

Classes and Objects (part II)

- Comments on homework
- A bit more on the practice problem from Thursday
- Some advanced techniques with classes

Comments on homework

- Test your program
 - Cases that should work—do they work?
 - Cases that should NOT work—do they fail gracefully?
- Do not meddle with the index of a `for` loop during the loop
- If you want to handle the incrementing yourself, use `while`

```
for index in range(0,20) :  
    newanswer = oldanswer * 2  
    index = index + 5 # this line does nothing useful!
```

Comments on homework

- What does `mylist[0:9]` mean?
- It means positions 0, 1, 2, 3, 4, 5, 6, 7, 8
- This is a total of 9 positions. It does not include position 9
- This is an example of a *half-open interval*
- It includes its left end but not its right end
- All slicing in Python uses half-open intervals

Comments on homework

- Indentation matters in Python
- Word processors or email programs can damage your indentation
- This makes it hard to grade!
- From now on, please email me a copy of your Python program, labeled with your name and the homework number
- If you don't, and the printed-out indentation is wrong, I will mark it down

Comments on homework

- It's important to read the assignment carefully
- Programs that do something slightly different are a bad idea in a lab environment
- Example:
 - In a PHYLIP file, the first 10 characters are the species name
 - I happened to show an example where the name ended with a space and had no spaces inside
 - Many students wrote code that relies on this, using `string.split()`
 - Legal PHYLIP files can break that code:
 - `Homosapie ATGCGGGCGGTCAATC` – OK
 - `HomosapienATGCGGGCGGTCAATC` – FAIL
 - `Homo sapieATGCGGGCGGTCAATC` – FAIL
- If you have a format specification, use it: don't rely on looking at the sample file

- (If the two don't match, talk to the person who asked you to write this program: they may be confused, and the sooner you find out, the better.)

Practice problem 1 solution

```
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}
```

Practice problem 2 solution

```
# It could also be done with 12 if statements but
# in general, shorter programs are easier to debug
```

```
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
```

Practice problem 2 alternate solution

```
def nextmonth(thismonth):
    nextmonthdict = {"January": "February",
                     "February": "March",
                     "March": "April",
                     "April": "May",
                     "May": "June",
                     "June": "July",
                     "July": "August",
                     "August": "September",
                     "September": "October",
                     "October": "November",
                     "November": "December",
                     "December": "January"}
    if thismonth in nextmonthdict :
        return nextmonthdict[thismonth]
    else :
        print "Illegal month", thismonth
```

Practice problem 3 solution

```
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)
```

date.add() changes its argument

- If you say `mybirthday.add(8)` you change my birthday
- It might be better to return a new date:

```
def addnew(self, numdays) :
    newmonth = self.mymonth
    newday = self.myday + numdays
    while (newday > daysinmonth[newmonth] :
        newday = newday - daysinmonth[newmonth]
        newmonth = nextmonth(newmonth)
    return date(newday,newmonth)
```

Using date.addnew()

```
>>> mybirthday = date(6,"July")
>>> mybirthday.printUS()
July 6
>>> party = mybirthday.addnew(4)
>>> mybirthday.printUS()
July 6
>>> party.printUS()
July 10
```

Advanced topic: Allowing the plus sign

We might want to write `party = mybirthday + 4`

```
def __add__(self, numdays) :
    newmonth = self.mymonth
    newday = self.myday + numdays
    while (newday > daysinmonth[newmonth] :
        newday = newday - daysinmonth[newmonth]
        newmonth = nextmonth(newmonth)
    return date(newday,newmonth)
```

usage example

```
>>> mybirthday = date(6,"July")
```

```
>>> party = mybirthday + 4
```

```
>>> party.printUS()
```

```
July 10
```

Operator overloading

- This shows the power of classes in Python
- We can make a new class, like date, behave like the built-in ones
- Operator overloads involve names with underscores

Common operator overloading methods

```
__init__      # object creation
__add__       # addition (+)
__mul__       # multiplication (*)
__sub__       # subtraction (-)
__lt__        # less than (<)
__str__       # printing
__call__      # function calls
```

Pros and Cons

- Good aspects of operator overloading
 - Make the date class easier to use
 - Can use your own classes just as you use built-in ones
 - It's very cool
- Bad aspects:
 - If you overload the + sign to do subtraction, you will make your life miserable
 - Must be sure that the resulting functions don't contain boobytraps
 - Cool code can distract you from getting the job done
- Bottom line: this is an advanced technique which you may or may not need
- One exception: almost all classes will need init functions

Practice problem 1

- Write a program which accepts four numbers from the command line:
- $P(AB)$, $P(Ab)$, $P(aB)$, $P(ab)$ in that order
- Check to see if they are legal:
 - Each one must be between 0.0 and 1.0
 - The sum of all four should be 1.0
 - Print "OK" or an error message

Practice problem 2

- Building on the last program:
- Calculate the allele frequencies of A, a, B and b
- Check to make sure that each one is legal (between 0 and 1)
- Make sure that $P(A) + P(a)$ is 1.0 and $P(B) + P(b)$ is 1.0

Practice problem 3

- You may have noticed incorrect results from problems 1 and 2
- The result of adding $1/3$ and $2/3$ in floating-point arithmetic is not guaranteed to be 1.0
- It may be 1.000000000001 or 0.999999999999
- Revise your solution to problem 2 by adding a function `equal_one` which tests if a number is “close enough” to 1
- We’ll define “close enough” as within 0.001
- Use this function each time you need such a test

Practice problem 1 – solution

```
import sys
if len(sys.argv) != 5 :
    print "this function takes 4 arguments"
    sys.exit()

names = {0:"P(AB)",1:"P(Ab)",2:"P(aB)",3:"P(ab)"}
msg = "OK"
sum = 0.0
for index in range(1,5) :
    if 0.0 <= float(sys.argv[index]) <= 1.0 :
        sum += float(sys.argv[index])
        continue
    msg = "argument " + str(index) + ", " + sys.argv[index] + ", "
    msg += "representing " + names[index-1]
    msg += " was not between 0.0 and 1.0, inclusive"
    break
```

```
if sum != 1.0 :  
    print "the arguments",sys.argv[1:], "should sum to 1.0"  
    sys.exit()
```

```
print msg
```

Practice problem 2 – solution

```
import sys
if len(sys.argv) != 5 :
    print "this function takes 4 arguments"
    sys.exit()

names = {0:"P(AB)",1:"P(Ab)",2:"P(aB)",3:"P(ab)"}
msg = "OK"
sum = 0.0
for index in range(1,5) :
    if 0.0 <= float(sys.argv[index]) <= 1.0 :
        sum += float(sys.argv[index])
        continue
    msg = "argument " + str(index) + ", " + sys.argv[index] + ", "
    msg += "representing " + names[index-1]
    msg += " was not between 0.0 and 1.0, inclusive"
    break
```

```

if sum != 1.0 :
    print "the arguments",sys.argv[1:], "sum to",sum,"but should sum
    sys.exit()

#relies on the exact input order and number
freqs = {"A":(float(sys.argv[1]) + float(sys.argv[2])),
        "a":(float(sys.argv[3]) + float(sys.argv[4])),
        "B":(float(sys.argv[1]) + float(sys.argv[3])),
        "b":(float(sys.argv[2]) + float(sys.argv[4]))}
freqkeys = freqs.keys()
for allele in freqkeys :
    if 0.0 <= freqs[allele] <= 1.0 :
        continue
    msg = "allele " + allele + " seems to have a frequency of "
    msg += str(freqs[allele]) + "."
    break

if freqs["A"] + freqs["a"] != 1.0 :
    print "the total for the A-a locus is",str(freqs["A"]+freqs["a"])
    print "when it should 1.0"

```

```
sys.exit()

if freqs["B"] + freqs["b"] != 1.0 :
    print "the total for the B-b locus is",str(freqs["B"]+freqs["b"])
    print "when it should 1.0"
    sys.exit()

print msg
```

Practice problem 3 – solution

```
def equal_one(value) :
    if value < 0.999 or value > 1.001 :
        return False
    return True

import sys
if len(sys.argv) != 5 :
    print "this function takes 4 arguments"
    sys.exit()

names = {0:"P(AB)",1:"P(Ab)",2:"P(aB)",3:"P(ab)"}
msg = "OK"
sum = 0.0
for index in range(1,5) :
    if 0.0 <= float(sys.argv[index]) <= 1.0 :
        sum += float(sys.argv[index])
        continue
```

```
msg = "argument " + str(index) + ", " + sys.argv[index] + ", "  
msg += "representing " + names[index-1]  
msg += " was not between 0.0 and 1.0, inclusive"  
break
```

```
if not equal_one(sum) :  
    print "the arguments",sys.argv[1:], "sum to",sum,  
    print "but should sum to 1.0"  
    sys.exit()
```

```
#relies on the exact input order and number  
freqs = {"A":(float(sys.argv[1]) + float(sys.argv[2])),  
         "a":(float(sys.argv[3]) + float(sys.argv[4])),  
         "B":(float(sys.argv[1]) + float(sys.argv[3])),  
         "b":(float(sys.argv[2]) + float(sys.argv[4]))}  
freqkeys = freqs.keys()  
for allele in freqkeys :  
    if 0.0 <= freqs[allele] <= 1.0 :  
        continue  
    msg = "allele " + allele + " seems to have a frequency of "
```

```
msg += str(freqs[allele]) + "."
break

if not equal_one(freqs["A"] + freqs["a"]) :
    print "the total for the A-a locus is",
    print str(freqs["A"]+freqs["a"]),
    print "when it should 1.0"
    sys.exit()

if not equal_one(freqs["B"] + freqs["b"]) :
    print "the total for the B-b locus is",
    print str(freqs["B"]+freqs["b"]),
    print "when it should 1.0"
    sys.exit()

print msg
```