

# Class Design

## Student Modelling

In this tutorial, we will write a simple program which keeps track of university students and the courses they take. The code below provides a starting point.

```
class Student(object) :
    """Simple representation of a university student."""
    def __init__(self, name) :
        """Create a student with a name.

        Parameters:
            name (str): The student's name.
        """
        self._name = name

    def get_name(self) :
        """(str) Returns the name of the student."""
        return self._name
```

Modify this class so that it also stores the student number, and degree program (as a string). Add methods `get_student_num` and `get_degree` which retrieve this information, and a method `set_degree` which changes the student's degree. For example, the resulting class might behave like this:

```
>>> s = Student('Michael Palin', 43215678, 'BInfTech')
>>> s.get_name()
'Michael Palin'
>>> s.get_student_num()
43215678
>>> s.get_degree()
'BInfTech'
>>> s.set_degree('BE')
>>> s.get_degree()
'BE'
```

Why might it be inappropriate to add a `set_student_num` method which modifies the student number; similar to `set_degree`?

Next, add the following methods. For our purposes, we will assume that the student's name is composed of two names (first name and last name) separated by a space.

- `get_first_name`, which returns the student's first name (e.g. 'Michael')
- `get_last_name`, which returns the student's surname (e.g. 'Palin')
- `get_email`, which returns the student's email address derived from their name (e.g. 'michael.palin@uq.net.au'). For this exercise, it is safe to assume the student's email is always in the format `firstname.lastname@uq.net.au`.
- `__str__`, which returns a string with the student's name, email, student number, and degree, in a format such as 'Michael Palin (michael.palin@uq.net.au, 43215678, BE)'
- `__repr__`, which returns a string which looks like the Python code to create the object; for example, `"Student('Michael Palin', 43215678, 'BE')"`

## Validity Checking

It is important that each student in the University has a different student number. To ensure this, we can take a list of `Student` objects, and check that no two students have the same student number.

Write a function `check_students` which takes a list of `Student` objects, and returns `True` if they all have different student numbers, and `False` if there are student numbers that have been repeated.

Make sure you write `check_students` as a function, not a method of the `Student` class! For example:

```
>>> students1 = [Student('Alice A', 1, 'BE'),
                  Student('Bob B', 2, 'BA'),
                  Student('Carol C', 4, 'BA')]
>>> check_students(students1)
True
>>> students2 = [Student('Alice A', 1, 'BE'),
                  Student('Bob B', 2, 'BA'),
                  Student('Carol C', 4, 'BA'),
                  Student('Dan D', 2, 'BInfTech')]
>>> check_students(students2)
False
```

In the first example, all students have different student numbers (1, 2, 4). In the second example, Bob and Dan have the same student number (2).

## Adding Courses

Write a class which represents a university course. The class should have getter methods for the course code (e.g. 'CSSE1001') and the course name (e.g. 'Introduction to Software Engineering').

Add functionality to the `Student` class so that each instance is capable of recording the courses for which a student has obtained a grade. You should write the following methods in the `Student` class:

- `add_grade(course, grade)`, which sets the student's grade for a given course. If the student already has a grade for that course, then replace the old grade with the new one.
- `gpa()`, which returns the student's GPA (the average of all their grades). (You may assume that each course carries the same weight.) For example:

```
>>> s = Student('Michael Palin', 43215678, 'BE')
>>> csse1001 = Course('CSSE1001', 'Intro to Software Engineering')
>>> deco1800 = Course('DECO1800', 'Design Computing Studio I')
>>> s.add_grade(csse1001, 4)
>>> s.gpa()
4.0
>>> s.add_grade(deco1800, 5)
>>> s.gpa()
4.5
>>> s.add_grade(csse1001, 6) # Overwrite the old grade
>>> s.gpa()
5.5
```

### **Challenge: Extending Further**

It is clear that our model of students and courses is still incomplete. Consider further extensions that can be made, and implement them. Possible improvements include:

- storing courses a student is currently taking within the 1 class,
- accessing a list of students enrolled in a course,
- storing the semester in which students took a course, or
- allowing for courses with different unit weightings in GPA calculation. As part of this process, you will need to consider which classes to alter, and which methods to add.