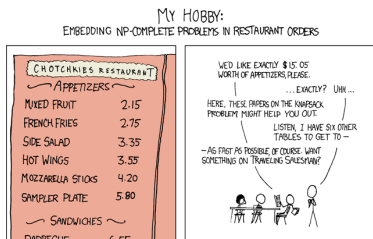
- Possible solutions:
 - ▶ 7 orders of mixed fruit, or

Design Challenge



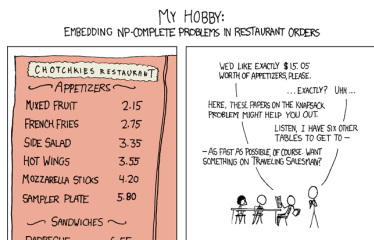
- Possible solutions:
 - ▶ 7 orders of mixed fruit, or
 - ▶ 2 orders hot wings, 1 order mixed fruit, and 1 sampler plate.

Design Challenge



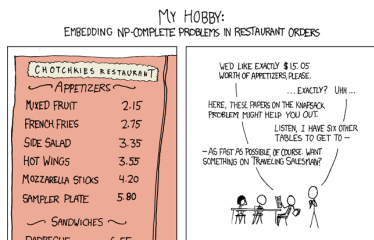
- Possible solutions:
 - ▶ 7 orders of mixed fruit, or
 - ▶ 2 orders hot wings, 1 order mixed fruit, and 1 sampler plate.
- **Input:** List of items with prices and amount to be spent.

Design Challenge



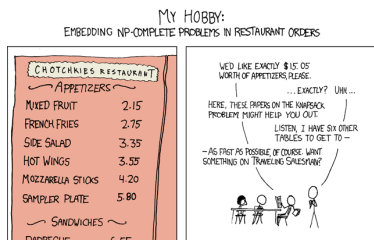
- Possible solutions:
 - ▶ 7 orders of mixed fruit, or
 - ▶ 2 orders hot wings, 1 order mixed fruit, and 1 sampler plate.
- **Input:** List of items with prices and amount to be spent.
- **Output:** An order that totals to the amount or empty list if none.

Design Challenge



- Possible solutions:
 - ▶ 7 orders of mixed fruit, or
 - ▶ 2 orders hot wings, 1 order mixed fruit, and 1 sampler plate.
- **Input:** List of items with prices and amount to be spent.
- **Output:** An order that totals to the amount or empty list if none.
- Possible algorithms: For each item on the list, divide total by price. If no remainder, return a list of that item. Repeat with two items, trying 1 of the first, 2 of the first, etc. Repeat with three items, etc.

Design Challenge



- Possible solutions:
 - ▶ 7 orders of mixed fruit, or
 - ▶ 2 orders hot wings, 1 order mixed fruit, and 1 sampler plate.
- **Input:** List of items with prices and amount to be spent.
- **Output:** An order that totals to the amount or empty list if none.
- Possible algorithms: For each item on the list, divide total by price. If no remainder, return a list of that item. Repeat with two items, trying 1 of the first, 2 of the first, etc. Repeat with three items, etc.
- “NP-Complete” problem: possible answers can be checked quickly, but not known how to compute quickly.

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():
    verse = "jan tomorrow and jan yesterday."
    print("The rule is.")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)

def mystery(s):
    print(s)
    c = v.count("jan")
    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():
    verse = "jan tomorrow and jan yesterday."
    print("The rule is.")
    c = mystery(verse)
    w = enigma(verse,c)
    print(c,w)
def mystery(s):
    print(s)
    c = v.count("jan")
    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)

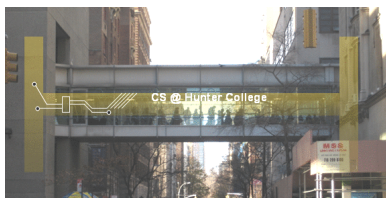
lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

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

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

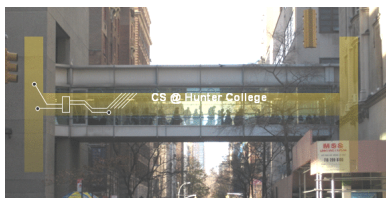
Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab

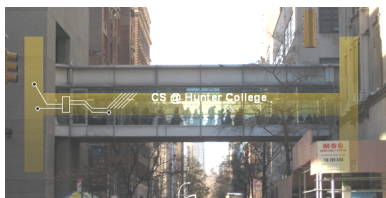
Weekly Reminders!



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

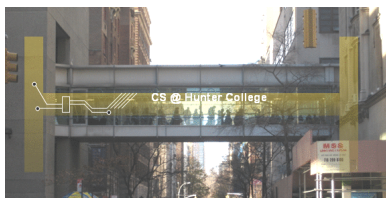
Weekly Reminders!



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

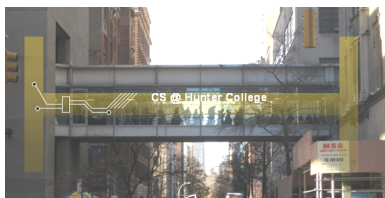
Weekly Reminders!



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**)

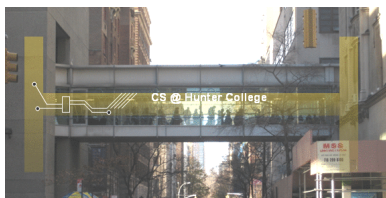
Weekly Reminders!



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.