Classes

Special thanks to Roy McElmurry, Scott Shawcroft, Ryan Tucker, and Paul Beck for their work on these slides.

Exceptions

```
raise type(message)
raise Exception(message)
```

Exceptions
AssertionError
TypeError
NameError
ValueError
IndexError
SyntaxError
ArithmeticError

http://docs.python.org/library/exceptions.html#bltin-exceptions

Class Syntax

- Recall the syntax for making a basic class

example.py

```
1  class ClassName:
2      def __init__(self, params, ...):
3          self.field1 = value
4          self.fieldn = value
5          #Your code here
6      def other_methods(self, params, ...):
7          #your code here
8
9
```

Inheritance

- Python has multiple inheritance
- This means that we can create a class that subclasses several classes
- Python makes an effort to mix super classes
  - Searches super classes from left to right
  - We can disambiguate if there are problems with this

example.py

```
1  class ClassName(SuperClass1, SuperClass2, ...):
2      def __init__(self, params, ...):
```

Commenting Your Classes

- Classes and functions have a built-in field called __doc__
- We can use this as a way to get more bang for our comments
- These __doc__ fields could be used like JavaDoc

example.py

```
1  class Point():
2      """This class defines a point in 2D space"""
3      def __init__(self, x, y):
4          """Post: returns a Point with the given x and y fields"""
```

Name Mangling

- Python does not have private methods
- Python does have name mangling, any method that starts with 2+ underscores and does not end in 2+ underscores with be renamed to _classname__method

example.py

```
1  class Foo():
2      def __init__(self):
3          self.__helper()
4      def __helper(self):
5          print("sneaky")
6
7  x = Foo()                    #output: sneaky
8  x._Foo__helper()             #output: sneaky
9  x.__helper()                 #output: AttributeError
```

Static Fields

- There is a subtle difference between declaring fields in the class and declaring them in the constructor
- Fields defined in the class can be used as static variables, meaning they belong to the class as a whole

example.py

```
1  class MovieTicket():
2      basePrice = 10
3      def __init__(self, fee):
4          self.price = self.basePrice + fee
5  x = MovieTicket(5)
6  print(x.price)                          #result: 15
7  print(MovieTicket.basePrice)            #result: 10
```

Static Methods

- We can use decorators to tell our function to be static, meaning they belong to the class, not an instance

example.py

```
1  class Point():
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5      @staticmethod
6      def distance(p1, p2):
7          d = sqrt((p1.x - p2.x)**2 + (p1.y - p2.y)**2 )
8          return d
9  x = Point(0, 0)
10 y = Point(0, 5)
11 print(Point.distance(x, y))                    #result: 5
```

Class Methods

- A class method receives a reference to the class instead of a reference to an instance
- You can use this class parameter (cls) to reference the static variables or methods
- One use of this ability is writing documentation methods

Class Methods

example.py

```
1  class Point():
2      """This class defines a point in 2D space."""
3      def __init__(self, x, y):
4          """Post: returns a Point with coordinates (x,y)"""
5          self.x = x
6          self.y = y
7      @classmethod
8      def help(cls):
9          for attr in cls.__dict__:
10           print(str(attr) + ": " + cls.__dict__
11                   [attr].__doc__)#result: 5
12
13 x = Point(0, 0)
14 x.help()
```

__str__()

- We already know about the __str__() method that allows a class to convert itself into a string

rectangle.py

```
1  class Point:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6      def __str__(self):
7          return "(" + str(self.x) + ", " +
8              str(self.y) + ")"
9
```

First Class Citizens

- For built-in types like `ints` and `strings` we can use operators like + and *.
- Our classes so far were forced to take back routes and use methods like `add()` or `remove()`
- Python is super cool, in that it allows us to define the usual operators for our class
- This brings our classes up to first class citizen status just like the built in ones

Underscored methods

- There are many other underscored methods that allow the built-in function of python to work
- Most of the time the underscored name matches the built-in function name

Built-In	Class Method
`str()`	`__str__()`
`len()`	`__len__()`
`abs()`	`__abs__()`

Underscored methods

- There are underscore methods that you can implement in order to define logical operations and arithmetic operations

Binary Operators

Operator	Class Method
-	`__sub__(self,other)`
+	`__add__(self, other)`
*	`__mul__(self, other)`
/	`__truediv__(self, other)`

Unary Operators

Operator	Class Method
-	`__neg__(self)`
+	`__pos__(self)`

Comparison Operators

Operator	Class Method
==	`__eq__(self,other)`
!=	`__ne__(self, other)`
<	`__lt__(self, other)`
>	`__gt__(self, other)`
<=	`__le__(self, other)`
>=	`__ge__(self, other)`
N/A	`__nonzero__(self)`

http://docs.python.org/reference/datamodel.html#sequence-types

Vector Class

Lets write a class that represents a Vector. A Vector is a Point that has some extra functionality. We should be able to add and subtract two Vectors, determine if two Vectors are equal. We should be able to multiply a Vector by a scalar and ask what the Vector's length is as an integer. In addition, Vectors should have these methods and fields.

Method/Field	**Functionality**
`origin`	`The origin as a field`
`isDiagonalInPointSet()`	`Returns whether this Vector lies on the diagonal and is contained in the given point set`
`slope()`	`Returns the slope between the two given Vectors`

python™