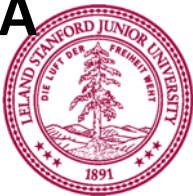
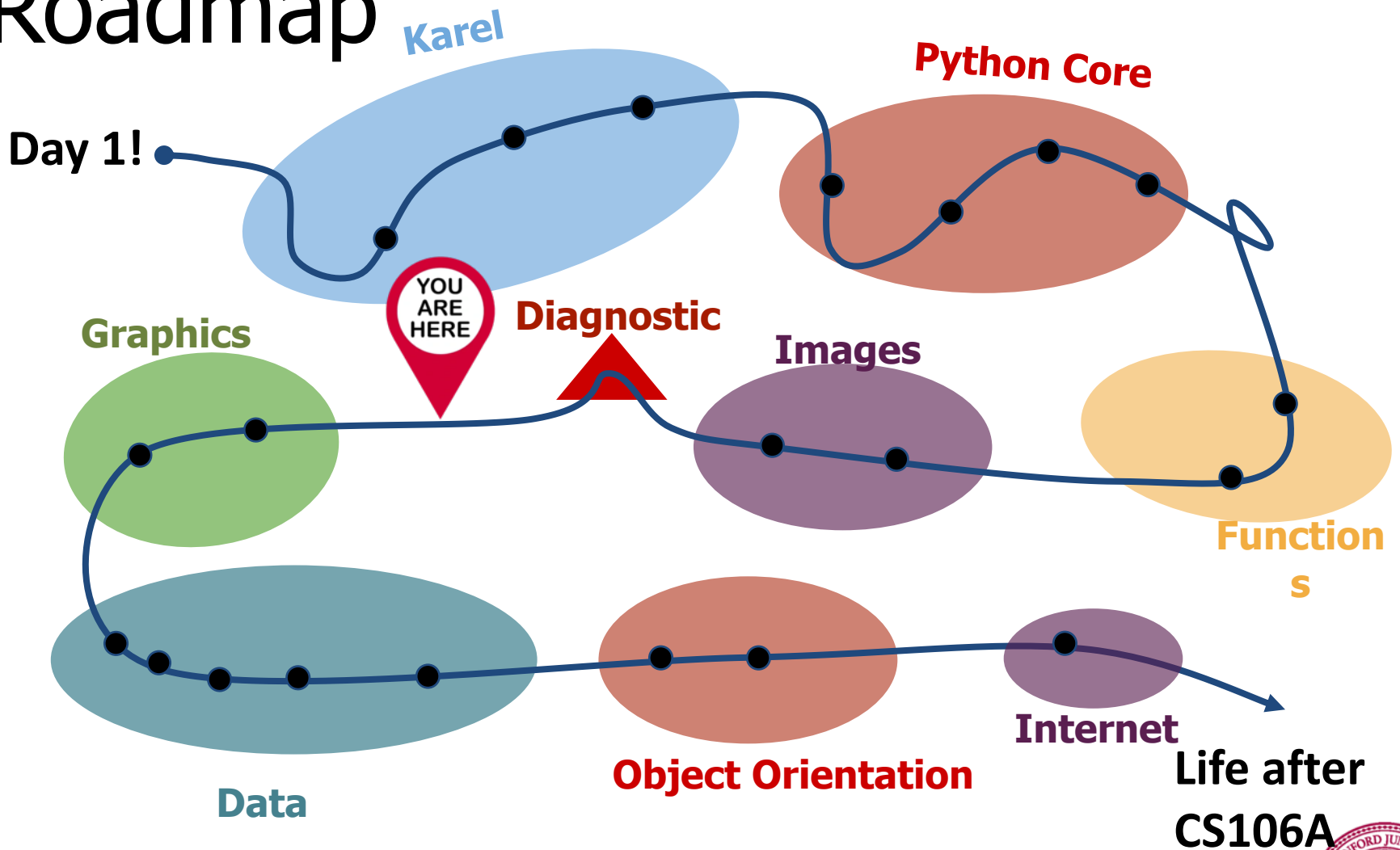




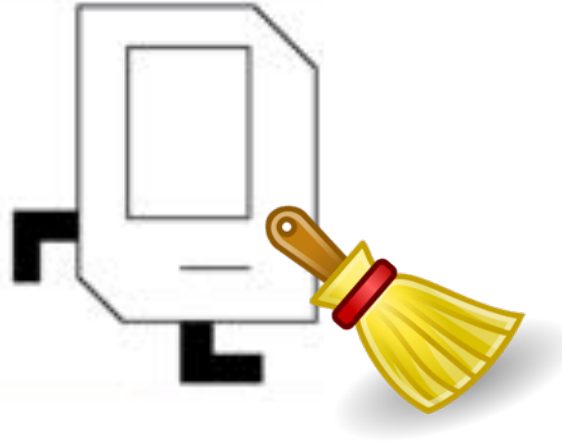
# Graphics

Chris Piech and Mehran Sahami  
CS106A, Stanford University

# Roadmap



# Housekeeping



- Give the diagnostic back at the end of lecture
- New handout on Binding vs Mutation



# New Handout!

The screenshot shows a web browser window with the URL `web.stanford.edu/class/cs106a/handouts/mutation...`. The page header includes 'CS106A' and a navigation menu with 'Lectures', 'Assignments', 'Section', and 'Handouts'. A dropdown menu is open under 'Handouts', listing various resources such as 'Karel Reader', 'Python Reader', 'General Information', 'Zoom Links', 'Course Communication', 'Course Placement', 'Honor Code', 'Installing PyCharm', 'Using Karel in PyCharm', 'Submitting Assignments', 'First Diagnostic Info', 'Image Reference', and 'Binding vs Mutation' (which is highlighted).

## Binding vs Mutation

OCTOBER 9TH, 2020

**New handout!** Hi all, this is a brand new handout for CS106A. Some students often find difficult. If you find something on Ed. You will help us make the handout better!

In this handout we are going to go on a little journey to understand how Python has three phases. **Part I:** We will start with a few observations of how Python works. **Part II:** We will formalize the pattern to go from code to a more abstract representation. **Part III:** We will take this chance to learn the details of how Python works under the hood.

**Erratum, Oct 9th:** based on a great question on Ed we updated the wording on The Re-Binding Rule and added examples to Binding in parameters and Binding in for each loops.

### Part I: Motivating examples!

First-time programmers bump into a need for a deeper understanding of what is actually going on under the hood when observing that in some **for loops**, and in some **helper functions**, changes to variables sometimes persist and sometimes do not! Lets first look at a few programs to build up this mystery:

Can you predict what each example prints? If not than read on to get your deep understanding.

**Example 1:** What will this print?

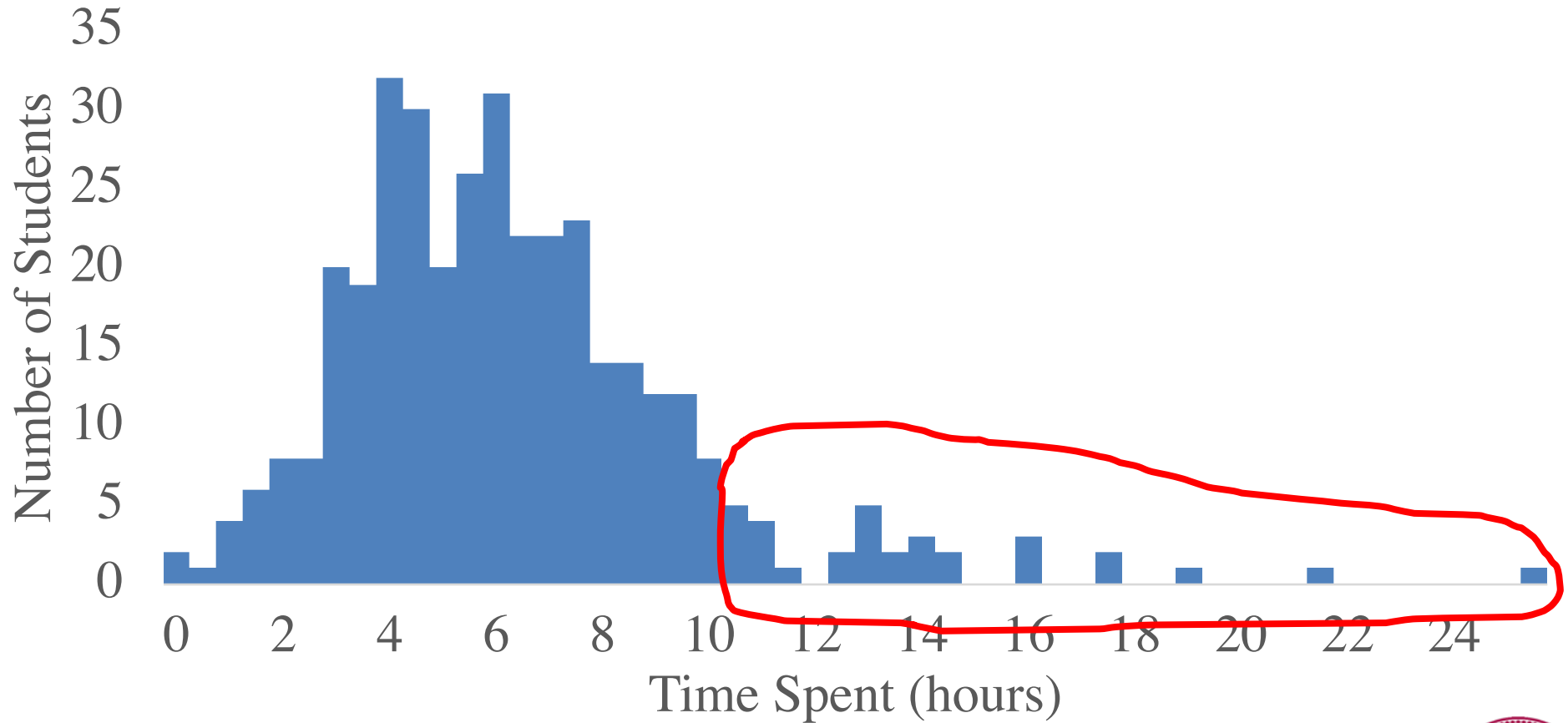
```
def main():  
    x = [1, 2, 3]
```



Get a little meta...


# Learn by Doing

## Assignment 2



# Learning to Program on the Internet

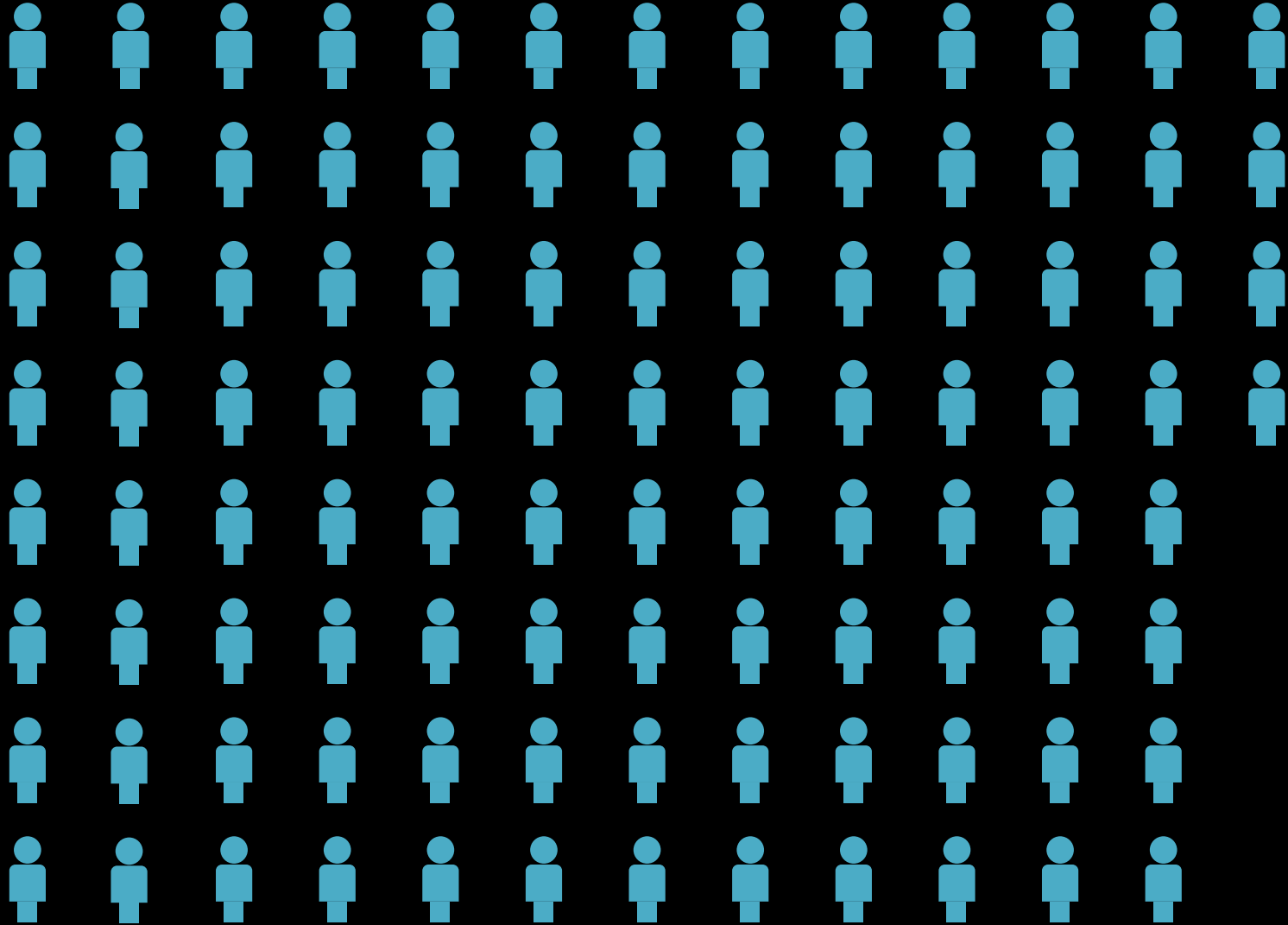


```
when run
  repeat until 
  do
    if path ahead ▼
    do
      move forward
    else
      turn left ↻ ▼
```

Task

Almost a hundred thousand  
unique solutions

# US K-12 Students

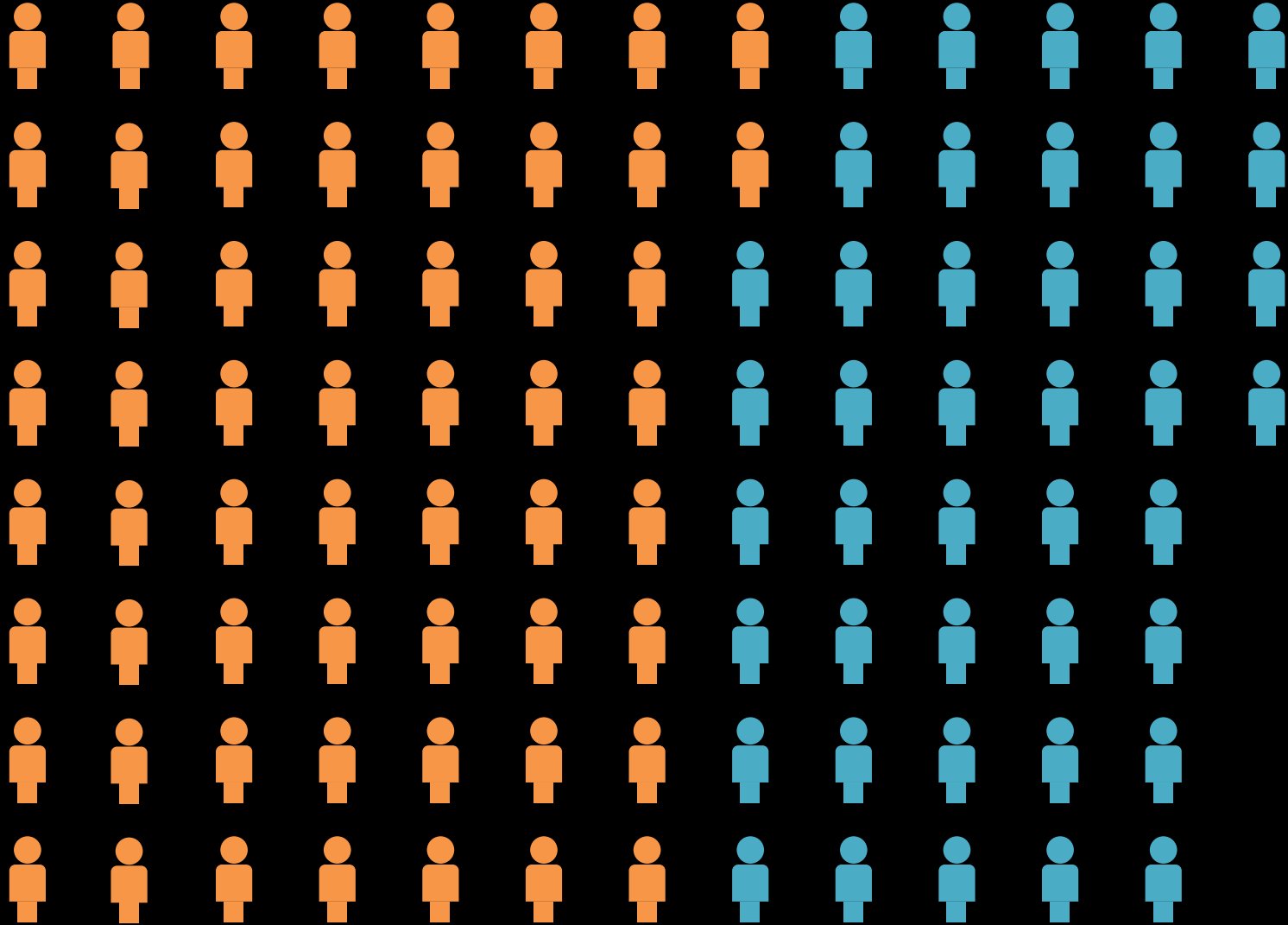


= 500,000 learners

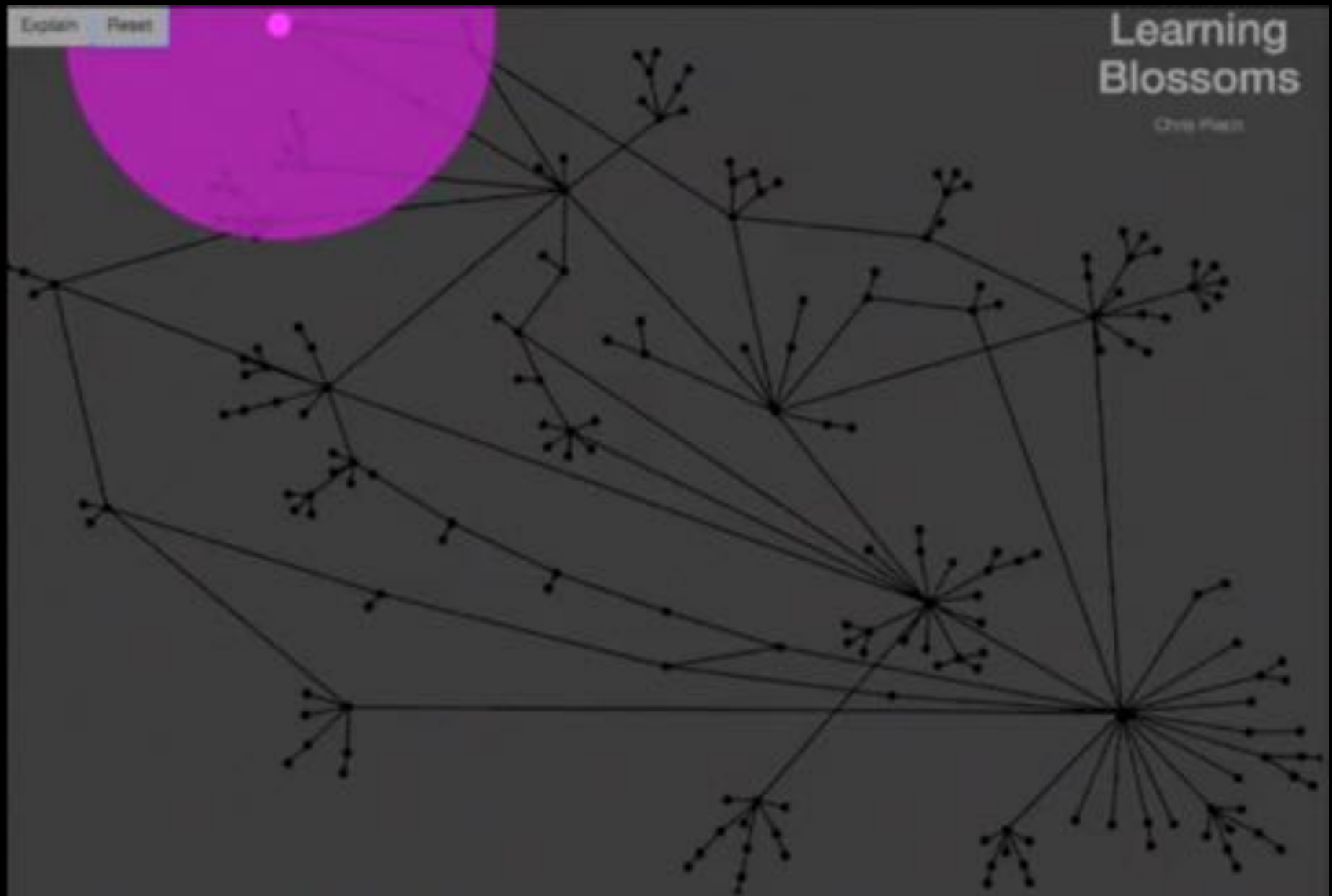




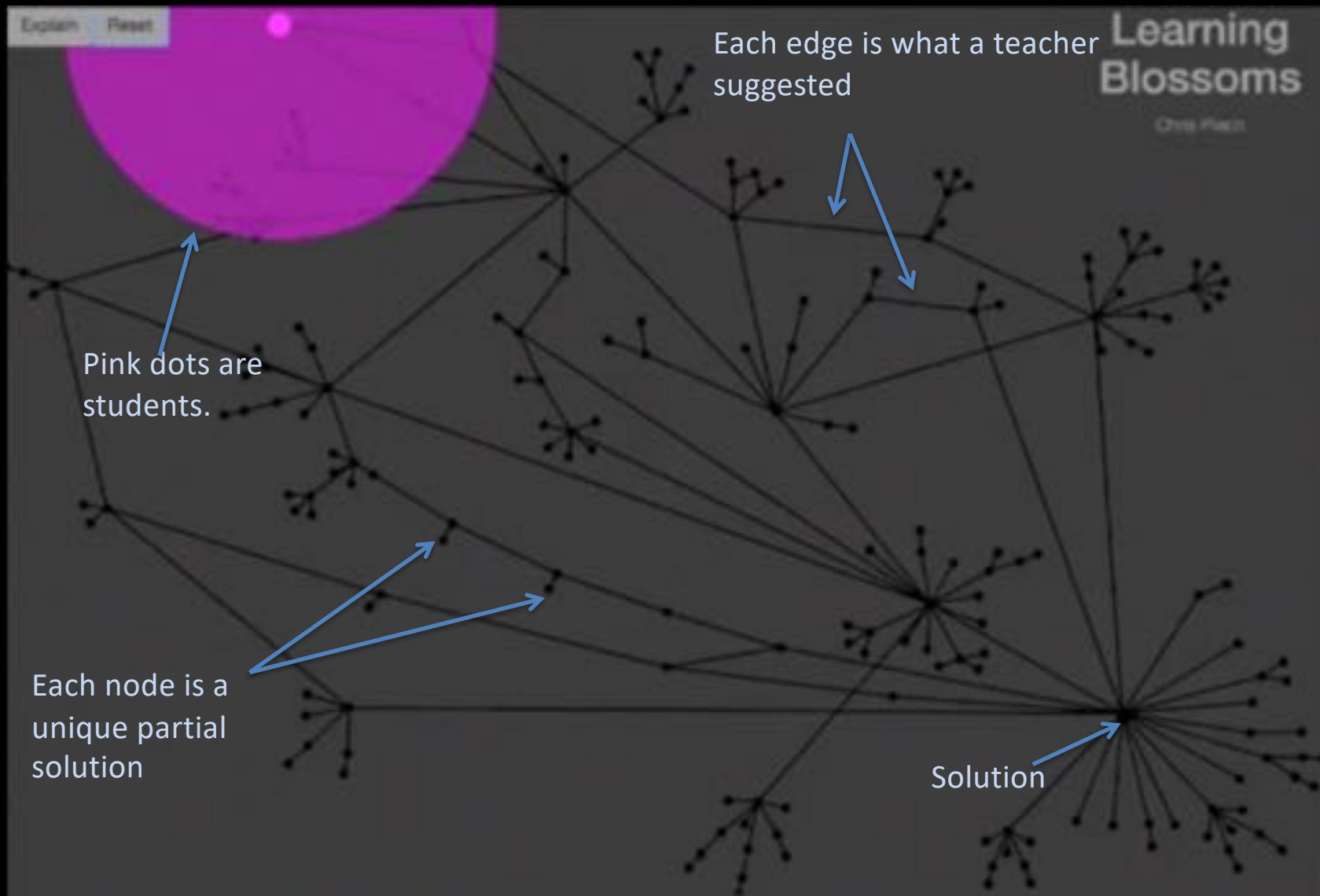
# Code.org Students

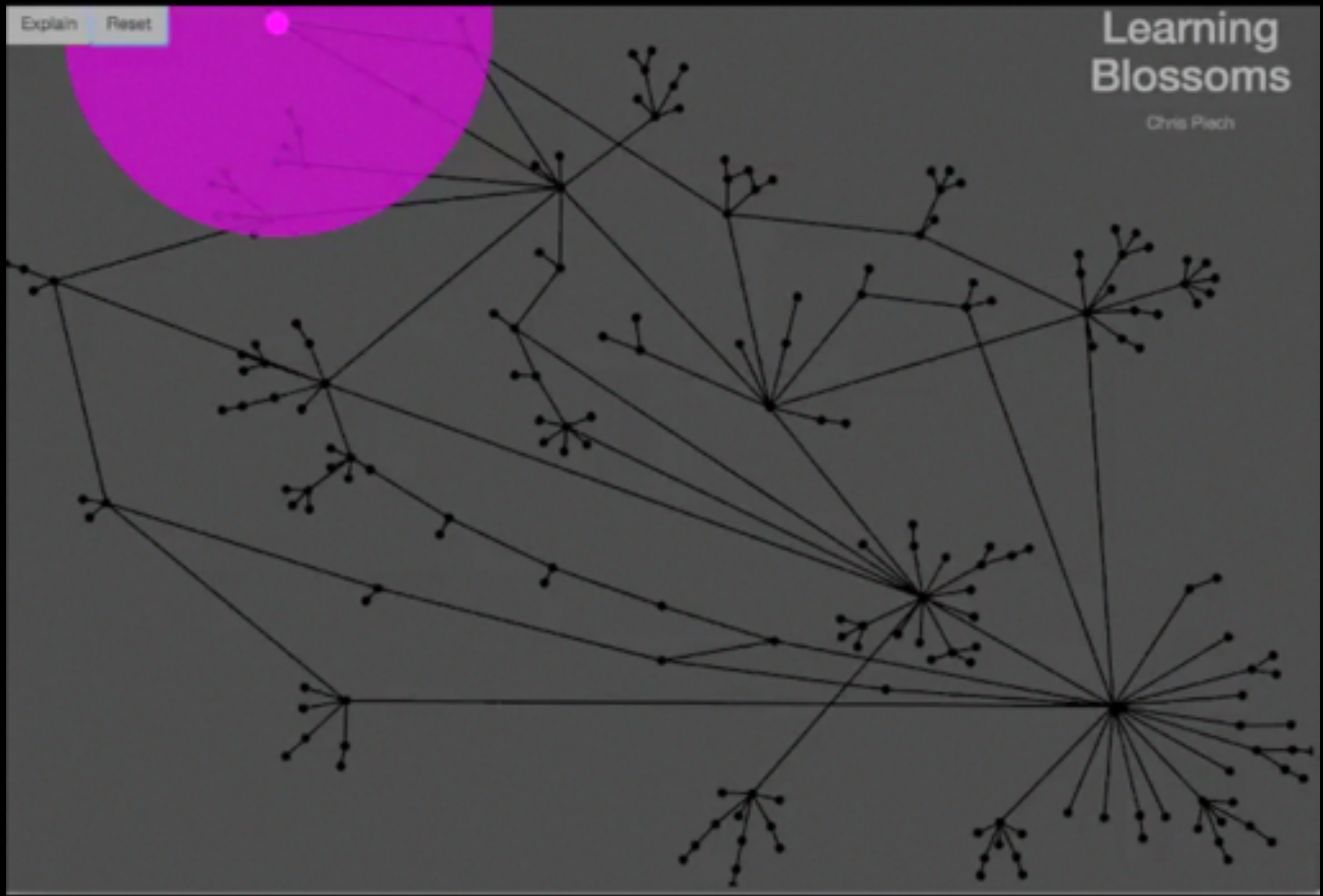


= 500,000 learners



Autonomously Generating Hints by Inferring Problem Solving Policies - Piech, Sahami et al.

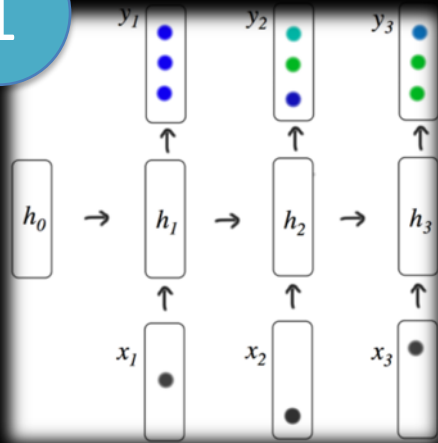




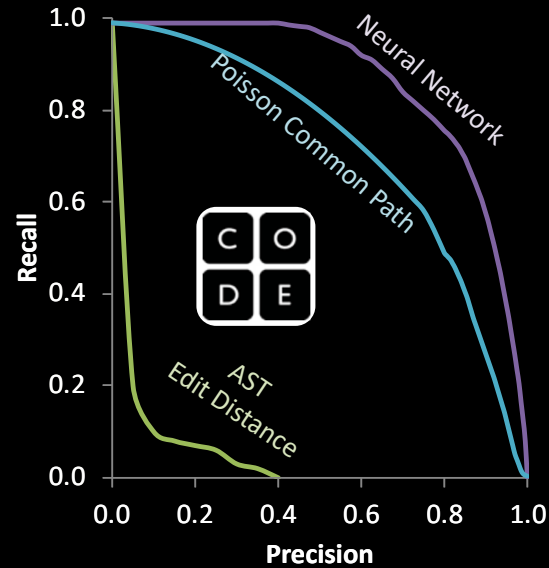
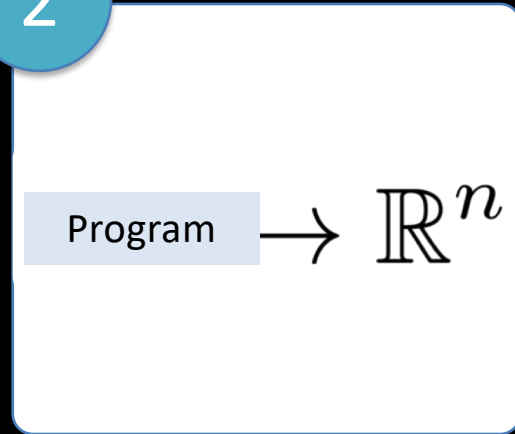
Autonomously Generating Hints by Inferring Problem Solving Policies - Piech, Sahami et al.

# Deep Learning Algorithms

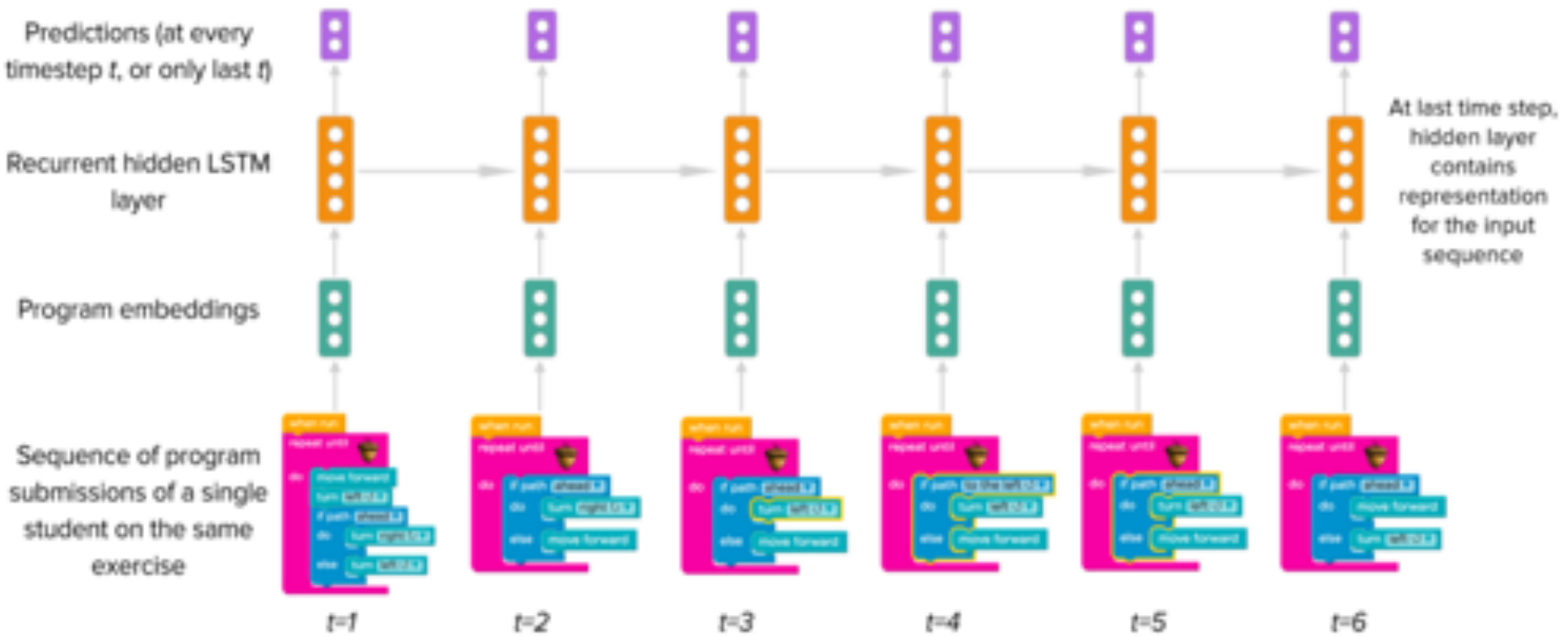
1



2



# Deep Learning on Trajectories

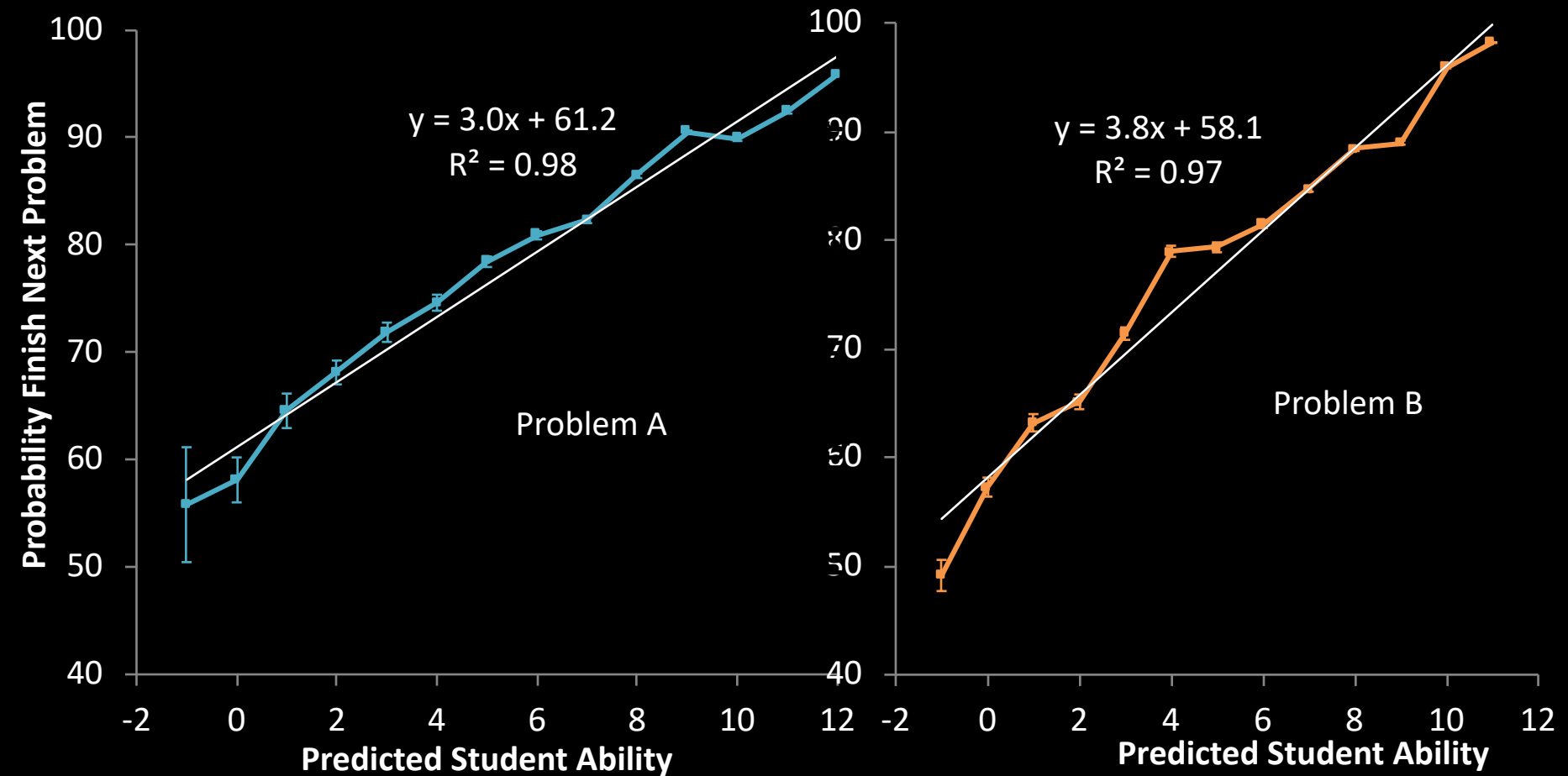


Research in collaboration with Lisa Wang

Piech + Sahami, CS106A, Stanford University



# Predicts Future Success



Most likely to succeed?



# Highly Rates Grit

1. Two compound errors

```
when run
repeat until [acorn]
do
  move forward
  turn left
  if path ahead
  do turn right
  else turn left
```

2. Solves first error

```
when run
repeat until [acorn]
do
  if path ahead
  do turn right
  else move forward
```

3. Starts reasonable attempt

```
when run
repeat until [acorn]
do
  if path ahead
  do turn left
  else move forward
```

4. Completes attempt

```
when run
repeat until [acorn]
do
  if path to the left
  do turn left
  else move forward
```

5. Backtracks

```
when run
repeat until [acorn]
do
  if path ahead
  do turn left
  else move forward
```

6. Finds solution

```
when run
repeat until [acorn]
do
  if path ahead
  do move forward
  else turn left
```



# Highly Rates Grit

1. Two compound errors

```
when run
repeat until acorn
do
  move forward
  turn left 90
  if path ahead
  do turn right 90
  else turn left 90
```

2. Solves first error

```
when run
repeat until acorn
do
  if path ahead
  do turn right 90
  else move forward
```

3. Starts reasonable attempt

```
when run
repeat until acorn
do
  if path ahead
  do turn left 90
  else move forward
```

4. Completes attempt

```
when run
repeat until acorn
do
  if path to the left
  do turn left 90
  else move forward
```

5. Backtracks

```
when run
repeat until acorn
do
  if path ahead
  do turn left 90
  else move forward
```

6. Finds solution

```
when run
repeat until acorn
do
  if path ahead
  do move forward
  else turn left 90
```



# Highly Rates Grit

1. Two compound errors

```
when run
repeat until [acorn]
do
  move forward
  turn left 90
  if path ahead
  do turn right 90
  else turn left 90
```

2. Solves first error

```
when run
repeat until [acorn]
do
  if path ahead
  do turn right 90
  else move forward
```

3. Starts reasonable attempt

```
when run
repeat until [acorn]
do
  if path ahead
  do turn left 90
  else move forward
```

4. Completes attempt

```
when run
repeat until [acorn]
do
  if path to the left
  do turn left 90
  else move forward
```

5. Backtracks

```
when run
repeat until [acorn]
do
  if path ahead
  do turn left 90
  else move forward
```

6. Finds solution

```
when run
repeat until [acorn]
do
  if path ahead
  do move forward
  else turn left 90
```



# Highly Rates Grit

1. Two compound errors

```
when run  
repeat until (acorn icon)  
do  
  move forward  
  turn left  
  if path ahead  
  do  
    turn right  
  else  
    turn left
```

2. Solves first error

```
when run  
repeat until (acorn icon)  
do  
  if path ahead  
  do  
    turn right  
  else  
    move forward
```

3. Starts reasonable attempt

```
when run  
repeat until (acorn icon)  
do  
  if path ahead  
  do  
    turn left  
  else  
    move forward
```

4. Completes attempt

```
when run  
repeat until (acorn icon)  
do  
  if path to the left  
  do  
    turn left  
  else  
    move forward
```

5. Backtracks

```
when run  
repeat until (acorn icon)  
do  
  if path ahead  
  do  
    turn left  
  else  
    move forward
```

6. Finds solution

```
when run  
repeat until (acorn icon)  
do  
  if path ahead  
  do  
    move forward  
  else  
    turn left
```



# Highly Rates Grit

1. Two compound errors

```
when run
repeat until [acorn]
do
  move forward
  turn left 90
  if path ahead
  do turn right 90
  else turn left 90
```

2. Solves first error

```
when run
repeat until [acorn]
do
  if path ahead
  do turn right 90
  else move forward
```

3. Starts reasonable attempt

```
when run
repeat until [acorn]
do
  if path ahead
  do turn left 90
  else move forward
```

4. Completes attempt

```
when run
repeat until [acorn]
do
  if path to the left
  do turn left 90
  else move forward
```

5. Backtracks

```
when run
repeat until [acorn]
do
  if path ahead
  do turn left 90
  else move forward
```

6. Finds solution

```
when run
repeat until [acorn]
do
  if path ahead
  do move forward
  else turn left 90
```



Review







Image processing - How is a ... x +

stackoverflow.com/questions/1061093/how-is-a-sepia-tone-created

stackoverflow Products Search... Log in Sign up

Our community has been nominated for a Webby Award for Best Community Website - thank you! Show the love and [vote here](#).

## How is a sepia tone created?

Asked 10 years, 10 months ago Active 7 years, 4 months ago Viewed 28k times

11

image-processing imagemagick

asked Jun 29 '09 at 23:37  
user83358  
854 +3 +10 +17

4 Answers

24

```
outputRed = (inputRed * .393) + (inputGreen * .769) + (inputBlue * .189)
outputGreen = (inputRed * .349) + (inputGreen * .686) + (inputBlue * .168)
outputBlue = (inputRed * .272) + (inputGreen * .534) + (inputBlue * .131)
```

if any of these output values is greater than 255, you simply set it to 255. These specific values are the values for sepia tone that are recommended by Microsoft.

answered Feb 25 '12 at 23:43  
Max Galkin  
15.9k +0 +58 +108

You will need to use Math.Min likely. I tried doing the check for 255 after those three lines and an error will occur. I was facing the same problem earlier today when I was trying to make a sepia tone for my program... - BigBug Feb 26 '12 at 6:54

But what if I want something different to change the filter then how can I get to these values? like my question is how we came to know about these values, do we need to just put different values again and again? - AHF Mar 23 '14 at 15:20

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).



# Sepia Example

```
def main():  
    image_name = input('enter an image name: ')  
    image = SimpleImage('images/' + image_name)  
    for pixel in image:  
        sepia_pixel(pixel)  
    image.show()
```

If you are unsure why this actually changes the image, check out binding vs mutation!

```
def sepia_pixel(pixel):  
    R = pixel.red  
    G = pixel.green  
    B = pixel.blue  
    pixel.red = 0.393 * R + 0.769 * G + 0.189 * B  
    pixel.green = 0.349 * R + 0.686 * G + 0.168 * B  
    pixel.blue = 0.272 * R + 0.534 * G + 0.131 * B
```



# Sepia Example

```
def main():
    image_name = input('enter an image name: ')
    image = SimpleImage('images/' + image_name)
    for y in range(image.height):
        for x in range(image.width):
            pixel = image.get_pixel(x, y)
            sepia_pixel(pixel)
    image.show()

def sepia_pixel(pixel):
    R = pixel.red
    G = pixel.green
    B = pixel.blue
    pixel.red = 0.393 * R + 0.769 * G + 0.189 * B
    pixel.green = 0.349 * R + 0.686 * G + 0.168 * B
    pixel.blue = 0.272 * R + 0.534 * G + 0.131 * B
```



# Sepia Example

```
def main():
    image_name = input('enter an image name: ')
    image = SimpleImage('images/' + image_name)
    for y in range(image.height):
        for x in range(image.width):
            pixel = image.get_pixel(x, y)
            sepia_pixel(pixel)
    image.show()

def sepia_pixel(pixel):
    R = pixel.red
    G = pixel.green
    B = pixel.blue
    pixel.red = 0.393 * R + 0.769 * G + 0.189 * B
    pixel.green = 0.349 * R + 0.686 * G + 0.168 * B
    pixel.blue = 0.272 * R + 0.534 * G + 0.131 * B
```



# Sepia Example

```
def main():  
  
    for y in range(600):  
        for x in range(800):  
            print(x, y)
```



# Sepia Example

```
def main():  
  
    for y in range(600):  
        for x in range(800):  
            print(x, y)
```



# Sepia Example

```
def main():  
  
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```



# Sepia Example

```
def main():
```

```
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```



Memory “stack”

main

row

0

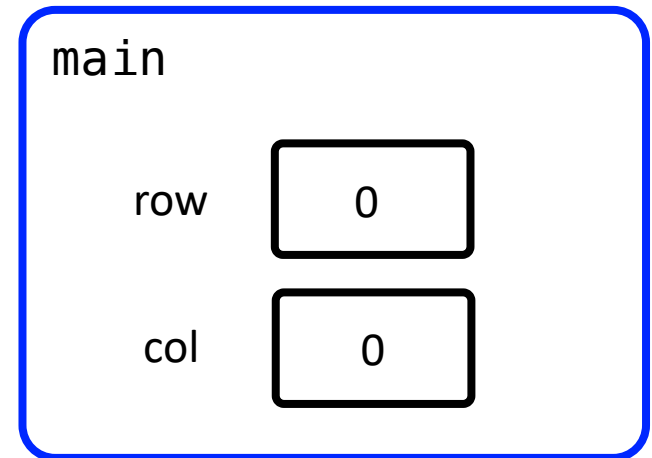


# Sepia Example

```
def main():  
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```

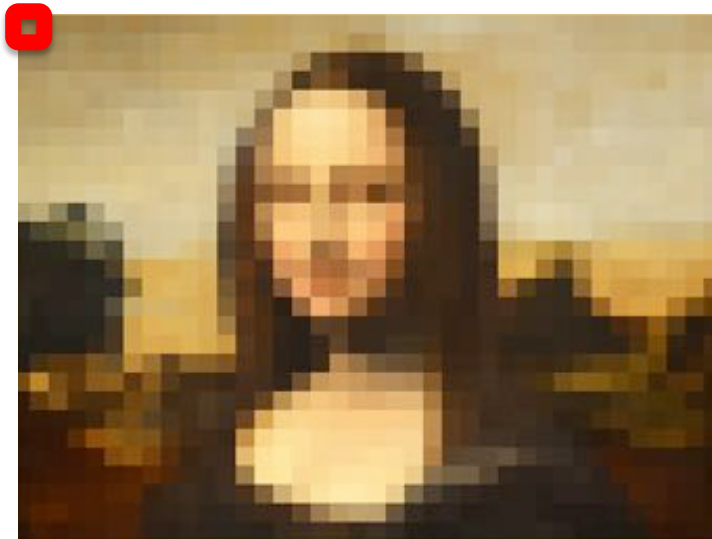


Memory “stack”

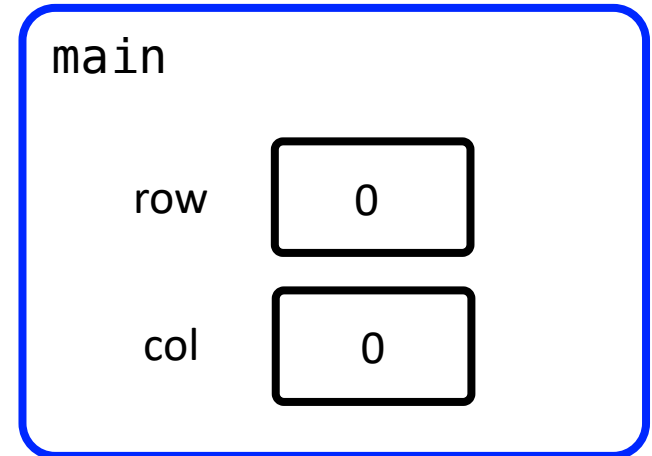


# Sepia Example

```
def main():  
  
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```

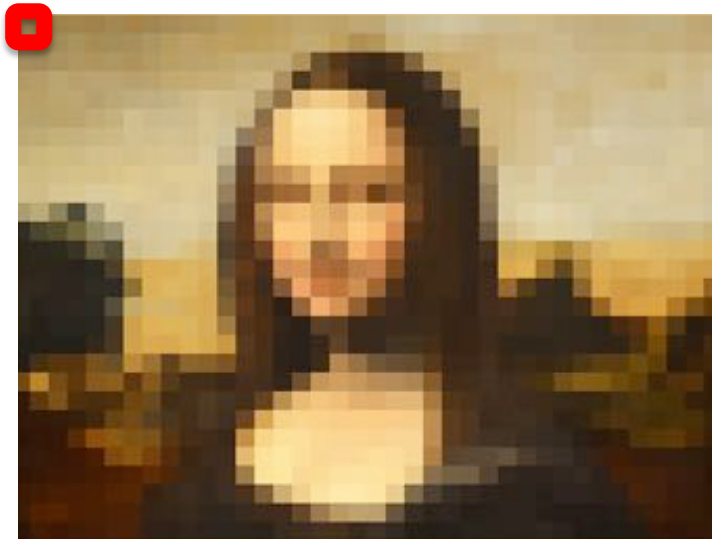


Memory “stack”

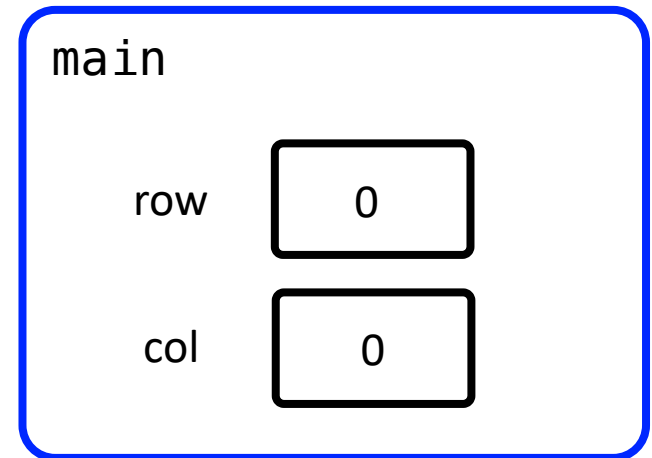


# Sepia Example

```
def main():  
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```

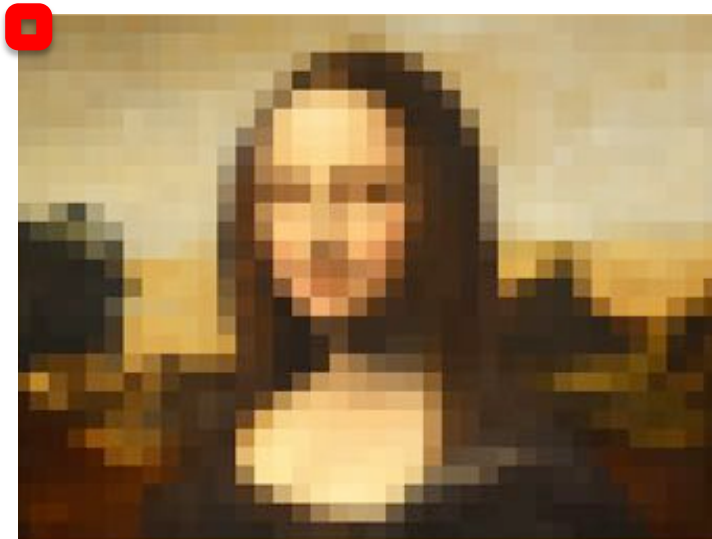


Memory “stack”

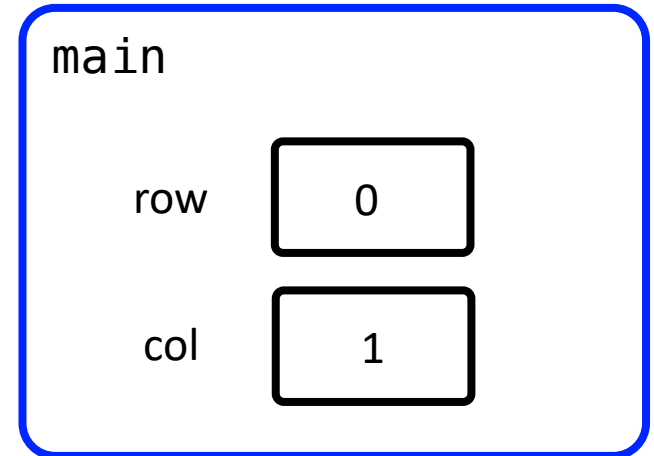


# Sepia Example

```
def main():  
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```

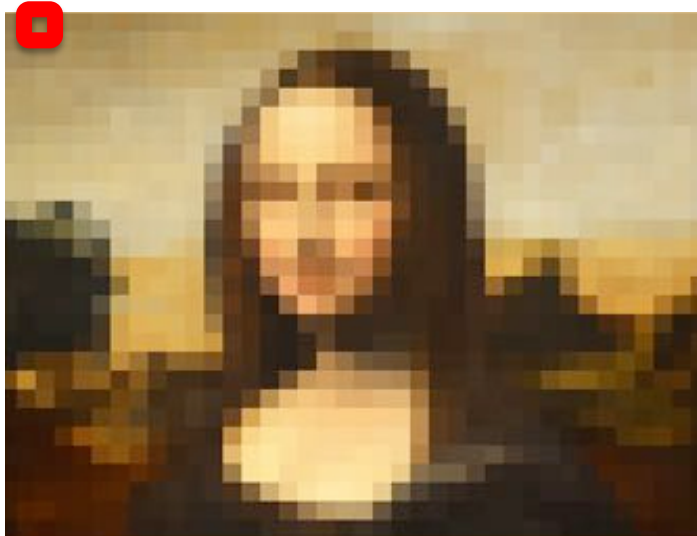


Memory “stack”

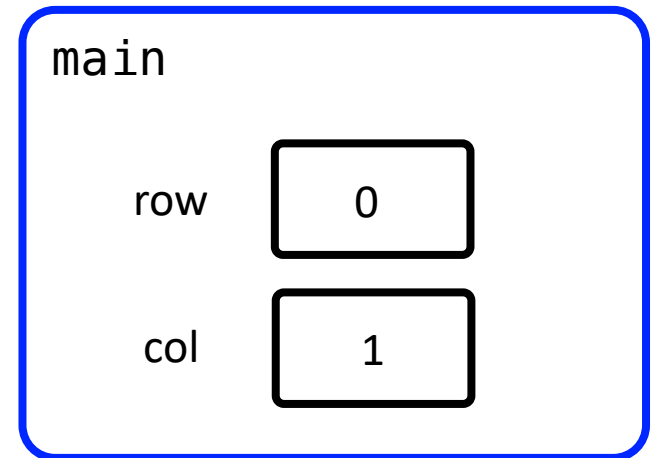


# Sepia Example

```
def main():  
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```

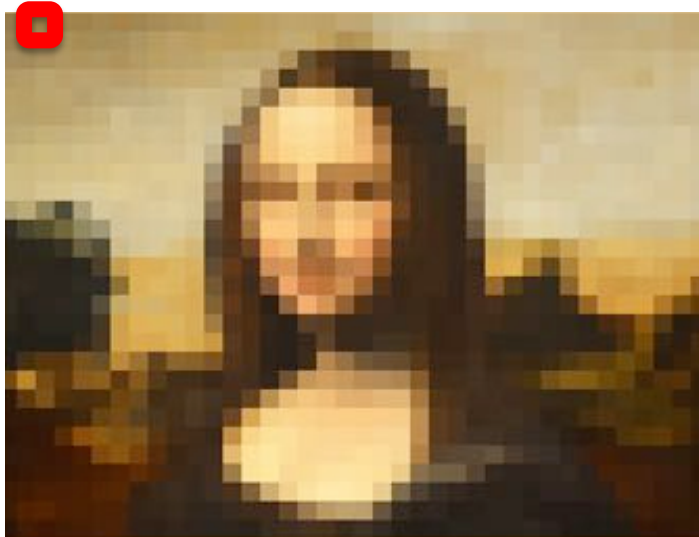


Memory “stack”

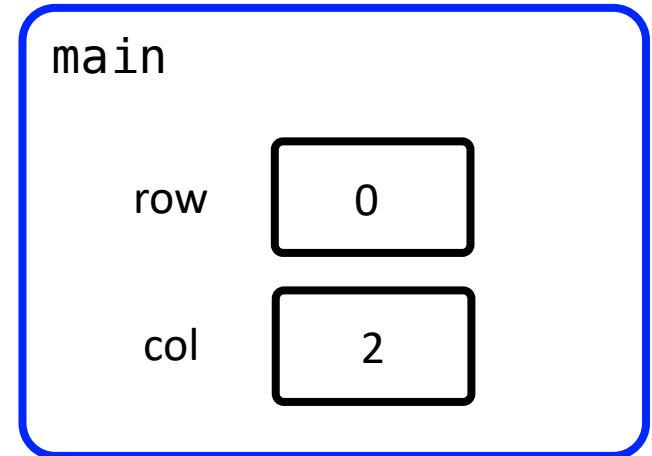


# Sepia Example

```
def main():  
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```

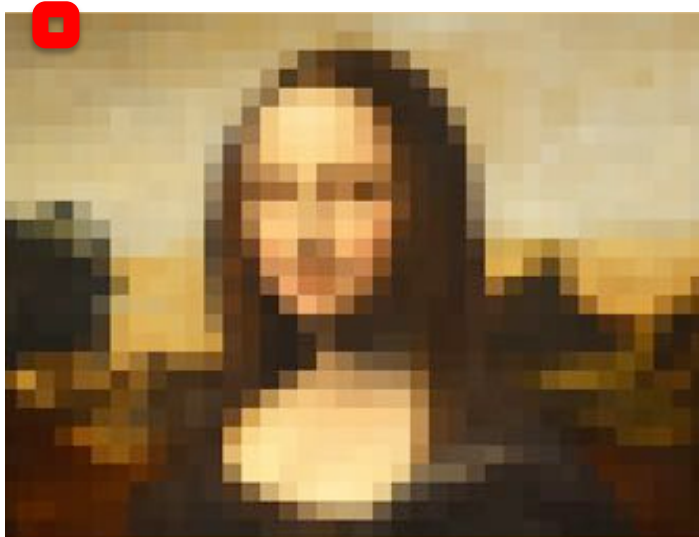


Memory “stack”

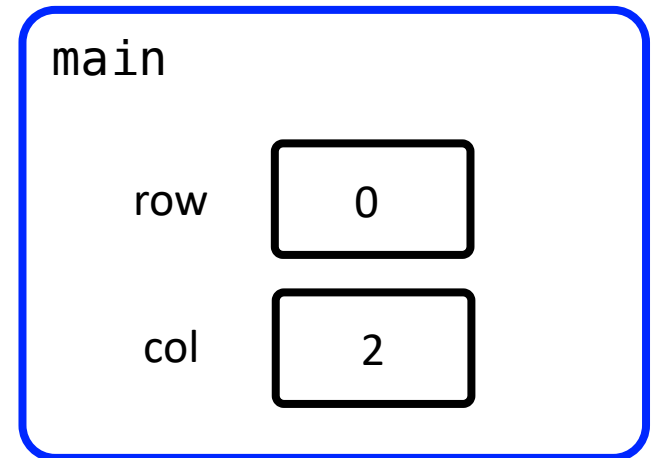


# Sepia Example

```
def main():  
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```



Memory “stack”

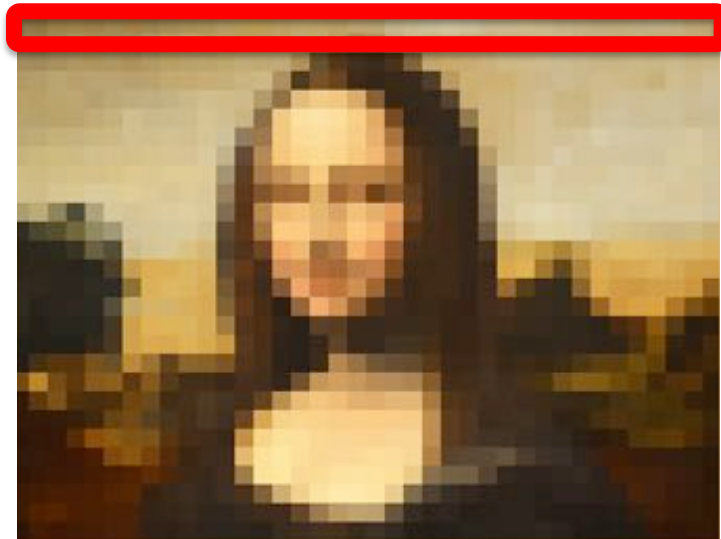


# Sepia Example

```
def main():
```

```
    for row in range(600):
```

```
        for col in range(800):  
            print(row, col)
```



Memory “stack”

main

row

0

col

*all*



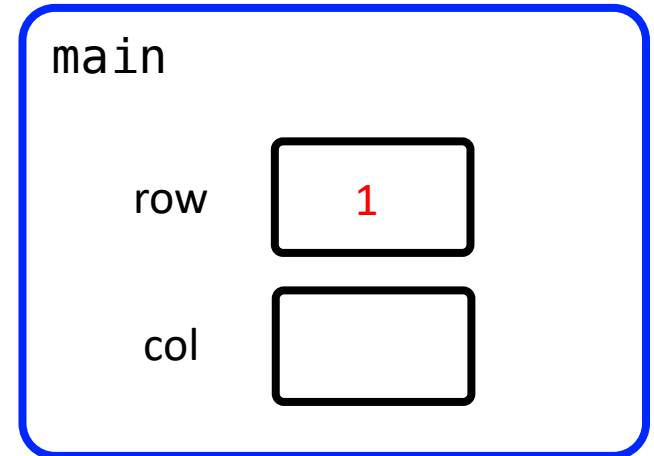
# Sepia Example

```
def main():
```

```
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```



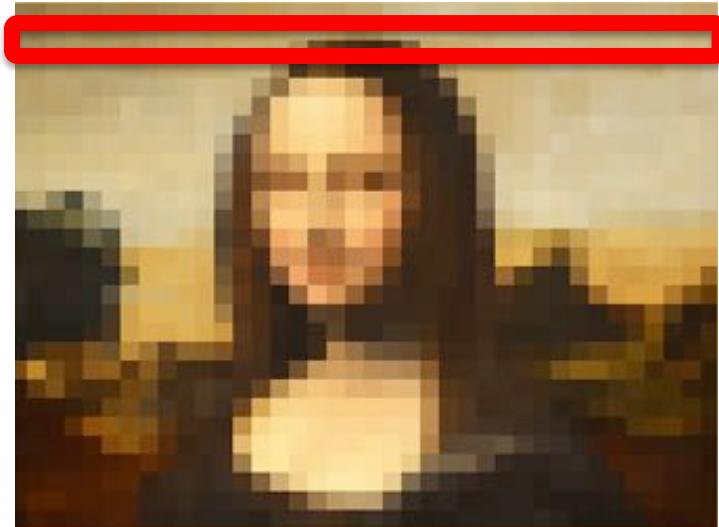
Memory “stack”



# Sepia Example

```
def main():
```

```
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```



Memory “stack”

main

row

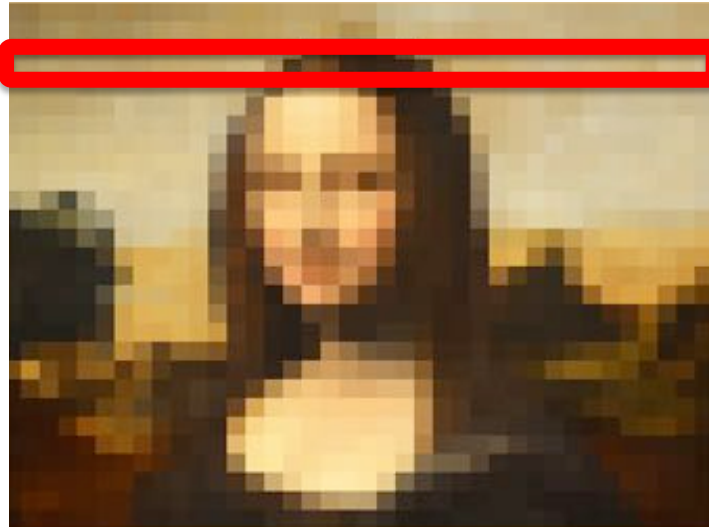
1

col

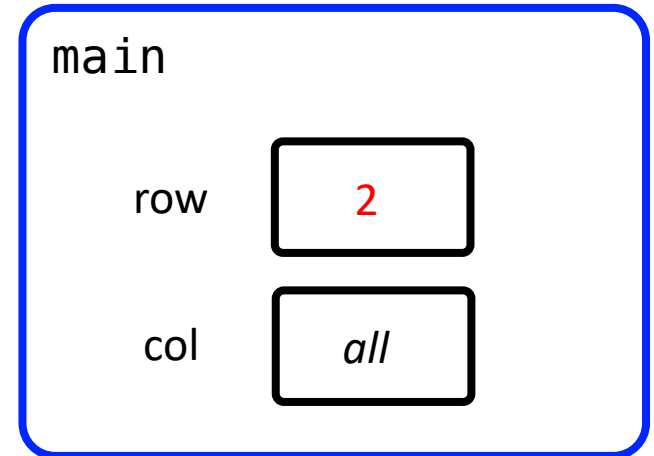
*all*

# Sepia Example

```
def main():  
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```



Memory “stack”



# Sepia Example

```
def main():
```

```
    for row in range(600):  
        for col in range(800):  
            print(row, col)
```



Memory “stack”

main

row

3

col

*all*

# Python Console

```
Python Console x
Python Console
>>> for r in range(5):
...     for c in range(5):
...         print(r, c)
...
0 0
0 1
0 2
0 3
0 4
1 0
1 1
1 2
1 3
1 4
2 0
2 1
2 2
2 3
2 4
3 0
3 1
3 2
3 3
3 4
4 0
4 1
4 2
4 3
4 4
>>>
```



**Mike Krieger**

**Kevin System**



End Review

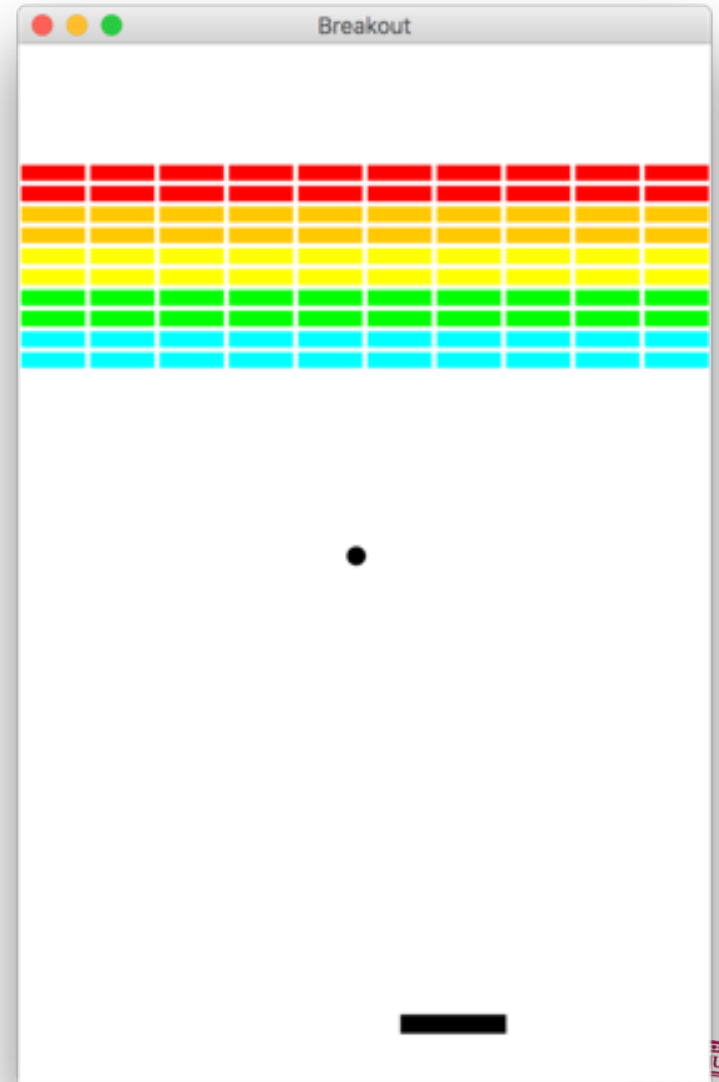
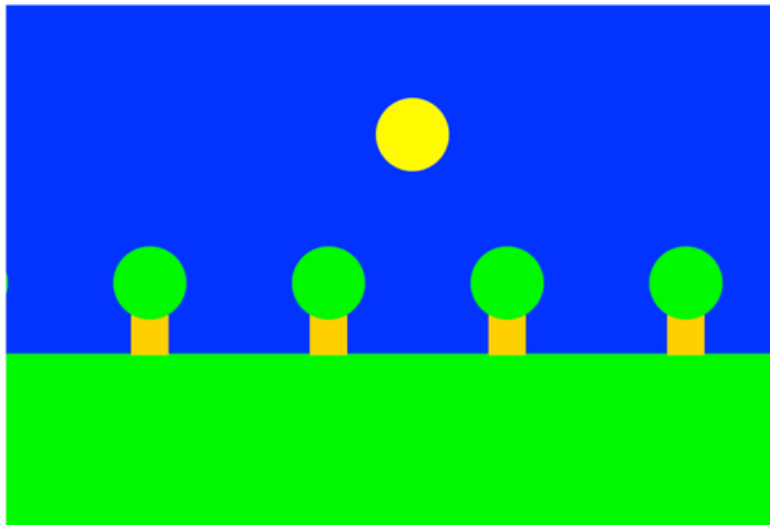
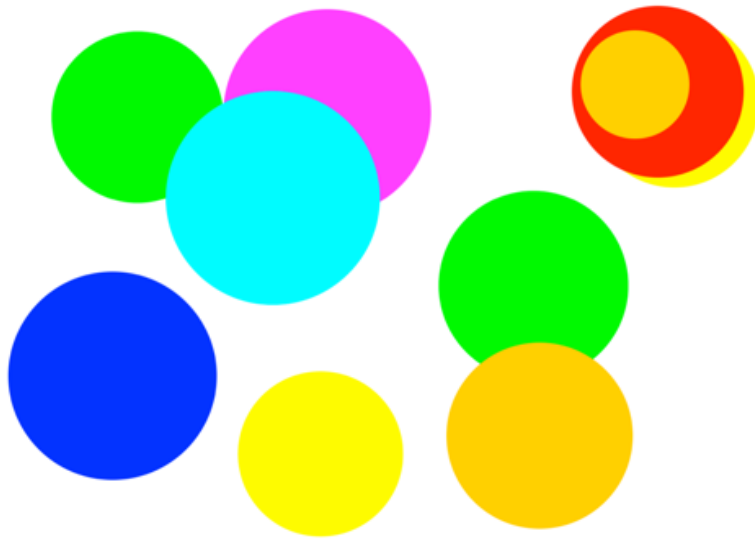
# Today's Goal

1. How do I draw shapes?





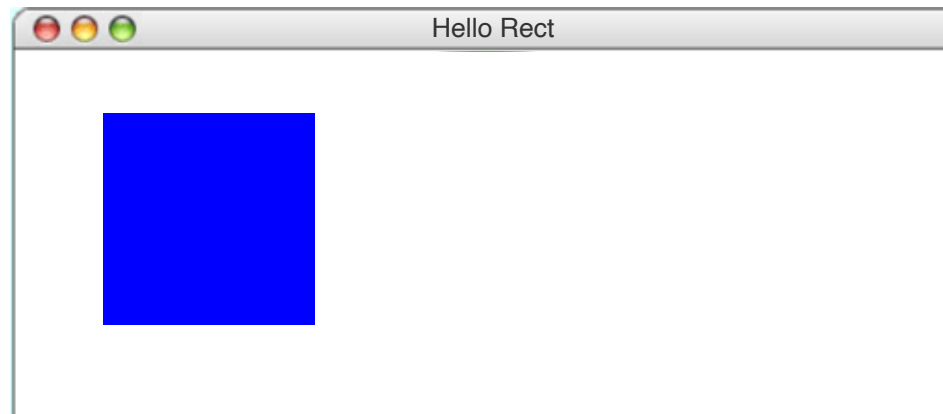
# Graphics Programs



# Draw a Rectangle

the following `main` method displays a blue square

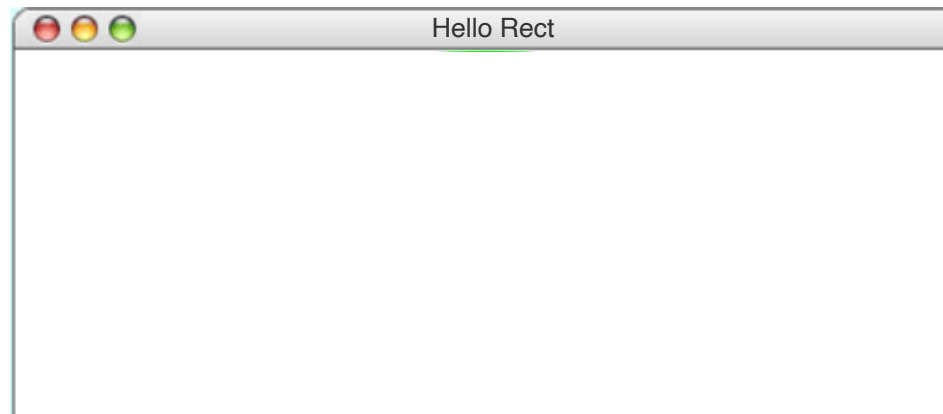
```
def main():  
    canvas = make_canvas(800, 200, 'Hello Rect')  
    canvas.create_rectangle(20, 20, 100, 100, fill="blue")  
    canvas.mainloop()
```



# Draw a Rectangle

the following `main` method displays a blue square

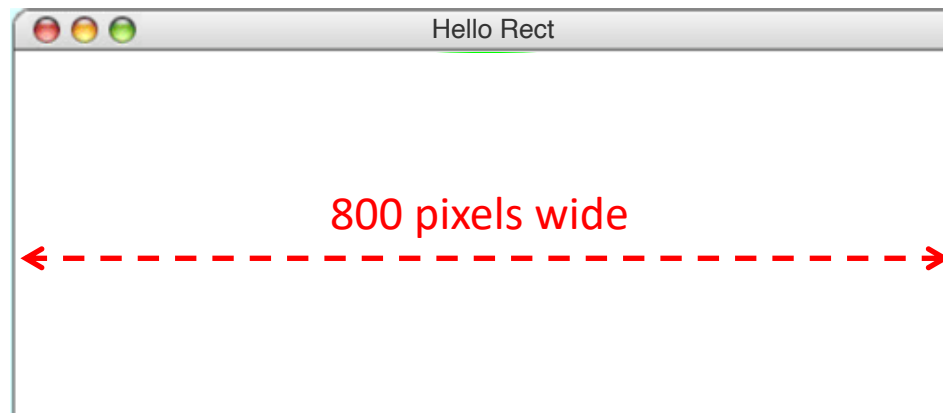
```
def main():  
    canvas = make_canvas(800, 200, 'Hello Rect')
```



# Draw a Rectangle

the following `main` method displays a blue square

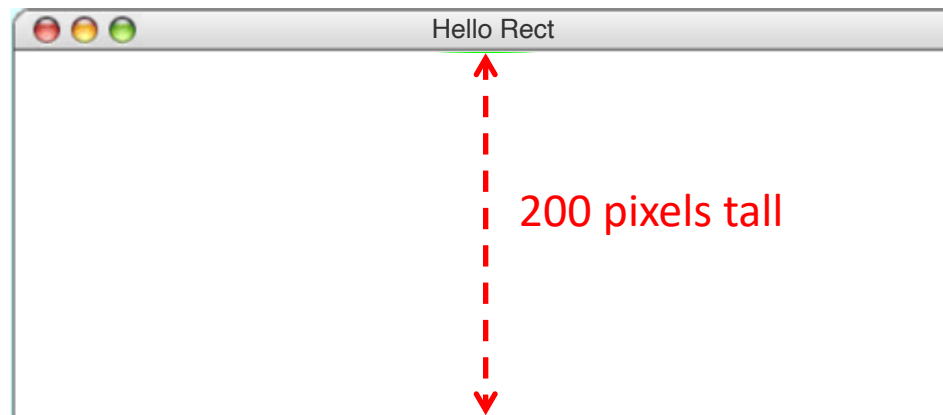
```
def main():  
    canvas = make_canvas(800, 200, 'Hello Rect')
```



# Draw a Rectangle

the following `main` method displays a blue square

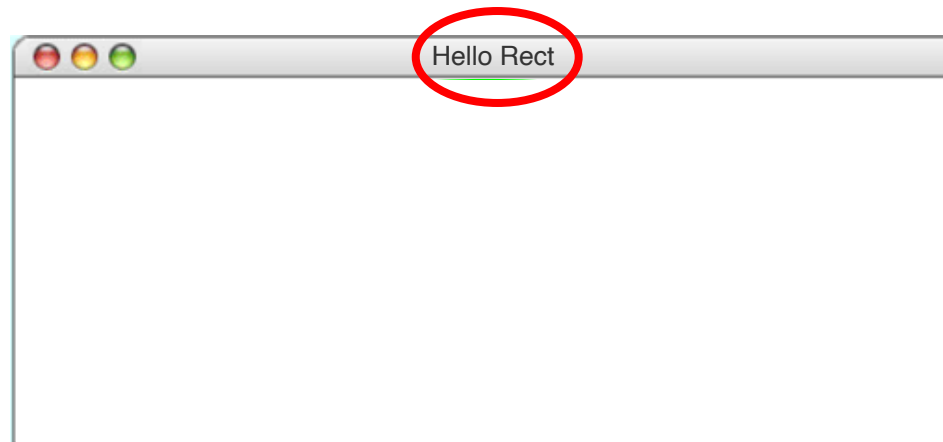
```
def main():  
    canvas = make_canvas(800, 200, 'Hello Rect')
```



# Draw a Rectangle

the following `main` method displays a blue square

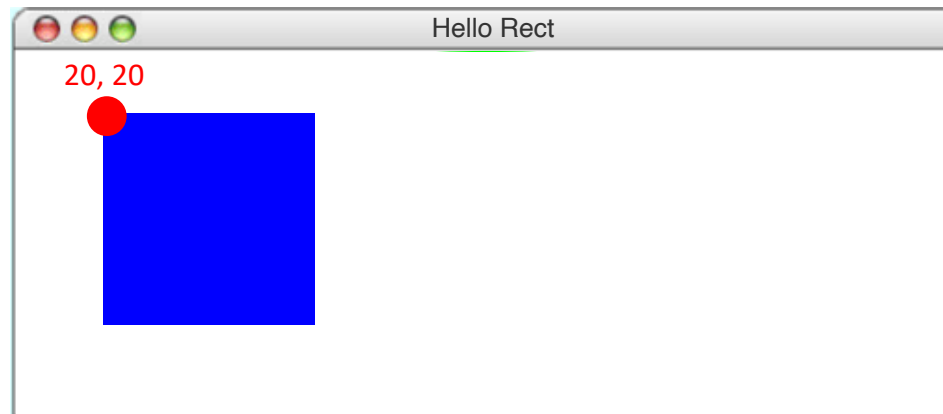
```
def main():  
    canvas = make_canvas(800, 200, 'Hello Rect')
```



# Draw a Rectangle

the following `main` method displays a blue square

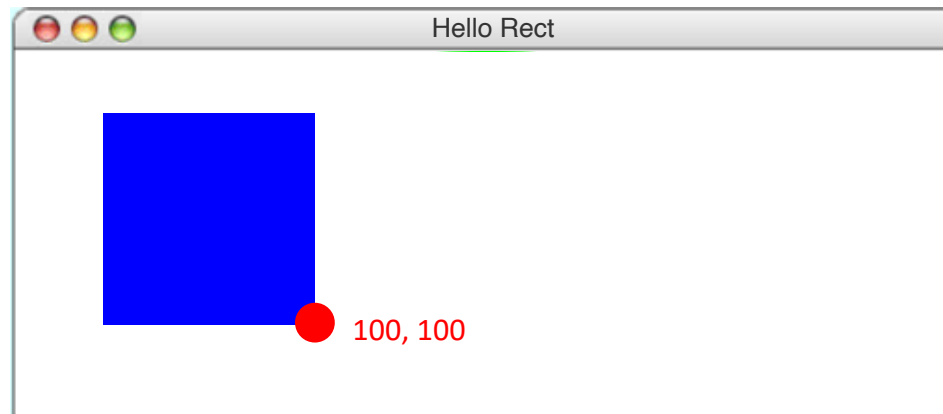
```
def main():  
    canvas = make_canvas(800, 200, 'Hello Rect')  
    canvas.create_rectangle(20, 20, 100, 100, fill="blue")
```



# Draw a Rectangle

the following `main` method displays a blue square

```
def main():  
    canvas = make_canvas(800, 200, 'Hello Rect')  
    canvas.create_rectangle(20, 20, 100, 100, fill="blue")
```

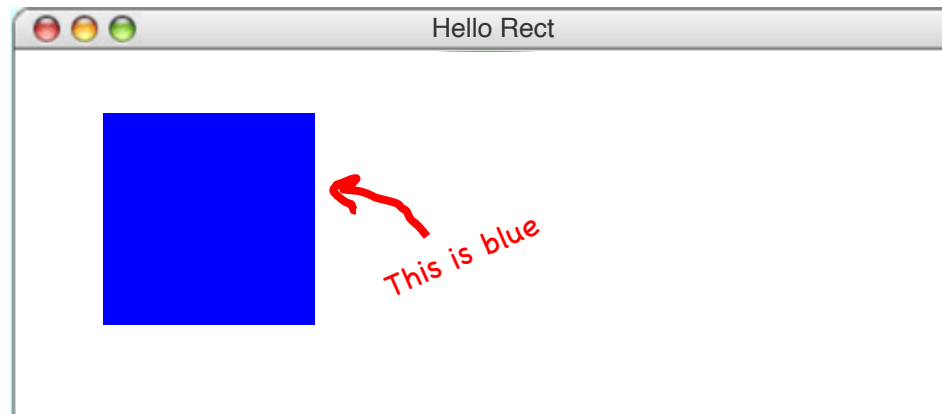




# Draw a Rectangle

the following `main` method displays a blue square

```
def main():  
    canvas = make_canvas(800, 200, 'Hello Rect')  
    canvas.create_rectangle(20, 20, 100, 100, fill="blue")
```



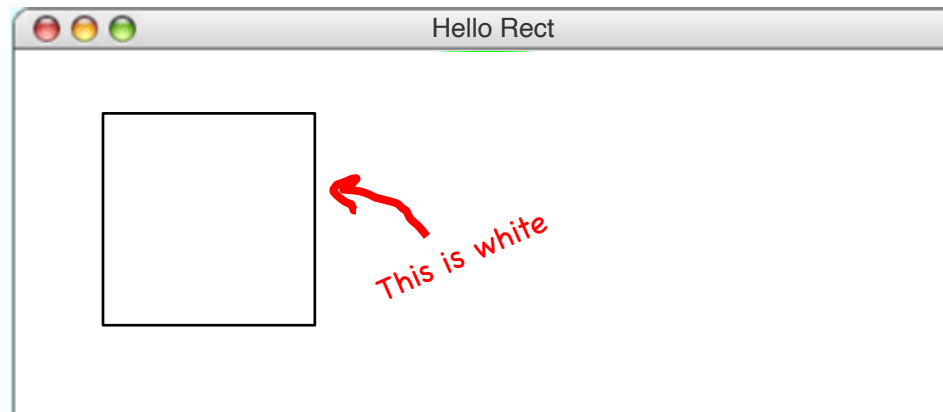
**Aside: Named Arguments**  
*This argument is named as `fill`. It allows functions to have arguments which you can ignore if you want a default value.*



# Draw a Rectangle

the following `main` method displays a blue square

```
def main():  
    canvas = make_canvas(800, 200, 'Hello Rect')  
    canvas.create_rectangle(20, 20, 100, 100)
```



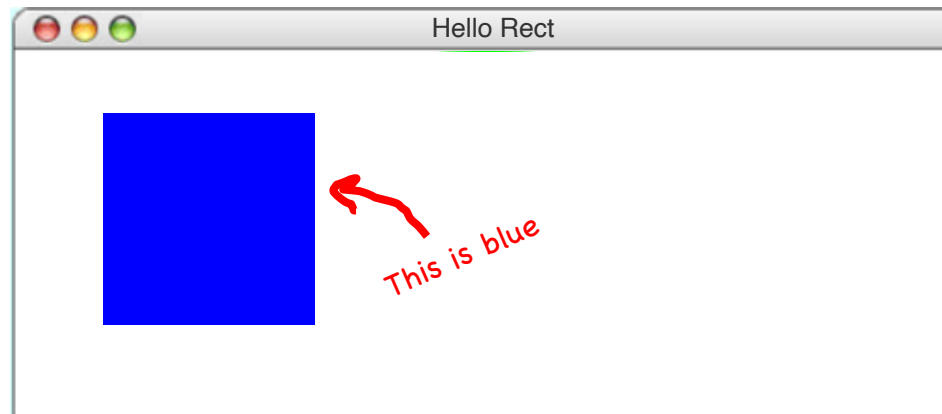
**Aside: Named Arguments**  
*This argument is named as `fill`. It allows functions to have arguments which you can ignore if you want a default value.*



# Draw a Rectangle

the following `main` method displays a blue square

```
def main():  
    canvas = make_canvas(800, 200, 'Hello Rect')  
    canvas.create_rectangle(20, 20, 100, 100, fill="blue")
```



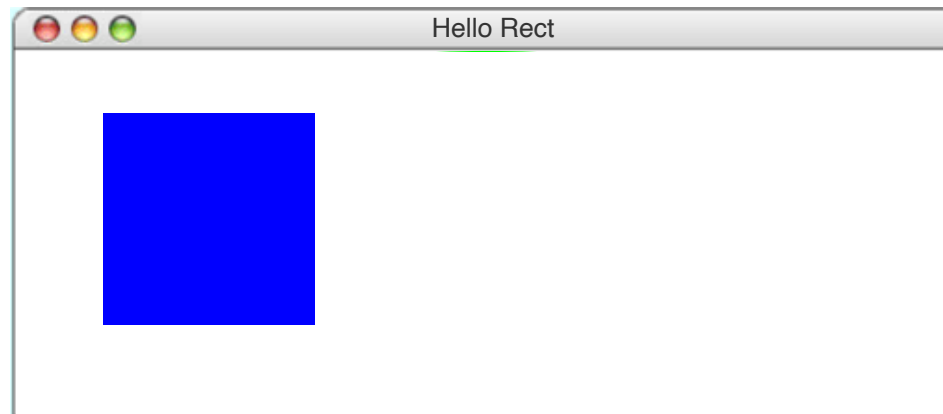
**Aside: Named Arguments**  
*This argument is named as filled. It allows functions to have arguments which you can ignore if you want a default value.*



# Draw a Rectangle

the following `main` method displays a blue square

```
def main():  
    canvas = make_canvas(800, 200, 'Hello Rect')  
    canvas.create_rectangle(20, 20, 100, 100, fill="blue")  
    canvas.mainloop()
```



# TK Natural Graphics



# Graphics Coordinates

0,0

x 40,20

x 120,40

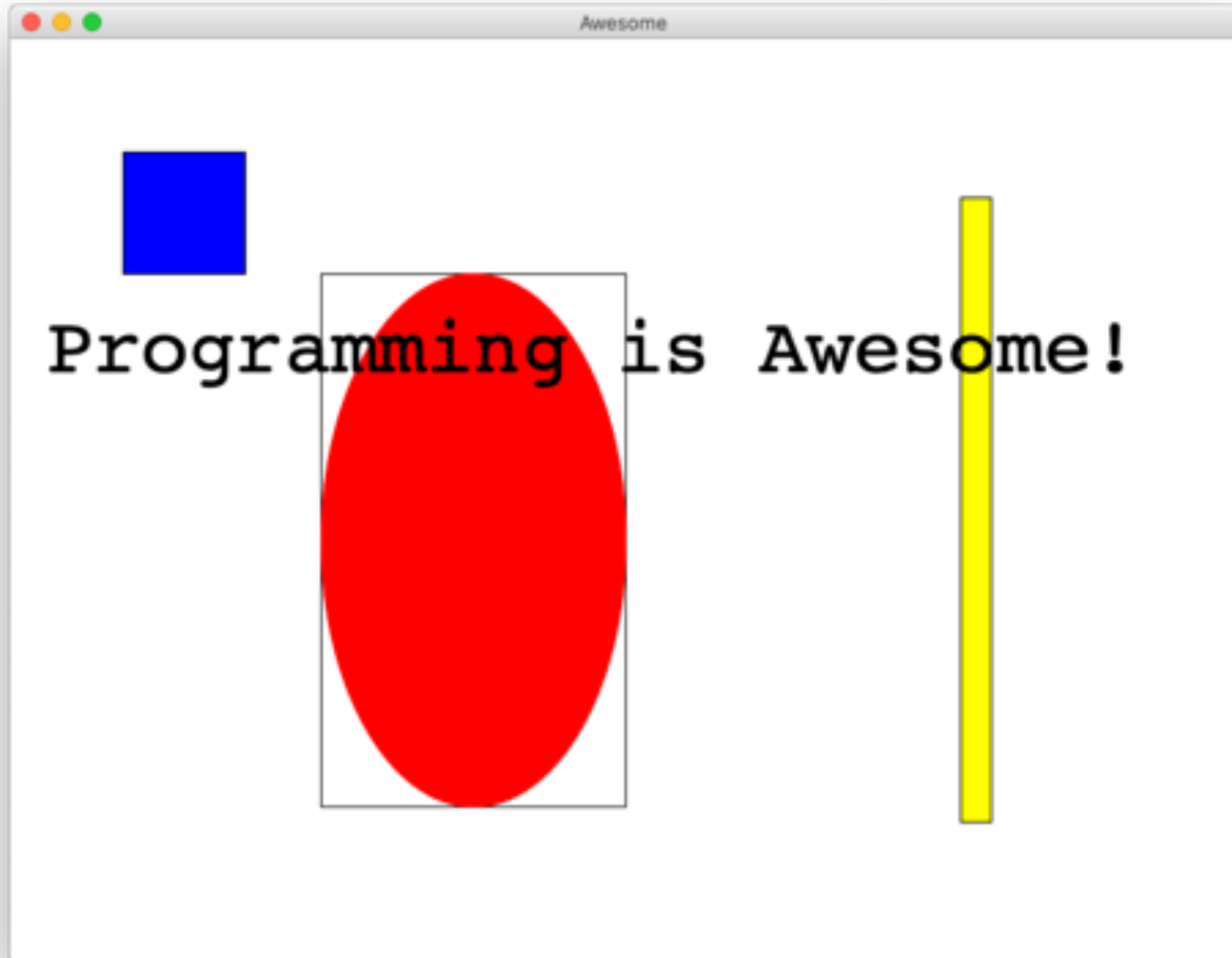
x 40,120

CANVAS\_WIDTH

CANVAS\_HEIGHT



# Rectangles, Ovals, Text



- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text()`





- `canvas.create_line(x1, y1, x2, y2)`
- `canvas.create_oval()`
- `canvas.create_text()`



- `canvas.create_line(x1, y1, x2, y2)`

- `canvas.create_oval()`

- `canvas.create_text()`



The first point of the  
line is  $(x_1, y_1)$



- `canvas.create_line(x1, y1, x2, y2)`

- `canvas.create_oval()`

- `canvas.create_text()`



The second point of the line is `(x2, y2)`



- `canvas.create_line(x1, y1, x2, y2)`

- `canvas.create_oval()`

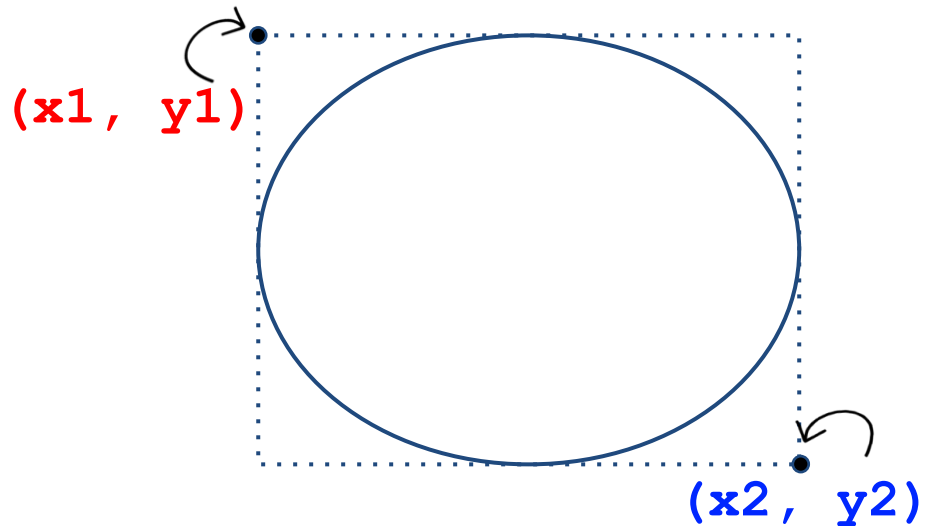
- `canvas.create_text()`



- `canvas.create_line()`
- `canvas.create_oval(x1, y1, x2, y2)`
- `canvas.create_text()`



- `canvas.create_line()`
- `canvas.create_oval(x1, y1, x2, y2)`
- `canvas.create_text()`

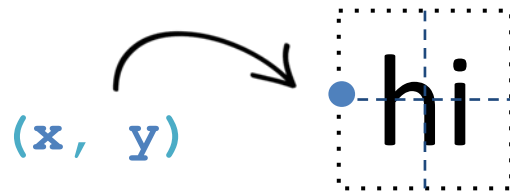


- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi')`

hi



- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi', anchor='w')`





# Pedagogy



CS106A Lectures ▾ Assignments ▾ Section ▾ Handouts ▾ Examples ▾ Schedule

## Programming is Awesome

BY CHRIS PIECH

Graphics are really fantastic in python, especially using the TK library (which is the standard for Python). There are a lot of details, and as such a great way to learn is to look at worked examples.



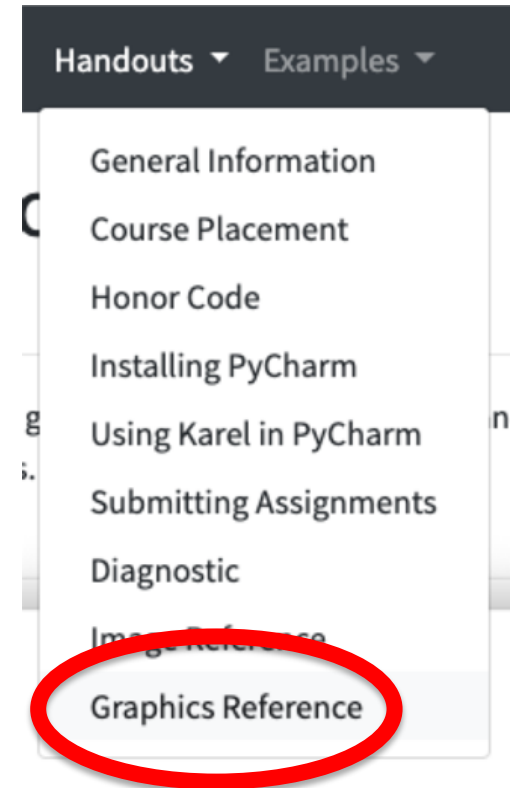
**Solution**

```
import tkinter
from PIL import ImageTk
from PIL import Image

CANVAS_WIDTH = 800
CANVAS_HEIGHT = 600

def main():
    canvas = make_canvas(CANVAS_WIDTH, CANVAS_HEIGHT, "Awesome")
    # a line for good measure!
    canvas.create_line(0, 0, 600, 600)

    # a blue square with width and height = 80
    canvas.create_rectangle(70, 70, 150, 150, fill="blue")
    # a yellow rectangle that is long and skinny
    canvas.create_rectangle(620, 100, 640, 510, fill="yellow")
```

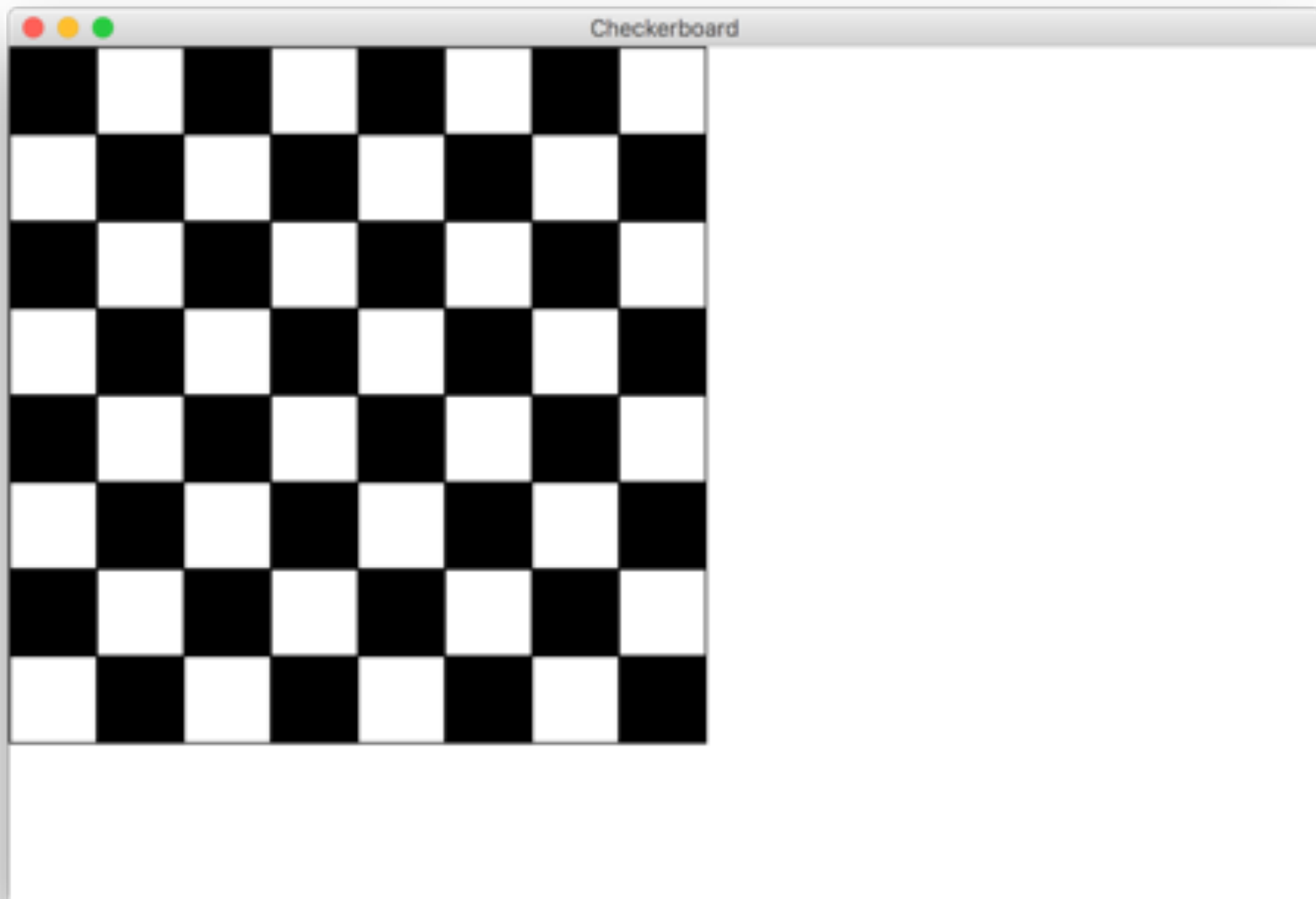


Handouts ▾ Examples ▾

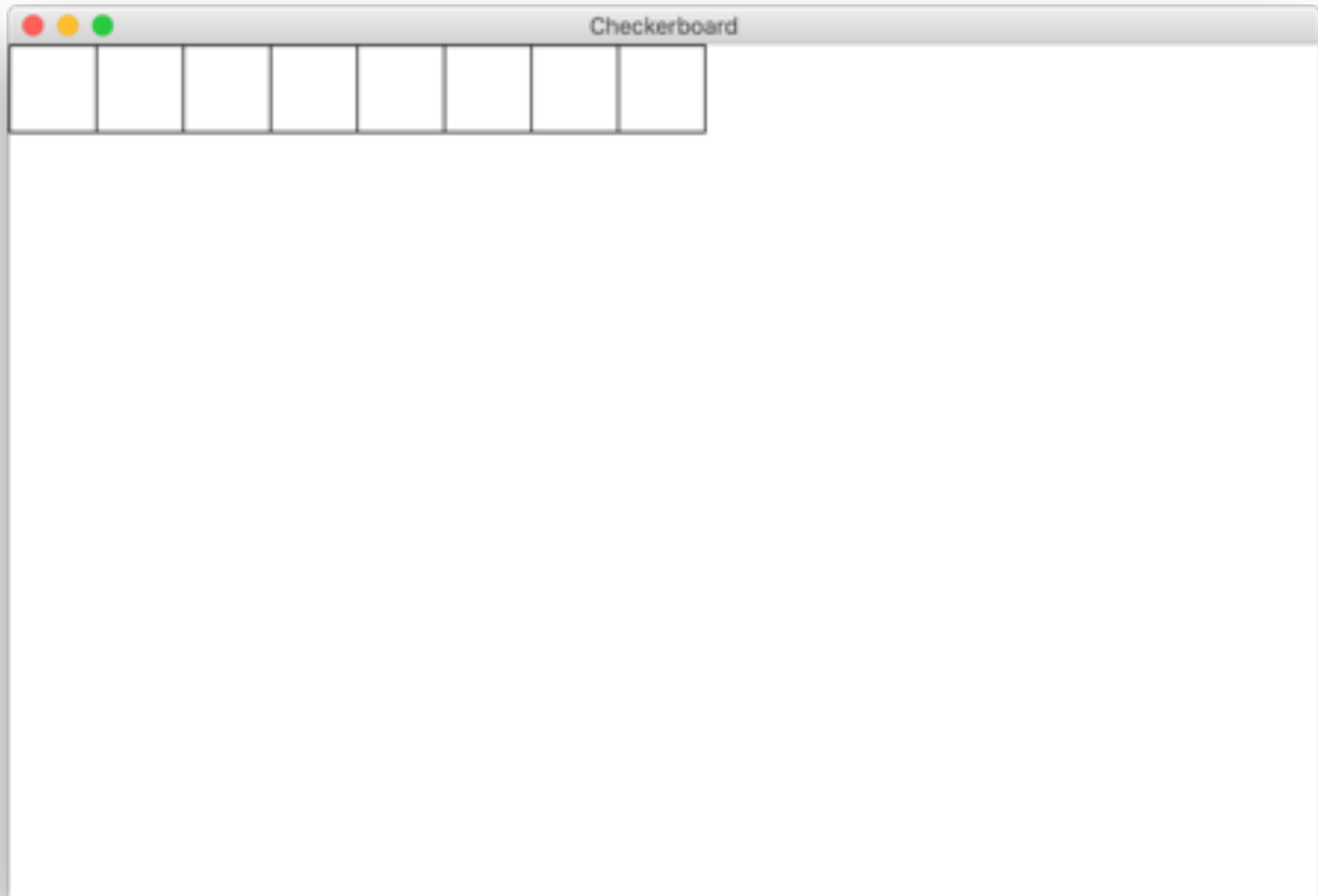
- General Information
- Course Placement
- Honor Code
- Installing PyCharm
- Using Karel in PyCharm
- Submitting Assignments
- Diagnostic
- Image Reference
- Graphics Reference**



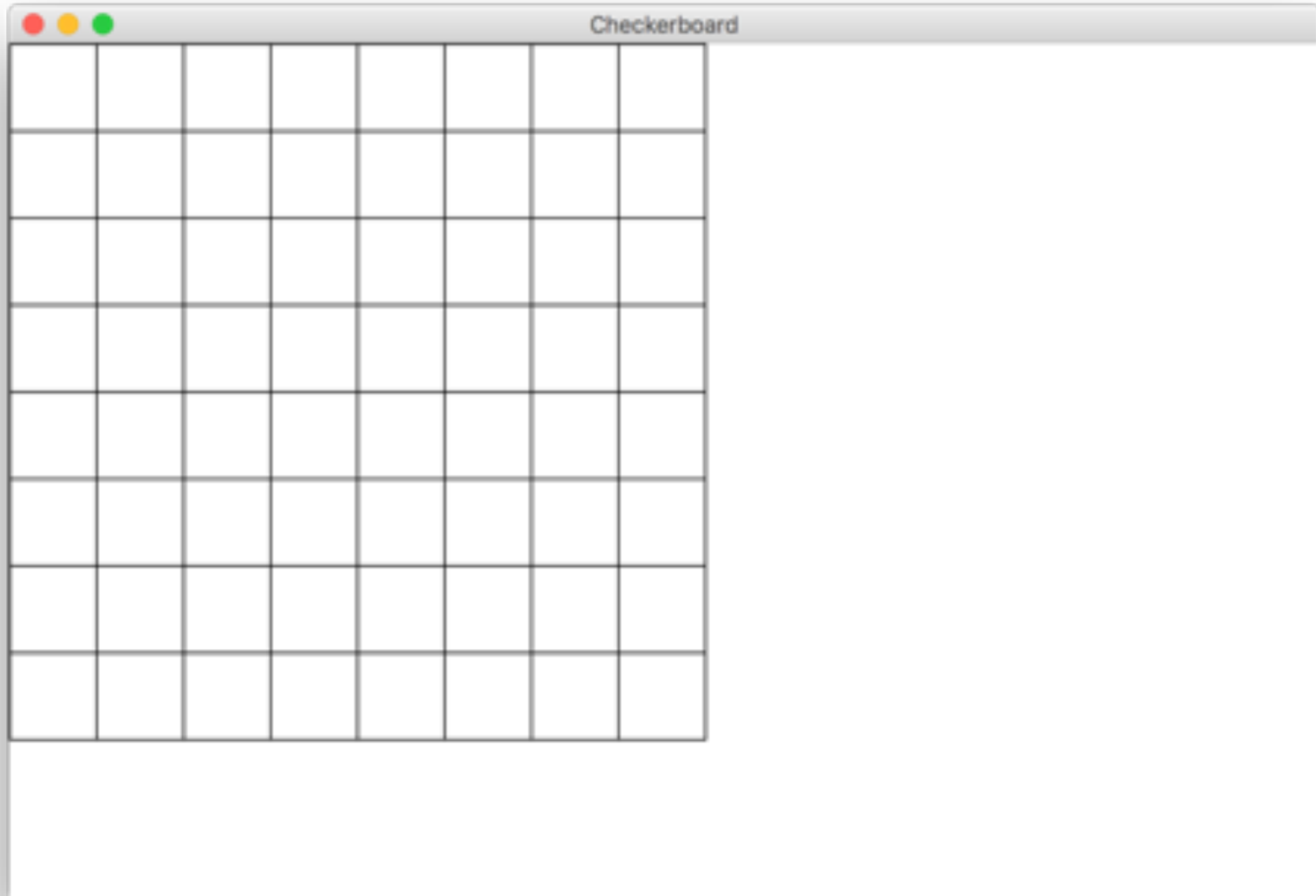
# Goal



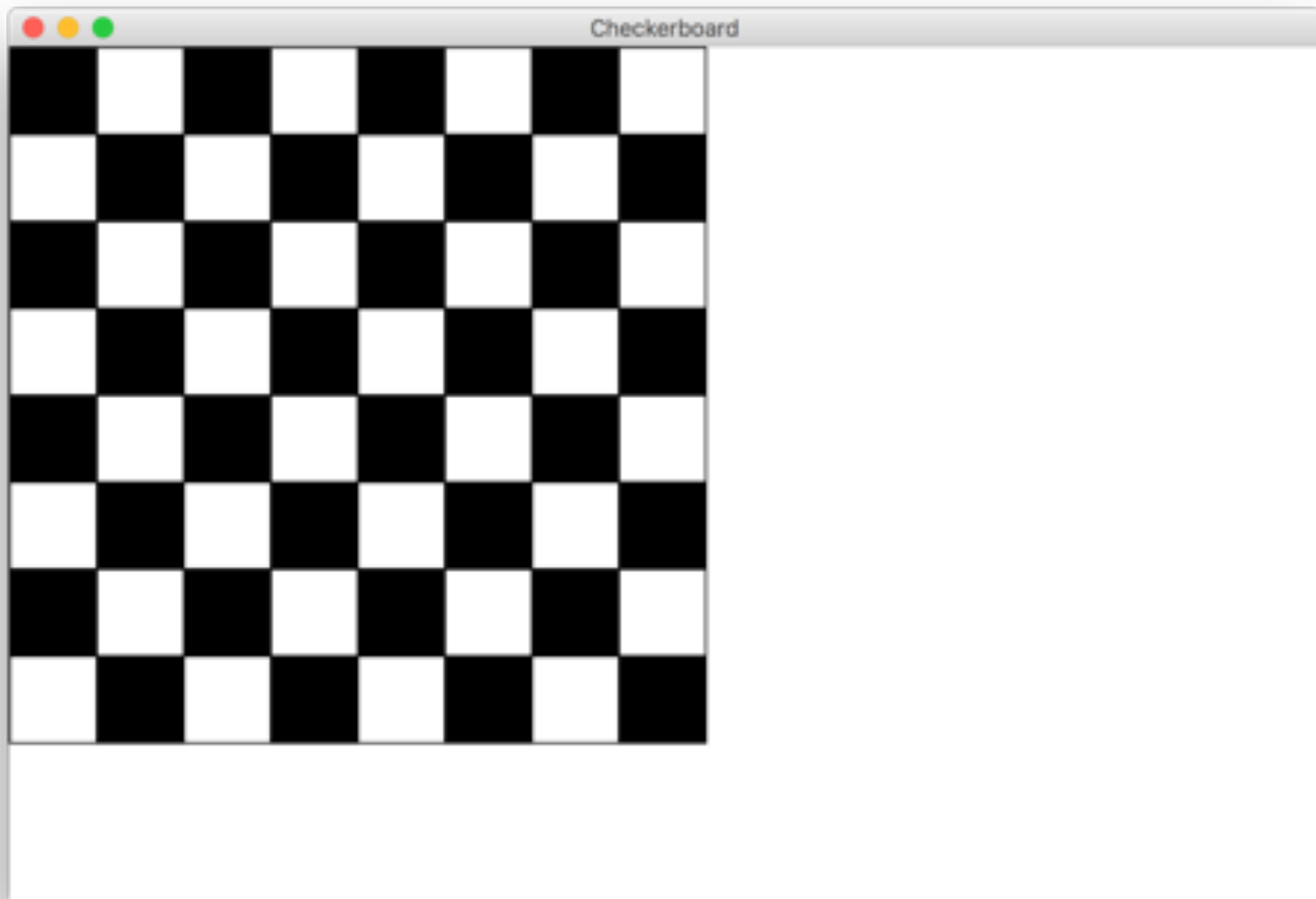
# Milestone 1

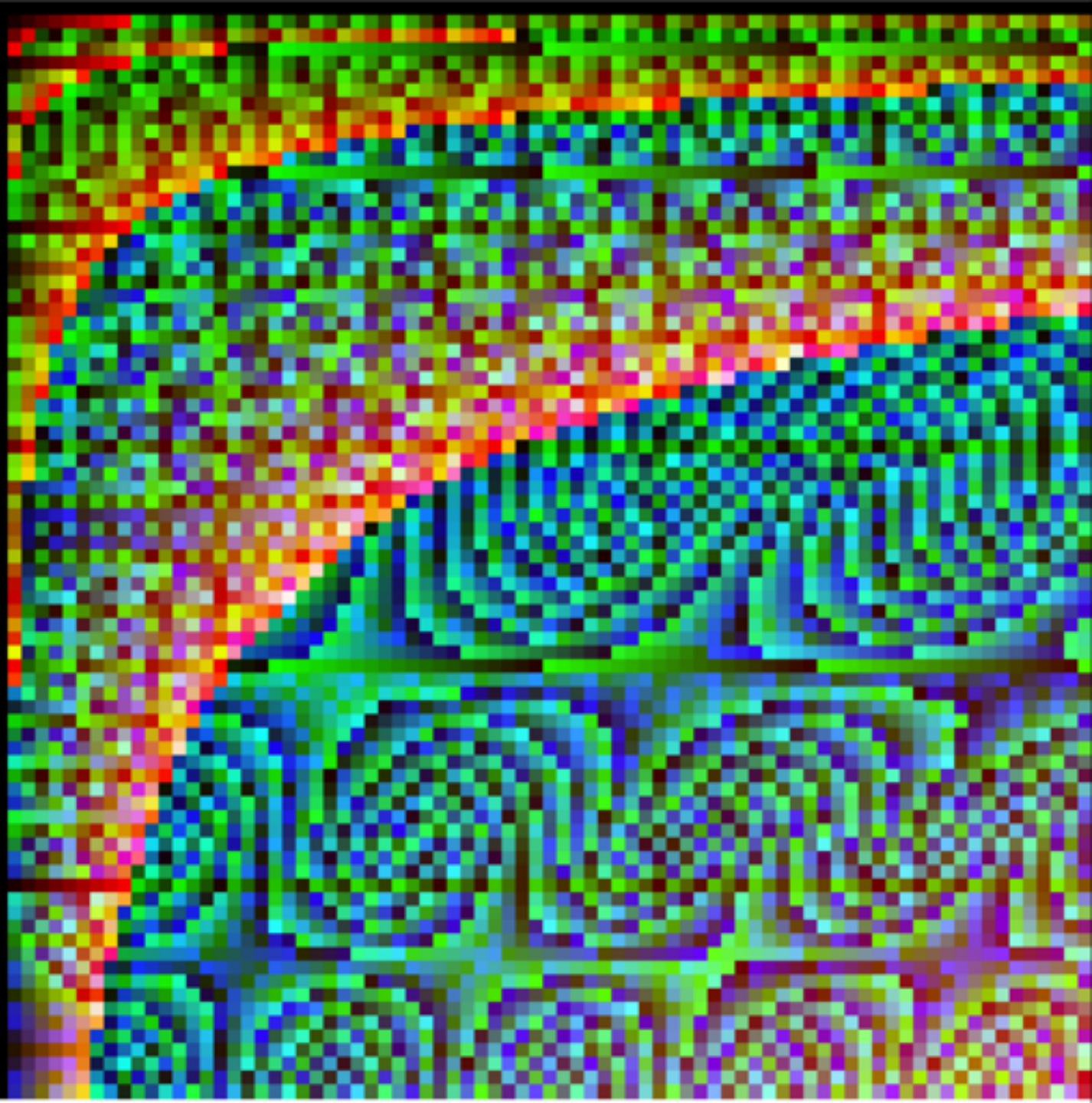


# Milestone 2



# Milestone 3







# Hold up!

```
def draw_square(canvas, row, col):
```

*If you get a copy when you pass a parameter. Does this copy the canvas??!!*

*Large variables are stored using something like a URL. The URL gets copied*





# How do you share google docs?



[https://docs.google.com/document/d/1eBtnEill3KHe\\_fFS-kSAOpXqeSXpbfTTMImOgj6I9dvk/](https://docs.google.com/document/d/1eBtnEill3KHe_fFS-kSAOpXqeSXpbfTTMImOgj6I9dvk/)



```
def main():
```

```
    canvas = make_canvas(...)
```

```
    draw_square(canvas)
```

```
def draw_square(canvas):
```

```
    canvas.create_rectangle(20, 20, 100, 100)
```

---

stack

heap

**main**



```
def main():
```

```
    canvas = make_canvas(...)
```

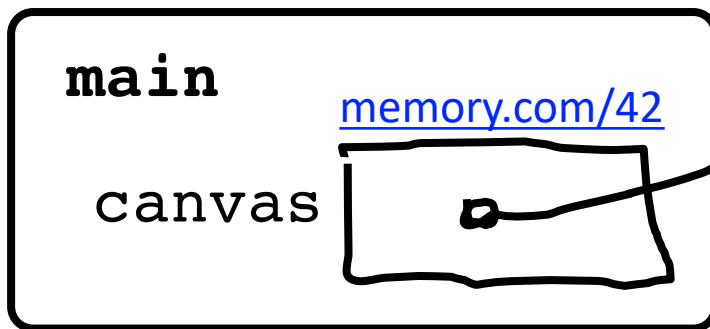
```
    draw_square(canvas)
```

```
def draw_square(canvas):
```

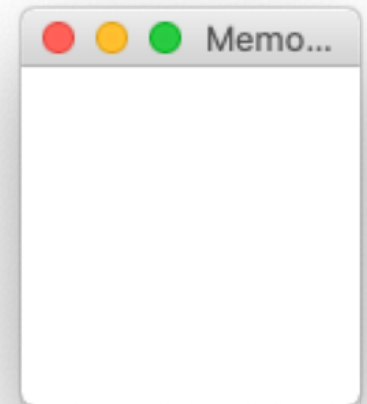
```
    canvas.create_rectangle(20, 20, 100, 100)
```

stack

heap

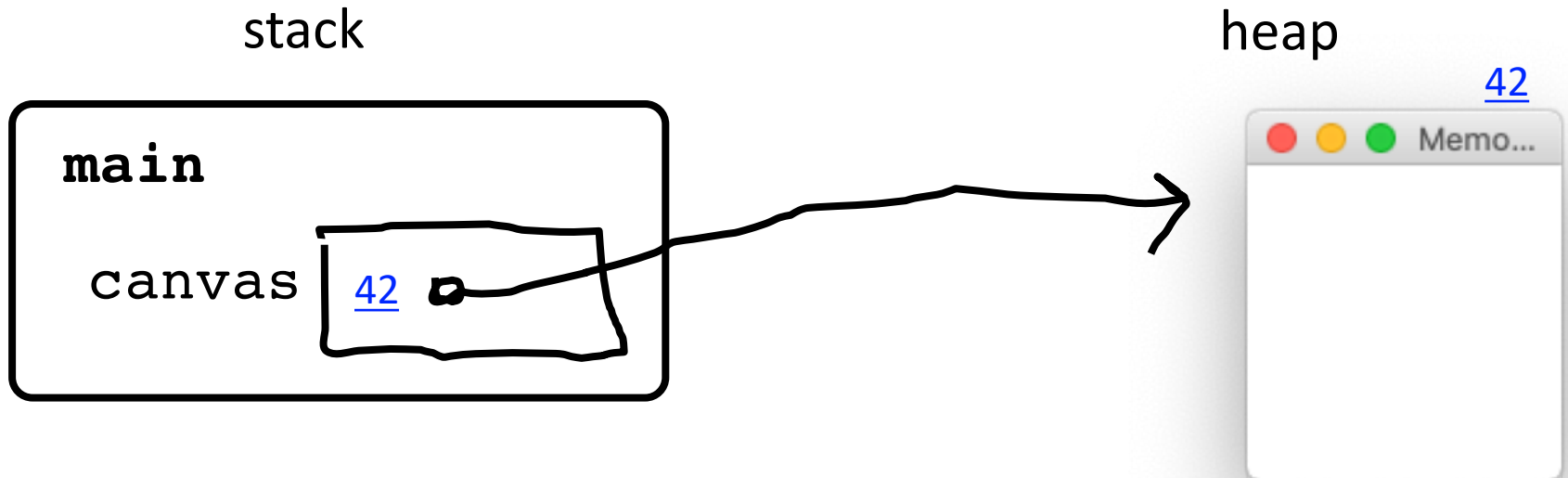


**\*\* Binding \*\***



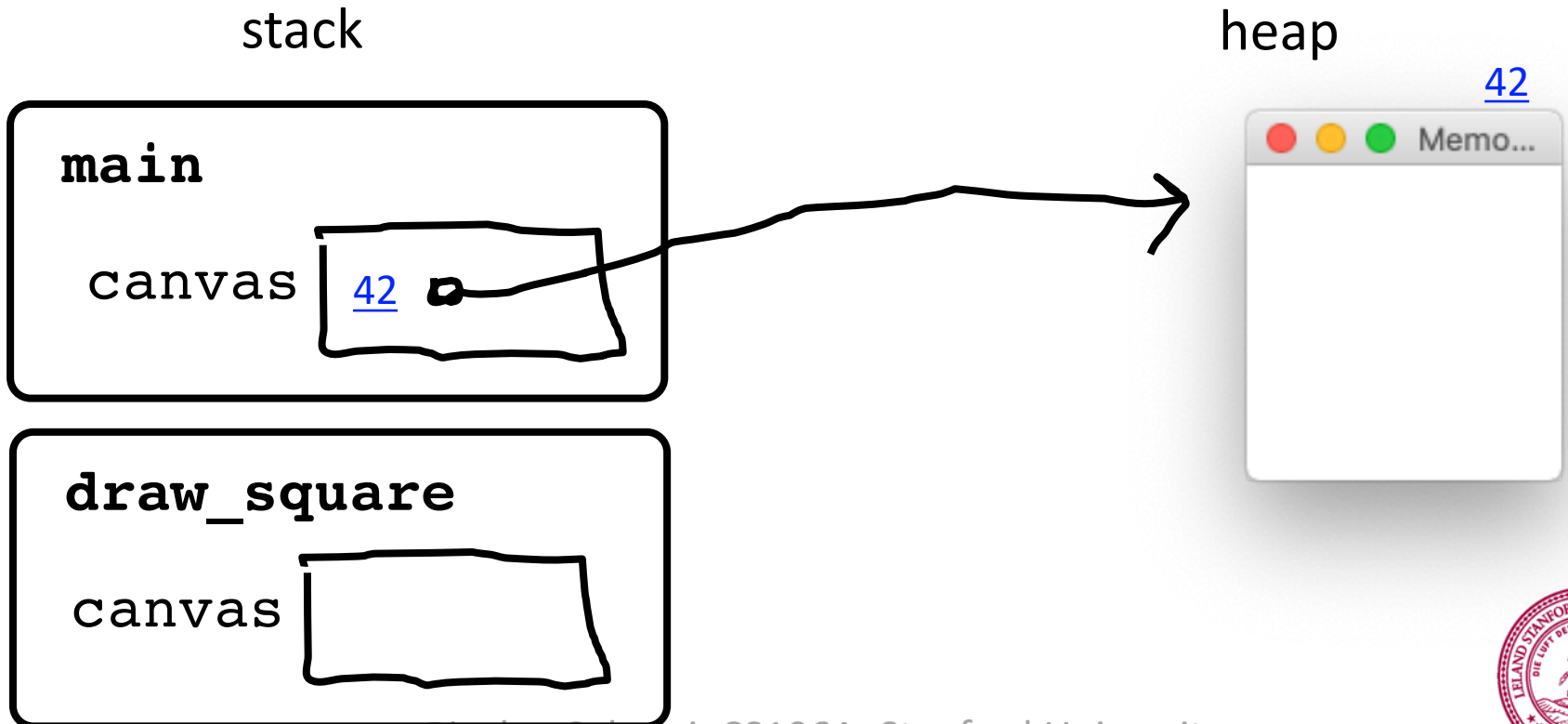
```
def main():  
    canvas = make_canvas(...)  
    draw_square(canvas)
```

```
def draw_square(canvas):  
    canvas.create_rectangle(20, 20, 100, 100)
```



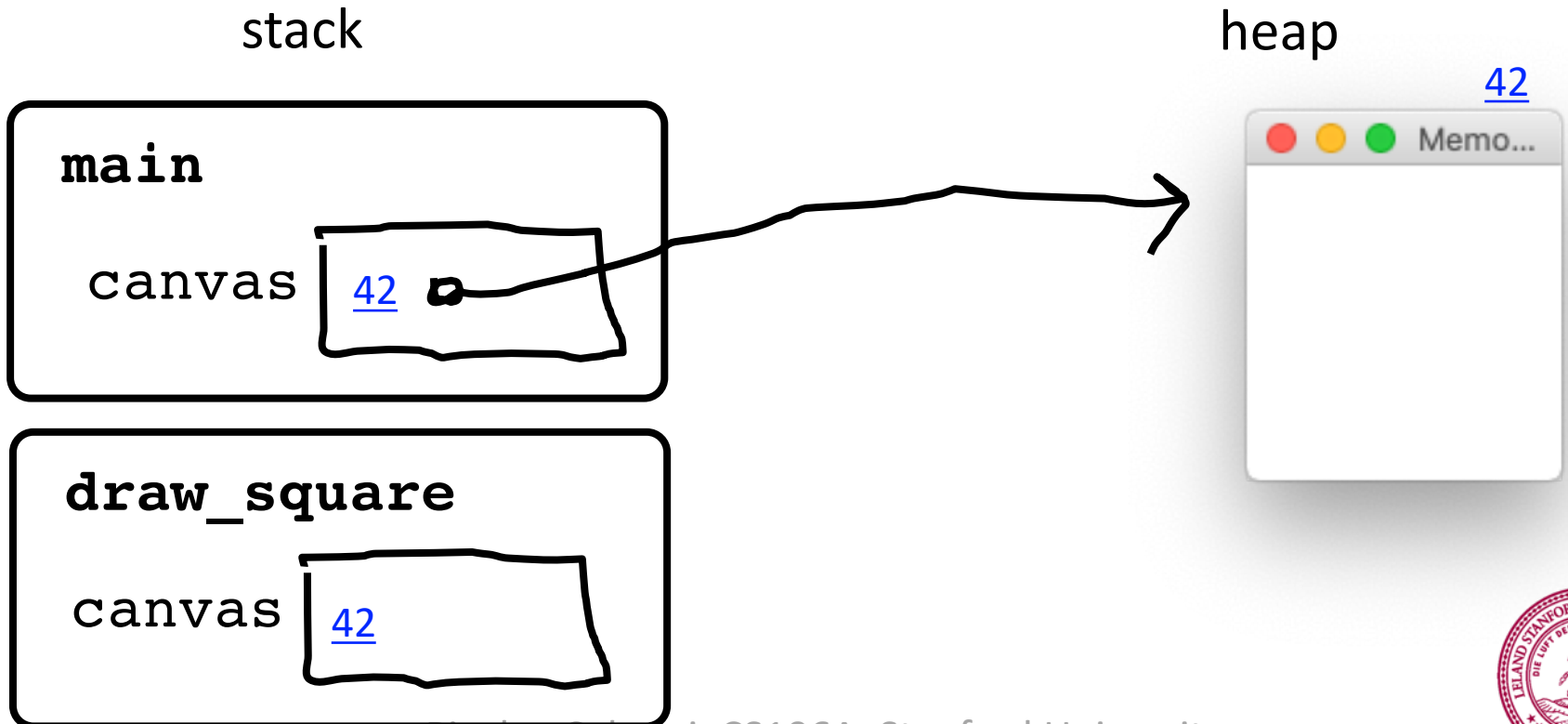
```
def main():  
    canvas = make_canvas(...)  
    draw_square(canvas)
```

```
def draw_square(canvas):  
    canvas.create_rectangle(20, 20, 100, 100)
```



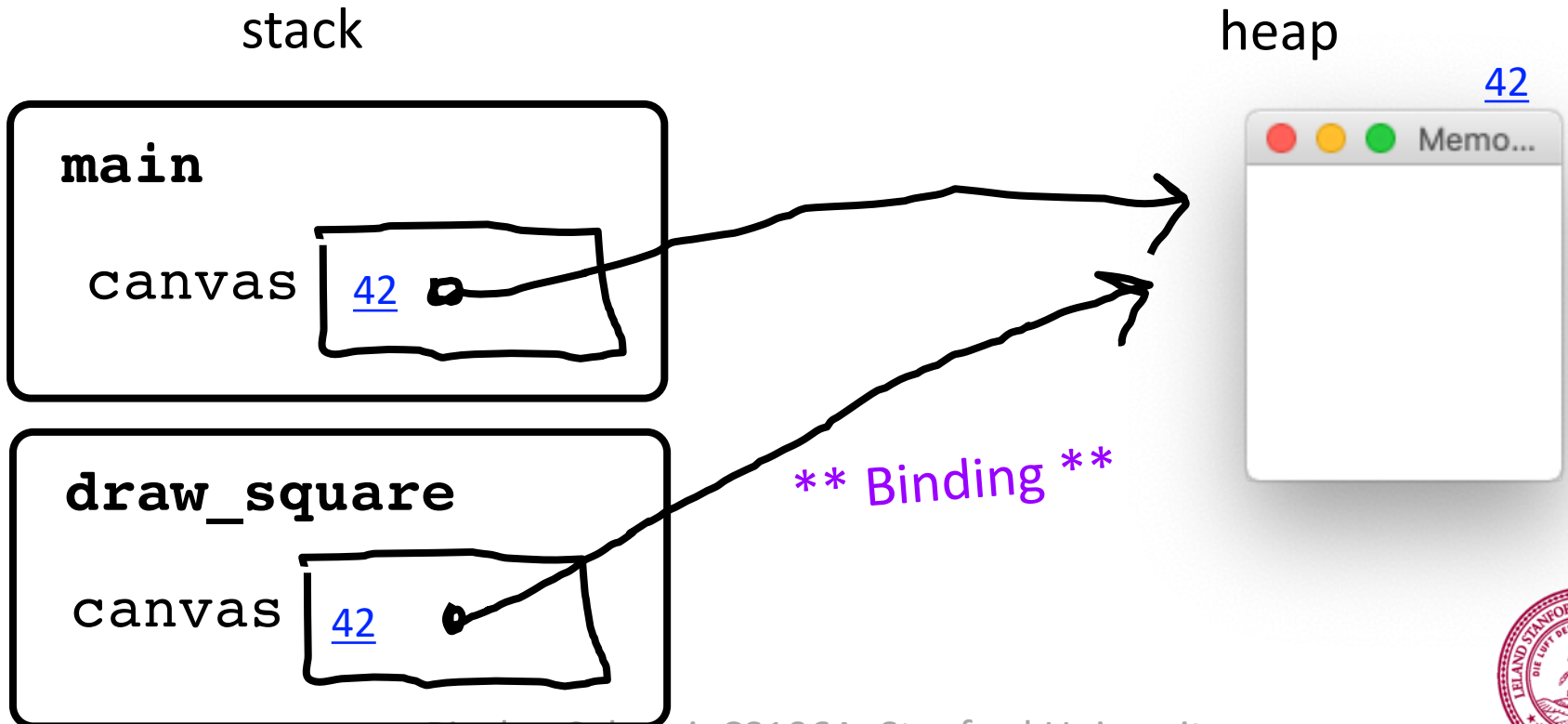
```
def main():  
    canvas = make_canvas(...)  
    draw_square(canvas)
```

```
def draw_square(canvas):  
    canvas.create_rectangle(20, 20, 100, 100)
```



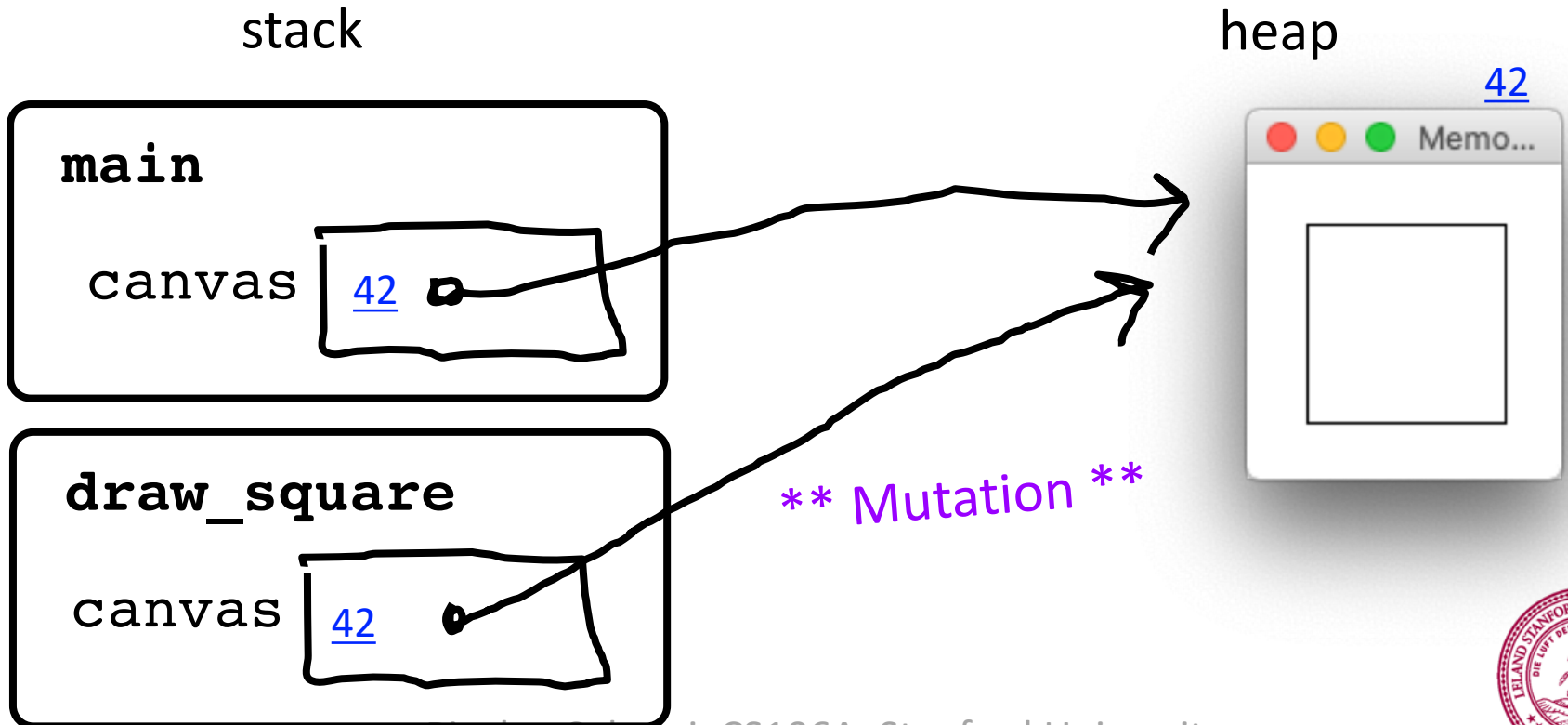
```
def main():  
    canvas = make_canvas(...)  
    draw_square(canvas)
```

```
def draw_square(canvas):  
    canvas.create_rectangle(20, 20, 100, 100)
```



```
def main():
    canvas = make_canvas(...)
    draw_square(canvas)
```

```
def draw_square(canvas):
    canvas.create_rectangle(20, 20, 100, 100)
```





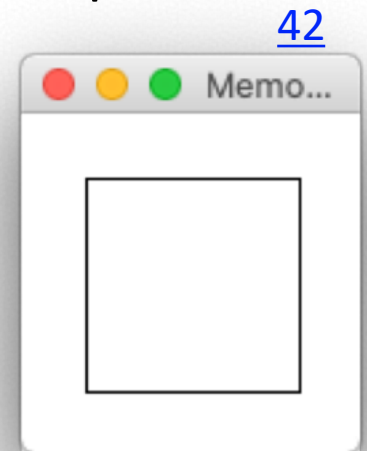
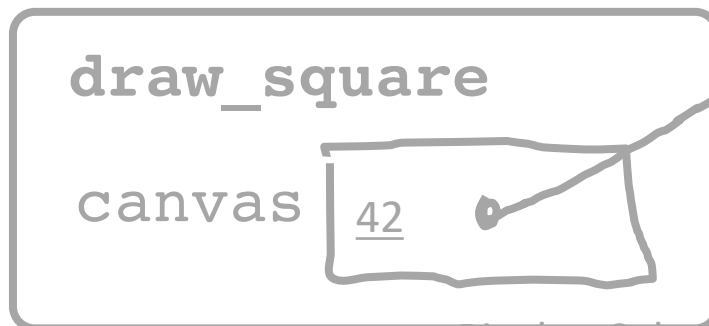
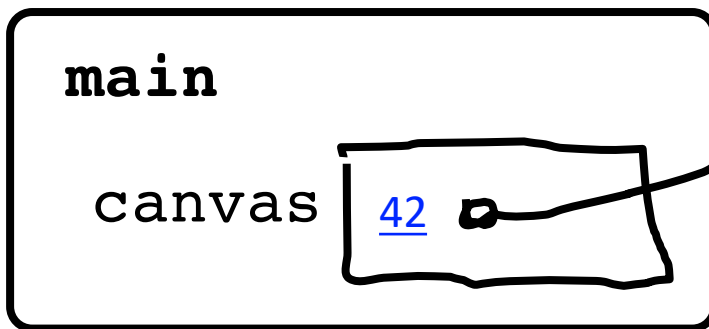
```
def main():
    canvas = make_canvas(...)
    draw_square(canvas)
```

```
def draw_square(canvas):
    canvas.create_rectangle(20, 20, 100, 100)
```



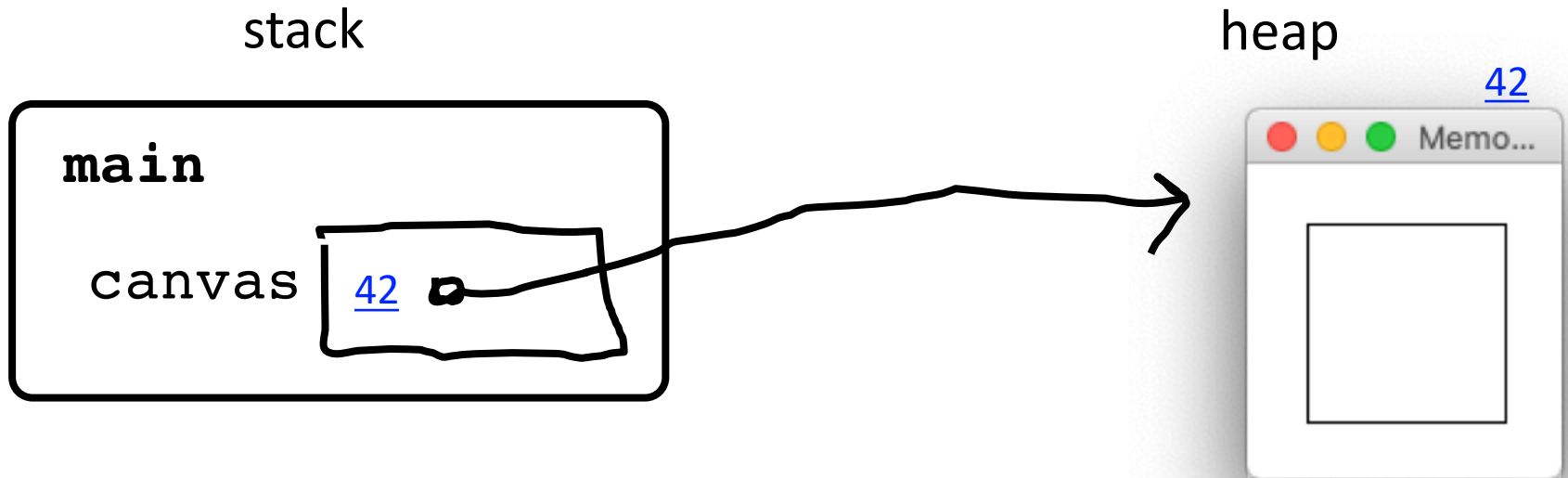
stack

heap



```
def main():  
    canvas = make_canvas(...)  
    draw_square(canvas)
```

```
def draw_square(canvas):  
    canvas.create_rectangle(20, 20, 100, 100)
```



Variables are stored as  
memory addresses.



**Binding** is changing  
memory address.

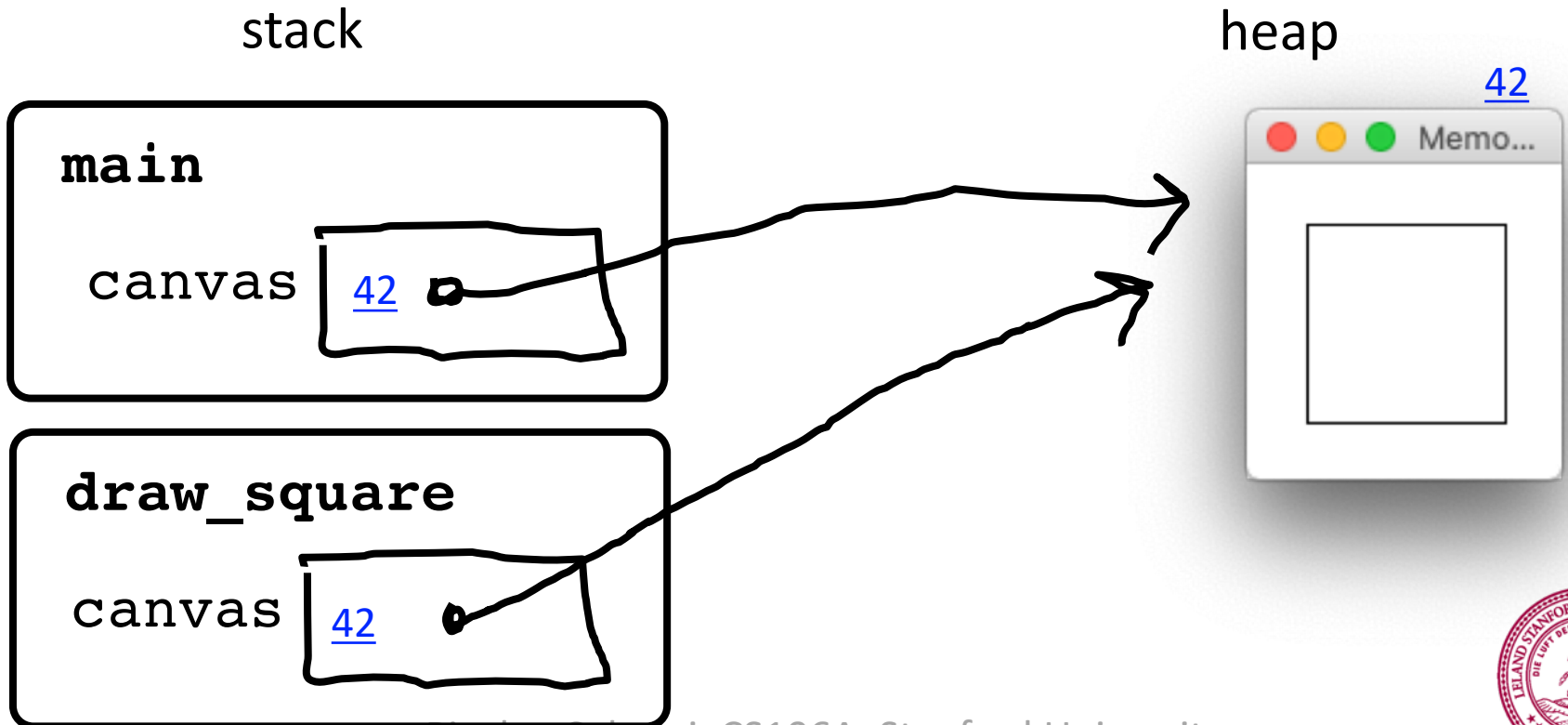
**Mutating** is editing whats  
on the other end.

(memory addresses are like  
memory URLs)



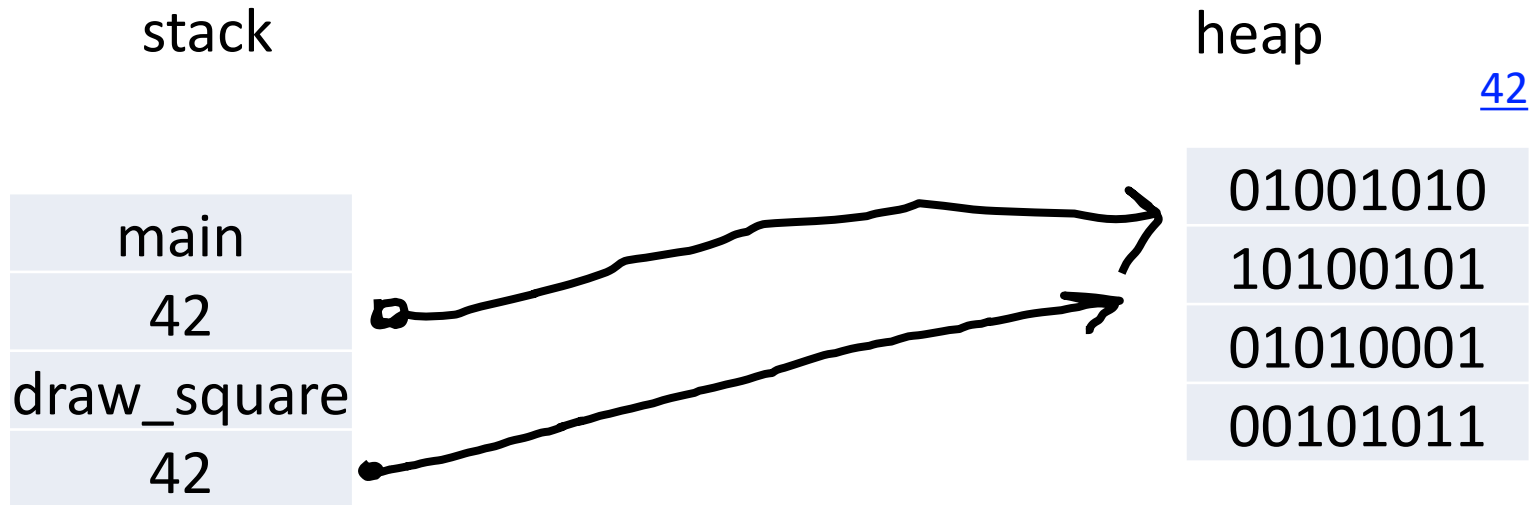
```
def main():  
    canvas = make_canvas(...)  
    draw_square(canvas)
```

```
def draw_square(canvas):  
    canvas.create_rectangle(20, 20, 100, 100)
```



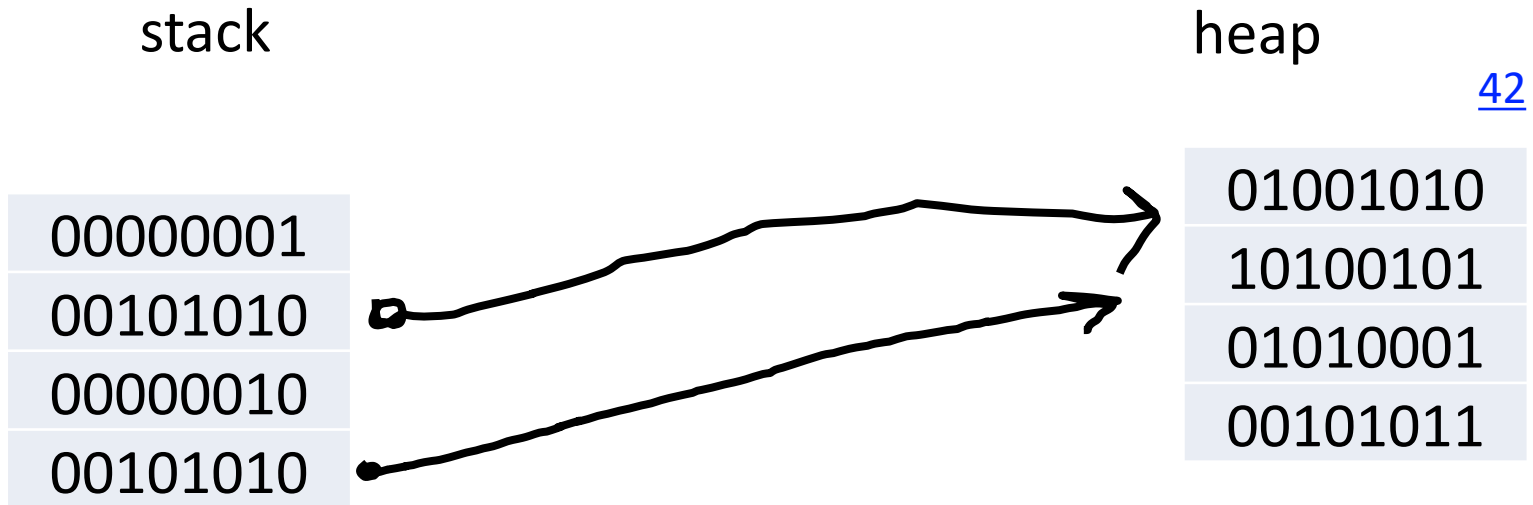
```
def main():  
    canvas = make_canvas(...)  
    draw_square(canvas)
```

```
def draw_square(canvas):  
    canvas.create_rectangle(20, 20, 100, 100)
```



```
def main():
    canvas = make_canvas(...)
    draw_square(canvas)
```

```
def draw_square(canvas):
    canvas.create_rectangle(20, 20, 100, 100)
```



```
def main():
    canvas = make_canvas(...)
    draw_square(canvas)
```

```
def draw_square(canvas):
    canvas.create_rectangle(20, 20, 100, 100)
```

---

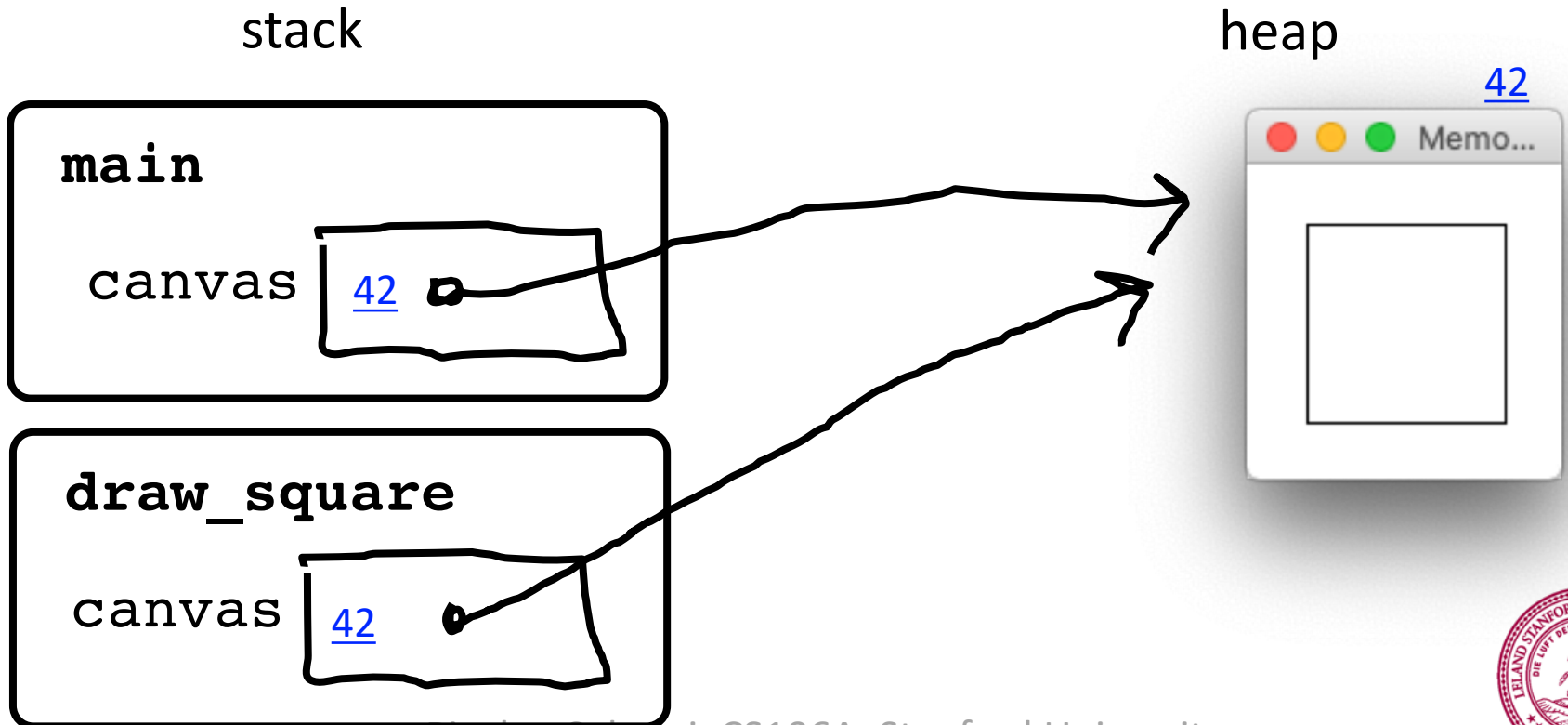
00000001	stack
00101010	↓
00000010	
00101010	

01001010	↑
10100101	
01010001	heap
00101011	



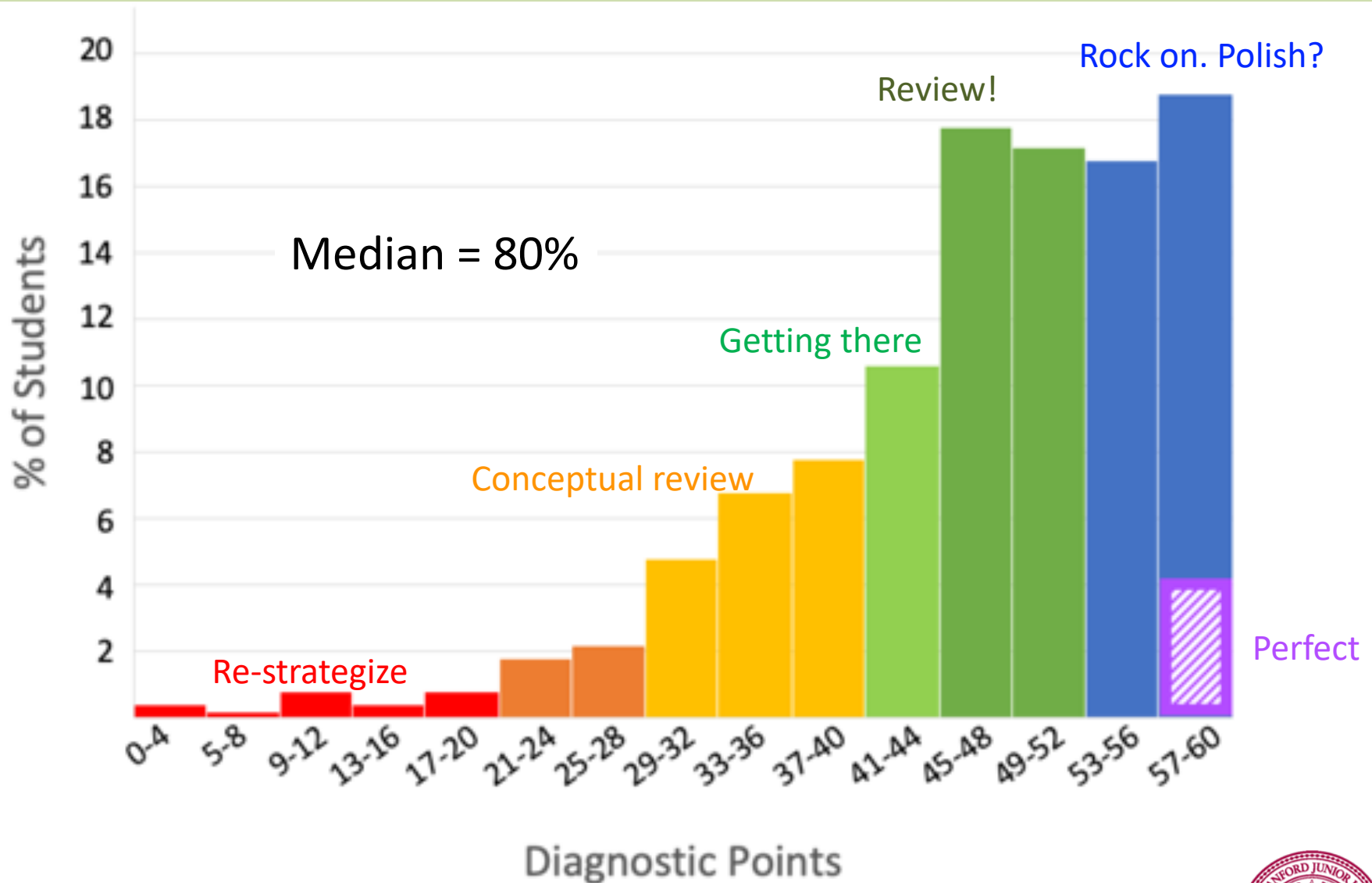
```
def main():  
    canvas = make_canvas(...)  
    draw_square(canvas)
```

```
def draw_square(canvas):  
    canvas.create_rectangle(20, 20, 100, 100)
```





# Diagnostic 1



# Diagnostic 1

- Mastery is still under your control
  - Feedback is how one gets better
  - Many people have trouble at this point, but hit their learning goals by the end of the quarter
- Grade is still under your control
  - We look at the delta between diagnostics
  - 10% was intentional
- First CS test!

