Table of Contents

실행시간 확인을 위한 함수

In [1]:
```
import time
from time import localtime, strftime

def checkTime(func):
    def new_func(*args, **kwargs):

        start = time.time()
        func(*args,**kwargs)
        end = time.time()

        print("Wn실행시간은:", end - start)

    return new_func
```

Student 클래스

In [2]:
```
class Student():
    def __init__(self, num, name, kor, eng, math):

        self._num = num
        self._name = name
        self._kor = kor
        self._eng = eng
        self._math = math

        self._total = self._kor + self._eng + self._math
        self._avg = self._total / 3
        self._order = 0

    @property
    def num(self):
            return self._num

    @property
    def name(self):
        return self._name

    @property
    def kor(self):
        return self._kor

    @property
    def eng(self):
        return self._eng

    @property
    def math(self):
        return self._math

    @property
    def total(self):
        return self._total

    @property
```

```
    def avg(self):
        return self._avg

    def get_order(self):
        return self._order

    def set_order(self, value):
        self._order = value

    order = property(get_order, set_order)
```

StudentGradeSystem

In [3]:

```
class StudentGradeSystem(object):
    def __init__(self, filename):
        self.filename = filename
        self.student_list = []

        self.total_avg = 0
        self.kor_avg = 0
        self.eng_avg = 0
        self.math_avg = 0

        self.kor_max = 0
        self.eng_max = 0
        self.math_max = 0
        self.kor_max_list = []
        self.eng_max_list = []
        self.math_max_list = []

    def register_student(self):
        f = open(self.filename, 'r', encoding='utf-8')
        lines = f.readlines()

        for line in lines:
            info = (line.strip()).split(",")
            num, name, kor, eng, math = [x.split()[1] for x in info]
            kor = int(kor)
            eng = int(eng)
            math = int(math)

            student = Student(num, name, kor, eng, math)
            self.student_list.append(student)

    def cal_order(self):
        order_list = sorted(self.student_list, key = lambda st:st.total, reverse = T
        order = 0
        temp = 0

        for student in order_list:
            if temp == student.total: # 동점 처리
                student.order = order
            else: # 그외
                temp = student.total
                order += 1
                student.order = order

        self.student_list = order_list


    def cal_total_avg(self):
        for student in self.student_list:
            self.total_avg += student.total
```

```
        self.total_avg /= len(self.student_list)

    def cal_kor_avg(self):
        for student in self.student_list:
            self.kor_avg += student.kor
        self.kor_avg /= len(self.student_list)

    def cal_eng_avg(self):
        for student in self.student_list:
            self.eng_avg += student.eng
        self.eng_avg /= len(self.student_list)

    def cal_math_avg(self):
        for student in self.student_list:
            self.math_avg += student.math
        self.math_avg /= len(self.student_list)

    def cal_kor_max(self):
        self.kor_max = max(self.student_list, key = lambda student:student.kor).kor

        for student in self.student_list:
            if student.kor == self.kor_max:
                self.kor_max_list.append(student.name)

    def cal_eng_max(self):
        self.eng_max = max(self.student_list, key = lambda student:student.eng).eng

        for student in self.student_list:
            if student.eng == self.eng_max:
                self.eng_max_list.append(student.name)

    def cal_math_max(self):
        self.math_max = max(self.student_list, key = lambda student:student.math).ma

        for student in self.student_list:
            if student.math == self.math_max:
                self.math_max_list.append(student.name)

    def process(self):
        self.cal_order()
        self.cal_total_avg()
        self.cal_kor_avg()
        self.cal_eng_avg()
        self.cal_math_avg()
        self.cal_kor_max()
        self.cal_eng_max()
        self.cal_math_max()

    def print_students(self):
        for st in self.student_list:
            print(f"번호: {st.num}, 이름: {st.name}, 국어: {st.kor}, 영어: {st.eng},
        print()

    def print_class_information(self):
        print("총점 반평균: {:.2f}".format(self.total_avg))
        print("국어 반평균: {:.2f}".format(self.kor_avg))
        print("영어 반평균: {:.2f}".format(self.eng_avg))
        print("수학 반평균: {:.2f}".format(self.math_avg))
        print()
        print("각 과목의 최고점과 명단")
        print(f"국어의 최고점 {int(self.kor_max)}, ", end="")
        for st in self.kor_max_list:
            print(st, end=" ")
        print()
```

```
        print(f"영어의 최고점 {int(self.eng_max)}, ", end="")
        for st in self.eng_max_list:
            print(st, end=" ")
        print()
        print(f"수학의 최고점 {int(self.math_max)}, ", end="")
        for st in self.math_max_list:
            print(st, end=" ")
        print()

    @checkTime
    def checktime(self):
        self.register_student()
        self.process()
        self.print_students()
        self.print_class_information()
```

실제 실행

실행 시간 체크를 위해 checktime() 메소드를 따로 만들어 RUN하였다.

In [4]:

```
grsys = StudentGradeSystem("SData.txt")
grsys.checktime()
```

```
번호: 2, 이름: 서강이, 국어: 90, 영어: 85, 수학: 95, 총점: 270, 평균: 90.00, 등수: 1
번호: 4, 이름: 서강사, 국어: 90, 영어: 92, 수학: 83, 총점: 265, 평균: 88.33, 등수: 2
번호: 5, 이름: 서강오, 국어: 85, 영어: 90, 수학: 90, 총점: 265, 평균: 88.33, 등수: 2
번호: 1, 이름: 서강일, 국어: 90, 영어: 80, 수학: 85, 총점: 255, 평균: 85.00, 등수: 3
번호: 3, 이름: 서강삼, 국어: 80, 영어: 80, 수학: 80, 총점: 240, 평균: 80.00, 등수: 4

총점 반평균: 259.00
국어 반평균: 87.00
영어 반평균: 85.40
수학 반평균: 86.60

각 과목의 최고점과 명단
국어의 최고점 90, 서강이 서강사 서강일
영어의 최고점 92, 서강사
수학의 최고점 95, 서강이

실행시간은: 0.001994609832763672
```

In [ ]: