

# Graphics

Chris Piech and Mehran Sahami  
CS106A, Stanford University

# Breakout Demo

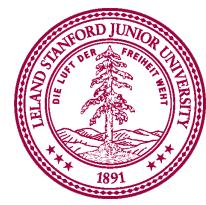
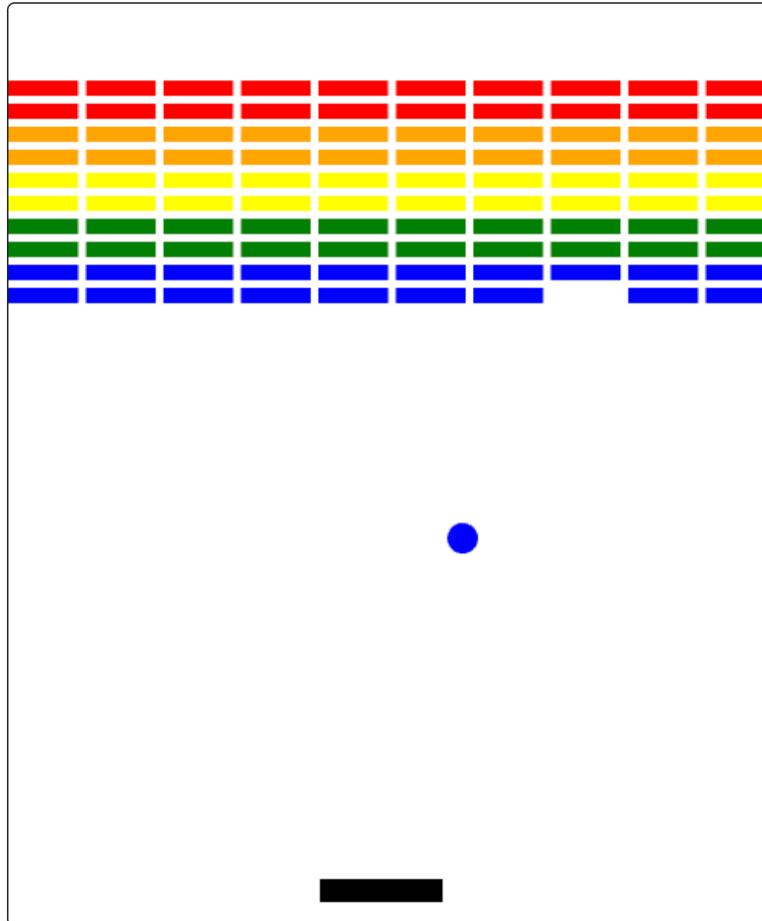


Created by Chris Piech ❤

Hello! I made this for CS106A!

Run

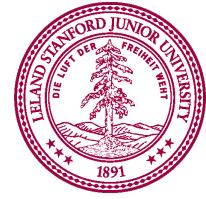
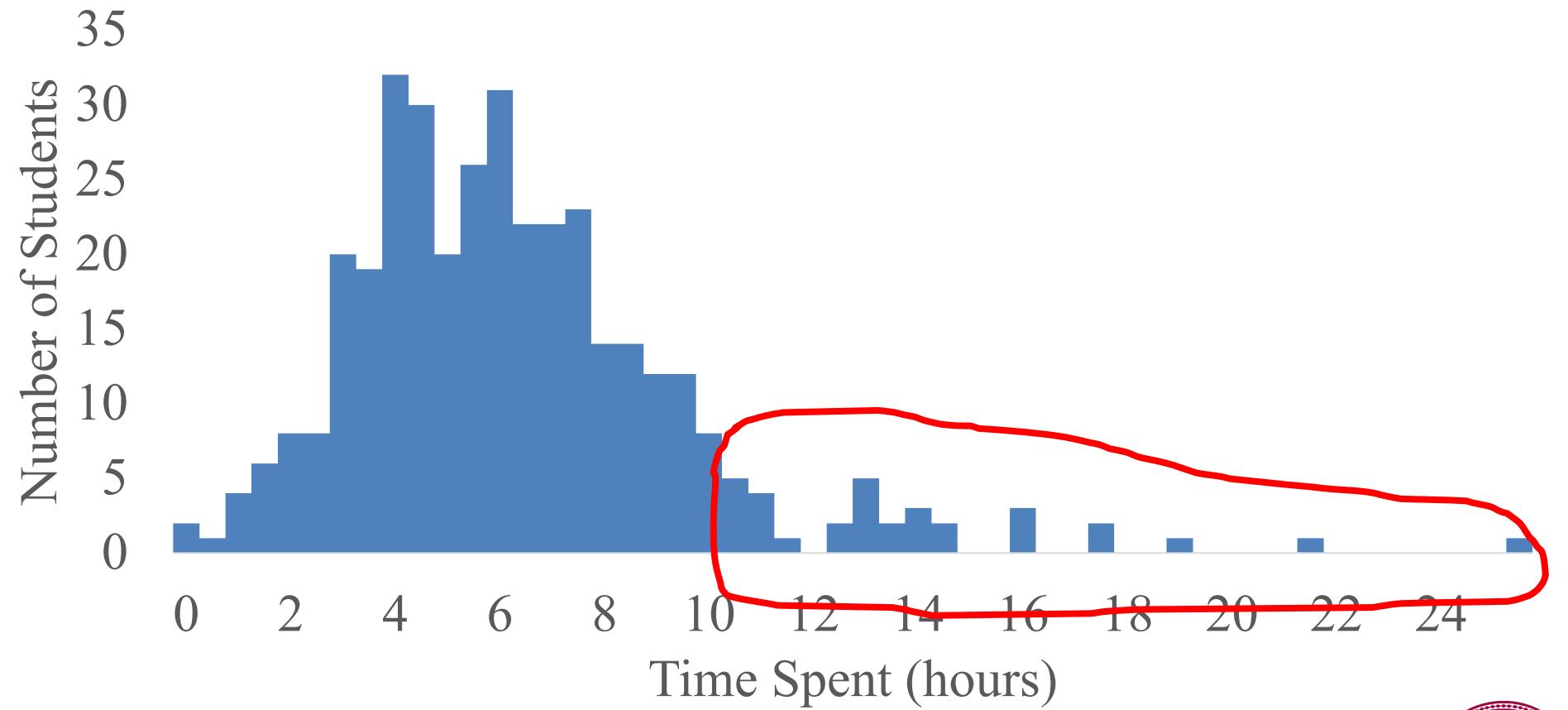
Editable (Owner)



Get a little meta...

# Learn by Doing

## Assignment 2



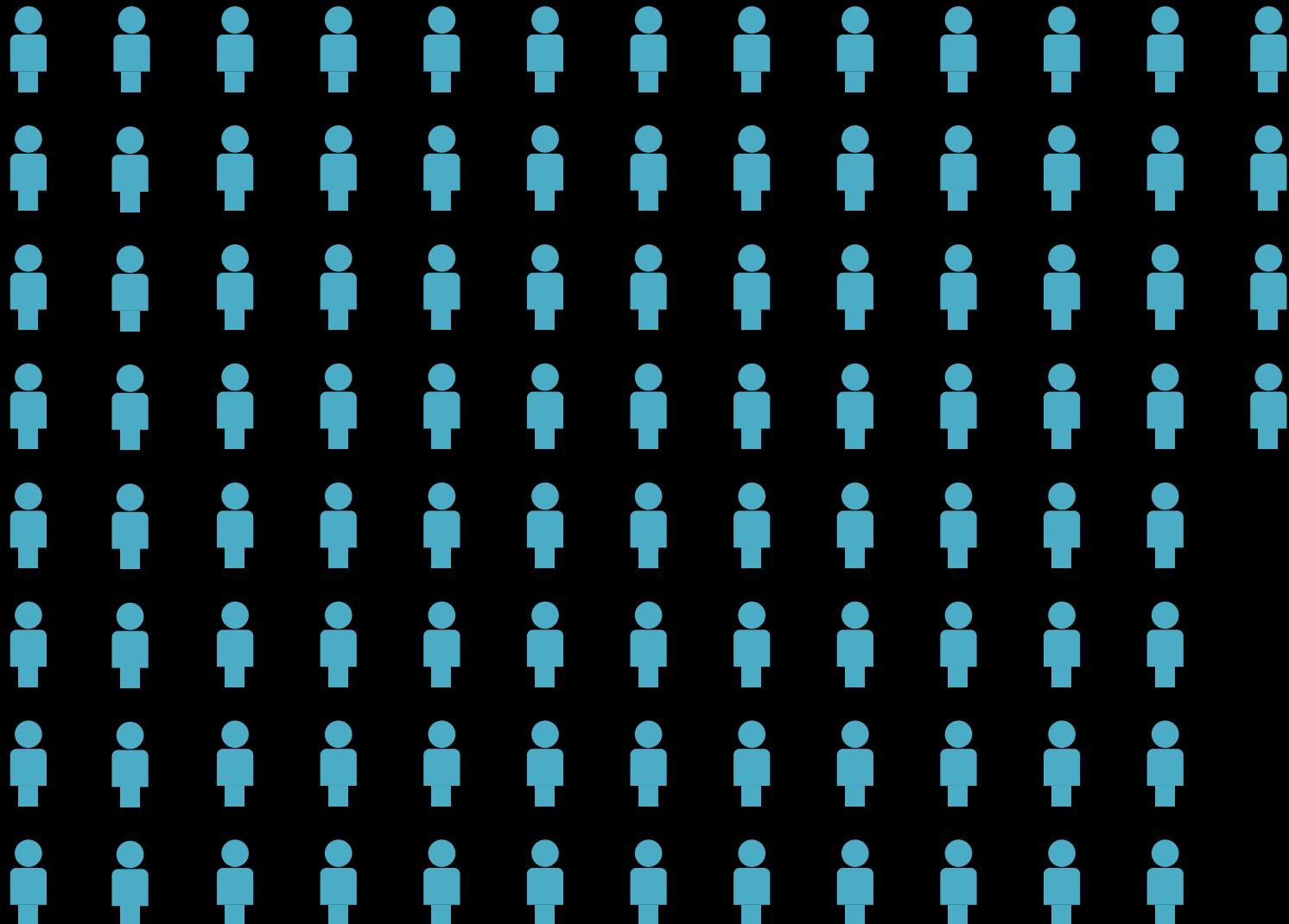
# Learning to Program on the Internet



Task

Almost a hundred thousand  
unique solutions

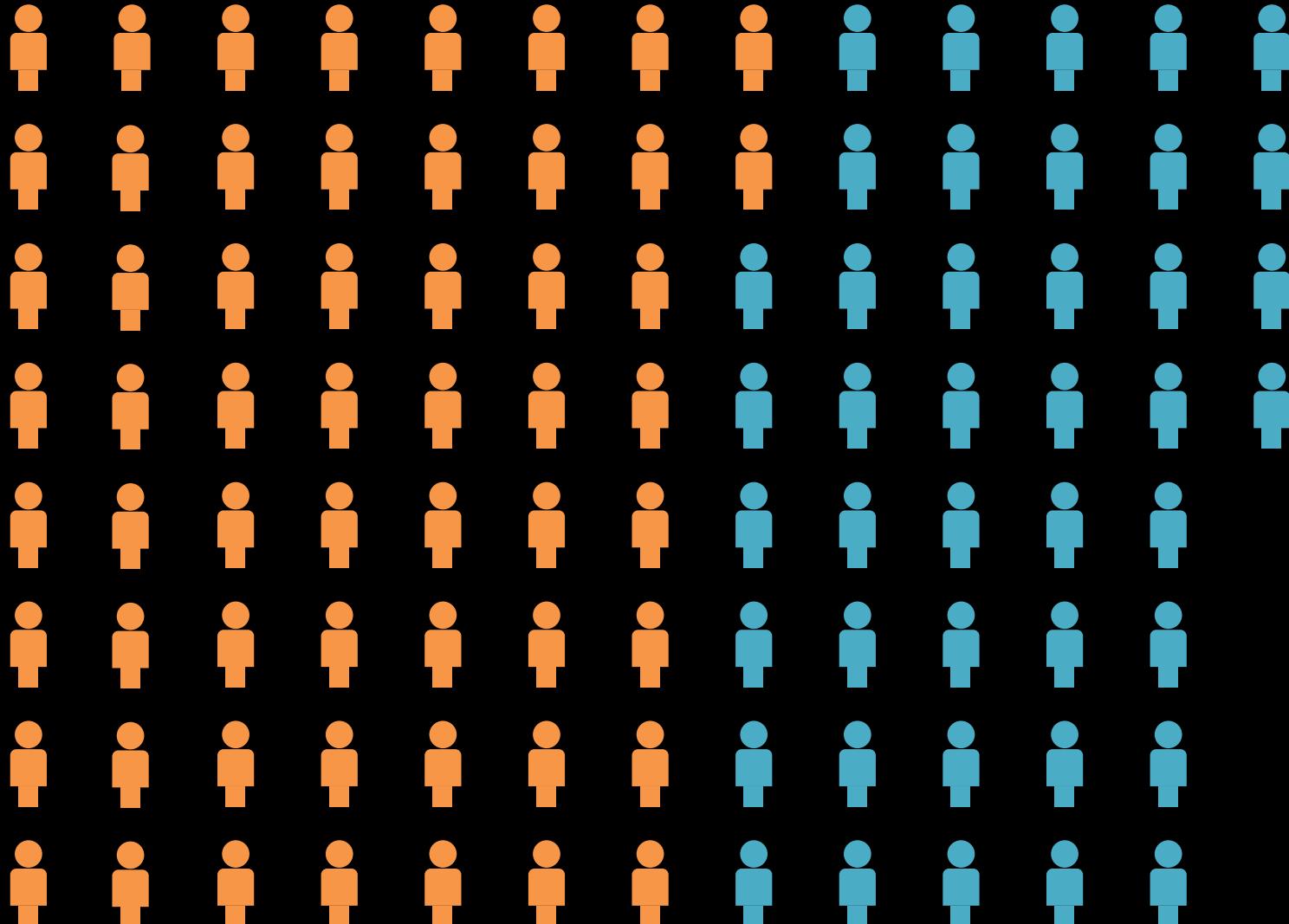
# US K-12 Students



= 500,000 learners

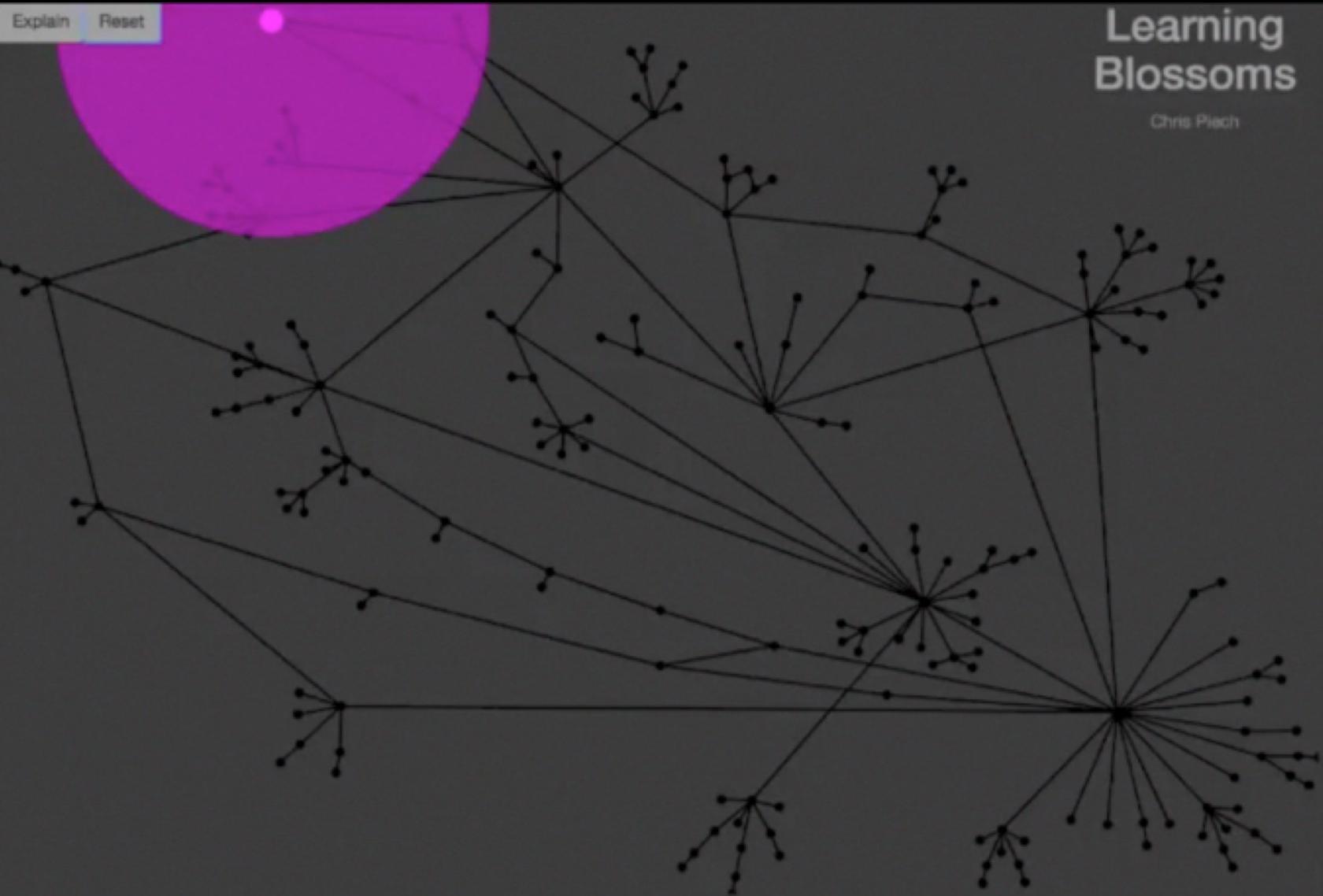


# Code.org Students

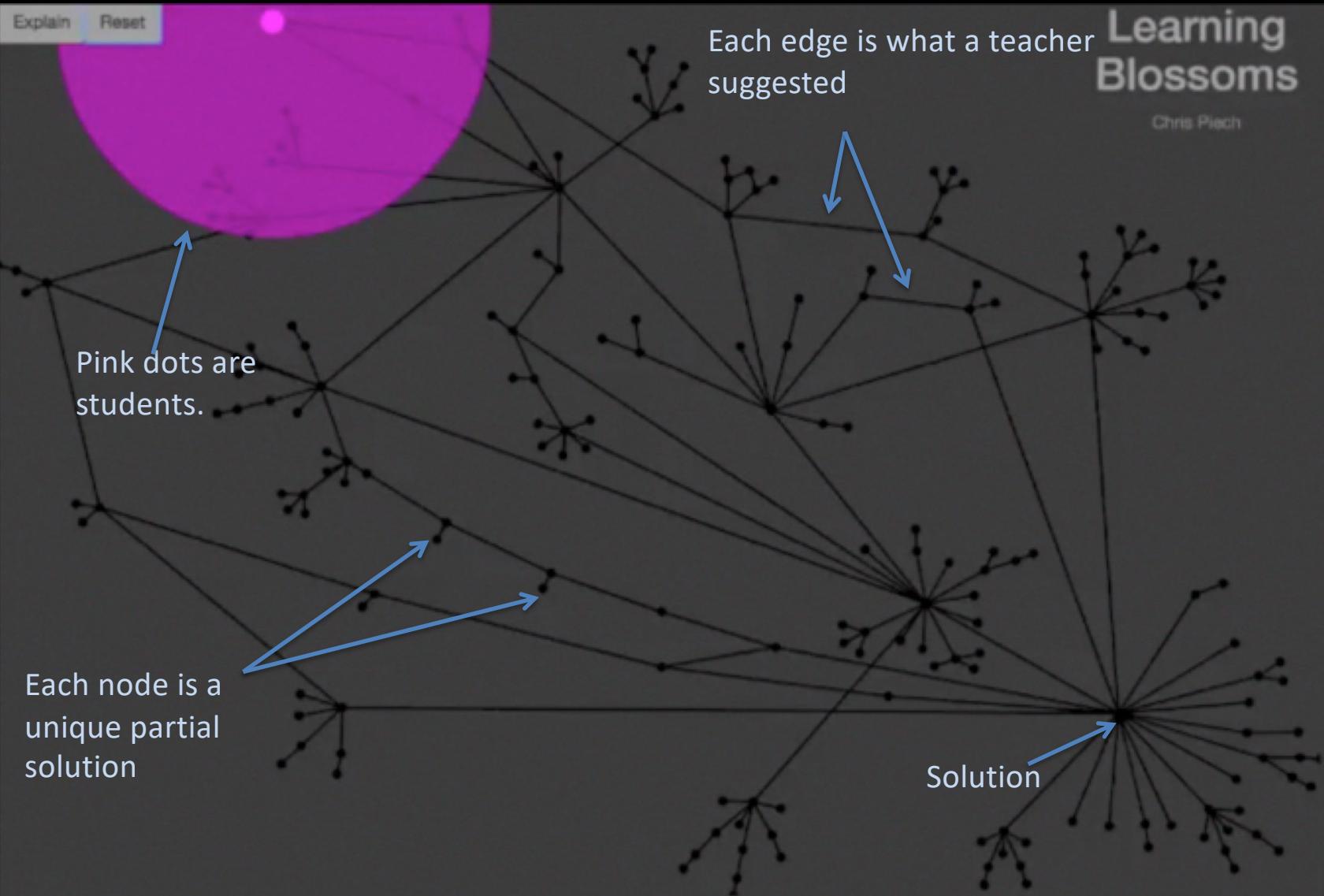


= 500,000 learners





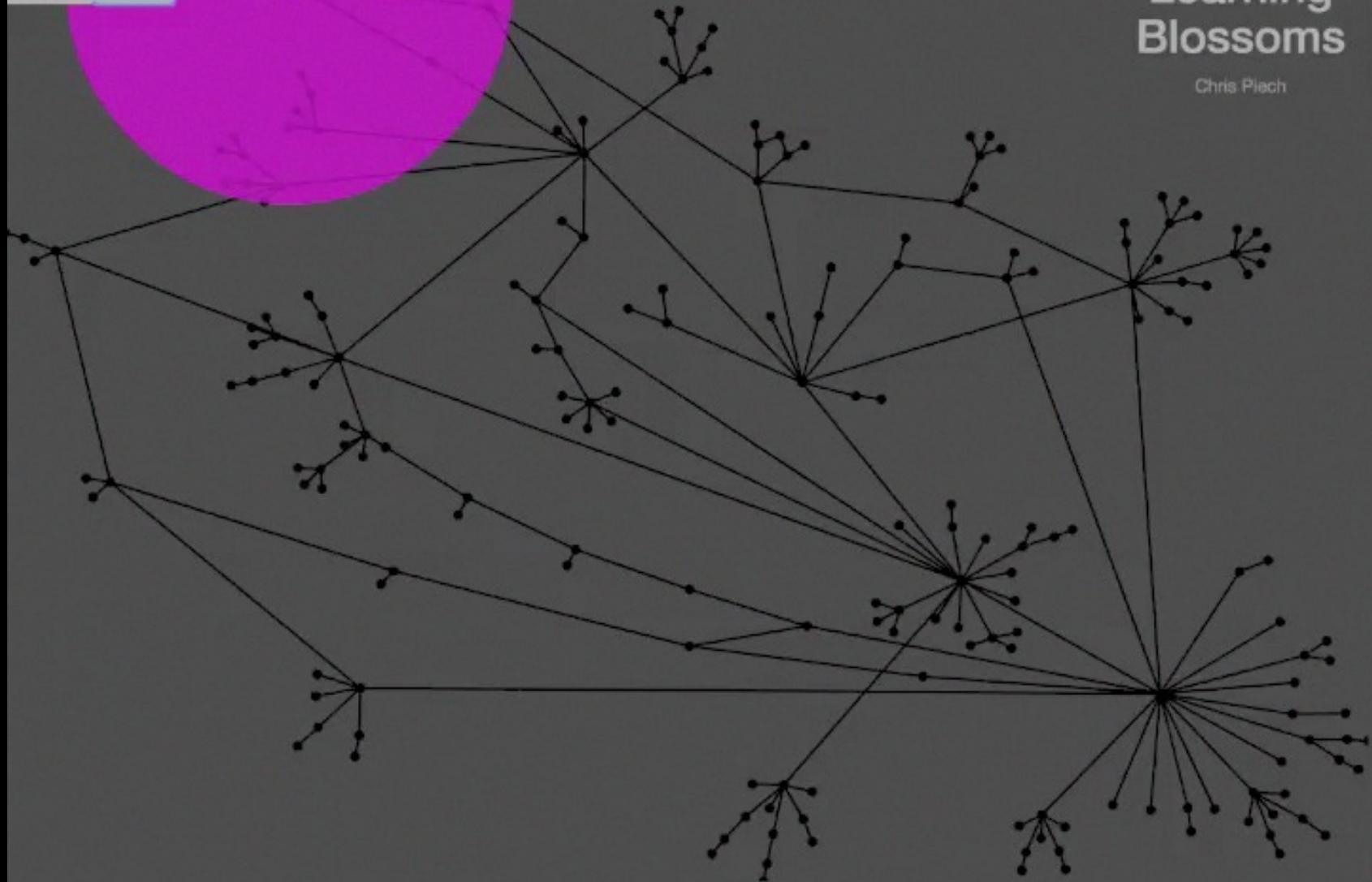
Autonomously Generating Hints by Inferring Problem Solving Policies - Piech, Sahami et al.



Explain Reset

# Learning Blossoms

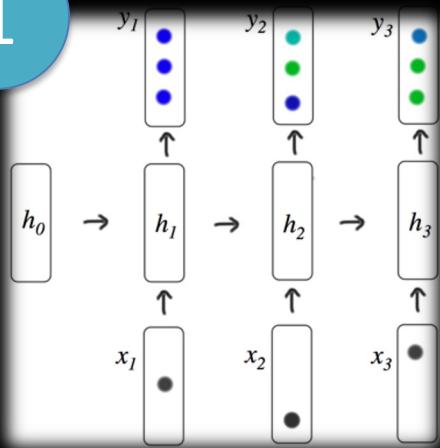
Chris Piech



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

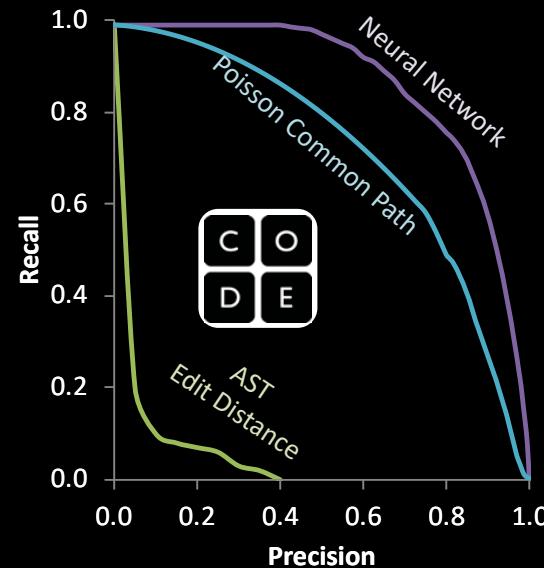
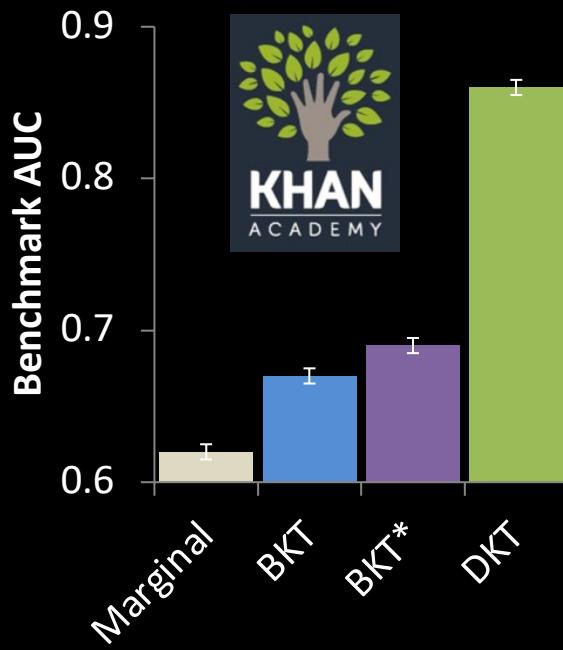
# Deep Learning Algorithms

1

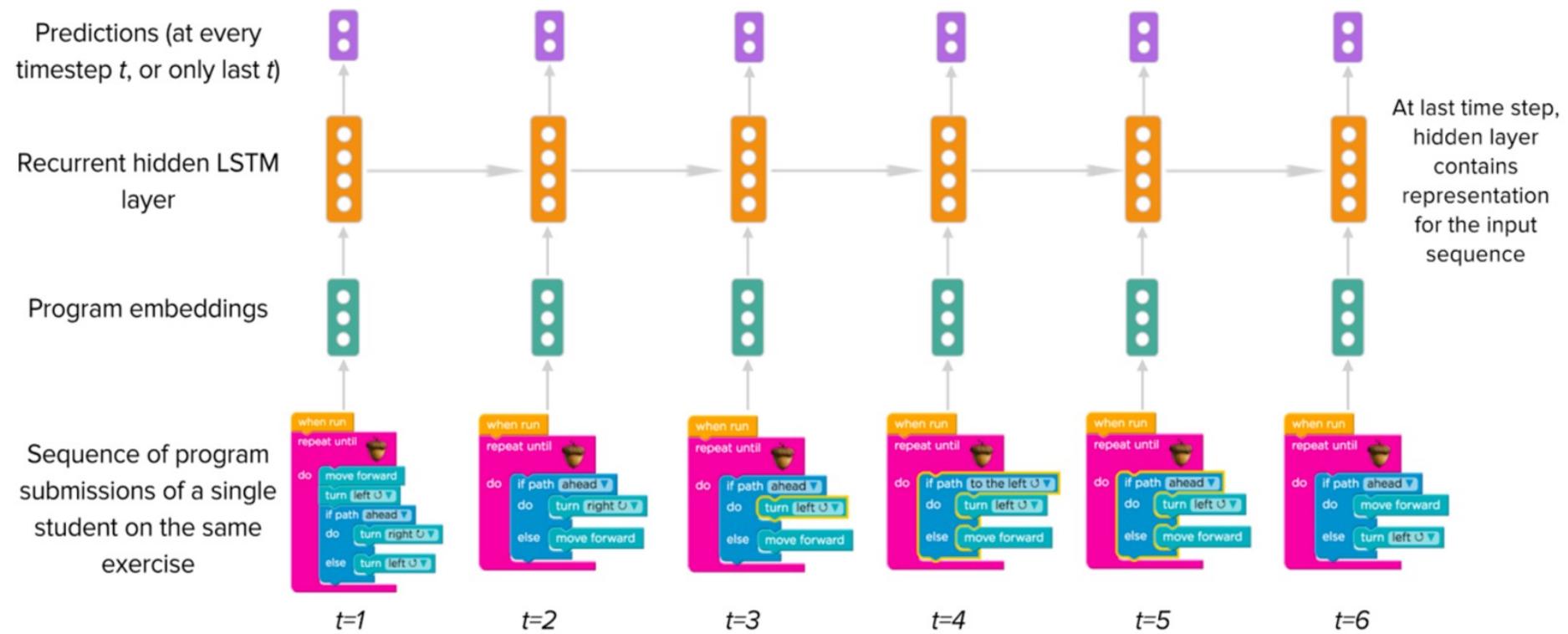


2

Program  $\rightarrow \mathbb{R}^n$



# Deep Learning on Trajectories

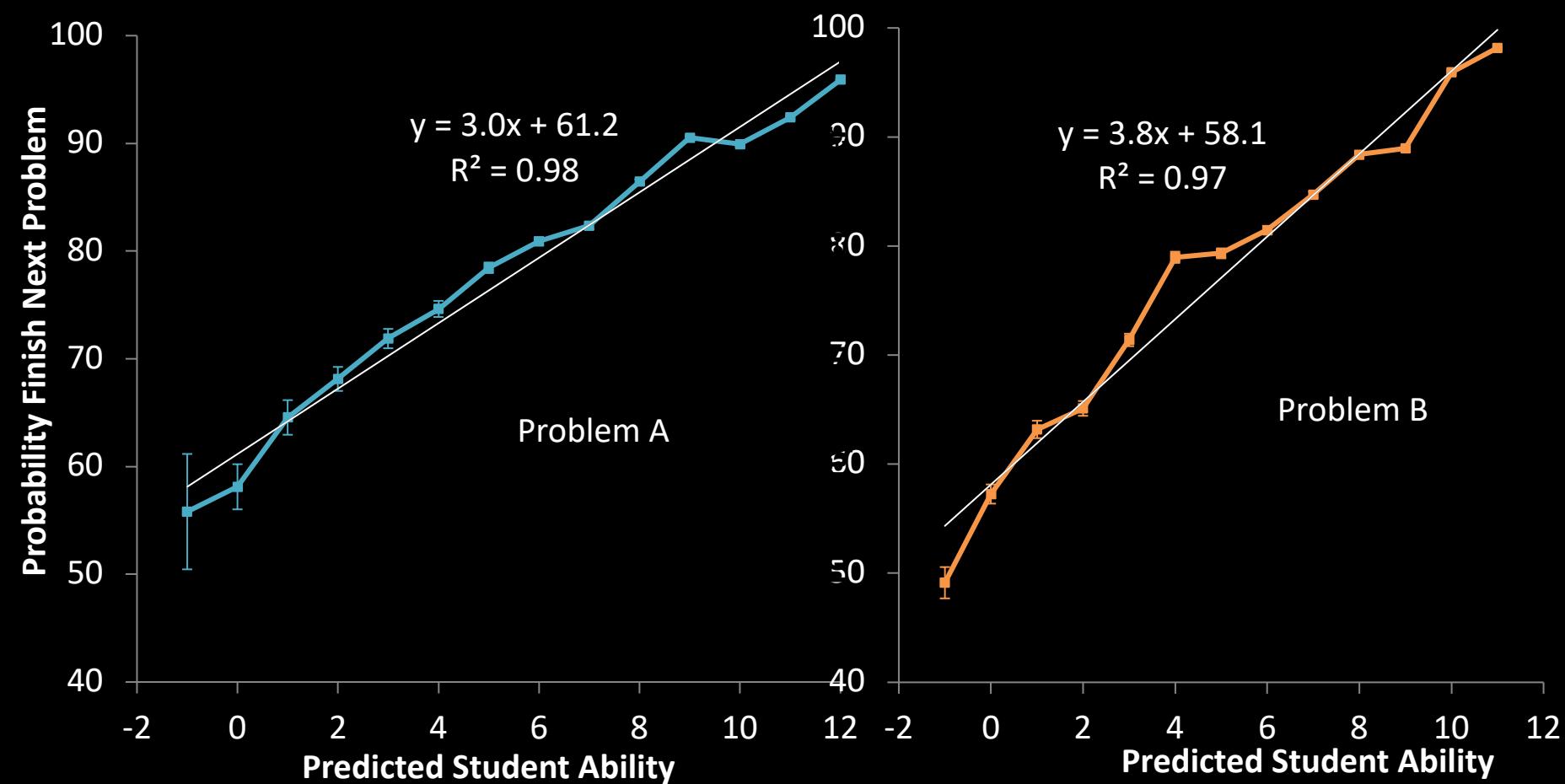


Research in collaboration with Lisa Wang

Piech + Sahami, CS106A, Stanford University



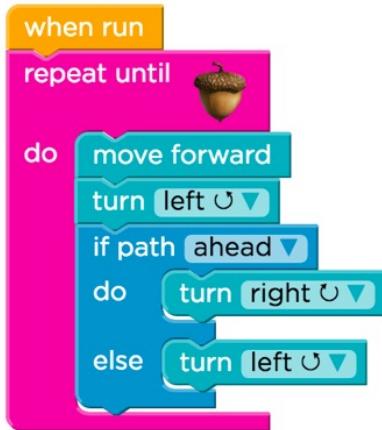
# Predicts Future Success



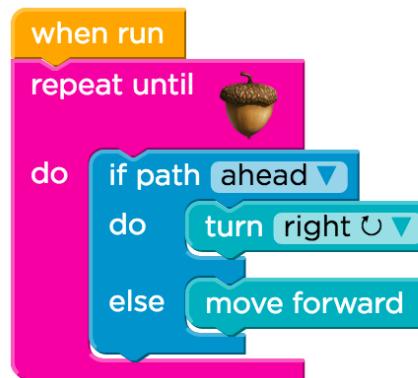
Most likely to succeed?

# Highly Rates Grit

1. Two compound errors



2. Solves first error



3. Starts reasonable attempt



4. Completes attempt



5. Backtracks



6. Finds solution



# Highly Rates Grit

1. Two compound errors



2. Solves first error



3. Starts reasonable attempt



4. Completes attempt



5. Backtracks

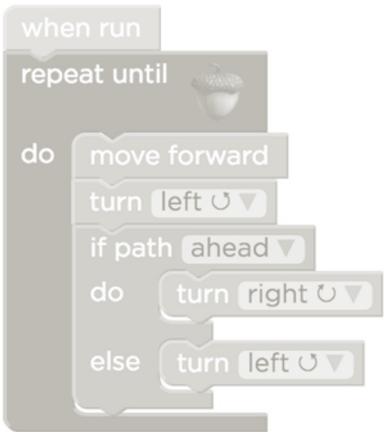


6. Finds solution

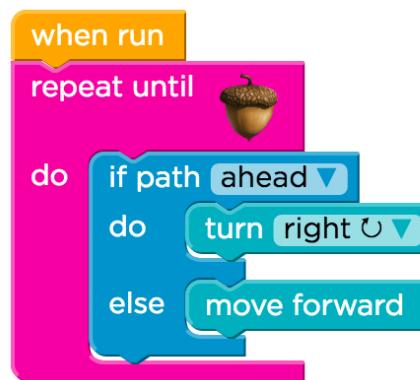


# Highly Rates Grit

1. Two compound errors



2. Solves first error



3. Starts reasonable attempt



4. Completes attempt



5. Backtracks

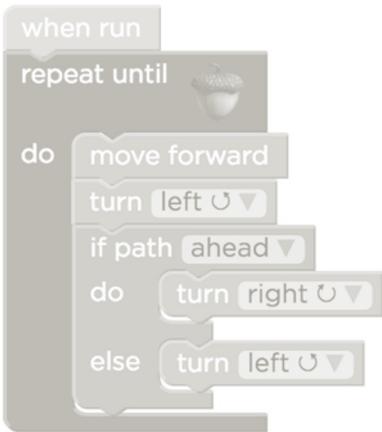


6. Finds solution

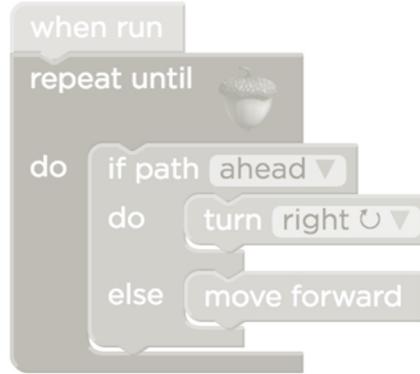


# Highly Rates Grit

1. Two compound errors



2. Solves first error



3. Starts reasonable attempt



4. Completes attempt



5. Backtracks

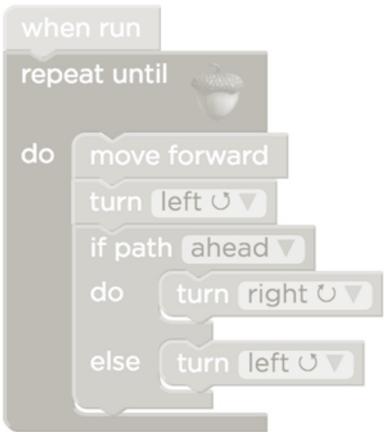


6. Finds solution



# Highly Rates Grit

1. Two compound errors



2. Solves first error



3. Starts reasonable attempt



4. Completes attempt



5. Backtracks



6. Finds solution



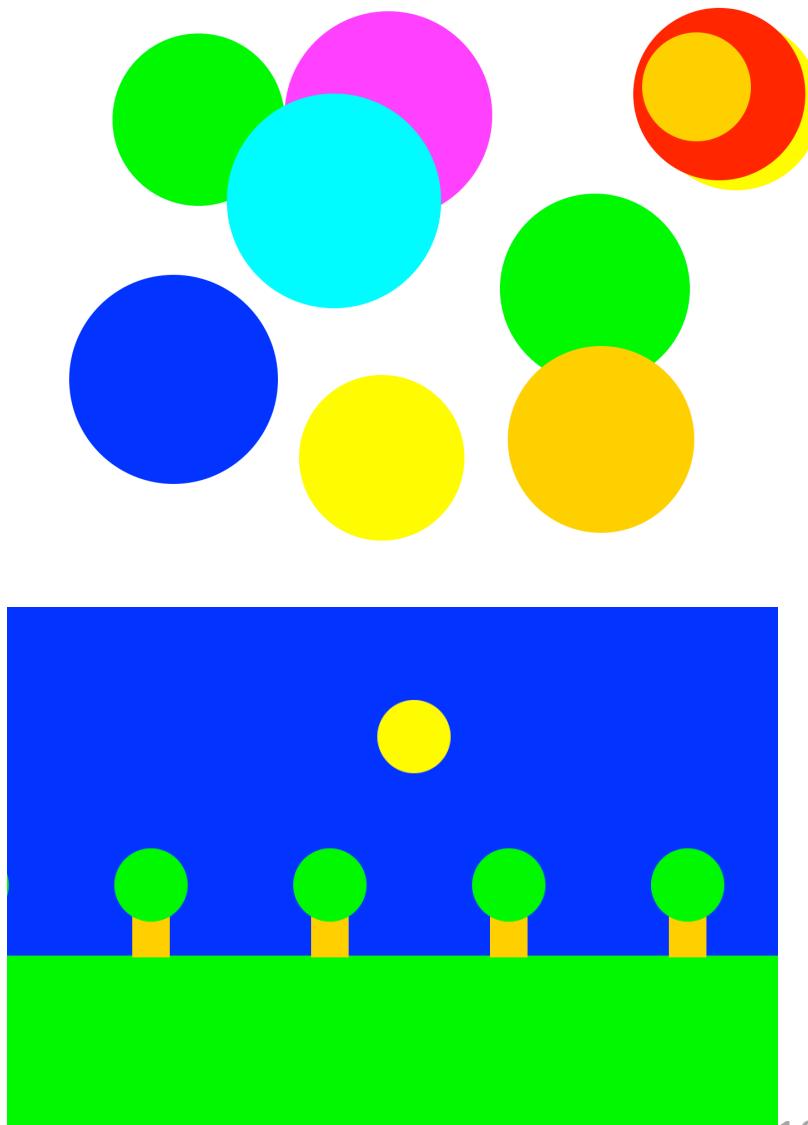
Coding is a new way to think  
Keep challenging yourself.

# Today's Goal

1. How do I draw things?



# Graphics Programs



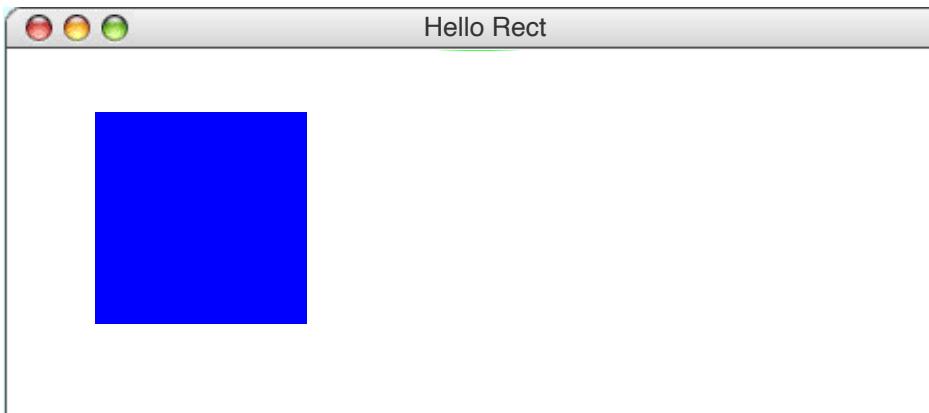
Plecn + Sanami, CS106A, Stanford University



# Draw a Rectangle

the following `main` method displays a blue square

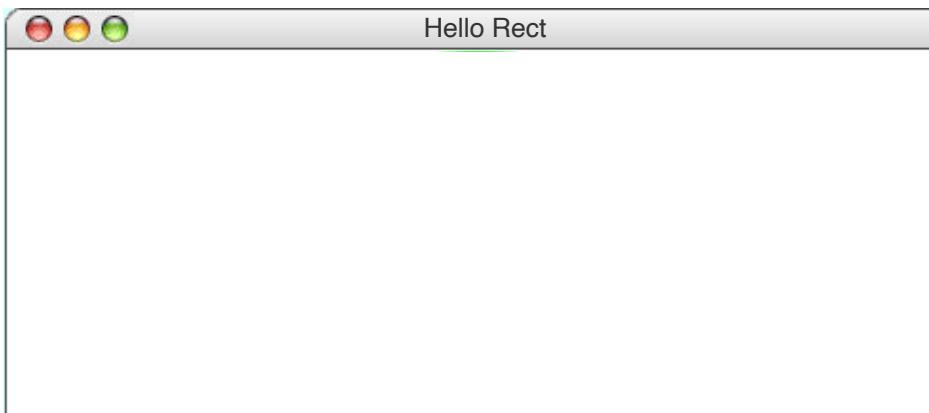
```
def main():
    canvas = Canvas(800, 200)
    canvas.create_rectangle(20, 20, 100, 100, "blue")
```



# Draw a Rectangle

the following `main` method displays a blue square

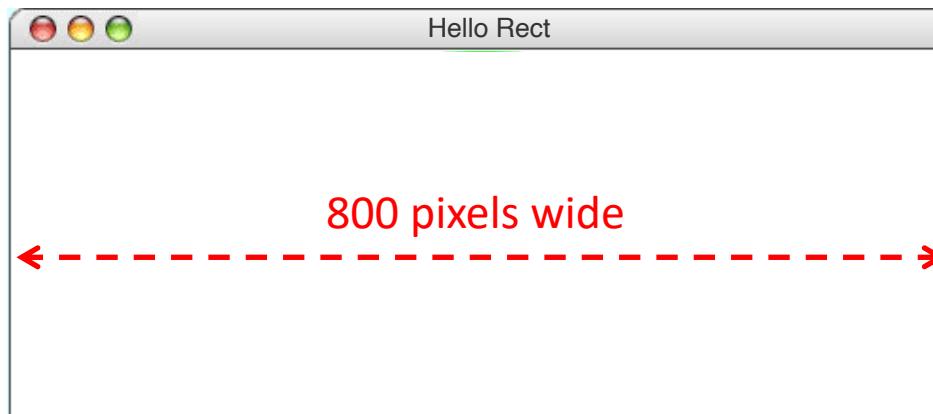
```
def main():
    canvas = Canvas(800, 200)
```



# Draw a Rectangle

the following `main` method displays a blue square

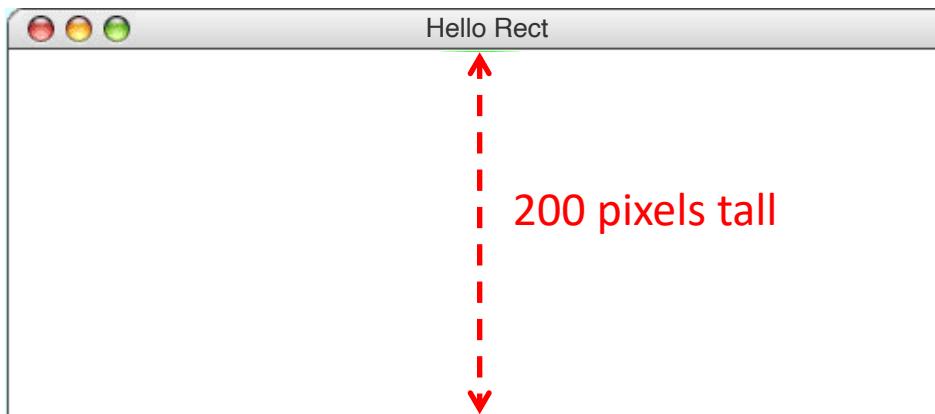
```
def main():
    canvas = Canvas(800, 200)
```



# Draw a Rectangle

the following `main` method displays a blue square

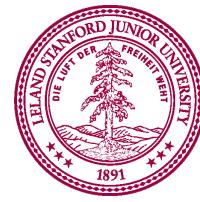
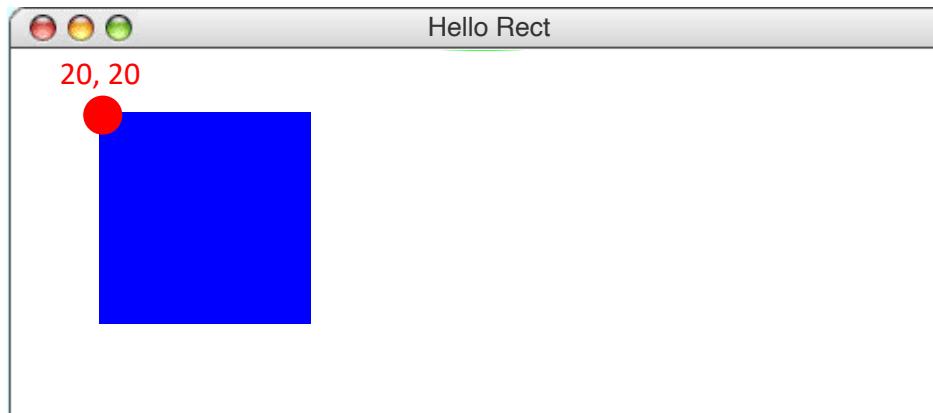
```
def main():
    canvas = Canvas(800, 200)
```



# Draw a Rectangle

the following `main` method displays a blue square

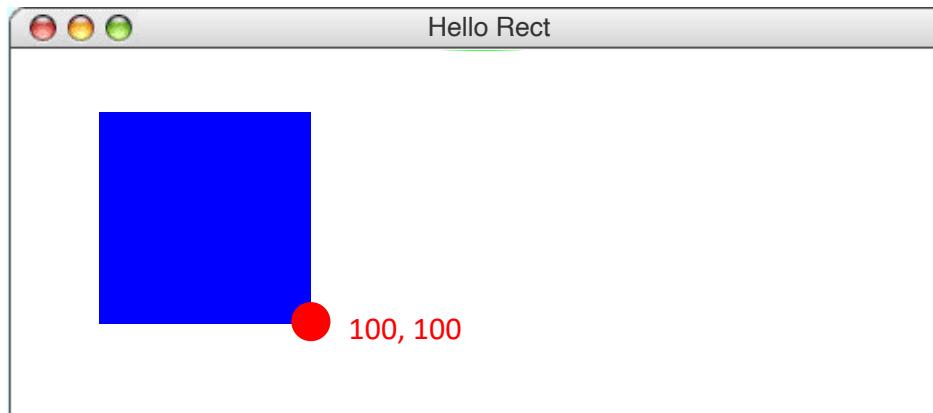
```
def main():
    canvas = Canvas(800, 200)
    canvas.create_rectangle(20, 20, 100, 100, "blue")
```



# Draw a Rectangle

the following `main` method displays a blue square

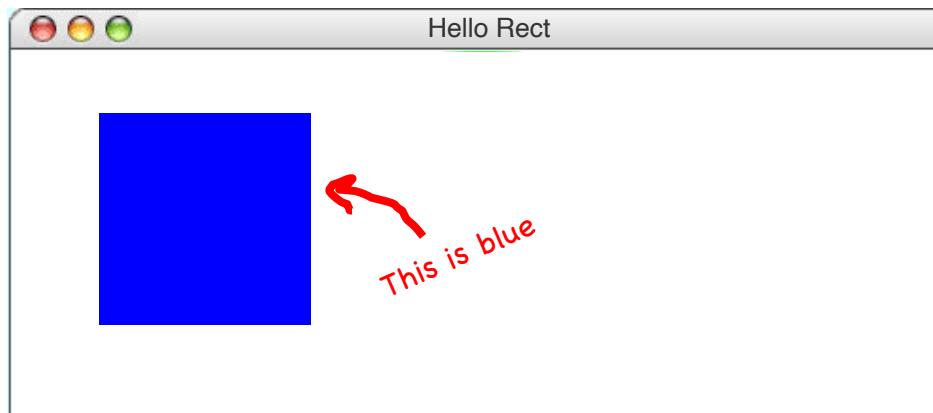
```
def main():
    canvas = Canvas(800, 200)
    canvas.create_rectangle(20, 20, 100, 100, "blue")
```



# Draw a Rectangle

the following `main` method displays a blue square

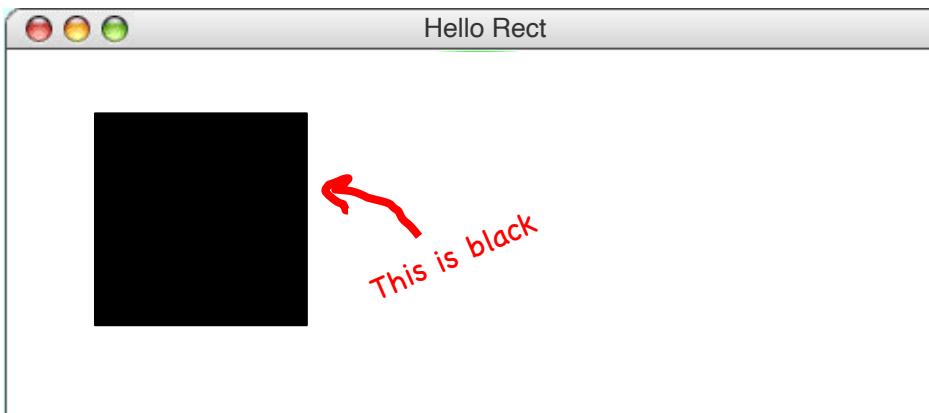
```
def main():
    canvas = Canvas(800, 200)
    canvas.create_rectangle(20, 20, 100, 100, "blue")
```



# Draw a Rectangle

the following `main` method displays a blue square

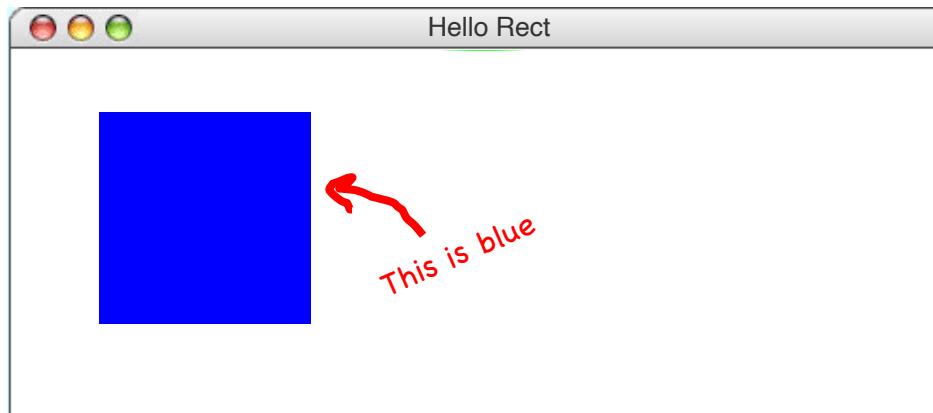
```
def main():
    canvas = Canvas(800, 200)
    canvas.create_rectangle(20, 20, 100, 100)
```



# Draw a Rectangle

the following `main` method displays a blue square

```
def main():
    canvas = Canvas(800, 200)
    canvas.create_rectangle(20, 20, 100, 100, "blue")
```



# TK Natural Graphics



# Graphics Coordinates

0,0

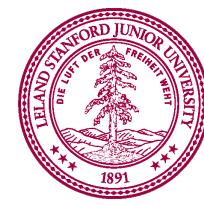
x 40,20

x 120,40

x 40,120

CANVAS\_WIDTH

CANVAS\_HEIGHT



# Rectangles, Ovals, Text

Code in Place    +

localhost:3000/cip3/share/vCPah8tdXp5x5249tU9S

## Programming is Awesome

Created by Chris Piech ❤️

This is a demo for class! Check it out. Graphics is fun.

Stop    Editable (Owner)

Programming is Awesome!!!

Copy shareable link!



# Rectangles, Ovals, Text

Code in Place

localhost:3000/cip3/ide/a/programmingisawesome

IDE | Programming is Awesome

Project Files

- Programming is Awesome
  - main.py
  - simba.png

main.py

```
1 from graphics import Canvas
2 import time
3
4 CANVAS_WIDTH = 500
5 CANVAS_HEIGHT = 500
6
7 def main():
8     canvas = Canvas(CANVAS_WIDTH, CANVAS_HEIGHT)
9
10    # a line for good measure!
11    canvas.create_line(0, 0, 500, 500)
12
13    # a blue square with width and height = 80
14    canvas.create_rectangle(70, 70, 150, 150, "blue")
15
16    # a red oval and a rectangle at the exact same spot!
17    canvas.create_rectangle(250, 150, 500, 500)
18    canvas.create_oval(250, 150, 500, 500, "red")
19
20    # images require the pillow library
21    canvas.create_image(0, 200, "simba.png")
22
23    # some text is written on the screen.
24    canvas.create_text(50, 20,
25                      "Programming is Awesome!!!",
26                      color="blue",
27                      font="Courier",
28                      font_size=28)
29
30    # dude, where's my rect?
31    canvas.create_rectangle(0, 800, 10, 810)
32
33    # this is just for demo purposes
34    show_mouse_position(canvas)
```

Canvas

The canvas displays several graphical elements: a blue square at (70, 70, 150, 150), a red oval at (250, 150, 500, 500) overlapping a small black rectangle, a photograph of a brown dog (Simba) at the bottom left, and the text "Programming is Awesome!!!" in blue Courier font at (50, 20). A line connects the text to the blue square.

Terminal

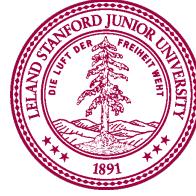
```
x = 17.015625, y = 7.1953125
KeyboardInterrupt
%
```



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

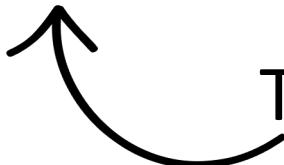


- `canvas.create_line(x1, y1, x2, y2)`
- `canvas.create_oval()`
- `canvas.create_text()`



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

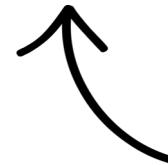
- `canvas.create_oval()`
- `canvas.create_text()`



The first point of the  
line is `(x1, y1)`



- `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)**

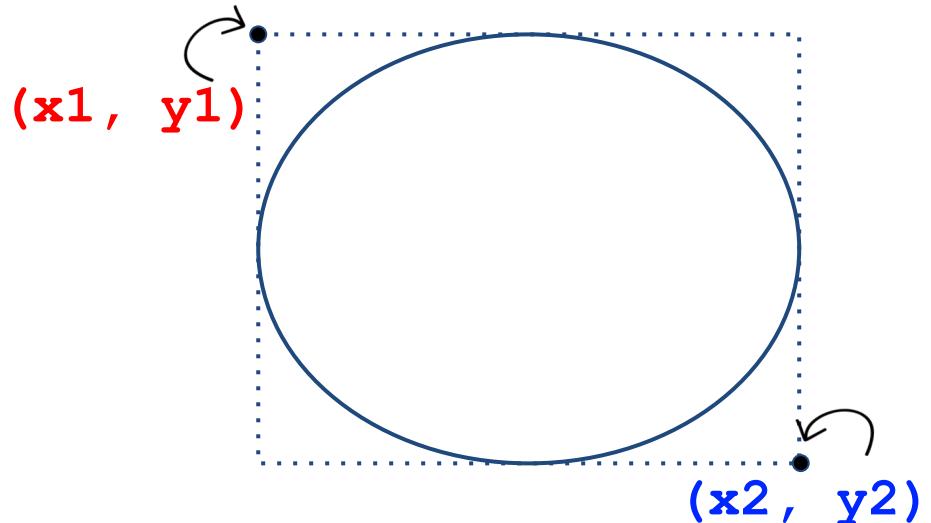
- **canvas.create\_oval()**
- **canvas.create\_text()**



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



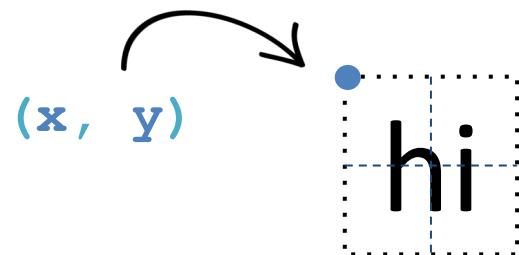
- `canvas.create_line()`
- `canvas.create_oval(x1, y1, x2, y2)`
- `canvas.create_text()`



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

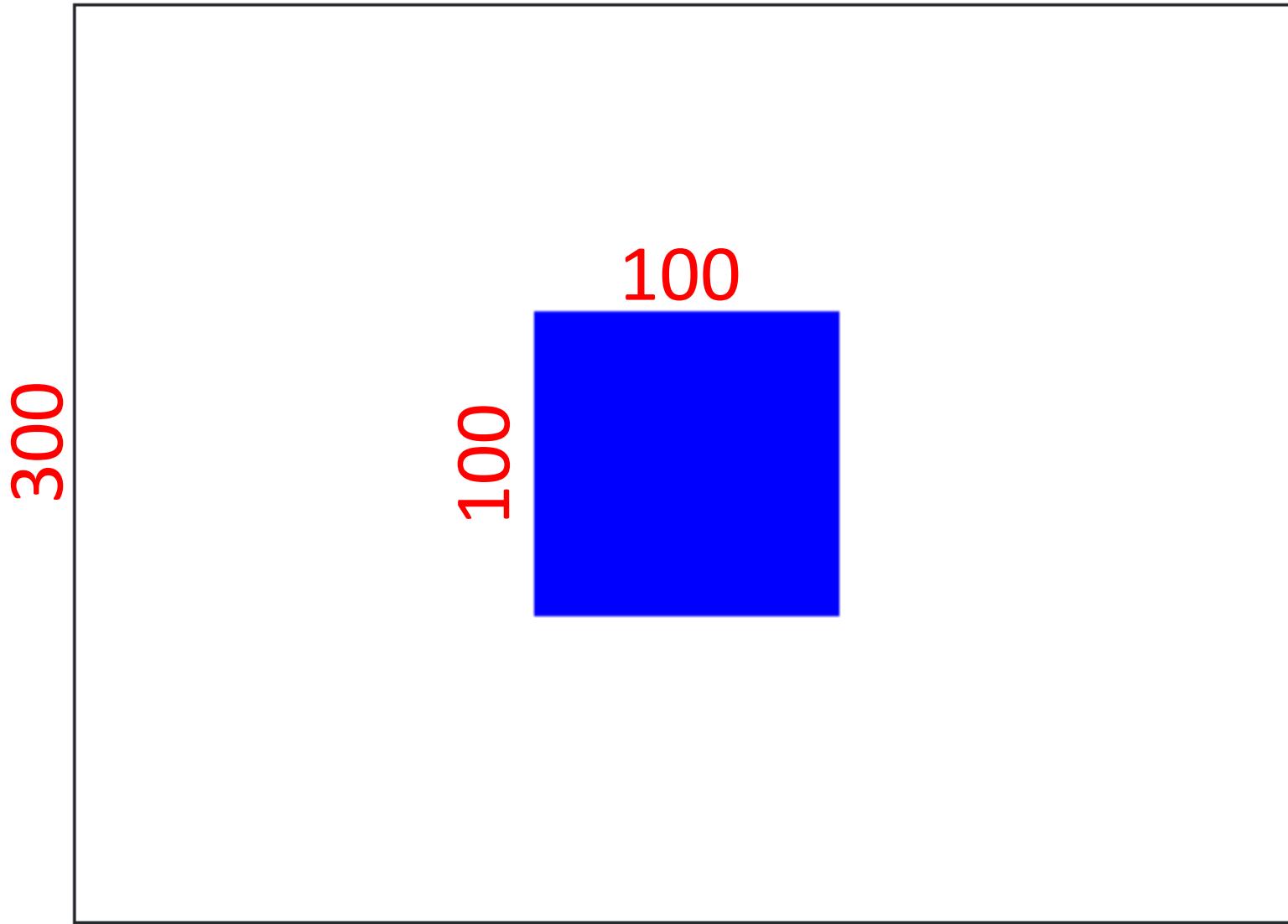


- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi')`



# Centered Square

400



# Centered Square

Code in Place

Code in Place

Code in Place

localhost:3000/cip3/ide/a/centeredsquare

Run

Share

## IDE | Centered Square

### Instructions

Write a program that draws a square, 200 pixels wide, by 200 pixels high, right in the center of the canvas. Warning: this example involves a little math. But it is math worth knowing!

This program is hard, because the first attempt that most students try does not work. It is tempting to try to place the square in the middle by setting the first two parameters of `create_rectangle` to be `middle_x` and `middle_y`:

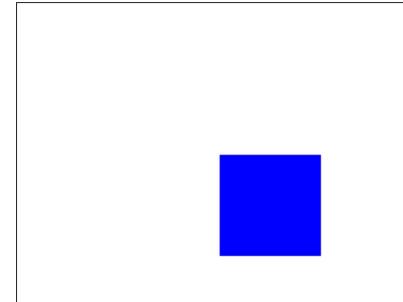
```
middle_x = CANVAS_WIDTH/2
middle_y = CANVAS_HEIGHT/2

# calculate the right and bottom of the
# square
right_x = middle_x + SQUARE_SIZE
bottom_y = middle_y + SQUARE_SIZE

# draw the square
canvas.create_rectangle(middle_x,
middle_y, right_x, bottom_y, 'blue')
```

However, instead of putting the square in the middle, this puts the top left corner of the square in the middle of the canvas.

Canvas >



```
from graphics import Canvas
CANVAS_WIDTH = 400
CANVAS_HEIGHT = 300
SQUARE_SIZE = 100

def main():
    canvas = Canvas(CANVAS_WIDTH, CANVAS_HEIGHT)

    # get the middle of the canvas
    middle_x = CANVAS_WIDTH/2
    middle_y = CANVAS_HEIGHT/2

    # calculate the top left of the square
    left_x = middle_x
    top_y = middle_y

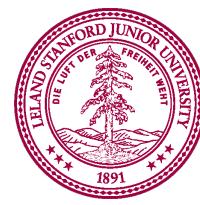
    # calculate the right and bottom of the square
    right_x = left_x + SQUARE_SIZE
    bottom_y = top_y + SQUARE_SIZE

    # draw the square
    canvas.create_rectangle(left_x, top_y, right_x, bottom_y, 'blue')
    print("Done!")

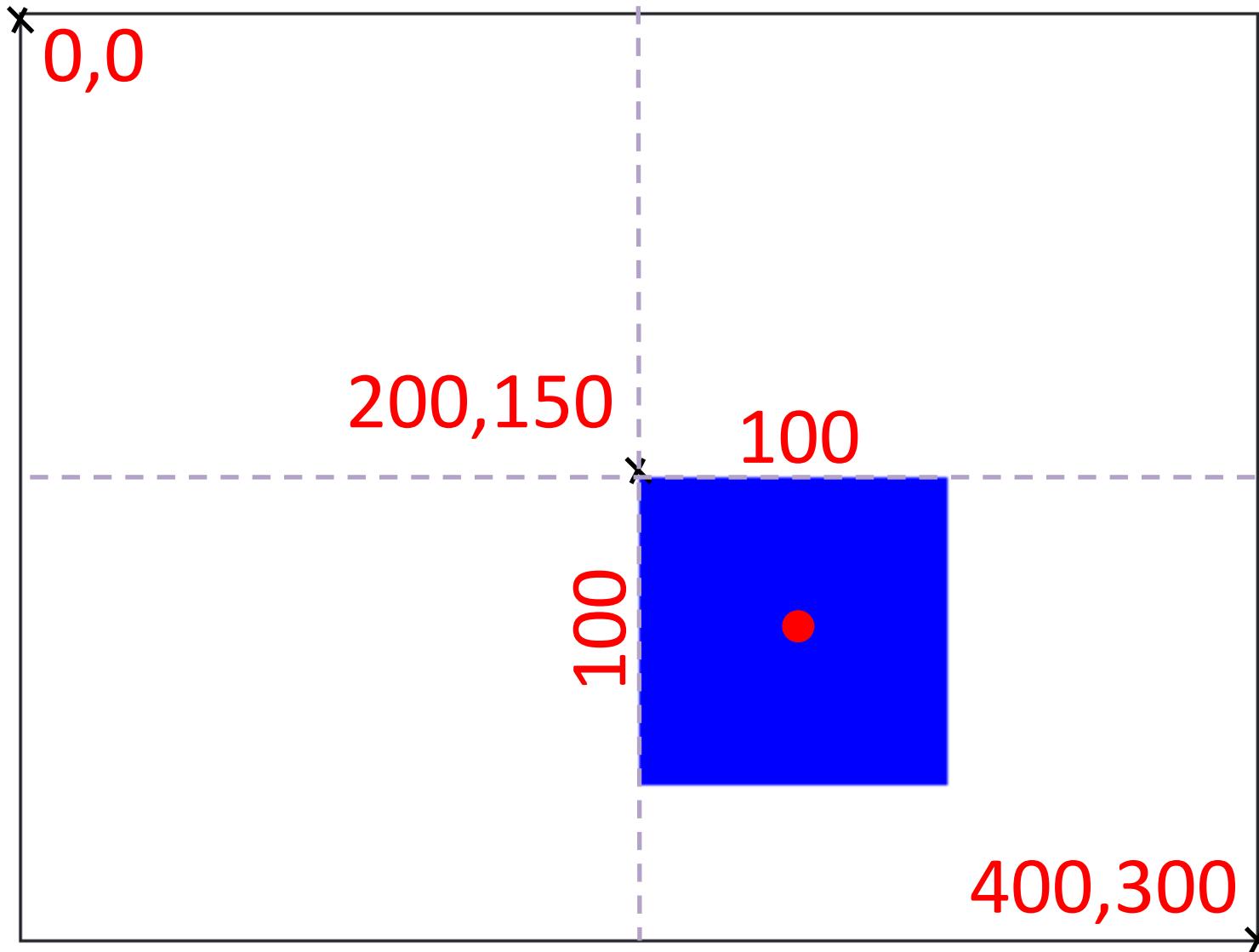
if __name__ == '__main__':
    main()
```

Terminal Error message type: Standard Error Message

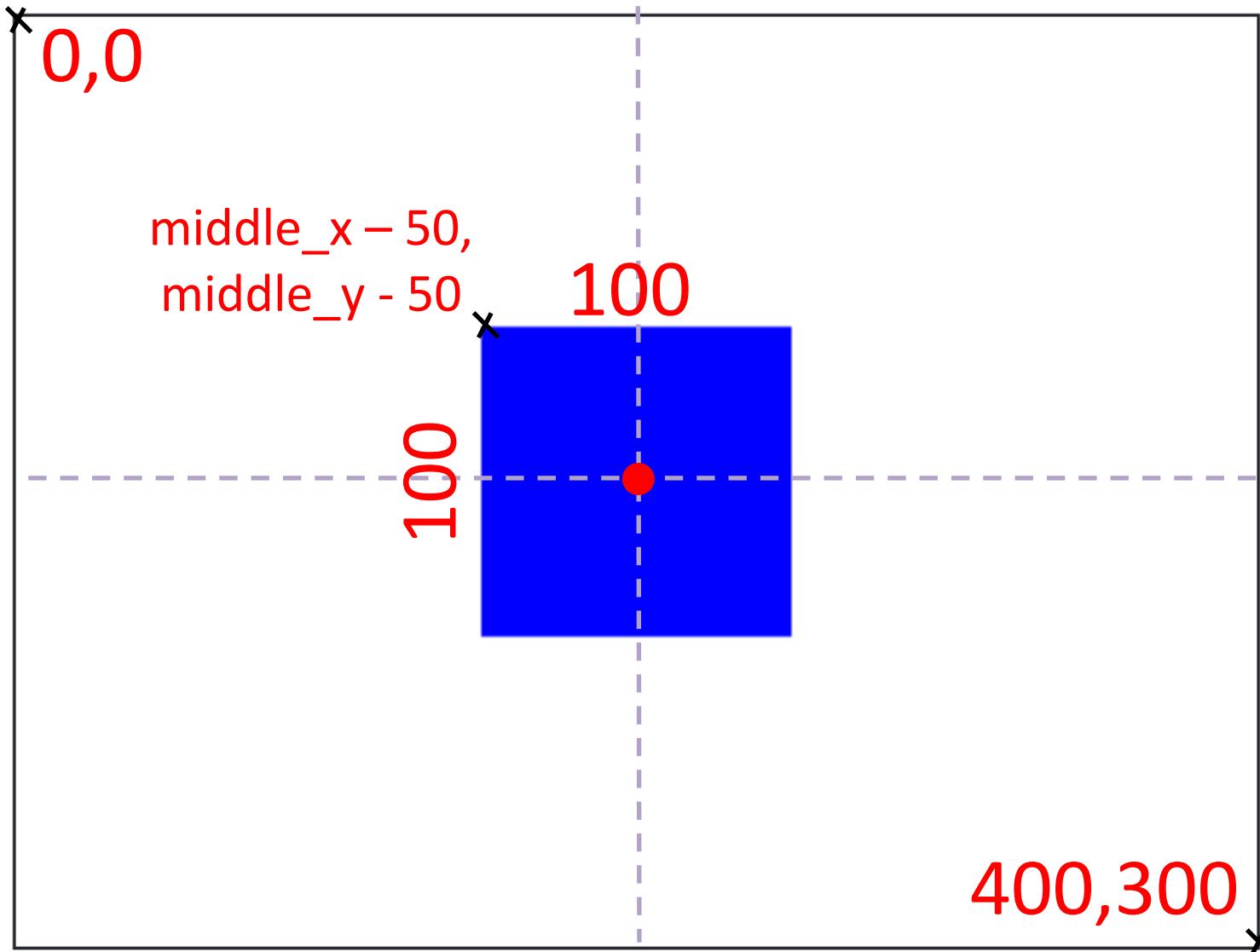
```
Done!
% python main.py
Done!
% python main.py
Done!
%
% python main.py
Done!
% []
```



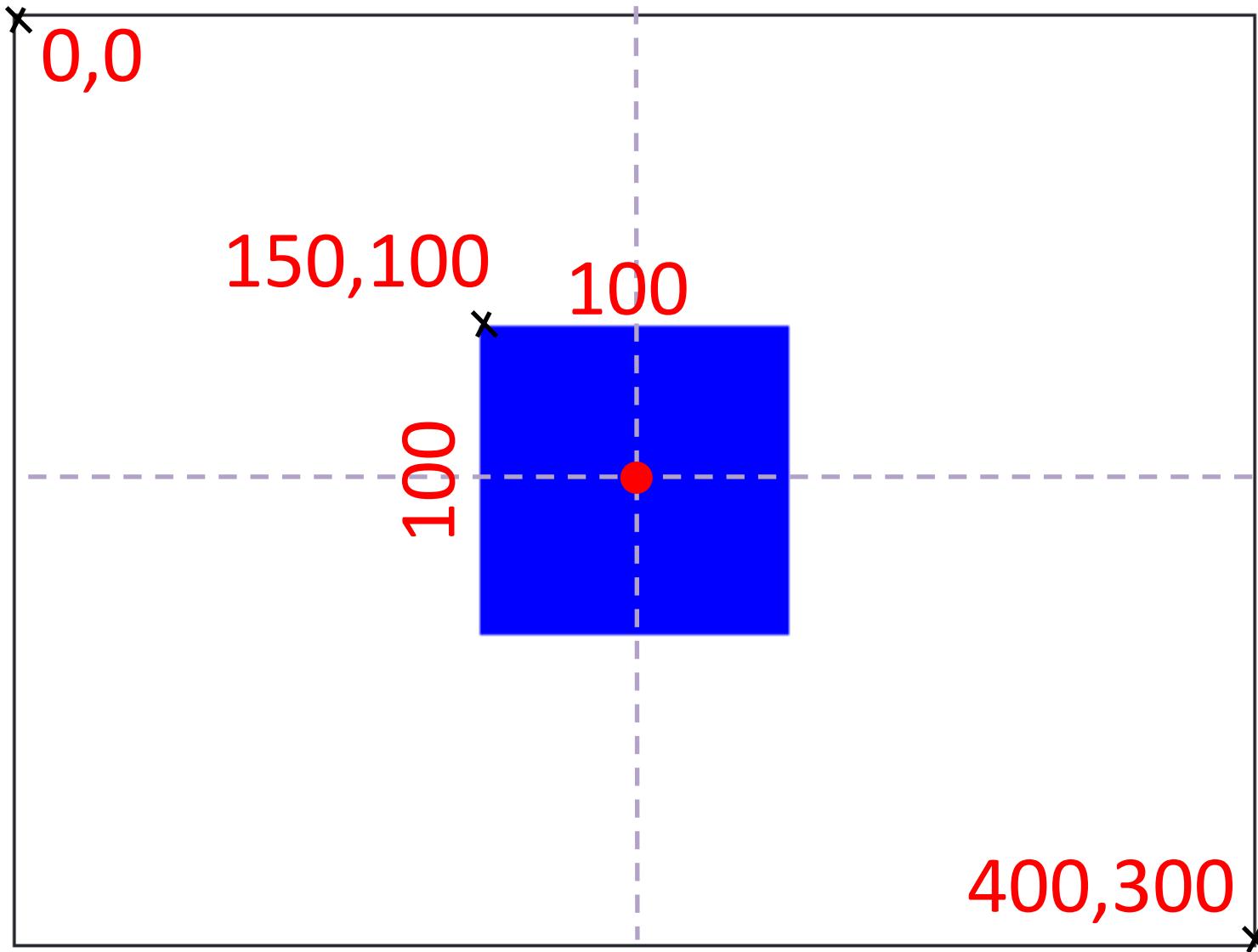
# Centered Square



# Centered Square



# Centered Square



# Centered Square

Code in Place

Code in Place

Code in Place

localhost:3000/cip3/ide/a/centeredsquare

IDE | Centered Square

Instructions

Write a program that draws a square, 200 pixels wide, by 200 pixels high, right in the center of the canvas. Warning: this example involves a little math. But it is math worth knowing!

This program is hard, because the first attempt that most students try does not work. It is tempting to try to place the square in the middle by setting the first two parameters of create\_rectangle to be middle\_x and middle\_y:

```
middle_x = CANVAS_WIDTH/2
middle_y = CANVAS_HEIGHT/2

# calculate the right and bottom of the
# square
right_x = middle_x + SQUARE_SIZE
bottom_y = middle_y + SQUARE_SIZE

# draw the square
canvas.create_rectangle(middle_x,
    middle_y, right_x, bottom_y, 'blue')
```

However, instead of putting the square in the middle, this puts the top left corner of the square in the middle of the canvas.

Run

Canvas

main.py

```
1  from graphics import Canvas
2
3  CANVAS_WIDTH = 400
4  CANVAS_HEIGHT = 300
5  SQUARE_SIZE = 100
6
7  def main():
8      canvas = Canvas(CANVAS_WIDTH, CANVAS_HEIGHT)
9
10     # get the middle of the canvas
11     middle_x = CANVAS_WIDTH/2
12     middle_y = CANVAS_HEIGHT/2
13
14     # calculate the top left corner position
15     left_x = middle_x - SQUARE_SIZE/2
16     top_y = middle_y - SQUARE_SIZE/2
17
18     # calculate the right and bottom of the square
19     right_x = left_x + SQUARE_SIZE
20     bottom_y = top_y + SQUARE_SIZE
21
22     # draw the square
23     canvas.create_rectangle(left_x, top_y, right_x, bottom_y, 'blue')
24
25 if __name__ == '__main__':
26     main()
```

Terminal

```
% python main.py
% 
```



# Step by Step

Code in Place    Code in Place    Code in Place

localhost:3000/cip3/ide/a/rectline

IDE | Rect Line Replay

Run

Share

Replay Mode

Variable	Value
canvas	<Canvas>
i	23
left_x	220
top_y	220
right_x	230
bottom_y	230

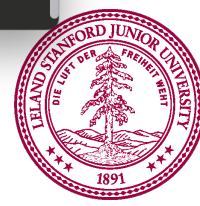
Replay Canvas >

main.py

```
1 import graphics
2
3 def main():
4     # makes a canvas
5     canvas = graphics.create_canvas(300, 300)
6     # make 20 some blue suares
7     for i in range(30):
8         # it can be helpful to store each coord into its own variable
9         left_x = i * 10
10        top_y = i * 10
11        right_x = left_x + 10
12        bottom_y = top_y + 10
13        canvas.create_rectangle(left_x,top_y,right_x,bottom_y, 'blue')
14
15        # keep track of i
16        print(i)
17
18 if __name__ == '__main__':
19     main()
```

Replay Terminal

```
13
14
15
16
17
18
19
20
21
22
```



# Pedagogy

Code in Place    X    Code in Place    X    +

codeinplace.stanford.edu/cip3/textbook/shapes

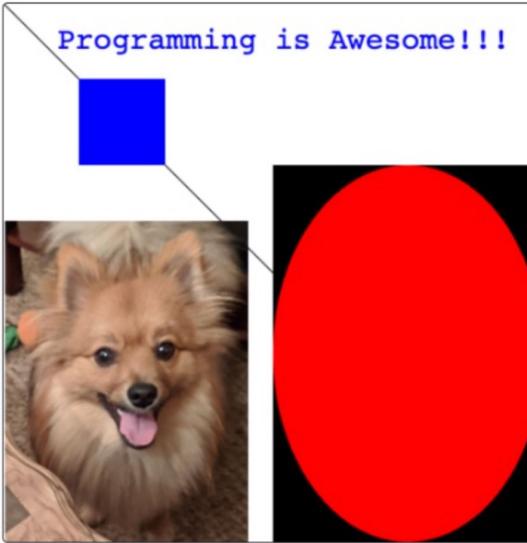
Conditionals  
Loops  
Nested Loops  
Random  
Nonetypes  
Advanced Arithmetic  
Floating Point  
Graphics  
Basic Shapes  
Animation  
Functions  
Anatomy of a Function  
Scope  
Parameters  
Return vs Print  
Multiple Returns  
Function Decomposition  
Update Functions

## Basic Shapes

**Introduction to Graphics**  
*Adapted from the Stanford CS106A Graphics Reference*

### Quest

Graphics!!!! Yep, it's time to give Python an upgrade! All the apps on your computer, everything from the calculator to the file explorer to the very browser you might be reading this in right now, are all programs that use graphics to make interactive, visual programs. You're about to get your first taste of graphics by learning how to draw shapes on the screen. Ready? By the end of this chapter you should be able to make programs that can draw!



Text vs Graphics

