Variables, References and Mutation

Aka, By Far the Hardest Topic from CSE8A, and 8B, and 11!

What is printed?

```
def silly(a, b):
    a = a + 1
    b = b + 2

a = 3
b = 6
silly(b,a)
print(a,b)
```

A. 3 6
B. 4 8
C. 7 5
D. 5 7
E. Something else

What is printed?

```
def silly(a, b):
    a = a + 1
    b = b + 2
    print(a,b)

a = 3
b = 6
silly(b,a)
```

A. 3 6
B. 4 8
C. 7 5
D. 5 7
E. Something else

What is printed?

```
def silly(a, b):
    a = a + 1
    b = b + 2
    return (a,b)

a = 3
b = 6
silly(b,a)
print(a,b)
```

A. 3 6
B. 4 8
C. 7 5
D. 5 7
E. Something else

What is printed?

```
def silly(a, b):
    a = a + 1
    b = b + 2
    return (a,b)

a = 3
b = 6
(a,b) = silly(b,a)
print(a,b)
```

Passing parameters to functions

What is shown?

```
def silly(im):
    for x in range(im.size[0]):
        im.putpixel( (x,0), (0,0,0) )
    return im

pic = Image.open('homer.jpg')
pic = silly(pic)
pic.show()
```

Passing parameters to functions

What is shown?

```
def silly(im):
    for x in range(im.size[0]):
        im.putpixel( (x,0), (0,0,0) )
    return im

pic = Image.open('homer.jpg')
silly(pic)
pic.show()
```

What happens now?

A. You get an error
B. An empty image is shown
C. The original image is shown
D. The modified image is shown
E. Something else

When you open a picture …

```
pic = Image.open('homer.jpg')
```

homer.jpg
A file on your computer

On your computer's hard drive

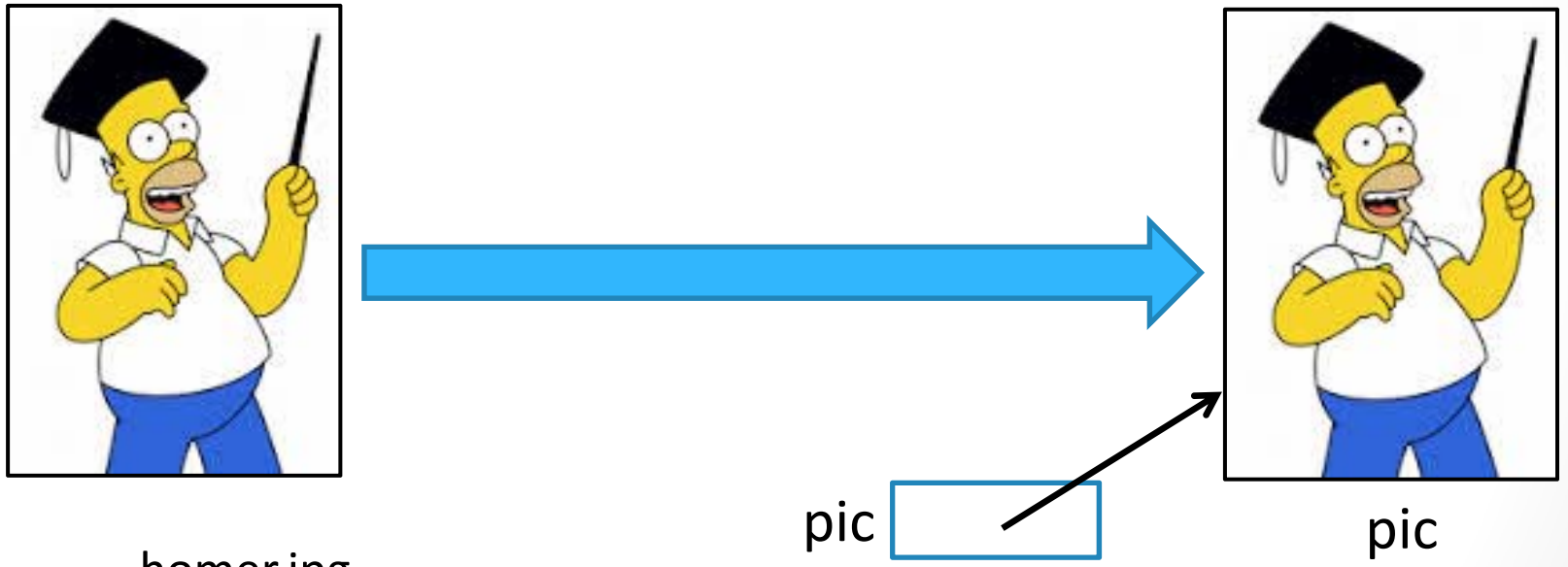
pic

A Python Picture Object

PIL's representation
In your computer's memory

When you open a picture …

```
pic = Image.open('homer.jpg')
```



homer.jpg
A file on your computer

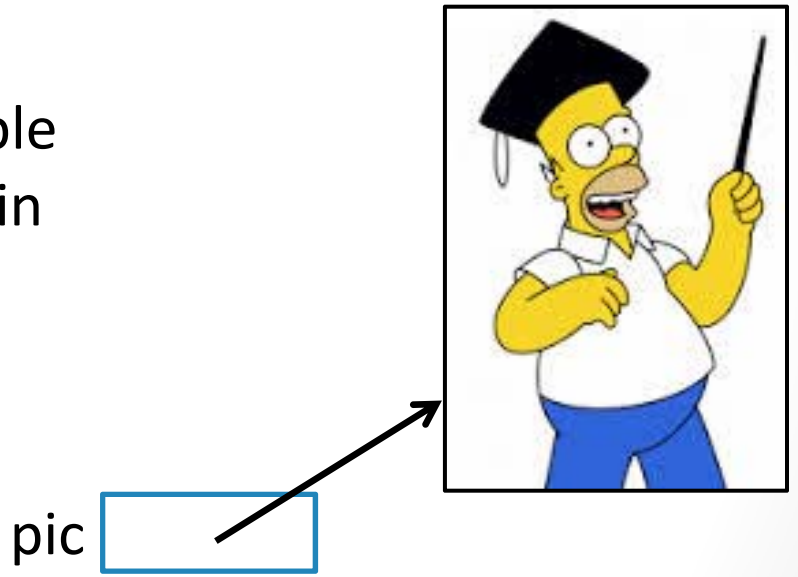
On your computer's hard drive

A Python Picture Object

PIL's representation
In your computer's memory

Objects in Python

The value of an object variable in Python (i.e., the **arrow** in the diagram) is a number that represents the location of that object in your computer's memory. The variable stores a *reference* to the object in memory.



A Python Picture Object

PIL's representation
In your computer's memory

* The fine print: technically ALL data in Python is an object, so all variables are object variables, but we will only talk about references when we talk about mutable objects. More on this shortly…

Objects in Python

The value of an object variable* in Python (i.e., the **arrow** in the diagram) is a number that represents the location of that object in your computer's memory. The variable stores a *reference* to the object in memory.

NOTE: This location is NOT on the stack. It is in a different part of memory called the heap.

pic 42428428

A Python Picture Object

PIL's representation
In your computer's memory

```
pic = Image.open('homer.jpg')
pic2 = Image.open('homer.jpg')
```

2 picture objects!

homer.jpg
A file on your computer

On your computer's hard drive

pic

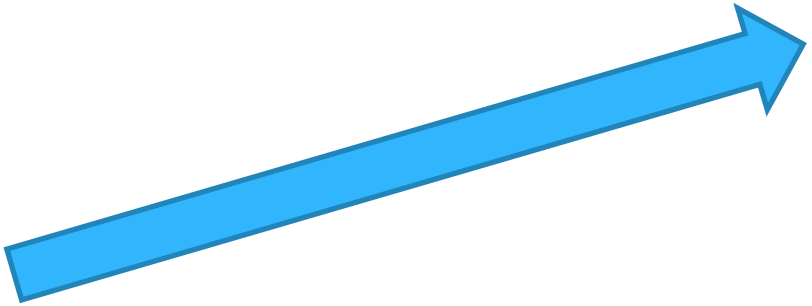
pic2

```
pic = Image.open('homer.jpg')
pic2 = Image.open('homer.jpg')
```

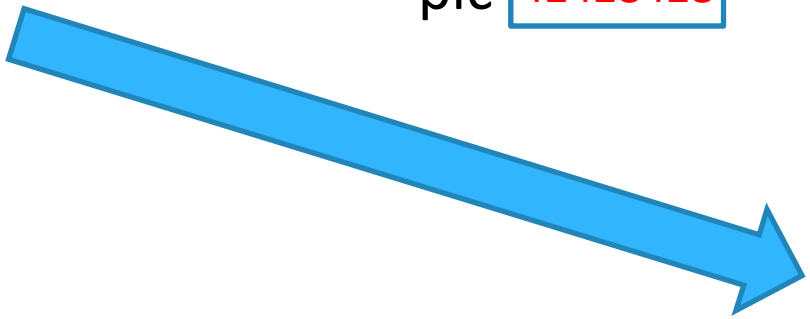
2 picture objects!

homer.jpg
A file on your computer

On your computer's hard drive



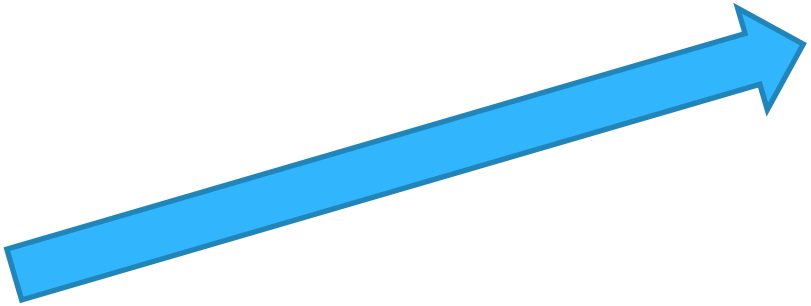
pic 42428428



pic2 30859432

```
pic = Image.open('flower.jpg')
pic2 = pic
```

1 picture object!



pic 42428428

pic2 42428428

homer.jpg
A file on your computer

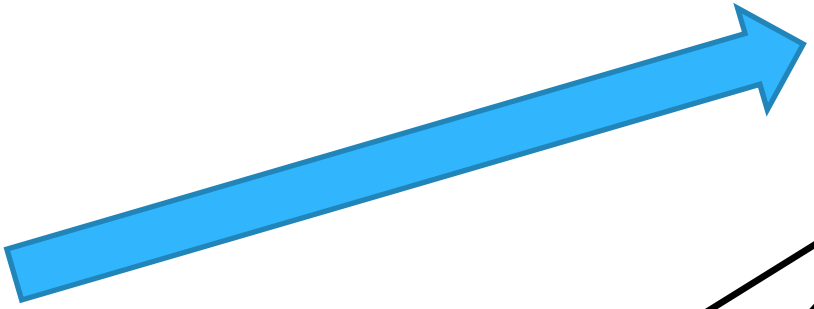
On your computer's hard drive

```
pic = Image.open('flower.jpg')
pic2 = pic
```

1 picture object!

homer.jpg
A file on your computer

On your computer's hard drive



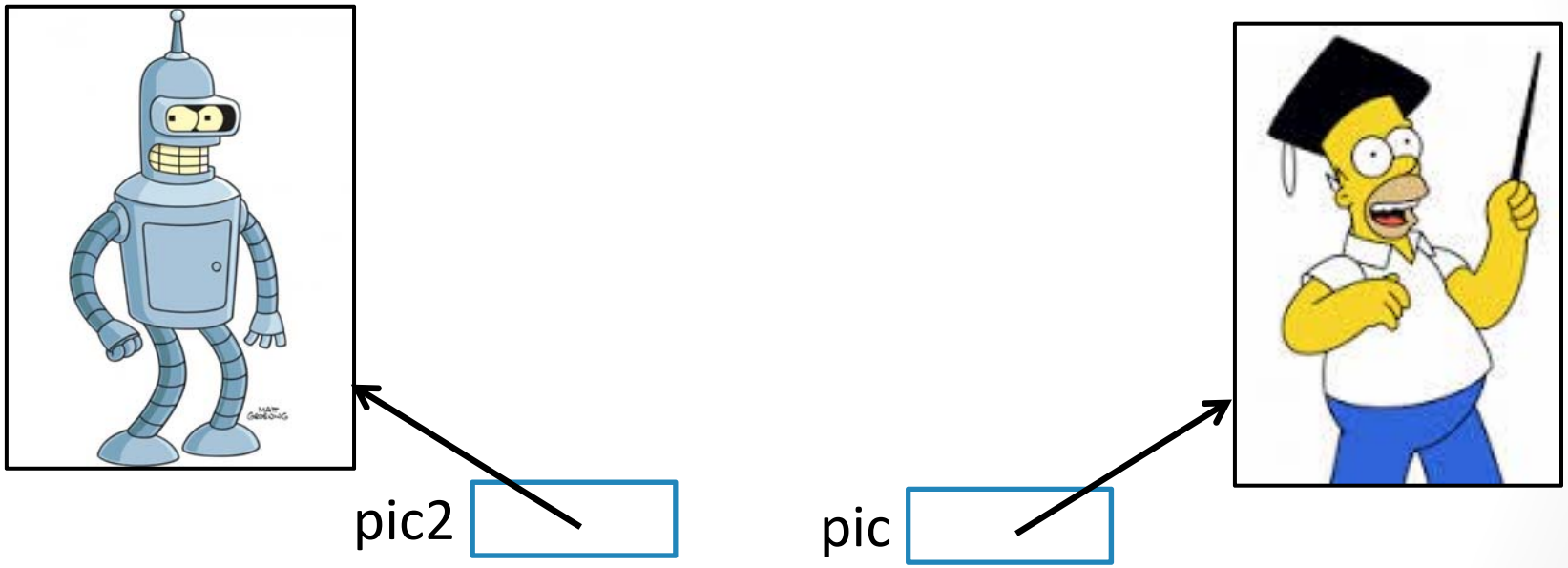
pic

pic2

Note, these arrows point to the whole object. It's not important where exactly we draw them.

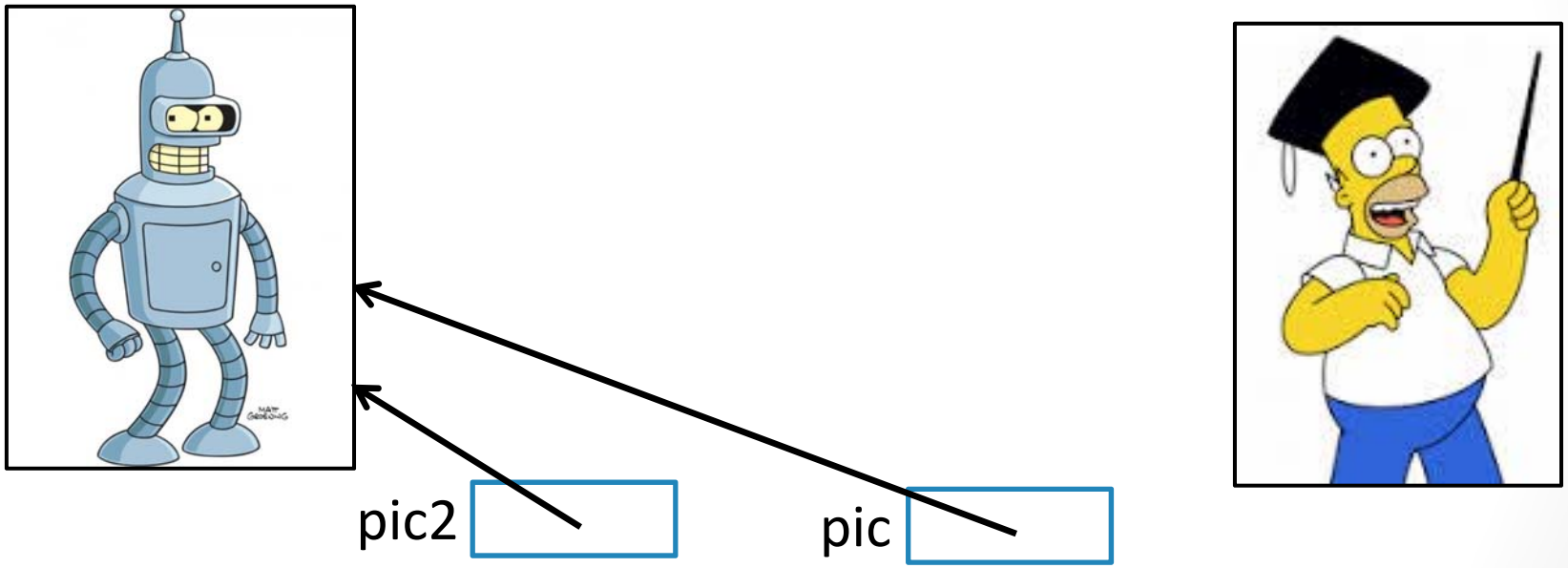
Reassignment

```
pic = Image.open('homer.jpg')
pic2 = Image.open('bender.jpg')
```



Reassignment

```
pic = Image.open('homer.jpg')
pic2 = Image.open('bender.jpg')
pic = pic2
```

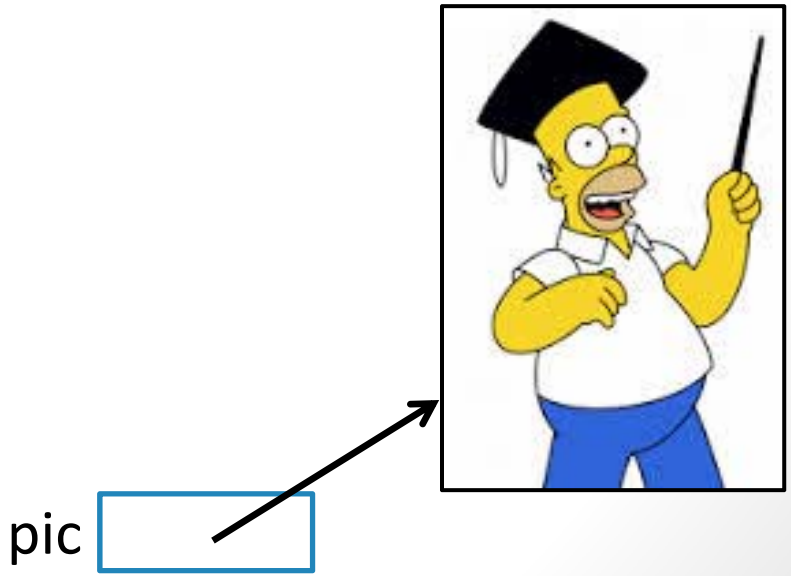


We can reassign the value of the variable, which results in it referencing something else in memory.

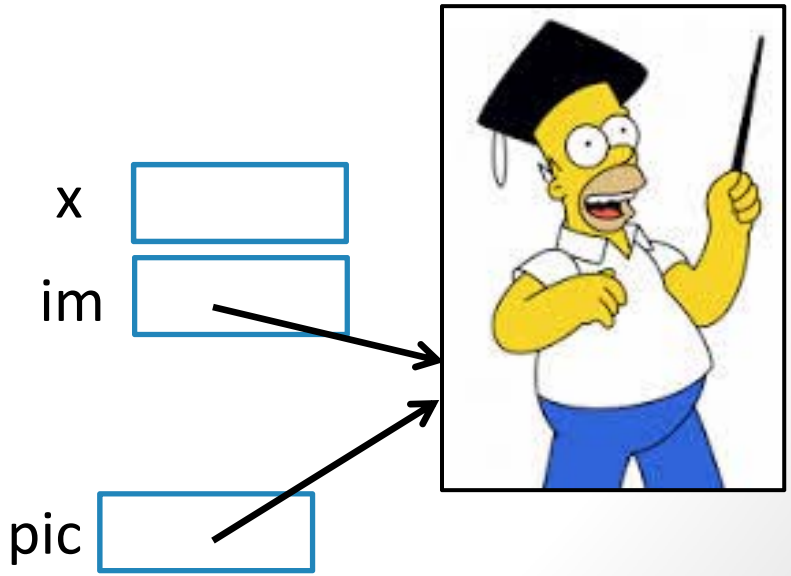
```
def silly(im):
    for x in range(im.size[0]):
        im.putpixel( (x,0), (0,255,0) )
    return im

pic = Image.open('homer.jpg')
silly(pic)
pic.show()
```

Our problem from before …

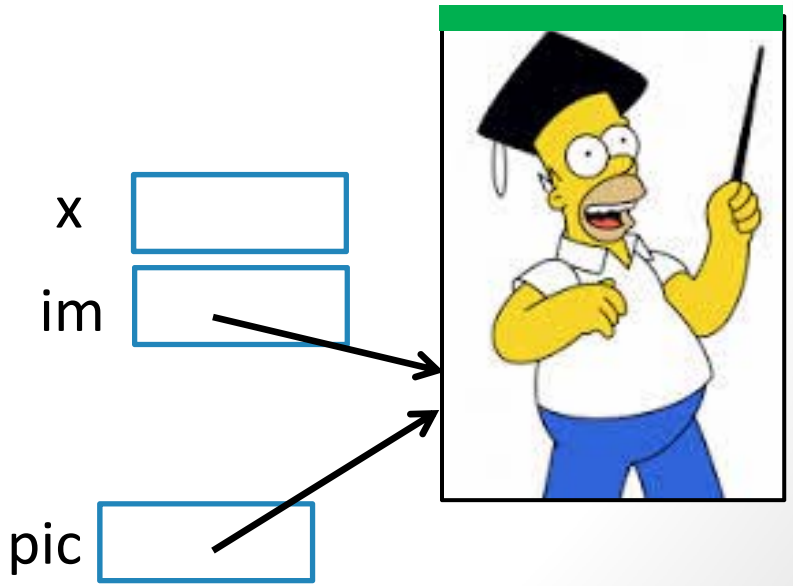
```
def silly(im):
    for x in range(im.size[0]):
        im.putpixel( (x,0), (0,255,0) )
    return im

pic = Image.open('homer.jpg')
silly(pic)
pic.show()
```



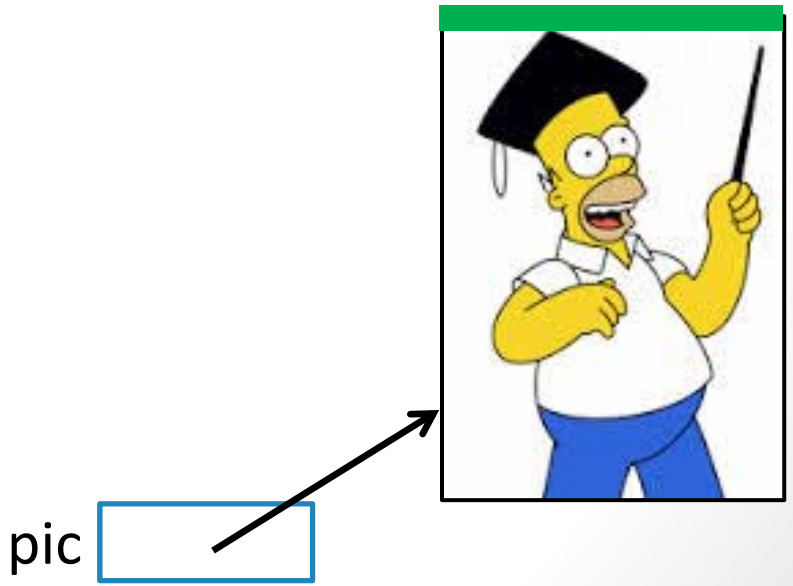
```
def silly(im):
    for x in range(im.size[0]):
        im.putpixel( (x,0), (0,255,0) )
    return im

pic = Image.open('homer.jpg')
silly(pic)
pic.show()
```



```
def silly(im):
    for x in range(im.size[0]):
        im.putpixel( (x,0), (0,255,0) )
    return im

pic = Image.open('homer.jpg')
silly(pic)
pic.show()
```



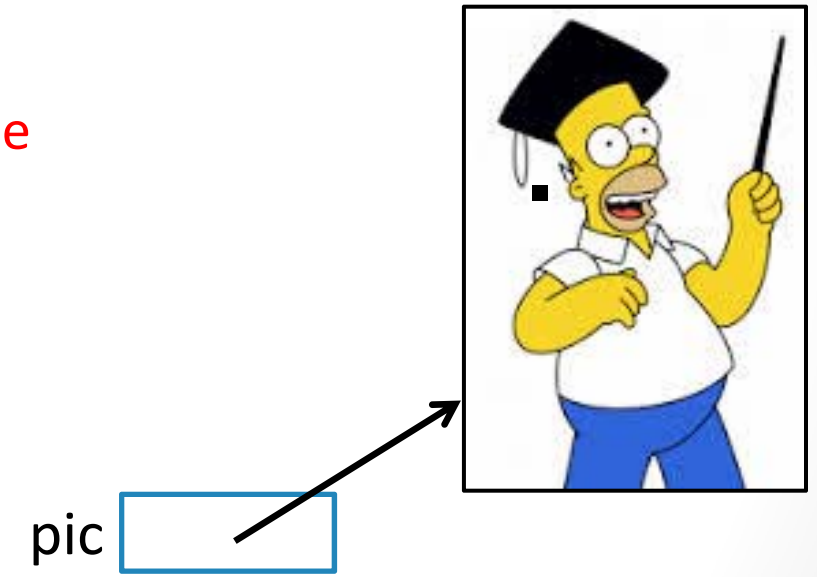
```
def silly(im):
    for x in range(im.size[0]):
        im.putpixel( (x,0), (0,255,0) )
    return im

pic = Image.open('homer.jpg')
silly(pic)
pic.show()
```



Objects are Mutable Data!

```
pic = Image.open('homer.jpg')
pic.putpixel( (3,4), (0,0,0) )
```

This is only possible because objects are MUTABLE



Via this reference we can change the value of the OBJECT. This is DIFFERENT FROM reassigning the value of the variable …

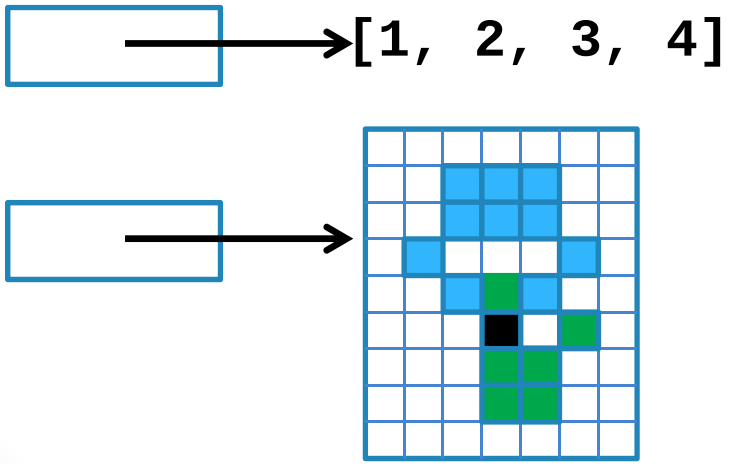
Mutable data vs. Immutable data

Changeable types:

`list`

`Image`

`(actually any user-defined object, but more on that in 8A/11)`



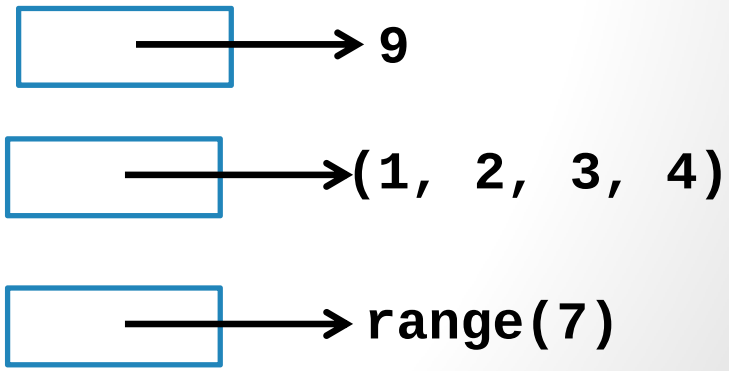
Unchangeable types:

`string` `float`

`int` `bool`

`tuple`

`range`



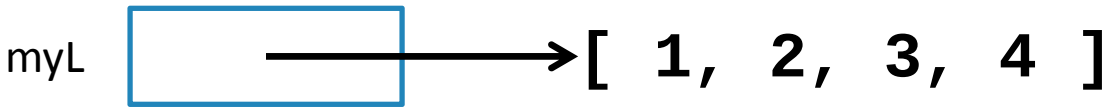
Reassignment vs. Data Mutation

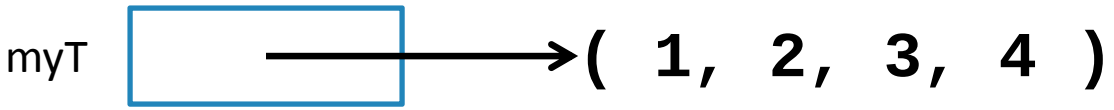
This is likely the most difficult topic you will learn in CSE8A/8B/11.
Mastering this topic is the key to acing your first year of CS!

Reassignment vs. Data Mutation

```
>>> myL = [1, 2, 3, 4]
```

```
>>> myT = (1, 2, 3, 4)
```

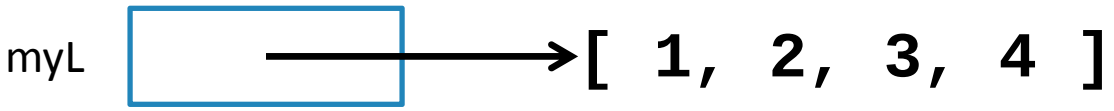


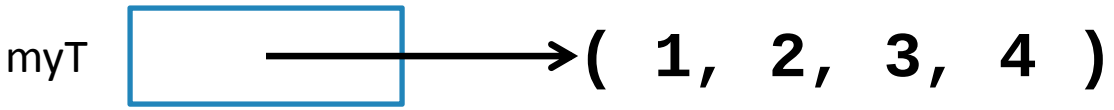


Reassignment vs. Data Mutation

```
>>> myL = [1, 2, 3, 4]
>>> myL = [10, 11, 12]
```

```
>>> myT = (1, 2, 3, 4)
>>> myT = (10, 11, 12)
```



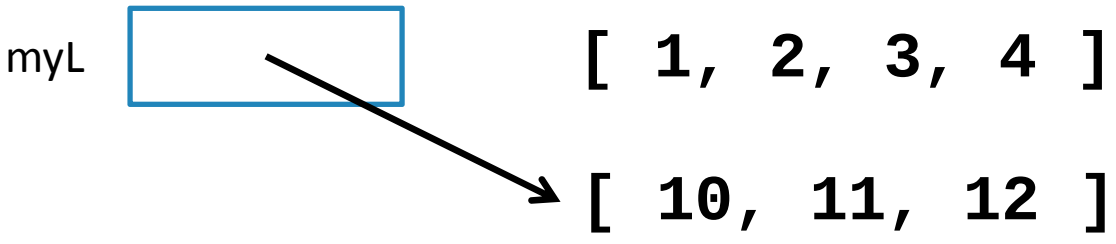


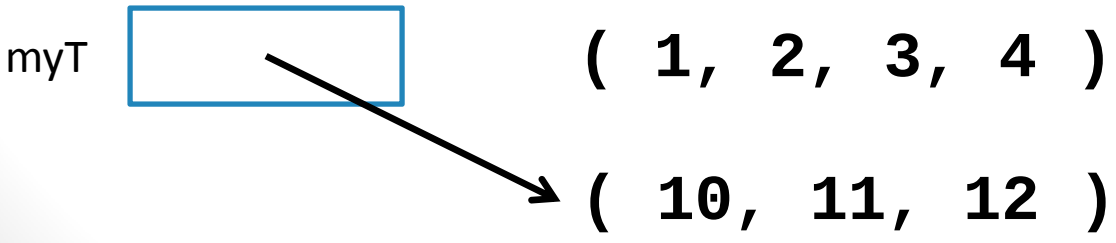
Reassignment vs. Data Mutation

```
>>> myL = [1, 2, 3, 4]
>>> myL = [10, 11, 12]
```

```
>>> myT = (1, 2, 3, 4)
>>> myT = (10, 11, 12)
```

Just like any assignment, myL and myT are REASSIGNED to a new value (i.e., a new location in memory)

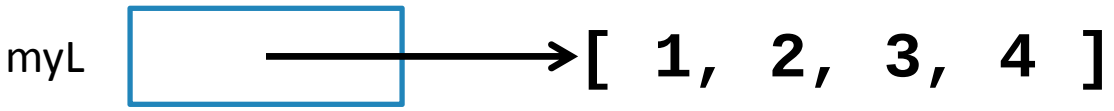


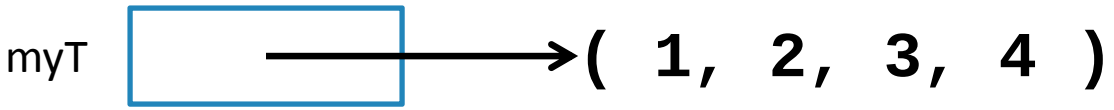


Reassignment vs. Data Mutation

```
>>> myL = [1, 2, 3, 4]
```

```
>>> myT = (1, 2, 3, 4)
```

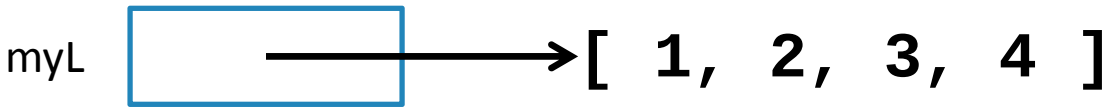


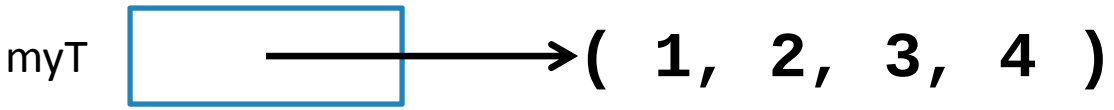


Reassignment vs. Data Mutation

```
>>> myL = [1, 2, 3, 4]
>>> myL[3] = 9
```

```
>>> myT = (1, 2, 3, 4)
>>> myT[3] = 9
```





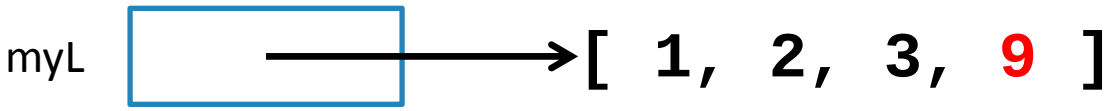
Reassignment vs. Data Mutation

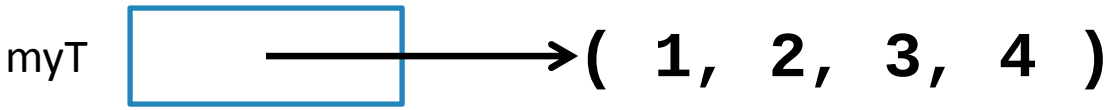
```
>>> myL = [1, 2, 3, 4]
>>> myL[3] = 9
```

Indexing MUTATES the list.

```
>>> myT = (1, 2, 3, 4)
>>> myT[3] = 9
```

Tuples are IMMUTABLE.
This statement will result in an error.



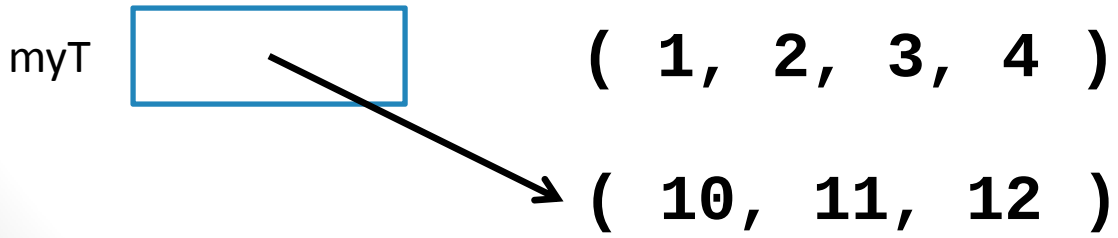


ERROR

Immutable data

```
>>> myT = (1, 2, 3, 4)
>>> myT = (10, 11, 12)
```

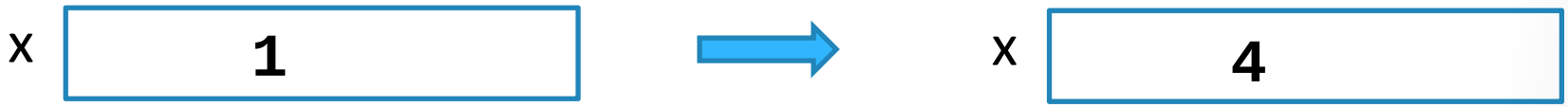
For immutable data, the fact that the variable stores a reference rather than the value itself is mostly irrelevant

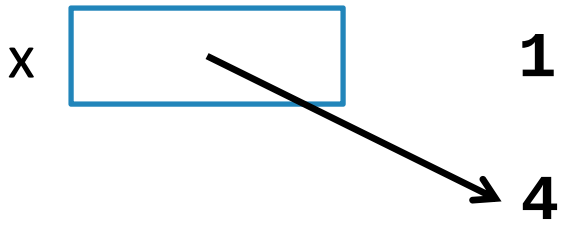


Immutable data

```
>>> x = 1
>>> x = 4
```

For immutable data, the fact that the variable stores a reference rather than the value itself is mostly irrelevant





THIS IS NOT THE CASE FOR MUTABLE DATA, WHERE MUTATION AND REASSIGNMENT ARE IMPORTANT

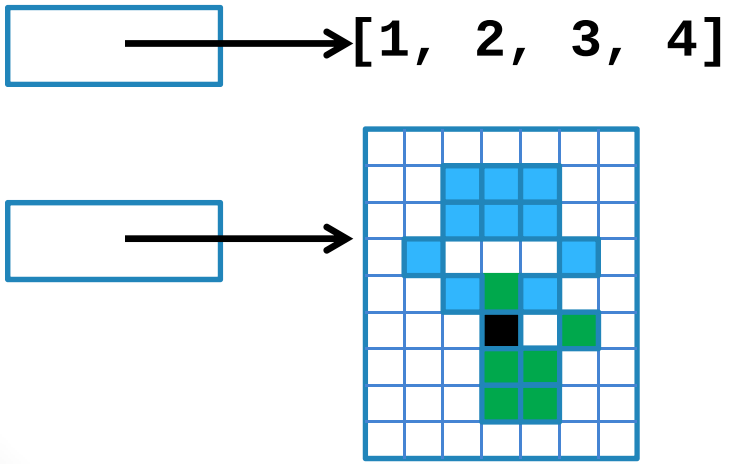
Mutable data vs. Immutable data

Changeable types:

`list`

`Image`

`(actually any user-defined object, but more on that in 8A/11)`



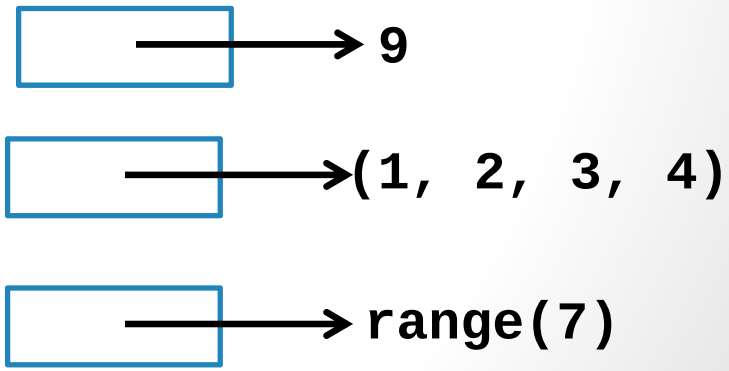
Unchangeable types:

`string` `float`

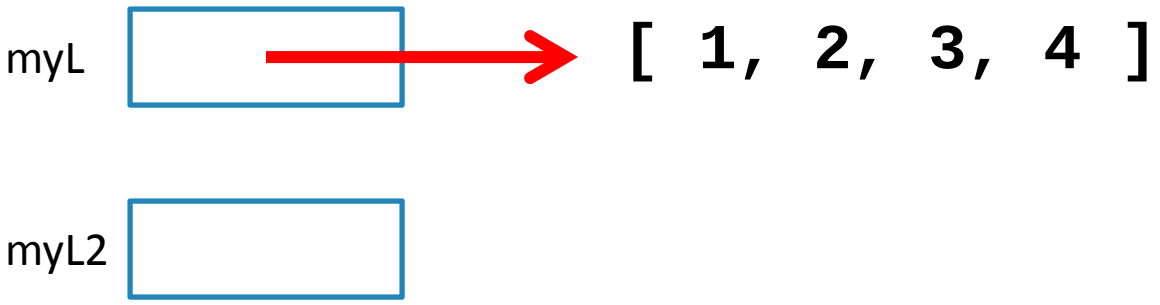
`int` `bool`

`tuple`

`range`



Reassignment vs. Data Mutation




```
>>> myL = [1, 2, 3, 4]
>>> myL2 = myL
>>> myL[1] = 5
>>> print(myL2[1])
```

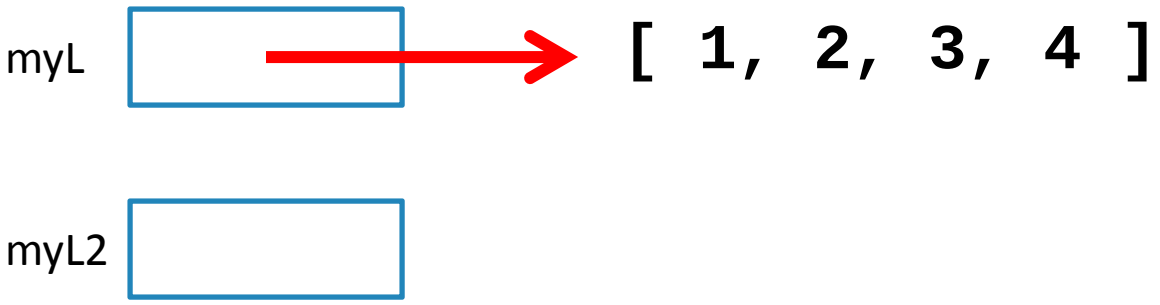
```
>>> myN = 1
>>> myN2 = myN
>>> myN = 5
>>> print(myN2)
```

What does this print?

A. 1
B. 2
C. 3
D. 5
E. Error

test02

Reassignment vs. Data Mutation



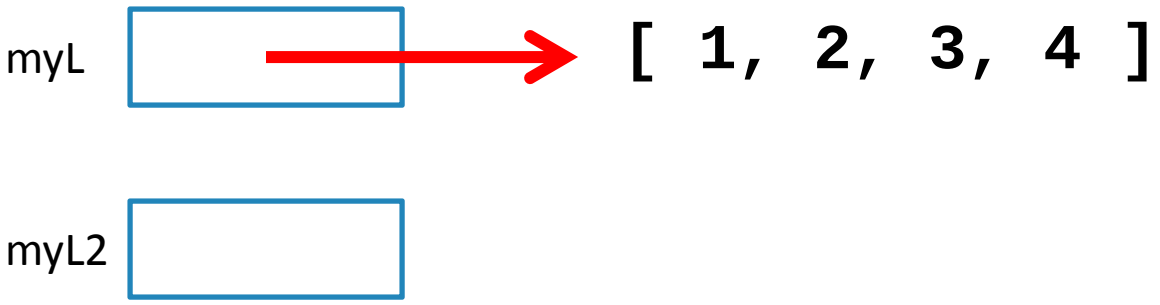

```
>>> myL = [1, 2, 3, 4]
>>> myL2 = myL
>>> myL = [5, 6, 7]
>>> myL[1] = 8
>>> print(myL2[1])
```

What does this print?

A. 2
B. 6
C. 8
D. Something else
E. Error

test03

Reassignment vs. Data Mutation




```
>>> myL = [1, 2, 3, 4]
>>> myL2 = [2, 5, 2]
>>> myL[1] = 8
>>> myL2 = myL
>>> myL = [5, 6, 7]
>>> print(myL2[1])
```

What does this print?

A. 2

B. 6

C. 8

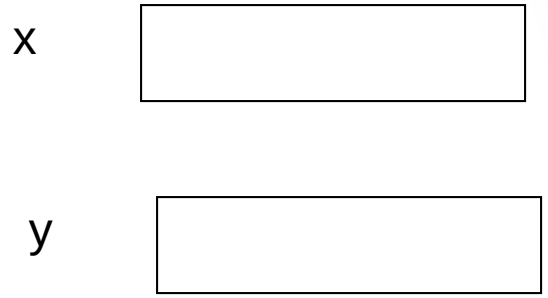
D. Something else

E. Error

test04

Swapping variable values

```
>>> x = 5
>>> y = 10
>>> x = y
>>> y = x
>>> print(x, y)
```

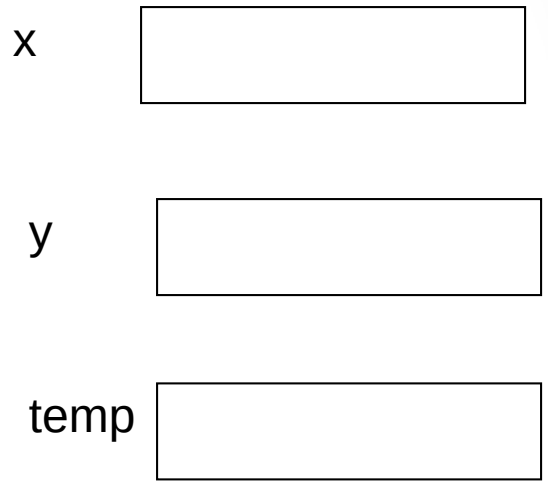


What does this print?

A. 5 10
B. 10 5
C. 5 5
D. 10 10
E. Something else

Swapping variable values

```
>>> x = 5
>>> y = 10
>>> temp = x
>>> x = y
>>> y = temp
>>> print(x, y)
```



What does this print?

A. 5 10
B. 10 5
C. 5 5
D. 10 10
E. Something else

Functions and (immutable) Variables

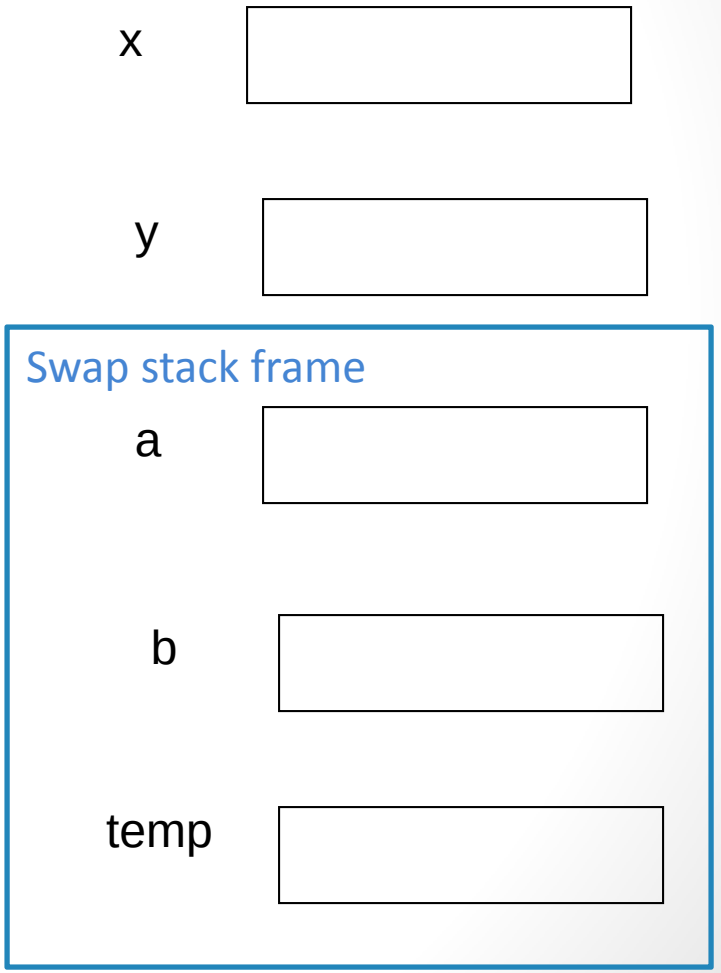
```
def swap(a, b):
    temp = a
    a = b
    b = temp

>>> x = 5
>>> y = 10
>>> swap(x, y)
>>> print(x, y)
```

What does this print?

A. 5 10
B. 10 5
C. 5 5
D. 10 10
E. Something else

Functions and (immutable) Variables

```
def swap(a, b):
    temp = a
    a = b
    b = temp

>>> x = 5
>>> y = 10
>>> swap(x, y)
>>> print(x, y)
```



What does this print?

A. 5 10
B. 10 5
C. 5 5
D. 10 10
E. Something else

Functions and Mutable Types

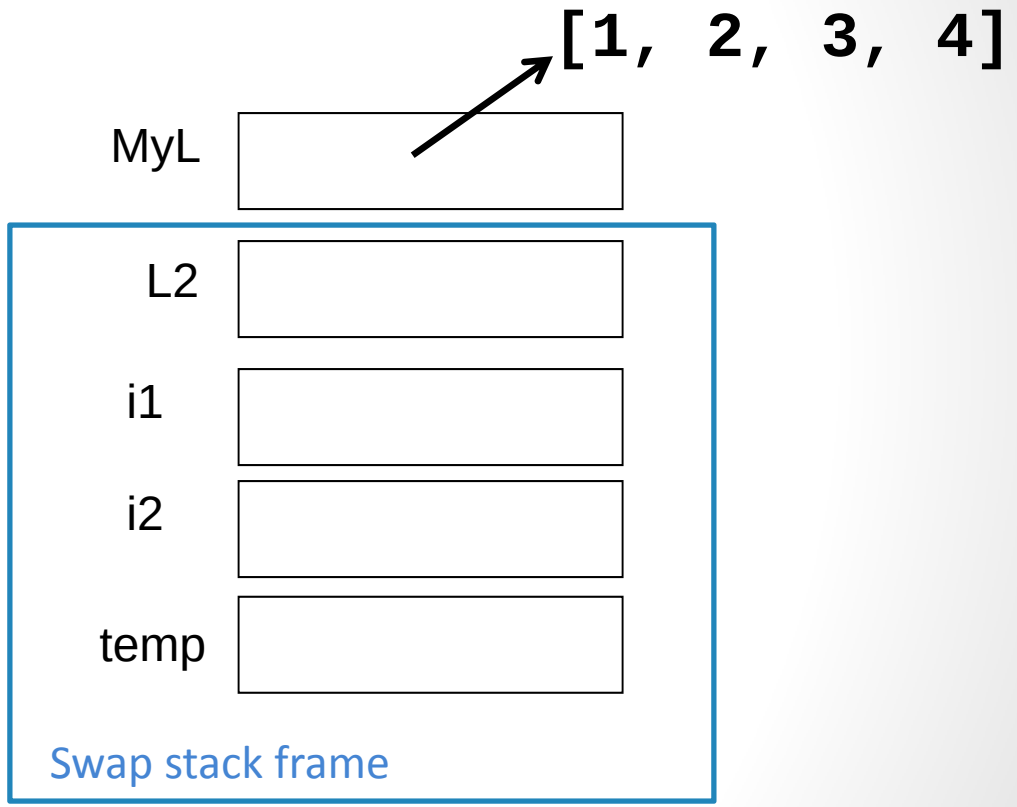
```
def swap(L2, i1, i2):
    temp = L2[i1]
    L2[i1] = L2[i2]
    L2[i2] = temp

>>> myL = [1, 2, 3, 4]
>>> swap(myL, 0, 3)
>>> print(myL)
```

What does this print?

A. [1, 2, 3, 4]
B. [3, 2, 3, 4]
C. [4, 2, 3, 1]
D. [1, 2, 4, 3]
E. Something else

Functions and Mutable Types

```
def swap(L2, i1, i2):
    temp = L2[i1]
    L2[i1] = L2[i2]
    L2[i2] = temp

>>> myL = [1, 2, 3, 4]
>>> swap(myL, 0, 3)
>>> print(myL)
```



What does this print?

A. [1, 2, 3, 4]
B. [3, 2, 3, 4]
C. [4, 2, 3, 1]
D. [1, 2, 4, 3]
E. Something else

The conclusion

You can change **the contents of lists (and pictures!)** in functions that take those lists as input.

(actually, lists or any mutable objects)

Those changes will be visible **everywhere**.

(immutable objects are safe, however)

The conclusion

Mutable data

```
def swap(L2, i1, i2):
    temp = L2[i1]
    L2[i1] = L2[i2]
    L2[i2] = temp

>>> myL = [2, 3, 4, 1]
>>> swap(myL, 0, 3)
>>> print(myL)
```

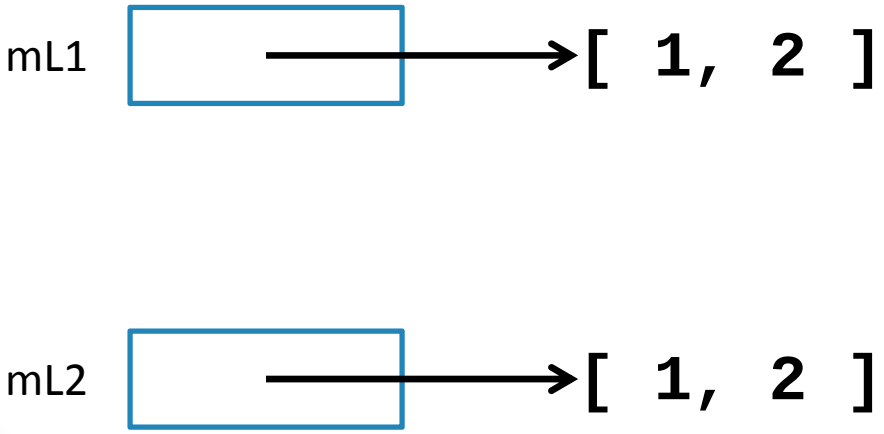
Immutable data

```
def swap(a, b):
    temp = a
    a = b
    b = temp
    return (a,b)

>>> x = 5
>>> y = 10
>>> (x,y) = swap(x, y)
>>> print(x, y)
```

Reassignment vs. Data Mutation

```
>>> mL1 = [1,2]
>>> mL1 = [3,4]

>>> print(mL1)
```

```
>>> mL2 = [1,2]
>>> mL2[0] = 3
>>> mL2[1] = 4

>>> print(mL2)
```



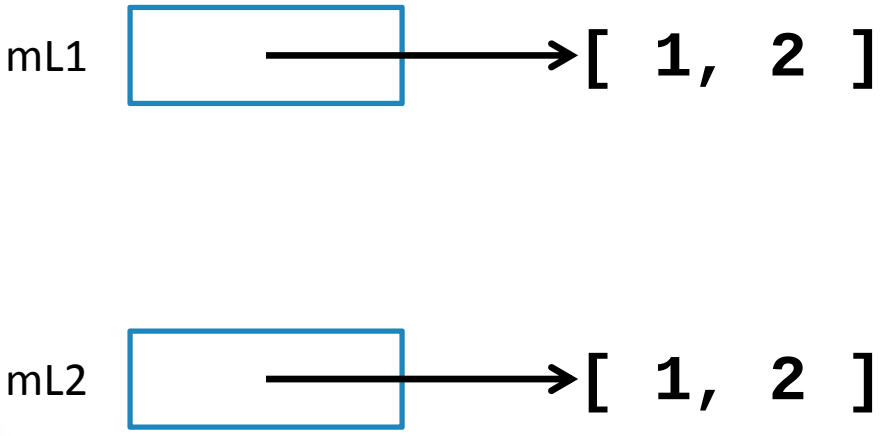
What do you think about this code?

A. It prints different results
B. It prints the same results

Reassignment vs. Data Mutation

```
>>> mL1 = [1,2]
>>> L1 = mL1
>>> mL1 = [3,4]

>>> print(mL1)
>>> print(L1)
```

```
>>> mL2 = [1,2]
>>> L2 = mL2
>>> mL2[0] = 3
>>> mL2[1] = 4

>>> print(mL2)
>>> print(L2)
```



What do you think about this code?

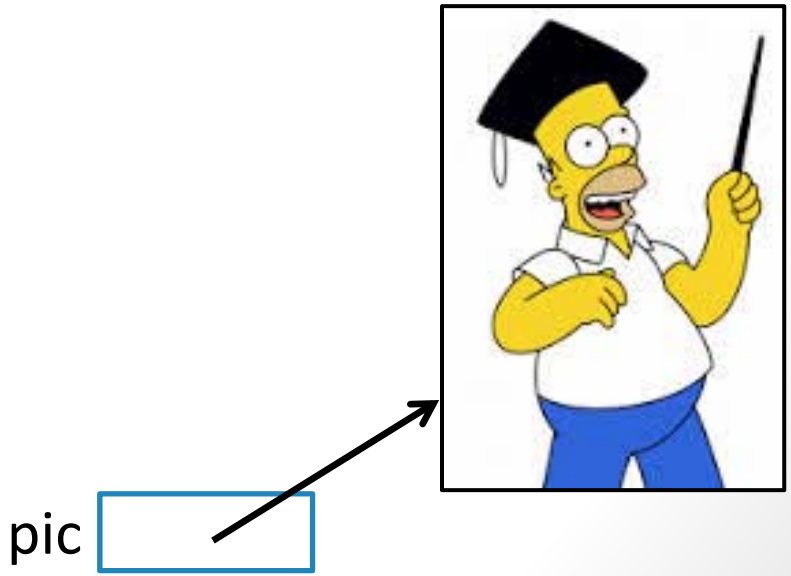
A. It prints different results
B. It prints the same results

```
def modify(im):
    for x in range(im.size[0]):
        im.putpixel( (x,0), (0,0,0) )
    return im


pic = Image.open('homer.jpg')
pic2 = modify(pic)
pic.show()
```

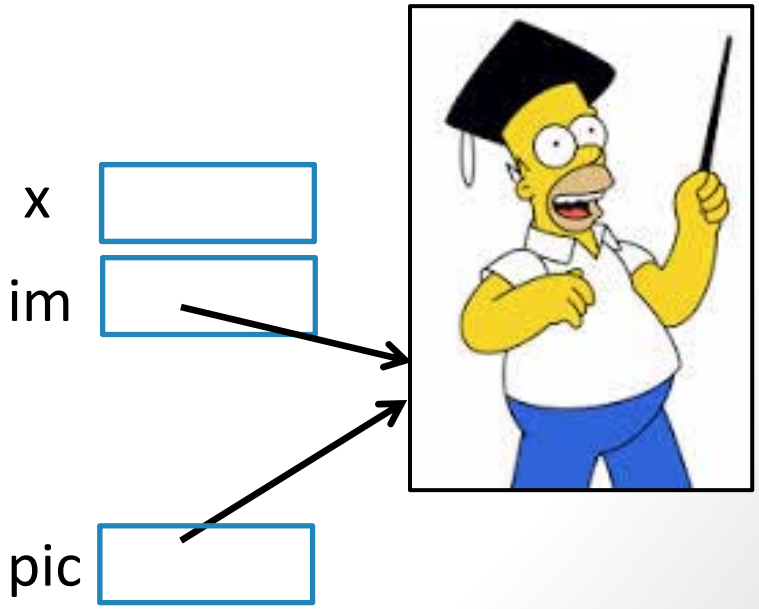
What happens?

A. You get an error
B. An empty image is shown
C. The original image is shown
D. The modified image is shown
E. Something else

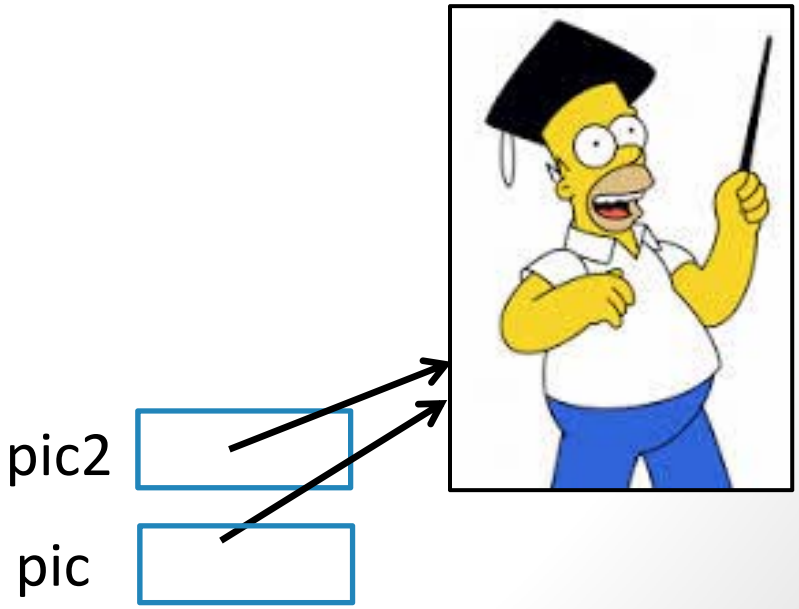
```
def modify(im):
    for x in range(im.size[0]):
        im.putpixel( (x,0), (0,0,0) )
    return im


pic = Image.open('homer.jpg')
pic2 = modify(pic)
pic.show()
```



```
def modify(im):
    for x in range(im.size[0]):
        im.putpixel( (x,0), (0,0,0) )
    return im


pic = Image.open('homer.jpg')
pic2 = modify(pic)
pic.show()
```



```
def modify(im):
    for x in range(im.size[0]):
        im.putpixel( (x,0), (0,0,0) )
    return im


pic = Image.open('homer.jpg')
pic2 = modify(pic)
pic.show()
```



```
def modify(im):
    for x in range(im.size[0]):
        im.putpixel( (x,0), (0,0,0) )


pic = Image.open('homer.jpg')
pic2 = pic
pic = Image.open('homer.jpg')
modify(pic2)
pic2 = pic
pic2.show()
```

What happens now?

A. You get an error
B. An empty image is shown
C. The original image is shown
D. The modified image is shown
E. Something else

test12