# CS 61A Extra Lecture 6

## Implementing an Object System

Brian Hou

March 5, 2015

## Announcements

- Extra Homework 2 due tonight!
- Extra Homework 3 due Thursday 4/2

## Objects in Python

## Review: Classes and Methods

```
class Adder:
    def __init__(self, a, b):
        self.a, self.b = a, b
    def total(self):
        return self.a + self.b
```

## Review: Classes and Methods

```
class Adder:
    def __init__(self, a, b):
        self.a, self.b = a, b
    def total(self):
        return self.a + self.b
```

```
>>> seven = Adder(6, 1)
```

## Review: Classes and Methods

```
class Adder:
    def __init__(self, a, b):
        self.a, self.b = a, b
    def total(self):
        return self.a + self.b
```

```
>>> seven = Adder(6, 1)

>>> seven.total
```

## Review: Classes and Methods

```
class Adder:
    def __init__(self, a, b):
        self.a, self.b = a, b
    def total(self):
        return self.a + self.b
```

```
>>> seven = Adder(6, 1)
>>> seven.total
<bound method Adder.total of ...>
```

## Review: Classes and Methods

```
class Adder:
    def __init__(self, a, b):
        self.a, self.b = a, b
    def total(self):
        return self.a + self.b
```

```
>>> seven = Adder(6, 1)
>>> seven.total
<bound method Adder.total of ...>
>>> seven.total()
```

## Review: Classes and Methods

```
class Adder:
    def __init__(self, a, b):
        self.a, self.b = a, b
    def total(self):
        return self.a + self.b
```

```
>>> seven = Adder(6, 1)
>>> seven.total
<bound method Adder.total of ...>
>>> seven.total()
7
```

## Review: Classes and Methods

```python
class Adder:
    def __init__(self, a, b):
        self.a, self.b = a, b
    def total(self):
        return self.a + self.b
```

```
>>> seven = Adder(6, 1)
>>> seven.total
<bound method Adder.total of ...>
>>> seven.total()
7
>>> Adder.total(seven)
```

## Review: Classes and Methods

```
class Adder:
    def __init__(self, a, b):
        self.a, self.b = a, b
    def total(self):
        return self.a + self.b
```

```
>>> seven = Adder(6, 1)
>>> seven.total
<bound method Adder.total of ...>
>>> seven.total()
7
>>> Adder.total(seven)
7
```

## Accessing Attributes

## Accessing Attributes

When we use object-oriented programming, there are two fundamental operations:

## Accessing Attributes

When we use object-oriented programming, there are two fundamental operations:

- looking up an attribute's value

## Accessing Attributes

When we use object-oriented programming, there are two fundamental operations:

- looking up an attribute's value
- defining an attribute's value

## Accessing Attributes

When we use object-oriented programming, there are two fundamental operations:

- looking up an attribute's value
- defining an attribute's value

We can use the `getattr` and `setattr` functions

## Accessing Attributes

When we use object-oriented programming, there are two fundamental operations:

- looking up an attribute's value
- defining an attribute's value

We can use the `getattr` and `setattr` functions

```
>>> getattr(seven, 'a') # seven.a
```

## Accessing Attributes

When we use object-oriented programming, there are two fundamental operations:

- looking up an attribute's value
- defining an attribute's value

We can use the `getattr` and `setattr` functions

```
>>> getattr(seven, 'a') # seven.a
6
```

## Accessing Attributes

When we use object-oriented programming, there are two fundamental operations:

- looking up an attribute's value
- defining an attribute's value

We can use the `getattr` and `setattr` functions

```
>>> getattr(seven, 'a') # seven.a
6
>>> setattr(seven, 'a', 7) # seven.a = 7
```

## Accessing Attributes

When we use object-oriented programming, there are two fundamental operations:

- looking up an attribute's value
- defining an attribute's value

We can use the `getattr` and `setattr` functions

```
>>> getattr(seven, 'a') # seven.a
6
>>> setattr(seven, 'a', 7) # seven.a = 7
>>> getattr(seven, 'total')()
```

## Accessing Attributes

When we use object-oriented programming, there are two fundamental operations:

- looking up an attribute's value
- defining an attribute's value

We can use the `getattr` and `setattr` functions

```
>>> getattr(seven, 'a') # seven.a

6

>>> setattr(seven, 'a', 7) # seven.a = 7

>>> getattr(seven, 'total')()

8
```

## Implementing an Object System

## Goals

## Goals

- Object instantiation and initialization

```
>>> seven = Adder(6, 1)
```

## Goals

- Object instantiation and initialization

```
>>> seven = Adder(6, 1)
```

- Attribute lookup and assignment

```
>>> seven.a = 8
```

## Goals

- Object instantiation and initialization

```
>>> seven = Adder(6, 1)
```

- Attribute lookup and assignment

```
>>> seven.a = 8
```

- Method invocation

```
>>> seven.total()
```

## Goals

- Object instantiation and initialization

```
>>> seven = Adder(6, 1)
```

- Attribute lookup and assignment

```
>>> seven.a = 8
```

- Method invocation

```
>>> seven.total()
```

- Inheritance

## Instances

```
def make_instance(cls):
```

## Instances

```
def make_instance(cls):







    instance = {'get': get_value, 'set': set_value} # dispatch dictionary
    return instance
```

## Instances

```
def make_instance(cls):
    attributes = {} # instance attributes, e.g. {'a': 6, 'b': 1}









    instance = {'get': get_value, 'set': set_value} # dispatch dictionary
    return instance
```

## Instances

```
def make_instance(cls):
    attributes = {} # instance attributes, e.g. {'a': 6, 'b': 1}
    def get_value(name):








    instance = {'get': get_value, 'set': set_value} # dispatch dictionary
    return instance
```

## Instances

```
def make_instance(cls):
    attributes = {} # instance attributes, e.g. {'a': 6, 'b': 1}
    def get_value(name):
        if name in attributes: # name is an instance attribute
            return attributes[name]






    instance = {'get': get_value, 'set': set_value} # dispatch dictionary
    return instance
```

## Instances

```
def make_instance(cls):
    attributes = {} # instance attributes, e.g. {'a': 6, 'b': 1}
    def get_value(name):
        if name in attributes: # name is an instance attribute
            return attributes[name]
        value = cls['get'](name) # look up name in class




    instance = {'get': get_value, 'set': set_value} # dispatch dictionary
    return instance
```

## Instances

```
def make_instance(cls):
    attributes = {} # instance attributes, e.g. {'a': 6, 'b': 1}
    def get_value(name):
        if name in attributes: # name is an instance attribute
            return attributes[name]
        value = cls['get'](name) # look up name in class
        return (bind_method(value, instance) if callable(value) else value)



    instance = {'get': get_value, 'set': set_value} # dispatch dictionary
    return instance
```

## Instances

```
def make_instance(cls):
    attributes = {} # instance attributes, e.g. {'a': 6, 'b': 1}
    def get_value(name):
        if name in attributes: # name is an instance attribute
            return attributes[name]
        value = cls['get'](name) # look up name in class
        return (bind_method(value, instance) if callable(value) else value)



    instance = {'get': get_value, 'set': set_value} # dispatch dictionary
    return instance
def bind_method(function, instance):
```

## Instances

```
def make_instance(cls):
    attributes = {} # instance attributes, e.g. {'a': 6, 'b': 1}
    def get_value(name):
        if name in attributes: # name is an instance attribute
            return attributes[name]
        value = cls['get'](name) # look up name in class
        return (bind_method(value, instance) if callable(value) else value)



    instance = {'get': get_value, 'set': set_value} # dispatch dictionary
    return instance
def bind_method(function, instance):
    return lambda *args: function(instance, *args)
```

## Instances

```
def make_instance(cls):
    attributes = {} # instance attributes, e.g. {'a': 6, 'b': 1}
    def get_value(name):
        if name in attributes: # name is an instance attribute
            return attributes[name]
        value = cls['get'](name) # look up name in class
        return (bind_method(value, instance) if callable(value) else value)
    def set_value(name, value):


    instance = {'get': get_value, 'set': set_value} # dispatch dictionary
    return instance
def bind_method(function, instance):
    return lambda *args: function(instance, *args)
```

## Instances

```
def make_instance(cls):
    attributes = {} # instance attributes, e.g. {'a': 6, 'b': 1}
    def get_value(name):
        if name in attributes: # name is an instance attribute
            return attributes[name]
        value = cls['get'](name) # look up name in class
        return (bind_method(value, instance) if callable(value) else value)
    def set_value(name, value):
        attributes[name] = value # assignment creates/modifies instance attrs
    instance = {'get': get_value, 'set': set_value} # dispatch dictionary
    return instance
def bind_method(function, instance):
    return lambda *args: function(instance, *args)
```

## Classes

```
def make_class(attributes={}):
```

## Classes

```
def make_class(attributes={}):










    cls = {'get': get_value, 'set': set_value, 'new': __new__}
    return cls
```

## Classes

```
def make_class(attributes={}):

    def get_value(name):









    cls = {'get': get_value, 'set': set_value, 'new': __new__}
    return cls
```

## Classes

```
def make_class(attributes={}):

    def get_value(name):

        if name in attributes: # name is a class attribute
            return attributes[name]









    cls = {'get': get_value, 'set': set_value, 'new': __new__}
    return cls
```

## Classes

```
def make_class(attributes={}):

    def get_value(name):

        if name in attributes: # name is a class attribute
            return attributes[name]

        else: # AttributeError!
            return None





    cls = {'get': get_value, 'set': set_value, 'new': __new__}
    return cls
```

## Classes

```
def make_class(attributes={}):
    def get_value(name):
        if name in attributes: # name is a class attribute
            return attributes[name]
        else: # AttributeError!
            return None
    def set_value(name, value):



    cls = {'get': get_value, 'set': set_value, 'new': __new__}
    return cls
```

## Classes

```
def make_class(attributes={}):

    def get_value(name):

        if name in attributes: # name is a class attribute
            return attributes[name]

        else: # AttributeError!
            return None

    def set_value(name, value):

        attributes[name] = value



    cls = {'get': get_value, 'set': set_value, 'new': __new__}
    return cls
```

## Classes

```
def make_class(attributes={}):

    def get_value(name):

        if name in attributes: # name is a class attribute
            return attributes[name]

        else: # AttributeError!
            return None

    def set_value(name, value):

        attributes[name] = value

    def __new__(*args):
        # Returns an instance of this class.

    cls = {'get': get_value, 'set': set_value, 'new': __new__}
    return cls
```

## Instantiation and Initialization

## Instantiation and Initialization

1. Make a new instance of this class with `make_instance`

## Instantiation and Initialization

1. Make a new instance of this class with `make_instance`
2. Call the instance's `__init__` method

## Instantiation and Initialization

1. Make a new instance of this class with `make_instance`
2. Call the instance's `__init__` method

```
def make_class(attributes={}):
    ...
    def __new__(*args):




    ...
```

## Instantiation and Initialization

1. Make a new instance of this class with `make_instance`
2. Call the instance's `__init__` method

```
def make_class(attributes={}):
    ...
    def __new__(*args):
        instance = make_instance(cls)


    ...
```

## Instantiation and Initialization

1. Make a new instance of this class with `make_instance`
2. Call the instance's `__init__` method

```
def make_class(attributes={}):
    ...
    def __new__(*args):
        instance = make_instance(cls)
        return init_instance(instance, *args)
    ...
```

## Instantiation and Initialization

1. Make a new instance of this class with `make_instance`
2. Call the instance's `__init__` method

```
def make_class(attributes={}):
    ...
    def __new__(*args):
        instance = make_instance(cls)
        return init_instance(instance, *args)
    ...
def init_instance(instance, *args):
```

## Instantiation and Initialization

1. Make a new instance of this class with `make_instance`
2. Call the instance's `__init__` method

```
def make_class(attributes={}):
    ...
    def __new__(*args):
        instance = make_instance(cls)
        return init_instance(instance, *args)
    ...
def init_instance(instance, *args):
    init = instance['get']('__init__')
```

## Instantiation and Initialization

1. Make a new instance of this class with `make_instance`
2. Call the instance's `__init__` method

```
def make_class(attributes={}):
    ...
    def __new__(*args):
        instance = make_instance(cls)
        return init_instance(instance, *args)
    ...
def init_instance(instance, *args):
    init = instance['get']('__init__')
    if callable(init):
```

## Instantiation and Initialization

1. Make a new instance of this class with `make_instance`
2. Call the instance's `__init__` method

```
def make_class(attributes={}):
    ...
    def __new__(*args):
        instance = make_instance(cls)
        return init_instance(instance, *args)
    ...
def init_instance(instance, *args):
    init = instance['get']('__init__')
    if callable(init):
        init(*args)
```

## Instantiation and Initialization

1. Make a new instance of this class with `make_instance`
2. Call the instance's `__init__` method

```
def make_class(attributes={}):
    ...
    def __new__(*args):
        instance = make_instance(cls)
        return init_instance(instance, *args)
    ...
def init_instance(instance, *args):
    init = instance['get']('__init__')
    if callable(init):
        init(*args)
    return instance
```

## Example: Account

## Defining an Account Class

```
class Account:
```

```
def make_account_class():
```

```
Account = make_account_class()
```

## Defining an Account Class

```
class Account:
    interest = 0.02
```

```
def make_account_class():
    interest = 0.02
```

```
Account = make_account_class()
```

## Defining an Account Class

```
class Account:
    interest = 0.02

    def __init__(self, account_holder):
        self.balance = 0 # with setattr?
        self.holder = account_holder
```

```
def make_account_class():
    interest = 0.02

    def __init__(self, account_holder):
        self['set']('balance', 0)
        self['set']('holder', account_holder)
```

```
Account = make_account_class()
```

## Defining an Account Class

```
class Account:
    interest = 0.02

    def __init__(self, account_holder):
        self.balance = 0 # with setattr?
        self.holder = account_holder

    def deposit(self, amt):
        balance = self.balance + amt
        self.balance = balance
        return self.balance
```

```
def make_account_class():
    interest = 0.02

    def __init__(self, account_holder):
        self['set']('balance', 0)
        self['set']('holder', account_holder)

    def deposit(self, amt):
        balance = self['get']('balance') + amt
        self['set']('balance', balance)
        return self['get']('balance')
```

```
Account = make_account_class()
```

## Defining an Account Class

```
class Account:
    interest = 0.02

    def __init__(self, account_holder):
        self.balance = 0 # with setattr?
        self.holder = account_holder

    def deposit(self, amt):
        balance = self.balance + amt
        self.balance = balance
        return self.balance

    def withdraw(self, amt):
        balance = self.balance
        if amt > balance:
            return 'Insufficient funds'
        self.balance = balance - amt
        return self.balance
```

```
def make_account_class():
    interest = 0.02

    def __init__(self, account_holder):
        self['set']('balance', 0)
        self['set']('holder', account_holder)

    def deposit(self, amt):
        balance = self['get']('balance') + amt
        self['set']('balance', balance)
        return self['get']('balance')

    def withdraw(self, amt):




Account = make_account_class()
```

## Defining an Account Class

```
class Account:
    interest = 0.02

    def __init__(self, account_holder):
        self.balance = 0 # with setattr?
        self.holder = account_holder

    def deposit(self, amt):
        balance = self.balance + amt
        self.balance = balance
        return self.balance

    def withdraw(self, amt):
        balance = self.balance
        if amt > balance:
            return 'Insufficient funds'
        self.balance = balance - amt
        return self.balance
```

```
def make_account_class():
    interest = 0.02

    def __init__(self, account_holder):
        self['set']('balance', 0)
        self['set']('holder', account_holder)

    def deposit(self, amt):
        balance = self['get']('balance') + amt
        self['set']('balance', balance)
        return self['get']('balance')

    def withdraw(self, amt):
        balance = self['get']('balance')
        if amt > balance:
            return 'Insufficient funds'
        self['set']('balance', balance - amt)
        return self['get']('balance')


Account = make_account_class()
```

## Defining an Account Class

```python
class Account:
    interest = 0.02

    def __init__(self, account_holder):
        self.balance = 0 # with setattr?
        self.holder = account_holder

    def deposit(self, amt):
        balance = self.balance + amt
        self.balance = balance
        return self.balance

    def withdraw(self, amt):
        balance = self.balance
        if amt > balance:
            return 'Insufficient funds'
        self.balance = balance - amt
        return self.balance
```

```python
def make_account_class():
    interest = 0.02

    def __init__(self, account_holder):
        self['set']('balance', 0)
        self['set']('holder', account_holder)

    def deposit(self, amt):
        balance = self['get']('balance') + amt
        self['set']('balance', balance)
        return self['get']('balance')

    def withdraw(self, amt):
        balance = self['get']('balance')
        if amt > balance:
            return 'Insufficient funds'
        self['set']('balance', balance - amt)
        return self['get']('balance')

    return make_class(locals())
Account = make_account_class()
```

## Using the Account Class

## Using the Account Class

(demo)

## Goals

## Goals

- Object instantiation and initialization?

## Goals

- Object instantiation and initialization?

```
>>> brian_acct = Account['new']('Brian')
```

## Goals

- Object instantiation and initialization?

```
>>> brian_acct = Account['new']('Brian')
```

- Attribute lookup and assignment?

## Goals

- Object instantiation and initialization?

```
>>> brian_acct = Account['new']('Brian')
```

- Attribute lookup and assignment?

```
>>> brian_acct['get']('holder')
'Brian'
>>> brian_acct['set']('interest', 0.08)
```

## Goals

- Object instantiation and initialization?

```
>>> brian_acct = Account['new']('Brian')
```

- Attribute lookup and assignment?

```
>>> brian_acct['get']('holder')
'Brian'
>>> brian_acct['set']('interest', 0.08)
```

- Method invocation?

## Goals

- Object instantiation and initialization?

```
>>> brian_acct = Account['new']('Brian')
```

- Attribute lookup and assignment?

```
>>> brian_acct['get']('holder')
'Brian'
>>> brian_acct['set']('interest', 0.08)
```

- Method invocation?

```
>>> brian_acct['get']('withdraw')(5)
```

## Goals

- Object instantiation and initialization?

```
>>> brian_acct = Account['new']('Brian')
```

- Attribute lookup and assignment?

```
>>> brian_acct['get']('holder')
'Brian'
>>> brian_acct['set']('interest', 0.08)
```

- Method invocation?

```
>>> brian_acct['get']('withdraw')(5)
```

- Inheritance?

## Goals

- Object instantiation and initialization?

```
>>> brian_acct = Account['new']('Brian')
```

- Attribute lookup and assignment?

```
>>> brian_acct['get']('holder')
'Brian'
>>> brian_acct['set']('interest', 0.08)
```

- Method invocation?

```
>>> brian_acct['get']('withdraw')(5)
```

- Inheritance? ...not yet

## Implementing Inheritance

## Inheritance

What do we need to change when we implement inheritance?

- `get_value`
- `set_value`

## Inheritance

What do we need to change when we implement inheritance?

- `get_value`
- `set_value`

Which `get_value` do we need to change?

- `make_instance`
- `make_class`

## Implementing Inheritance: make_instance

```
def make_instance(cls):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        value = cls['get'](name) # look up name in class
        return bind_method(value, instance) if callable(value) else value
    ...
```

## Implementing Inheritance: make_instance

```
def make_instance(cls):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        value = cls['get'](name) # look up name in class
        return bind_method(value, instance) if callable(value) else value
    ...
```

No change necessary!

## Implementing Inheritance: make_class

```
def make_class(attributes={}):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        else:
            return None
    ...
```

## Implementing Inheritance: make_class

```
def make_class(attributes={}):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        else:
            return None
    ...
```

```
def make_class(attributes={},
               base_class=None):
```

## Implementing Inheritance: make_class

```
def make_class(attributes={}):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        else:
            return None
    ...
```

```
def make_class(attributes={},
               base_class=None):

    def get_value(name):
```

## Implementing Inheritance: make_class

```
def make_class(attributes={}):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        else:
            return None
    ...
```

```
def make_class(attributes={},
               base_class=None):

    def get_value(name):

        if name in attributes:
            return attributes[name]
```

## Implementing Inheritance: make_class

```
def make_class(attributes={}):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        else:
            return None
    ...
```

```
def make_class(attributes={},
               base_class=None):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        elif base_class is not None:
            return base_class['get'](name)
```

## Implementing Inheritance: make_class

```
def make_class(attributes={}):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        else:
            return None
    ...
```

```
def make_class(attributes={},
               base_class=None):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        elif base_class is not None:
            return base_class['get'](name)
        else:
            return None
    ...
```

## Using Inheritance

## Using Inheritance

```
class CheckingAccount(Account):
```

```
def make_checking_account_class():






    return make_class(locals(), Account)
CheckingAccount =
  make_checking_account_class()
```

## Using Inheritance

```
class CheckingAccount(Account):
    interest = 0.01
    withdraw_fee = 1
```

```
def make_checking_account_class():
    interest = 0.01
    withdraw_fee = 1









    return make_class(locals(), Account)
CheckingAccount =
  make_checking_account_class()
```

## Using Inheritance

```
class CheckingAccount(Account):
    interest = 0.01
    withdraw_fee = 1
    def withdraw(self, amount):
```

```
def make_checking_account_class():
    interest = 0.01
    withdraw_fee = 1
    def withdraw(self, amount):




    return make_class(locals(), Account)
CheckingAccount =
  make_checking_account_class()
```

## Using Inheritance

```python
class CheckingAccount(Account):

    interest = 0.01
    withdraw_fee = 1

    def withdraw(self, amount):

        fee = self.withdraw_fee
        return Account.withdraw(
          self, amount + fee
        )
```

```python
def make_checking_account_class():

    interest = 0.01
    withdraw_fee = 1

    def withdraw(self, amount):

        fee = self['get']('withdraw_fee')




    return make_class(locals(), Account)
CheckingAccount =
  make_checking_account_class()
```

## Using Inheritance

```
class CheckingAccount(Account):
    interest = 0.01
    withdraw_fee = 1
    def withdraw(self, amount):
        fee = self.withdraw_fee
        return Account.withdraw(
          self, amount + fee
        )
```

```
def make_checking_account_class():
    interest = 0.01
    withdraw_fee = 1
    def withdraw(self, amount):
        fee = self['get']('withdraw_fee')
        return Account['get']('withdraw')(
          self, amount + fee
        )
    return make_class(locals(), Account)
CheckingAccount =
  make_checking_account_class()
```

## Objects in Python

## Recap

## Recap

- We've implemented objects with dictionaries and functions!

## Recap

- We’ve implemented objects with dictionaries and functions!
- Who cares?

## The `__dict__` Attribute

## The `__dict__` Attribute

- A user-defined class automagically has a `__dict__` “attribute”

## The `__dict__` Attribute

- A user-defined class automagically has a `__dict__` “attribute”
- This attribute contains an object’s instance attributes!

## The `__dict__` Attribute

- A user-defined class automagically has a `__dict__` “attribute”
- This attribute contains an object’s instance attributes!

(demo)

## Recap

- We've implemented objects with dictionaries and functions!
- Who cares?
- When am I ever going to use this?

## JavaScript

Brendan Eich, creator of JavaScript

## JavaScript

Brendan Eich, creator of JavaScript

- How to create a language in 10 days.

## JavaScript

Brendan Eich, creator of JavaScript

- How to create a language in 10 days.
- Originally, a simple language for the Web.

## JavaScript

Brendan Eich, creator of JavaScript

- How to create a language in 10 days.
- Originally, a simple language for the Web.
- Now, one of the most commonly used languages in the world.

## JavaScript

Brendan Eich, creator of JavaScript

- How to create a language in 10 days.
- Originally, a simple language for the Web.
- Now, one of the most commonly used languages in the world.
- Object-oriented JavaScript? Dictionaries!