Homework 2

- Name of activity (Module 02 Exercise: Python)
- Dmitry Mikhaylov
- Your UVA computing ID: agp7dp

```
class Student:
    # fields: name, id, grades(a list)

    #Local variable
    grades = [] # initially empty

    def __init__(self, name, id):  # constructor
        self.name = name
        self.id = id

    def addGrade(self, grade): # add the grade to the list of grades
        self.grades.append(grade)

    def showGrades(self): # displaying the grades
        grds = '' # empty string
        for grade in self.grades: # Loop through grades list
            grds += str(grade) + ' '  # assign each grade to the string grds
        return grds


student1 = Student('Jones', '123')
print(str(student1.name) + ', ' + str(student1.id)) # Output: Jones, 123
student1.addGrade(88)
student1.addGrade(72)
student1.addGrade(100)
print("Grades: " + student1.showGrades()) # showing grades for student1
# print(student1) # Will NOT work, since we do not have a "to-string" (__str__) method
# Output of the above line will be a memory address like:
#      <__main__.Student object at 0x00000220B8611BE0>
```

```
Jones, 123
Grades: 88 72 100
```

```
# ** TO THINK ABOUT: **
# This is fine, however, what happens if you create a second Student object??
# Local ("global") variable grades (a list - which is a mutable object) will
# accumulate grades from ALL students... this behavior is not what we want.
# Uncomment lines 46-51 below and see what happens.  How would you fix this??
# For now, the above file is fine for the above scenario.

# ================================================================================
s2 = Student('Clayton', '115')
print(str(s2.name) + ', ' + str(s2.id))
s2.addGrade(85)
s2.addGrade(95)
s2.addGrade(99)
print("Grades: " + s2.showGrades())  # !!!
# ================================================================================
```

```
    Clayton, 115
    Grades: 88 72 100 85 95 99
```

Discussion:

The problem is that every new instance of `Student` shares access to the list of `grades` and thus no instance really has a unique record of grades. This can be solved by the following:

1. Derive new class `GradedStudent` that takes in `Student` as a base class
2. Add `self.grades = list(grades)` into the new constructor
3. When averaging the grades make sure to account for possibility of no grades, such as in case of a new studnet.

```
class GradedStudent(Student):


  def __init__(self, name, id, grades): # modified constructor
    Student.__init__(self, name, id)
    self.grades = list(grades)

  def average(self): # average the list of grades
    try:
      avg = sum(self.grades) / len(self.grades)
    except (IndexError, ZeroDivisionError): # if no grades yet, return 0
      avg = 0
    return avg
```

```
    return avg

  def __str__(self): # string representation of the class
    result =  "Graded student {} with ID {} has grades {} averaging {}.".format(
        self.name, str(self.id), str(self.grades), str(self.average())
        )
    return result


gs = GradedStudent("Bob", 123, [33, 33, 22, 222])


print(gs)
```

```
Graded student Bob with ID 123 has grades [33, 33, 22, 222] averaging 77.5.
```

```
another_gs = GradedStudent("Alice", 123, [777777, 888888, 99999])


print(another_gs)
```

```
Graded student Alice with ID 123 has grades [777777, 888888, 99999] averaging 588888.0.
```

```
new_gs = GradedStudent("Cat", 2242, []) # new studnet without any grades yet


print(new_gs)
```

```
Graded student Cat with ID 2242 has grades [] averaging 0.
```

```
new_gs.addGrade(44)


new_gs.showGrades()
```

```
'44 '
```

```
print(new_gs)
```

```
Graded student Cat with ID 2242 has grades [44] averaging 44.0.
```

✓ 0s completed at 3:42 PM