IoT exercises – Week 7-8

For the following two weeks, please first finish the undone exercises for Week 6-7. Then we will start using the NodeMCU/ESP8266 to request information from Internet. Particularly, time will be required by using sntp and rtctime modules to update the local clock from Internet. With the local clock refreshed, the cron module can be used to replace the tmr in certain applications (eg. an alarm). The http and sjson modules will be introduced as the requesting tool and decoding tool for APIs provided by public websites for various data.

The official documentation is provided here again for your reference. Please remember, when you are building your own IoT project in the future, always refer to the bespoke built-in modules and read their documentation first, which will be helpful!!!

https://nodemcu.readthedocs.io/en/master/

The details of sntp, rtctime, cron, http and sjson module can be found in

https://nodemcu.readthedocs.io/en/master/modules/sntp/

https://nodemcu.readthedocs.io/en/master/modules/rtctime/

https://nodemcu.readthedocs.io/en/master/modules/cron/

https://nodemcu.readthedocs.io/en/master/modules/http/

https://nodemcu.readthedocs.io/en/master/modules/sjson/

Exercise 1:

In this exercise, find an NTP server yourself and use it for the NodeMCU/ESP8266 synchronization, where **sntp** will connect to the NTP server and synchronize the local clock and **rtctime** will convert the system clock to a format of stamp time.

==========

```
tmsrv = "uk.pool.ntp.org"

--an optional NTP server

function stampTime()

--get the stamp of the time from synchronized clock
```

```
sec,microsec,rate = rtctime.get()

tm = rtctime.epoch2cal(sec,microsec,rate)

print(string.format("%04d/%02d/%02d %02d:%02d:%02d", tm["year"], tm["mon"], tm["day"], tm["hour"], tm["min"], tm["sec"]))

end

sntp.sync(tmsrv,function()

    print(“Sync succeeded”)

    mytimer:stop()

    stampTime()

    end,function()

    print(“Synchronization failed!”)

    end, 1

)
```

==========

Exercise 2:

In this exercise, the synchronized local clock will be used together with the **cron** module as a job scheduler to replace the **tmr** for certain applications.

==========

```
--cron.schedule()

--refer to https://en.wikipedia.org/wiki/Cron

--for the mask "* * * * *" information

--minute, hour, day of a month, month of a year, day of the week

cron.schedule("* * * * *", function(e)

    print("For every minute function will be executed once")

end)

cron.schedule("*/5 * * * *", function(e)

    print("For every 5 minutes function will be executed once")

end)
```

```
cron.schedule("0 7 * * *", function(e)
    print("\n Alarm Clock \n It is 07:00!!! \n Get UP! \n")
end)
--set your own alarm at 07:00
--please change it to 2 minutes later from now on
--and see if it will alarm you according to the synchronized time.
```

==========

Optional: for more details about cron scheduler, refer to https://en.wikipedia.org/wiki/Cron

Exercise 3:

Now you have the clock your NodeMCU/ESP8266 synchronized with Internet. Now we look at the requests for more general data from the API of public websites.

- In most cases, the returned data are in json format, decoding is required after the successful requests. Particularly we can use the **sjson** module to concert the json to a table and then filter the data.
- Let's take the current weather data of Portsmouth from Open Weather Map (https://openweathermap.org/) as an example. Please find the address of the API to weather data first (https://openweathermap.org/current).
- Read about the API description. Know how to call them to return a json response.
- Then the following code might be helpful for you to read and filter the data.

==========

```
keyAPI = "9f418d12e2c0f5e721197b55a91010a3"
--This free API key allows no more than 1 request per second
--You can register your own if it fails which is free - Dalin
lat =
lon =
--Using the longitude and latitude of Portsmouth to retrieve weather data
urlAPI = "http://api.openweathermap.org....."
http.get(urlAPI, nil, function(code, data)
    if (code < 0) then
```

```
--200 is success code
--for the rest, refer to https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
            print("http request failed")
        else
            print (code)
            print (data)
        end
end)
```

==========

- Now further use the **sjson** module to convert the json to readable data for display.

https://nodemcu.readthedocs.io/en/master/modules/sjson/

Exercise 4:

Now please integrate the **sntp** and **rtctime** based Internet time synchronizer, the **cron** based scheduler, and the **http** and **sjson** based website API reader together. Create your Digital Alarm Clock with Weather Function.

Now you have implemented the requests to Internet for Time and Data through a bespoke module using their IP address and API.

(Optional) Exercise 5:

In this exercise, you are expected to integrate the previous exercises and create a platform which can retrieve time/weather with alarm function. The info can be updated and checked in your browser in LAN. The button based ON/OFF control and photoresistor + pot resistor + ADC based adjustable control can be included to switch between functions or setting parameters. As a server or a client, the messages, system info can be transmitted with the tester that you have used in Week 6-7.