COMP1531

4.1 - SDLC Development - Data Transfer

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

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. It's easy to convert between JSON and YAML online.

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

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