

CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Frequently Asked Questions

From email

Frequently Asked Questions

From email

Frequently Asked Questions

From email

- **How do I know the height and width of an image?**

Frequently Asked Questions

From email

- **How do I know the height and width of an image?**

When you read an image file using pyplot, you can access the number of rows (height) and the number of columns (width) using the shape attribute of a numpy array.

Frequently Asked Questions

From email

- **How do I know the height and width of an image?**

When you read an image file using pyplot, you can access the number of rows (height) and the number of columns (width) using the shape attribute of a numpy array.

We will start with that today.

Frequently Asked Questions

From email

- **How do I know the height and width of an image?**

When you read an image file using pyplot, you can access the number of rows (height) and the number of columns (width) using the shape attribute of a numpy array.

We will start with that today.

- **Why are we looking at NYC historical population and CUNY enrollment data?**

Frequently Asked Questions

From email

- **How do I know the height and width of an image?**

When you read an image file using pyplot, you can access the number of rows (height) and the number of columns (width) using the shape attribute of a numpy array.

We will start with that today.

- **Why are we looking at NYC historical population and CUNY enrollment data?**

We are showing you how to access and analyze data. The tools we are exploring can be applied to many different datasets.

Frequently Asked Questions

From email

- **How do I know the height and width of an image?**

When you read an image file using pyplot, you can access the number of rows (height) and the number of columns (width) using the shape attribute of a numpy array.

We will start with that today.

- **Why are we looking at NYC historical population and CUNY enrollment data?**

We are showing you how to access and analyze data. The tools we are exploring can be applied to many different datasets.

We will explore many more in the coming weeks!

- **What is the difference between [] and ()?**

Frequently Asked Questions

From email

- **How do I know the height and width of an image?**

When you read an image file using pyplot, you can access the number of rows (height) and the number of columns (width) using the shape attribute of a numpy array.

We will start with that today.

- **Why are we looking at NYC historical population and CUNY enrollment data?**

We are showing you how to access and analyze data. The tools we are exploring can be applied to many different datasets.

We will explore many more in the coming weeks!

- **What is the difference between [] and ()?**

Parenthesis () generally follow function names, e.g. `print()`.

You may also find them in mathematical and boolean expressions, e.g. `(x == 2(y+3))` and `(x < 10)`*

Frequently Asked Questions

From email

- **How do I know the height and width of an image?**

When you read an image file using pyplot, you can access the number of rows (height) and the number of columns (width) using the shape attribute of a numpy array.

We will start with that today.

- **Why are we looking at NYC historical population and CUNY enrollment data?**

We are showing you how to access and analyze data. The tools we are exploring can be applied to many different datasets.

We will explore many more in the coming weeks!

- **What is the difference between [] and ()?**

Parenthesis () generally follow function names, e.g. print().

You may also find them in mathematical and boolean expressions, e.g. ($x == 2(y+3)$) and ($x < 10$)*

We use square brackets [] to index or slice,

i.e. take a piece, of a string, list or numpy array: my_string[2:5]

Today's Topics



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

Today's Topics



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

Challenge: Cropping Images

Crop an image to select the top quarter (upper 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()
```

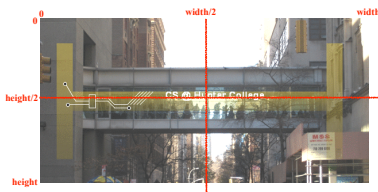
Challenge: Cropping Images

```
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()
```



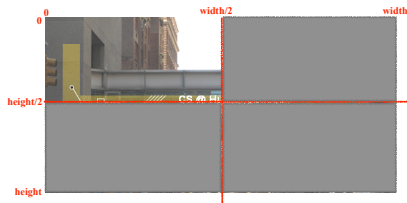
Challenge: Cropping Images

```
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()
```



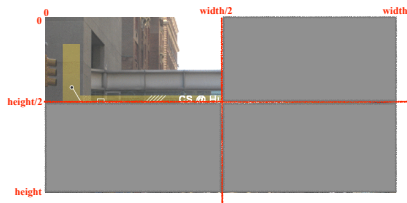
Challenge: Cropping Images

```
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

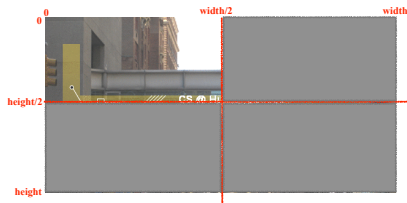
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



- 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()
```

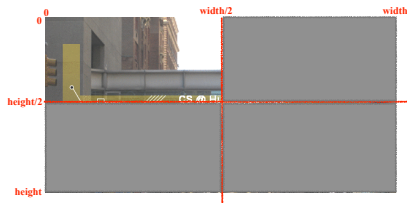


- How would you select the lower left corner?

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

Challenge: Cropping Images

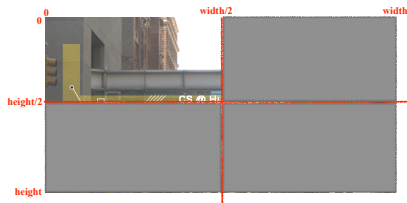
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



- 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

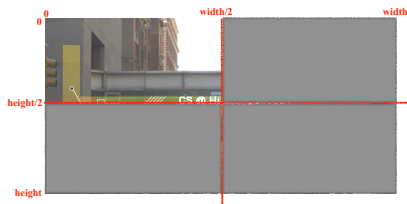
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



- 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

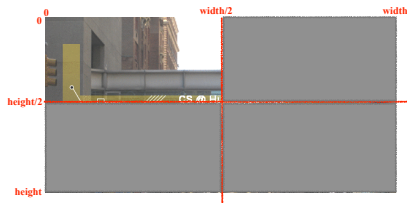
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



- 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()
```



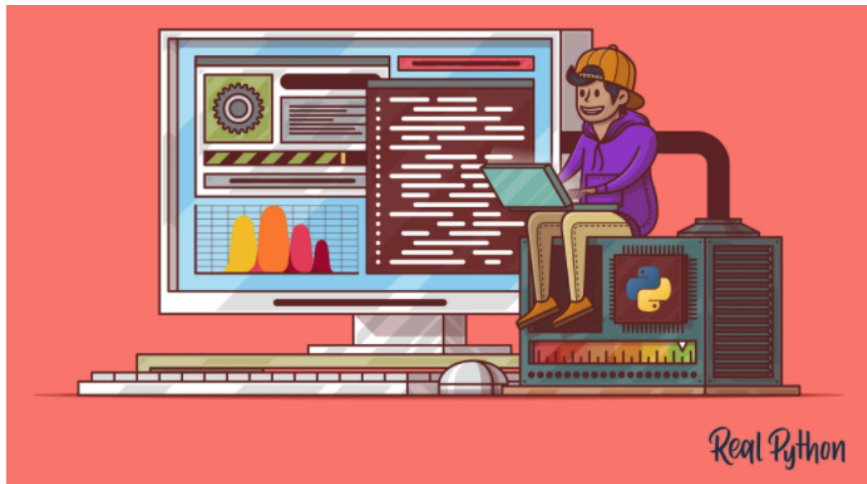
- How would you select the lower left corner?
`img2 = img[height//2:, :width//2]`
- How would you select the upper right corner?
`img2 = img[:height//2, width//2:]`
- How would you select the lower right corner?
`img2 = img[height//2:, width//2:]`

Today's Topics

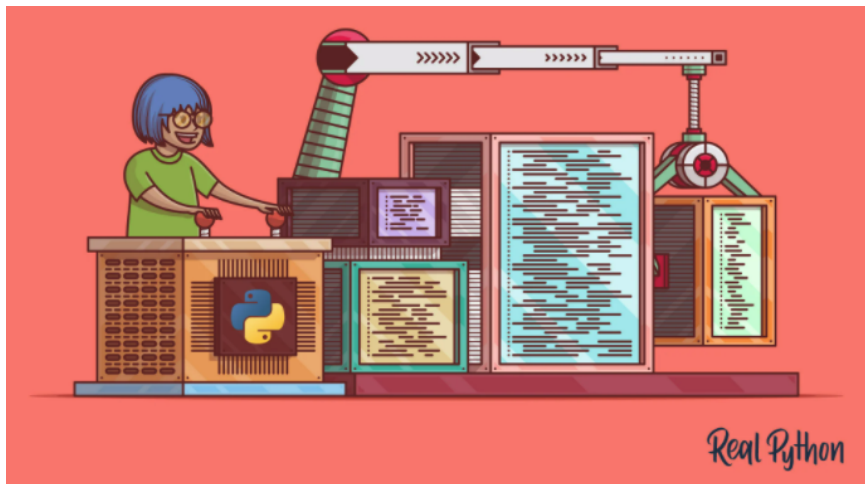


- Recap: Slicing & Images
- **Introduction to Functions**
- NYC Open Data

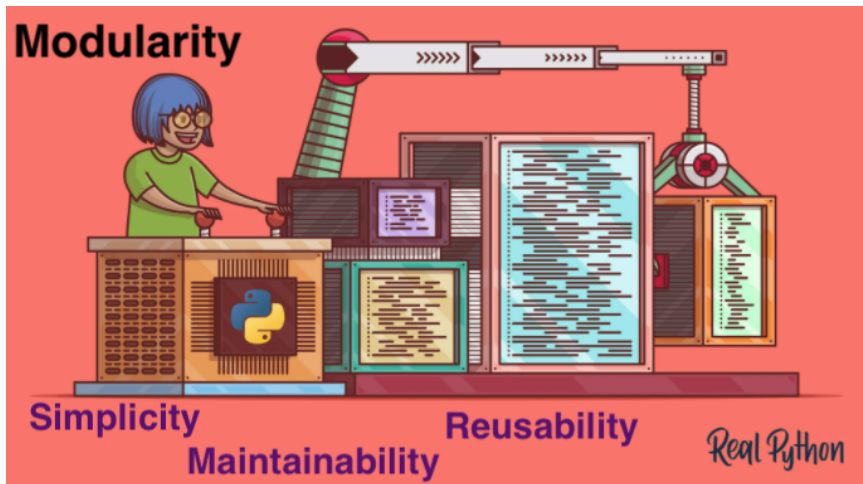
Scripts



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()
```

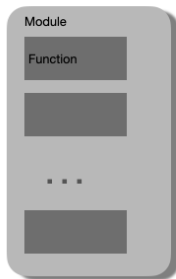
Python Tutor

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

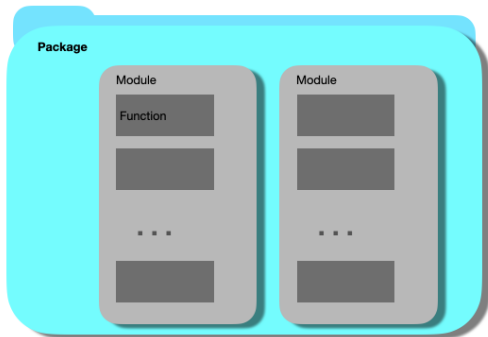
```
def main():  
    print("Hello, World!")  
  
if __name__ == "__main__":  
    main()
```

(Demo with pythonTutor)

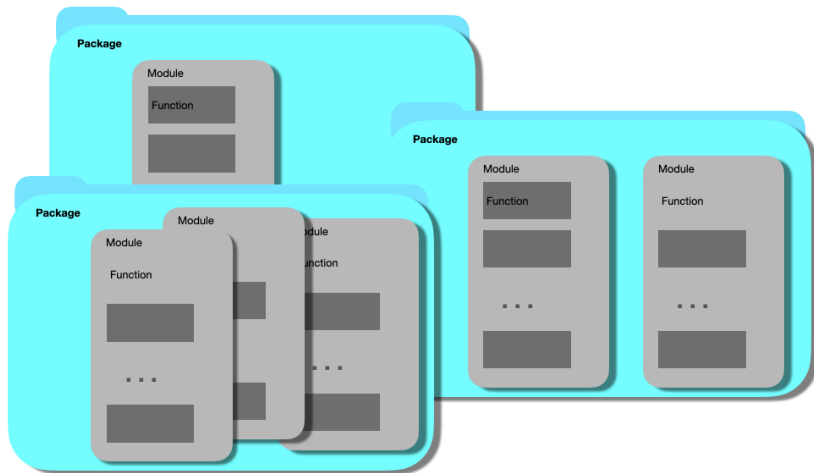
functions - modules - packages



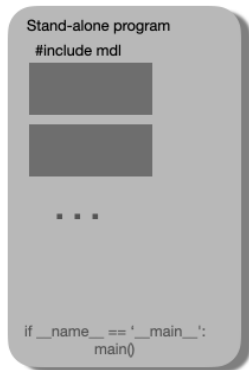
functions - modules - packages



functions - modules - packages



Stand-alone program



Challenge:

Predict what the code will do:

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

Python Tutor

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

(Demo with pythonTutor)

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)
```

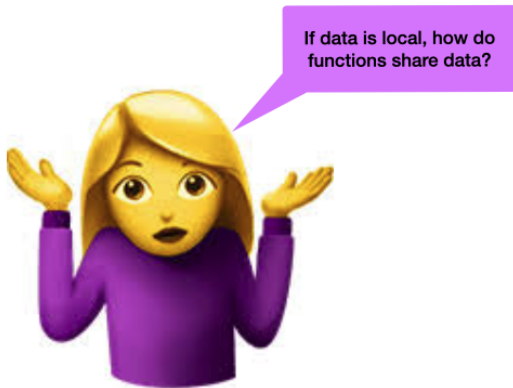
- You can have multiple functions.
- Each function defines the **scope** of its local variables

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?



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

- 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**.

```
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

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 the arguments in function calls to the parameters in function definitions. Red arrows, labeled "Formal Parameters", point from the parameter names in function definitions to the function call. Specifically, purple arrows point from `verse` in `mystery(verse)` to `v` in `mystery(v)`, and from `verse` and `c` in `enigma(verse, c)` to `v` and `c` in `enigma(v, c)`. Red arrows point from `v` in `mystery(v)` and `v, c` in `enigma(v, c)` to the `mystery` and `enigma` calls within `prob4()`.

Challenge:

Predict what the code will do:

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

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:

```
#Greet loop example

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

greetLoop("Thomas")
```

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

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

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

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

Python Tutor

```
#Greet loop example

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

greetLoop("Thomas")
```

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

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

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

```
sing('Fred')
sing('Thomas')
sing('Hunter')
```

(Demo with pythonTutor)

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()
```

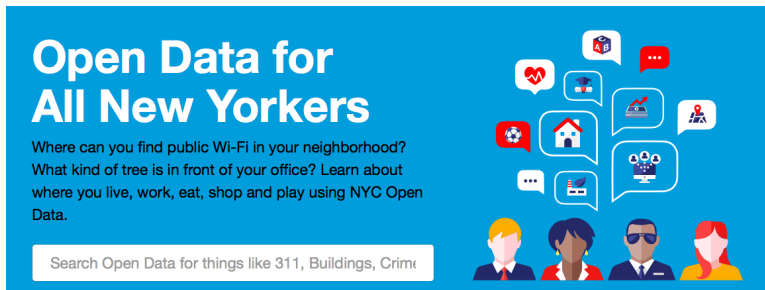
- Functions are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked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

Today's Topics



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

Accessing Structured Data: NYC Open Data

A blue banner for NYC Open Data. On the left, the text "Open Data for All New Yorkers" is in large white font. Below it, in smaller white text, are the questions: "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 is a white search bar with the placeholder text "Search Open Data for things like 311, Buildings, Crime". On the right side, there are several white speech bubbles containing various icons: a heart with a pulse line, a graduation cap, a red location pin, a soccer ball, a house, a factory with smoke, a person with a magnifying glass, a bar chart, and a group of people. Below the speech bubbles are four stylized human avatars with different skin tones and hairstyles, wearing professional attire.

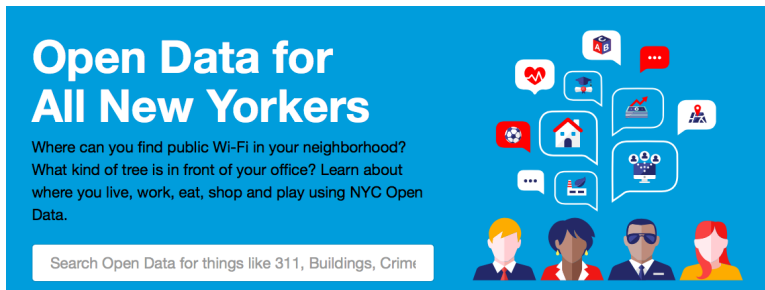
Open Data for All New Yorkers

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

Search Open Data for things like 311, Buildings, Crime

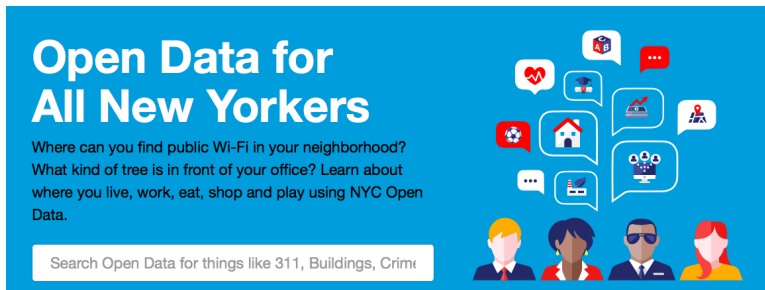
- Freely available source of data.

Accessing Structured Data: NYC Open Data

A blue banner for NYC Open Data. On the left, the text "Open Data for All New Yorkers" is in large white font. Below it, in smaller white text, are the questions: "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 is a white search bar with the placeholder text "Search Open Data for things like 311, Buildings, Crime". On the right side, there are several white speech bubbles containing various icons: a heart with a pulse line, a graduation cap, a red speech bubble with three dots, a soccer ball, a house, a laptop with a bar chart, a location pin, a person, a bar chart, and a person with a magnifying glass. Below the speech bubbles are four stylized human figures with different skin tones and hair colors (yellow, dark blue, dark blue with sunglasses, and red).

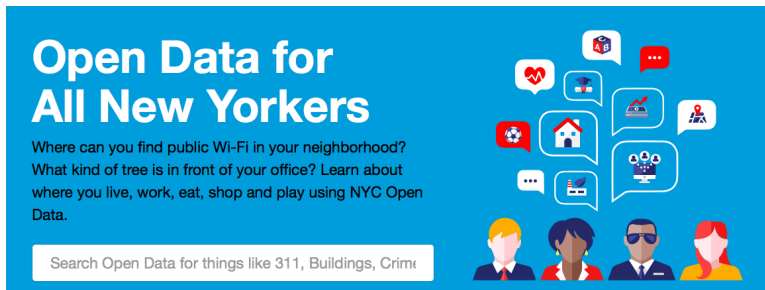
- Freely available source of data.
- Maintained by the NYC data analytics team.

Accessing Structured Data: NYC Open Data

A blue banner for NYC Open Data. On the left, the text "Open Data for All New Yorkers" is in large white font. Below it, in smaller white text, are the questions: "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 is a white search bar with the placeholder text "Search Open Data for things like 311, Buildings, Crime". On the right side, there are several white speech bubbles containing various icons: a heart with a pulse line, a graduation cap, a red speech bubble with three dots, a soccer ball, a house, a factory with smoke, a location pin, a person with a magnifying glass, a person with a gear, and a person with a lightbulb. Below the speech bubbles are four stylized human figures with different hair colors and styles: blonde, dark skin with short hair, a man with sunglasses, and a woman with red hair.

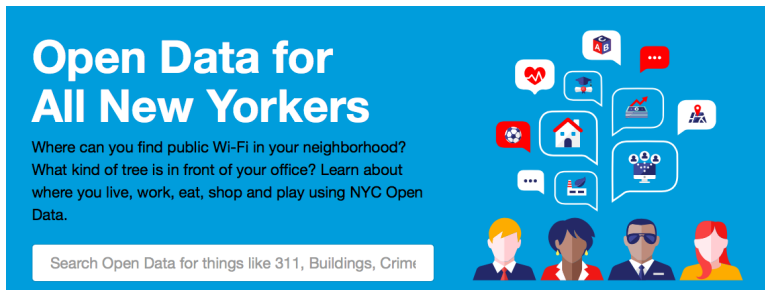
- Freely available source of data.
- Maintained by the NYC data analytics team.
- We will use several different ones for this class.

Accessing Structured Data: NYC Open Data

A blue banner for NYC Open Data. On the left, the text 'Open Data for All New Yorkers' is in large white font. Below it, in smaller white text, are the questions: '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 is a white search bar with the placeholder text 'Search Open Data for things like 311, Buildings, Crime'. On the right side, there are several white speech bubbles containing icons: a heart with a pulse line, a graduation cap, a red speech bubble with three dots, a soccer ball, a house, a factory with smoke, a location pin, a person with a magnifying glass, and a person with a gear. Below the speech bubbles are four stylized human avatars 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.

Accessing Structured Data: NYC Open Data

A blue banner for NYC Open Data. On the left, the text "Open Data for All New Yorkers" is in large white font. Below it, in smaller white text, are the questions: "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 is a white search bar with the placeholder text "Search Open Data for things like 311, Buildings, Crime". On the right side, there are several white speech bubbles containing icons: a heart with a pulse line, a graduation cap, a red speech bubble with three dots, a soccer ball, a house, a laptop with a bar chart, a location pin, a person, a bar chart, and a person with a magnifying glass. 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.
- Lab 7 covers accessing and downloading NYC OpenData datasets.

Example: OpenData Film Permits

Example: OpenData Film Permits

- What's the most popular street for filming?

Example: OpenData Film Permits

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

Example: OpenData Film Permits

- 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 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/stations/permits/when-permit-required-page>

More Views Filter Visualize Export Discuss Embed About

EventID	EventType	StartDateT...	EndDateT...	EnteredO...	EventAg...	ParkingIn...	Borne...	Com...	Felice...	Categ...	SubC...	Count...	ZipCo...
45003	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:35...	Mayor's Offi...	STARKE AVENUE S...	Queens	2	108	Television	Epicodic s...	United Sta...	11101
45497	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
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	75, 88	Still Photo...	Net Appari...	United Sta...	11217, 11...
45400	Shooting Permit	12/06/2018 12:00...	12/06/2018 11:59...	12/06/2018 03:28...	Mayor's Offi...	13 AVENUE boro...	Queens	1, 3, 7	108, 75, 88	Film	Feature	United Sta...	10802, 11...
454914	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/06/2018 03:05...	Mayor's Offi...	ELBERT STREET b...	Brooklyn	4, 6	104, 75, 88	Television	Epicodic s...	United Sta...	11209, 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...	36 STREET boro...	Queens	1	114	Television	Cable-epi...	United Sta...	11101, 11...

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

Example: OpenData Film Permits

NYC OpenData

Home Data About Learn Alerts Contact Us Blog Sign In

Film Permits

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

More Views: [View](#) [Visualize](#) [Export](#) [Discuss](#) [Embed](#) [About](#)

EventID	EventType	StartDateT...	EndDateTime	EnteredOn	EventAg...	ParkingInfo	Borne...	Com...	Felice...	Categ...	SubC...	Count...	ZipCo...
45003	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:35...	Mayor's Offi...	STARKE AVENUE S...	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 09:01...	Mayor's Offi...	EAGLE STREET B...	Brooklyn	1	84	Television	Episodic s...	United Sta...	11222
45491	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/06/2018 05:44...	Mayor's Offi...	SOUTH OXFORD ...	Brooklyn	2, 6	75, 88	Still Photo...	Not Applica...	United Sta...	11217, 11...
45400	Shooting Permit	12/06/2018 12:00...	12/06/2018 11:59...	12/06/2018 03:28...	Mayor's Offi...	13 AVENUE Sene...	Queens	1, 3, 7	108, 7, 98	Film	Feature	United Sta...	10802, 11...
45414	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/06/2018 03:05...	Mayor's Offi...	ELBERT STREET S...	Brooklyn	4, 6	104, 75, 83	Television	Episodic s...	United Sta...	11209, 11...
45489	Shooting Permit	12/05/2018 08:00...	12/05/2018 09:00...	12/06/2018 02:45...	Mayor's Offi...	ELBERT STREET S...	Brooklyn	4	83	Television	Episodic s...	United Sta...	11227
45485	Shooting Permit	12/06/2018 07:00...	12/06/2018 10:00...	12/06/2018 02:17...	Mayor's Offi...	36 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 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/staterenewpermits/when-permit-required.page>

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

EventID	EventType	StartDateT...	EndDateTime	EnteredOn...	EventAg...	ParkingInfo	Borne...	Com...	Felice...	Categ...	SubC...	Count...	ZipCo...
45003	Shooting Permit	12/06/2018 07:00...	12/06/2018 19:00...	12/05/2018 12:35...	Mayor's Offi...	STARBUCKS AVENUE S...	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 09:01...	Mayor's Offi...	EAGLE STREET B...	Brooklyn	1	84	Television	Episodic s...	United Sta...	11222
45491	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/06/2018 05:44...	Mayor's Offi...	SOUTH OXFORD...	Brooklyn	2, 6	75, 88	Still Photo...	Not Applica...	United Sta...	11217, 11...
45400	Shooting Permit	12/06/2018 12:00...	12/06/2018 11:00...	12/06/2018 03:28...	Mayor's Offi...	13 AVENUE SENE...	Queens	1, 3, 7	108, 7, 98	Film	Feature	United Sta...	10802, 11...
454914	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/06/2018 03:05...	Mayor's Offi...	ELBERT STREET S...	Brooklyn	4, 6	104, 75, 83	Television	Episodic s...	United Sta...	11202, 11...
454089	Shooting Permit	12/05/2018 08:00...	12/05/2018 09:00...	12/06/2018 02:45...	Mayor's Offi...	ELBERT STREET S...	Brooklyn	4	83	Television	Episodic s...	United Sta...	11227
45485	Shooting Permit	12/06/2018 07:00...	12/06/2018 10:00...	12/06/2018 02:17...	Mayor's Offi...	36 STREET BROW...	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

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/staterenewpermits/when-permit-required.page>

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

EventID	EventType	StartDateT...	EndDateTime	EnteredOn	EventAg...	ParkingHeld	Borne...	Com...	Felice...	Categ...	SubC...	Count...	ZipCo...
45003	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:35...	Mayor's Offi...	STARKE AVENUE S...	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 09:01...	Mayor's Offi...	EAGLE STREET B...	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 05:44...	Mayor's Offi...	SOUTH OXFORD ...	Brooklyn	2, 6	75, 88	Still Photo...	Not Applic...	United Sta...	11217, 11...
45420	Shooting Permit	12/06/2018 12:00...	12/06/2018 11:59...	12/06/2018 03:28...	Mayor's Offi...	13 AVENUE Sene...	Queens	1, 3, 7	108, 7, 98	Film	Feature	United Sta...	10802, 11...
45414	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/06/2018 03:05...	Mayor's Offi...	ELBERT STREET S...	Brooklyn	4, 6	104, 75, 83	Television	Epicodic s...	United Sta...	11202, 11...
45489	Shooting Permit	12/05/2018 08:00...	12/05/2018 09:00...	12/06/2018 02:45...	Mayor's Offi...	ELBERT STREET S...	Brooklyn	4	83	Television	Epicodic 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...	36 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:

```
#Sci 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

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/stations/permits/when-permit-required.page>

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

EventID	EventType	StartDateT...	EndDateTime	EnteredOn...	EventStg...	ParkingHeld	Borne...	Com...	Felice...	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 S...	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 09:01...	Mayor's Offi...	EAGLE STREET b...	Brooklyn	1	84	Television	Epicodic s...	United Sta...	11222
454941	Shooting Permit	12/06/2018 07:00...	12/06/2018 07:00...	12/06/2018 05:44...	Mayor's Offi...	SOUTH OXFORD ...	Brooklyn	2, 6	76, 88	Still Photo...	Net Appari...	United Sta...	11217, 11...
454920	Shooting Permit	12/06/2018 11:00...	12/06/2018 11:00...	12/06/2018 03:28...	Mayor's Offi...	13 AVENUE Sene...	Queens	1, 3, 7	108, 7, 98	Film	Feature	United Sta...	10802, 11...
454914	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/06/2018 03:05...	Mayor's Offi...	ELBERT STREET S...	Brooklyn	4, 6	104, 76, 83	Television	Epicodic s...	United Sta...	11209, 11...
454909	Shooting Permit	12/05/2018 08:00...	12/05/2018 09:00...	12/06/2018 02:45...	Mayor's Offi...	ELBERT STREET S...	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...	36 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 street, or a park. See <http://www1.nyc.gov/staterenewpermits/when-permit-required.page>

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

EventID	EventType	StartDateT...	EndDateTime	EnteredOn...	EventAg...	ParkingHeld	Borne...	Com...	Felice...	Categ...	SubC...	Count...	ZipCo...
45003	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:35...	Mayor's Offi...	STARKE AVENUE S...	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 09:01...	Mayor's Offi...	EAGLE STREET B...	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 05:44...	Mayor's Offi...	SOUTH OXFORD ...	Brooklyn	2, 6	75, 88	Still Photo...	Net Appari...	United Sta...	11217, 11...
45420	Shooting Permit	12/06/2018 11:00...	12/06/2018 11:00...	12/06/2018 03:28...	Mayor's Offi...	13 AVENUE Sene...	Queens	1, 3, 7	108, 7, 98	Film	Feature	United Sta...	10002, 11...
45414	Shooting Permit	12/06/2018 08:00...	12/06/2018 11:00...	12/06/2018 03:05...	Mayor's Offi...	ELBERT STREET S...	Brooklyn	4, 6	104, 75, 83	Television	Epicodic S...	United Sta...	11202, 11...
45489	Shooting Permit	12/05/2018 08:00...	12/05/2018 09:00...	12/06/2018 02:45...	Mayor's Offi...	ELBERT STREET S...	Brooklyn	4	83	Television	Epicodic 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...	36 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

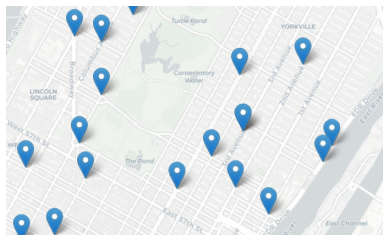
Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/mome/permits/when-permit-required.page>

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

Can approach the other questions in the same way:

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

Design Question



Design an algorithm that finds the collision that is closest to input location.

DATE	TIME	BOROUGH	ZIP CODE	LATITUDE	LONGITUDE	LOCATION	ON STREET	CROSS STREET	OFF STREET	NUMBER OF
12/31/16	9:56						2 AVENUE			0
12/31/16	9:55	BRONX	10462	40.83521	-73.85497	(40.8352098	UNIONPORT	OLMSTEAD AVENUE		0
12/31/16	9:50						JESUP AVENUE			0
12/31/16	9:40	BROOKLYN	11225	40.66911	-73.95335	(40.6691137	ROGERS AVE	UNION STREET		0
12/31/16	20:23	BROOKLYN	11209	40.62578	-74.02415	(40.6257805	80 STREET	5 AVENUE		0
12/31/16	20:20	QUEENS	11375	40.71958	-73.83977	(40.719584,	ASCAN AVEN	QUEENS BOULEVARD		0
12/31/16	20:15	BROOKLYN	11204				60 STREET	BAY PARKWAY		0
12/31/16	20:10			40.66479	-73.82047	(40.6647944,	-73.8204653)			0
12/31/16	20:10						69 STREET	37 AVENUE		0
12/31/16	20:05	BRONX	10457	40.85429	-73.90026	(40.8542925	RYER AVENUE	EAST 181 STREET		0

Design Question

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

Design Question

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

How to approach this:

- Create a “To Do” list of what your program has to accomplish.

Design Question

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

How to approach this:

- Create a “To Do” list of what your program has to accomplish.
- Read through the problem, and break it into “To Do” items.

Design Question

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

How to approach this:

- Create a “To Do” list of what your program has to accomplish.
- Read through the problem, and break it into “To Do” items.
- Don't worry if you don't know how to do all the items you write down.

Design Question

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

How to approach this:

- Create a “To Do” list of what your program has to accomplish.
- Read through the problem, and break it into “To Do” items.
- Don’t worry if you don’t know how to do all the items you write down.
- Example:

Design Question

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

How to approach this:

- Create a “To Do” list of what your program has to accomplish.
- Read through the problem, and break it into “To Do” items.
- Don't worry if you don't know how to do all the items you write down.
- Example:
 - ① Find data set (great place to look: NYC OpenData).

Design Question

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

How to approach this:

- Create a “To Do” list of what your program has to accomplish.
- Read through the problem, and break it into “To Do” items.
- Don’t worry if you don’t know how to do all the items you write down.
- Example:
 - ① Find data set (great place to look: NYC OpenData).
 - ② Ask user for current location.

Design Question

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

How to approach this:

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

Design Question

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

How to approach this:

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

Design Question

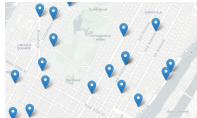
Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

How to approach this:

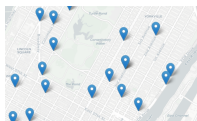
- Create a “To Do” list of what your program has to accomplish.
- Read through the problem, and break it into “To Do” items.
- Don't worry if you don't know how to do all the items you write down.
- Example:
 - ① Find data set (great place to look: NYC OpenData).
 - ② Ask user for current location.
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

Recap

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

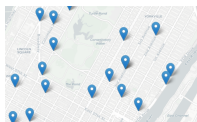


Recap



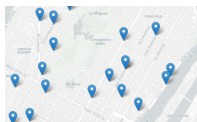
- **Functions** are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:

Recap



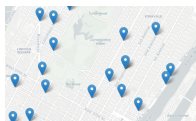
- **Functions** are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
Example: `print("Hello", "World")`

Recap



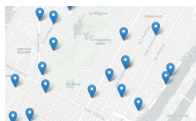
- **Functions** are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
Example: `print("Hello", "World")`
- Can write, or **define** your own functions,

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("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():
    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 Fall 17 V2, #4(b).

Weekly Reminders!



Before next lecture, don't forget to:

- Read and work through Lab 7!

Weekly Reminders!



Before next lecture, don't forget to:

- Read and work through Lab 7!
- Submit this week's programming assignments

Weekly Reminders!



Before next lecture, don't forget to:

- Read and work through Lab 7!
- Submit this week's programming assignments
- Tutoring and help is available through cscisummer23@gmail.com