Machine Learning for IoT
Lab 5 – Communication

<u>HowTo 1:</u> JSON in Python

JSON (JavaScript Object Notation) is a lightweight data-interchange format that is used for exchanging data between different systems. JSON is commonly used for transmitting data over a network connection (e.g., between a server and a web application) and for storing data in a file or database.
JSON is a widely used format because it is easy to read and write, and it is supported by many programming languages and tools. For example, you can use the *json* module in Python to encode and decode JSON data.

`json.loads(json_string):` convert a JSON string to a Python object

```
string = '{"name": "Tony", "surname": "Stark"}'
obj = json.loads(string)
```

`json.dumps(object):` convert a Python object to a JSON string

```
obj = {"num1": 12,"num2": 34}
string = json.dumps(obj)
```

<u>HowTo 2:</u> Setup REST Client in VSCode

1. Open VSCode and install *REST Client* extension by Huachao Mao.
2. Create a new file and save it with the *.http* extension.
3. In the file, specify the base URL of the API and the endpoint that you want to test. For example:

```
GET https://jsonplaceholder.typicode.com/users
```

<u>Exercise 1:</u> MQTT in Python with PAHO

1.1 Develop an MQTT publisher to send:

- every 5 seconds the date and time in the format dd-mm-yyyy hh:mm:ss.
- every 10 seconds the timestamp.

Use hierarchical topics for the two messages, with your student ID as the first topic level (e.g., *s001122/datetime* and *s001122/timestamp*)
Use the message broker provided by eclipse (mqtt.eclipseprojects.io at port 1883).

1.2 In VSCode, develop a first MQTT subscriber to receive only the messages about date and time and print the information in a user-friendly format.

1.3 In Deepnote, develop a second MQTT subscriber to receive only the messages about the timestamp and print the information in a user-friendly format.

Exercise 2: REST Client with VSCode

In VSCode, create a new file named *lab5_ex2.http* to develop a REST Client that retrieve information related to cryptocurrencies, such as prices, market data, trading information, and more from a third-party service, the Coinbase Data API (see the documentation: https://docs.cloud.coinbase.com/sign-in-with-coinbase/docs/api-currencies).

In the file, define HTTP requests to retrieve the following information:

- Get a list of all available currencies.
- Get the exchange rates for the EUR currency.
- Get the price in EUR to buy one bitcoin (BTC).
- Get the price in EUR to sell one bitcoin (BTC).

Exercise 3: REST Client with Python

In VSCode, write a Python script named *lab5_ex3.py* to retrieve information from the Coinbase Data API using the *requests* package. The script should be run from the command line interface and should take as input a single argument called --*currency* that specifies the currency code.
Then, it should print the buy and sell price for one bitcoin in human readably format. If the currency does not exist, print an error message.

Suggestion: check the official Python documentation of the *requests* package at:
https://requests.readthedocs.io/en/latest/user/quickstart/#make-a-request

Example:
```
python lab5_ex3.py --currency EUR
```

Output:
```
The buy price in EUR for one BTC is: 16367.88
The sell price in EUR for one BTC is: 16203.09
```

Example:
```
python lab5_ex3.py --currency AAA
```

Output:
```
Error: AAA currency not found.
```

Exercise 4: REST Server with Cherrypy

4.1 Develop a to-do list application using *cherrypy* as web framework and Redis as database.
The to-do list application should provide a simple REST API for creating and managing tasks in a user's to-do list. The API enables the user to create tasks, retrieve a list of tasks, update tasks, and delete tasks. The API must be compliant with the following specifications:

RESOURCES

Resource: Item

Parameter	Description
id	string, identifier
message	string, description of the to-do item.
completed	boolean, a flag indicating whether the to-do item has been marked as completed (True) or not (False).

Example:
```
{
  id: "7556bd85-8ac4-4d18-88e1-b72d54c89298",
  message: "Submit Homework #2",
  completed: true
}
```

ENDPOINTS

Endpoint /online

- **GET /online**

  Description: Return the status of the REST Server.

  Path Parameters: N/A

  Query Parameters: N/A

  Response Status Code:
  - 200 – OK: Everything worked as expected.

  Response Schema:

Parameter	Description
status	string, equal to "online" if the REST Server is online.

  Response Example:
  ```
  {
    status: "online"
  }
  ```

Endpoint /todos

- **GET /todos**

  Description: Get the list of the to-do items. This endpoint allows you to search for items with a given text in the message and/or completed status.

  Path Parameters: N/A

Query Parameters:

Parameter	Description
message	string (optional), the text query to search for in the items message. If not provided, items with any message will be returned.
completed	boolean (optional), a flag indicating whether to return only completed items (true) or uncompleted items (false). If not provided, items with any completed value will be returned.

Response Status Code:

- 200 – OK: Everything worked as expected.

Response Schema: List of Item resources.

Response Example:
```
[
  {
    "message": "Do homework 3",
    "completed": false,
    "id": "70391b99-c9f3-4903-9043-00195d0c970b"
  },
  {
    "message": "Do homework 2",
    "completed": true,
    "id": "7556bd85-8ac4-4d18-88e1-b72d54c89298"
  }
]
```

- **POST /todos**

  Description: Add a new to-do item.

  Path Parameters: N/A

  Query Parameters: N/A

  Body Parameters:

Parameter	Description
message	string (required), description of the to-do item.

Response Status Code:

- 200 – OK: Everything worked as expected.

Response Schema: N/A

Endpoint /todo/{id}

- **GET /todo/{id}**
  Description: Retrieve the specified to-do item.

  Path Parameters:

Parameter	Description
id	string (required), the id of the item to retrieve

  Query Parameters: N/A

  Response Status Code:
  - 200 – OK: Everything worked as expected.
  - 400 – Bad Request: missing id.
  - 404 – Not Found: invalid id.

  Response Schema: Item

  Response Example:
  ```
  {
    "message": "Do homework 3",
    "completed": false,
    "id": "70391b99-c9f3-4903-9043-00195d0c970b"
  }
  ```

- **PUT /todo/{id}**
  Description: Update the specified to-do item.

  Path Parameters:

Parameter	Description
id	string (required), the id of the item to update

  Query Parameters: N/A

  Body Parameters:

Parameter	Description
message	string (required), description of the to-do item.
completed	boolean (required), a flag indicating whether to mark the item as completed (true) or not (false).

  Response Status Code:
  - 200 – OK: Everything worked as expected.
  - 400 – Bad Request: missing id.
  - 404 – Not Found: invalid id.

  Response Schema: N/A

- **DELETE /todo/{id}**
  Description: Delete the specified to-do item

  Path Parameters:

Parameter	Description
id	string (required), the id of the item to update

  Query Parameters: N/A

  Response Status Code:
    - 200 – OK: Everything worked as expected.
    - 400 – Bad Request: missing id.
    - 404 – Not Found: invalid id.

  Response Schema: N/A

4.2 Test the API with a VSCode Client. The test must sequentially execute the following actions:
  - Check the server status.
  - Add three to-do items.
  - Print the list of all to-do items.
  - Print the list of the *completed* to-do items.
  - Modify the *completed* flag of the second to-do item.
  - Delete the third to-do item.

4.3 Test the API with a Python Client. Repeat the same actions of 4.2.

4.4 Deploy the REST API on Deepnote and repeat the tests.
- In Deepnote, create a Python notebook. Copy the code of *Exercise 4.1* to the notebook.
- In the *Project* tab, go to *Environment*, then click on the wheel icon and enable “Allow incoming connections”. Copy the tunnelling link.
- Run the notebook.
- In the VSCode REST Client, replace the host with tunnelling link of Deepnote and repeat the tests.