COMP1531

4.4 - Packages

Virtual Environments

A virtual environment is a tool that helps to keep dependencies required by different projects separate by creating isolated python virtual environments for them.

You can read more about them here and here.

You may be asked a question about them on the exam, but you will never be required to use them.

They are often required for use with CI/CD

Virtual Environments

```
pip3 install virtualenv
python3 -m virtualenv venv/
source venv/bin/activate

# Do stuff

deactivate

pip3 freeze > requirements.txt # Save modules
pip3 install -r requirements.txt # Install modules
```

Overview

- Data storage
- Data transfer

Data

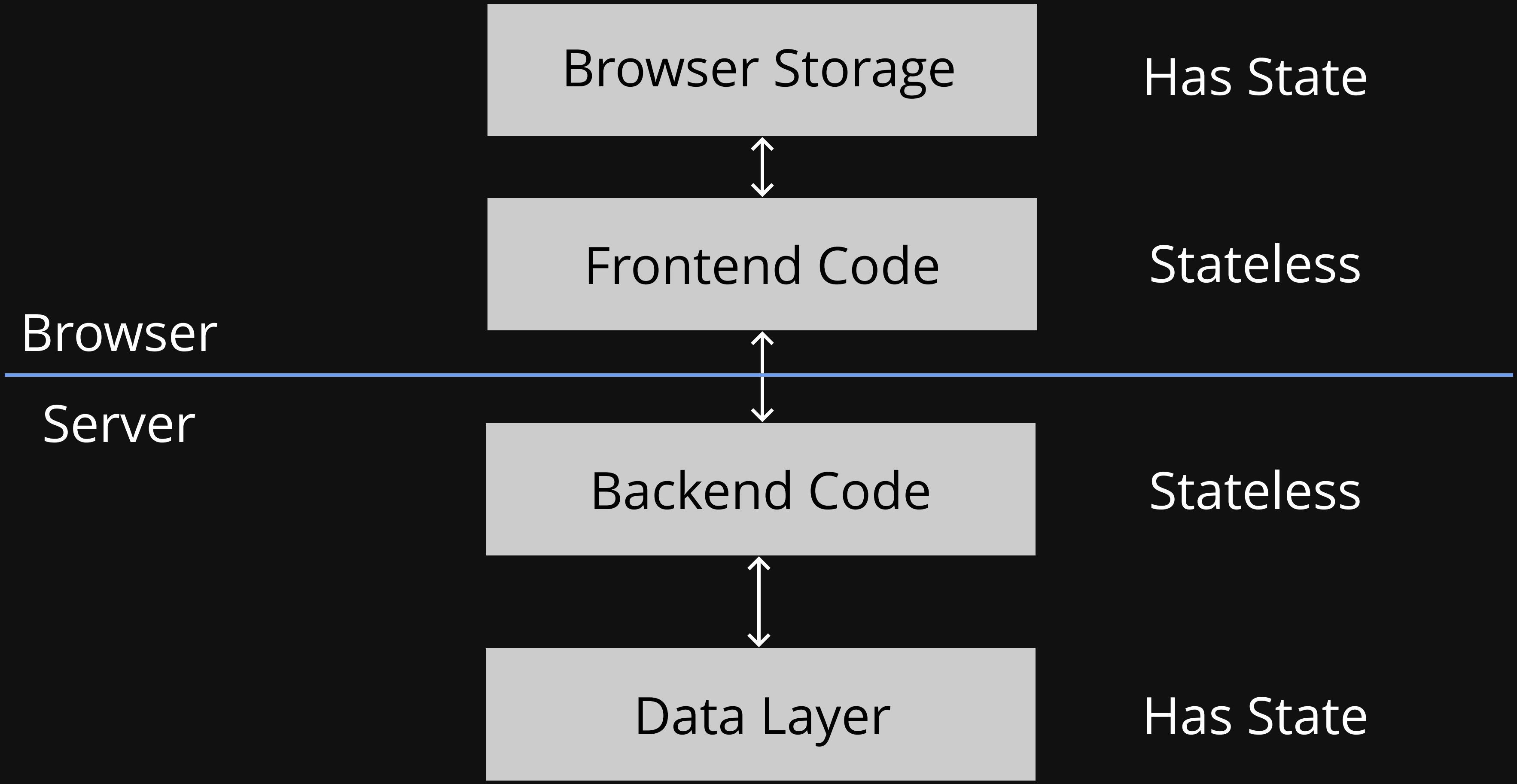
Data: Facts that can be recorded and have implicit meaning

Data is one of the fastest and most rapidly growing areas within software.

From **data (raw)** was can find **insights (information)** that allow us to make **decisions**.

Data Layer

Data Layer: The part of the tech stack that provides persistence



Browser Storage — Has State

Frontend Code — Stateless

Browser

Server

Backend Code — Stateless

Data Layer — Has State

Databases

Data is only as powerful as you can store and access it. Study COMP3311 to learn more about efficient data storage.

There are 3 main ways to store data:

1. In-memory (non-persistent)
2. In-file
3. In-database (SQL)

As you move down the list, barrier to entry becomes higher, but so does performance.

In COMP1531 we will only explore (2)

Storing Data: Persistence

Persistence: When program state outlives the process that created it. This is achieved by storing the state as data in computer data storage

What is storage?

- CPU cache?
- RAM?
- Hard disk? (we usually mean this one)

Storing Data: Persistence

Most modern backend/server applications are just source code + data

Storing Data: In practice

A very common and popular method of storing data in python is to "pickle" the file.

- Pickling a file is a lot like creating a .zip file for a variable.
- This "variable" often consists of many nested data structures (a lot like your iteration 2 data)

Storing Data: In practice

Let's look at an example

pickle_it.py

unpickle_it.py

Standard Interfaces

In any field in engineering, we often have systems, components, and designs built by different parties for different purposes.

How do all of these systems connect together? Through standard interfaces

Standard Interfaces

Data Interchange Formats

When it comes to **transferring data**, we also need common interface(s) that people all send or store data in universal ways to be shared between applications or shared over networks.

Three main interchange formats we will talk about:

- JSON
- YAML
- XML

Data Interchange Formats

When it comes to **transferring data**, we also need common interface(s) that people all send or store data in universal ways to be shared between applications or shared over networks.

Three main interchange formats we will talk about:

- JSON
- YAML
- XML

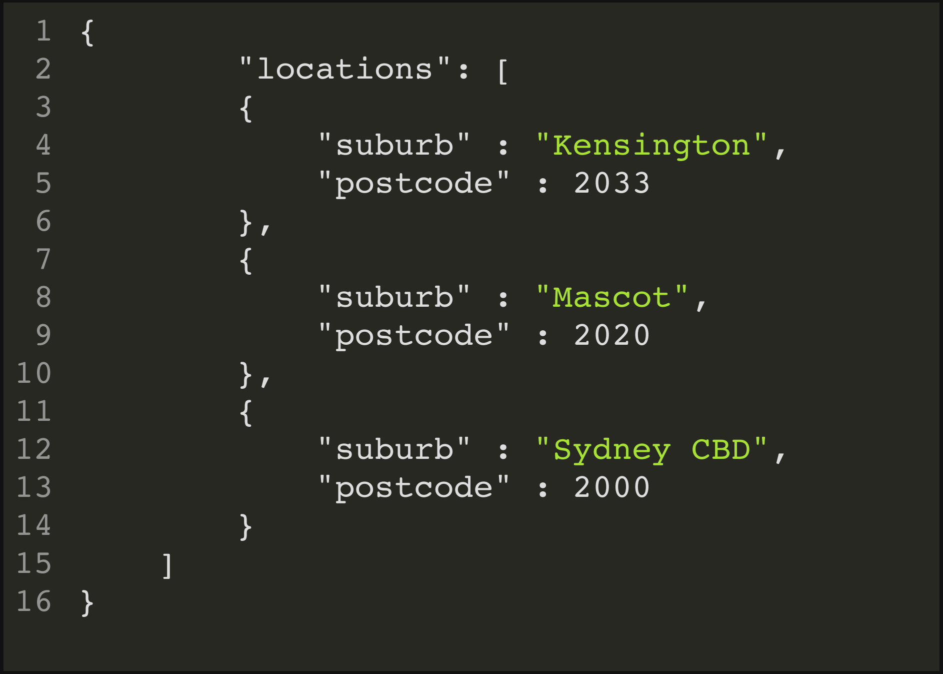
JSON

JavaScript Object Notation (JSON) - TFC 7159

A format made up of braces for dictionaries, square brackets for lists, where all non-numeric items must be wrapped in quotations. Very similar to python data structures.

JSON

Let's represent a structure that contains a list of locations, where each location has a suburb and postcode:



```
{
        "locations": [
        {
            "suburb" : "Kensington",
            "postcode" : 2033
        },
        {
            "suburb" : "Mascot",
            "postcode" : 2020
        },
        {
            "suburb" : "Sydney CBD",
            "postcode" : 2000
        }
    ]
}
```

Note:

- No trailing commas allowed
- Whitespace is ignored

JSON - Writing & Reading

Python has powerful built in libraries to write and read json.

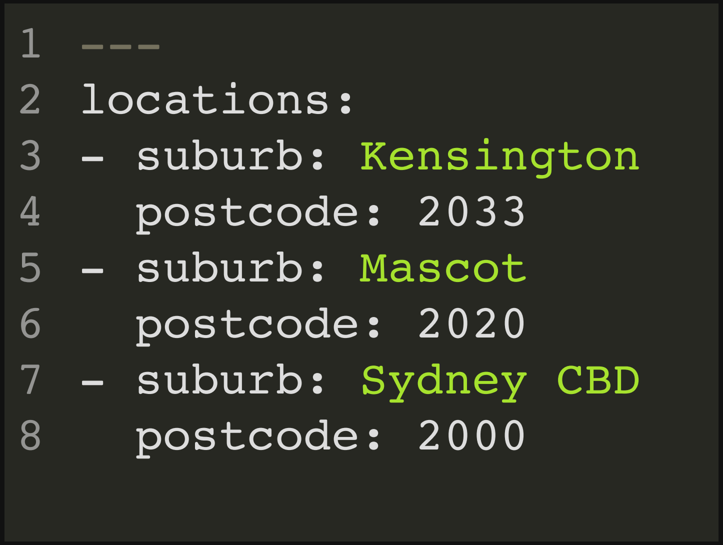
This involves converting JSON between a python-readable data structure, and a text-based dump of JSON

json_it.py

unjson_it.py

YAML

YAML Ain't Markup Language (YAML) is a popular modern interchange format due it's ease of editing and concise nature



```yaml
---
locations:
- suburb: Kensington
  postcode: 2033
- suburb: Mascot
  postcode: 2020
- suburb: Sydney CBD
  postcode: 2000
```

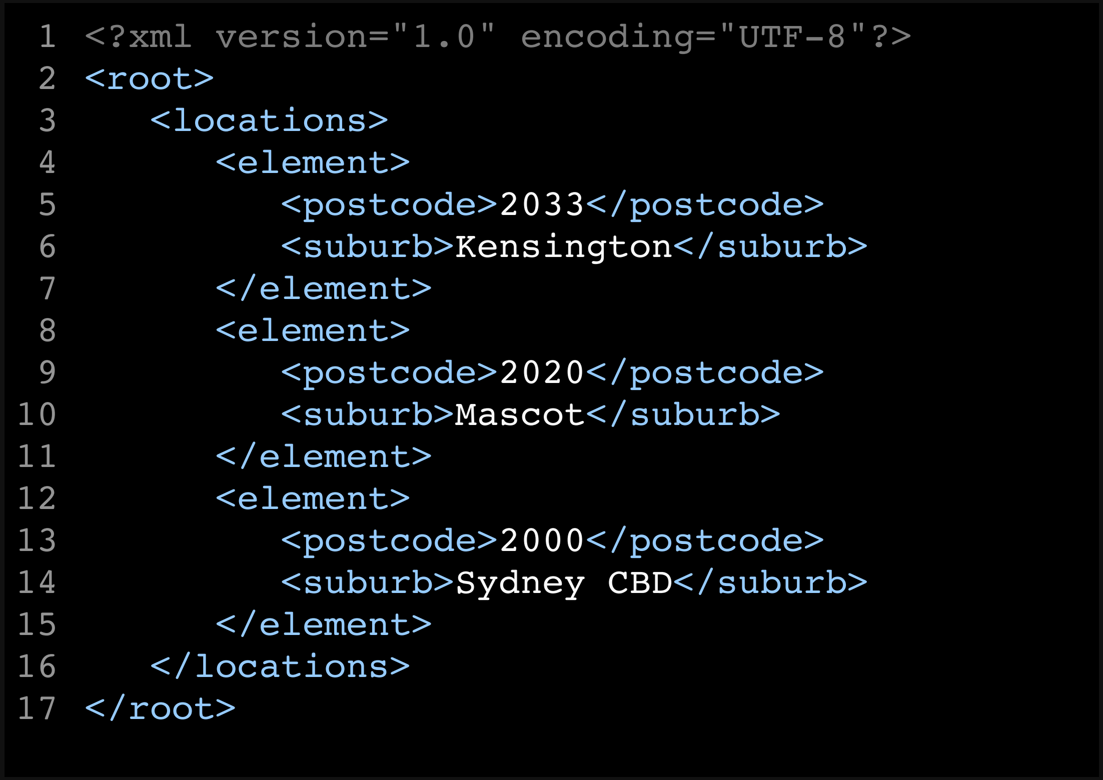
Same example from previous slide

Note:

- Like python, indentation matters
- A dash is used to begin a list item
- very common for configuration(s)

XML

eXtensible Markup Language (XML) is more of a legacy interchange format being used less and less



```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <locations>
      <element>
         <postcode>2033</postcode>
         <suburb>Kensington</suburb>
      </element>
      <element>
         <postcode>2020</postcode>
         <suburb>Mascot</suburb>
      </element>
      <element>
         <postcode>2000</postcode>
         <suburb>Sydney CBD</suburb>
      </element>
   </locations>
</root>
```

XML

Issues with XML include:

- More verbose (harder to read at a glance)
- More demanding to process/interpret
- More bytes required to store (due to open/closing tags)

While you will find very few modern applications choose to use XML as an interchange format, many legacy systems will still use XML as a means of storing data