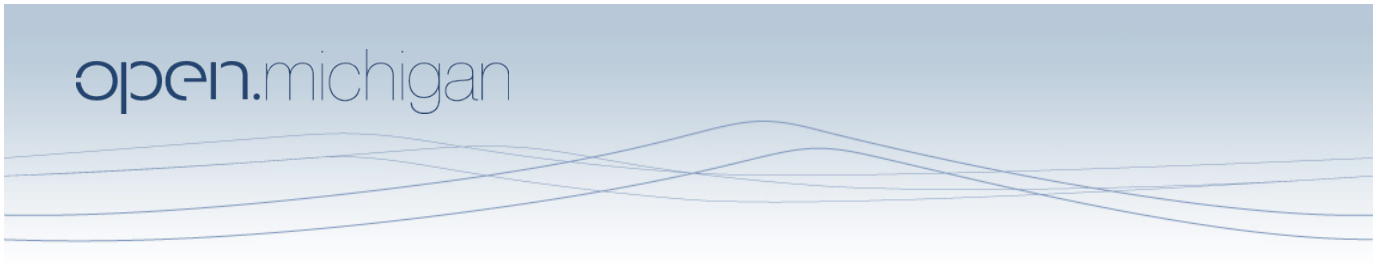
Python Objects

Jim Eng / Chuck Severance

http://www.python.org/doc/2.5.2/tut/node11.html
http://en.wikipedia.org/wiki/Object-oriented_programming



Unless otherwise noted, the content of this course material is licensed under a Creative Commons Attribution 3.0 License.
http://creativecommons.org/licenses/by/3.0/.

Copyright 2009-2010, Charles Severance, Jim Eng

Warning

- This lecture is very much about definitions and mechanics for objects
- This lecture is a lot more about "how it works" and less about "how you use it"
- You won't get the entire picture until this is all looked at in the context of a real problem
- So please suspend disbelief for the next 50 or so slides..

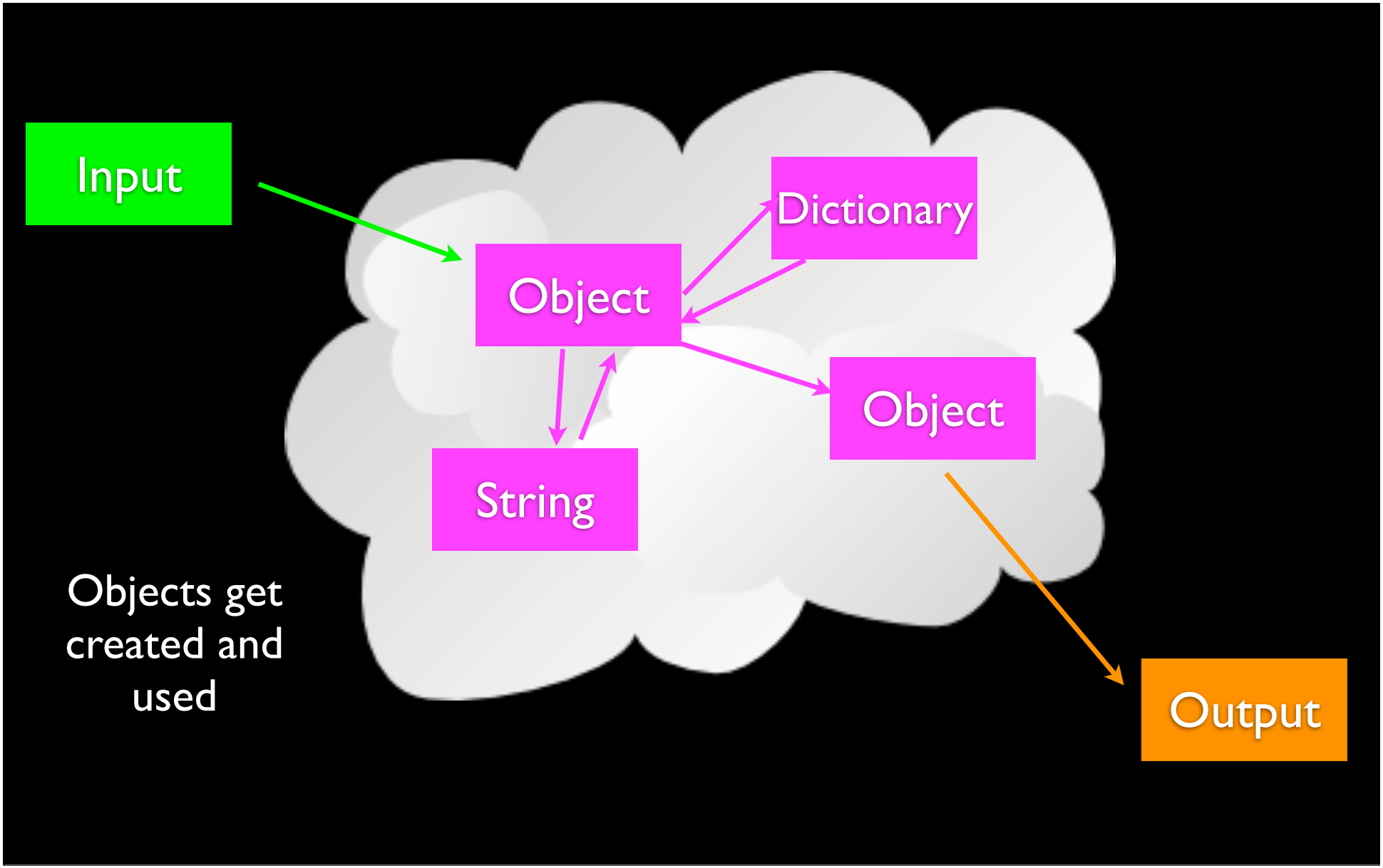
Review of Programs

```
usf = input('Enter the US Floor Number: ')
wf = usf - 1
print 'Non-US Floor Number is',wf
```

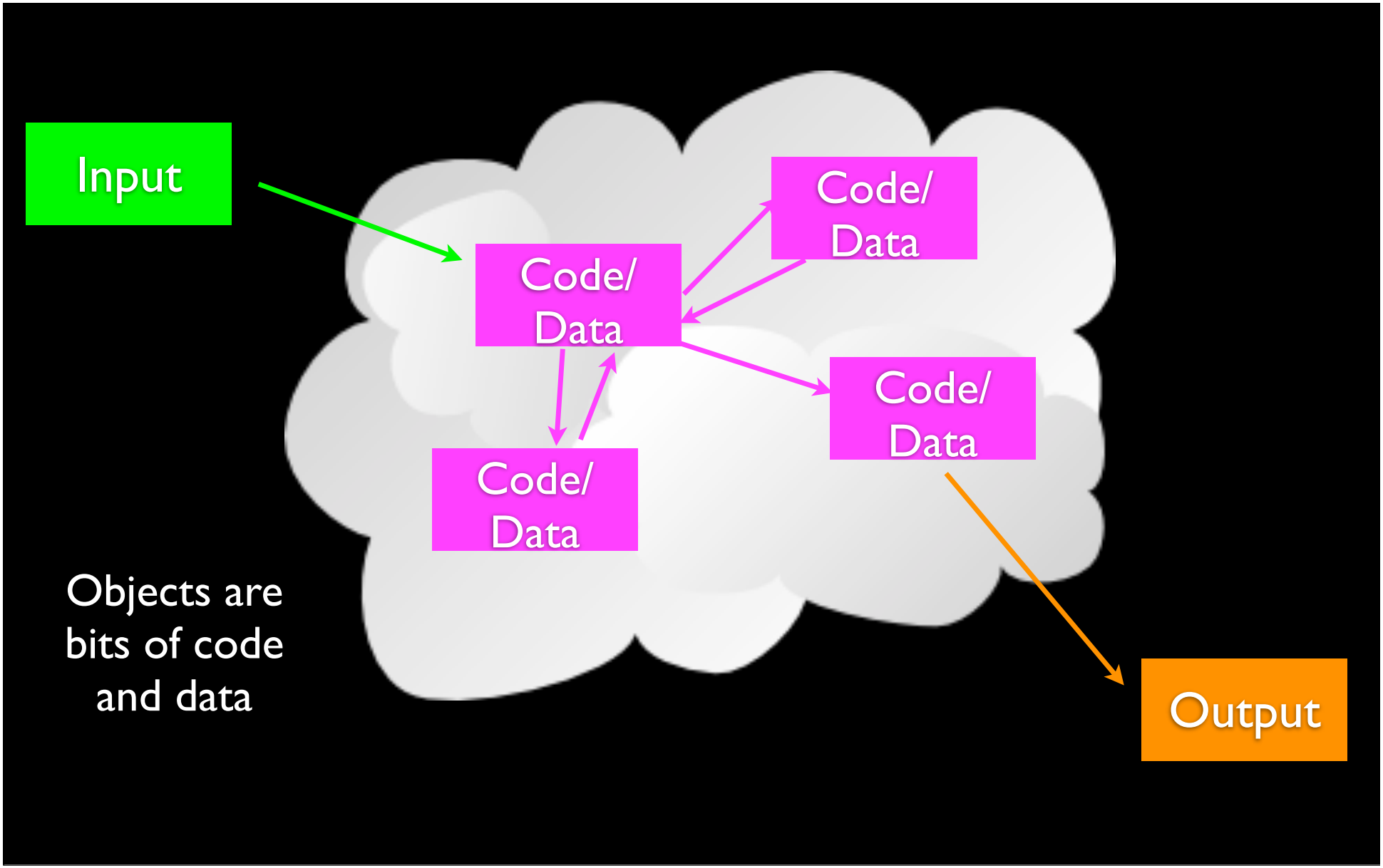
```
python elev.py
Enter the US Floor Number: 2
Non-US Floor Number is 1
```

Object Oriented

- A program is made up of many cooperating objects
- Instead of being the “whole program” - each object is a little “island” within the program and cooperatively working with other objects.
- A program is made up of one or more objects working together - objects make use of each other’s capabilities



Objects get created and used


Input
Code/
Data
Code/
Data
Code/
Data
Code/
Data
Output
Objects are
bits of code
and data

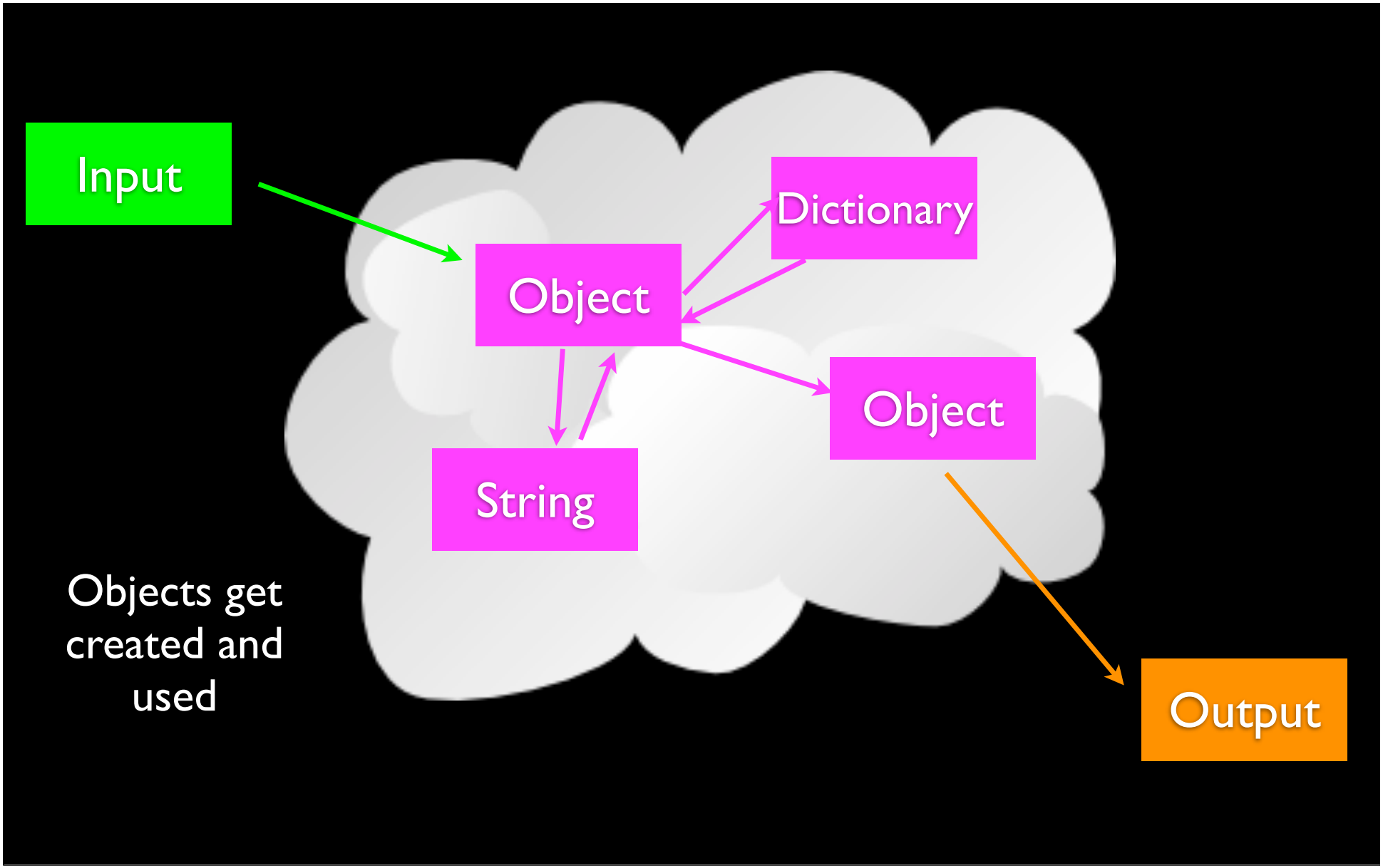
```
movies = list()
movie1  = dict()
movie1['Director'] = 'James Cameron'
movie1['Title'] = 'Avatar'
movie1['Release Date'] = '18 December 2009'
movie1['Running Time'] = '162 minutes'
movie1['Rating'] = 'PG-13'
movies.append(movie1)
movie2 = dict()
movie2['Director'] = 'David Fincher'
movie2['Title'] = 'The Social Network'
movie2['Release Date'] = '01 October 2010'
movie2['Running Time'] = '120 min'
movie2['Rating'] = 'PG-13'
movies.append(movie2)
```

```
movies = list()
movie1  = dict()
movie1['Director'] = 'James Cameron'
movie1['Title'] = 'Avatar'
movie1['Release Date'] = '18 December 2009'
movie1['Running Time'] = '162 minutes'
movie1['Rating'] = 'PG-13'
movies.append(movie1)
movie2 = dict()
movie2['Director'] = 'David Fincher'
movie2['Title'] = 'The Social Network'
movie2['Release Date'] = '01 October 2010'
movie2['Running Time'] = '120 min'
movie2['Rating'] = 'PG-13'
movies.append(movie2)
```

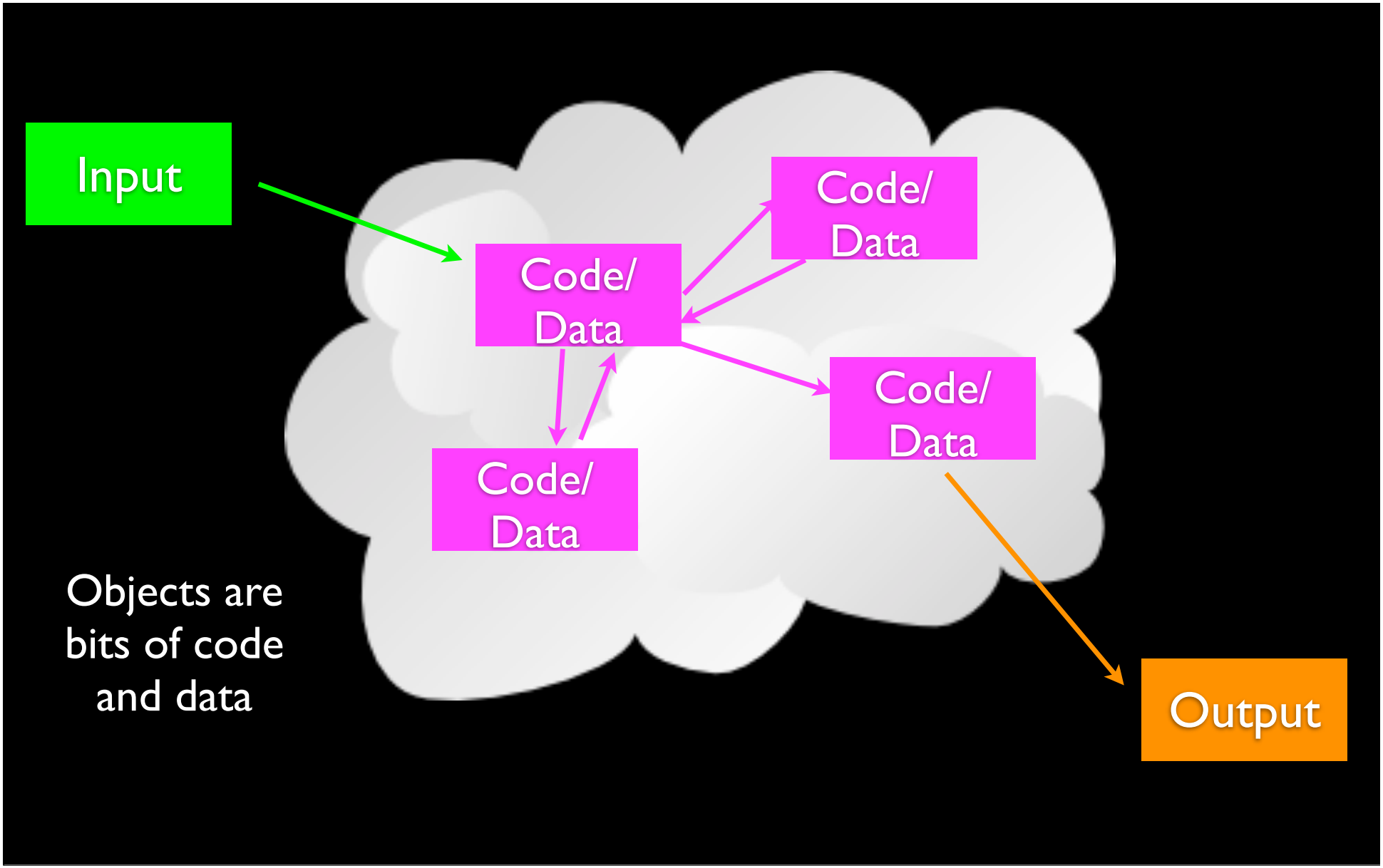
```
movies = list()
movie1  = dict()
movie1['Director'] = 'James Cameron'
movie1['Title'] = 'Avatar'
movie1['Release Date'] = '18 December 2009'
movie1['Running Time'] = '162 minutes'
movie1['Rating'] = 'PG-13'
movies.append(movie1)
movie2 = dict()
movie2['Director'] = 'David Fincher'
movie2['Title'] = 'The Social Network'
movie2['Release Date'] = '01 October 2010'
movie2['Running Time'] = '120 min'
movie2['Rating'] = 'PG-13'
movies.append(movie2)
```

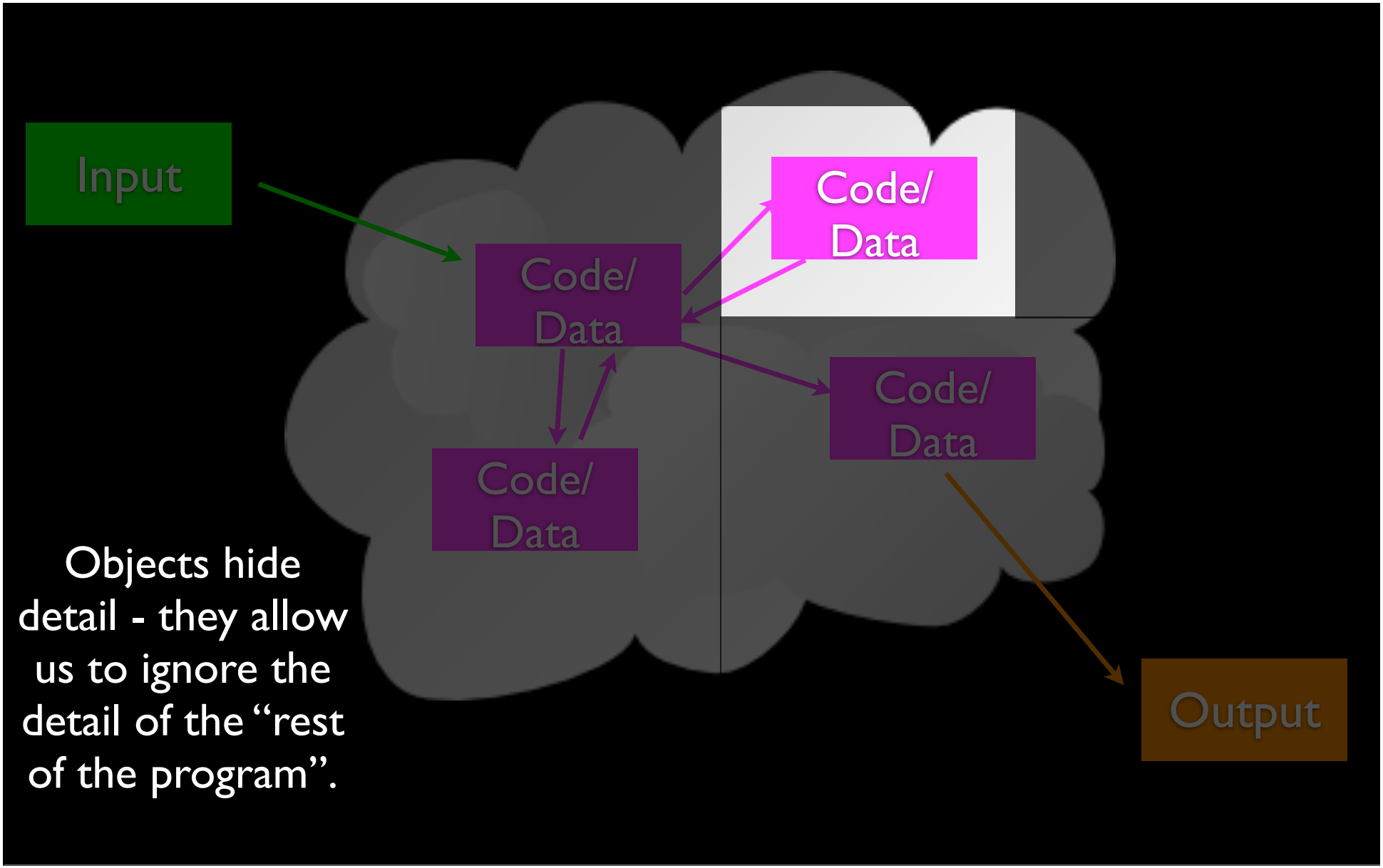
```
keys = ['Title', 'Director', 'Rating', 'Running Time']

print '-----------'
print movies
print '-----------'
print keys

for item in movies:
    print '-----------'
    for key in keys:
        print key,': ', item[key]

print '-----------'
```

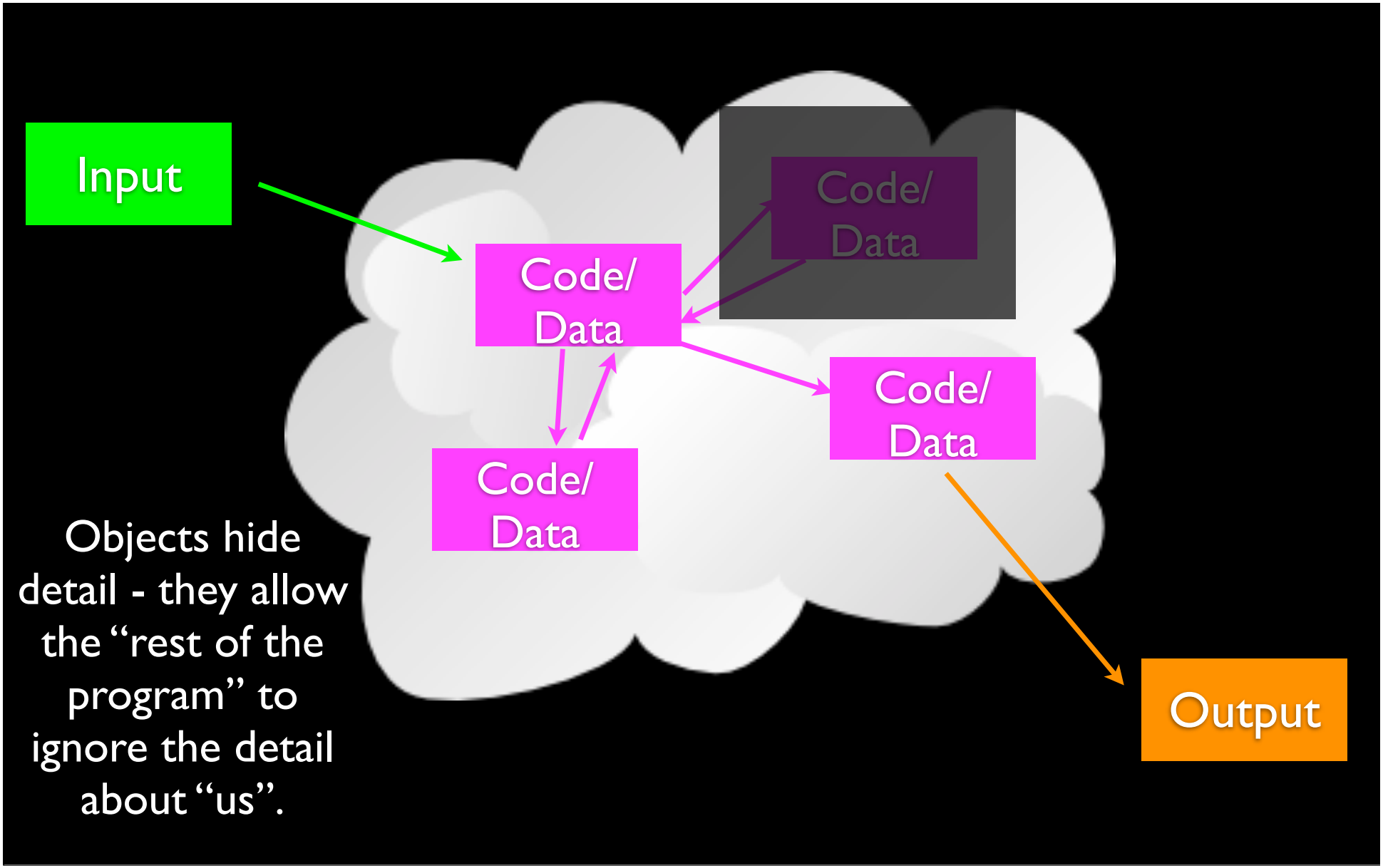
```
keys = ['Title', 'Director', 'Rating', 'Running Time']

print '----------'
print movies
print '----------'
print keys

for item in movies:
    print '----------'
    for key in keys:
        print key,': ', item[key]

print '----------'
```



Objects get created and used


Input
Code/
Data
Code/
Data
Code/
Data
Code/
Data
Output
Objects are
bits of code
and data


Input
Code/
Data
Code/
Data
Code/
Data
Code/
Data
Output
Objects hide detail - they allow us to ignore the detail of the "rest of the program".


Input
Code/
Data
Code/
Data
Code/
Data
Code/
Data
Output
Objects hide detail - they allow the "rest of the program" to ignore the detail about "us".

Object

- An Object is a bit of self-contained Code and Data
- A key aspect of the Object approach is to break the problem into smaller understandable parts (divide and conquer)
- Objects have boundaries that allow us to ignore un-needed detail
- We have been using objects all along: String Objects, Integer Objects, Directory Objects, List Objects...

Definitions

- Class - a template - Dog
- Method or Message - A defined capability of a class - bark()
- Object or Instance - A particular instance of a class - Lassie

Terminology: Class

Defines the abstract characteristics of a thing (object), including the thing's characteristics (its attributes, fields or properties) and the thing's behaviors (the things it can do, or methods, operations or features). One might say that a class is a blueprint or factory that describes the nature of something. For example, the class Dog would consist of traits shared by all dogs, such as breed and fur color (characteristics), and the ability to bark and sit (behaviors).

http://en.wikipedia.org/wiki/Object-oriented_programming

Terminology: Class

A pattern (exemplar) of a class. The class of Dog defines all possible dogs by listing the characteristics and behaviors they can have; the object Lassie is one particular dog, with particular versions of the characteristics. A Dog has fur; Lassie has brown-and-white fur.

http://en.wikipedia.org/wiki/Object-oriented_programming

Terminology: Instance

One can have an instance of a class or a particular object. The instance is the actual object created at runtime. In programmer jargon, the Lassie object is an instance of the Dog class. The set of values of the attributes of a particular object is called its state. The object consists of state and the behavior that's defined in the object's class.

Object and Instance are often used interchangeably.

http://en.wikipedia.org/wiki/Object-oriented_programming

Terminology: Method

An object's abilities. In language, methods are verbs. Lassie, being a Dog, has the ability to bark. So bark() is one of Lassie's methods. She may have other methods as well, for example sit() or eat() or walk() or save_timmy(). Within the program, using a method usually affects only one particular object; all Dogs can bark, but you need only one particular dog to do the barking

Method and Message are often used interchangeably.

http://en.wikipedia.org/wiki/Object-oriented_programming

A Sample Class

class is a reserved word.

Each PartyAnimal object has a bit of code.

Tell the object to run the party () code.

```
class PartyAnimal:
   x = 0

   def party(self) :
     self.x = self.x + 1
     print "So far",self.x

an = PartyAnimal()

an.party()
an.party()
an.party()
```

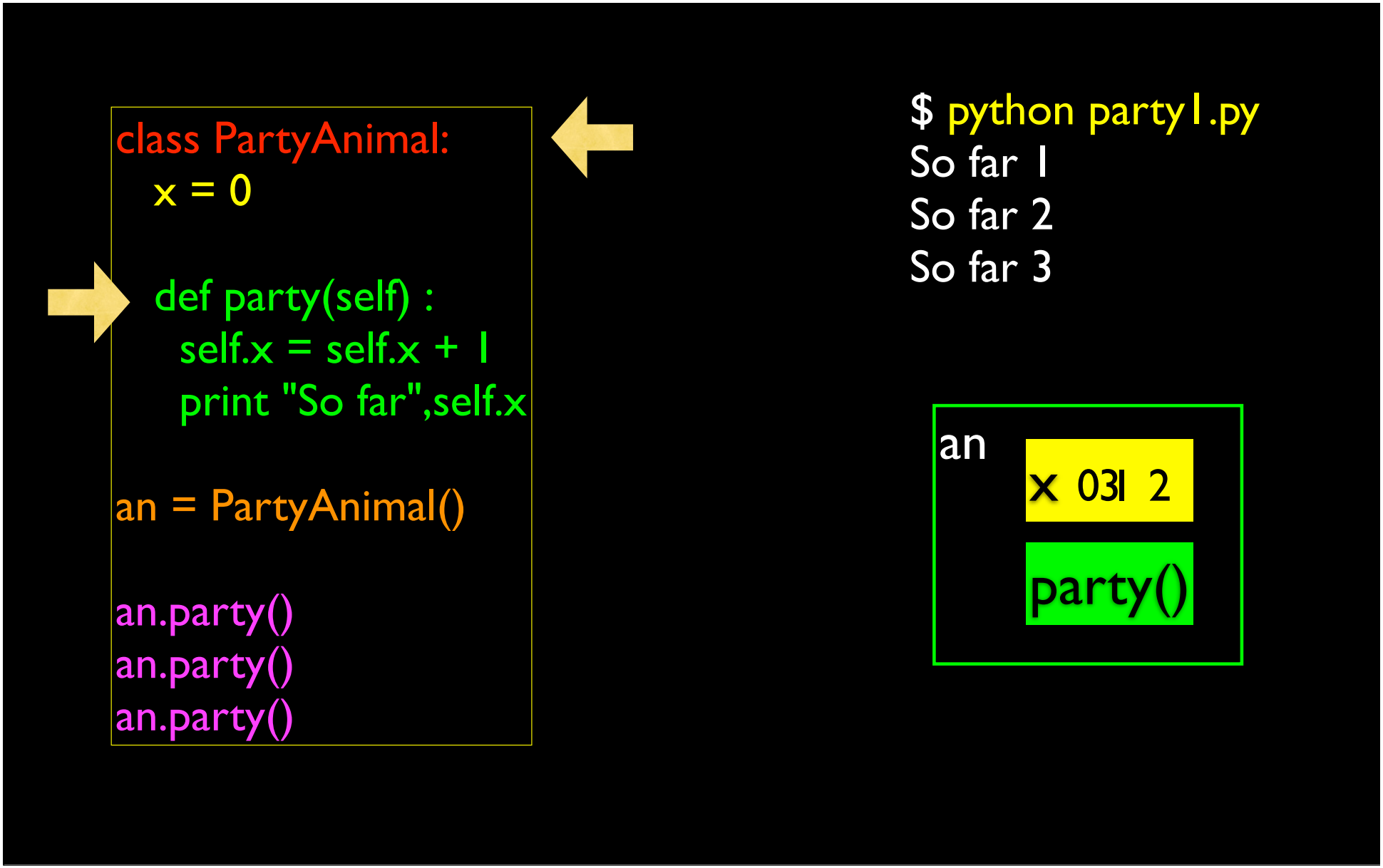
This is the template for making PartyAnimal objects.

Each PartyAnimal object has a bit of data.

Create a PartyAnimal object.

PartyAnimal.party(an)

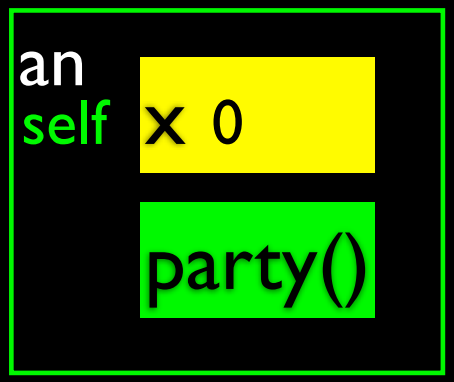
run party() *within* the object an


class PartyAnimal:
x = 0
def party(self) :
self.x = self.x + 1
print "So far",self.x
an = PartyAnimal()
an.party()
an.party()
an.party()
$ python party1.py
So far 1
So far 2
So far 3
an
x 0 1 2
party()

```
class PartyAnimal:
   x = 0

   def party(self) :
     self.x = self.x + 1
     print "So far",self.x

an = PartyAnimal()

an.party()
an.party()
an.party()
```

"self" is a formal argument that refers to the object itself.

self.x is saying "x within self"



self is "global within this object"

Definitions

- Class - a template - Dog
- Method or Message - A defined capability of a class - bark()
- Object or Instance - A particular instance of a class - Lassie

Playing with dir() and type()

A Nerdy Way to Find Capabilities

- The dir() command lists capabilities
  - Ignore the ones with underscores - these are used by Python itself
  - The rest are real operations that the object can perform
- It is like type() - it tells us something *about* a variable

```
>>> x = list()
>>> type(x)
<type 'list'>
>>> dir(x)
['__add__', '__class__',
'__contains__', '__delattr__',
'__delitem__', '__delslice__',
'__doc__', '__eq__',
'__setitem__', '__setslice__',
'__str__', 'append', 'count',
'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
>>>
```

Try dir() with a String

```
>>> y = “Hello there”
>>> dir(y)
['__add__', '__class__', '__contains__', '__delattr__',
'__doc__', '__eq__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__getslice__', '__gt__',
'__hash__', '__init__', '__le__', '__len__', '__lt__', '__repr__',
'__rmod__', '__rmul__', '__setattr__', '__str__', 'capitalize',
'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find',
'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

```
class PartyAnimal:
   x = 0

   def party(self) :
     self.x = self.x + 1
     print "So far",self.x

an = PartyAnimal()

print "Type", type(an)
print "Dir ", dir(an)
```

We can use dir() to find the "capabilities" of *our* newly created class.

```
$ python party2.py
Type <type 'instance'>
Dir  ['__doc__', '__module__',
'party', 'x']
```

Object Life Cycle

http://en.wikipedia.org/wiki/Constructor_(computer_science)

Object Life Cycle

- Objects are created, used and discarded
- We have special blocks of code (methods) that get called
  - At the moment of creation (constructor)
  - At them moment of destruction (destructor)
- Constructors are used a lot
- Destructors are seldom used

Constructor

- The primary purpose of the constructor is to set up some instance variables to have the proper initial values when the object is created

```
class PartyAnimal:
   x = 0

   def __init__(self):
     print "I am constructed"

   def party(self) :
     self.x = self.x + 1
     print "So far",self.x

   def __del__(self):
     print "I am destructed", self.x

an = PartyAnimal()
an.party()
an.party()
an.party()
```

```
$ python party2.py
I am constructed
So far 1
So far 2
So far 3
I am destructed 3
```

The constructor and destructor are optional. The constructor is typically used to set up variables. The destructor is seldom used.

Constructor

- In object-oriented programming, a constructor in a class is a special block of statements called when an object is created

http://en.wikipedia.org/wiki/Constructor_(computer_science)

Many Instances

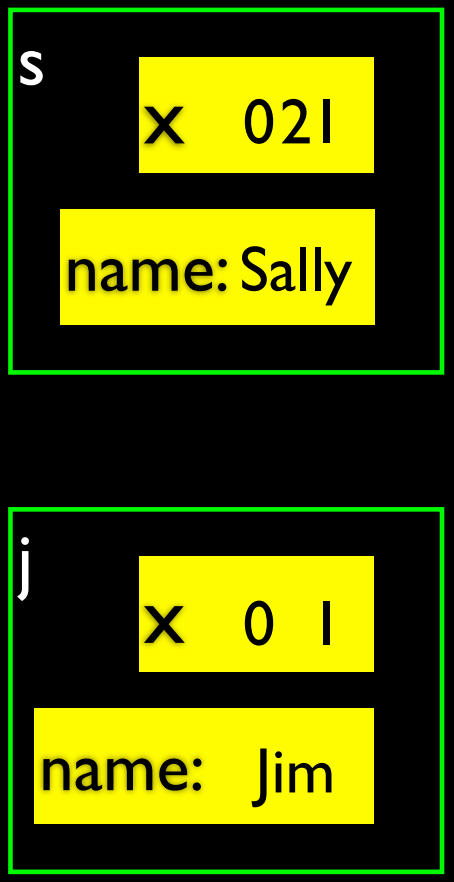
- We can create lots of objects - the class is the template for the object
- We can store each distinct object in its own variable
- We call this having multiple instances of the same class
- Each instance has its own copy of the instance variables

```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print self.name,"constructed"

   def party(self) :
     self.x = self.x + 1
     print self.name,"party count",self.x

s = PartyAnimal("Sally")
s.party()

j = PartyAnimal("Jim")
j.party()
s.party()
```

Constructors can have additional parameters. These can be used to setup instance variables for the particular instance of the class (i.e. for the particular obect).

```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, z):
     self.name = z
     print self.name,"constructed"

   def party(self) :
     self.x = self.x + 1
     print self.name,"party count",self.x

s = PartyAnimal("Sally")
s.party()

j = PartyAnimal("Jim")
j.party()
s.party()
```

We have two independent instances.



PartyAnimal.party(j)

Definitions

- Class - a template - Dog
- Method or Message - A defined capability of a class - bark()
- Object or Instance - A particular instance of a class - Lassie
- Constructor - A method which is called when the instance / object is created

Inheritance

http://www.python.org/doc/2.5.2/tut/node11.html
http://www.ibiblio.org/g2swap/byteofpython/read/inheritance.html

Inheritance

- When we make a new class - we can reuse an existing class and inherit all the cababilities of an existing class and then add our own little bit to make our new class
- Another form of store and reuse
- Write once - reuse many times
- The new class (child) has all the capabilities of the old class (parent) - and then some more

Terminology: Inheritance

‘Subclasses’ are more specialized versions of a class, which inherit attributes and behaviors from their parent classes, and can introduce their own.

http://en.wikipedia.org/wiki/Object-oriented_programming

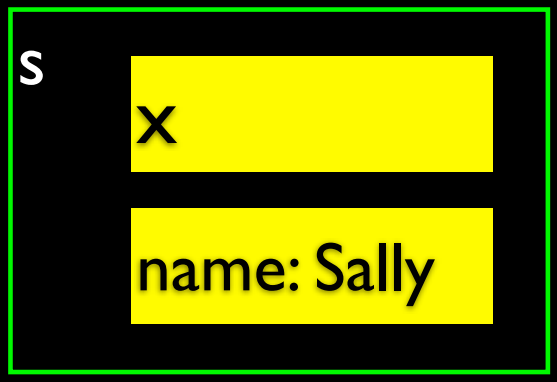
```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print self.name,"constructed"

   def party(self) :
     self.x = self.x + 1
     print self.name,"party count",self.x

class FootballFan(PartyAnimal):
   points = 0
   def touchdown(self):
      self.points = self.points + 7
      self.party()
      print self.name,"points",self.points
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```

FootballFan is a class which extends PartyAnimal. It has all the capabilities of PartyAnimal and more.

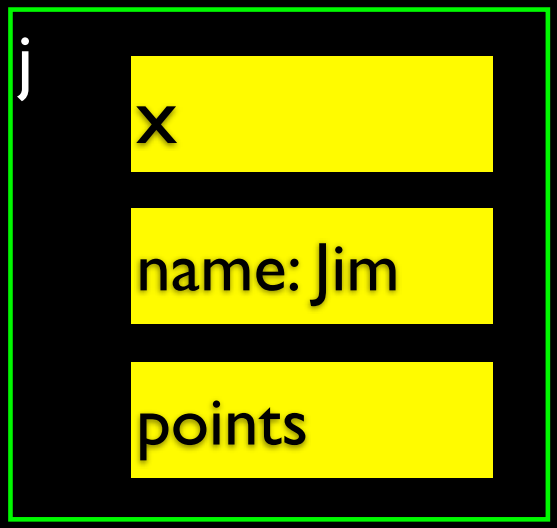
```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print self.name,"constructed"

   def party(self) :
     self.x = self.x + 1
     print self.name,"party count",self.x

class FootballFan(PartyAnimal):
   points = 0
   def touchdown(self):
      self.points = self.points + 7
      self.party()
      print self.name,"points",self.points
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```



```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print self.name,"constructed"

   def party(self) :
     self.x = self.x + 1
     print self.name,"party count",self.x

class FootballFan(PartyAnimal):
   points = 0
   def touchdown(self):
      self.points = self.points + 7
      self.party()
      print self.name,"points",self.points
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```



Definitions

- Class - a template - Dog
- Method or Message - A defined capability of a class - bark()
- Object or Instance - A particular instance of a class - Lassie
- Constructor - A method which is called when the instance / object is created
- Inheritance - the ability to take a class and extend it to make a new class.

Summary

- Object Oriented programming is a very structured approach to code reuse.
- We can group data and functionality together and create many independent instances of a class

Questions...