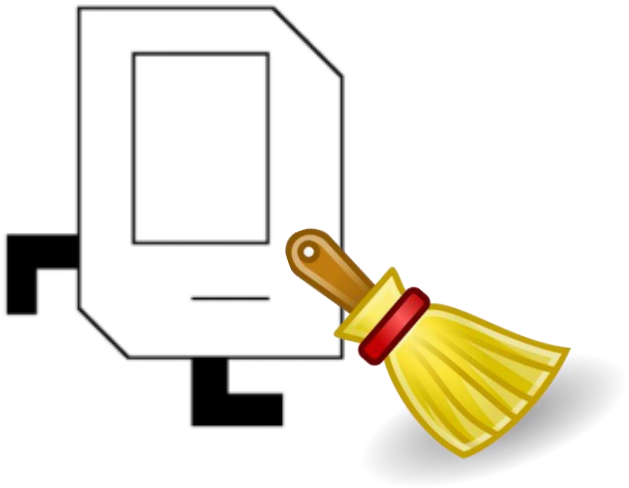
Classes + Memory

CS106A, Stanford University

Housekeeping



- Survey about Embedded EthiCS
  - https://forms.gle/GB9LzheQaUTfETMaA
  - Or, use QR Code → → →

1-minute survey on Embedded EthiCS @ Stanford

Results will be reported to & used to improve the Embedded Ethics program

And now… a song about tuples!

Learning Goals

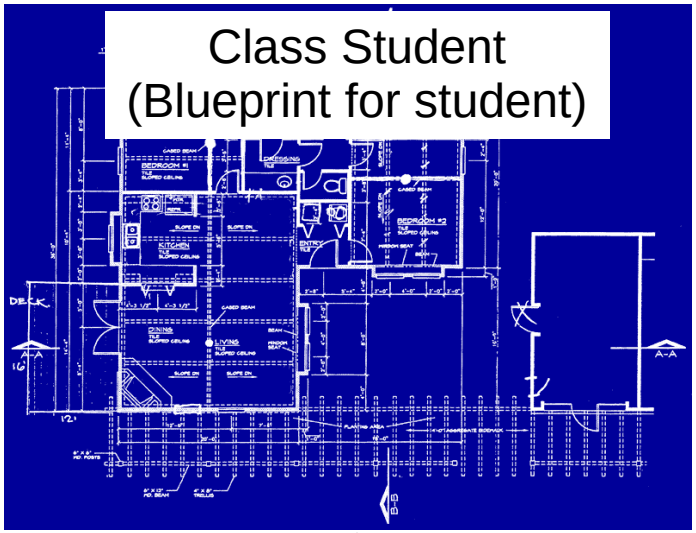
1. More practice with classes
2. See how to trace memory

Review:
classes and objects

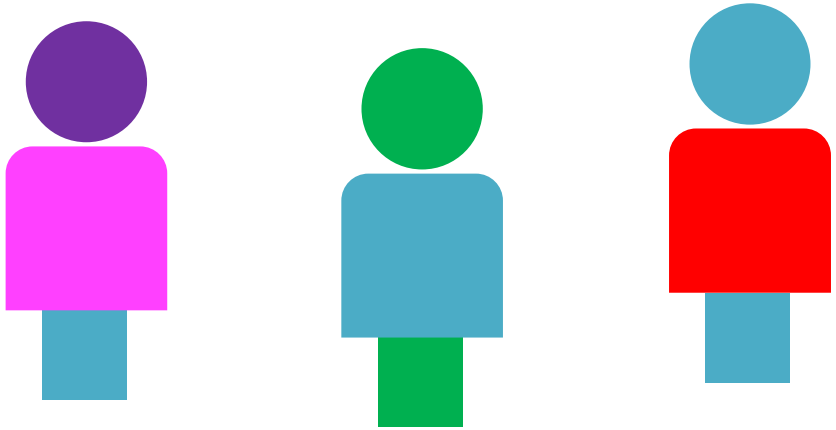
Classes are like blueprints

class: A template for a new type of variable

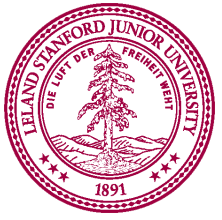
A blueprint is a helpful analogy



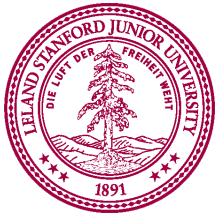
When defining a new variable type you make a blueprint



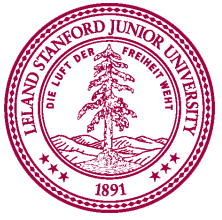
Student *instances*
(Objects)



Classes define new variable *types*



Classes help decompose your program across files



Classes Can Include Three Things

- Constructor
  - Method (function) called when a new object is being created
- Methods
  - Functions that you can call on an instance (object) of that class
- Instance variables
  - Variables inside each object of that class
  - Referred to using `self.`***variable_name***

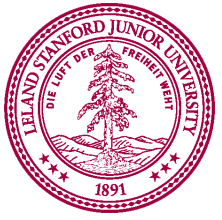
Classes Review

dog.py

```
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1
```

dogworld.py

```
from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

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



Classes Review

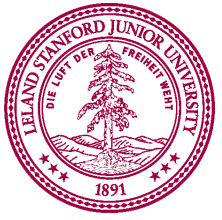
dog.py

```
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1
```

dogworld.py

```
from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

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

- Constructor is called each time we create a new object



Classes Review

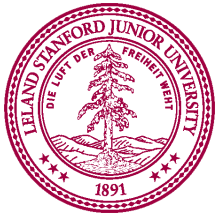
dog.py

```
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1
```

dogworld.py

```
from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

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

- Instance variables are stored inside each object
- Each object has its own version of the instance variables



Classes Review

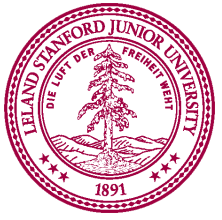
dog.py

```
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1
```

dogworld.py

```
from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

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

- Methods are functions that can be called on a particular object



Classes Review

dog.py

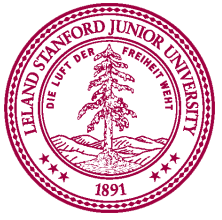
```
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1
```

dogworld.py

```
from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

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

- When authoring a class, self means:

  "the instance (aka object) I am currently working with"



Classes Review

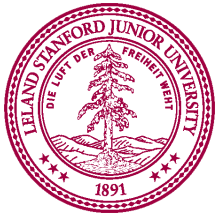
dog.py

```
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1
```

dogworld.py

```
from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

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

```
A new dog is born
```



Classes Review

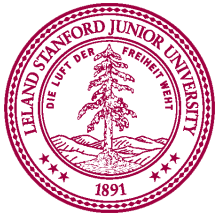
dog.py

```
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1
```

dogworld.py

```
from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

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

```
A new dog is born
A new dog is born
```



Classes Review

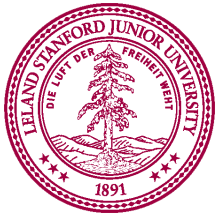
dog.py

```
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1
```

dogworld.py

```
from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

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

```
A new dog is born
A new dog is born
yip
```



Classes Review

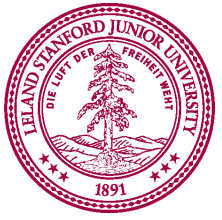
dog.py

```
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1
```

dogworld.py

```
from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

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

```
A new dog is born
A new dog is born
yip
woof
```



Classes Review

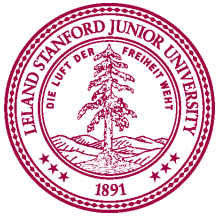
dog.py

```
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1
```

dogworld.py

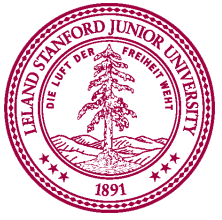
```
from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

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

```
A new dog is born
A new dog is born
yip
woof
yip
```



`.__dict__`

- Instance variables in an object are stored in internal `.__dict__` variable



Classes Review

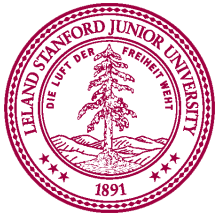
dog.py

```
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1
```

dogworld.py

```
from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

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

```
A new dog is born
A new dog is born
yip
woof
yip
simba {'times_barked': 2, 'breed': 'pomeranian'}
```



Classes Review

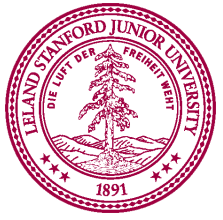
dog.py

```
class Dog:

    def __init__(self, breed):
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1
```

dogworld.py

```
from dog import Dog

def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

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

```
A new dog is born
A new dog is born
yip
woof
yip
simba {'times_barked': 2, 'breed': 'pomeranian'}
juno {'times_barked': 1, 'breed': 'great dane'}
```



Recall Functions?

Coder: **Function Author**

Writes helper functions others can use

Coder: **Function Caller**

Uses helper functions

Classes also split up the work!

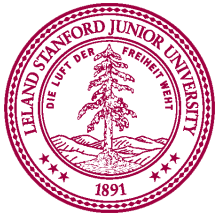
Coder: **Class Author**

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



dog.py

```
"""
File: dog.py
------------
Defines a Dog class.
"""

class Dog:

    def __init__(self, breed):
        """
        Constructor for Dog.  We set the breed of the dog when
        the dog object is created.
        """
        print('A new dog is born')
        self.times_barked = 0
        self.breed = breed

    def bark(self):
        """
        The kind of bark the dog makes depends on its breed.
        """
        if self.breed == 'pomeranian':
            print('yip')
        else:
            print('woof')
        self.times_barked += 1

```

dogworld.py

```
"""
File: dogworld.py
-----------------
Defines a Dog class.
"""

from dog import Dog


def main():
    simba = Dog('pomeranian')
    juno = Dog('great dane')

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

    print('simba', simba.__dict__)
    print('juno',  juno.__dict__)


if __name__ == '__main__':
    main()

```

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

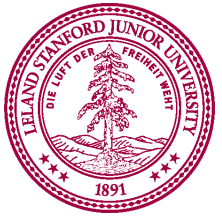
Class Client: Uses the new variable type to solve problems (often from main).

Next step in writing large programs:
Better understand memory

You are now ready…

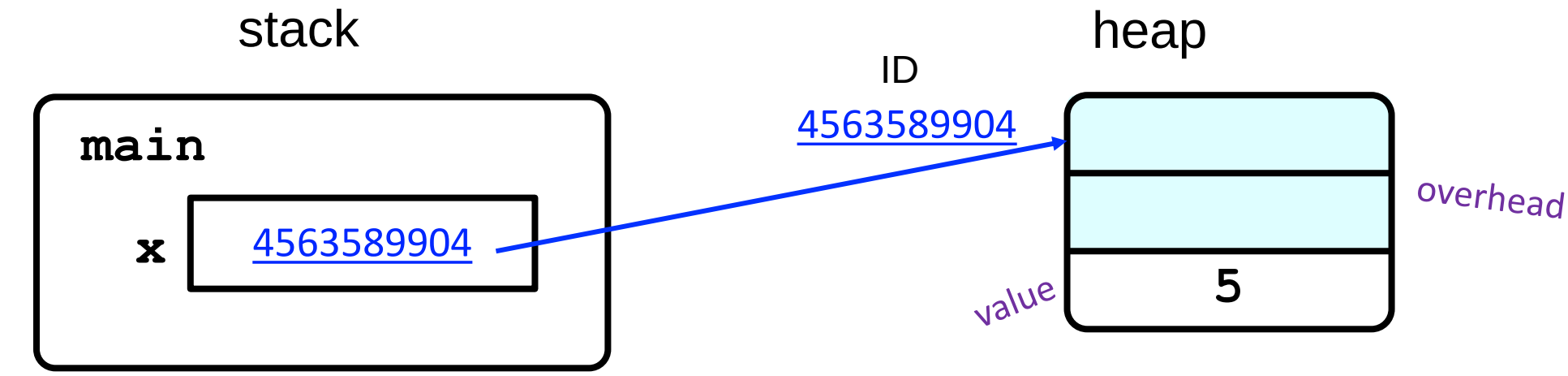
What does this do?

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



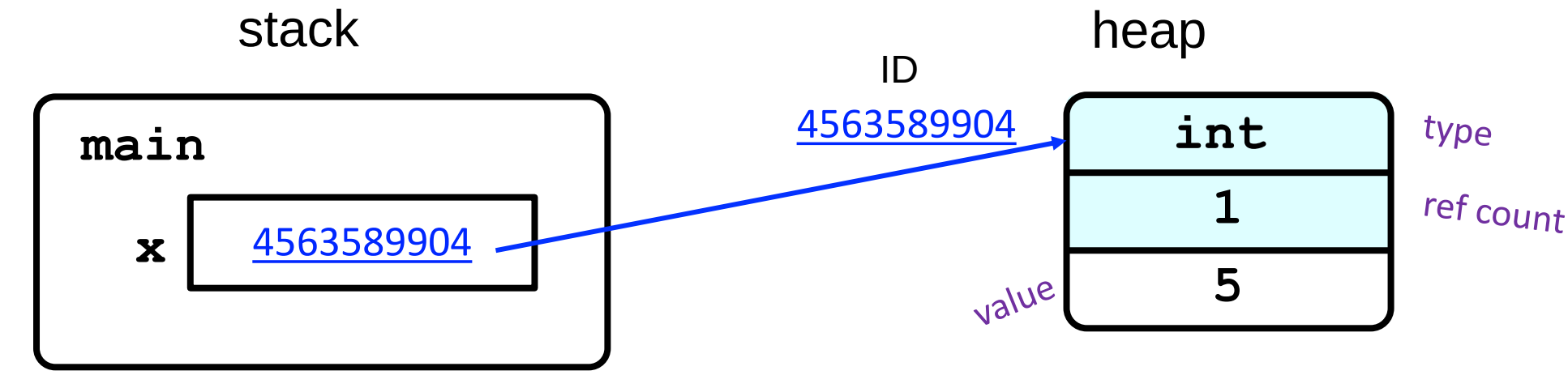
What does this do?

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



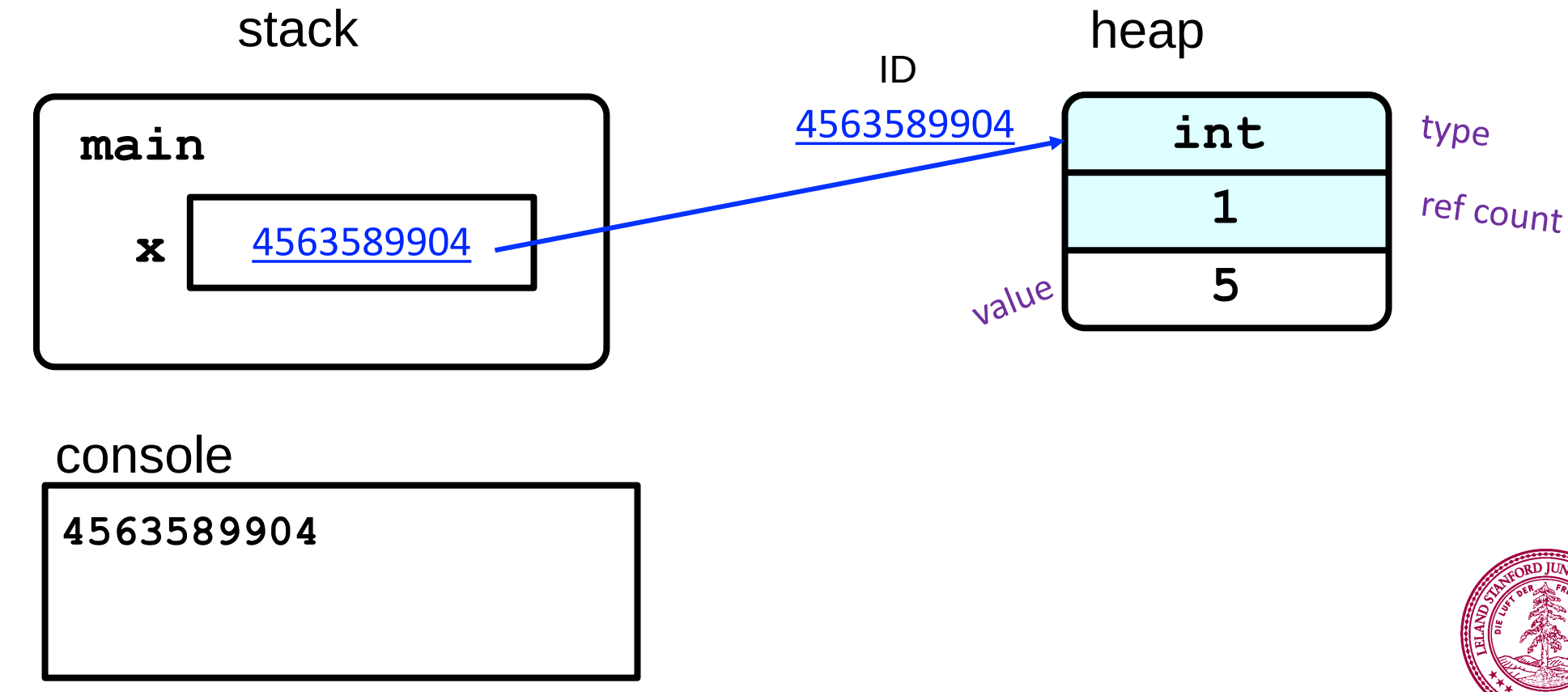
What does this do?

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



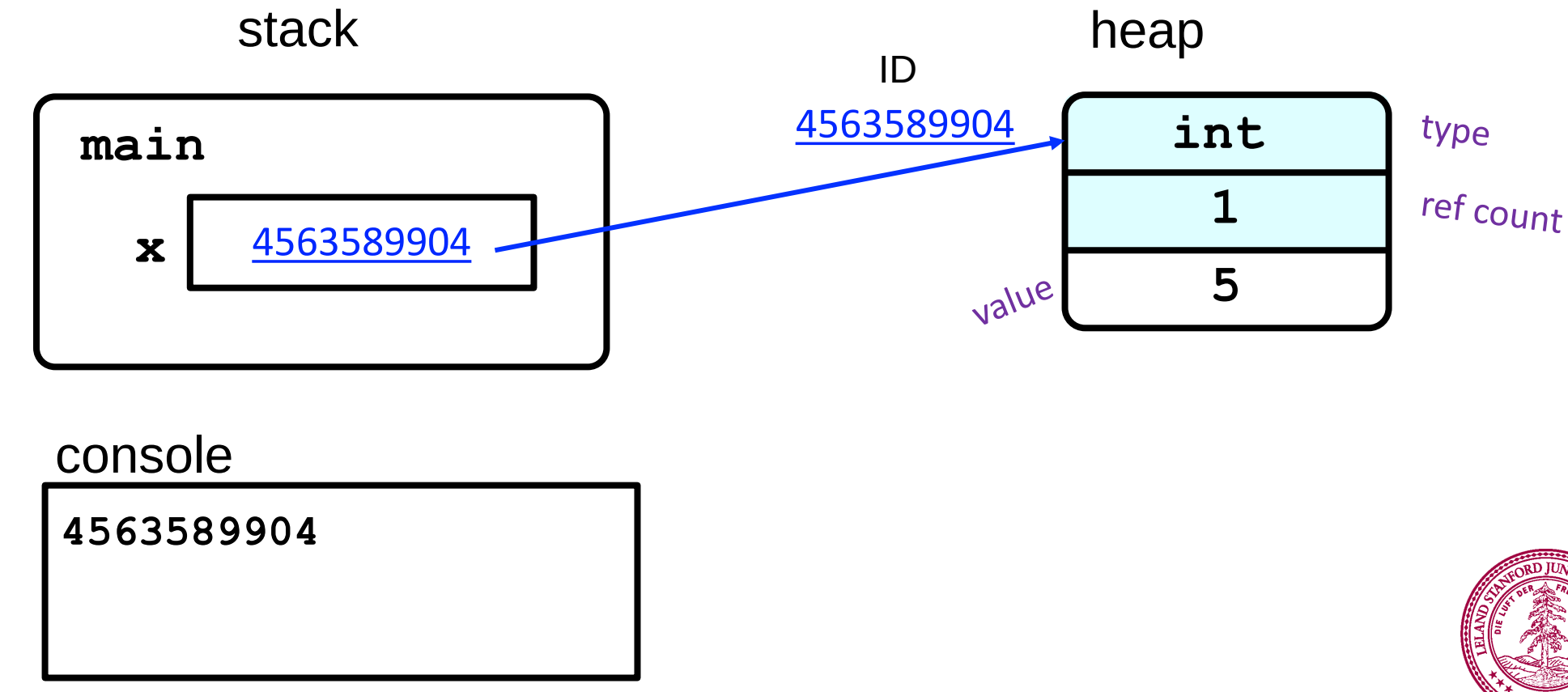
What does this do?

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



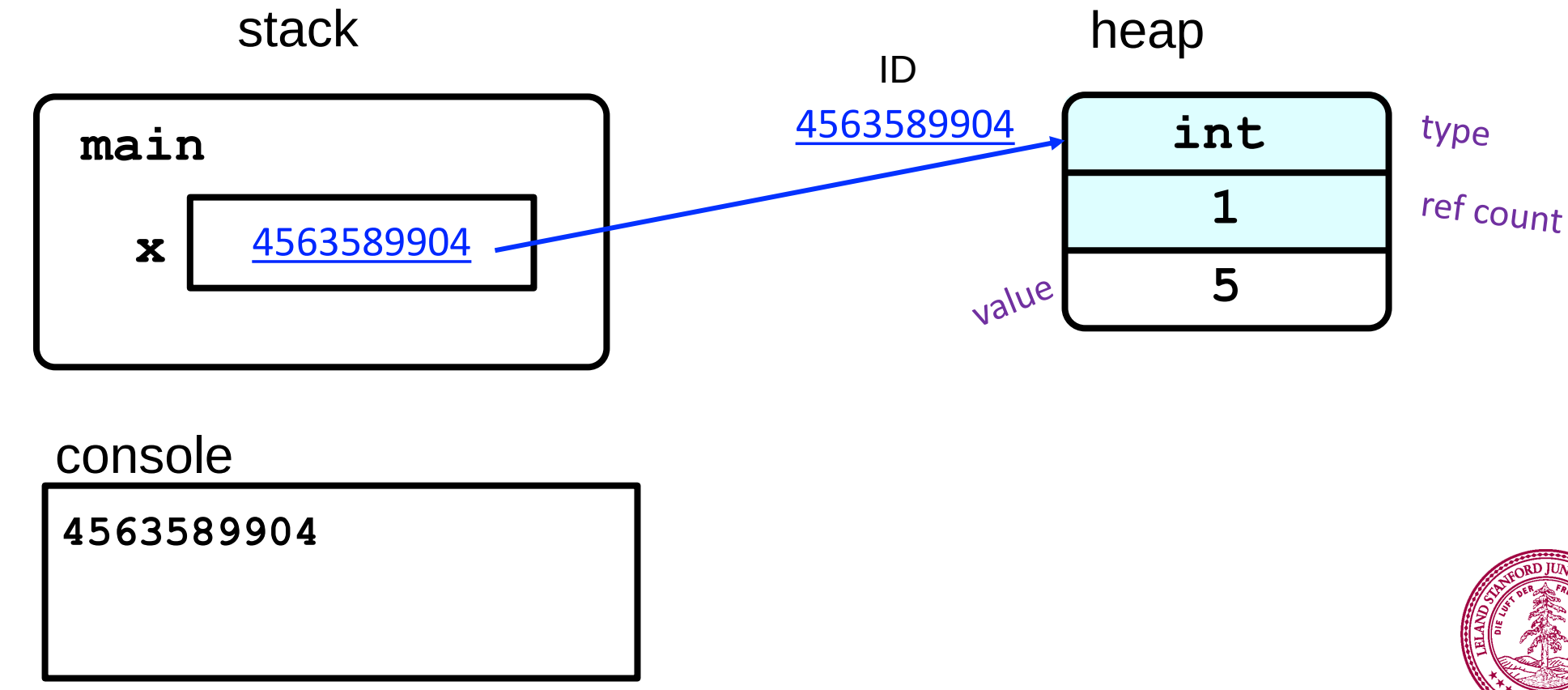
What does this do?

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



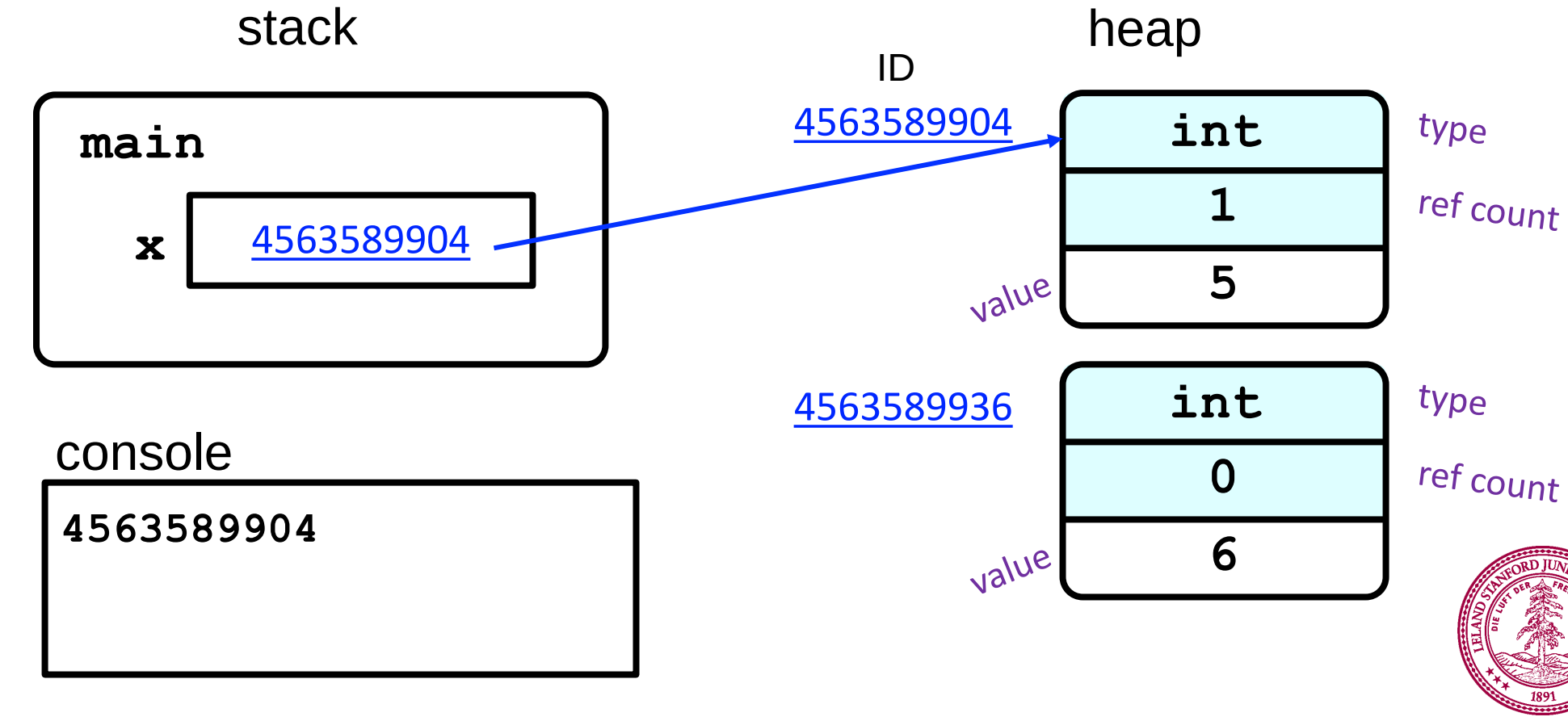
What does this do?

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



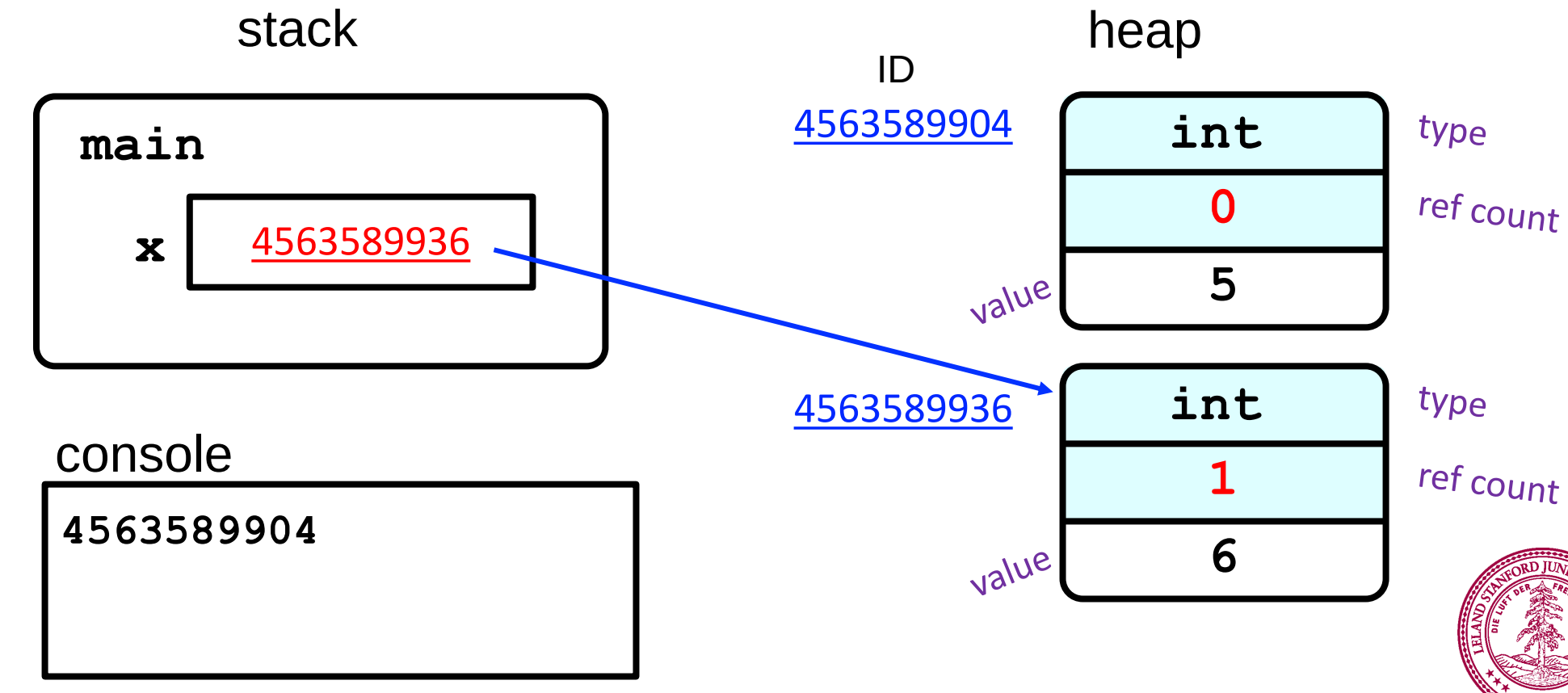
What does this do?

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



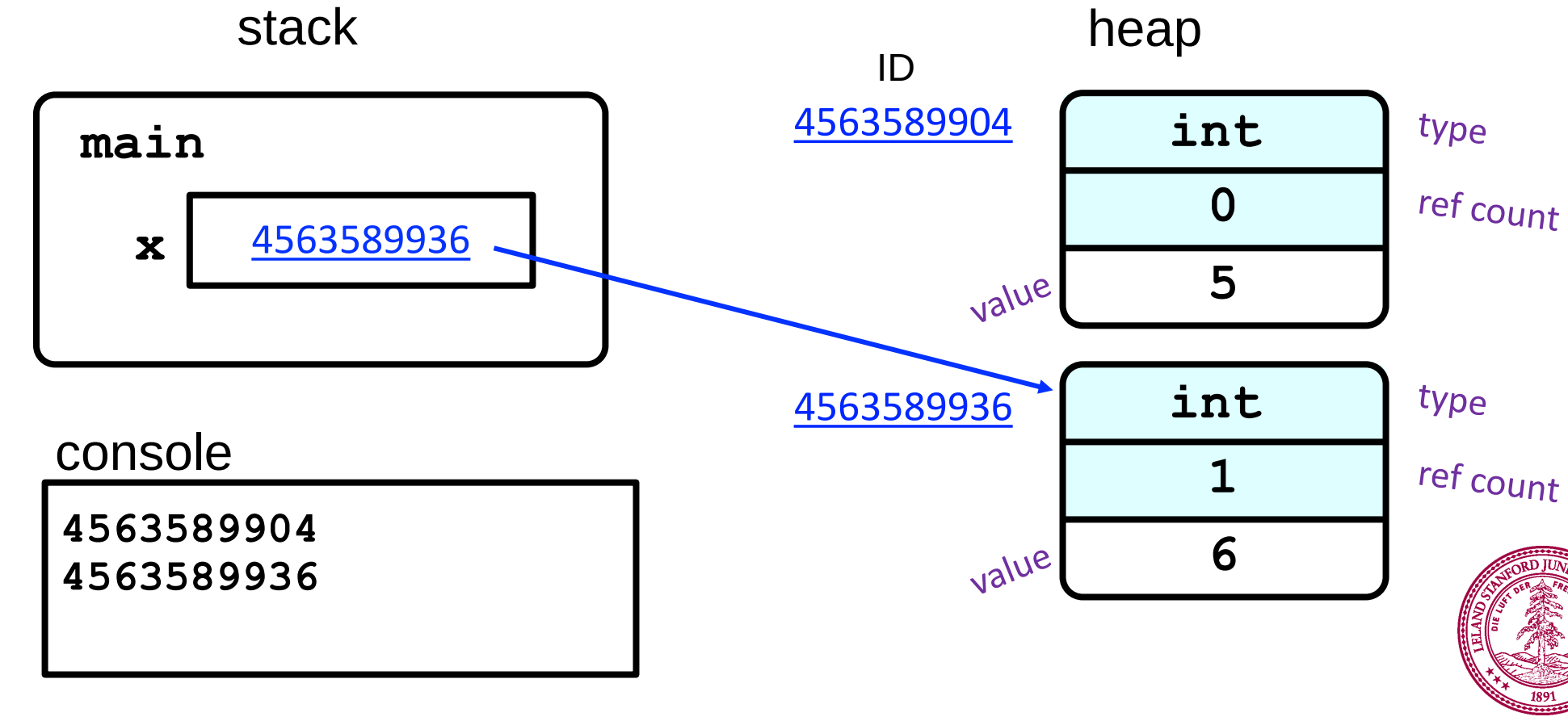
What does this do?

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

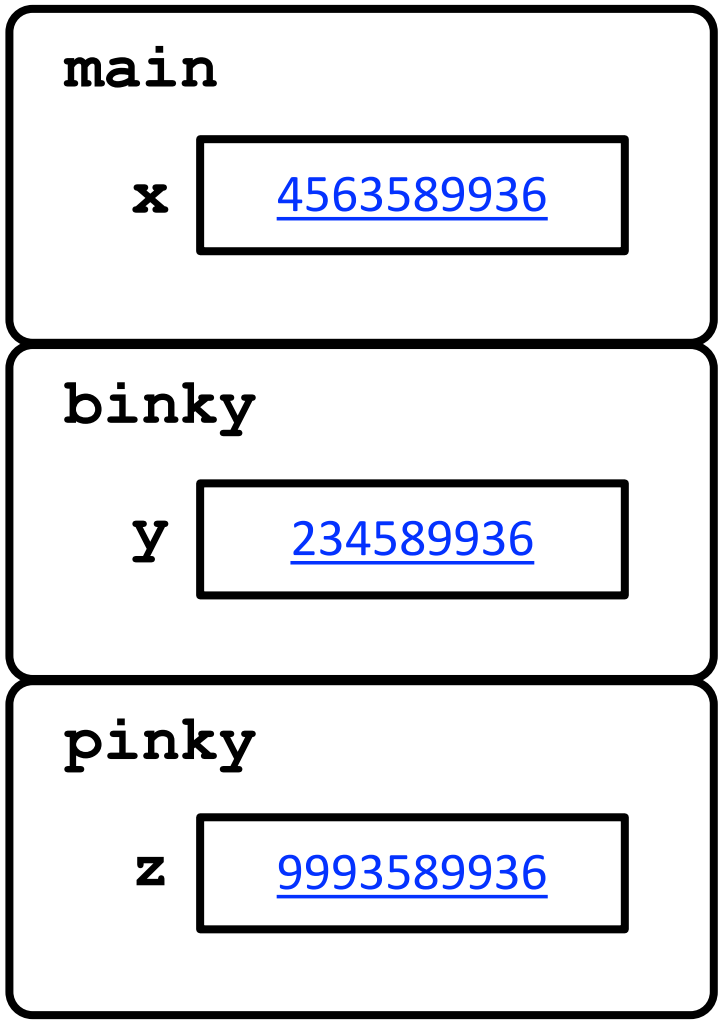


What does this do?

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



The stack

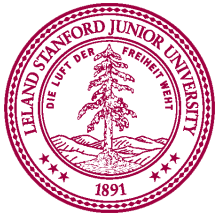


Each time a function is called, a new frame of memory is created.

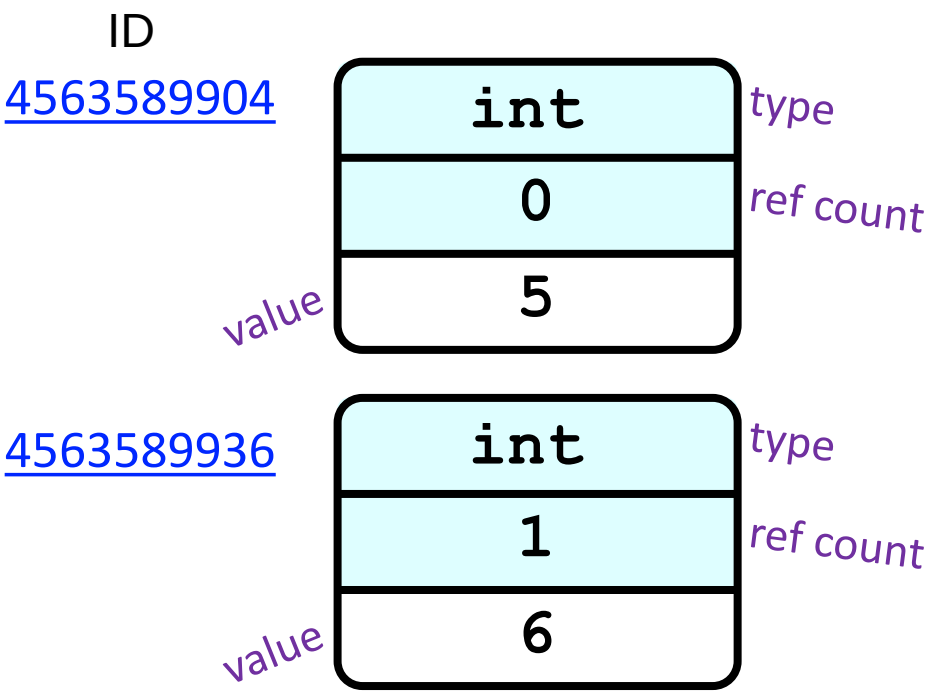
Each frame has space for all the local variables declared in the function, and parameters

Each variable has a reference which is like a URL

When a function returns, its frame is destroyed.



The heap

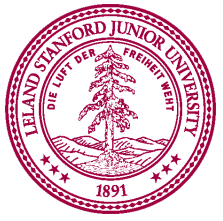


Where values are stored

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.



What does this do?

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

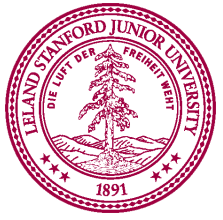
When a variable is “used”
you are accessing its **value**

What does this do?

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

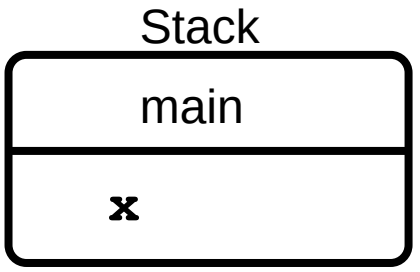
When a variable is “assigned” via *binding* you are changing its **reference**

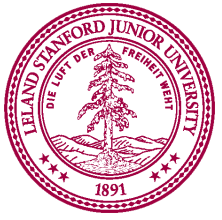
You know a variable is being assigned to if it is on the left hand side of an **=** sign



What does this do?

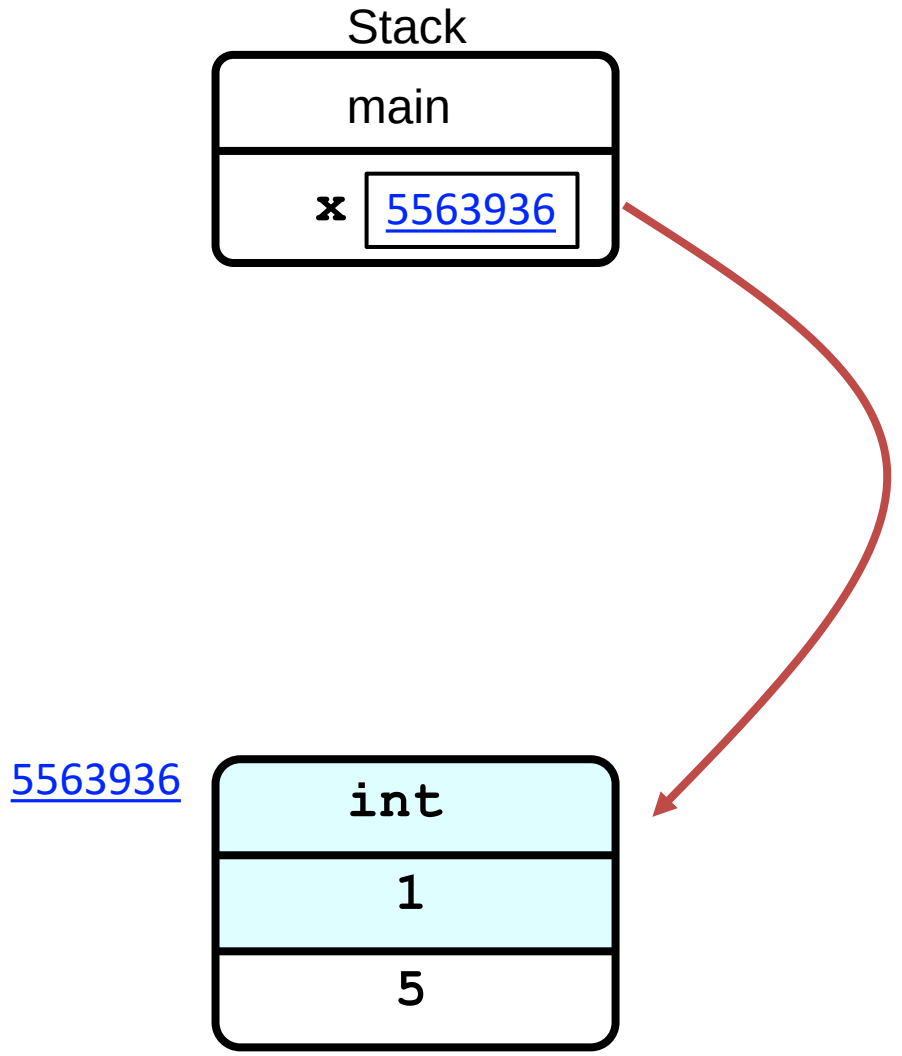
```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

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





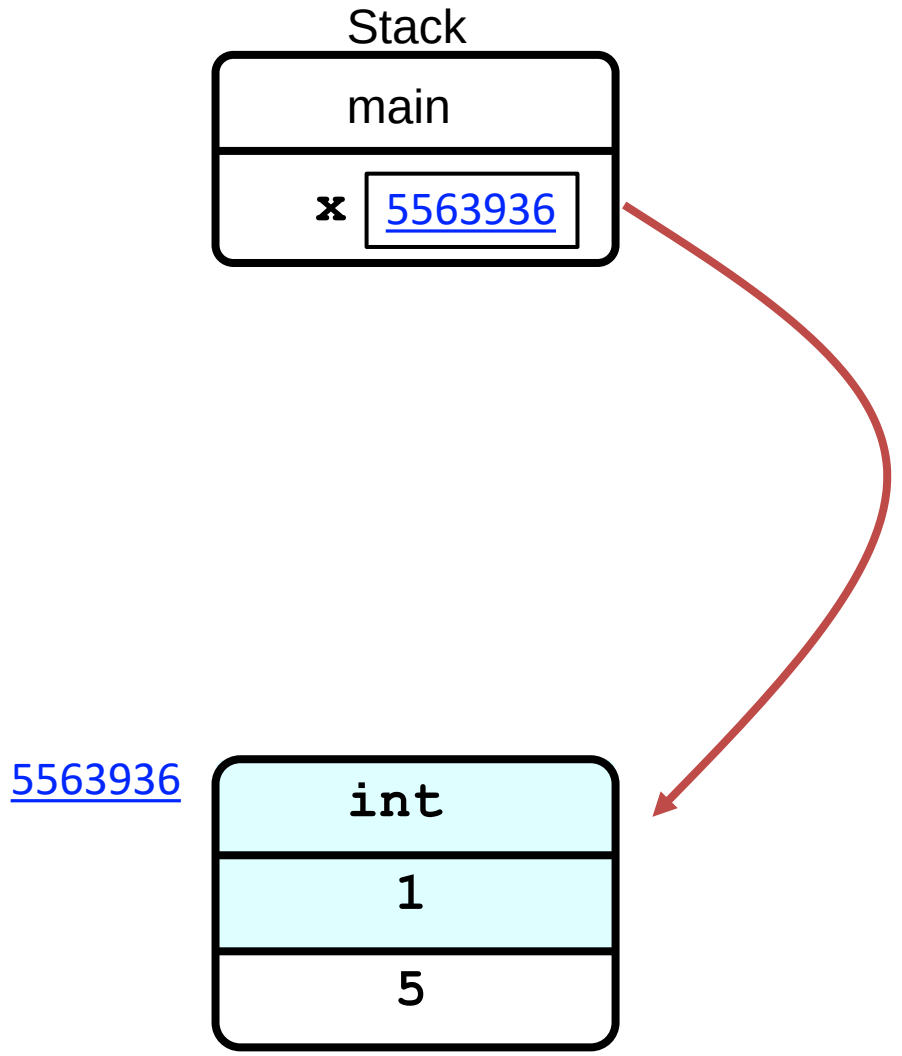
What does this do?

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

def binky(y):
    pinky(y)

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



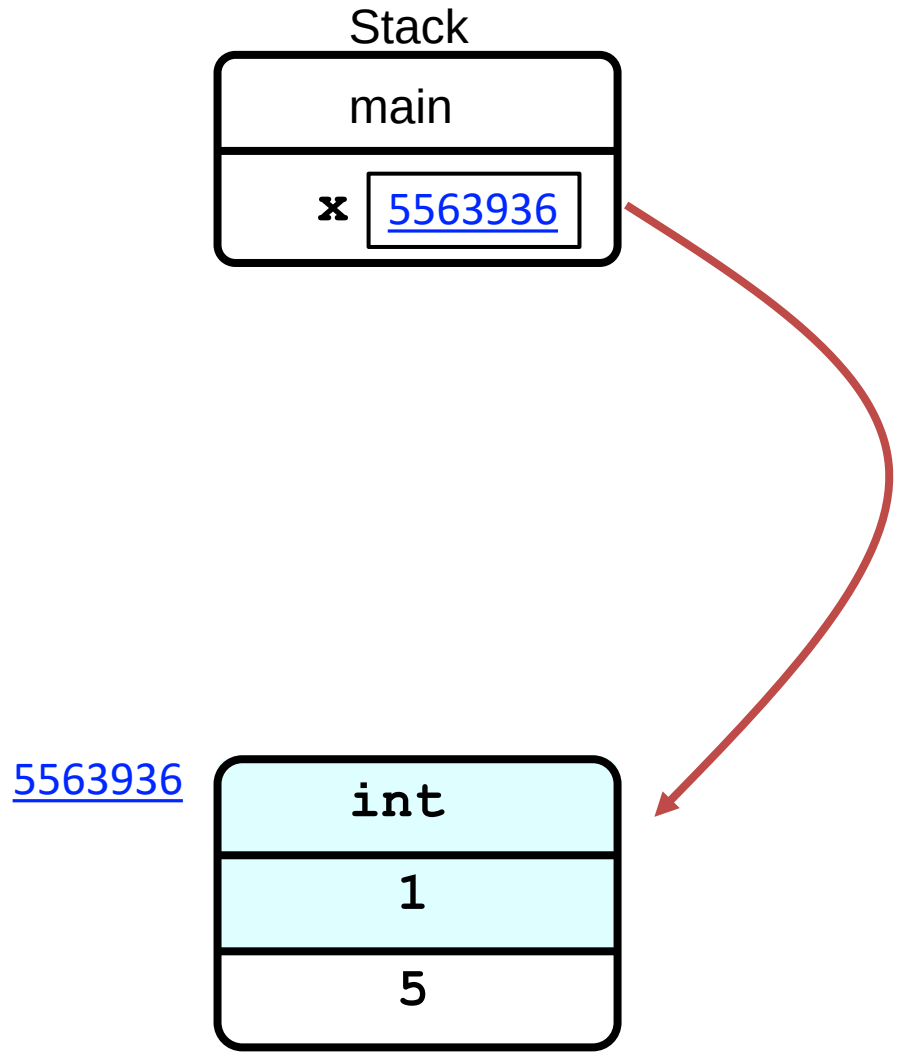
What does this do?

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

def binky(y):
    pinky(y)

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



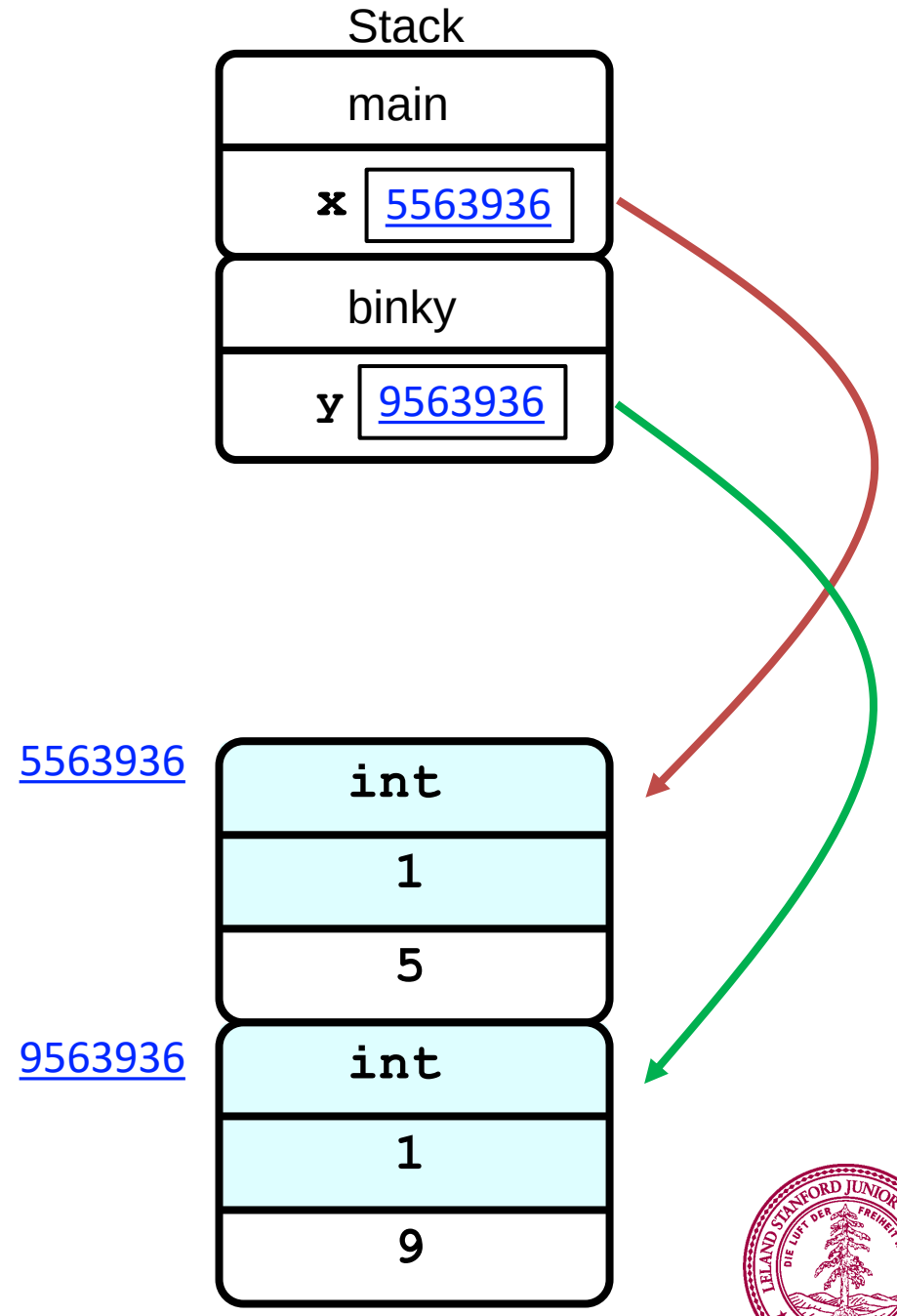
What does this do?

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

def binky(y):
    pinky(y)

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



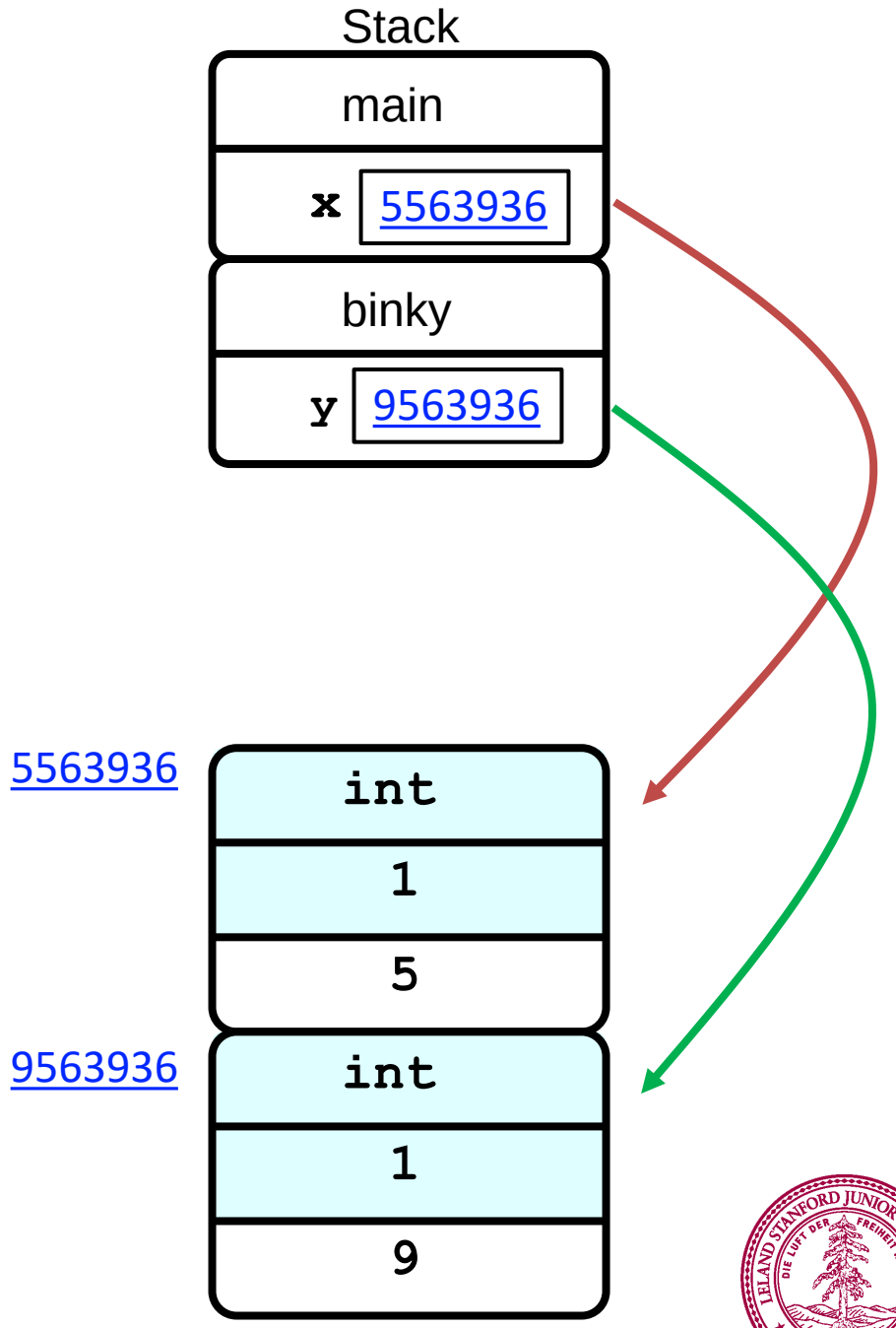
What does this do?

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

def binky(y):
    pinky(y)

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



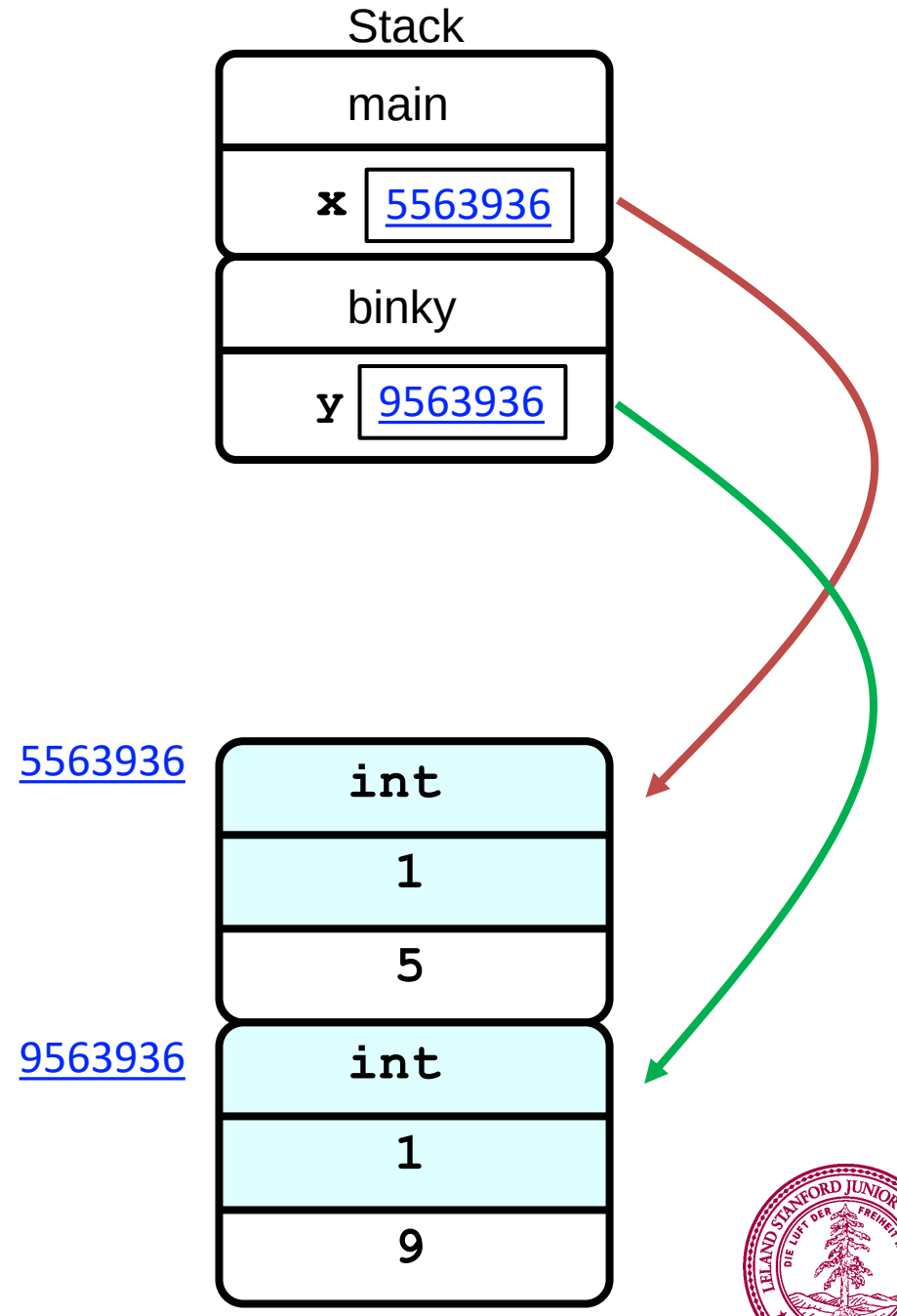
What does this do?

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

def binky(y):
    pinky(y)

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



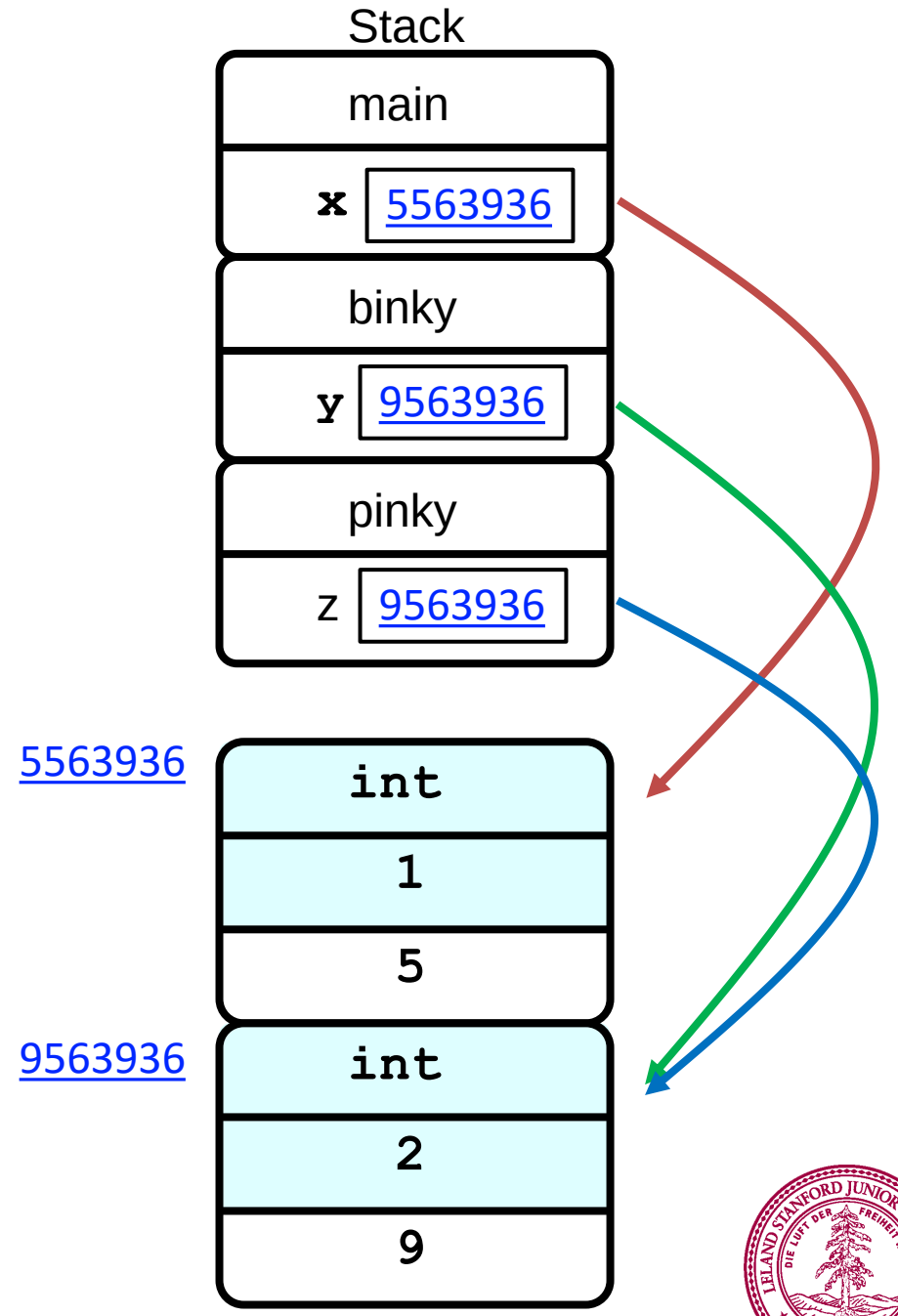
What does this do?

```
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)
```

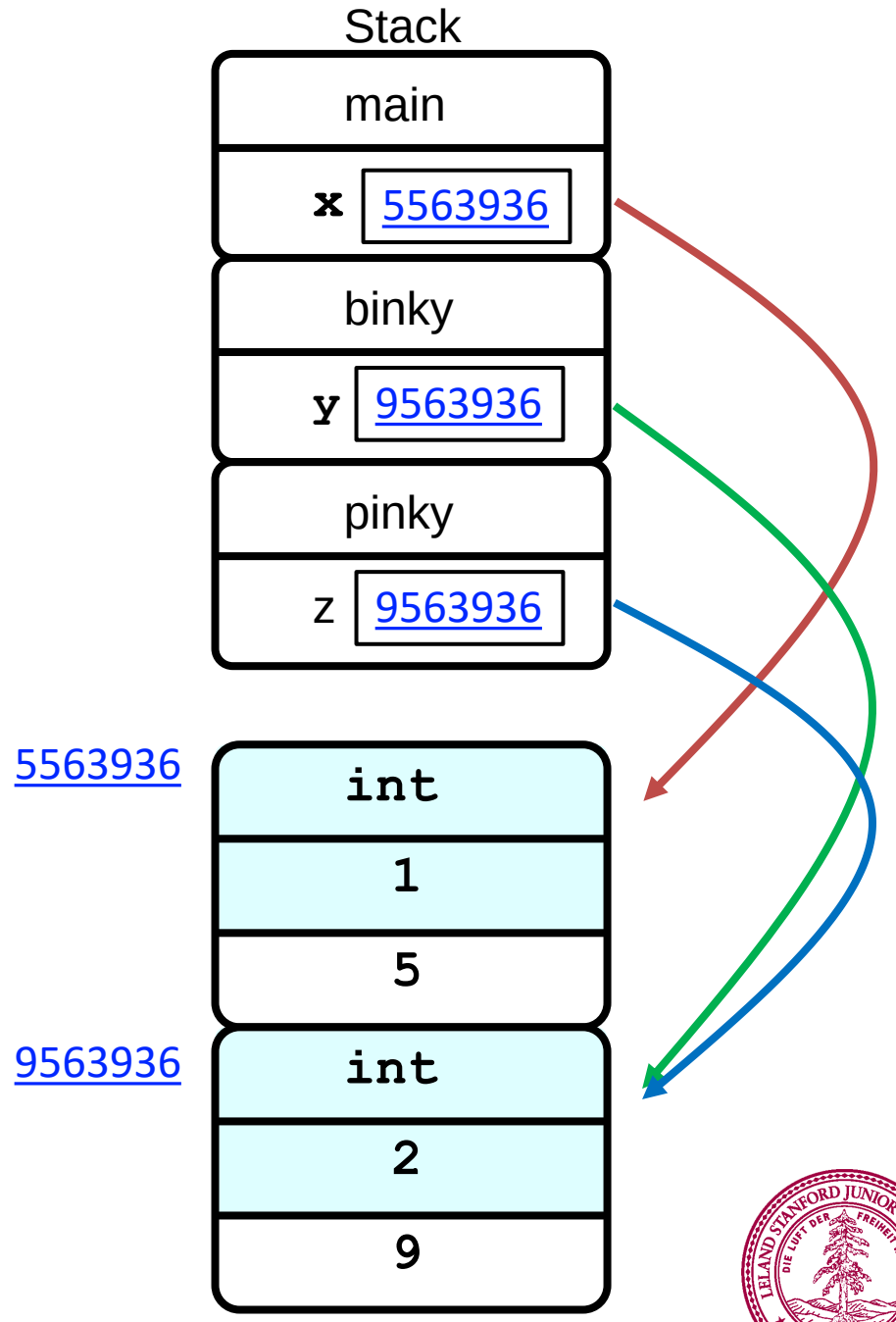


What does this do?

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

def binky(y):
    pinky(y)

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

console

```
9
```

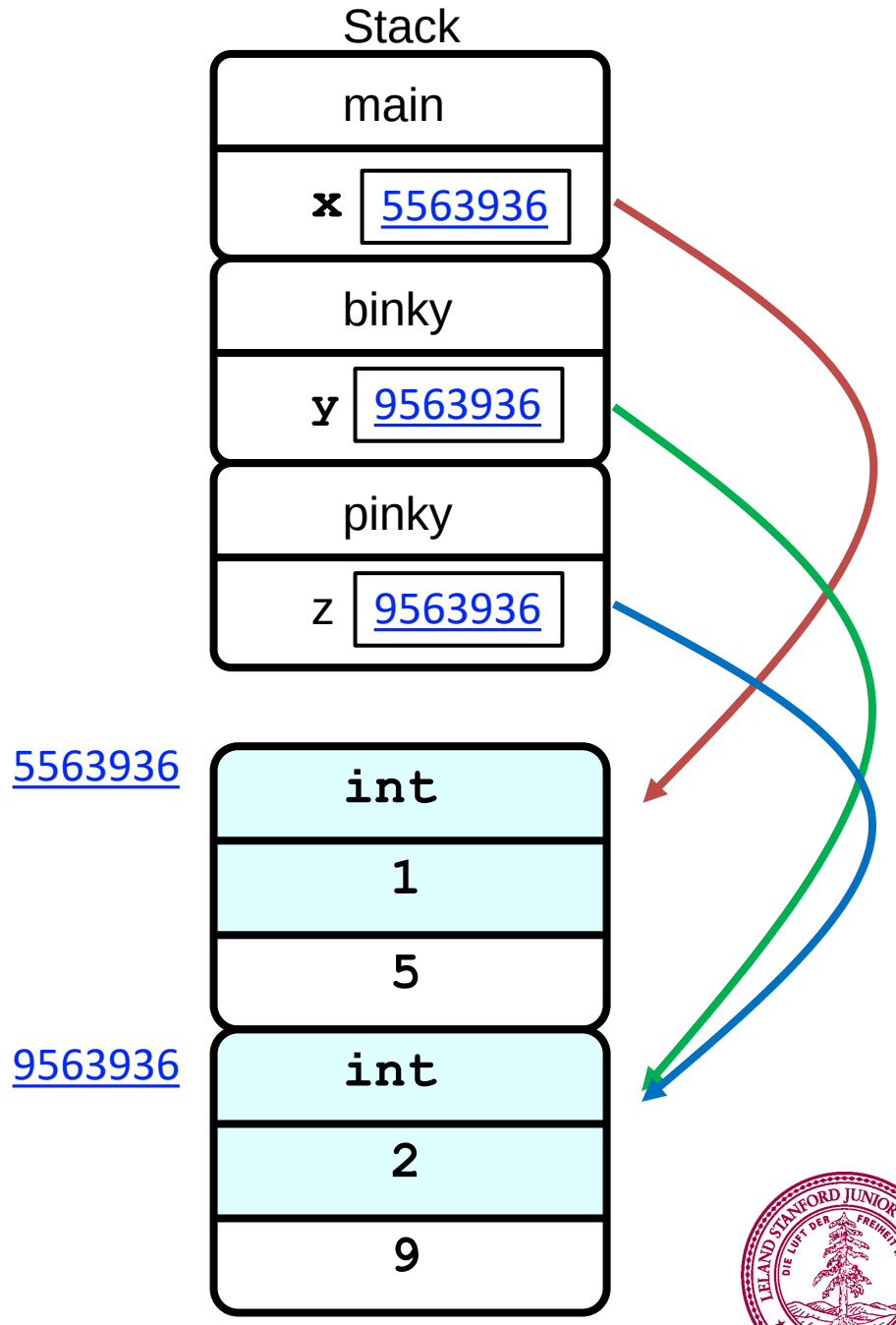


What does this do?

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

def binky(y):
    pinky(y)

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

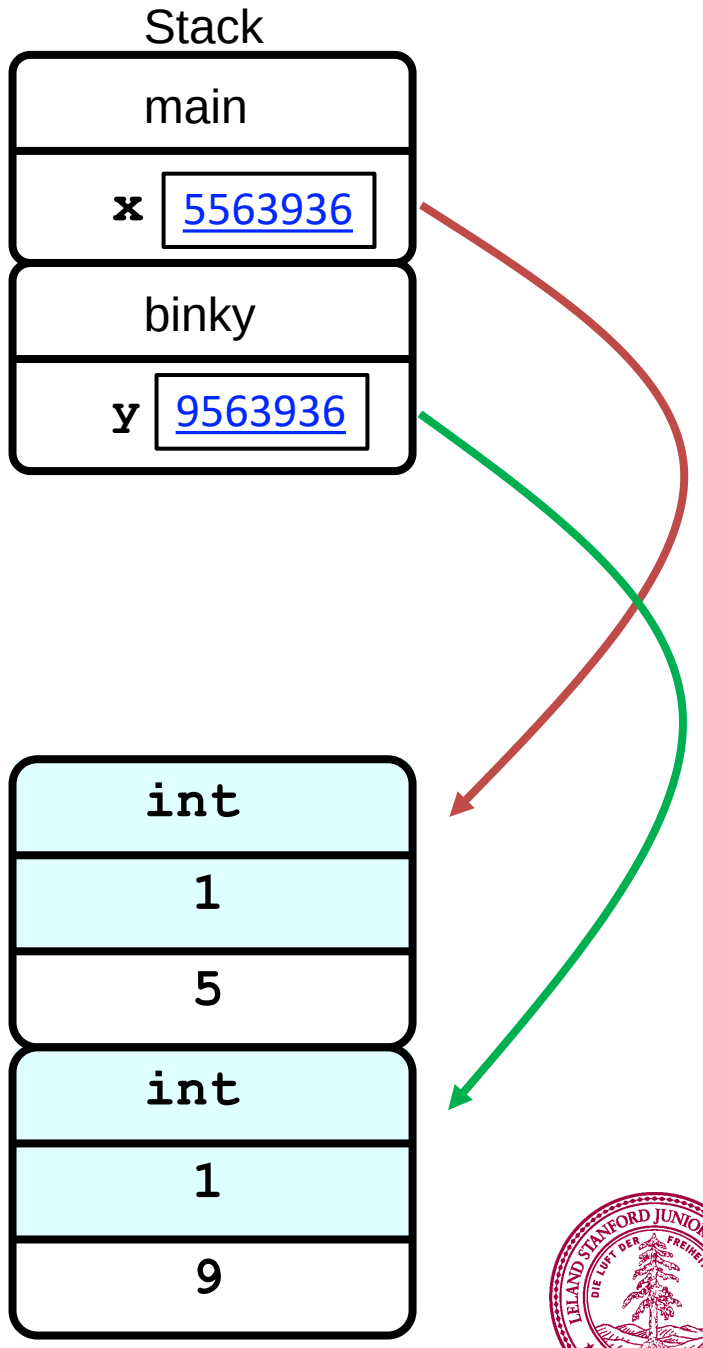
console

```
9
```



What does this do?

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

def binky(y):
    pinky(y)

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

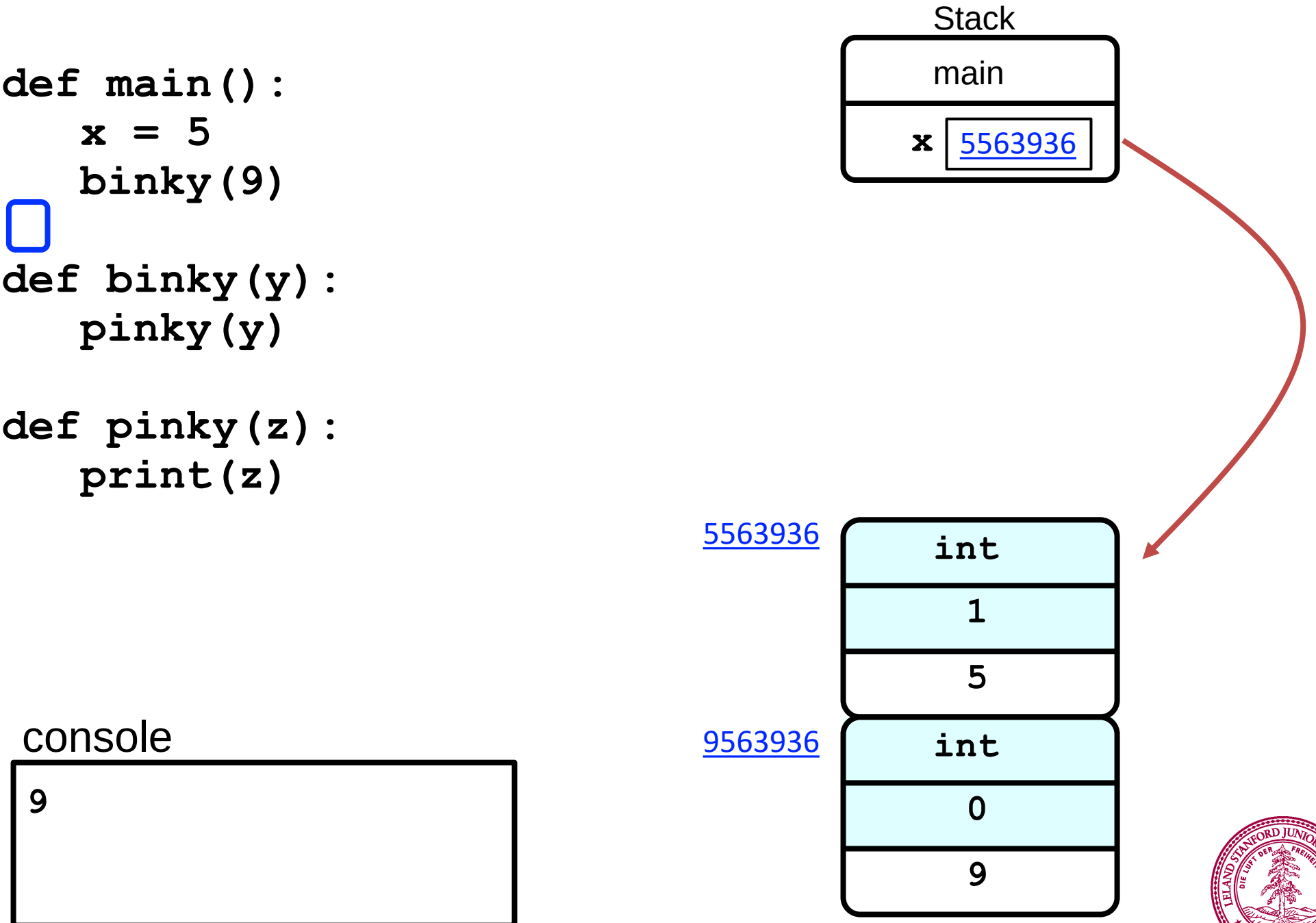


console

```
9
```

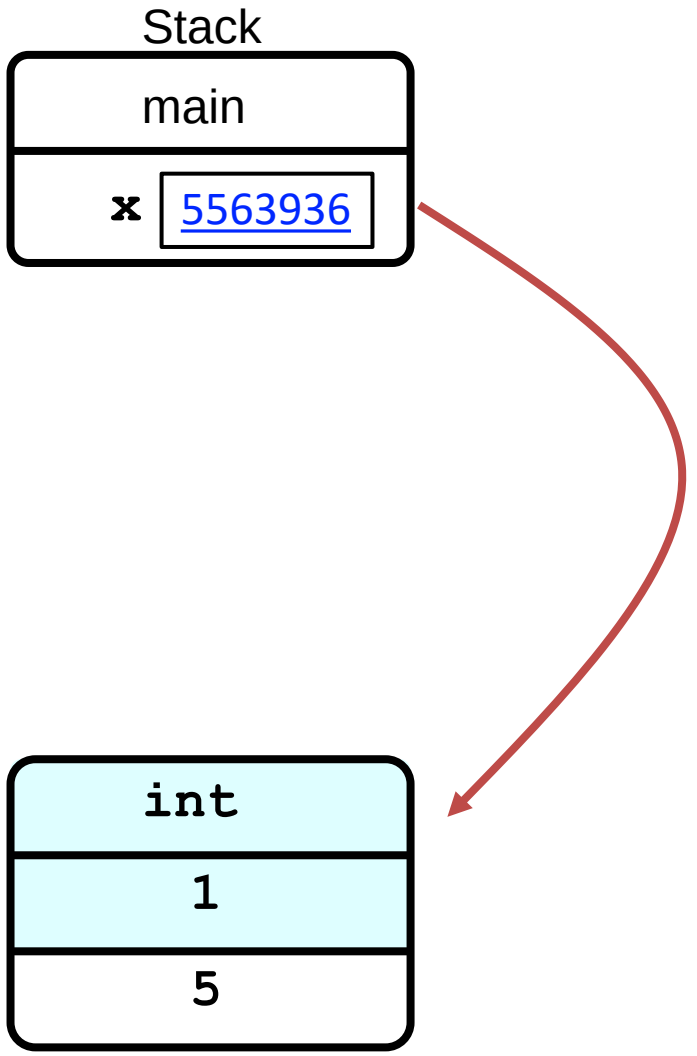
What does this do?

```
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)
```



console

```
9
```

What does this do?

Stack

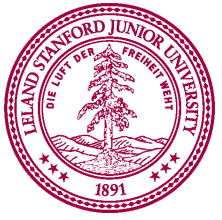
```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

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

5563936

int
0
5

console

```
9
```



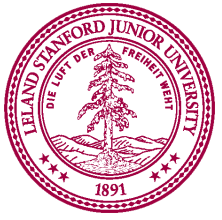
What does this do?

Stack

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

def binky(y):
    pinky(y)

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

console

```
9
```



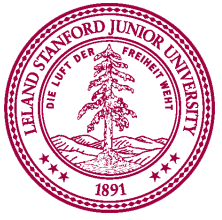
What does this do?

Stack

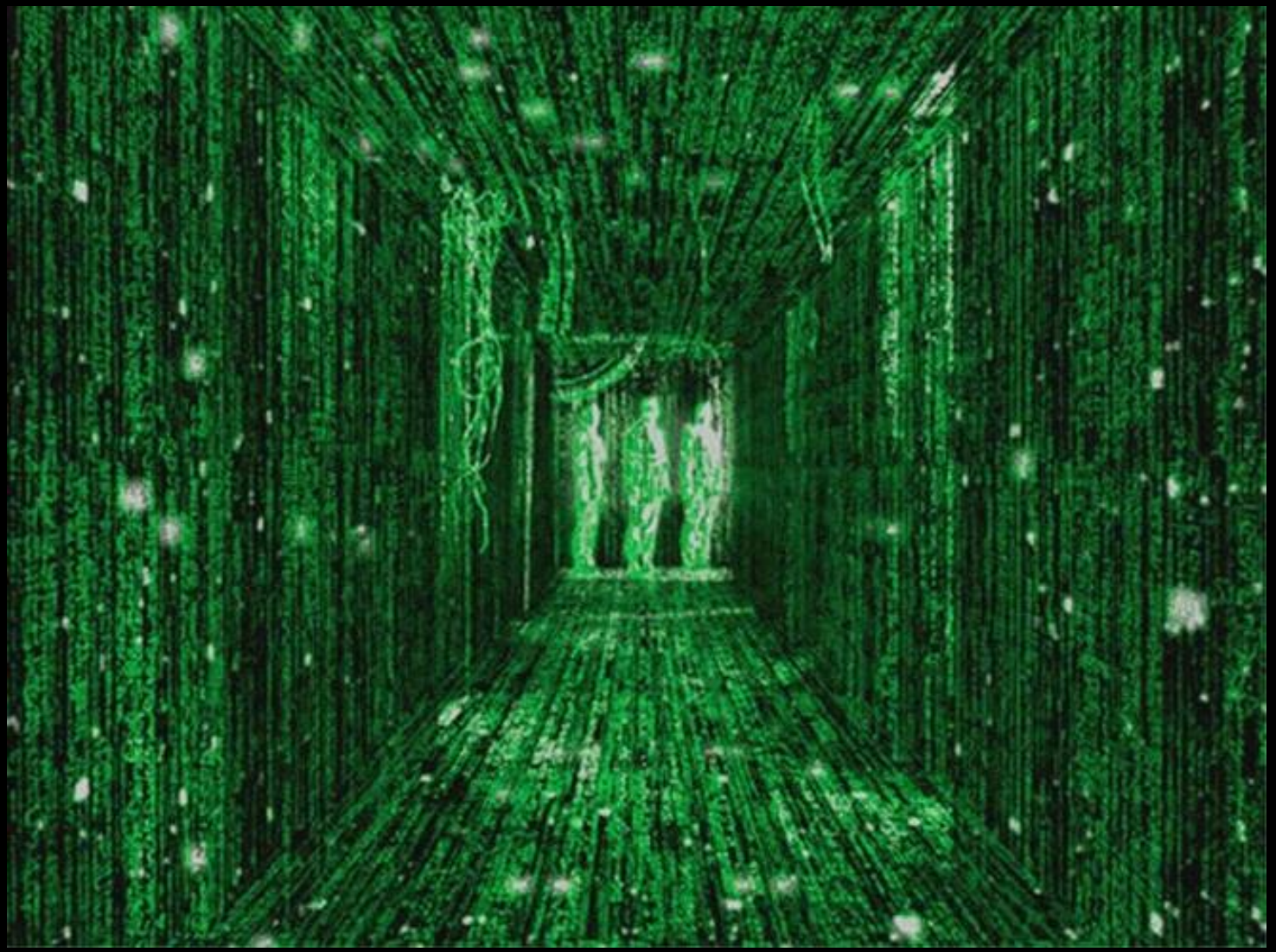
```
def main():
    x = 5
    binky(9)

def binky(y):
    pinky(y)

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

console

```
9
```



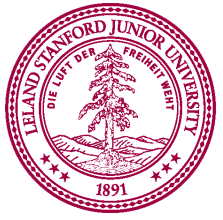
Whoa!



Bring it!

```
def main():
    x = ['a', 'b', 'c']
    print(x)
    update_in_list(x)
    print(x)
    update_list(x)
    print(x)

def update_in_list(x):
    x[0] = 'z'

def update_list(x):
    x = ['m', 'n', 'o']

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

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



Learning Goals

1. More practice with classes
2. See how to trace memory