Review

- +=, -=, //, and % operators
- Equations of Motion
- Bouncing balls with functions and lists
- Objects – encapsulate state and behavior
- class statement
- __init__(self, …) constructor
- Instance variables and methods
- Creating instances

Assignment #3 Requirement Clarification

- Two functions, each of which draws a separate object.
- Several versions (at least 3) of each object are drawn.
- The position and size of the object must be arguments passed to the functions.
- The values of the position and size of the objects should be randomized each time your program runs. ~~the sketch is drawn.~~
- At least one of the objects must react to the position of the mouse.
- At least one of the objects must react when the mouse is over the object, or when the object is clicked.
- Includes proper header and adequate comments

The Python Class Statement

```
class MyClass:

    # Constructor
    def __init__(self, arg1, arg2):

        # Init instance variables
        self.ivar1 = arg1
        self.ivar2 = arg2
        self.ivar3 = 64       # All inst's init'd to same val

    # Define methods
    def incrementBy(self, val):
        # Increment an instance variable
        self.ivar1 += val

    # Another method
    def remainderVar1(self, mod):
        # Return a computed value
        return self.ivar1 % mod
```

What is the type of an object instance?

```
# Simple class
class Test:
    def __init__(self, name):
        self.name = name

t = Test('Fred')
print( type( t ) )
print( type( Test ) )
```

What is the type of an object instance?

```
# Simple class
class Test:
    def __init__(self, name):
        self.name = name

t = Test('Fred')
print( type( t ) )
print( type( Test ) )
```

```
>>> <type 'instance'>
>>> <type 'classobj'>
```

What is the output when you print object instance?

```
# Simple class
class Test:
    def __init__(self, name):
        self.name = name

t = Test('Fred')
print( t )
```

What is the output when you print object instance?

```
# Simple class
class Test:
    def __init__(self, name):
        self.name = name

t = Test('Fred')
print( t )
```

```
>>> <<module>.Test instance at 0x000000000000002D>
```

Not useful!

A special method named __str__()

- Add a __str__() method to the class definition to define a string representation for you objects
- The __str__() method must return the string representation, which can be built using instance variables

What is the output when you print object instance?

```
# Simple class
class Test:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return 'My name is ' + self.name

t = Test('Fred')
print( t )
```

What is the output when you print object instance?

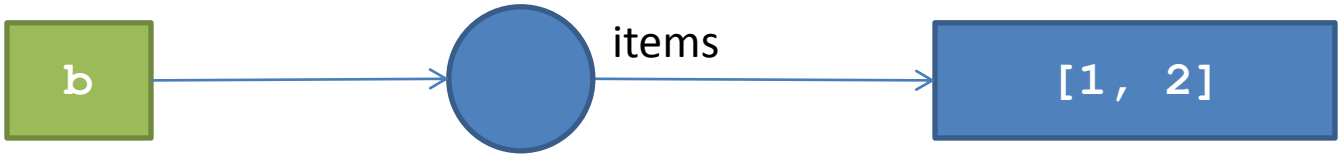
```
# Simple class
class Test:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return 'My name is ' + self.name

t = Test('Fred')
print( t )

>>> My name is Fred
```

Useful!

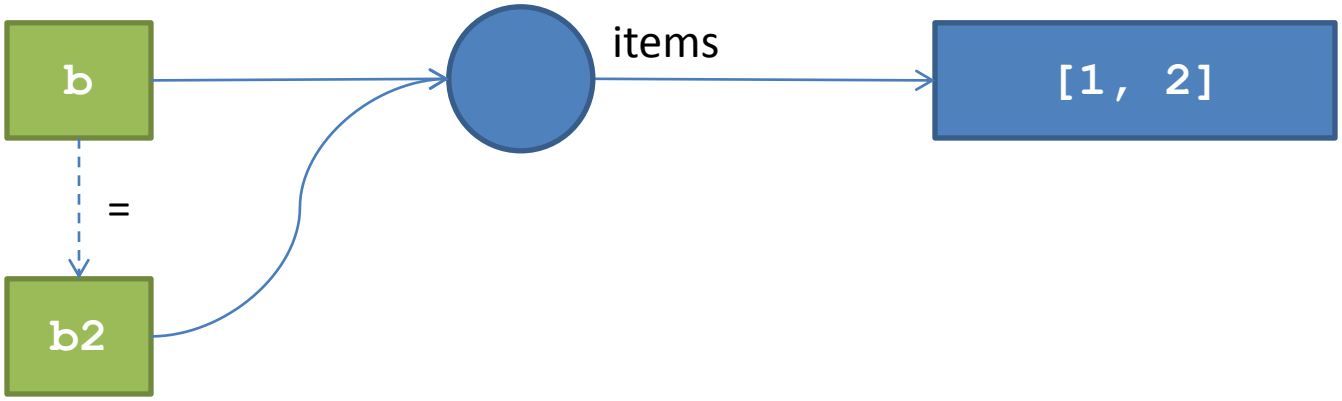
Assignment vs. Shallow Copy vs. Deep Copy

```
class Bag:
    def __init__(self):
        self.items = []

    def add(self, item):
        self.items.append( item )

    def __str__(self):
        return str(self.items)


b = Bag()
b.add(1)
b.add(2)
print(b)        # [1, 2]

b2 = b          # Assignment
b2.add(3)
print( b2 )     # [1, 2, 3]
print( b )      # [1, 2, 3] !!!!
```

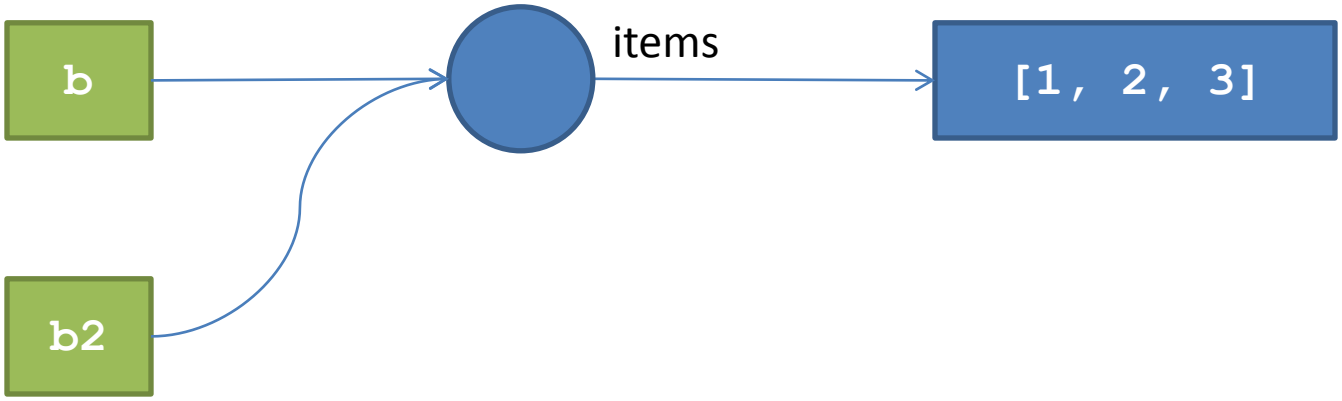
Assignment vs. Shallow Copy vs. Deep Copy



Assignment vs. Shallow Copy vs. Deep Copy



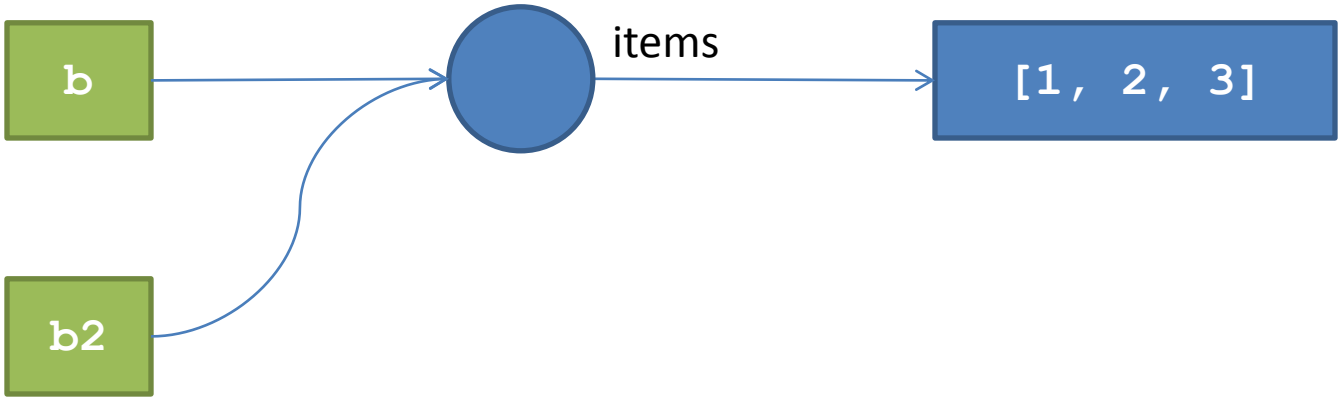
Assignment vs. Shallow Copy vs. Deep Copy



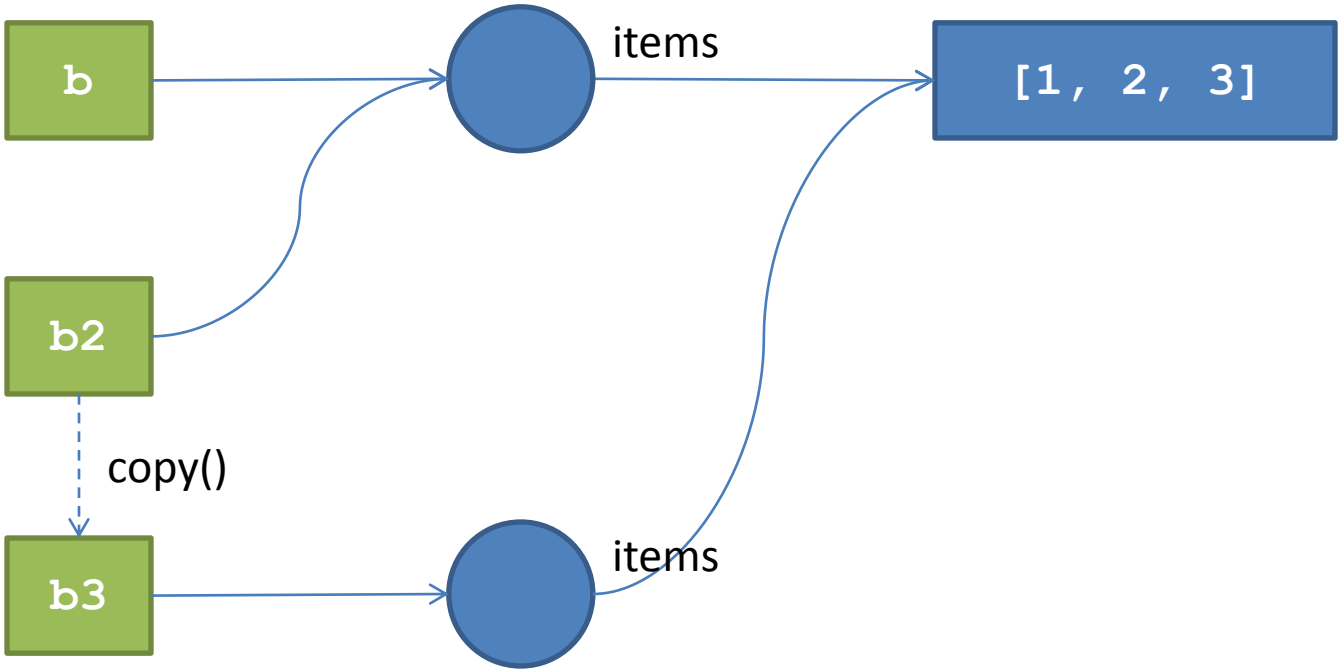
Assignment vs. Shallow Copy vs. Deep Copy

```
b = Bag()
b.add(1)
b.add(2)
print(b)        # [1, 2]

b2 = b          # Assignment
b2.add(3)
print( b2 )     # [1, 2, 3]
print( b )      # [1, 2, 3] !!!!

from copy import copy, deepcopy

b3 = copy(b)
b3.add(4)
print(b3)       # [1, 2, 3, 4]
print(b)        # [1, 2, 3, 4] Still?
```

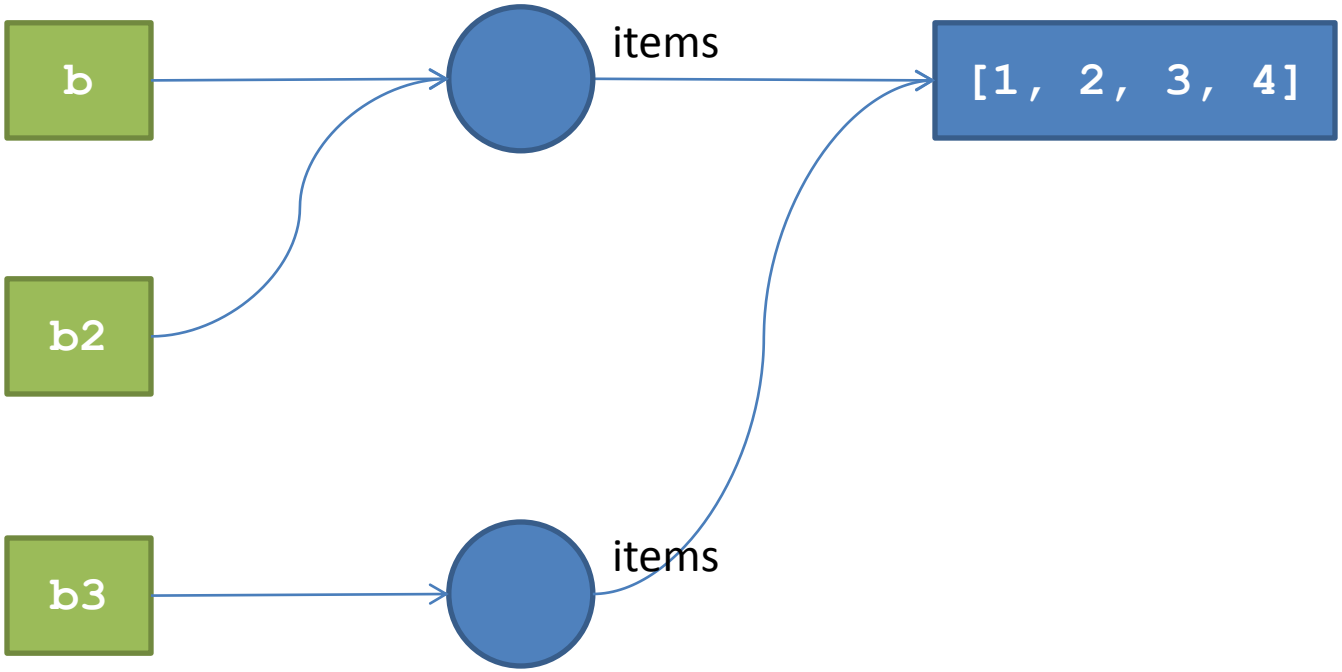
Assignment vs. Shallow Copy vs. Deep Copy



Assignment vs. Shallow Copy vs. Deep Copy



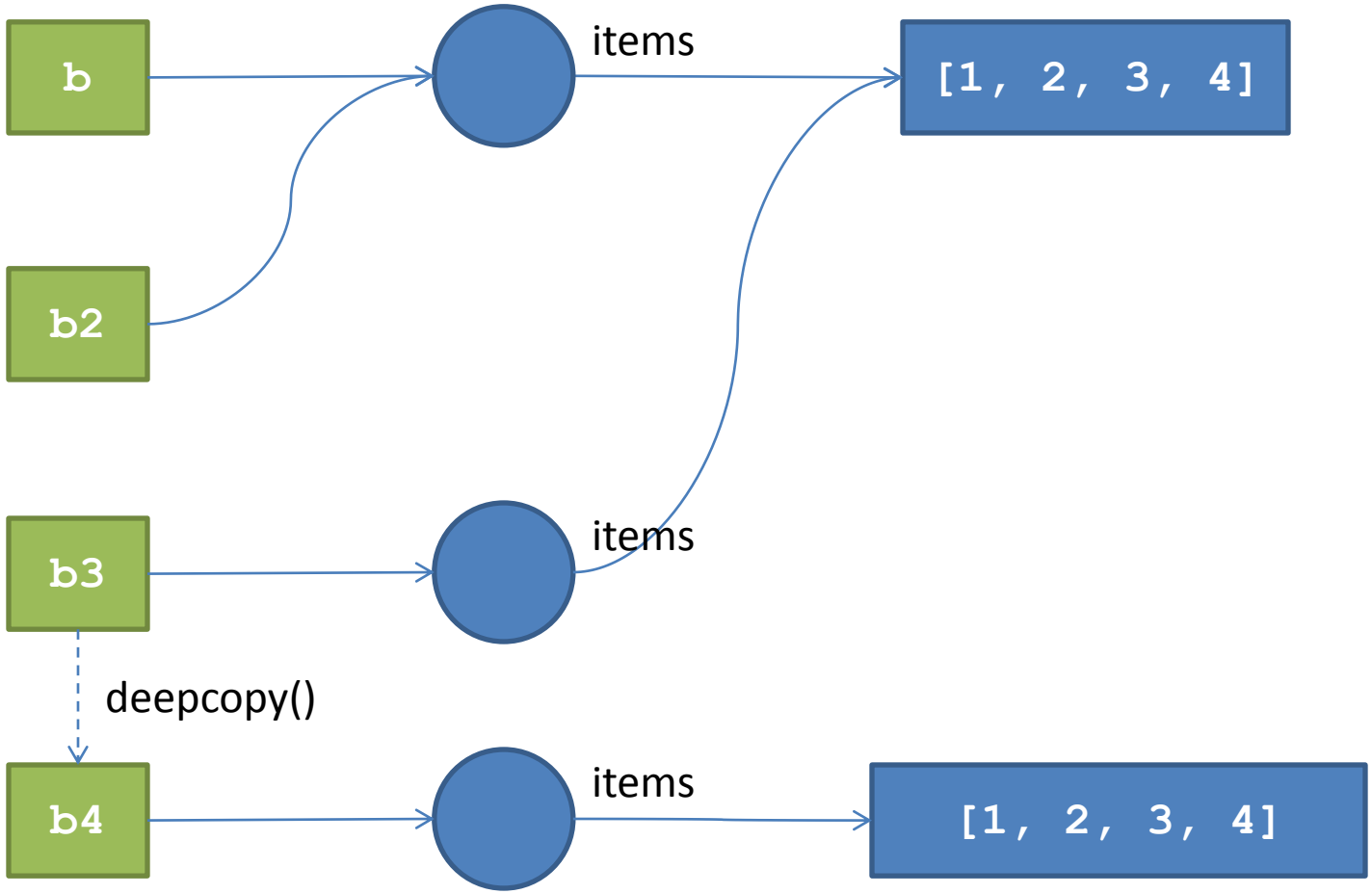
Assignment vs. Shallow Copy vs. Deep Copy



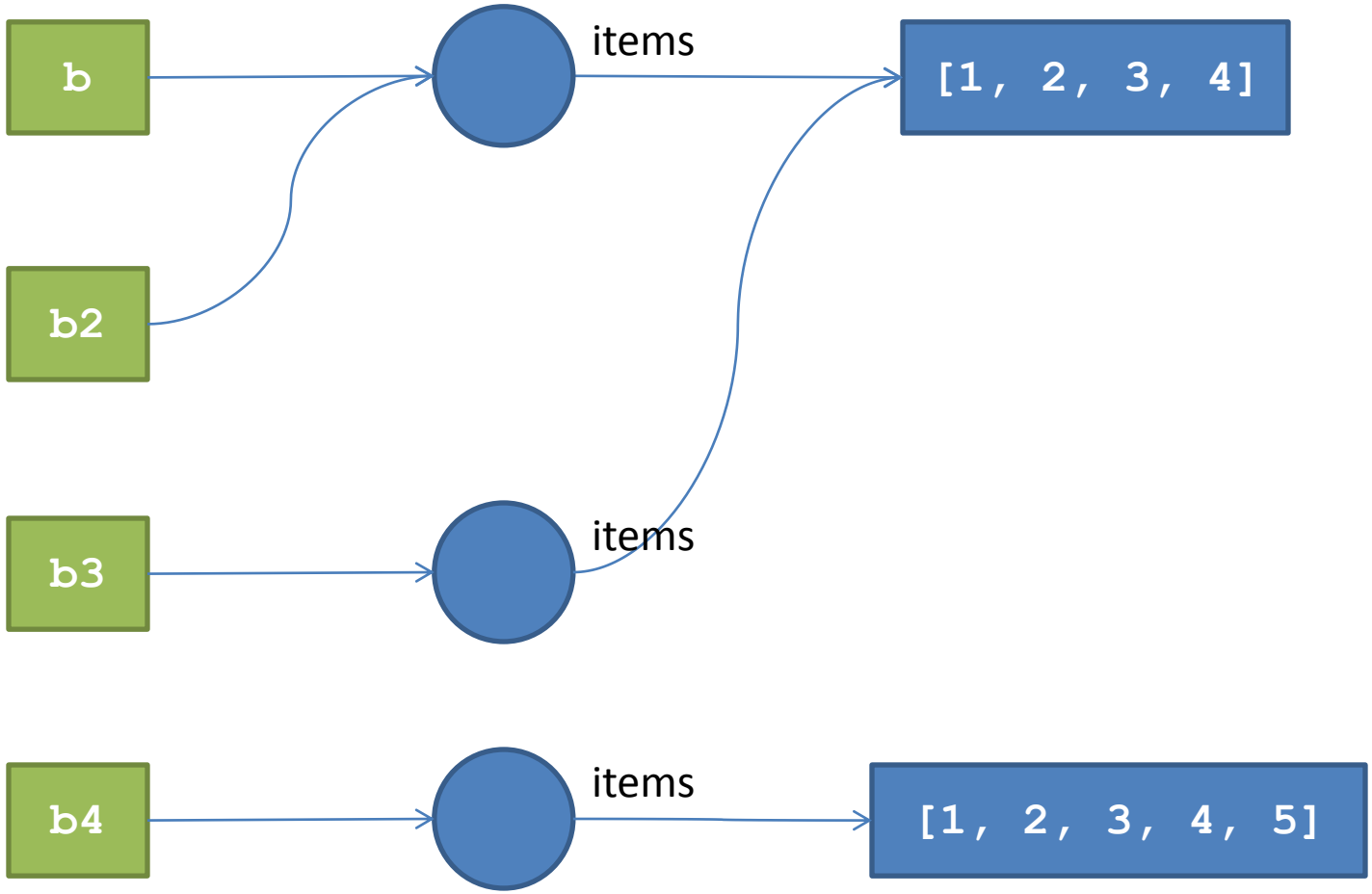
Assignment vs. Shallow Copy vs. Deep Copy

```
b = Bag()
b.add(1)
b.add(2)
print(b)        # [1, 2]

b2 = b          # Assignment
b2.add(3)
print( b2 )     # [1, 2, 3]
print( b )      # [1, 2, 3] !!!!

from copy import copy, deepcopy

b3 = copy(b)
b3.add(4)
print(b3)       # [1, 2, 3, 4]
print(b)        # [1, 2, 3, 4] Still?

b4 = deepcopy(b)
b4.add(5)
print(b4)       # [1, 2, 3, 4, 5]
print(b)        # [1, 2, 3, 4] Yes!
```

Assignment vs. Shallow Copy vs. Deep Copy



Assignment vs. Shallow Copy vs. Deep Copy



Exam 1 Review Problems