

## User defined data types: Classes

### What is a class?

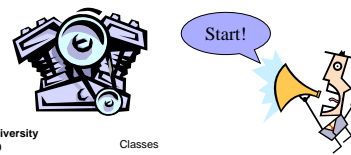
- If you have done anything in computer science before, you likely will have heard the term object oriented programming (OOP)
- What is OOP, and why should I care?

### Short answer

- The short answer is that object oriented programming is a way to think about “objects” in a program (such as variables, functions, etc)
- A program becomes less a list of instruction and more a set of objects and how they interact

### Responding to “messages”

- As a set of interacting objects, each object responds to “messages” sent to it
- The interaction of objects via messages makes a high level description of what the program is doing.



### Everything in Python is an object

- in case you hadn't noticed, everything in Python is an object
- Thus Python embraces OOP at a fundamental level

### OOP helps for software engineering

- software engineering is the discipline of managing code to ensure its long-term use
- remember, SE via refactoring
- refactoring:
  - takes existing code and modifies it
  - makes the overall code simpler, easier to understand
  - doesn't change the functionality, only the form!

## More refactoring

- Hiding the details of what the message entails means that changes can be made to the object and the flow of messages (and their results) can stay the same
- Thus the implementation of the message can change but its intended effect stay the same.
- This is **encapsulation**



Michigan State University  
CSE 231, Fall 2009

Classes

## OOP principles

- encapsulation: hiding design details to make the program clearer and more easily modified later
- modularity: the ability to make objects "stand alone" so they can be reused (our modules). Like the math module
- inheritance: create a new object by inheriting (like father to son) many object characteristics while creating or over-riding for this object
- polymorphism: (hard) Allow one message to be sent to any object and have it respond appropriately based on the type of object it is.



Michigan State University  
CSE 231, Fall 2009

Classes

## Class versus instance

- One of the harder things to get is what a class is and what an instance of a class is.
- The analogy of the cookie cutter and a cookie.



Michigan State University  
CSE 231, Fall 2009

Classes

## Template vs exemplar

- The cutter is a template for "stamping out" cookies, the "cookie" is what is made each time the cutter is used
- One template can be used to make an infinite number of cookies, each one just like the other.
- No one confuses a cookie for a cookie cutter, do they?



Michigan State University  
CSE 231, Fall 2009

Classes

## Same in OOP

- You define a class as a way to generate new instances of that class.
- Both the instances and the classes are themselves objects
- the structure of an instance starts out the same, as dictated by the class.
- The instances respond to the messages defined as part of the class.



Michigan State University  
CSE 231, Fall 2009

Classes

## OK, a different tack

- Ever hear of Plato?
  - the "second" western philosopher
  - 428-347 B.C, in Greece (Athens)
  - student of Socrates (the "first" western philosopher), teacher of Aristotle
  - prolific writer and first of the Greek philosophers we have writings from



Michigan State University  
CSE 231, Fall 2009

Classes

## Theory of Forms

- Plato worried about the problem of universals.
- For example, “Have you ever seen a perfect circle?”
- Probably not, nearly any circle I were to draw (even given great care) would be flawed in some way



Michigan State University  
CSE 231, Fall 2009

Classes

## So what is “circleness”

- So if you have never seen a perfect circle, how do you know one when you see one?
- We assume that somehow we can compare our experiences to some “universal” (a perfect circle), but how can we compare to something we have never seen?



Michigan State University  
CSE 231, Fall 2009

Classes

## Forms

- Plato assume there must be some universal concepts that we, as cognitive humans, could “participate” in (we could think about) that allows us to compare particulars to universals.
- He called these universals the Forms



Michigan State University  
CSE 231, Fall 2009

Classes

## And, um, how does that help?

- In a way, OOP is related to a Form view of the world
  - instances are made in the image of its class
  - though an instance may change (no longer be a perfect class instance) it is still an instance of the class (its form)
- See, ancient philosophy comes to the rescue!



Michigan State University  
CSE 231, Fall 2009

Classes

## Why a class

- We make classes because we need more complicated, user-defined data types to construct instances we can use.
- Each class has potentially two aspects:
  - the data (types, number, names) that each instance might contain
  - the messages that each instance can respond to.



Michigan State University  
CSE 231, Fall 2009

Classes

## A first class

```
class myClass(Object):
    def __init__(self,param1=10):
        self.theValue = param1
    def printIt(self):
        print self.theValue
```



Michigan State University  
CSE 231, Fall 2009

Classes

## Details

keyword: defines a class

class name follows name rules

default class we are a subclass of in the {}

```
class myClass(object):
```

class methods indented under the class

```
    def __init__(self, param1=10):
        self.theValue = param1
    def printIt(self):
        print self.theValue
```

Michigan State University  
CSE 231, Fall 2009

Classes

## “dot” reference

- we can refer to the attributes of an object by doing a “dot” reference, of the form: `object.attribute`
- the attribute can be a variable or a function
- it is part of the object, either directly or by that object being part of a class

Michigan State University  
CSE 231, Fall 2009

Classes

## examples

- print `myInst.myVal`
  - print a variable associated with the object `myInst`
- `myInst.myMethod()`
  - call a method associated with the object `myInst`
- variable versus method, you can tell by the parenthesis at the end of the reference

Michigan State University  
CSE 231, Fall 2009

Classes

## Methods

## Remember our first class

```
class myClass(object):
    def __init__(self, param1=10):
        self.theValue = param1
    def printIt(self):
        print self.theValue
```

Michigan State University  
CSE 231, Fall 2009

Classes

## method versus function

discussed before, a method and a function are closely related. They are both “small programs” that have parameters, perform some operation and (potentially) return a value

main difference is that methods are functions tied to a particular object

Michigan State University  
CSE 231, Fall 2009

Classes

## difference in calling

functions are called, methods are called in the context of an object:

- function:

```
dosomething(param1)
```

- method:

```
anObject.doSomething(param1)
```

This means that the object that the method is called on is always implicitly a parameter!



Michigan State University  
CSE 231, Fall 2009

Classes

## difference in definition

- methods are defined *inside* the suite of a class
- methods always bind the first parameter in the definition to the object that called it
- This parameter can be named anything, but traditionally it is named **self**

```
class myClass(object):
    def myMethod(self, otherparam):
        doSomething
```



Michigan State University  
CSE 231, Fall 2009

Classes

## more on self

- `self` is an important variable. In any method it is bound to the object that called the method
- through `self` we can access the internal structure of the instance



Michigan State University  
CSE 231, Fall 2009

Classes

## Back to the example

```
class myClass(Object):
    def __init__(self, param1=10):
        self.theValue = param1
    def printIt(self):
        print(self.theValue)
```



Michigan State University  
CSE 231, Fall 2009

Classes

## Binding self

```
newInstance = myClass()
```

```
newInstance.printIt()
```

```
def myClass (object):
```

```
...
```

```
def printIt(self):
```

```
...
```



Michigan State University  
CSE 231, Fall 2009

Classes

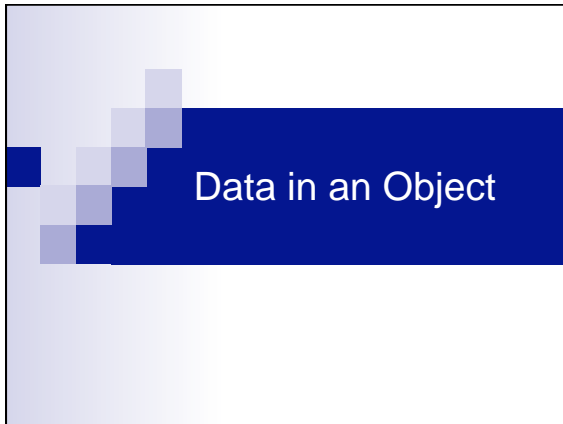
## self is bound for us

- when a dot method call is made, the object that called the method is **automatically** assigned to `self`
- we can use `self` to remember, and therefore refer, to the calling object
- to reference any part of the calling object, we must always precede it with `self`.
- The method can be written generically, dealing with all calling objects through `self`



Michigan State University  
CSE 231, Fall 2009

Classes



### How to make an object-local value

- once an object is made, the data is made the same way as in any other Python situation, by assignment
- Any object can thus be augmented by adding a variable

```
myInst.newVal = 123
```



Michigan State University  
CSE 231, Fall 2009

Classes

### Constructor

- there are some special methods that have certain pre-defined roles for all classes
- One of the first we will learn is a **constructor**.
- Constructor is called when an instance is made, and provides the class designer the opportunity to set up the instance with variables, by assignment



Michigan State University  
CSE 231, Fall 2009

Classes

### special python keywords

- again, python has special uses for keywords that begin and end with `__`
- so far we have seen the `__doc__` attributed of a function with a doc string
- In classes, we will see more of these special values.



Michigan State University  
CSE 231, Fall 2009

Classes

### the `__init__` method

- one of the special method names in a class is the constructor name, `__init__`
- by assigning values in the constructor, every instance will start out with the same variables
- you can also pass arguments to a constructor through its init method



Michigan State University  
CSE 231, Fall 2009

Classes

### calling a constructor

- a constructor is called by using the name of the class as a function call (by adding `()` after the class name)

```
myInst = myClass()
```

- creates a new instance using `myClass`



Michigan State University  
CSE 231, Fall 2009

Classes

## Remember our class

```
class myClass(object):
    def __init__(self,param1=10):
        self.theValue = param1
    def printIt(self):
        print self.theValue
```



Michigan State University  
CSE 231, Fall 2009

Classes

## Using our class

```
myInst = myClass() # default constructor
print myInst # prints <__main__.myClass object at 0x13870f0>
print myInst.theValue # prints 10
yourInst = myClass(127) # constructor with argument
print yourInst.theValue # prints 127
```



Michigan State University  
CSE 231, Fall 2009

Classes

## default constructor

- if you don't provide a constructor, then a default constructor is provided by the system
- the default constructor does "system stuff" to create the instance, nothing more
- you cannot pass arguments to the default constructor.



Michigan State University  
CSE 231, Fall 2009

Classes

## simple Date class

## Date class methods, 1 function

- `__init__` method which assigns values to variables in each instance that is made
- `printDate` method which nicely formats the date for printing
- `strUpdate` method which takes in a string of the form '01/15/1957' and assigns it to the date instance
- `validDate` function that "crudely" checks for valid dates



Michigan State University  
CSE 231, Fall 2009

Classes

## `__init__`

- constructor method
- takes 4 arguments: `self` (required) and 3 named parameters (`day,month,year`)
- remember, `self` is bound to the object that called the method
- `self.day = day` means that the `self` object creates a `day` variable (by assignment) and refers to the parameter value `day`
- now every instance gets a `day` variable (by the assignment done in the constructor)



Michigan State University  
CSE 231, Fall 2009

Classes

## the list of months

- note that the `__init__` method also creates in every instance a `monthList` variable
- This contains an enumeration of the names of all the months
- We can convert the month integer to a string by “properly” indexing into that list



Michigan State University  
CSE 231, Fall 2009

Classes

## printDate

- only takes a self parameter
- uses string formatting to print out the date in a nice formats
- indexes into the `monthList` variable since the names and number are “off by one”



Michigan State University  
CSE 231, Fall 2009

Classes

## strUpdate

- takes a string parameter of the form `'03/15/1492'`
- `month,day,year=[int(n) for n in dateStr.split('/')]`
  - split pulls out the three parts of the string, separated by a `'/'`
  - the list comprehension collects a list of the `int()` conversion of each of those strings
  - do multiple assignment to get the values



Michigan State University  
CSE 231, Fall 2009

Classes

## validDate

- just a function, no self in the parameters, not called by an object
- just does a crude check to make sure that the dates are “roughly” correct
- misses many details:
  - doesn't distinguish months
  - no leap years
  - can't distinguish AD and BC



Michigan State University  
CSE 231, Fall 2009

Classes