Technologies

Outline

- Rapid prototyping for development
- Technology stacks
- Choice of programming language
- Choice of database

Prototyping for development

- Recall: In **design context**, prototyping…
  - Refers to creation, testing of UX/UI designs
  - Generally, **no code** involved
- In **development context**, prototyping…
  - Refers to preliminary implementation of design
  - Experimenting with coded solutions (**yes code**)

Rapid prototyping

When prototyping (for development)…

- Want something working ASAP
    - To validate design
    - To compare technologies
    - Etc.
- Don’t worry about non-functional requirements
    - Might call these “quality attributes”
    - *What do you think might be some examples?*

Quality attributes

- Non-functional requirements used to evaluate the performance of a system
- Some key attributes:
  - Dependability: “aggregate of availability, reliability, safety, integrity and maintainability”
  - Integrity: “depends on security and survivability”
  - Security: “composite of confidentiality, integrity and availability"

https://en.wikipedia.org/wiki/List_of_system_quality_attributes

Iteration/evolution

- When prototyping: develop quickly
- As confidence in design/approach grows…
  - Will consider quality attributes more seriously
  - Will likely choose to redevelop parts/all of system
- Over multiple iterations, system evolves
  - Systems may require more significant overhauls
  - Perhaps new system developed that deprecates the "old" one—old system is decommissioned

Selecting tools

- When doing rapid prototyping…
  - Select tools that enable quick development
  - Aim is quick-and-dirty solution, **not** perfection
- Later can consider pros/cons of various tools
  - Learn from prototype what you actually need
  - Then can find the tools that give a good balance of your highest-priority quality attributes
  - Usually done by ≥ senior developers (junior developers told what tools to use)

Outline

- Rapid prototyping for development
- Technology stacks
- Choice of programming language
- Choice of database

Outline

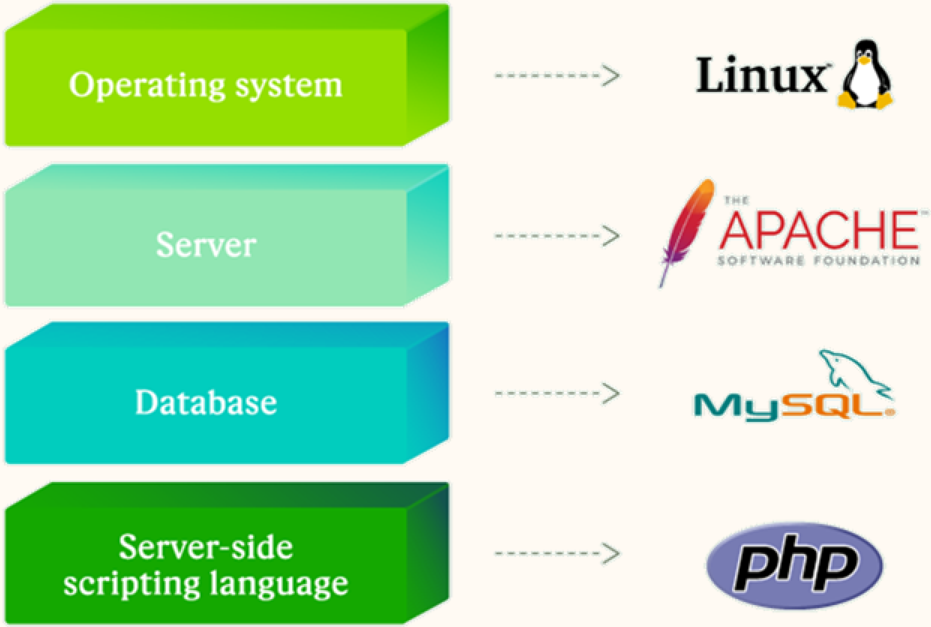
- Rapid prototyping for development
- Technology stacks
- Choice of programming language
- Choice of database

Outline

- Rapid prototyping for development
- Technology stacks
  - What are they, front/back-ends, development
  - Web development stacks
  - Stacks for this class
- Choice of programming language
- Choice of database

Tech Stack (aka solution/software stack)

- Set of software subsystems/components needed to create a complete ***platform***
  - No additional software needed to support app
  - Applications said to "run on (top of)“ platform
- Example: classic LAMP web software stack →



Operating system

Server

Database

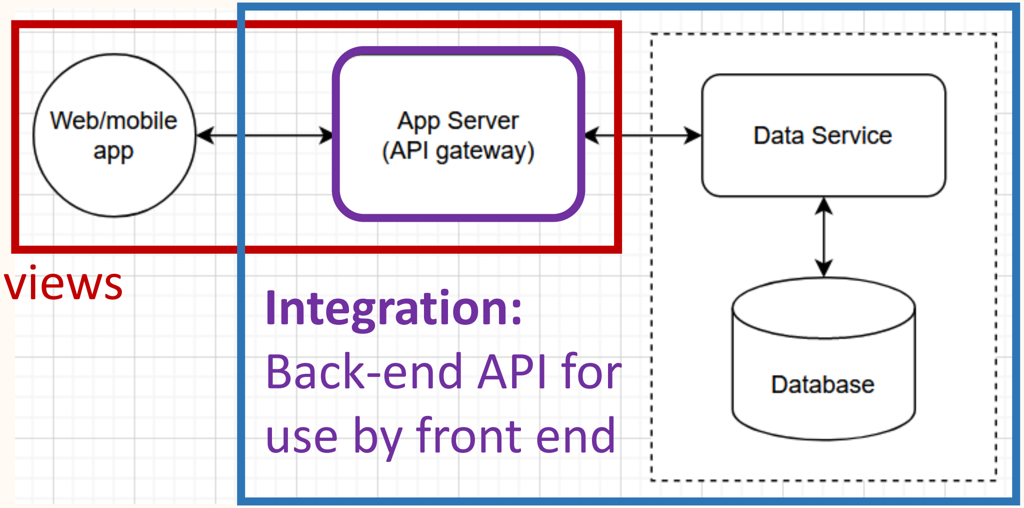
Server-side scripting language

Front-end and back-end

- “Stacks” aren’t usually linear any longer
- But do tend to have different regions
  - Front end:
    - Client-facing/client-side
    - Focus on “presentation layer” (UI, interaction)
  - Back end:
    - Server side (web/app server, service calls, API…)
    - Services and databases
    - Focus on “business logic” and data

Specialization of developers

- Developer usually specializes in certain parts
  - **Front-end** developer: user-facing websites/apps
  - **Back-end** developer: business logic, data, etc.
- **Full-stack developer** works all along the stack



Back end: Develop API routes for access to data, services

Front end: Develop UI views

Integration: Back-end API for use by front end

Why full-stack development?

- Working throughout front-, back-ends can be more efficient than separate developers
  - Communication barriers / lags are removed
- Can divide work differently
  - By user story or product feature
  - By customer (semi-custom-tailored solutions)
  - Etc.

Outline

- Rapid prototyping for development
- Technology stacks
  - What are they, front/back-ends, development
  - Web development stacks
  - Stacks for this class
- Choice of programming language
- Choice of database

Front-end stack/frameworks

- All front-end web technology is based on
  - HTML
  - CSS
  - JavaScript
- Front-end toolkits, frameworks
  - **Bootstrap**—free, open-source CSS framework for responsive, mobile-first front-end web dev.

BEYOND COURSE SCOPE

  - **React** (aka React.js or ReactJS)–free, open-source front-end JavaScript library for building UIs

Modern stacks

Many of the most popular stacks built using…

- **JavaScript**
  - Originally designed to run in the browser
  - But it’s limited in browser (e.g. you can't do file I/O)
- **Node.js**
  - Runtime environment for JavaScript (JS)
  - Can run JS outside the browser, like a "regular" programming language

Modern stacks: MEAN variants

- MEAN stack:
  - Database: ~~MongoDB~~
  - Web server: Express.js
  - Front-end: ~~Angular~~
  - Back-end: Node.js
- Variations replace one or more components

PostgreSQL → PERN

ex
node
JS
M
E
A
N

React.js → MERN

Vue.js → MEVN

Modern stacks: Python

Two most popular:

- Django
  - Full-stack framework
  - “Opinionated”—Designers have built “happy path” that makes dev. faster/easier if follow specific assumptions
- Flask
  - “micro-framework”—lightweight but extensible
  - Not “opinionated” —more flexible

Modern stacks: Serverless

Serverless computing

- Cloud computing execution model
  - Cloud provider allocates resources on demand
  - Takes care of the servers on behalf of their customers
- Misnomer: servers still used by cloud provider
- Developers not concerned with server details
  - e.g., capacity planning, configuration, management, maintenance, fault tolerance, or scaling of containers, VMs, or physical servers

Modern stacks: FaaS

AWS Lambda

Azure Functions

Function as a service (FaaS)

- Category of cloud computing services…
  - Provides platform to develop, run, manage application functionalities, reducing complexity
  - Maintains infrastructure typically associated with developing and launching an app (so dev. doesn’t)
- One way of achieving a "serverless" architecture
- Typically used when building microservices applications

Outline

- Rapid prototyping for development
- Technology stacks
  - What are they, front/back-ends, development
  - Web development stacks
  - Stacks for this class
- Choice of programming language
- Choice of database

Choosing a tech stack

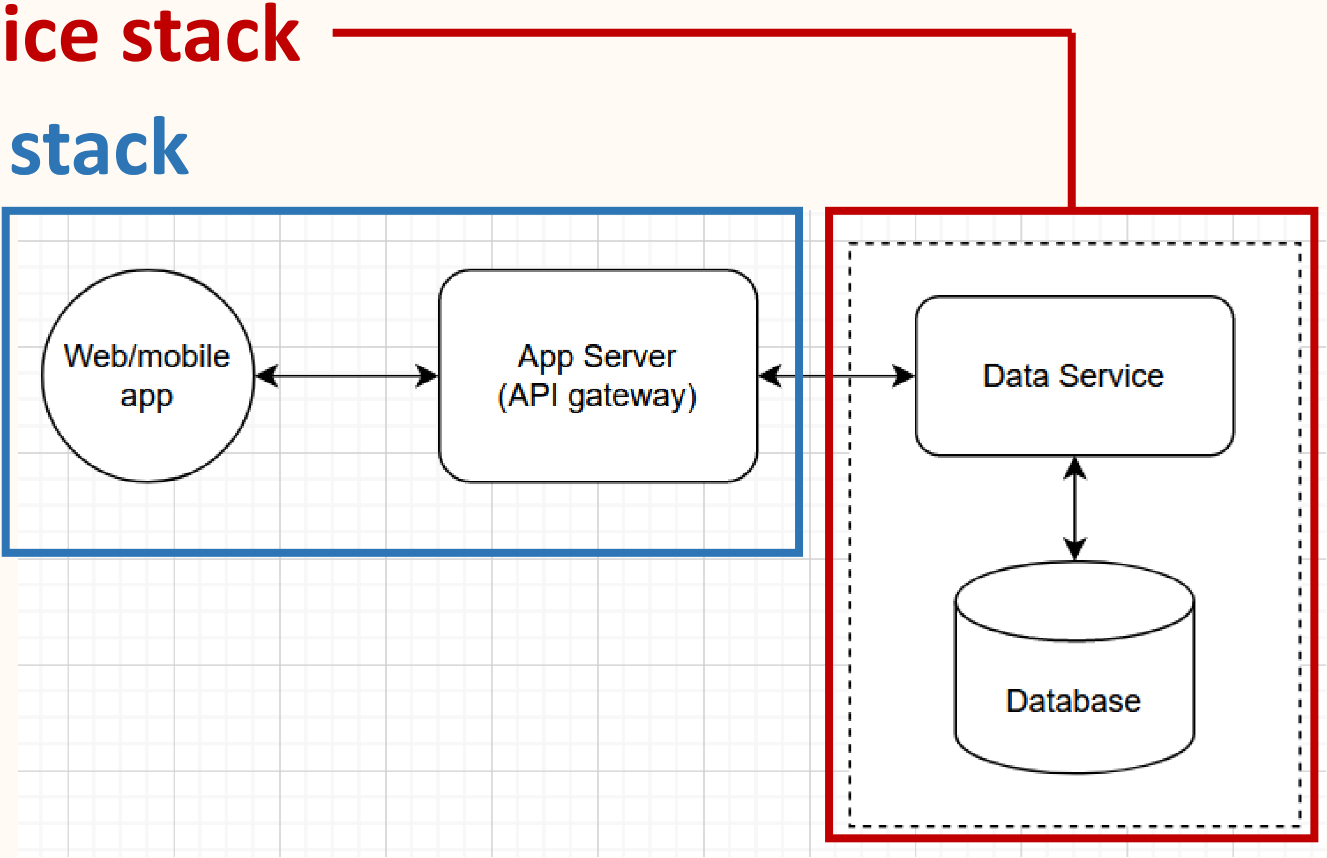
BEYOND COURSE SCOPE

Many things to consider, including…

- Target user platform (Web? Mobile?)
- Size/complexity of project
- Scalability (How many users at once?)
- Performance
- Cost
- Existing expertise on team
- How actively are components used/updated
- …

Your project will have two stacks

- **Service stack**
- **App stack**



Service stack for this class

• Serverless:	Azure functions
• Language:	Python
• Database:	MongoDB

App stack for this class

- Database:
  - not needed right now
  - data management done by service(s)
- Back-end/server: Python/Flask
- Front-end: Jinja
  - Jinja is a web template engine for Python

Outline

- Rapid prototyping for development
- Technology stacks
- Choice of programming language
- Choice of database

Criteria for language selection

- Some criteria for selecting a language
  - Readability — Easy to read, write, and maintain
  - Type mechanism
  - Supported paradigms
  - Libraries — Lots of ready-to-use code
  - Popularity — Good support community
- What will allow for **rapid prototyping**?

Language for project: Python

Why?

- High-level, general-purpose scripting language
- Design philosophy emphasizes **readability**
  - Indentation, minimal punctuation, for example
- Dynamically-typed, garbage collected
- Supports structural, OO, functional paradigms
- "Batteries included"—comprehensive **standard library**
- Consistently **popular** w/ strong community

“Python where we can…”

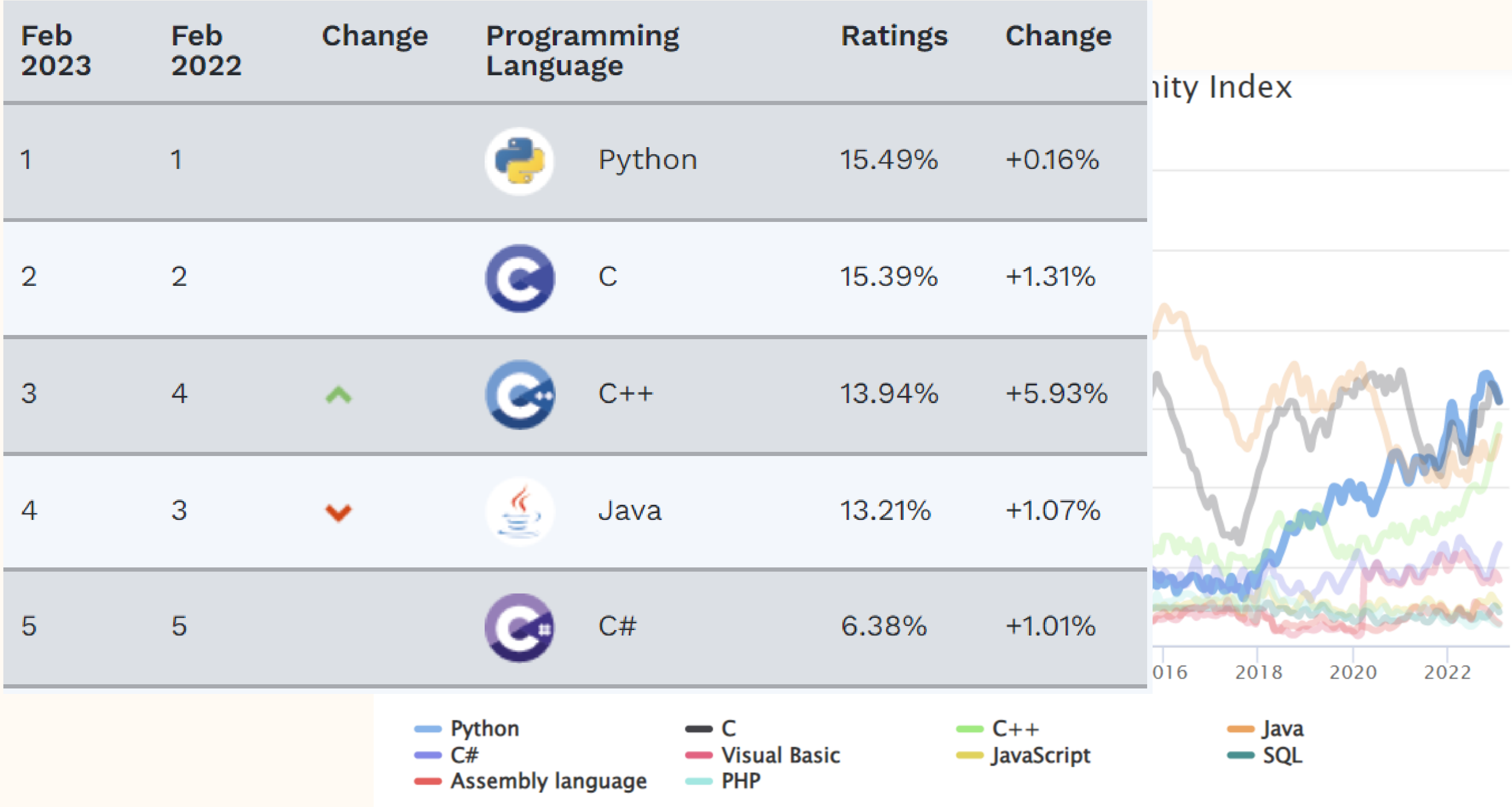
- Founders of Google made the decision: “**Python** where we can, **C++** where we must.”
- **C++** was used when
  - Memory control vital (efficient memory use)
  - Low latency desired (efficient processing)
- In the other facets, **Python** enabled
  - Ease of maintenance
  - Relatively fast delivery

A few more words about Python

Python is…

- Interpreted: Write code, test it w/o build files
- Flexible: easily applied to many situations
- High-level: Focus on main ideas, not minor details
- Extendable: Many modules exist, are easy to learn/use
- Well-supported: Runs on most platforms
- Popular: somebody on your team likely can read your code and cooperate with you

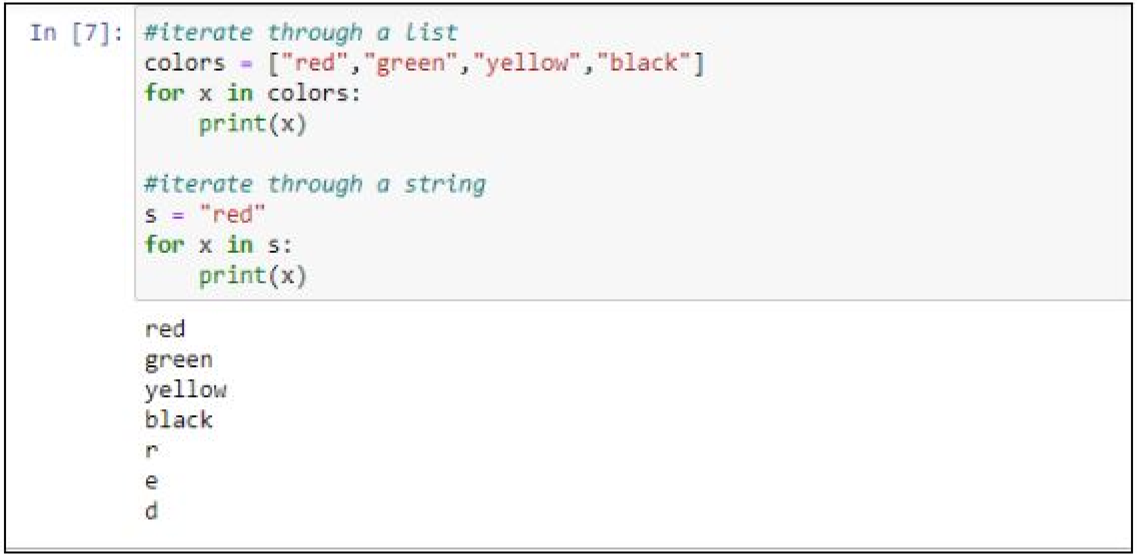
TIOBE Index



Feb 2023	Feb 2022	Change	Programming Language	Ratings	Change
1	1		Python	15.49%	+0.16%
2	2		C	15.39%	+1.31%
3	4	▲	C++	13.94%	+5.93%
4	3	▼	Java	13.21%	+1.07%
5	5		C#	6.38%	+1.01%

Basic Python syntax

- White-space delimited
- Blocks of code are indented (no Java braces)
- For-in loop example ➔



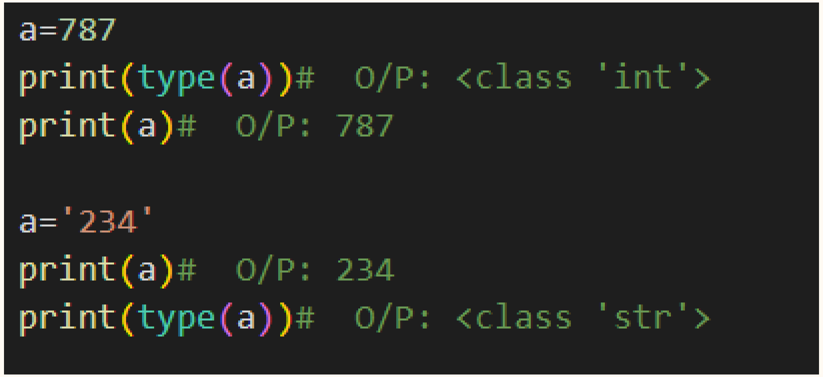
```
#iterate through a list
colors = ["red","green","yellow","black"]
for x in colors:
    print(x)

#iterate through a string
s = "red"
for x in s:
    print(x)
```

```
red
green
yellow
black
r
e
d
```

Python type mechanism

- Dynamically typed; but also strongly typed
  - Type of variable determined at runtime
  - Types must be compatible for operators & methods

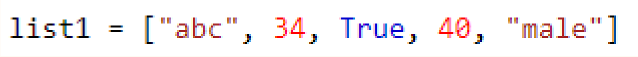


```
a=787
print(type(a))#  O/P: <class 'int'>
print(a)#  O/P: 787

a='234'
print(a)#  O/P: 234
print(type(a))#  O/P: <class 'str'>
```

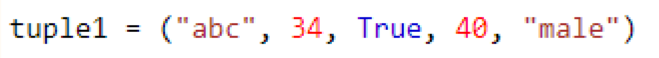
- Examples
  - OK: Add integer to floating point number
  - Error: Add integer to string

Python collections



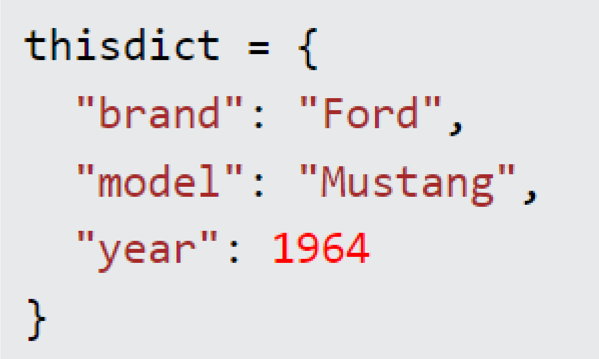
```
list1 = ["abc", 34, True, 40, "male"]
```

- Lists—look like arrays storing different types
- Tuples—on-the-fly structure



```
tuple1 = ("abc", 34, True, 40, "male")
```

- Sets—no duplicates allowed, unordered
- Dictionaries—Associative arrays (maps)
- To get efficient arrays…
  - Use **NumPy** array type
  - Supports multiple dimensions



```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Python in a pinch

- Here are some tutorials/references to try
  - https://docs.python.org/3/tutorial/
  - https://www.w3schools.com/python/
- If you know Java or C++, it’s easy to pick up!

- Preview lab: cs518.lab3a.python-db-essentials

Outline

- Rapid prototyping for development
- Technology stacks
- Choice of programming language
- Choice of database

Outline

- Rapid prototyping for development
- Technology stacks
- Choice of programming language
- Choice of database
  - Database types
  - MongoDB & PyMongo

What is a database?

- Organized collection of data stored and accessed electronically
  - Small databases stored on a file system
  - Large databases hosted on clusters or cloud
- Database design involves many concerns
  - data modeling, efficient data representation and storage, query languages, data security and privacy, distributed computing issues (including support for concurrent access, fault tolerance)

Types of database

- **SQL**–Structured Query Language
  - Decades-old access method: relational databases
  - Most who work with databases familiar with it
- **NoSQL** databases created to leverage
  - Unstructured data
  - Ever larger amounts of storage
  - Ever more-powerful processors
  - New types of analytics

SQL vs. NoSQL

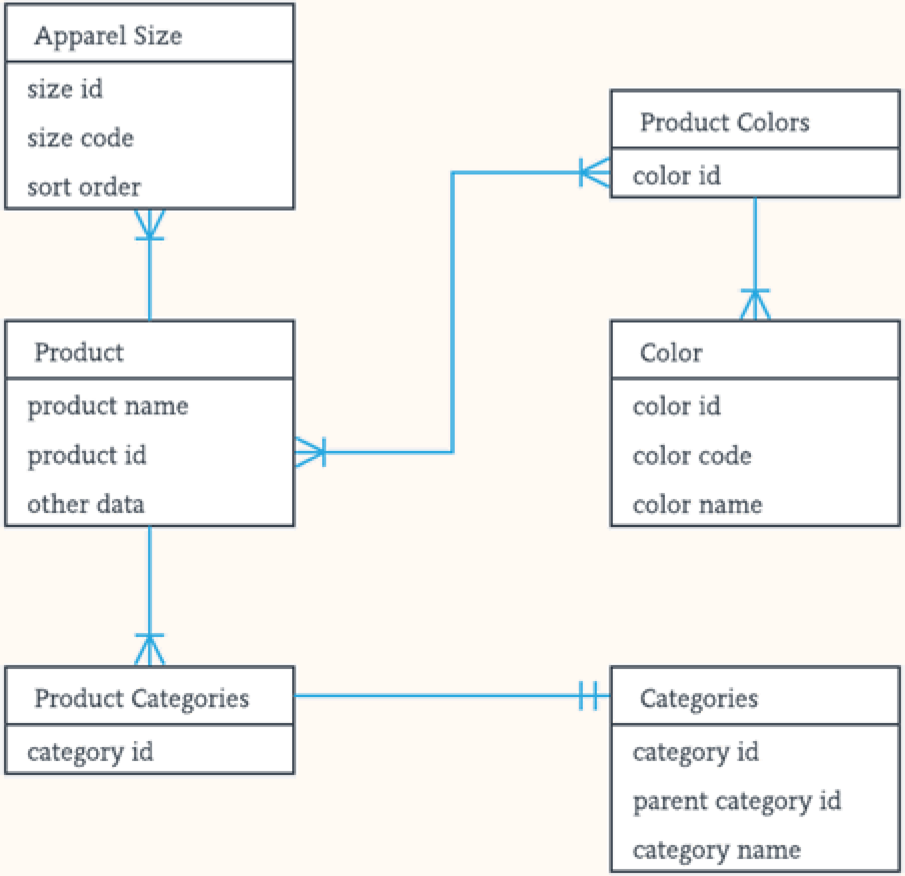
- Relational vs. non-relational
- Predefined schema vs. dynamic schema
- Scales well vertically vs. horizontally
- Types of data they handle best
- Better suited to multi-row data or unstructured data

Relational databases

- Data is stored in rows (like structures)
- Tables contain many rows (like arrays)
- Tables are linked in certain ways
  - Often linked using unique identifiers (IDs)
    - Allows for 1-1, many-1, and many-many relationships
  - Indexed by IDs for extremely efficient access
- Complex SQL queries can be efficient even with vast amounts of data

DB schema and ER diagrams

- DB schema
  - Structure of database
- ER Diagram
  - ER = Entity Relationship
  - Displays relationships of entity sets stored in DB
  - Explain logical structure
  - 3 concepts: entities, attributes, relationships



NoSQL databases

- Originally referred to "non-SQL" or "non-relational" database
- Provides for storage/retrieval of data in ways other than tabular relations
  - Key-value pairs
  - Wide columns
  - Graphs
  - Documents
  - Etc.

Why NoSQL?

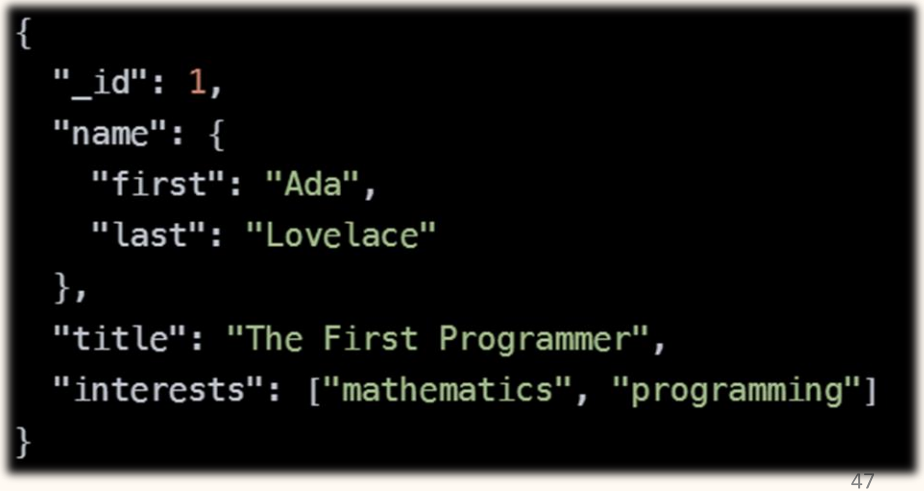
- Simplicity of design
- Simpler "horizontal" scaling to clusters of machines (hard for relational databases)
- Finer control over availability
- Limits the object-relational impedance mismatch
  - Object-oriented data is surprisingly challenging to model well in relational databases

Document-oriented databases

- AKA document store
- Designed for managing document-oriented information (AKA semi-structured data)
- Inherently a subclass of the key-value store
- Store all information for an object in a single instance in the database
    - Relational: one object spread across many tables
- Each object can be different from every other

Documents, JSON

- Documents encapsulate data in some standard format or encoding
- JSON (JavaScript object notation)
  - One of these standard formats for documents
  - Same way objects defined in JavaScript
  - Used by many programming languages



```
{
  "_id": 1,
  "name": {
    "first": "Ada",
    "last": "Lovelace"
  },
  "title": "The First Programmer",
  "interests": ["mathematics", "programming"]
}
```

Outline

- Rapid prototyping for development
- Technology stacks
- Choice of programming language
- Choice of database
  - Database types
  - MongoDB & PyMongo

MongoDB

- Source-available, cross-platform document-oriented database program
- Classified as NoSQL database program
- Uses JSON-like documents with optional schemas

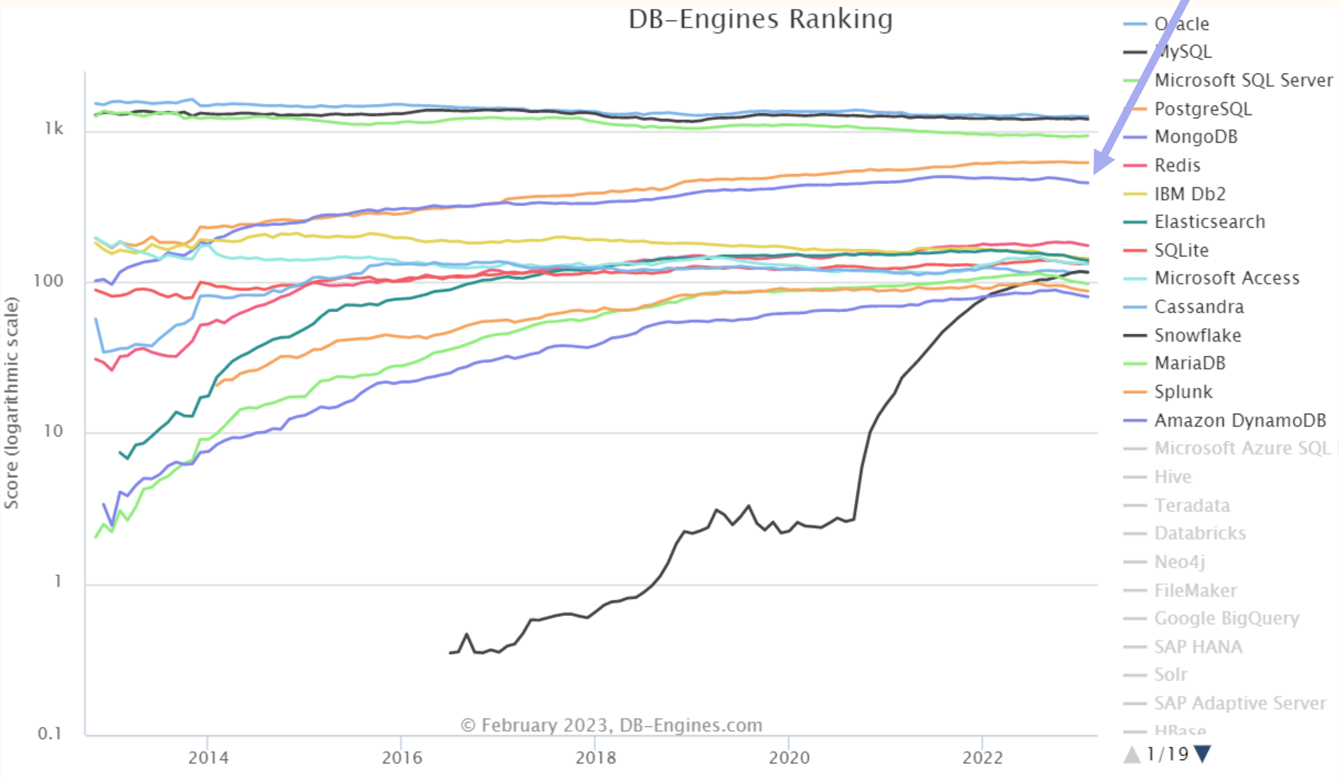
Why MongoDB?

Leverages advantages of a document-oriented NoSQL databases

- Scale cheaper
- Query faster
- ***Pivot more easily***
  - When requirements mean changes to database, much easier to make than relational databases
- ***Program more rapidly***
  - MongoDB documents map directly to data structures

Database engine rankings

- MongoDB: most popular NoSQL database engine



DBMS—tie DB to your programs

- Database management system (DBMS)
  - Software that allows end users & applications to interact with database
  - Facilities database administration
- Database System =
  - Database + DBMS + associated applications
  - Term *database* may also be used loosely to refer to the database system or any significant part

PyMongo

- Python DBMS
  - Has tools for working with MongoDB
  - Recommended way to work with MongoDB
- Python dictionaries similar to JSON docs
  - PyMongo allows you to insert python dictionaries
- Further reading:

  https://www.w3schools.com/python/python_mongodb_getstarted.asp
- Lab preview:
  - cs518.lab3b.pymongo-tutorial

Summary/Questions?

- Rapid prototyping for development
- Technology stacks
  - What are they, front/back-ends, development
  - Web development stacks
  - Stacks for this class
- Choice of programming language
- Choice of database
  - Database types
  - MongoDB & PyMongo