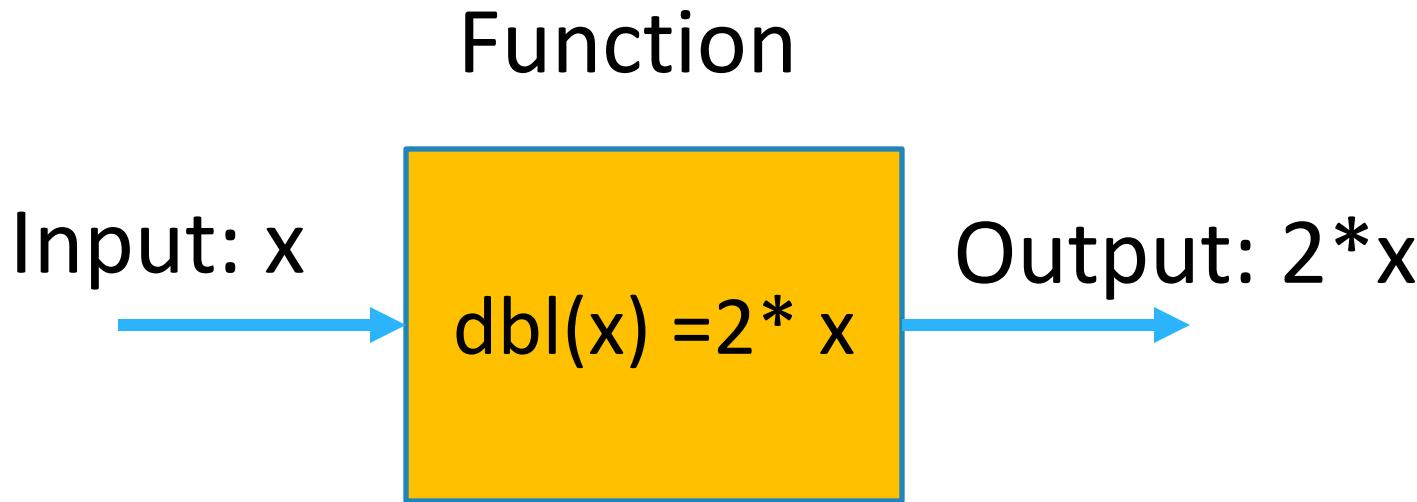


Print vs. Return Data Mutation

Understanding the behavior of functions

Understanding the behavior of functions



What are the different ways to "output" the result of the function?



Print vs. Return

```
def return dbl( x ):  
    return x*2
```

```
def print dbl( x ):  
    print(x*2)
```

```
>>> a = 32  
>>> return dbl(a)  
?(1)  
>>> print dbl(a)  
?(2)
```

Will the output of (1) and (2) be the same?

- A. Yes
- B. No

Print vs. Return

```
def return dbl( x ):  
    return x*2
```

```
def print dbl( x ):  
    print(x*2)
```

```
>>> a = 32  
>>> return dbl(a)  
?(1)  
>>> print dbl(a)  
?(2)
```

Will the output of (1) and (2) be the same?

- A. Yes
- B. No

Print vs. Return

```
def return_dbl( x ):  
    return x*2
```

```
def print_dbl( x ):  
    print(x*2)
```

```
>>> a = 32  
>>> print(return_dbl(a))  
?(1)  
>>> print(print_dbl(a))  
?(2)
```

- Will the output of (1) and (2) be the same?
- A. Yes
 - B. No

Print vs. Return

```
def return dbl( x ):  
    return x*2
```

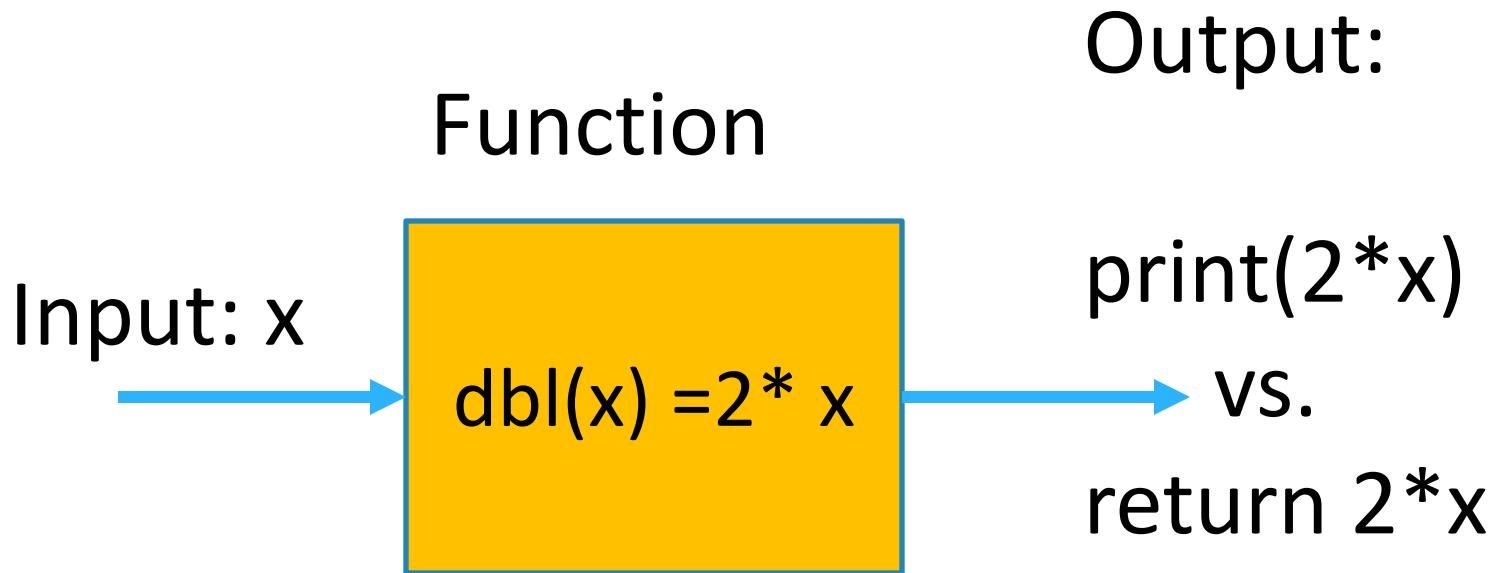
```
def print dbl( x ):  
    print(x*2)
```

```
>>> a = 32  
>>> print(return dbl(a))  
?(1)  
>>> print(print dbl(a))  
?(2)
```

Will the output of (1) and (2) be the same?

- A. Yes
- B. No

Understanding the behavior of functions



Printing vs. returning the output can lead to very different behaviors!!!!!



What is printed? (Draw boxes!)

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

```
>>> x = 67  
>>> y = 13  
>>> silly( y, x )
```

- A. 67, 13
- B. 68, 34
- C. 14, 7
- D. 8, 7
- E. Something else

Can the `silly` function change the value of parameters `x` and `y`?



What is printed? (Draw boxes!)

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

```
>>> a = 67  
>>> b = 13  
>>> silly( b, a )  
>>> print(a, ", ", b)
```

- A. 67, 13
- B. 68, 34
- C. 14, 7
- D. 8, 7
- E. Something else

Can the `silly` function change the value of the shell variables `a`, `b`?



What is printed? (Draw boxes!)

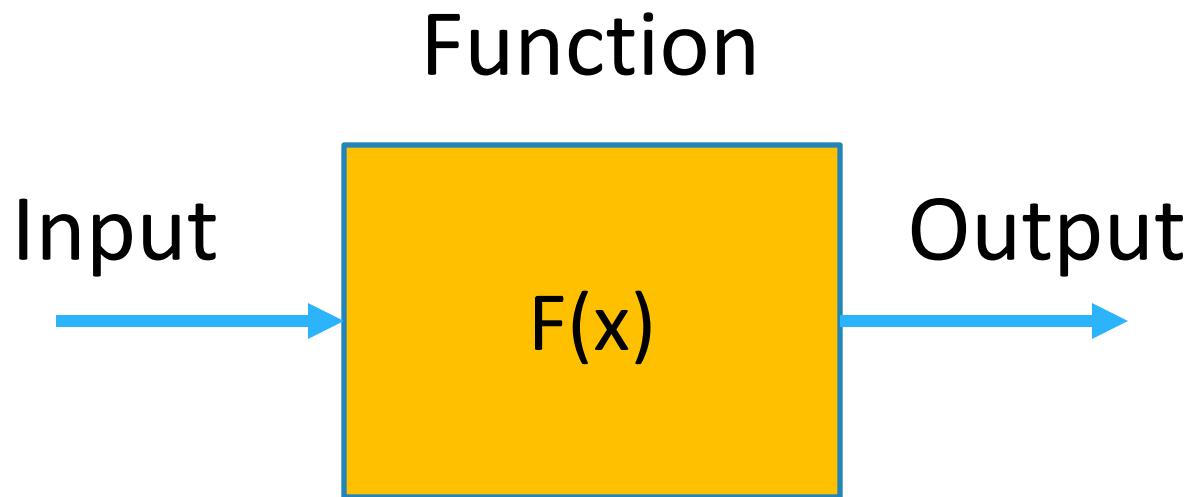
```
def silly( a, b ):  
    a = b + 1  
    b = a/2
```

```
>>> a = 67  
>>> b = 13  
>>> silly( b, a )  
>>> print(a, ", ", b)
```

Different a's and b's! Reassignment within the function has NO EFFECT on the variables in the interaction pane.

- A. 67, 13
- B. 68, 34
- C. 14, 7
- D. 8, 7
- E. Something else

Can a function change the value of the parameter y?



```
>>>F(y) # Calling function F
```



What is printed? (Draw boxes!)

```
def mutate(a):  
    a[0] = a[1] + 1  
    a[1] = a[0]/2
```

```
>>> x = [67, 13]  
>>> mutate( x )  
>>> print(x)
```

Can the `mutate` function change the value of `x`?

- A. [67, 13]
- B. [68, 34]
- C. [14, 7]
- D. [8, 7]
- E. Something else



Mutable vs. Immutable data

Changeable types:

`list`
`Pixel`
`Picture`
`Turtle`
(actually any user-defined object)

Unchangeable types:

`float`
`string`
`bool`
`int`

Lists are Mutable Data

This list “lives” in your computer’s memory



```
>>> myL = [1, 2, 3, 4] # same as myL = list(range(1,5))  
>>> myL[3] = 42 # Indexing MUTATES the list!
```

Reassignment vs. Data Mutation

DANGER! This is likely the MOST DIFFICULT topic you will learn in But mastering this topic is the key to acing this class!

Reassignment vs. Data Mutation

myL  [1 , 2 , 3 , 4]

[10 , 11 , 12]

```
>>> myL = list(range(1, 5))  
>>> myL = list(range(10, 13))
```

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

Reassignment vs. Data Mutation

myL  [1 , 2 , 3 , 4]

```
>>> myL = list(range(1, 5))
>>> myL[1] = 10
>>> myL[2] = 11
```

But these statements CHANGE the object that myL references

Reassignment vs. Data Mutation

myL  [1 , 2 , 3 , 4]

myL2 

```
>>> myL = list(range(1, 5))
>>> myL2 = myL
>>> print( myL2[1] )
```

What does the above print?

- A. 1
- B. 2
- C. 3
- D. 4
- E. Error

Reassignment vs. Data Mutation

myL  [1 , 2 , 3 , 4]

myL2 

```
>>> myL = list(range(1, 5))  
>>> myL2 = myL  
>>> myL[1] = 42  
>>> print( myL2[1] )
```

What does the above print?

- A. 1
- B. 2
- C. 42
- D. Error

Reassignment vs. Data Mutation

myL  [1 , 2 , 3 , 4]

myL2 

```
>>> myL = list(range(1, 5))
>>> myL2 = myL
>>> myL = list(range(10, 13))
>>> myL[1] = 42
>>> print( myL2[1] )
```

What does the above print?

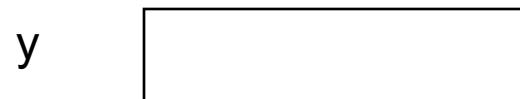
- A. 2
- B. 42
- C. 11
- D. Error
- E. Something else

Functions and (immutable) Variables

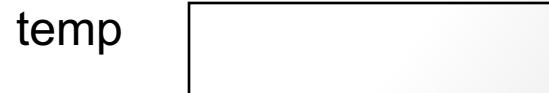
```
def swap(a, b):  
    temp = a  
    a = b  
    b = temp  
  
->>> x = 5  
->>> y = 10  
->>> swap(x, y)  
->>> print(x, y)  
??
```

What is printed?

- A. 5, 10
- B. 10, 5
- C. Something else



Swap stack frame



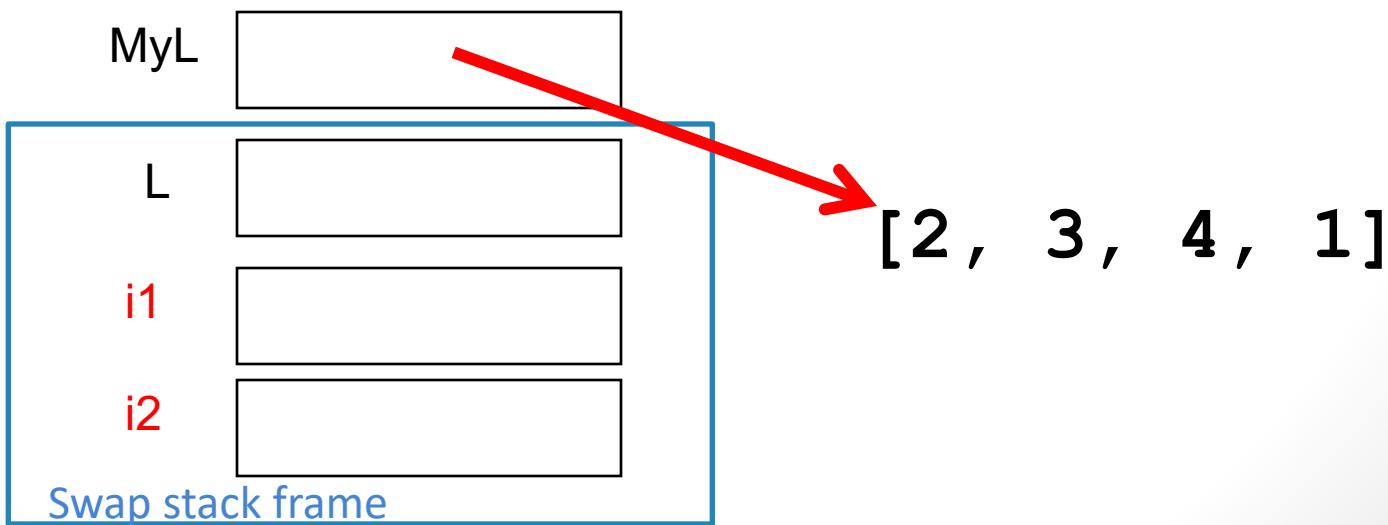
Functions and Mutable Types

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

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

What gets printed?

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



Reference vs. Value

Mutable types:

dictionary

list

Unmutable types:

tuple

float

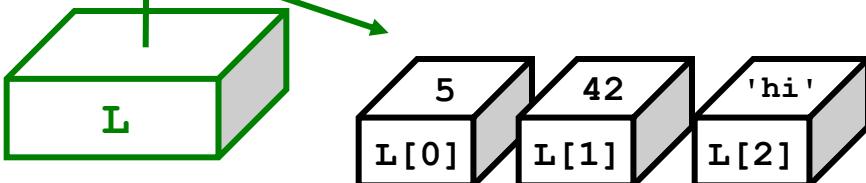
string

bool

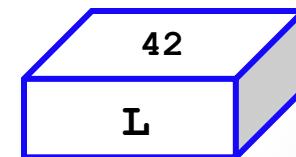
int

`L = [5, 42, 'hi']`

Reference, Pointer, id



`L = 42`



What is printed? (Draw boxes!)

```
def mutate(a):  
    a[0] = a[1] + 1  
    a[1] = a[0]/2
```

```
>>> x = [67, 13]  
>>> mutate( x )  
>>> print(x)
```

Can the `mutate` function change the value of `x`?

- A. [67, 13]
- B. [68, 34]
- C. [14, 7]
- D. [8, 7]
- E. Something else



What is printed?

```
def mutate(a):  
    a = "Diba"
```

```
>>> x = "Adib"  
>>> mutate( x )  
>>> print(x)
```

Can the `mutate` function
change the value of `x`?

- A. Diba
- B. Adib
- C. Something else

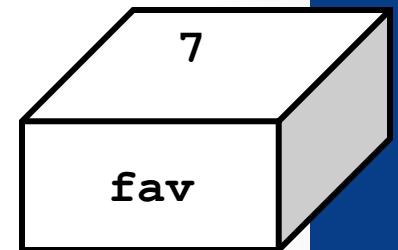


“Pass By Value”

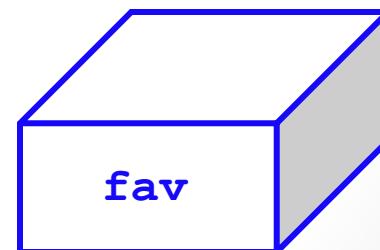
```
def main()
    """ calls conform """
    print " Welcome to Conformity, Inc. "

    fav = 7
    conform(fav)

    print " My favorite number is", fav
```



```
def conform(fav)
    """ sets input to 42 """
    fav = 42
```

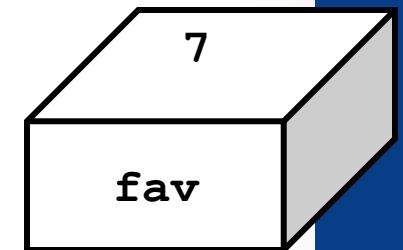


“Pass By Value”

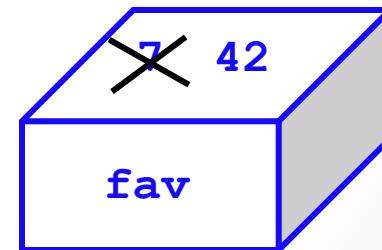
```
def main()
    """ calls conform """
    print " Welcome to Conformity, Inc. "

    fav = 7
    conform(fav)

    print " My favorite number is", fav
```



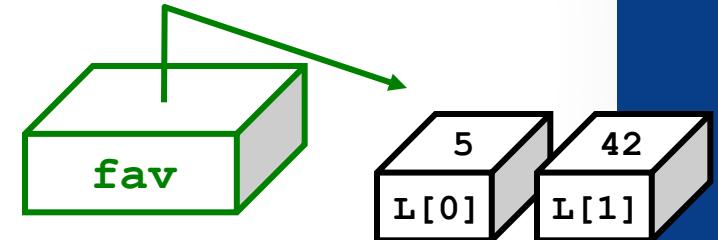
```
def conform(fav)
    """ sets input to 42 """
    fav = 42
```



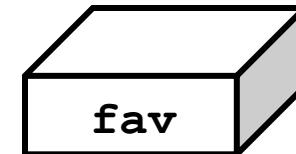
“Pass by value” means that data is **copied** when sent to a method

Passing *lists* by value...

```
def main()
    """ calls conform2 """
    print " Welcome to Conformity, Inc. "
    fav = [ 7, 11 ]
    conform2(fav)
    print " My favorite numbers are", fav
```



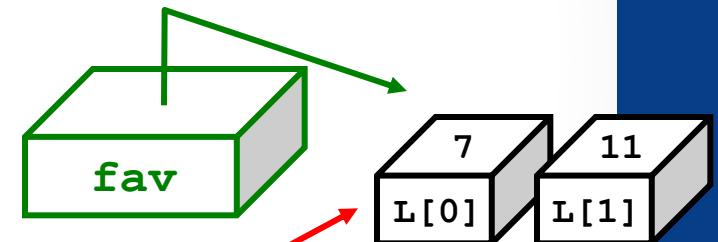
```
def conform2(fav)
    """ sets all of fav to 42 """
    fav[0] = 42
    fav[1] = 42
```



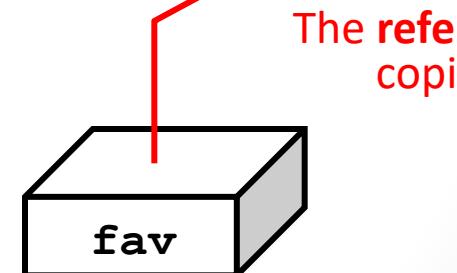
What gets passed by
value here?

Passing *lists* by value...

```
def main():
    """ calls conform2 """
    print " Welcome to Conformity, Inc. "
    fav = [ 7, 11 ]
    conform2(fav)
    print " My favorite numbers are", fav
```



```
def conform2(fav):
    """ sets all of fav to 42 """
    fav[0] = 42
    fav[1] = 42
```



The reference is copied!

can change data elsewhere!

The conclusion

You can change **the contents of lists** 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)