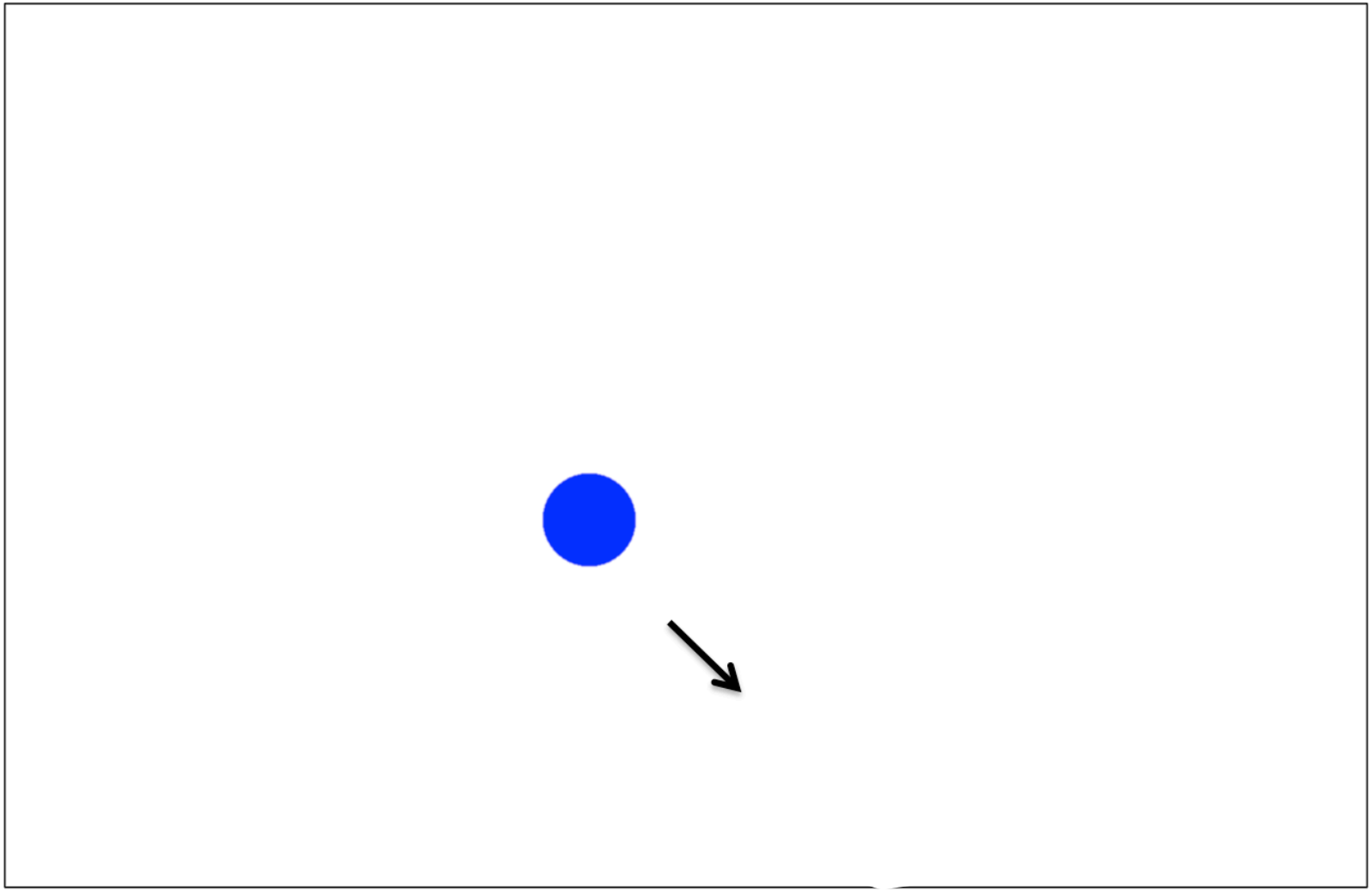
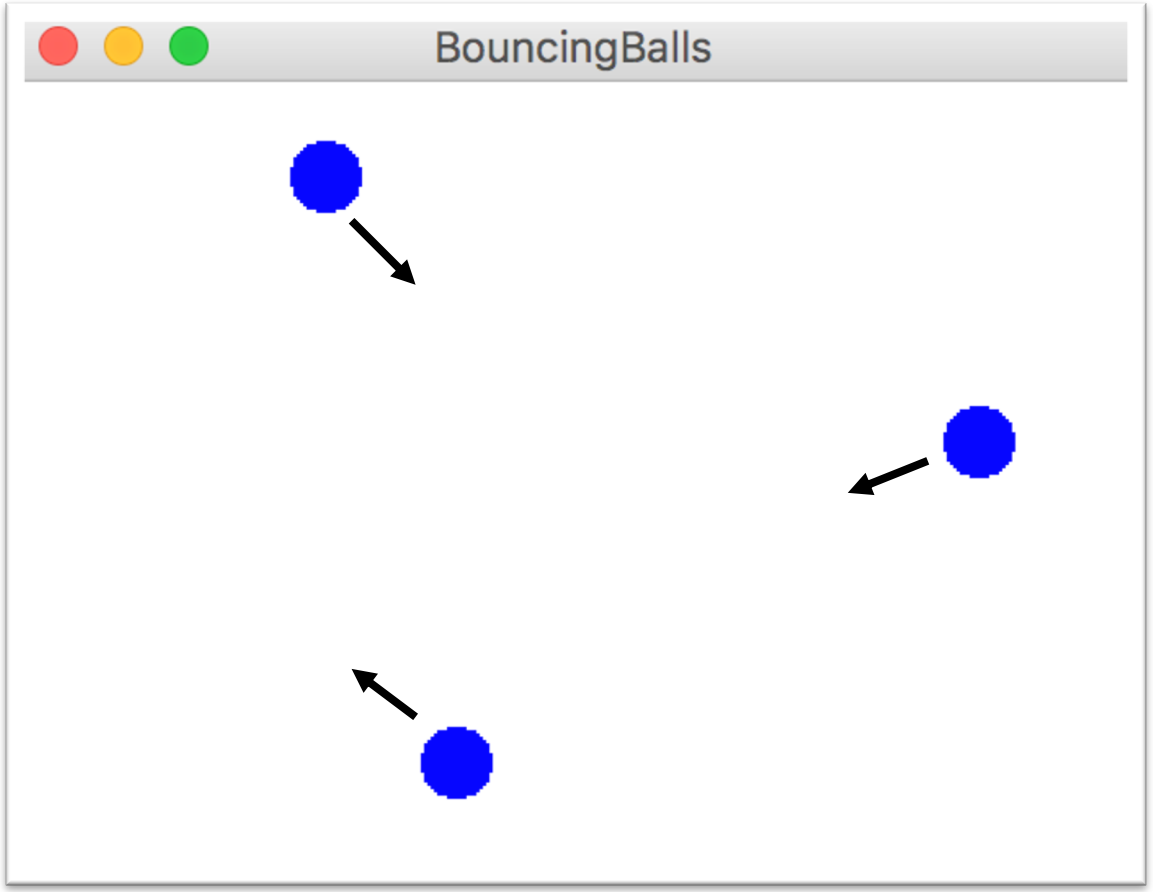
Classes + Memory

Chris Piech and Mehran Sahami
CS106A, Stanford University

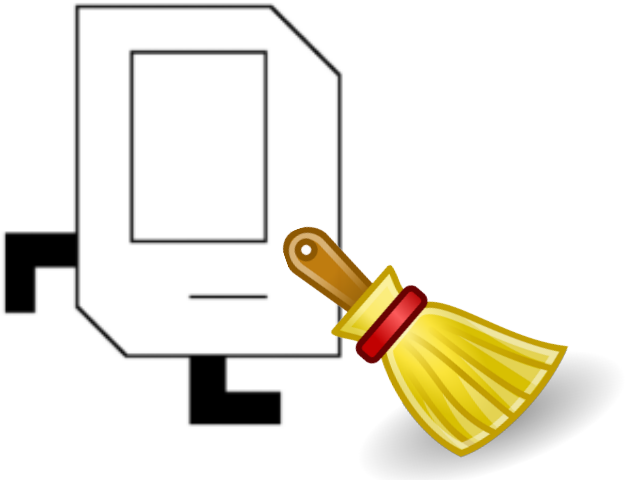
Remember this?



Bouncing Balls



Housekeeping



- Chris OH changed one time only. This week at 11a

Learning Goals

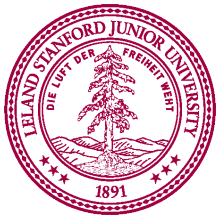
1. Practice with classes
2. See how to trace memory with classes

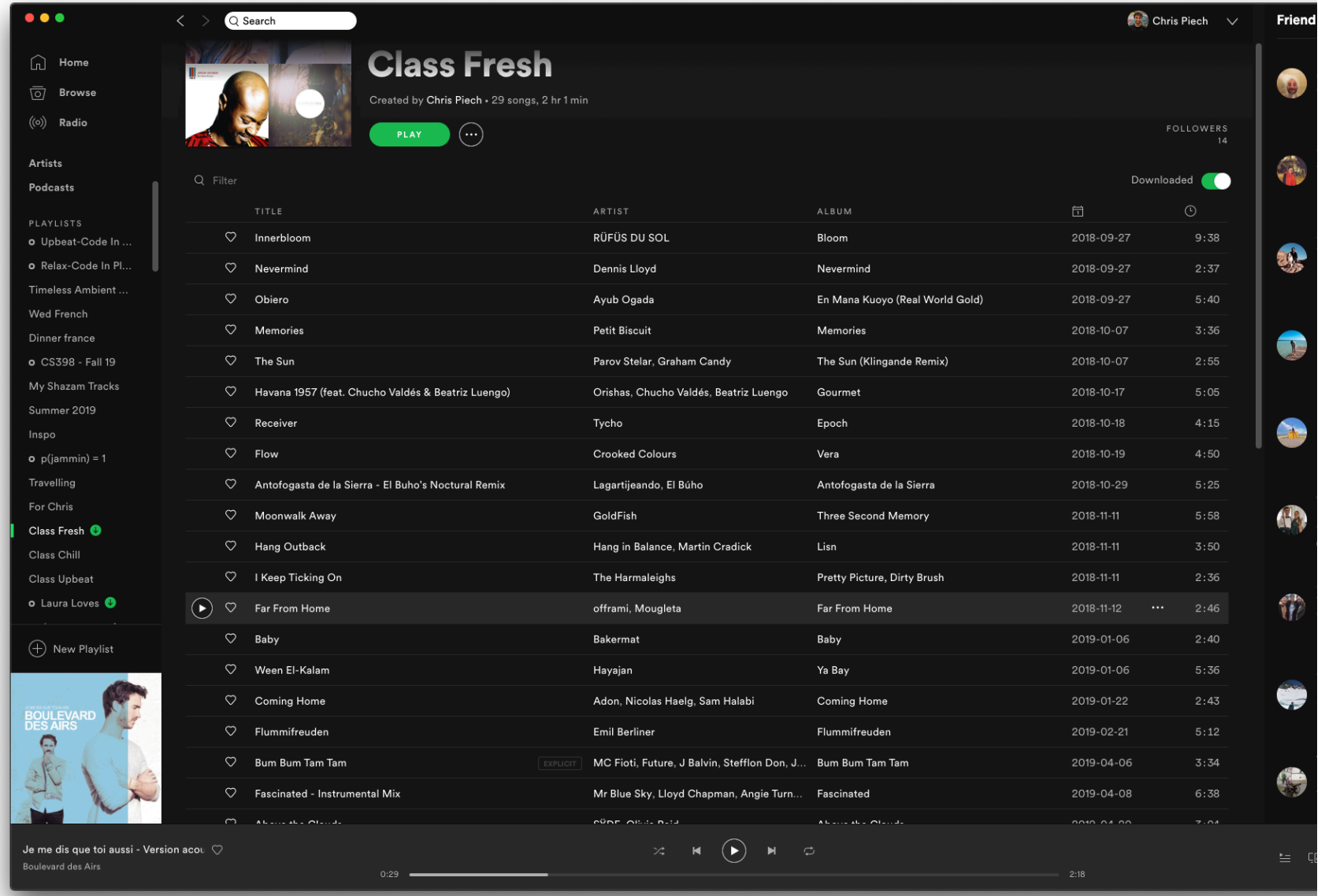
Guiding question for today:

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


Search
Chris Piech
Friend
Home
Browse
Radio
Artists
Podcasts
PLAYLISTS
Upbeat-Code In ...
Relax-Code In Pl...
Timeless Ambient ...
Wed French
Dinner france
CS398 - Fall 19
My Shazam Tracks
Summer 2019
Inspo
p(jammin) = 1
Travelling
For Chris
Class Fresh
Class Chill
Class Upbeat
Laura Loves
New Playlist
Class Fresh
Created by Chris Piech • 29 songs, 2 hr 1 min
PLAY
FOLLOWERS
14
Filter
Downloaded
TITLE
ARTIST
ALBUM
Innerbloom RÜFÜS DU SOL Bloom 2018-09-27 9:38
Nevermind Dennis Lloyd Nevermind 2018-09-27 2:37
Obiero Ayub Ogada En Mana Kuoyo (Real World Gold) 2018-09-27 5:40
Memories Petit Biscuit Memories 2018-10-07 3:36
The Sun Parov Stelar, Graham Candy The Sun (Klingande Remix) 2018-10-07 2:55
Havana 1957 (feat. Chucho Valdés & Beatriz Luengo) Orishas, Chucho Valdés, Beatriz Luengo Gourmet 2018-10-17 5:05
Receiver Tycho Epoch 2018-10-18 4:15
Flow Crooked Colours Vera 2018-10-19 4:50
Antofogasta de la Sierra - El Buho's Noctural Remix Lagartijeando, El Búho Antofogasta de la Sierra 2018-10-29 5:25
Moonwalk Away GoldFish Three Second Memory 2018-11-11 5:58
Hang Outback Hang in Balance, Martin Cradick Lisn 2018-11-11 3:50
I Keep Ticking On The Harmaleighs Pretty Picture, Dirty Brush 2018-11-11 2:36
Far From Home offrami, Mougleta Far From Home 2018-11-12 2:46
Baby Bakermat Baby 2019-01-06 2:40
Ween El-Kalam Hayajan Ya Bay 2019-01-06 5:36
Coming Home Adon, Nicolas Haelg, Sam Halabi Coming Home 2019-01-22 2:43
Flummifreuden Emil Berliner Flummifreuden 2019-02-21 5:12
Bum Bum Tam Tam EXPLICIT MC Fioti, Future, J Balvin, Stefflon Don, J... Bum Bum Tam Tam 2019-04-06 3:34
Fascinated - Instrumental Mix Mr Blue Sky, Lloyd Chapman, Angie Turn... Fascinated 2019-04-08 6:38
BOULEVARD DES AIRS
Je me dis que toi aussi - Version acou
Boulevard des Airs
0:29
2:18

How?

Define New Variable Types

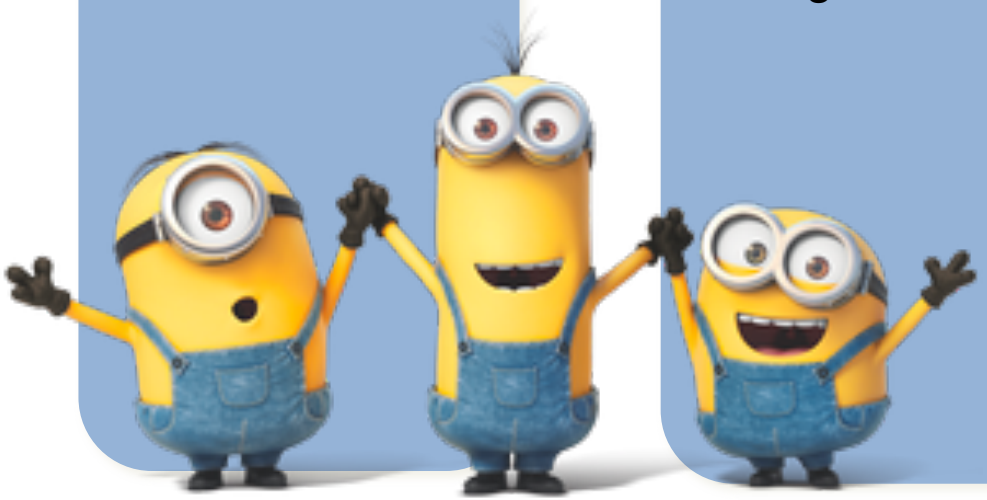
Song

Playlist

User

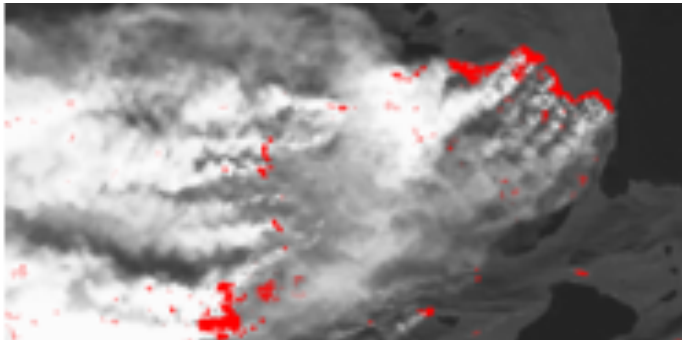
Song Player

Song Retriever

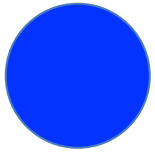


You Have Been *Using* Variable Types

`SimpleImage`



`Canvas`



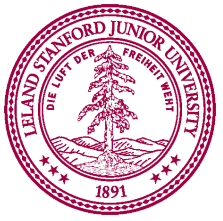
`Karel`

`String`

`int`

What would it take to define your own?

type


LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

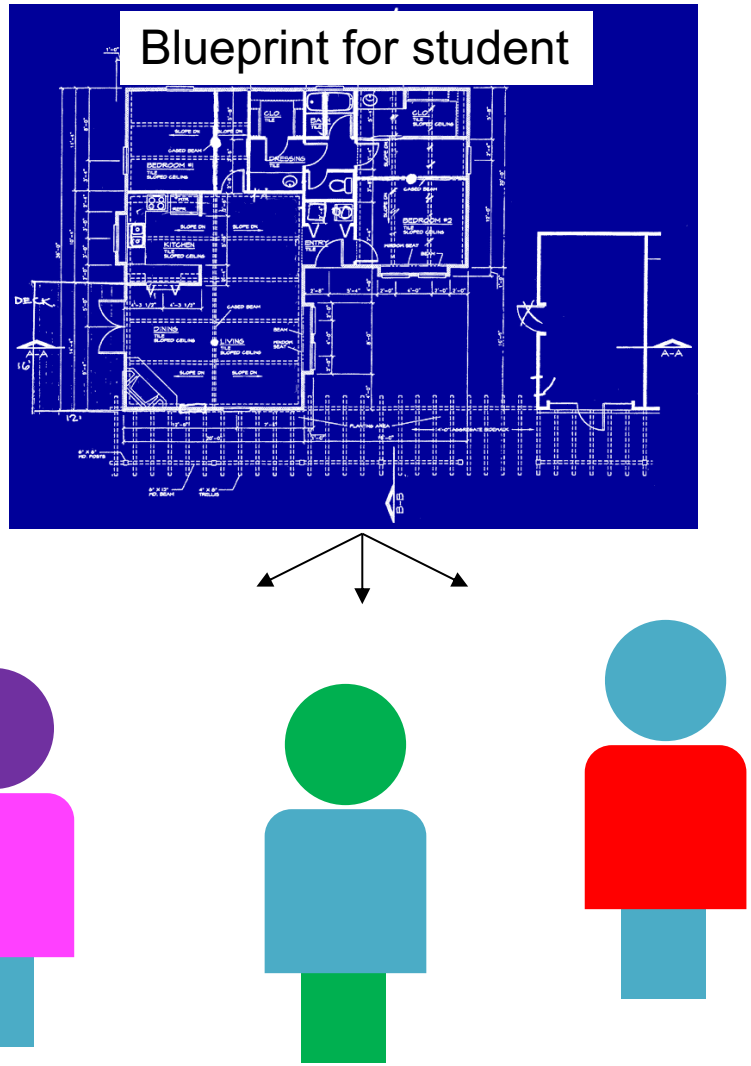
Classes define new variable *types*

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



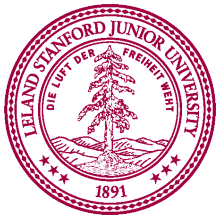
You must define three things

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

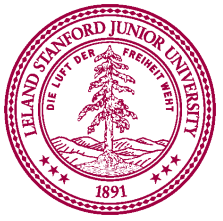
2. What **methods** can you call on an instance?

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

*details on how to define these three things coming soon



.__dict__


LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

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

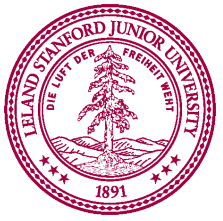
Dog.py

```
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?

Classes Review

Dog.py

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

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

life.py

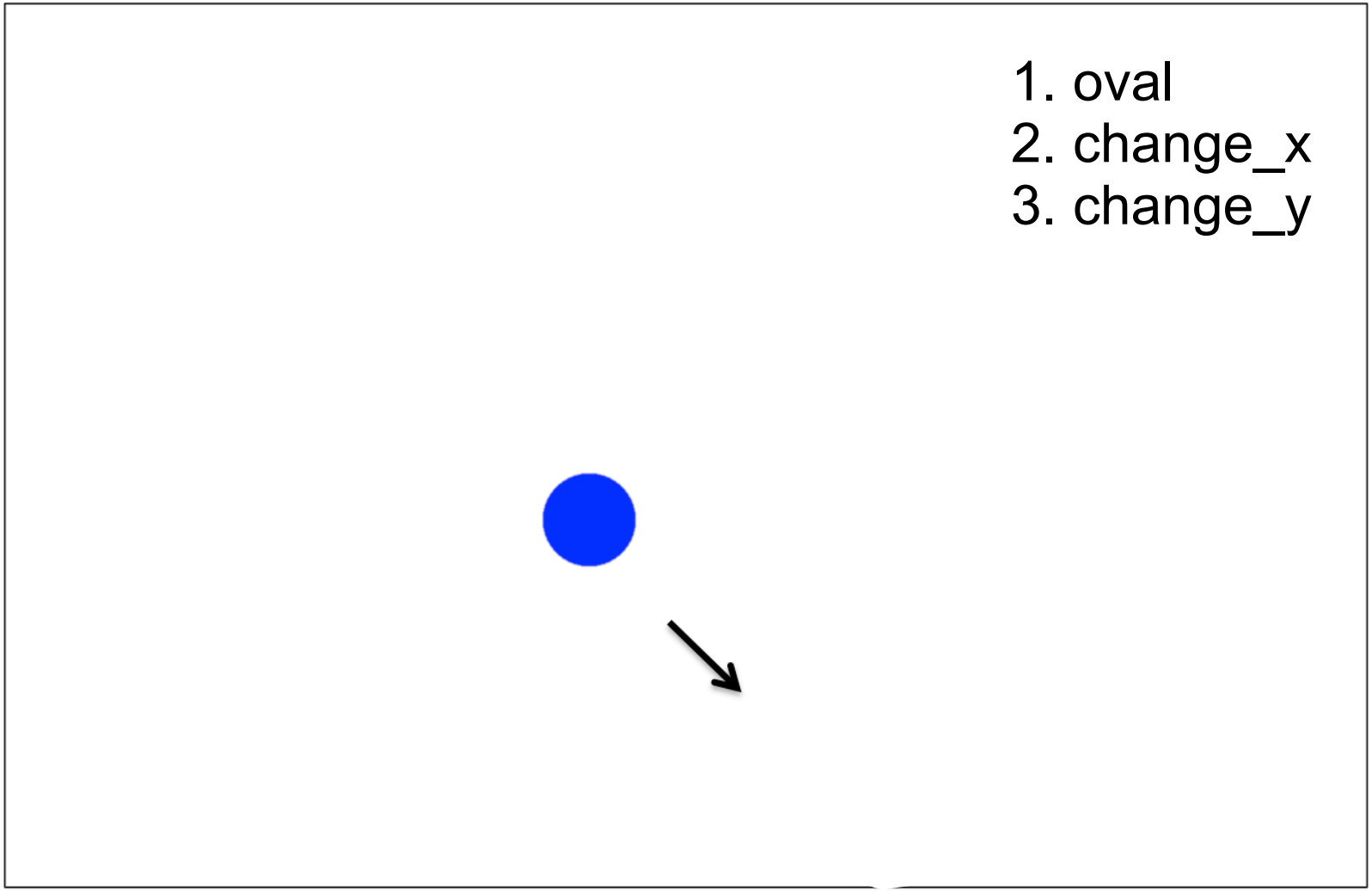
```
def main():
    simba = Dog()
    juno = Dog()

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

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

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

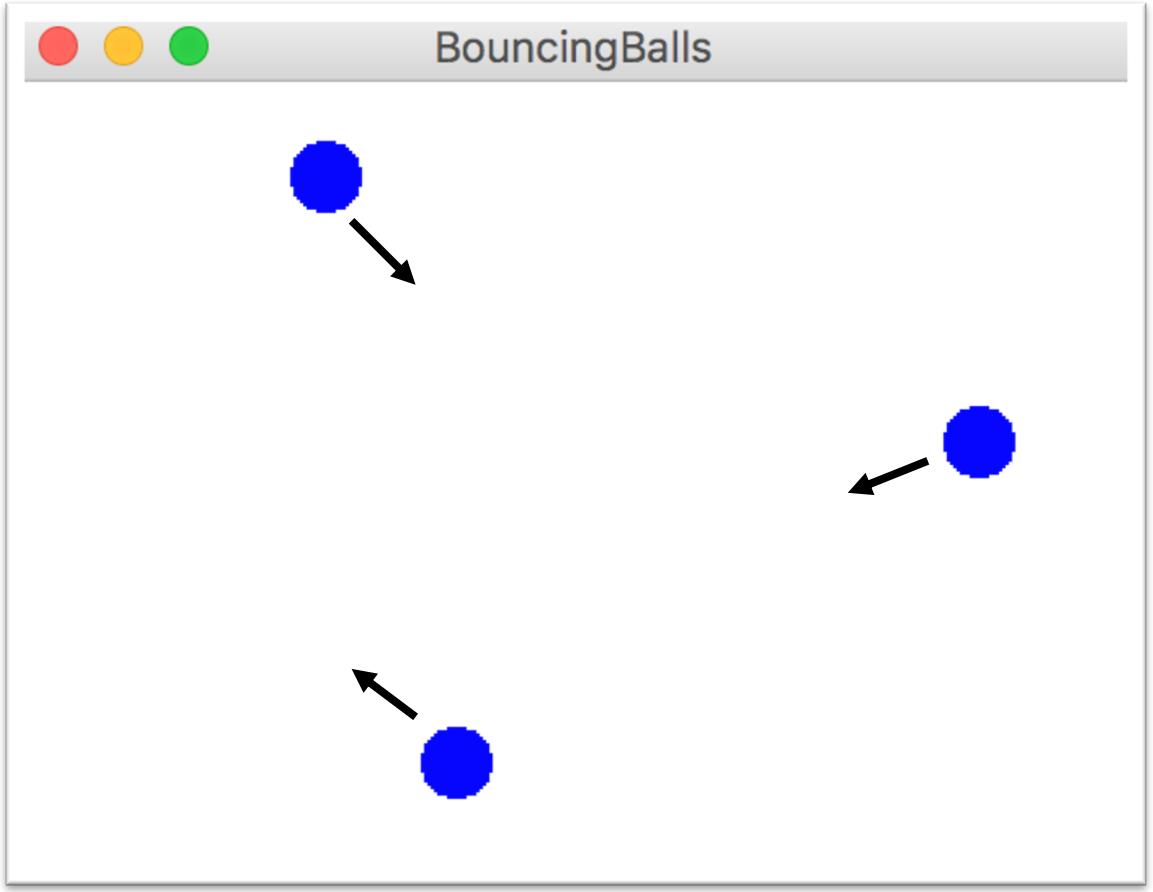
What is a class?

A class defines a new variable type

How many variables for the ball?



Bouncing Balls



1: Store a list of dictionaries

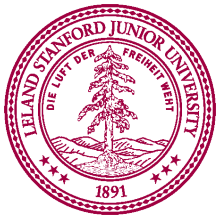
2: Store a list of Balls

Next step in writing large programs:
Better understand memory

You are now ready…

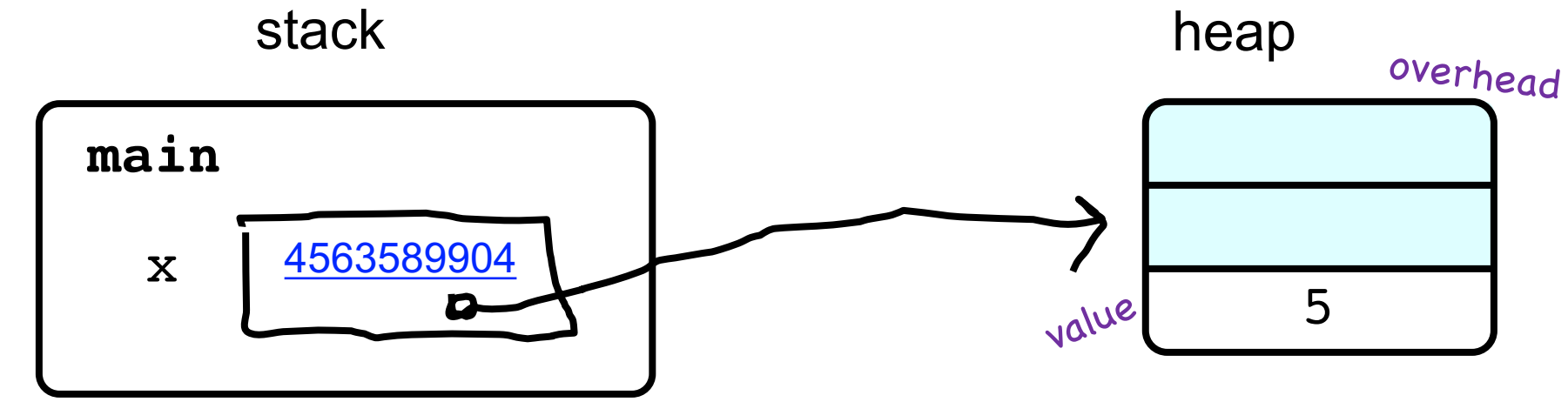
What does this do?

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

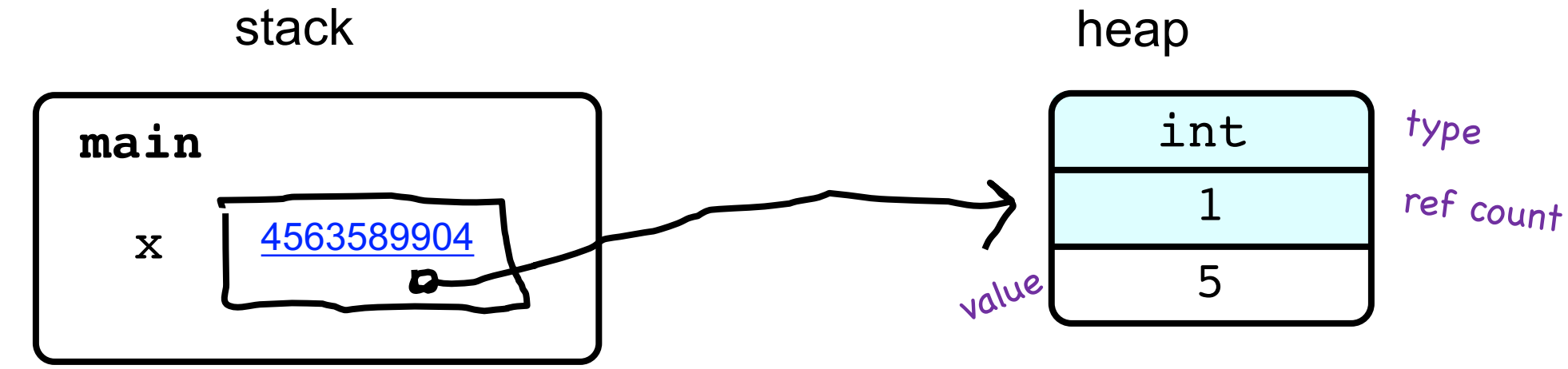
What does this do?

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



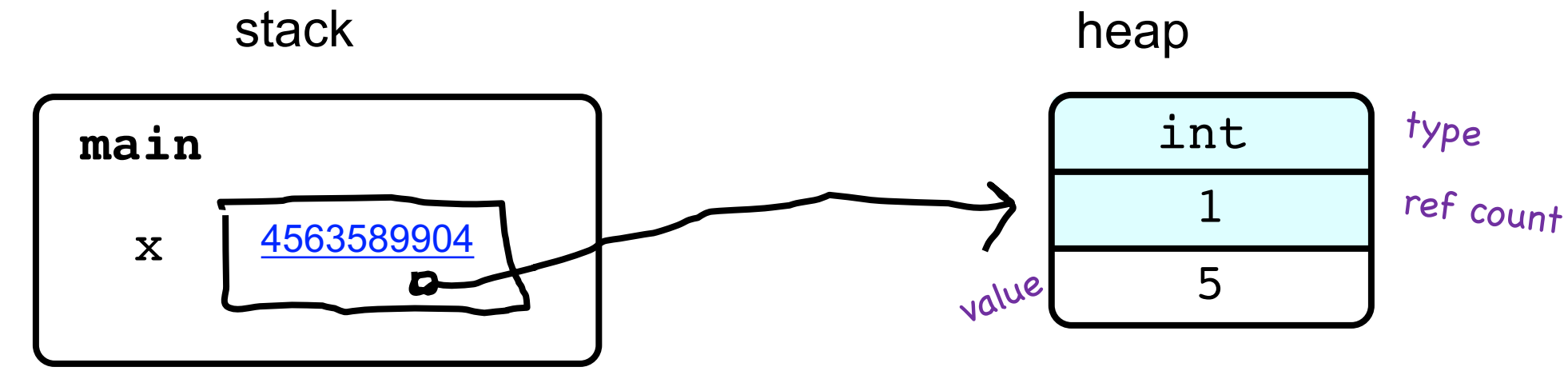
What does this do?

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



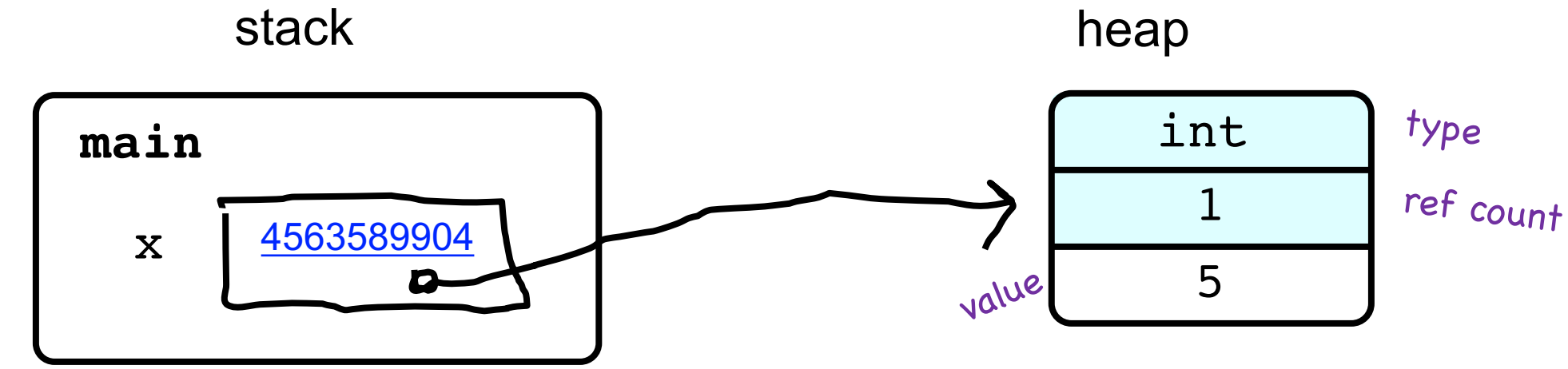
What does this do?

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



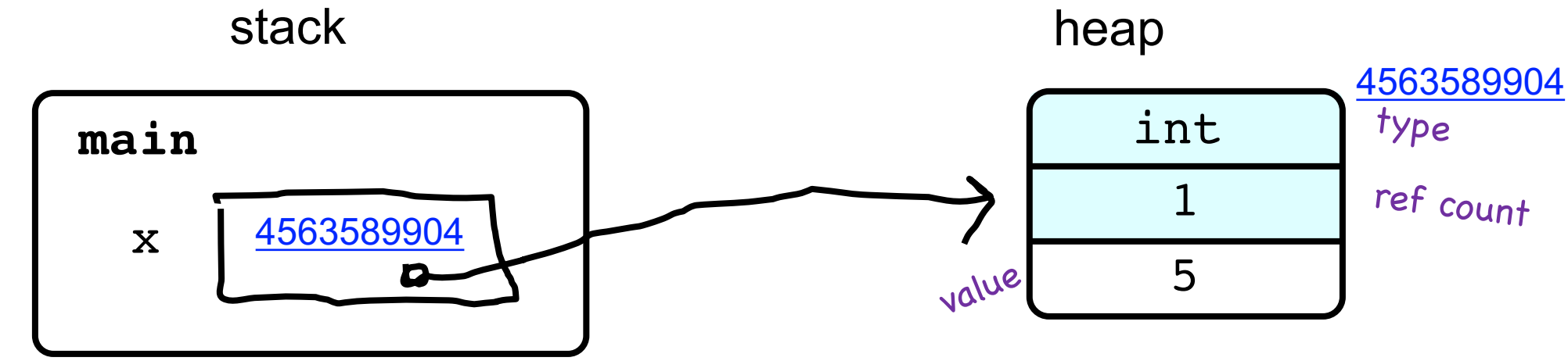
What does this do?

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



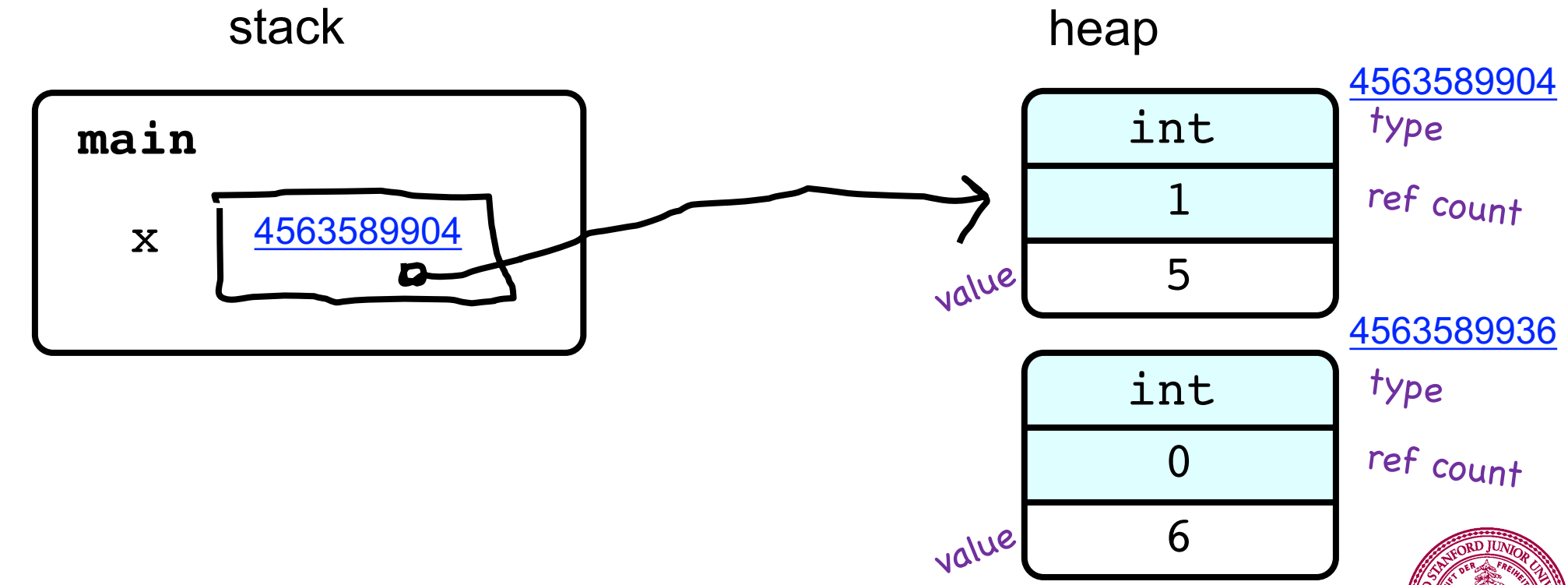
What does this do?

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



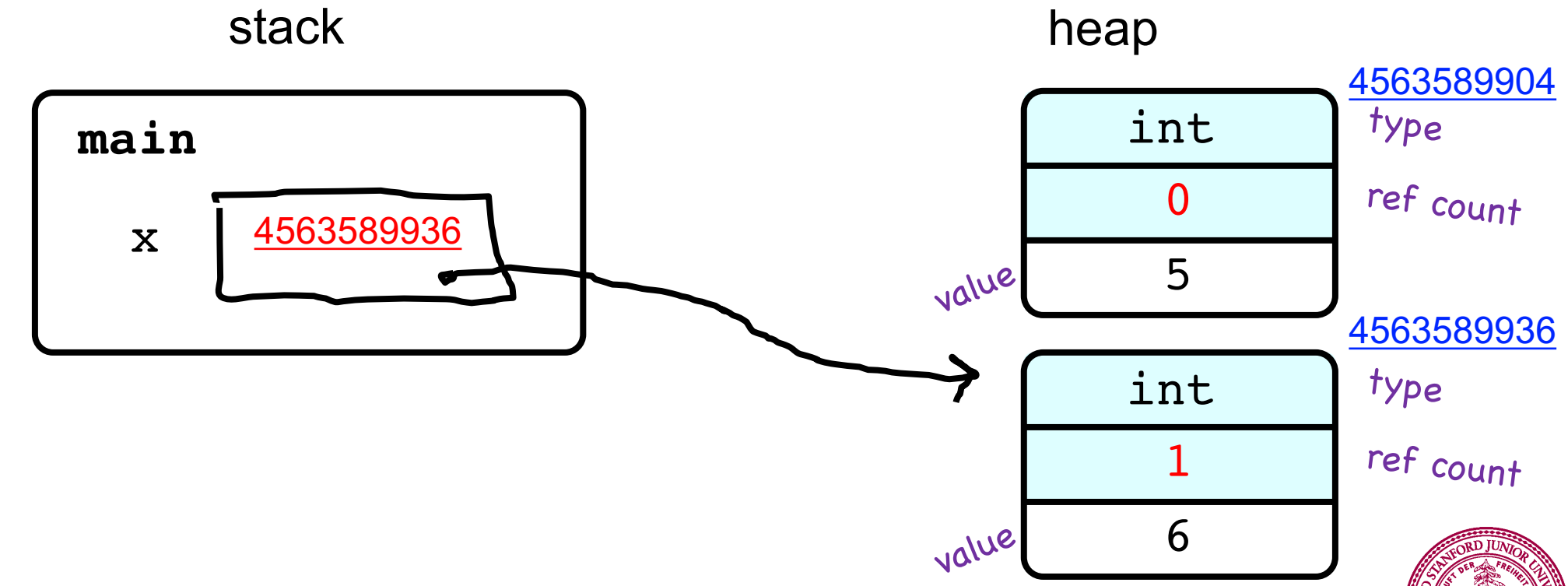
What does this do?

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



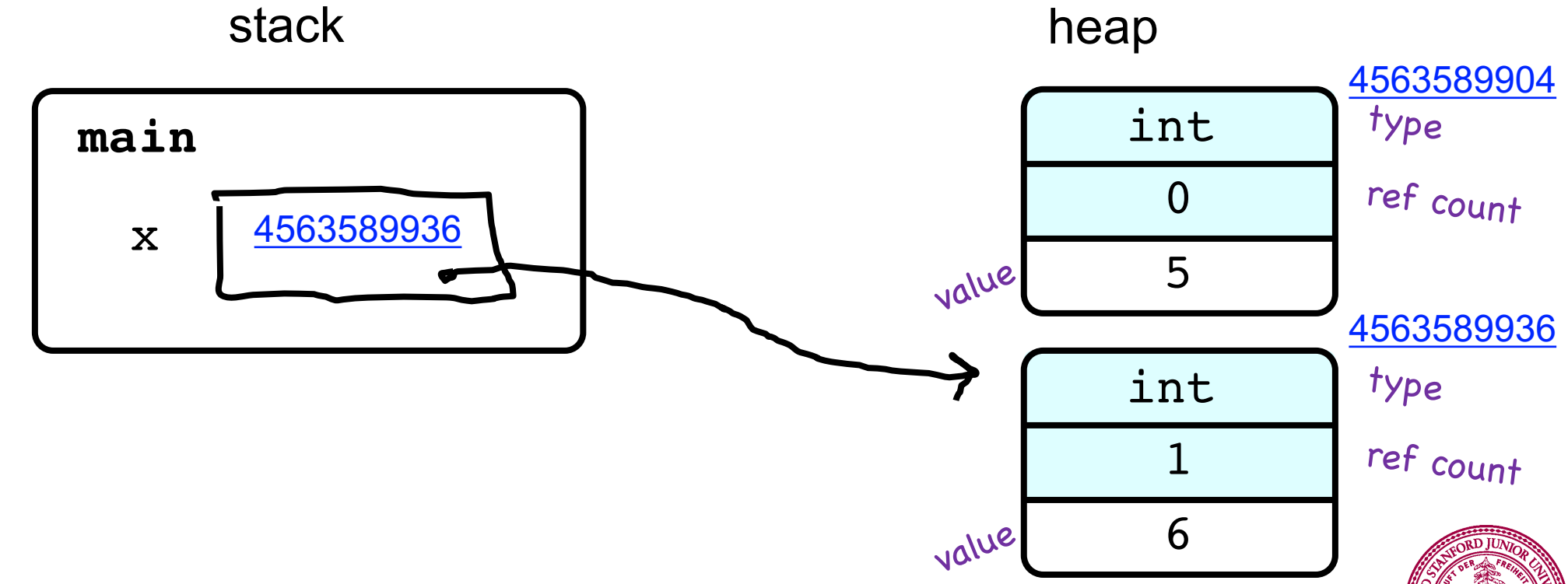
What does this do?

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

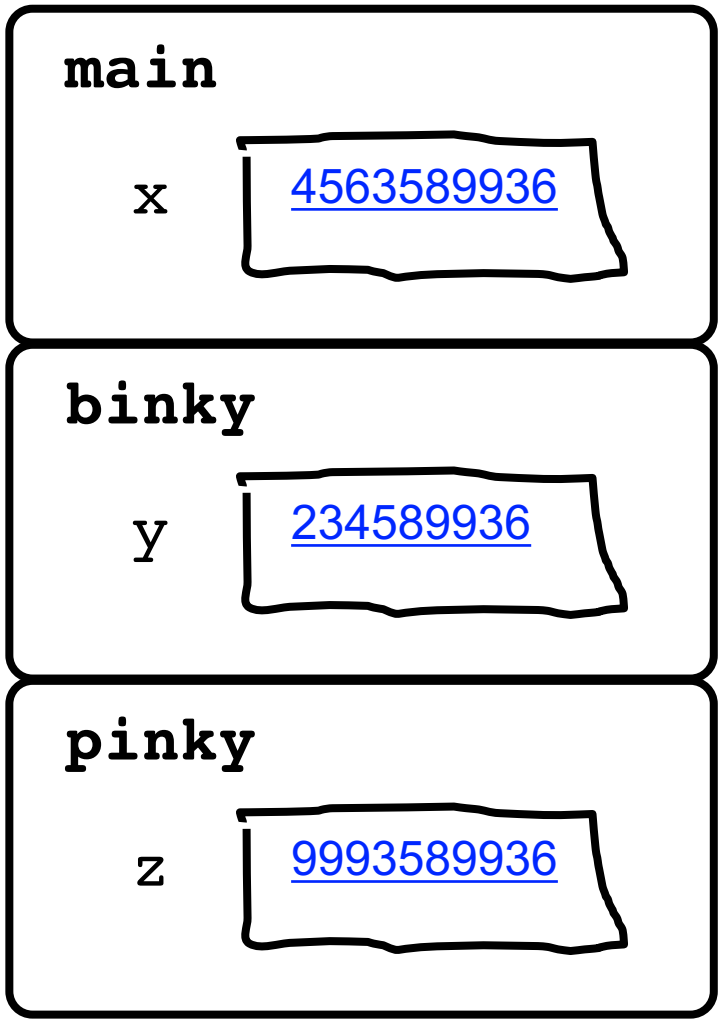


What does this do?

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



The stack



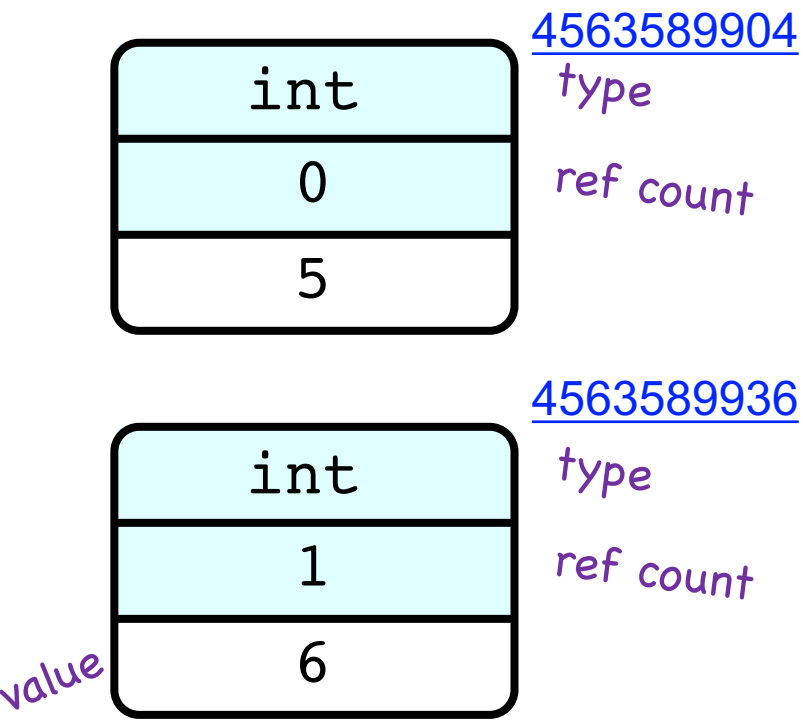
Each time a function is called, a new frame of memory is created.

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

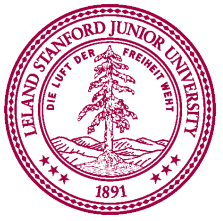
Every value has an address (like a URL address)

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” you are changing its **reference**

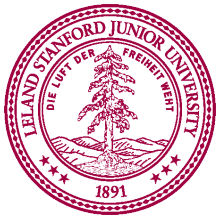
You know a variable is being assigned to if it is on the left hand side of an = sign

What does this do?

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

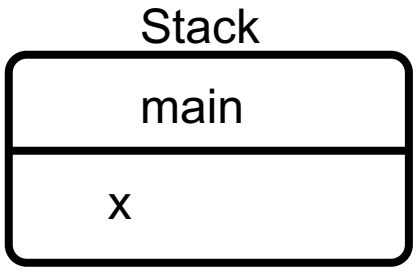
When a variable is “used” you are accessing its **value**

You know a variable is being used to if it is **not** on the left hand side of an = sign



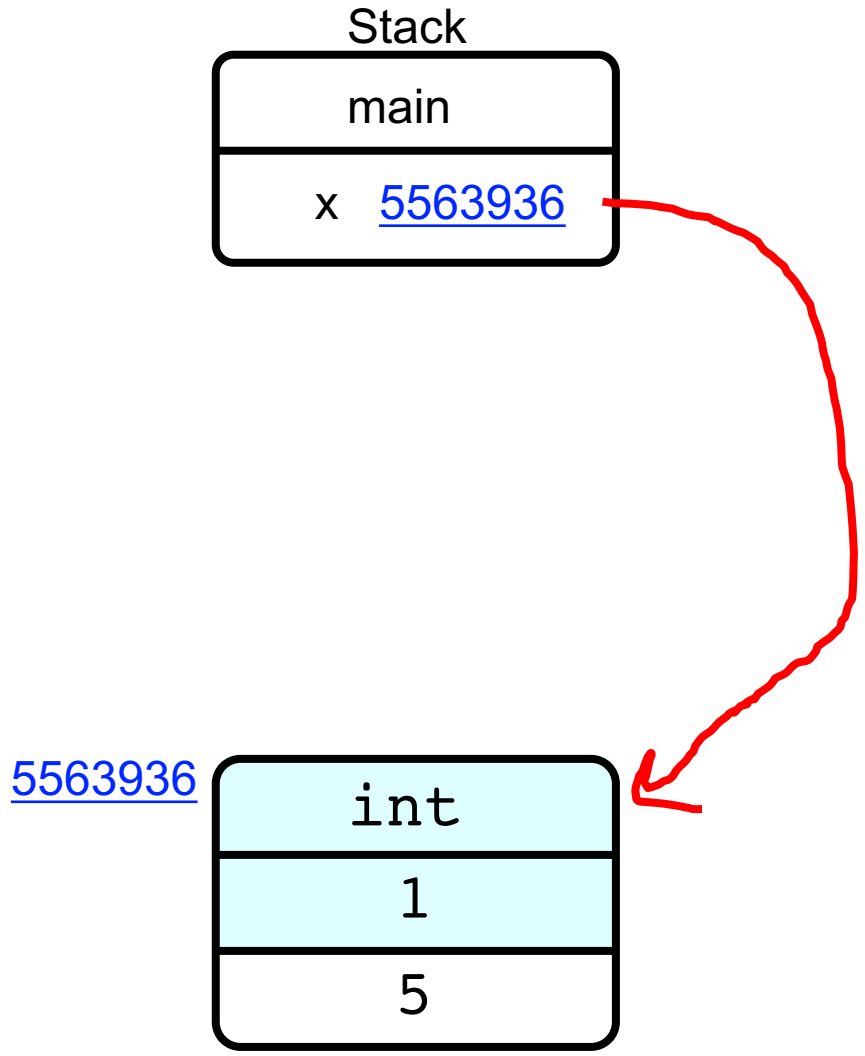
What does this do?

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

def binky(y):
    pinky(y)

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



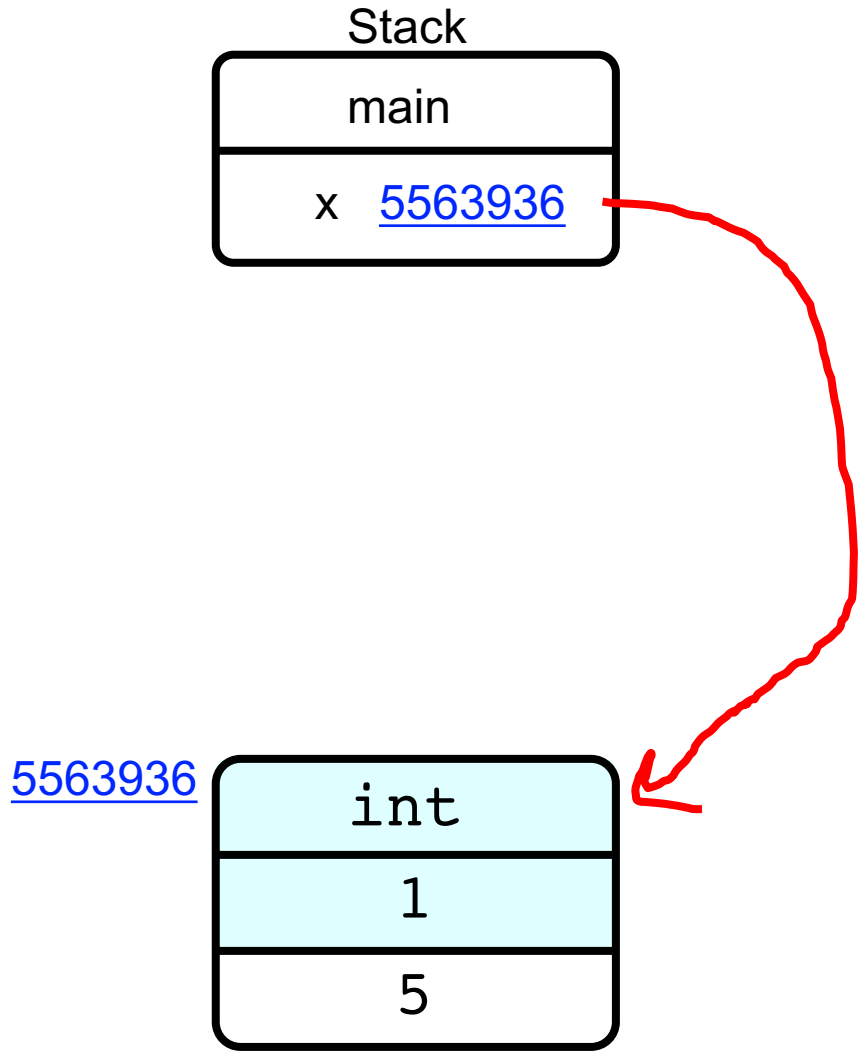
What does this do?

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

def binky(y):
    pinky(y)

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



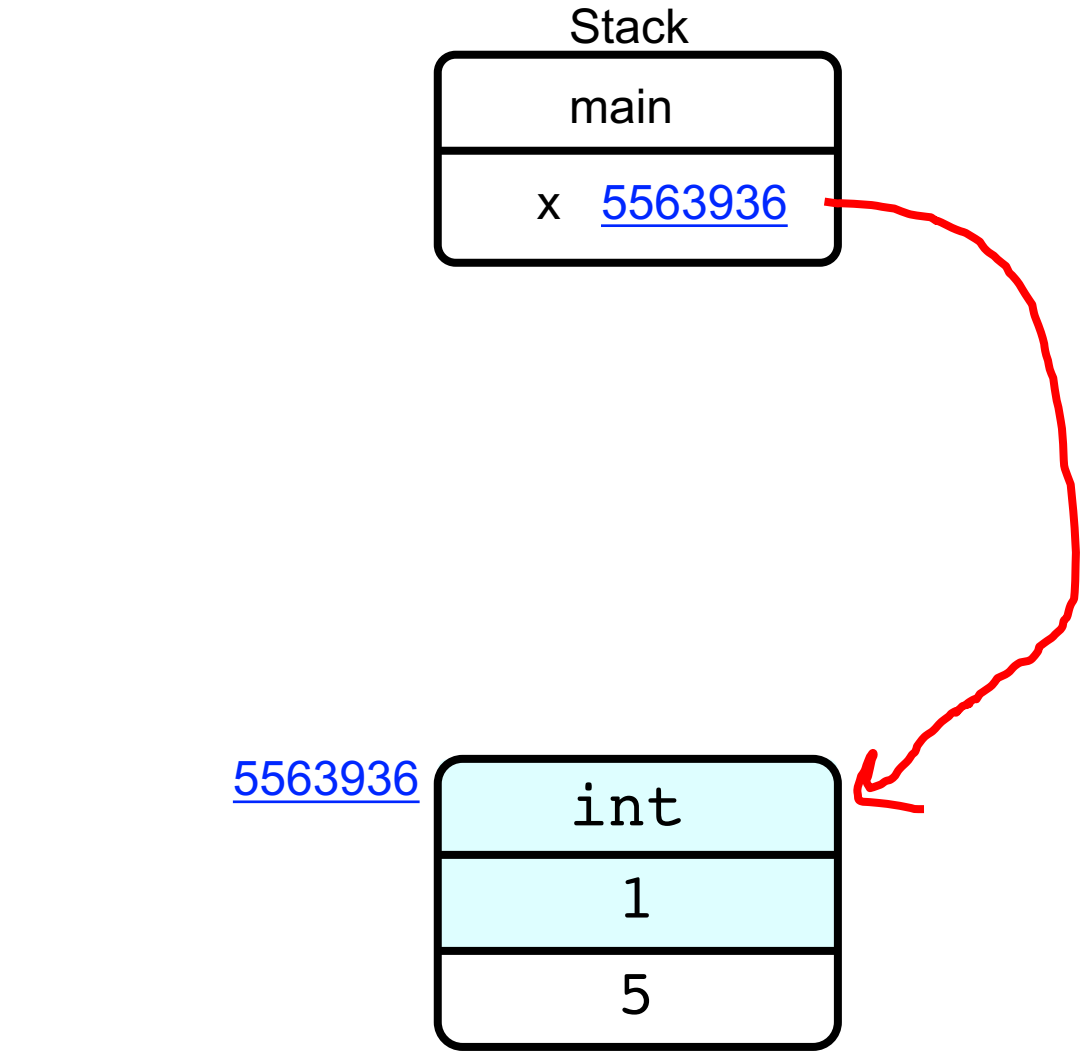
What does this do?

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

def binky(y):
    pinky(y)

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



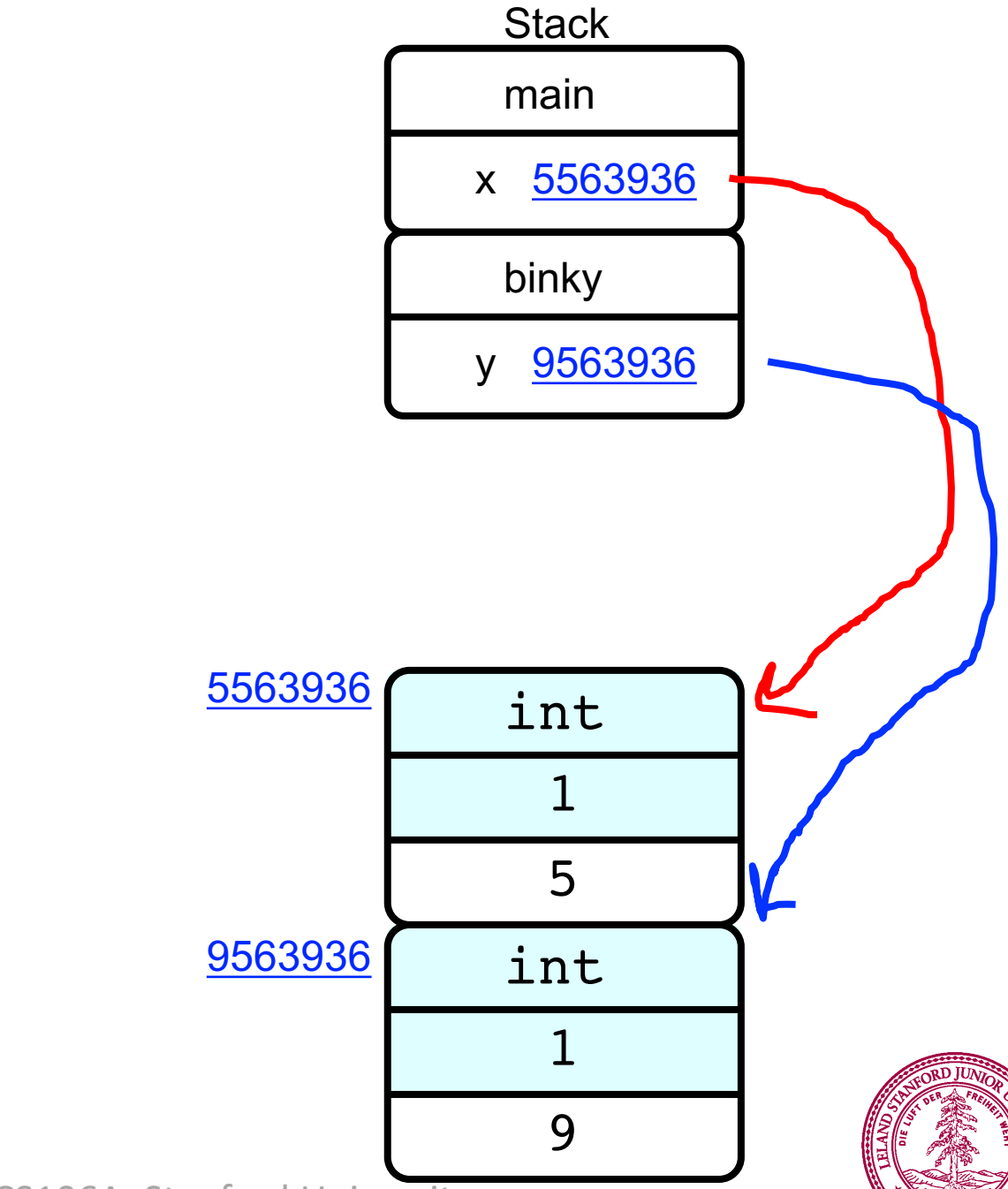
What does this do?

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

def binky(y):
    pinky(y)

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



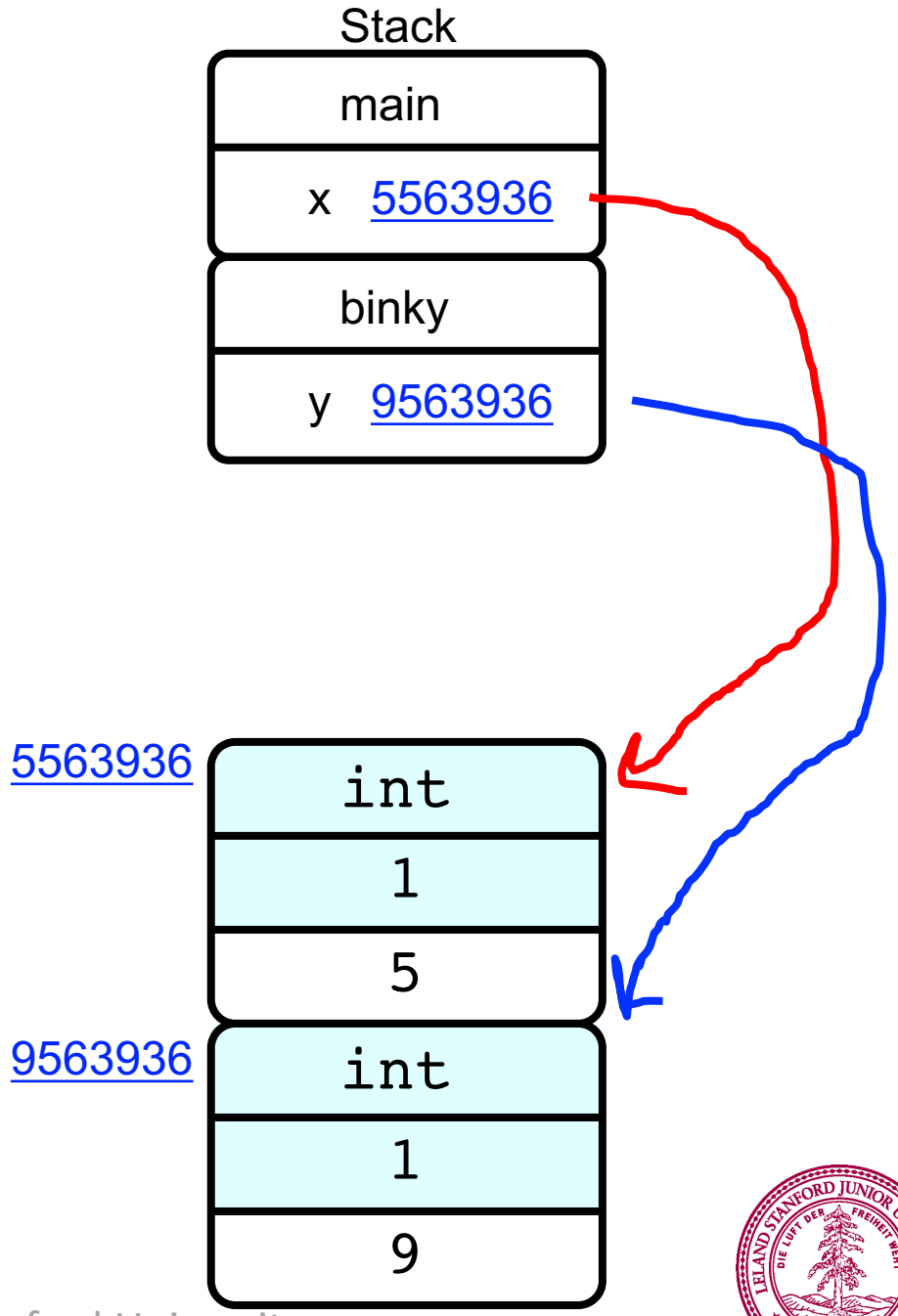
What does this do?

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

def binky(y):
    pinky(y)

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



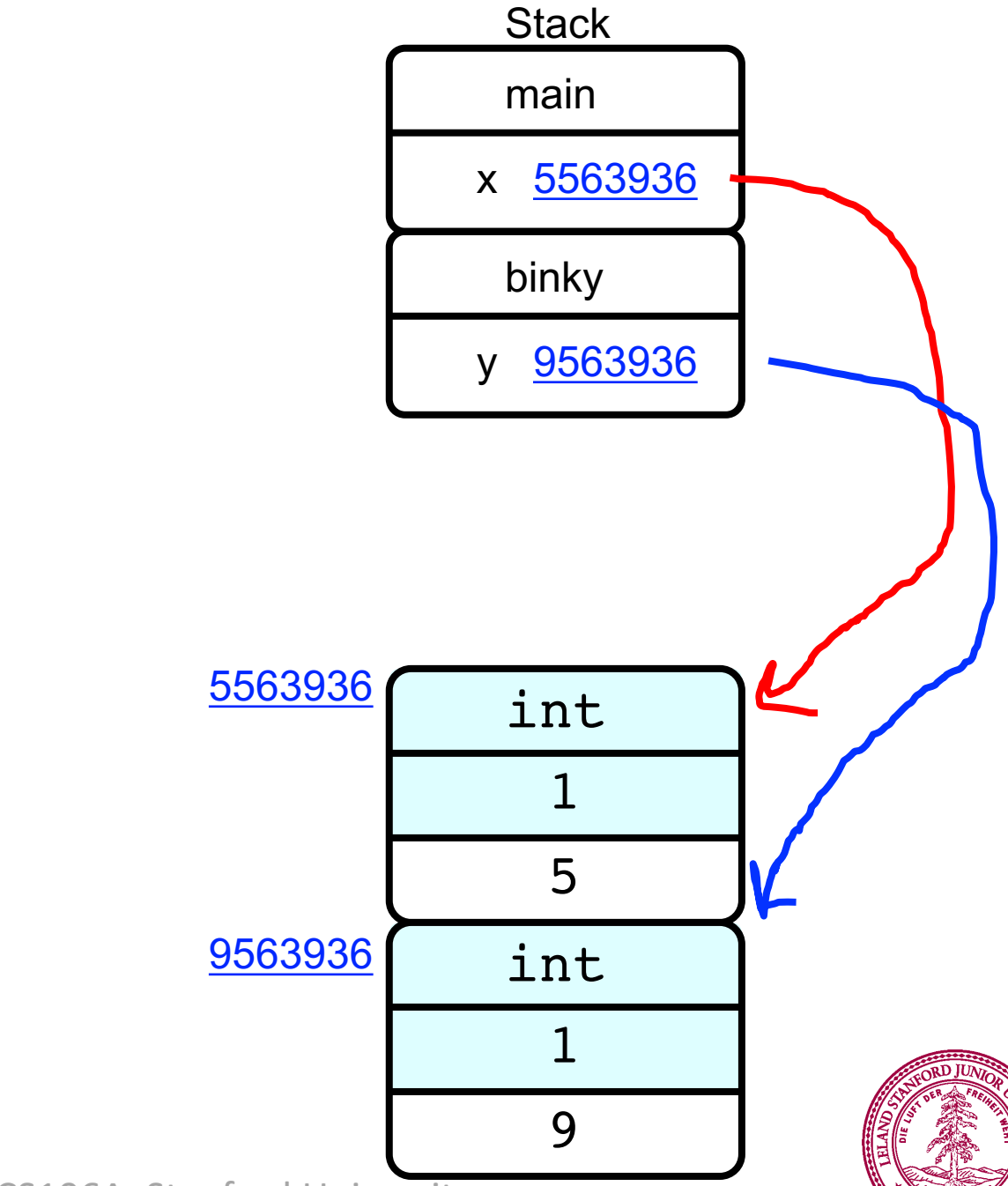
What does this do?

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

def binky(y):
    pinky(y)

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



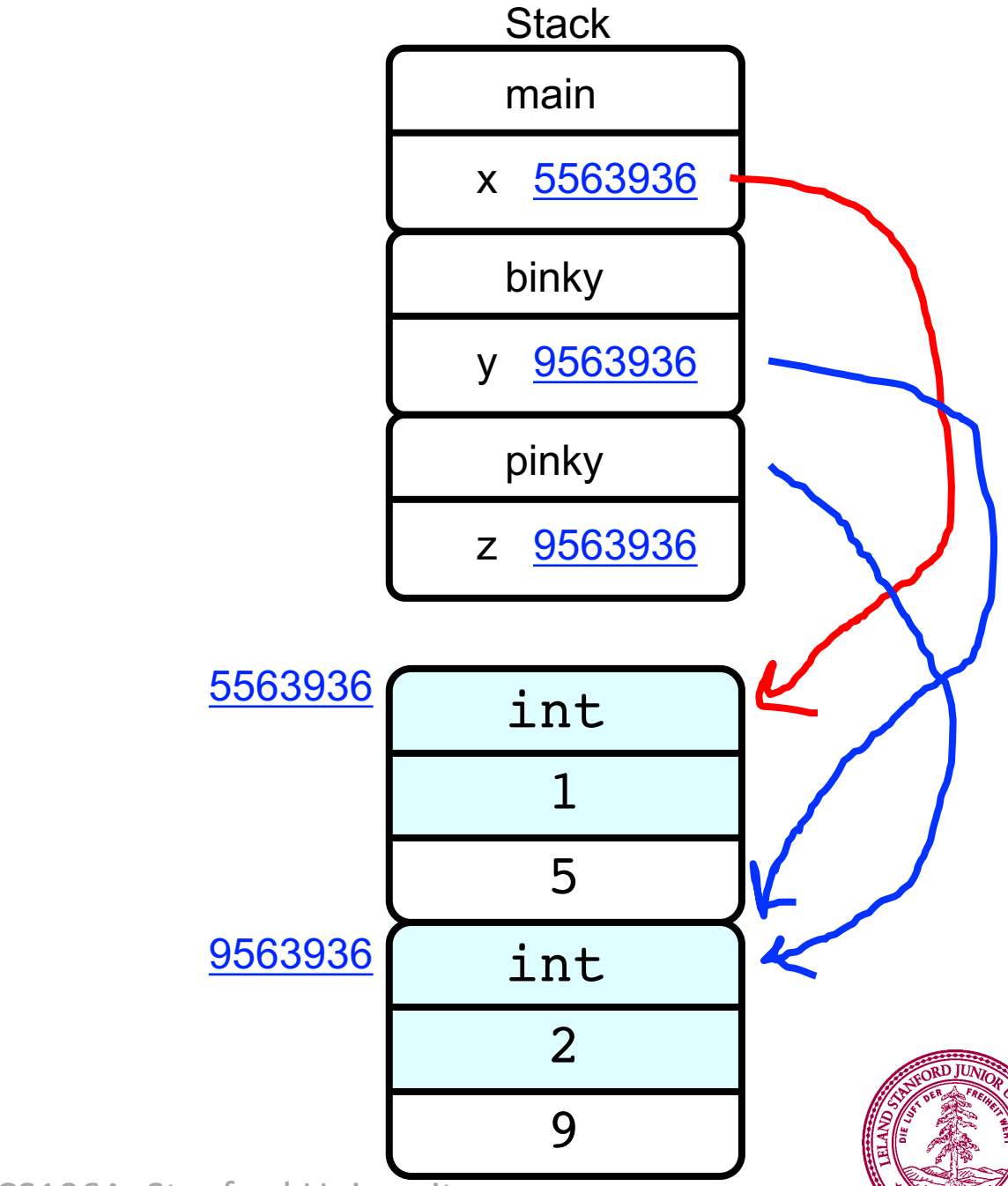
What does this do?

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

def binky(y):
    pinky(y)

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



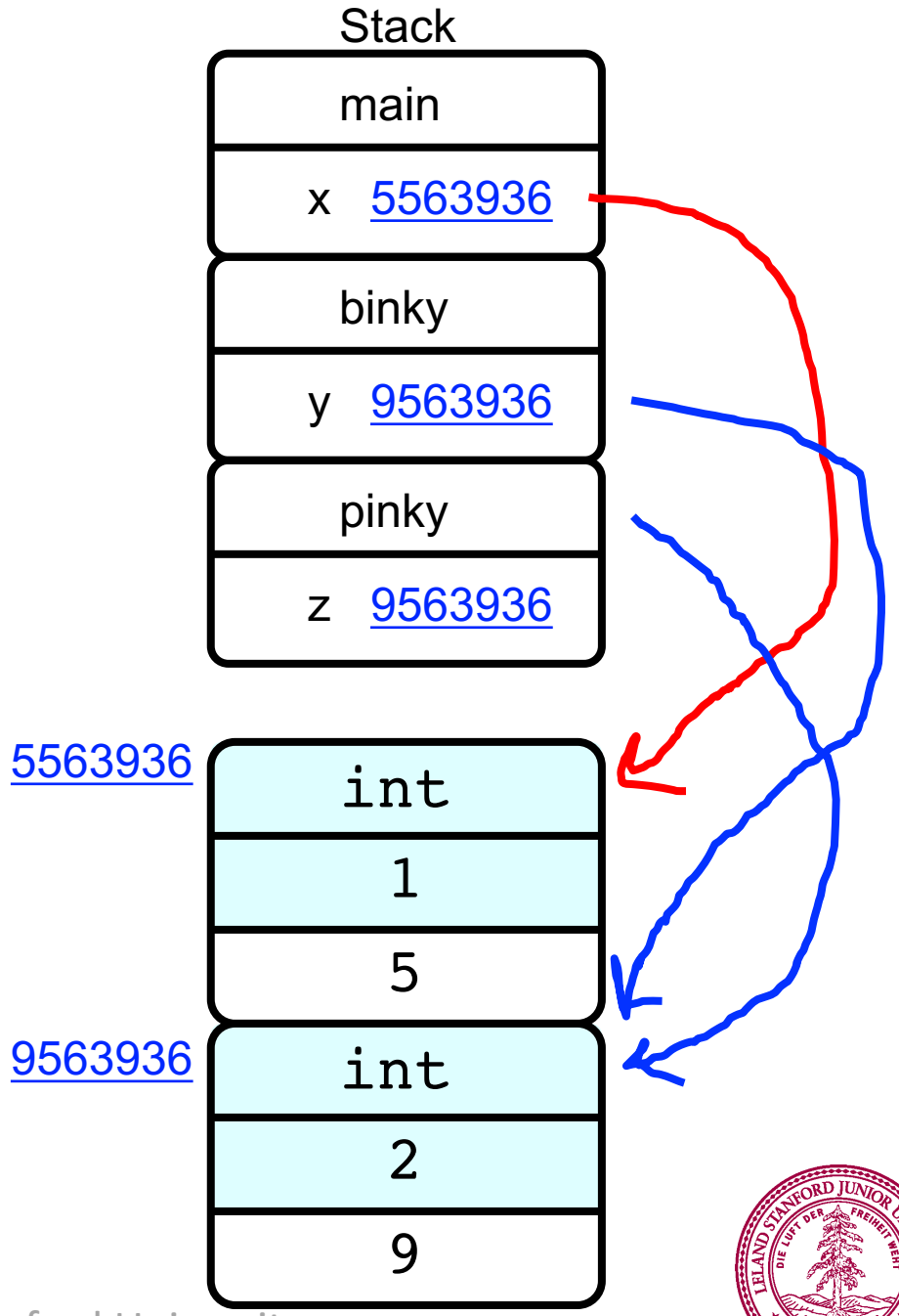
What does this do?

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

def binky(y):
    pinky(y)

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



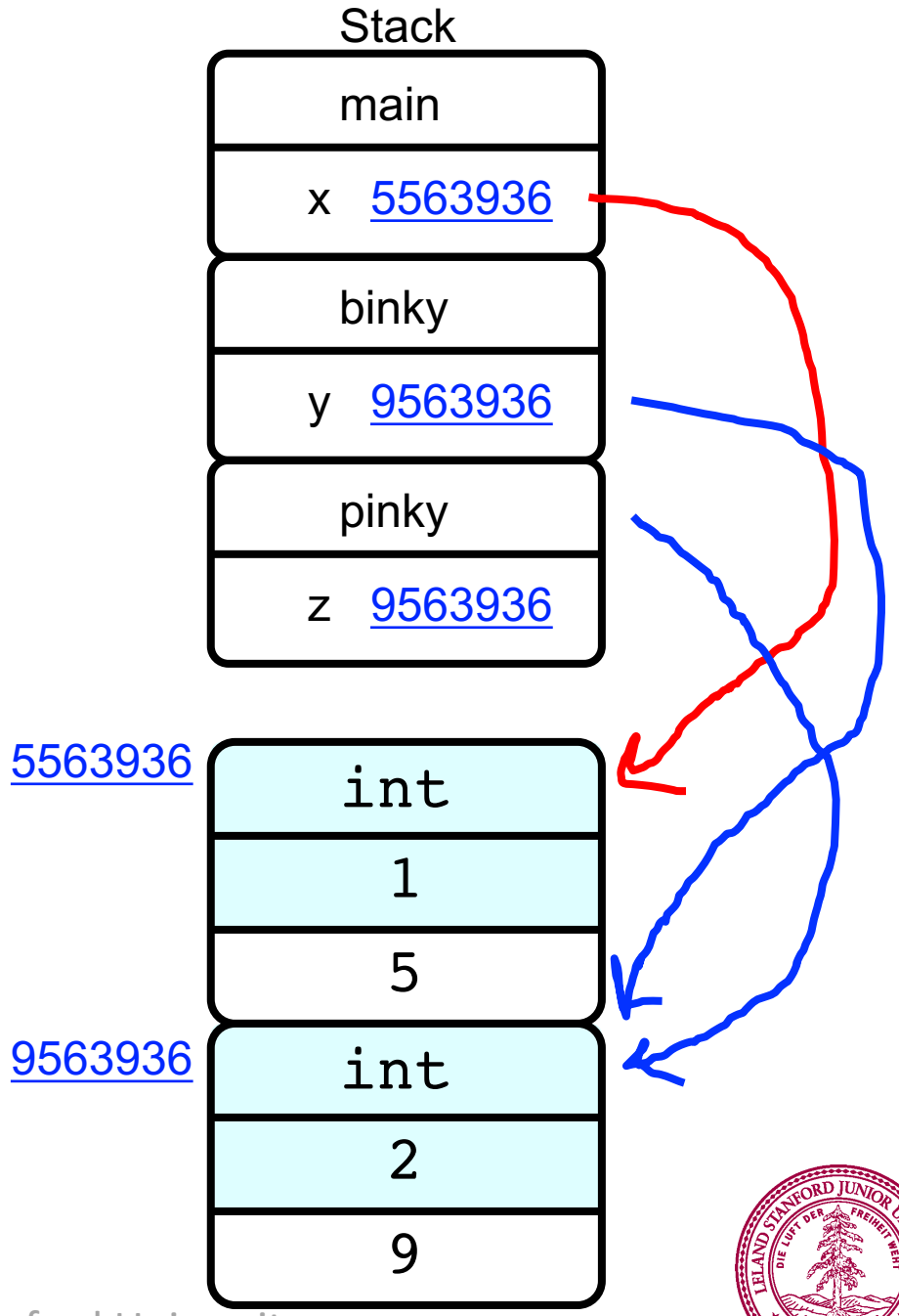
What does this do?

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

def binky(y):
    pinky(y)

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



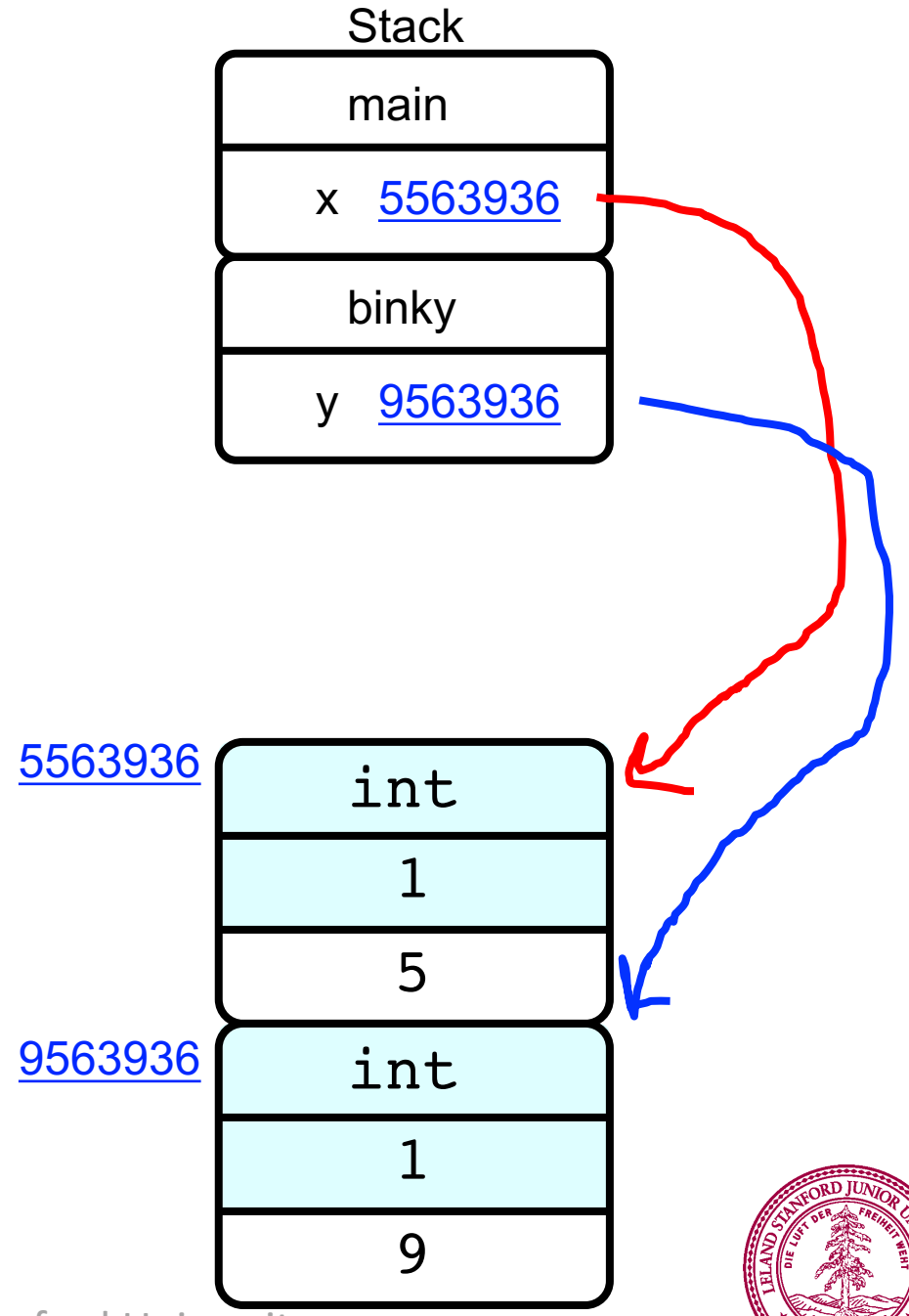
What does this do?

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

def binky(y):
    pinky(y)

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



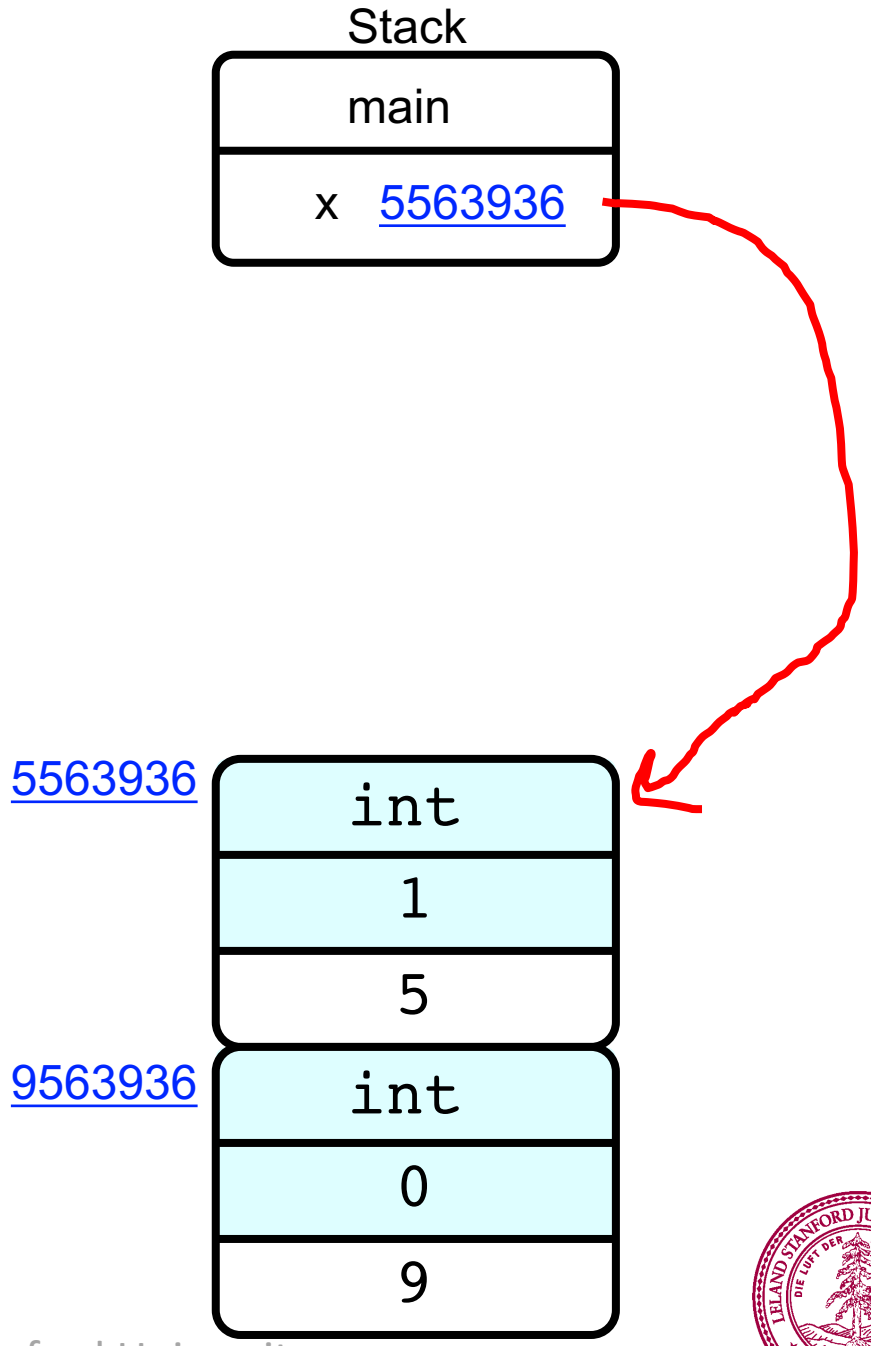
What does this do?

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

def binky(y):
    pinky(y)

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



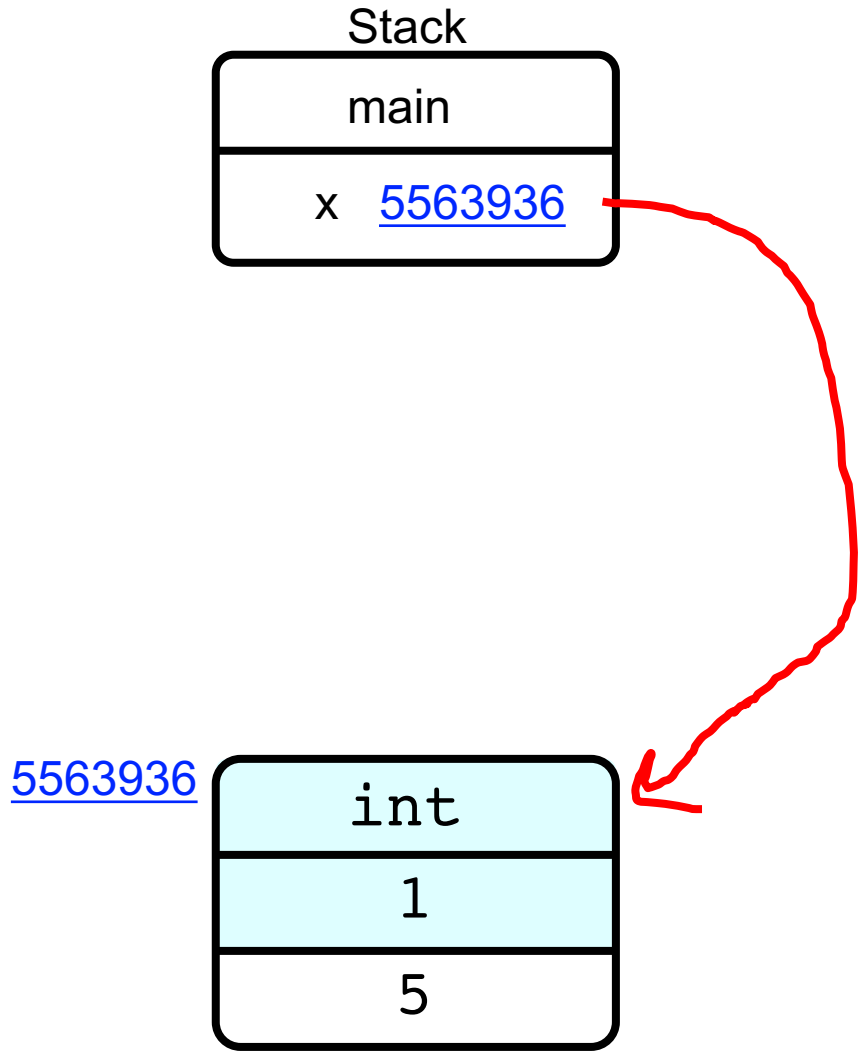
What does this do?

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

def binky(y):
    pinky(y)

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



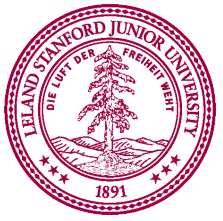
What does this do?

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

def binky(y):
    pinky(y)

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

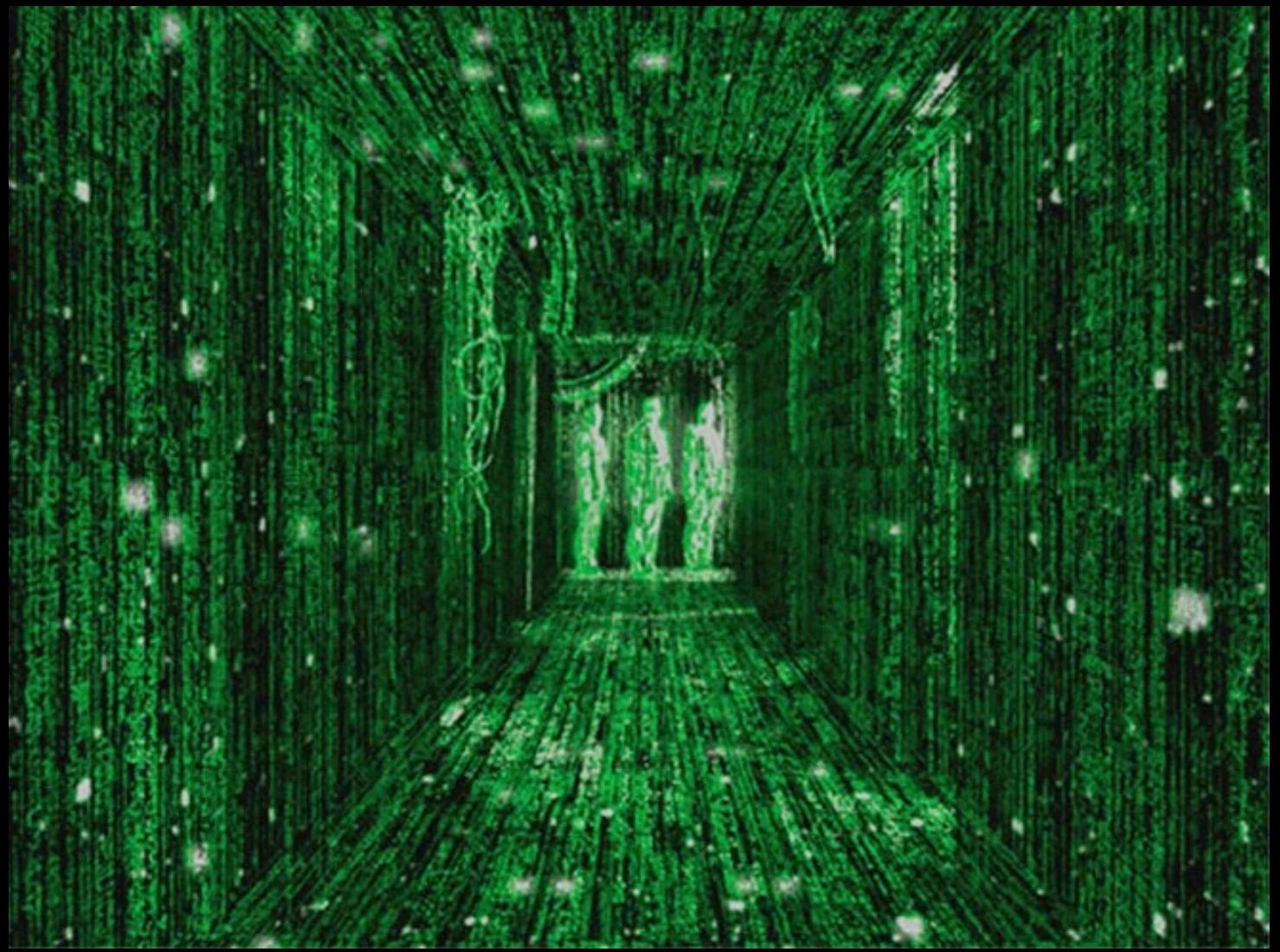


What does this do?

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

def binky(y):
    pinky(y)

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

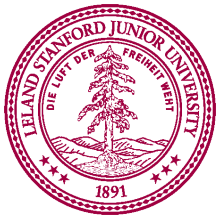
What is… the matrix?



The matrix origins

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

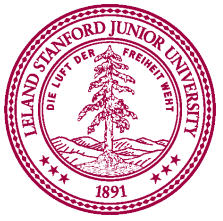
```
def main():
    x = ['a', 'b', 'c']
    update(x)

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

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

What is `self`?

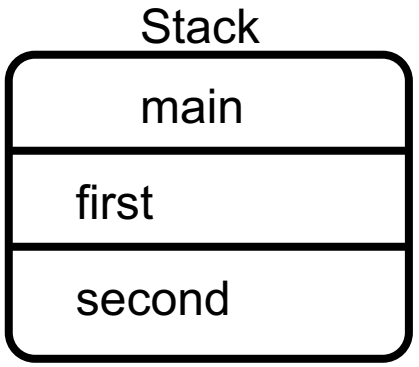
What does this do?

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

# put in another file...
def main():
    first = Dog('simba')
    print(first)
    print(type(first))
    print(id(first))
    print(first.__dict__)
```

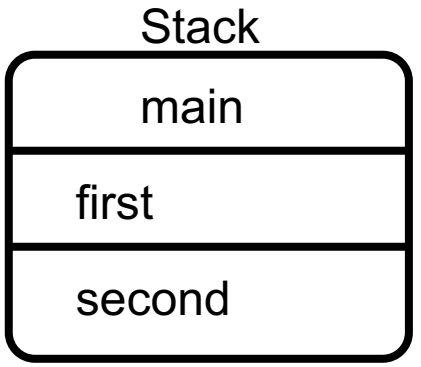
What does this do?

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

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

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

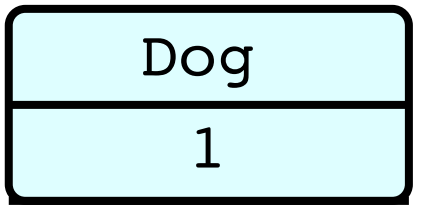


What does this do?

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

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

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

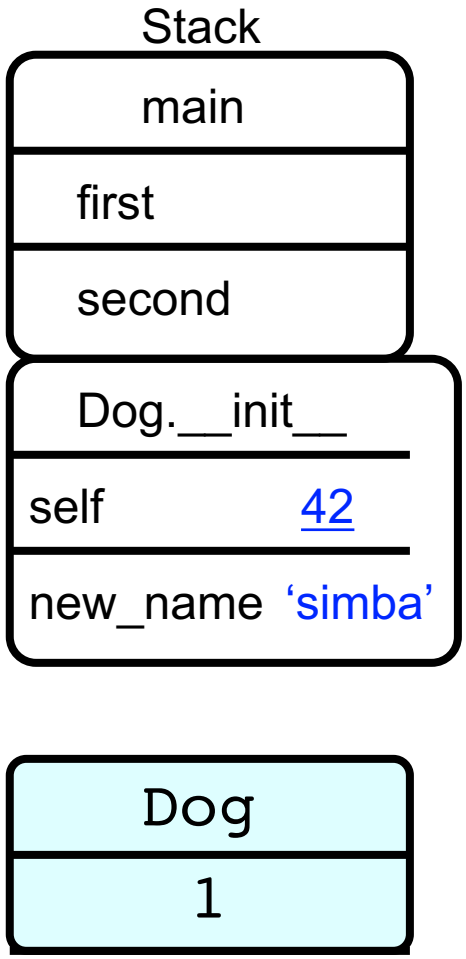


42



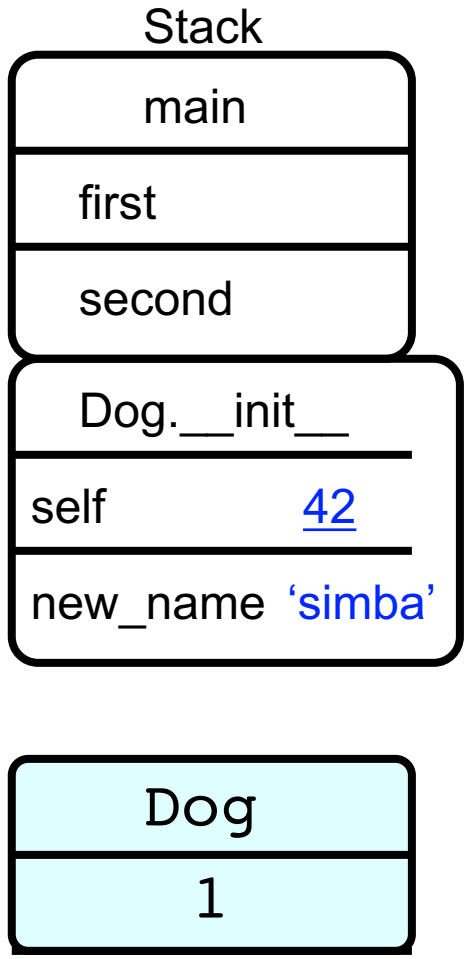
What does this do?

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

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

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



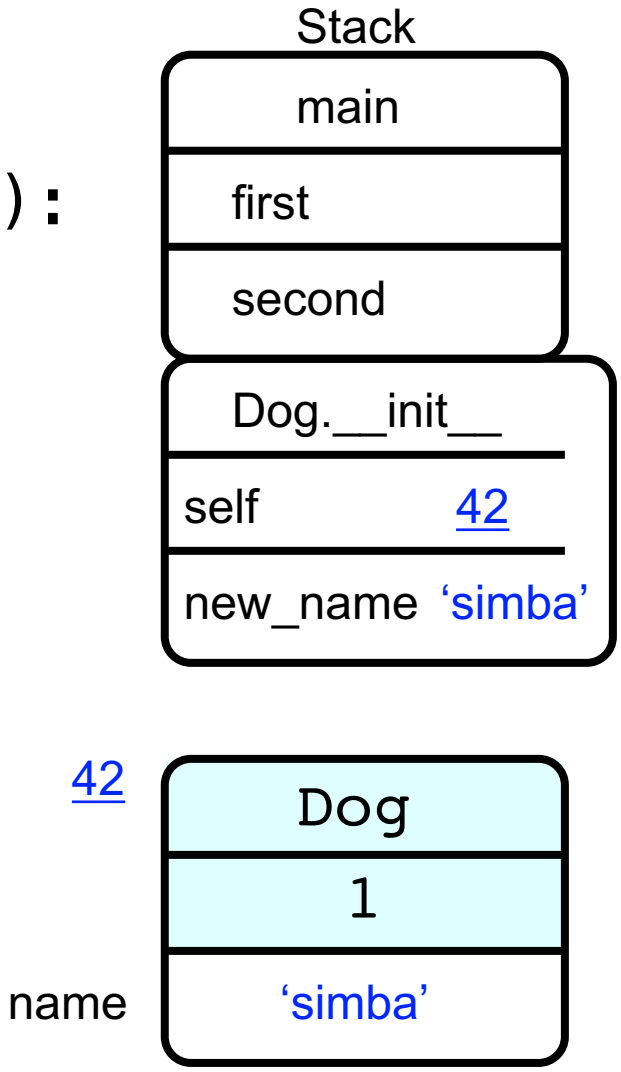
What does this do?

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

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

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



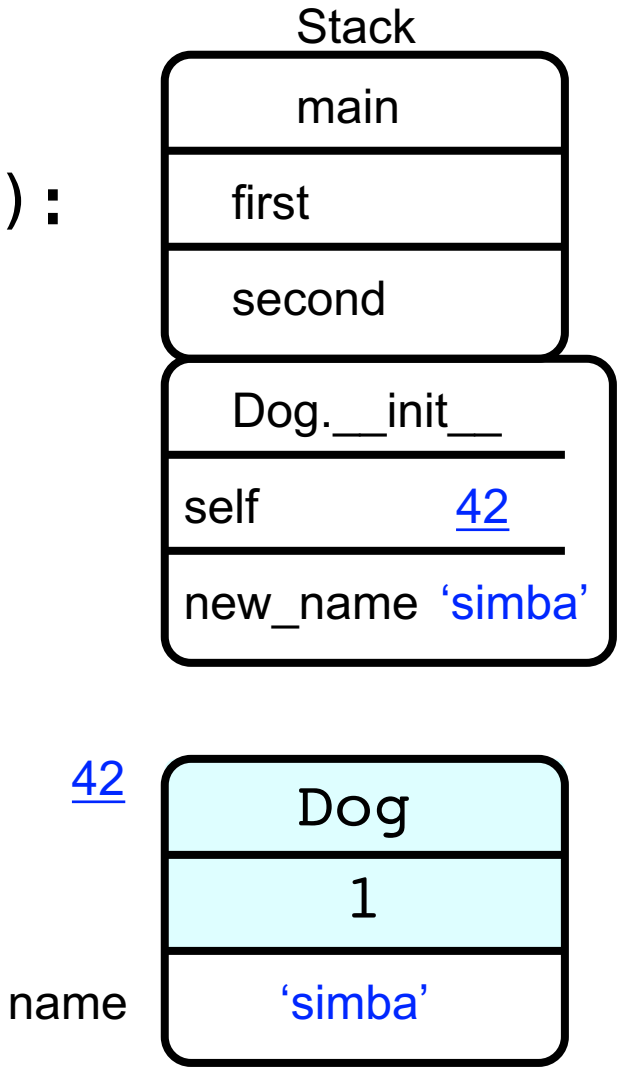
What does this do?

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

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

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



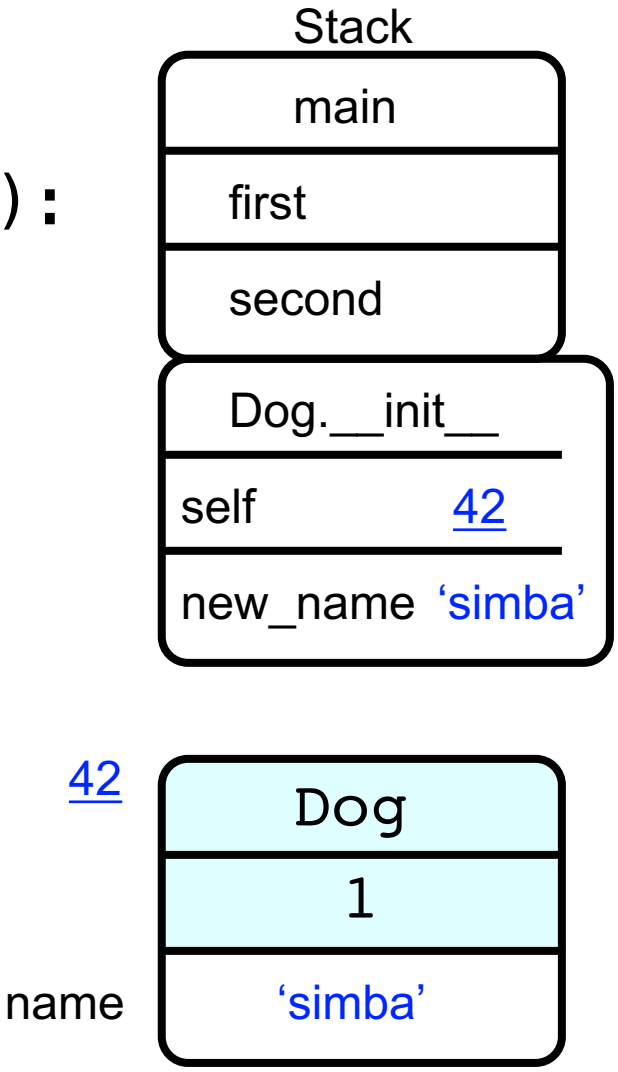
What does this do?

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

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

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



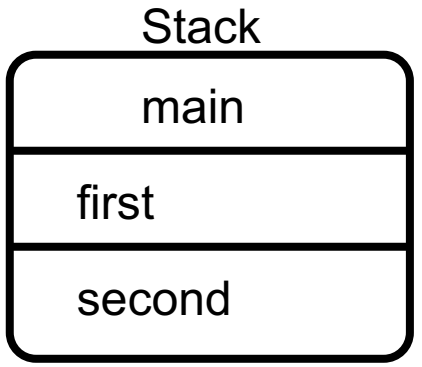
What does this do?

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

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

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



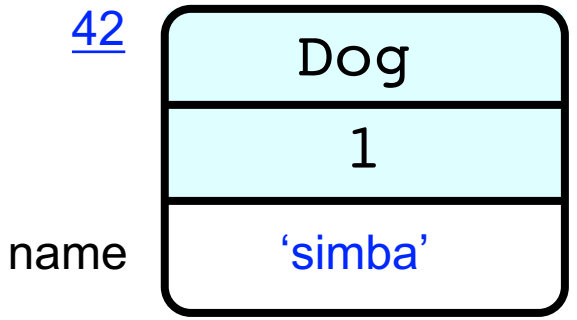
What does this do?

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

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

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



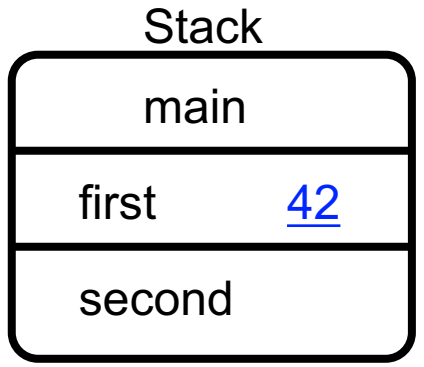
42

42



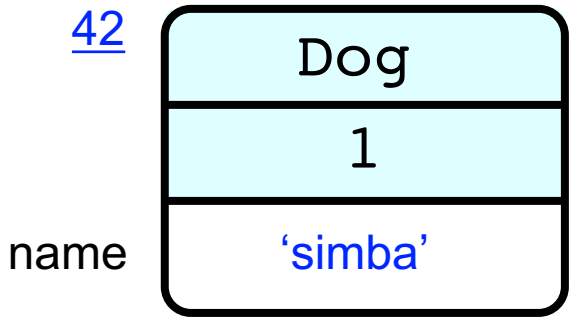
What does this do?

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

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

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



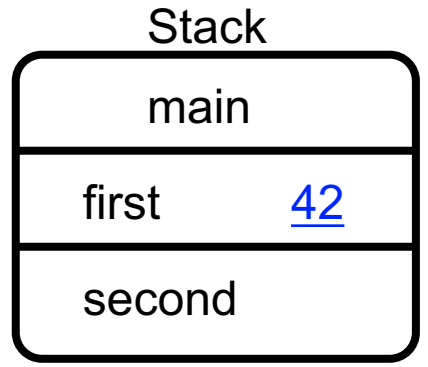
42

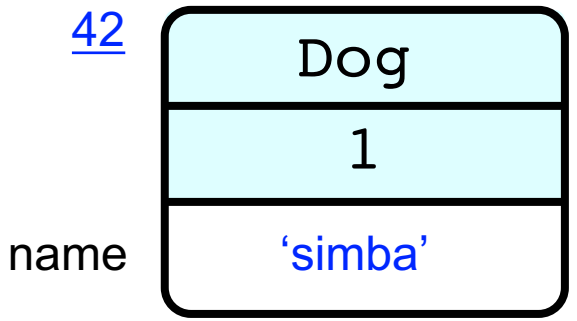
42



What does this do?

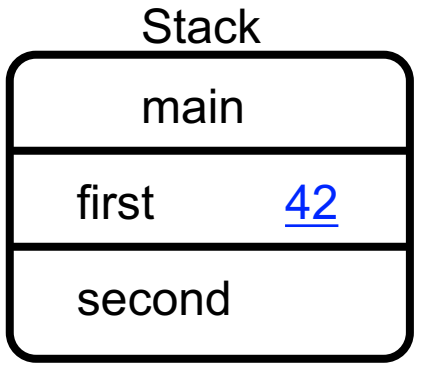
```
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

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

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

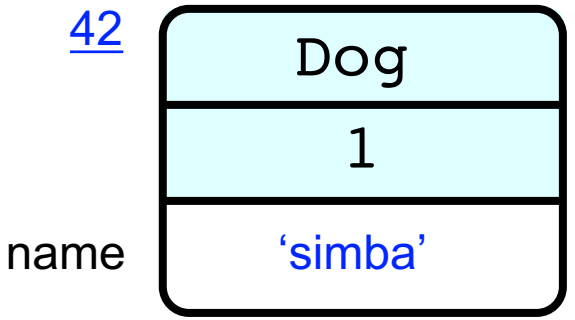




What does this do?

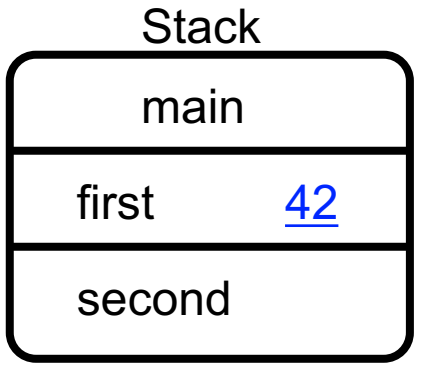
```
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

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

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

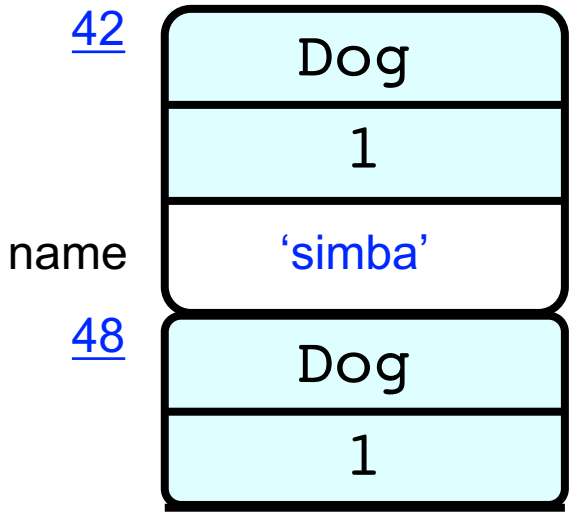




What does this do?

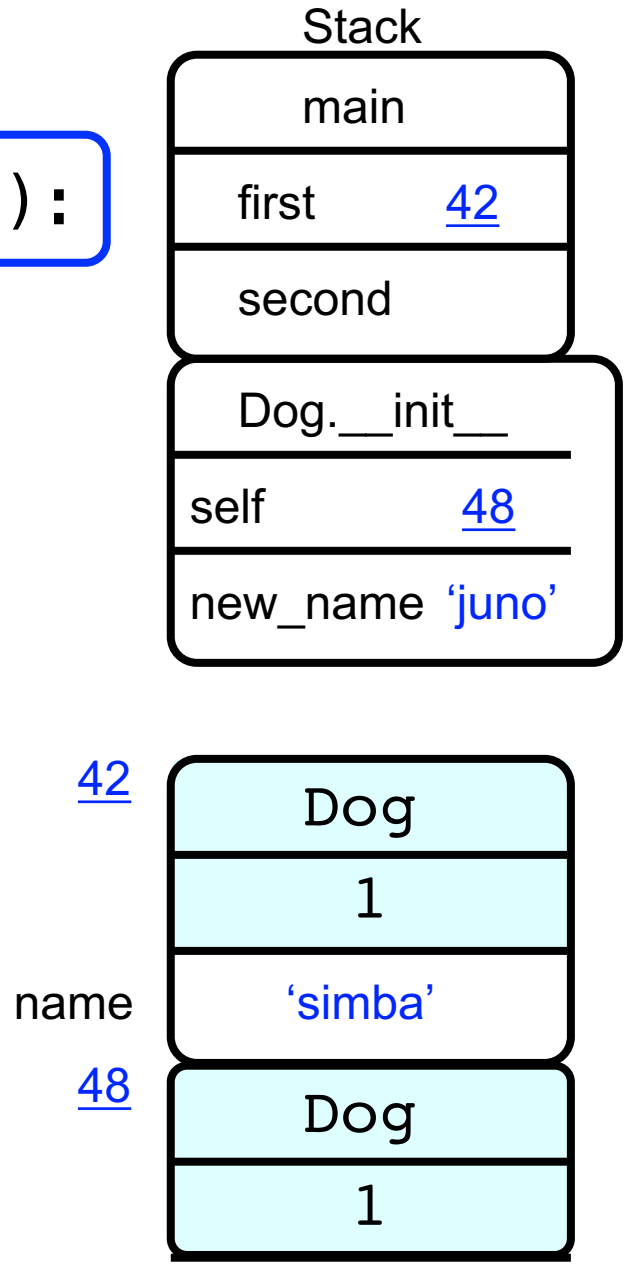
```
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

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

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





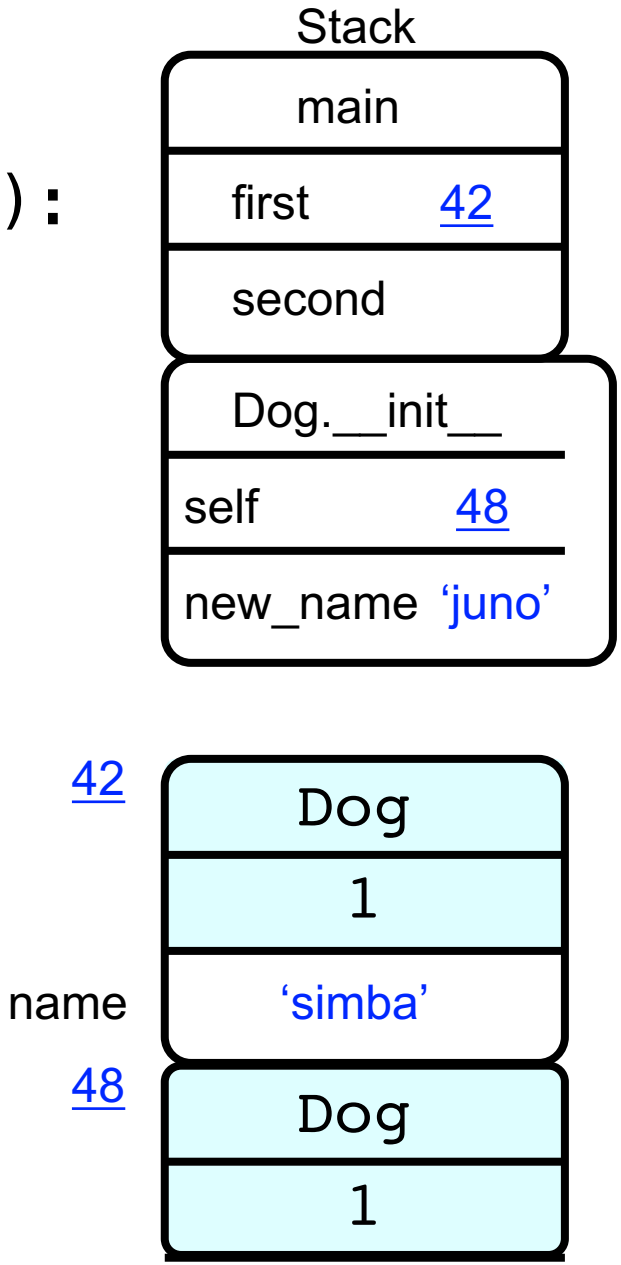
What does this do?

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

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

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



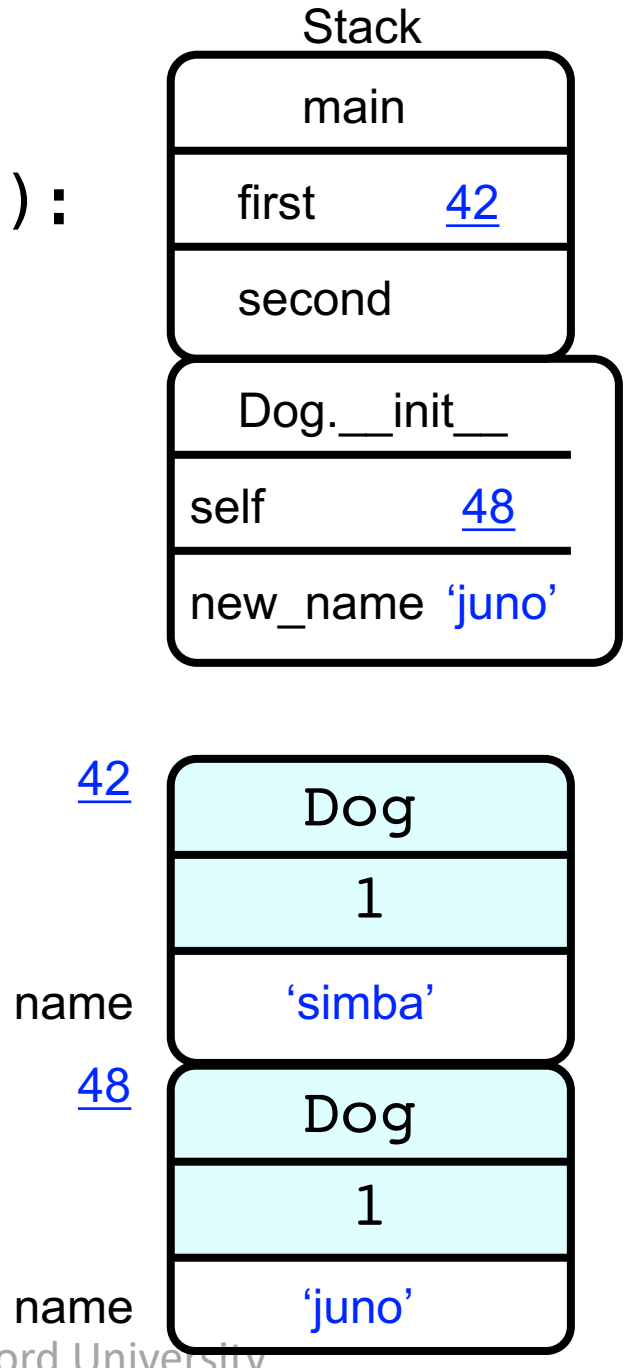
What does this do?

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

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

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



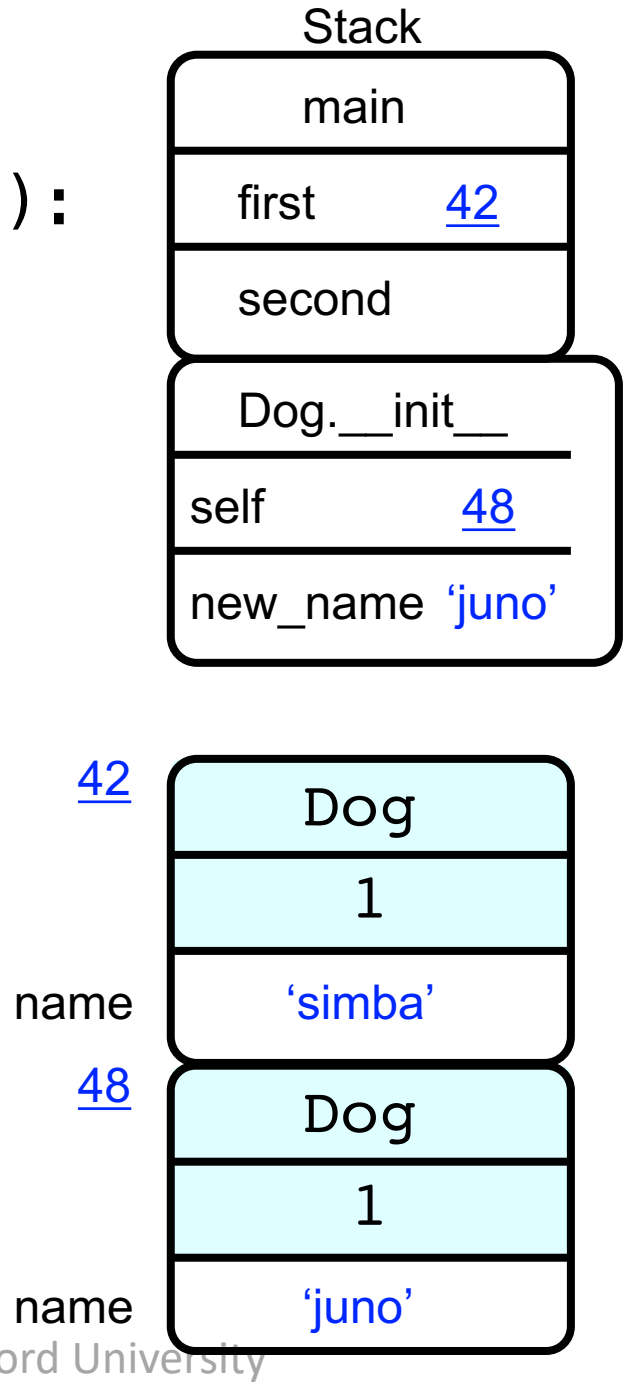
What does this do?

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

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

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



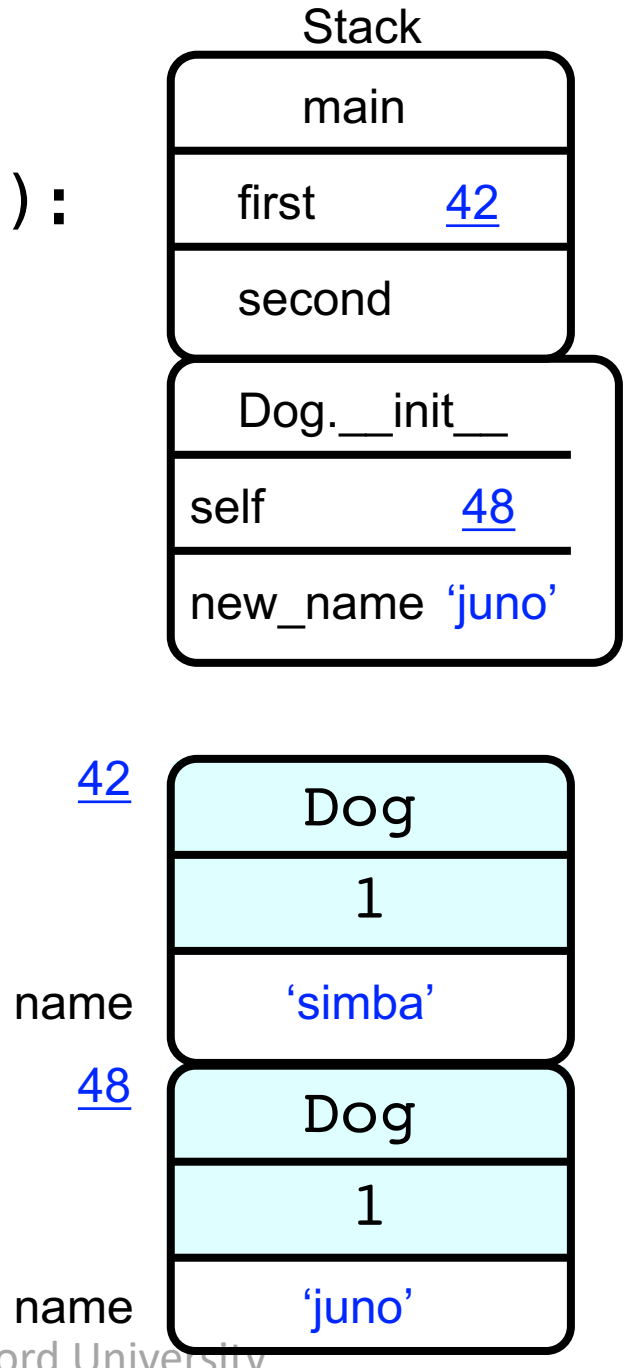
What does this do?

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

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

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



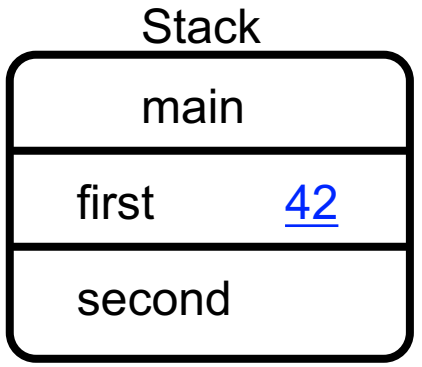
What does this do?

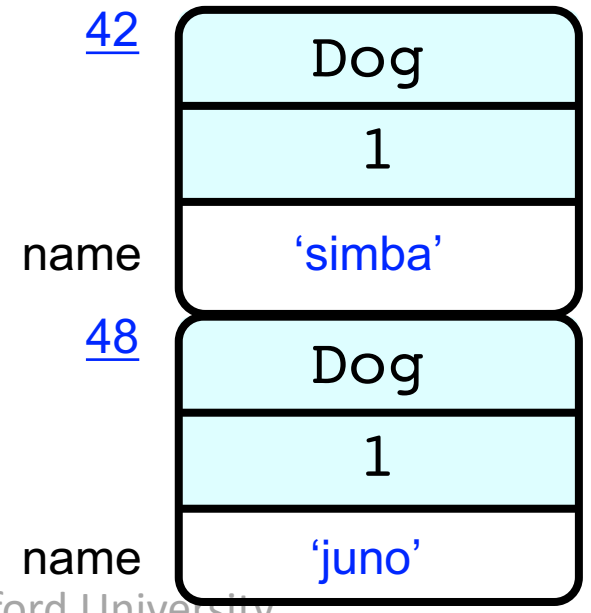
```
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

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

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



What does this do?

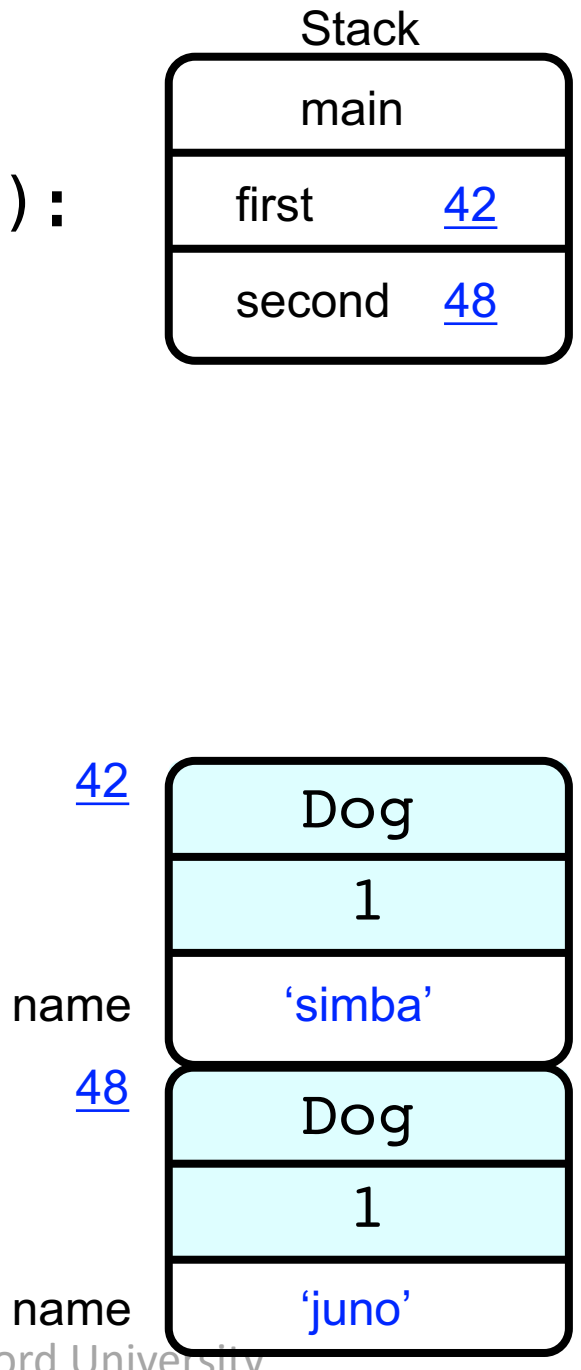
```
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

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

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





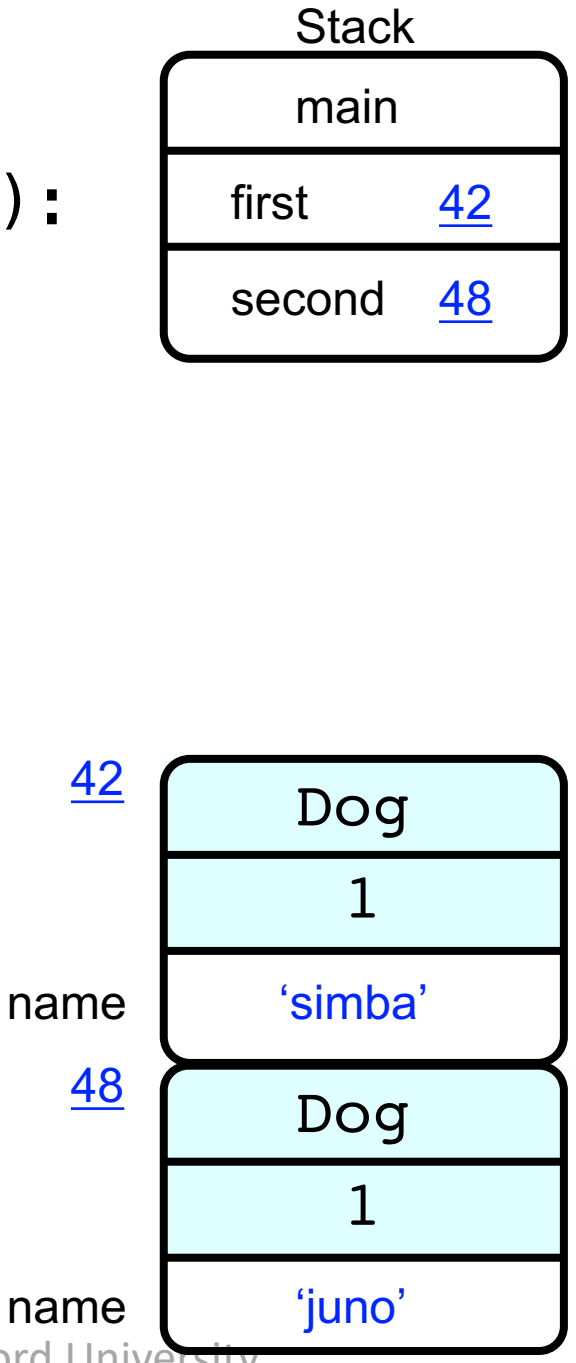
What does this do?

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

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

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



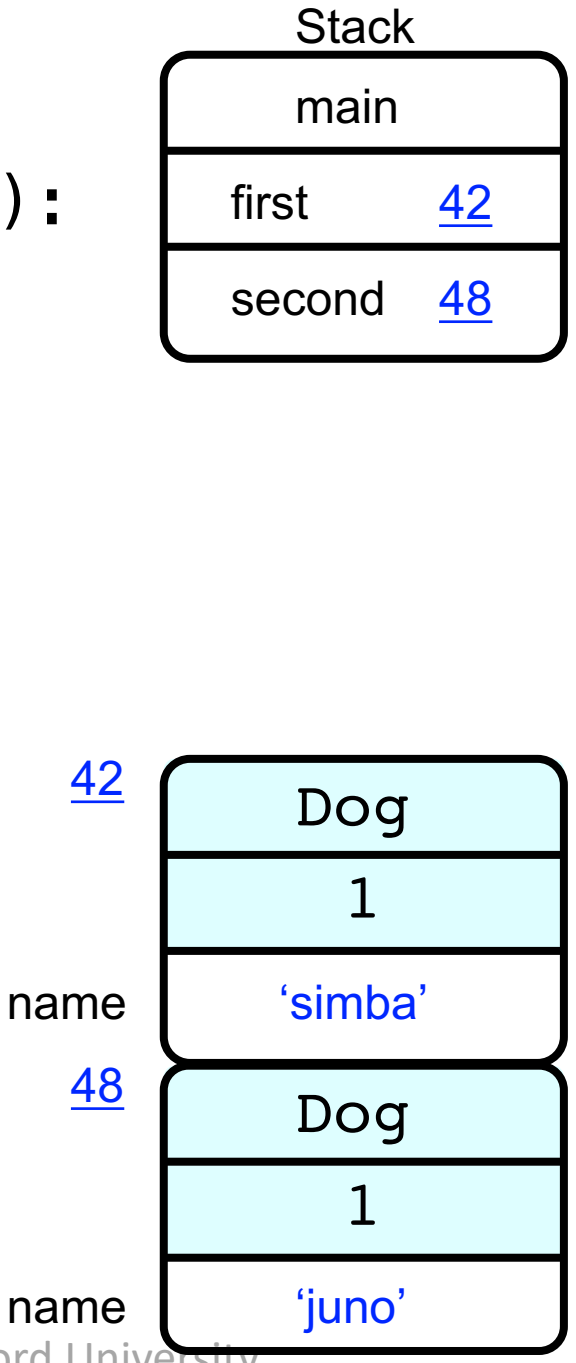
What does this do?

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

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

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



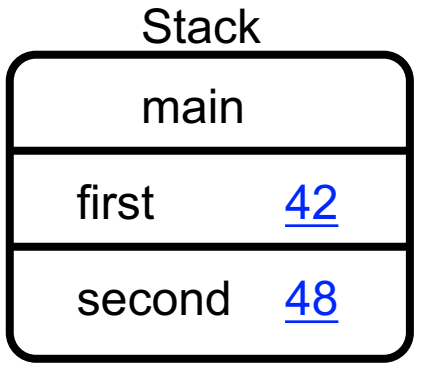
What does this do?

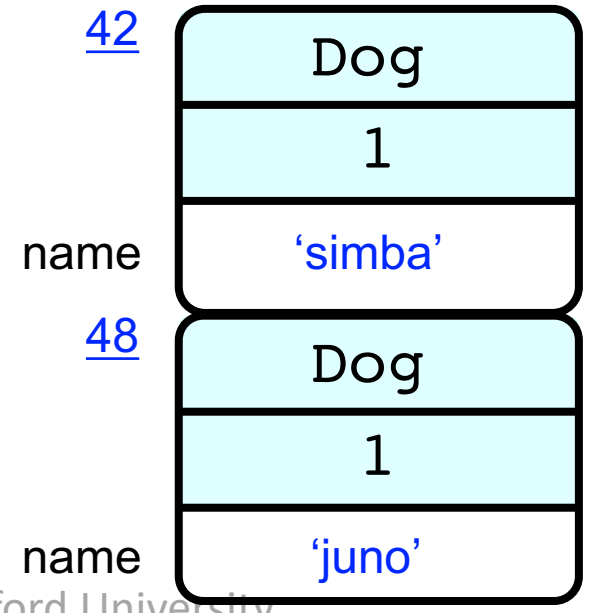
```
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

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

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



What does this do?

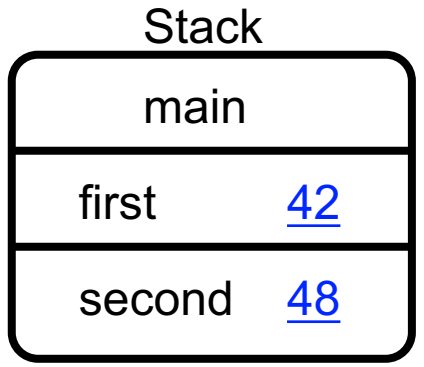
```
class Dog:
    def __init__(self, new_name):
        print(self)
        self.name = new_name
        print(self.name)

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

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

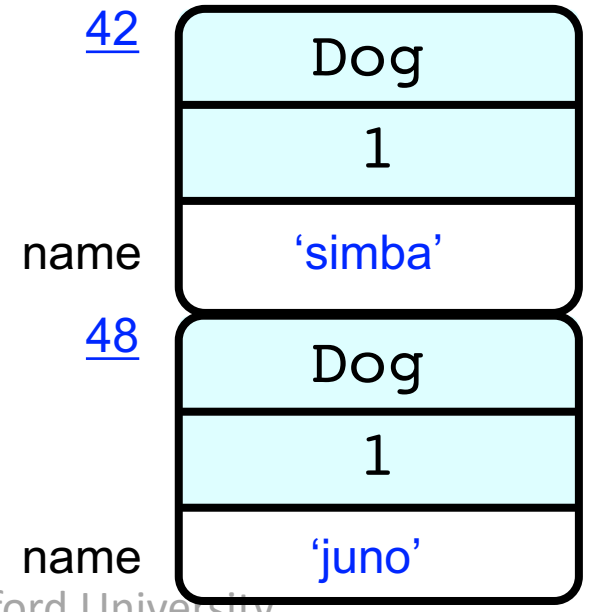




What does this do?

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

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

    print(first)
    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