

Projection: Getting only what you need

INTRODUCTION TO MONGODB IN PYTHON

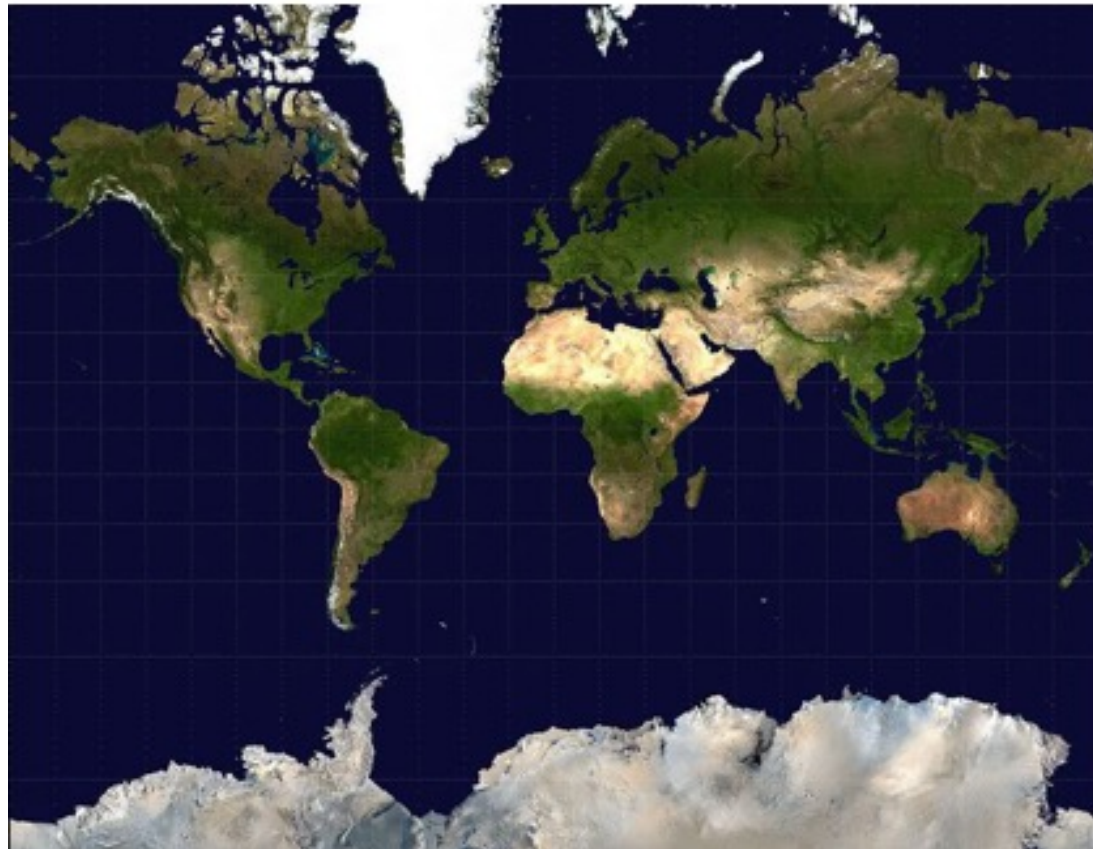


Donny Winston
Instructor

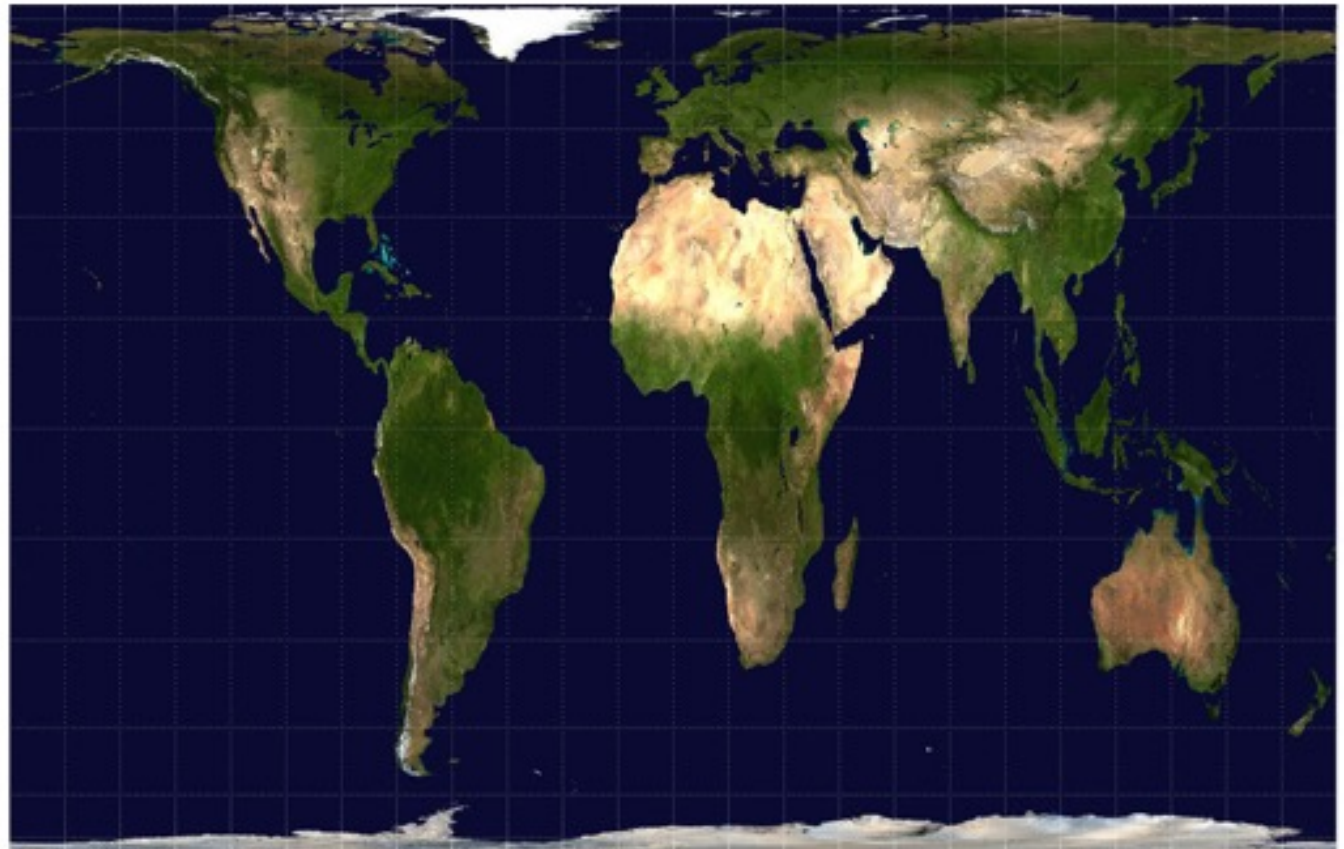
What is "projection"?

- reducing data to fewer dimensions
- asking certain data to "speak up"!

Mercator projection



Gall-Peters projection



Projection in MongoDB

```
# include only prizes.affiliations
# exclude _id
docs = db.laureates.find(
    filter={},
    projection={"prizes.affiliations": 1,
               "_id": 0})

type(docs)
```

```
<pymongo.cursor.Cursor at 0x10d6e69e8>
```

Projection as a dictionary:

- Include fields: `"field_name" : 1`
- `"_id"` is included by default

Projection in MongoDB

```
# include only prizes.affiliations
# exclude _id
docs = db.laureates.find(
    filter={},
    projection={"prizes.affiliations": 1,
               "_id": 0})

type(docs)
```

```
<pymongo.cursor.Cursor at 0x10d6e69e8>
```

```
# convert to list and slice
list(docs)[:3]
```

```
[{'prizes': [{'affiliations': [{'city': 'Munich',
                                'country': 'Germany',
                                'name': 'Munich University'}]}]},
 {'prizes': [{'affiliations': [{'city': 'Leiden',
                                'country': 'the Netherlands',
                                'name': 'Leiden University'}]}]},
 {'prizes': [{'affiliations': [{'city': 'Amsterdam',
                                'country': 'the Netherlands',
                                'name': 'Amsterdam University'}]}]}]
```

Missing fields

```
# use "gender":"org" to select organizations
# organizations have no bornCountry
docs = db.laureates.find(
    filter={"gender": "org"},
    projection=["bornCountry", "firstname"])
list(docs)
```

```
[{'_id': ObjectId('5bc56154f35b634065ba1dff'),
  'firstname': 'United Nations Peacekeeping Forces'},
 {'_id': ObjectId('5bc56154f35b634065ba1df3'),
  'firstname': 'Amnesty International'},
 ...
]
```

Projection as a list

- list the fields to include
["field_name1", "field_name2"]
- "_id" is included by default

Missing fields

```
# use "gender":"org" to select organizations
# organizations have no bornCountry
docs = db.laureates.find(
    filter={"gender": "org"},
    projection=["bornCountry", "firstname"])
list(docs)
```

```
[{'_id': ObjectId('5bc56154f35b634065ba1dff'),
  'firstname': 'United Nations Peacekeeping Forces'},
 {'_id': ObjectId('5bc56154f35b634065ba1df3'),
  'firstname': 'Amnesty International'},
 ...
]
```

- only projected fields that *exist* are returned

```
docs = db.laureates.find({}, ["favoriteIceCreamFlavor"])
list(docs)
```

```
[{'_id': ObjectId('5bc56154f35b634065ba1dff')},
 {'_id': ObjectId('5bc56154f35b634065ba1df3')},
 {'_id': ObjectId('5bc56154f35b634065ba1db1')},
 ...
]
```

Simple aggregation

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

n_prizes = 0
for doc in :
    # count the number of pizes in each doc
    n_prizes += len(doc["prizes"])
print(n_prizes)
```

941

```
# using comprehension
sum([len(doc["prizes"]) for doc in docs])
```

941

Let's project!

INTRODUCTION TO MONGODB IN PYTHON

Sorting

INTRODUCTION TO MONGODB IN PYTHON



Donny Winston

Donny Winston

Sorting post-query with Python

```
docs = list(db.prizes.find({"category": "physics"}, ["year"]))

print([doc["year"] for doc in docs][:5])
```

```
['2018', '2017', '2016', '2015', '2014']
```

```
from operator import itemgetter

docs = sorted(docs, key=itemgetter("year"))
print([doc["year"] for doc in docs][:5])
```

```
['1901', '1902', '1903', '1904', '1905']
```

```
docs = sorted(docs, key=itemgetter("year"), reverse=True)
print([doc["year"] for doc in docs][:5])
```

```
['2018', '2017', '2016', '2015', '2014']
```

```
['2018', '2017', '2016', '2015', '2014']
```

Sorting in-query with MongoDB

```
cursor = db.prizes.find({"category": "physics"}, ["year"],
                        sort=[("year", 1)])
print([doc["year"] for doc in cursor][:5])
```

```
['1901', '1902', '1903', '1904', '1905']
```

```
cursor = db.prizes.find({"category": "physics"}, ["year"],
                        sort=[("year", -1)])
print([doc["year"] for doc in cursor][:5])
```

```
['2018', '2017', '2016', '2015', '2014']
```

```
['2018', '2017', '2016', '2015', '2014']
```

Primary and secondary sorting

```
for doc in db.prizes.find(
    {"year": {"$gt": "1966", "$lt": "1970"}},
    ["category", "year"],
    sort=[("year", 1), ("category", -1)]):
    print("{year} {category}".format(**doc))
```

```
1967 physics
1967 medicine
1967 literature
1967 chemistry
1968 physics
1968 peace
1968 medicine
1968 literature
1968 chemistry
1969 physics
1969 peace
1969 medicine
1969 literature
1969 economics
1969 chemistry
```

Sorting with pymongo versus MongoDB shell

In MongoDB shell:

- Example `sort` argument: `{"year": 1, "category": -1}`
- JavaScript objects retain key order as entered

In Python (< 3.7):

```
{"year": 1, "category": 1}
```

```
{'category': 1, 'year': 1}
```

```
[("year", 1), ("category", 1)]
```

```
[('year', 1), ('category', 1)]
```

Let's get sorted!

INTRODUCTION TO MONGODB IN PYTHON

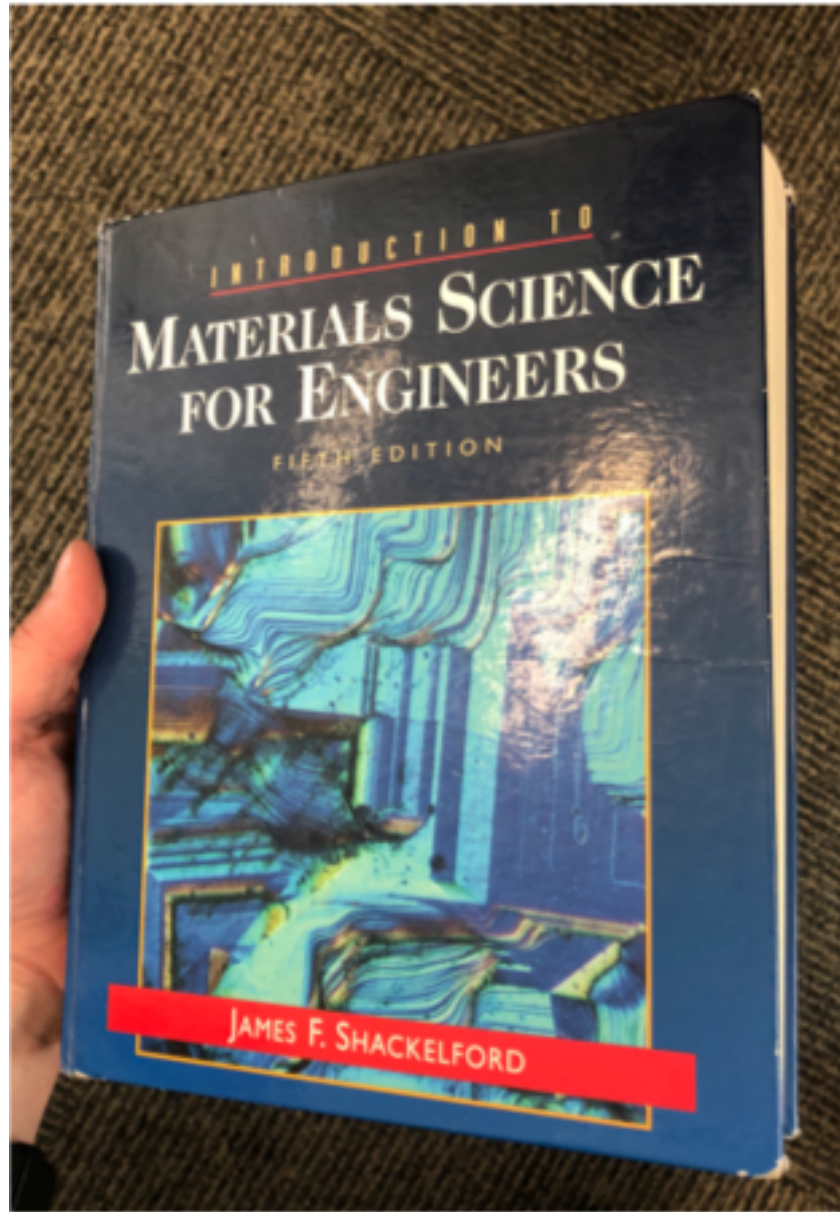
What are indexes?

INTRODUCTION TO MONGODB IN PYTHON

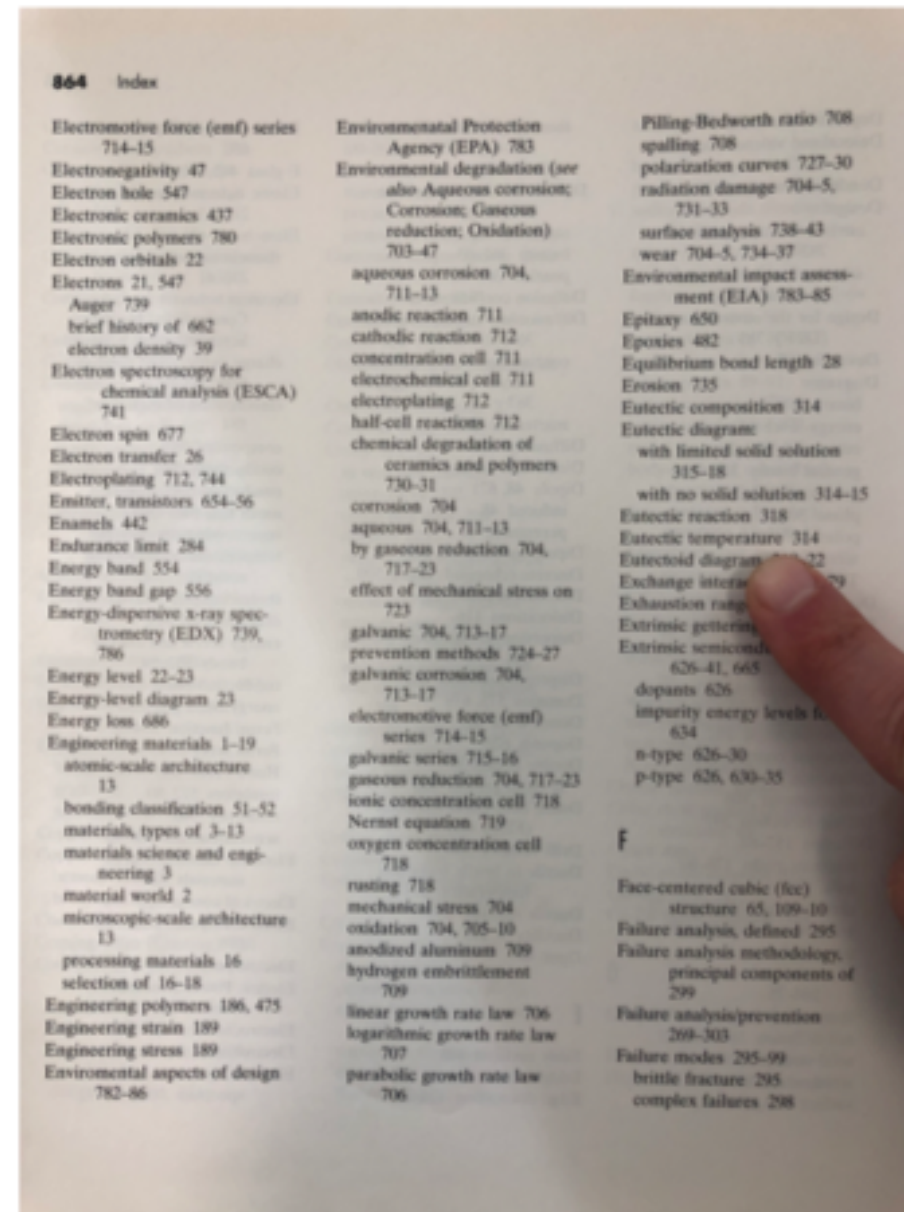
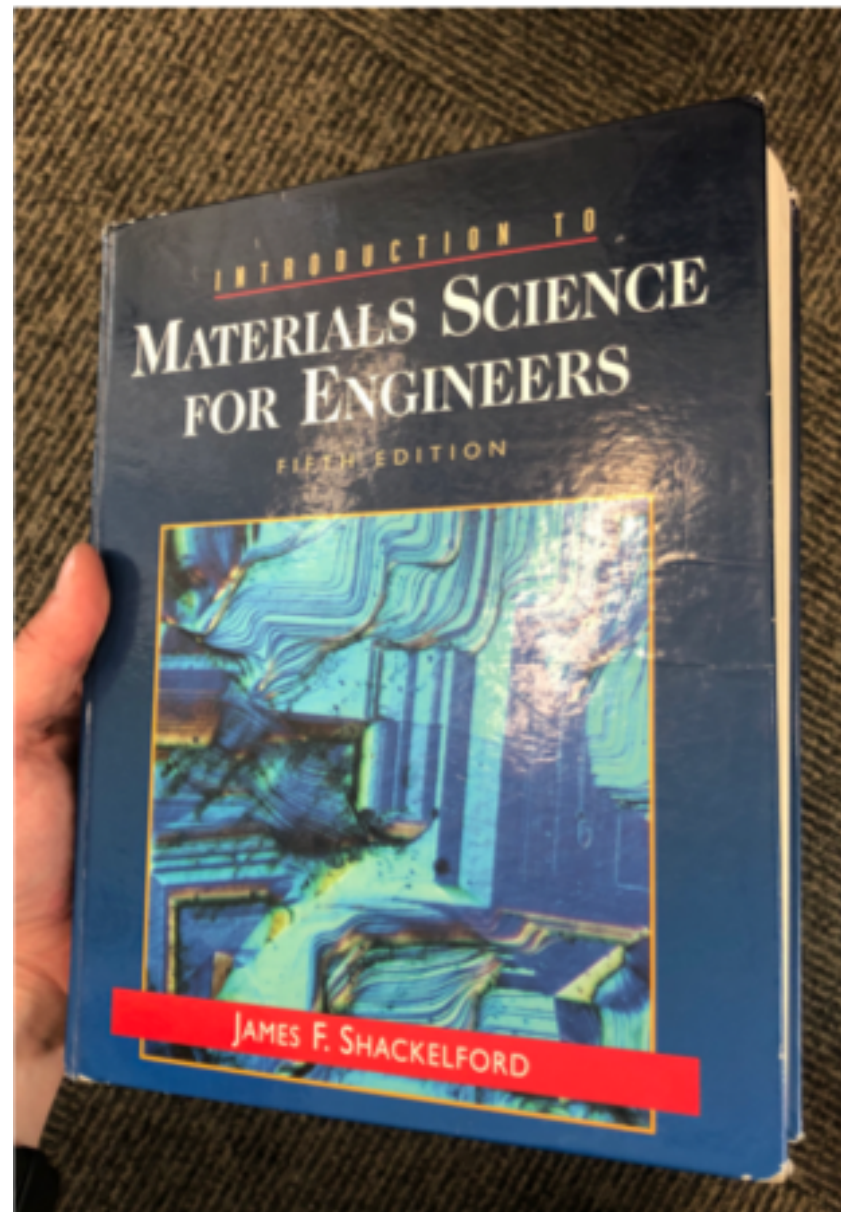


Donny Winston
Instructor

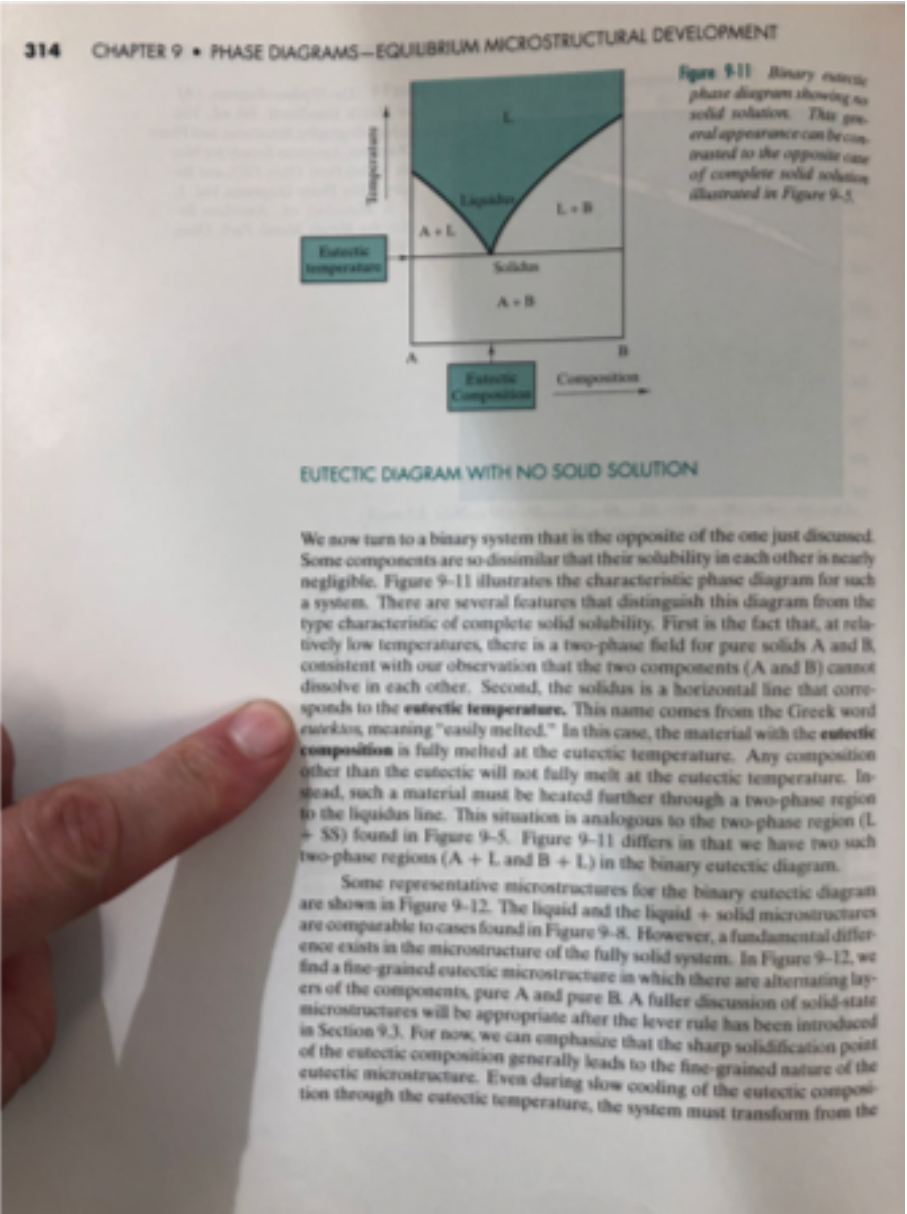
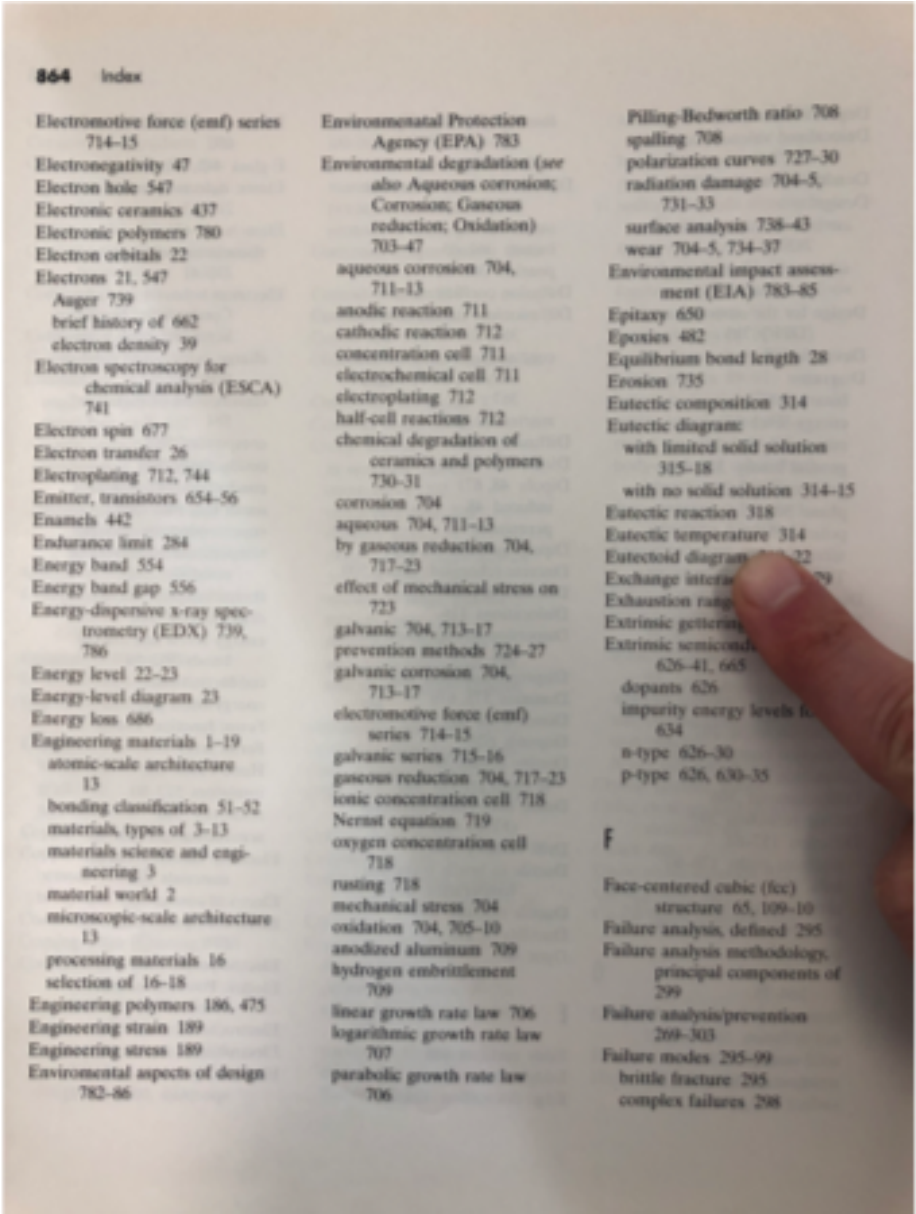
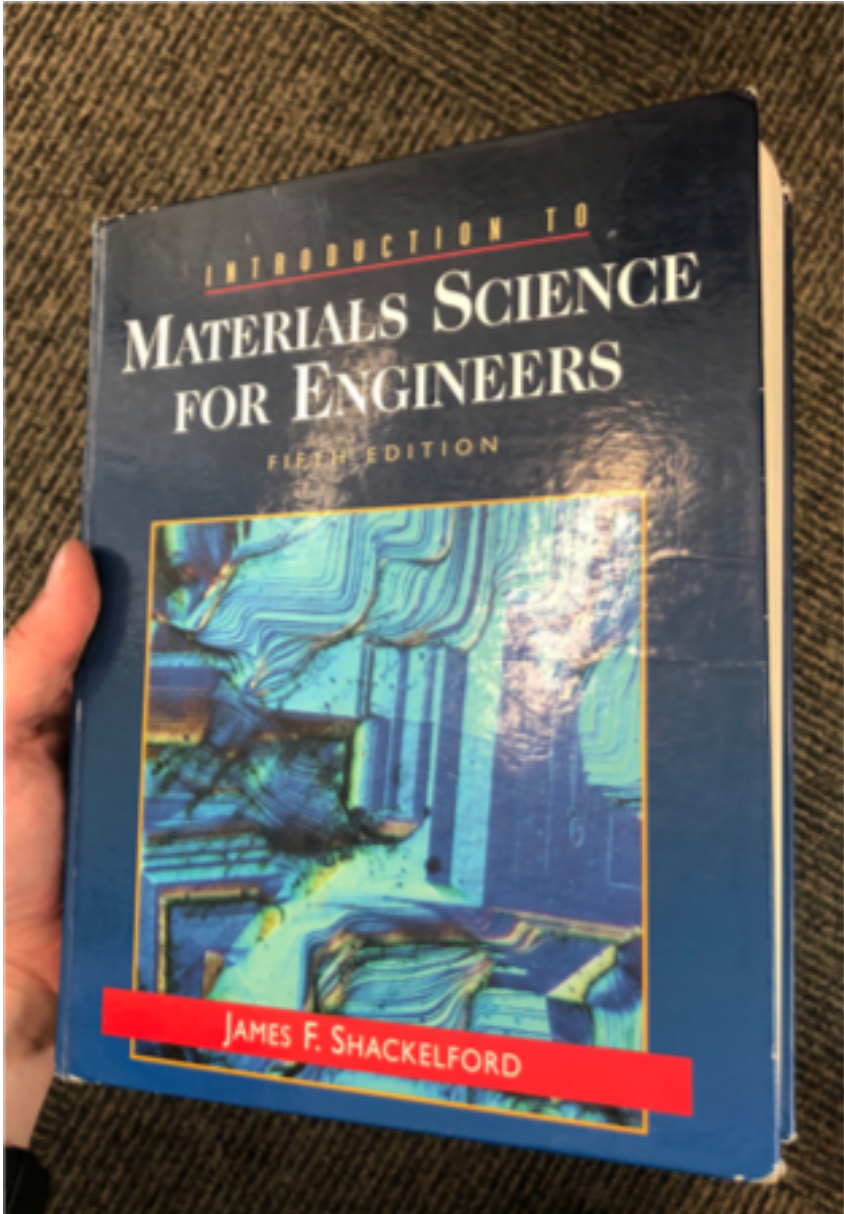
What are indexes?



What are indexes?



What are indexes?



When to use indexes?

- Queries with high specificity
- Large documents
- Large collections

Gauging performance before indexing

Jupyter Notebook `%%timeit` magic (same as `python -m timeit "[expression]"`)

```
%%timeit
docs = list(db.prizes.find({"year": "1901"}))
```

524 μ s \pm 7.34 μ s per loop (mean \pm std. dev. of 7 runs, 1000 loops each)

```
%%timeit
docs = list(db.prizes.find({}, sort=[("year", 1)]))
```

5.18 ms \pm 54.9 μ s per loop (mean \pm std. dev. of 7 runs, 100 loops each)

Adding a single-field index

- index model: list of (field, direction) pairs.
- directions: 1 (ascending) and -1 (descending)

```
db.prizes.create_index([("year", 1)])
```

```
'year_1'
```

```
%%timeit  
# Previously: 524  $\mu$ s  $\pm$  7.34  $\mu$ s  
docs = list(db.prizes.find({"year": "1901"}))
```

```
379  $\mu$ s  $\pm$  1.62  $\mu$ s per loop  
(mean  $\pm$  std. dev. of 7 runs, 1000 loops each)
```

```
%%timeit  
# Previously: 5.18 ms  $\pm$  54.9  $\mu$ s  
docs = list(db.prizes.find({}, sort=[("year", 1)]))
```

```
4.28 ms  $\pm$  95.7  $\mu$ s per loop  
(mean  $\pm$  std. dev. of 7 runs, 100 loops each)
```

```
4.28 ms  $\pm$  95.7  $\mu$ s per loop (mean  $\pm$  std. dev. of 7 runs, 100 loops each)
```

Adding a compound (multiple-field) index

```
db.prizes.create_index([("category", 1), ("year", 1)])
```

- index "covering" a query with projection

```
list(db.prizes.find({"category": "economics"},  
                    {"year": 1, "_id": 0}))
```

```
# Before  
645 µs ± 3.87 µs per loop  
(mean ± std. dev. of 7 runs, 1000 loops each)  
# After  
503 µs ± 4.37 µs per loop  
(mean ± std. dev. of 7 runs, 1000 loops each)
```

- index "covering" a query with projection and sorting

```
db.prizes.find_one({"category": "economics"},  
                   {"year": 1, "_id": 0},  
                   sort=[("year", 1)])
```

```
# Before  
673 µs ± 3.36 µs per loop  
(mean ± std. dev. of 7 runs, 1000 loops each)  
# After  
407 µs ± 5.51 µs per loop  
(mean ± std. dev. of 7 runs, 1000 loops each)
```


Learn more: ask your collection and your queries

```
db.laureates.index_information() # always an index on "_id" field
```

```
{'_id_': {'v': 2, 'key': [('_id', 1)], 'ns': 'nobel.laureates'}}
```

```
db.laureates.find(  
    {"firstname": "Marie"}, {"bornCountry": 1, "_id": 0}).explain()
```

```
...  
'winningPlan': {'stage': 'PROJECTION',  
    'transformBy': {'bornCountry': 1, '_id': 0},  
    'inputStage': {'stage': 'COLLSCAN',  
    ...
```

```
db.laureates.create_index([("firstname", 1), ("bornCountry", 1)])  
db.laureates.find(  
    {"firstname": "Marie"}, {"bornCountry": 1, "_id": 0}).explain()
```

```
...  
'winningPlan': {'stage': 'PROJECTION',  
    'transformBy': {'bornCountry': 1, '_id': 0},  
    'inputStage': {'stage': 'IXSCAN',  
    'keyPattern': {'firstname': 1, 'bornCountry': 1},  
    'indexName': 'firstname_1_bornCountry_1',  
    ...
```

Let's practice!

INTRODUCTION TO MONGODB IN PYTHON

Limits and Skips with Sorts, Oh My!

INTRODUCTION TO MONGODB IN PYTHON



Donny Winston
Instructor

Limiting our exploration

```
for doc in db.prizes.find({}, ["laureates.share"]):
    share_is_three = [laureate["share"] == "3"]
    for laureate in doc["laureates"]
    assert all(share_is_three) or not any(share_is_three)
```

```
for doc in db.prizes.find({"laureates.share": "3"}):
    print("{year} {category}".format(**doc))
```

```
2017 chemistry
2017 medicine
2016 chemistry
2015 chemistry
2014 physics
2014 chemistry
2013 chemistry
...
```

```
for doc in db.prizes.find({"laureates.share": "3"}, limit=3):
    print("{year} {category}".format(**doc))
```

```
2017 chemistry
2017 medicine
2016 chemistry
```

Skips and paging through results

```
for doc in db.prizes.find({"laureates.share": "3"}, limit=3):  
    print("{year} {category}".format(**doc))
```

```
2017 chemistry  
2017 medicine  
2016 chemistry
```

```
for doc in db.prizes.find({"laureates.share": "3"}, skip=3, limit=3):  
    print("{year} {category}".format(**doc))
```

```
2015 chemistry  
2014 physics  
2014 chemistry
```

```
for doc in db.prizes.find({"laureates.share": "3"}, skip=6, limit=3):  
    print("{year} {category}".format(**doc))
```

```
2013 chemistry  
2013 medicine  
2013 economics
```

Using cursor methods for {sort, skip, limit}

```
for doc in db.prizes.find({"laureates.share": "3"}).limit(3):  
    print("{year} {category}".format(**doc))
```

```
2017 chemistry  
2017 medicine  
2016 chemistry
```

```
for doc in (db.prizes.find({"laureates.share": "3"}).skip(3).limit(3)):  
    print("{year} {category}".format(**doc))
```

```
2015 chemistry  
2014 physics  
2014 chemistry
```

```
for doc in (db.prizes.find({"laureates.share": "3"})  
            .sort([("year", 1)])  
            .skip(3)  
            .limit(3)):  
    print("{year} {category}".format(**doc))
```

```
1954 medicine  
1956 physics  
1956 medicine
```

Simpler sorts of sort

```
cursor1 = (db.prizes.find({"laureates.share": "3"}).skip(3).limit(3)
          .sort([("year", 1)]))
```

```
cursor2 = (db.prizes.find({"laureates.share": "3"}).skip(3).limit(3)
          .sort("year", 1))
```

```
cursor3 = (db.prizes.find({"laureates.share": "3"}).skip(3).limit(3)
          .sort("year"))
```

```
docs = list(cursor1)
assert docs == list(cursor2) == list(cursor3)
for doc in docs:
    print("{year} {category}".format(**doc))
```

```
1954 medicine
1956 physics
1956 medicine
```

```
doc = db.prizes.find_one({"laureates.share": "3"},
                          skip=3, sort=[("year", 1)])
print("{year} {category}".format(**doc))
```

Limit or Skip Practice? Exactly.

INTRODUCTION TO MONGODB IN PYTHON