# Classes

**This third notebook is an optional extension describing how classes work in python. Classes are very useful in developing software, but you will likely not need to know how to define them if you are going to use python for analysing data.**

**Classes are effectively a nice way to package functionality and data together that has to be applied to multiple different objects of the same type. An instance of a class is called an object and can have attributes attached to it for maintaining its state. Class instances can also have methods for modifying its state.**

**Classes are defined as follows:**

```
class className:
    globalValue = "global"
    # methods that belong to the class
    def __init__(self, name):
        # this method is called whenever a new instance is created
        self._instanceName = name

    def classMethod(self):
        # this is a method that belonds to the class
        # Note how we have an argument self, which is a reference to the object it
self

    def classMethod2(self):
        # Another method
```

**We define a new object as:**

```
newObject = className("Example")
```

**You use classes to define objects that represent the concepts and things that your program will work with. For example, if your program managed exam results of students, then you may create one class that represents an Exam, and another that represents a Student.**

**Let us see how this can be implemented:**

In [ ]:

```
class Exam:
    def __init__(self, max_score=100):
        self._max_score = max_score
        self._actual_score = 0

    def percent(self):
        return 100.0 * self._actual_score / self._max_score

    def setResult(self, score):
        if score < 0:
            self._actual_score = 0
        elif score > self._max_score:
            self._actual_score = self._max_score
        else:
            self._actual_score = score

    def grade(self):
        if self._actual_score == 0:
            return "U"
        elif self.percent() > 70.0:
            return "A"
        elif self.percent() > 60.0:
            return "B"
        elif self.percent() > 50.0:
```

```
            return "C"
        else:
            return "F"

class Student:
    def __init__(self, name):
        self._exams = {}
        self._name = name

    def addExam(self, name, exam):
        self._exams[name] = exam

    def addResult(self, name, score):
        self._exams[name].setResult(score)

    def result(self, exam):
        return self._exams[exam].percent()

    def grade(self, exam):
        return self._exams[exam].grade()

    def grades(self):
        g = {}
        for exam in self._exams.keys():
            g[exam] = self.grade(exam)
        return g
```

**We can now create a `Student` object and add exams to it:**

In [ ]:

```
# You will need to add a name for the student below
s = Student("Ignat")  # create new object s of type Student
s.addExam("Maths", Exam(20))  # use Students' method
s.addExam("Chemistry", Exam(75))
```

**The student now has exams which he has attended. Now we have to give him grades:**

In [ ]:

```
s.addResult("Maths", 15)
s.addResult("Chemistry", 62)
s.grades()
```

**Programming with classes makes the code easier to read, as the code more closely represents the concepts that make up the program. For example, here we have a class that represents a full school of students.**

In [ ]:

```
class School:
    def __init__(self):
        self._students = {}
        self._exams = []

    def addStudent(self, name):
        self._students[name] = Student(name)

    def addExam(self, exam, max_score):
        self._exams.append(exam)

        for key in self._students.keys():
            self._students[key].addExam(exam, Exam(max_score))

    def addResult(self, name, exam, score):
        self._students[name].addResult(exam, score)

    def grades(self):
        grades = {}
        for name in self._students.keys():
```

```
            grades[name] = self._students[name].grades()
        return grades
```

**Now we can populate the school with students and their grades:**

In [ ]:

```
school = School()
school.addStudent("Andrew")
school.addStudent("James")
school.addStudent("Laura")
school.addExam("Maths", 20)
school.addExam("Physics", 50)
school.addExam("English", 30)
school.grades()
```

**We have a school of students but sadly all of them have grades U which stands for unassigned. How can we add grades? Functions and loops come to the rescue!**

**The grades which the examiners have returned are:**

In [ ]:

```
maths_results = {"Andrew" : 13, "James" : 17, "Laura" : 14}
physics_results = {"Andrew" : 34, "James" : 44, "Laura" : 27}
english_results = {"Andrew" : 26, "James" : 14, "Laura" : 29}
```

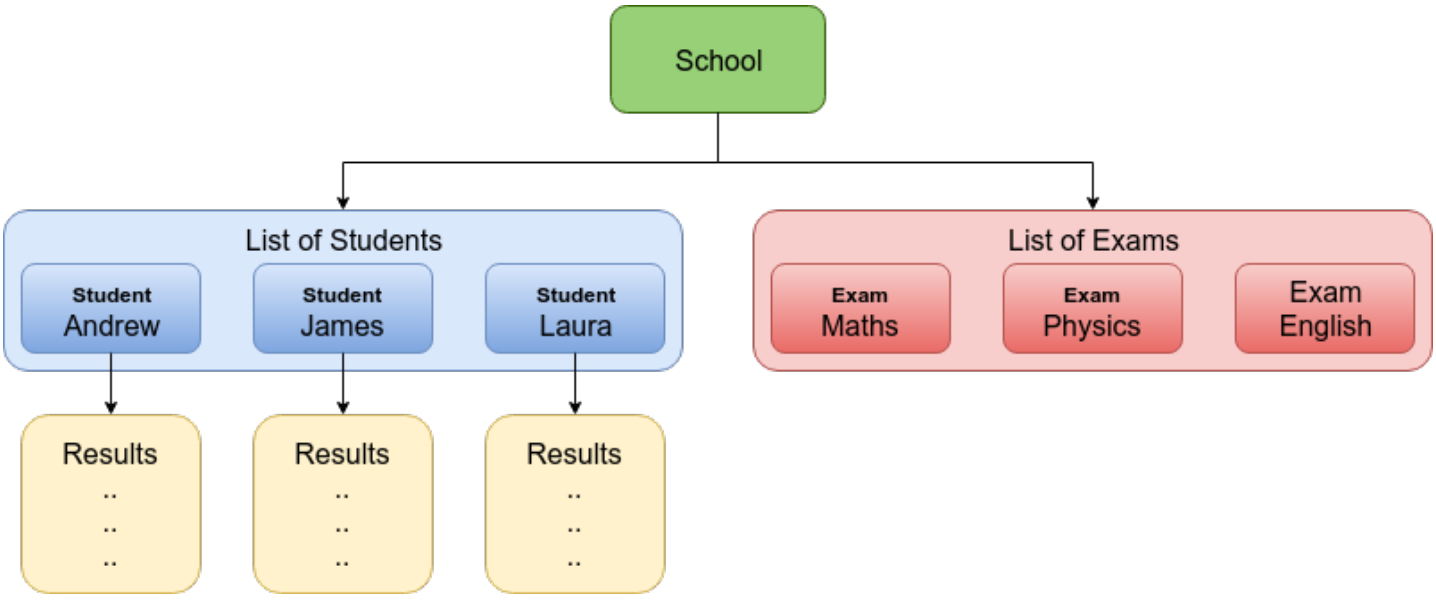
In [ ]:

```
def addResults(school, exam, results):
    for student in results.keys():
        school.addResult(student, exam, results[student])

addResults(school, "Maths", maths_results)
addResults(school, "Physics", physics_results)
addResults(school, "English", english_results)

school.grades()
```

**Let us take a step back and have a look the big picture now:**




**Hopefully this makes using the whole datastructure more intuitive and easy, which was the initial goal. Now we can easily change things and add new Students or Exams (and resulsts).**

**Taking a step back even further, everything in Python is an object which was defined in a class somewhere. That is why it is reffered to as a object-orientated programming language.**

**Strings, floats, integers, etc. Everything is a object, has its own values and methods.**

**For an exercise exploring classes, see the notebook python-intro-exercises.**

In [ ]:

```
```