Classes

Not like the class you are taking - a kind you can make!

Stanford | ENGINEERING
Computer Science

Frankie Cerkvenik, CS106A, 2023

Housekeeping

- Sorry for the recorded lecture! Thanks for tuning in!

- BiasBars (assignment 5) has been released - it is longer but very interesting! We hope you enjoy

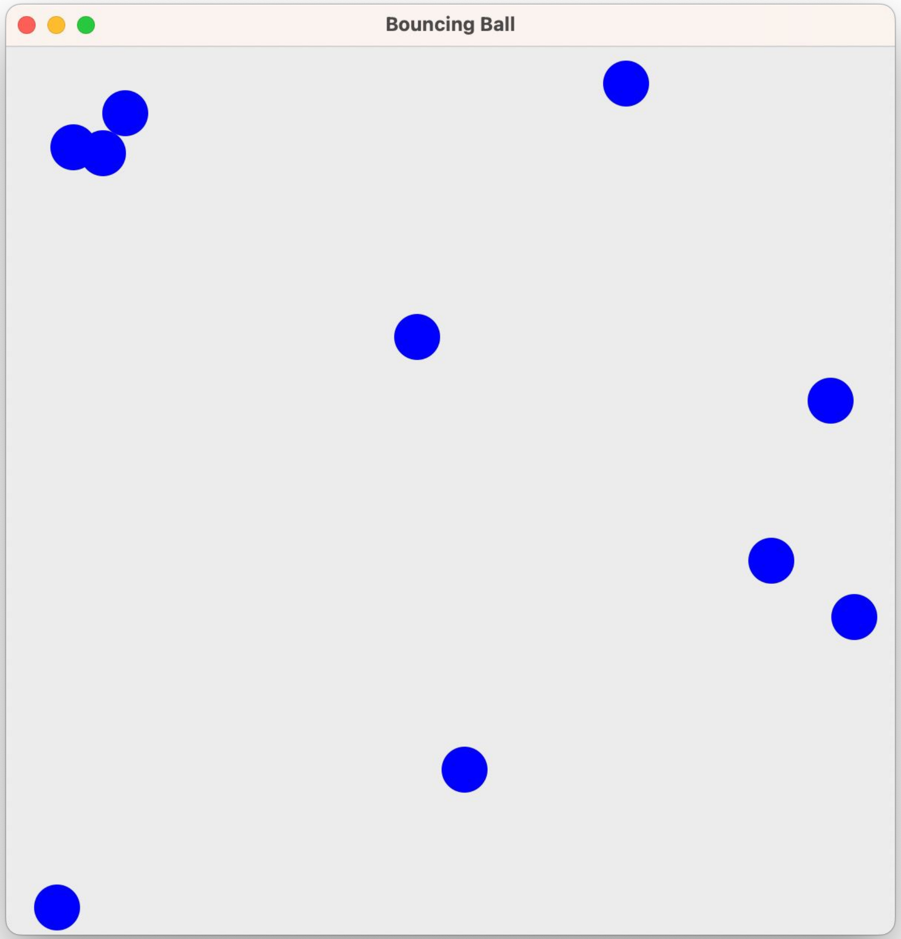
- It is due **August 8 (which is Tuesday) at 11:59**- grace period until August 9

Stanford | ENGINEERING
Computer Science

Today

- We are moving into “exposure” concepts
    - concepts that you will certainly see again,
    - But aren’t **super** the focus of 106A
    - Won’t be as emphasized on the homework assignments, but **fair game for the final**!
- Today and tomorrow (and into next week): Classes
    - How to define your own custom type!
    - Object-oriented programming

How would you make this?



Stanford | ENGINEERING
Computer Science

Python's Variable Types

- Int
- Boolean
- Float
- String
- List
- Dictionary

Stanford | ENGINEERING
Computer Science

“Custom” variable types

- Bit

```
bit  =  Bit(filename)
```

- SimpleImage

```
image = SimpleImage(filename)
```

- Canvas

```
canvas = Canvas(200, 400, ‘Example’)
```

Stanford | ENGINEERING
Computer Science

You can make your own types!

- Someone (not the Python people - someone at Stanford!) wrote the Canvas type (its in graphics.py)

- You import the Canvas type with:

```
from graphics import Canvas
```

- You use the canvas type when you make a variable of type Canvas:

```
my_canvas = Canvas(200, 400, ‘example’)
```

Stanford | ENGINEERING
Computer Science

Our first custom type: ServiceLine

- Say we wanted a variable to store information that represented a ServiceLine - like at the grocery store or the DMV

- First, we need to define a “line”- what pieces of information do we need to store a line?
  - Name - what the line is for (eg “DMV”, “Deli”, “waterslide”)
  - Names of the people in line (a list of some sort)
  - Average wait time per person

- Next, we need to implement a “class” that stores and manages those pieces of information

Classes need 4 things

0 A name (like ServiceLine!)

1 **Constructor**
What happens when you make a ServiceLine

2 **Instance Variables**
What sub variables each ServiceLine stores

3 **Methods**
What functions you can call on a ServiceLine

Example of a class in Python

serviceline.py:

```
class ServiceLine:
  def __init__(self, name, avg_wait_time):
    self.name = name
    self.people_waiting = []
    self.wait_time = 10

  def add_person(self, name):
    self.people_waiting.append(name)

  def serve_next_person(self):
     self.people_waiting.pop()

 def get_wait_time(self):
     return self.wait_time * len(self.people_waiting)
```

Stanford | ENGINEERING
Computer Science

Example of a class in Python

serviceline.py:

```
class ServiceLine:
  def __init__(self, name, avg_wait_time_mins):
    self.name = name
    self.people_waiting = []
    self.wait_time = avg_wait_time_mins
```

- **This is the constructor** - it is the code that runs when someone makes a new “ServiceLine” type variable
- It takes in a **name** and an **average wait time** as **parameters** - self is a special, invisible-ish parameter that means “the variable I’m currently making”

```
# making ServiceLine variable runs the constructor
dmv_line = ServiceLine(“DMV”, 90)
deli_line =  ServiceLine(“Deli”, 15)
```

Example of a class in Python

- **These are "methods" -** things ServiceLine variables can do

```
dmv_line = ServiceLine("DMV", 90)
dmv_line.get_wait_time() # returns 0
# adds 'frankie' to internal list
dmv_line.add_person("Frankie")

dmv_line.get_wait_time() # returns 90
```

```
cl
 (
def add_person(self, name):
   self.people_waiting.append(name)

def serve_next_person():
    self.people_waiting.pop(self)

def get_wait_time(self):
    return self.wait_time * len(self.people_waiting)
```

Stanford | ENGINEERING
Computer Science

What are Classes and Objects?

- Classes are like blueprints
  - They provide a template for a kind of object
  - They define a new **type**
  - ServiceLine is a class

- Objects are *instances* of Classes
  - Can have multiple objects of the same Class type
  - E.g., **`dmv_line`** and **`deli_line`** are two different instances of ServiceLine
  - They each have their own versions of their internal variables - called **instance variables**!
  - E.g., **`dmv_line`** had a wait time of 90 and a list of length 1, and **`deli_line`** had a wait time of 15 and a list of length 0

To pycharm: lines.py

Lets see it in action

Stanford | ENGINEERING
Computer Science

Objects are Mutable

- When you pass an object as a parameter, mutations in that function persist after function ends

```
from service_line import ServiceLine

def big_rush(line, lots_of_people):
   for person in people:
    line.add_person(person)

def main():
  dmv_line = ServiceLine("DMV", 90)
  big_rush(dmv_line, ["frankie", "ecy", "chris"])
  print(dmv_line.get_wait_time()) # prints 90*3!
```

Remember: moves never persist!

- This isn’t special for classes - but good to remember!

```
from service_line import ServiceLine

def big_rush(line, lots_of_people):
   line = ServiceLine(“New Line”, 90)
   for person in people:
     line.add_person(person)

def main():
  dmv_line = ServiceLine("DMV", 90)
  big_rush(dmv_line, ["frankie", "ecy", "chris"])
  print(dmv_line.get_wait_time()) # prints 90*0!
```

General Form for Writing a Class

- Filename for class is usually ***classname*.py**
  - Filename is usually lowercase version of class name in file

```
class ClassName:
  def __init__(self, var_val):
    # constructor sets up instance variables
    self.instance_var = var_val

  def method1(self):
    # methods do something with an instance
    # they always take in 'self'
  def method2(self, x):
    # methods can take in other params too!
```

Constructor of a Class

- Called when a new object is being created
  - Does not explicitly specify a return value
  - New object is created and returned
    - Can think of constructor as the "factory" that creates new objects
  - Responsible for initializing object (setting initial values)
  - Generally, where instance variables are created (with **self**)

```
class Classname:
   def __init__(self, …):
     # create instance variables
     self.instance_variable_name = value
```

Instance Variables

- Instance variables are variables associated with objects
  - Each object get its **own set** of instance variables
  - Generally, they are initialized in constructor for class
  - They're accessed in the class definition using **`self`**:

    `self.`***variable_name*** `=` ***value***
  - Self really refers to the object that a method is called on

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```

`count` is an instance variable for the counter class!

Stanford | ENGINEERING
Computer Science

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```

```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```

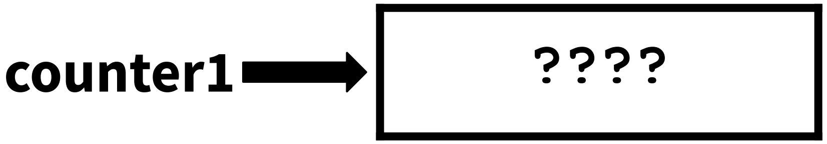
Stanford | ENGINEERING
Computer Science

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```

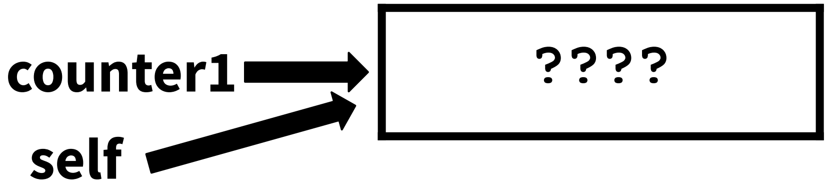
```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```



Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
     self.count += 1
```



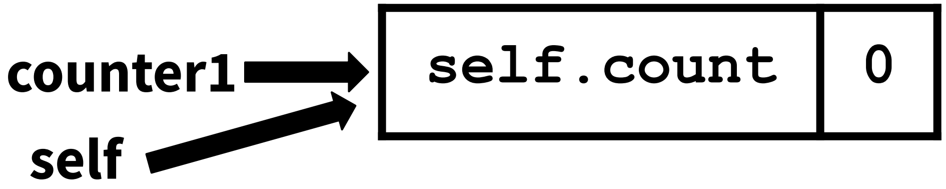
```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```

Stanford | ENGINEERING
Computer Science

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```

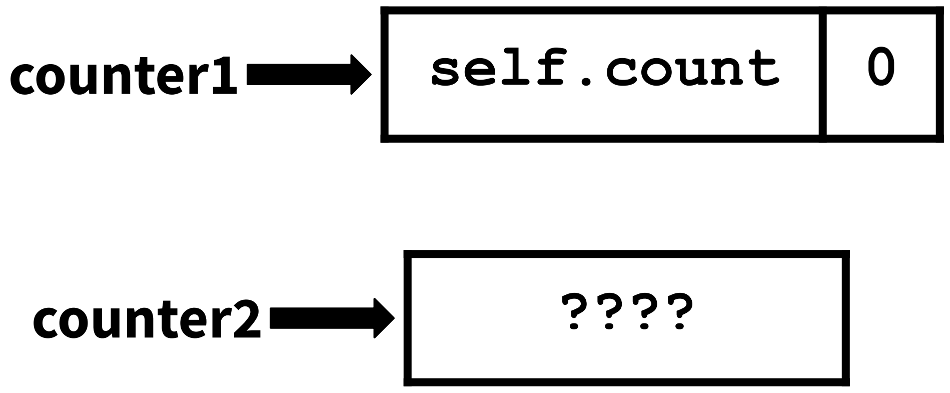


```
from counter import Counter

def main():
  counter1 = Counter()
  counter2 = Counter()
  counter1.next()
  counter1.next()
  counter2.next()
```

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```



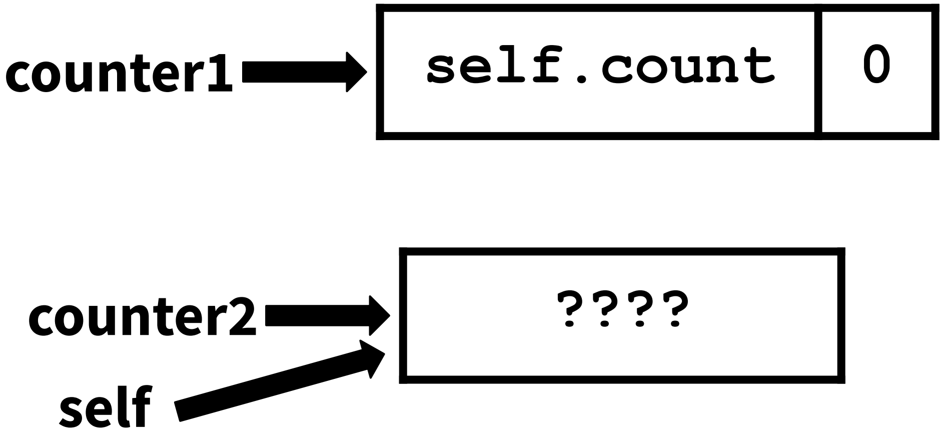
```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```

Stanford | ENGINEERING
Computer Science

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```

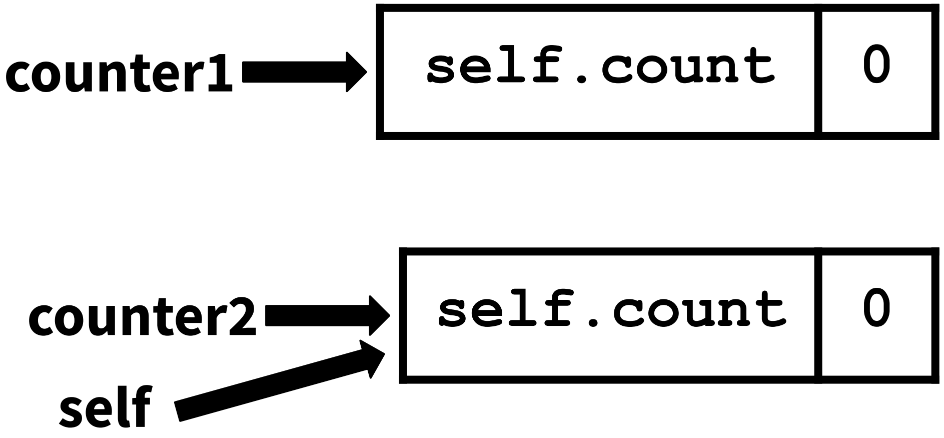


```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```

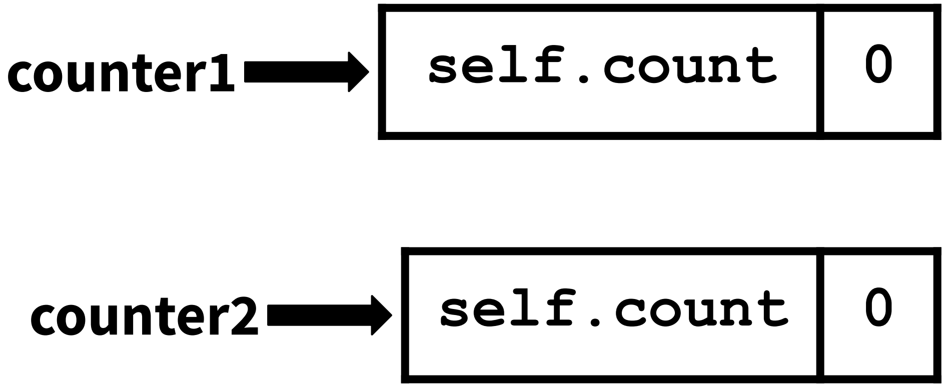


```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```

Instance Variables

- Each Counter has its own count - trace!

```python
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```



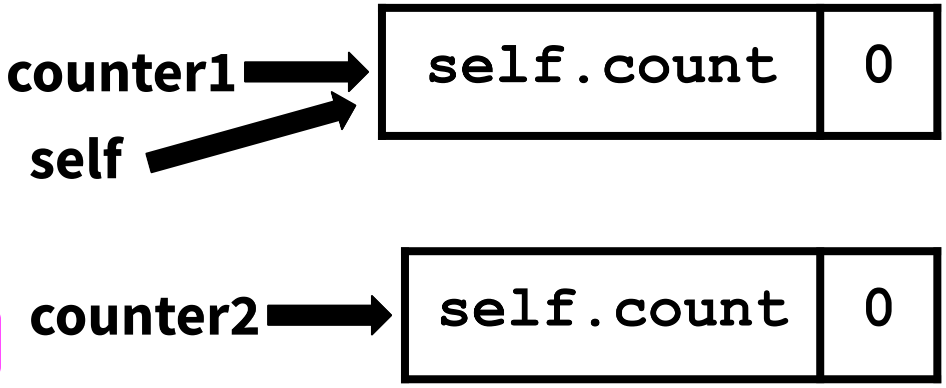
```python
from counter import Counter

def main():
    counter1 = Counter()
    counter2 = Counter()
    counter1.next()
    counter1.next()
    counter2.next()
```

Stanford | ENGINEERING
Computer Science

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```



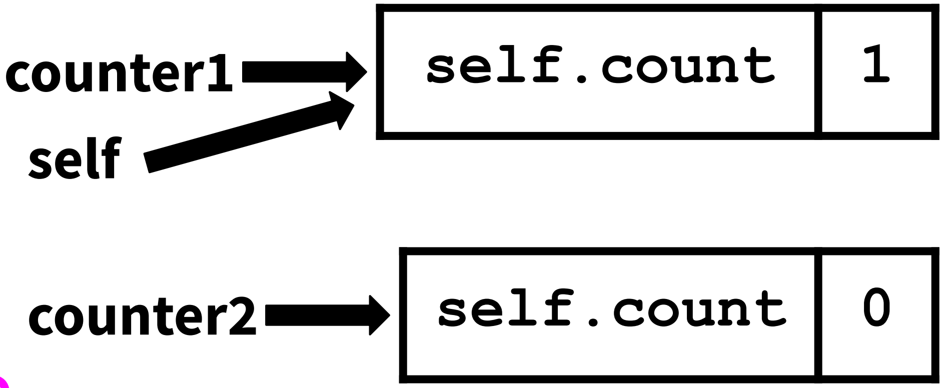
```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```

Stanford | ENGINEERING
Computer Science

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
      self.count += 1
```



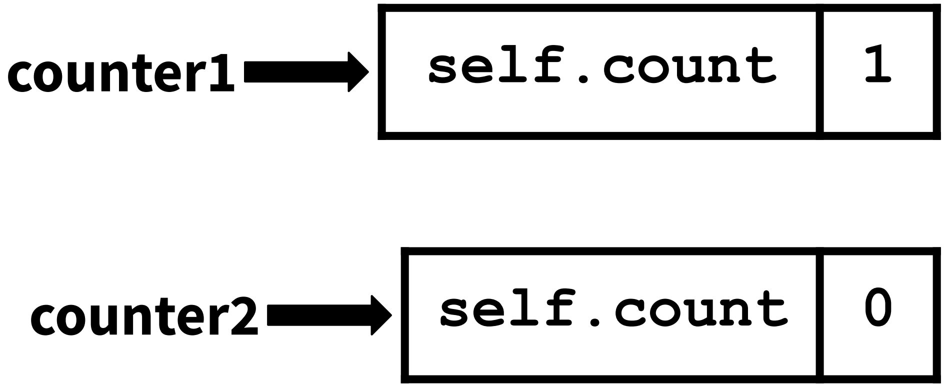
```
from counter import Counter

def main():
    counter1 = Counter()
    counter2 = Counter()
    counter1.next()
    counter1.next()
    counter2.next()
```

Stanford | ENGINEERING
Computer Science

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```

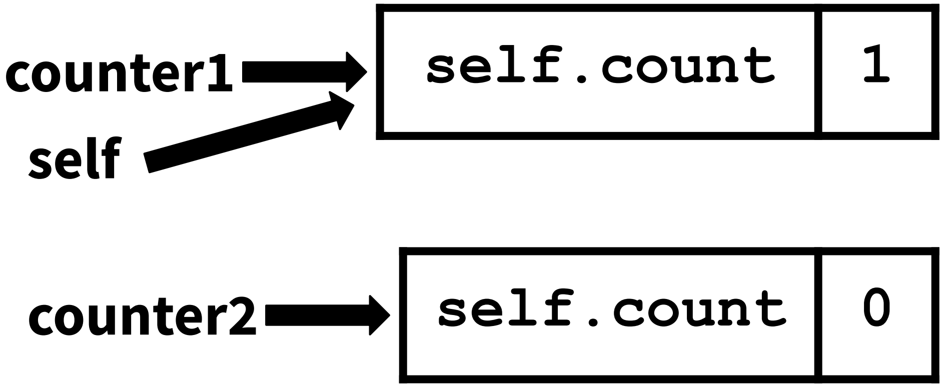


```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```



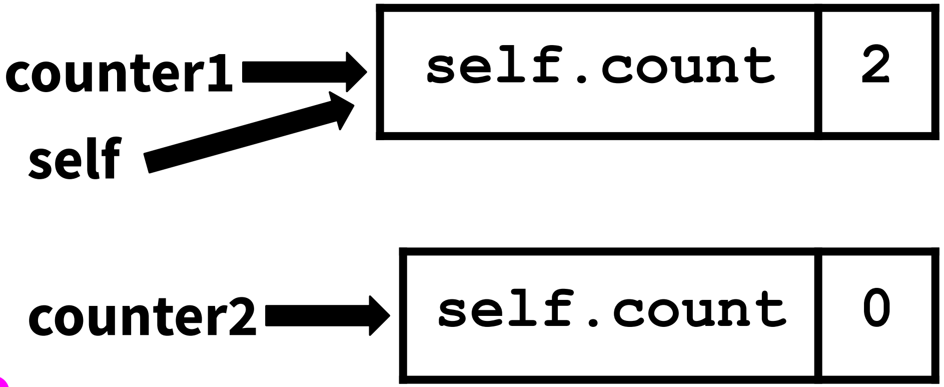
```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```

Stanford | ENGINEERING
Computer Science

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```

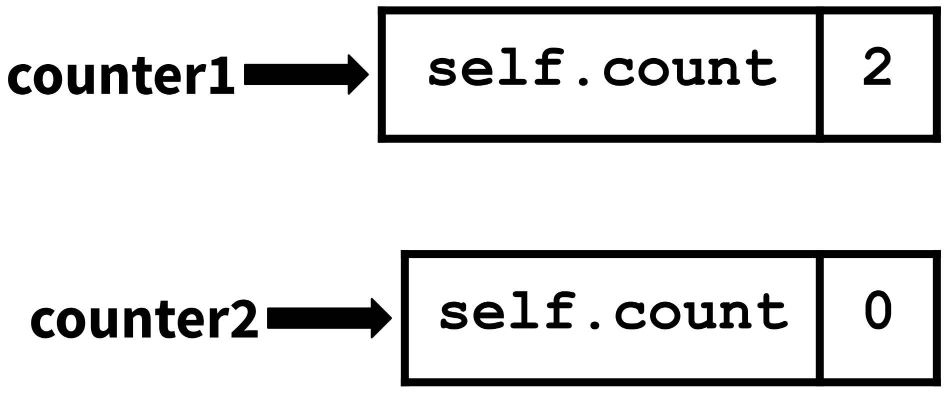


```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```

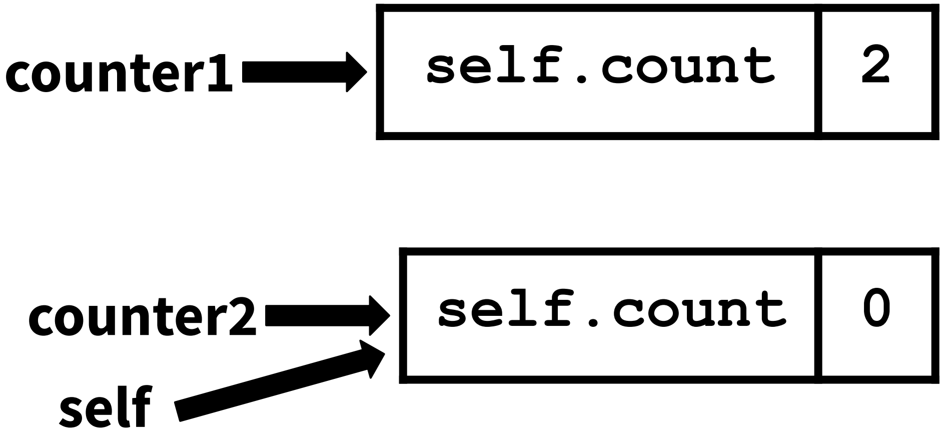


```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
    self.count += 1
```

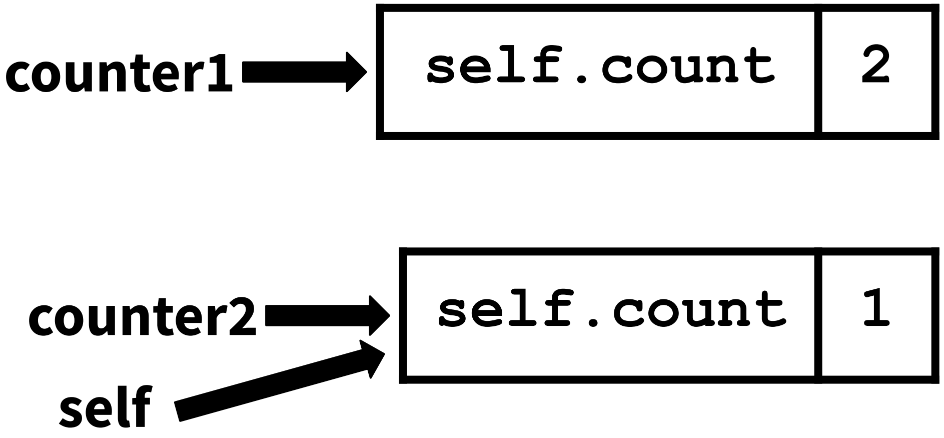


```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```

Instance Variables

- Each Counter has its own count - trace!

```
class Counter:
  def __init__(self):
      self.count = 0

  def next(self):
      self.count += 1
```



```
from counter import Counter

def main():
   counter1 = Counter()
   counter2 = Counter()
   counter1.next()
   counter1.next()
   counter2.next()
```

Stanford | ENGINEERING
Computer Science

Methods (Functions) in Class

- Methods (name used for functions in objects)
  - Syntax:

```
def method_name(self, additional_params):
    body
```

- Works like a regular function in Python
  - Can return values (like a regular function)
  - Has access to *instance* variables (through **`self`**):

```
self.variable_name = value
```

  - Called using an object:

```
object_name.method_name(additional_parameters)
```

  - Recall, parameter **`self`** is automatically set by Python as the object that this method is being called on
    - You write: **`count1.next_value()`**
    - Python treats it as: **`next_value(count1)`**

Next class: bouncing_balls.py

woohoo!

Stanford | ENGINEERING
Computer Science

Recap

- We can define our own custom types 🧠🧠🧠🧠

- To do so, we define a **Class** with a constructor, instance variables and methods

- The we can use as many variables of that type as we want! And they are all independent!