Classes and Objects (part I)

• What is a class?

• What is an object?

• Why use one?

• How to define and use an object
A class is a defined data type

- Built-in classes in Python include string and dictionary
- A class defines the kinds of data and functions that are available
An object is an instance (example) of a class

• For example:
  – string is a class
  – mystring = "AGGCGT" creates an object of class string

• You can only have one class named “string”

• You can have many objects which all belong to class string:
  – mystring = "AGGCGT"
  – yourstring = "Fred"

• The string class provides many useful functions which all string objects can use
  – mystring.upper(), yourstring.split(), etc.
Why use a class?

- Keep related data together
- Keep functions connected to the data they work on

Example:
- A date class could keep the day and month together
- It could offer functions such as "Add a number to a date"

- This could be done without classes, but classes are convenient and help organize the information

- A date class could help avoid the error where you add 15 to February 23 and get February 38
Defining a new class

- As an example, we will build up a simple date class
- A date consists of a day and month
- We will also provide a way to add a number to a date and get a correct answer
- A real date class would need a few more functions
- Years would be helpful!
- More error checking would be important too
Defining a new class

class date:
    def __init__(self, day, month):
        self.myday = day
        self.mymonth = month
    def printdate(self):
        print self.myday, self.mymonth

mydate = date(15, "January")
mydate.printdate()  
15 January
What does that do??

- The class statement creates a new class

- Inside the class, the special name “self” means the current object of that class

- Any variable named self.something is a member of the class

- Every object of the class will have a variable of that name

- This class has variables self.myday and self.mymonth
More features of our class

• All functions in a class start with “self” as an argument

• printdate(self) is a straightforward function

• It prints the object’s day and month

• __init__ is a special function that is run whenever an object of this class is created

• We use it to give the new object its values

• Almost all classes will want an init function
A fancier date class

class date:
    def __init__(self, day, month):
        self.myday = day
        self.mymonth = month
    def printUS(self):
        print self.mymonth, self.myday
    def printUK(self):
        print self.myday, self.mymonth

mydate = date(15,"January")
mydate.printUK()
15 January
mydate.printUS()
January 15
Adding a number to a date

- We would like a function on our date class that allows us to add a number to a date
- This is fairly tricky; we’ll build it in stages
- Rules:
  - Try adding the number to the day
  - If this goes past the end of a month, advance to the next month
  - Ignore the leap year problem
Practice problem 1

• Create and fill up a dictionary:
  – Key is name of month
  – Entry is number of days in month
Practice problem 2

• Write a function nextmonth()

• Argument: name of a month

• Return value: name of the next month
  – If it receives “July” it should return “August”
  – If it receives “December” it should return “January”

• You can do this with a big if statement, but there are easier ways

• (Hint: make a list of months with an extra “January” at the end)
Practice problem 3

- Copy the class definition into your program file
- Add a new class function add(self, numdays)
- This function accepts only positive number arguments
- It should use the dictionary to find the number of days in a month, and the nextmonth function to find the next month
Use your new date class

- Create an object of your date class, containing a date:

  - \texttt{birthday = date(6, "July")}

- Try adding various numbers to it:

  \begin{verbatim}
  birthday.printUS()
  July 6
  birthday.add(8)
  birthday.printUS()
  July 14
  birthday.add(30)
  birthday.printUS()
  August 13
  \end{verbatim}
daysinmonth = {
    "January":31,
    "February":28,
    "March":31,
    "April":30,
    "May":31,
    "June":30,
    "July":31,
    "August":31,
    "September":30,
    "October":31,
    "November":30,
    "December":31}
def nextmonth(thismonth):
    monthlist = ["January","February","March",
                 "April","May","June",
                 "July","August","September",
                 "October","November","December",
                 "January"]
    for index in range(0,len(monthlist)) :
        if (monthlist[index] == thismonth) :
            return monthlist[index + 1]
    print "Illegal month", thismonth
```python
class Date:
    def __init__(self, day, month):
        self.myday = day
        self.mymonth = month
    def printUS(self):
        print self.mymonth, self.myday
    def printUK(self):
        print self.myday, self.mymonth
    def add(self, numdays):
        self.myday = self.myday + numdays
        while (self.myday > daysinmonth[self.mymonth]):
            self.myday = self.myday - daysinmonth[self.mymonth]
            self.mymonth = nextmonth(self.mymonth)
```