Smith College
Computer Science

CSC111

Week 8 — Spring 2018

Dominique Thiébaut
dthiebaut@smith.edu

HACKATSMITH.COM

CODE. EAT. REPEAT.

H A C K   S M I T H

< APRIL 7-8TH, 2018 />

Outline

Boolean Operators

Exercises

If Statements and Graphics

Organization of a Graphics Program

Measuring Distances

Graphics: Obstacles

Eliza

Boolean Operators

Boolean Operators

And, Or, Not

__True__ and __False__ are Python values!

Boolean Operators

And, Or, Not

```
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement
```

```
         True                      True
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement
```

```
                    True
         True                     True
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement
```

```
                          True
            True                         True
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement
```

False

False **True**

```
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement
```

False

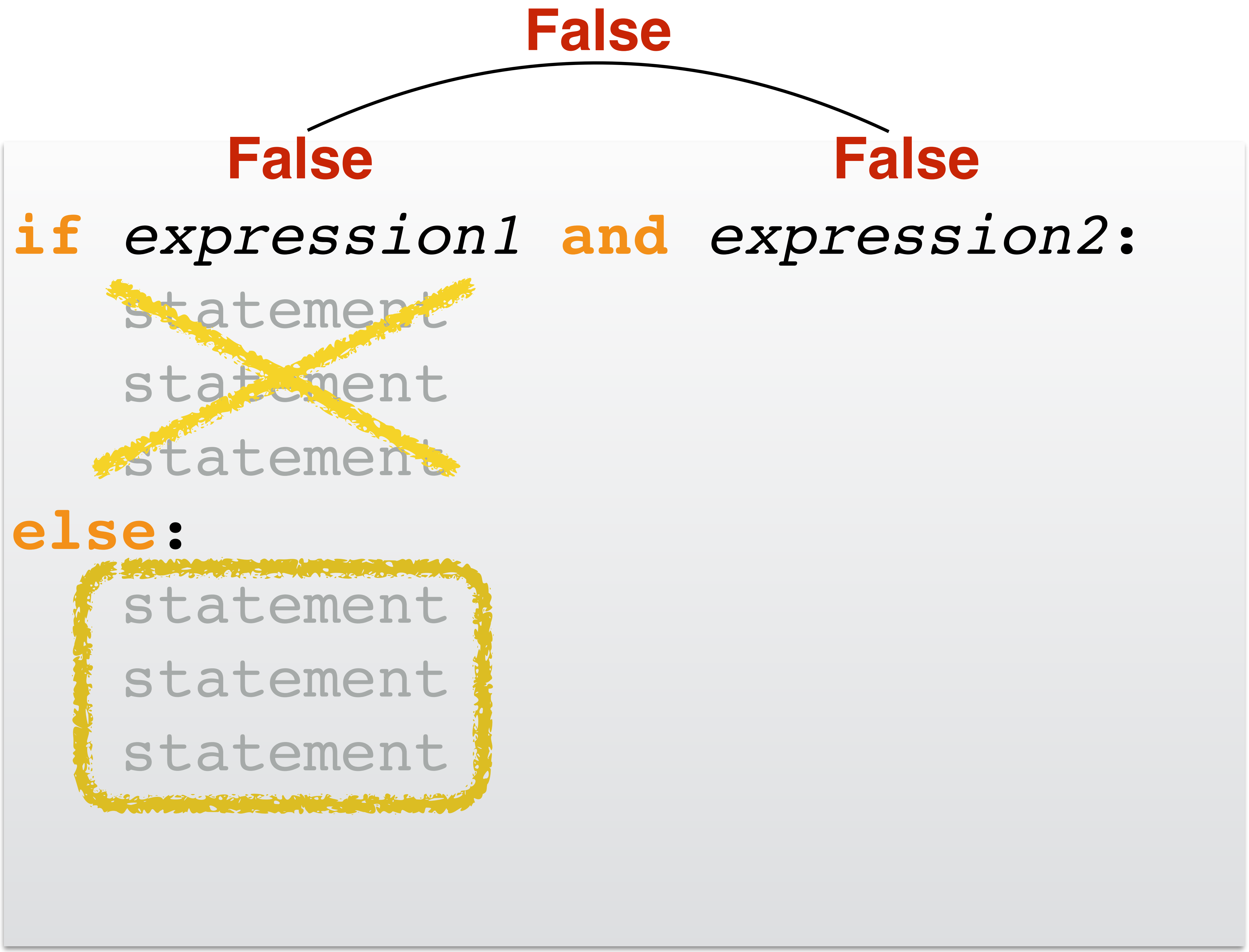
False True

```
if expression1 and expression2:
    ~~statement~~
    ~~statement~~
    ~~statement~~
else:
    statement
    statement
    statement
```

False

True **False**

```
if expression1 and expression2:
    ~~statement~~
    ~~statement~~
    ~~statement~~
else:
    statement
    statement
    statement
```



```
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement
```

And

```
if expression1 and expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement
```

True True

True False
False True
False False

Or

```
if expression1 or expression2:
    statement
    statement
    statement
else:
    statement
    statement
    statement
```

True	False
False	True
True	True

False	False

Not

```
if not expression:
    statement
    statement
    statement
else:
    statement
    statement
    statement
```

False

True

else is not always used...

```
if no20s == 1:
    print( no20s, "$20-bill" )
else:
    print( no20s, "$20-bills" )
```

else is not always used...

```
caption = "$20-bill"
if no20s != 1:
    caption = caption + "s"

print( no20s, caption )
```

Outline

Boolean Operators

Exercises

If Statements and Graphics

Organization of a Graphics Program

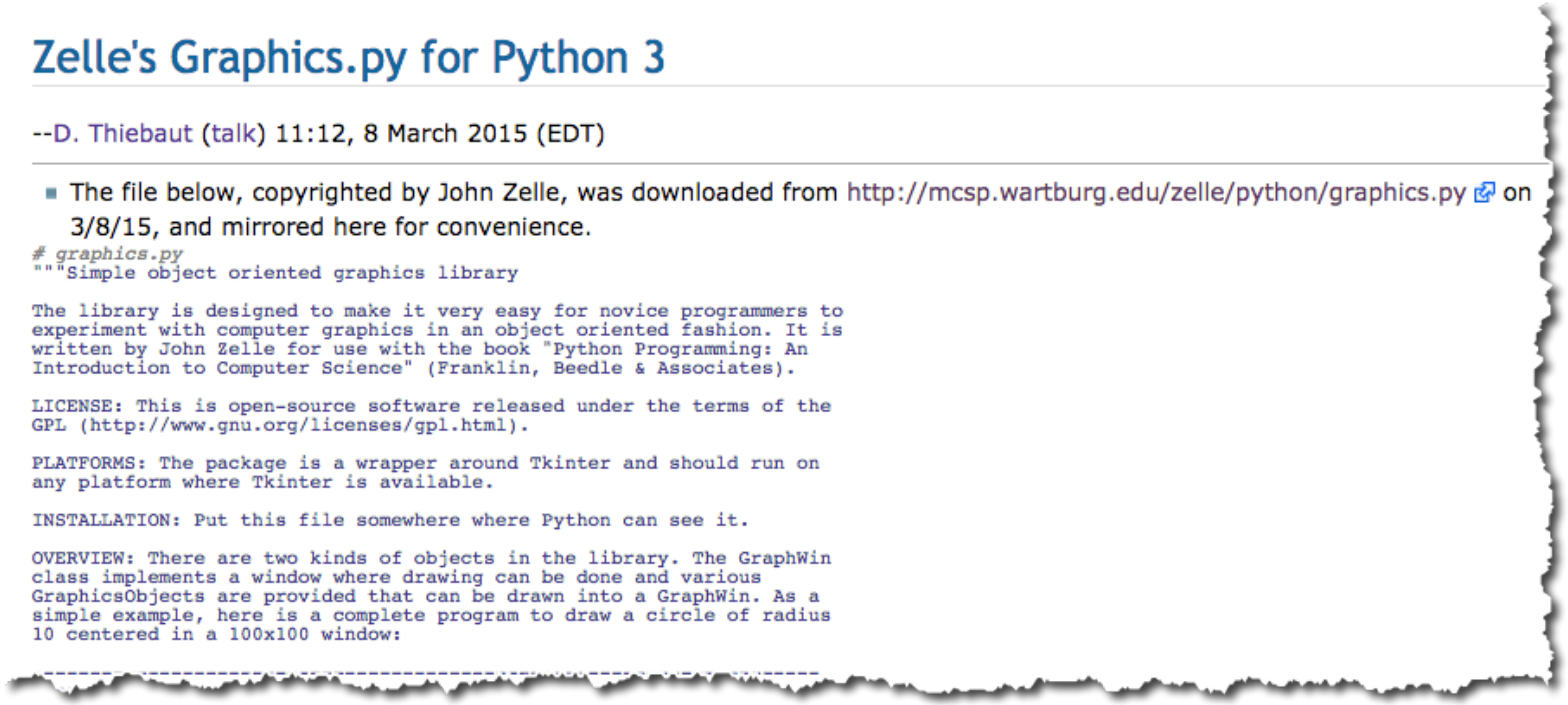
Measuring Distances

Graphics: Obstacles

Eliza

If-Statements and Graphics

Where are Graphic Objects Defined?



Zelle's Graphics.py for Python 3

--D. Thiebaut (talk) 11:12, 8 March 2015 (EDT)

- The file below, copyrighted by John Zelle, was downloaded from http://mcsp.wartburg.edu/zelle/python/graphics.py on 3/8/15, and mirrored here for convenience.

```
# graphics.py
"""Simple object oriented graphics library

The library is designed to make it very easy for novice programmers to
experiment with computer graphics in an object oriented fashion. It is
written by John Zelle for use with the book "Python Programming: An
Introduction to Computer Science" (Franklin, Beedle & Associates).

LICENSE: This is open-source software released under the terms of the
GPL (http://www.gnu.org/licenses/gpl.html).

PLATFORMS: The package is a wrapper around Tkinter and should run on
any platform where Tkinter is available.

INSTALLATION: Put this file somewhere where Python can see it.

OVERVIEW: There are two kinds of objects in the library. The GraphWin
class implements a window where drawing can be done and various
GraphicsObjects are provided that can be drawn into a GraphWin. As a
simple example, here is a complete program to draw a circle of radius
10 centered in a 100x100 window:
```

http://cs.smith.edu/dftwiki/index.php/Zelle%27s_Graphics.py_for_Python_3

Every element is an OBJECT

Examples of If-Statements in Graphics

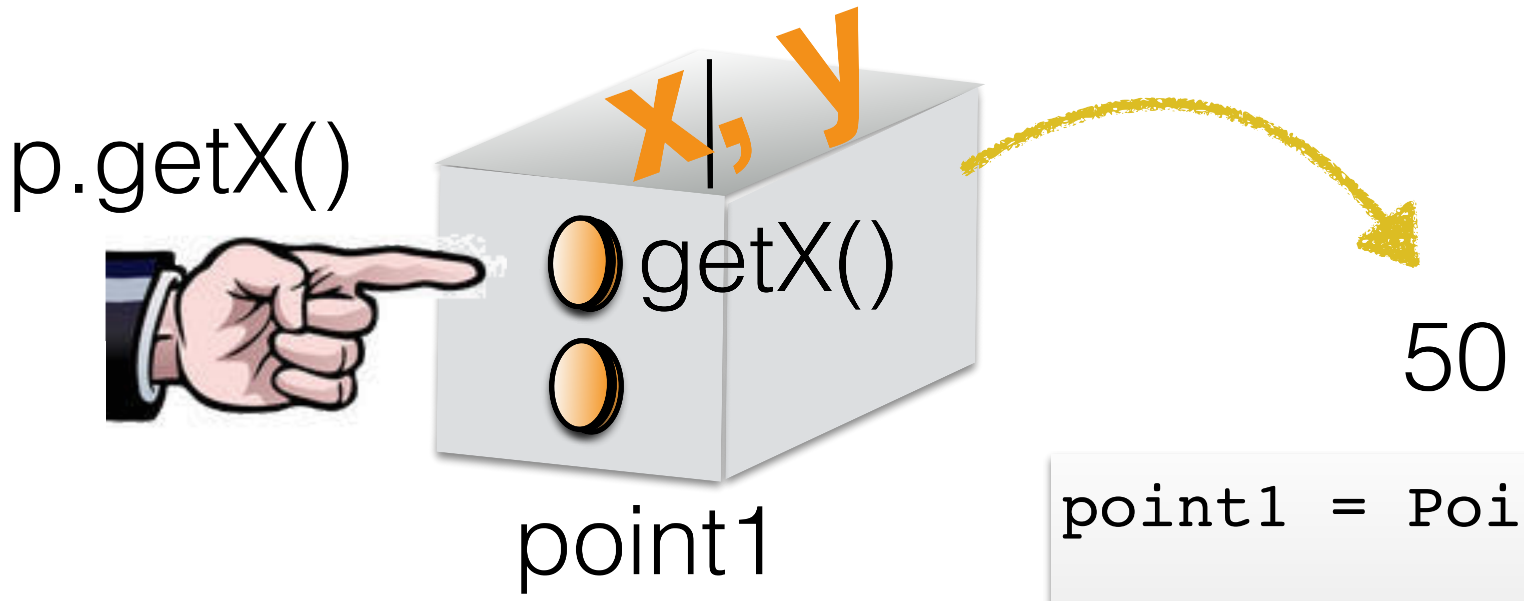
Organization of a graphic program

Something completely different…

Examples of If-Statements in Graphics

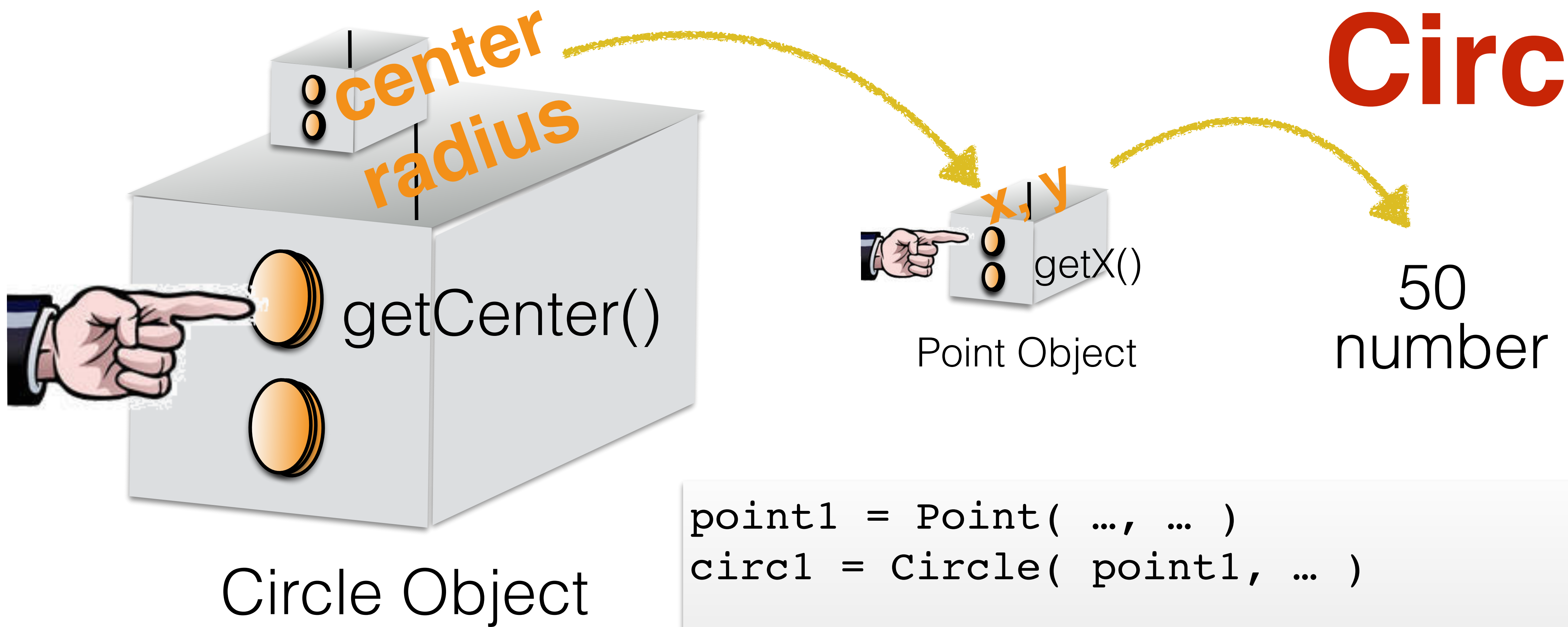
Point

p = Point(50, 150)



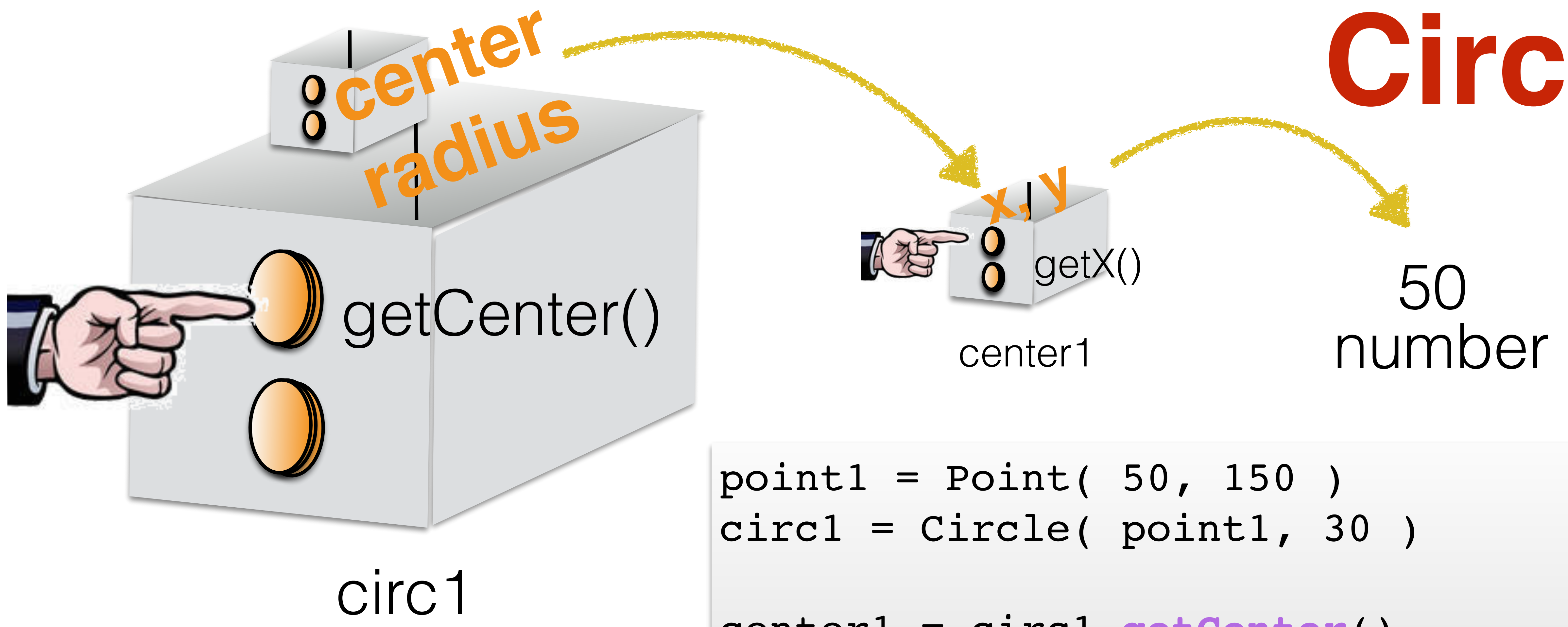
```
point1 = Point( 50, 150 )

x = point1.getX()
y = point1.getY()
if  x <= 0 or y <= 0:
    # the point is outside the window
    …
```

Circle



```
point1 = Point( …, … )
circ1 = Circle( point1, … )

center1 = circ1.getCenter()
x = center1.getX()
y = center1.getY()
if  x <= 0 or y <= 0:
    # the center is outside the window
    …
```

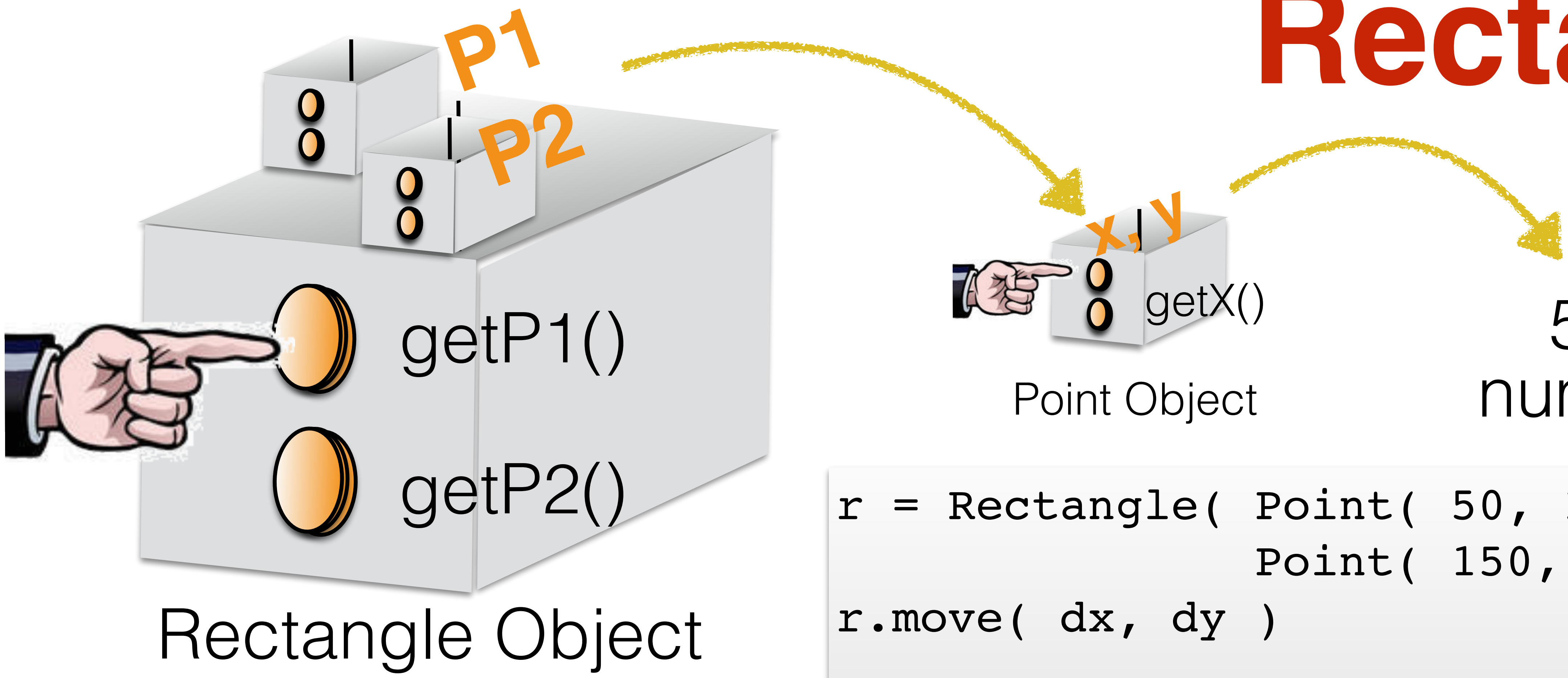
Circle



```
point1 = Point( 50, 150 )
circ1 = Circle( point1, 30 )

center1 = circ1.getCenter()
x = center1.getX()
y = center1.getY()
if  x <= 0 or y <= 0:
    # the center is outside the window
    …

x = circ1.getCenter().getX()
y = circ1.getCenter().getY()
if x <= 0 or y <= 0:
    #
```

Same computation

Rectangle



```
r = Rectangle( Point( 50, 150 ),
               Point( 150, 150 ) )
r.move( dx, dy )

x1 = r.getP1().getX()
y1 = r.getP1().getY()
x2 = r.getP2().getX()
y2 = r.getP2().getY()

mouseP = win.checkMouse()
if mouseP != None:
   x = mouseP.getX()
   y = mouseP.getY()
   if x1 <= x <= x2 and …
```

Outline

Boolean Operators

Exercises

If Statements and Graphics

Organization of a Graphics Program

Measuring Distances

Graphics: Obstacles

Eliza

Organization of a Graphic Program

Skeleton Program

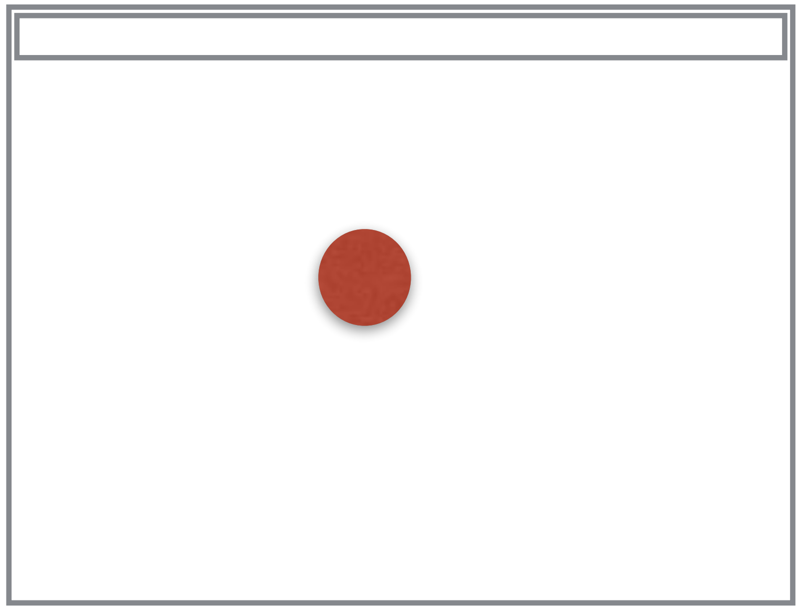
```
def main():
    # open the graphic window


    # define  and initialize the graphic objects





    # start animation loop.  Stop on specific user
    interaction
    while win.checkMouse() == None:

        # move/update each object according to its speed
        # and direction



    # Loop is over.
    # close the graphic window
```

Skeleton Program

```
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define  and initialize the graphic objects




    # start animation loop.  Stop on specific user
    interaction


        # move/update each object according to its speed
        # and direction



    # Loop is over.
    # close the graphic window
```

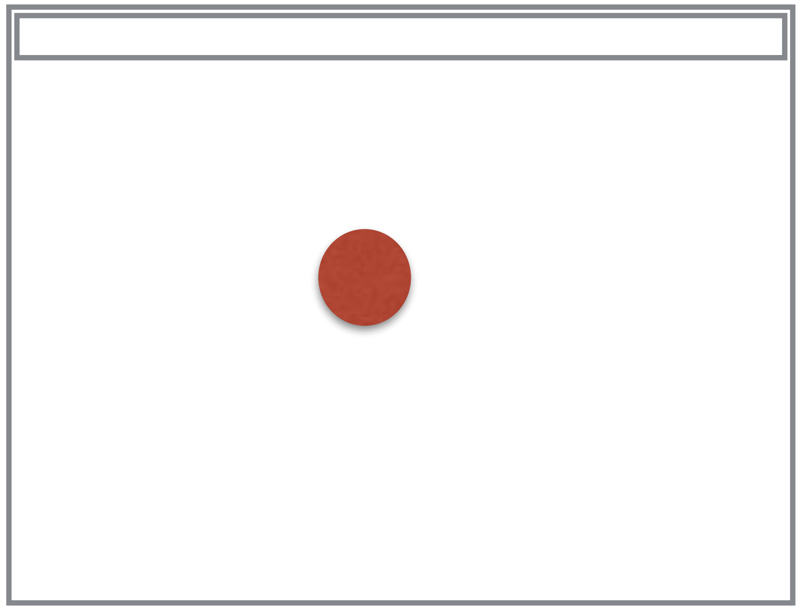
Skeleton Program

```
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define  and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop.  Stop on specific user
    interaction


        # move/update each object according to its speed
        # and direction



    # Loop is over.
    # close the graphic window
```



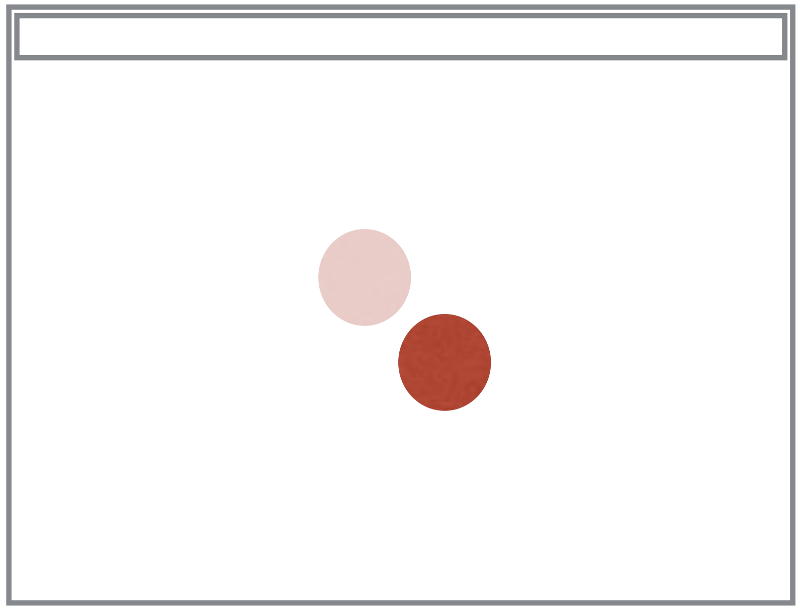
Skeleton Program

```
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define  and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop.  Stop on specific user
    interaction
    while win.checkMouse() == None:

        # move/update each object according to its speed
        # and direction



    # Loop is over.
    # close the graphic window
```



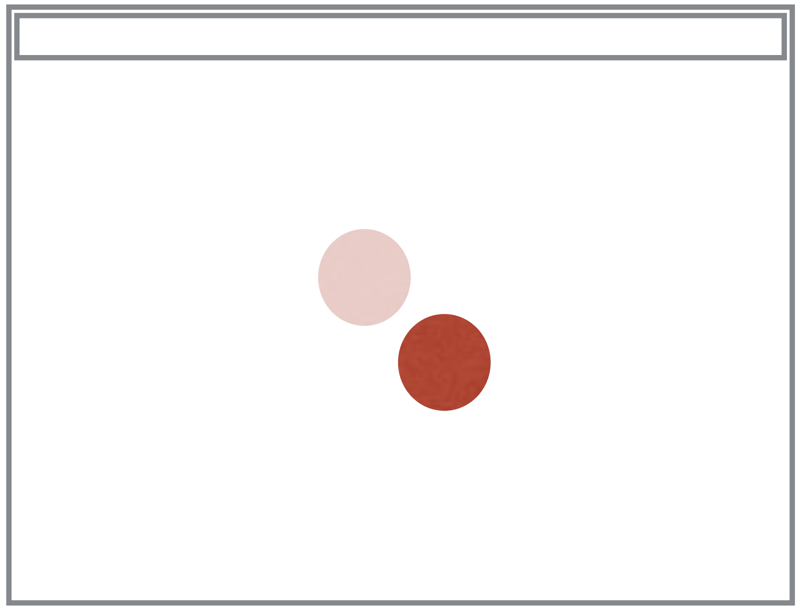
Skeleton Program

```
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define  and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop.  Stop on specific user
    interaction
    while win.checkMouse() == None:

        # move/update each object according to its speed
        # and direction
        circ.move( dx, dy )


    # Loop is over.
    # close the graphic window
```



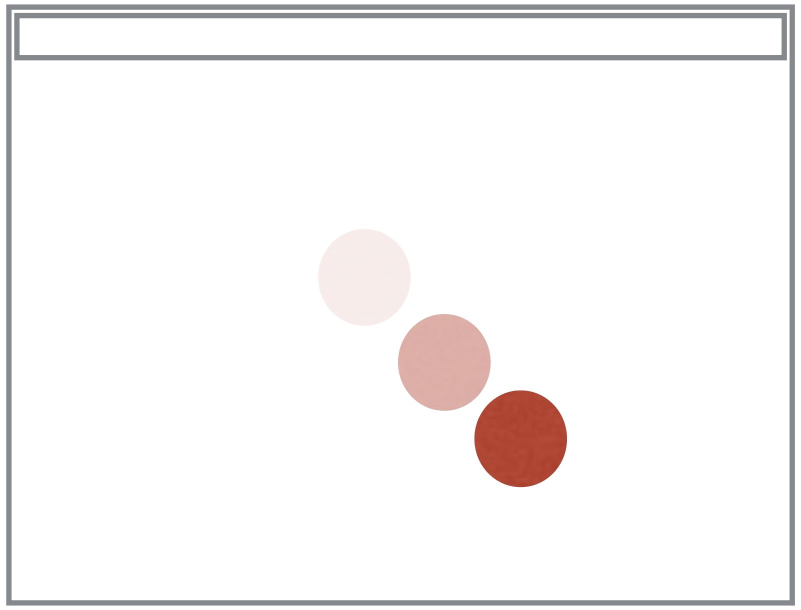
Skeleton Program

```
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define  and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop.  Stop on specific user interaction
    while win.checkMouse() == None:

        # move/update each object according to its speed
        # and direction
        circ.move( dx, dy )


    # Loop is over.
    # close the graphic window
```



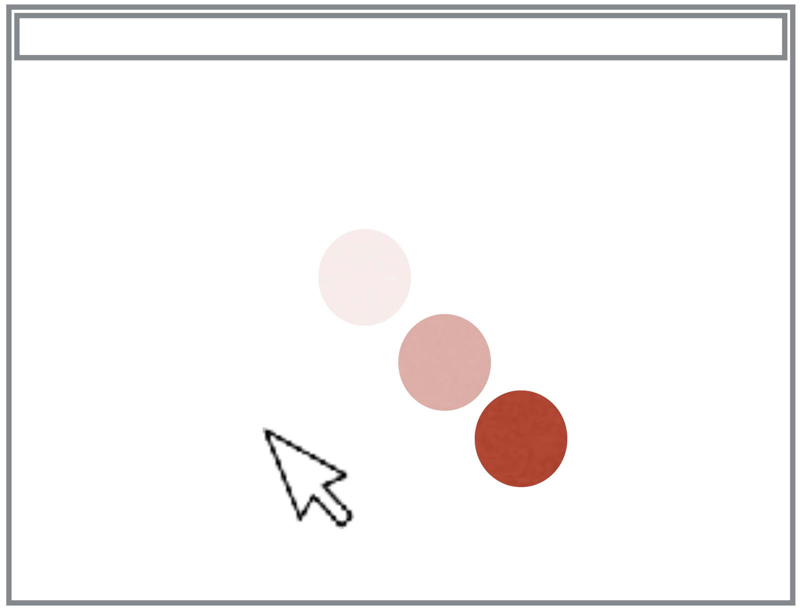
Skeleton Program

```
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define  and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop.  Stop on specific user
    interaction
    while win.checkMouse() == None:

        # move/update each object according to its speed
        # and direction
        circ.move( dx, dy )


    # Loop is over.
    # close the graphic window
```



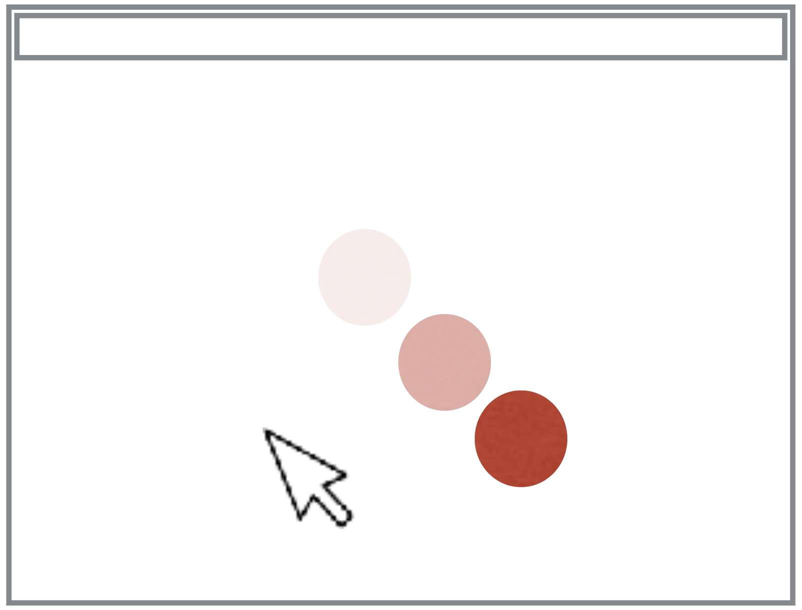
Skeleton Program

```
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define  and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop.  Stop on specific user
    interaction
    while win.checkMouse() == None:

        # move/update each object according to its speed
        # and direction
        circ.move( dx, dy )


    # Loop is over.
    # close the graphic window
```



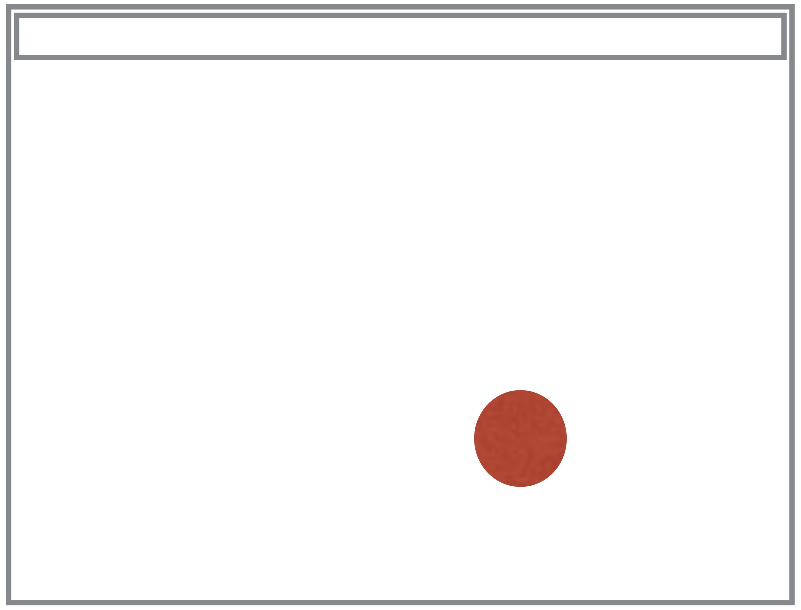
Skeleton Program

```
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define  and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop.  Stop on specific user
    interaction
    while win.checkMouse() == None:

        # move/update each object according to its speed
        # and direction
        circ.move( dx, dy )


    # Loop is over.
    # close the graphic window
```



Skeleton Program

```
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define  and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop.  Stop on specific user
    interaction
    while win.checkMouse() == None:

        # move/update each object according to its speed
        # and direction
        circ.move( dx, dy )


    # Loop is over.
    # close the graphic window
    win.close()
```



Skeleton Program

```
def main():
    # open the graphic window
    win = GraphWin( "Demo", 600, 400 )

    # define  and initialize the graphic objects
    circ = Circle( Point( 100, 100 ), 30 )
    circ.setFill( 'red' )
    circ.draw( win )
    dx, dy = 3, 2

    # start animation loop.  Stop on specific user
    interaction
    while win.checkMouse() == None:

        # move/update each object according to its speed
        # and direction
        circ.move( dx, dy )


    # Loop is over.
    # close the graphic window
    win.close()
```

Outline

Boolean Operators

Exercises

If Statements and Graphics

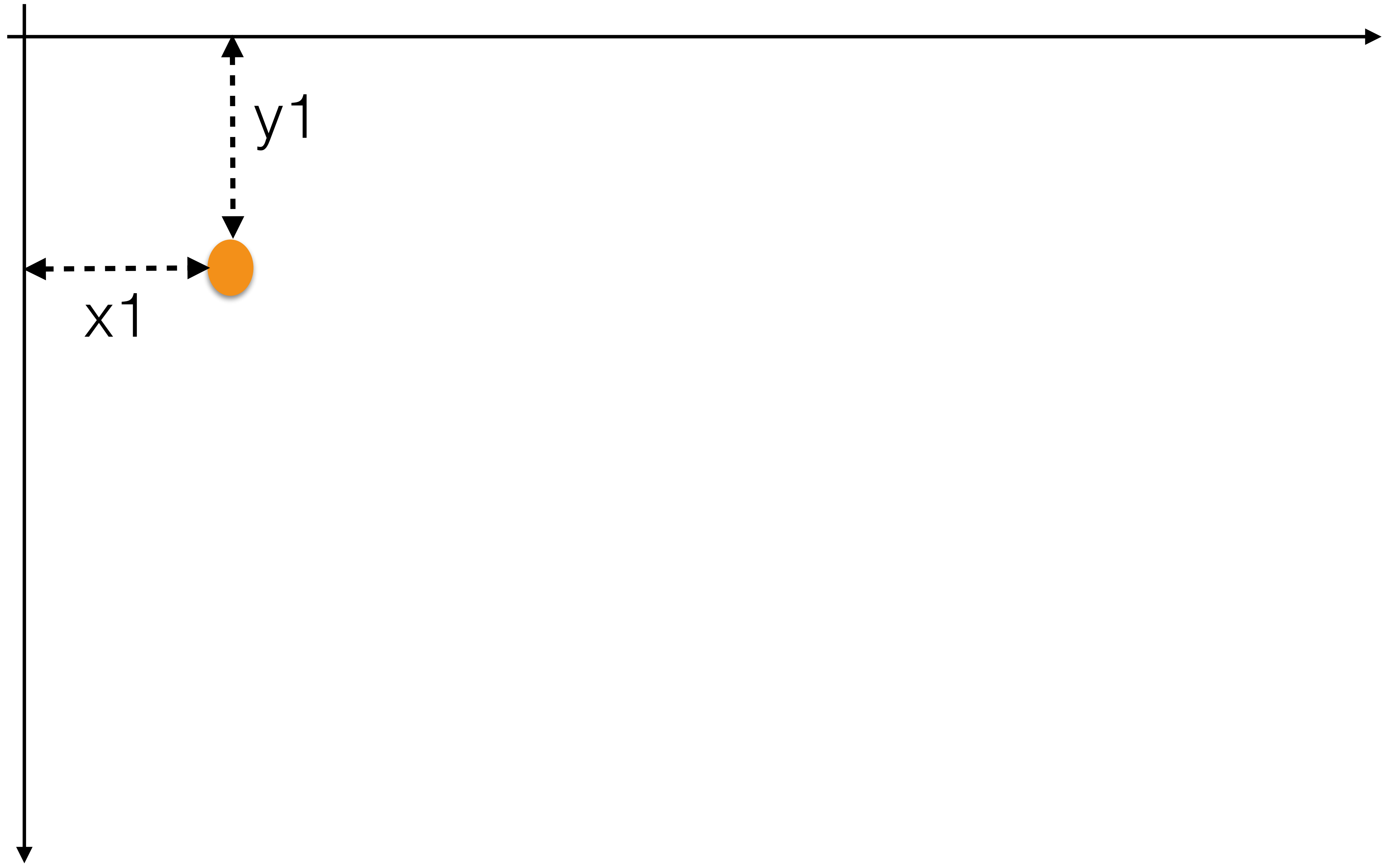
Organization of a Graphics Program

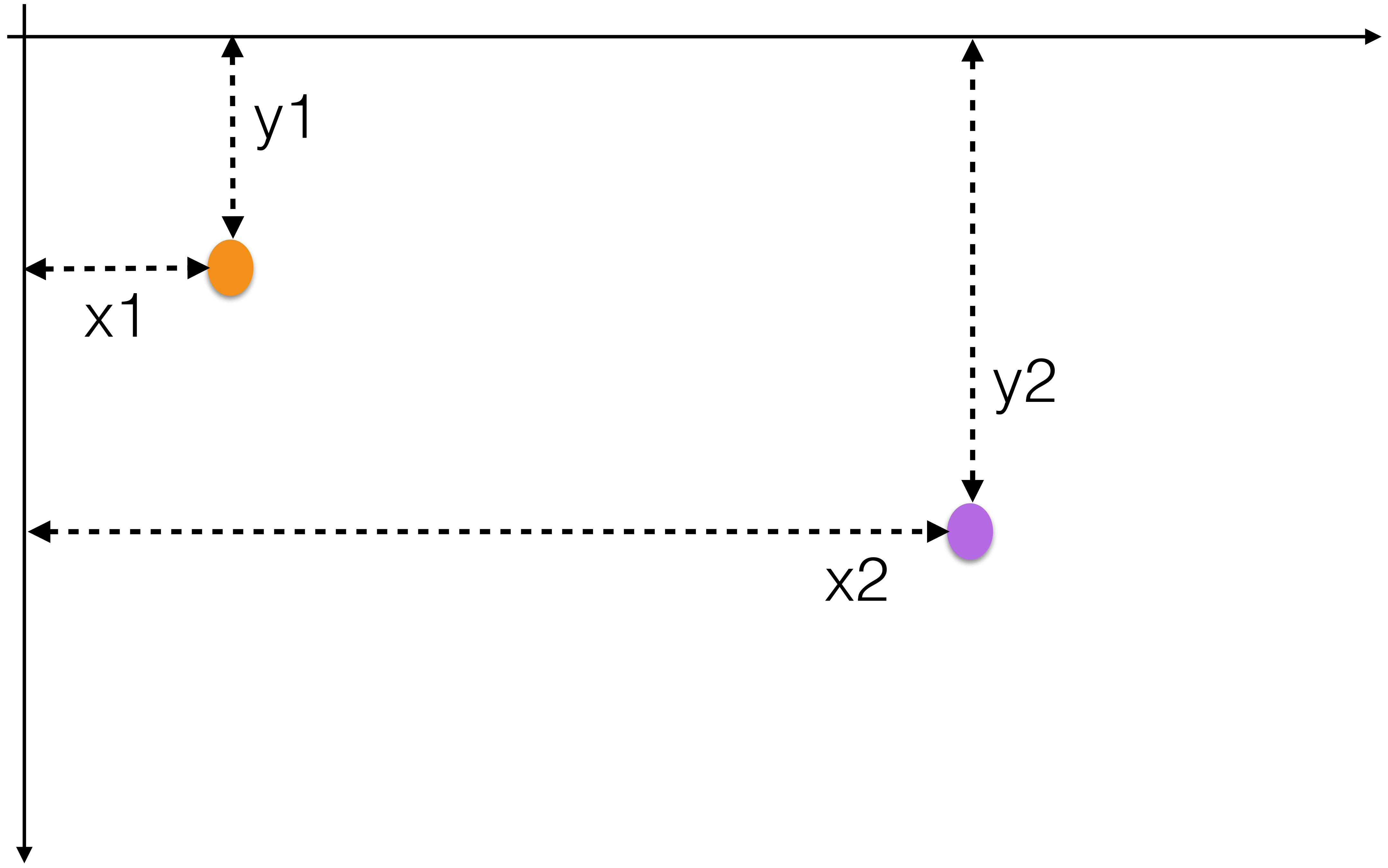
Measuring Distances

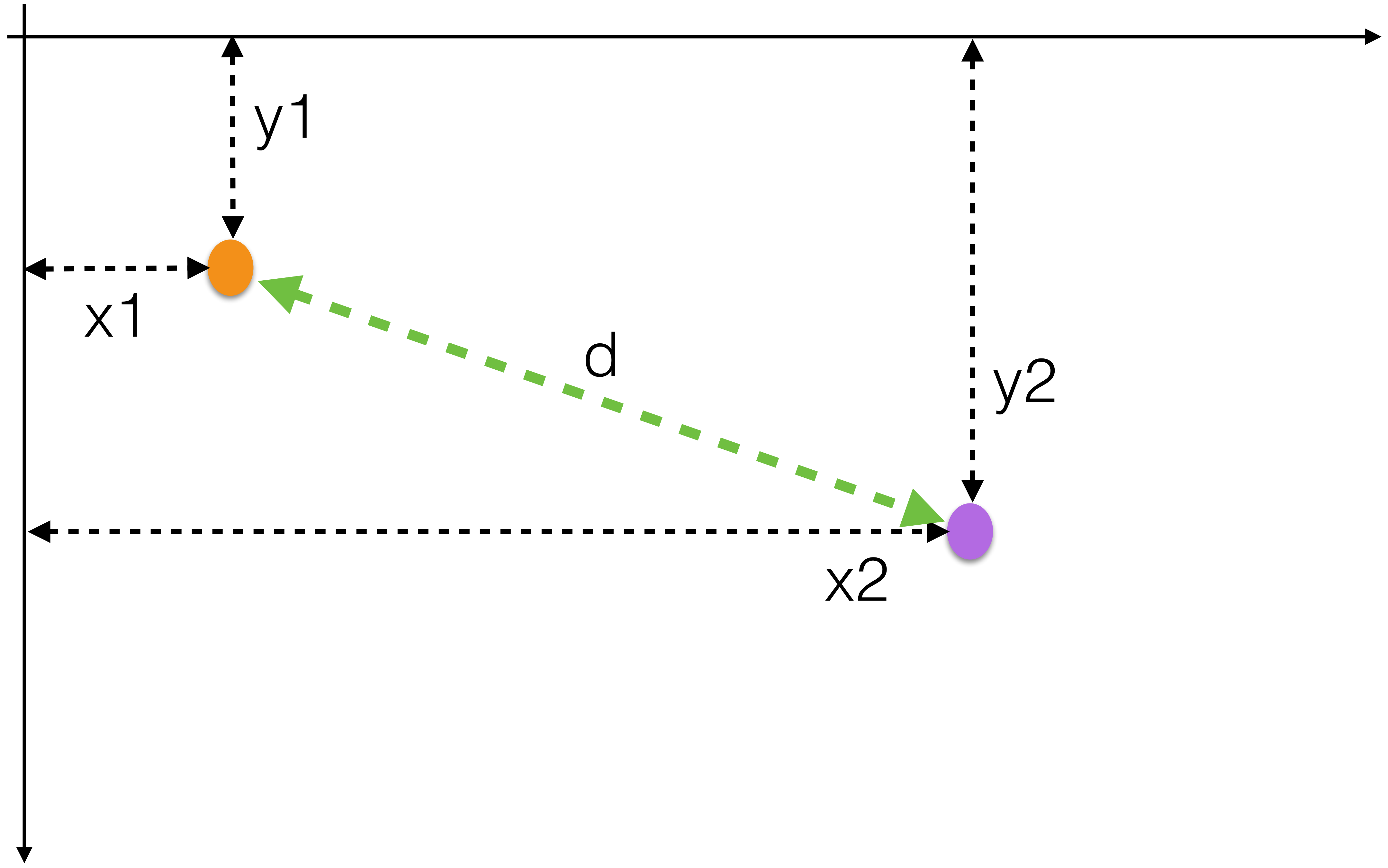
Graphics: Obstacles

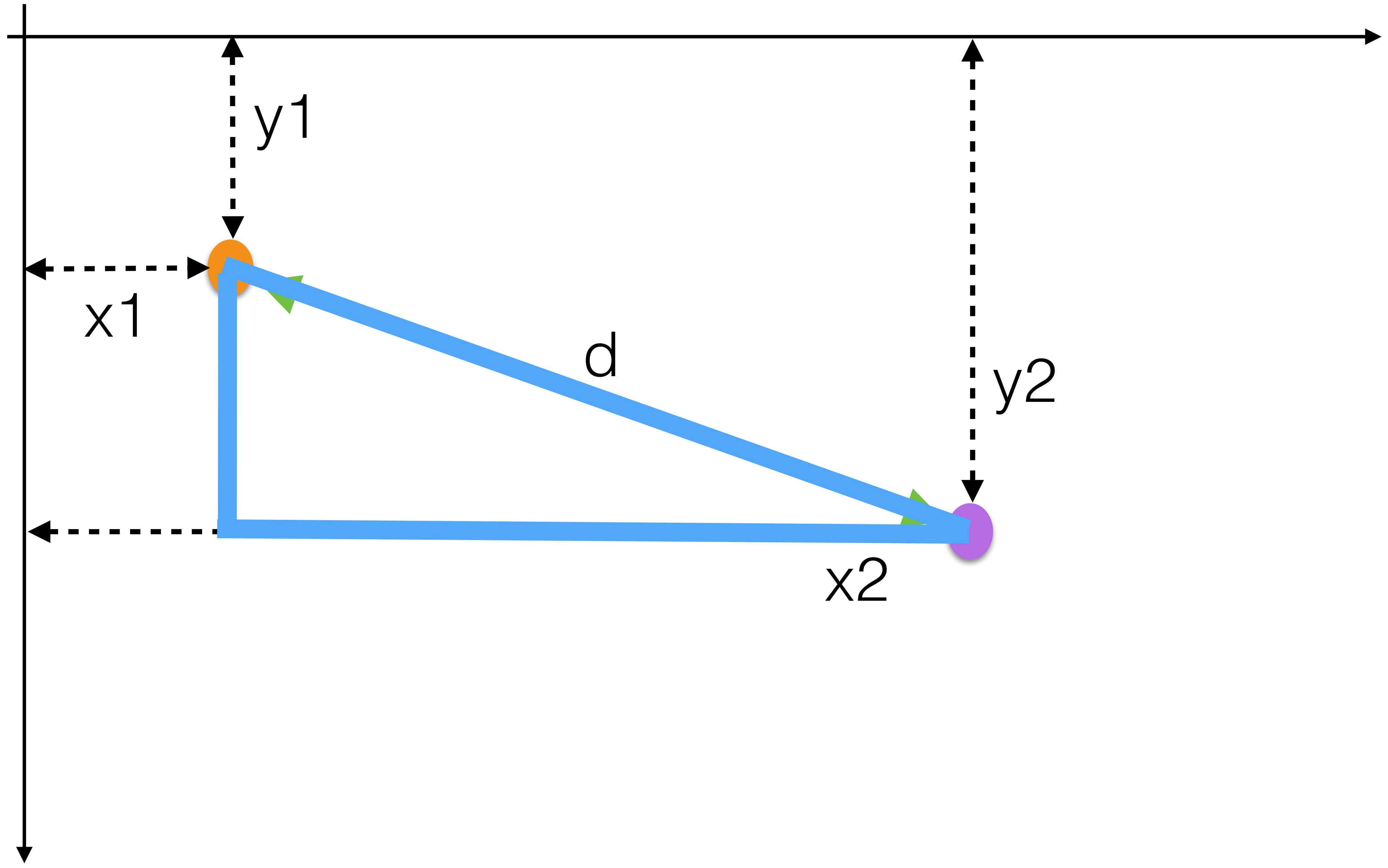
Eliza

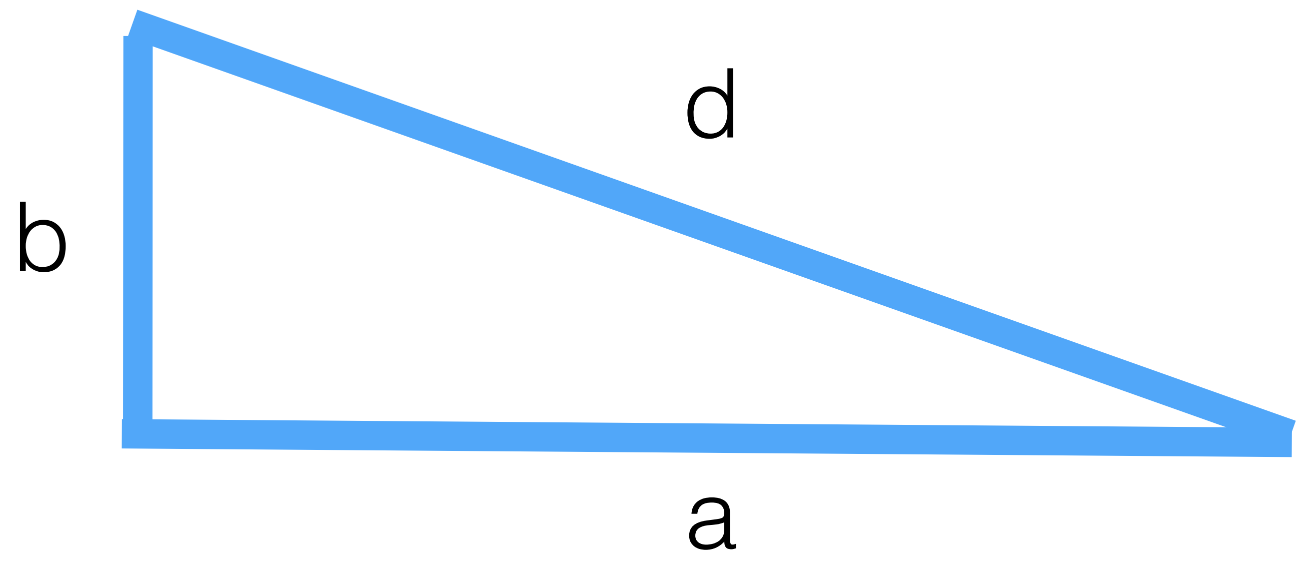
Measuring Distances


y1
x1

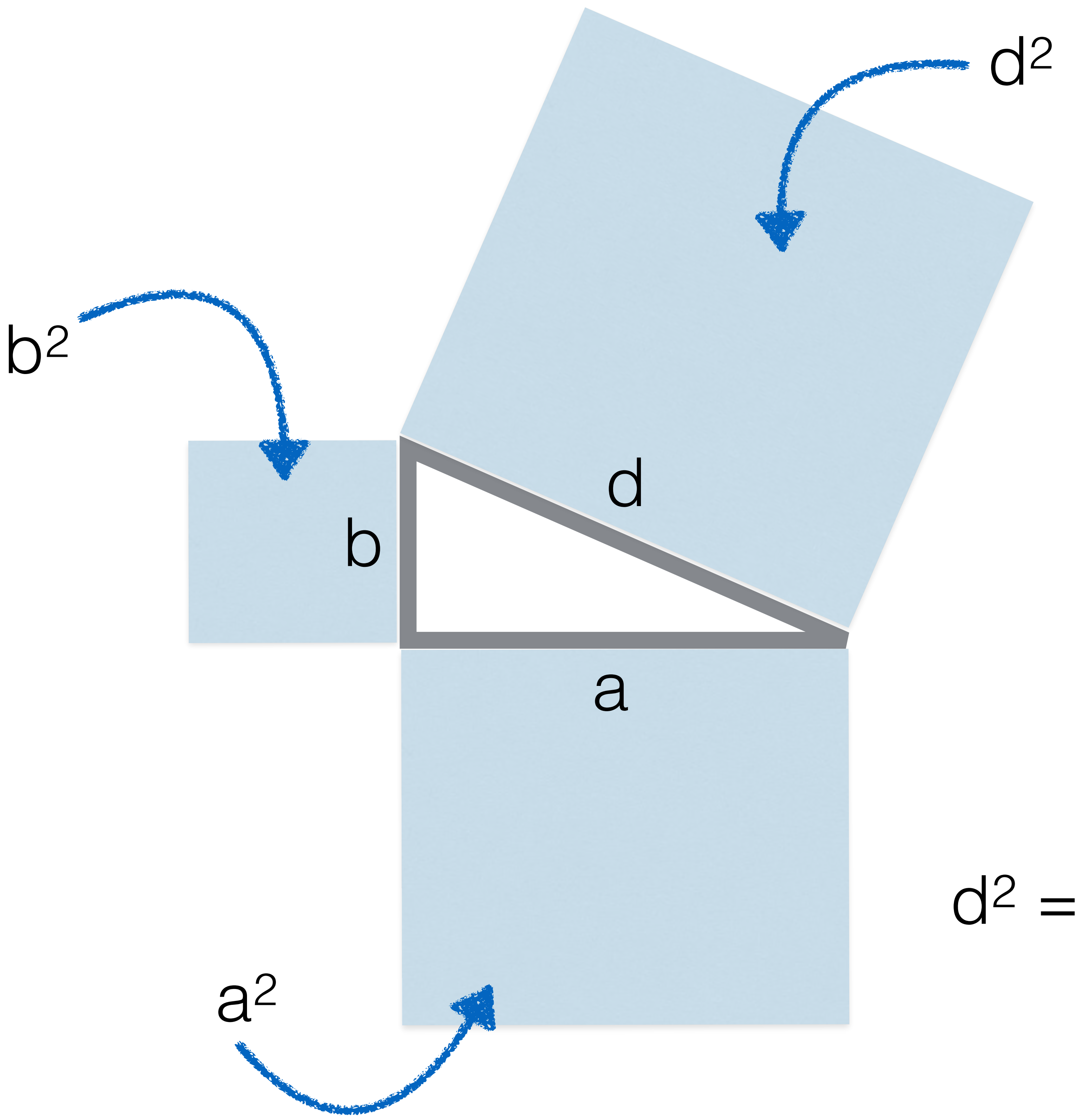

y1
x1
y2
x2


y1
x1
d
y2
x2

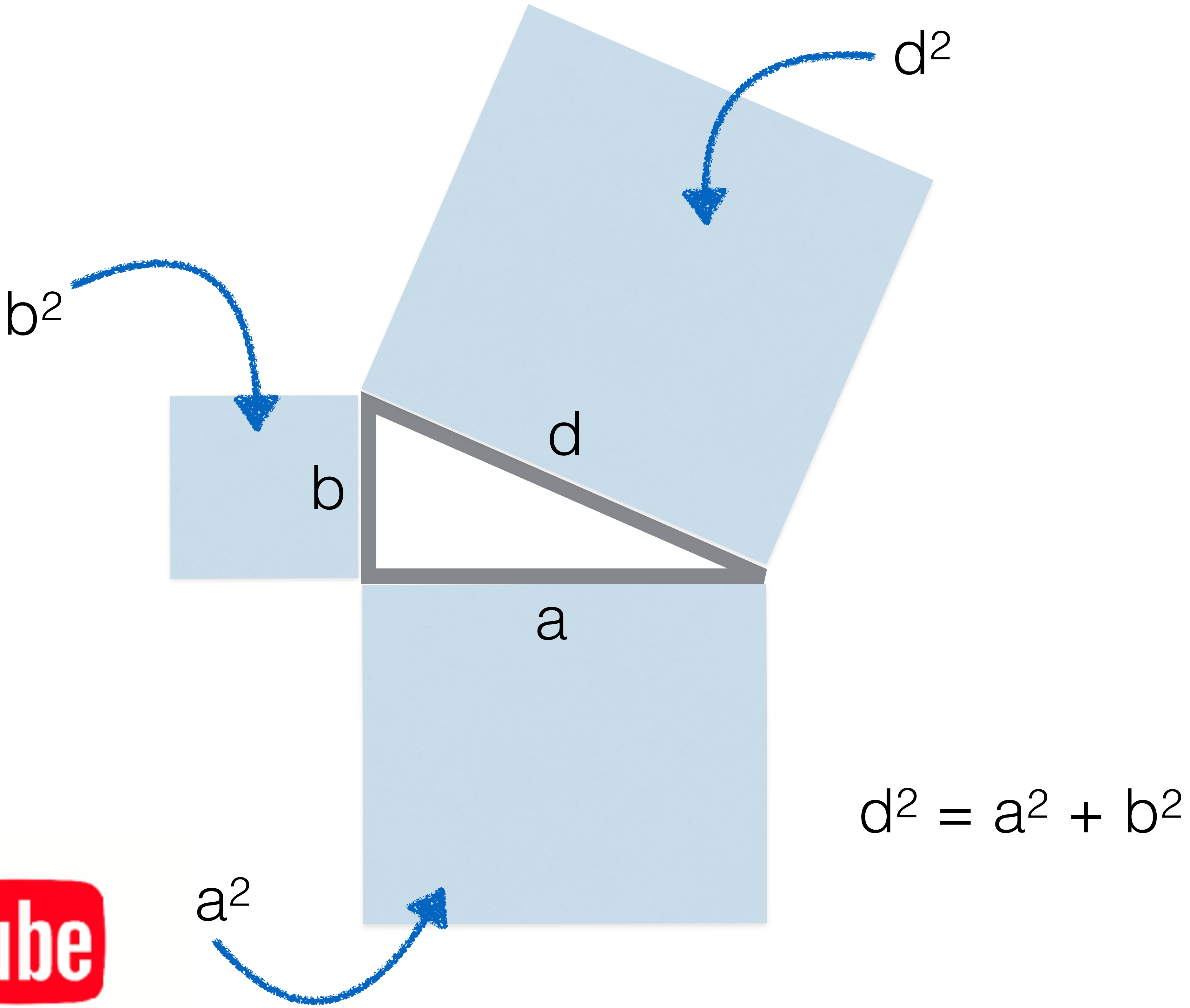

y1
x1
d
y2
x2



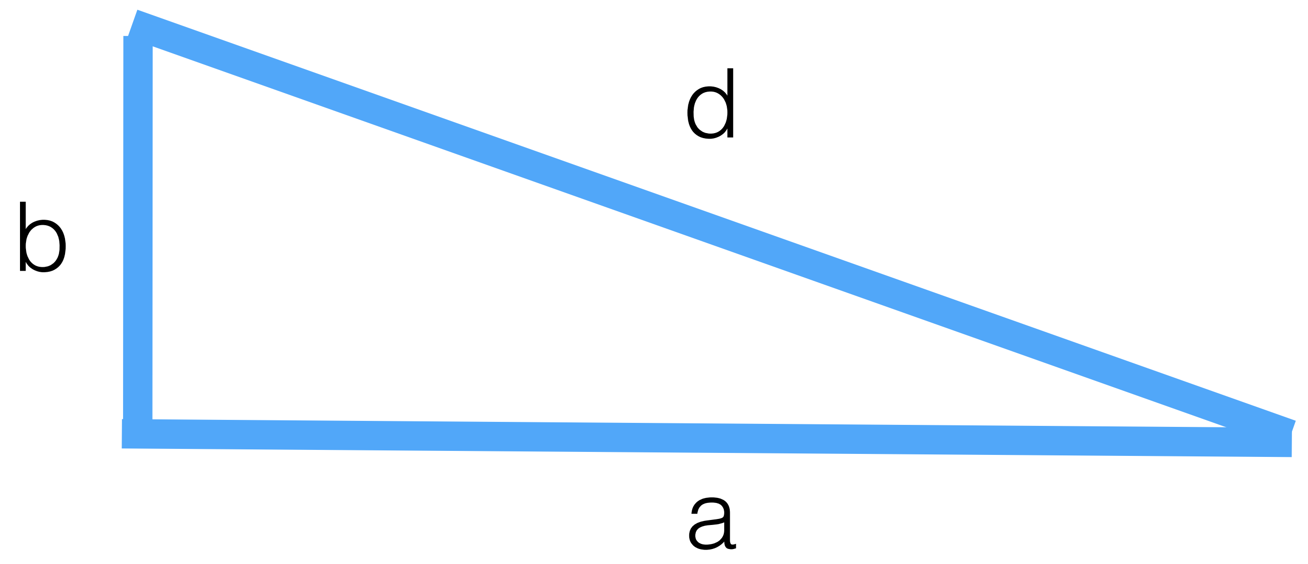
$$d^2 = a^2 + b^2$$



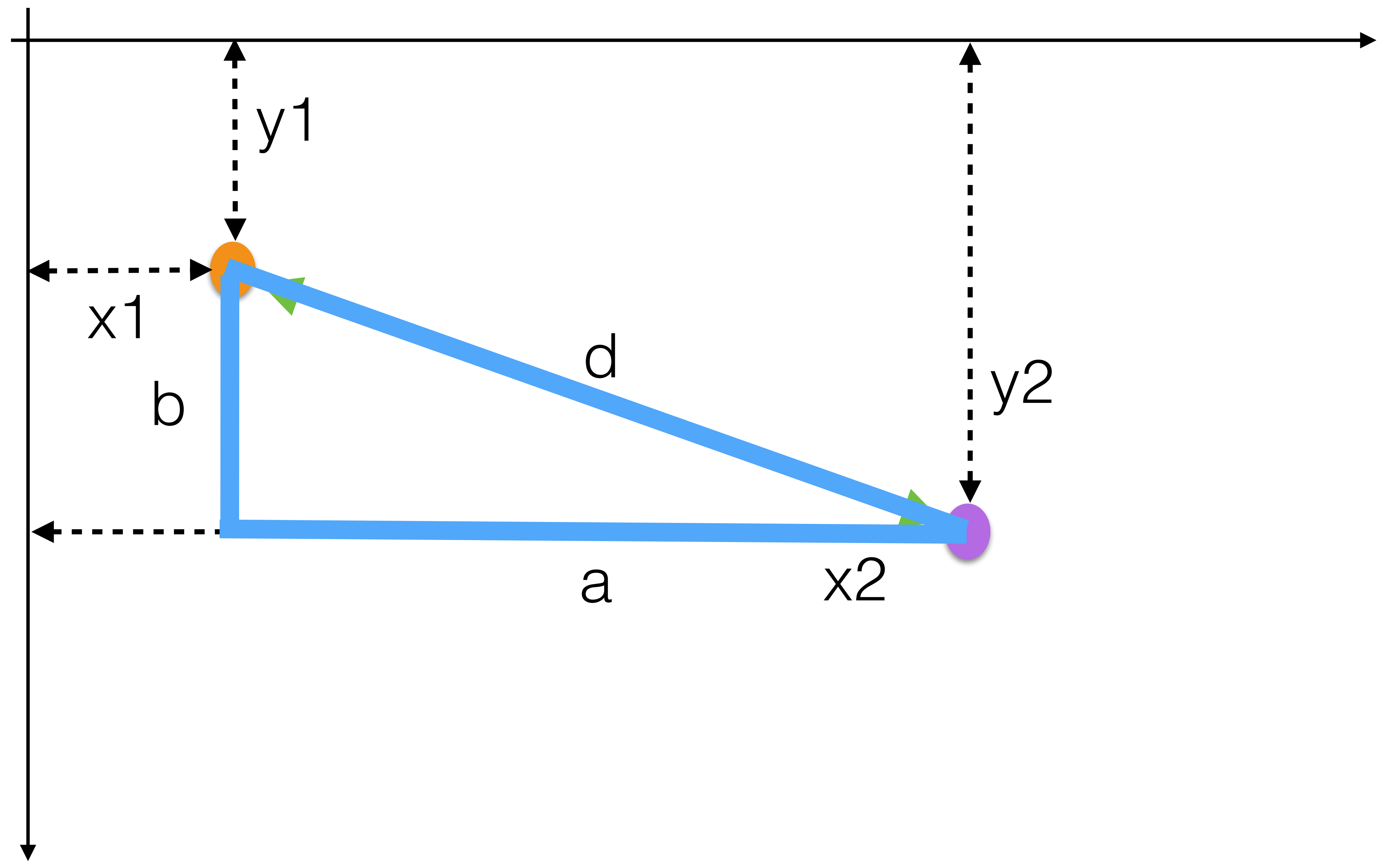
$$d^2 = a^2 + b^2$$

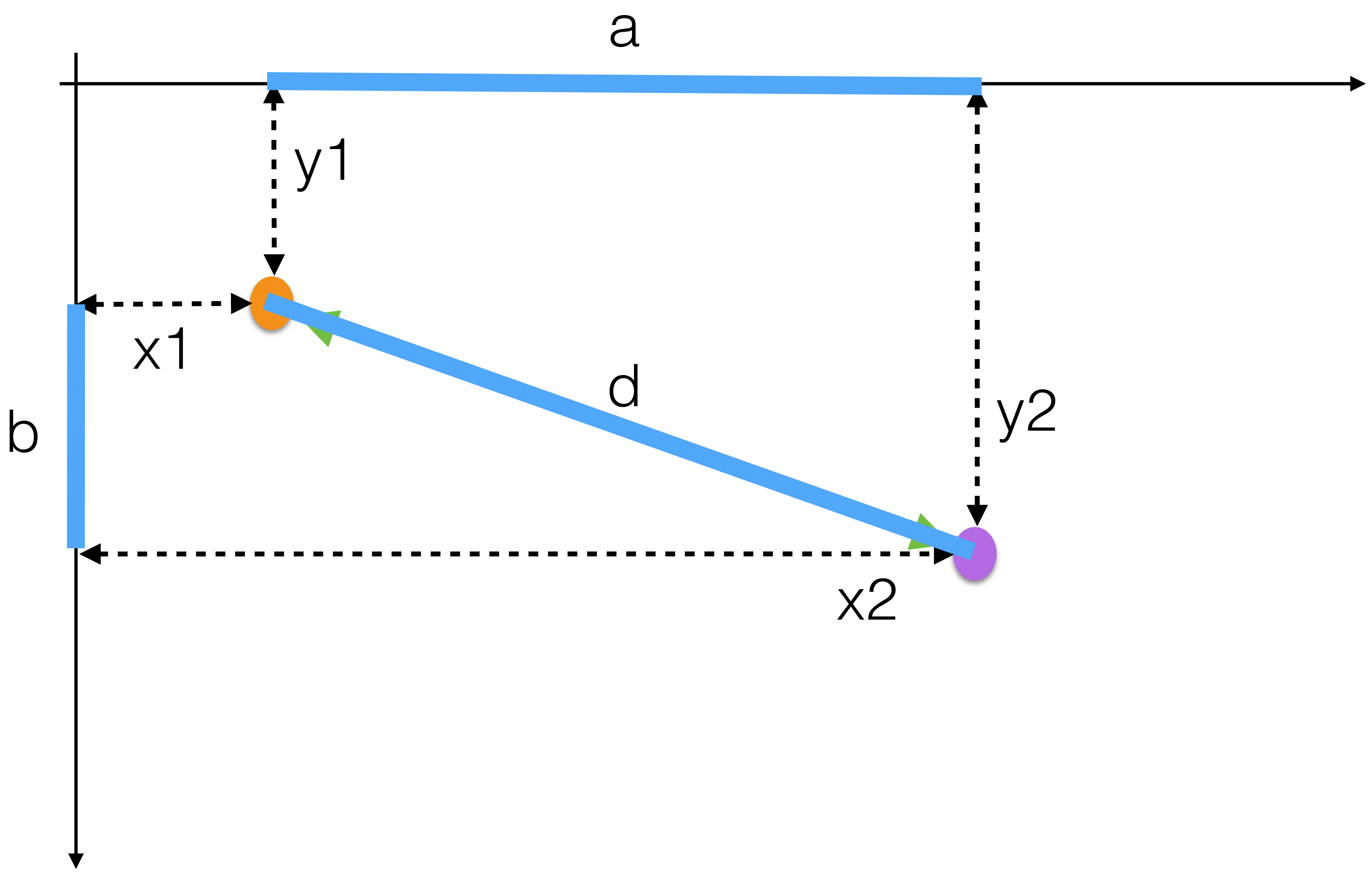


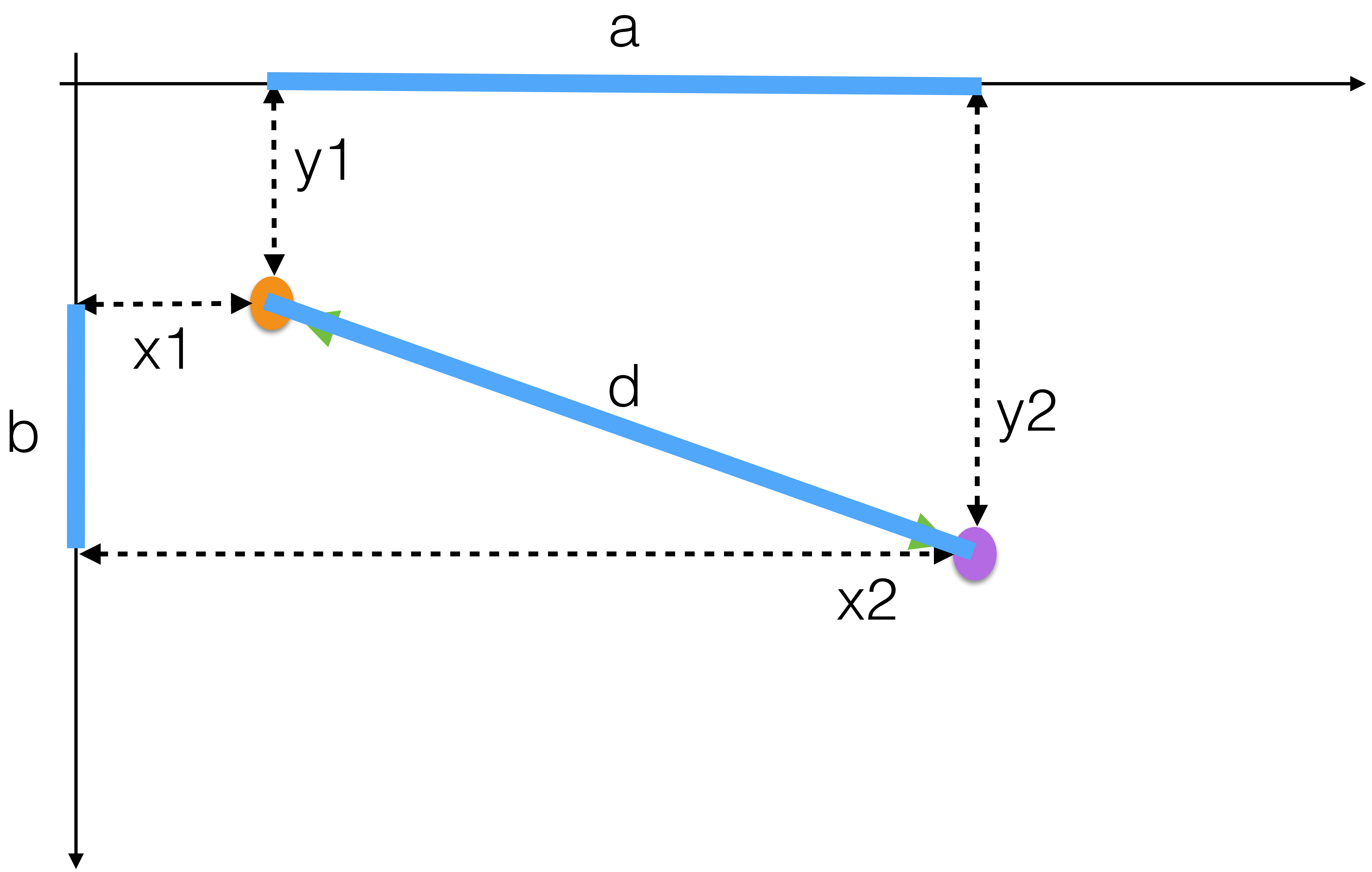
$d^2 = a^2 + b^2$



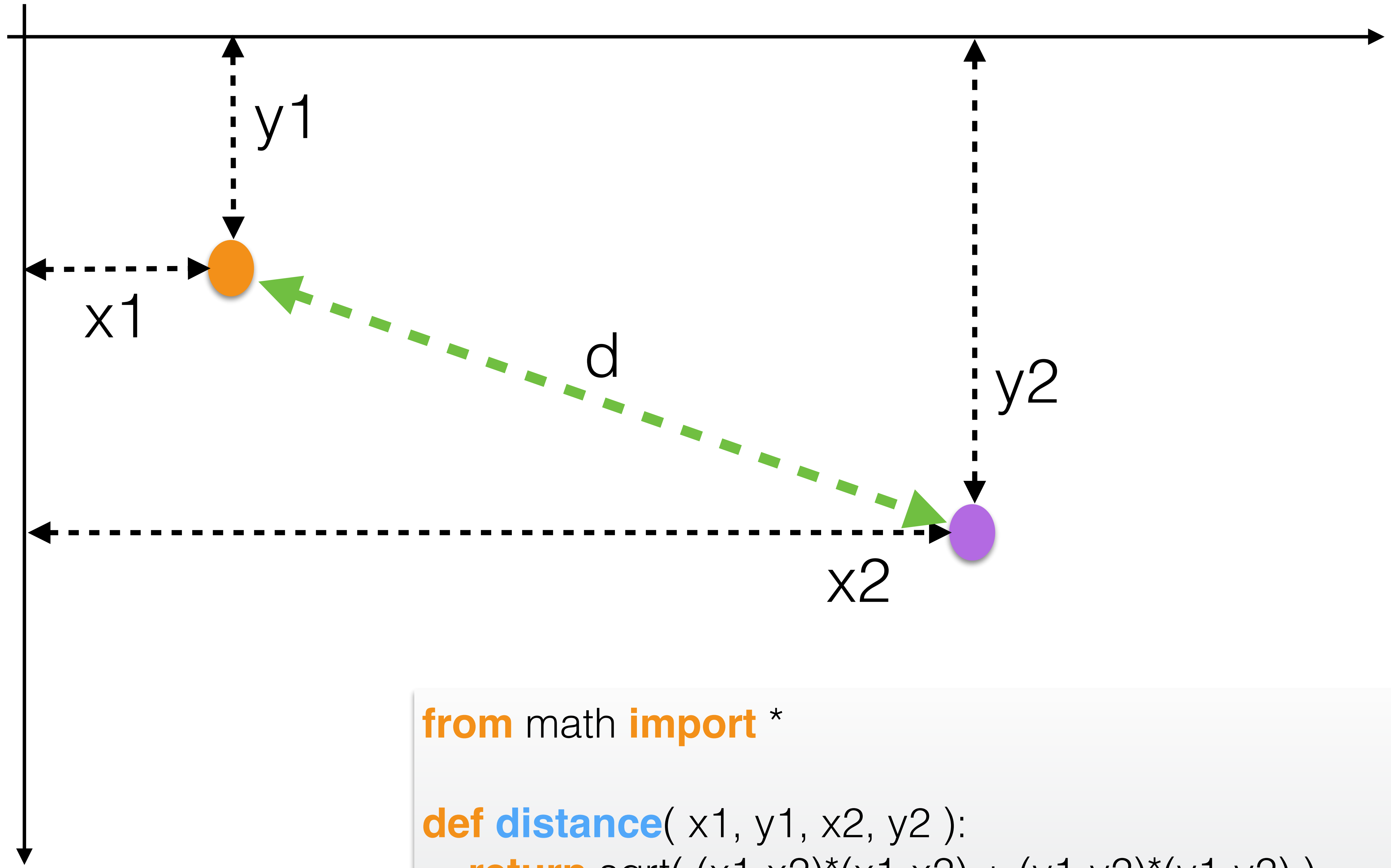
$d^2 = a^2 + b^2$


y1
x1
b
d
y2
a
x2

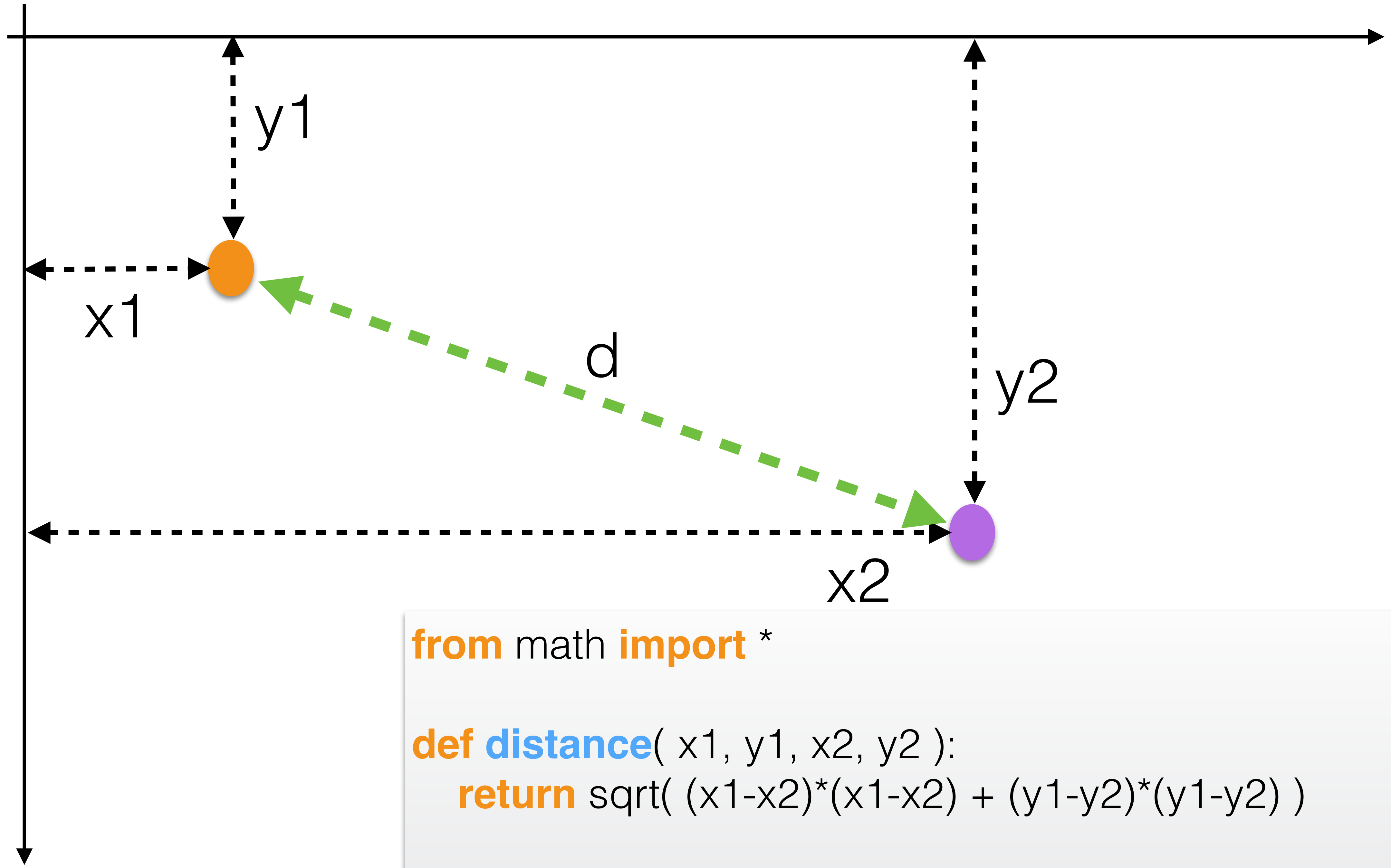

a
y1
x1
d
y2
b
x2



d = sqrt((y2-y1)*(y2-y1) + (x2-x1)*(x2-x1))

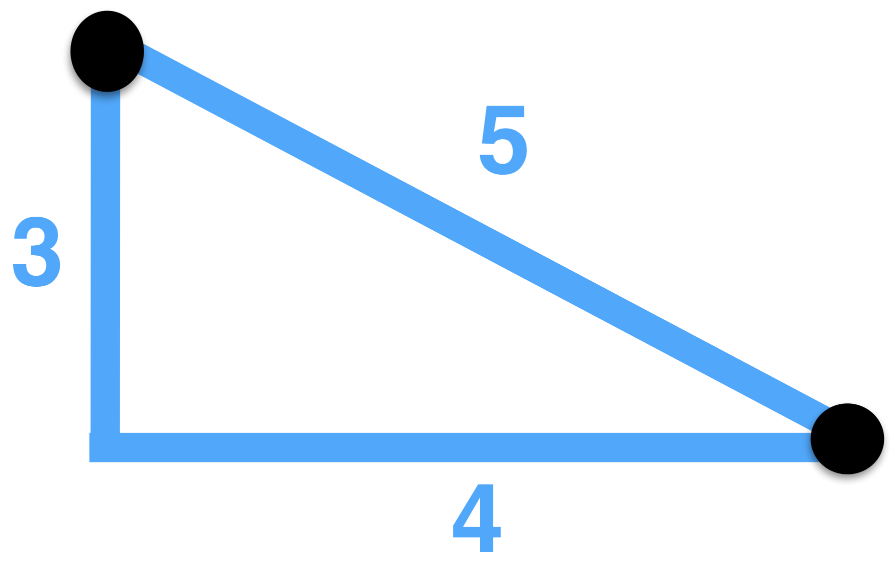


```
from math import *

def distance( x1, y1, x2, y2 ):
    return sqrt( (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) )
```

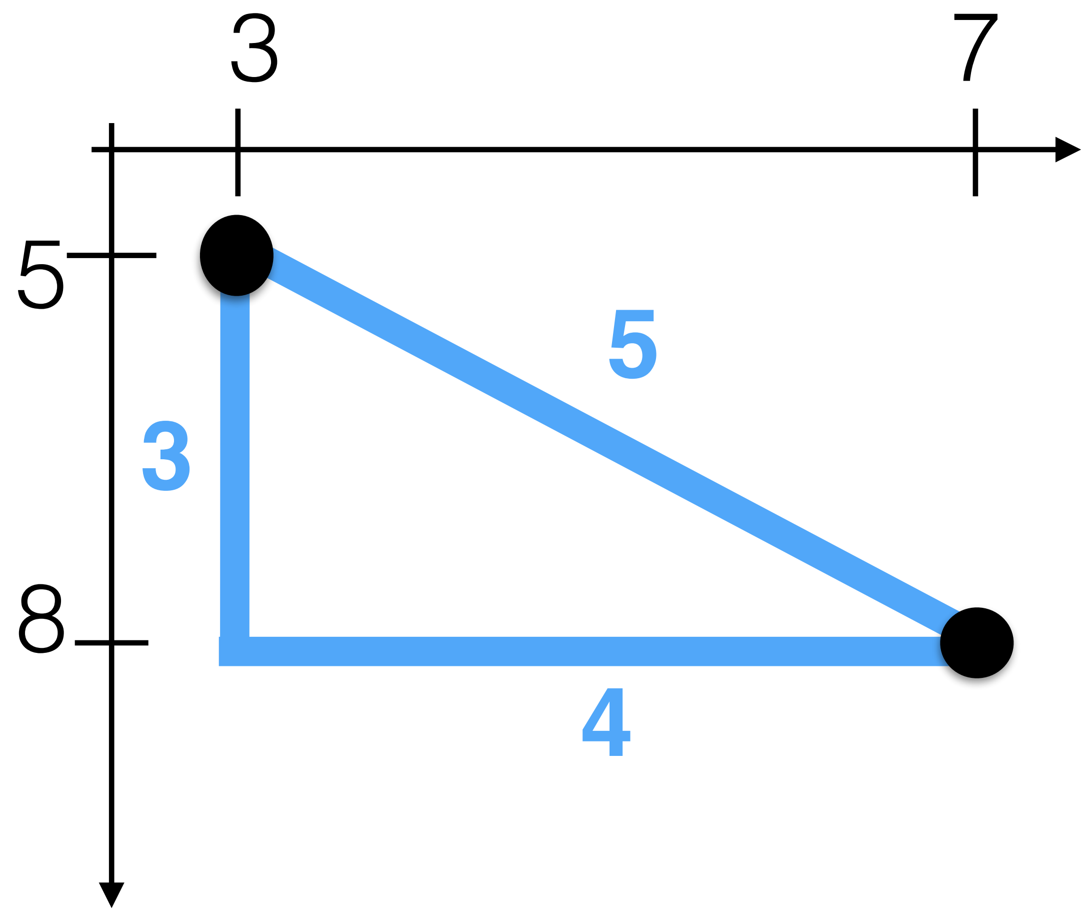


```
from math import *

def distance( x1, y1, x2, y2 ):
    return sqrt( (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) )

def distanceP( p1, p2 ):
    x1, y1 = p1.getX(), p1.getY()
    x2, y2 = p2.getX(), p2.getY()
    return distance( x1, y1, x2, y2 )
```

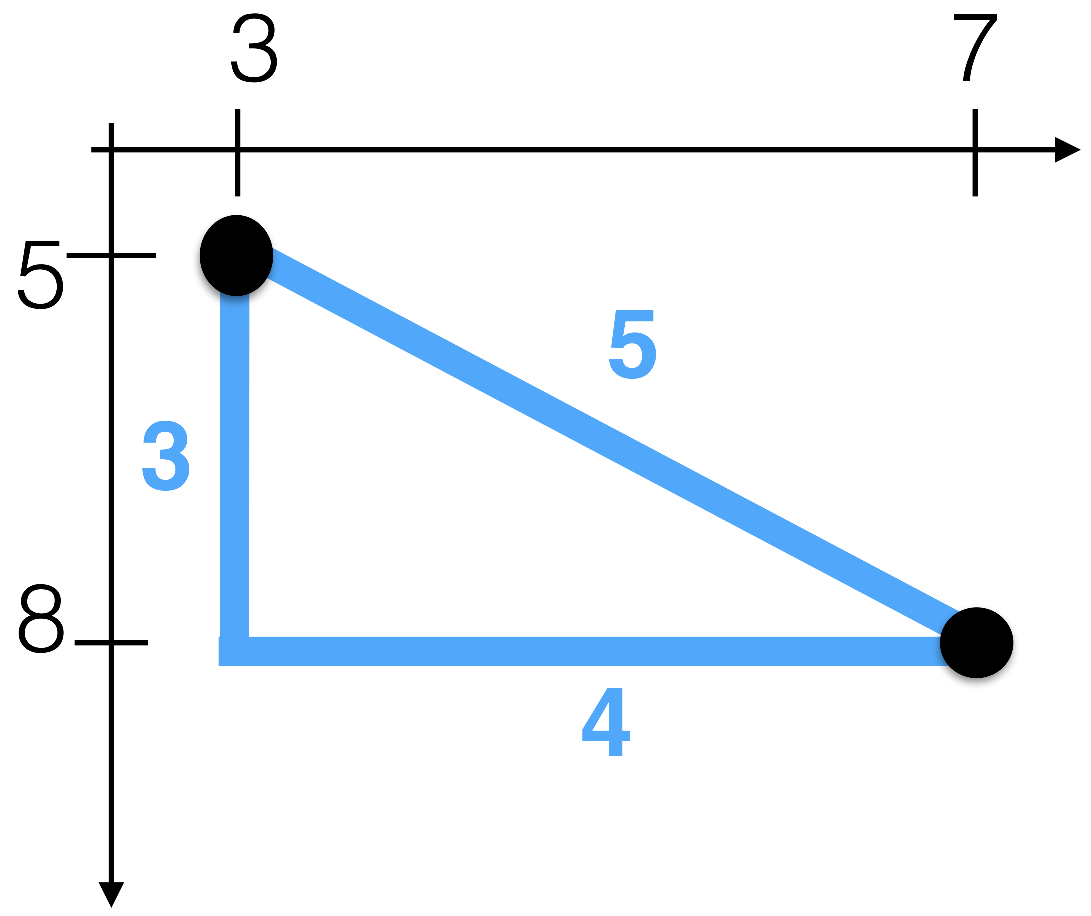
Testing

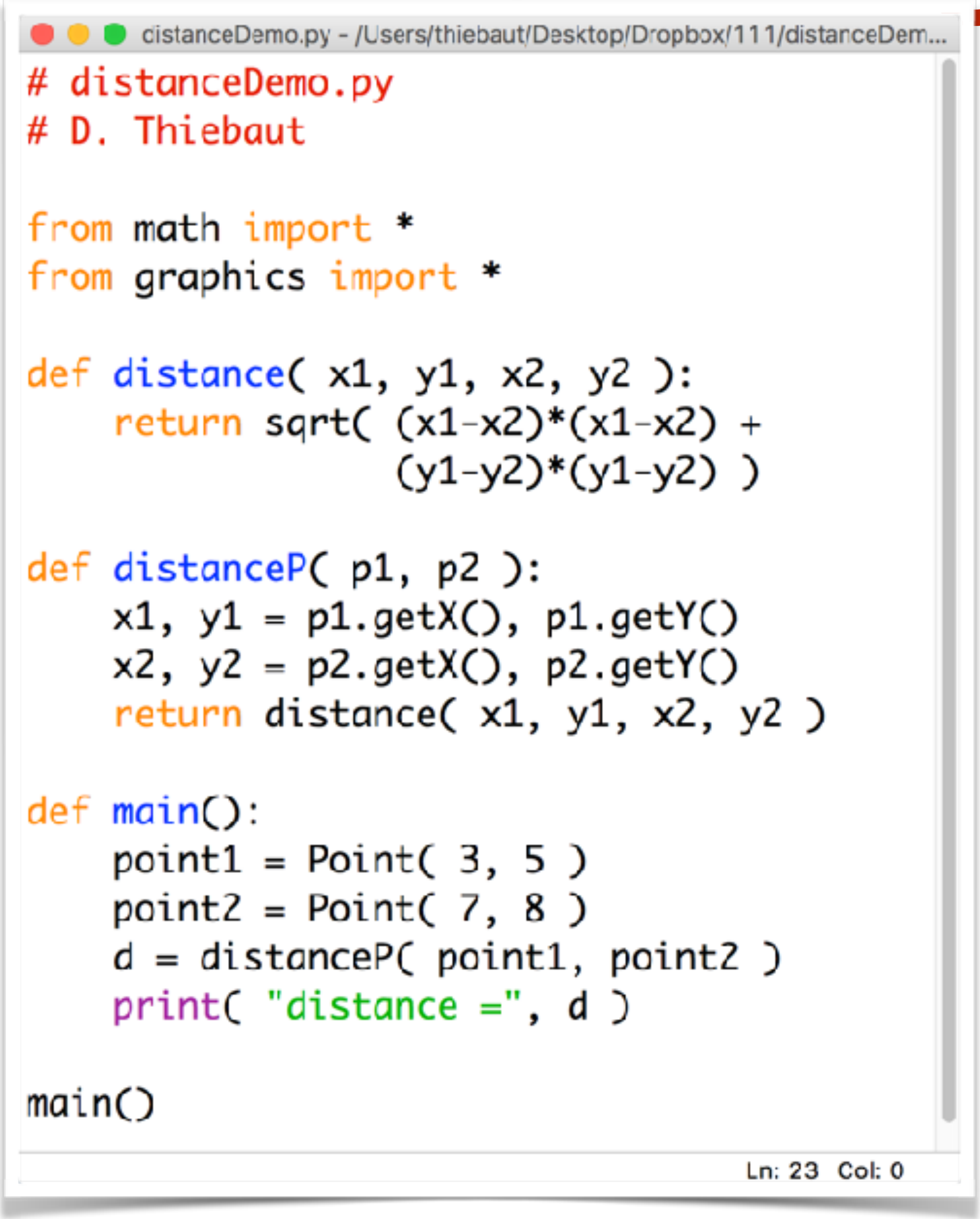


Testing



Testing



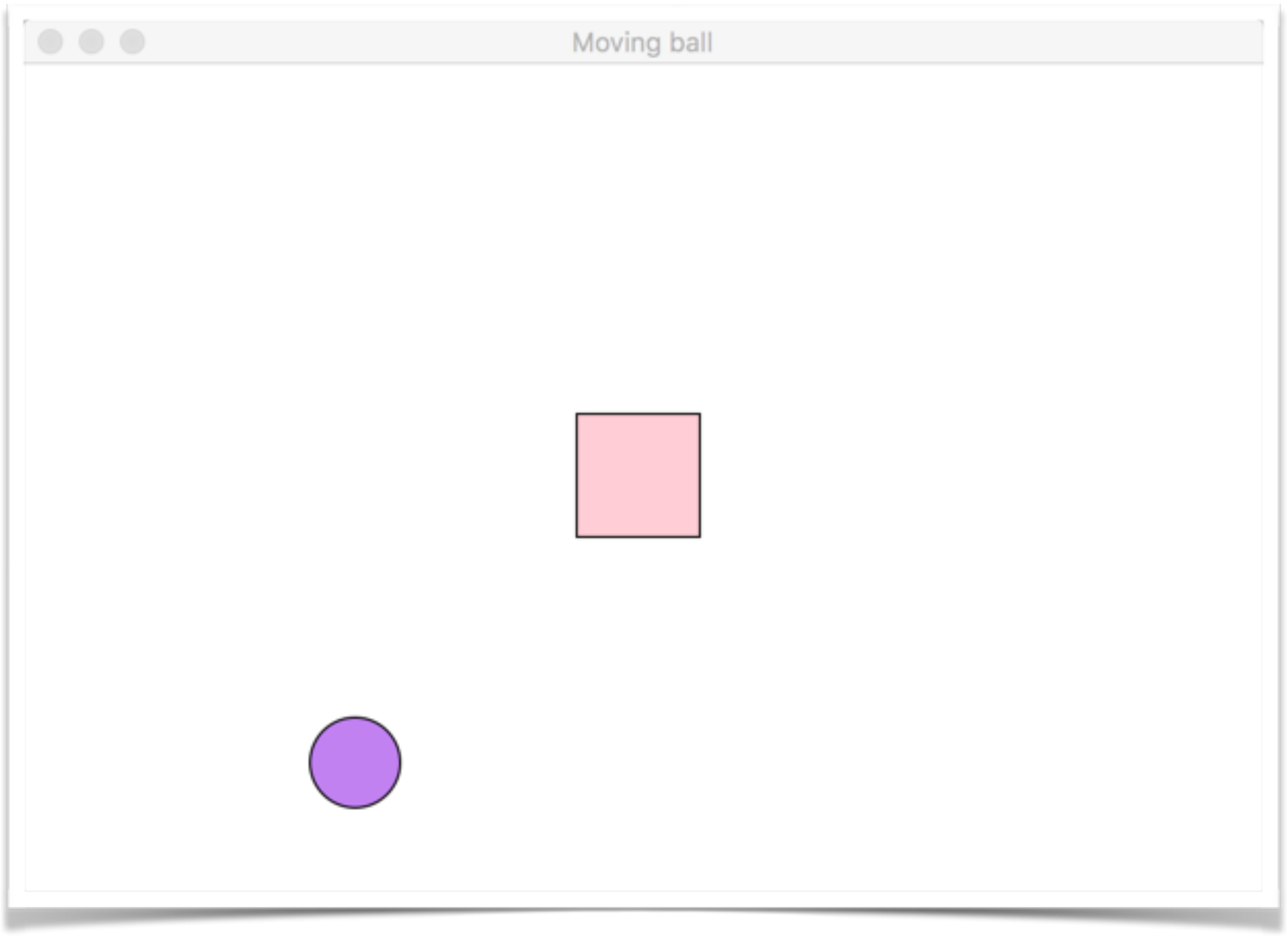

distanceDemo.py - /Users/thiebaut/Desktop/Dropbox/111/distanceDem...

```
# distanceDemo.py
# D. Thiebaut

from math import *
from graphics import *

def distance( x1, y1, x2, y2 ):
    return sqrt( (x1-x2)*(x1-x2) +
                 (y1-y2)*(y1-y2) )

def distanceP( p1, p2 ):
    x1, y1 = p1.getX(), p1.getY()
    x2, y2 = p2.getX(), p2.getY()
    return distance( x1, y1, x2, y2 )

def main():
    point1 = Point( 3, 5 )
    point2 = Point( 7, 8 )
    d = distanceP( point1, point2 )
    print( "distance =", d )

main()
```

Graphics:
Detecting Obstacles

Exercise: Obstacle

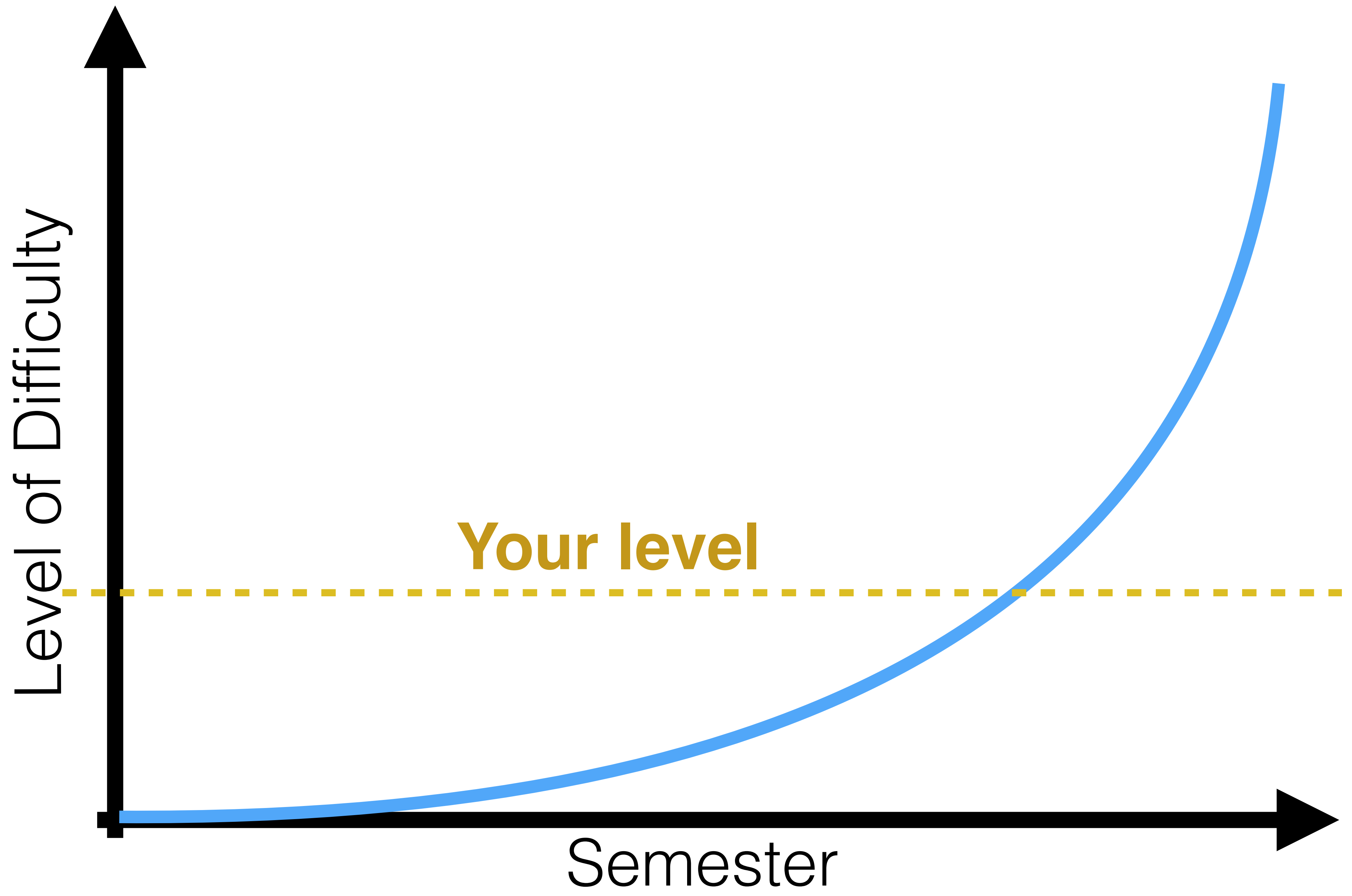
Take the graphic program moving a circle around, and create an obstacle.



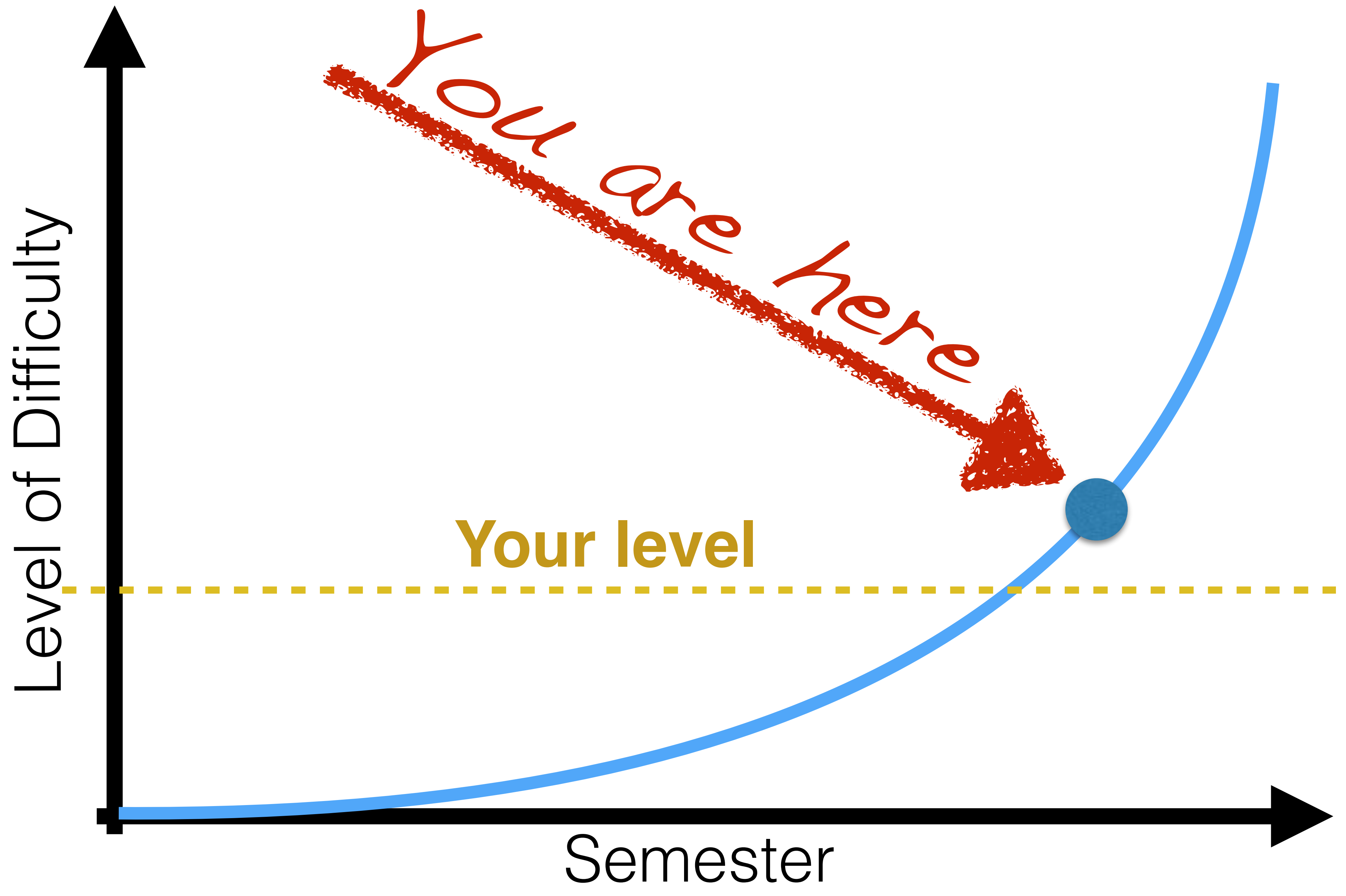
© Can Stock Photo

We stopped here last time…

CSC111: Amount of Work



CSC111: Amount of Work



Outline

Boolean Operators

Exercises

If Statements and Graphics

Organization of a Graphics Program

Measuring Distances

Graphics: Obstacles

Eliza

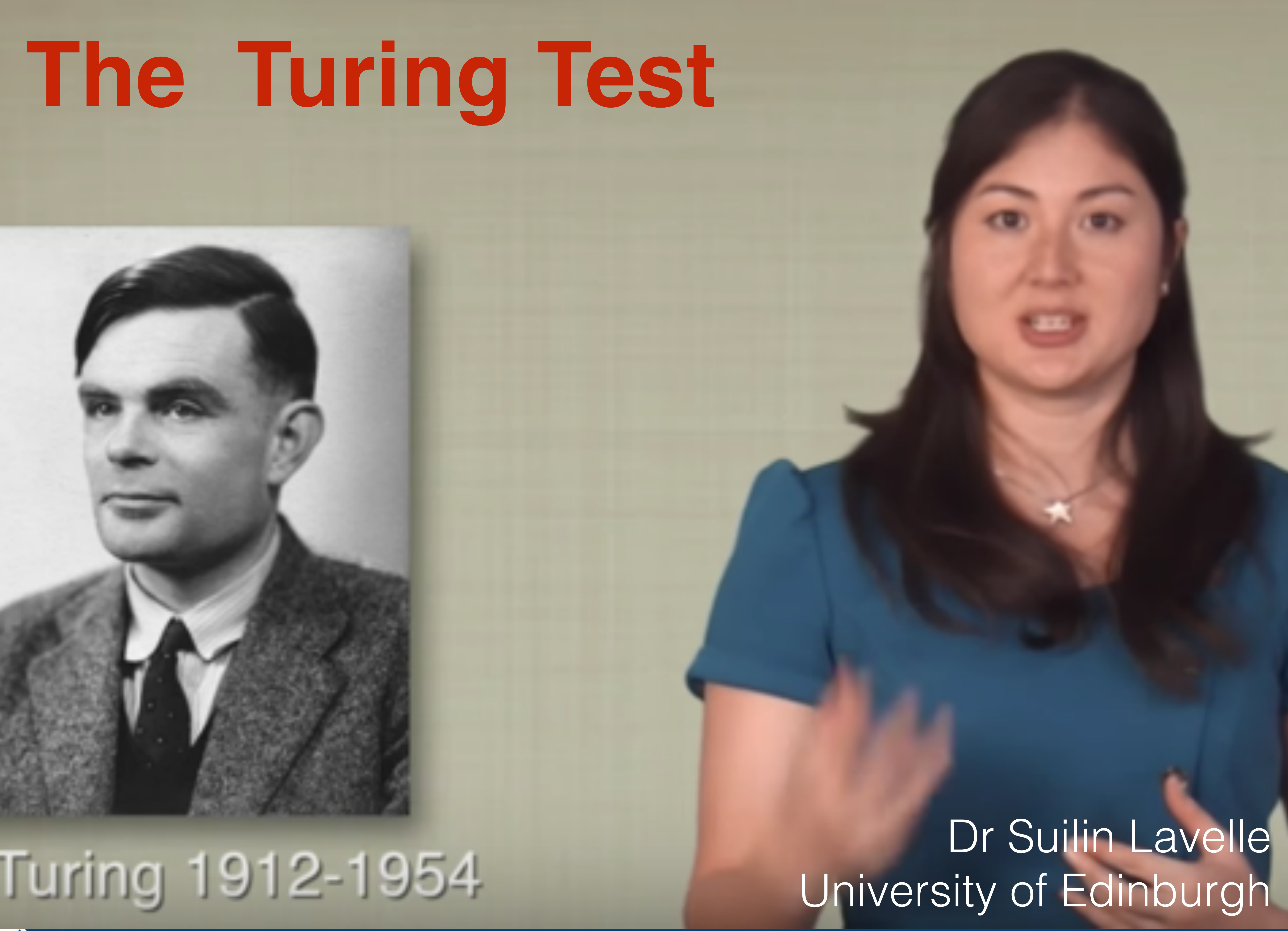
Concepts to Explore With the Bouncing Ball

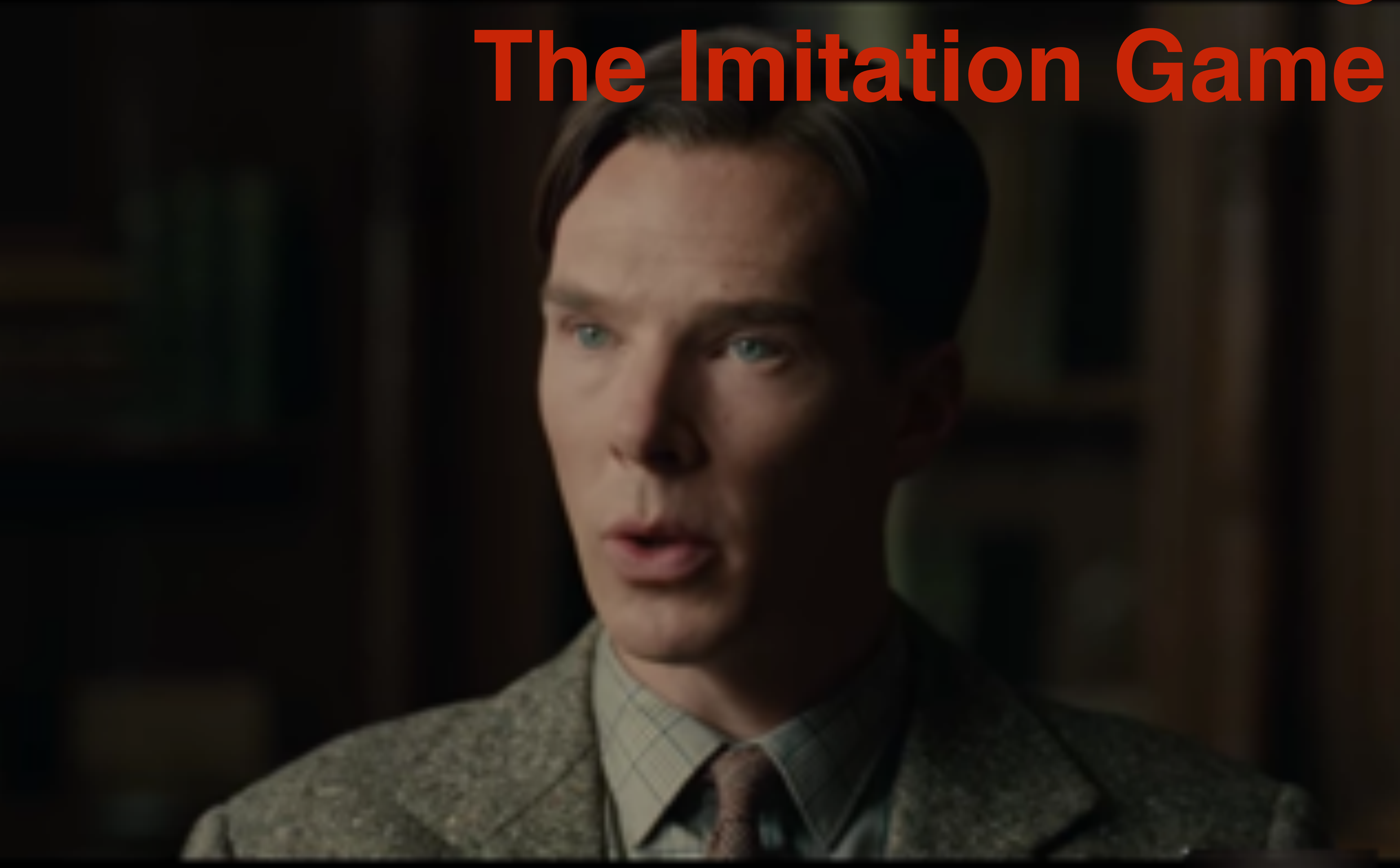
- Do this on your own (not in the lab… The lab is dense this week!)
- Multiple balls
- Balls lose energy every time they hit a wall
- Balls lose energy as they move around

Outline

Eliza

The Turing Test





Turing
The Imitation Game

BBC Series On Turing

Blade Runner

Eliza

- Example of Natural Language Processing (NLP)
- MIT, 1964, Joseph Weizenbaum
- One of the first chat-bots (Amazon Alexa)
- Emulates a **Rogerian** psychotherapist
- Example of dialogs: https://web.stanford.edu/group/SHR/4-2/text/dialogues.html

First Python Version of Eliza

```python
# Eliza1.py
# D. Thiebaut
# A very short beginning program for Eliza
# ---------------------------------------------------------------------------
# just print the string to the console
# will be transformed to something better later...
def myprint( string ):
    print( string )

# sayGoodBye
# say goodbye to the user.
def sayGoodBye( name ):
    myprint( "Good bye " + name )

# isGoodBye
# checks to see if what the user said is one of the keywor
# ending the conversation.
def isGoodBye( userAnswer ):
    if userAnswer.lower().strip() in [ "bye",
                                       "goodbye", "ciao" ]:
        return True
    else:
        return False

def greetings():
    myprint( "Hello there!" )
    myprint( "What is your name?" )
    name = input( "> " )
    myprint( "Welcome " + name )
    return name
```

```python
# ----------------------------------------
# main function
# ----------------------------------------
def main():

    # greet user and get her name
    userName = greetings()

    # conversation: get user input, and respond
    for i in range( 1000 ):

        # get user's statement
        userAnswer = input( "> " )

        # if it is a goodbye statement, exit the loop
        if isGoodBye( userAnswer ) == True:
            break

        # tell the user to continue speaking
        myprint( "Please tell me more..." )


    # if we're here, it's because the loop stopped.
    # say goodbye to the user
    sayGoodBye( userName )


main()
```

Adding Randomness To Eliza's Dialogs

- Use the random library

```
from random import *
…

prompts = ["Please go on...", "Please tell me more...",
           "Interesting... Go on, please!",
           "Yes?  Really? Go on",
           "Weird... I'm not sure what to think of that..." ]

…

myprint( choice( prompts ) )
```

Looking for String Patterns

The Problem

User types: "I had a HUGE fight with my brother"

Program knows: ["mother", "father", "brother", "sister"]

Option 1

"I had a HUGE fight with my brother"

split()

[
"I",
"had",
"a",
"huge",
"fight",
"with",
"my",
"brother"
]

in

["mother", "father", "brother", "sister"]

"I had a HUGE fight with my brother"

Option 1

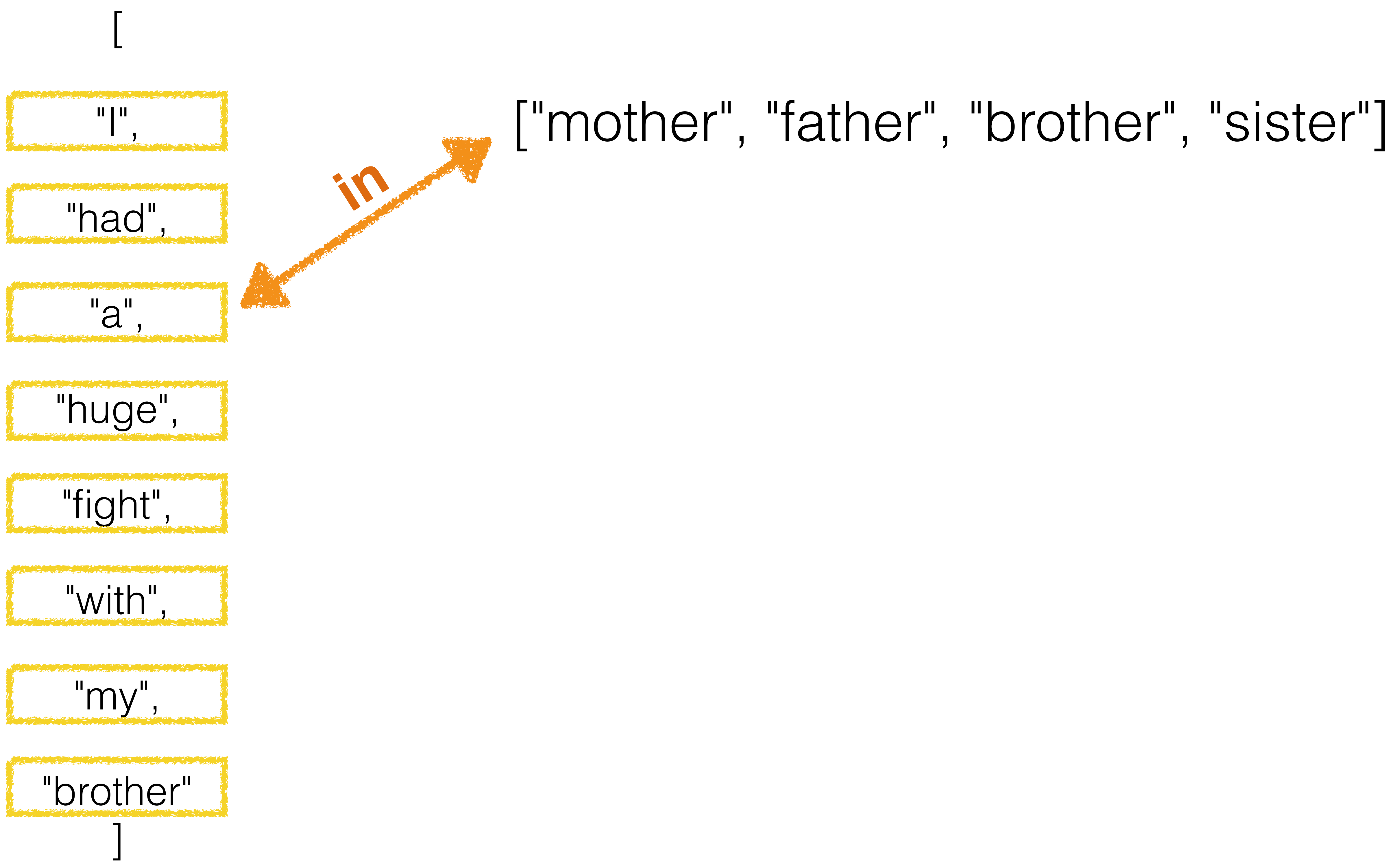
```
[
"I",
"had",
"a",
"huge",
"fight",
"with",
"my",
"brother"
]
```

in

["mother", "father", "brother", "sister"]

"I had a HUGE fight with my brother"

Option 1

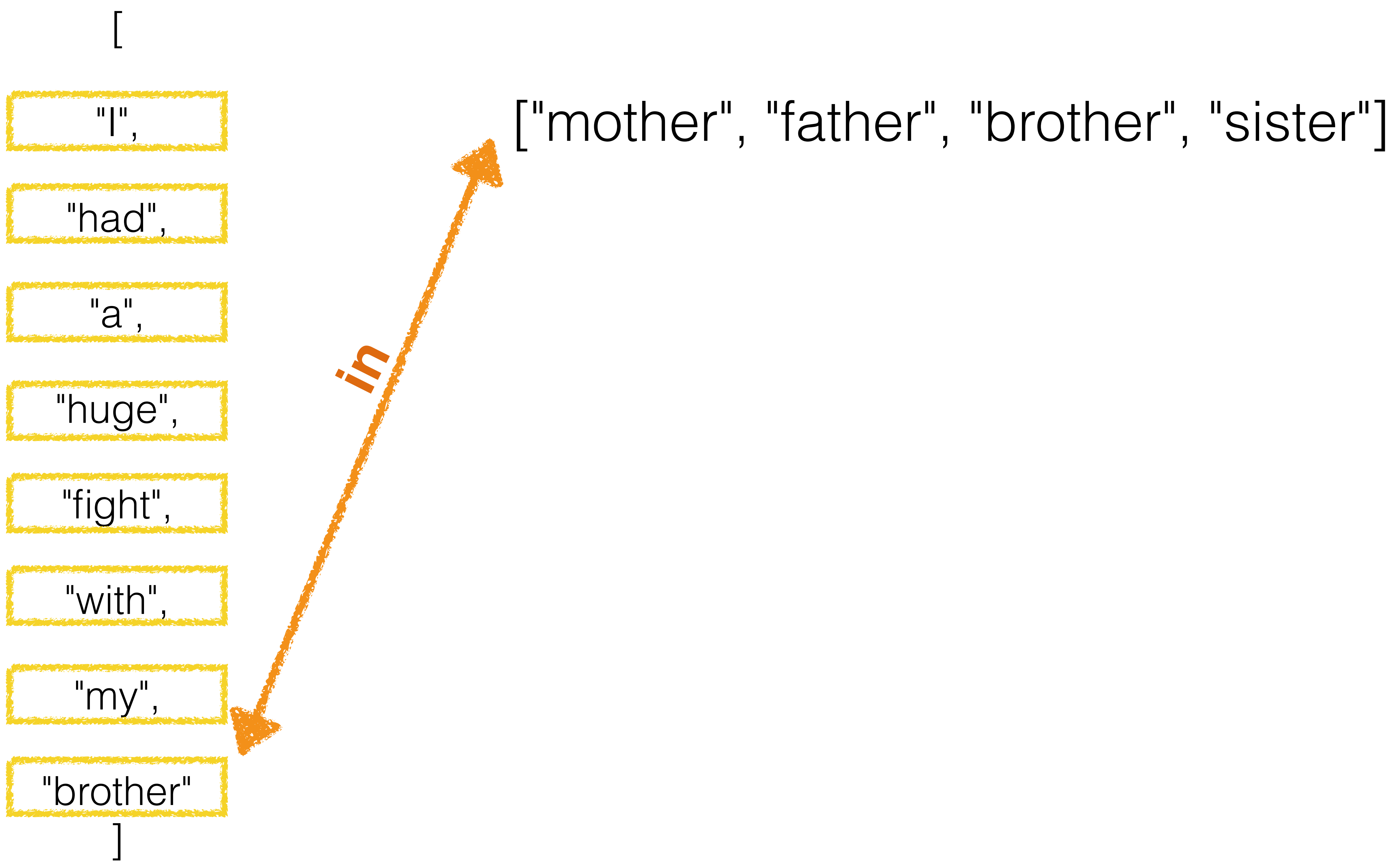


```
[
"I",
"had",
"a",
"huge",
"fight",
"with",
"my",
"brother"
]
```

["mother", "father", "brother", "sister"]

"I had a HUGE fight with my brother"

Option 1



```
[
"I",
"had",
"a",
"huge",
"fight",
"with",
"my",
"brother"
]
```

```
["mother", "father", "brother", "sister"]
```

"I had a HUGE fight with my brother"

Option 1

[
"I",
"had",
"a",
"huge",
"fight",
"with",
"my",
"brother"
]

["mother", "father", "brother", "sister"]

```
family = ["mother", "father", "brother",
"sister"]

userInput = input( "> " )
words = userInput.lower().split()

familyMatter = False
for word in words:
    if word in family:
        familyMatter = True

if familyMatter == True:
    doSomething()
```

Option 2

"I had a HUGE fight with my brother"

```
[
"mother",
"father",
"brother",
"sister"
]
```

Option 2

"I had a HUGE fight with my brother" $\xrightarrow{\text{.find}}$

[

$\boxed{\text{"mother",}}$

"father",

"brother",

"sister"

]

Option 2

"I had a HUGE fight with my brother"

$\xrightarrow{\text{.find}}$

[

"mother",

"father",

"brother",

"sister"

]

Option 2

"I had a HUGE fight with my brother"

.find

```
[
"mother",
"father",
"brother",
"sister"
]
```

Option 2

"I had a HUGE fight with my brother"

```
family = ["mother", "father",
          "brother", "sister"]

userInput = input( "> " ).lower()

familyMatter = False
for word in family:
    if userInput.find( word ) != -1:
        familyMatter = True


if familyMatter == True:
    doSomething()
```

[

"mother",

"father",

"brother",

"sister"

]

Ways to Make Eliza Program Smarter

TO BE DONE IN THE LAB

- Respond to "No", "Never", "Nope" with a different answer
- Detect "I xxx you" and respond with "You xxx me?"
- Add generated "You xxx me?" to canned answers

We stopped here last time…

Indefinite Loops (Chapter 8)

Reviewing For-Loops

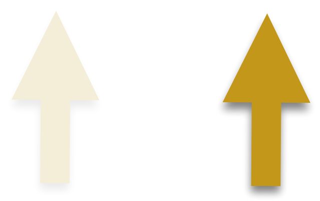
Applications

While Loops for Robustness

Break & Continue

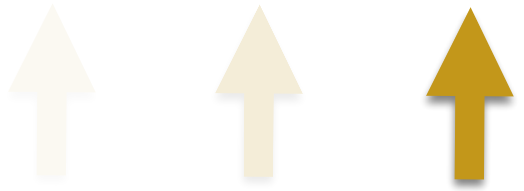
For-Loops

Items: [dog, cat, horse, hen, pig]



For-Loops

Items: [dog, cat, horse, hen, pig]

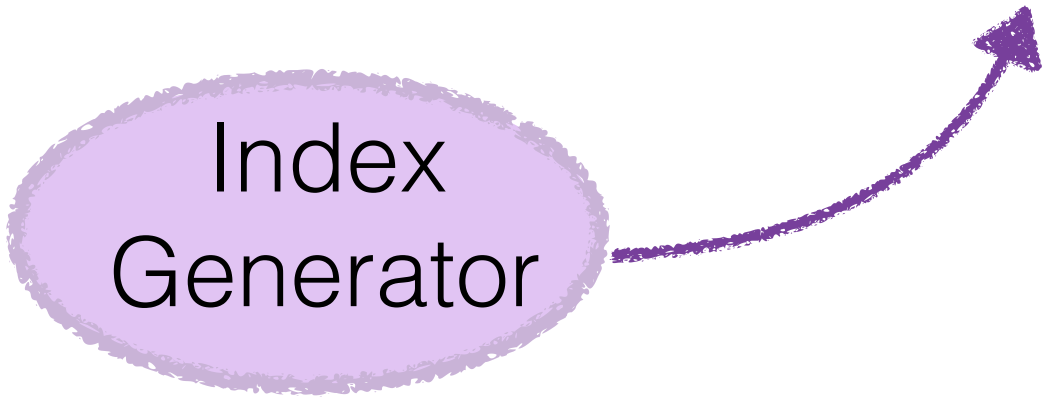


```
list = [ dog, cat, horse, hen, pig ]
for x in list:
     process( x )
```

For-Loops

0 **1** 2 3 4

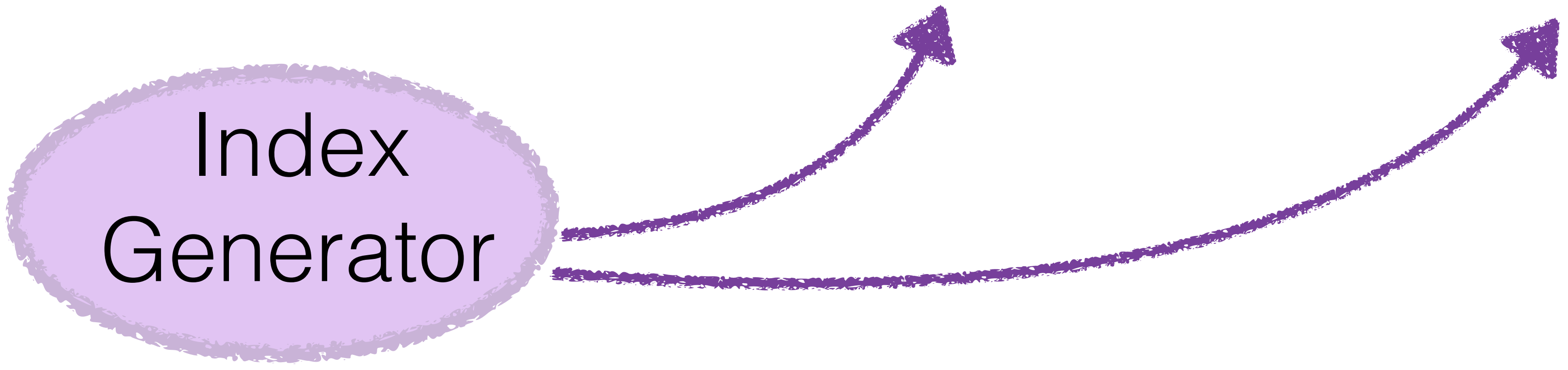
Items: [dog, cat, horse, hen, pig]



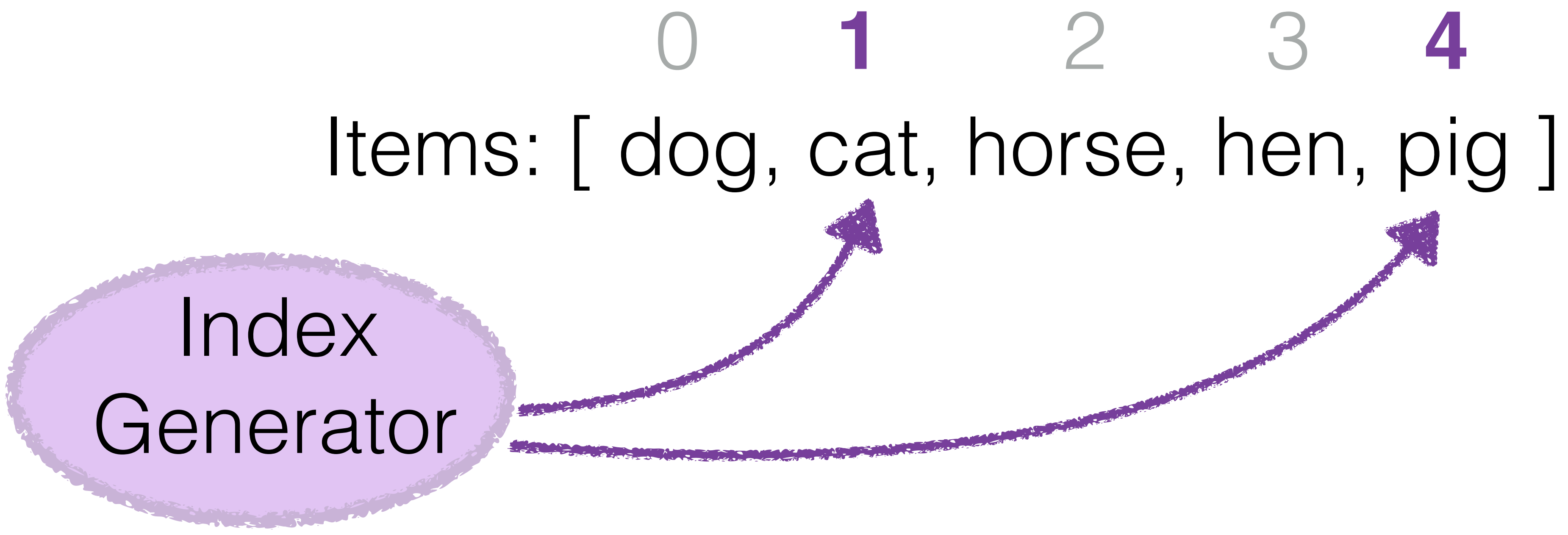
For-Loops

0 **1** 2 3 **4**

Items: [dog, cat, horse, hen, pig]



For-Loops



```
list = [ dog, cat, horse, hen, pig ]
for i in range( 1, len(list), 3):
      x = list[i]
      process( x )
```

For loops in context

Applications

While Loops for Robustness

Break & Continue

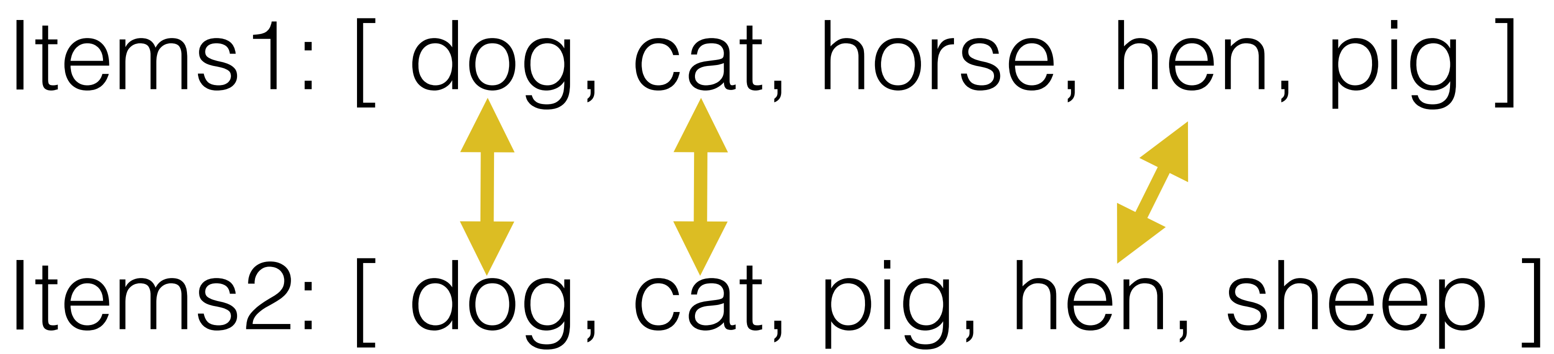
Applications

Count Matching Items

Items1: [dog, cat, horse, hen, pig]

Items2: [dog, cat, pigeon, hen, sheep]

Count Matching Items



Items1: [dog, cat, horse, hen, pig]

Items2: [dog, cat, pig, hen, sheep]

Exact Place Matching

```
#                 0      1      2         3      4
items1 = [ dog, cat, horse, hen, pig ]
items2 = [ dog, cat, pig,    hen, sheep ]

count = 0
for i in range( len( items1 ) ):
    if items1[i]==items2[i]:
        count += 1
```

What if the lists do not have the same length?

```
#                 0    1    2      3    4
items1 = [ dog, cat, horse, hen, pig ]
items2 = [ dog, cat, pig ]

count = 0
for ???:
    if items1[i]==items2[i]:
        count += 1
```

What is the risk?

What could go wrong?

What kind of error?

What is the risk?

What could go wrong?

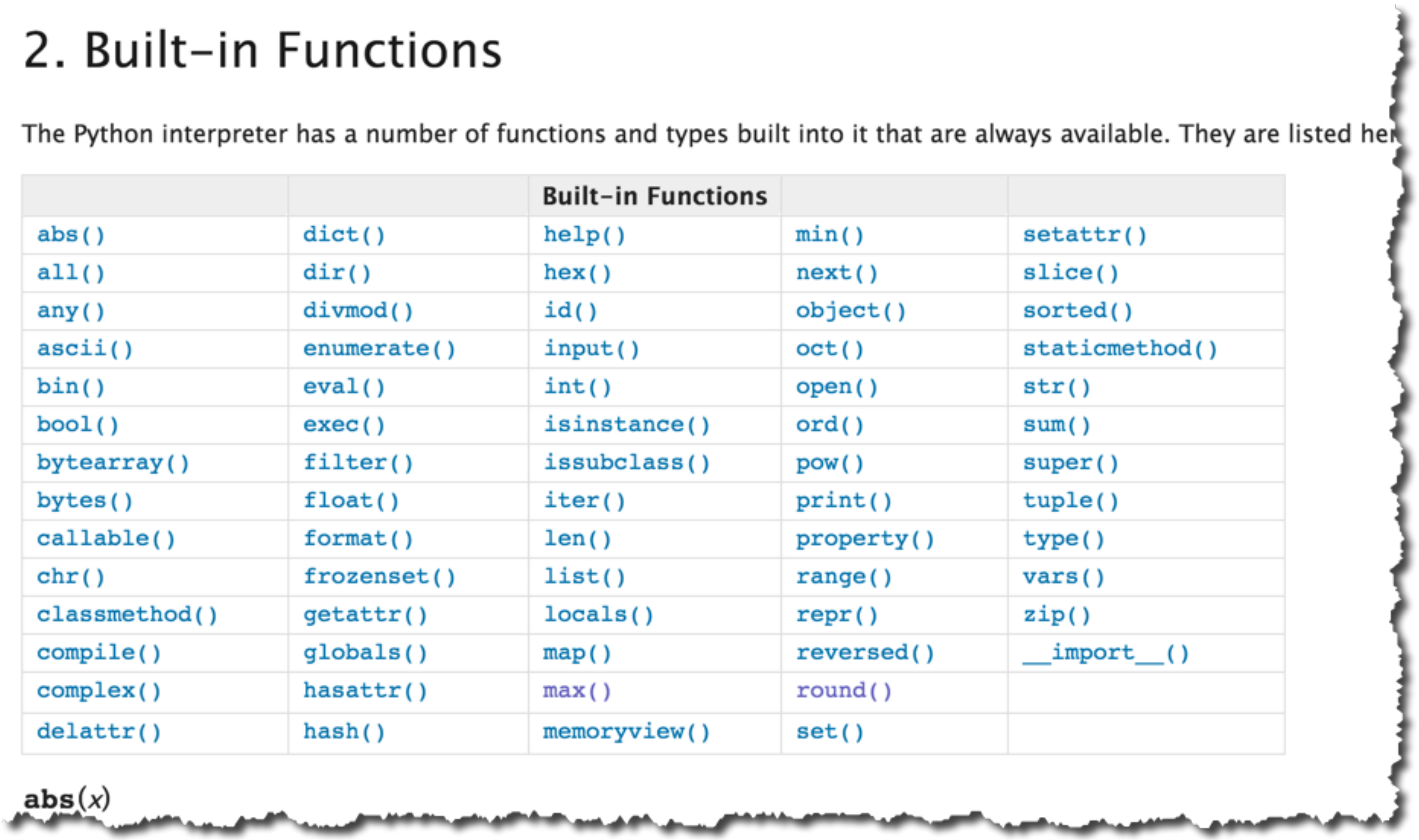
What kind of error?

What if the lists do not have the same length?

```
#                 0    1    2       3    4
items1 = [ dog, cat, horse, hen, pig ]
items2 = [ dog, cat, pig ]

count = 0
for i in range( len( items1 ) ):
    if items1[i]==items2[i]:
       count += 1
```

items1 may not be the longest list

```
#                0     1     2     3     4
items1 = [ dog, cat, horse]
items2 = [ dog, cat, pig, hen, pig ]

count = 0
for i in range( len( items1 ) ):
    if items1[i]==items2[i]:
        count += 1
```



2. Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed he

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

abs(*x*)

https://docs.python.org/3.4/library/functions.html

```
#               0    1    2    3    4
items1 = [ dog, cat, horse]
items2 = [ dog, cat, pig, hen, pig ]

count = 0
len1 = len( items1 )
len2 = len( items2 )
for i in range( min( len1, len2 ) ):
    if items1[i]==items2[i]:
       count += 1
```

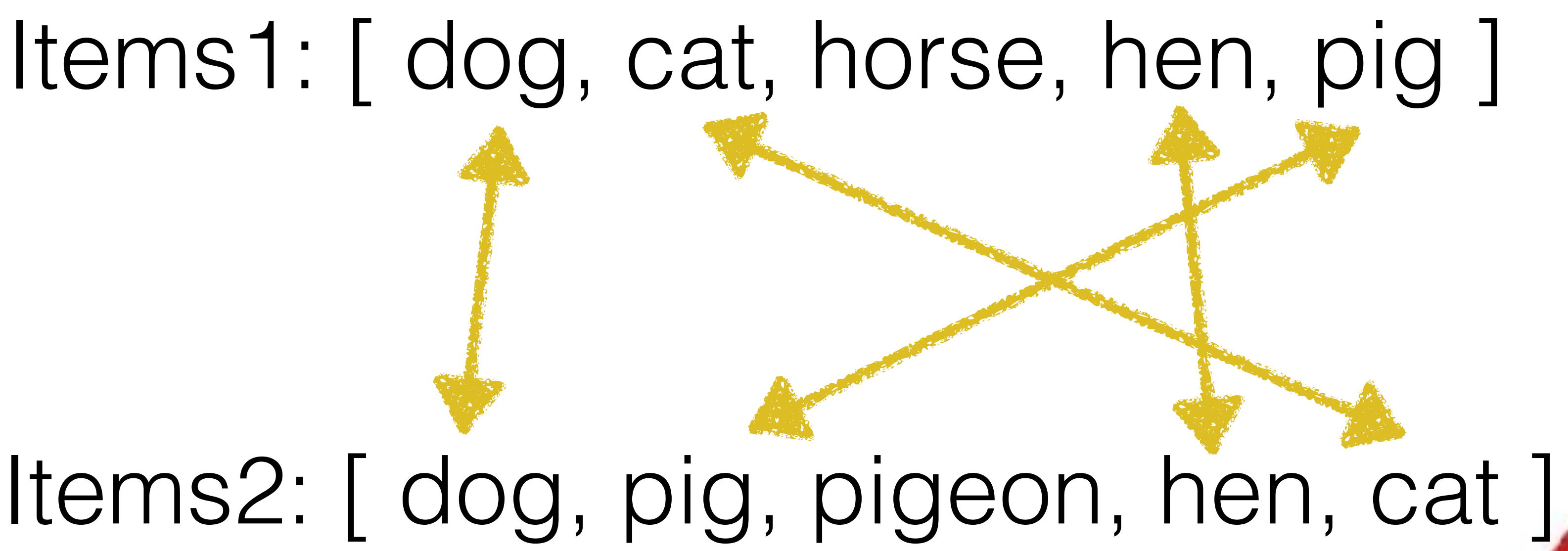
Applications, #2

Count Matching Misplaced Items

Items1: [dog, cat, horse, hen, pig]

Items2: [cat, pig, pigeon, hen, dog]

Count Matching Misplaced Items


Items1: [dog, cat, horse, hen, pig]

Items2: [dog, pig, pigeon, hen, cat]

Algorithm

```
         0     1     2      3     4
Items1: [ dog, cat, horse, hen, pig ]

         0    1     2       3     4
Items2: [ dog, pig, pigeon, hen, cat ]
```

Algorithm

```
          i
          0      1      2      3      4
Items1: [ dog, cat, horse, hen, pig ]
          j
          0      1      2        3      4
Items2: [ dog, pig, pigeon, hen, cat ]
```

Algorithm

```
          i
          0     1     2      3     4
Items1: [ dog, cat, horse, hen, pig ]
          j
          0    1     2        3      4
Items2: [ dog, pig, pigeon, hen, cat ]
```

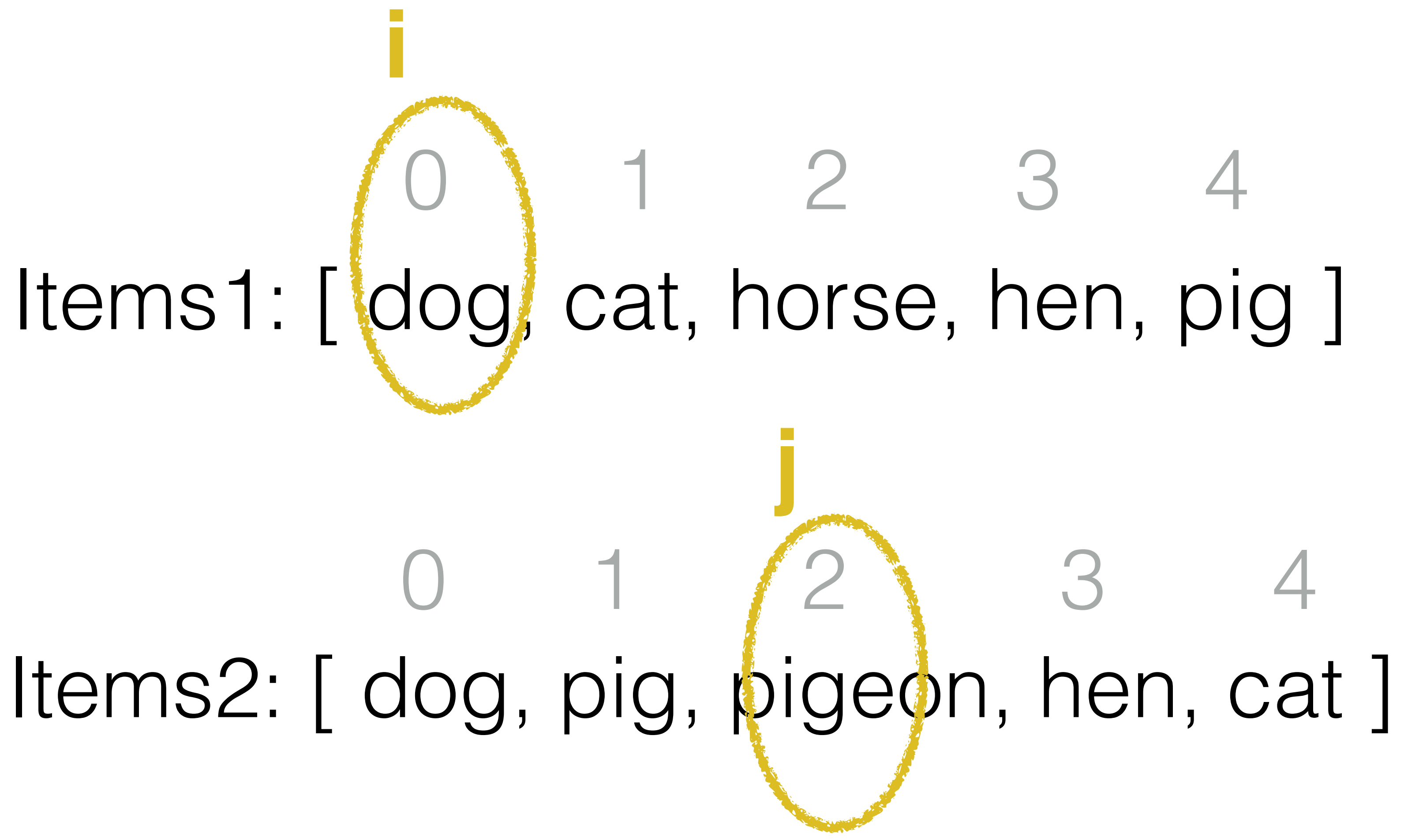
count: 1

Algorithm

```
               i
               0      1      2      3      4
Items1: [ dog, cat, horse, hen, pig ]
                     j
               0      1      2         3      4
Items2: [ dog, pig, pigeon, hen, cat ]
```

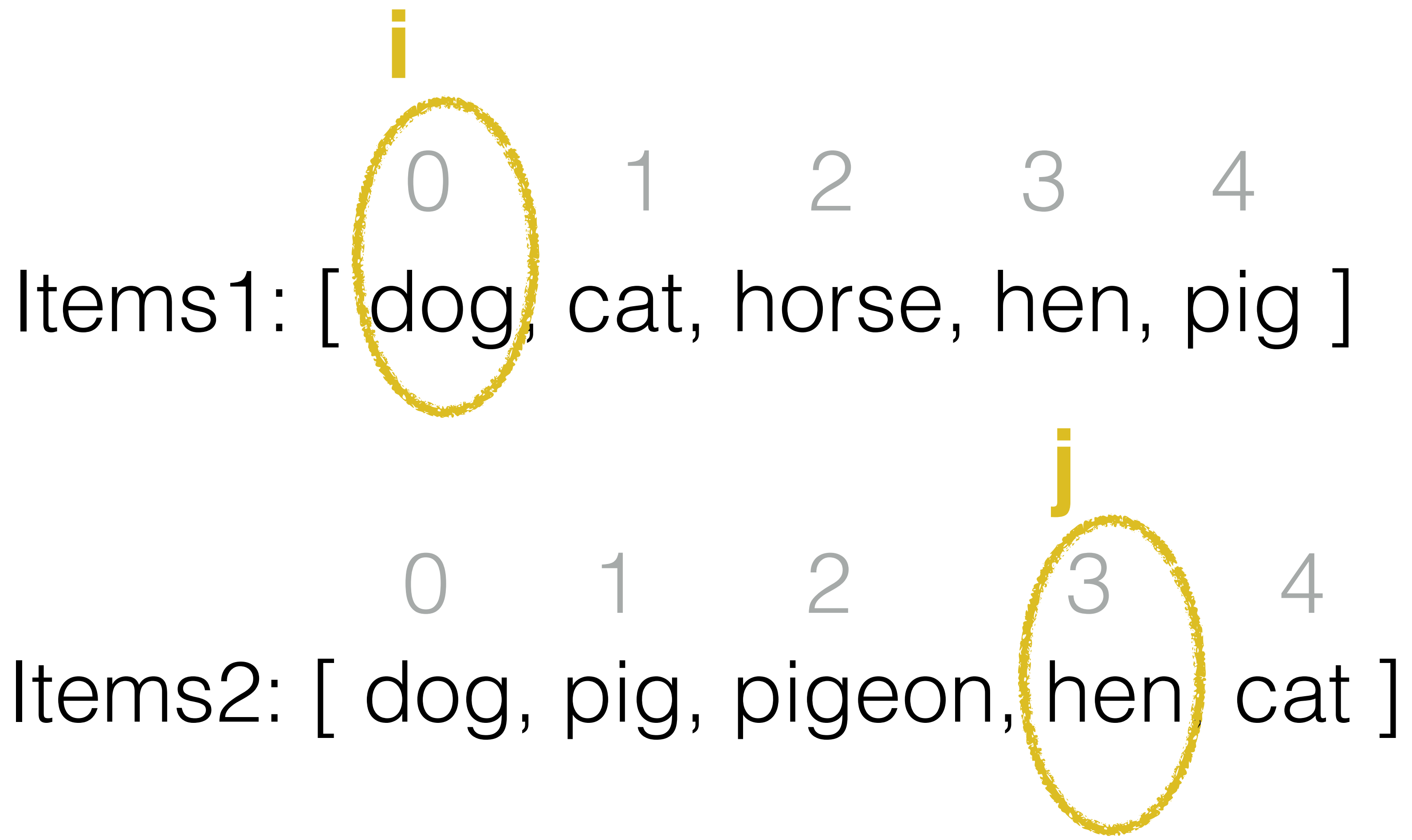
count: 1

Algorithm


i
0 1 2 3 4
Items1: [dog, cat, horse, hen, pig]
j
0 1 2 3 4
Items2: [dog, pig, pigeon, hen, cat]

count: 1

Algorithm


i
0 1 2 3 4
Items1: [dog, cat, horse, hen, pig]
j
0 1 2 3 4
Items2: [dog, pig, pigeon, hen, cat]

count: 1

Algorithm

```
           i
           0     1     2      3     4
Items1: [ dog, cat, horse, hen, pig ]

                                    j
           0    1     2        3     4
Items2: [ dog, pig, pigeon, hen, cat ]
```

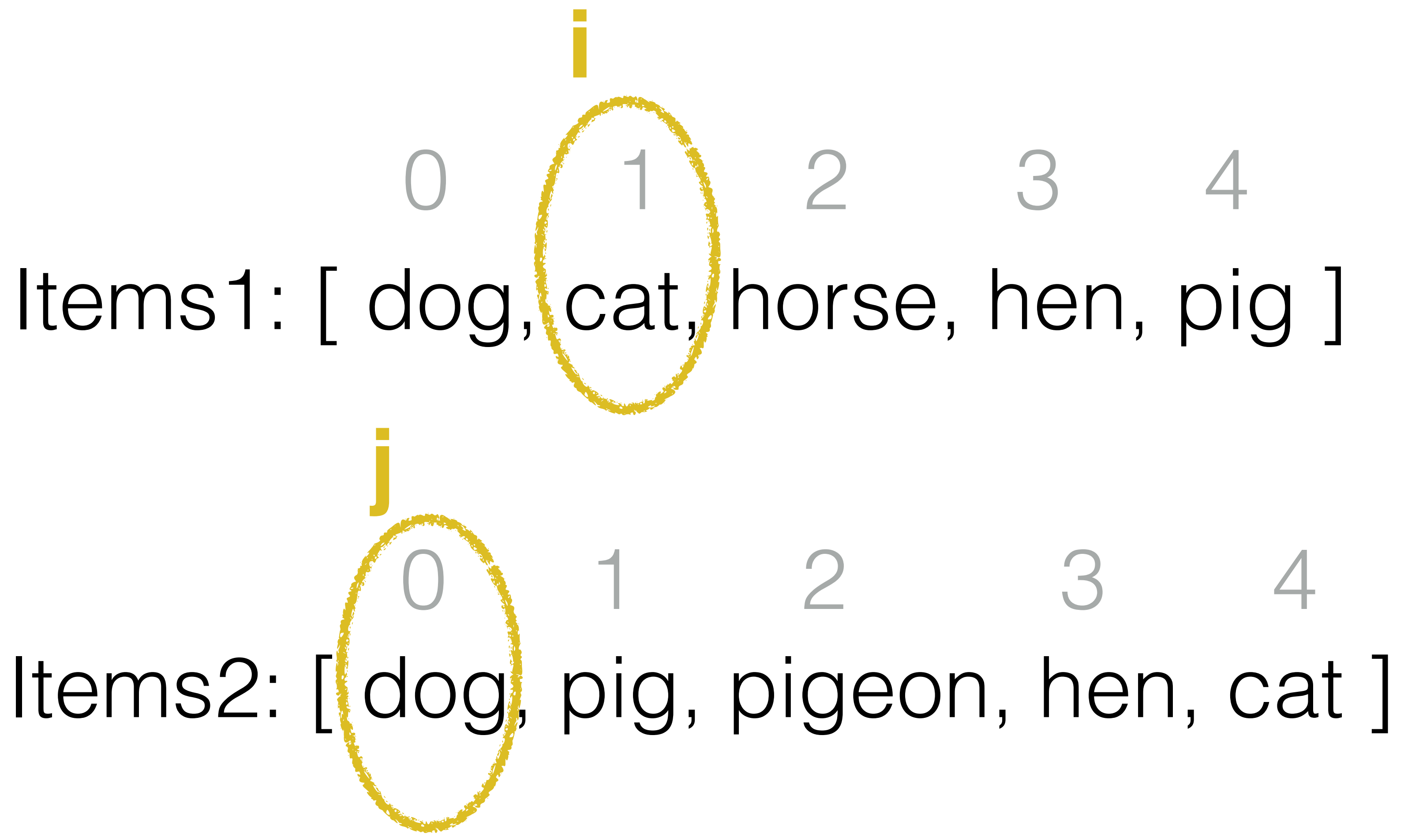
count: 1

Algorithm

```
                 i
        0        1       2        3       4
Items1: [ dog,  cat,   horse,   hen,    pig ]

        0       1       2          3        4
Items2: [ dog,  pig,   pigeon,   hen,    cat ]
```

count: 1

Algorithm


i
0 1 2 3 4
Items1: [dog, cat, horse, hen, pig]
j
0 1 2 3 4
Items2: [dog, pig, pigeon, hen, cat]

count: 1

Algorithm

```
                  i
          0       1       2       3       4
Items1: [ dog, cat, horse, hen, pig ]
                  j
          0       1       2         3       4
Items2: [ dog, pig, pigeon, hen, cat ]
```

count: 1

Algorithm

i

0 1 2 3 4

Items1: [dog, cat, horse, hen, pig]

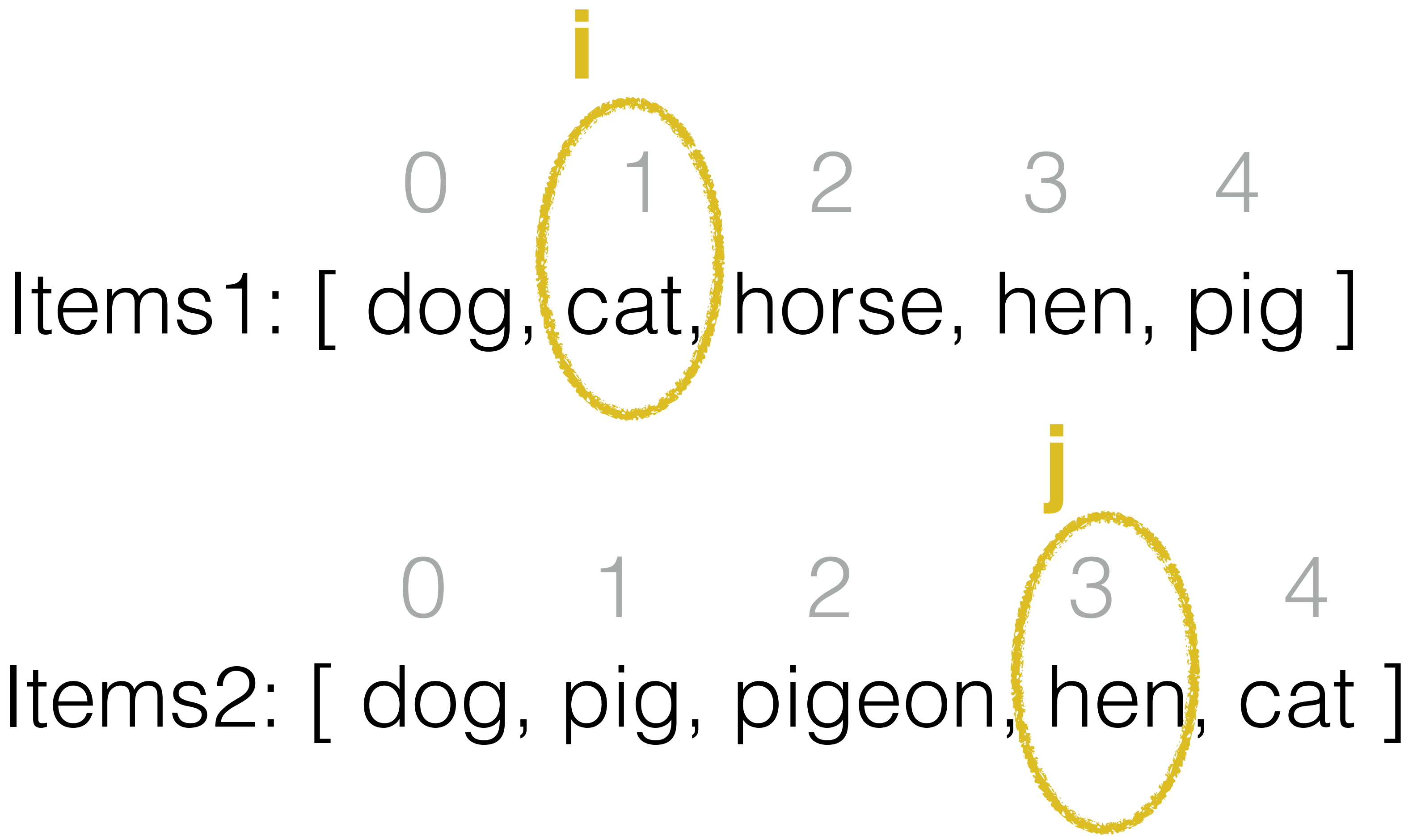
j

0 1 2 3 4

Items2: [dog, pig, pigeon, hen, cat]

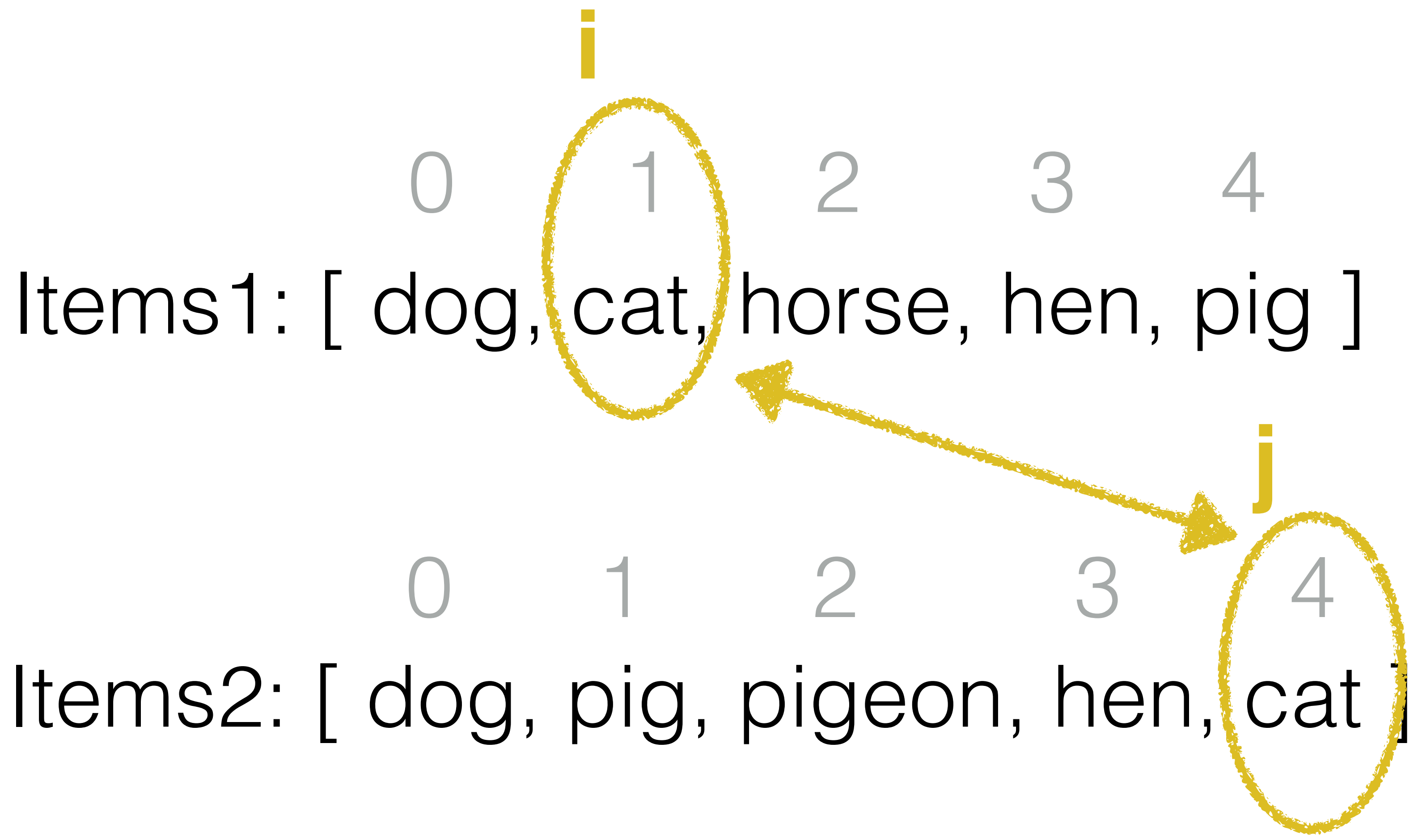
count: 1

Algorithm


i
0 1 2 3 4
Items1: [dog, cat, horse, hen, pig]
j
0 1 2 3 4
Items2: [dog, pig, pigeon, hen, cat]

count: 1

Algorithm


i
0 1 2 3 4
Items1: [dog, cat, horse, hen, pig]
j
0 1 2 3 4
Items2: [dog, pig, pigeon, hen, cat]

count: ~~1~~ 2

```
#                0    1     2       3    4
items1 = [ dog, cat, horse,   hen, pig]
items2 = [ dog, pig, pigeon, hen, cat]

count = 0

for i in range( len( items1 ) ):
    for j in range( len( items2 ) ):
        if items1[i]==items2[j]:
            count += 1
```

For loops in context

Applications

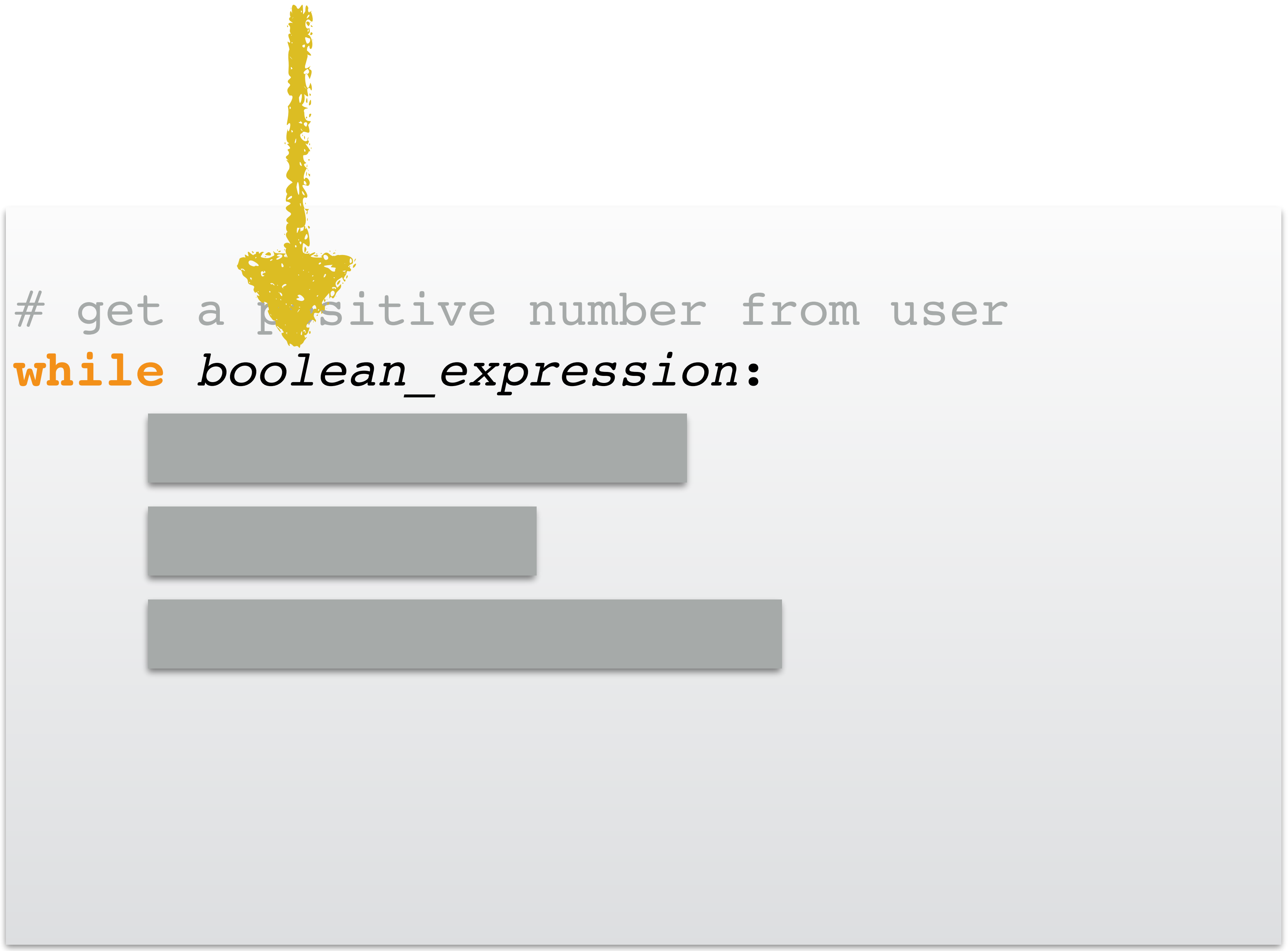
While Loops for Robustness

Break & Continue

While Loop

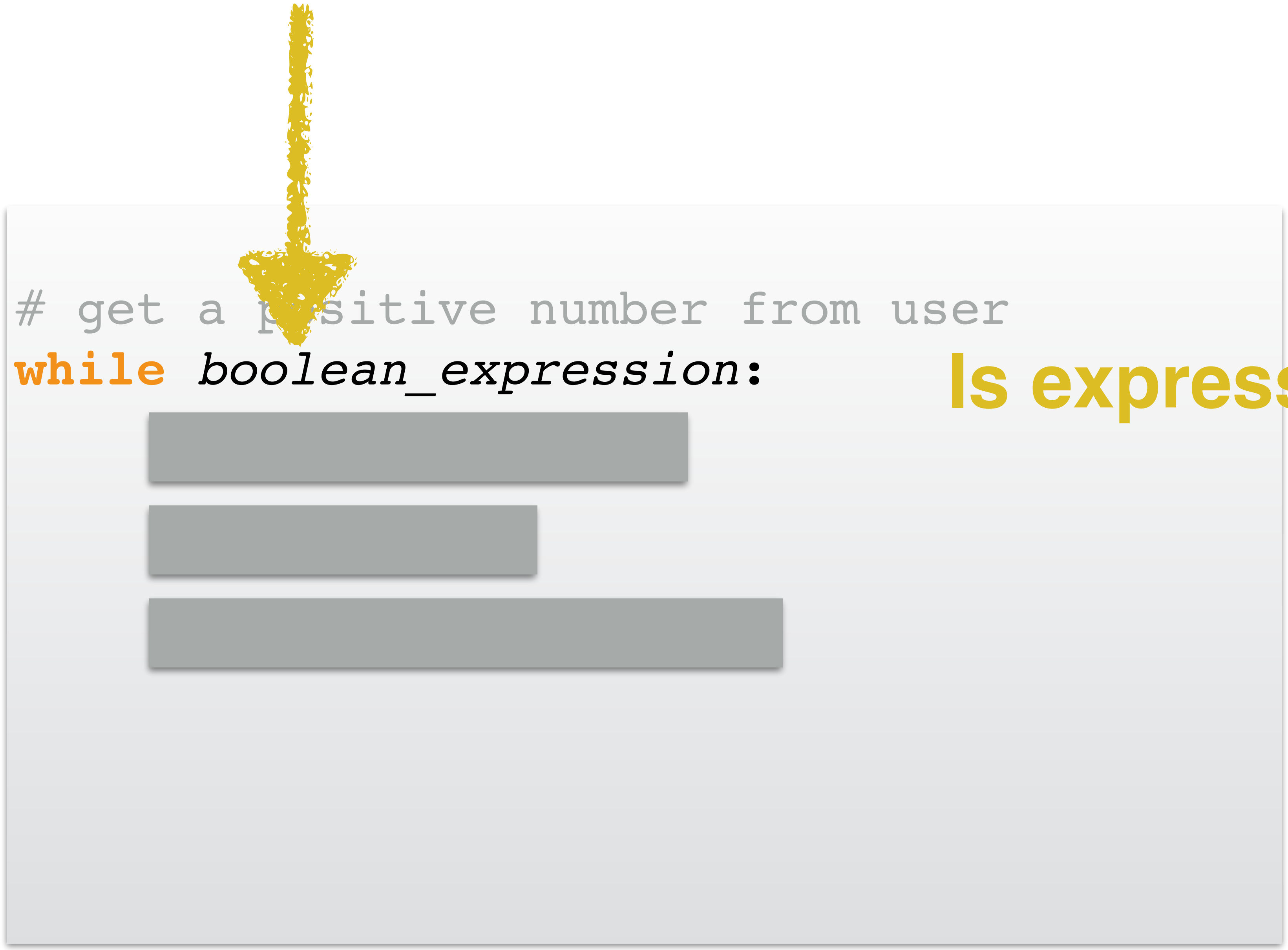
```
# get a positive number from user
while boolean_expression:
```

While Loop



```
# get a positive number from user
while boolean_expression:
```

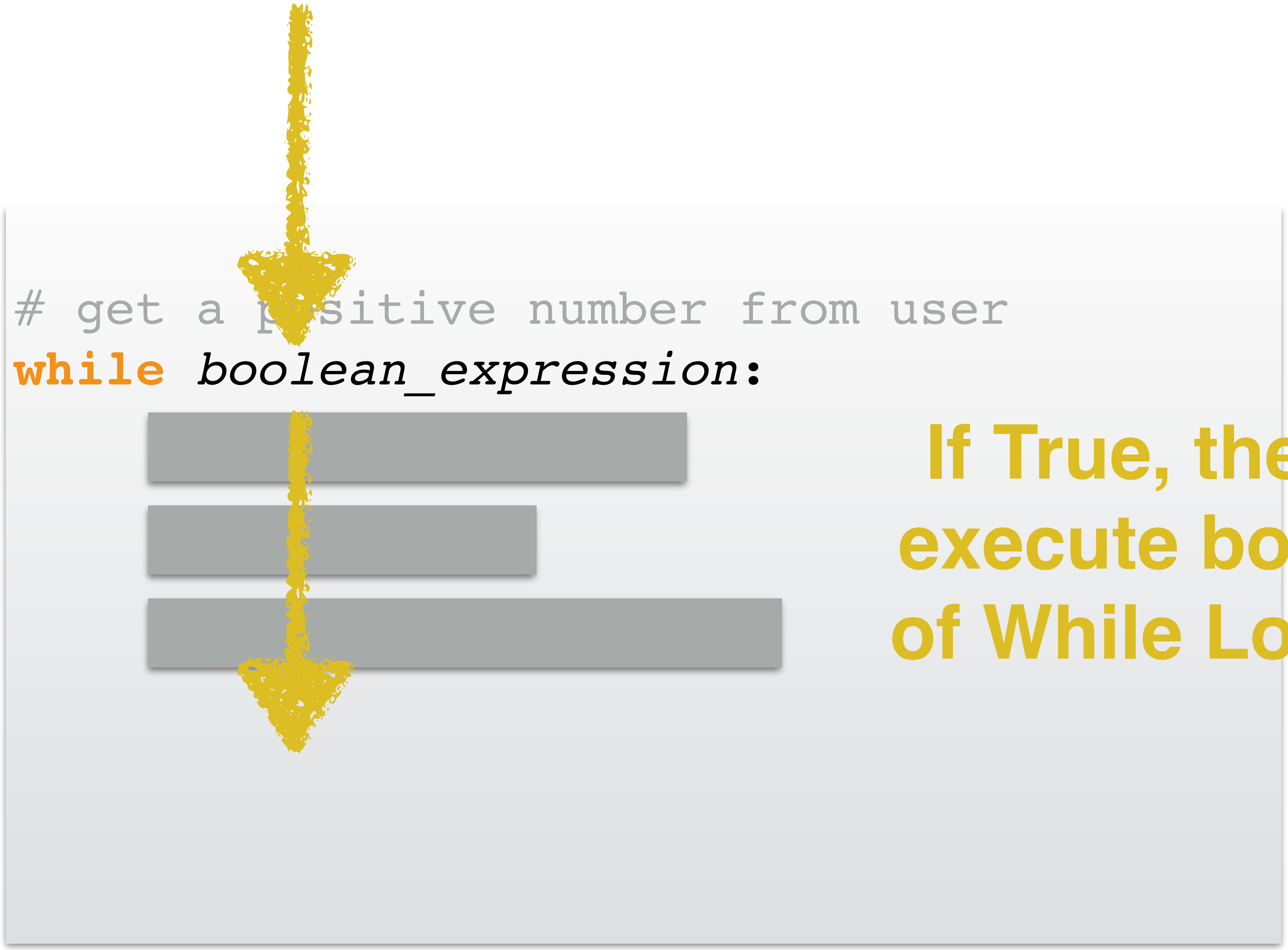
While Loop



```
# get a positive number from user
while boolean_expression:
```

Is expression True?

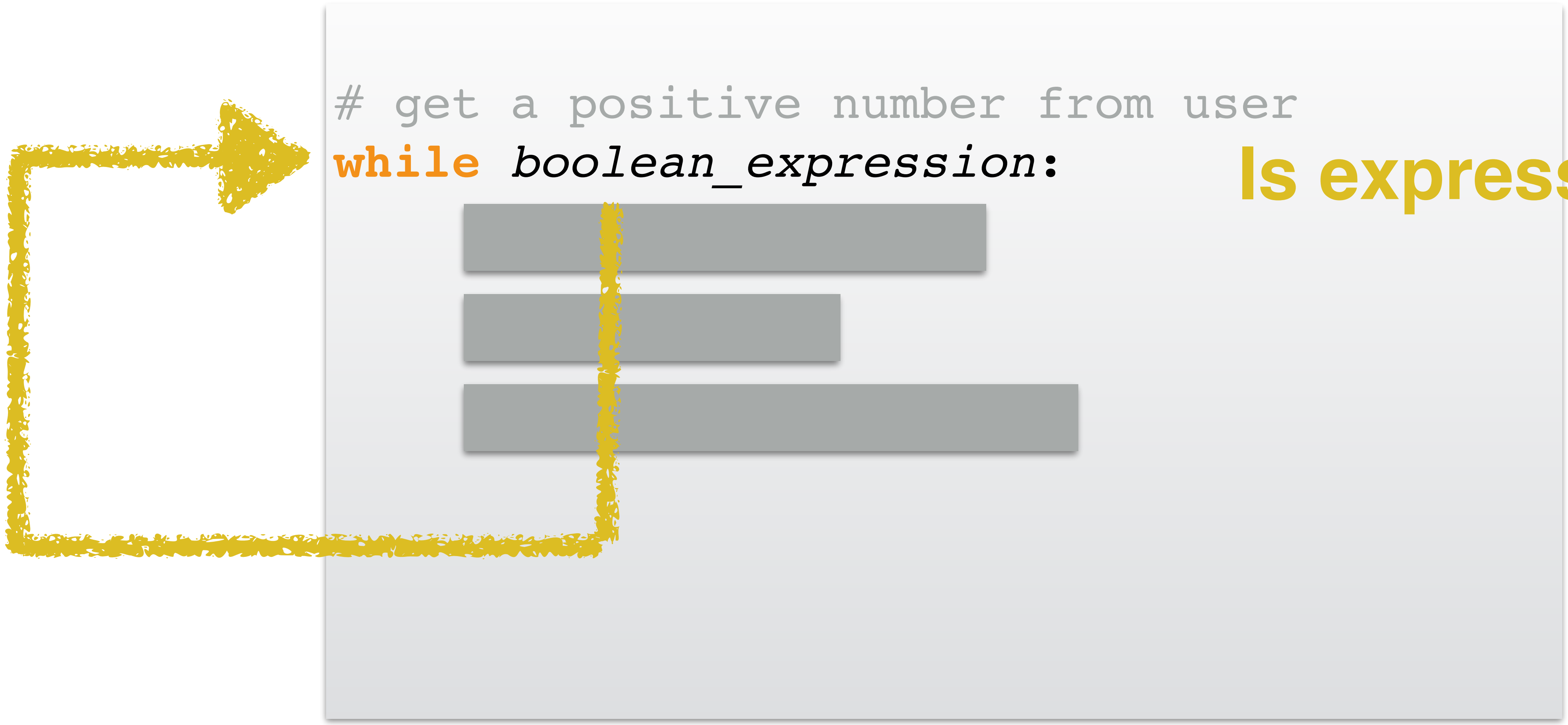
While Loop



```
# get a positive number from user
while boolean_expression:
```

If True, then execute body of While Loop

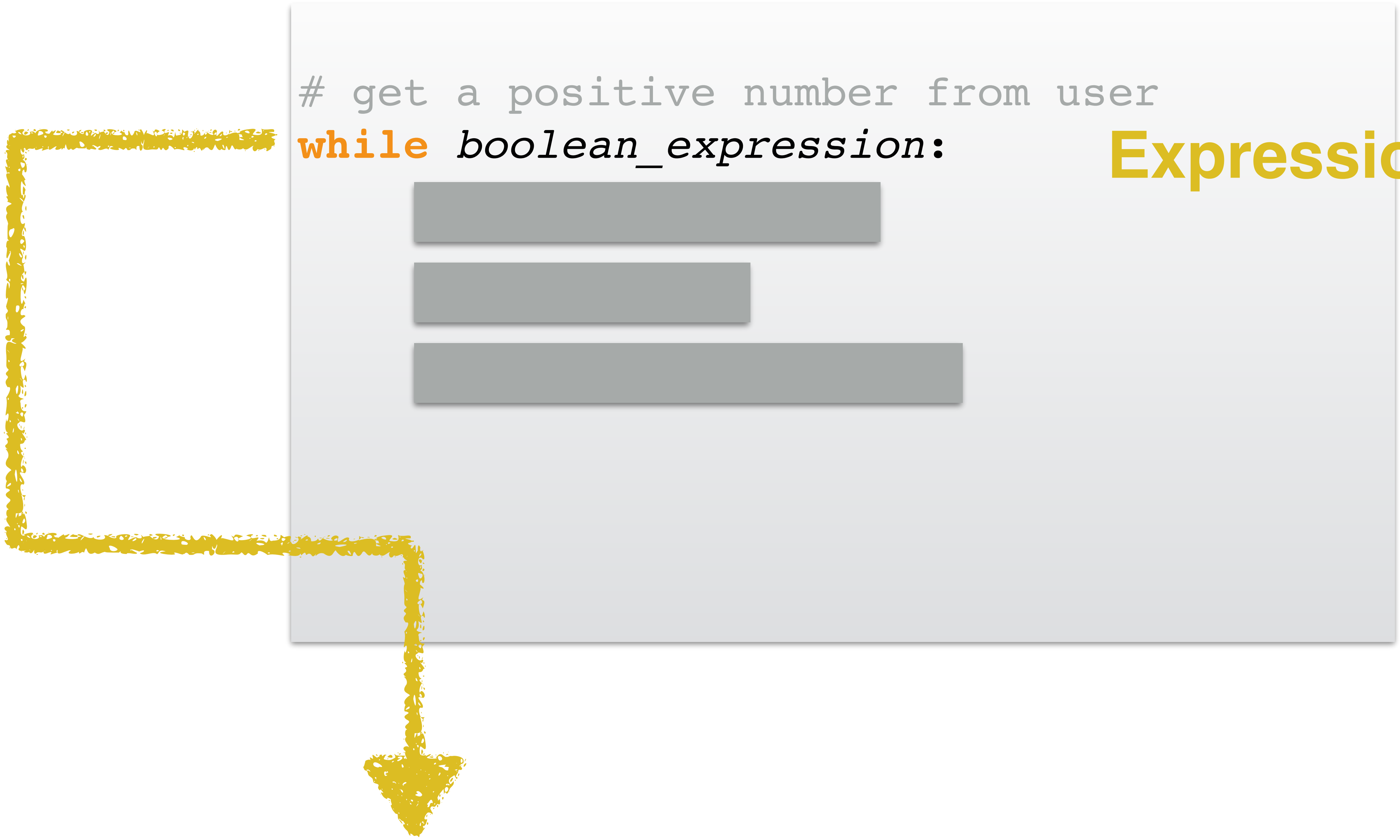
While Loop



```
# get a positive number from user
while boolean_expression:
```

Is expression True?

While Loop



```
# get a positive number from user
while boolean_expression:
```

Robust Input With While Loops

- Example 1: While quantity not valid
- Example 2: While item not in list

```
# get a positive number from user
x = eval( input( "> " ) )

. . .
```

What if user enters negative number?

```
# get a positive number from user
x = eval( input( "> " ) )

. . .
```

Solution: keep on asking until input is ok

```
# get a positive number from user
x = eval( input( "> " ) )
while x < 0:
    x = eval( input( "Invalid number\n> " ) )
```

Write Robust Functions That Prompt for Quantities

Solution 1

```
# get a positive number from user
def getPositiveInt():
   x = int( input( "> " ) )
   while x < 0:
      x = int( input( "Invalid number\n> " ) )
   return x


x = getPositiveInt()
```

Robust Input With While Loops

- Example 1: While quantity not valid
- Example 2: While item not in list

```
# get a YES/NO answer from user
def getAnswerYesNo():
   x = input( “Continue (Yes/No)? " ) )
   while ???:
      print( “Invalid input, must be YES or NO” )
      x = input( “Continue (Yes/No)? " ) )
   return x


ans = getAnswerYesNo()
```

```
# get a YES/NO answer from user
def getAnswerYesNo():
   x = input( “Continue (Yes/No)? " ) ).upper()

   while ( x in [ “YES”, “NO” ] ) == False:
      print( “Invalid input, must be YES or NO” )
      x = input( “Continue (Yes/No)? " ) ).upper()

   return x


ans = getAnswerYesNo()
```

Alternative Coding
(harder to grasp, but shorter)

```
# get a YES/NO answer from user
def getAnswerYesNo():
    x = input( “Continue (Yes/No)? " ) ).upper()

    while not ( x in [ “YES”, “NO” ] ):
        print( “Invalid input, must be YES or NO” )
        x = input( “Continue (Yes/No)? " ) ).upper()

    return x


ans = getAnswerYesNo()
```

For loops in context

Applications

While Loops for Robustness

Break & Continue

Break and **Continue**

Monopoly = loop

break = ?

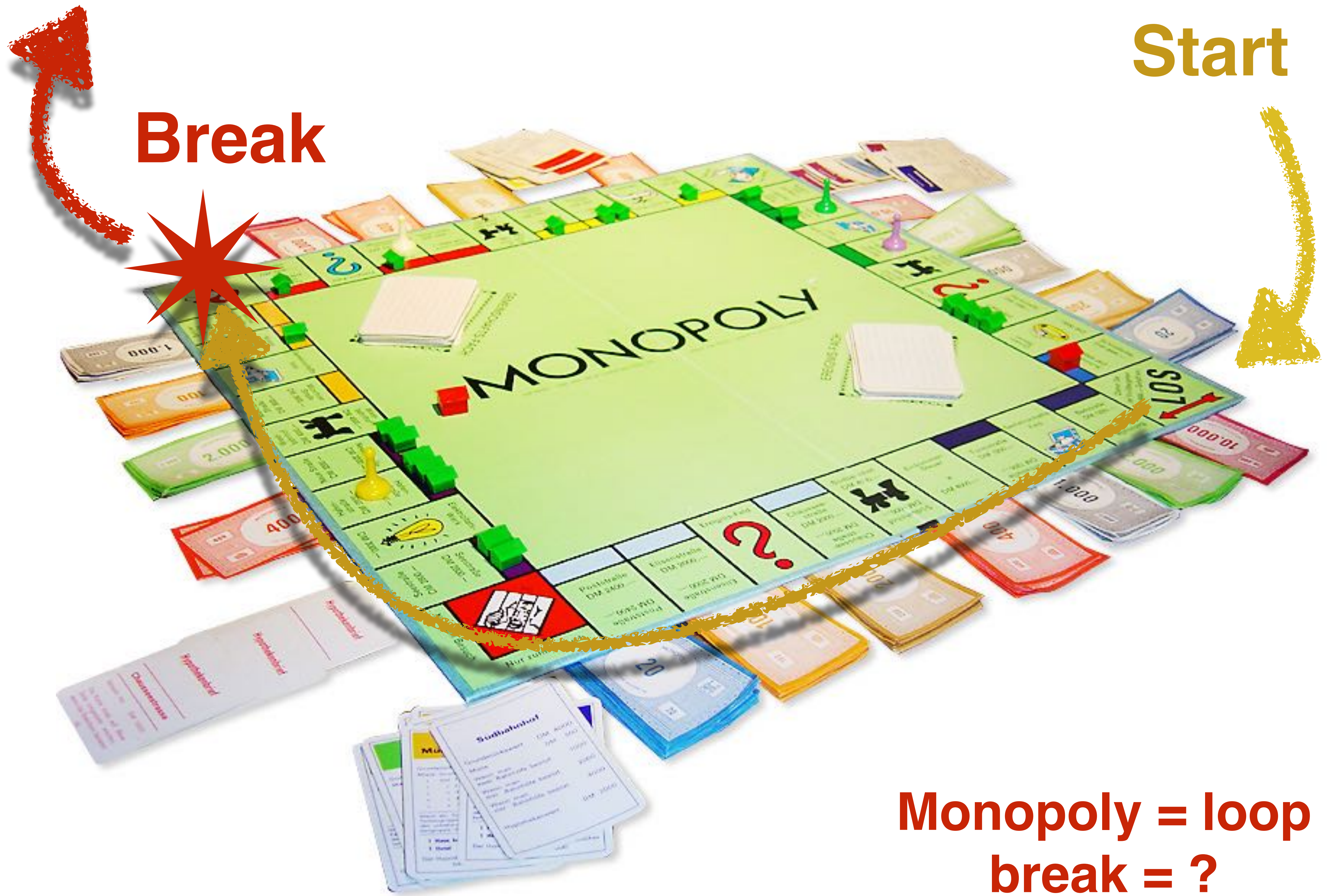
continue = ?

http://en.wikipedia.org/wiki/File:German_Monopoly_board_in_the_middle_of_a_game.jpg

Start

Monopoly = loop
break = ?
continue = ?

Monopoly = loop
break = ?
continue = ?



Monopoly = loop
break = ?
continue = ?

Monopoly = loop
break = ?
continue = ?

Monopoly = loop
break = ?
continue = ?

Applying *Break* and *Continue* to Eliza