# CS 61A DISCUSSION 5

## MUTATION / OBJECT-ORIENTED PROGRAMMING

# ANNOUNCEMENTS

- Midterm scores have been released; submit regrade requests by March 5th.
- Hog composition revisions are due on March 5th.
- My mom's birthday is on March 5th. Hi mom.
- Maps has been released. Submit by Monday @ 11:59pm to get an extra credit point. If you need help, there'll be project parties on Thursday and Monday from 6:30-8:30pm in 247 Cory.

# MORE ANNOUNCEMENTS

- We now have "exam prep" office hours. During these sessions, you'll work on past exam questions and then TAs will go over them with you.
- We also have 2X speed discussion sections now – these will be from 5-6:30 in 3105 Etcheverry.
- Check OK to make sure that your scores for labs 0-3, HWs 1-3, and the Hog project all make sense.

# WHY ARE THERE SO MANY ANNOUNCEMENTS?

- There'll be a guerrilla section on trees, linked lists, and mutation on Saturday from 12-3 in 247 Cory.
- There is no lecture on Friday. I do not know why.
- Free one-on-one tutoring is now available. Sign up on Friday at 3:30pm on Piazza.

"Softmax softmax softmax softmax softmax softmax softmax softmax softmax softmax softmax softmax"

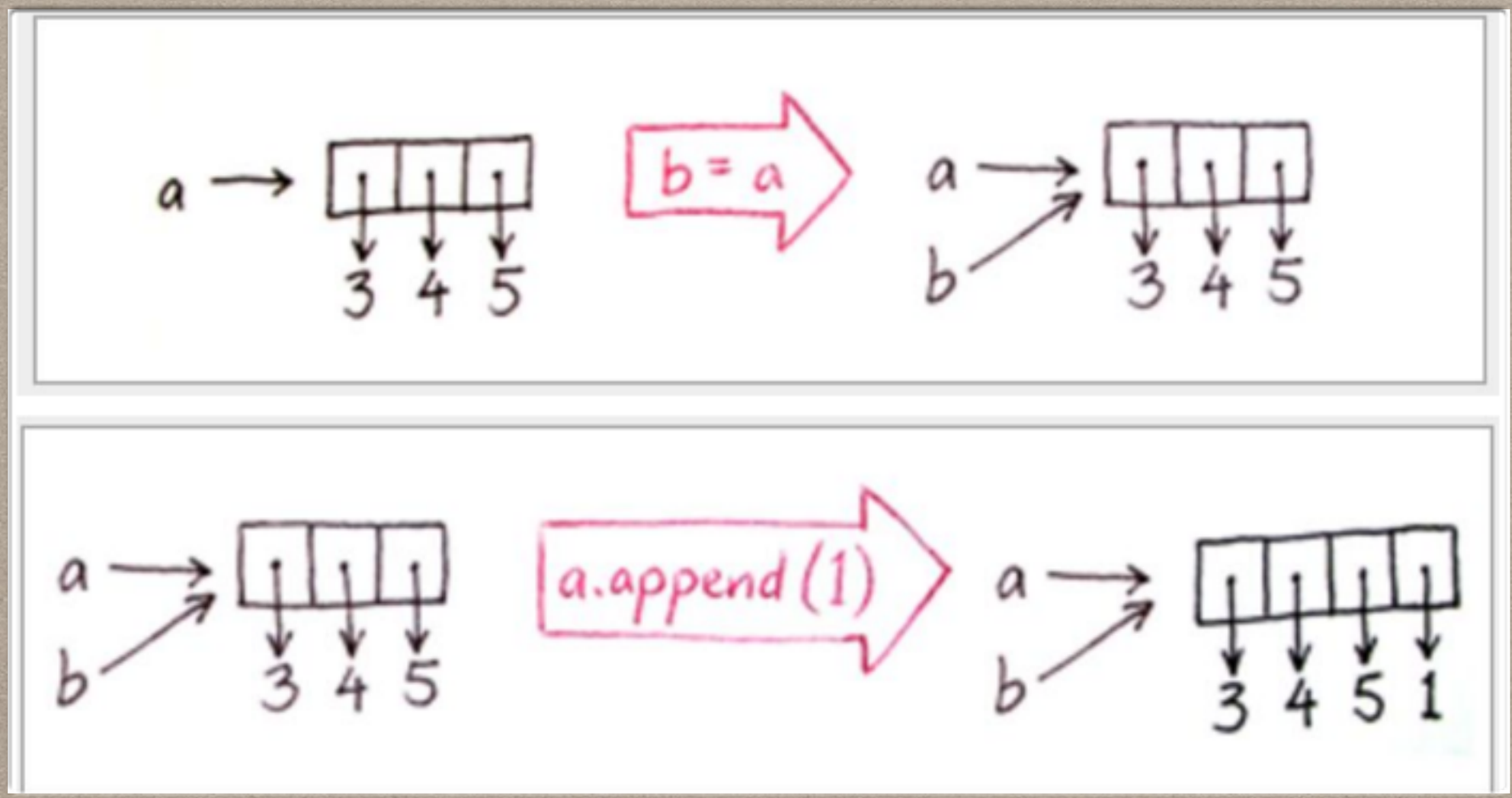
*- Sigmoid*

# MUTATION

## PYTHON MEETS THE X-MEN

# MUTATION: CHANGING THE ACTUAL OBJECT IN MEMORY

- as opposed to, say, creating a copy of an object



# LIST MUTATION

- `lst[i] = elt`
  - puts `elt` into the list at index `i`
- `lst1 += lst2`
  - extends `lst1` by the list `lst2`
- `append(elt)`
  - adds `elt` to the end of the list
- `insert(i, elt)`
  - adds `elt` to the list at index `i`
- `remove(elt)`
  - removes `elt` from the list
- `pop(i)`
  - removes and returns the item at index `i`

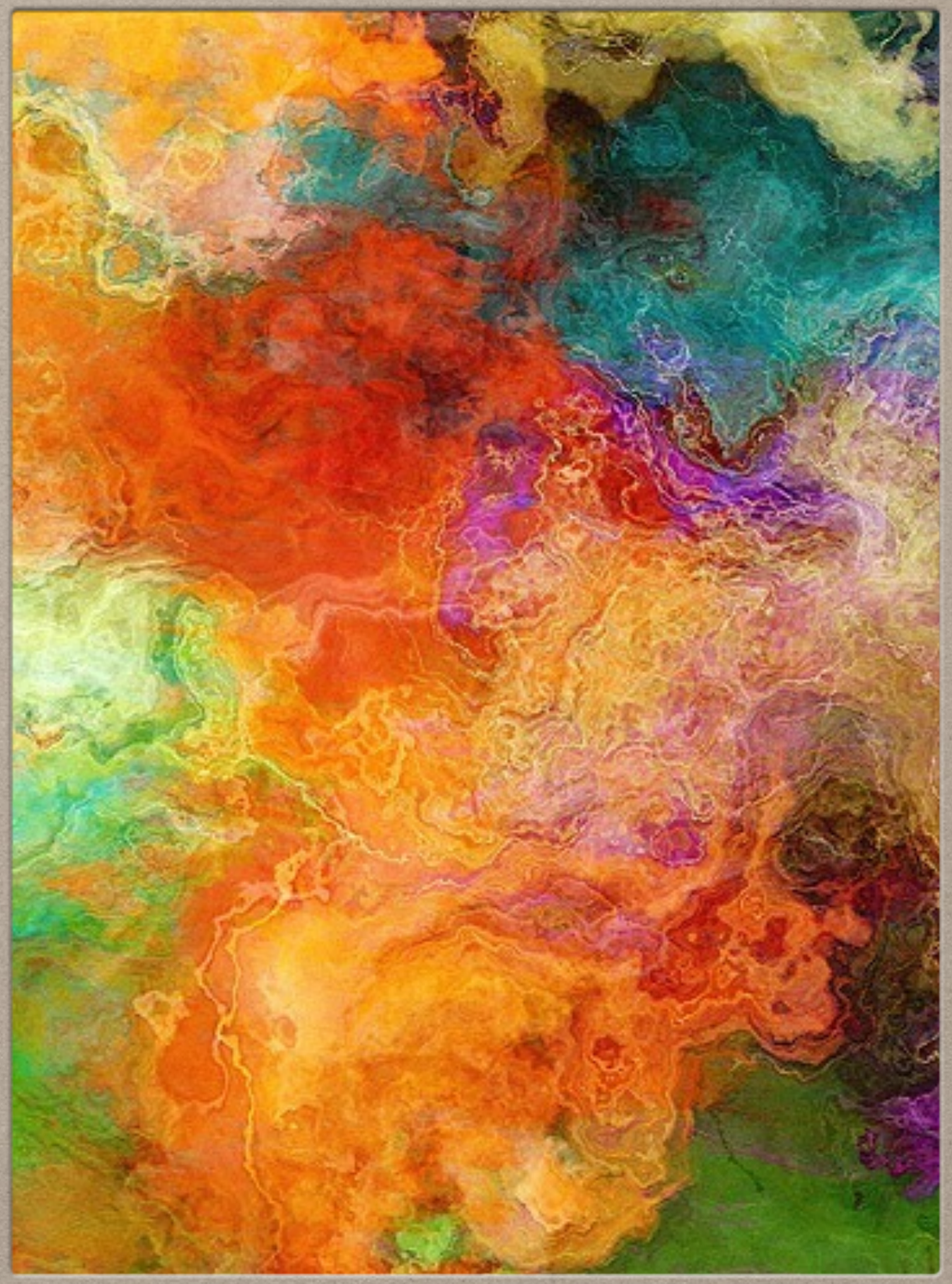
# OBJECT-ORIENTED PROGRAMMING

## BRINGING THE BOURGEOISIE TO PYTHON

# DO YOU LOVE ABSTRACTION?

IF SO, YOU'LL LOVE OBJECT-ORIENTED PROGRAMMING.

IF NOT, WHY ARE YOU HERE? (JK)



# ABSTRACTION MEANS THAT YOU HAVE TO THINK LESS

YOU DON'T ALWAYS NEED ALL THE DETAILS.

# BASIC IDEA:

MODEL REALITY (OR FANTASY... YOUR CHOICE) USING OBJECTS.

THEN YOU CAN THINK OF YOUR DATA AS WHOLESOME ENTITIES, WHICH WILL HOPEFULLY HELP MAKE SENSE OF THINGS.

OOP IS REALLY JUST A FORMALIZED VERSION OF DATA ABSTRACTION.

# OBJECTS ARE BASED AROUND CLASSES

- A **class** is like the type of an object. It's the blueprint that defines object attributes.
- An **object** is an instance of a class. Objects are created using `<ClassName>(...)`, which calls `<ClassName>`'s `__init__` method.
- In the OOP paradigm, a **constructor** is implemented as the `__init__` method, which creates an object. In it, we initialize relevant data.

# THINGS TO KNOW

- **Instance attributes** are attributes specific to an instance. (**Methods** are functions specific to an instance.)
- **Class attributes** are attributes specific to a class.
- You can access instance attributes and class attributes via **dot notation**:
  - `<object or class>.<attribute>`
  - If `<object or class>` is an object and `<attribute>` is the name of both an instance attribute and a class attribute, dot notation will prioritize the instance attribute.

# SELF

- `self` is what you should associate with objects (aka instances).
  - If you have a method defined with arguments `(self, ...)`, it's an instance method.
- `self` is always bound to a specific object instance. This happens automatically when an instance method is called or accessed using dot notation.
- To clarify: if you've got a method with an **object** on the left of the dot (as in `obj.method`), then `self` - or whatever the first formal parameter is - has already been given the value of `obj`!
- If you have a **class** on the left, though, nothing is automatically bound.

# INHERITANCE

- Inheritance is another way in which OOP models the real world.
- A **subclass** is a more specific version of a **parent class.**
  - *(A Square **is a** Rectangle, which **is a** Shape. A Car **is a** Vehicle, and so is a Bus.)*
- Subclasses inherit all of the methods and attributes of the parent class. They can also override or add more attributes.
- All classes are derived from `object`, which contains a few methods common to all classes (`__repr__`, `__str__`, etc.).