* Save a list, called db\_names, of the names of the databases managed by our connected client.
* Similarly, save a list, called nobel\_coll\_names, of the names of the collections managed by the "nobel" database.

# Save a list of names of the databases managed by client

db\_names = client.list\_database\_names()

print(db\_names)

# Save a list of names of the collections managed by the "nobel" database

nobel\_coll\_names = client.nobel.list\_collection\_names()

print(nobel\_coll\_names)

* Connect to the *nobel* database.

Fetch **one** document from each of the prizes and laureates collections, and then take a look at the output in the console to see the format and type of the documents in Python.

Since prize and laureate are dictionaries, you can use the .keys() method to return the keys (i.e. the field names). But it's often more convenient to work with *lists* of fields.

* Use the list() constructor to save a ***list*** of the fields present in the prize and laureate documents.

# Connect to the "nobel" database

db = client.nobel

# Retrieve sample prize and laureate documents

prize = db.prizes.find\_one()

laureate = db.laureates.find\_one()

# Print the sample prize and laureate documents

print(prize)

print(laureate)

print(type(laureate))

# Get the fields present in each type of document

prize\_fields = list(prize.keys())

laureate\_fields = list(laureate.keys())

print(prize\_fields)

print(laureate\_fields)

Create a filter criteria to count laureates who died ("diedCountry") in the USA ("USA"). Save the document count as count.

# Create a filter for laureates who died in the USA

criteria = {"diedCountry": "USA"}

# Save the count of these laureates

count = db.laureates.count\_documents(criteria)

print(count)

* Create a filter to count laureates who died in the United States but were born ("bornCountry") in Germany.

# Create a filter for laureates who died in the USA but were born in Germany

criteria = {"bornCountry": "Germany",

"diedCountry": "USA"}

# Save the count

count = db.laureates.count\_documents(criteria)

print(count)

Count laureates who died in the USA, were born in Germany, and whose first name ("firstname") was "Albert". # Create a filter for Germany-born laureates who died in the USA and with the first name "Albert"

criteria = {"firstname": "Albert",

"bornCountry": "Germany",

"diedCountry": "USA"}

# Save the count

count = db.laureates.count\_documents(criteria)

print(count)

* How many laureates were born in "USA", "Canada", or "Mexico"? Save a filter as criteria and your count as count.

# Save a filter for laureates born in the USA, Canada, or Mexico

criteria = { "bornCountry":

{ "$in": ["USA", "Canada", "Mexico"]}

}

# Count them and save the count

count = db.laureates.count\_documents(criteria)

print(count)

* How many laureates died in the USA but were not born there? Save your filter as criteria and your count as count.

# Save a filter for laureates who died in the USA and were not born there

criteria = { "diedCountry": 'USA',

"bornCountry": { "$ne": 'USA'},

}

# Count them

count = db.laureates.count\_documents(criteria)

print(count)

* Save a filter criteria for laureates born in (bornCountry) "Austria" with a non-Austria prizes.affiliations.country.
* Save your count of laureates as count.

# Filter for laureates born in Austria with non-Austria prize affiliation

criteria = {"bornCountry": "Austria",

"prizes.affiliations.country": {"$ne": "Austria"}}

# Count the number of such laureates

count = db.laureates.count\_documents(criteria)

print(count)

* Use a filter document (criteria) to count the documents that don't have a "born" field.

# Filter for documents without a "born" field

criteria = {"born": {"$exists": False}}

# Save count

count = db.laureates.count\_documents(criteria)

print(count)

Use a filter document (criteria) to **find** a document for a laureate with at least three elements in its "prizes" array. In other words, does a third element exist for the array? Remember about the zero-based indexing! # Filter for laureates with at least three prizes

criteria = {"prizes.2": {"$exists": True}}

# Find one laureate with at least three prizes

doc = db.laureates.find\_one(criteria)

# Print the document

print(doc)

There are some recorded countries of death ("diedCountry") that do not appear as a country of birth ("bornCountry") for laureates. One such country is "East Germany".

* Return a set of all such countries as countries.

# Countries recorded as countries of death but not as countries of birth

countries = set(db.laureates.distinct("diedCountry")) - set(db.laureates.distinct("bornCountry"))

print(countries)

* Determine the number of distinct countries recorded as part of an affiliation for laureates' prizes. Save this as count

# The number of distinct countries of laureate affiliation for prizes

count = len(db.laureates.distinct('prizes.affiliations.country' ))

print(count)

In which countries have USA-born laureates had affiliations for their prizes?

db.laureates.distinct('prizes.affiliations.country',{"bornCountry": "USA"} )

* Australia, Denmark, United Kingdom, USA
* Save a filter document criteria that, when passed to db.prizes.distinct, returns all prize categories shared by three or more laureates. That is, "laureates.2" must exist for such documents.
* Save these prize categories as a Python set called triple\_play\_categories.
* Confirm via an assertion that "literature" is the only prize category with no prizes shared by three or more laureates.

# Save a filter for prize documents with three or more laureates

criteria = {"laureates.2": {"$exists": True}}

# Save the set of distinct prize categories in documents satisfying the criteria

triple\_play\_categories = set(db.prizes.distinct("category", criteria))

# Confirm literature as the only category not satisfying the criteria.

assert set(db.prizes.distinct("category")) - triple\_play\_categories == {"literature"}

* Save an $elemMatch filter unshared to count laureates with unshared prizes in categories other than ("not in") ["physics", "chemistry", "medicine"] in or after 1945.
* Save an $elemMatch filter shared to count laureates with shared (i.e., "share" is not "1") prizes in categories other than ["physics", "chemistry", "medicine"] in or after 1945.

# Save a filter for laureates with unshared prizes

unshared = {

"prizes": {"$elemMatch": {

"category": {"$nin": ["physics", "chemistry", "medicine"]},

"share": "1",

"year": {"$gte": "1945"},

}}}

# Save a filter for laureates with shared prizes

shared = {

"prizes": {"$elemMatch": {

"category": {"$nin": ["physics", "chemistry", "medicine"]},

"share": {"$ne": "1"},

"year": {"$gte": "1945"},

}}}

ratio = db.laureates.count\_documents(unshared) / db.laureates.count\_documents(shared)

print(ratio)

* You won't need the $elemMatch operator at all for this exercise.
* Save a filter before to count organization laureates with prizes won before 1945. Recall that organization status is encoded with the "gender" field, and that dot notation is needed to access a laureate's "year" field within its "prizes" array.
* Save a filter in\_or\_after to count organization laureates with prizes won in or after 1945

# Save a filter for organization laureates with prizes won before 1945

before = {

"gender": "org",

"prizes.year": {"$lt": "1945"},

}

# Save a filter for organization laureates with prizes won in or after 1945

in\_or\_after = {

"gender": "org",

"prizes.year": {"$gte": "1945"},

}

n\_before = db.laureates.count\_documents(before)

n\_in\_or\_after = db.laureates.count\_documents(in\_or\_after)

ratio = n\_in\_or\_after / (n\_in\_or\_after + n\_before)

print(ratio)

Evaluate the expression

db.laureates.count\_documents({"firstname": Regex(\_\_\_\_), "surname": Regex(\_\_\_\_)})

in the console, filling in the blanks appropriately.

db.laureates.count\_documents({"firstname": Regex(“^G”), "surname": Regex(“^S”)})

Use a regular expression object to filter for laureates with "Germany" in their "bornCountry" value.

from bson.regex import Regex

# Filter for laureates with "Germany" in their "bornCountry" value

criteria = {"bornCountry": Regex("Germany",0)}

print(set(db.laureates.distinct("bornCountry", criteria)))

* Use a regular expression object to filter for laureates with a "bornCountry" value starting with "Germany".

from bson.regex import Regex

# Filter for laureates with a "bornCountry" value starting with "Germany"

criteria = {"bornCountry": Regex("^Germany",0)}

print(set(db.laureates.distinct("bornCountry", criteria)))

* Use a regular expression object to filter for laureates born in what was at the time Germany but is now another country.

from bson.regex import Regex

# Fill in a string value to be sandwiched between the strings "^Germany " and "now"

criteria = {"bornCountry": Regex("^Germany " + "\(" + "now")}

print(set(db.laureates.distinct("bornCountry", criteria)))

* Use a regular expression object to filter for laureates born in what is now Germany but at the time was another country.

from bson.regex import Regex

# Fill in a string value to be sandwiched between the strings "now" and "$"

criteria = {"bornCountry": Regex("now" + " Germany\)" + "$")}

print(set(db.laureates.distinct("bornCountry", criteria)))

* Save a filter criteria that finds laureates with prizes.motivation values containing "transistor" as a substring. The substring can appear anywhere within the value, so no anchoring characters are needed.
* Save to first and last the field names corresponding to a laureate's first name and last name (i.e. "surname") so that we can print out the names of these laureates.

from bson.regex import Regex

# Save a filter for laureates with prize motivation values containing "transistor" as a substring

criteria = {"prizes.motivation": Regex("transistor")}

# Save the field names corresponding to a laureate's first name and last name

first, last = "firstname", "surname"

print([(laureate[first], laureate[last]) for laureate in db.laureates.find(criteria)])

* First, use regular expressions to fetch the documents for the laureates whose "firstname" starts with "G" and whose "surname" starts with "S".

In the previous step, we fetched all the data for all the laureates with initials G.S. This is unnecessary if we only want their full names!

* Use projection and adjust the query to select only the "firstname" and "surname" fields

Now the documents you fetched contain only the relevant information!

* Iterate over the documents, and for each document, concatenate the first name and the surname fields together with a space in between to obtain full names.

# Use projection to select only firstname and surname

docs = db.laureates.find(

filter= {"firstname" : {"$regex" : "^G"},

"surname" : {"$regex" : "^S"} },

projection= ["firstname", "surname"] )

# Iterate over docs and concatenate first name and surname

full\_names = [doc["firstname"] + " " + doc["surname"] for doc in docs]

# Print the full names

print(full\_names)

* Save a list of prizes (prizes), projecting out only the "laureates.share" values for each prize.
* For each prize, compute the total share as follows:
  + Initialize the variable total\_share to 0.
  + Iterate over the laureates for each prize, converting the "share" field of the "laureate" to float and adding the reciprocal of it (that is, 1 divided by it) to total\_share.

# Save documents, projecting out laureates share

prizes = db.prizes.find({}, ["laureates.share"])

# Iterate over prizes

for prize in prizes:

# Initialize total share

total\_share = 0

# Iterate over laureates for the prize

for laureate in prize["laureates"]:

# add the share of the laureate to total\_share

total\_share += 1 / float(laureate["share"])

# Print the total share

print(total\_share)

Complete the definition of all\_laureates(prize). Within the body of the function:

* Sort the "laureates" list of the prize document according to the "surname" key.
* For each of the laureates in the sorted list, extract the "surname" field.
* The code for joining the last names into a single string is already written for you.

Take a look at the console to make sure the output looks like what you'd expect!

Find the documents for the prizes in the physics category, sort them in chronological order (by "year", ascending), and only fetch the "year", "laureates.firstname", and "laureates.surname" fields.

Now that you have the prizes, and the function to extract laureates from a prize, print the year and the names of the laureates (use your all\_laureates() function) for each prize document.

from operator import itemgetter

def all\_laureates(prize):

# sort the laureates by surname

sorted\_laureates = sorted(prize["laureates"], key=itemgetter("surname"))

# extract surnames

surnames = [laureate["surname"] for laureate in sorted\_laureates]

# concatenate surnames separated with " and "

all\_names = " and ".join(surnames)

return all\_names

# find physics prizes, project year and name, and sort by year

docs = db.prizes.find(

filter= {"category": "physics"},

projection= ["year", "laureates.firstname", "laureates.surname"],

sort= [("year", 1)])

# print the year and laureate names (from all\_laureates)

for doc in docs:

print("{year}: {names}".format(year=doc["year"], names=all\_laureates(doc)))

* Find the original prize categories established in 1901 by looking at the distinct values of the "category" field for prizes from year 1901.
* Fetch ONLY the year and category from all the documents *(without the* "\_id" *field)*.
* Sort by "year" in **descending** order, then by "category" in **ascending** order.

# original categories from 1901

original\_categories = db.prizes.distinct("category", {"year": "1901"})

print(original\_categories)

# project year and category, and sort

docs = db.prizes.find(

filter={},

projection={"year":1, "category":1, "\_id":0},

sort=[("year", -1), ("category", 1)]

)

#print the documents

for doc in docs:

print(doc)

* Specify an index model that indexes first on category (ascending) and second on year (descending).
* Save a string report for printing the last single-laureate year for each distinct category, one category per line. To do this, for each distinct prize category, find the latest-year prize (requiring a descending sort by year) of that category (so, find matches for that category) with a laureate share of "1".

# Specify an index model for compound sorting

index\_model = [("category", 1), ("year", -1)]

db.prizes.create\_index(index\_model)

# Collect the last single-laureate year for each category

report = ""

for category in sorted(db.prizes.distinct("category")):

doc = db.prizes.find\_one(

{"category": category, "laureates.share": "1"},

sort=[("year", -1)]

)

report += "{category}: {year}\n".format(\*\*doc)

print(report)

* Create an index on country of birth ("bornCountry") for db.laureates to ensure efficient gathering of distinct values and counting of documents
* Complete the skeleton dictionary comprehension to construct n\_born\_and\_affiliated, the count of laureates as described above for each distinct country of birth. For each call to count\_documents, ensure that you use the value of country to filter documents properly.

from collections import Counter

# Ensure an index on country of birth

db.laureates.create\_index([("bornCountry", 1)])

# Collect a count of laureates for each country of birth

n\_born\_and\_affiliated = {

country: db.laureates.count\_documents({

"bornCountry": country,

"prizes.affiliations.country": country

})

for country in db.laureates.distinct("bornCountry")

}

five\_most\_common = Counter(n\_born\_and\_affiliated).most\_common(5)

print(five\_most\_common)

* Save to filter\_ the filter document to fetch only prizes with one or more quarter-share laureates, i.e. with a "laureates.share" of "4".
* Save to projection the list of field names so that prize category, year and laureates' motivations ("laureates.motivation") may be fetched for inspection.
* Save to cursor a cursor that will yield prizes, sorted by ascending year. Limit this to five prizes, and sort using the most concise specification.

from pprint import pprint

# Fetch prizes with quarter-share laureate(s)

filter\_ = {"laureates.share": "4"}

# Save the list of field names

projection = ["category", "year", "laureates.motivation"]

# Save a cursor to yield the first five prizes

cursor = db.prizes.find(filter\_, projection).sort("year").limit(5)

pprint(list(cursor))

* Complete the function get\_particle\_laureates that, given page\_number and page\_size, retrieves a given page of prize data on laureates who have the word "particle" (use $regex) in their prize motivations ("prizes.motivation"). Sort laureates first by ascending "prizes.year" and next by ascending "surname".
* Collect and save the first nine pages of laureate data to pages.

from pprint import pprint

# Write a function to retrieve a page of data

def get\_particle\_laureates(page\_number=1, page\_size=3):

if page\_number < 1 or not isinstance(page\_number, int):

raise ValueError("Pages are natural numbers (starting from 1).")

particle\_laureates = list(

db.laureates.find(

{"prizes.motivation": {"$regex": "particle"}},

["firstname", "surname", "prizes"])

.sort([("prizes.year", 1), ("surname", 1)])

.skip(page\_size \* (page\_number - 1))

.limit(page\_size))

return particle\_laureates

# Collect and save the first nine pages

pages = [get\_particle\_laureates(page\_number=page) for page in range(1,9)]

pprint(pages[0])

Translate the above cursor cursor to an equivalent aggregation cursor, saving the pipeline stages to pipeline. Recall that the find collection method's "filter" parameter maps to the "$match" aggregation stage, its "projection" parameter maps to the "$project" stage, and the "limit" parameter (or cursor method) maps to the "$limit" stage.

# Translate cursor to aggregation pipeline

pipeline = [

{"$match": {"gender": {"$ne": "org"}}},

{"$project": {"bornCountry": 1, "prizes.affiliations.country": 1}},

{"$limit": 3}

]

for doc in db.laureates.aggregate(pipeline):

print("{bornCountry}: {prizes}".format(\*\*doc))

Save to pipeline an aggregation pipeline to collect prize documents as detailed above. Use Python's collections.OrderedDict to specify any sorting.

from collections import OrderedDict

from itertools import groupby

from operator import itemgetter

original\_categories = set(db.prizes.distinct("category", {"year": "1901"}))

# Save an pipeline to collect original-category prizes

pipeline = [

{"$match": {"category": {"$in": list(original\_categories)}}},

{"$project": {"category": 1, "year": 1}},

{"$sort": OrderedDict([("year", -1)])}

]

cursor = db.prizes.aggregate(pipeline)

for key, group in groupby(cursor, key=itemgetter("year")):

missing = original\_categories - {doc["category"] for doc in group}

if missing:

print("{year}: {missing}".format(year=key, missing=", ".join(sorted(missing))))

* Fill out pipeline to determine the number of prizes awarded (at least partly) to organizations. To do this, you'll first need to $match on the "gender" that designates organizations.
* Then, use a field path to project the number of prizes for each organization as the "$size" of the "prizes" array. Recall that to specify the value of a field "<my\_field>", you use the field path "$<my\_field>".
* Finally, use a single group {"\_id": None} to sum over the *values* of all organizations' prize counts.

# Count prizes awarded (at least partly) to organizations as a sum over sizes of "prizes" arrays.

pipeline = [

{"$match": {"gender": "org"}},

{"$project": {"n\_prizes": {"$size": "$prizes"}}},

{"$group": {"\_id": None, "n\_prizes\_total": {"$sum": "$n\_prizes"}}}

]

print(list(db.laureates.aggregate(pipeline)))

* Make the $group stage output a document for each prize year (set "\_id" to the *field path* for year) with the set of categories awarded that year.
* Given your intermediate collection of year-keyed documents, $project a field named "missing" with the (original) categories *not* awarded that year. Again, mind your *field path*s!
* Use a $match stage to only pass through documents with at least one missing prize category.
* Finally, add sort documents in descending order.

from collections import OrderedDict

original\_categories = sorted(set(db.prizes.distinct("category", {"year": "1901"})))

pipeline = [

{"$match": {"category": {"$in": original\_categories}}},

{"$project": {"category": 1, "year": 1}},

# Collect the set of category values for each prize year.

{"$group": {"\_id": "$year", "categories": {"$addToSet": "$category"}}},

# Project categories \*not\* awarded (i.e., that are missing this year).

{"$project": {"missing": {"$setDifference": [original\_categories, "$categories"]}}},

# Only include years with at least one missing category

{"$match": {"missing.0": {"$exists": True}}},

# Sort in reverse chronological order. Note that "\_id" is a distinct year at this stage.

{"$sort": OrderedDict([("\_id", -1)])},

]

for doc in db.prizes.aggregate(pipeline):

print("{year}: {missing}".format(year=doc["\_id"],missing=", ".join(sorted(doc["missing"]))))

* Use $unwind stages to ensure a single prize affiliation country per pipeline document.
* Filter out prize-affiliation-country values that are "empty" (null, not present, etc.) -- ensure values are "$in" the list of known values.
* Produce a count of documents for each value of "affilCountrySameAsBorn" (a field we've projected for you using the $indexOfBytes operator) by adding 1 to the running sum.

key\_ac = "prizes.affiliations.country"

key\_bc = "bornCountry"

pipeline = [

{"$project": {key\_bc: 1, key\_ac: 1}},

# Ensure a single prize affiliation country per pipeline document

{"$unwind": "$prizes"},

{"$unwind": "$prizes.affiliations"},

# Ensure values in the list of distinct values (so not empty)

{"$match": {key\_ac: {"$in": db.laureates.distinct(key\_ac)}}},

{"$project": {"affilCountrySameAsBorn": {

"$gte": [{"$indexOfBytes": ["$"+key\_ac, "$"+key\_bc]}, 0]}}},

# Count by "$affilCountrySameAsBorn" value (True or False)

{"$group": {"\_id": "$affilCountrySameAsBorn",

"count": {"$sum": 1}}},

]

for doc in db.laureates.aggregate(pipeline): print(doc)

* $unwind the laureates array field to output one pipeline document for each array element.
* After pulling in laureate bios with a $lookup stage, unwind the new laureate\_bios array field (each laureate has only a single biography document).
* Collect the set of bornCountries associated with each prize category.
* Project out the size of each category's set of bornCountries.

pipeline = [

# Unwind the laureates array

{"$unwind": "$laureates"},

{"$lookup": {

"from": "laureates", "foreignField": "id",

"localField": "laureates.id", "as": "laureate\_bios"}},

# Unwind the new laureate\_bios array

{"$unwind": "$laureate\_bios"},

{"$project": {"category": 1,

"bornCountry": "$laureate\_bios.bornCountry"}},

# Collect bornCountry values associated with each prize category

{"$group": {"\_id": "$category",

"bornCountries": {"$addToSet": "$bornCountry"}}},

# Project out the size of each category's (set of) bornCountries

{"$project": {"category": 1,

"nBornCountries": {"$size": "$bornCountries"}}},

{"$sort": {"nBornCountries": -1}},

]

for doc in db.prizes.aggregate(pipeline): print(doc)

* In your aggregation pipeline pipeline, use the "gender" field to limit results to people (that is, not organizations).
* Count prizes for which the laureate's "bornCountry" is not also the "country" of any of their affiliations for the prize. Be sure to use field paths (precede a field name with "$") when appropriate.

pipeline = [

# Limit results to people; project needed fields; unwind prizes

{"$match": {"gender": {"$ne": "org"}}},

{"$project": {"bornCountry": 1, "prizes.affiliations.country": 1}},

{"$unwind": "$prizes"},

# Count prizes with no country-of-birth affiliation

{"$addFields": {"bornCountryInAffiliations": {"$in": ["$bornCountry", "$prizes.affiliations.country"]}}},

{"$match": {"bornCountryInAffiliations": False}},

{"$count": "awardedElsewhere"},

]

print(list(db.laureates.aggregate(pipeline)))

* Construct a stage added\_stage that filters for laureate "prizes.affiliations.country" values that are non-empty, that is, are $in a list of the distinct values that the field takes in the collection.
* Insert this stage into the pipeline so that it filters out single prizes (not arrays) and precedes any test for membership in an array of countries. Recall that the first parameter to <list>.insert is the (zero-based) index for insertion.

pipeline = [

{"$match": {"gender": {"$ne": "org"}}},

{"$project": {"bornCountry": 1, "prizes.affiliations.country": 1}},

{"$unwind": "$prizes"},

{"$addFields": {"bornCountryInAffiliations": {"$in": ["$bornCountry", "$prizes.affiliations.country"]}}},

{"$match": {"bornCountryInAffiliations": False}},

{"$count": "awardedElsewhere"},

]

# Construct the additional filter stage

added\_stage = {"$match": {"prizes.affiliations.country": {"$in": db.laureates.distinct("prizes.affiliations.country")}}}

# Insert this stage into the pipeline

pipeline.insert(3, added\_stage)

print(list(db.laureates.aggregate(pipeline)))