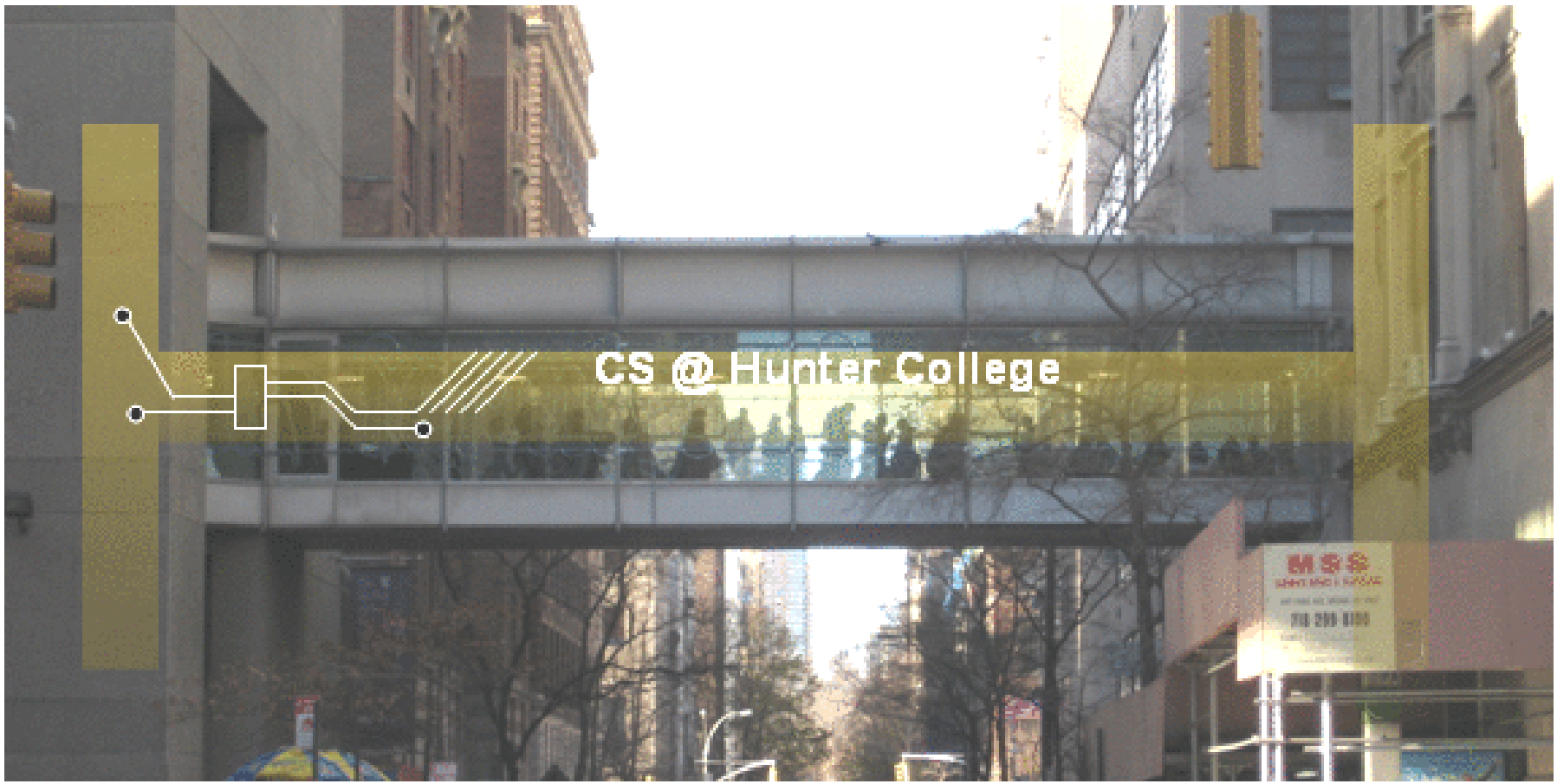
CSci 127: Introduction to Computer Science



Finished the lecture preview?

hunter.cuny.edu/csci

Today's Topics

- Recap: Colors
- 2D Arrays & Image Files
- Design Challenge: Airplanes
- Decisions

index, slice, and split of string and list

```
string = "I love python!" #Can we name string as str?
#Better not. Otherwise, error happens when calling
#str(12) to convert integer 12 to string "12" later
    on.
#This is because, by
#str = "I love python!",
#str is redefined as a variable. As a result, str
#cannot be used as a function name in the same
    program.

print(string[2:6])
print(string[-7:-1])
print(string[2:6:2])
print(string[-7:-1:2])
print(string[-1])
print(string[:-1])
```

index, slice, and split of string and list: II

```
#Get a list of words from a sentence. That is,
#split a string to a list of words by a delimiter.
mylist = string[:-1]. split (' ') #delimiter is ' '

#Concatenate elements of a list to get a string
#using join method.
string2 = ' '.join(mylist)
print("string2 =", string2)
```

index, slice, and split of string and list: III

```
print(mylist)
print(len(mylist))
print(mylist[0])
print(mylist[0:2])
print(mylist[-1])
print(mylist[0::2])
```

index, slice, and split of string and list: IV

```
abbr = ""
for word in mylist: #mylist is ['I', 'love', 'python']
    abbr = word[-1] + abbr
    #word[-1] is the last character in word,
    #which is a string object.
    #pad last character of word to left of abbr

print(abbr)
```

index, slice, and split of string and list: V

```
abbr2 = ""
for word in mylist: #mylist is ['I', 'love', 'python']
    abbr2 += word[-1] #same as abbr2 = abbr2 + word
        [-1]
    #pad last character of word to right of abbr2

print(abbr2)
```

link to program

Today's Topics

- **Recap: Colors**
- 2D Arrays & Image Files
- Design Challenge: Airplanes
- Decisions

Challenge (Group Work)

EmpID: CSci 127 Mock Final, S19

2. (a) Fill in the boxes with the appropriate hexcode to change the color to match the comments:

```
import turtle
thomasH = turtle.Turtle()
```

i. `#Change thomasH to be the color black:`

`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

ii. `#Change thomasH to be the color white:`

`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

iii. `#Change thomasH to be the brightest color blue:`

`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

iv. `#Change thomasH to be the color purple:`

`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

v. `#Change thomasH to be the color gray:`

`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

Challenge (Group Work)

EmpID: CSci 127 Mock Final, S19

2. (a) Fill in the boxes with the appropriate hexcode to change the color to match the comments:

```
import turtle
thomasH = turtle.Turtle()
```

i. `#Change thomasH to be the color black:`
`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

ii. `#Change thomasH to be the color white:`
`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

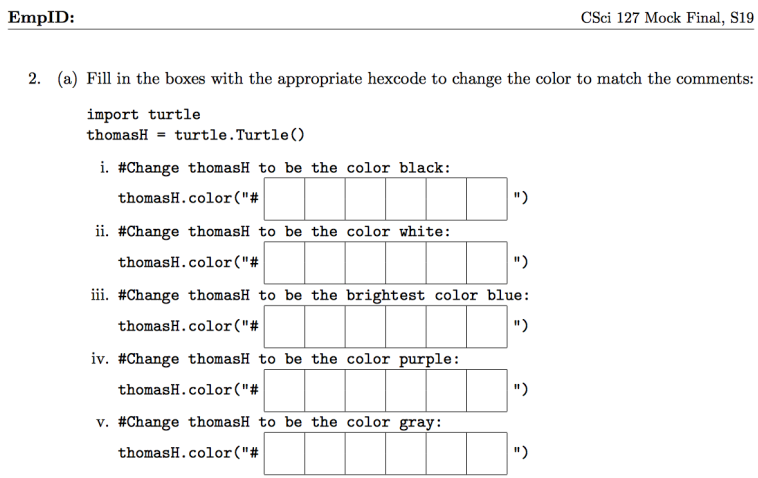
iii. `#Change thomasH to be the brightest color blue:`
`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

iv. `#Change thomasH to be the color purple:`
`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

v. `#Change thomasH to be the color gray:`
`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

- Need to fill in hexcodes (always start with #):

Challenge (Group Work)



EmpID: CSci 127 Mock Final, S19

2. (a) Fill in the boxes with the appropriate hexcode to change the color to match the comments:

```
import turtle
thomasH = turtle.Turtle()
```

i. `#Change thomasH to be the color black:`
`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

ii. `#Change thomasH to be the color white:`
`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

iii. `#Change thomasH to be the brightest color blue:`
`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

iv. `#Change thomasH to be the color purple:`
`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

v. `#Change thomasH to be the color gray:`
`thomasH.color("#` [ ][ ][ ][ ][ ][ ] `")`

- Need to fill in hexcodes (always start with #): `R R G G B B`
- Black: `0 0 0 0 0 0`
- White: `F F F F F F`
- Blue: `0 0 0 0 F F`
- Purple: `F F 0 0 F F`
- Gray: `4 2 4 2 4 2` (any choice where RR = GG = BB and RR, GG, BB not 00 or FF).

Recap: Colors

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	

- Can specify by name. See named color in python and scroll down to section CSS color.
- Can specify by numbers:
  - Amount of Red, Green, and Blue (RGB).
  - Adding light, not paint:
    - ★ Black: 0% red, 0% green, 0% blue
    - ★ White: 100% red, 100% green, 100% blue

Recap: Colors

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	

- Can specify by numbers (RGB):
  - Fractions of each:
    e.g. `(1.0, 0, 0)` is 100% red, no green, and no blue.
  - 8-bit colors: numbers from 0 to 255 $= 2^8 - 1$, a total of $2^8 = 256$ choices (related: 3-bit has 2^3 choices, from 0 to $2^3 - 1 = 7$):
    e.g. `(0, 255, 0)` is no red, 100% green, and no blue.
  - Hexacodes (base-16 numbers)...

Decimal and Hexadecimal

	decimal	hexadecimal
base	10	16
digits	0-9	0-9, A (10) - F (15)
eg	$123 = 1 * 10^2 + 2 * 10^1 + 3 * 10^0$	$7B_{16} = 7 * 16^1 + B * 16^0 = 112 + 11 * 1 = 123_{10}$
	$255 = 2 * 10^2 + 5 * 10^1 + 5 * 10^0$	$FF_{16} = 15 * 16^1 + 15 = 255_{10}$

Steps to convert hexadecimal to decimal

1. Start from rightmost to leftmost digit, label exponent as 0, 1, 2, ...
2. Multiple each digit by $base^{exponent}$, where base is 16 for hexadecimal numbers. Digit A - F are converted to 10 - 15, respectively.
3. Add the products in the second step up.

Decimal and Hexadecimal: II

Steps to convert decimal to hexadecimal

1. Divide the number by base 16. Calculate quotient and remainder.
2. Set the quotient to be the number. Repeat the above step until quotient is zero.
3. Connect the remainders **backwards**.

```
Convert 123 to decimal  Convert 123 to hexadecimal

10 | 123                16 | 123
   +-----                  +-----
10 |  12    3           16 |   7     11 (B in hexadecimal)
   +-----                  +-----
10 |   1    2                0      7
   +-----
       0    1
```

Colors

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	

- Can specify by numbers (RGB):
  - Fractions of each:
    e.g. `(1.0, 0, 0)` is 100% red, no green, and no blue.
  - 8-bit colors: numbers from 0 to 255:
    e.g. `(0, 255, 0)` is no red, 100% green, and no blue.
  - Hexcodes (base-16 numbers):

Colors

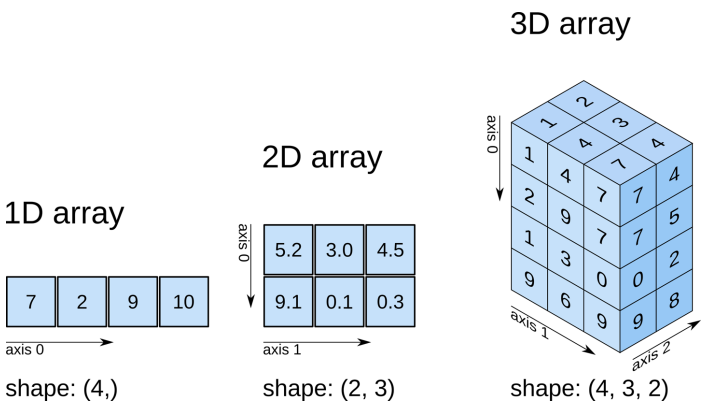
Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	

- Can specify by numbers (RGB):
  - Fractions of each:
    e.g. `(1.0, 0, 0)` is 100% red, no green, and no blue.
  - 8-bit colors: numbers from 0 to 255:
    e.g. `(0, 255, 0)` is no red, 100% green, and no blue.
  - Hexcodes (base-16 numbers):
    e.g. `#0000FF` is no red, no green, and 100% blue.

Today's Topics

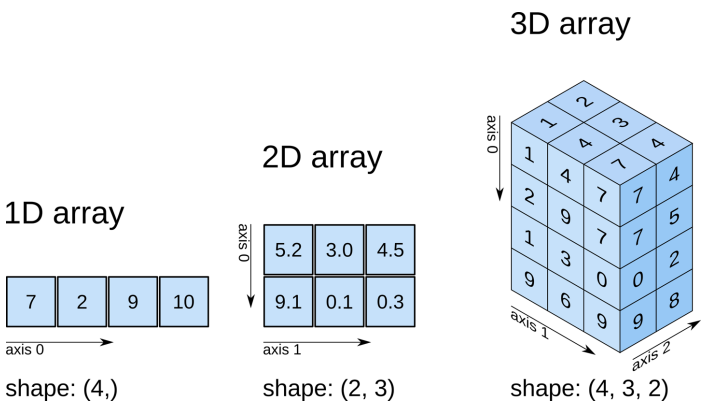
- Recap: Colors
- **2D Arrays & Image Files**
- Design Challenge: Airplanes
- Decisions

Arrays



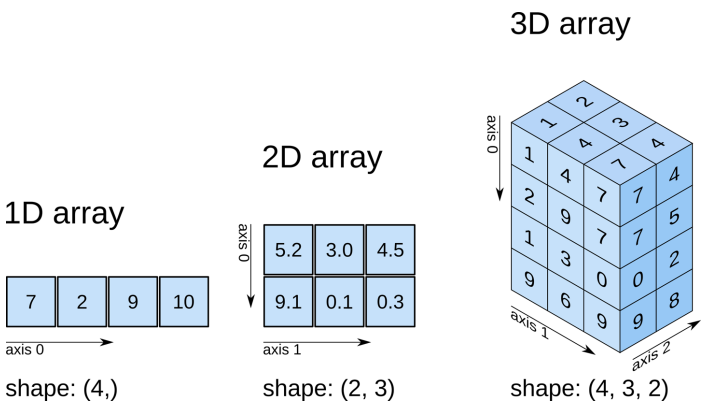
- An **array** is a sequence of elements, much like a list.

Arrays



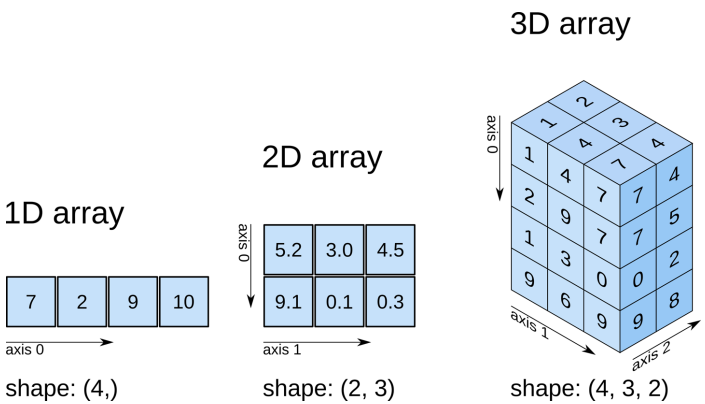
- An **array** is a sequence of elements, much like a list.
- A **2D array** is like a grid of elements, think a list of lists.

Arrays



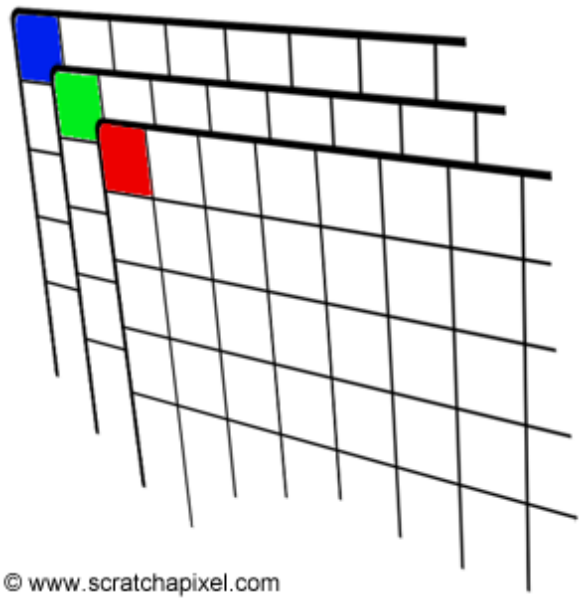
- An **array** is a sequence of elements, much like a list.
- A **2D array** is like a grid of elements, think a list of lists.
- Can keep on adding dimensions (3D, etc.)

Arrays

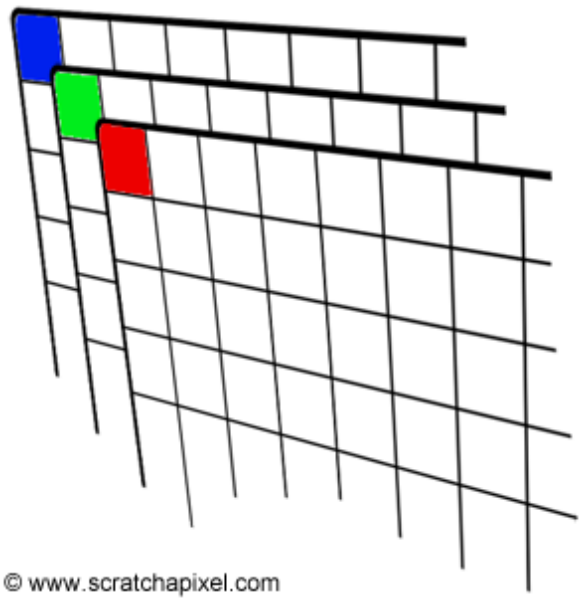


- An **array** is a sequence of elements, much like a list.
- A **2D array** is like a grid of elements, think a list of lists.
- Can keep on adding dimensions (3D, etc.)
- Can access pieces/slices as we do with strings and lists

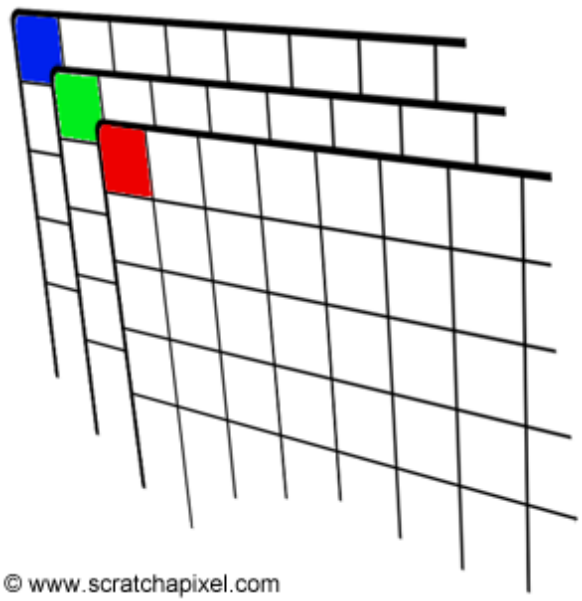
Images

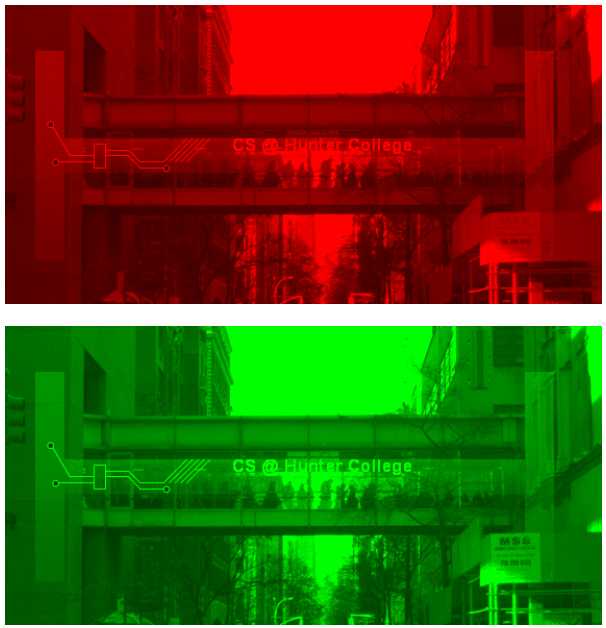


Images

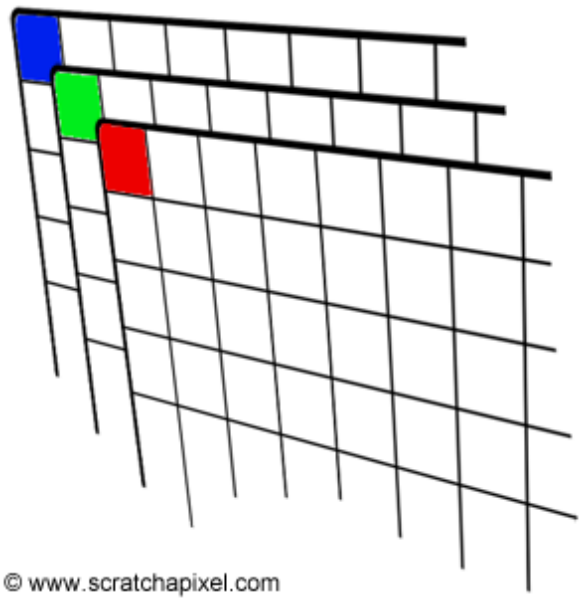


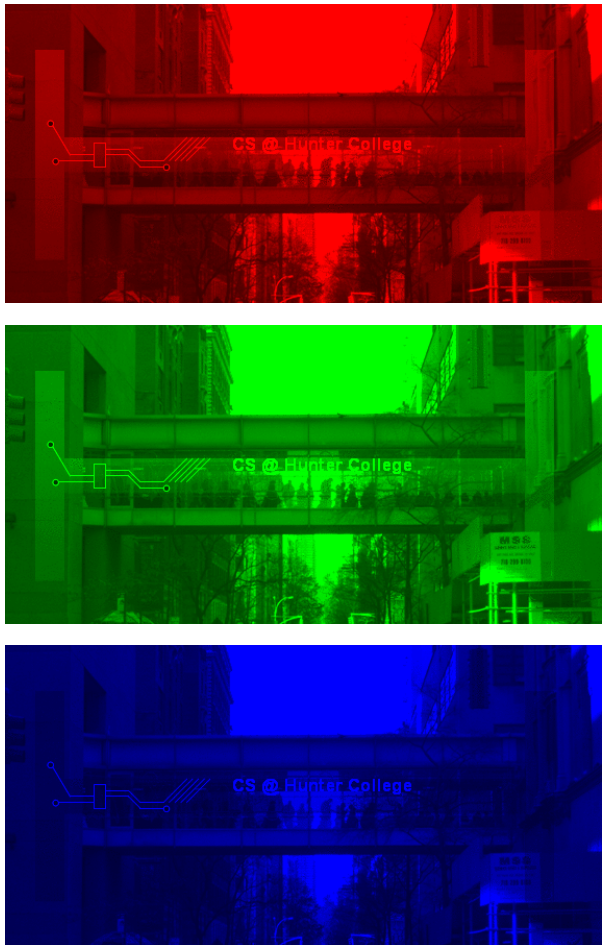
Images



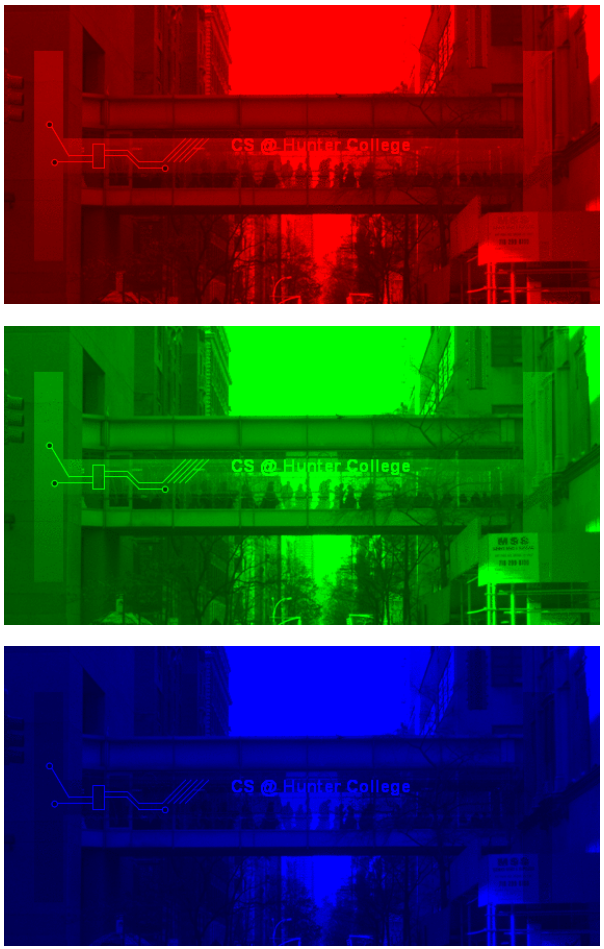


Images


© www.scratchapixel.com

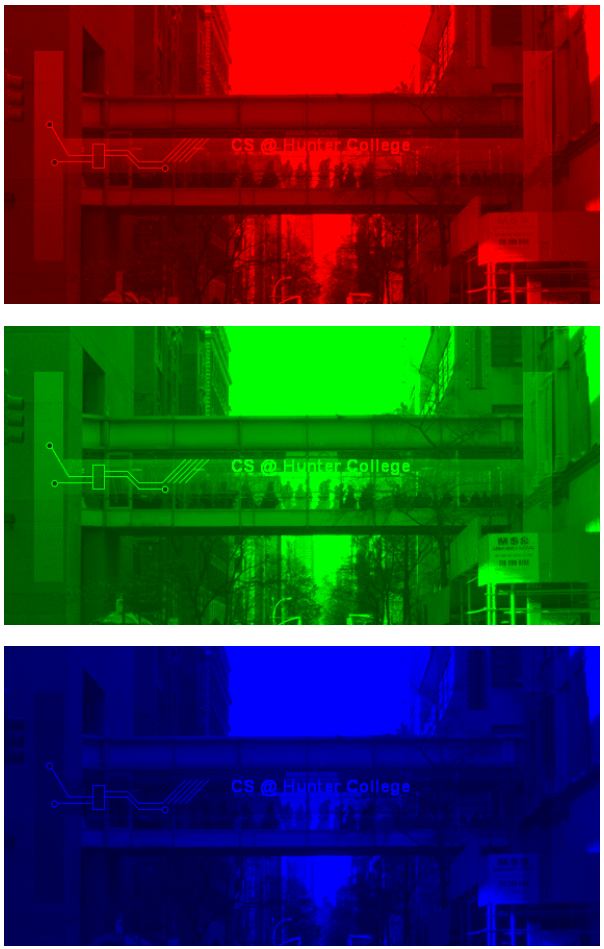

CS @ Hunter College
CS @ Hunter College
CS @ Hunter College

Useful Packages



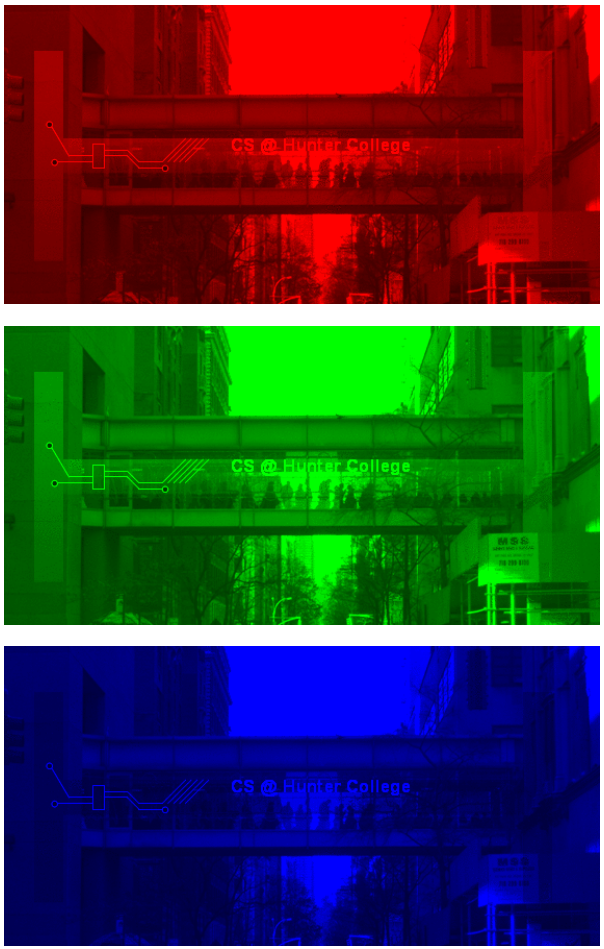
- We will use 2 useful packages for images:

Useful Packages



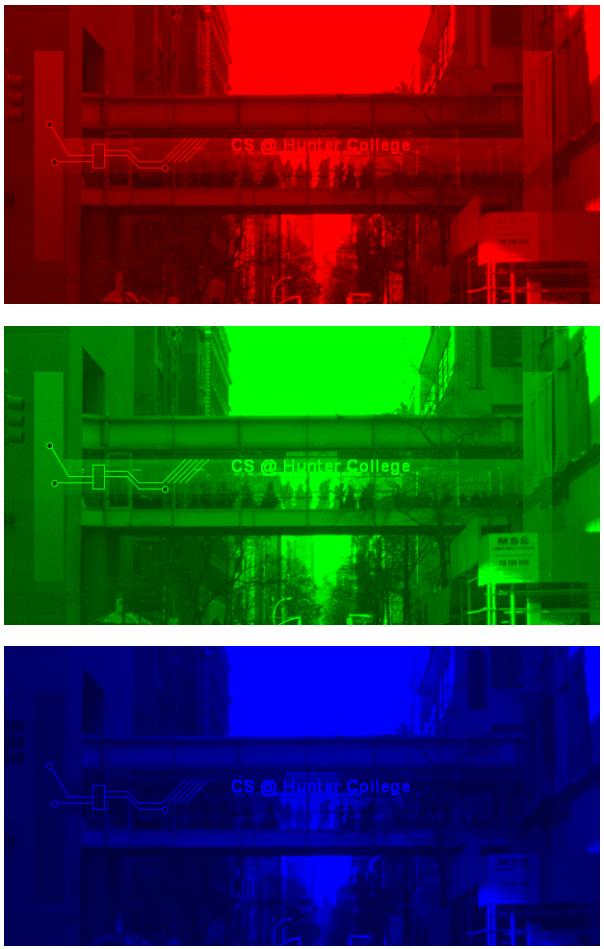
- We will use 2 useful packages for images:
  - `numpy`: numerical analysis package

Useful Packages



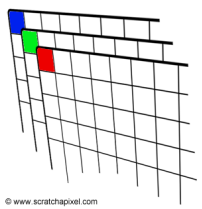
- We will use 2 useful packages for images:
  - `numpy`: numerical analysis package
  - `pyplot`: part of `matplotlib` for making graphs and plots

Useful Packages



- We will use 2 useful packages for images:
  - `numpy`: numerical analysis package
  - `pyplot`: part of `matplotlib` for making graphs and plots
- See lab notes for installing on your home machine.

Images with pyplot and numpy



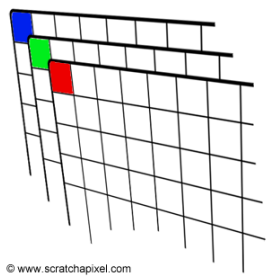

```
1  import matplotlib.pyplot as plt
2  #plt reads a picture and put the result in an array
3  #might need to run the following for one time
4  #if matplotlib is not installed before
5  #pip3 install matplotlib
6  import numpy as np
7  #np is to work with array
8
9  img = plt.imread('csBridge.png')
10 plt.imshow(img)
11 plt.show()
12
13 img2 = img.copy() #copy img to img2
```

Images with `pyplot` and `numpy`

```
1  img2[:, :, 1] = 0 #turn green channel at 1 to 0
2  img2[:, :, 2] = 0 #turn blue channel at 2 tp 0
3
4  plt.imshow(img2)
5  plt.show()
6
7  plt.imsave('red_csBridge.png', img2)
8  #save img2 to red_csBridge.png
```

Creating Images

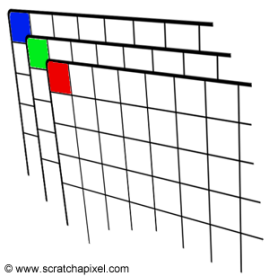
To create an image from scratch:



Creating Images

To create an image from scratch:

1. Import the libraries.

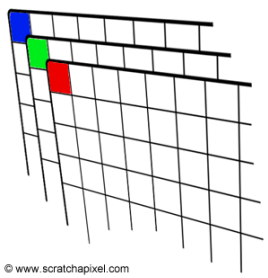


Creating Images

To create an image from scratch:

1. Import the libraries.

```
import matplotlib.pyplot as plt
import numpy as np
```



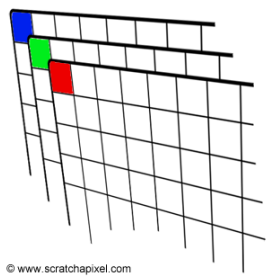
Creating Images

To create an image from scratch:

1. Import the libraries.

```
import matplotlib.pyplot as plt
import numpy as np
```

2. Create the image– easy to set all color



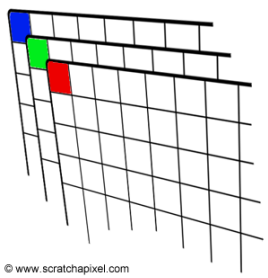
Creating Images

To create an image from scratch:

1. Import the libraries.

```
import matplotlib.pyplot as plt
import numpy as np
```

2. Create the image– easy to set all color
   1. to 0% (black):



Creating Images

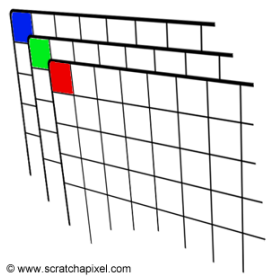
To create an image from scratch:

1. Import the libraries.

```
import matplotlib.pyplot as plt
import numpy as np
```

2. Create the image– easy to set all color
   1. to 0% (black):

```
img = np.zeros( (num,num,3) )
```



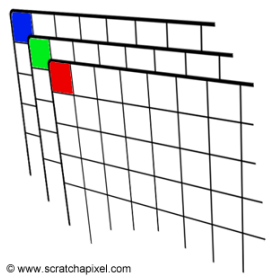
Creating Images

To create an image from scratch:

1. Import the libraries.

```
import matplotlib.pyplot as plt
import numpy as np
```

2. Create the image– easy to set all color
   1. to 0% (black):
      `img = np.zeros( (num,num,3) )`
   2. to 100% (white):



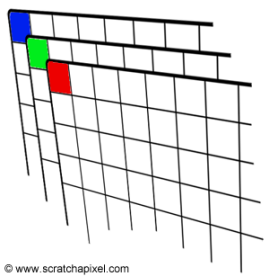
Creating Images

To create an image from scratch:

1. Import the libraries.

```
import matplotlib.pyplot as plt
import numpy as np
```

2. Create the image– easy to set all color
   1. to 0% (black):
      `img = np.zeros( (num,num,3) )`
   2. to 100% (white):
      `img = np.ones( (num,num,3) )`



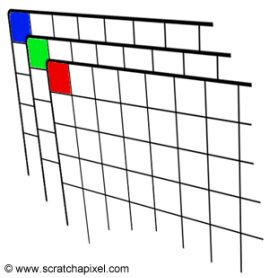
Creating Images

To create an image from scratch:

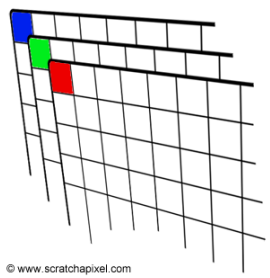
1. Import the libraries.

```
import matplotlib.pyplot as plt
import numpy as np
```

2. Create the image– easy to set all color
   1. to 0% (black):
      `img = np.zeros( (num,num,3) )`
   2. to 100% (white):
      `img = np.ones( (num,num,3) )`
3. *Do stuff to the pixels to make your image*



Creating Images



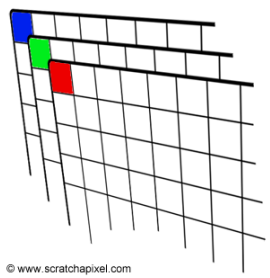
To create an image from scratch:

1. Import the libraries.

```
import matplotlib.pyplot as plt
import numpy as np
```

2. Create the image– easy to set all color
   1. to 0% (black):
      `img = np.zeros( (num,num,3) )`
   2. to 100% (white):
      `img = np.ones( (num,num,3) )`
3. *Do stuff to the pixels to make your image*
4. You can display your image:

Creating Images



To create an image from scratch:

1. Import the libraries.

```
import matplotlib.pyplot as plt
import numpy as np
```

2. Create the image– easy to set all color
   1. to 0% (black):
      `img = np.zeros( (num,num,3) )`
   2. to 100% (white):
      `img = np.ones( (num,num,3) )`
3. *Do stuff to the pixels to make your image*
4. You can display your image:

```
plt.imshow(img)
plt.show()
```

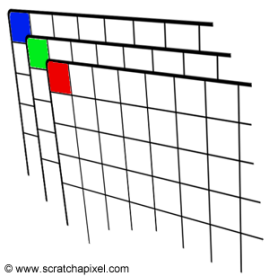
Creating Images

To create an image from scratch:

1. Import the libraries.

```
import matplotlib.pyplot as plt
import numpy as np
```

2. Create the image– easy to set all color
   1. to 0% (black):
      `img = np.zeros( (num,num,3) )`
   2. to 100% (white):
      `img = np.ones( (num,num,3) )`



3. *Do stuff to the pixels to make your image*
4. You can display your image:

```
plt.imshow(img)
plt.show()
```

5. And save your image:

Creating Images

To create an image from scratch:

1. Import the libraries.

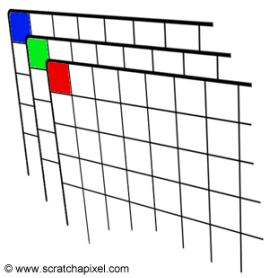
```
import matplotlib.pyplot as plt
import numpy as np
```

2. Create the image– easy to set all color
   1. to 0% (black):
      `img = np.zeros( (num,num,3) )`
   2. to 100% (white):
      `img = np.ones( (num,num,3) )`
3. *Do stuff to the pixels to make your image*
4. You can display your image:

```
plt.imshow(img)
plt.show()
```

5. And save your image:

```
plt.imsave('myImage.png', img)
```



Two Dimensional Array Slicing

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

Two Dimensional Array Slicing: II

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

Two Dimensional Array Slicing: III

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

Two Dimensional Array Slicing: III

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

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

Two Dimensional Array Slicing: III

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

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

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

print

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

Two Dimensional Array Slicing: IV

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

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

Two Dimensional Array Slicing: IV

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

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

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

Print out

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

Two Dimensional Array Slicing: V

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

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

Two Dimensional Array Slicing: V

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

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

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

Print out

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

Two Dimensional Array Slicing: VI

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

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

Two Dimensional Array Slicing: VI

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

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

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

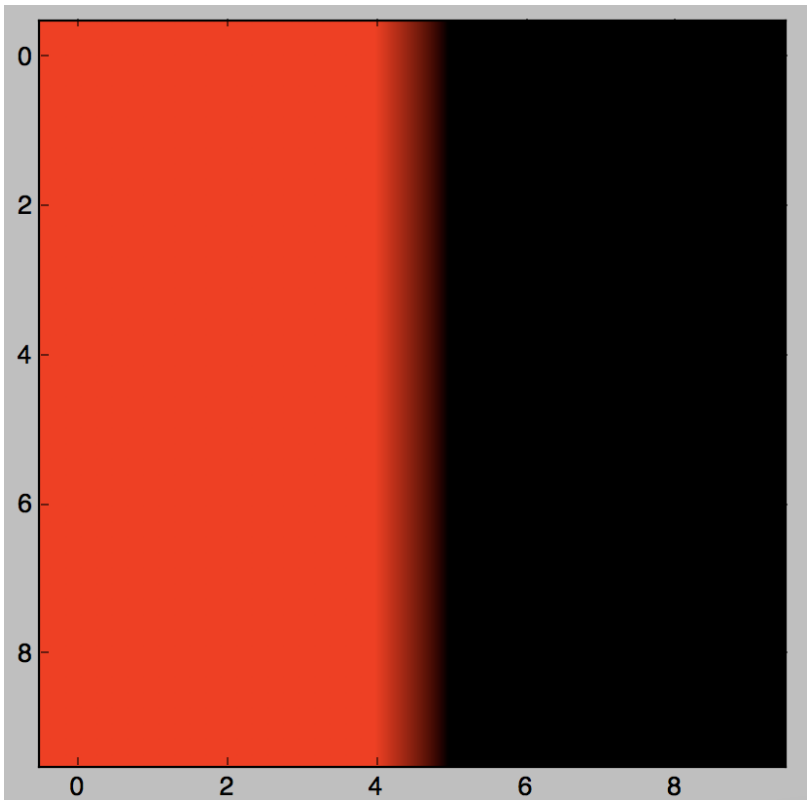
print

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

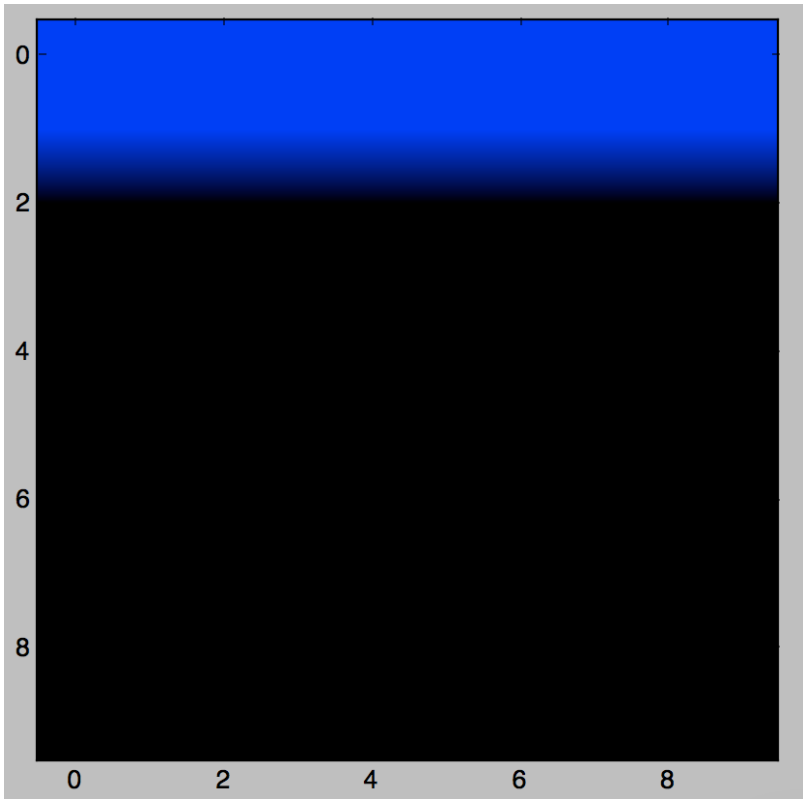
Slicing & Image Examples

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.

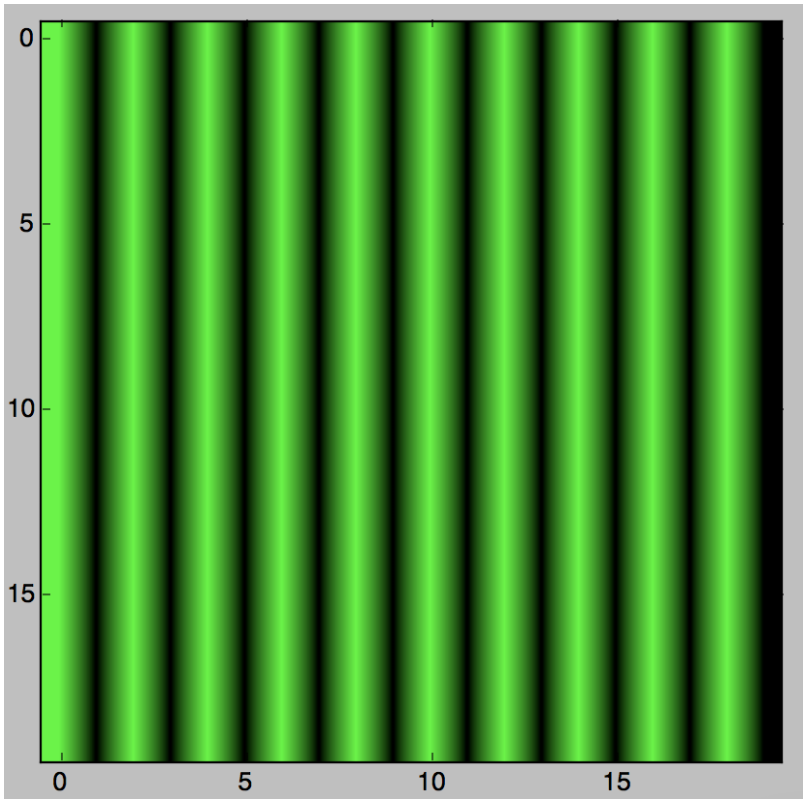
Slicing & Image Examples

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:
  - ```
    img = np.zeros( (10,10,3) )
    img[0:10,0:5,0:1] = 1
    ```
    ```

Slicing & Image Examples

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:
  - ```
    img = np.zeros( (10,10,3) )
    img[0:10,0:5,0:1] = 1
    ```


    ```

Slicing & Image Examples

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:
  - ```
    num = 10
    img = np.zeros( (num,num,3) )
    img[0:2,:,2:3] = 1.0
    ```
    ```

Slicing & Image Examples

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:
  - ```
    num = 10
    img = np.zeros( (num,num,3) )
    img[0:2,:,2:3] = 1.0
    ```


    ```

Slicing & Image Examples

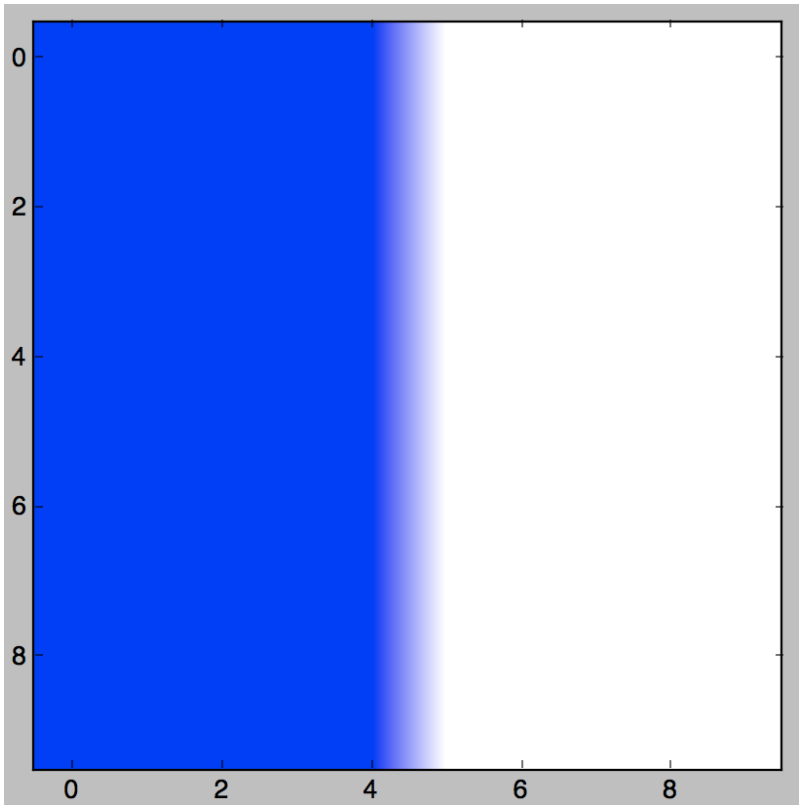
- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:
  - ```
    num = int(input('Enter size'))
    img = np.zeros( (num,num,3) )
    img[:,::2,1] = 1.0
    ```
    ```

Slicing & Image Examples

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:
  - ```
    num = int(input('Enter size'))
    img = np.zeros( (num,num,3) )
    img[:,::2,1] = 1.0
    ```


    ```

Challenge (Group Work)

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:
  - ```
    img = np.ones( (10,10,3) )
    img[0:10,0:5,0:2] = 0
    ```
    ```

Challenge (Group Work)

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:

  - ```
    img = np.ones( (10,10,3) )
    img[0:10,0:5,0:2] = 0
    ```

  - ```
    num = int(input('Enter size '))
    img = np.ones( (num,num,3) )
    img[::2,:,1:]  = 0
    ```

Challenge (Group Work)

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:

  - ```
    img = np.ones( (10,10,3) )
    img[0:10,0:5,0:2] = 0
    ```

  - ```
    num = int(input('Enter size '))
    img = np.ones( (num,num,3) )
    img[::2,:,1:]  = 0
    ```

  - ```
    img = np.zeros( (8,8,3) )
    img[::2,::2,0] = 1
    ```
    ```

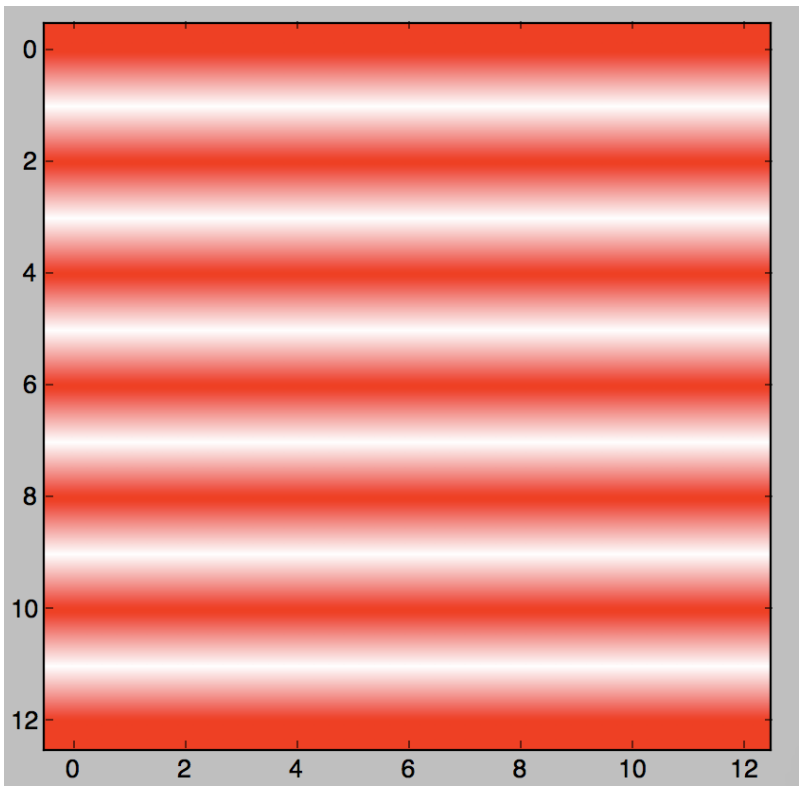
Challenge (Group Work):

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:

  - ```
    img = np.ones( (10,10,3) )
    img[0:10,0:5,0:2] = 0
    ```
    ```

Challenge (Group Work):

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:

  - ```
    img = np.ones( (10,10,3) )
    img[0:10,0:5,0:2] = 0
    ```


    ```

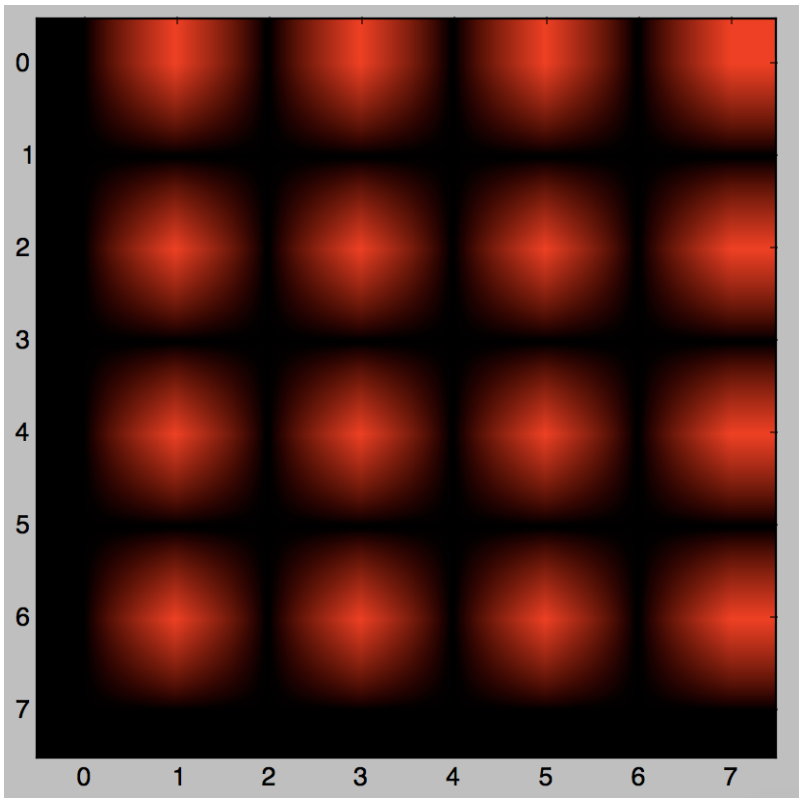
Challenge (Group Work)

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:

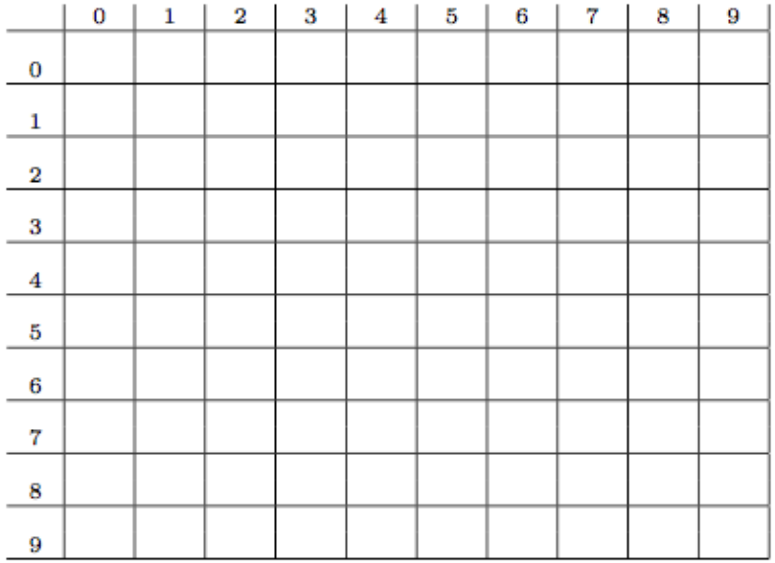
  - ```
    num = int(input('Enter size '))
    img = np.ones( (num,num,3) )
    img[::2,:,1:]  = 0
    ```
    ```

Challenge (Group Work)

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:

  - ```
    num = int(input('Enter size '))
    img = np.ones( (num,num,3) )
    img[::2,:,1:]  = 0
    ```


    ```

Challenge (Group Work)

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:

  - ```
    img = np.zeros( (8,8,3) )
    img[::2,1::2,0] = 1
    ```
    ```

Challenge (Group Work)

- Basic pattern: *img[rows, columns, channels]* with: *start:stop:step*.
- Assuming the libraries are imported, what do the following code fragments produce:

  - ```
    img = np.zeros( (8,8,3) )
    img[::2,1::2,0] = 1
    ```


    ```

Challenge (Group Work)



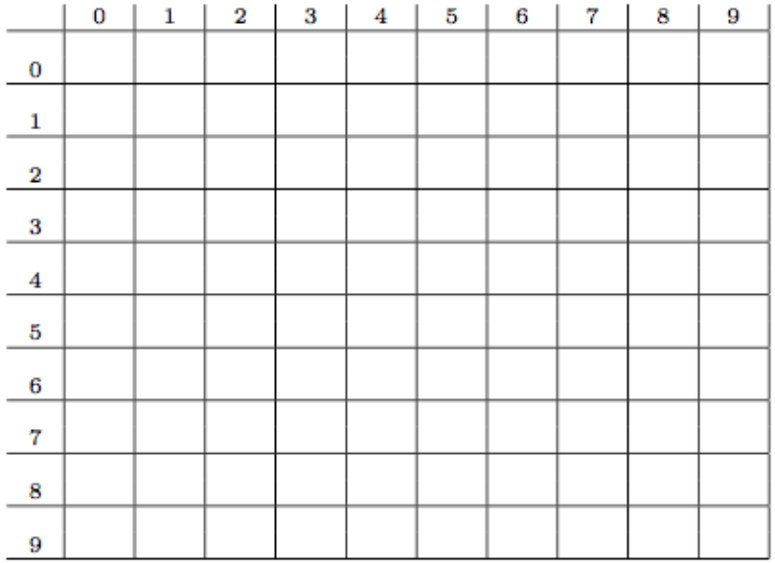
1. Design a 10 by 10 logo for Hunter College that contains a purple 'H'.

Challenge (Group Work)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

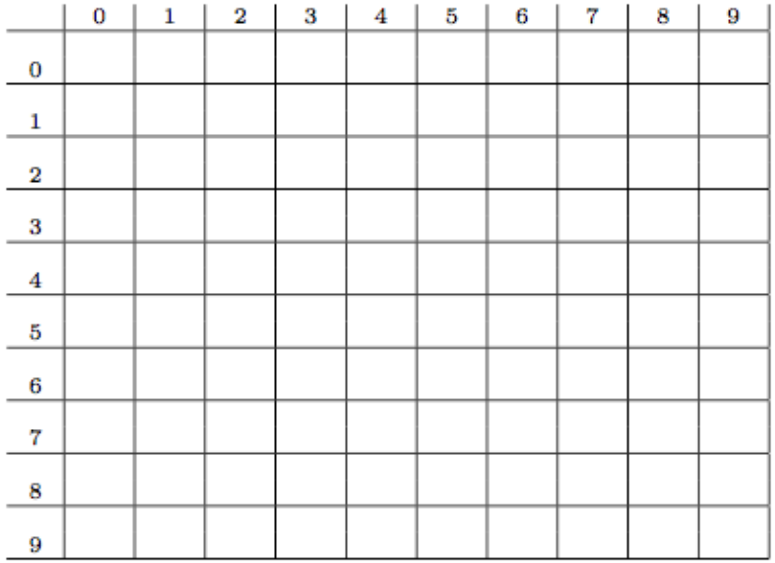
1. Design a 10 by 10 logo for Hunter College that contains a purple 'H'.
2. Your logo should only contain the colors purple and white.

Challenge (Group Work)



1. Design a 10 by 10 logo for Hunter College that contains a purple 'H'.
2. Your logo should only contain the colors purple and white.
3. How can you make Python draw the logo?
   Write down a "To Do" list of things you need to do.

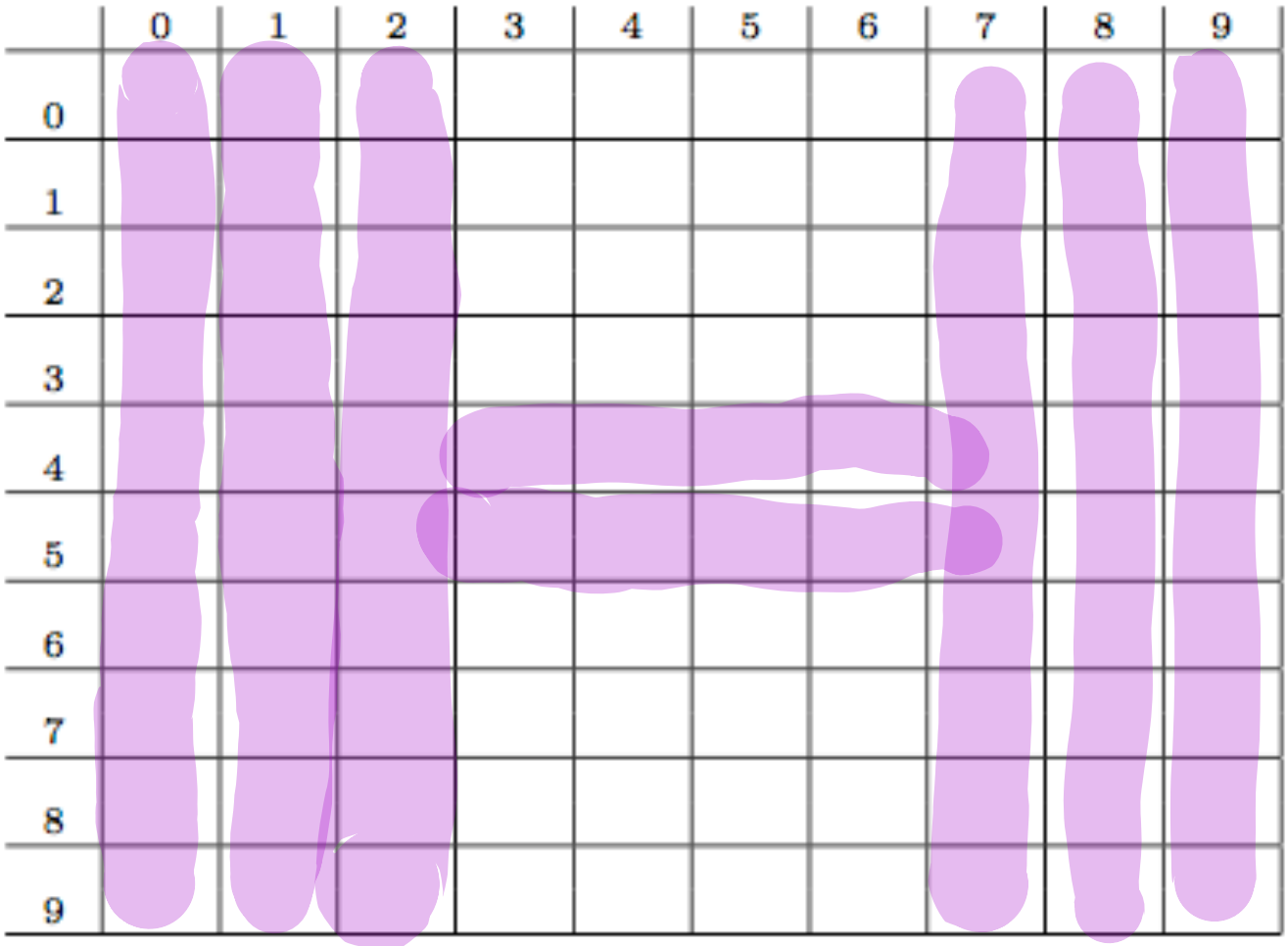
Challenge (Group Work)



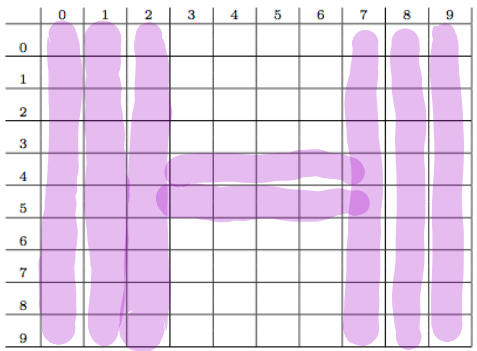
1. Design a 10 by 10 logo for Hunter College that contains a purple 'H'.
2. Your logo should only contain the colors purple and white.
3. How can you make Python draw the logo?
   Write down a "To Do" list of things you need to do.
4. If time, refine your steps above into a Python program.

Design a Hunter Logo

One possible solution:

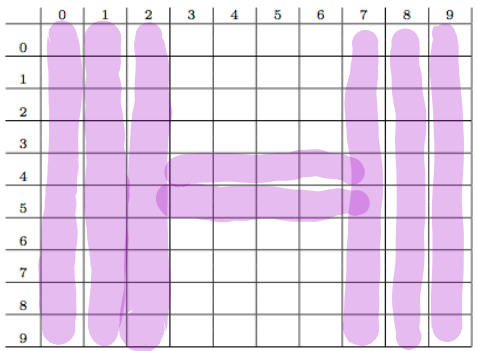


Design a Hunter Logo



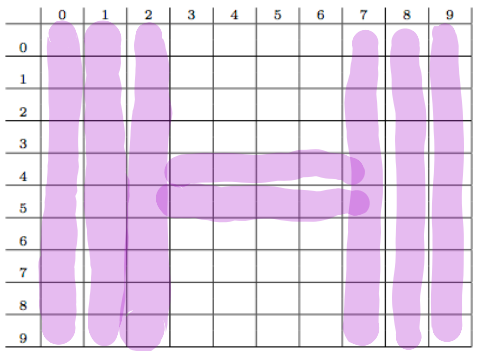
1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

Design a Hunter Logo



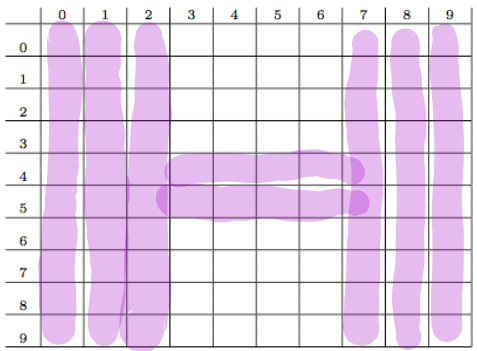
1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.
2. Set the 3 left columns to be purple.

Design a Hunter Logo



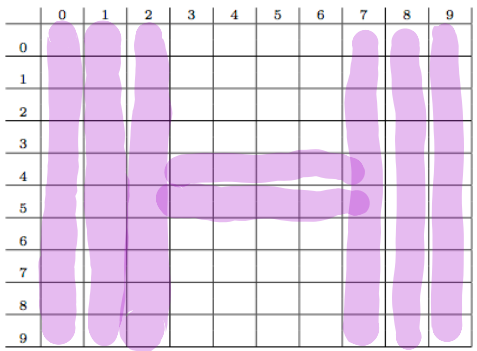
1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.
2. Set the 3 left columns to be purple.
3. Set the 3 right columns to be purple.

Design a Hunter Logo



1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.
2. Set the 3 left columns to be purple.
3. Set the 3 right columns to be purple.
4. Set the middle 2 rows to be purple.

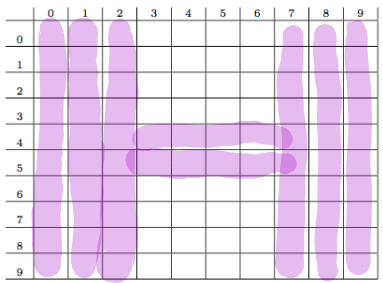
Design a Hunter Logo



1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.
2. Set the 3 left columns to be purple.
3. Set the 3 right columns to be purple.
4. Set the middle 2 rows to be purple.
5. Save `logo` array to a file.

Translating the Design to Code

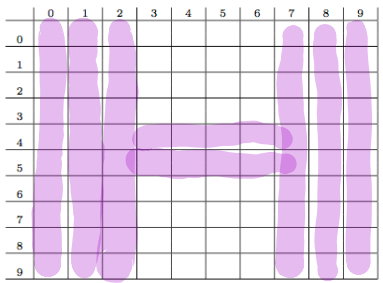
1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.



Translating the Design to Code

1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

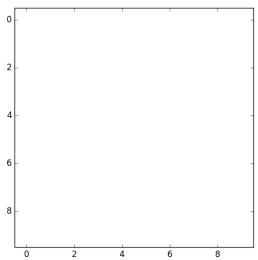
```
import matplotlib.pyplot as plt #import libraries for plotting
import numpy as np              #and for arrays (to hold images)
logoImg = np.ones((10,10,3))    #10x10 array with 3 sheets of 1's
```



Translating the Design to Code

1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

```
import matplotlib.pyplot as plt #import libraries for plotting
import numpy as np              #and for arrays (to hold images)
logoImg = np.ones((10,10,3))    #10x10 array with 3 sheets of 1's
```

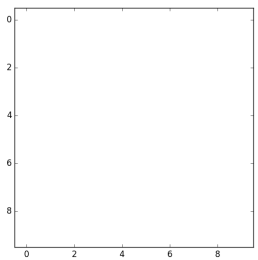


Translating the Design to Code

1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

```
import matplotlib.pyplot as plt #import libraries for plotting
import numpy as np              #and for arrays (to hold images)
logoImg = np.ones((10,10,3))    #10x10 array with 3 sheets of 1's
```

2. Set the 3 left columns to be purple.



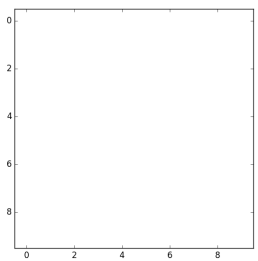
Translating the Design to Code

1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

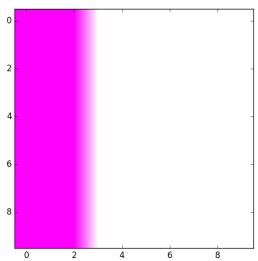
```
import matplotlib.pyplot as plt #import libraries for plotting
import numpy as np              #and for arrays (to hold images)
logoImg = np.ones((10,10,3))    #10x10 array with 3 sheets of 1's
```

2. Set the 3 left columns to be purple.

```
#To make purple, we'll keep red and blue at 100% and turn green to 0%
logoImg[:,:3,1] = 0 #Turn the green to 0 for first 3 columns
```



Translating the Design to Code



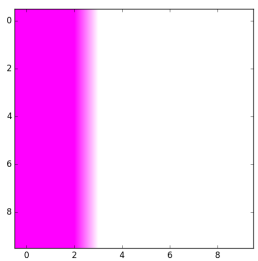
1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

```
import matplotlib.pyplot as plt #import libraries for plotting
import numpy as np              #and for arrays (to hold images)
logoImg = np.ones((10,10,3))    #10x10 array with 3 sheets of 1's
```

2. Set the 3 left columns to be purple.

```
#To make purple, we'll keep red and blue at 100% and turn green to 0%
logoImg[:,:3,1] = 0 #Turn the green to 0 for first 3 columns
```

Translating the Design to Code



1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

```
import matplotlib.pyplot as plt #import libraries for plotting
import numpy as np              #and for arrays (to hold images)
logoImg = np.ones((10,10,3))    #10x10 array with 3 sheets of 1's
```

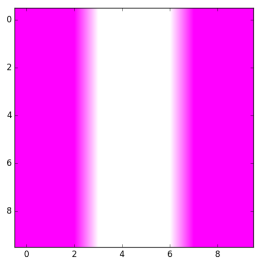
2. Set the 3 left columns to be purple.

```
#To make purple, we'll keep red and blue at 100% and turn green to 0%
logoImg[:,:3,1] = 0 #Turn the green to 0 for first 3 columns
```

3. Set the 3 right columns to be purple.

```
logoImg[:,-3:,1] = 0 #Turn the green to 0 for last 3 columns
```

Translating the Design to Code



1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

```
import matplotlib.pyplot as plt #import libraries for plotting
import numpy as np              #and for arrays (to hold images)
logoImg = np.ones((10,10,3))    #10x10 array with 3 sheets of 1's
```

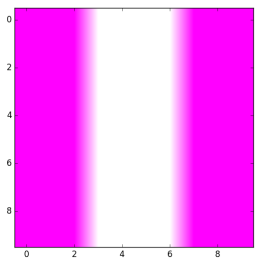
2. Set the 3 left columns to be purple.

```
#To make purple, we'll keep red and blue at 100% and turn green to 0%
logoImg[:,:3,1] = 0 #Turn the green to 0 for first 3 columns
```

3. Set the 3 right columns to be purple.

```
logoImg[:,-3:,1] = 0 #Turn the green to 0 for last 3 columns
```

Translating the Design to Code



1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

```
import matplotlib.pyplot as plt #import libraries for plotting
import numpy as np              #and for arrays (to hold images)
logoImg = np.ones((10,10,3))    #10x10 array with 3 sheets of 1's
```

2. Set the 3 left columns to be purple.

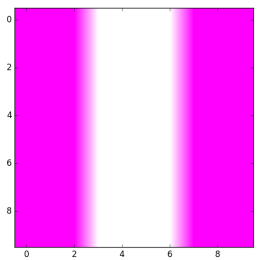
```
#To make purple, we'll keep red and blue at 100% and turn green to 0%
logoImg[:,:3,1] = 0 #Turn the green to 0 for first 3 columns
```

3. Set the 3 right columns to be purple.

```
logoImg[:,-3:,1] = 0 #Turn the green to 0 for last 3 columns
```

4. Set the middle 2 rows to be purple.

Translating the Design to Code



1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

```
import matplotlib.pyplot as plt #import libraries for plotting
import numpy as np              #and for arrays (to hold images)
logoImg = np.ones((10,10,3))    #10x10 array with 3 sheets of 1's
```

2. Set the 3 left columns to be purple.

```
#To make purple, we'll keep red and blue at 100% and turn green to 0%
logoImg[:,:3,1] = 0 #Turn the green to 0 for first 3 columns
```

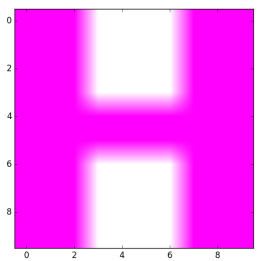
3. Set the 3 right columns to be purple.

```
logoImg[:,-3:,1] = 0 #Turn the green to 0 for last 3 columns
```

4. Set the middle 2 rows to be purple.

```
logoImg[4:6,:,1] = 0 #Turn the green to 0 for middle rows
```

Translating the Design to Code



1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

```
import matplotlib.pyplot as plt #import libraries for plotting
import numpy as np              #and for arrays (to hold images)
logoImg = np.ones((10,10,3))    #10x10 array with 3 sheets of 1's
```

2. Set the 3 left columns to be purple.

```
#To make purple, we'll keep red and blue at 100% and turn green to 0%
logoImg[:,:3,1] = 0 #Turn the green to 0 for first 3 columns
```

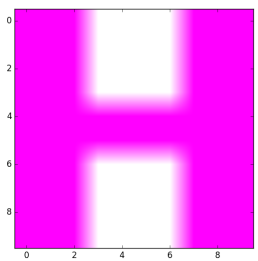
3. Set the 3 right columns to be purple.

```
logoImg[:,-3:,1] = 0 #Turn the green to 0 for last 3 columns
```

4. Set the middle 2 rows to be purple.

```
logoImg[4:6,:,1] = 0 #Turn the green to 0 for middle rows
```

Translating the Design to Code



1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

```
import matplotlib.pyplot as plt #import libraries for plotting
import numpy as np              #and for arrays (to hold images)
logoImg = np.ones((10,10,3))    #10x10 array with 3 sheets of 1's
```

2. Set the 3 left columns to be purple.

```
#To make purple, we'll keep red and blue at 100% and turn green to 0%
logoImg[:,:3,1] = 0 #Turn the green to 0 for first 3 columns
```

3. Set the 3 right columns to be purple.

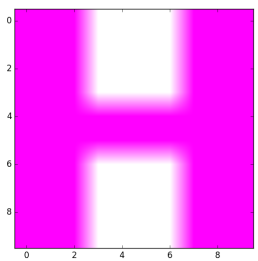
```
logoImg[:,-3:,1] = 0 #Turn the green to 0 for last 3 columns
```

4. Set the middle 2 rows to be purple.

```
logoImg[4:6,:,1] = 0 #Turn the green to 0 for middle rows
```

5. Save logo array to file.

Translating the Design to Code



1. Create a 10 by 10 array, `logo`, that starts out as all white pixels.

```
import matplotlib.pyplot as plt #import libraries for plotting
import numpy as np              #and for arrays (to hold images)
logoImg = np.ones((10,10,3))    #10x10 array with 3 sheets of 1's
```

2. Set the 3 left columns to be purple.

```
#To make purple, we'll keep red and blue at 100% and turn green to 0%
logoImg[:,:3,1] = 0 #Turn the green to 0 for first 3 columns
```

3. Set the 3 right columns to be purple.

```
logoImg[:,-3:,1] = 0 #Turn the green to 0 for last 3 columns
```

4. Set the middle 2 rows to be purple.

```
logoImg[4:6,:,1] = 0 #Turn the green to 0 for middle rows
```

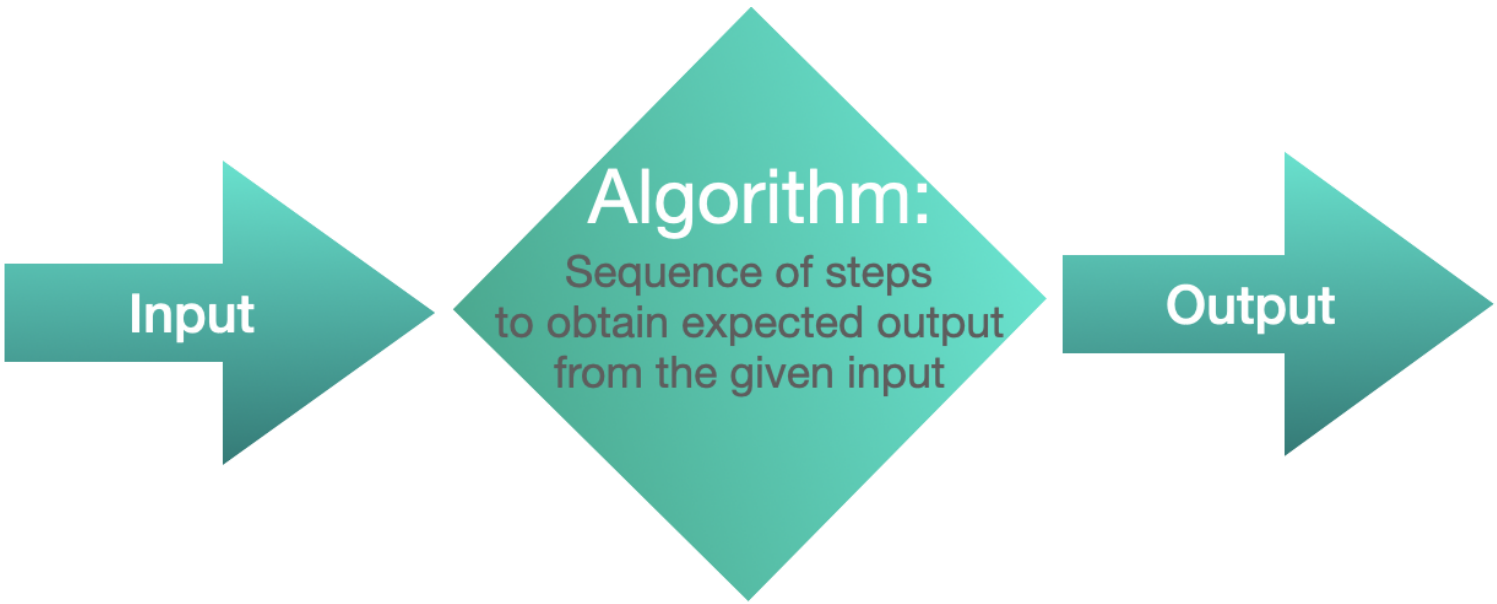
5. Save logo array to file.

```
plt.imsave("logo.png", logoImg) #Save the image to logo.png
```

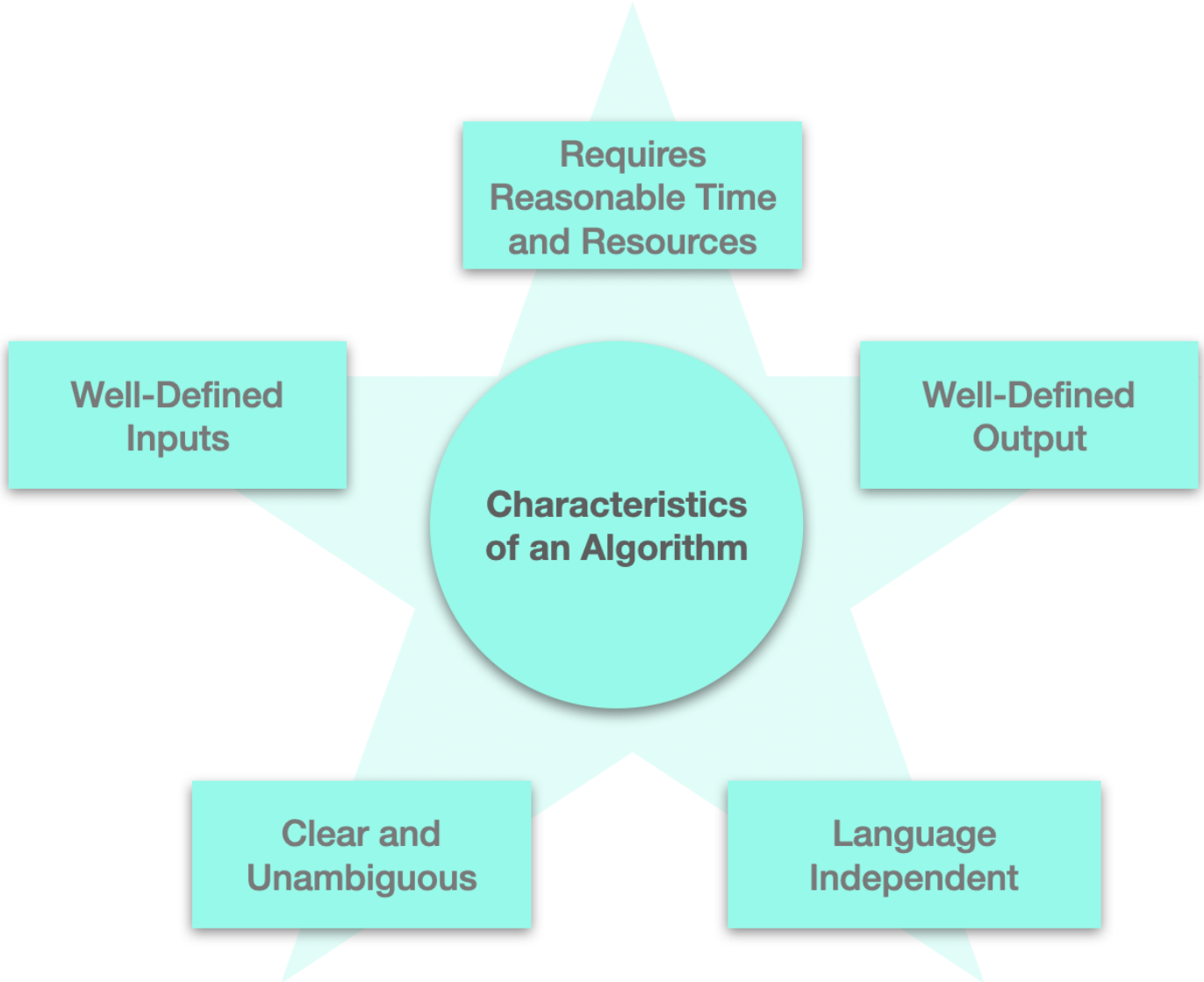
Today's Topics

- Recap: Colors
- 2D Arrays & Image Files
- **Design Challenge: Airplanes**
- Decisions

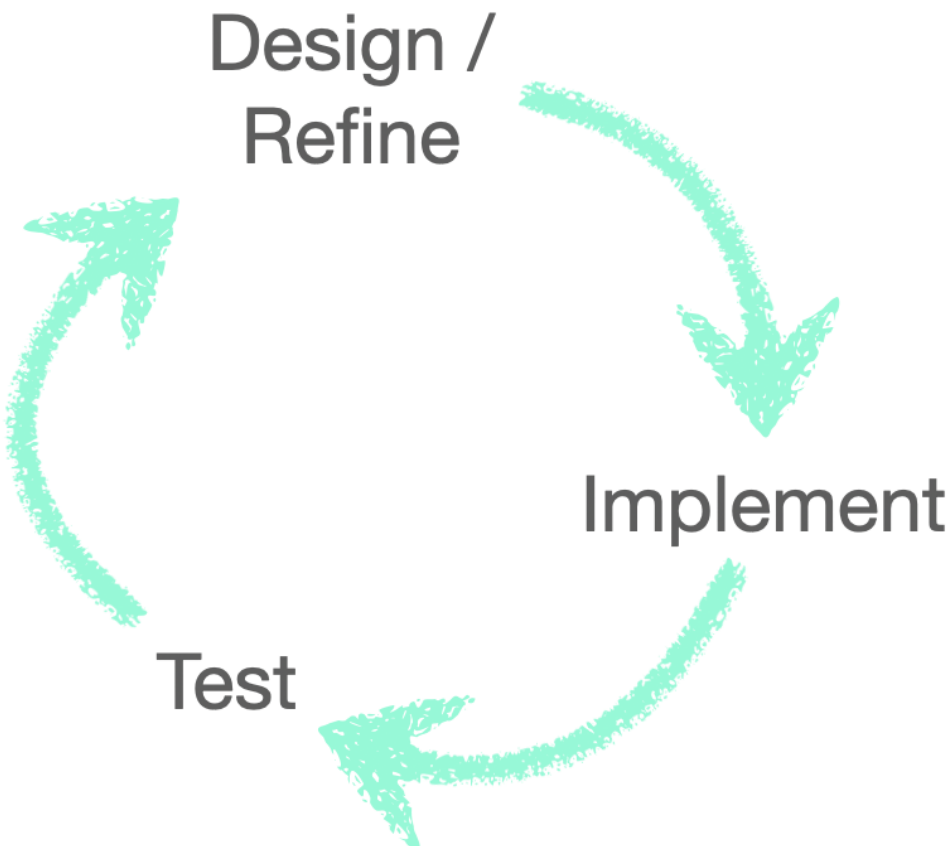
What is an Algorithm?



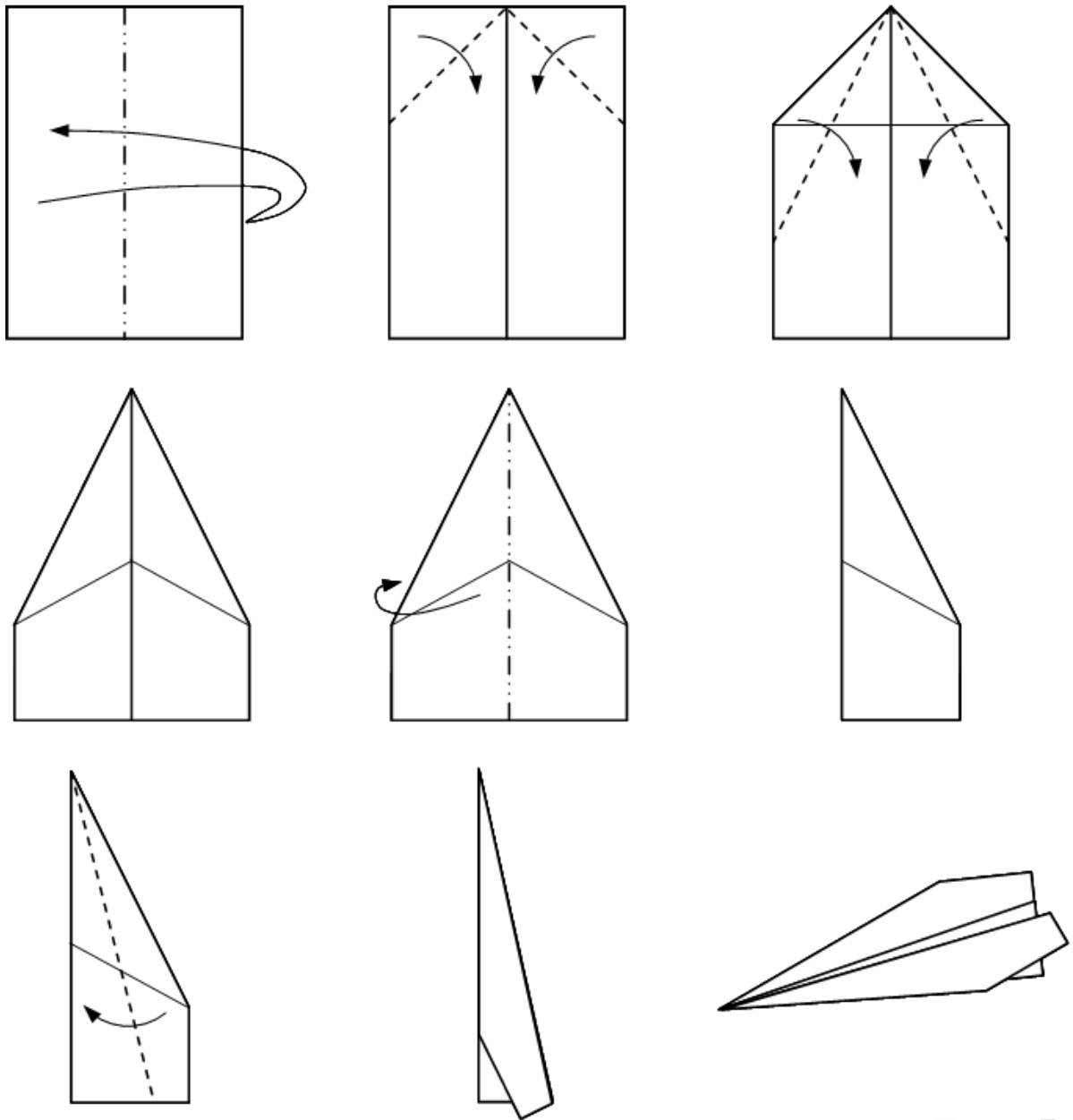
Characteristics of an Algorithm



Algorithm Design Cycle

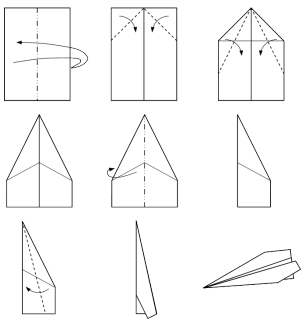


Design Challenge: Planes



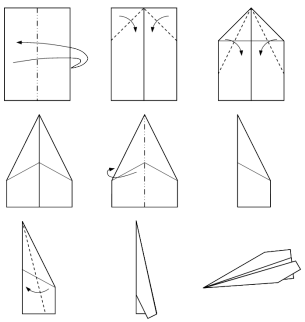
Design Challenge: Planes

- A classic write-an-algorithm challenge for introductory programming.



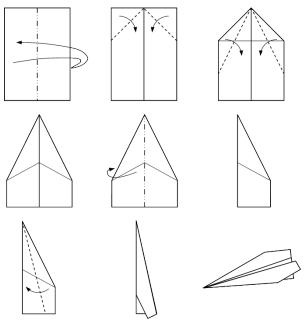
Design Challenge: Planes

- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist:



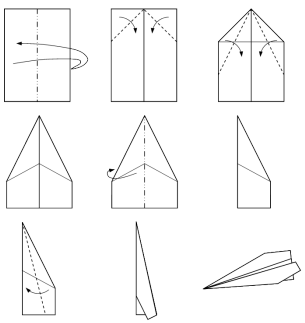
Design Challenge: Planes

- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs



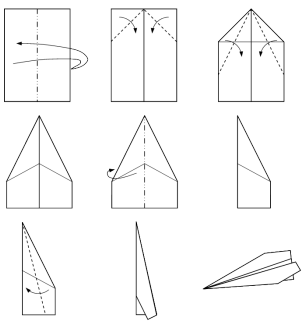
Design Challenge: Planes

- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.

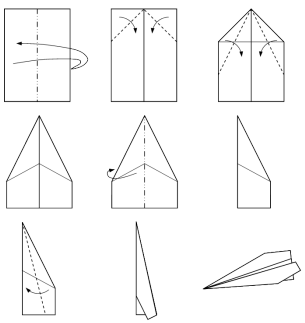


Design Challenge: Planes

- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - Exchange with another team.

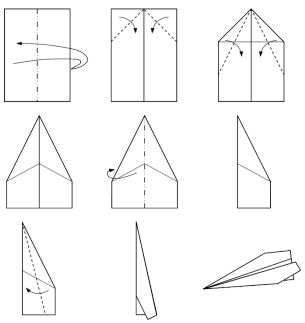


Design Challenge: Planes



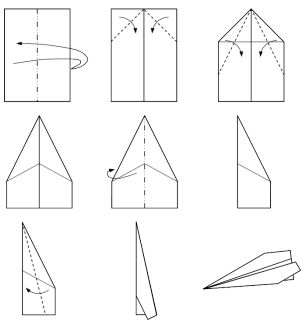
- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - Exchange with another team.
  - They build an airplane to your design (TEST FLIGHT) **without consulting you**.

Design Challenge: Planes



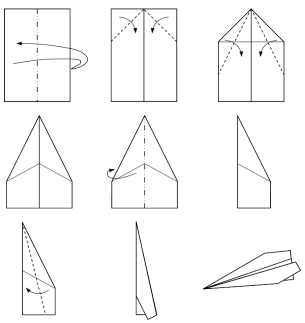
- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - Exchange with another team.
  - They build an airplane to your design (TEST FLIGHT) **without consulting you**.
  - You exchange test planes, and **revise your algorithm**.

Design Challenge: Planes



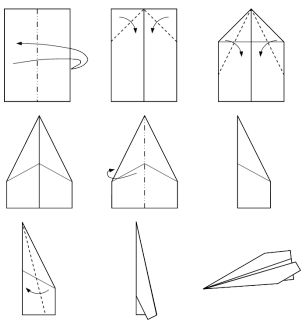
- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - Exchange with another team.
  - They build an airplane to your design (TEST FLIGHT) **without consulting you**.
  - You exchange test planes, and **revise your algorithm**.
  - The build team makes a copy of your revised paper airplane (FINAL FLIGHT)

Design Challenge: Planes



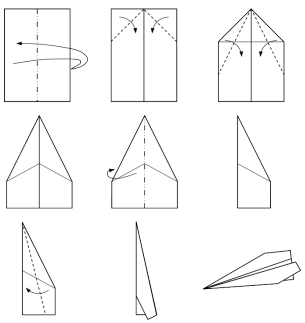
- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - Exchange with another team.
  - They build an airplane to your design (TEST FLIGHT) **without consulting you**.
  - You exchange test planes, and **revise your algorithm**.
  - The build team makes a copy of your revised paper airplane (FINAL FLIGHT) and flies it from the balcony (must be behind first row of seats).

Design Challenge: Planes



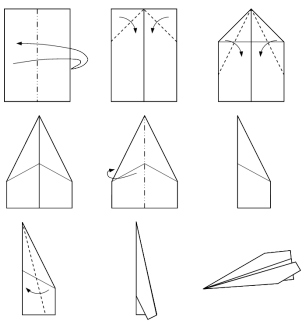
- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - Exchange with another team.
  - They build an airplane to your design (TEST FLIGHT) **without consulting you**.
  - You exchange test planes, and **revise your algorithm**.
  - The build team makes a copy of your revised paper airplane (FINAL FLIGHT) and flies it from the balcony (must be behind first row of seats).
  - Will be judged on closeness to the stage.

Design Challenge: Planes



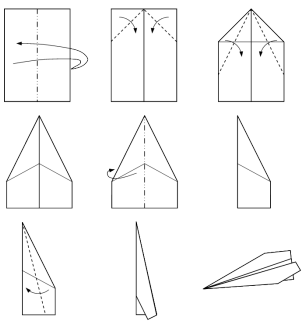
- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - Exchange with another team.
  - They build an airplane to your design (TEST FLIGHT) **without consulting you**.
  - You exchange test planes, and **revise your algorithm**.
  - The build team makes a copy of your revised paper airplane (FINAL FLIGHT) and flies it from the balcony (must be behind first row of seats).
  - Will be judged on closeness to the stage.
  - Winning design/build team gets chocolate.

Design Challenge: Planes



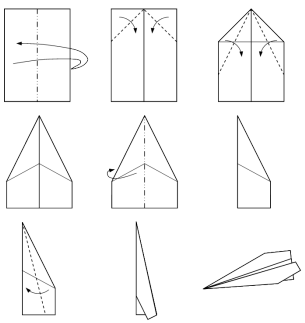
- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - Exchange with another team.
  - They build an airplane to your design (TEST FLIGHT) **without consulting you**.
  - You exchange test planes, and **revise your algorithm**.
  - The build team makes a copy of your revised paper airplane (FINAL FLIGHT) and flies it from the balcony (must be behind first row of seats).
  - Will be judged on closeness to the stage.
  - Winning design/build team gets chocolate.
- Remember to pick up all your airplanes!

Design Challenge: Initial Design **(2 Minutes)**



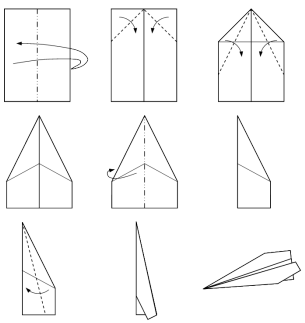
- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - **As a team, write down your design.**
  - Exchange with another team.
  - They build an airplane to your design (TEST FLIGHT) without consulting you.
  - You exchange test planes, and revise your algorithm.
  - The build team makes a copy of your revised paper airplane (FINAL FLIGHT) and flies it from the balcony (must be behind first row of seats).
  - Will be judged on closeness to the stage.
  - Winning design/build team gets chocolate.
- Remember to pick up all your airplanes!

Design Challenge: Test Build **(2 Minutes)**



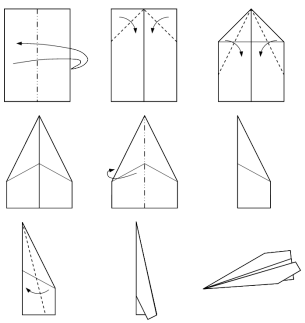
- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - **Exchange with another team.**
  - **They build an airplane to your design (TEST FLIGHT) without consulting you.**
  - You exchange test planes, and revise your algorithm.
  - The build team makes a copy of your revised paper airplane (FINAL FLIGHT) and flies it from the balcony (must be behind first row of seats).
  - Will be judged on closeness to the stage.
  - Winning design/build team gets chocolate.
- Remember to pick up all your airplanes!

Design Challenge: Revise Design **(3 Minutes)**



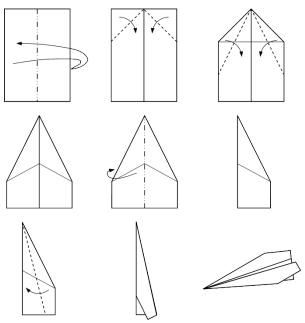
- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - Exchange with another team.
  - They build an airplane to your design (TEST FLIGHT) without consulting you.
  - **You exchange test planes, and revise your algorithm**.
  - The build team makes a copy of your revised paper airplane (FINAL FLIGHT) and flies it from the balcony (must be behind first row of seats).
  - Will be judged on closeness to the stage.
  - Winning design/build team gets chocolate.
- Remember to pick up all your airplanes!

Design Challenge: Build Final Planes **(2 Minutes)**



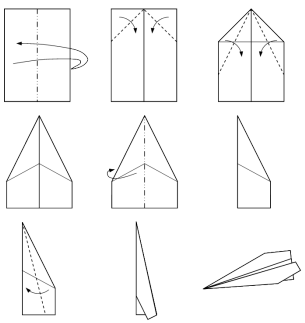
- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - Exchange with another team.
  - They build an airplane to your design ((TEST FLIGHT) without consulting you.
  - You exchange test planes, and revise your algorithm.
  - **The build team makes a copy of your revised paper airplane (FINAL FLIGHT)** and flies it from the balcony (must be behind first row of seats).
  - Will be judged on closeness to the stage.
  - Winning design/build team gets chocolate.
- Remember to pick up all your airplanes!

Design Challenge: Test Planes **(3 Minutes)**



- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - Exchange with another team.
  - They build an airplane to your design (TEST FLIGHT) without consulting you.
  - You exchange test planes, and revise your algorithm.
  - The build team makes a copy of your revised paper airplane (FINAL FLIGHT) **and flies it from the balcony (must be behind first row of seats)**.
  - Will be judged on closeness to the stage.
  - Winning design/build team gets chocolate.
- Remember to pick up all your airplanes!

Design Challenge: Retrieve Planes **(2 Minutes)**



- A classic write-an-algorithm challenge for introductory programming.
- With a slight twist: refining designs
  - As a team, write down your design.
  - Exchange with another team.
  - They build an airplane to your design (TEST FLIGHT) without consulting you.
  - You exchange test planes, and revise your algorithm.
  - The build team makes a copy of your revised paper airplane (FINAL FLIGHT) and flies it from the balcony (must be behind first row of seats).
  - Will be judged on closeness to the stage.
  - Winning design/build team gets chocolate.
- **Remember to pick up all your airplanes!**

Today's Topics

- Recap: Colors
- 2D Arrays & Image Files
- Design Challenge: Airplanes
- **Decisions**

Challenge (Group Work)

Predict what these will do (novel concepts):

```
yearBorn = int(input('Enter year born: '))
if yearBorn < 1946:
    print("Greatest Generation")
elif yearBorn <= 1964:
    print("Baby Boomer")
elif yearBorn <= 1984:
    print("Generation X")
elif yearBorn <= 2004:
    print("Millennial")
else:
    print("TBD")

x = int(input('Enter number: '))
if x % 2 == 0:
    print('Even number')
else:
    print('Odd number')
```

```
import turtle

tess = turtle.Turtle()
myWin = turtle.Screen()     #The graphics window
commands = input("Please enter a command string: ")

for ch in commands:
    #perform action indicated by the character
    if ch == 'F':           #move forward
        tess.forward(50)
    elif ch == 'L':         #turn left
        tess.left(90)
    elif ch == 'R':         #turn right
        tess.right(90)
    elif ch == '^':         #lift pen
        tess.penup()
    elif ch == 'v':         #lower pen
        tess.pendown()
    elif ch == 'B':         #go backwards
        tess.backward(50)
    elif ch == 'r':         #turn red
        tess.color("red")
    elif ch == 'g':         #turn green
        tess.color("green")
    elif ch == 'b':         #turn blue
        tess.color("blue")
    else:                   #for any other character
        print("Error: do not know the command:", c)
```

Python Tutor

```
yearBorn = int(input('Enter year born: '))
if yearBorn < 1946:
    print("Greatest Generation")
elif yearBorn <= 1964:
    print("Baby Boomer")
elif yearBorn <= 1984:
    print("Generation X")
elif yearBorn <= 2004:
    print("Millennial")
else:
    print("TBD")

x = int(input('Enter number: '))
if x % 2 == 0:
    print('Even number')
else:
    print('Odd number')
```

(Demo with pythonTutor)

IDLE

```
import turtle

tess = turtle.Turtle()
myWin = turtle.Screen()     #The graphics window
commands = input("Please enter a command string: ")

for ch in commands:
    #perform action indicated by the character
    if ch == 'F':           #move forward
        tess.forward(50)
    elif ch == 'L':         #turn left
        tess.left(90)
    elif ch == 'R':         #turn right
        tess.right(90)
    elif ch == '^':         #lift pen
        tess.penup()
    elif ch == 'v':         #lower pen
        tess.pendown()
    elif ch == 'B':         #go backwards
        tess.backward(50)
    elif ch == 'r':         #turn red
        tess.color("red")
    elif ch == 'g':         #turn green
        tess.color("green")
    elif ch == 'b':         #turn blue
        tess.color("blue")
    else:                   #for any other character
        print("Error: do not know the command:", c)
```

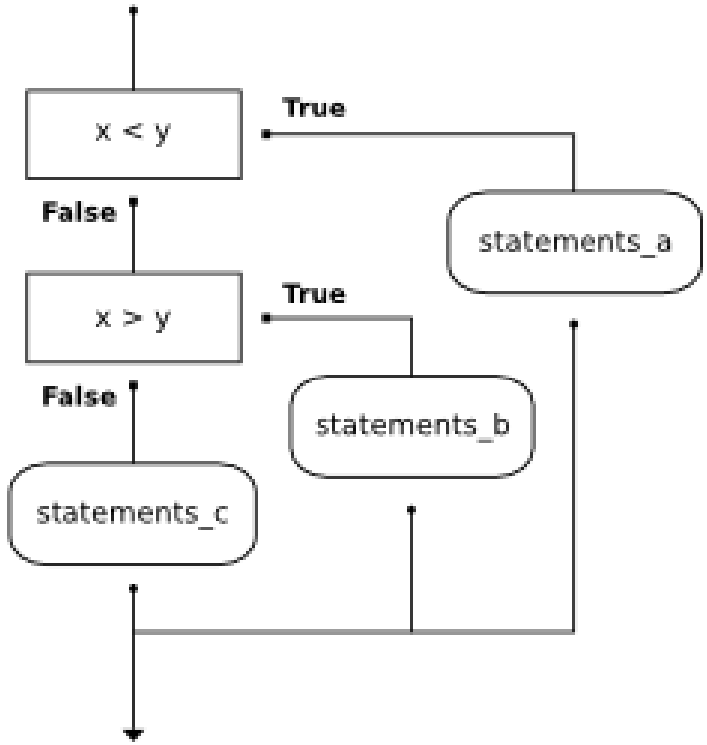
(Demo with IDLE)

Decisions

```
if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x and y must be equal")
```

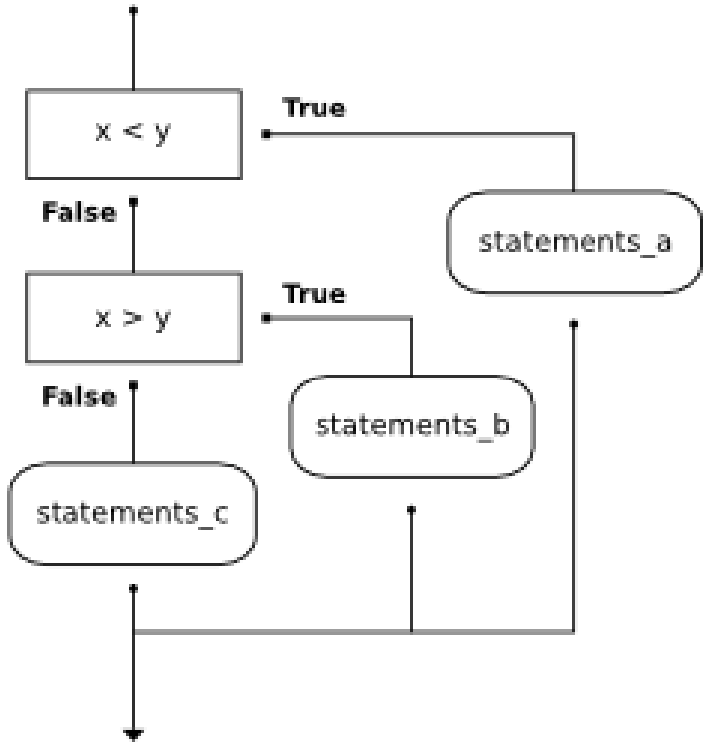
Decisions

```
if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x and y must be equal")
```



Decisions

```
if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x and y must be equal")
```



(This was just a first glance, will do much more on decisions over the next several weeks.)

Recap

- In Python, we introduced:

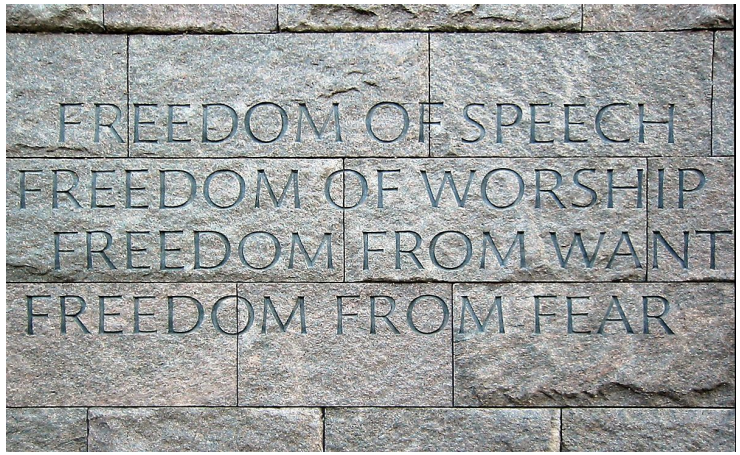
Recap

- In Python, we introduced:
  - Recap: Colors
  - 2D Array & Image Files
  - Decisions

Practice Quiz & Final Questions

(NYTimes)

(Hunter College)



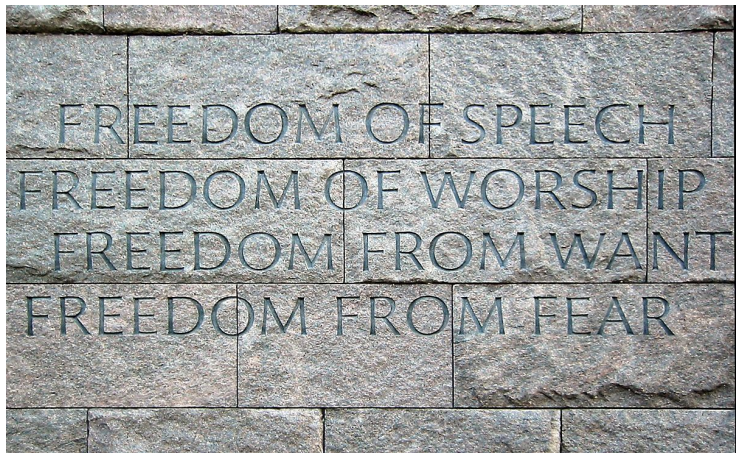
(FDR 4 FP)

- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).

Practice Quiz & Final Questions

(NYTimes)

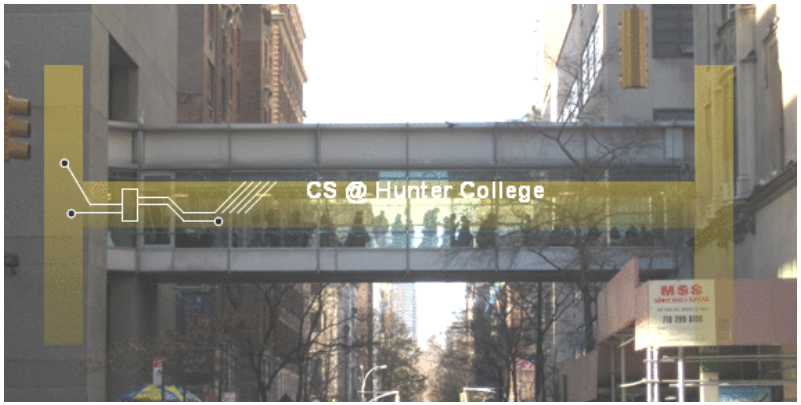
(Hunter College)



(FDR 4 FP)

- 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).
- We are starting with Fall 2019, Version 1.

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab

Weekly Reminders!

Before next lecture, don't forget to:

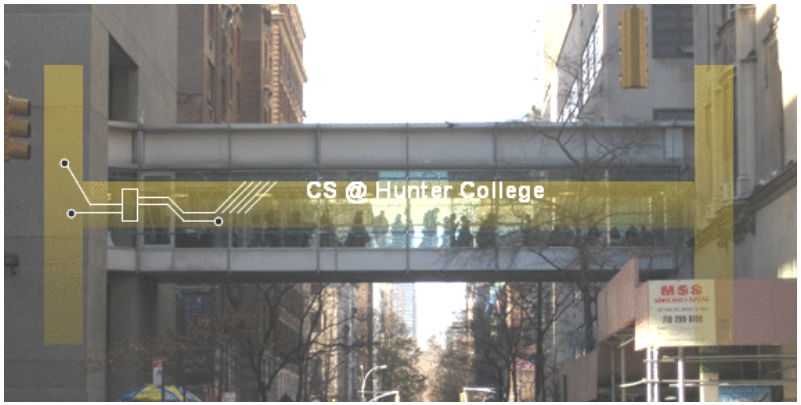
- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North

Weekly Reminders!

Before next lecture, don't forget to:

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

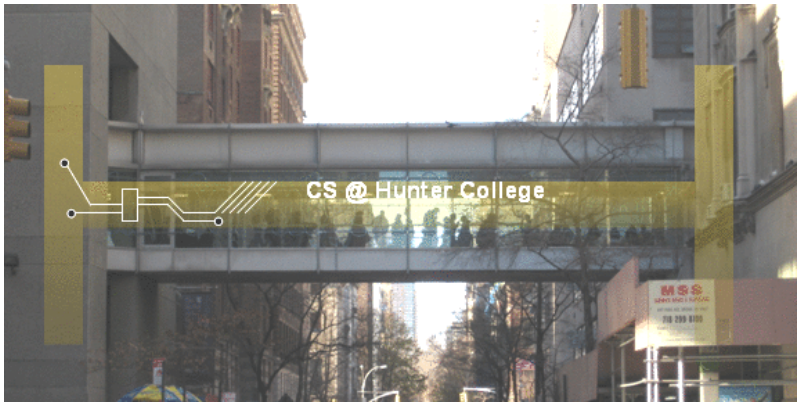
Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments **(programs 16-20)**

Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments **(programs 16-20)**
- If you need help, schedule an appointment for Tutoring in lab 1001G 11:30am-5:15pm

Weekly Reminders!

Before next lecture, don't forget to:

- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001G Hunter North
- If you haven't already, schedule an appointment to take the Code Review (**one every week**) in lab 1001G Hunter North
- Submit this week's 5 programming assignments **(programs 16-20)**
- If you need help, schedule an appointment for Tutoring in lab 1001G 11:30am-5:15pm
- Take the Lecture Preview on Blackboard on Monday (or no later than 10:15am on Tuesday)

Lecture Slips & Writing Boards

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