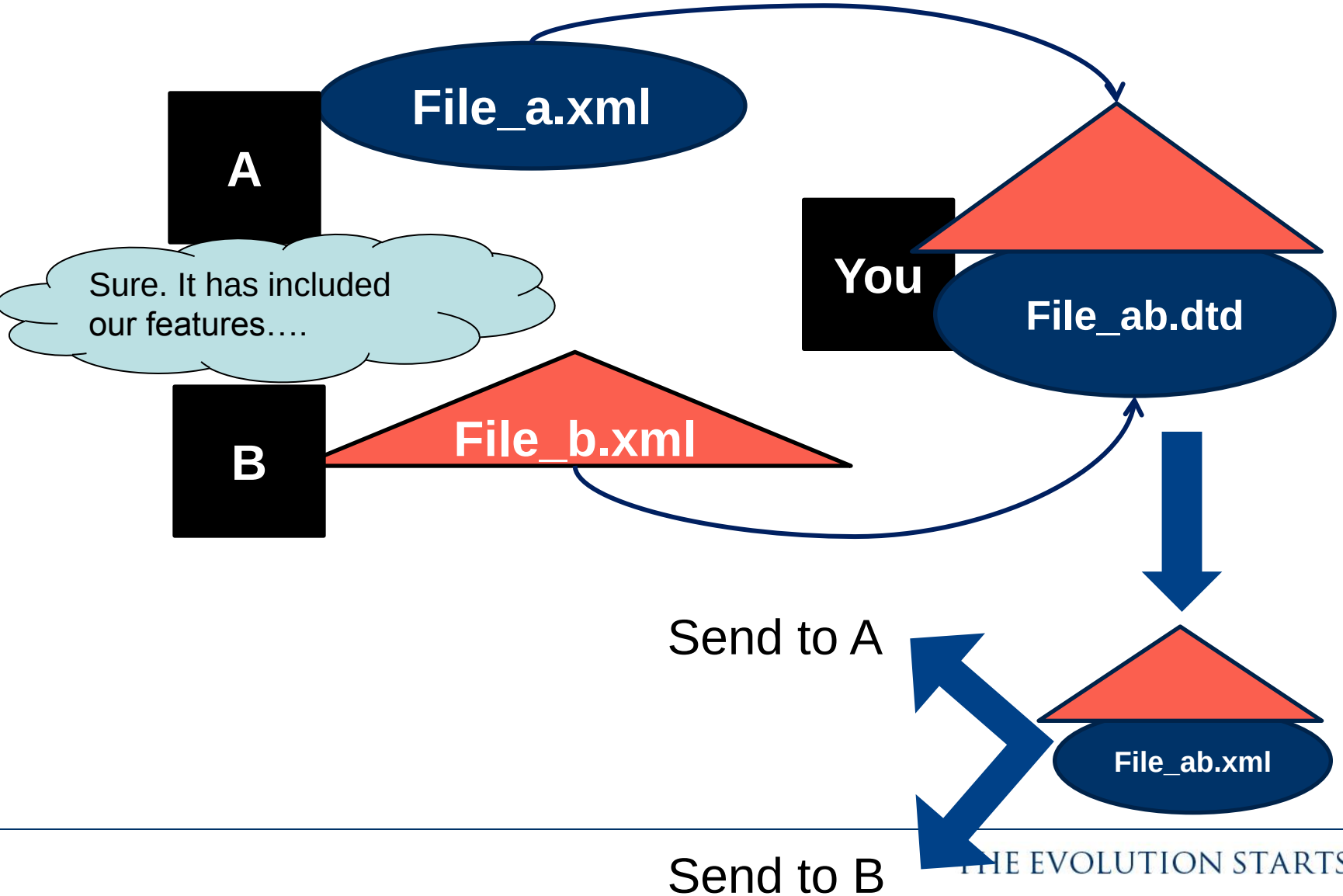
My email address:
ylu11@unimelb.edu.au

Document type definition (DTD)



Exercise 5: DTD validation

- **Exercise 5**
- 1. Validate your **simplebook.xml** against the DTD, which is already part of the file. Use the following W3C online validator: http://validator.w3.org/#validate_by_input

- 2. Now modify simplebook.xml by changing the <writer> tag to <reader>, and revalidate the XML. Inspect the response given by the validator.

```
<!ELEMENT book (writer,title)>
<!ELEMENT writer (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ATTLIST book id CDATA #REQUIRED>
```

Exercise 6: DTD and XML Design

- Back to the XML data: **book.xml** and **book2.xml**
  1. Design a new DTD for describing the new structure by reconciling two XML formats.
  2. Rewrite book.xml and book2.xml based on the new DTD. Called them **newbook.xml** and **newbook2.xml**.
  3. Include the DTD within newbook.xml and reference the DTD externally in newbook2.xml.
     - Hints: newbook2.xml needs a publicly accessible address to reference DTD (by IVLE)
       - –a. new a DTD file and put it into a folder (e.g. book)
       - –b. “publish” the folder
       - –c. “share” the DTD file and get the link
       - –d. use the link to reference
  4. Compare your DTD with the student next to you. Discuss how to redesign a third DTD to reconcile your two DTDs.

DTD Elements Rules (the second lecture)

Element Type	
(#PCDATA)	Parsed character (text) only
(child1, child2, child3, …)	Declare order of child elements
EMPTY	Element must be empty
(child1 | child2 | child3 |…)	Declare choice of child elements
ANY	Any type of content (no restrictions)

Symbol	
?	Allow zero or one of the item
+	Allow one or more of the item
*	Allow zero or more of the item

DTD Attribute Rules

Attribute Types	
CDATA	Character data ($ for instance)
(value1| value2 | …)	Allowed values (enumerated list)
ID	Used to create a unique identifier for an attribute
IDREF	Allows an attribute to **reference the ID attribute** from another element
NMTOKEN	A name token whose value is restricted to a valid XML name
ENTITY	A reference to an external file, usually one containing non-XML data

DTD Attribute Rules

Attribute default	
#REQUIRED	The **attribute must appear** with every occurrence of the element
#IMPLIED	Attribute is optional
“default”	Attribute is optional, but if not specified, XML parser supplies default value
#FIXED “default”	Attribute is optional, but if value is specified it must match the default value

Exercise 7: Advanced practice

```
<!DOCTYPE album
[
<!ELEMENT album (title,creator,release-date,publisher?,formats,award-name*,track-list)>

<!ELEMENT title (#PCDATA)>
<!ELEMENT creator (group-name|person-name+)>
<!ELEMENT release-date (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT formats (media+)>
<!ELEMENT award-name (#PCDATA)>
<!ELEMENT track-list (track+)>

<!ELEMENT group-name (#PCDATA)>
<!ELEMENT person-name (#PCDATA)>
<!ELEMENT media (#PCDATA)>
<!ELEMENT track (track-title,track-length)>
<!ELEMENT track-title (#PCDATA)>
<!ELEMENT track-length (#PCDATA)>

<!ATTLIST album uid CDATA #REQUIRED>
<!ATTLIST album language (english|french|german|klingon) "english">
<!ATTLIST creator type (group|person) #REQUIRED>
<!ATTLIST media media-count CDATA #IMPLIED>
<!ATTLIST track uid CDATA #REQUIRED>
]>
```

Create two XML files based on the DTD above to represent some information about the following two albums:

http://www.amazon.com/Walking-Dream-Empire-Sun/dp/B001GXPHX0/

http://www.amazon.com/Raising-Robert-Plant-Alison-Krauss/dp/B000UMQDHC/

Validate your XML to ensure that you apply the DTD correctly.

- Consider following questions:
  - –1. Can DTD guarantee there is no ambiguous data of a particular piece of information? Could there be multiple interpretations of a DTD?

    A: There are limitations in DTD. Even though structure of the XML can be embedded within DTD, people still need to make judgement on what the DTD means and how the entities within it relate to the real things that they want to represent.

  - –2. Can you modify the DTD (below) in order to ensure that if the type attribute of creator element is equal to "group", then the creator element can only contain “group-name” but not “person-name” ?
  - –A: There is no way to constraint existence of an element based on the attribute of the parent element.

Web Infrastructure and JSON

INFO20002: Foundations of Informatics

Web Infrastructure and JSON

- Outcomes:
  - Understand the infrastructure concepts, HTTP, HTML as well as the mechanism of information transferring across the network
  - Use “Flask” to develop simple web applications
  - JSON (JavaScript Object Notation)
    - Re-iterate the syntax rules
    - Validate (JSONLint)
    - HTML transforming – XML/HTML and JSON/HTML

- HTTP (hyper-text transfer protocol)
  - How the data is transferred over the World Wide Web.
- HTML (hyper-text markup language)
  - The markup language for describing web pages

echanic of

- Ope ct to
  www ds from
  Note returned.

PuTTY (inactive)

```
GET / HTTP/1.1
HOST: www.unimelb.edu.au
Connection: close;

HTTP/1.1 200 OK
Date: Sun, 13 Mar 2016 22:31:36 GMT
Server: Apache
Accept-Ranges: bytes
Cache-Control: max-age=0, must-revalidate
Expires: Sun, 13 Mar 2016 22:31:36 GMT
Vary: Accept-Encoding
Content-Length: 45154
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
                <html>
                        <head>
                                        <meta charset="utf-8">
                                                                <title>The University
 of Melbourne, Australia - Australia's best university and one of the world's fi
nest.</title>
                        <meta content="width=device-width, initial-scale=0.9" name="vie
wport">
```

  - a) Where does the response information start?
  - b) What server engine is hosting this site?
  - c) What HTTP code has been returned?
  - d) Try again with the input “GET /test.html HTTP/1.1” on the first line. What code is returned?

    - https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

- Compare the output from the Putty terminal and the output of the web-page in a browser. How do they differ?

- **Flask** allows you to develop a web application, which is basically a service responses to user interactively via a web browser.
  - A micro web framework written in Python
  - Has been included in **anaconda** 2

"Recipe" – hello.py

```
#import the Flask class
from flask import Flask
# create an instance of "Flask" class
app = Flask(__name__, static_folder='.', static_url_path='')
# route() decorator to tell app what URL can trigger functions.
@app.route("/")
#define a function which is to display the message on the browser
def root():
  return 'Hello World!'
# using run() to specify how to run the app
if __name__ == "__main__":
  app.run (debug=True, host='127.0.0.1', port=80)
```

Exercise 2 Make functions in hello.py

- Add another function under the address “localhost/main” to present “current time” (use strftime from time module).

Exercise 3 Make a Web Form

- Write a Python script that process the **book.xml** (downloadable) as a **web page.**
  - Hints:
    - 1. transform XML into HTML

```
<table border="0" cellpadding="5" cellspacing="1">
<tr bgcolor="#CCCC99"><td><b>Author</b></td><td>Salinger, J. D.</td></tr>
<tr bgcolor="#CCCC99"><td><b>Title</b></td><td>The Catcher in the Rye</td></tr>
...
```

    - 2. Process the output HTML as a web page (see the example of “New York temperature”)

Author	Salinger, J. D.
Title	The Catcher in the Rye
Price	44.95
Language	English
Publish_date	1951-07-16
Publisher	Little, Brown and Company
Isbn	0-316-76953-3
Description	A story about a few important days in the life of Holden Caulfield

Exercise 4 – HTML tags

- Resource

```
<!doctype html>
<html>
<body>
<form method="get" action="handler">
  <p>Your name: <input type="text" name="name"/></p>
  <p>Gender:
    <input type="radio" name="gender" value="male"/>Male
    <input type="radio" name="gender" value="female"/> Female
  </p>
  <p>Interested in:
    <input type="checkbox" name="interests" value="men"/>Men
    <input type="checkbox" name="interests" value="women"/>Women
  </p>
  <p>Favourite car:
    <select name="car">
      <option value="volvo">Volvo</option>
      <option value="saab">Saab</option>
      <option value="mercedes">Mercedes</option>
      <option value="audi">Audi</option>
    </select>
  </p>
  <p>Description:
    <textarea name="description">Tell me more</textarea>
  </p>
  <p>
    <input type="submit" />
  </p>
</form>
</body>
</html>
```

```python
@app.route('/handler', methods=['POST', 'GET'])
def handler():
  body = '''Name: %s
Gender: %s
Interests: %s
Car: %s
Description: %s
'''

  if request.method == 'POST':
    name = request.form['name']
    gender = request.form['gender']
    interests = request.form.getlist('interests')
    car = request.form['car']
    description = request.form['description']
  else:
    name = request.args.get('name')
    gender = request.args.get('gender')
    interests = request.args.getlist('interests')
    car = request.args.get('car')
    description = request.args.get('description')

  body = body % (name, gender, interests, car, description)
  return body, 200, {'Content-Type': 'text/plain'}
```

```
<form method="get" action="handler">
```

Exercise 4 – HTTP method – GET and POST

```
127.0.0.1 - - [13/Mar/2016 19:31:15] "GET /form.html HTTP/1.1" 200 -
127.0.0.1 - - [13/Mar/2016 19:31:17] "GET /form.html HTTP/1.1" 304 -
127.0.0.1 - - [13/Mar/2016 19:31:18] "GET /form.html HTTP/1.1" 304 -
127.0.0.1 - - [13/Mar/2016 19:31:24] "GET /handler?name=&car=volvo&description=T
ell+me+more HTTP/1.1" 200 -

C:\Users\luy4\Desktop\exercise1\form\codes>python form.py
 * Restarting with stat
 * Debugger is active!
 * Debugger pin code: 113-816-601
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
127.0.0.1 - - [13/Mar/2016 19:31:50] "GET /form.html HTTP/1.1" 200 -
127.0.0.1 - - [13/Mar/2016 19:31:52] "POST /handler HTTP/1.1" 400 -
127.0.0.1 - - [13/Mar/2016 19:32:06] "POST /handler HTTP/1.1" 200 -
```

Exercise 4 – HTTP method – GET and POST

- **GET:** Appends **form-data** into the URL in name/value pairs
  - The length of a URL is limited (about 3000 characters)
  - Never use GET to send sensitive data! (will be visible in the URL)
  - Useful for form submissions where a user want to bookmark the result
  - GET is better for non-secure data, like **query strings** in Google

- **POST:** Appends form-data inside the body of the HTTP request (data is not shown is in URL)
  - Has no limitation on the size
  - Form submissions with POST cannot be bookmarked (form parameters)

JSON Workshop

INFO20002: Foundations of Informatics

XML Example

```
<?xml version="1.0" encoding="utf-8"?>
  <book id="book001">
    <author>Salinger, J. D.</author>
    <title>The Catcher in the Rye</title>
    <price>44.95</price>
    <language>English</language>
    <publish_date>1951-07-16</publish_date>
    <publisher>Little, Brown and Company</publisher>
    <isbn>0-316-76953-3</isbn>
    <description>A story about a few important days
in the life of Holden Caulfield</description>
  </book>
```

JSON Example

```
{
  "firstName": “David",
  "lastName": “Lynn",
  "isAlive": true,
  "age": 25,
  "height_cm": 167.6,
  "address": {
      "streetAddress": "211 Fox Street",
      "city": “Greenville",
      "state": "NH",
      "postalCode": “80021“
  }
}
```

JSON syntax rules

- JSON **Objects**

```
{"firstName":"John", "lastName":"Doe"}
```

  - It is organised in name/value pairs

```
"firstName": "John"
```

- JSON **Arrays** (values)

```
{"employees":
   [
     {"firstName":"John", "lastName":"Doe"},
     {"firstName":"Anna", "lastName":"Smith"},
     {"firstName":"Peter", "lastName":"Jones"}
    ]
}
```

- Objects can repeat recursively down a hierarchy as needed.

JSON syntax rules

- JSON value types:
  - A number (integer or floating point)
  - A string (in double quotes)
  - A boolean (true or false)
  - An array (in square brackets)
  - An object (in curly braces)
  - null

Comparison with XML

- Similarities:
  - They are both human-readable formats
  - Extensible

- Differences:
  - XML is more **verbose** than JSON, so it is faster to write JSON for programmers/ lightweight
  - XML is used to describe the structured data, which doesn't include **arrays** whereas JSON does.

Exercise 1 Validate JSON

- Represent book.xml as JSON book.json. Begin with the following text and expand from there:

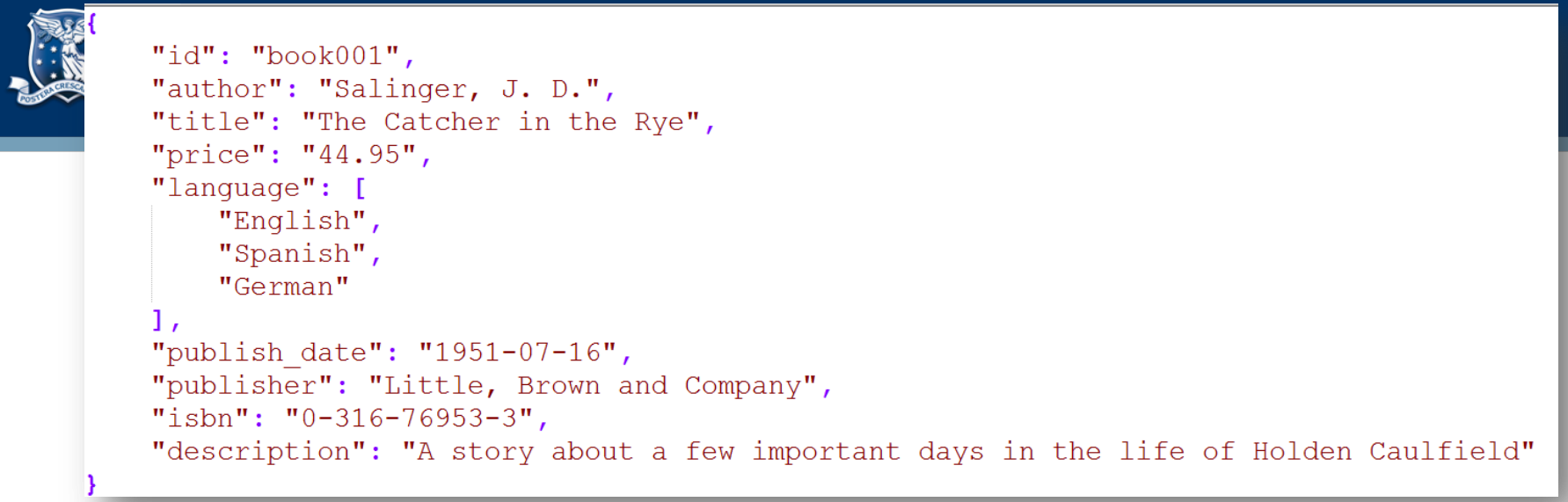
```
{
  "id": "book001",
  "author": "Salinger, J. D."
}
```

- Validate your JSON solution against JSONLint. [Note: as with the XML validator, be careful of **white-space** or missing control **characters**]

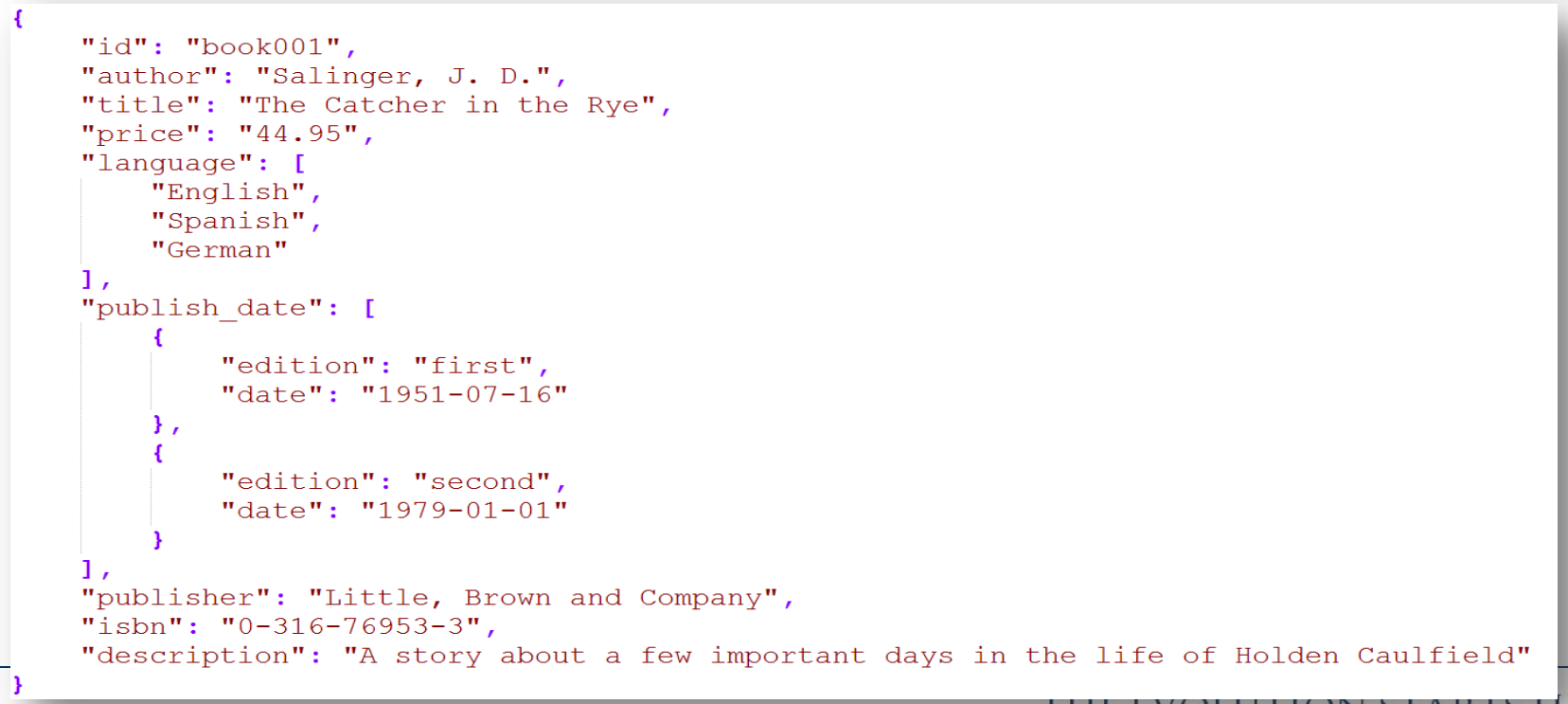
Exercise 2 Edit JSON

- 1. In your JSON solution, add **Spanish** and **German** as two extra languages represented as an array. Save this file as book2.json. Validate it on JSONLint.

- 2. Now modify the **publish_date** parameter. Make this an array of two objects that have properties of edition (first, second) and date (1951-07-16,1979-01-01) respectively. Save this file as book3.json. Validate it on JSONLint.



```
{
    "id": "book001",
    "author": "Salinger, J. D.",
    "title": "The Catcher in the Rye",
    "price": "44.95",
    "language": [
        "English",
        "Spanish",
        "German"
    ],
    "publish_date": "1951-07-16",
    "publisher": "Little, Brown and Company",
    "isbn": "0-316-76953-3",
    "description": "A story about a few important days in the life of Holden Caulfield"
}
```



```
{
    "id": "book001",
    "author": "Salinger, J. D.",
    "title": "The Catcher in the Rye",
    "price": "44.95",
    "language": [
        "English",
        "Spanish",
        "German"
    ],
    "publish_date": [
        {
            "edition": "first",
            "date": "1951-07-16"
        },
        {
            "edition": "second",
            "date": "1979-01-01"
        }
    ],
    "publisher": "Little, Brown and Company",
    "isbn": "0-316-76953-3",
    "description": "A story about a few important days in the life of Holden Caulfield"
}
```

Exercise 3 Web application based on JOSN

- Now create a python code that represents the HTML table (seen before earlier in the workshop) by using json library. The library allows you to read JSON file and access its content as dictionary-like object in Python. You can simply make your code to write an HTML page or you can use Flask to produce an application that write HTML from the JSON file dynamically.

```
import json

json_obj = json.load(open('book.json'))

# write your Python code from this point
# ...
```

```
import json

def json2html():
  json_obj = json.load(open('book.json'))

  str = '<html><body><table border="0" cellpadding="5" cellspacing="1" bgcolor="#CC9933">'
  for entry in json_obj:
    str += '<tr><td>' + entry + '</td><td>' +  json_obj[entry] + '</td></tr>'
  str += '</table></body></html>'
  return str


from flask import Flask
app = Flask(__name__, static_folder='.', static_url_path='')

@app.route("/")
def main():
    return json2html() # process json data and present it to the website

if __name__ == "__main__":
    app.run(debug=True, host = '127.0.0.1', port = 80)
```

THE UNIVERSITY OF

MELBOURNE