03-0: **Python Objects**

- Simpliest possible object:

  ```
  class simple:
      pass
  ```

- pass is python's no-op

  - needed because there are no ;, {}, etc.

- We can create an instance variable by assinging a value

  ```
  >>> x = simple()
  >>> x.instanceVar = 7
  >>> x.instanceVar
  7
  ```

03-1: **Methods & Constructor**

```
class simple:
   def __init__(self, initVal = 0):
      self.instance = initVal

   def inc(self)
      self.instance = self.instance + 1
```

- All methods take explicit "self" parameter

- Access instance variables, other methods through self

- Constructor is method named __init__

03-2: **"Private" instance vars**

- Can make a variable (or method) private by starting with __, (and not ending with __)

- Not *really* private, name mangling

  - _classname is appended outside of class context
  - prevents mistakes, not malice

03-3: **Class varaibles**

- One instance per class

- Don't need an instance to access

- Handy for constants, etc

```
class foo:
    classVar = 3

>>> foo.classVar
3
>>> x = foo()
>>> x.classVar
3
```

03-4: **Built-in Methods**

- Can override default methods / operators:

    - `__repr__` how a method is printed

    - `__lt__`, `__gt__`, `__le__`, `__gt__`, `__cmp__` comparison ops

    - `__add__`, `__sub__`, `__mul__`, `__div__`: arithmetic ops

03-5: **Inheritance**

```
class point:
   def __init__(this, x = 0, y = 0):
      this.x = x
      this.y = y

   def __repr__(this):
      return "(" + str(this.x) + "," + str(this.y) + ")"

class circle(point):
   def __init__(this, x = 0, y = 0, radius = 0):
      point.__init__(this, x, y)
      this.radius = radius

   def __repr__(this):
      return "center:" + point.__repr__(this) + \
         ", radius = " + str(this.radius)
```

03-6: **List Comprehensions**

   [f(x) for x in L]

```
>>> [x*x for x in range(1,10)]
[1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> x = ["hello", "there"]
>>> [y.upper() for y in x]
['HELLO', 'THERE']
```

03-7: **List Comprehensions**

   [f(x) for x in L if test]

```
>>> [x for x in range(1,10) if x % 2 == 0]
[2, 4, 6, 8]
>>> [x for x in range(20)) if prime(x)]
 [2, 3, 5, 7, 11, 13, 17, 19]
>>> d = {"dog" : 1, "cat" : 2, "mouse" : 3}
>>> [x for (x,y) in d.items()]
>>> [y for (x,y) in d.items()]
>>> dict([(y,x) for (x,y) in d.items()])
```

03-8: **Functions as Data**

```
def cube(x):
   return x * x * x


def myMap(f, L):
   result = []
   for (x) in L:
      result.append(f(x))
   return result
```

03-9: **Introspection**

- Can ask a function for documentation

  - help(foo)
  - `foo.__doc__`
  - dir(foo)
  - type(foo)

03-10: **Exception Handling**

- Python also has execptions

- Much like Java / C++

```
try:
   fsock = open("/BadFileName")
except IOError:
   print "could not find the file"

try:
   x = d["badKey"]
except KeyError:
   print "key does not exist"
```

03-11: **Final Thoughts**

- Remember to always use self to reference member variables and other methods from within objects

- Use help() and dir() for quick documentation

  - Especially helpful for strings & such

- When at all possible, use iterators and "in" instead of doing loops yourself

  - Don't code like a C programmer: Harder to read, and less efficient!