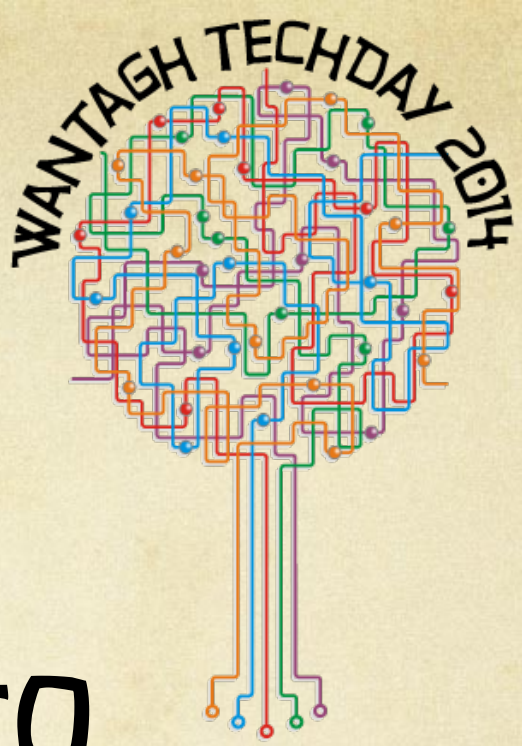
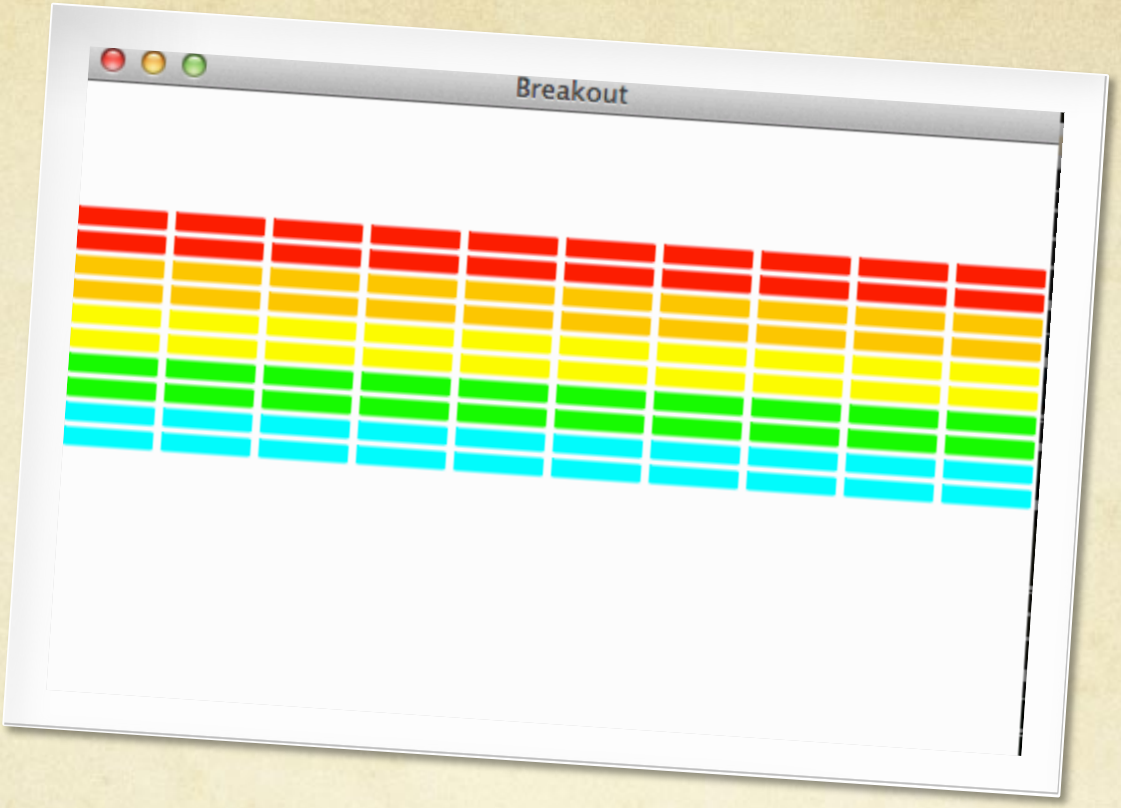


INTRODUCTION TO PROGRAMMING USING PYTHON

CS.003



LET'S PLAY!

CLICK THE GAME BREAKOUT ON YOUR SCREEN, AND GIVE IT A TRY!

Outline

- Introduction to coding
- Types (String, Int, Float, Bool, List)
- Operations on types
- Expressions
- Conditionals
- Loops

LOGIN TO YOUR GOOGLE DRIVE

http://drive.google.com
Username: schoolid@wantaghschools.org
Password: what you set it to!

If you have not yet set your password, it should be Password1, feel free to change it whenever you want!

```
        First, this method calls another hidden method _getCollidingObject(),
        and stores it in a variable called collision.  That method detects
        collisions, and if a collision does occur this method will do something,
        if not it will simply have just called the _getCollidingObject() method.
        If a collision occurs with the ball and the paddle, when the ball has a
        negative velocity, the ball's y component of the velocity is negated. If
        there is a collision with the ball and paddle, but the ball is going up,
        nothing happens. If there is a collision between the ball and a brick,
        the brick is removed, and the ball's y component of the velocity is
        negated. Also plays a specific sound depending on what object the ball
        hits."""

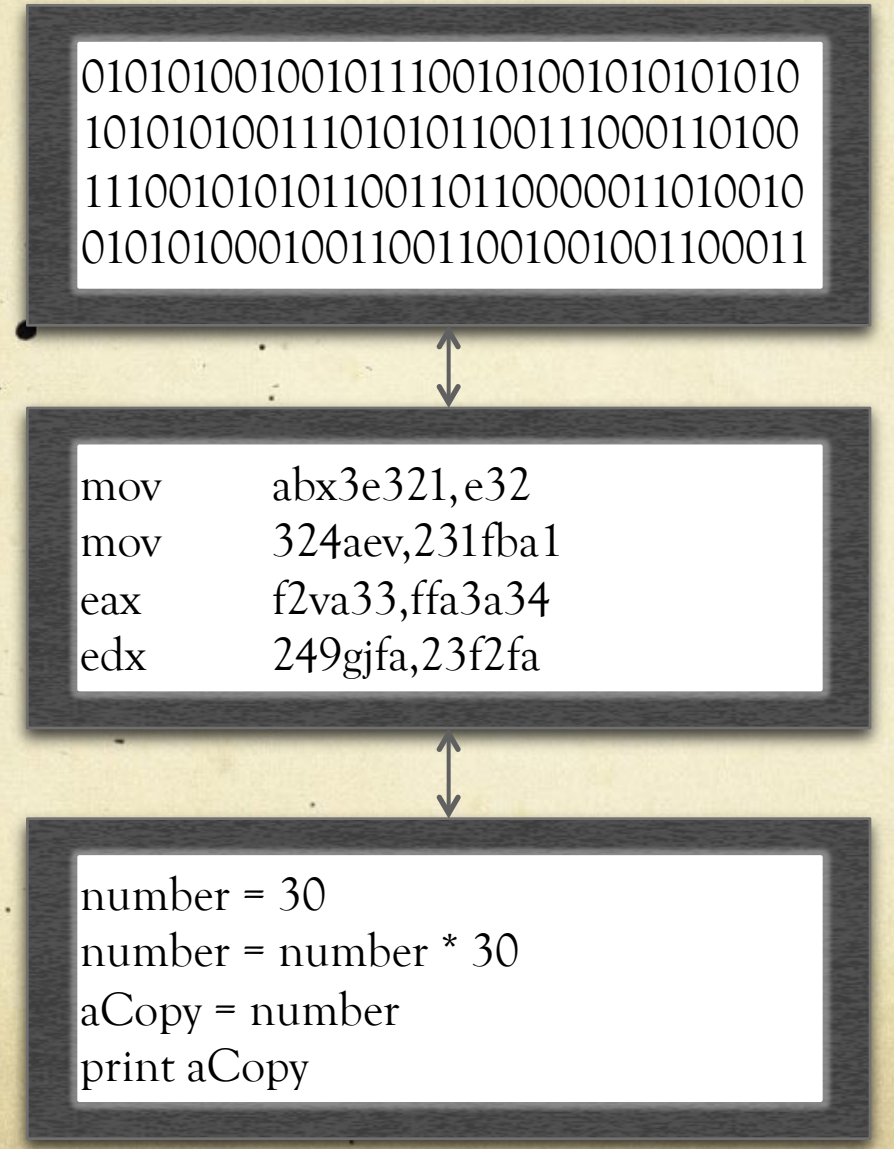
        collision = self._getCollidingObject() # Checks for Collisions

        if collision == self._paddle and self._ball.vy > 0:
            pass # nothing happens when the ball is going up and hits the paddle
        elif collision == self._paddle and self._ball.vy < 0:
            if self._soundToggle:
                paddleSound = Sound('bounce.wav')
                paddleSound.play()
            self._ball.vy = - self._ball.vy # change direction
        elif collision in self._bricks:
            if self._soundToggle:
                brickNoise = Sound(BRICKS_AUDIO[self._audioCounter])
                brickNoise.play()
                self._audioCounter = self._audioCounter + 1
                if self._audioCounter == len(BRICKS_AUDIO):
                    self._audioCounter = 0
            self._ball.vy = - self._ball.vy # change direction
            self.view.remove(collision) # remove from view
            self._bricks.remove(collision) # remove from field
            self._score = self._score + 10
            self._scoreKeeper.text = 'Current Score : '+`self._score`
```

WHAT IS CODE?

AND HOW IS IT PREVALENT IN TODAY'S WORLD?

CODE

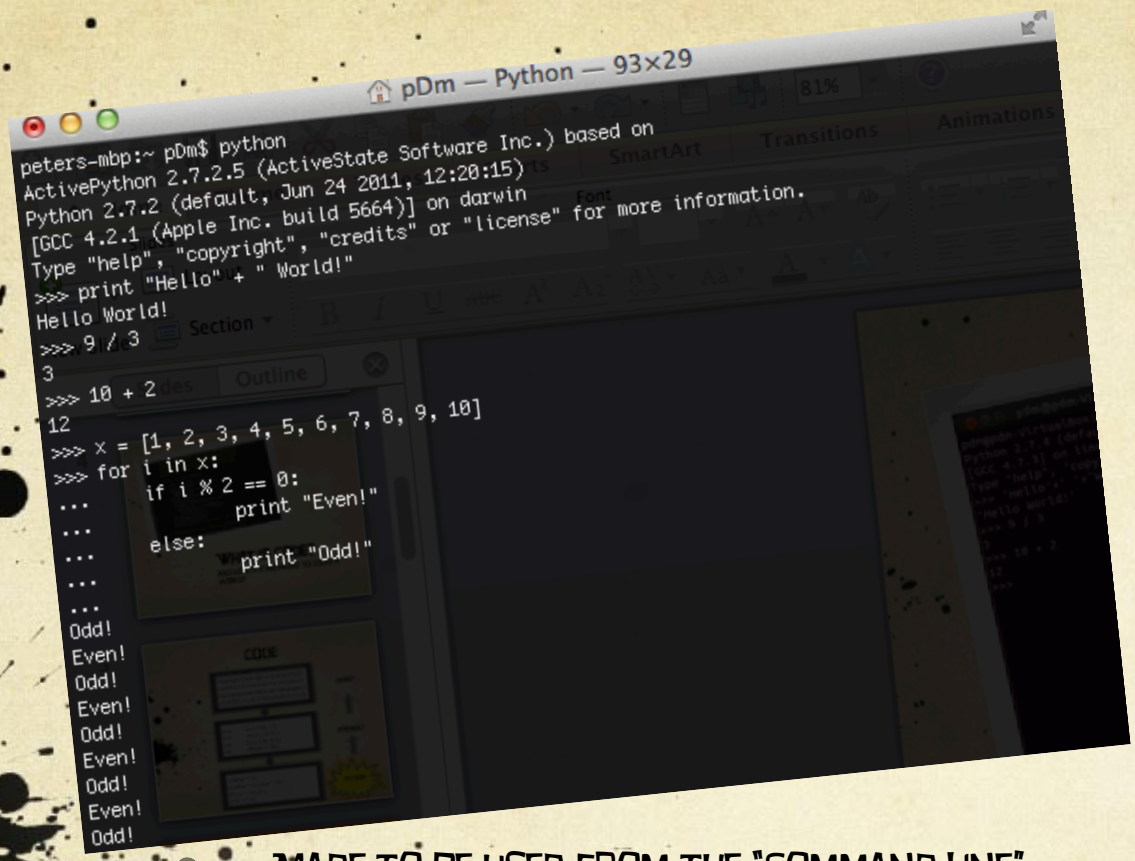


BINARY

ASSEMBLY

PYTHON

PYTHON - THE BEGINNINGS



- MADE TO BE USED FROM THE "COMMAND LINE"
  - WINDOWS (WHAT WE'RE USING TODAY): COMMAND PROMPT
  - OS X/LINUX: TERMINAL
- TO BEGIN, TYPE "PYTHON"
  - STARTS AN *INTERACTIVE SHELL*
  - TYPE COMMANDS, AND THE SHELL WILL RESPOND
- WE ARE GOING TO BE USING A TEXT EDITOR INSTEAD (EASIER)...

Starting / Using Sublime Text 3

- To start Sublime Text 3
  - Click the Windows *Start* button, the Start menu will appear
  - Click on the Sublime Text 3 quick-launch icon 

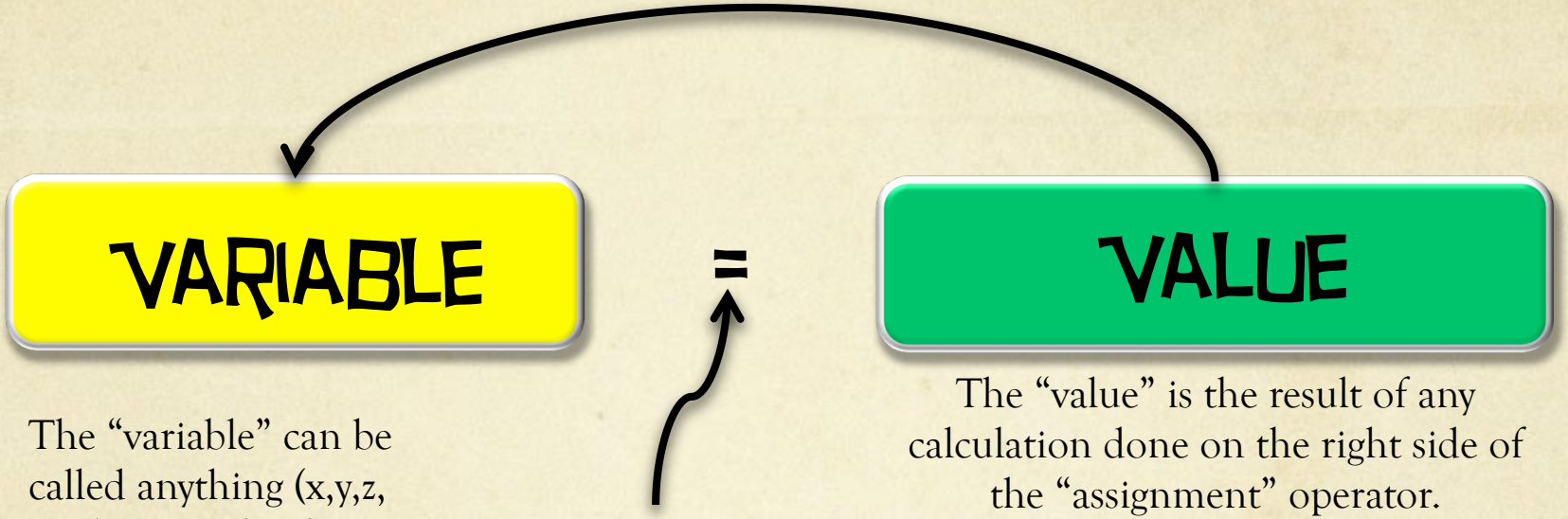
- To begin using Sublime Text 3
  - Click *File -> New File (Ctrl + N)*
  - Save the new file
    - Click *File -> Save (Ctrl + S)*
    - Save the file as helloWorld.py in your project folder

- Now let's write our first program!

TYPES

- Numbers (Integers & Floats) – store numeric values
  - Integers (`int`) – whole numbers from negative infinity to infinity, such as 1, 0, -5, etc.
  - Float (float) – any rational number, usually used with decimals such as 2.8 or 3.14159
- Strings (`str`) – a set of letters, numbers, or other characters
  - Enclosed within quotation marks like 'Hello World!' or "Hello World!"
  - Can be single or double quotes in Python (just need to be consistent within the string)
- Booleans (`bool`) – truth values (think back to logic in math)
  - Can either be True or False
- Lists (`list`) – a list of values without a fixed number of elements, i.e. x = [1,2,3]
  - Square brackets represent a list in Python
  - Can mix and match different types within a list
    - [1, 'a', True, 4.294, [3, 2, 'Hello']]

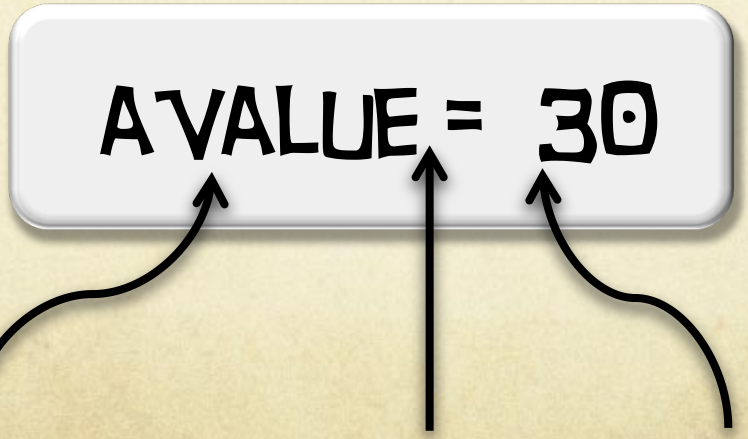
VARIABLES AND ASSIGNMENT



The "variable" can be called anything (x,y,z, apple,orange). The variable "holds" the value calculated on the right side of the assignment operator.

Assignment, NOT **equals**!

The "value" is the result of any calculation done on the right side of the "assignment" operator.
Ex: 30 + 30 (value is 60)
15 * 2 (value is 30)
'hello'+'!' (values is "hello!")



This is the *variable*

"Assignment" Operator

This is the *value*

ARITHMETIC OPERATIONS

- Assume variable *a* holds 10 and variable *b* holds 20, then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0
**	Exponent - Performs exponential (power) calculation on operators	a**b will give 10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 is equal to 4 and 9.0//2.0 is equal to 4.0

VARIABLES AND ASSIGNMENT

OPERATION	EVALUATION
NUMBER = 30	NUMBER --> 30
NUMBER = 15 * 2	NUMBER --> 30
NUMBER = 30 – 15	NUMBER --> 15
NUMBER = 3 ** 3	NUMBER --> 27
NUMBER = 2 ** (2 + 3)	NUMBER --> 2 ** (5) --> 32
AVALUE = 5	AVALUE = 5
NUMBER = AVALUE * 3	NUMBER --> (5) * 3 --> 15
NUMBER = AVALUE + 15	NUMBER = (15) + 15 --> 30
NUMBER = 10	NUMBER --> 10
NUMBER = NUMBER ** 2	NUMBER --> (10) ** 2 --> 100

The calculation of the *value* to be assigned to the *variable* follows PEMDAS rules.

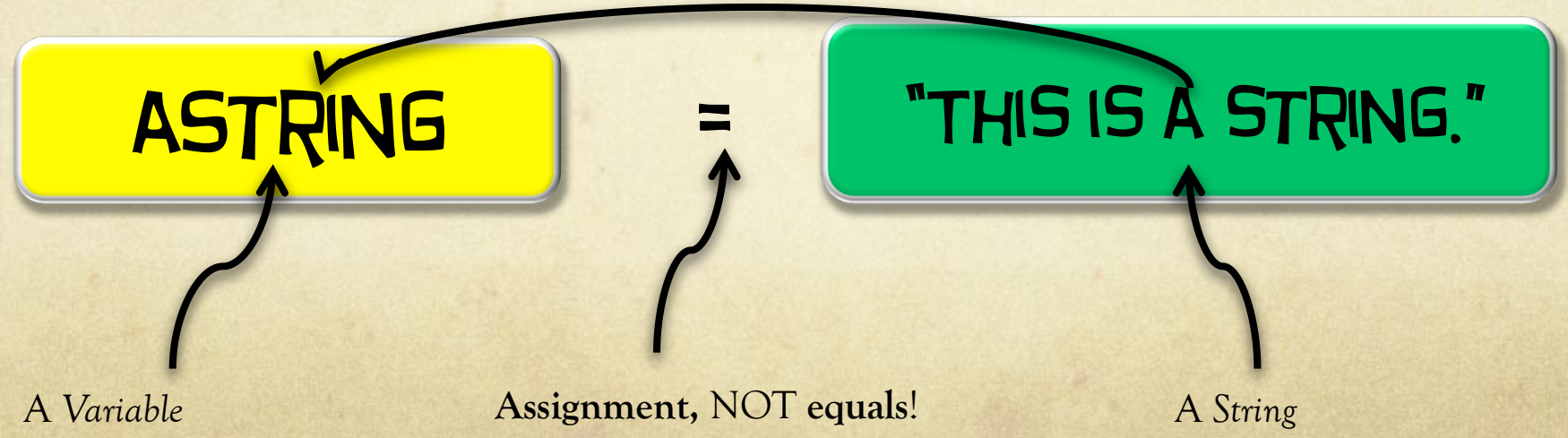
A value calculation can include other **variables** (using their stored value), in fact a calculation can include the variables current value, which will be changed to the new value following assignment.

ADVANCED ASSIGNMENT OPERATORS

Operator	Description	Example
=	Simple assignment operator; Assigns values from the right side operand to the left side operand	c = a + b will assign value of a + b into c
+=	Add AND assignment operator; it adds the right operand to the left operand and assigns the result to the left operand	c += a is equivalent to c = c + a
-=	Subtract AND assignment operator; it subtracts the right operand to the left operand and assigns the result to the left operand	c -= a is equivalent to c = c - a
*=	Multiply AND assignment operator; it multiplies the right operand to the left operand and assigns the result to the left operand	c *= a is equivalent to c = c * a
/=	Divide AND assignment operator; it divides the right operand to the left operand and assigns the result to the left operand	c /= a is equivalent to c = c / a
%=	Modulus AND assignment operator; it takes the modulus using two operands and assigns the result to the left operand	c %= a is equivalent to c = c % a
**=	Exponent AND assignment operator; performs exponential (power) calculation on operators and assigns the value to the left operand	c **= a is equivalent to c = c ** a
//=	Floor Division AND assignment operator; performs floor division on operators and assigns the value to the left operand	c //= a is equivalent to c = c // a

STRINGS

- A *VARIABLE* CAN STORE THINGS BESIDES NUMBERS AS ITS VALUE.
  - TEXT, LISTS, ETC. . . *(WE WILL GET TO THIS LAST ONE IN A FEW MINUTES)*
- A *STRING* IS A VALUE THAT HOLDS TEXT, WHICH CAN BE A SINGLE LETTER OR AN ENTIRE PARAGRAPH.
  - A STRING IS MADE SIMPLY BY PUTTING " " AROUND A WORD OR PHRASE.
    - EX: *"THIS IS A STRING"* CAN BE ASSIGNED TO A VARIABLE LIKE A NUMBER.



Strings

- What can String be used for?
  - Combine with *conditionals* and *print* commands to allow the program to give feedback to the user.
  - Can be used to get user *input*.
    - Ex: `answer = raw_input("Enter something: ")`
      - The above statement will ask the user for some sort of input. The input (a string) will be assigned to the variable *answer*.
- How can we manipulate strings?
  - Strings *can* be added together.
    - Ex: `aStr = "the cat" + " in the hat." = "the cat in the hat."`
  - Strings *cannot* be subtracted from one another.

STRING OPERATORS AND FUNCTIONS

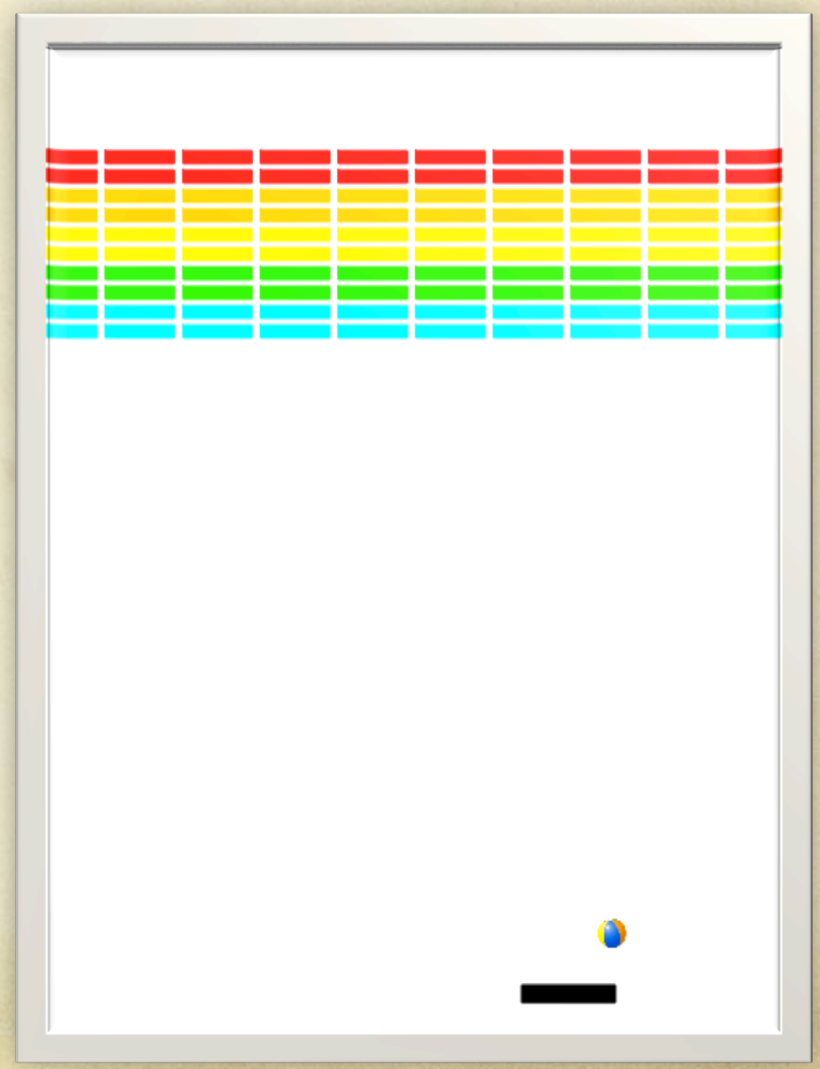
Assume variable *a* holds 'Hello', variable *b* holds 'World' and variable *c* holds 10, then:

Operator/Function	Description	Example
+	Concatenation – adds values on either side of the operator	a + b will give 'HelloWorld'
*	Repetition – creates new strings, concatenating multiple copies of the same string	a * 2 will give 'HelloHello'
+=	Concatenation AND assignment operator; it adds the right operand to the left operand and assigns the result to the left operand	a += b will give 'HelloWorld'
str(c)	Function which casts other types to string	d = a + str(c) will return "Hello10"
len(c)	Function which returns the length (the number of items in an object). For strings, it returns the number of characters in a string.	newVar = a + ' ' + b len(newVar) will return 11

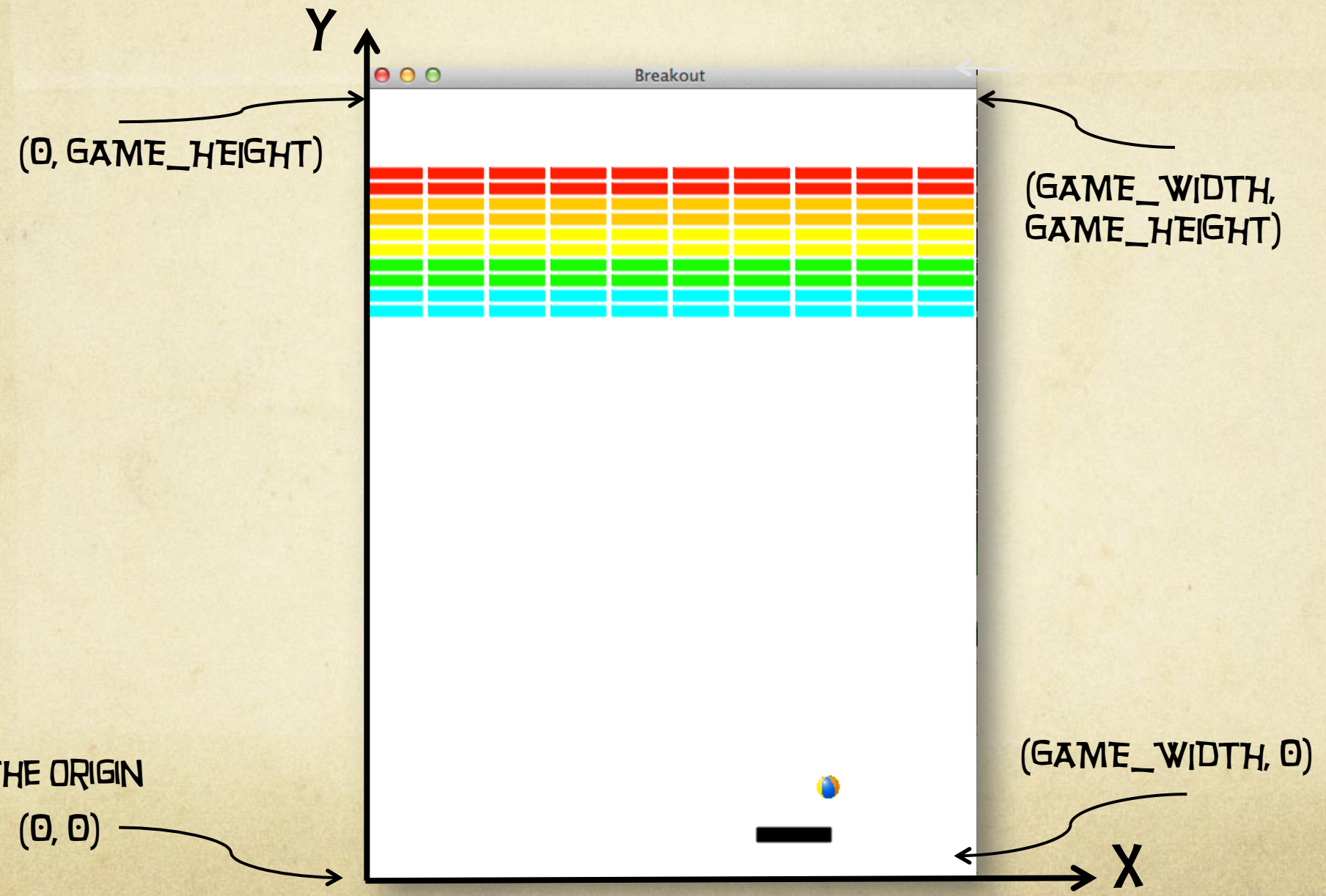
NOW TRY THIS YOURSELF

A PRACTICAL APPLICATION
BREAKOUT

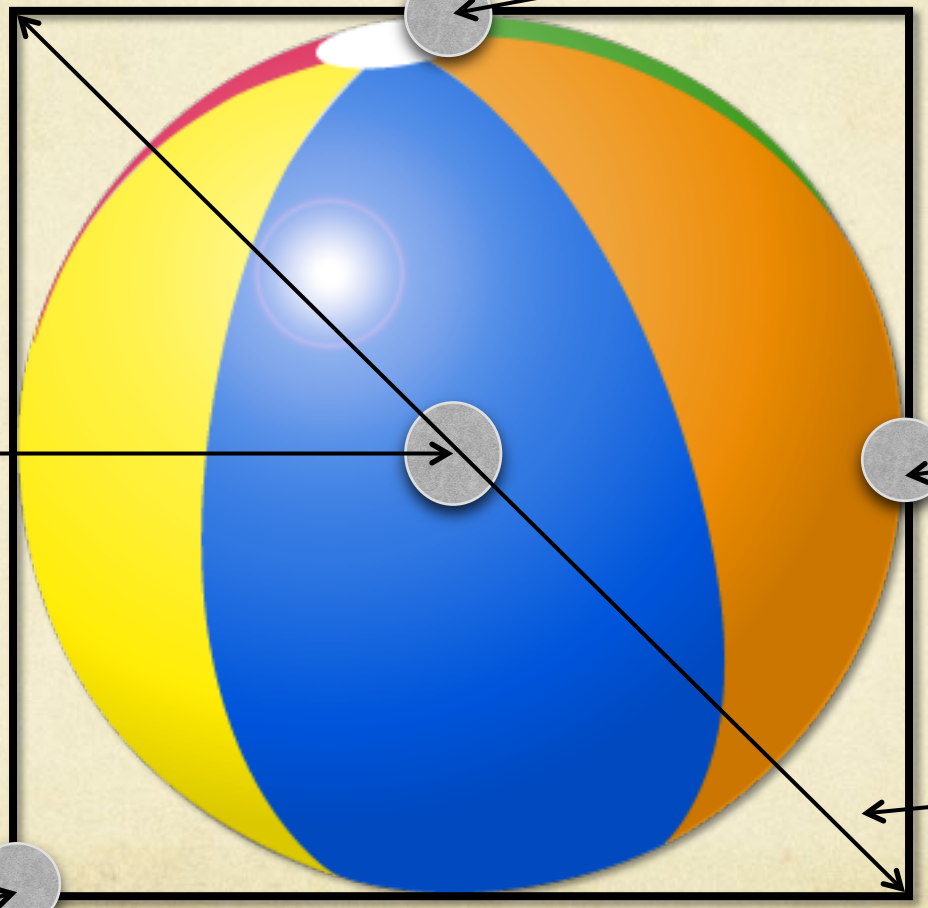
- Breakout is a simple arcade game that was released in 1976 by Atari. It was first coded by Steve Wozniak (other founder of Apple).
- Each of you has your own copy of the game's python code in your personal folder.
- There are a few *"bugs"* and missing pieces in the game's code. Let's use what we've learned thus far to fix the first one.
- Open controller.py file that is found in the breakout folder in your directory, then open CMD and run the program



BREAKOUT COORDINATE SYSTEM



BREAKOUT BALL



BALL.TOP = Y + HEIGHT

BALL.RIGHT
= X + WIDTH

BALL.CENTER
(CENTER_X, CENTER_Y)

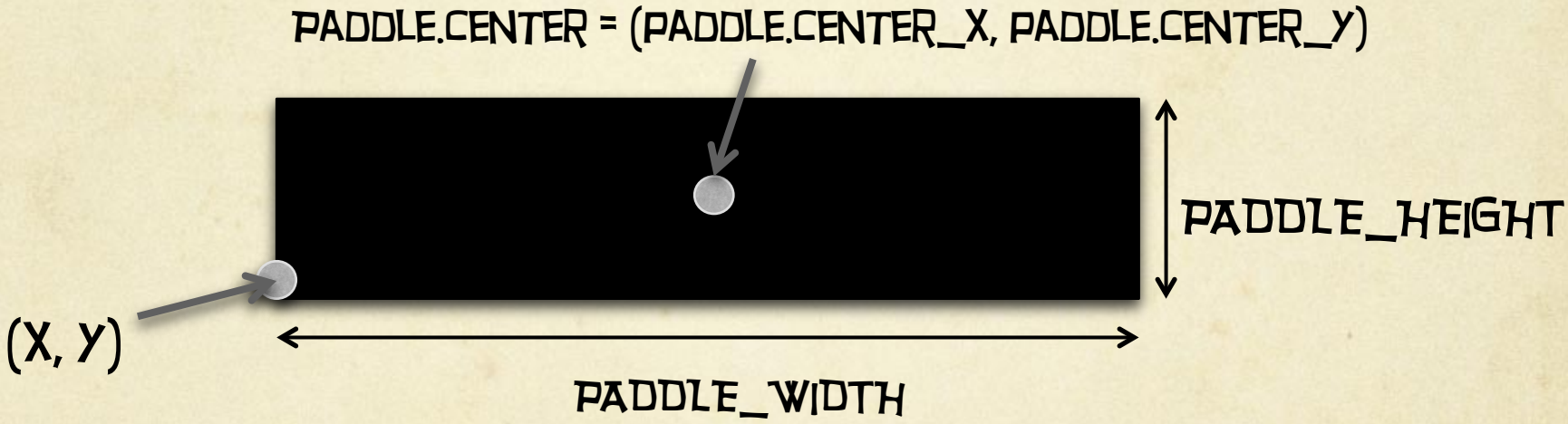
center_x = x + width / 2

center_y = y + width / 2

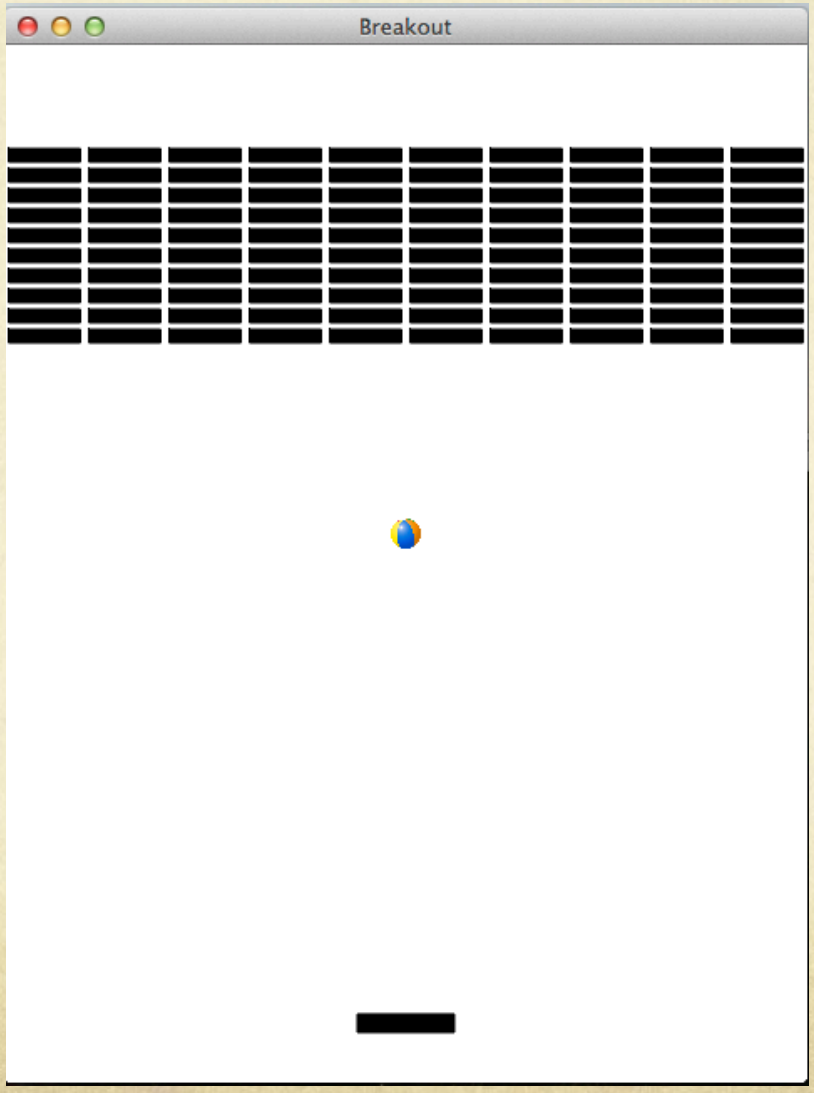
BALL_DIAMETER

(X, Y)

BREAKOUT PADDLE



WHAT'S WRONG?

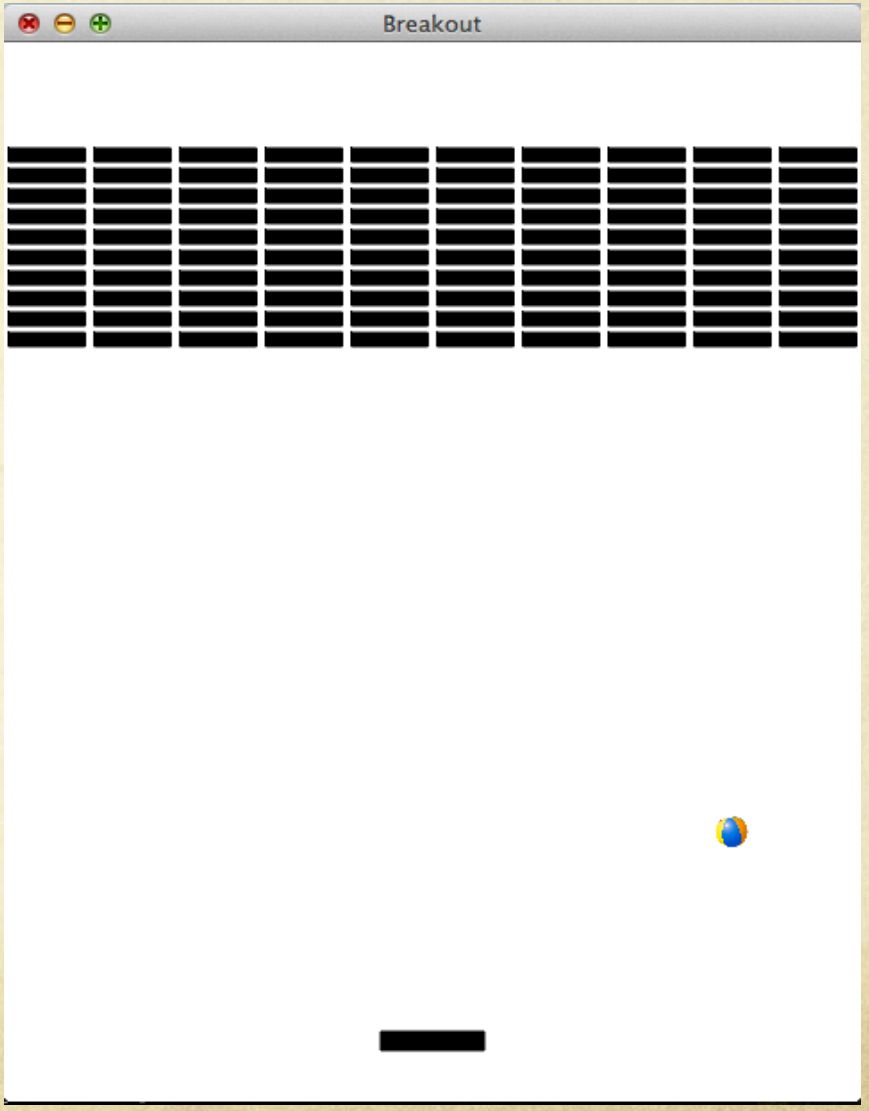


MOVE THE BALL!

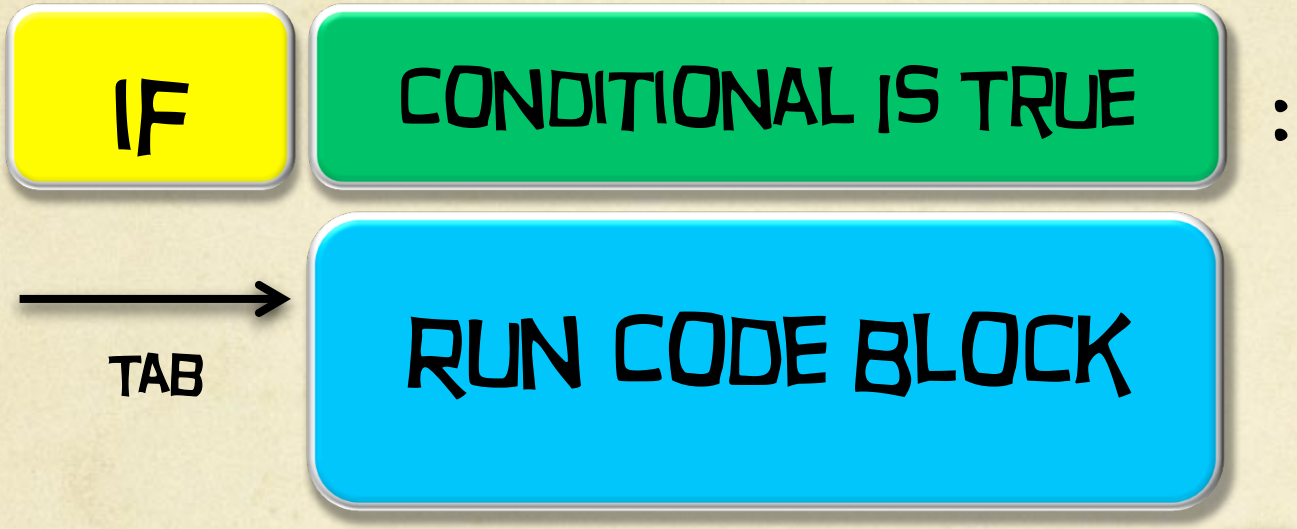
- WE NEED TO FIGURE OUT A WAY TO GET THE BALL MOVING...
- IN THE CONTROLLER.PY FILE, SCROLL TO LINE 533 WHERE IT SAYS
  - ```
    def updatePos(self):
    """Helps move the ball one step at
    a time. This is accomplished by adding
    the ball's velocity components to the
    ball's corresponding position
    coordinates. This function is not
    hidden, because Breakout must be able to
    "see" this function in order to call
    it."""
    ```
    ```

THE BALL MOVES!

- HOWEVER, THE BALL IS NOW GOING OFF THE SCREEN...
- WE'RE NOT QUITE READY TO SOLVE THIS PROBLEM, SO LET'S LEARN A FEW MORE TRICKS!



CONTROLLING ASSIGNMENT
CONDITIONALS



The *Assignment*

The *print statement*.
Ex:
print variable

```
NUMBER = 0
IF (NUMBER == 0):
    ACOPY = 10
    NUMBER = ACOPY * 2

PRINT NUMBER
```

The *Conditional*

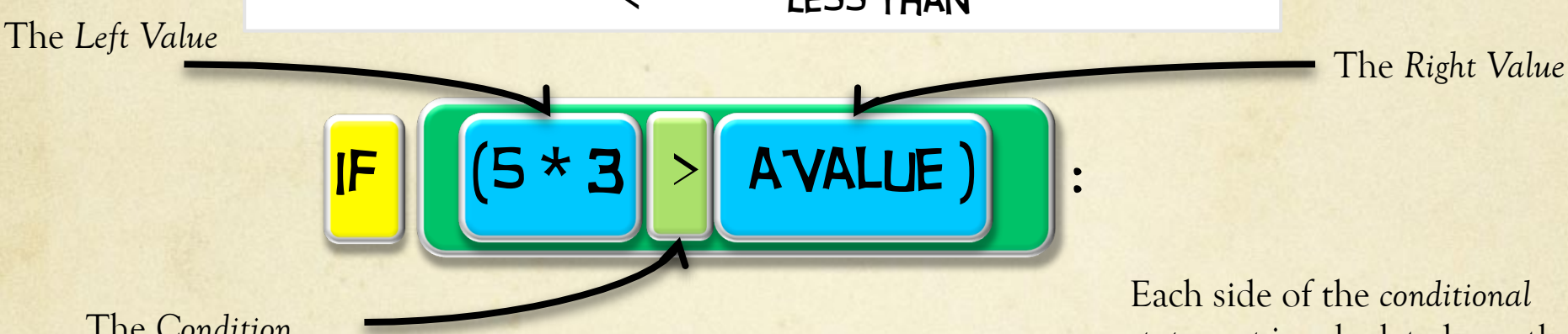
The equation in here is known as a *logic operation*. All conditionals will either be *True* or *False*.
If true, the program runs the code block, otherwise it skips it.

CONTROLLING ASSIGNMENT

CONDITIONALS

TYPES OF CONDITIONS

==	EQUALS
!=	NOT EQUALS
>	GREATER THAN
<	LESS THAN



IF

15 == 15	TRUE
15 == 16	FALSE
3 + 4 == 7	TRUE
8 != 7	TRUE
98 > (3*15)	TRUE

Each side of the *conditional statement* is calculated exactly like the right side of an *assignment* of operation. The resulting values on each side are compared to one another according to the type of *condition* imposed (==, != , > , <) and returns either *True* or *False*.

KEEP THE BALL IN BOUNDS!

- Let's scroll to line 384 where it says

```
def _checkBallBounds(self):

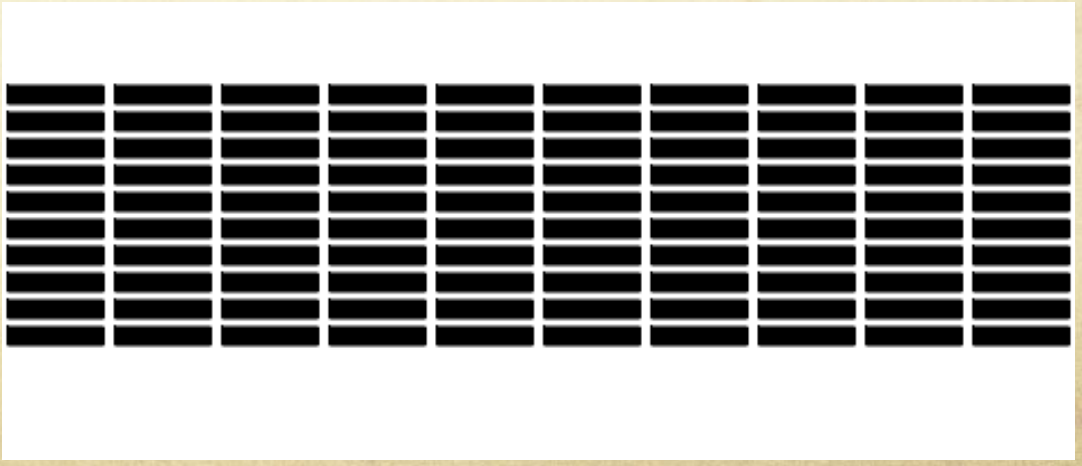
    """Checks the boundaries of the playing field. If the
       ball hits the bottom wall, the player loses a life.  If
       the ball hits any of the three other walls its
       respective component of its velocity is negated.
       First, we will implement this function that if the
       ball hits the bottom wall, negate its vertical velocity
       component (i.e., we're going to let the ball bounce
       around."""
```

Make Sure the Paddle Stays On the Screen!

- Great job on keeping the ball in bounds, but if you start to drag the paddle around, you'll realize pieces of the paddle go off the side of the screen
- How can we fix that...?
  - Doesn't it seem like a similar problem to the ball?
- Let's scroll to the method (function) right below the _checkBallBounds, that says
  - def _keepPaddleInBounds(self):

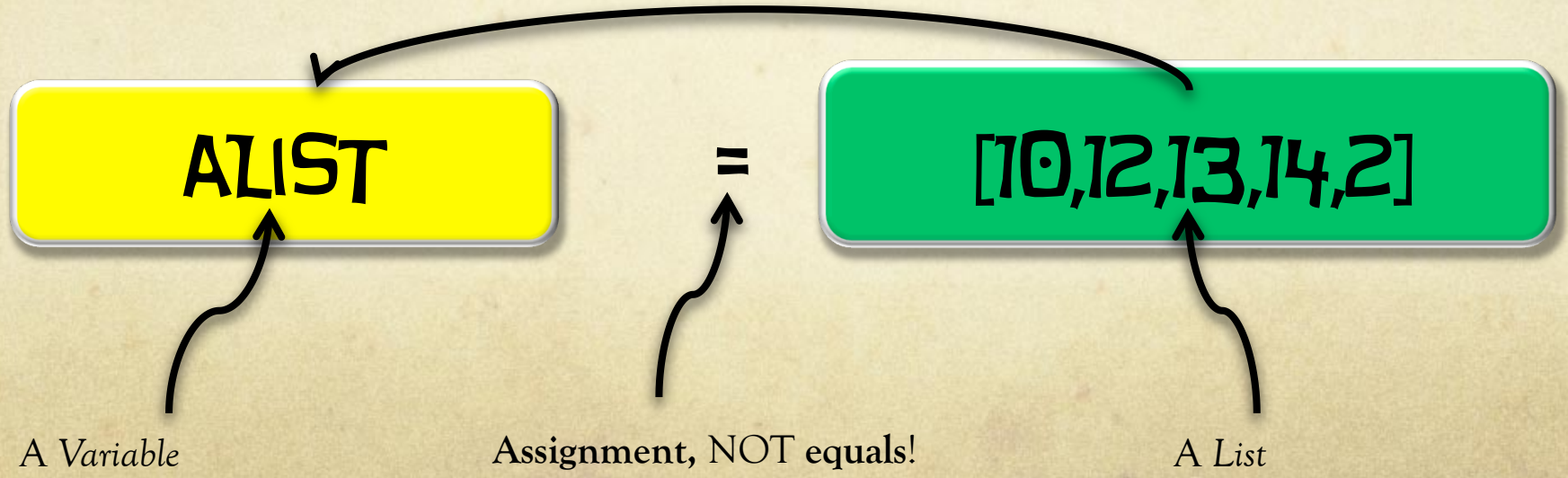
Let's fix those ugly bricks!

- Now that we have the ball moving (and staying in bounds!), and we also have the paddle staying in bounds, let's do something about those ugly bricks
- First, let's learn some more tricks to get the job done...



Lists

- A *variable* can store things besides numbers as its value.
  - Lists, text , etc... *(we already covered the second one)*
- A *list* is a collection of a bunch of similar items (numbers, strings, etc...).
  - A list is made by putting a series of values (numbers or strings) together inside brackets [ ], separated by commas.



LISTS

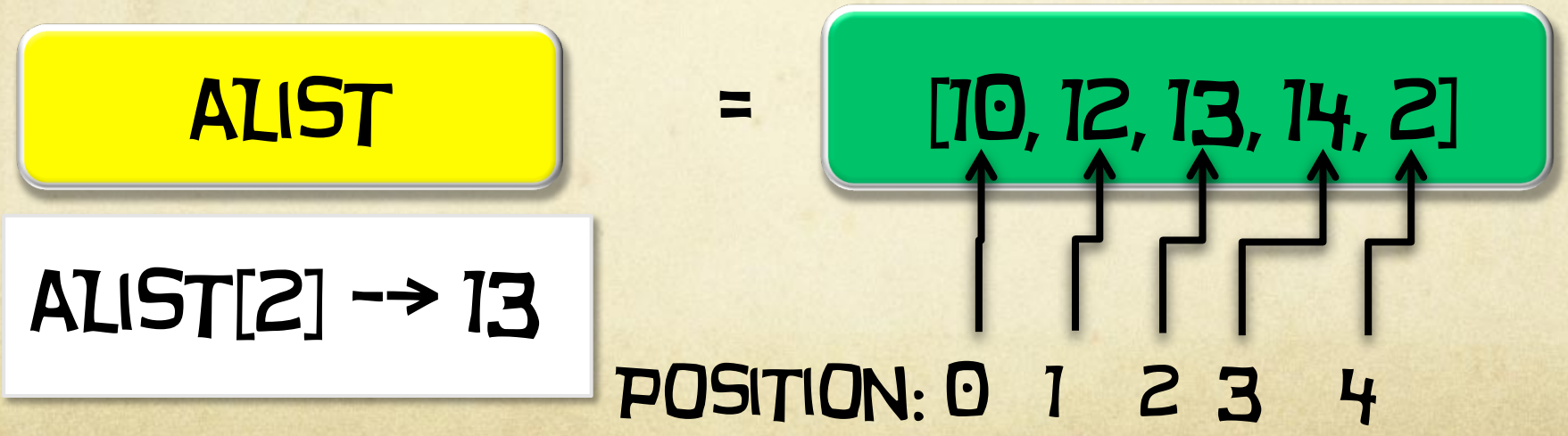
Examples of Lists

myList = [3,4,1,63,12]	List of Numbers
myList = [“aString”,”anotherString”]	List of Strings
myList = [anotherList,anotherList2]	List of Lists

- ADDING AND ITEM TO A LIST:
  - MYLIST.APPEND(ITEM)
- REMOVING AN ITEM FROM A LIST:
  - MYLIST.REMOVE(ITEM)
- HOW DO WE ACCESS ITEMS THAT WE PUT IN OUR LIST?

LISTS

- INDEXING
  - A LIST HOLDS A COLLECTION OF ITEMS, EACH OF WHICH IS IN A SPECIFIC POSITION WITHIN A LIST.
  - YOU CAN RETRIEVE ITEMS FROM A LIST THROUGH A PROCESS CALLED *INDEXING*.
    - ALIST[POSITION] WILL RETURN THE VALUE STORED AT ALIST'S POSITION
    - POSITIONS START AT 0

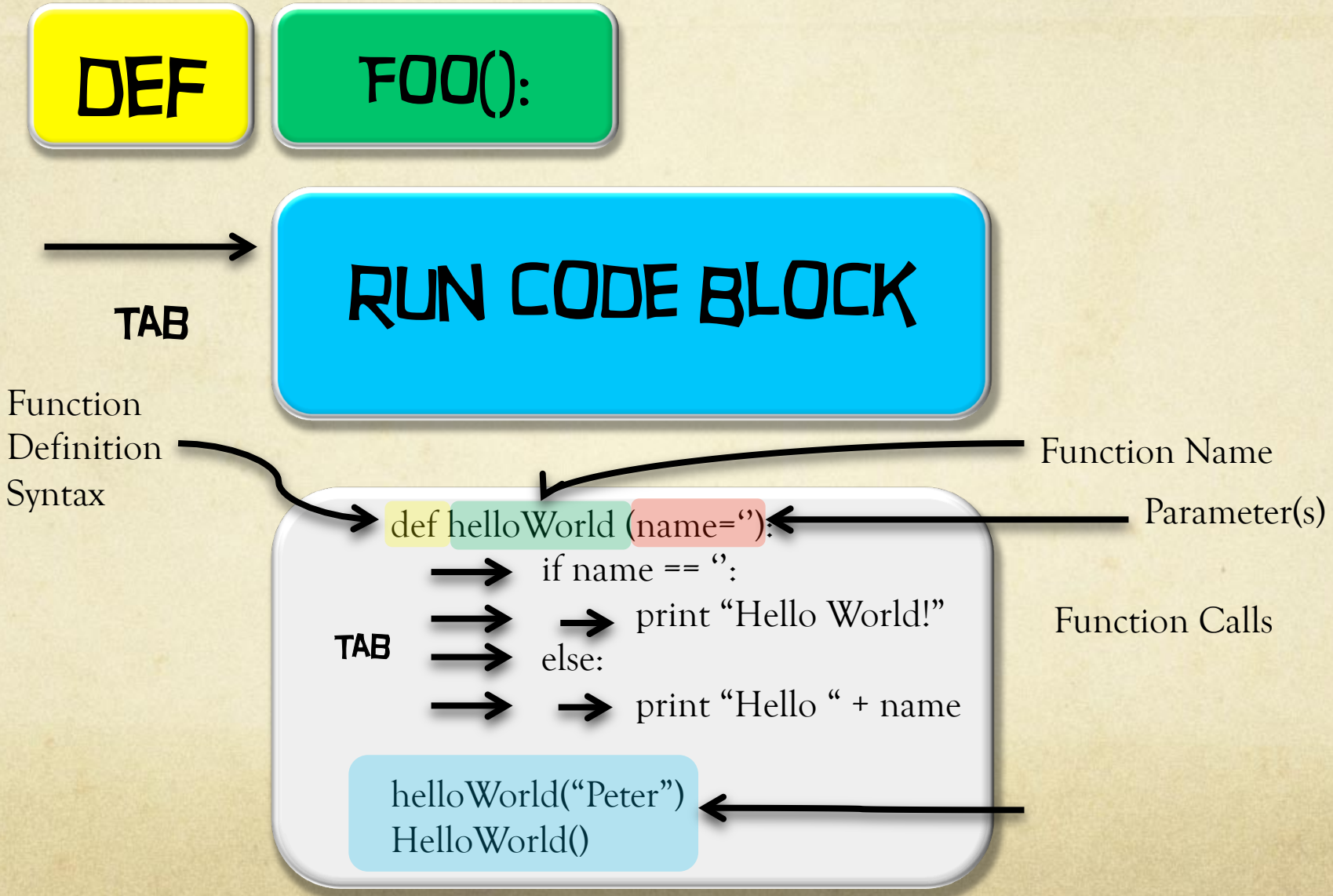


ALIST[2] --> 13

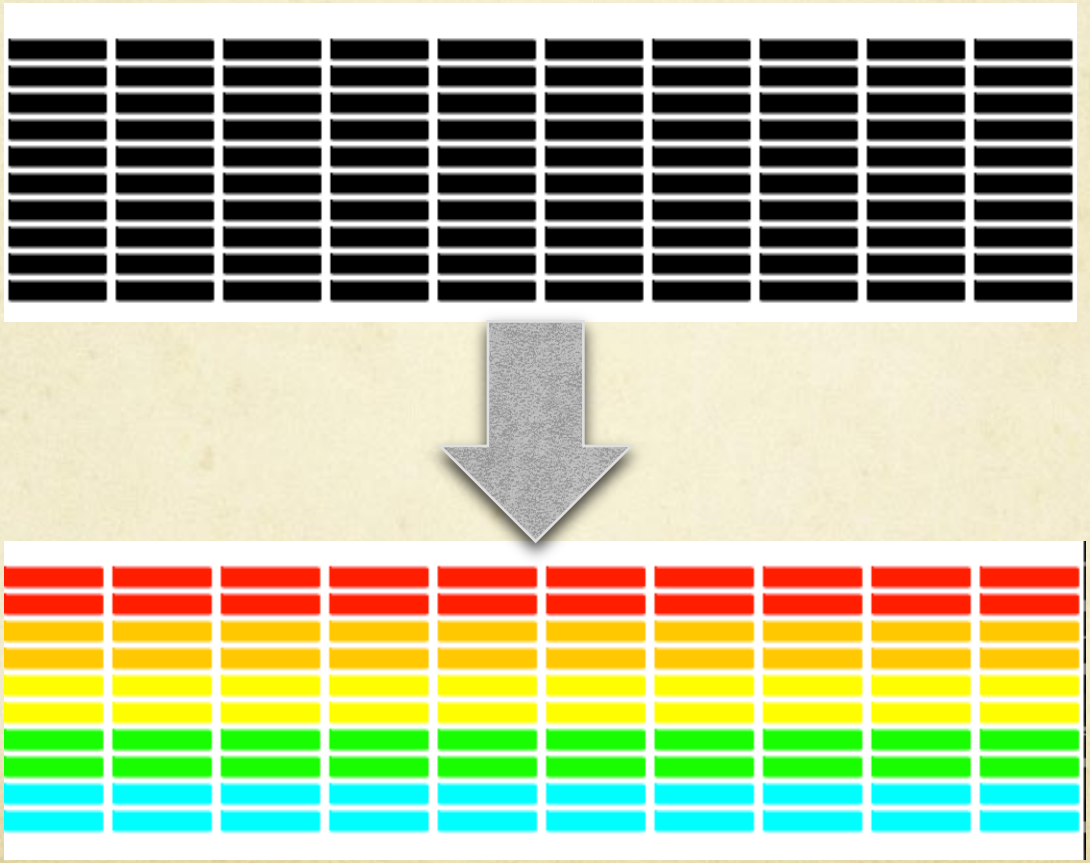
Functions

- A function is just a piece of code in a program that performs a specific task
  - Reduces duplication of code
  - Helps one decompose complex problems into manageable pieces
  - Improves clarity of the code
  - Gives one the ability to "reuse" code
- Two basic "types" of functions
  - User-defined functions
    - We will see in the next slide…
  - Built-in functions
    - Functions premade by Python
    - **Ex:** len(), str(), abs(), type(), max(), min()

USER-DEFINED FUNCTIONS



NOW TRY THIS YOURSELF



Fixing the Brick Color

- Each row "couple" (set of two) should be the same color
- Although the game can handle any number of rows from user input (we'll see this later), let's make it simple and work with 10 rows (you'll see later why our solution will work for any number of rows)
  - Rows 1 & 2 - red
  - Rows 3 & 4 - orange
  - Rows 5 & 6 - yellow
  - Rows 7 & 8 - green
  - Rows 8 & 9 - cyan
- How can we write this function?
- Go to `def _findBrickColor(self, rowNum):`

WHAT ABOUT COLLISIONS?!

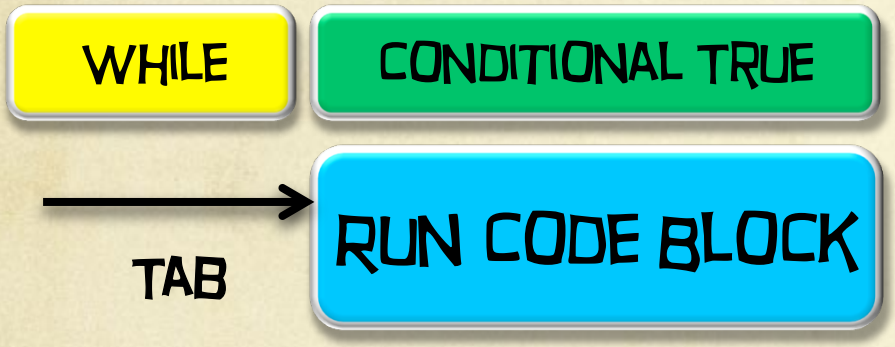
- THERE'S ONE MORE IMPORTANT COMPUTER SCIENCE CONCEPT WE NEED TO LEARN ABOUT BEFORE WE CAN FIX THE LAST "BUG" IN BREAKOUT!

- ONCE WE LEARN ABOUT LOOPS, WE CAN TRY TO GET OUR GAME TO RESPOND TO COLLISIONS BETWEEN THE BALL AND DIFFERENT OBJECTS (LIKE THE PADDLE AND BRICKS)

LOOPS

- A *LOOP* IS A POWERFUL WAY TO REPEAT THE SAME BLOCK OF CODE MULTIPLE TIMES.
- TWO IMPORTANT TYPES OF LOOP:

WHILE LOOP



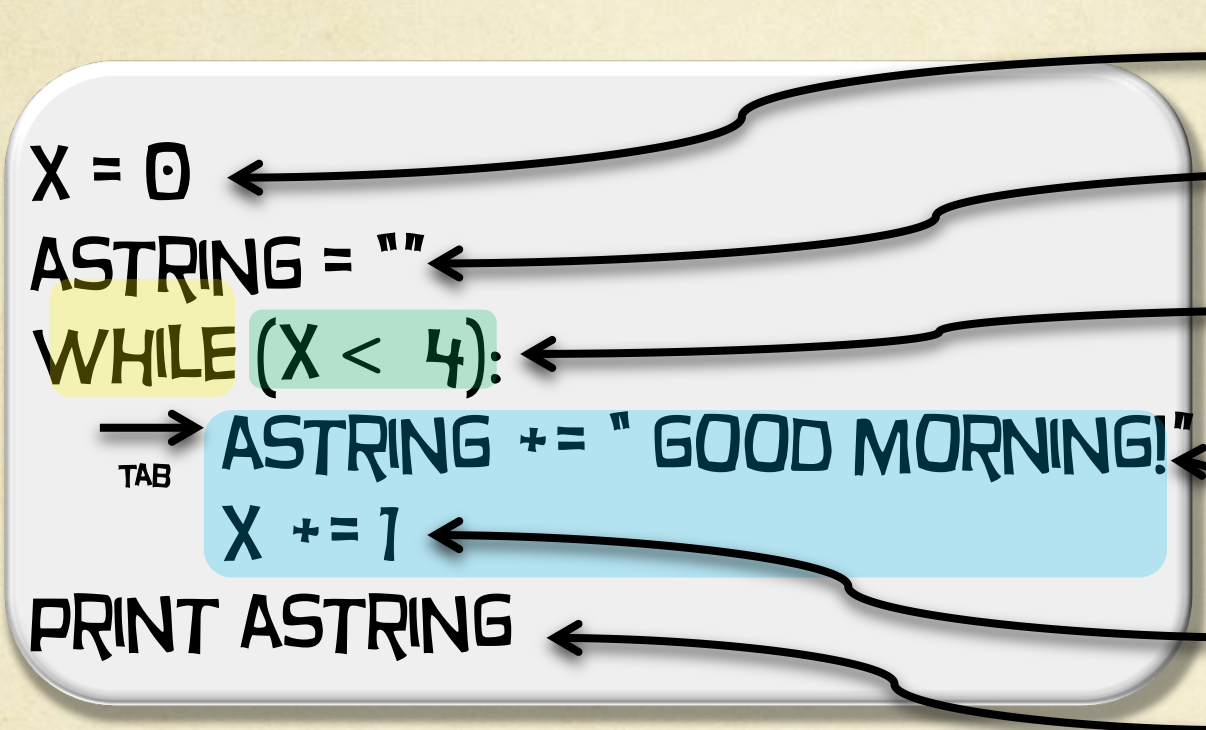
- A *WHILE LOOP* IS SIMILAR TO AN *IF STATEMENT* IN THAT THE CODE BLOCK WILL KEEP REPEATING WHILE THE GIVEN CONDITIONAL IS STILL TRUE.

FOR LOOP

- A *FOR LOOP* WILL RUN THE CODE BLOCK *ONCE FOR EVERY VALUE IN A LIST*. FOR ANY GIVEN CYCLE OF THE CODE BLOCK, *X* HOLDS THE CURRENT LIST VALUE.

LOOPS

- EXAMPLE OF A *WHILE LOOP*:



- RESULT: " GOOD MORNING! GOOD MORNING! GOOD MORNING! GOOD MORNING!"

LOOPS

- WHAT IS THE *RANGE(N)* FUNCTION?
  - IN PYTHON, TYPING RANGE(*N*) WHERE *N* IS A NUMBER WILL GIVE YOU A *LIST OF ELEMENTS* STARTING FROM 0 AND GOING UP BY ONE (1,2,3,ETC…) UNTIL *ONE LESS THAN N*.
    - *EX: RANGE(5)* == [0,1,2,3,4]
    - THIS IS AN EASY AND COMMONLY USED WAY TO MAKE LISTS FOR *FOR LOOPS*.
- EXAMPLE OF A *FOR LOOP*:

The variable *x* will cycle through the range 0 to 4

The *range* here is: 0,1,2,3,4

```
FOR X IN RANGE(0,5):
    PRINT (2 * X)
```

TAB

- RESULT: 0 2 4 6 8

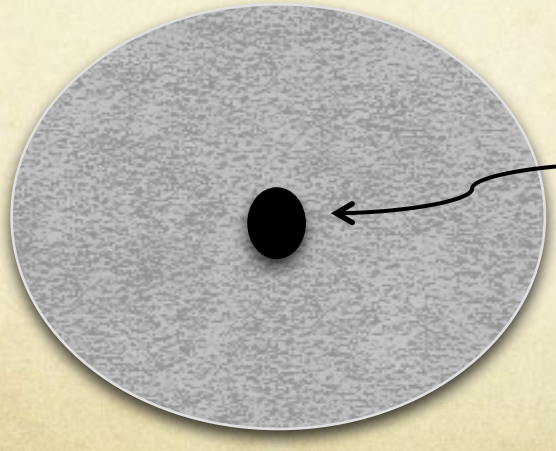
```
def _getCollidingObject(self):
```

- RETURNS: GRAPHICSOBJECT THAT HAS COLLIDED WITH THE BALL
- THIS METHOD CHECKS THE FOUR CORNERS OF THE BALL, ONE AT A TIME. IF ONE OF THESE POINTS COLLIDES WITH EITHER THE PADDLE, OR A BRICK, IT STOPS THE CHECKING IMMEDIATELY AND RETURNS THE OBJECT INVOLVED IN THE COLLISION.
- IT RETURNS NONE IF NO COLLISION OCCURRED.

Let's Think About Collisions

- As scientists often do, we make a simplifying assumption and then relax the assumption later.

- Suppose the ball were a single point (*x*,*y*) rather than a circle. Then, for any GraphicsObject (foo for example), the method call: foo.collide_point(x,y) returns true, if the point is inside of the object, and false if it isn't.



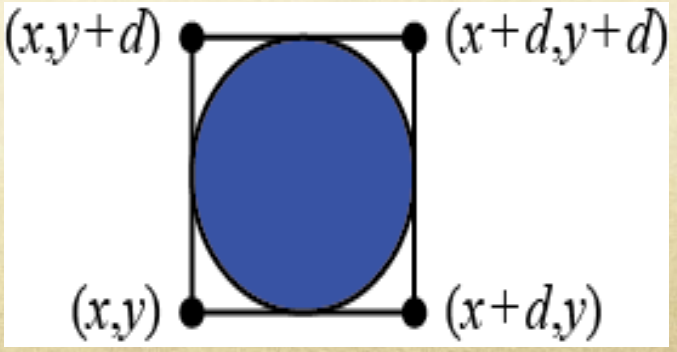
Imagine this is the point (x, y), what's the problem with this?

COLLISIONS

- Unfortunately, the ball is not a single point. It occupies a physical area, so it may collide with something on the screen even though its center doesn't. For our purposes, the easiest thing to do — which is typical of the simplifying assumptions made in real computer games — is to check a few carefully chosen points on the outside of the ball, and then see whether any of those points has collided with anything. As soon as you find something at one of those points (other than the ball, of course) you can declare that the ball has collided with that object.

COLLISIONS

- One of the easiest ways to come up with these "carefully chosen points" is to treat everything in the game as rectangles. A graphicsEllipse is actually defined in terms of its bounding rectangle (i.e., the rectangle in which it is inscribed). Therefore the lower left corner of the ball is at the point (x,y) and the other corners are at the locations shown in the diagram to the right (d is the diameter of the ball). These points are outside of the ball, but they are close enough to make it appear that a collision has occurred.



Let's implement it!

- Go to (within the controller.py file)
- `def _getCollidingObject(self):`
  - Let's get started!

We finished!

- The game is complete, you guys can take a few minutes to play your fully functioning game
- We don't know how much time is left...
- If there's a bunch of time left, think of ways to extend the game
  - If you look inside the CS.003 Resources folder within the student share drive, you'll see a PDF that's called Extensions...on there is some ideas for extensions (sounds, levels, etc.)
  - Try to implement those (here & at home)
- Don't stop learning!

ANY QUESTIONS?

CONGRATULATIONS!

```
for i in range(len(students)):

    print “You can now code, ”+ i + “!”
```

Time to take our survey!

If you don’t do it...we’re not putting you in the raffle drawing (we’re using a random number generator that pulls from our survey responses on Google Drive…using Python!