1 Administrative Topics

- I believe our year-getting code from Mon is perfectly good.
- Bruce has updated the saveCanvas code.Oops. But it still doesn’t work. I’ll get back to you on that one.

2 More about Classes

Recall the Student class.

It has data fields

- name
- year

and methods

- getName (an accessor method)
- getYear (an accessor method)
- setYear (a mutator method)
• `__init__`

• `getGradeTitle` or `getSeniority` (depending on which section you are in)

Note that there is an entry for each data field and each method in a Student object’s symbol table. One implication is that it means you cannot make methods and data fields with the same name.

Also, note that the methods are added to the object’s symbol table when Python first creates the object. The data fields are added when `__init__` is executed. As a general rule, if any data field that is ever going to be in an object should be added in the `__init__` method. This makes for easy reading later.

Today, we will continue developing the Student class, making it more sophisticated. My goal is for you to be prepared for lab today when Bruce will guide you through the process of rewriting the L-system and interpreter modules as classes.

Let’s add the possibility for Student objects to include grades. Let’s be super simple about this. Grades are simply stored in a list. They can be strings (e.g. “P” or “F” or “I”) or numbers (between 0 and 4).

A student record is initialized without any grades (a new student doesn’t have any).

Grades are added one by one. For this, we need a mutator method named `addGrade`.

We should be able to get the list of grades, so we need an accessor method `getGrades`.

And we should be able to compute a GPA, but that computation should take into account only number grades. We must write two methods for this – one that returns a list of just the number grades. Another which computes the mean of those grades.

Today, we will write these four new methods and add support for the new field.

First, we add the line `self.grades = []` to the `init` function:

```python
# init with name and year
def __init__(self, name, year):
```

2
This ensures the data field is there, and is consistent with our statement that no student enters Colby with grades.

Next, we write addGrade:

```python
# add a grade (grade must be a string or number)
def addGrade( self, grade ):
    self.grades.append( grade )
```

Next, we write getGrades. Note that it would be straight-forward to write code that simply returns `self.grades`. But we don’t want to do this. We want to “protect” our data. First, remember that we want code within class methods to manipulate data in the object’s symbol table and NO ONE ELSE. Second, remember what type `self.grades` is. It is a list. Lists are mutable. If we simply return a reference the grades field, then we are giving the caller (which could be code outside the class) the ability to edit the list. Ahhh. We don’t want that. Instead, we want to return a clone of the list. To do that, we simply make a new list, and copy each element in. Since the elements are numbers or strings, we don’t need to do that in any special way. So, the code for getGrades is:

```python
# return a clone of the grades
def getGrades( self ):
    #return self.grades # straight-forward, but not safe!
    grades = []
    for grade in self.grades:
        grades.append( grade )
    return grades
```

The next two methods to write are getNumberGrades and getGPA. Since we need to compute the GPA from the number grades only, we will probably need to call getNumberGrades from getGPA, so let’s write getNumberGrades first. It is similar to `lstinline getGrades`, but requires an if-statement to prevent us from coping text grades into the return list:

```python
# return a list of numbers
def getNumberGrades( self ):
    numgrades = []
```
for grade in self.grades:
    if type(grade) == int or type(grade) == float:
        numgrades.append(grade)
return numgrades

Finally, we are ready to write getGPA. The most interesting aspect of this function is the syntax for calling one method from another. We use the dot notation with the self variable as the object.

```python
# return the average of the number grades
def getGPA( self ):
    ng = self.getNumberGrades()
    if len(ng) == 0:
        return 0.0
    avg = 0.0
    for g in ng:
        avg += g  # equiv to avg = avg + g
    avg /= len(ng)  # equiv to avg = avg / len(ng)
return avg
```

The complete code has been uploaded to the course web page.