# TP 5

**Exercice 1**
Écrire une classe `Resultat` qui possède deux attributs d'instance : l'intitulé du cours ainsi que la note obtenue. Écrire, le constructeur et les accesseurs et la méthode `__str__`.

**Exercice 2**
On souhaite gérer un ensemble d'étudiants. Chaque étudiant sera défini par une classe `Etudiant` et devra présenter les informations suivantes :

- un numéro d'étudiant,
- un nom,
- un prénom,
- une liste de `Resultats`.

Écrire le constructeur, celui-ci permettra l'instanciation d'un étudiant avec une liste de résultats vide. Écrire les accesseurs (on n'écrira pas le setter pour l'attribut qui contient la liste de résultats) et la méthode `__str__`. Écrire également Une méthode `add_resultat` permettant d'ajouter une note dans un cours à l'étudiant.

**Exercice 3**
Une classe `GroupeTD` permettra de gérer une liste d'étudiants figurant dans ce groupe. Ajouter un constructeur qui construit un groupe vide, puis ajouter une méthode qui permet l'ajout d'un étudiant.

**Exercice 4**
On souhaite mettre en place une classe capable de réaliser des statistiques sur une liste d'objets, comme par exemple, des Etudiants, des Résultats, ... Cette classe, qui sera nommée `Stats`, pourra ainsi calculer le maximum, le minimum et la moyenne d'une liste d'objets. On considère que les objets contenus dans la liste sont « statisticables »(cf. exercice suivant) et définissent la méthode `get_value()` qui permet de connaitre la valeur à considérer pour effectuer les différentes opérations. Écrire la classe `Stats` qui fournit les méthodes statiques suivantes :

- moyenne (l) : fournit la moyenne de la liste passée en argument,
- min (l) : fournit la valeur minimale de la liste passée en argument,
- max (l) : fournit la valeur maximale de la liste passée en argument.

Petit rappel : les méthodes statiques sont définies sur la classe et non sur les instances (objets) de celle-ci. Ces méthodes ont une connexion logique avec la classe, mais n'utilisent pas l'état de la classe ou de l'objet. Une méthode statique ne reçoit pas de premier argument implicitement. Voilà comment déclarer une méthode statique :

```
class C:
    @staticmethod
    def f(arg1, arg2, ...): ...
```

Une méthode statique peut être appelée sur la classe (`C.f()`).

**Exercice 5**
Toutes les classes qui peuvent faire l'objet de statistiques implémenteront l'interface `Statisticable` :

```
class Statisticable:
  def get_value(self):
    raise NotImplementedError()
```

Tout objet « statisticable » doit donc avoir une certaine valeur (et donc implémenter la méthode `get_value`) ; pour un `Etudiant`, on choisit de prendre la moyenne de ses notes comme valeur de l'Etudiant. Pour un groupe de TD, la moyenne des étudiants du groupe et pour un résultat la note obtenue. Ainsi pour un étudiant donné, on peut connaitre sa moyenne ainsi que sa

note minimale et sa note maximale. Pour un groupe de TD, il est possible de connatre la moyenne du groupe, la moyenne la plus basse et la moyenne la plus haute. Ajouter la méthode `get_value(self)` à la classe `Etudiant` et faites-la hériter de `Statisticable`. De même pour les classes `GroupeTD` et `Resultat`

**Exercice 6**

On souhaite pouvoir classer la liste d'étudiants suivant leur moyenne. Pour ce faire, il faut que les étudiants (ou plutôt leurs résultats) soient « comparables ». Pour cela, il faut implémenter la méthode `__lt__` (et idéalement `__eq__`) dans la classe `Etudiant`. Une fois cela réalisé, on pourra trier la liste d'étudiants à l'aide de la méthode list.sort.

Tester toutes vos classes, en créant un groupe de TD. Ajoutez-y des étudiants, ajoutez leurs des notes, affichez les étudiants par ordre de mérite...).