EPFL, CIVIL-127

Programming and software development for engineers

Schedule

- Tuesday, May 13th
  - We'll discuss writing maintainable code, data formats, databases, and cloud services
- Tuesday, May 20th
  - Are there any topics you would like to learn about?
- Friday, May 23rd
  - Project due at 6pm
- Tuesday, May 27th
  - Some of you will present your one-pager in front of the class

Writing maintainable code

Writing maintainable code

- Code is maintainable if your future self or someone else can easily fix bugs or add features to the existing codebase
  - Knowledge currently in your head will be forgotten when you re-visit the code. Write down information you think will be useful later
- For the final project:
  - “We'll review your code structure and comments. We want to see class-level and method-level comments. You will earn up to 1 point for how your code is written.”

Code comments

```
class Cal():

    def __init__(self):
        ...

    def print(self):
        ...

    def week_start(self, day):
        ...
    ...
```

- Comment your classes, methods, and functions

Code comments

```
# Implements the API per
# https://github.com/vita-epfl/…/Lab4_instructions.
class Cal():
    # Initializes the class. The current month and year
are used by default.
    # The week starts with Monday.
    def __init__(self):
        ...

    # Sets the start day of the week. Raises an
exception if day is invalid.
    def week_start(self, day):
        ...

    def print(self):
        ...
    ...
```

- Use the correct comment style:
  - use `'''` for class and function/method comments
  - # for code comments

Code comments

```
class Cal():
    '''
    Implements the API per
    https://github.com/vita-epfl/…/Lab4_instructions.md
    '''

    def __init__(self):
        '''
        Initializes the class. The current month and
year are used by default.
        The week starts with Monday.
        '''
        ...

    ...
```

- Make sure your comments are adding value. E.g. no need to comment the initializer if you don’t have anything to say
- Use links, as long as the links are public
- Explain why you are doing something
- Use drawings if they help explain concepts
- Describe your assumptions!

Writing maintainable code

- Real world example: lib/heapq.py
  - ~120 lines of comment at the top of the file, gives an overview
  - ASCII diagram
  - http links to additional resources
  - Every method is commented
  - Tricky pieces of the code are also commented

General documentation

- https://diataxis.fr/
  - four forms of documentation: tutorials, how-to guides, technical reference, and explanation
- https://passo.uno/seven-action-model/
  - Seven-action Documentation Model
- https://docsfordevelopers.com/
- https://readthedocs.com and https://writethedocs.org/
- Lots of other projects aim to help software engineers write better documentation

Technical writing

- Technical writing is an important skill, for **all fields of engineering**
  - “You can have brilliant ideas, but if you can't get them across, your ideas won't get you anywhere.” Lee Iacocca
- Like most other skills, you can invest time now to improve the skill for the rest of your life
- Online resources
  - https://developers.google.com/tech-writing/one
  - https://developers.google.com/tech-writing/two
  - https://developers.google.com/tech-writing/error-messages
  - https://en.wikipedia.org/wiki/BLUF_(communication)

Data Interchange Formats

Data Interchange Formats

- Provide a shared format for computers to exchange data
- Enable storing data for later use
- It is less error prone and faster to use an existing format vs inventing your own bespoke format

JavaScript Object Notation (JSON)

```
{
 "name": "John Doe",
 "age": 20,
 "isAsleep": false,
 "nationality": null,
 "hobbies": [
   "Cooking",
   "Football",
   "Coding"
 ],
 "address": {
   "country": "Switzerland",
   "state": "Vaud",
   "city": "Lausanne",
   "street": "Av. de la gare 10"
 }
}
```

- JSON is based on a very restricted subset of JavaScript literals
- Very widely used file format
- E.g.: format for the project's config file
- 6 datatypes
  - String
  - Floating point (*)
  - Boolean
  - Null
  - Array
  - Dictionary

  * JavaScript float type can represent integers from $-(2^{53} - 1)$ to $(2^{53} - 1)$. For 64-bit integers, it is common to piggyback on strings
- https://www.rfc-editor.org/rfc/rfc8259

JSON

Pros

- Very widely used
- Human readable
- Self-describing

Cons

- No comments
- Space inefficient
- Renaming fields can be hard

Protocol Buffers (protobuf)

Schema

```
message Person {
  string name = 1;
  int32 age = 2;
  bool isAsleep = 3;
  optional string nationality = 4;
  repeated string hobbies = 5;
  Address address = 6;
}

message Address {
  string country = 1;
  string state = 2;
  string city = 3;
  string street = 4;
}
```

Value

```
0a084a6f686e20446f65101418002a07436f6f6b696e672a08466f6f7462616c6c2a06436f64696e6732300a0b537769747a65726c616e64120456175641a084c617573616e6e65221141762e2064652065206c612067617265203130
```

- Explicit schema
- Binary message format
- Widely used within data centers when servers are communicating with each other (e.g. microservices)

Comma/Tab separated values (CSV/TSV)

```
name,age,address
John Doe,20,"Av. de la gare 10, Lausanne"
Alice,21,"Rte. de la Sorge 23, Lausanne"
```

- Used to represent tabular data
- Lots of data import issues in practice, important to understand your specific implementations limitations and test edge cases
  - Not all implementations support newlines within quotes or quotes inside quotes
  - Data type confusion (e.g. Excel auto-converting text values to dates)

Pickle

```
class Person:
   def __init__(self, name, age, is_asleep,
nationality, hobbies, address):
        self.name = name
        self.age = 20
        self.isAsleep = is_asleep
        self.nationality = nationality
        self.hobbies = hobbies
        self.address = address


john = Person(...)
```

800495e4000000000000008c085f5f6d61696e5f5f948c06506572736f6e9493942981947d94288c046e616d65948c084a6f686e20446f65948c03616765944b148c08697341736c65657094898c0b6e6174696f6e616c697479944e8c07686f6262696573945d94288c07436f6f6b696e67948c08466f6f7462616c6c948c06436f64696e6794658c0761646472657373947d94288c07636f756e747279948c0b537769747a65726c616e64948c057374617465948c0456617564948c04636974794948c084c617573616e6e65948c0673747265657494 8c1141762e206465206c612067617265203130947575622e

- Python's native object serializer/deserializer
- **Warning: The pickle module is not secure!**
- Historically, all native object deserializers have been prone to arbitrary code execution. This has been true for Python, Java, PHP, etc.
- Used in some cases to exchange ML data

Extensible Markup Language (XML)

```xml
<?xml version="1.0"?>
<person name="John Doe" age="20">
 <isAsleep>false</isAsleep>
 <hobbies>
   <hobby>Cooking</hobby>
   <hobby>Football</hobby>
   <hobby>Coding</hobby>
 </hobbies>
 <address country="Switzerland" state="Vaud"
city="Lausanne" street="Av. de la gare 10"/>
</person>
```

- A tree of nodes
- Nodes are defined with an opening and closing tag
- Nodes can have key-value attributes
- Nodes can have child nodes
- The document type definition (DTD) specifies the schema for a document
- As an author: not always clear when to use an attribute vs a child node
- Examples
  - Vector graphics (svg)
  - Microsoft Office XML formats (docx, xlsx)

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <meta name="viewport"
content="width=device-width,
initial-scale=1">
 <title>HTML sample</title>
 <link rel="stylesheet" href="style.css">
</head>
<body>
 <h1>Page Title</h1>
 <script src="scripts.js"></script>
</body>
</html>
```

- Like XML with minor differences
  - Some tags don't need a closing tag
- Used for web pages and web applications

Markdown

```
# Page title


- Bullet point 1
- Bullet point 2


Some text with [a link](https://…)
```

- Easy-to-read text format with support for basic formatting
- Exact syntax and features differ between implementations
- Tip: VSCode lets you preview markdown files as you type

HTTP API server

- What is an API server?
  - HTTP Server which exposes endpoints (e.g. /foo and /bar)
  - Typically, takes a JSON request and returns a JSON response
- Two main choices for Python: Flask and Django

API Server

```
app = Flask("demo")

class Temperature(BaseModel):
   temp: int
   note: str

@app.route("/temp")
def temp():
   temp = 20
   if temp < 0:
       note = "it's freezing"
   elif temp < 18:
       note = "it's cold"
   else:
       note = "it's nice"

   t = Temperature(temp=temp, note=note)
   return t.model_dump_json()
```

Databases

Structured Query Language (SQL)

- Relational data is stored in a collection of tables
- SQL provides a language to insert, update, and query the data
  - In 2025, SQL might seem primitive but it is an industry-wide standard
- SQL is implemented by lots of engines: SQLite, MySQL/MariaDB, Postgres, Microsoft SQL Server, Oracle, etc. These engines share a common set of features

SQL

- Creating tables
  - CREATE TABLE *table_name* (*field1_name field1_type*, *field2_name*, *field2_type*, ...)
- Inserting data
  - INSERT INTO *table_name* (*field1_name*, *field2_name*, ...) VALUES (*field1_value*, *field2_value*, ...)
- Viewing data
  - SELECT * FROM *table_name*
  - SELECT * FROM *table_name* WHERE *some_condition*
- Updating data
  - UPDATE *table_name* SET *field1_name=new_value* WHERE *some_condition*
- Programming languages typically offer object-relational mapper (ORM) libraries, making it easier to interact with SQL databases

Example: books + authors

- We want an `authors` table, with `id` and `name` columns
- `CREATE TABLE authors(id integer primary key autoincrement, name string);`
- `INSERT INTO authors (name) VALUES ("Jeff Zentner")`
- …
- `sqlite> select * from authors;`

```
+----+--------------------+
| id |        name        |
+----+--------------------+
| 1  | Jeff Zentner       |
| 2  | Brittany Cavallaro |
| 3  | J. R. R. Tolkien   |
+----+--------------------+
```

Example: books + authors

- Next, we create a `books` table with `id` and `title` columns
- `CREATE TABLE books (id integer primary key autoincrement, title string);`
- `INSERT INTO books (title) VALUES ("Sunrise Nights")`
- …
- `sqlite> select * from BOOKS;`

```
+----+-----------------------+
| id |         title         |
+----+-----------------------+
| 1  | Sunrise Nights        |
| 2  | The Hobbit            |
| 3  | The Lord of the Rings |
+----+-----------------------+
```

Example: books + authors

- Finally, we create a `books_authors` table. This table enable us to have multiple authors per book and also track all the books a given has published
- `CREATE TABLE books_authors (book_id integer, author_id integer);`
- `INSERT INTO books_authors (book_id, author_id) VALUES (1, 1)`
- …
- `sqlite> select * from books_authors;`

```
+---------+-----------+
| book_id | author_id |
+---------+-----------+
| 1       | 1         |
| 1       | 2         |
| 2       | 3         |
| 3       | 3         |
+---------+-----------+
```

Example: books + authors

- We can run queries to join data between different tables
- `sqlite> select books.id, books.title, authors.id, authors.name from books_authors join books on books_authors.book_id=books.id join authors on books_authors.author_id=authors.id;`

```
+----+-----------------------+----+--------------------+
| id |         title         | id |        name        |
+----+-----------------------+----+--------------------+
| 1  | Sunrise Nights        | 1  | Jeff Zentner       |
| 1  | Sunrise Nights        | 2  | Brittany Cavallaro |
| 2  | The Hobbit            | 3  | J. R. R. Tolkien   |
| 3  | The Lord of the Rings | 3  | J. R. R. Tolkien   |
+----+-----------------------+----+--------------------+
```

Example: books + authors

- We can filter rows, e.g. by authors who have published at least two books:
- ```
  sqlite> select authors.id, authors.name, count(1) as c from books_authors join authors on
  books_authors.author_id=authors.id group by author_id having c>=2;
  ```

```
+----+------------------+---+
| id |       name       | c |
+----+------------------+---+
| 3  | J. R. R. Tolkien | 2 |
+----+------------------+---+
```
```

Example: books + authors

- Or by books which were published by at least two authors:
- `sqlite> select books.id, books.title, count(1) as c from books_authors join books on books_authors.book_id=books.id group by book_id having c>=2;`

```
+----+----------------+---+
| id |     title      | c |
+----+----------------+---+
| 1  | Sunrise Nights | 2 |
+----+----------------+---+
```

Why do people love SQL?

- SQL's main strength is that it enables building systems which are (usually) fast at answering queries about the data
- This enables implementing complex computer systems today which can service future needs (i.e. you don't need to think about the shape of future queries)

Why do people love SQL?

- SQL provides transactions: a set of writes will either all succeed or fail (*) as well as isolation

  * different products provide vastly different transactional guarantees (see https://jepsen.io/)

NoSQL

- Sometimes you just want a fast key-value store
- Or you want to store large json blobs
- Examples:
  - Redis
  - DynamoDB
  - Cassandra
  - MongoDB
  - etc.
- It can be hard to pick among these products!

Realtime databases

- Applications are automatically informed as new data is added to the database
- Each query creates a data stream, which gets populated by the database engine
- Example:
  - Parse
  - Firebase
  - RethinkDB
  - etc.

Graph-oriented databases

- Most popular: GraphQL (older solutions, such as Neo4J, do exist)
- Minimizes bandwidth and round trips when dealing with graph-oriented data
- Can be used for reads only or reads+writes
- Typically also provides real time updates

Analytics databases

- Query TB or PB-sized datasets
- Typically, query is split and processed on hundreds of servers at the same time
- Some analytics engines are able to instantly return approximate responses
- Some analytics products implement SQL, others work differently
- Examples
  - Hive
  - PrestoDB
  - etc.

Time-series databases

- Store events
- Can quickly return counts, max, min, average over arbitrary windows
- Typically handles compressions and stores historical data at a coarser granularity
  - E.g. 1-min resolution for last 7 days, 1-hour resolution for last 30 days, 1-day resolution for last year.
- Typically coupled with monitoring and notifications

Cloud Infrastructure

Providers

- Big Three
  - Amazon Web Services (AWS)
  - Google Cloud Platform (GCP)
  - Microsoft Azure
- A whole bunch of smaller players
  - IBM Cloud, Oracle Cloud Infrastructure, Cloudflare, Salesforce, etc.

Providers

- On-demand data storage and compute
  - Typically billed at a per-MB and per-second granularity
- In theory, cheaper than to buy or rent your own hardware and datacenter space
- Makes sense for sporadic or unpredictable workloads
  - E.g. training a ML model, it's cheaper to rent a GPU-month than to buy the hardware
  - Serverless functions provide a way to handle requests without having to allocate a fixed number of servers

Providers

- The industry is slowly moving towards standardization
- E.g.
  - Terraform to define resources
  - Containers to package applications
  - Kubernetes (K8s) to manage deployments
  - “Cattle vs Pets” mentality is spreading
- But in practice, moving from one cloud provider to another is difficult
  - Vendor lock-in!

Amazon AWS database-related products

- Amazon Aurora
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon Keyspaces (for Apache Cassandra)
- Amazon MemoryDB
- Amazon Neptune
- Amazon Relational Database Service
- Amazon RDS for Db2
- Amazon RDS on VMware
- Amazon Quantum Ledger Database (Amazon QLDB)
- Amazon Timestream
- Amazon DocumentDB (with MongoDB compatibility)
- Amazon Lightsail managed databases

- Amazon Athena
- Amazon CloudSearch
- Amazon DataZone
- Amazon EMR
- Amazon FinSpace
- Amazon Kinesis
- Amazon Data Firehose
- Amazon Managed Service for Apache Flink
- Amazon Kinesis Data Streams
- Amazon Kinesis Video Streams
- Amazon OpenSearch Service
- Amazon OpenSearch Serverless
- Amazon Redshift
- Amazon Redshift Serverless
- QuickSight
- AWS Clean Rooms
- AWS Data Exchange
- AWS Data Pipeline
- AWS Entity Resolution
- AWS Glue
- AWS Lake Formation
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)

Amazon AWS other major products

- EC2: elastic compute
- S3: key-value storage
- Lambda: serverless compute
- Cognito: user authentication layer
- SNS: email/SMS notifications
- SQS: queue
- Lots of ML and AI-related products

More about cloud providers

- AWS has over 200 products
- GCP and Azure have similar products, with different names
- Similar products can result in vastly different performance and costs
- Picking the right set of tools is hard!