Accelerating Information Technology Innovation

http://aiti.mit.edu

Nigeria Summer 2012
Lecture 6– Objects

MISTI

The History of Objects

- Objects weren't always supported by programming languages
- Idea first originated at MIT in the 1960s and was officially incorporated in a few languages in the same decade
- OOP (Object Oriented Programming) has now become a core feature of nearly all languages

Object Oriented Programming (OOP)

- A certain style of computer programming
- Centered around data structures called “objects”
- Many pros and cons, but almost every language and decent sized project uses it

What is an Object?

- A standard way to organize information (data)
- Holds similar information about a single “thing” in one place
- For example, in a soccer tournament, a “tournament” object could hold:
- A list of teams and points of teams
- The name of the tournament
- A list of stadiums
- A procedure to make a new game by picking the teams and stadium
- In fact, all the data structures you've learned as well as procedures are also objects (lists, strings, dictionaries)

The String Object

```
original_string = '    some text    ' #instantiate a string object
#original_string = str( '    some text    ' ) is equivalent to the above line
# remove leading and trailing whitespace by calling string's strip method
string1 = original_string.strip()

# make uppercase
string2 = string1.upper()
print string2 #SOME TEXT

# make lowercase
string2.lower() == string1
True
```

Pointers/References

```
a = 5
b = a
a = 4
print b # 5
c = [5]
d = c # point to the same object as c
c[0] = 8
print d[0] # 8
e = [2]
f = e[:] # make a copy of e
e[0] = 6
print f[0] # 2
```

Defining a Class

```
class Car():
    wheels = 4
print Car.wheels #4
myCar = Car() #instantiation
print myCar.wheels #4
Car.wheels = 5 # change the class variable
print Car.wheels #5
print myCar.wheels #5
```

The Constructor

```
class Car():
    wheels = 4
    def __init__(self, color):
        self.color = color
#print Car.color <-- AttributeError: class Car
   has no attribute 'color'
myCar = Car("red")
print myCar.color # red
```

Adding Procedures

```
class Car():
    wheels = 4
    def __init__(self, color):
        self.color = color
    def fade(self):
        self.color = self.color + "ish"
myCar = Car("red")
print myCar.color #red
myCar.fade()
print myCar.color #redish
```

Static Procedures

```
class Car():
    wheels = 4
    def __init__(self, color):
        self.color = color
    def fade(self):
        self.color = self.color + "ish"

    @staticmethod
    def isOld(miles):
```

Inner Classes

```
class Car():
    wheels = 4
    def __init__(self, color, horsepower):
        self.color = color
        self.engine = self.Engine(horsepower)
    class Engine():
        def __init__(self, horsepower):
            self.horsepower = horsepower
        def getWatts(self):
            return self.horsepower * 745.7

myCar = Car('red', 400)
print myCar.engine.getWatts() #298280.0
```

Instance vs Class Variables

```
class Person():
    eyes = 2
    fingers = 10
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def setFingers(self):
        self.fingers = 9
    def is_old(self):
        return self.age > 40
```

```
(person1, person2) =
    Person('Larry', 70),
    Person('Doug', 20)

print person1.eyes, person2.eyes
    #2 2

Person.eyes = 3

print person1.eyes #3 because
    person1 doesn't own it's own
    eyes variable

print person2.eyes #3 because
    person2 doesn't own it's own
    eyes variable

####
```

Instance vs Class Variables

print person1.fingers, person2.fingers #10 10

person1.fingers = 9

print person1.fingers #9 because person1 owns it's own fingers variable and it was changed

print person2.fingers #10 because person2's fingers variable wasn't changed

#####

Person.age = 5

print person1.age #70 because person1 has it's own age variable

del person1.age # delete person1's age variable

print person1.age #5