mongoDB Project
Relational databases & Document-Oriented databases

Athens University of Economics and Business
Dpt. Of Management Science and Technology
Prof. Damianos Chatziantoniou

Lamprini Koutsokera (8130074) | lkoutsokera@gmail.com
Stratos Gounidellis (8130029) | stratos.gounidellis@gmail.com

BDSMasters

SQL Server vs. mongoDB

	Microsoft SQL Server	mongoDB
Description	Microsoft’s relational DBMS	One of the most popular document stores
Database model	Relational DBMS	Document store
Implementation language	C++	C ++
Data scheme	yes	schema-free
Triggers	yes	no
Replication methods	yes, depending the SQL-Server Edition	Master-slave replication
Partitioning methods	tables can be distributed across several files, sharding through federation	Sharding

References: [1]

From **theory** to **practice**

Tools

mongoDB®

python™

jupyter

JavaScript

ANACONDA®

NumPy

GitHub

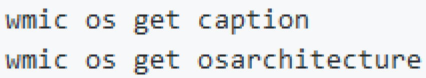
pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Required installations

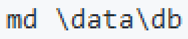
Mongodb installation

1. Determine which **MongoDB build** you need.



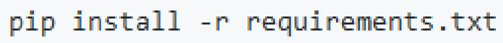
```
wmic os get caption
wmic os get osarchitecture
```

2. Download MongoDB for **Windows**
3. Install MongoDB **Community Edition**.
4. Set up the MongoDB **environment**.

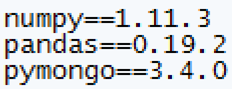


```
md \data\db
```

Required python packages installation



```
pip install -r requirements.txt
```

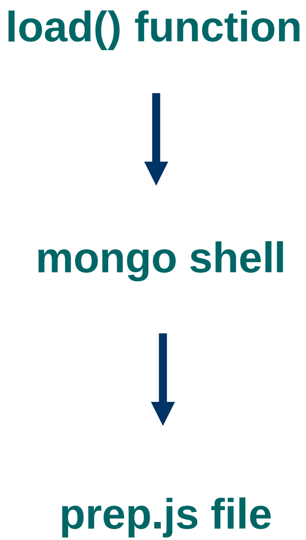


```
numpy==1.11.3
pandas==0.19.2
pymongo==3.4.0
```

References: [2]

From **software configuration** to **coding**

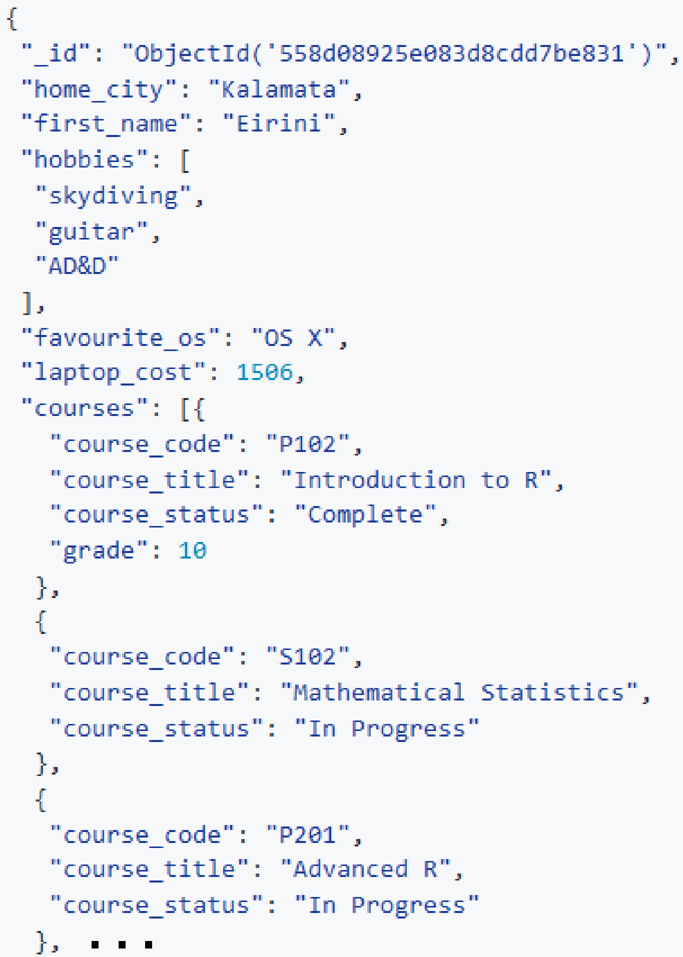
Part 1 - Queries and the Aggregation Pipeline [1]



load() function

mongo shell

prep.js file

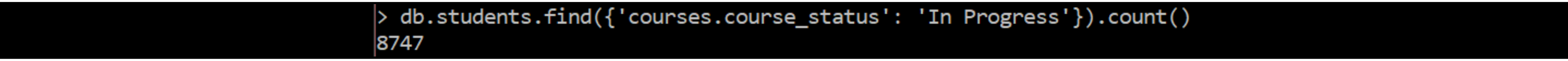


```
{
 "_id": "ObjectId('558d08925e083d8cdd7be831')",
 "home_city": "Kalamata",
 "first_name": "Eirini",
 "hobbies": [
  "skydiving",
  "guitar",
  "AD&D"
 ],
 "favourite_os": "OS X",
 "laptop_cost": 1506,
 "courses": [{
   "course_code": "P102",
   "course_title": "Introduction to R",
   "course_status": "Complete",
   "grade": 10
  },
  {
   "course_code": "S102",
   "course_title": "Mathematical Statistics",
   "course_status": "In Progress"
  },
  {
   "course_code": "P201",
   "course_title": "Advanced R",
   "course_status": "In Progress"
  }, ...
```

References: [3]

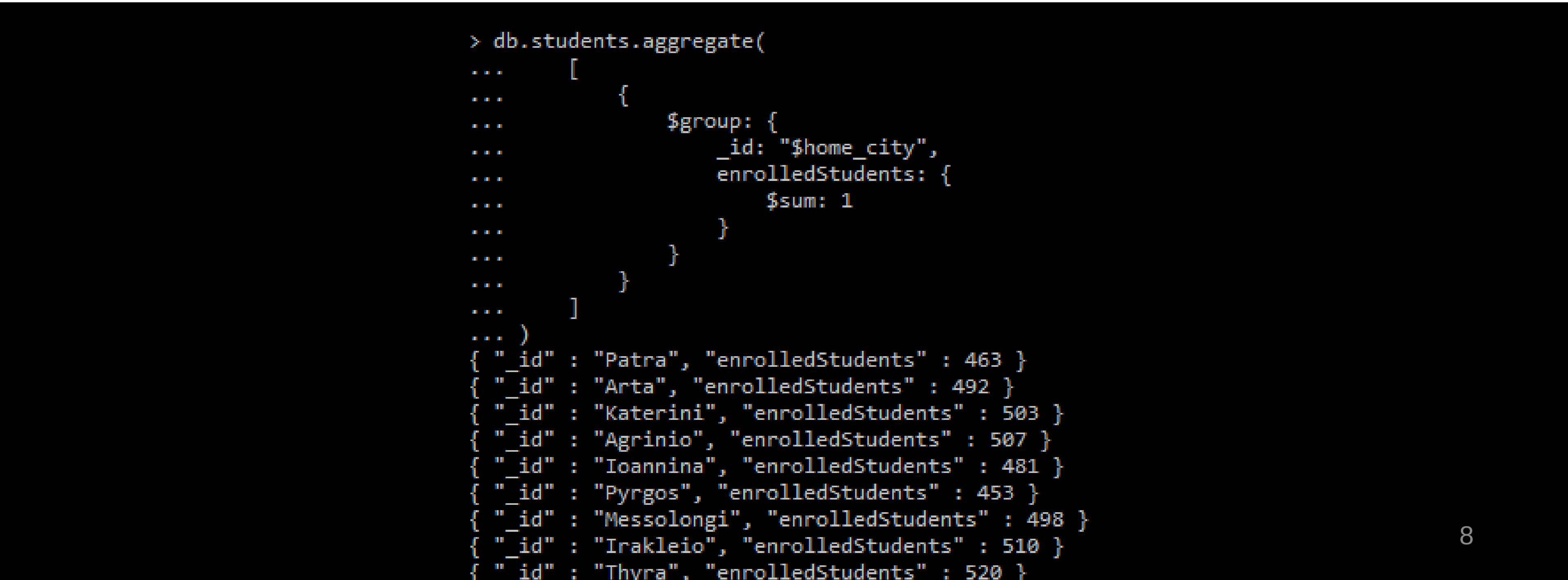
Part 1 - Queries and the Aggregation Pipeline [2]

Query 1 : How many students in your database are currently taking at least 1 class (i.e. have a class with a course_status of “In Progress”)?



```
> db.students.find({'courses.course_status': 'In Progress'}).count()
8747
```

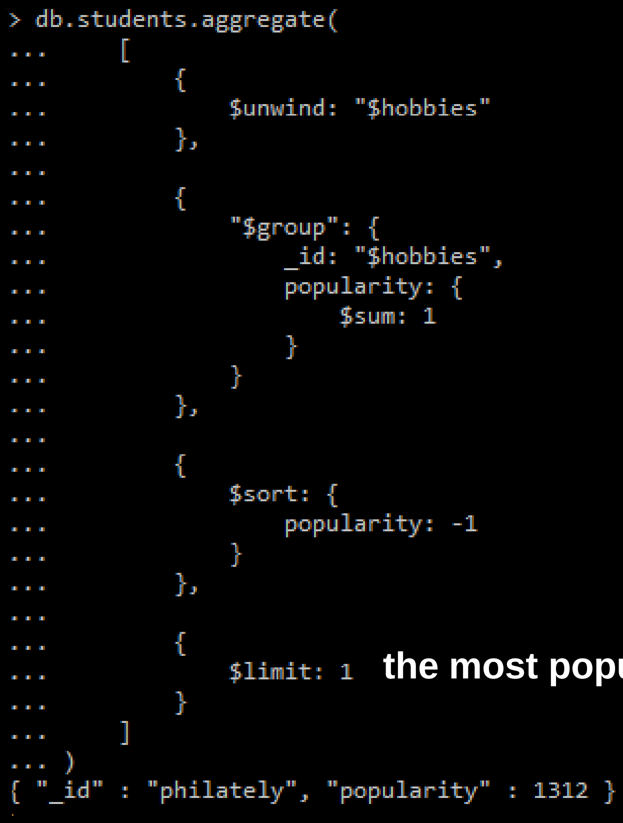
Query 2 : Produce a grouping of the documents that contains the name of each home city and the number of students enrolled from that home city.



```
> db.students.aggregate(
...     [
...         {
...             $group: {
...                 _id: "$home_city",
...                 enrolledStudents: {
...                     $sum: 1
...                 }
...             }
...         }
...     ]
... )
{ "_id" : "Patra", "enrolledStudents" : 463 }
{ "_id" : "Arta", "enrolledStudents" : 492 }
{ "_id" : "Katerini", "enrolledStudents" : 503 }
{ "_id" : "Agrinio", "enrolledStudents" : 507 }
{ "_id" : "Ioannina", "enrolledStudents" : 481 }
{ "_id" : "Pyrgos", "enrolledStudents" : 453 }
{ "_id" : "Messolongi", "enrolledStudents" : 498 }
{ "_id" : "Irakleio", "enrolledStudents" : 510 }
{ "_id" : "Thyra", "enrolledStudents" : 520 }
```

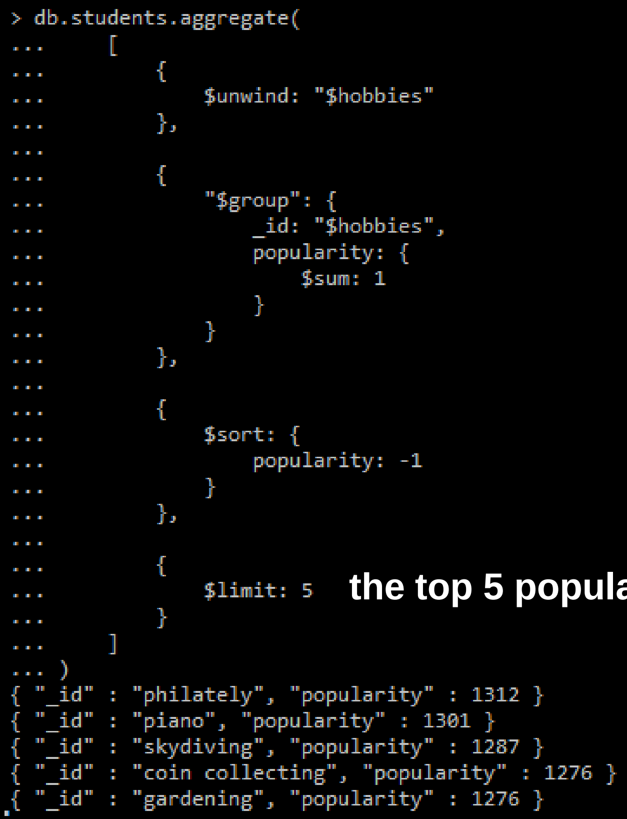
Part 1 - Queries and the Aggregation Pipeline [3]

Query 3 : Which hobby or hobbies are the most popular?



```
> db.students.aggregate(
...     [
...         {
...             $unwind: "$hobbies"
...         },
...
...         {
...             "$group": {
...                 _id: "$hobbies",
...                 popularity: {
...                     $sum: 1
...                 }
...             }
...         },
...
...         {
...             $sort: {
...                 popularity: -1
...             }
...         },
...
...         {
...             $limit: 1
...         }
...     ]
... )
{ "_id" : "philately", "popularity" : 1312 }
```

the most popular

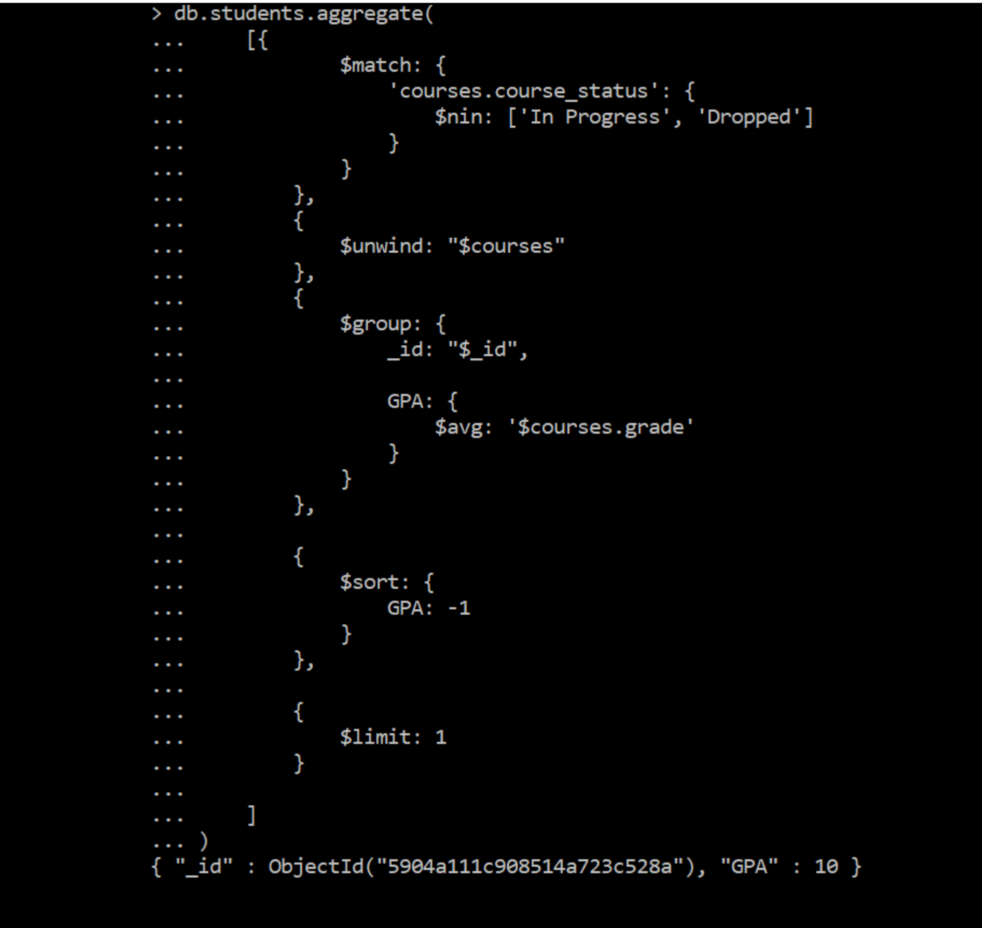


```
> db.students.aggregate(
...     [
...         {
...             $unwind: "$hobbies"
...         },
...
...         {
...             "$group": {
...                 _id: "$hobbies",
...                 popularity: {
...                     $sum: 1
...                 }
...             }
...         },
...
...         {
...             $sort: {
...                 popularity: -1
...             }
...         },
...
...         {
...             $limit: 5
...         }
...     ]
... )
{ "_id" : "philately", "popularity" : 1312 }
{ "_id" : "piano", "popularity" : 1301 }
{ "_id" : "skydiving", "popularity" : 1287 }
{ "_id" : "coin collecting", "popularity" : 1276 }
{ "_id" : "gardening", "popularity" : 1276 }
```

the top 5 popular

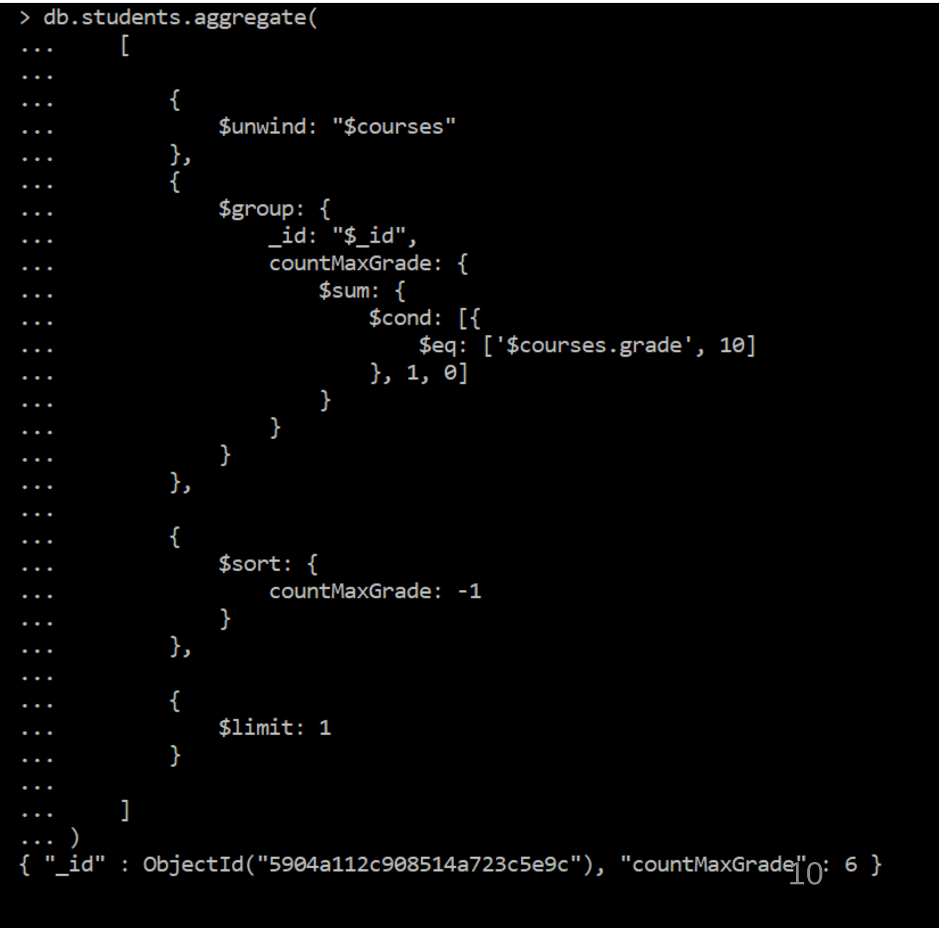
Part 1 - Queries and the Aggregation Pipeline [4]

Query 4 : What is the GPA (ignoring dropped classes and in progress classes) of the best student?



```
> db.students.aggregate(
...     [{
...             $match: {
...                 'courses.course_status': {
...                     $nin: ['In Progress', 'Dropped']
...                 }
...             }
...         },
...         {
...             $unwind: "$courses"
...         },
...         {
...             $group: {
...                 _id: "$_id",
...
...                 GPA: {
...                     $avg: '$courses.grade'
...                 }
...             }
...         },
...
...         {
...             $sort: {
...                 GPA: -1
...             }
...         },
...
...         {
...             $limit: 1
...         }
...
...     ]
... )
{ "_id" : ObjectId("5904a111c908514a723c528a"), "GPA" : 10 }
```

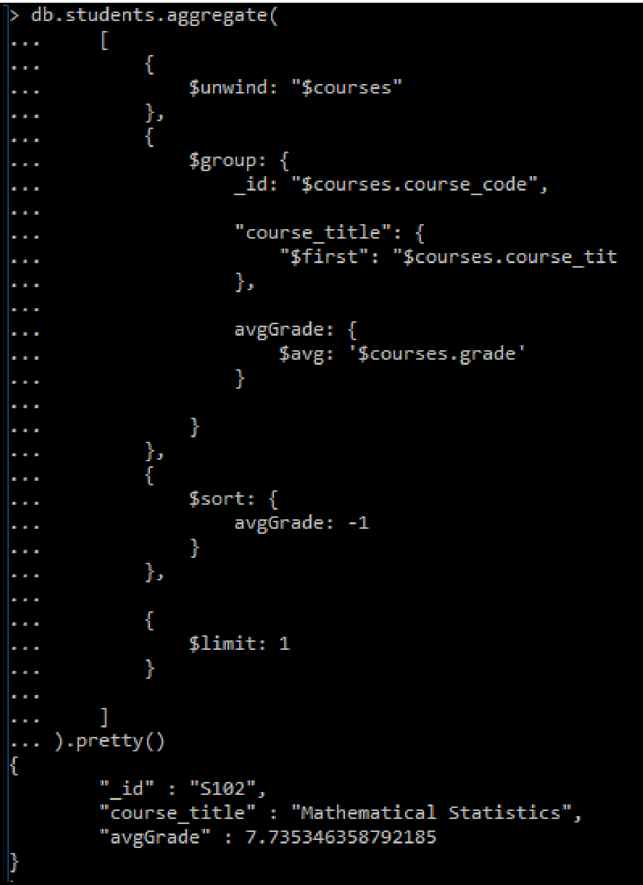
Query 5 : Which student has the largest number of grade 10’s?



```
> db.students.aggregate(
...     [
...
...         {
...             $unwind: "$courses"
...         },
...         {
...             $group: {
...                 _id: "$_id",
...                 countMaxGrade: {
...                     $sum: {
...                         $cond: [{
...                             $eq: ['$courses.grade', 10]
...                         }, 1, 0]
...                     }
...                 }
...             }
...         },
...
...         {
...             $sort: {
...                 countMaxGrade: -1
...             }
...         },
...
...         {
...             $limit: 1
...         }
...
...     ]
... )
{ "_id" : ObjectId("5904a112c908514a723c5e9c"), "countMaxGrade" : 6 }
```

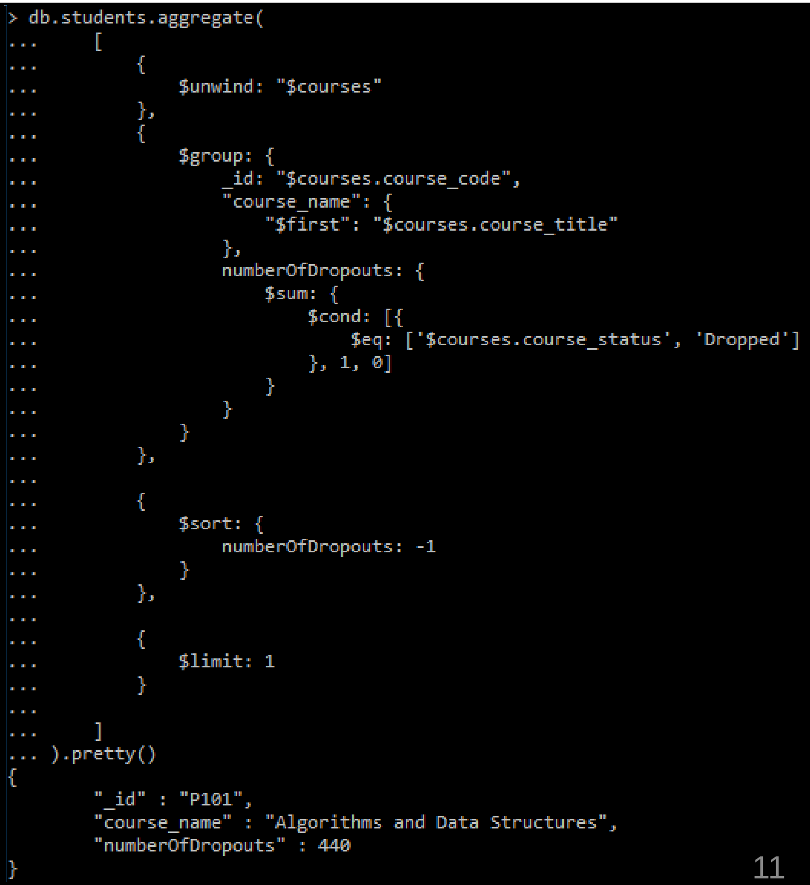
Part 1 - Queries and the Aggregation Pipeline [5]

Query 6 : Which class has the highest average GPA?



```
> db.students.aggregate(
...     [
...         {
...             $unwind: "$courses"
...         },
...         {
...             $group: {
...                 _id: "$courses.course_code",
...
...                 "course_title": {
...                     "$first": "$courses.course_tit
...                 },
...
...                 avgGrade: {
...                     $avg: '$courses.grade'
...                 }
...
...             }
...         },
...         {
...             $sort: {
...                 avgGrade: -1
...             }
...         },
...
...         {
...             $limit: 1
...         }
...
...     ]
... ).pretty()
{
        "_id" : "S102",
        "course_title" : "Mathematical Statistics",
        "avgGrade" : 7.735346358792185
}
```

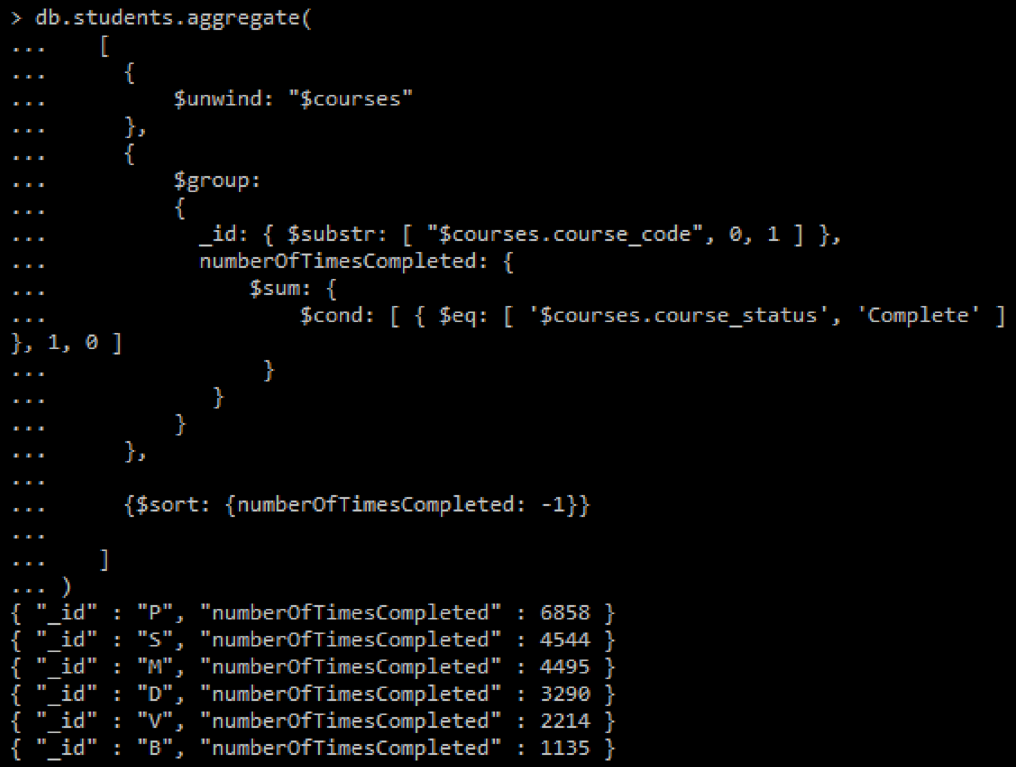
Query 7 : Which class has been dropped the most number of times?



```
> db.students.aggregate(
...     [
...         {
...             $unwind: "$courses"
...         },
...         {
...             $group: {
...                 _id: "$courses.course_code",
...                 "course_name": {
...                     "$first": "$courses.course_title"
...                 },
...                 numberOfDropouts: {
...                     $sum: {
...                         $cond: [{
...                             $eq: ['$courses.course_status', 'Dropped']
...                         }, 1, 0]
...                     }
...                 }
...             }
...         },
...
...         {
...             $sort: {
...                 numberOfDropouts: -1
...             }
...         },
...
...         {
...             $limit: 1
...         }
...
...     ]
... ).pretty()
{
        "_id" : "P101",
        "course_name" : "Algorithms and Data Structures",
        "numberOfDropouts" : 440
}
```

Part 1 - Queries and the Aggregation Pipeline [6]

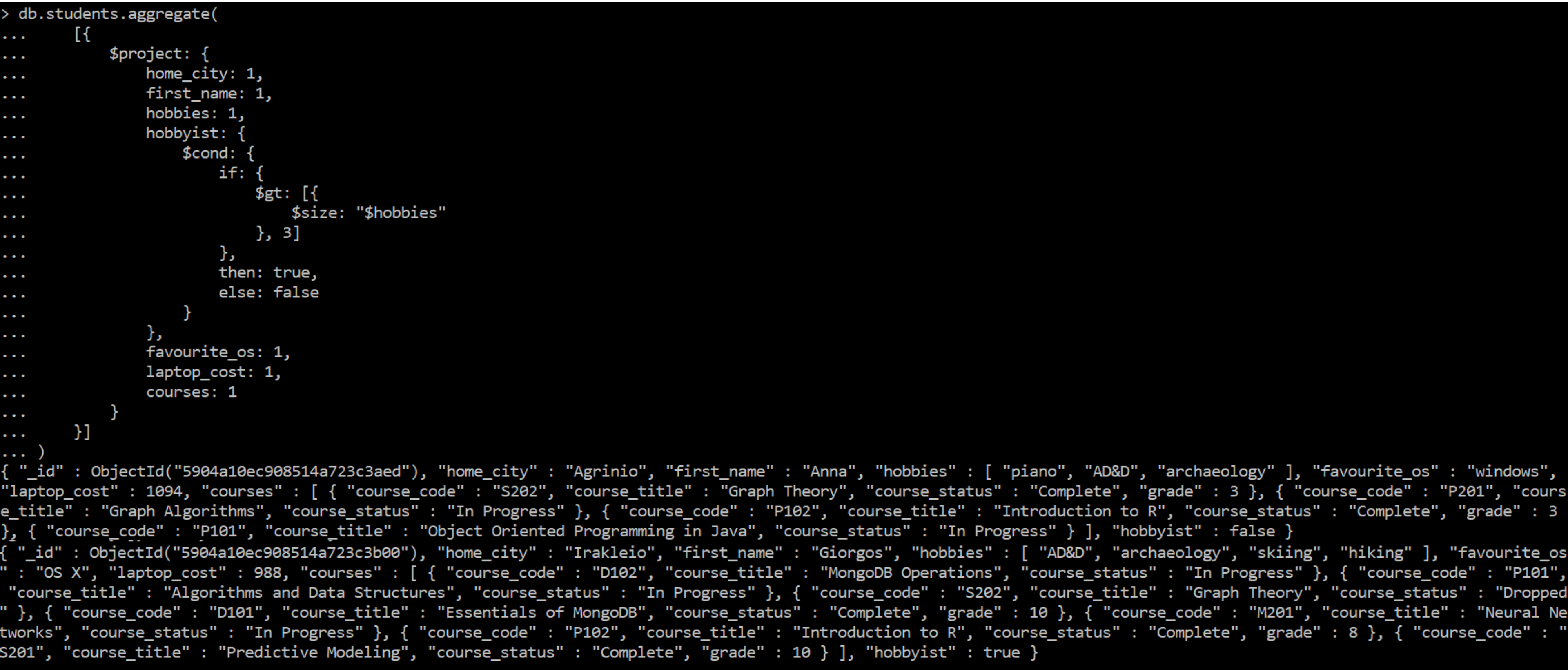
Query 8 : Produce of a count of classes that have been COMPLETED by class type. The class type is found by taking the first letter of the course code so that M102 has type M.



```
> db.students.aggregate(
...    [
...      {
...          $unwind: "$courses"
...      },
...      {
...          $group:
...          {
...            _id: { $substr: [ "$courses.course_code", 0, 1 ] },
...            numberOfTimesCompleted: {
...                $sum: {
...                    $cond: [ { $eq: [ '$courses.course_status', 'Complete' ]
}, 1, 0 ]
...                 }
...             }
...          }
...      },
...
...      {$sort: {numberOfTimesCompleted: -1}}
...
...    ]
... )
{ "_id" : "P", "numberOfTimesCompleted" : 6858 }
{ "_id" : "S", "numberOfTimesCompleted" : 4544 }
{ "_id" : "M", "numberOfTimesCompleted" : 4495 }
{ "_id" : "D", "numberOfTimesCompleted" : 3290 }
{ "_id" : "V", "numberOfTimesCompleted" : 2214 }
{ "_id" : "B", "numberOfTimesCompleted" : 1135 }
```

Part 1 - Queries and the Aggregation Pipeline [7]

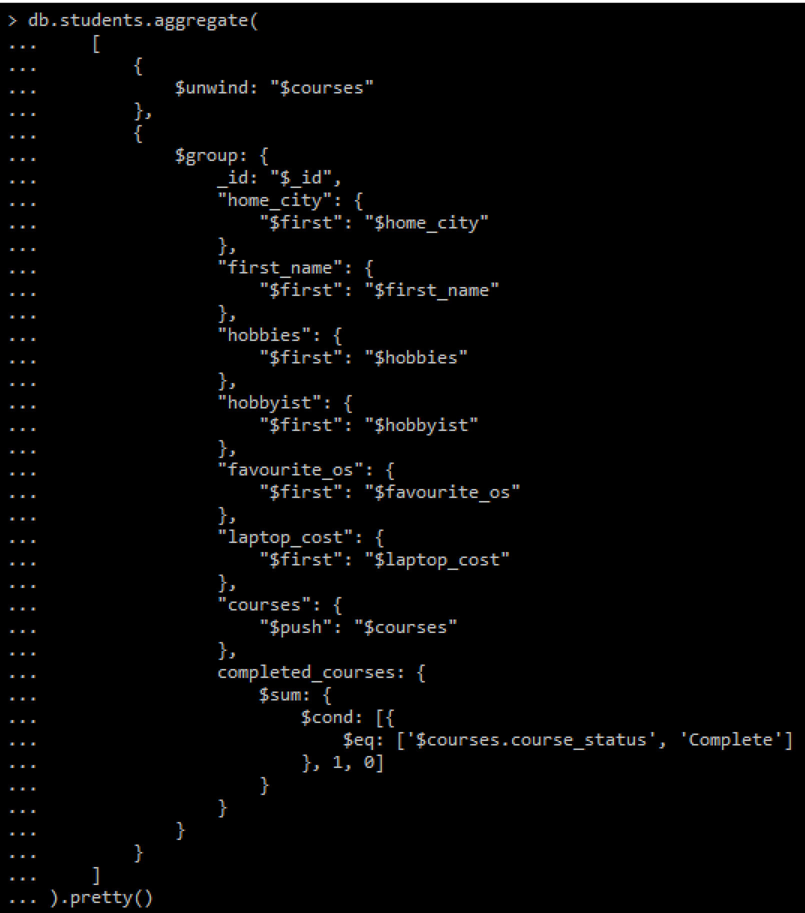
Query 9 : Produce a transformation of the documents so that the documents now have an additional boolean field called “hobbyist” that is true when the student has more than 3 hobbies and false otherwise.



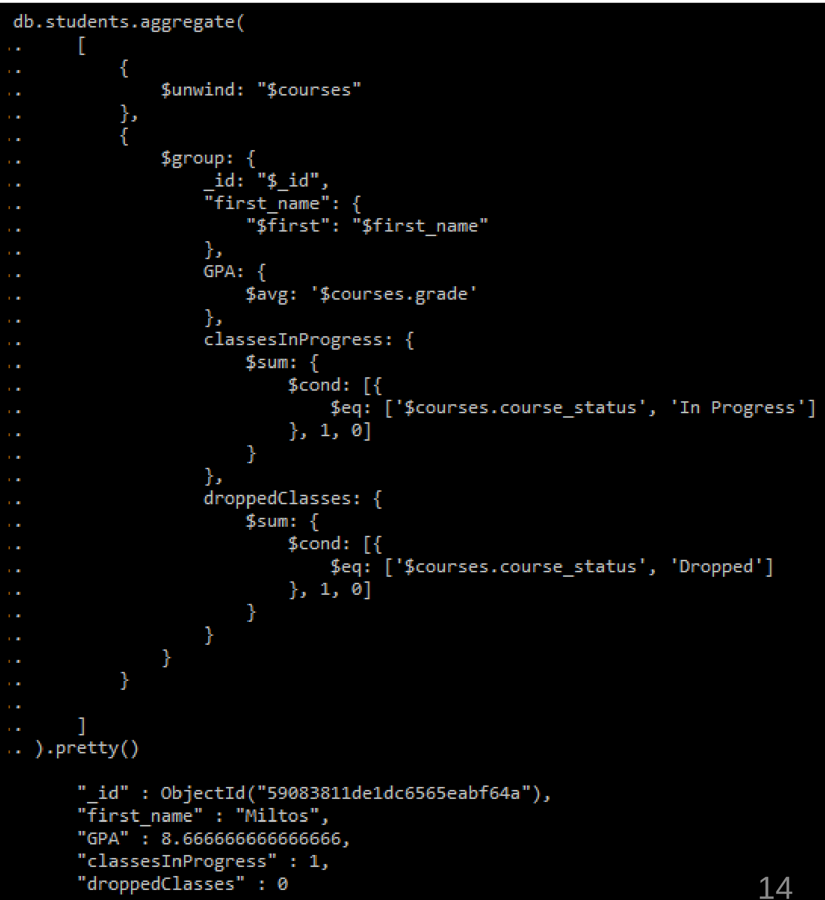
```
> db.students.aggregate(
...     [{
...         $project: {
...             home_city: 1,
...             first_name: 1,
...             hobbies: 1,
...             hobbyist: {
...                 $cond: {
...                     if: {
...                         $gt: [{
...                             $size: "$hobbies"
...                         }, 3]
...                     },
...                     then: true,
...                     else: false
...                 }
...             },
...             favourite_os: 1,
...             laptop_cost: 1,
...             courses: 1
...         }
...     }]
... )
{ "_id" : ObjectId("5904a10ec908514a723c3aed"), "home_city" : "Agrinio", "first_name" : "Anna", "hobbies" : [ "piano", "AD&D", "archaeology" ], "favourite_os" : "windows",
"laptop_cost" : 1094, "courses" : [ { "course_code" : "S202", "course_title" : "Graph Theory", "course_status" : "Complete", "grade" : 3 }, { "course_code" : "P201", "cours
e_title" : "Graph Algorithms", "course_status" : "In Progress" }, { "course_code" : "P102", "course_title" : "Introduction to R", "course_status" : "Complete", "grade" : 3
}, { "course_code" : "P101", "course_title" : "Object Oriented Programming in Java", "course_status" : "In Progress" } ], "hobbyist" : false }
{ "_id" : ObjectId("5904a10ec908514a723c3b00"), "home_city" : "Irakleio", "first_name" : "Giorgos", "hobbies" : [ "AD&D", "archaeology", "skiing", "hiking" ], "favourite_os
" : "OS X", "laptop_cost" : 988, "courses" : [ { "course_code" : "D102", "course_title" : "MongoDB Operations", "course_status" : "In Progress" }, { "course_code" : "P101",
 "course_title" : "Algorithms and Data Structures", "course_status" : "In Progress" }, { "course_code" : "S202", "course_title" : "Graph Theory", "course_status" : "Dropped
" }, { "course_code" : "D101", "course_title" : "Essentials of MongoDB", "course_status" : "Complete", "grade" : 10 }, { "course_code" : "M201", "course_title" : "Neural Ne
tworks", "course_status" : "In Progress" }, { "course_code" : "P102", "course_title" : "Introduction to R", "course_status" : "Complete", "grade" : 8 }, { "course_code" : "
S201", "course_title" : "Predictive Modeling", "course_status" : "Complete", "grade" : 10 } ], "hobbyist" : true }
```

Part 1 - Queries and the Aggregation Pipeline [8]

Query 10 : Produce a transformation of the documents so that the documents now have an additional field that contains the number of classes that the student has completed.



```
> db.students.aggregate(
...     [
...         {
...             $unwind: "$courses"
...         },
...         {
...             $group: {
...                 _id: "$_id",
...                 "home_city": {
...                     "$first": "$home_city"
...                 },
...                 "first_name": {
...                     "$first": "$first_name"
...                 },
...                 "hobbies": {
...                     "$first": "$hobbies"
...                 },
...                 "hobbyist": {
...                     "$first": "$hobbyist"
...                 },
...                 "favourite_os": {
...                     "$first": "$favourite_os"
...                 },
...                 "laptop_cost": {
...                     "$first": "$laptop_cost"
...                 },
...                 "courses": {
...                     "$push": "$courses"
...                 },
...                 completed_courses: {
...                     $sum: {
...                         $cond: [{
...                             $eq: ['$courses.course_status', 'Complete']
...                         }, 1, 0]
...                     }
...                 }
...             }
...         }
...     ]
... ).pretty()
```

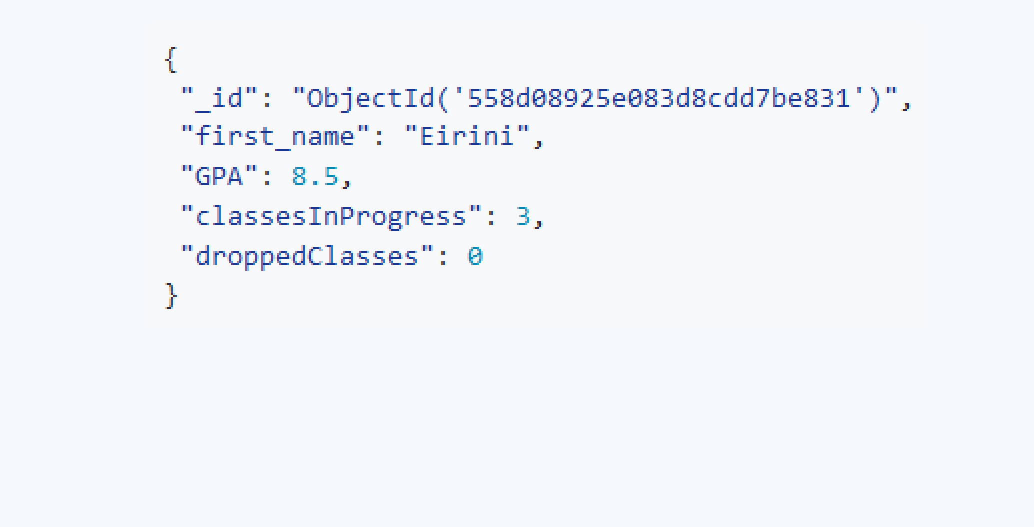


```
db.students.aggregate(
..      [
..          {
..              $unwind: "$courses"
..          },
..          {
..              $group: {
..                  _id: "$_id",
..                  "first_name": {
..                      "$first": "$first_name"
..                  },
..                  GPA: {
..                      $avg: '$courses.grade'
..                  },
..                  classesInProgress: {
..                      $sum: {
..                          $cond: [{
..                              $eq: ['$courses.course_status', 'In Progress']
..                          }, 1, 0]
..                      }
..                  },
..                  droppedClasses: {
..                      $sum: {
..                          $cond: [{
..                              $eq: ['$courses.course_status', 'Dropped']
..                          }, 1, 0]
..                      }
..                  }
..              }
..          }
..
..      ]
.. ).pretty()

        "_id" : ObjectId("59083811de1dc6565eabf64a"),
        "first_name" : "Miltos",
        "GPA" : 8.666666666666666,
        "classesInProgress" : 1,
        "droppedClasses" : 0
```

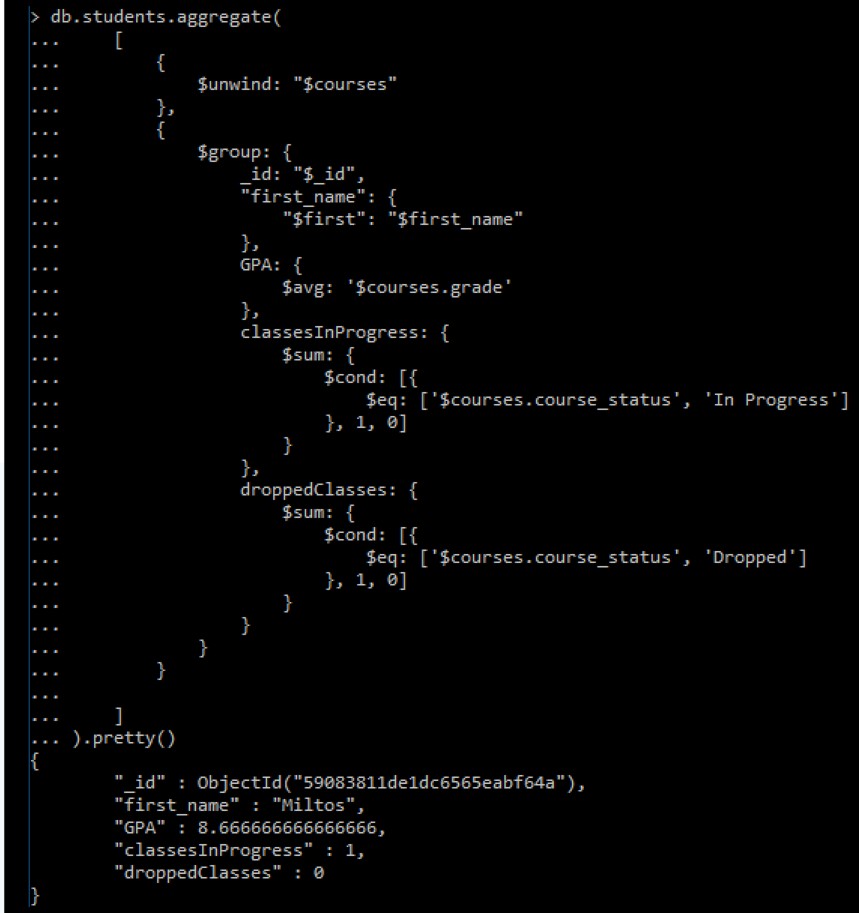
Part 1 - Queries and the Aggregation Pipeline [9]

Query 11 : Produce a transformation of the documents in the collection so that they look like the following output.

The GPA is the average grade of all the completed classes. The other two computed fields are the number of classes currently in progress and the number of classes dropped. No other fields should be in there. No other fields should be present.



```
{
 "_id": "ObjectId('558d08925e083d8cdd7be831')",
 "first_name": "Eirini",
 "GPA": 8.5,
 "classesInProgress": 3,
 "droppedClasses": 0
}
```

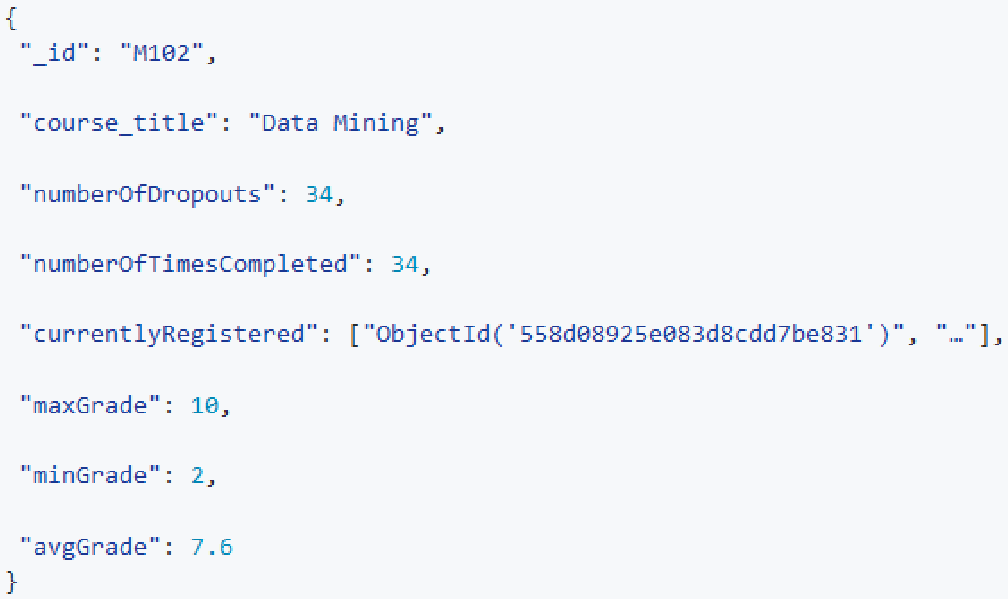


```
> db.students.aggregate(
...     [
...         {
...             $unwind: "$courses"
...         },
...         {
...             $group: {
...                 _id: "$_id",
...                 "first_name": {
...                     "$first": "$first_name"
...                 },
...                 GPA: {
...                     $avg: '$courses.grade'
...                 },
...                 classesInProgress: {
...                     $sum: {
...                         $cond: [{
...                             $eq: ['$courses.course_status', 'In Progress']
...                         }, 1, 0]
...                     }
...                 },
...                 droppedClasses: {
...                     $sum: {
...                         $cond: [{
...                             $eq: ['$courses.course_status', 'Dropped']
...                         }, 1, 0]
...                     }
...                 }
...             }
...         }
...
...     ]
... ).pretty()
{
        "_id" : ObjectId("59083811de1dc6565eabf64a"),
        "first_name" : "Miltos",
        "GPA" : 8.666666666666666,
        "classesInProgress" : 1,
        "droppedClasses" : 0
}
```

Part 1 - Queries and the Aggregation Pipeline [10]

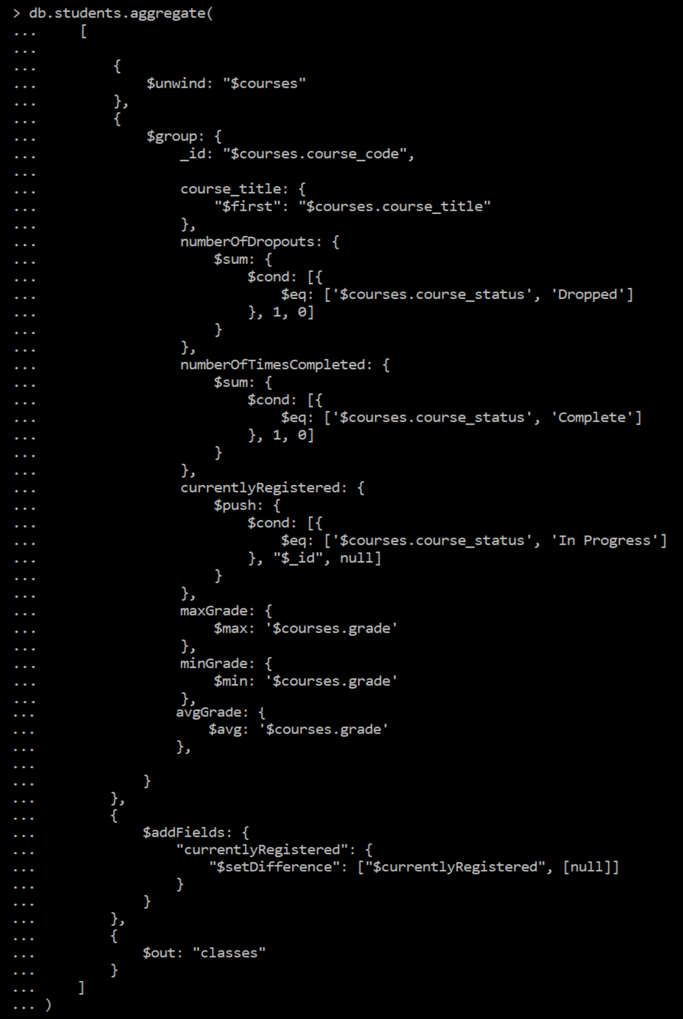
Query 12 : Produce a NEW collection (HINT: Use $out in the aggregation pipeline) so that the new documents in this correspond to the classes on offer. The structure of the documents should be like the following output. The _id field should be the course code.

The course_title is what it was before. The numberOfDropouts is the number of students who dropped out. The numberOfTimesCompleted is the number of students that completed this class. The currentlyRegistered array is an array of ObjectID’s corresponding to the students who are currently taking the class. Finally, for the students that completed the class, the maxGrade, minGrade and avgGrade are the summary statistics for that class.

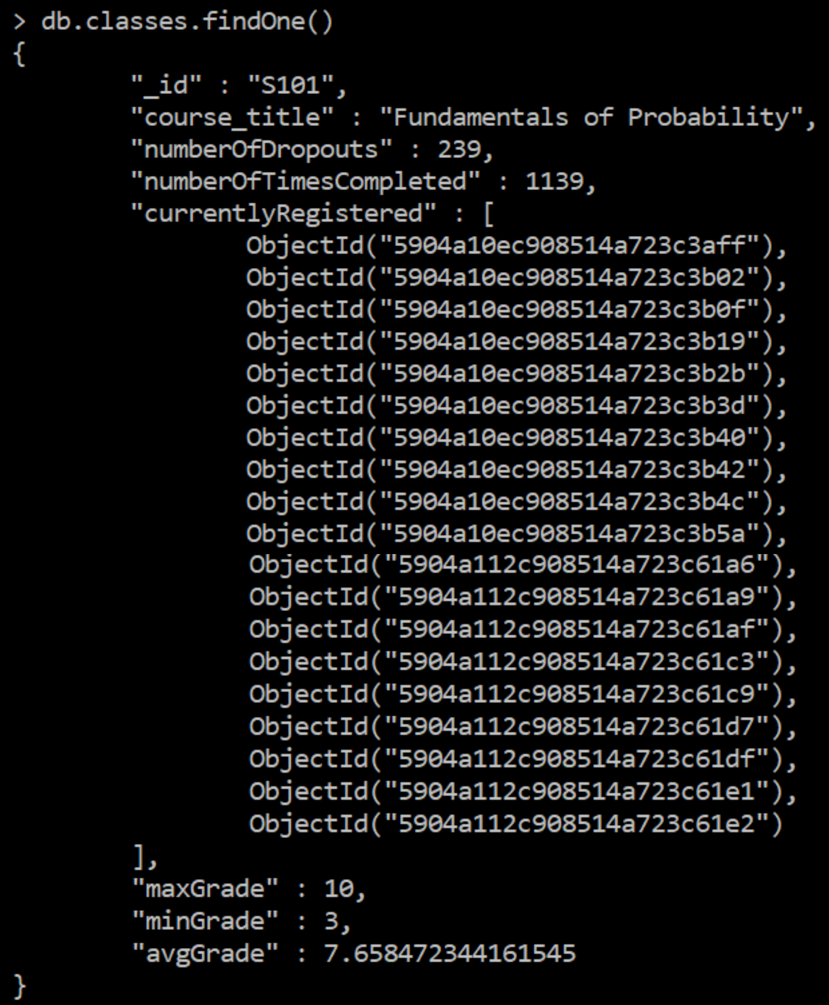


```
{
 "_id": "M102",

 "course_title": "Data Mining",

 "numberOfDropouts": 34,

 "numberOfTimesCompleted": 34,

 "currentlyRegistered": ["ObjectId('558d08925e083d8cdd7be831')", "…"],

 "maxGrade": 10,

 "minGrade": 2,

 "avgGrade": 7.6
}
```

Part 1 - Queries and the Aggregation Pipeline [11]



```
> db.students.aggregate(
...     [
...
...         {
...             $unwind: "$courses"
...         },
...         {
...             $group: {
...                 _id: "$courses.course_code",
...
...                 course_title: {
...                     "$first": "$courses.course_title"
...                 },
...                 numberOfDropouts: {
...                     $sum: {
...                         $cond: [{
...                             $eq: ['$courses.course_status', 'Dropped']
...                         }, 1, 0]
...                     }
...                 },
...                 numberOfTimesCompleted: {
...                     $sum: {
...                         $cond: [{
...                             $eq: ['$courses.course_status', 'Complete']
...                         }, 1, 0]
...                     }
...                 },
...                 currentlyRegistered: {
...                     $push: {
...                         $cond: [{
...                             $eq: ['$courses.course_status', 'In Progress']
...                         }, "$_id", null]
...                     }
...                 },
...                 maxGrade: {
...                     $max: '$courses.grade'
...                 },
...                 minGrade: {
...                     $min: '$courses.grade'
...                 },
...                avgGrade: {
...                    $avg: '$courses.grade'
...                },
...
...             }
...         },
...         {
...             $addFields: {
...                 "currentlyRegistered": {
...                     "$setDifference": ["$currentlyRegistered", [null]]
...                 }
...             }
...         },
...         {
...             $out: "classes"
...         }
...     ]
... )
```

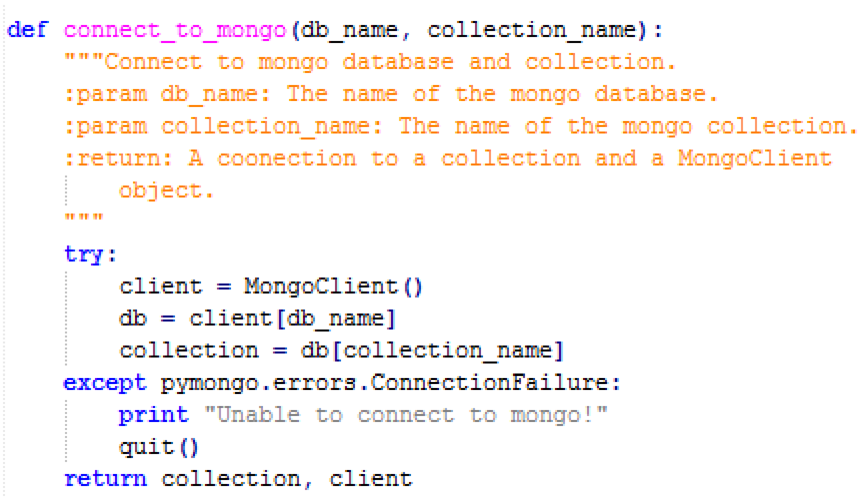


```
> db.classes.findOne()
{
        "_id" : "S101",
        "course_title" : "Fundamentals of Probability",
        "numberOfDropouts" : 239,
        "numberOfTimesCompleted" : 1139,
        "currentlyRegistered" : [
                ObjectId("5904a10ec908514a723c3aff"),
                ObjectId("5904a10ec908514a723c3b02"),
                ObjectId("5904a10ec908514a723c3b0f"),
                ObjectId("5904a10ec908514a723c3b19"),
                ObjectId("5904a10ec908514a723c3b2b"),
                ObjectId("5904a10ec908514a723c3b3d"),
                ObjectId("5904a10ec908514a723c3b40"),
                ObjectId("5904a10ec908514a723c3b42"),
                ObjectId("5904a10ec908514a723c3b4c"),
                ObjectId("5904a10ec908514a723c3b5a"),
                ObjectId("5904a112c908514a723c61a6"),
                ObjectId("5904a112c908514a723c61a9"),
                ObjectId("5904a112c908514a723c61af"),
                ObjectId("5904a112c908514a723c61c3"),
                ObjectId("5904a112c908514a723c61c9"),
                ObjectId("5904a112c908514a723c61d7"),
                ObjectId("5904a112c908514a723c61df"),
                ObjectId("5904a112c908514a723c61e1"),
                ObjectId("5904a112c908514a723c61e2")
        ],
        "maxGrade" : 10,
        "minGrade" : 3,
        "avgGrade" : 7.658472344161545
}
```

Part 2 - Python & MongoDB [1]

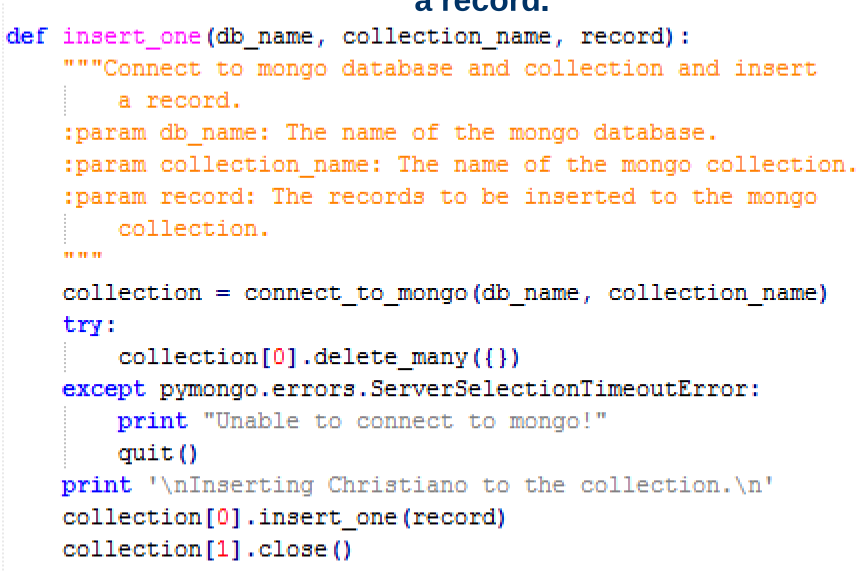
python_mongodb.py: Implement simple operations on mongo database.

Connect to mongo database and collection.



```
def connect_to_mongo(db_name, collection_name):
    """Connect to mongo database and collection.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    :return: A coonection to a collection and a MongoClient
        object.
    """
    try:
        client = MongoClient()
        db = client[db_name]
        collection = db[collection_name]
    except pymongo.errors.ConnectionFailure:
        print "Unable to connect to mongo!"
        quit()
    return collection, client
```

Connect to mongo database and collection and insert a record.

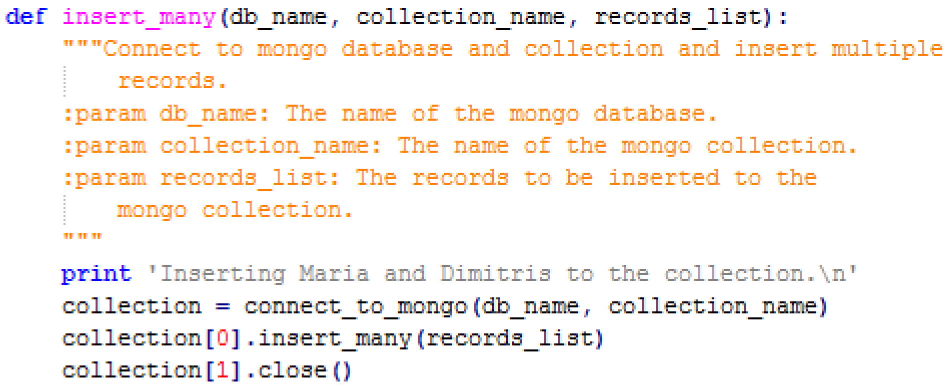


```
def insert_one(db_name, collection_name, record):
    """Connect to mongo database and collection and insert
        a record.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    :param record: The records to be inserted to the mongo
        collection.
    """
    collection = connect_to_mongo(db_name, collection_name)
    try:
        collection[0].delete_many({})
    except pymongo.errors.ServerSelectionTimeoutError:
        print "Unable to connect to mongo!"
        quit()
    print '\nInserting Christiano to the collection.\n'
    collection[0].insert_one(record)
    collection[1].close()
```

Part 2 - Python & MongoDB [2]

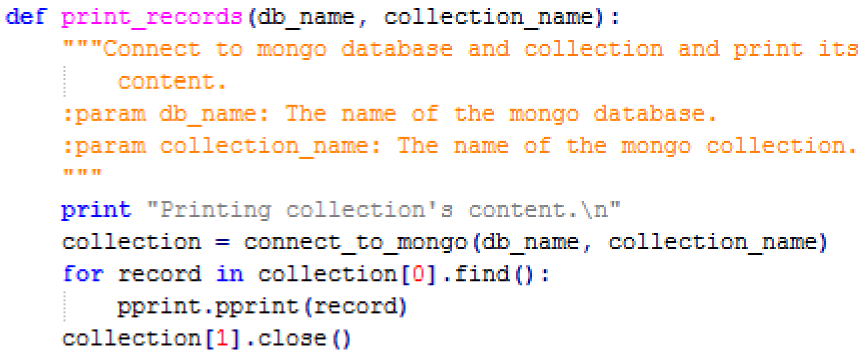
python_mongodb.py: Implement simple operations on mongo database.

Connect to mongo database and collection and insert multiple records.



```
def insert_many(db_name, collection_name, records_list):
    """Connect to mongo database and collection and insert multiple
        records.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    :param records_list: The records to be inserted to the
        mongo collection.
    """
    print 'Inserting Maria and Dimitris to the collection.\n'
    collection = connect_to_mongo(db_name, collection_name)
    collection[0].insert_many(records_list)
    collection[1].close()
```

Connect to mongo database and collection and print its content.

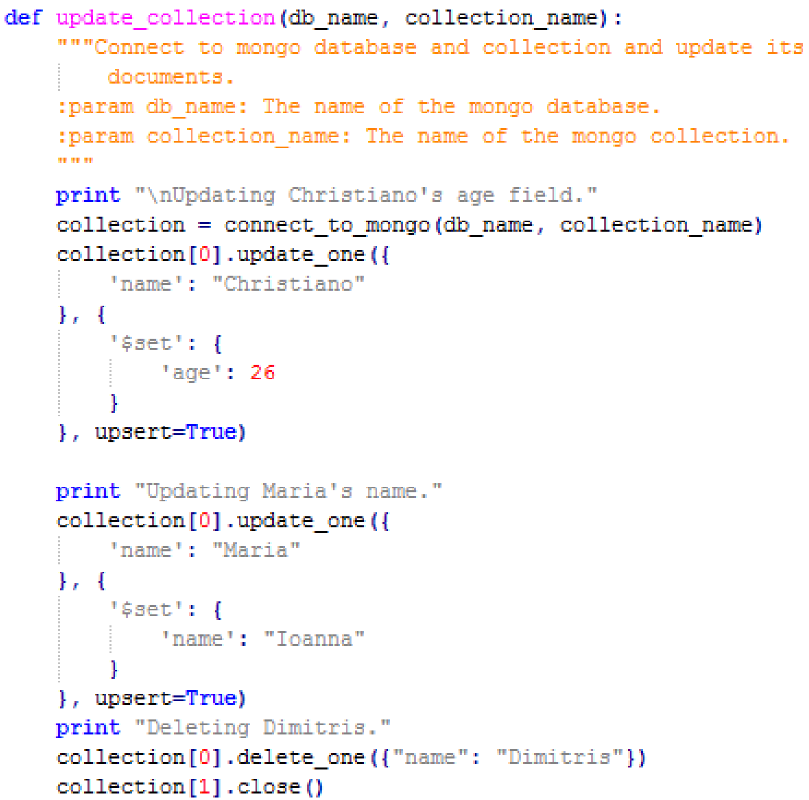


```
def print_records(db_name, collection_name):
    """Connect to mongo database and collection and print its
        content.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    """
    print "Printing collection's content.\n"
    collection = connect_to_mongo(db_name, collection_name)
    for record in collection[0].find():
        pprint.pprint(record)
    collection[1].close()
```

Part 2 - Python & MongoDB [3]

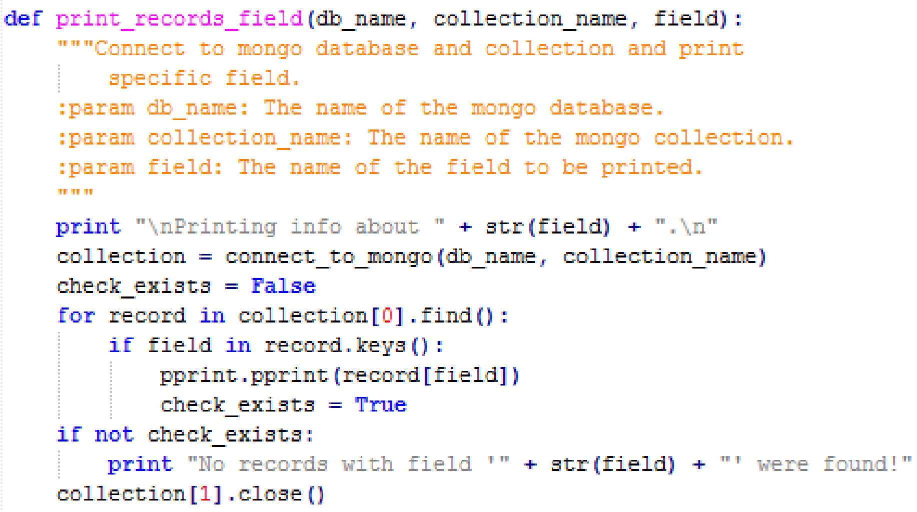
python_mongodb.py: Implement simple operations on mongo database.

Connect to mongo database and collection and update Its documents.



```
def update_collection(db_name, collection_name):
    """Connect to mongo database and collection and update its
        documents.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    """
    print "\nUpdating Christiano's age field."
    collection = connect_to_mongo(db_name, collection_name)
    collection[0].update_one({
        'name': "Christiano"
    }, {
        '$set': {
            'age': 26
        }
    }, upsert=True)

    print "Updating Maria's name."
    collection[0].update_one({
        'name': "Maria"
    }, {
        '$set': {
            'name': "Ioanna"
        }
    }, upsert=True)
    print "Deleting Dimitris."
    collection[0].delete_one({"name": "Dimitris"})
    collection[1].close()
```

Connect to mongo database and collection and print specific field.

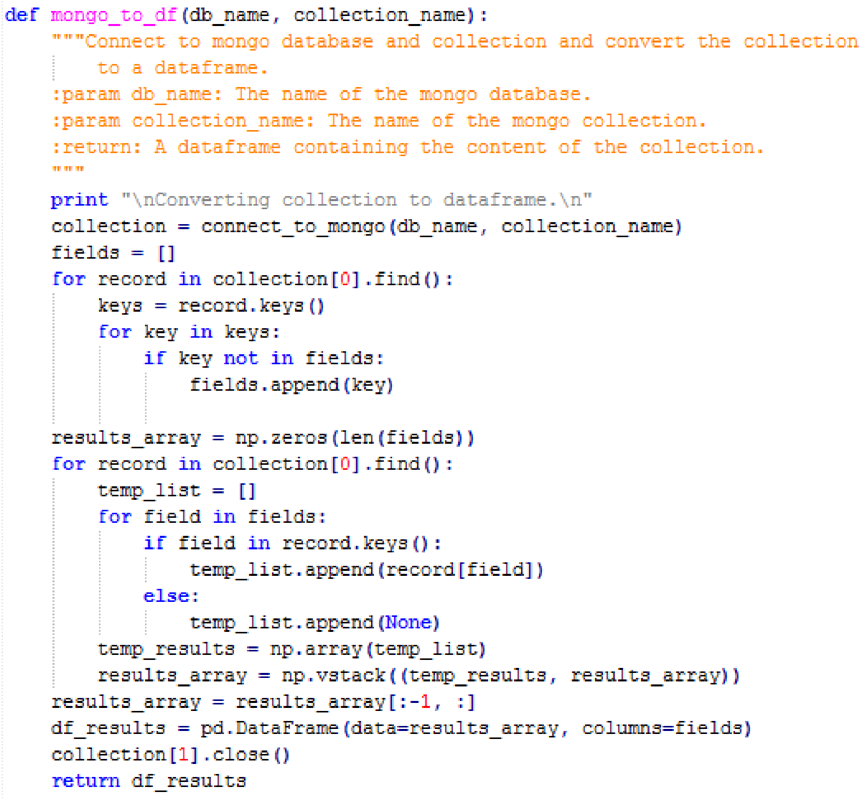


```
def print_records_field(db_name, collection_name, field):
    """Connect to mongo database and collection and print
        specific field.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    :param field: The name of the field to be printed.
    """
    print "\nPrinting info about " + str(field) + ".\n"
    collection = connect_to_mongo(db_name, collection_name)
    check_exists = False
    for record in collection[0].find():
        if field in record.keys():
            pprint.pprint(record[field])
            check_exists = True
    if not check_exists:
        print "No records with field '" + str(field) + "' were found!"
    collection[1].close()
```

Part 2 - Python & MongoDB [4]

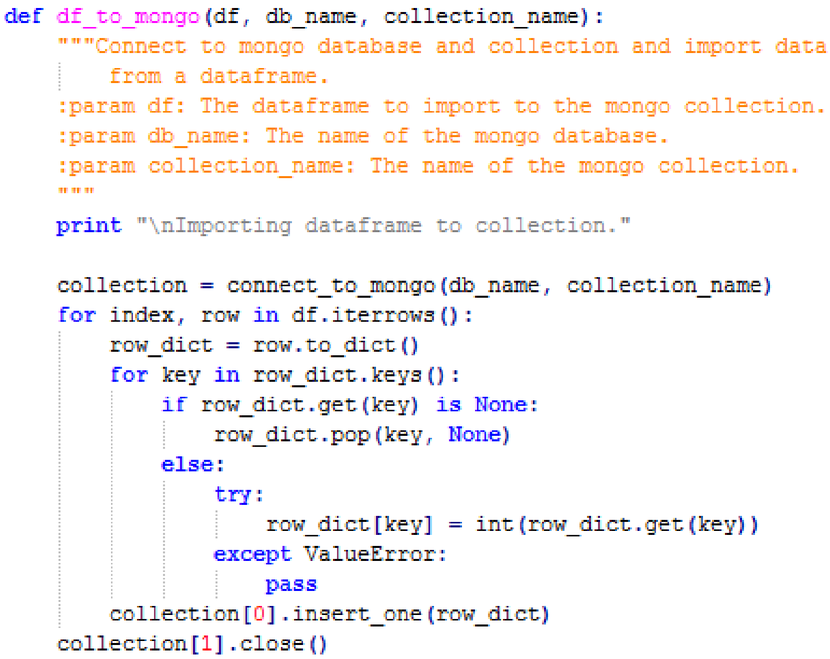
python_mongodb.py: Implement simple operations on mongo database.

Connect to mongo database and collection and convert the collection to a dataframe.



```python
def mongo_to_df(db_name, collection_name):
    """Connect to mongo database and collection and convert the collection
        to a dataframe.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    :return: A dataframe containing the content of the collection.
    """
    print "\nConverting collection to dataframe.\n"
    collection = connect_to_mongo(db_name, collection_name)
    fields = []
    for record in collection[0].find():
        keys = record.keys()
        for key in keys:
            if key not in fields:
                fields.append(key)

    results_array = np.zeros(len(fields))
    for record in collection[0].find():
        temp_list = []
        for field in fields:
            if field in record.keys():
                temp_list.append(record[field])
            else:
                temp_list.append(None)
        temp_results = np.array(temp_list)
        results_array = np.vstack((temp_results, results_array))
    results_array = results_array[:-1, :]
    df_results = pd.DataFrame(data=results_array, columns=fields)
    collection[1].close()
    return df_results
```

Connect to mongo database and collection and import data from a dataframe.

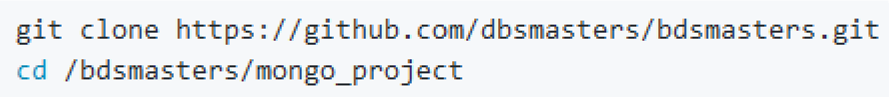


```python
def df_to_mongo(df, db_name, collection_name):
    """Connect to mongo database and collection and import data
        from a dataframe.
    :param df: The dataframe to import to the mongo collection.
    :param db_name: The name of the mongo database.
    :param collection_name: The name of the mongo collection.
    """
    print "\nImporting dataframe to collection."

    collection = connect_to_mongo(db_name, collection_name)
    for index, row in df.iterrows():
        row_dict = row.to_dict()
        for key in row_dict.keys():
            if row_dict.get(key) is None:
                row_dict.pop(key, None)
            else:
                try:
                    row_dict[key] = int(row_dict.get(key))
                except ValueError:
                    pass
        collection[0].insert_one(row_dict)
    collection[1].close()
```

Part 2 - Python & MongoDB [5]

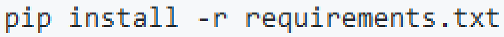
python_mongodb.py: Implement simple operations on mongo database.

1. Clone this repository:



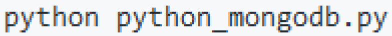
```
git clone https://github.com/dbsmasters/bdsmasters.git
cd /bdsmasters/mongo_project
```

2. Install the required python packages.



```
pip install -r requirements.txt
```

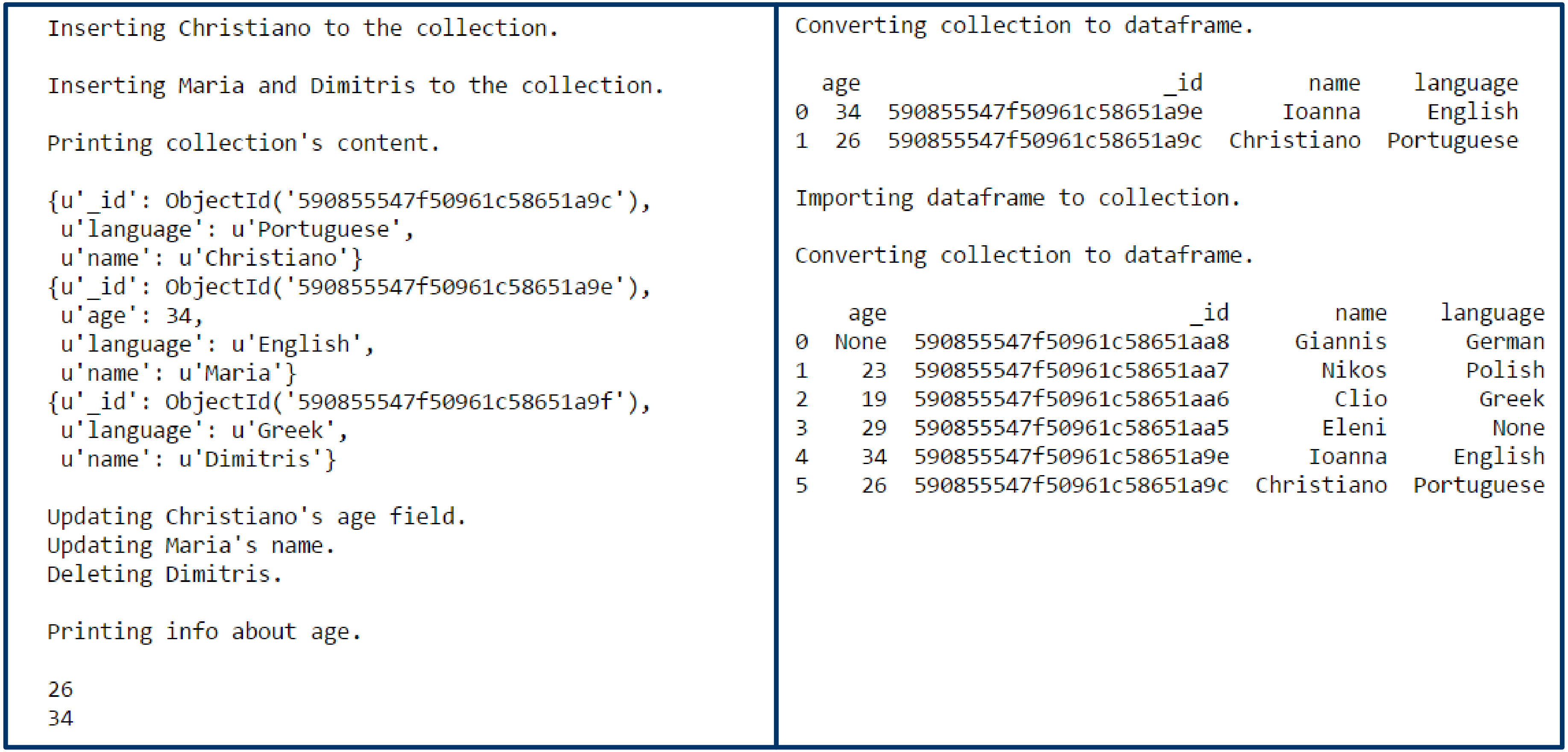
3. Run python_mongodb.py to implement basic operations (insert_one, insert_many, update, delete_one, delete_many, etc.) on mongodb.



```
python python_mongodb.py
```

Part 2 - Python & MongoDB [6]

Output

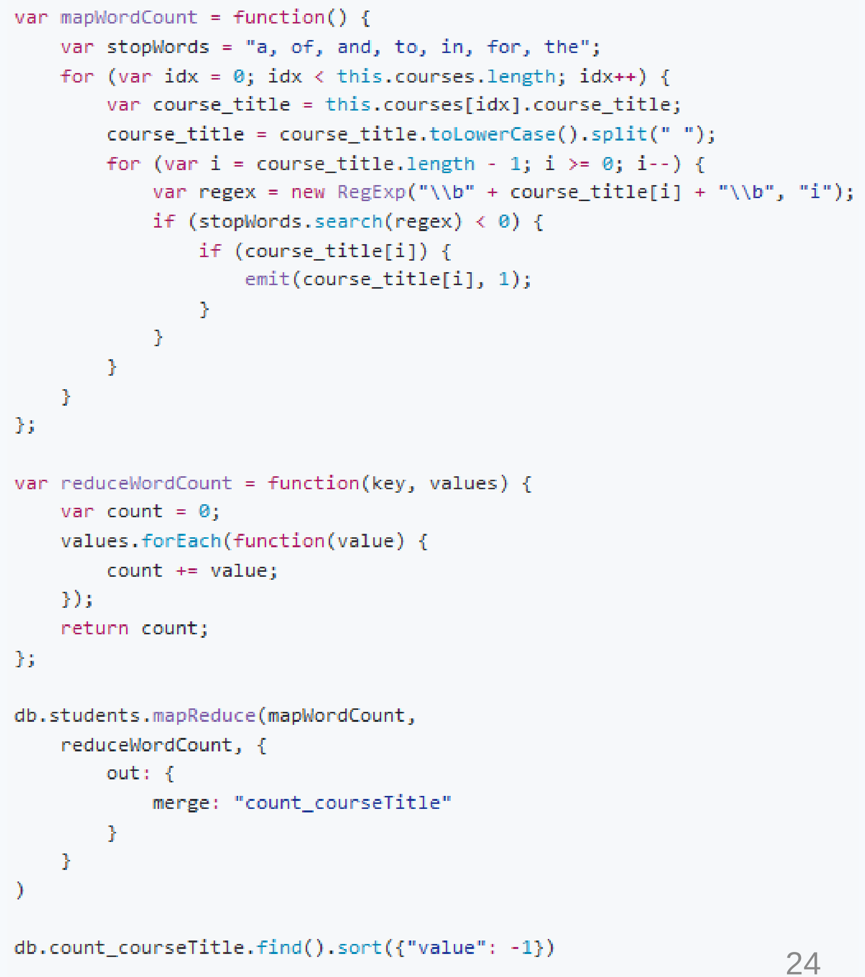


```
Inserting Christiano to the collection.

Inserting Maria and Dimitris to the collection.

Printing collection's content.

{u'_id': ObjectId('590855547f50961c58651a9c'),
 u'language': u'Portuguese',
 u'name': u'Christiano'}
{u'_id': ObjectId('590855547f50961c58651a9e'),
 u'age': 34,
 u'language': u'English',
 u'name': u'Maria'}
{u'_id': ObjectId('590855547f50961c58651a9f'),
 u'language': u'Greek',
 u'name': u'Dimitris'}

Updating Christiano's age field.
Updating Maria's name.
Deleting Dimitris.

Printing info about age.

26
34
```

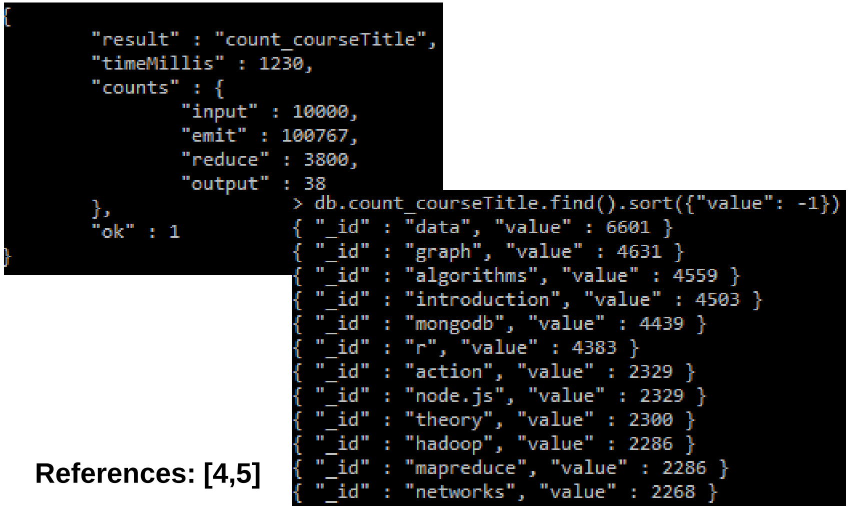
```
Converting collection to dataframe.

  age                       _id        name    language
0  34  590855547f50961c58651a9e      Ioanna     English
1  26  590855547f50961c58651a9c  Christiano  Portuguese

Importing dataframe to collection.

Converting collection to dataframe.

    age                       _id        name    language
0  None  590855547f50961c58651aa8     Giannis      German
1    23  590855547f50961c58651aa7       Nikos      Polish
2    19  590855547f50961c58651aa6        Clio       Greek
3    29  590855547f50961c58651aa5       Eleni        None
4    34  590855547f50961c58651a9e      Ioanna     English
5    26  590855547f50961c58651a9c  Christiano  Portuguese
```

Part 3 - MapReduce (Word Count) [1]

MapReduce 1 : Write a map reduce job on the students collection similar to the classic word count example. More specifically, implement a word count using the course title field as the text. In addition, exclude stop words from this list. You should find/write your own list of stop words. (Stop words are the common words in the English language like “a”, “in”, “to”, “the”, etc.)



```
var mapWordCount = function() {
    var stopWords = "a, of, and, to, in, for, the";
    for (var idx = 0; idx < this.courses.length; idx++) {
        var course_title = this.courses[idx].course_title;
        course_title = course_title.toLowerCase().split(" ");
        for (var i = course_title.length - 1; i >= 0; i--) {
            var regex = new RegExp("\\b" + course_title[i] + "\\b", "i");
            if (stopWords.search(regex) < 0) {
                if (course_title[i]) {
                    emit(course_title[i], 1);
                }
            }
        }
    }
};

var reduceWordCount = function(key, values) {
    var count = 0;
    values.forEach(function(value) {
        count += value;
    });
    return count;
};

db.students.mapReduce(mapWordCount,
    reduceWordCount, {
        out: {
            merge: "count_courseTitle"
        }
    }
)

db.count_courseTitle.find().sort({"value": -1})
```



```
{
        "result" : "count_courseTitle",
        "timeMillis" : 1230,
        "counts" : {
                "input" : 10000,
                "emit" : 100767,
                "reduce" : 3800,
                "output" : 38
        },
        "ok" : 1
}
```

```
> db.count_courseTitle.find().sort({"value": -1})
{ "_id" : "data", "value" : 6601 }
{ "_id" : "graph", "value" : 4631 }
{ "_id" : "algorithms", "value" : 4559 }
{ "_id" : "introduction", "value" : 4503 }
{ "_id" : "mongodb", "value" : 4439 }
{ "_id" : "r", "value" : 4383 }
{ "_id" : "action", "value" : 2329 }
{ "_id" : "node.js", "value" : 2329 }
{ "_id" : "theory", "value" : 2300 }
{ "_id" : "hadoop", "value" : 2286 }
{ "_id" : "mapreduce", "value" : 2286 }
{ "_id" : "networks", "value" : 2268 }
```

References: [4,5]

Part 3 - MapReduce (Word Count) [2]

course_title

[Predictive Modeling]

[MongoDB Operations]

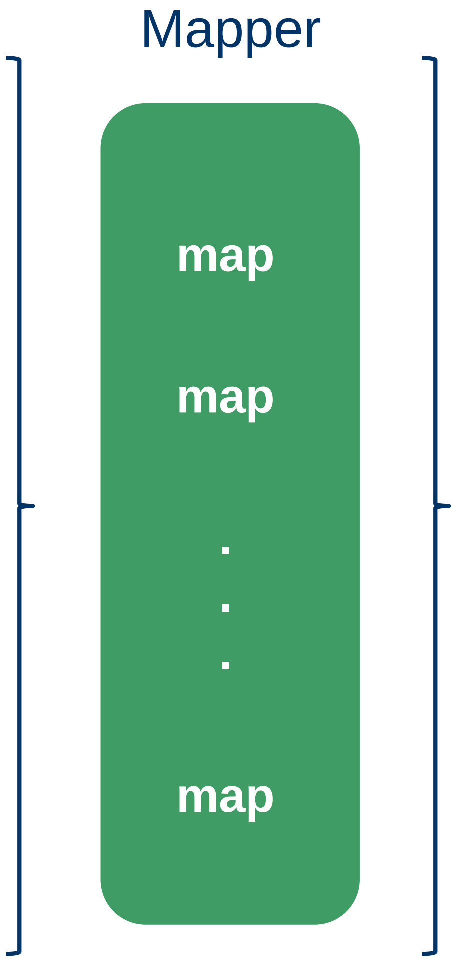
[Hadoop and MapReduce]

[Data Mining]

[MongoDB Operations]

[Data Mining]

. . . .



Mapper

map

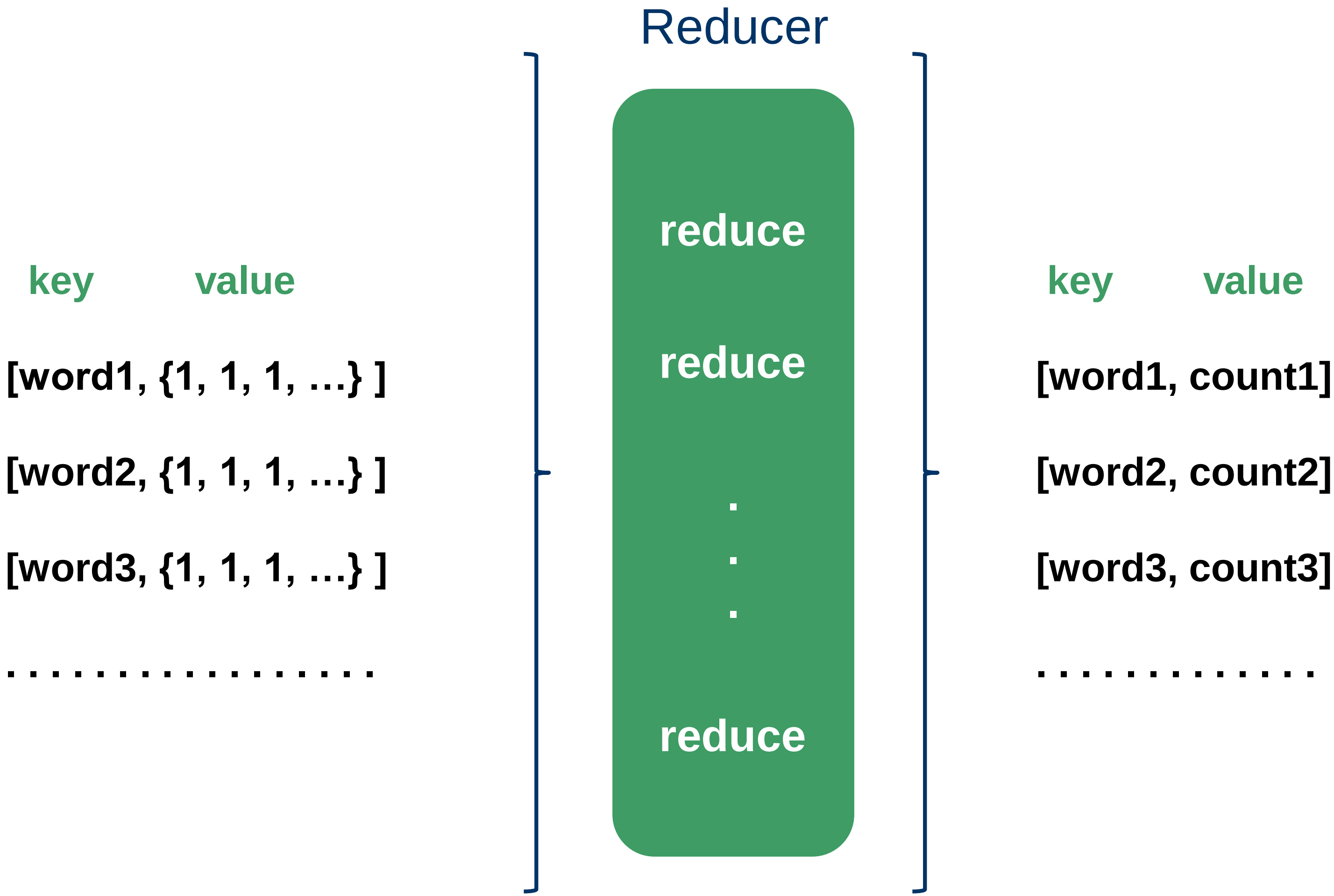
map

.
.
.

map

key	value
[word1,	1]
[word2,	1]
[word3,	1]
[word2,	1]
[word4,	1]
. . . .	
[wordx,	1]

Part 3 - MapReduce (Word Count) [3]



Reducer

key value

[word1, {1, 1, 1, …}]

[word2, {1, 1, 1, …}]

[word3, {1, 1, 1, …}]

. . . .

reduce

reduce

.
.
.

reduce

key value

[word1, count1]

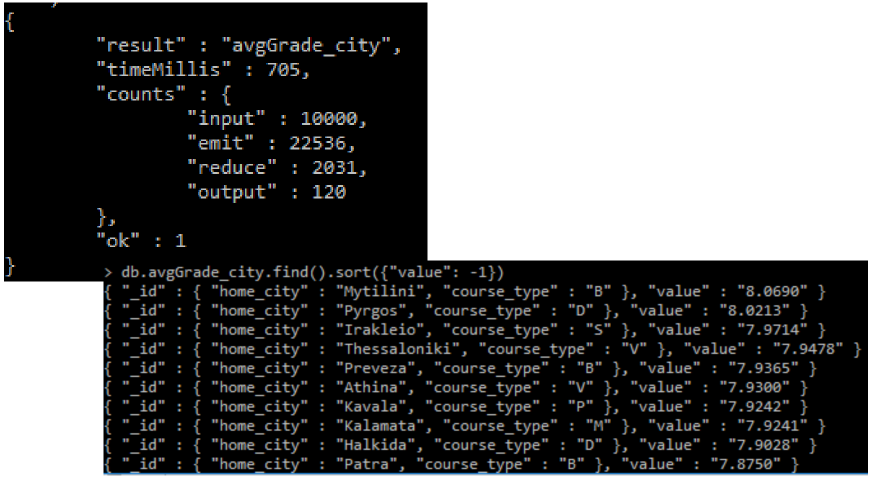
[word2, count2]

[word3, count3]

. . . .

Part 3 - MapReduce (Average grade) [1]

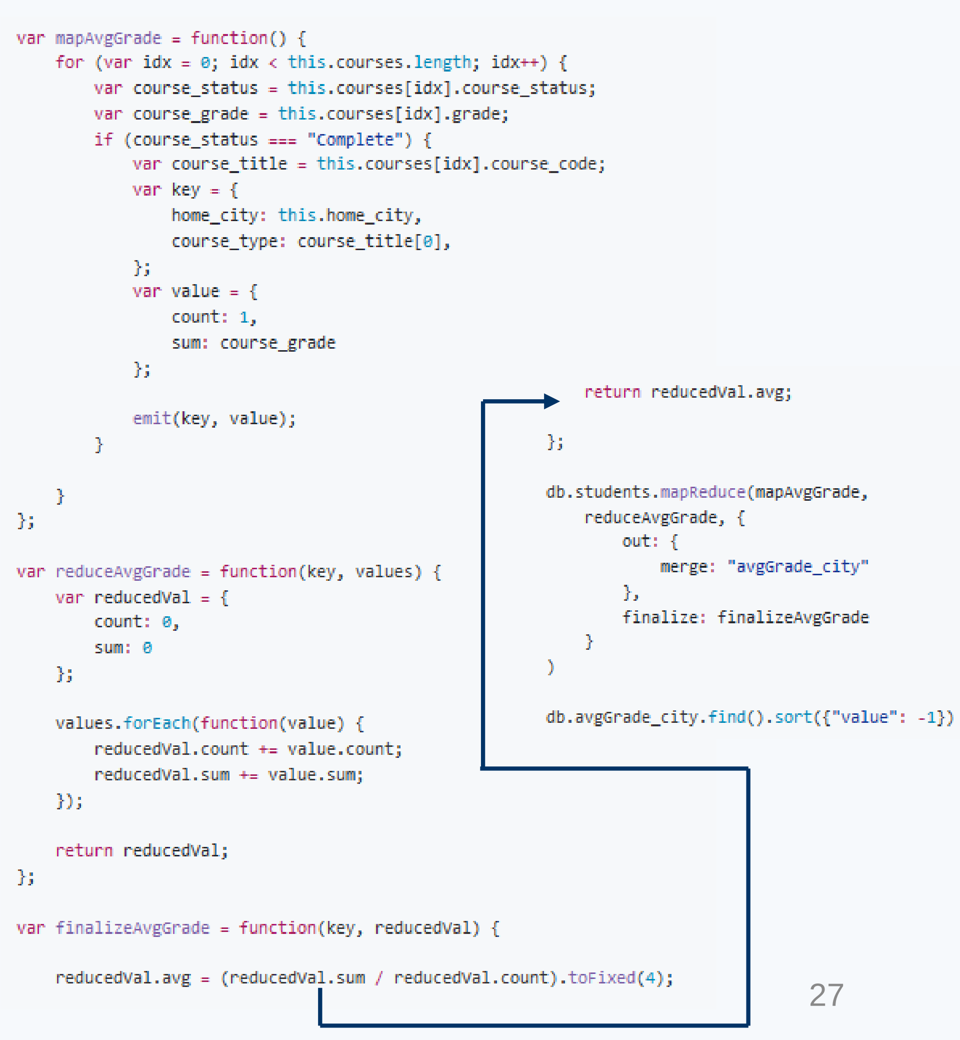
MapReduce 2 : Write a map reduce job on the students collection whose goal is to compute average GPA scores for completed courses by home city and by course type (M, B, P, etc.).



```
{
        "result" : "avgGrade_city",
        "timeMillis" : 705,
        "counts" : {
                "input" : 10000,
                "emit" : 22536,
                "reduce" : 2031,
                "output" : 120
        },
        "ok" : 1
}
```

```
> db.avgGrade_city.find().sort({"value": -1})
{ "_id" : { "home_city" : "Mytilini", "course_type" : "B" }, "value" : "8.0690" }
{ "_id" : { "home_city" : "Pyrgos", "course_type" : "D" }, "value" : "8.0213" }
{ "_id" : { "home_city" : "Irakleio", "course_type" : "S" }, "value" : "7.9714" }
{ "_id" : { "home_city" : "Thessaloniki", "course_type" : "V" }, "value" : "7.9478" }
{ "_id" : { "home_city" : "Preveza", "course_type" : "B" }, "value" : "7.9365" }
{ "_id" : { "home_city" : "Athina", "course_type" : "V" }, "value" : "7.9300" }
{ "_id" : { "home_city" : "Kavala", "course_type" : "P" }, "value" : "7.9242" }
{ "_id" : { "home_city" : "Kalamata", "course_type" : "M" }, "value" : "7.9241" }
{ "_id" : { "home_city" : "Halkida", "course_type" : "D" }, "value" : "7.9028" }
{ "_id" : { "home_city" : "Patra", "course_type" : "B" }, "value" : "7.8750" }
```

References: [4,5]



```
var mapAvgGrade = function() {
    for (var idx = 0; idx < this.courses.length; idx++) {
        var course_status = this.courses[idx].course_status;
        var course_grade = this.courses[idx].grade;
        if (course_status === "Complete") {
            var course_title = this.courses[idx].course_code;
            var key = {
                home_city: this.home_city,
                course_type: course_title[0],
            };
            var value = {
                count: 1,
                sum: course_grade
            };

            emit(key, value);
        }

    }
};

var reduceAvgGrade = function(key, values) {
    var reducedVal = {
        count: 0,
        sum: 0
    };

    values.forEach(function(value) {
        reducedVal.count += value.count;
        reducedVal.sum += value.sum;
    });

    return reducedVal;
};

var finalizeAvgGrade = function(key, reducedVal) {

    reducedVal.avg = (reducedVal.sum / reducedVal.count).toFixed(4);

    return reducedVal.avg;

};

db.students.mapReduce(mapAvgGrade,
    reduceAvgGrade, {
        out: {
            merge: "avgGrade_city"
        },
        finalize: finalizeAvgGrade
    }
)

db.avgGrade_city.find().sort({"value": -1})
```

Part 3 - MapReduce (Average grade) [2]

(course_code, home_city), grade

[(S201, Athina), 10]

[(M101, Mytilini), 9]

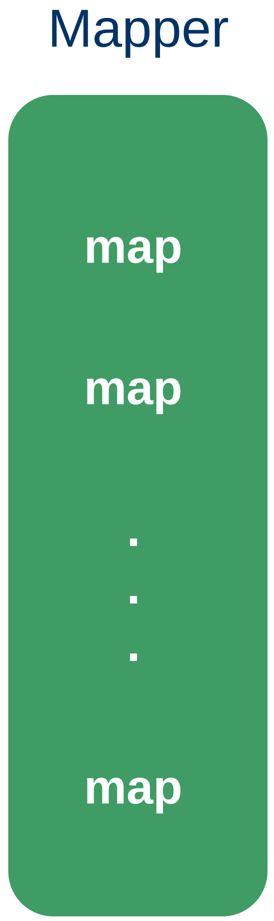
[(S202, Kavala), 3]

[(D102, Chania), 5]

[(P103, Athina), 6]

[(P101, Arta), 10]

. . . .



key value

[{home_city: Athina, course_type: M}, {count: 1, sum: 8}]

[{home_city: Chania, course_type: P}, {count: 1, sum: 6}]

[{home_city: Thyra, course_type: V}, {count: 1, sum: 3}]

[{home_city: Arta, course_type: M}, {count: 1, sum: 10}]

. . . .

Part 3 - MapReduce (Average grade) [3]

key **value**

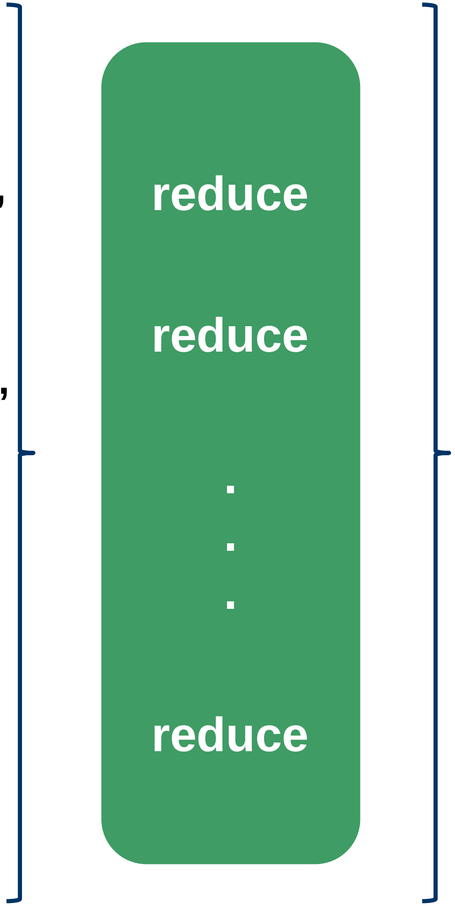
[{home_city: Athina, course_type: M}, [{count: 1, sum: 8}, [{count: 1, sum: 3}, [{count: 1, sum: 7}]

[{home_city: Chania, course_type: P}, {count: 1, sum: 6} , [{count: 1, sum: 5}, [{count: 1, sum: 4}]]

[{home_city: Thyra, course_type: V}, {count: 1, sum: 3} , [{count: 1, sum: 7}, [{count: 1, sum: 10}]]

. . . . .

Reducer



key **value**

[{home_city: Athina, course_type: M}, {count: 22, sum: 80}]

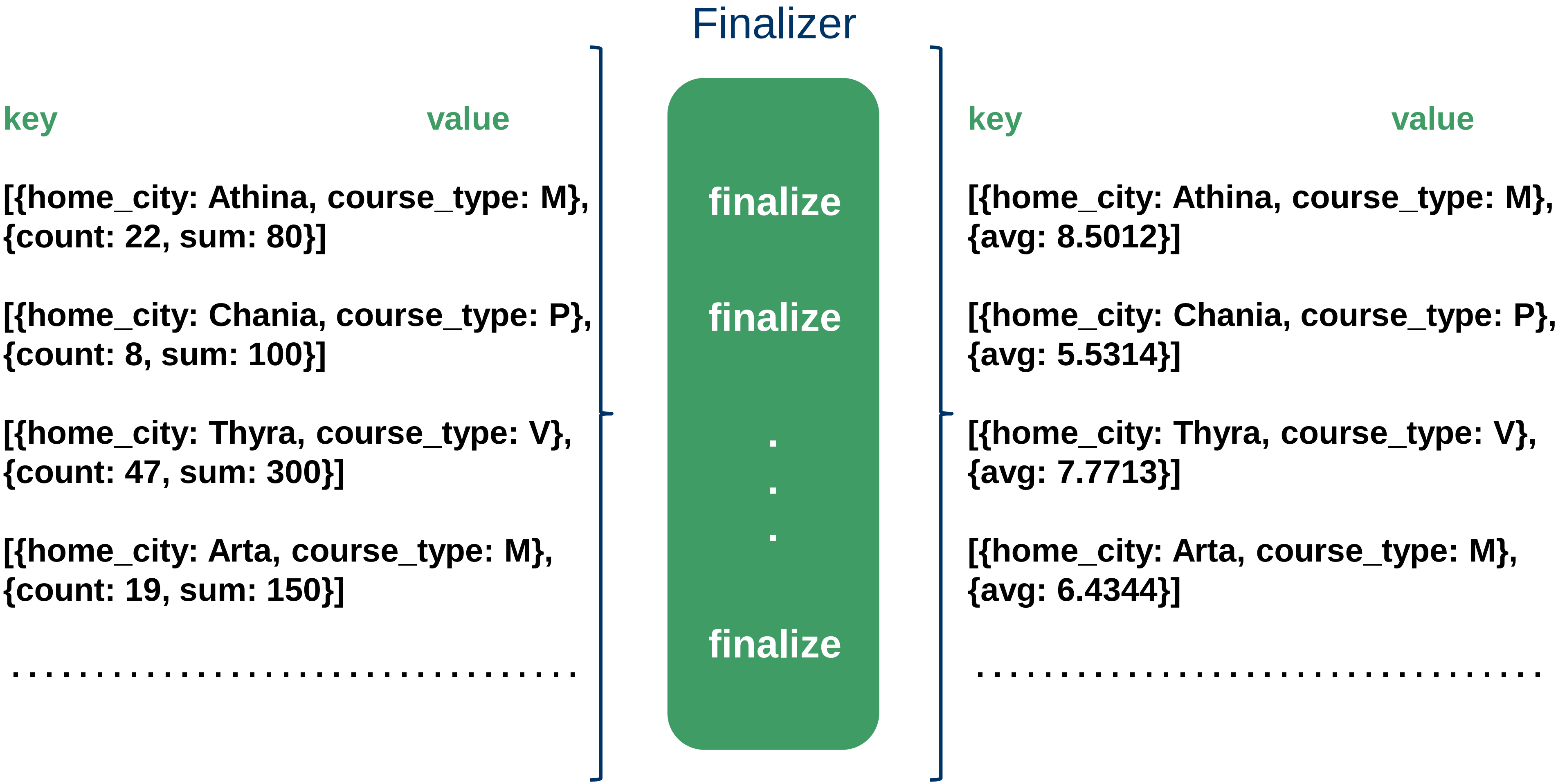
[{home_city: Chania, course_type: P}, {count: 8, sum: 100}]

[{home_city: Thyra, course_type: V}, {count: 47, sum: 300}]

[{home_city: Arta, course_type: M}, {count: 19, sum: 150}]

. . . . .

Part 3 - MapReduce (Average grade) [4]



Finalizer

key value

[{home_city: Athina, course_type: M},
{count: 22, sum: 80}]

[{home_city: Chania, course_type: P},
{count: 8, sum: 100}]

[{home_city: Thyra, course_type: V},
{count: 47, sum: 300}]

[{home_city: Arta, course_type: M},
{count: 19, sum: 150}]

. . . .

key value

[{home_city: Athina, course_type: M},
{avg: 8.5012}]

[{home_city: Chania, course_type: P},
{avg: 5.5314}]

[{home_city: Thyra, course_type: V},
{avg: 7.7713}]

[{home_city: Arta, course_type: M},
{avg: 6.4344}]

. . . .

References

[1] Db-engines.com. (n.d.). *System Properties Comparison Microsoft SQL Server vs. MongoDB vs. Oracle NoSQL* [online] Available at: https://db-engines.com/en/system/Microsoft+SQL+Server%3BMongoDB%3BOracle+NoSQL [Accessed 5 May 2017].

[2] Install MongoDB Community Edition on Windows — MongoDB Manual 3.4. https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/t [Accessed 2 May 2017].

[3] Aggregation Pipeline — MongoDB Manual 3.4 https://docs.mongodb.com/manual/core/aggregation-pipeline/ [Accessed 2 May 2017].

[4] Map-Reduce Examples — MongoDB Manual 3.4 https://docs.mongodb.com/manual/tutorial/map-reduce-examples/ [Accessed 2 May 2017].

[5] Map-Reduce — MongoDB Manual 3.4 https://docs.mongodb.com/manual/core/map-reduce/ [Accessed 2 May 2017].