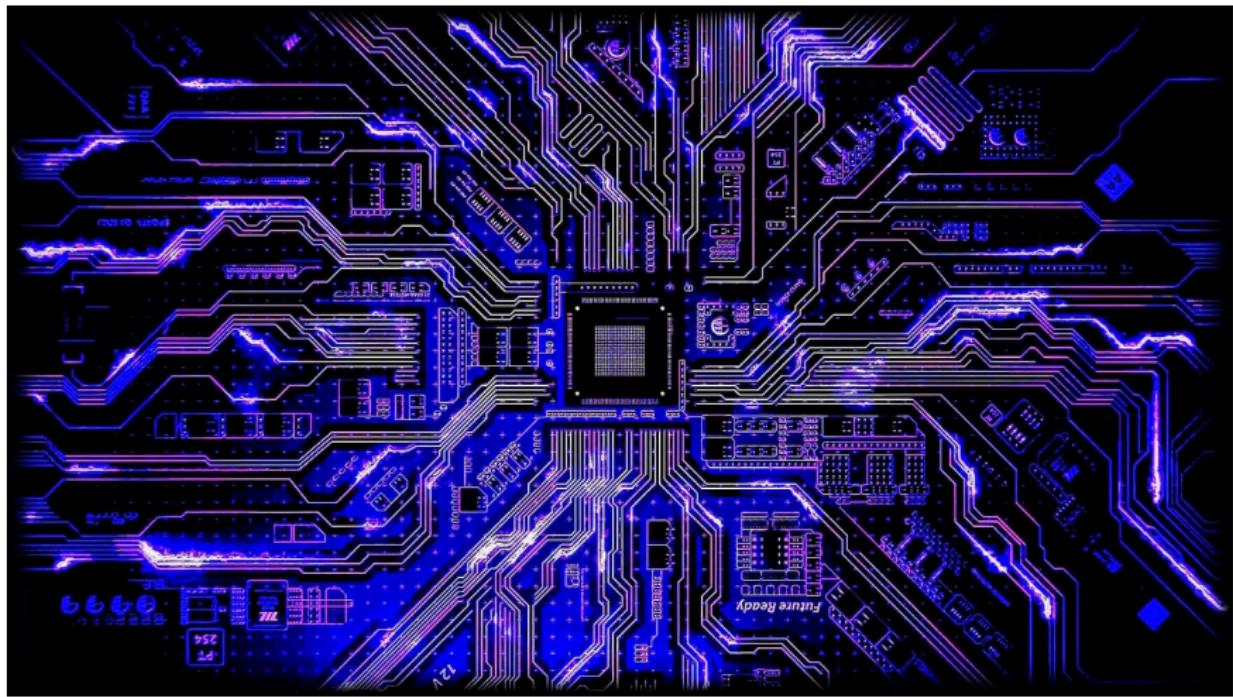


CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

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()
```

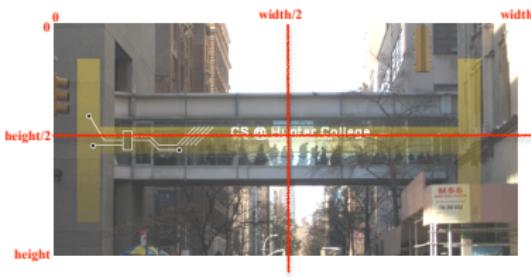
Challenge: Cropping Images

```
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()
```



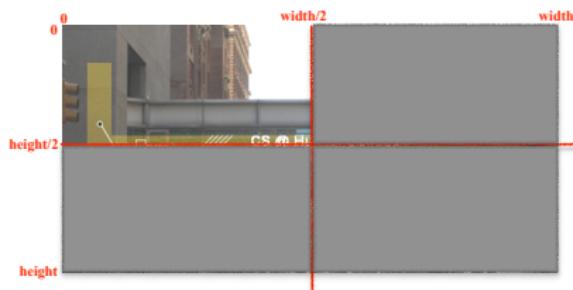
Challenge: Cropping Images

```
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()
```



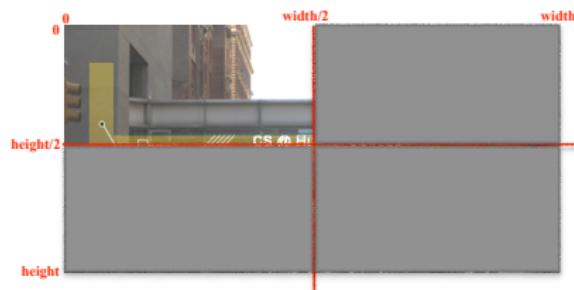
Challenge: Cropping Images

```
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

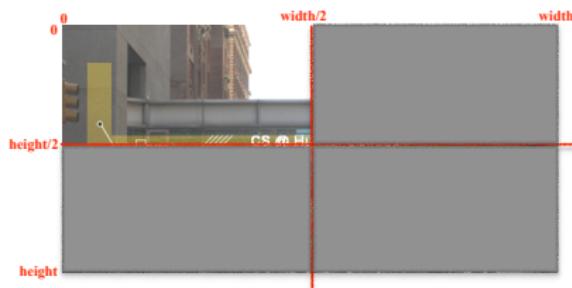
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



- 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()
```

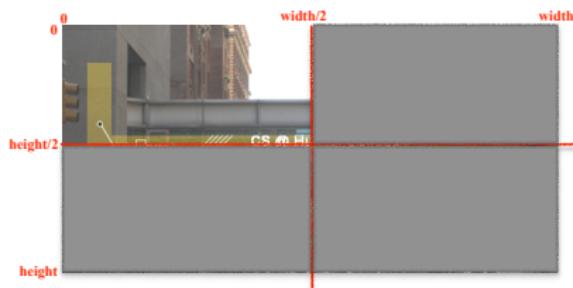


- How would you select the lower left corner?

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

Challenge: Cropping Images

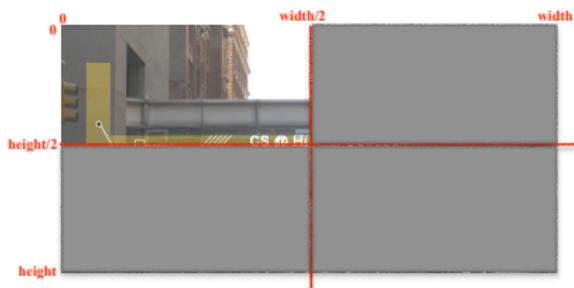
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



- 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?

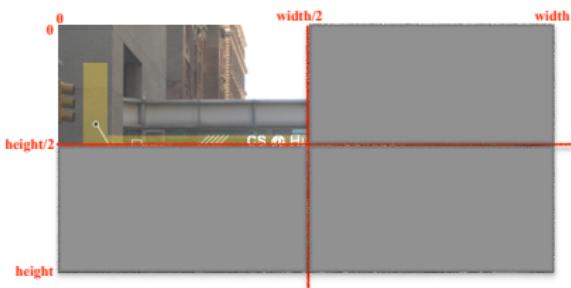
```
img2 = img[height//2:, :width//2]
```

- How would you select the upper right corner?

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

Challenge: Cropping Images

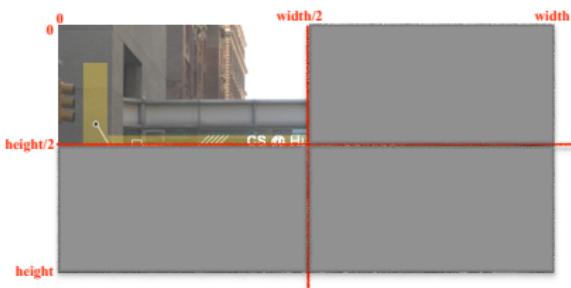
```
import matplotlib.pyplot as plt
import numpy as np
img = plt.imread('csBridge')
plt.imshow(img)
plt.show()
height = img.shape[0]
width = img.shape[1]
img2 = img[:height//2, :width//2]
plt.imshow(img2)
plt.show()
```



- 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()
```



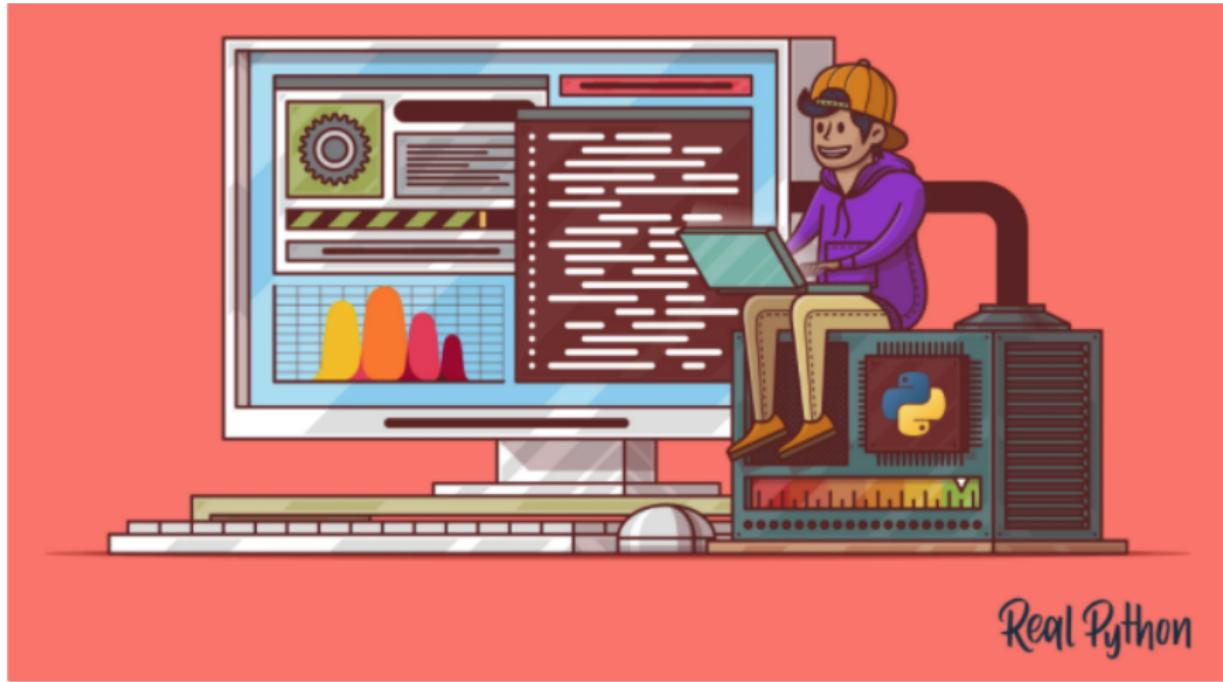
- How would you select the lower left corner?
`img2 = img[height//2:, :width//2]`
- How would you select the upper right corner?
`img2 = img[:height//2, width//2:]`
- How would you select the lower right corner?
`img2 = img[height//2:, width//2:]`

Today's Topics



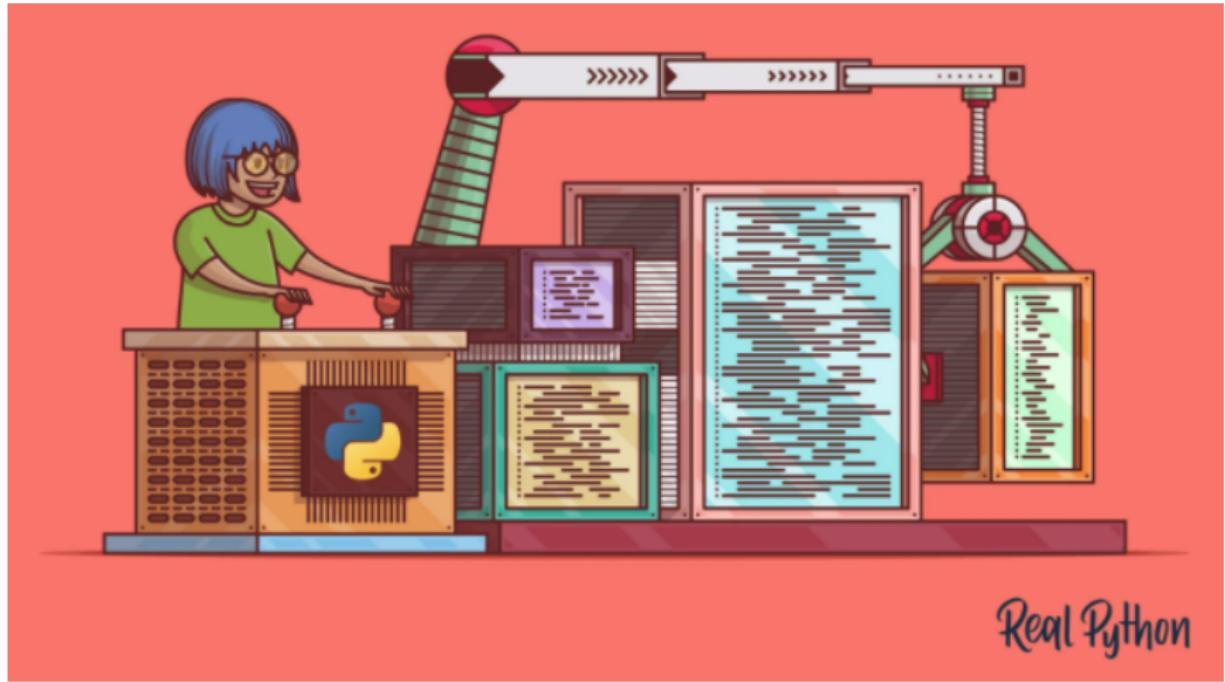
- Recap: Slicing & Images
- **Introduction to Functions**
- NYC Open Data

Scripts

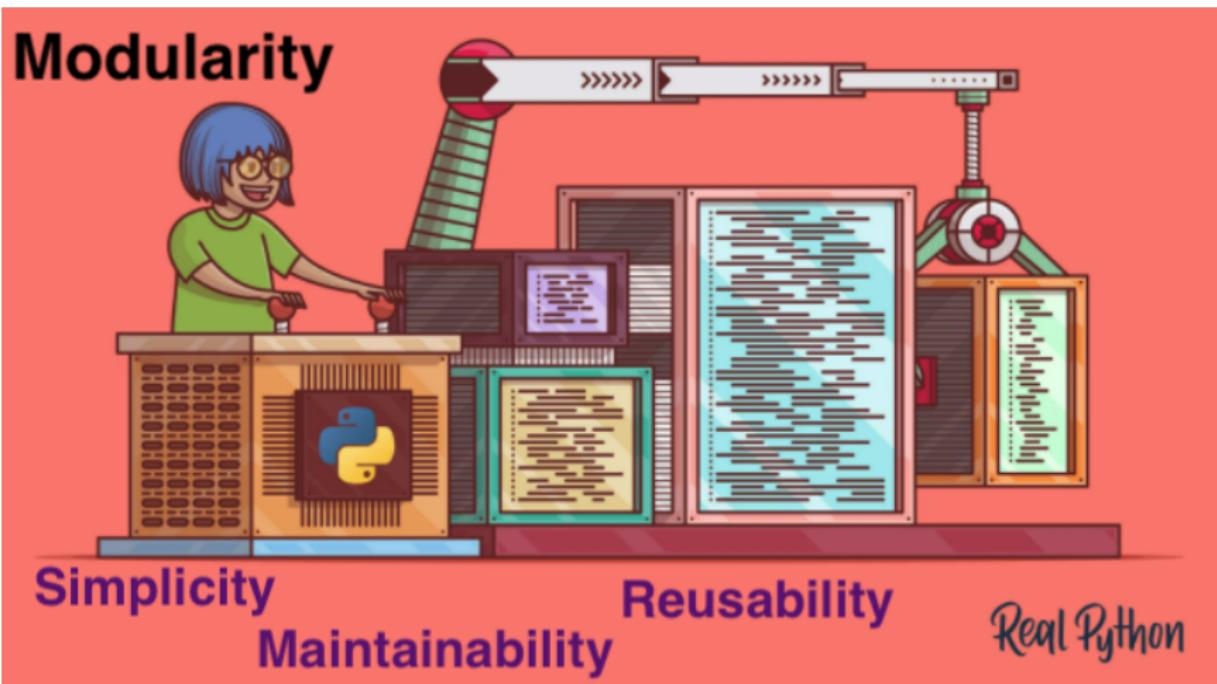


Real Python

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

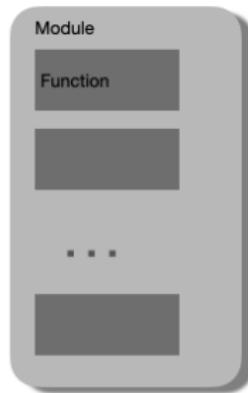
```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#     says hello to the world!

def main():
    print("Hello, World!")

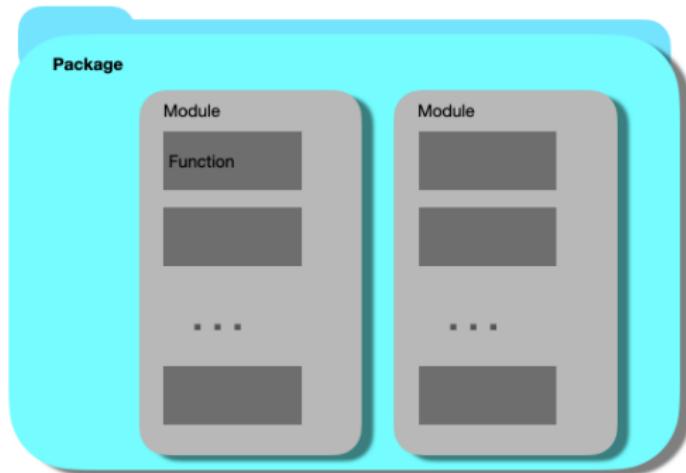
if __name__ == "__main__":
    main()
```

(Demo with pythonTutor)

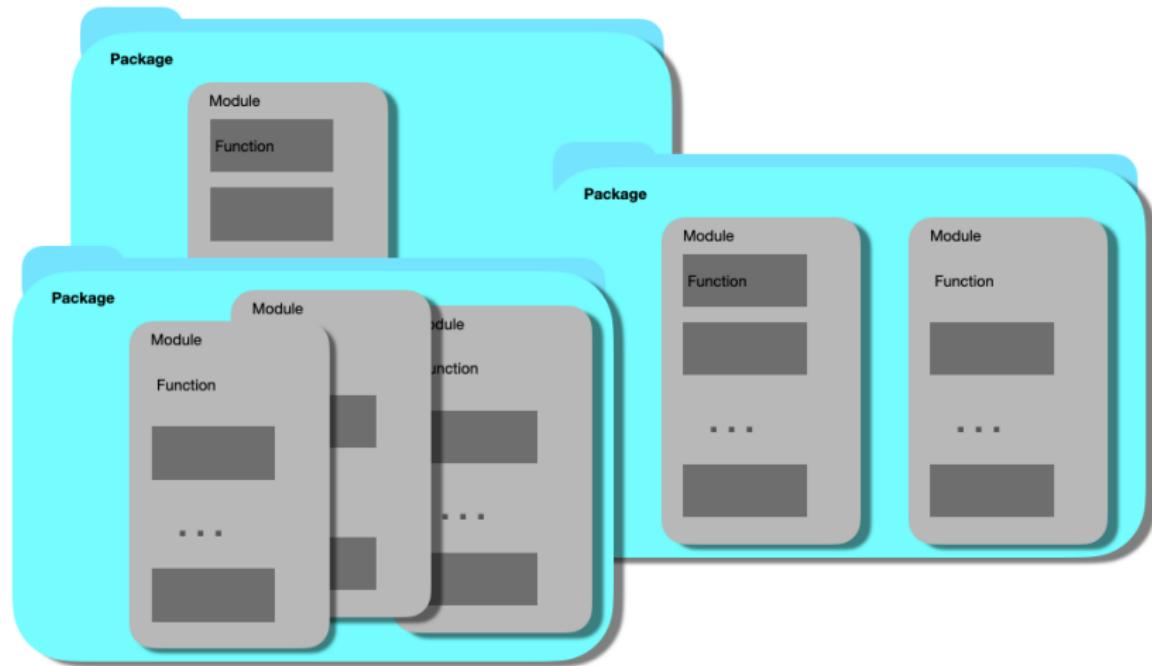
functions - modules - packages



functions - modules - packages



functions - modules - packages



Stand-alone program

Stand-alone program

```
#include mdl
```



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

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

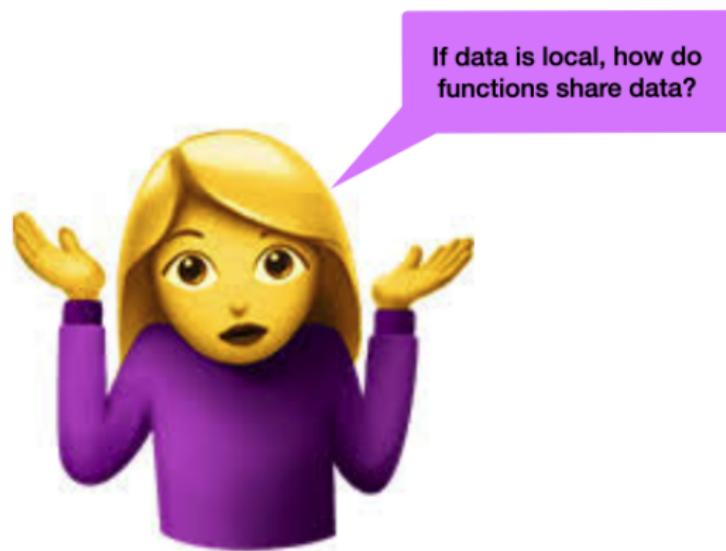
Scope

```
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

```
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)
                                         Actual Parameters

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)
                                         Actual Parameters

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

- Functions can have **input parameters**.
- Surrounded by parentheses, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**.
- Functions can also **return values** to where it was called.

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 binding of actual parameters to formal parameters in the `prob4()` function. Red arrows point from the underlined identifiers to their corresponding actual parameters: `prob4()` to `prob4()`, `v` to `verse`, `c` to `c`, and `enigma()` to `enigma()`. Purple ovals highlight the actual parameters `verse` and `c` in the `mystery()` call, and `verse` and `c` in the `enigma()` call. A purple arrow points from the label "Actual Parameters" to the ovals. A red arrow points from the label "Formal Parameters" to the underlined identifiers.

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: '))
    mString = monthString(n)
    print('The month is', mString)
```

(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

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 central cluster of twelve speech bubbles in various sizes and colors (white, red, blue) containing icons such as a house, a map, a graph, a graduation cap, a heart, a soccer ball, a person at a computer, and a location pin. Below this cluster are four stylized human figures: a man with blonde hair, a woman with dark skin and short hair, a man with dark skin and 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, Crime



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

Accessing Structured Data: NYC Open Data

Open Data for All New Yorkers

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

Search Open Data for things like 311, Buildings, Crime



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

Accessing Structured Data: NYC Open Data

Open Data for All New Yorkers

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

Search Open Data for things like 311, Buildings, Crime



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

Accessing Structured Data: NYC Open Data

Open Data for All New Yorkers

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

Search Open Data for things like 311, Buildings, Crime



- 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

Film Permits

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

EventID	EventType	StartDateL	EndDateW	EventD	EventM	ParkingHd	BoroL	Com.	PermitL	Categ.	SubC.	Count.	ZipCo.
455083	Shooting Permit	12/09/2018 0000..	12/09/2018 0500..	12/09/2018 1230..	May 09/18	STAR AVENUE b..	Queens	2	108	Television	Episodic L..	United Sta..	11101
454677	Shooting Permit	12/09/2018 0000..	12/09/2018 0500..	12/09/2018 0011..	May 09/18	EAGLE STREET b..	Brooklyn	1	84	Television	Episodic L..	United Sta..	11222
454541	Shooting Permit	12/09/2018 0000..	12/09/2018 0700..	12/09/2018 0744..	May 09/18	SOUTH OXFORD ..	Brooklyn	2, 8	70, 88	Still Photo ..	Not Applic..	United Sta..	11217, 11..
454900	Shooting Permit	12/09/2018 1000..	12/09/2018 1159..	12/09/2018 0232..	May 09/18	12 AVENUE betw..	Queens	1, 3, 7	108, 7, 08	Film	Feature	United Sta..	1002, 11..
454914	Shooting Permit	12/09/2018 0000..	12/09/2018 1100..	12/09/2018 0308..	May 09/18	ELDRY STREET b..	Brooklyn	4, 5	104, 76, 89	Television	Episodic S..	United Sta..	11207, 11..
454889	Shooting Permit	12/09/2018 0800..	12/09/2018 0900..	12/09/2018 0245..	May 09/18	ELDER STREET b..	Brooklyn	4	83	Television	Episodic L..	United Sta..	11227
454865	Shooting Permit	12/09/2018 0700..	12/09/2018 1030..	12/09/2018 0211..	May 09/18	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.

Example: OpenData Film Permits

The screenshot shows the NYC OpenData website with a search bar and navigation menu. Below the menu is a table titled "Film Permits" with a note about permits required for city property like sidewalks or streets. The table has columns for EventID, EventType, StartDate, EndDate, ExtendedBy, EventMgt, ParkingHeld, Block, Comm., Police, Gang, SubC., County, and ZipCode. The data includes rows for various shooting permits issued between 2018 and 2019 across different boroughs and zip codes.

EventID	EventType	StartDate	EndDate	ExtendedBy	EventMgt	ParkingHeld	Block	Comm.	Police	Gang	SubC.	County	ZipCode
455083	Shooting Permit	12/09/2018 0000..	12/09/2018 0500..	12/09/2018 1230..	Mayer's OFC..	STAIR AVENUE b..	Queens	2	108	Television	Episodic s..	United States	11101
454677	Shooting Permit	12/06/2018 0000..	12/06/2018 0500..	12/06/2018 0011..	Mayer's OFC..	EAGLE STREET b..	Brooklyn	1	84	Television	Episodic s..	United States	11222
454541	Shooting Permit	12/09/2018 0700..	12/09/2018 0750..	12/09/2018 0941..	Mayer's OFC..	SOUTH OXFORD ..	Brooklyn	2, 8	70, 88	Still Photo..	Not Applicable	United States	11217, 11218
454900	Shooting Permit	12/05/2018 1000..	12/05/2018 1159..	12/05/2018 0232..	Mayer's OFC..	13 AVENUE betw..	Queens	1, 3, 7	108, 7, 06	Film	Feature	United States	10002, 11101
454914	Shooting Permit	12/09/2018 0000..	12/09/2018 1100..	12/09/2018 0208..	Mayer's OFC..	ELDERT STREET b..	Brooklyn	4, 5	104, 76, 89	Television	Episodic s..	United States	11207, 11222
454809	Shooting Permit	12/05/2018 0800..	12/05/2018 0900..	12/04/2018 0245..	Mayer's OFC..	ELDERT STREET b..	Brooklyn	4	83	Television	Episodic s..	United States	11227
454805	Shooting Permit	12/06/2018 0700..	12/06/2018 1030..	12/06/2018 0211..	Mayer's OFC..	36 STREET betwe..	Queens	1	114	Television	Cable-sports..	United States	11101, 11222

- 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

The screenshot shows the NYC OpenData homepage with the title "Film Permits". Below it is a detailed table of permit data. The table has 16 columns: EventID, EventType, StartDate, EndDate, Extended, EventMg., ParkingHeld, Block, Com., Police, Gang, SubC., Count., and ZipCode. The data includes various permit types like Shooting Permit, dates ranging from 2018-05-01 to 2018-06-06, and locations across Brooklyn and Queens. A navigation bar at the top includes Home, Data, About, Learn, Alerts, Contact Us, Blog, a search bar, and a sign-in button.

EventID	EventType	StartDate	EndDate	Extended	EventMg.	ParkingHeld	Block	Com.	Police	Gang	SubC.	Count.	ZipCode
455083	Shooting Permit	12/05/2018 0000..	12/06/2018 0500..	12/05/2018 1230..	Mayer ORPC	STAIR AVENUE b..	Queens	2	108	Television	Episodic s..	United Sta..	11101
454677	Shooting Permit	12/05/2018 0000..	12/06/2018 0500..	12/05/2018 0011..	Mayer ORPC	EAGLE STREET b..	Brooklyn	1	84	Television	Episodic s..	United Sta..	11222
454541	Shooting Permit	12/05/2018 0000..	12/06/2018 0730..	12/04/2018 0041..	Mayer ORPC	SOUTH OXFORD ..	Brooklyn	2, 8	70, 88	Still Photo ..	Not Applic..	United Sta..	11217, 11..
454900	Shooting Permit	12/05/2018 0000..	12/06/2018 1159..	12/05/2018 0232..	Mayer ORPC	12 AVENUE betw..	Queens	1, 3, 7	108, 7, 06	Film	Feature	United Sta..	10002, 11..
454914	Shooting Permit	12/05/2018 0000..	12/06/2018 1100..	12/05/2018 0038..	Mayer ORPC	ELDERT STREET b..	Brooklyn	4, 5	104, 76, 89	Television	Episodic s..	United Sta..	11207, 11..
454809	Shooting Permit	12/05/2018 0000..	12/05/2018 0000..	12/04/2018 0245..	Mayer ORPC	ELDERT STREET b..	Brooklyn	4	83	Television	Episodic s..	United Sta..	11227
454805	Shooting Permit	12/05/2018 0000..	12/06/2018 1030..	12/05/2018 0211..	Mayer ORPC	36 STREET betwe..	Queens	1	114	Television	Cable epis..	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

EventID	EventType	StartDate	EndDate	ExtendedBy	EventMg.	ParkingHeld	Block	Com.	Police	Gang	SubC.	Count.	ZipCode
455083	Shooting Permit	12/06/2018 0800..	12/06/2018 0900..	12/06/2018 1230..	Mayer's OFC..	STAIR AVENUE b..	Queens	2	108	Television	Episodic s..	United Sta..	11101
454677	Shooting Permit	12/06/2018 0800..	12/06/2018 0800..	12/06/2018 0810..	Mayer's OFC..	EAGLE STREET b..	Brooklyn	1	84	Television	Episodic s..	United Sta..	11222
454841	Shooting Permit	12/06/2018 0700..	12/06/2018 0700..	12/06/2018 0744..	Mayer's OFC..	SOUTH OXFORD ..	Brooklyn	2, 8	70, 88	Star Photo..	Not Applic..	United Sta..	11217, 11..
454900	Shooting Permit	12/06/2018 1000..	12/06/2018 1159..	12/06/2018 0230..	Mayer's OFC..	12 AVENUE betw..	Queens	1, 3, 7	108, 7, 06	Film	Feature	United Sta..	10002, 11..
454914	Shooting Permit	12/06/2018 0800..	12/06/2018 0900..	12/06/2018 0830..	Mayer's OFC..	ELDERT STREET b..	Brooklyn	4, 5	104, 76, 89	Television	Episodic s..	United Sta..	11207, 11..
454889	Shooting Permit	12/05/2018 0800..	12/05/2018 0900..	12/04/2018 0245..	Mayer's OFC..	ELDERT STREET b..	Brooklyn	4	83	Television	Episodic s..	United Sta..	11227
454865	Shooting Permit	12/06/2018 0700..	12/06/2018 1000..	12/06/2018 0217..	Mayer's OFC..	36 STREET betwe..	Queens	1	114	Television	Cable-sport..	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

Film Permits

Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See http://www.nyc.gov/html/nyparks/when_permit_required.page

EventID	EventType	StartDateL	EndDateW	ExtendedD	EventMg.	ParkingHeld	BoroL	Com.	PermitL	Categ.	SubC.	Count.	ZipCo.
455083	Shooting Permit	12/05/2018 0000L	12/06/2018 0500L	12/05/2018 1230L	Mayer ORPC	STAIR AVENUE L.	Queens	2	108	Television	Episodic L.	United States	11101
454677	Shooting Permit	12/06/2018 0000L	12/06/2018 0500L	12/06/2018 0011L	Mayer ORPC	EAGLE STREET Bk.	Brooklyn	1	84	Television	Episodic L.	United States	11222
454541	Shooting Permit	12/06/2018 0000L	12/06/2018 0730L	12/06/2018 0041L	Mayer ORPC	SOUTH OXFORD L.	Brooklyn	2, 8	70, 88	Still Photo	Not Applicable	United States	11271, 11...
454900	Shooting Permit	12/06/2018 1000L	12/06/2018 1159L	12/06/2018 0232L	Mayer ORPC	12 AVENUE betw.	Queens	1, 3, 7	108, 7, 06	Film	Feature	United States	10002, 11...
454914	Shooting Permit	12/06/2018 0000L	12/06/2018 0500L	12/06/2018 0038L	Mayer ORPC	ELDERT STREET L.	Brooklyn	4, 5	104, 76, 89	Television	Episodic L.	United States	11207, 11...
454889	Shooting Permit	12/05/2018 0000L	12/05/2018 0500L	12/04/2018 0245L	Mayer ORPC	ELDERT STREET L.	Brooklyn	4	83	Television	Episodic L.	United States	11227
454865	Shooting Permit	12/06/2018 0000L	12/06/2018 1030L	12/06/2018 0217L	Mayer ORPC	36 STREET betwe..	Queens	1	114	Television	Cable-epic.	United States	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

Film Permits

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

EventID	EventType	StartDateL	EndDateW	ExtendedD	EventMg.	ParkingHeld	BoroL	Com.	PermitL	Categ.	SubC.	Count.	ZipCo.
455083	Shooting Permit	12/06/2018 0800L		12/06/2018 0900L	12/06/2018 1230L	Mayer ORPC	STAIR AVENUE L	Queens	2	108	Television	Episodic L	United States 11101
454677	Shooting Permit	12/06/2018 0800L		12/06/2018 0800L	12/06/2018 0810L	Mayer ORPC	EAGLE STREET Bk	Brooklyn	1	84	Television	Episodic L	United States 11222
454841	Shooting Permit	12/06/2018 0700L		12/06/2018 0710L	12/06/2018 0941L	Mayer ORPC	SOUTH OXFORD	Brooklyn	2, 8	70, 88	Still Photo	Not Applicable	United States 11271, 11...
454900	Shooting Permit	12/06/2018 0800L		12/06/2018 1159L	12/06/2018 0202L	Mayer ORPC	12 AVENUE betw.	Queens	1, 3, 7	108, 7, 08	Film	Feature	United States 10002, 11...
454914	Shooting Permit	12/06/2018 0800L		12/06/2018 0900L	12/06/2018 0938L	Mayer ORPC	ELDER STREET Bk	Brooklyn	4, 5	104, 76, 89	Television	Episodic L	United States 11207, 11...
454889	Shooting Permit	12/05/2018 0800L		12/05/2018 0900L	12/04/2018 0245L	Mayer ORPC	ELDER STREET Bk	Brooklyn	4	83	Television	Episodic L	United States 11227
454865	Shooting Permit	12/06/2018 0700L		12/06/2018 1030L	12/06/2018 0217L	Mayer ORPC	36 STREET betwe..	Queens	1	114	Television	Cable epis.	United States 11101, 11...

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

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

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

Example: OpenData Film Permits

NYC OpenData

Home Data About Learn Alerts Contact Us Blog Sign In



Film Permits

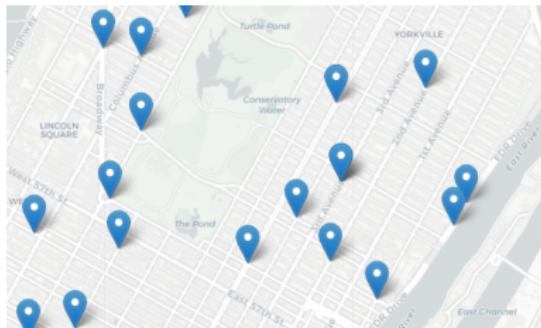
Permits are generally required when asserting the exclusive use of city property, like a sidewalk, a street, or a park. See <http://www1.nyc.gov/site/mome/permits/when-permit-required.page>

Film Permits													Find in this Dataset
EventID	EventType	StartDateTime	EndDateTime	EnteredOn	EventAgency	ParkingHeld	Borough	CommunityBoard	PolicePrecinct	Category	SubCategory	Count	ZipCode
455063	Shooting Permit	12/06/2018 07:00...	12/06/2018 09:00...	12/05/2018 12:36...	Mayor's Office...	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 Office...	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 Office...	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 Office...	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 Office...	ELDER 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 Office...	ELDER 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 Office...	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, -73.85497]	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, -73.95335]	ROGERS AVE	UNION STREET		0
12/31/16	20:23	BROOKLYN	11209	40.62578	-74.02415	[40.6257805, -74.02415]	80 STREET	5 AVENUE		0
12/31/16	20:20	QUEENS	11375	40.71958	-73.83977	[40.719584, -73.83977]	ASCAN AVENUE	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, -73.90026]	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

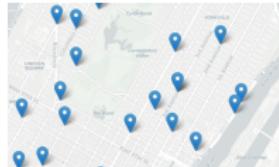
Design an algorithm that uses NYC OpenData collision data and computes the closest collision to the location the user provides.

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



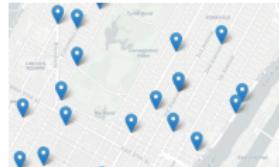
- **Functions** are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:

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



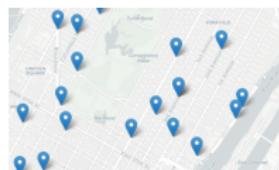
- **Functions** are a way to break code into pieces, that can be easily reused.
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:
Example: `print("Hello", "World")`
- Can write, or **define** your own functions,

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 = "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()
```

- 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: '))
dTTotal = totalWithTax(dinner, dtip)
print('Dinner total is', dTotal)
```

```
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.capitalize()
    return(c)

def enigma(v,w):
    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 = "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()
```

- 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).

(b) What is the output:

```
#Mystery program

def select(nums):
    m = nums[0]
    for n in nums:
        if n < m:
            m = n
    print(m)
    return(m)

def truncate(userList):
    if len(userList) < 5:
        best = select(userList)
    else:
        best = select(userList[2:])
    print("Best is", best)
```

i. For `truncate([10,2])`?

Output:

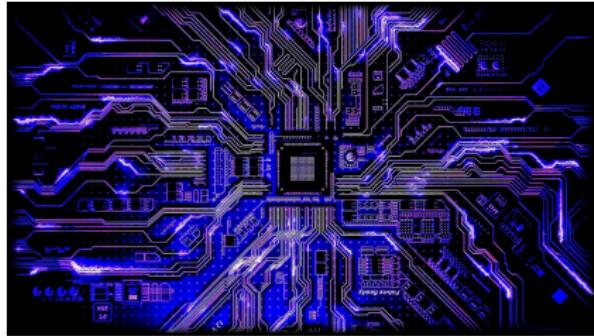
ii. For `truncate([7,2,0,1])`?

Output:

iii. For `truncate([0,2,10,9,1,-1])`?

Output:

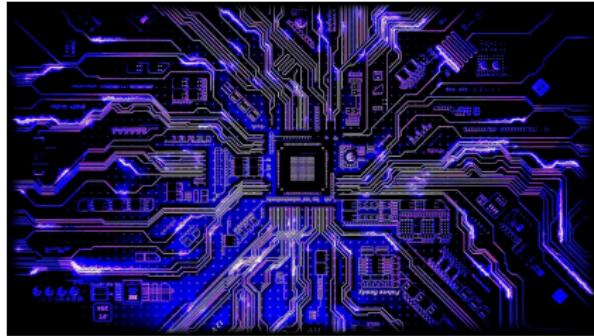
Weekly Reminders!



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