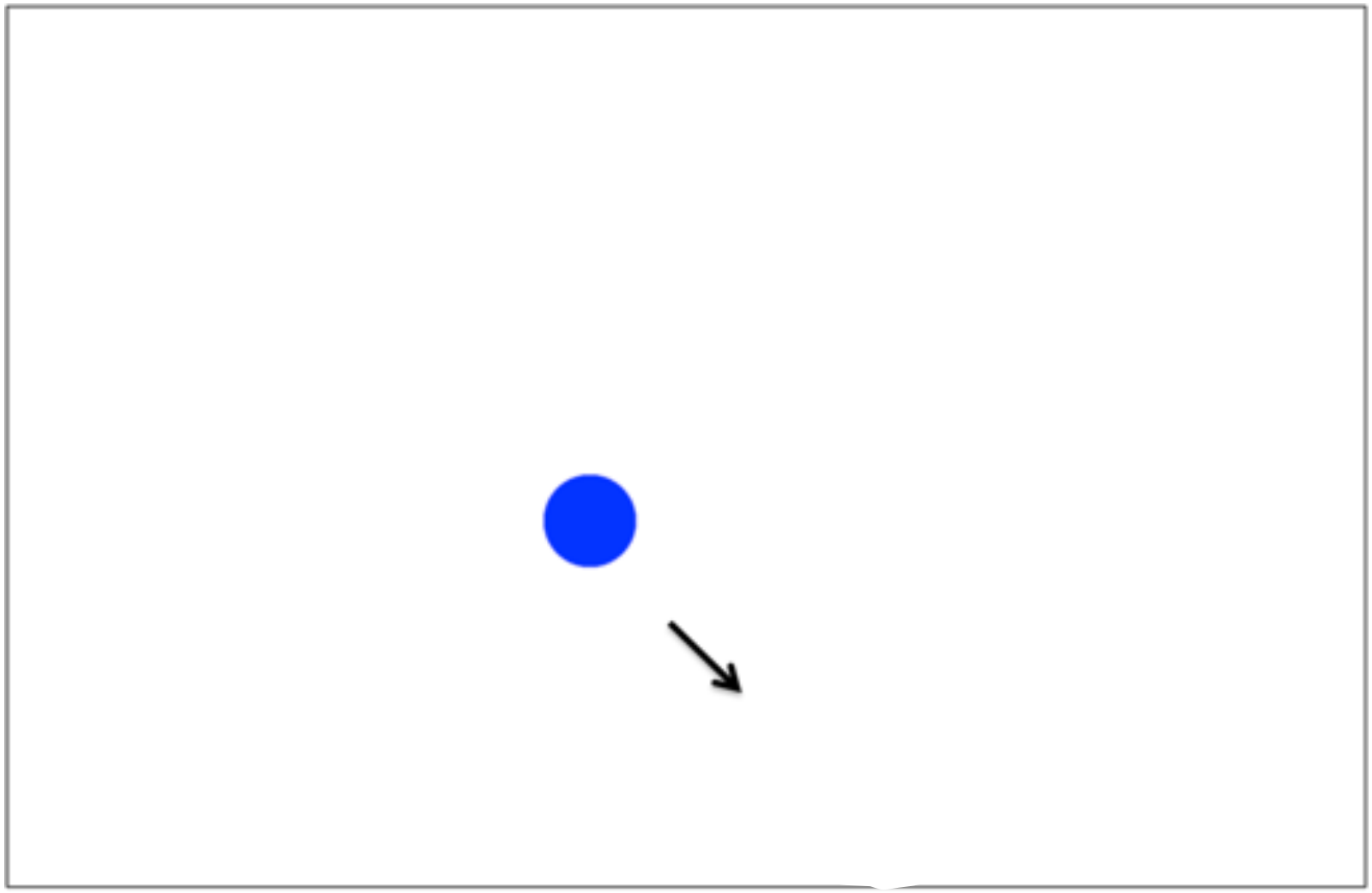
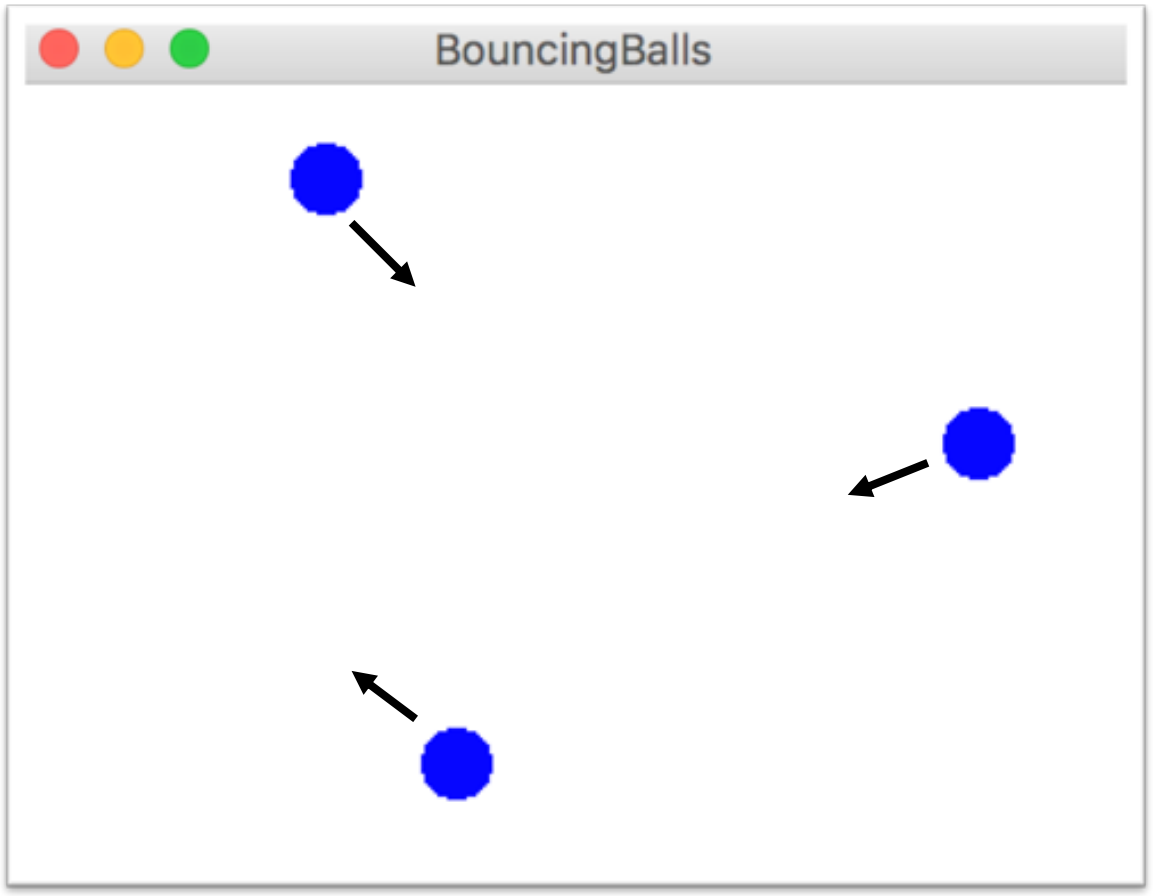
Classes + Memory

Chris Piech and Mehran Sahami
CS106A, Stanford University

Remember this?



Bouncing Balls



Housekeeping

- Hope you are all doing well

Learning Goals

1. Practice with classes
2. See how to trace memory with classes

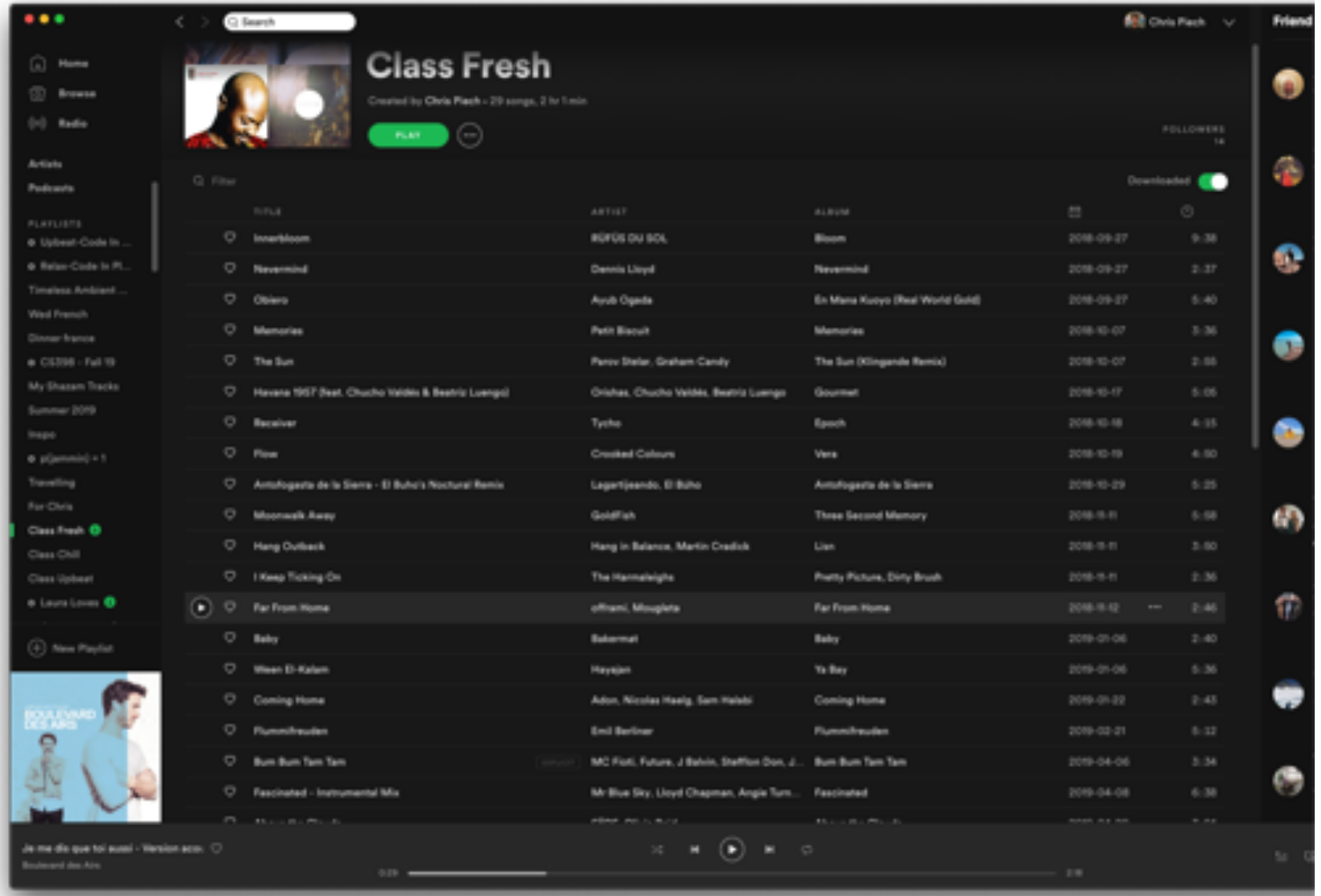
Guiding question for today:

what does it take to go from
what you know to writing
big-scale software?

Some *large* programs are in Python

YouTube


LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891


Class Fresh
Created by Chris Piech · 29 songs, 2 hr 1 min
PLAY
Downloaded
TITLE
ARTIST
ALBUM
Innerbloom
RÜFÜS DU SOL
Bloom
Nevermind
Dennis Lloyd
Nevermind
Obiero
Ayub Ogada
En Mana Kuoyo (Real World Gold)
Memories
Petit Biscuit
Memories
The Sun
Penny Stellar, Graham Candy
The Sun (Klingande Remix)
Havana 1957 (feat. Chucho Valdés & Beatriz Luengo)
Orishas, Chucho Valdés, Beatriz Luengo
Gourmet
Receiver
Tycho
Epoch
Flow
Crooked Colours
Vera
Antofagasta de la Sierra - El Buho's Noctural Remix
Lagartijeando, El Buho
Antofagasta de la Sierra
Moonwalk Away
GoldFish
Three Second Memory
Hang Outback
Hang in Balance, Martin Cradick
Lion
I Keep Ticking On
The Hammeleighs
Pretty Picture, Dirty Brush
Far From Home
ofnami, Mougleta
Far From Home
Baby
Bakermat
Baby
Ween El-Kalam
Hayajan
Ya Bey
Coming Home
Adon, Nicolas Haelg, Sam Halabi
Coming Home
Flummifreuden
Emil Berliner
Flummifreuden
Bum Bum Tam Tam
MC Fioti, Future, J Balvin, Stefflon Don, J...
Bum Bum Tam Tam
Fascinated - Instrumental Mix
Mr Blue Sky, Lloyd Chapman, Angie Turn...
Fascinated
Je me dis que toi aussi - Version acou...
Boulevard des Airs
Chris Piech
Friend

How?

Define New Variable Types

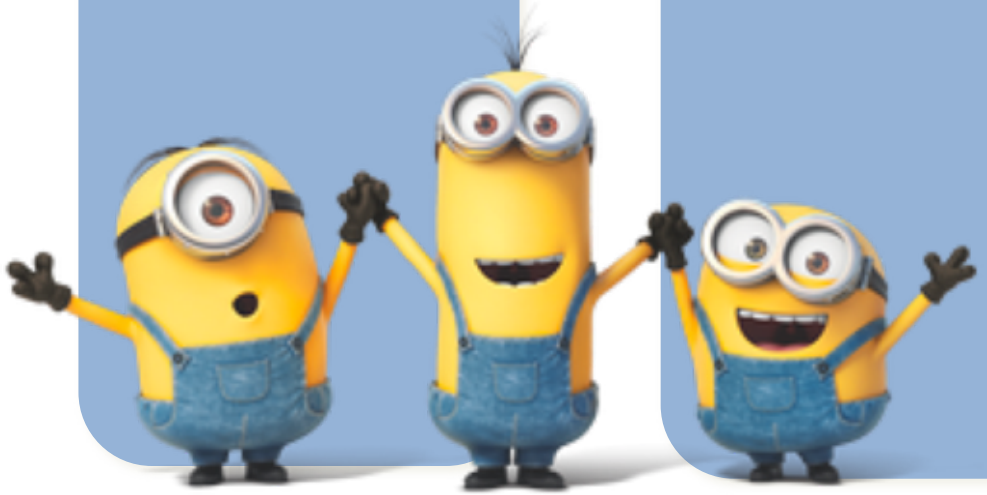
Song

Playlist

User

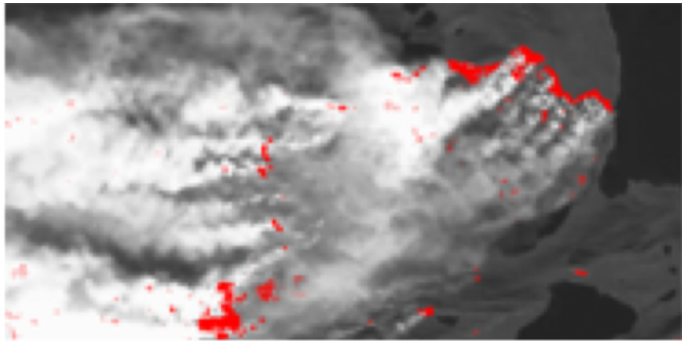
Song Player

Song Retriever

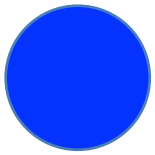


You Have Been *Using* Variable Types

`SimpleImage`



`Canvas`



`Karel`

`String`

`int`

What would it take to define your own?

type


LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

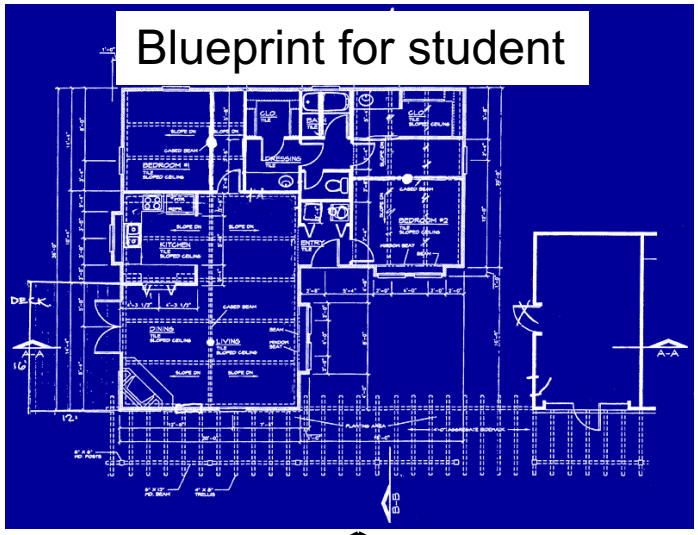
Classes define new variable *types*

Classes decompose your program across files

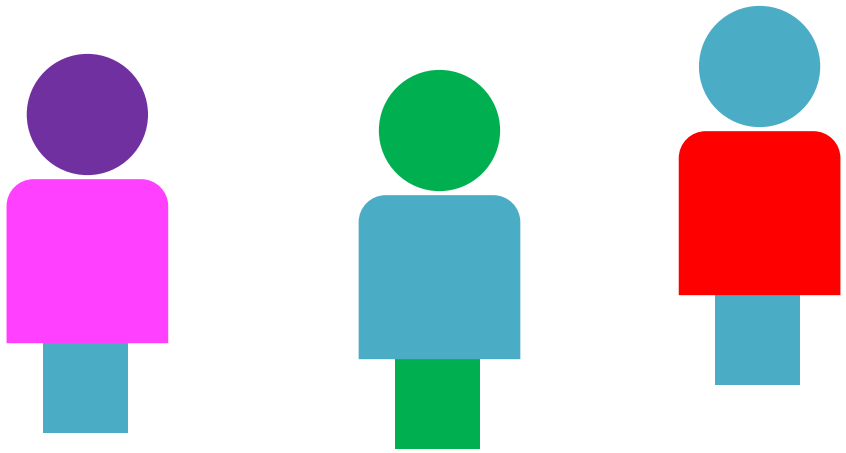
Classes are like blueprints

class: A template for a new type of variable.

A blueprint is a helpful analogy



When defining a new variable type you make a blueprint



You must define three things

Instance Variables

1. What **sub-variables** does each instance store?

Instance Methods

2. What **methods** can you call on an instance?

Constructor

3. What happens when you make a `new` one?

*details on how to define these three things coming soon

Classes Review

dog.py

```
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        print('woof')
        self.times_barked += 1
```

life.py

```
def main():
    simba = Dog()
    juno = Dog()

    simba.bark()
    juno.bark()
    simba.bark()

    print(simba.__dict__)
    print(juno.__dict__)
```

Classes Review

dog.py

```
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        print('woof')
        self.times_barked += 1
```

life.py

```
def main():
    simba = Dog()
    juno = Dog()

    simba.bark()
    juno.bark()
    simba.bark()

    print(simba.__dict__)
    print(juno.__dict__)
```

1. What **variables** does each instance store?

Classes Review

dog.py

```
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        print('woof')
        self.times_barked += 1
```

life.py

```
def main():
    simba = Dog()
    juno = Dog()

    simba.bark()
    juno.bark()
    simba.bark()

    print(simba.__dict__)
    print(juno.__dict__)
```

2. What **methods** can you call on an instance?

Classes Review

dog.py

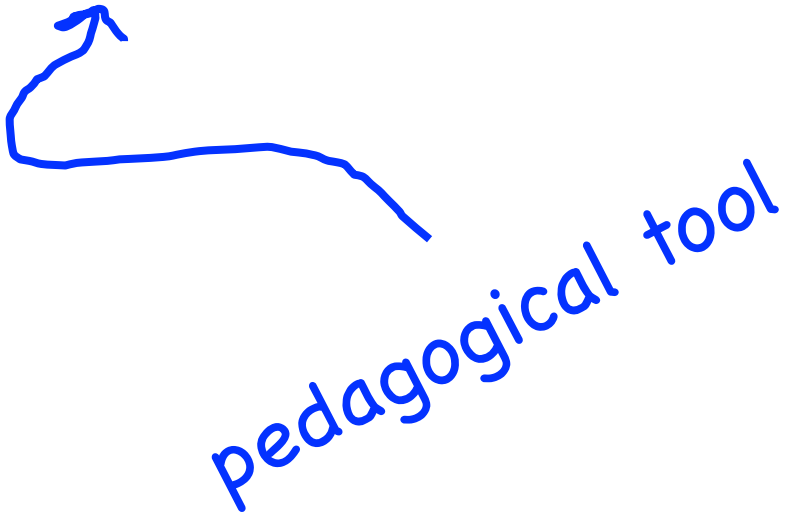
```
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        print('woof')
        self.times_barked += 1
```

life.py

```
def main():
    simba = Dog()
    juno = Dog()

    simba.bark()
    juno.bark()
    simba.bark()

    print(simba.__dict__)
    print(juno.__dict__)
```

3. What happens when you make a **new** one?

.__dict__



Classes Review

dog.py

```
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        print('woof')
        self.times_barked += 1
```

life.py

```
def main():
    simba = Dog()
    juno = Dog()

    simba.bark()
    juno.bark()
    simba.bark()

    print(simba.__dict__)
    print(juno.__dict__)
```

Classes Review

dog.py

```
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        print('woof')
        self.times_barked += 1
```

life.py

```
def main():
    simba = Dog()
    juno = Dog()

    simba.bark()
    juno.bark()
    simba.bark()

    print(simba.__dict__)
    print(juno.__dict__)
```

Did I mention that a class is like a fancy dictionary?

What is self?

dog.py

```
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        print('woof')
        self.times_barked += 1
```

life.py

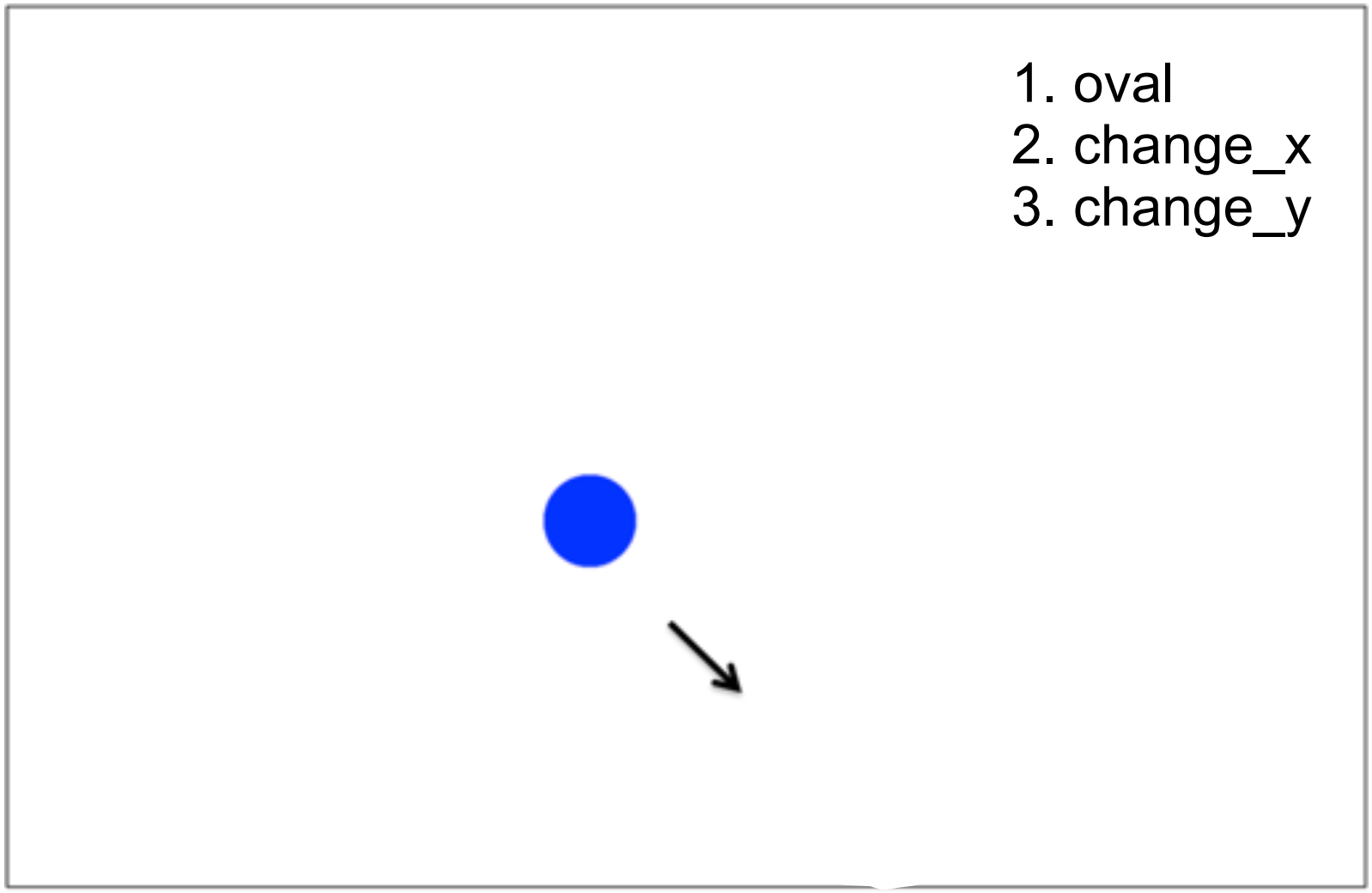
```
def main():
    simba = Dog()
    juno = Dog()

    simba.bark()
    juno.bark()
    simba.bark()

    print(simba.__dict__)
    print(juno.__dict__)
```

When authoring a class, `self` means:
"the instance (aka object) I am currently working with"

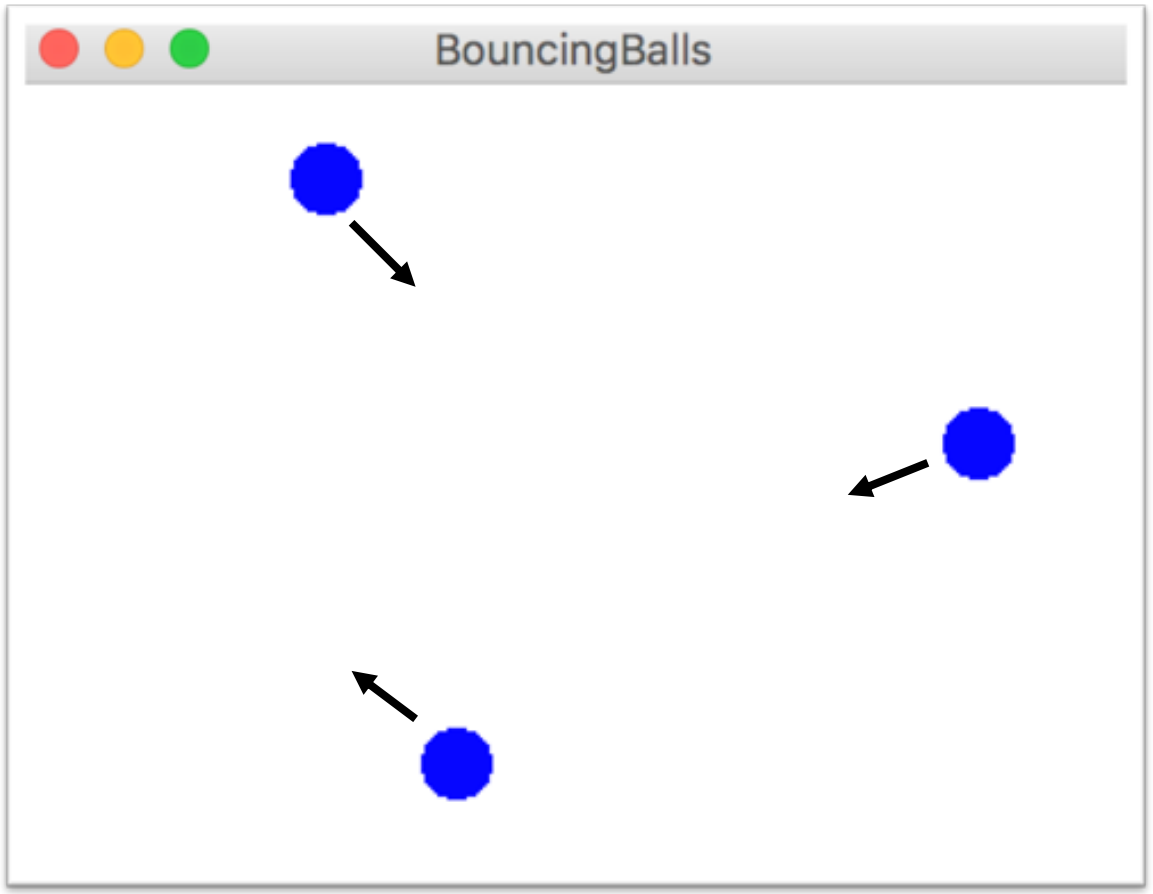
What does a `class` do?

A class defines a new variable type

How many variables for the ball?



How many variables for 3 balls?



1: Store a list of dictionaries

2: Store a list of Balls

Recall Functions?

Coder: **Function Author**

Writes helper functions others can use

Coder: **Function Caller**

Uses helper functions

Classes also split up the work!

Coder: **Class Author**

Writes the class (often in its own file), thus defining a new variable type

Coder: **Class Client**

Uses the new variable type to solve problems (often from main).

Because they are classy

```
def main():
    # gotta have a canvas!
    canvas = Canvas(CANVAS_WIDTH, CANVAS_HEIGHT,
    balls = create_balls(canvas)
    while True:
        # update world
        for ball in balls:
            ball.update(canvas)
        # redraw canvas
        canvas.update()
        time.sleep(1/50.)

def create_balls(canvas):
    balls = []
    for i in range(N_BALLS):
        ball = Ball(canvas)
        balls.append(ball)
    return balls

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

```
import random
from constants import *

class Ball:
    """
    This is the blueprint for a new variable type
    Every ball has three things: an oval, a change
    Every ball supports the "update" method which

    When defining a new variable type you must spe
    (1) What happens when you create a new instanc
    (2) What sub-variables does every instance hav
    (3) What methods (aka functions) can you call
    """

    # (1) This defines the "constructor": what hap
    # instance of type Ball. Note the use of 'canv
    def __init__(self, canvas):
        # When making a new ball, first chose its
        x_1 = random.randint(0, CANVAS_WIDTH - BAL
        y_1 = random.randint(0, CANVAS_HEIGHT - BA
        x_2 = x_1 + BALL_SIZE
```

Class Client: Uses the new variable type to solve problems (often from main).

Class Author: Writes the class, thus defining a new variable type (often in its own file)

Next step in writing large programs:
Better understand memory

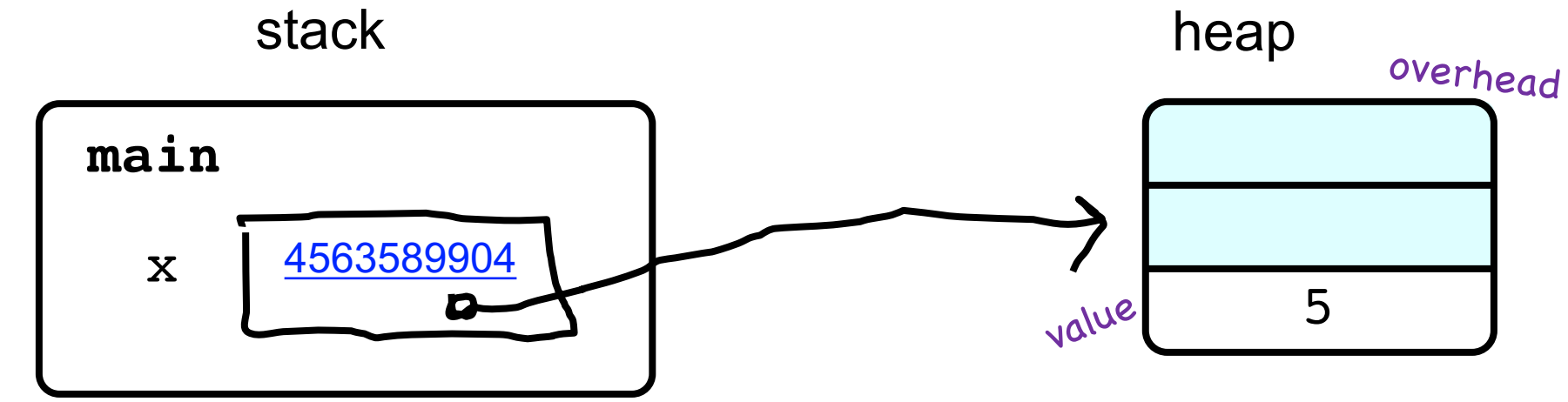
You are now ready…

What does this do?

```
def main():
    x = 5
    print(id(x))
    x += 1
    print(id(x))
```

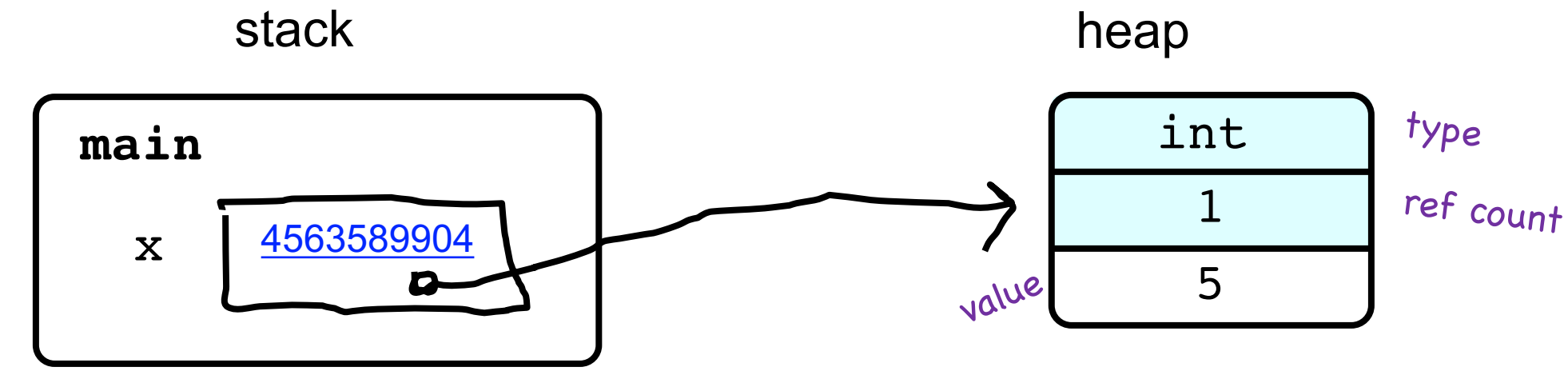

What does this do?

```
def main():
    x = 5
    print(id(x))
    x += 1
    print(id(x))
```



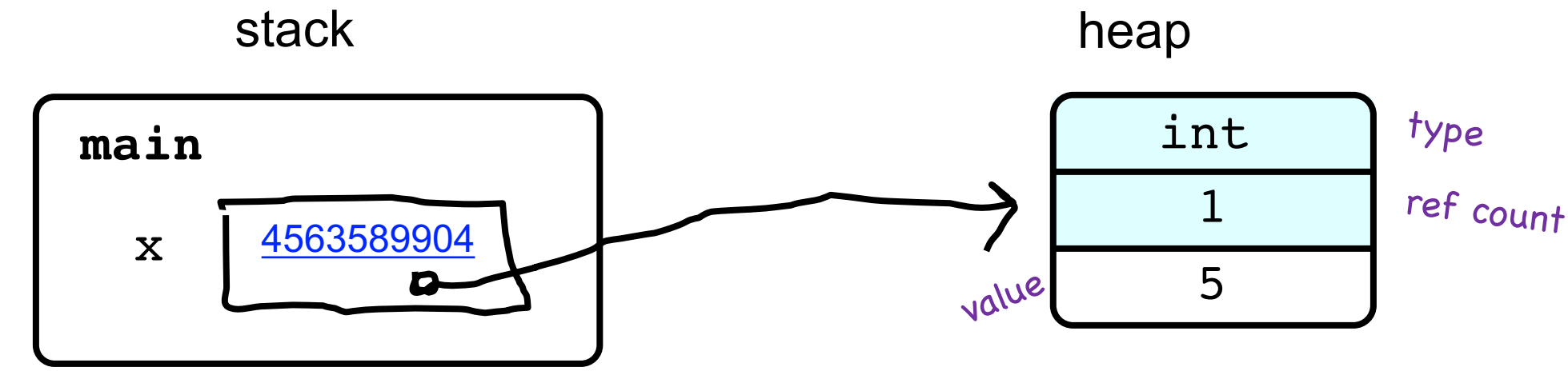
What does this do?

```
def main():
    x = 5
    print(id(x))
    x += 1
    print(id(x))
```



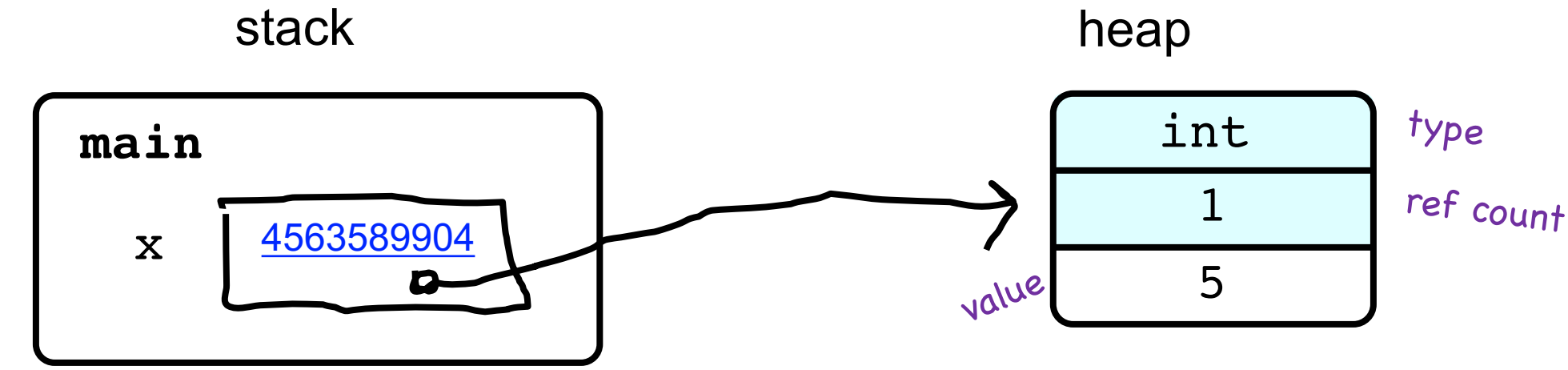
What does this do?

```
def main():
    x = 5
    print(id(x))
    x += 1
    print(id(x))
```



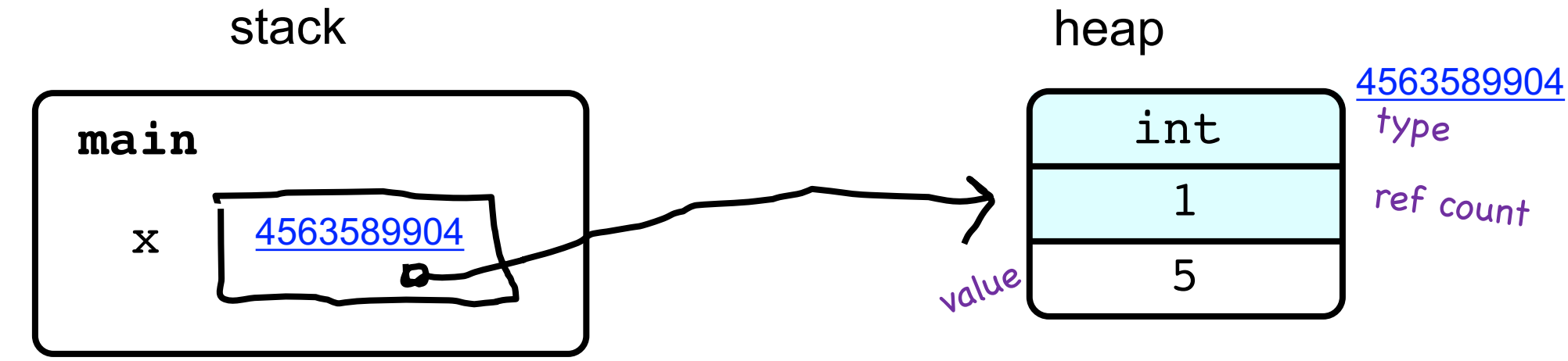
What does this do?

```
def main():
    x = 5
    print(id(x))
    x += 1
    print(id(x))
```



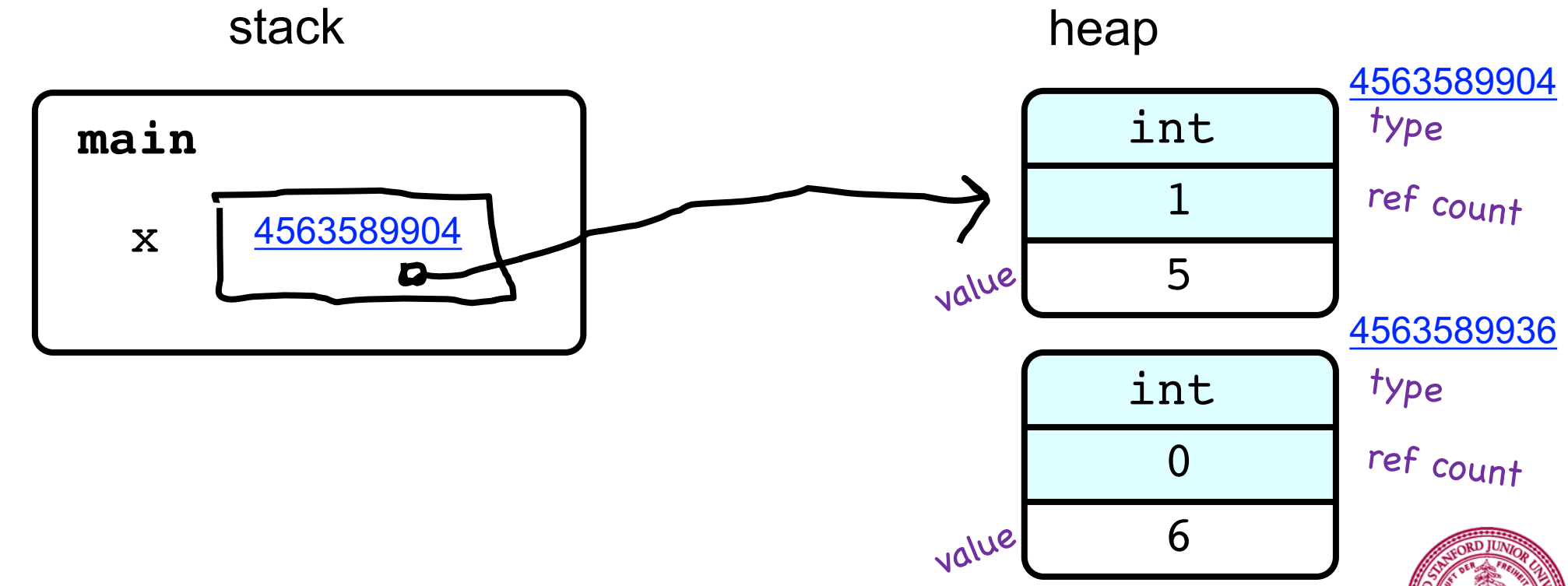
What does this do?

```
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))
```



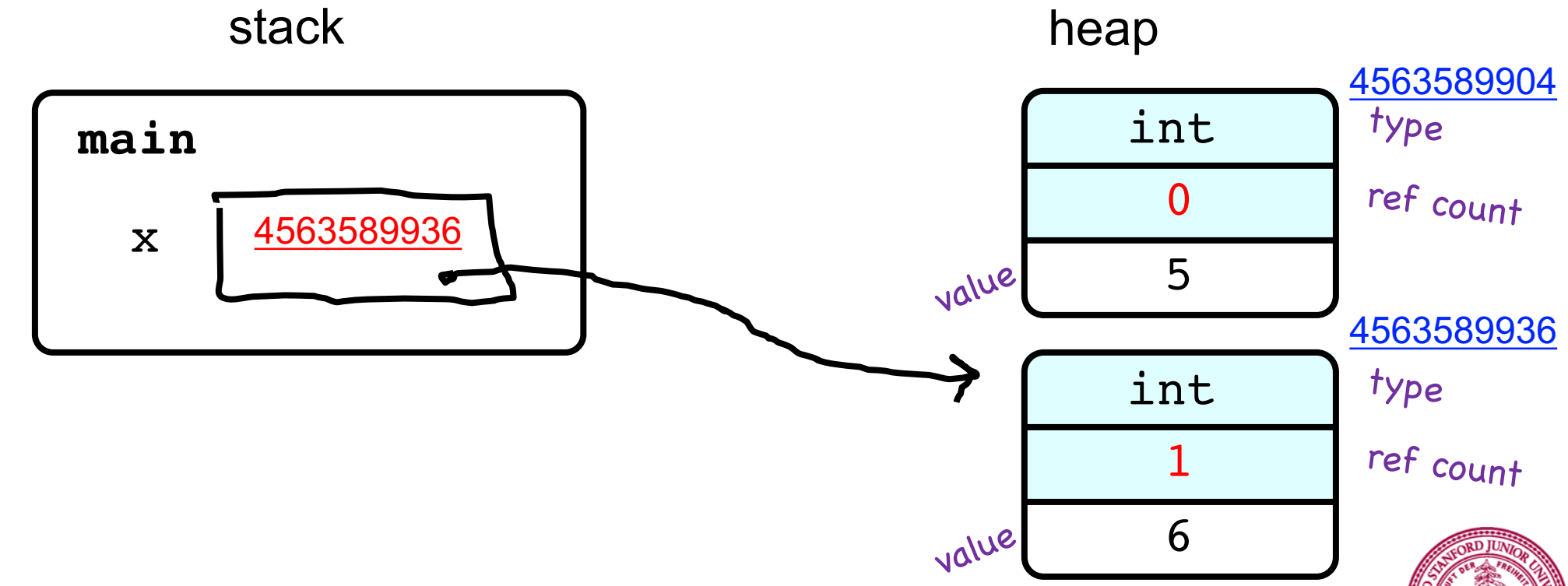
What does this do?

```
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))
```



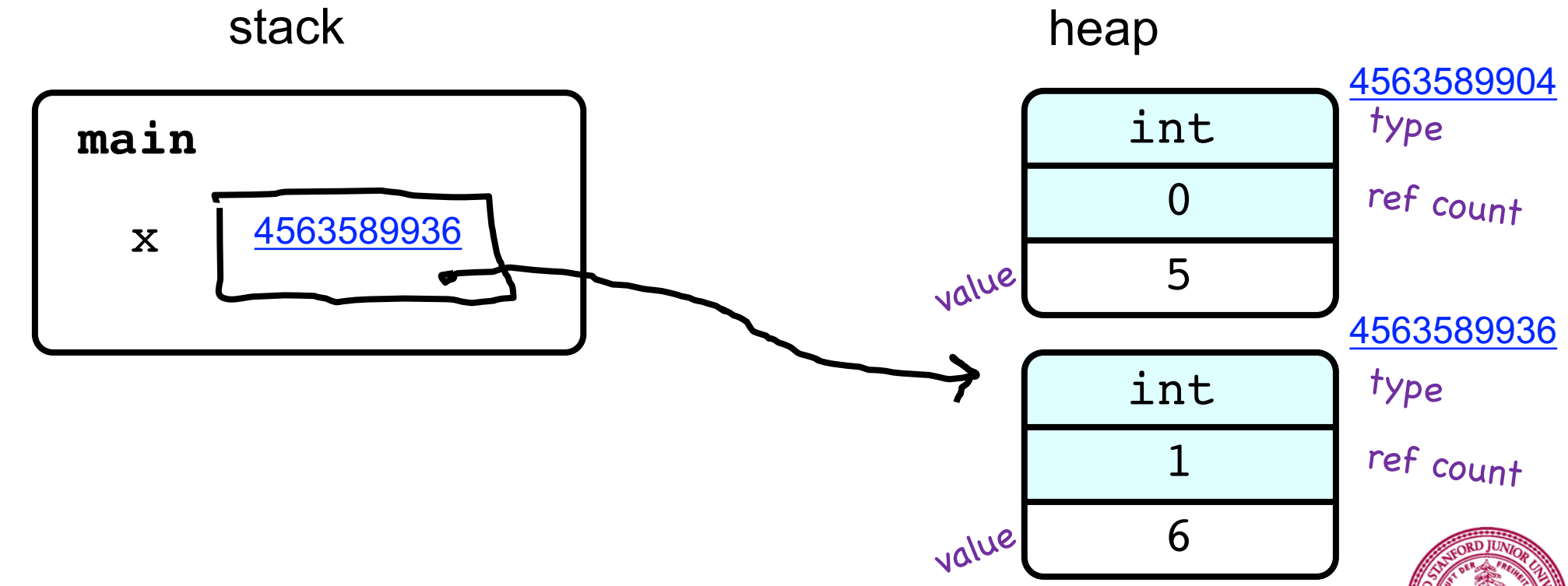
What does this do?

```
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))
```

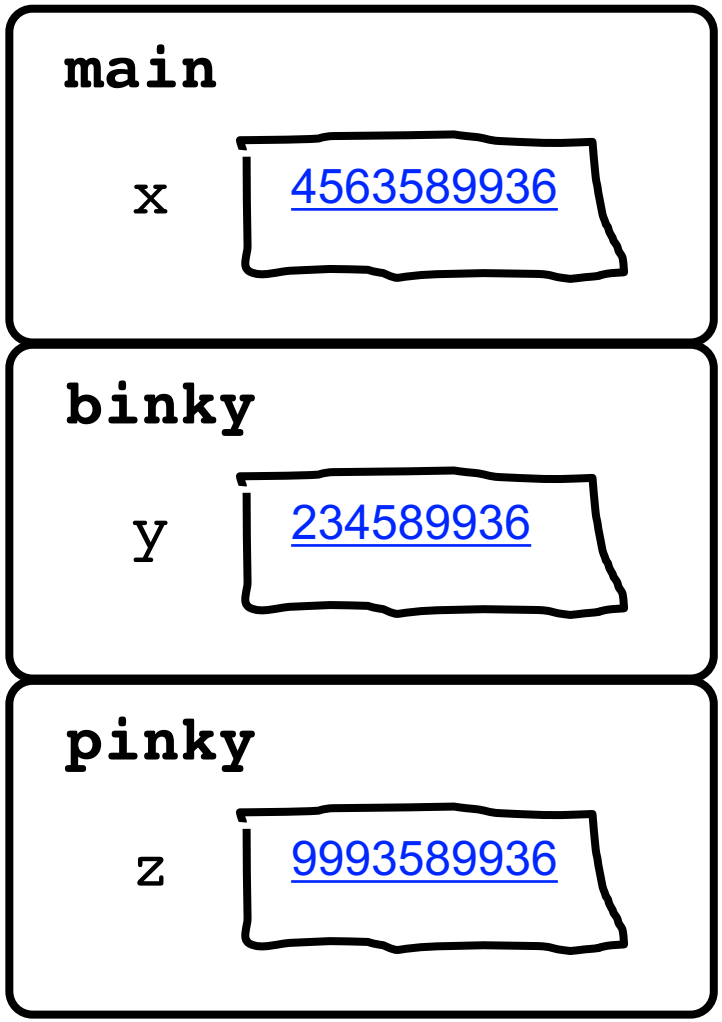


What does this do?

```
def main():
    x = 5
    print(id(x))
    x = x + 1
    print(id(x))
```



The stack



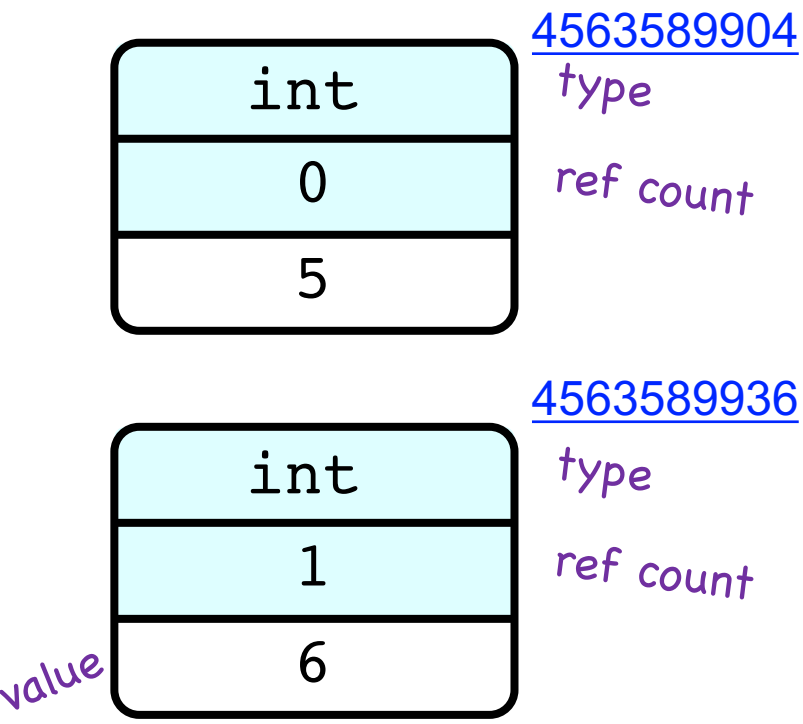
Each time a function is called, a new frame of memory is created.

Each frame has space for all the local variables declared in the function, and parameters

Each variable has a reference which is like a URL

When a function returns, its frame is destroyed.

The heap



Where values are stored

Every value has an address (like a URL address)

Values don't go away when functions return

Memory is recycled when its no longer used.

Deconstructed Samosa

```
def main():
    x = 5
    x = x + 1
```


What does this do?

```
def main():
    x = 5
    x = x + 1
```

When a variable is “assigned” via binding you are changing its **reference**

You know a variable is being assigned to if it is on the left hand side of an = sign

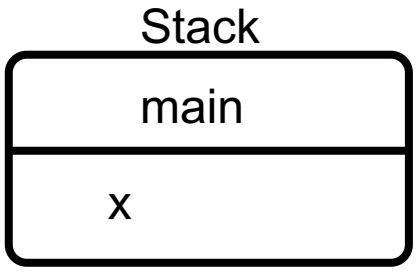
What does this do?

```
def main():
    x = 5
    x = x + 1
```

When a variable is “used” you are accessing its **value**

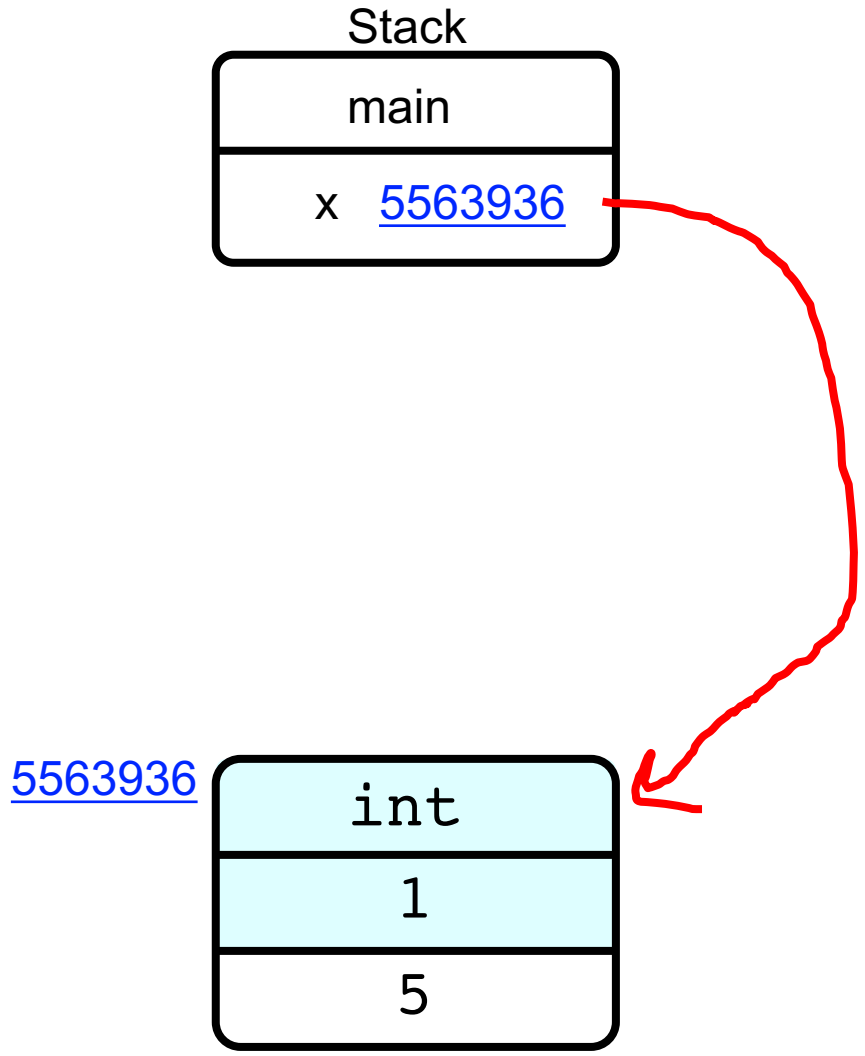
What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



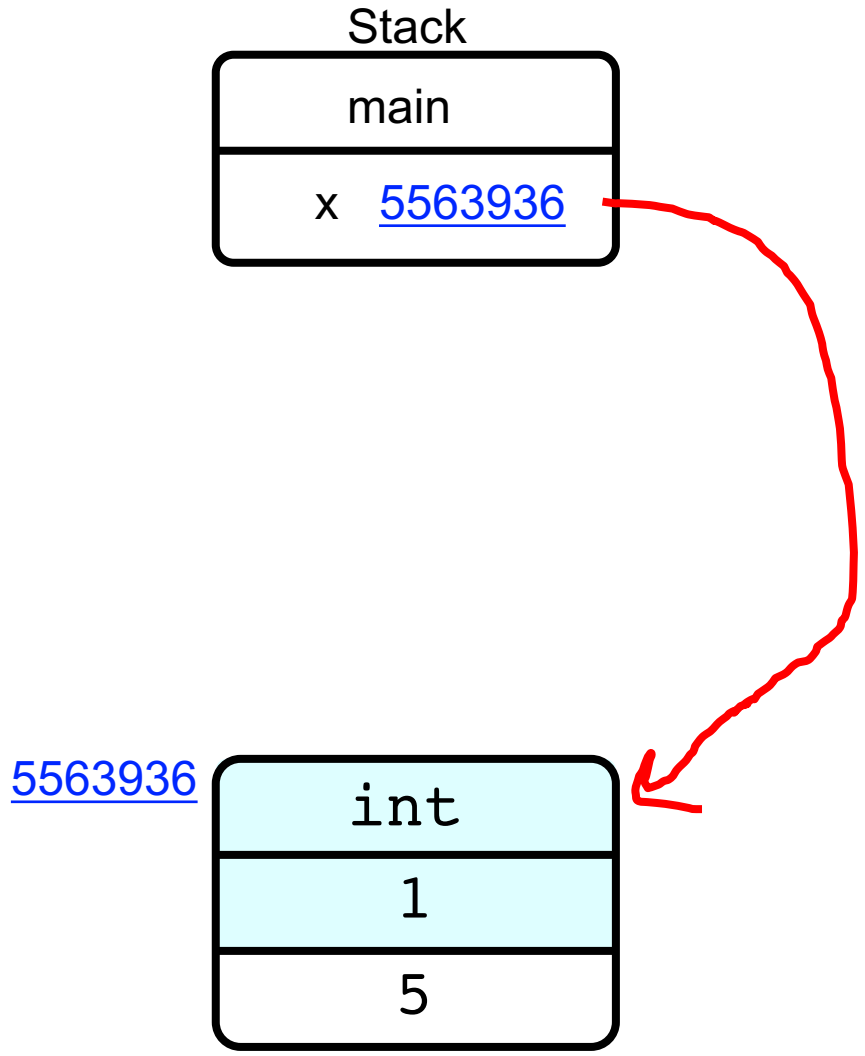
What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



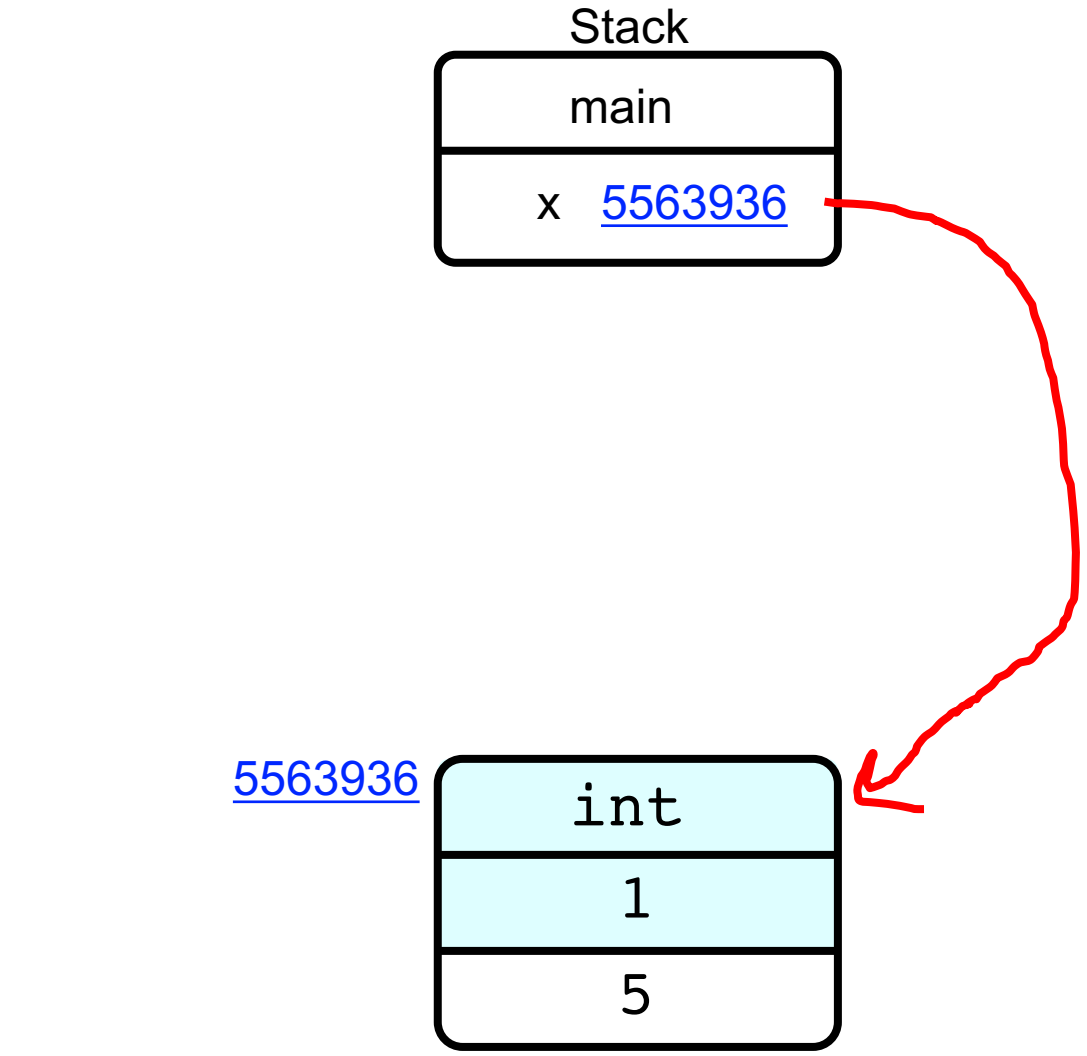
What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



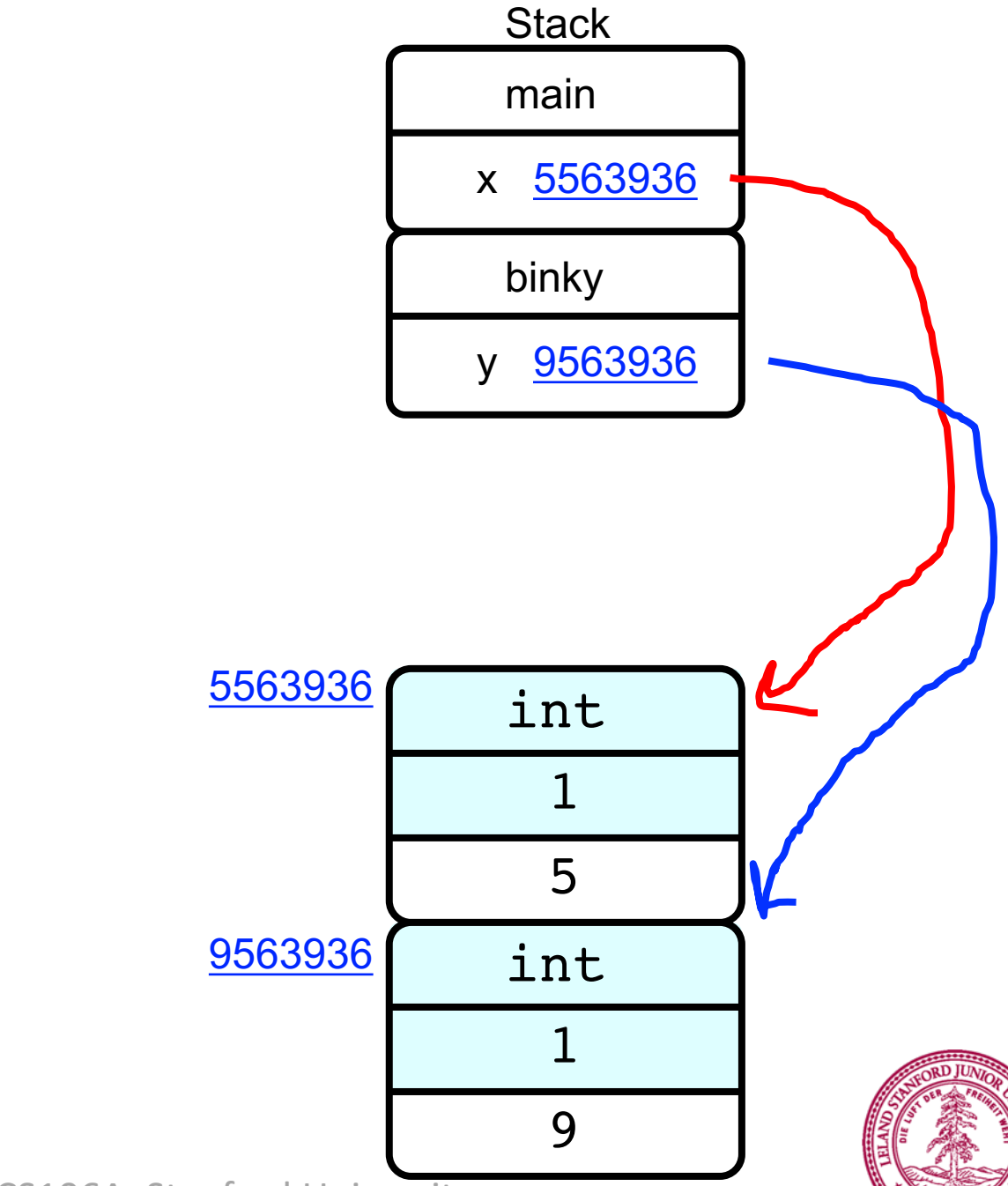
What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



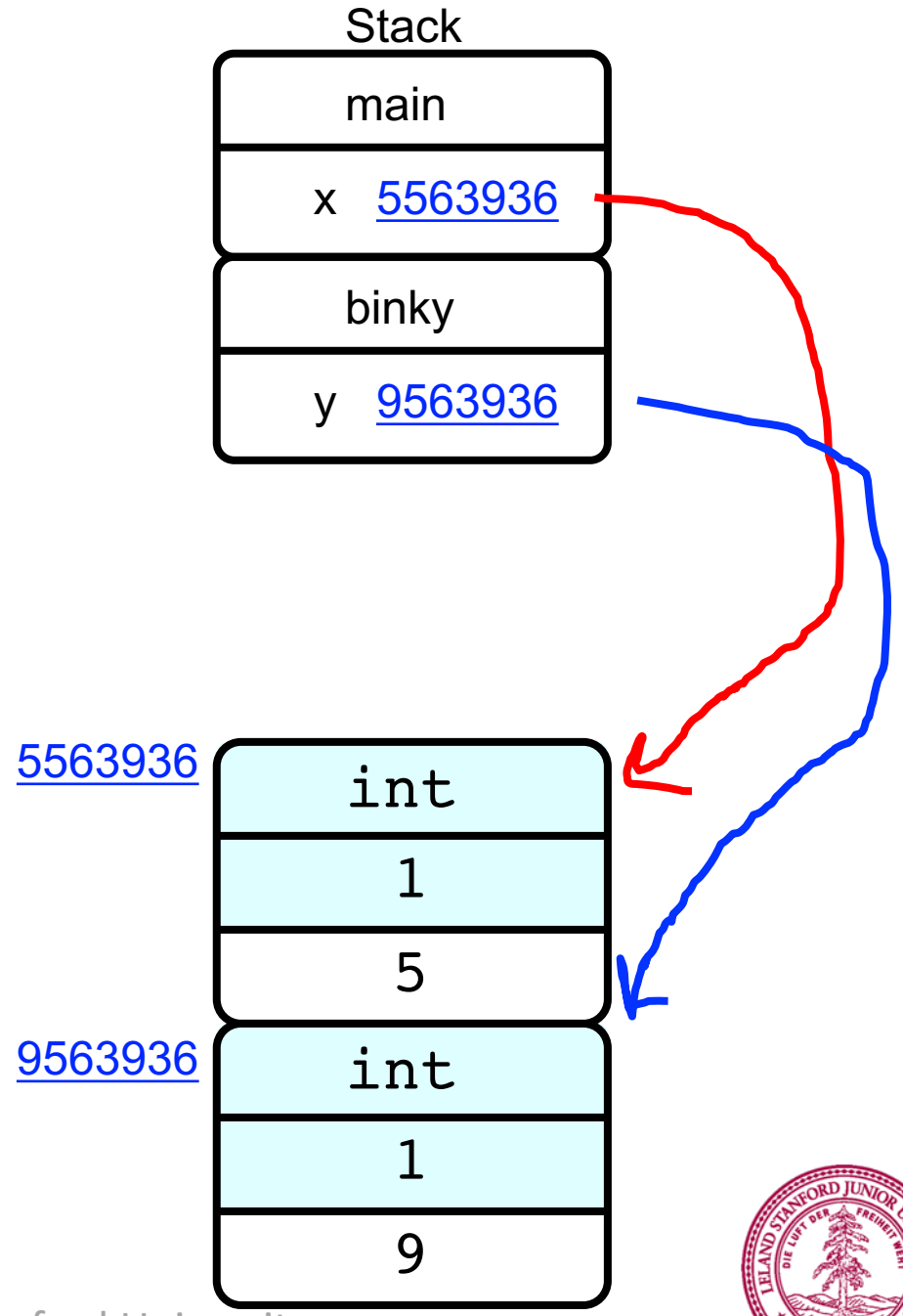
What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



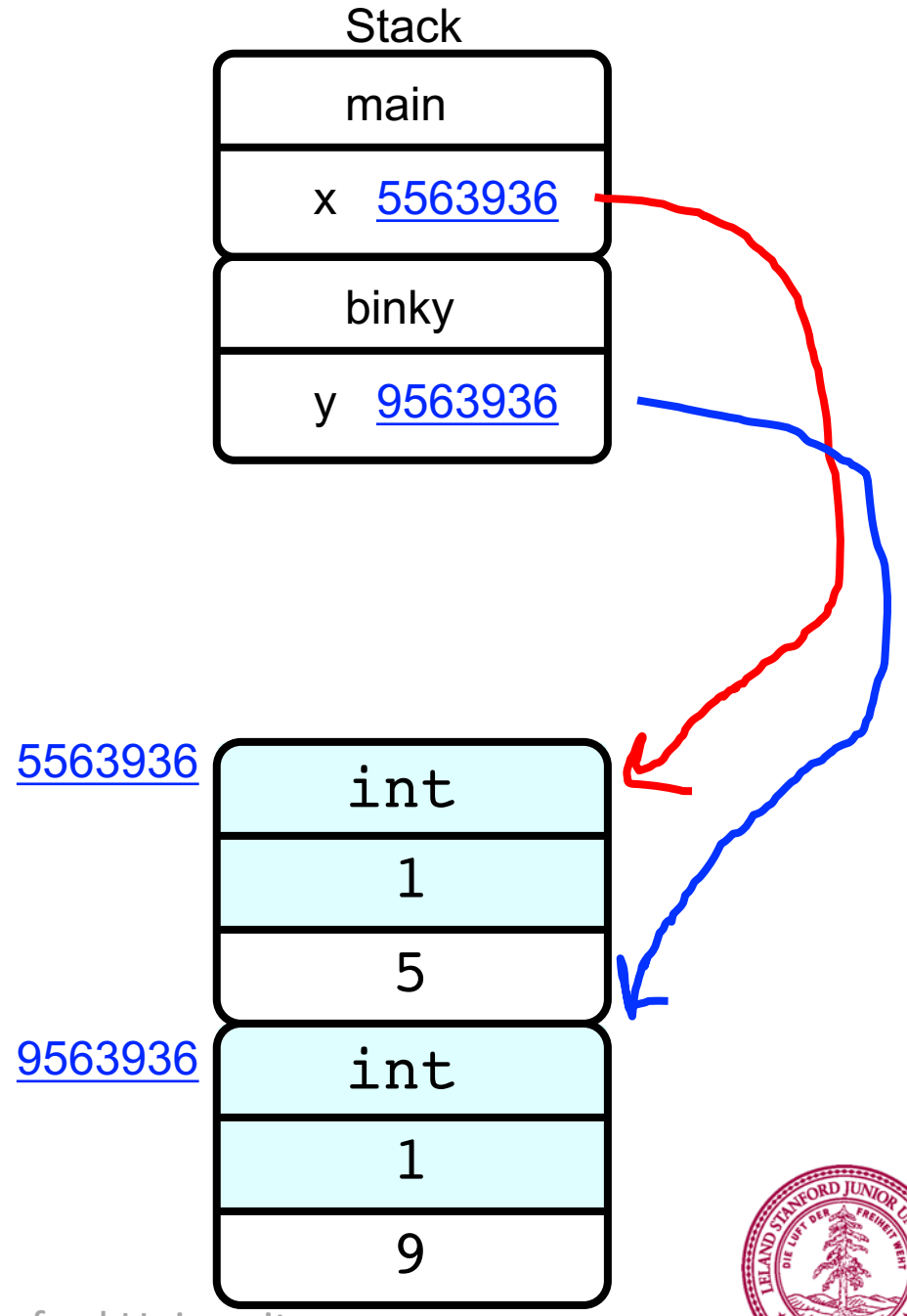
What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



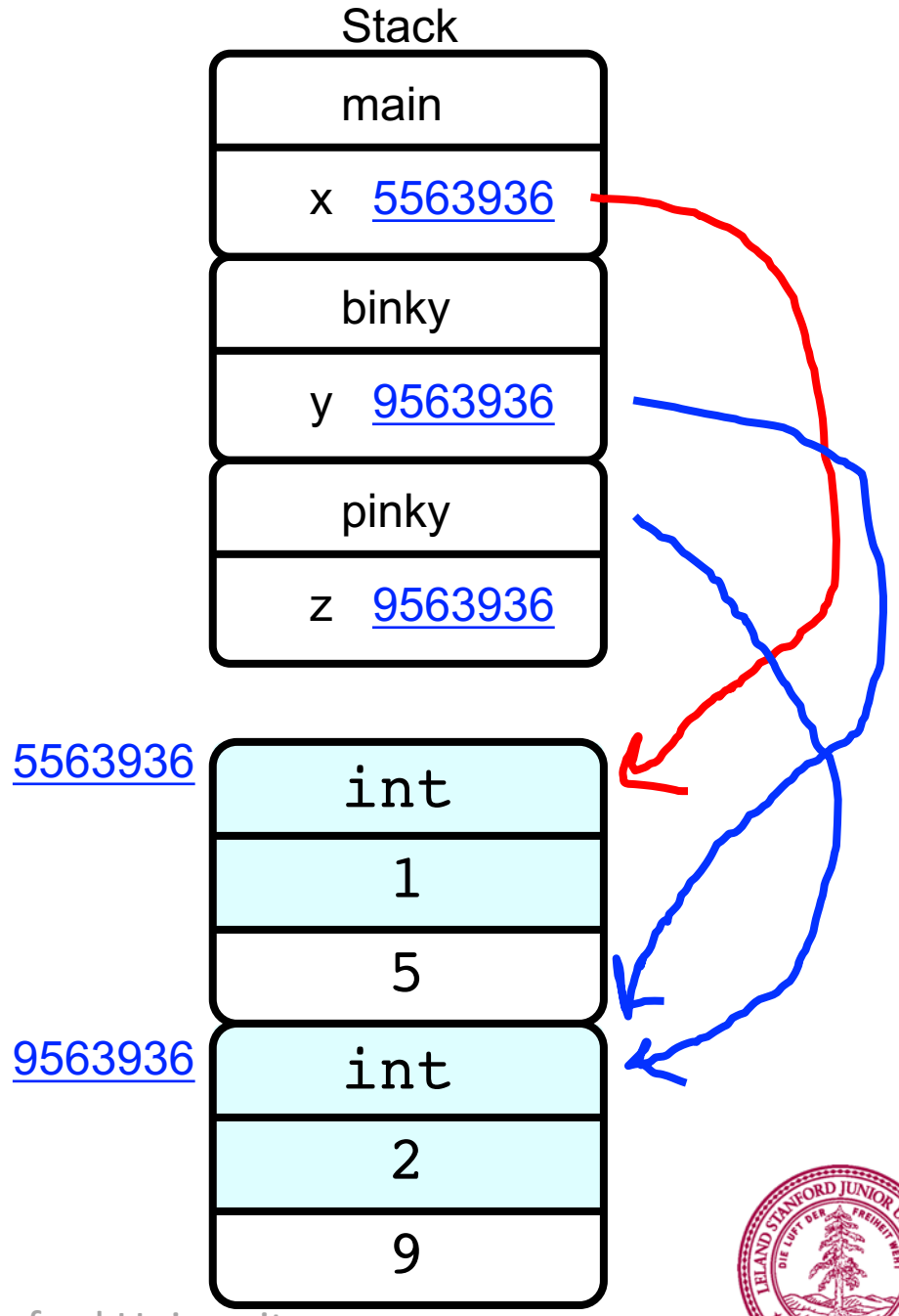
What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



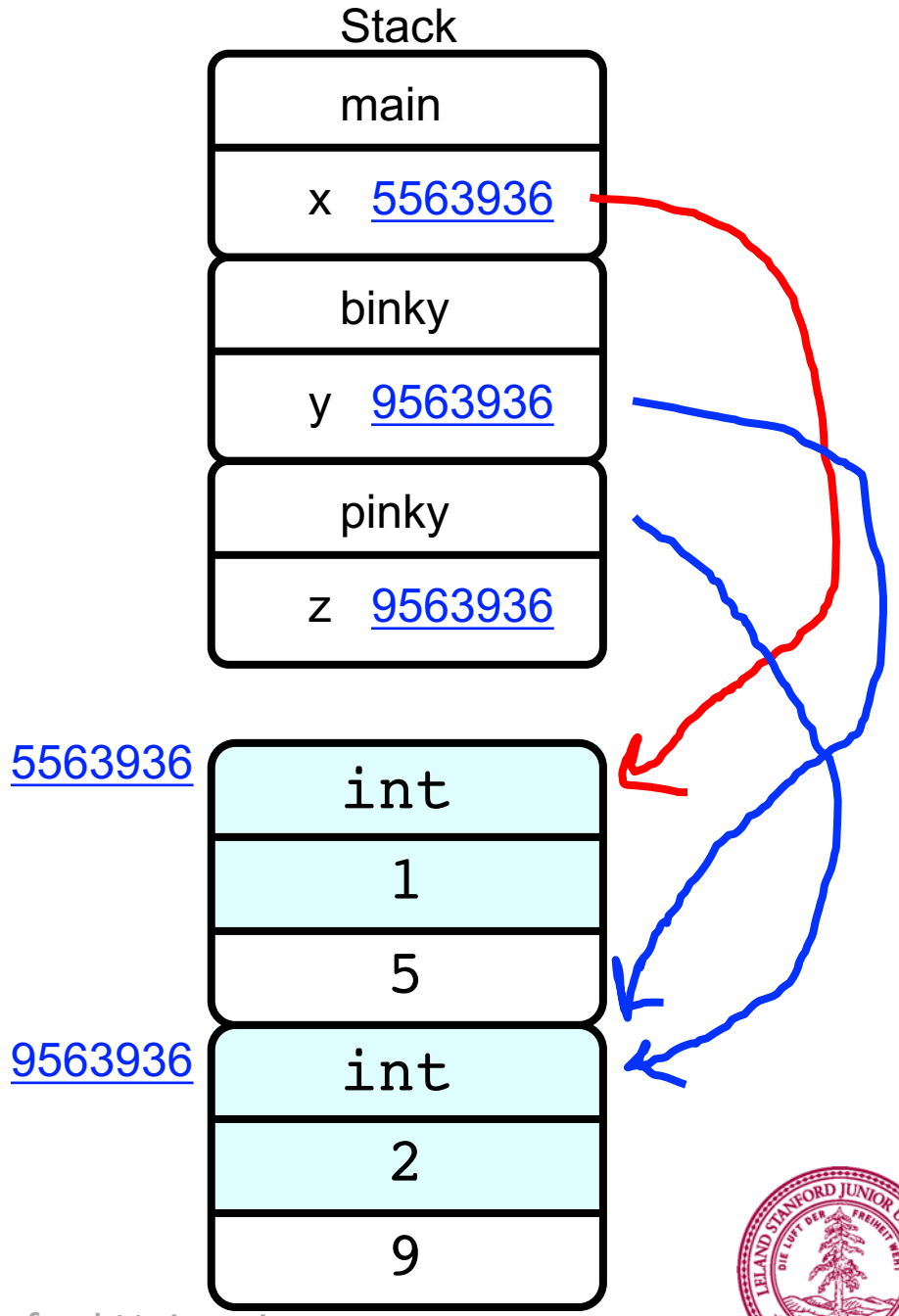
What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



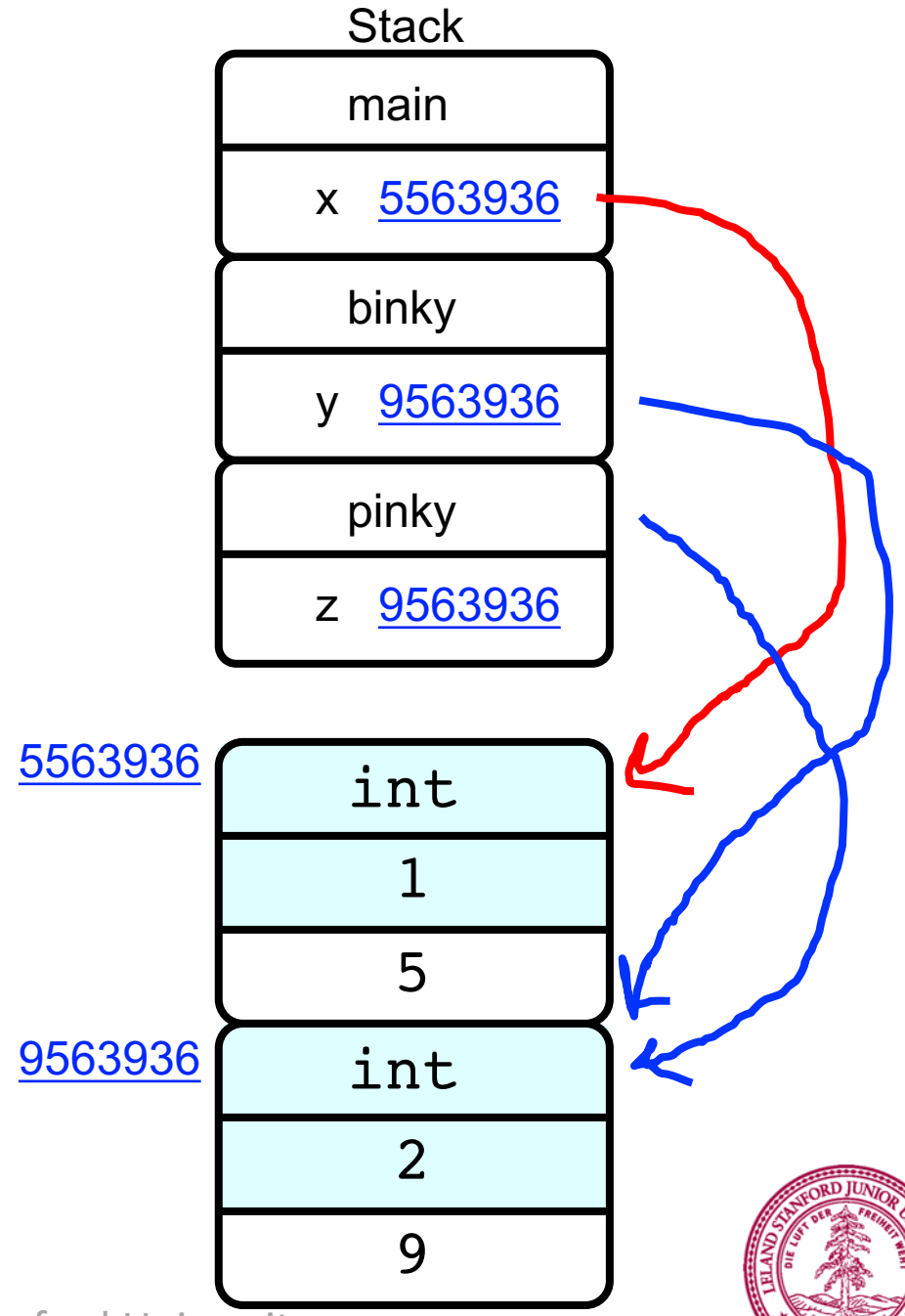
What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



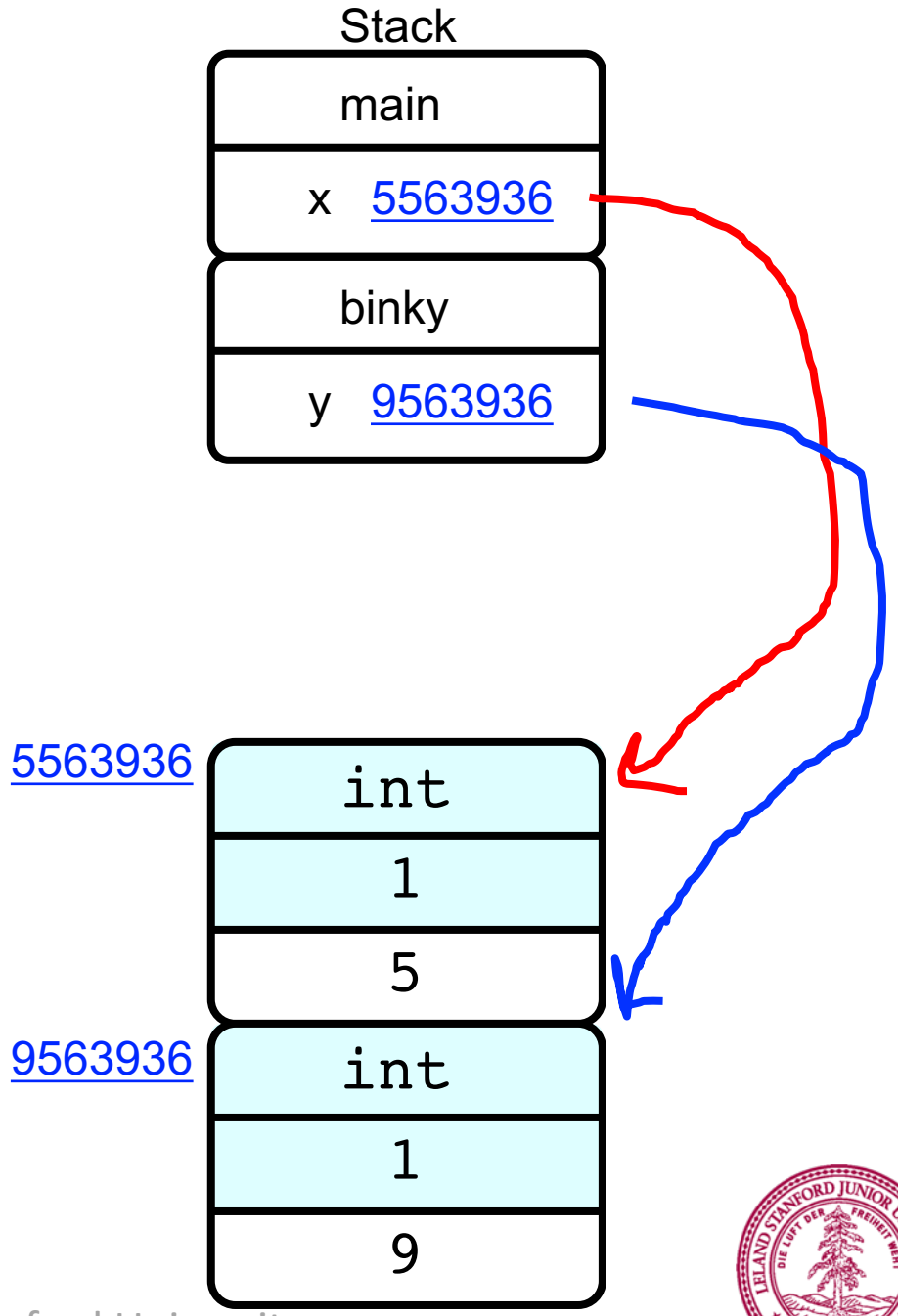
What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



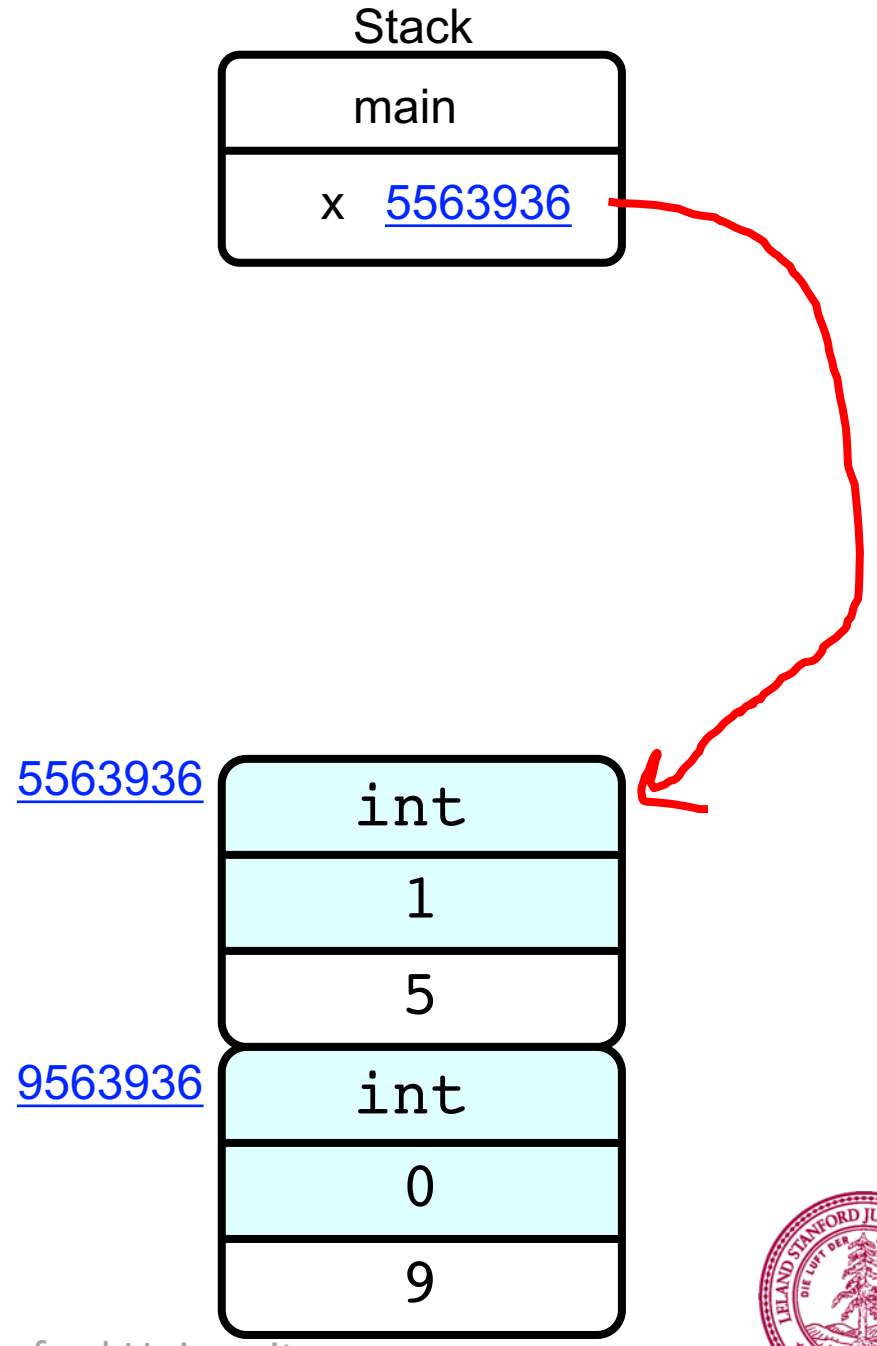
What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



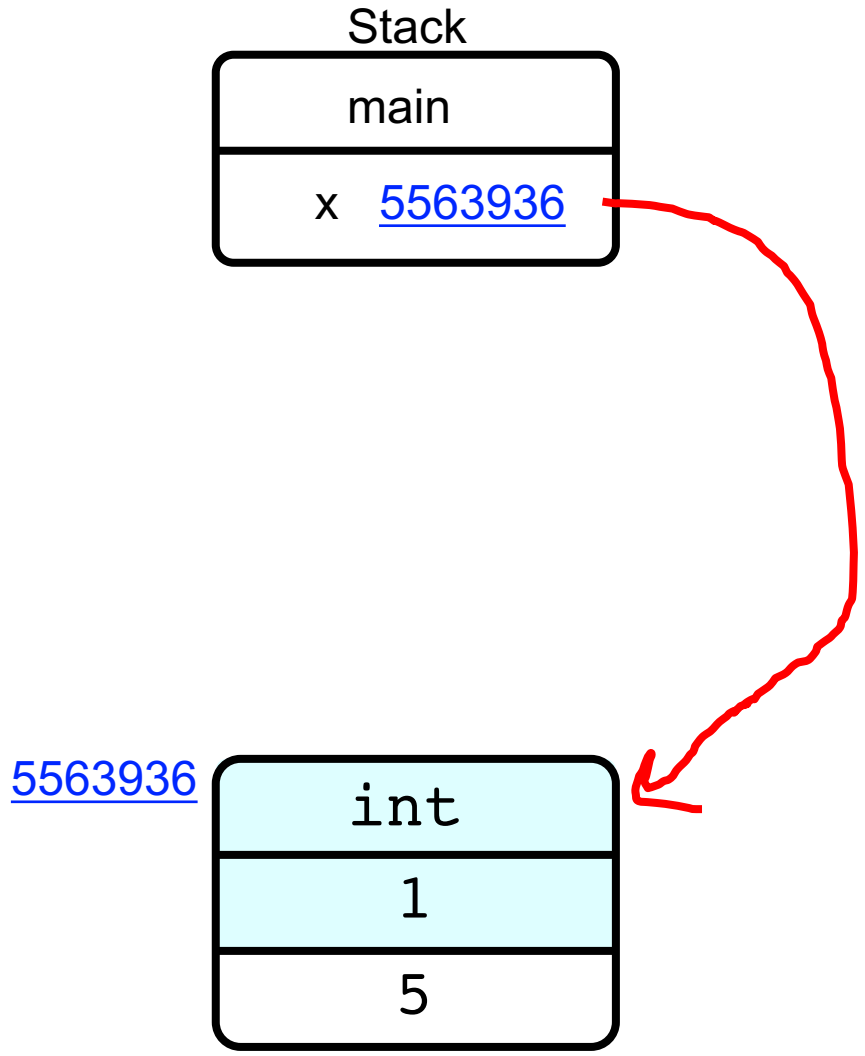
What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```



What does this do?

```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

def pinky(z):
    print(z)
```


This is the real matrix…

The matrix origins

http://www.pythontutor.com/visualize.html

```
def main():
  x = ['a', 'b', 'c']
  update(x)

def update(x):
  for v in x:
    print(type(v), v)
    v = v + '!'
    print(v)

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

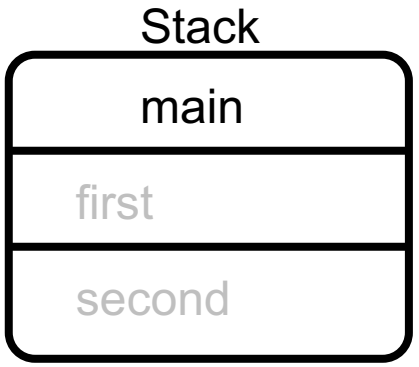

What is `self`?

What does this do?

```
class Dog:
  def __init__(self, name):

    self.name = new_name
    print(self.name)

# put in another file...
def main():
  first = Dog('simba')

  print(type(first))
  print(id(first))
  print(first.__dict__)
```

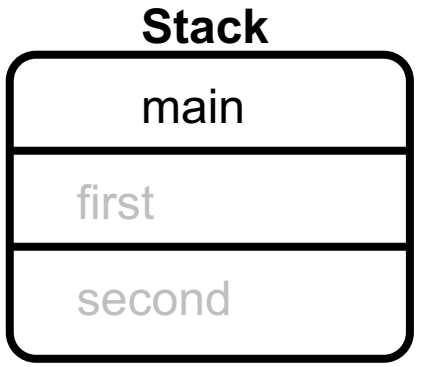

What does this do?

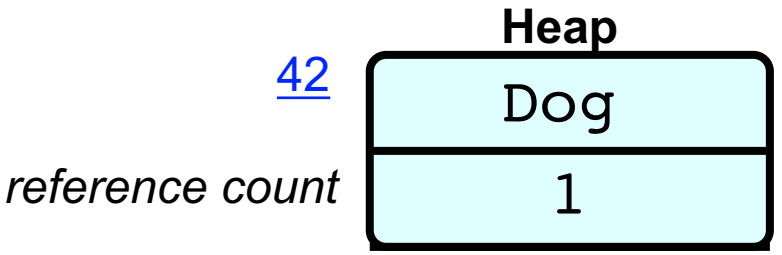
```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



What does this do?

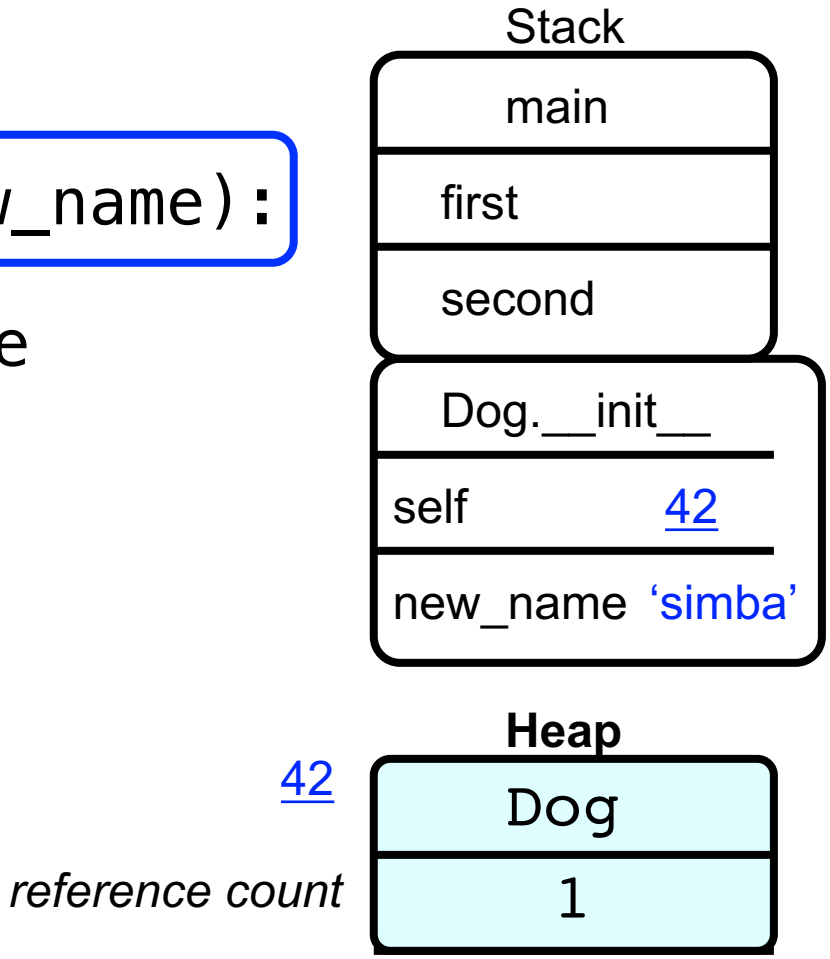
```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```





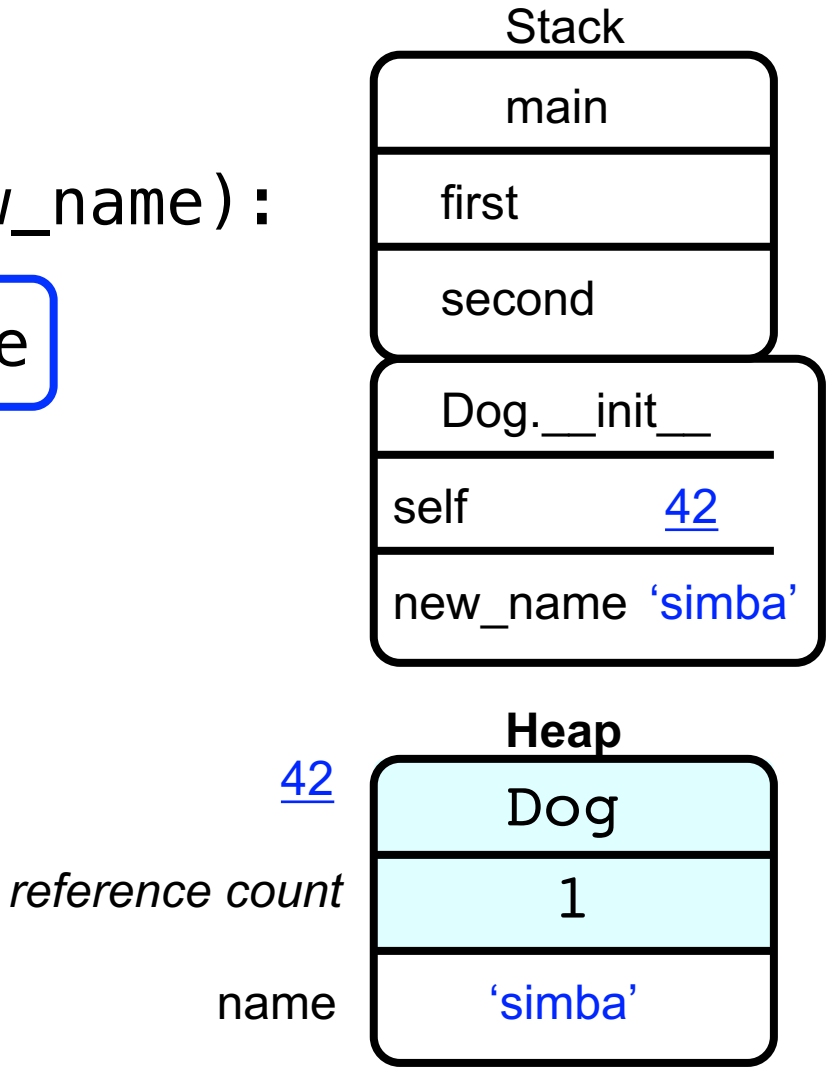
What does this do?

```
class Dog:
    def __init__(self, new_name):
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



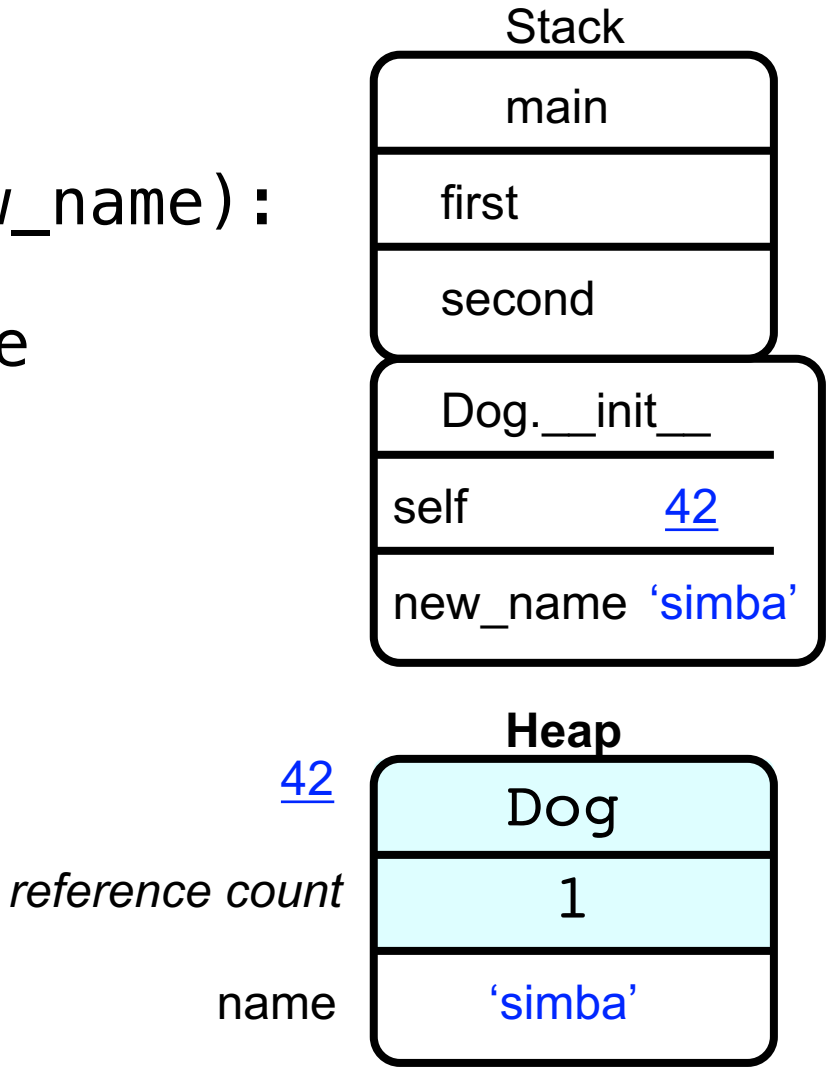
What does this do?

```
class Dog:
    def __init__(self, new_name):
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



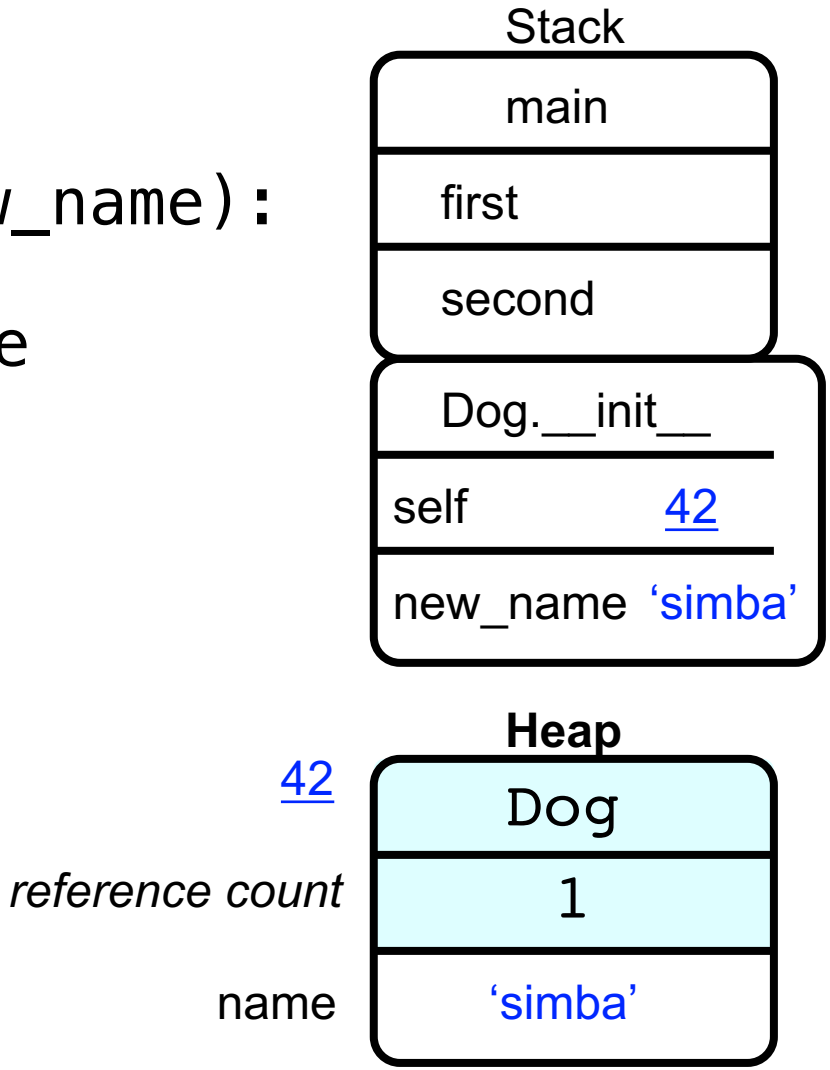
What does this do?

```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



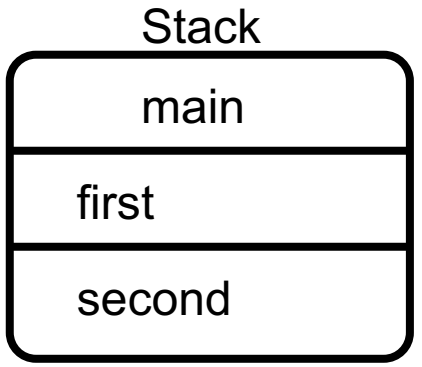
What does this do?

```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```

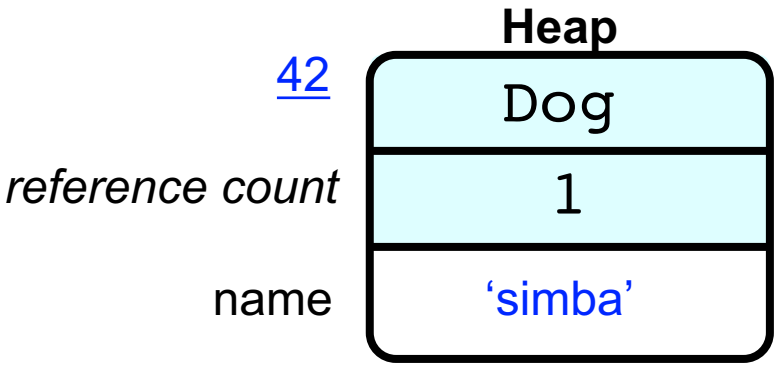


What does this do?

```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```

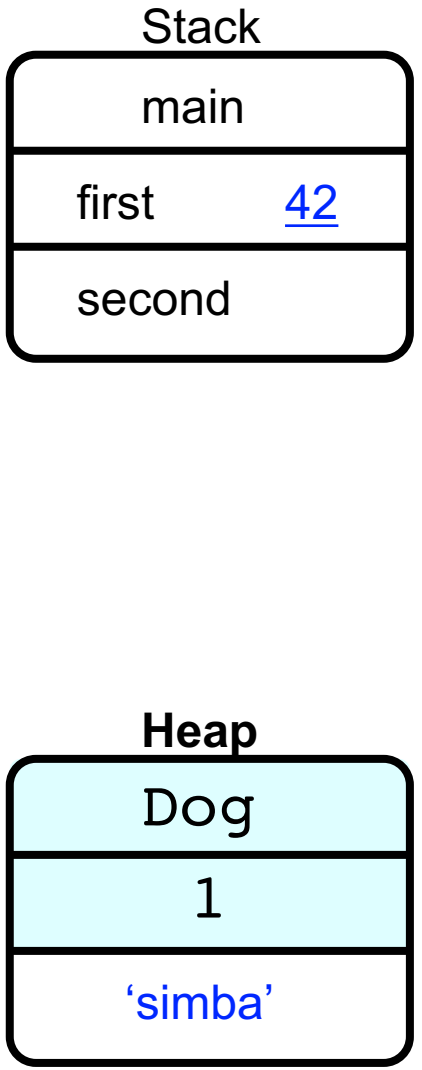


42



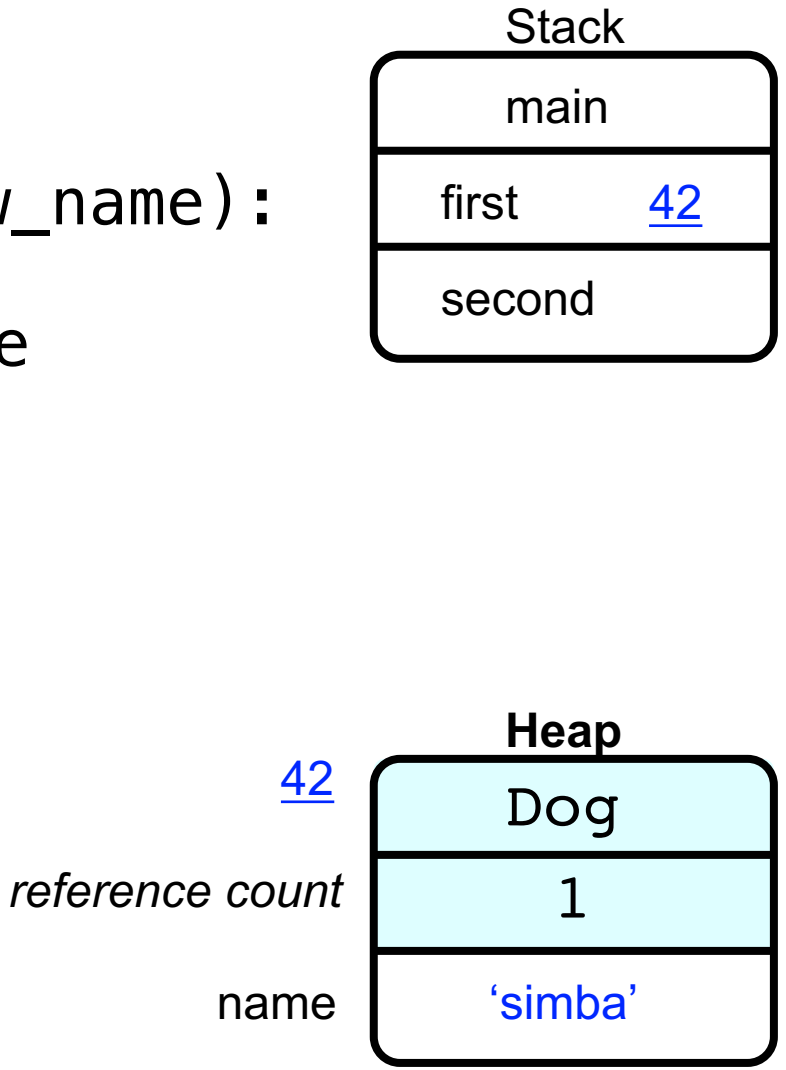
What does this do?

```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



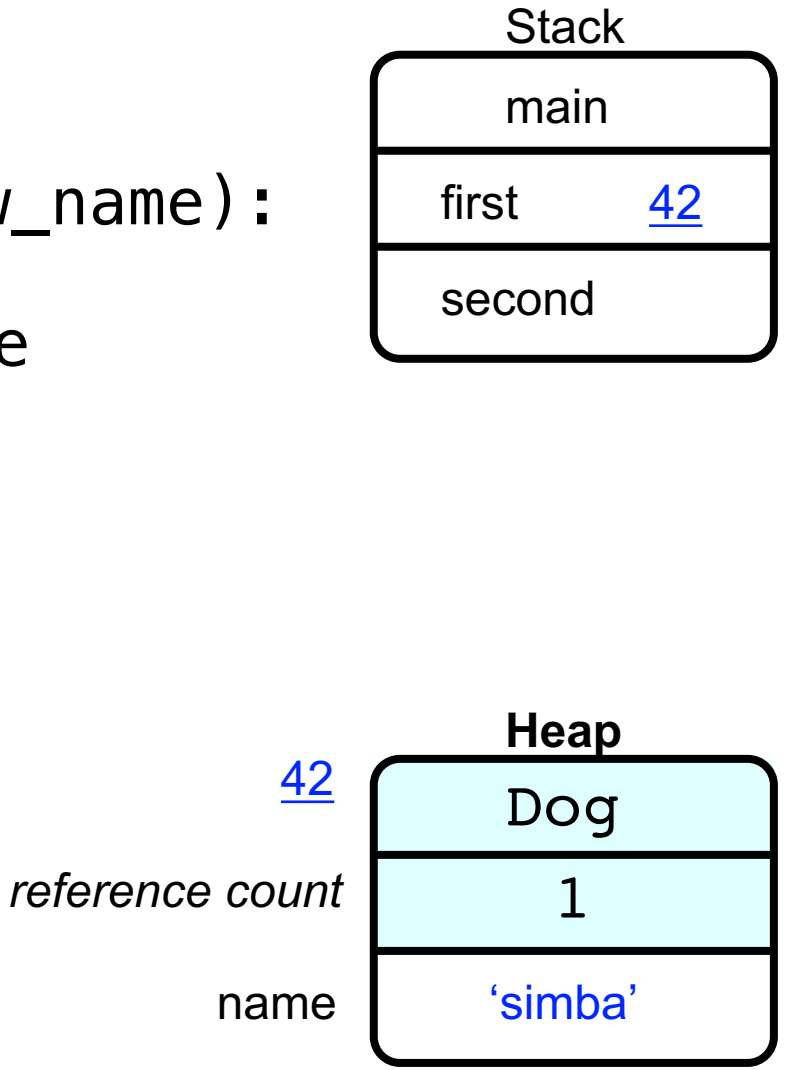
What does this do?

```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



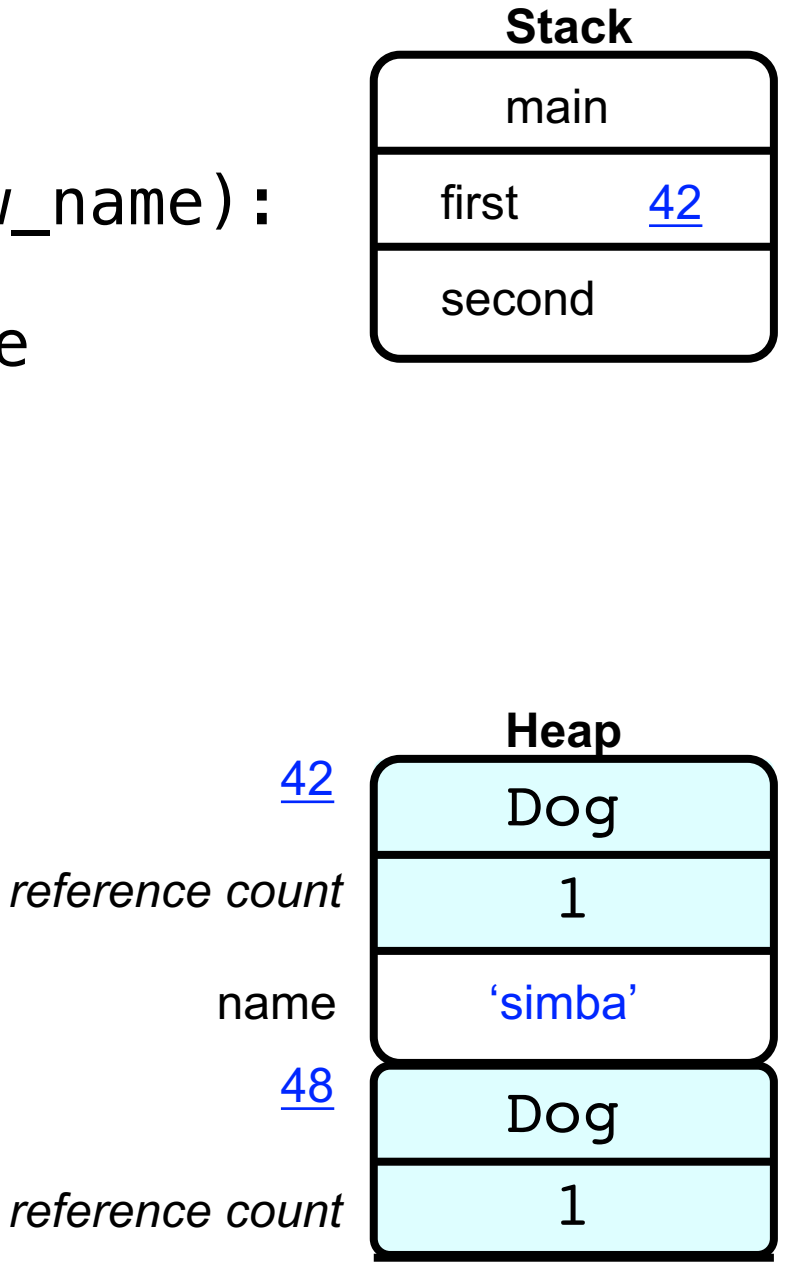
What does this do?

```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



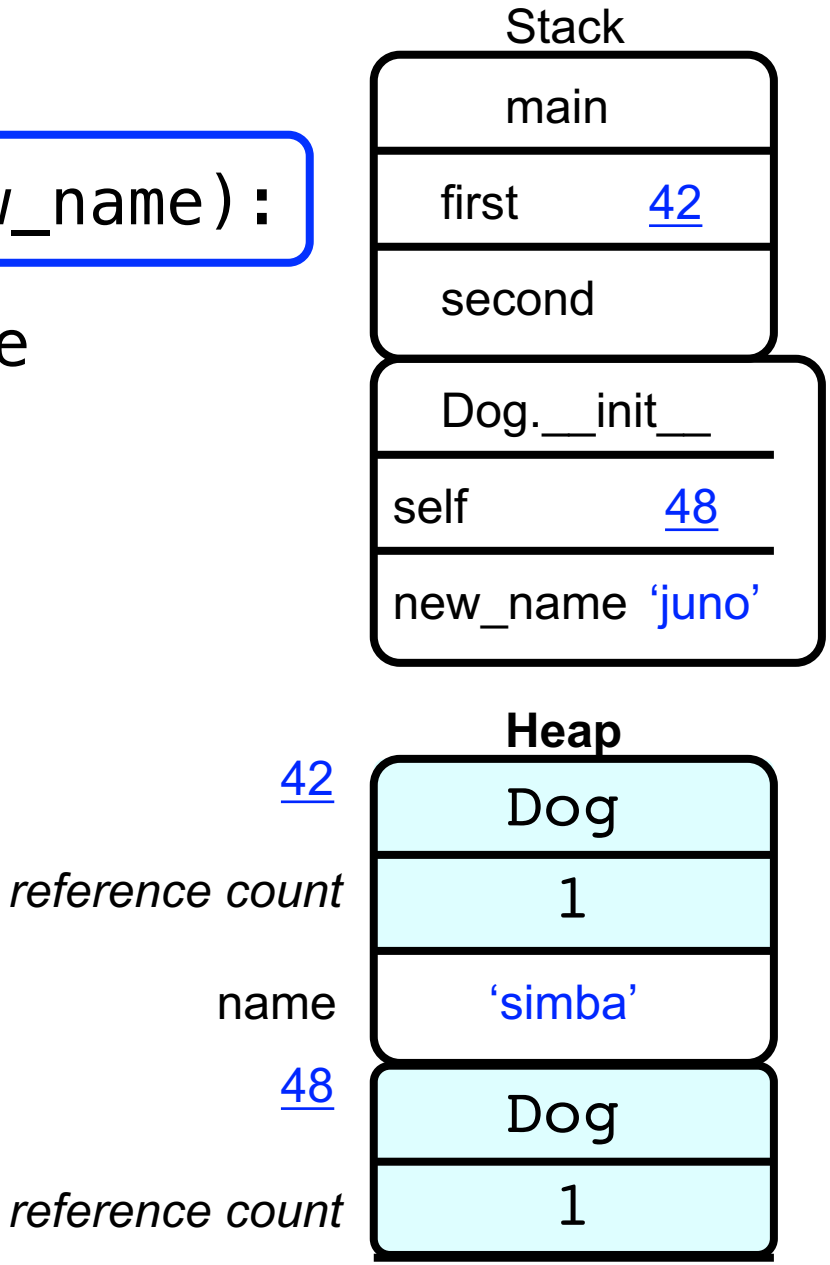
What does this do?

```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



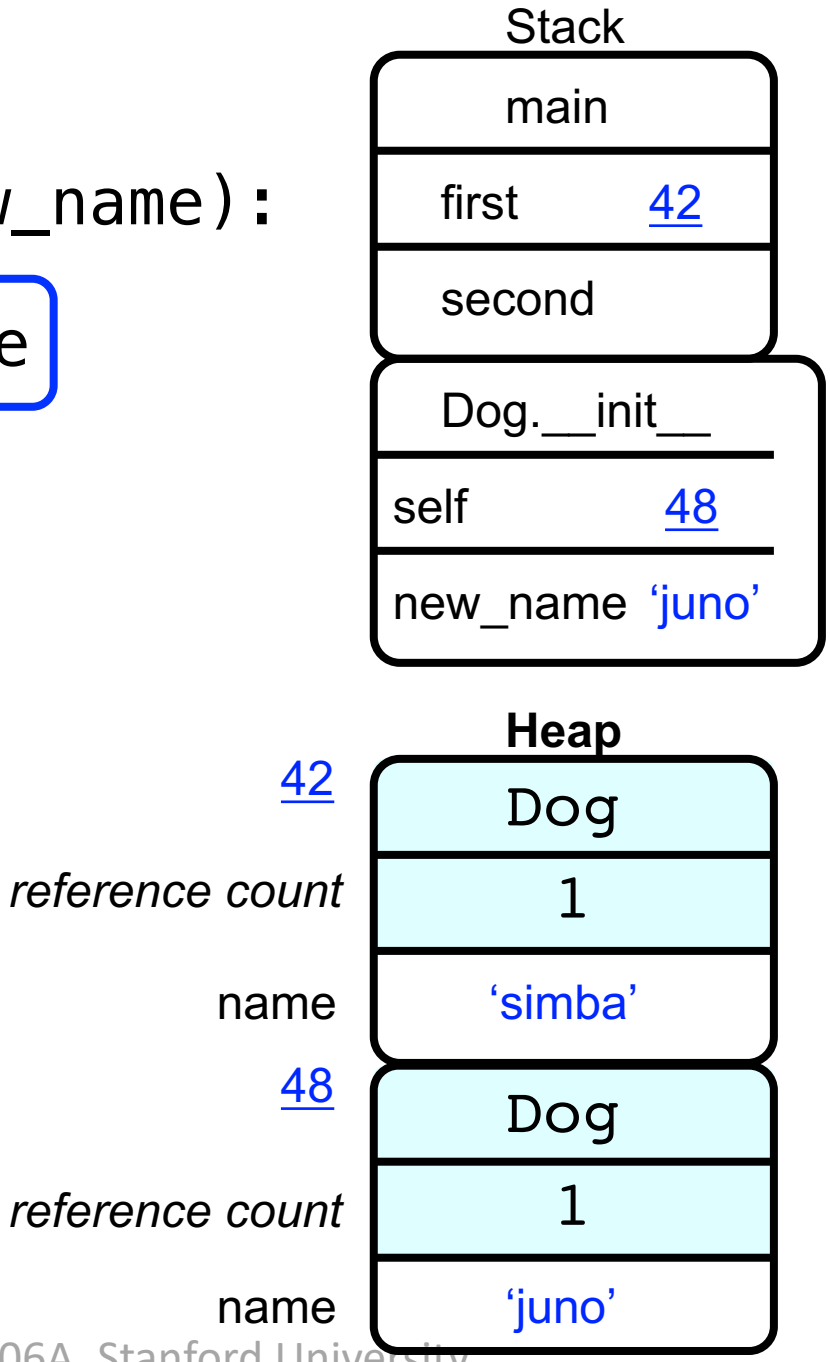
What does this do?

```
class Dog:
    def __init__(self, new_name):
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')

    print(type(first))
    print(id(first))
    print(first.__dict__)
```



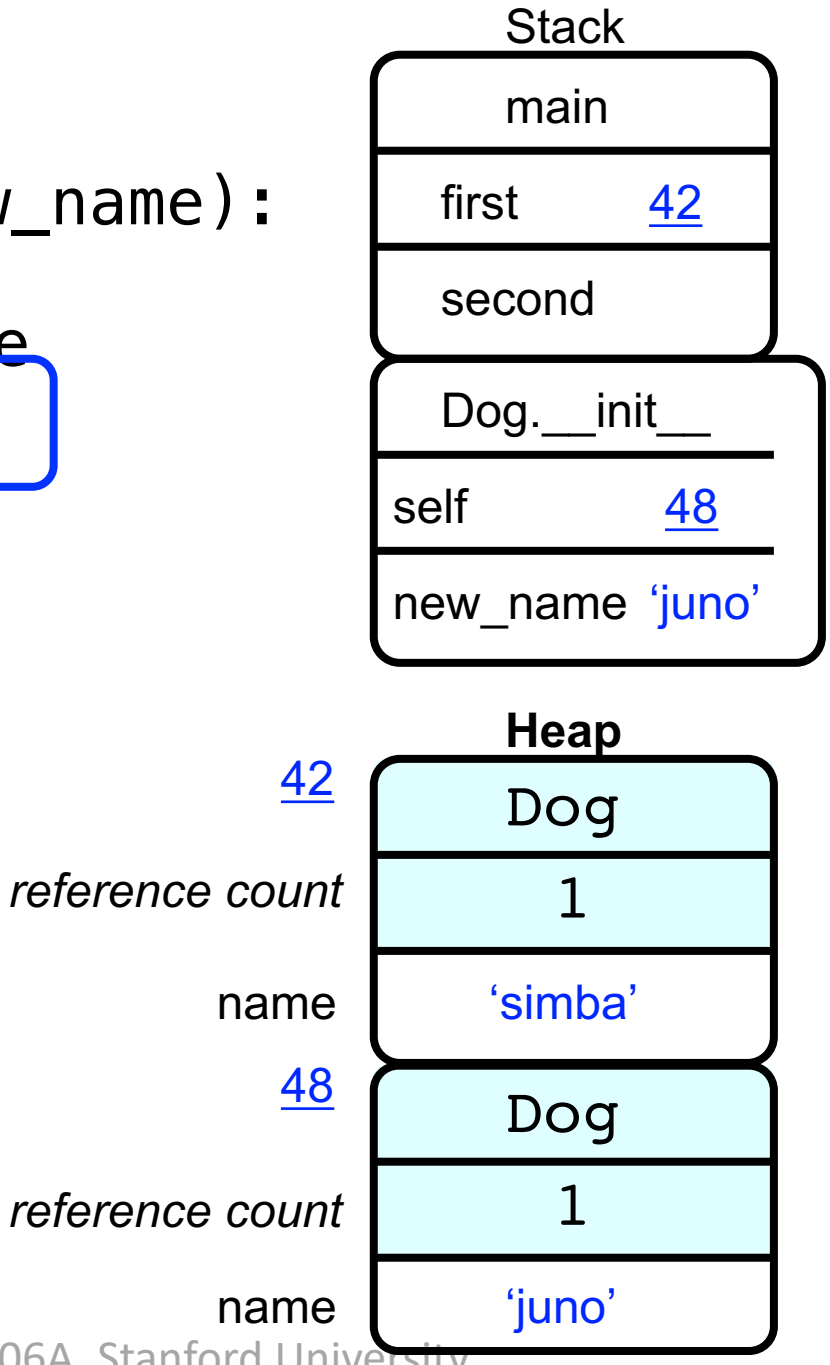
What does this do?

```
class Dog:
    def __init__(self, new_name):
        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



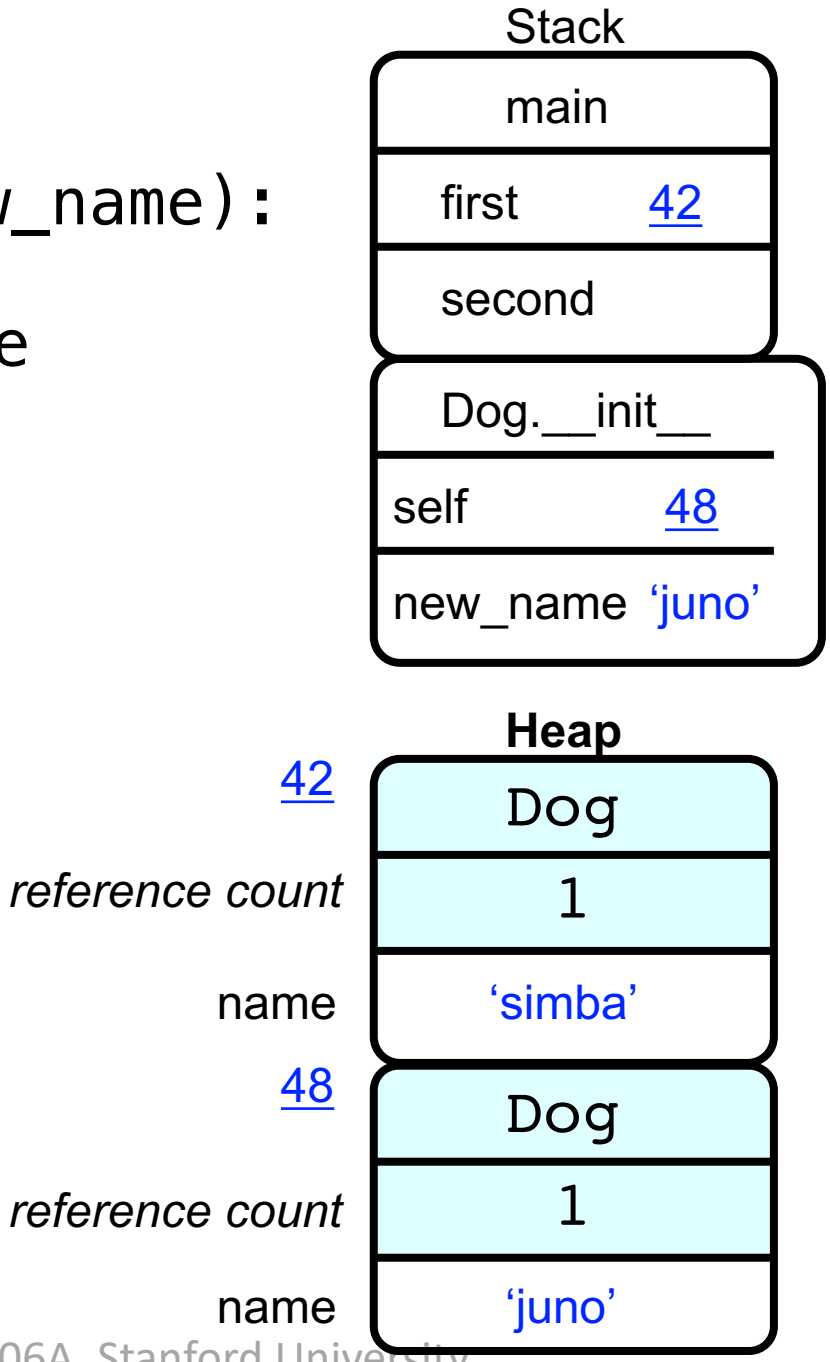
What does this do?

```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



What does this do?

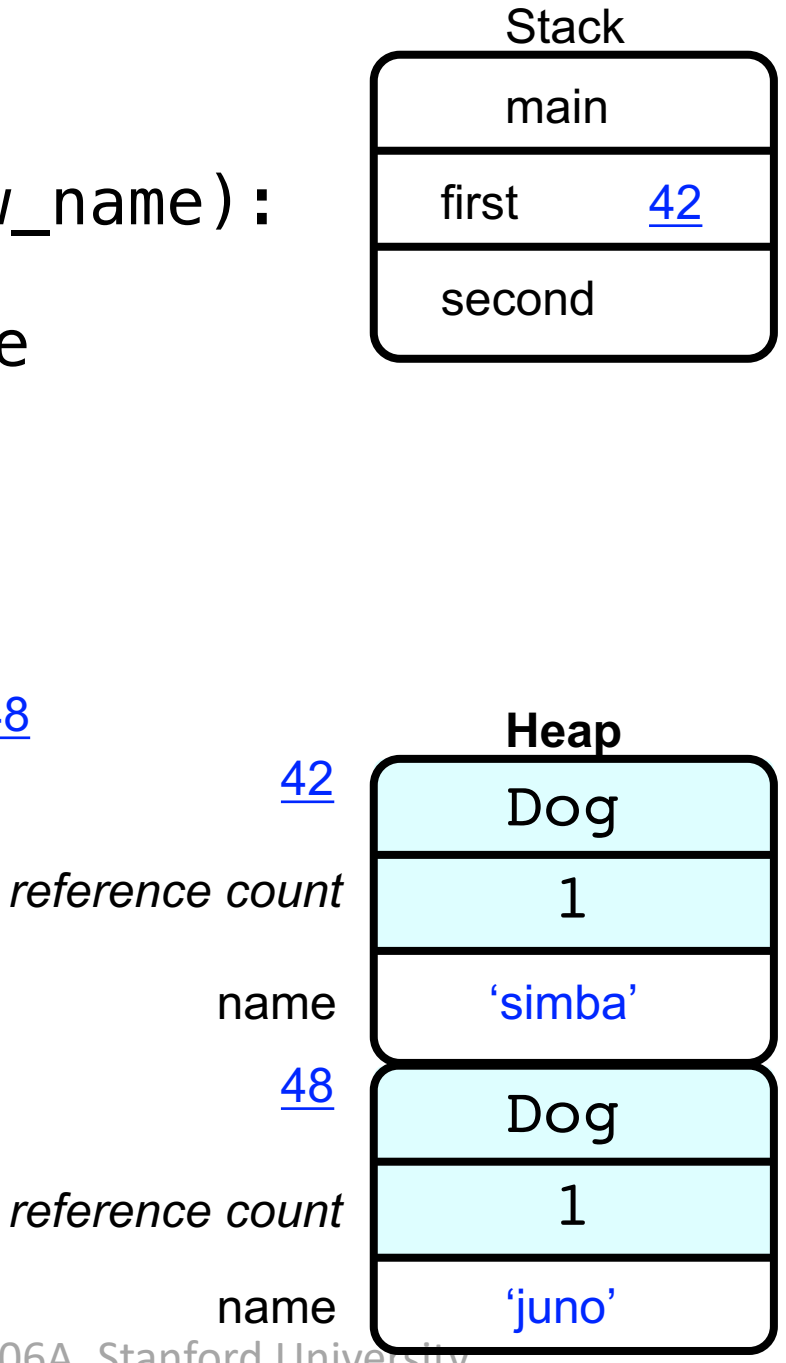
```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



What does this do?

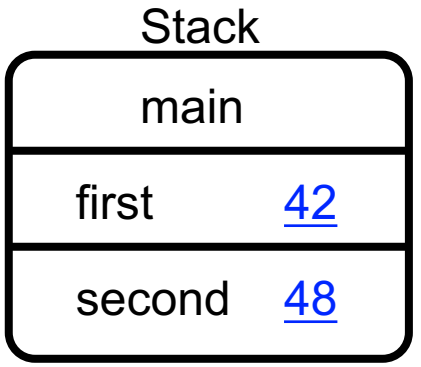
```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```

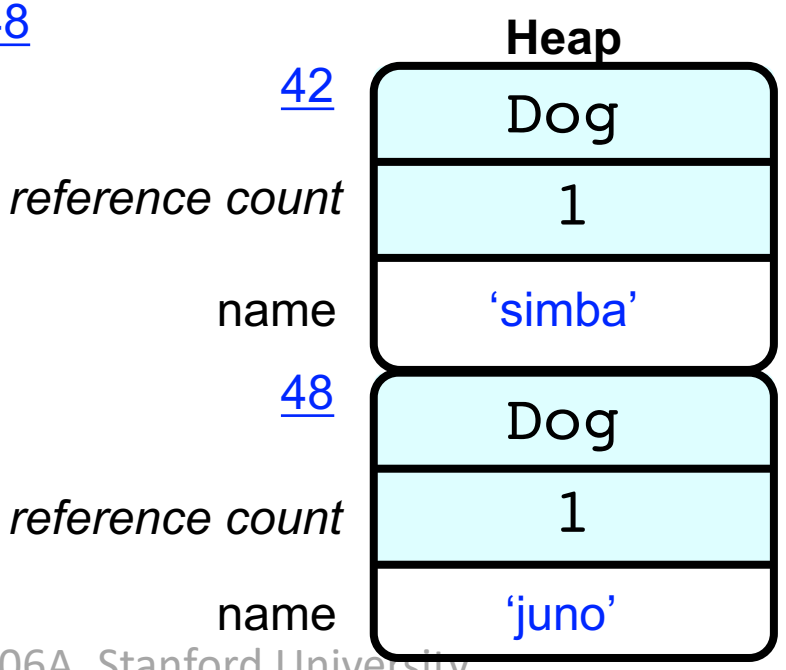
48



What does this do?

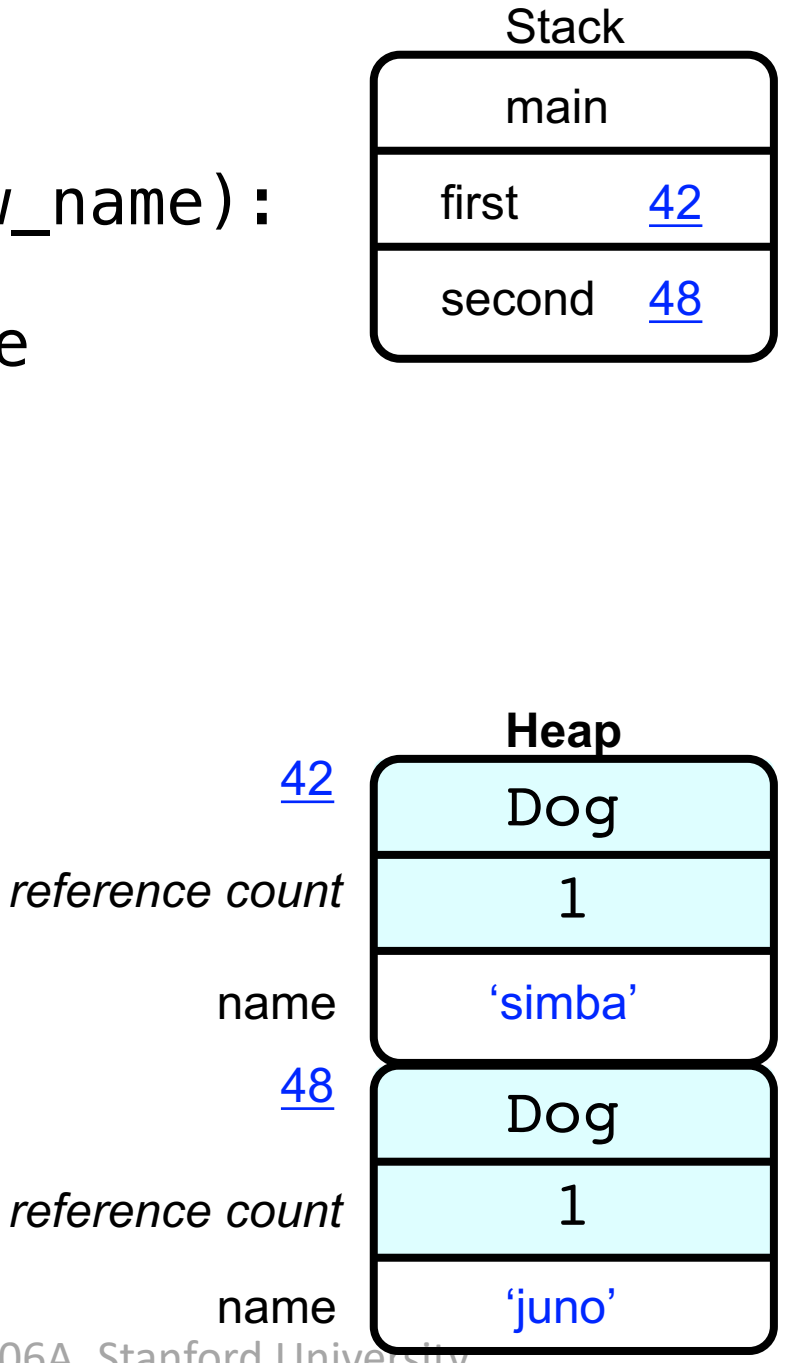
```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```





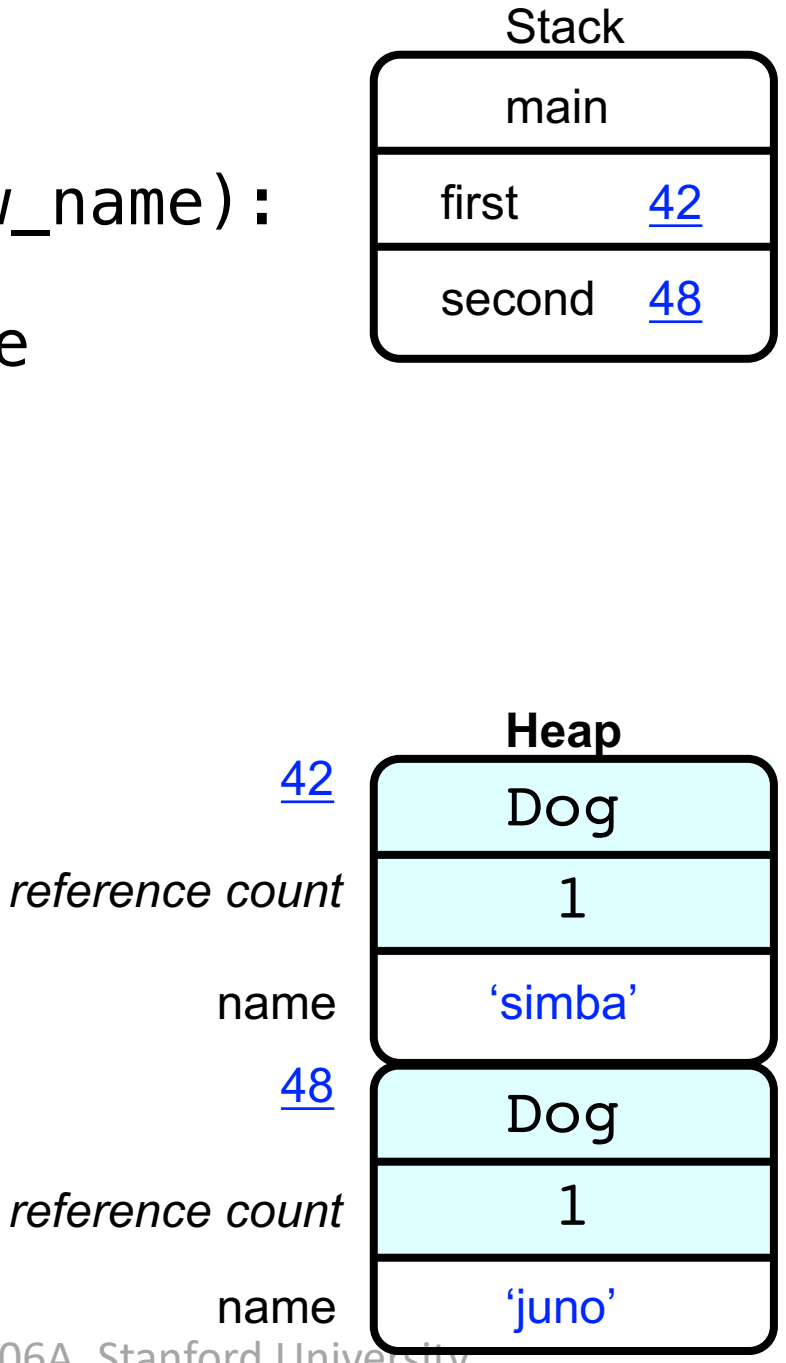
What does this do?

```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



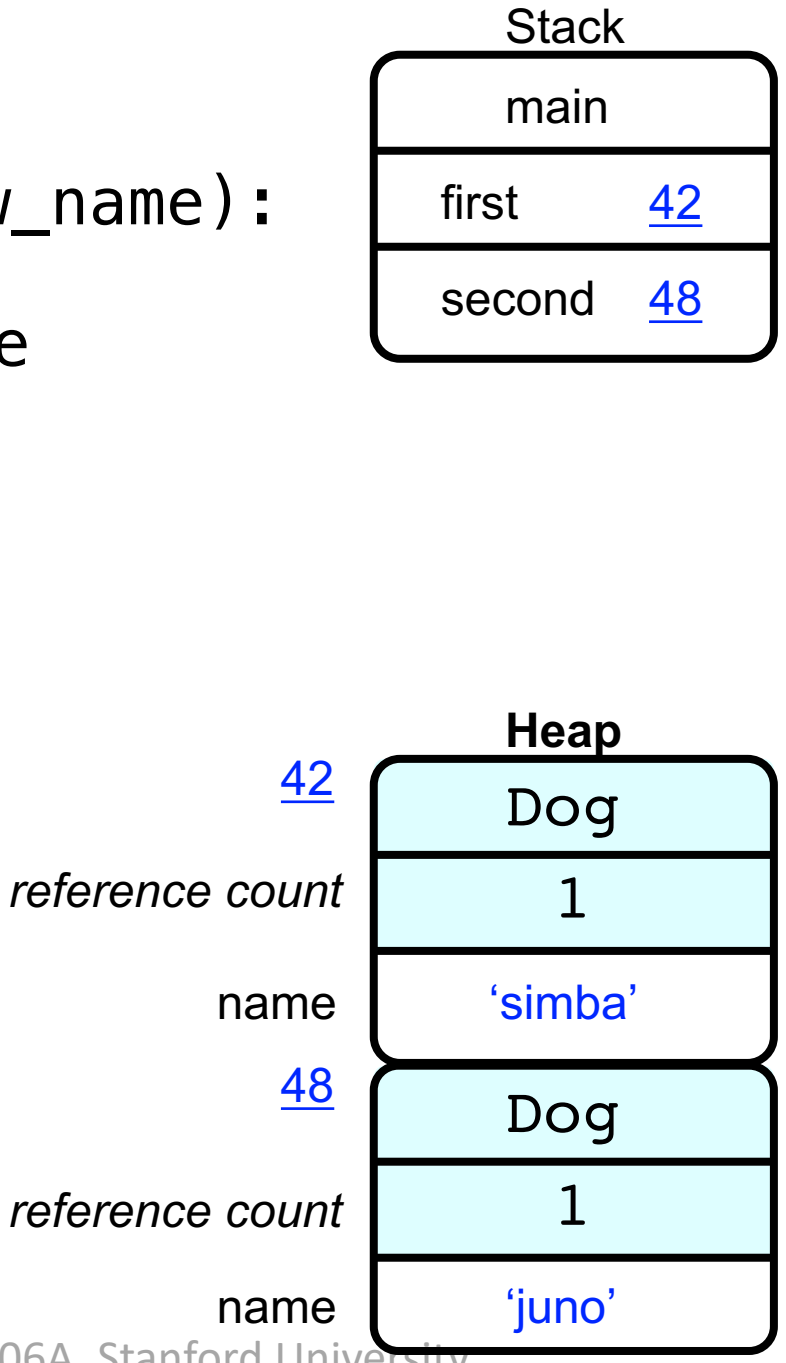
What does this do?

```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



What does this do?

```
class Dog:
    def __init__(self, new_name):

        self.name = new_name
        print(self.name)

# put in another file...
def main():
    first = Dog('simba')
    second = Dog('juno')


    print(type(first))
    print(id(first))
    print(first.__dict__)
```



Challenge: Trace This!

dog.py

```
class Dog:
    def __init__(self):
        self.times_barked = 0

    def bark(self):
        print('woof')
        self.times_barked += 1
```

life.py

```
def main():
    simba = Dog()
    juno = Dog()

    simba.bark()
    juno.bark()
    simba.bark()

    print(simba.__dict__)
    print(juno.__dict__)
```

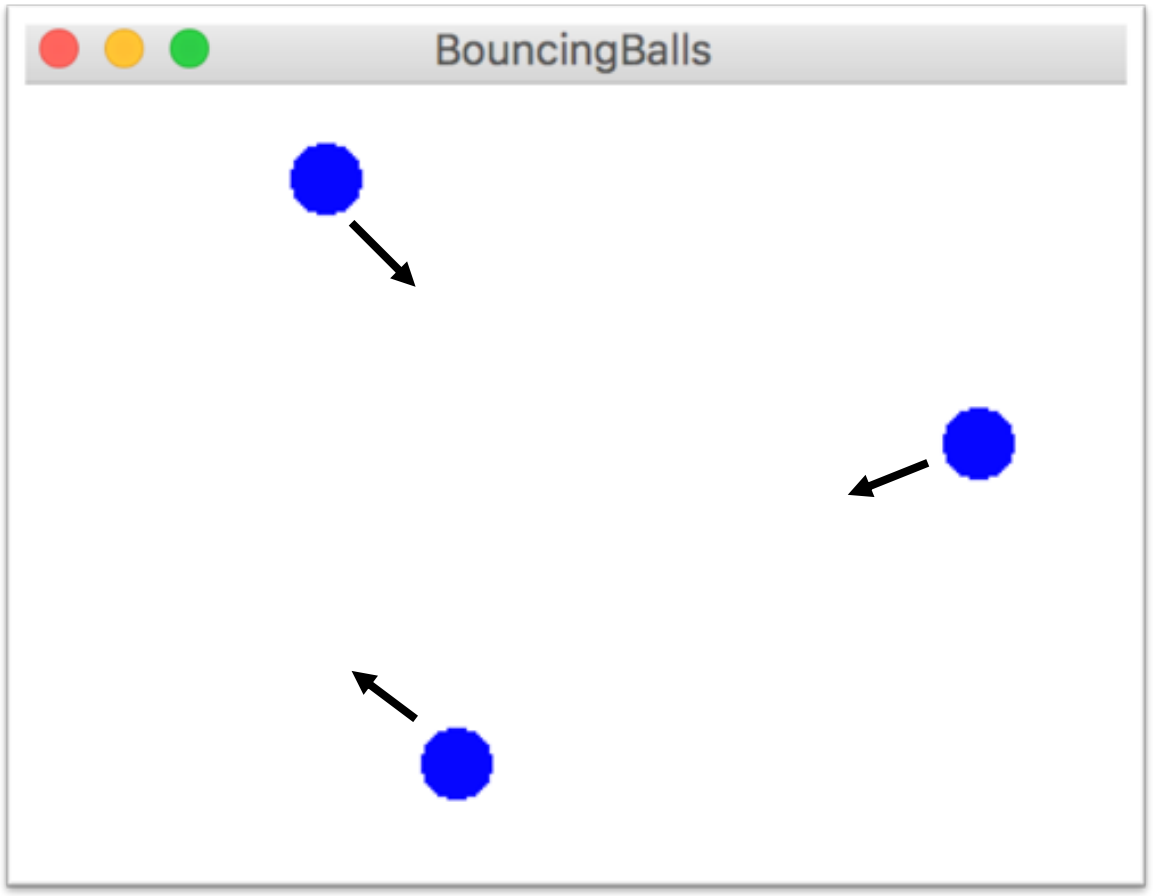
Learning Goals

1. Practice with classes
2. See how to trace memory with classes

Guiding question for today:

what does it take to go from
what you know to writing
big-scale software?

Bouncing Balls



What does a `class` do?

A class defines a new variable type