LST SITD
Dept GE&GM
FST SETTAT
Pr : Karim TAHIRY

TP IoT

Introduction

The best way to develop quickly an IoT application with less Integrated circuits to add is to choose this circuit "NodeMCU". Today,we will give a detailed Introduction on NodeMCU V3. It is an open-source firmware and development kit that plays a vital role in designing a proper IoT product using a few script lines.

The module is mainly based on ESP8266 that is a low-cost Wi-Fi microchip incorporating both a full TCP/IP stack and microcontroller capability. It is introduced by manufacturer Espressif Systems. The ESP8266 NodeMcu is a complex device, which combines some features of the ordinary Arduino board with the possibility of connecting to the internet.
Arduino Modules and Microcontrollers have always been a great choice to incorporate automation into the relevant project. But these modules come with a little drawback as they don't feature a built-in WiFi capability, subsequently, we need to add external WiFi protocol into these devices to make themcompatible with the internet channel.

This is the famous NodeMCU which is based on ESP8266 WiFi SoC. This is version 3 and it is based on ESP-12E (An ESP8266 based WiFi module). NodeMCU is also an open-source firmware and development kit that helps you to prototype your IOT product within a few LUA script lines, and of course you can always program it with Arduino IDE.
In this article, We will try present useful details related to this WiFi Development Kit, its main features, pinout and everything we need to know about this module and the application domain.

NodeMCU V3

NodeMCU V3 is an open-source firmware and development kit that plays a vital role in designing an IoT product using a few script lines.

Multiple GPIO pins on the board allow us to connect the board with other peripherals and are capable of generating PWM, I2C, SPI, and UART serial communications.

- The interface of the module is mainly divided into two parts including both Firmware and Hardware where former runs on the ESP8266 Wi-Fi SoC and later is based on the ESP-12 module.

The firmware is based on Lua – A scripting language that is easy to learn, giving
a simple programming environment layered with a fast scripting language that connects you with a well-known developer community.

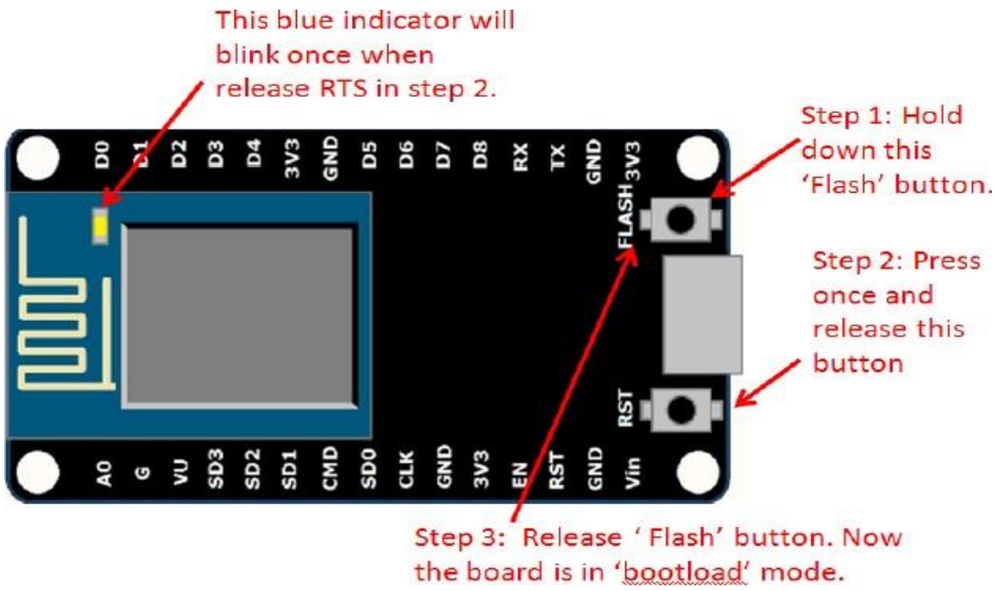


Figure 1

And open-source firmware gives you the flexibility to edit, modify and rebuilt the existing module and keep changing the entire interface until you succeed in optimizing the module as per your requirements.

- USB to UART converter is added on the module that helps in converting USB data to UART data which mainly understands the language of serial communication.

Instead of the regular USB port, MicroUSB port is included in the module that connects it with the computer for dual purposes: programming and powering up the board.

- The board incorporates status LED that blinks and turns off immediately, giving you the current status of the module if it is running properly when connected with the computer.

The ability of module to establish a flawless WiFi connection between two channels make it an ideal choice for incorporating it with other embedded devices like Raspberry Pi.

NodeMCU V3 Pinout

NodeMCU V3 comes with a number of GPIO Pins. Following figure shows the Pinout of the board.

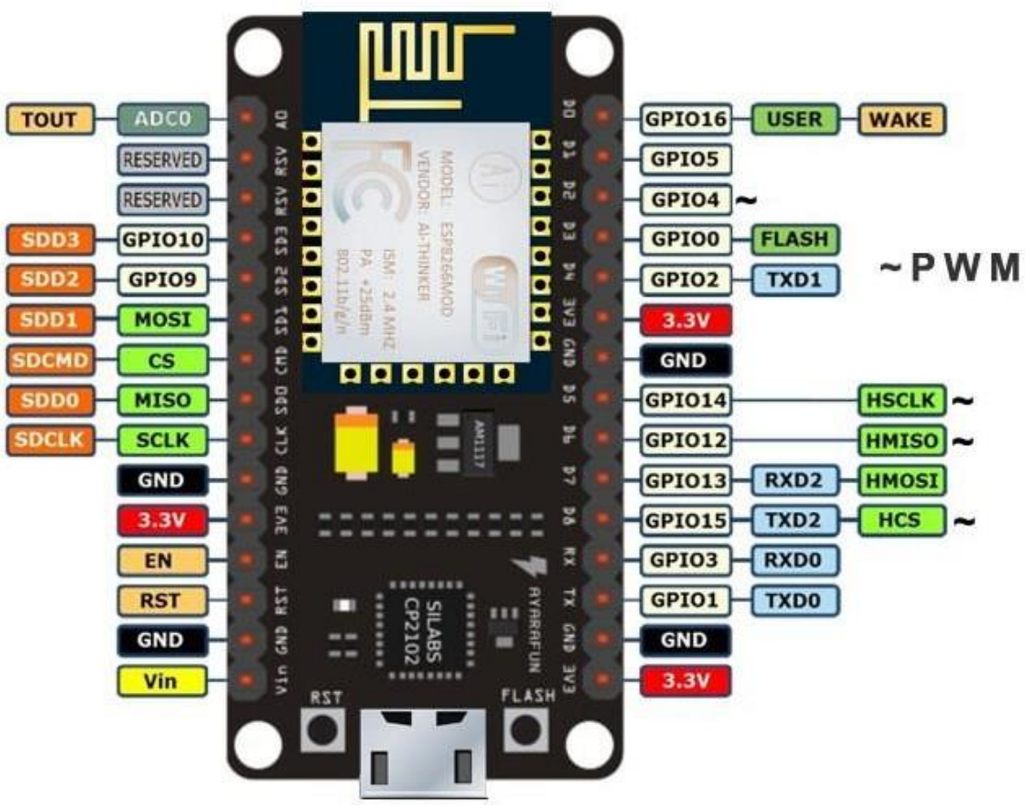


Figure 2

- There is a candid difference between Vin and VU where former is the regulated voltage that may stand somewhere between 7 to 12 V while later is the power voltage for USB that must be kept around 5 V.

Features

1. Open-source
2. Arduino-like hardware
3. Status LED
4. MicroUSB port
5. Reset/Flash buttons
6. Interactive and Programmable
7. Low cost
8. ESP8266 with inbuilt wifi
9. USB to UART converter
10. GPIO pins
11. Arduino-like hardware IO
12. Advanced API for hardware IO, which can dramatically reduce the redundant work for configuring and manipulating hardware.
13. Code like arduino, but interactively in Lua script.
14. Nodejs style network API
15. Event-driven API for network applicaitons, which faciliates developers writing code running on a 5mm*5mm sized MCU in Nodejs style.

16. Greatly speed up your IOT application developing process.
17. Lowest cost WI-FI
18. Less than $2 WI-FI MCU ESP8266 integrated and esay to prototyping development kit.
19. We provide the best platform for IOT application development at the lowest cost.

As mentioned above, a cable supporting micro-USB port is used to connect the board. As you connect the board with a computer, LED will flash. You may need some drivers to be installed on your computer if it fails to detect the NodeMCU board. You can download the driver from this page.

Note: We use Arduino IDE software for programming this module. It is important to note that the pin configuration appearing on the board is different from the configuration we use to program the board on the software i.e. when we write code for targeting pin 16 on the Arduino IDE, it will actually help is laying out the communication with the D0 pin on the module.
Following figure the shows the pin configuration to use in Arduino IDE.

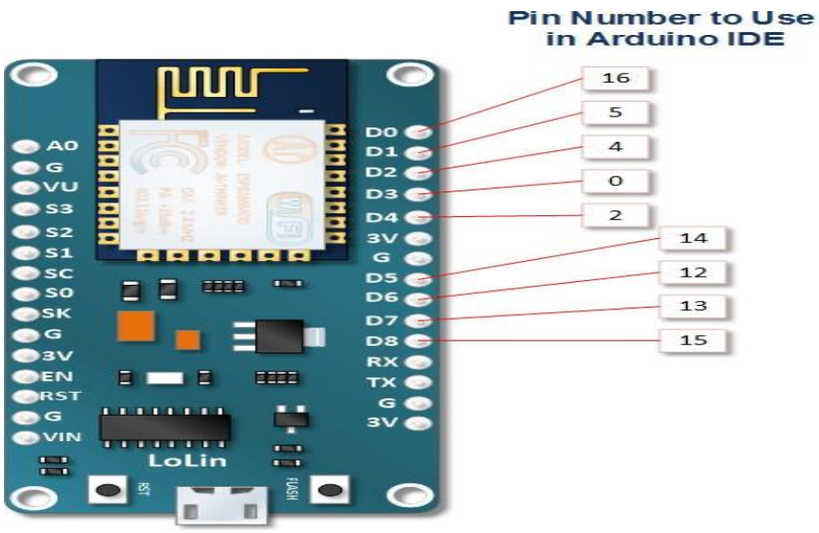


Figure 3

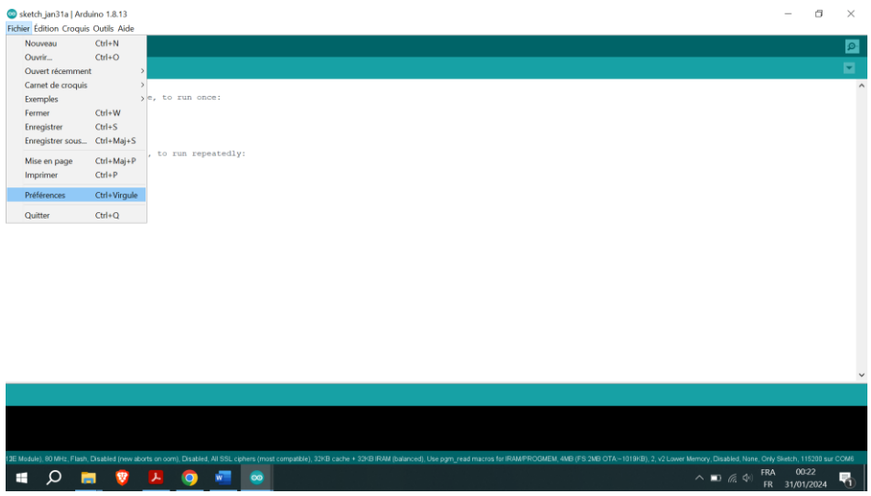
How to Power NodeMCU V3

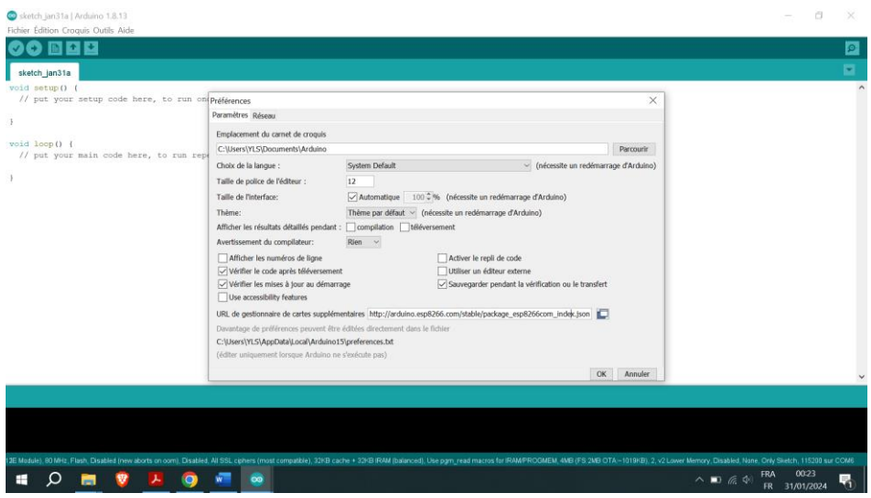
We can see from the pinout image above, there are five ground pins and three 3V3 pins on the board. The board can be powered up using the following three ways.

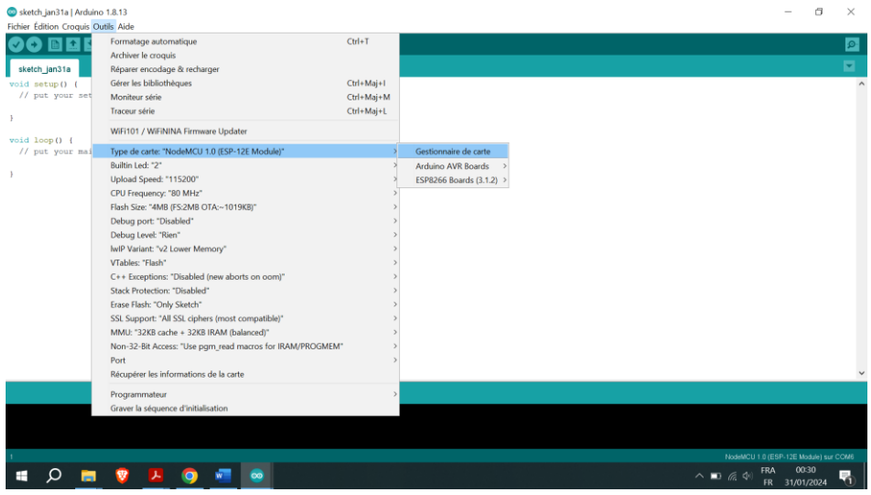
USB Power. It proves to an ideal choice for loading programs unless the project you aim to design requires separate interface i.e. disconnected from the computer.
Provide 3.3V. This is another great option to power up the module. If you have your own off-board regulator, you can generate an instant power source for your development kit.
Power Vin. This is a voltage regulator that comes with the ability to support up to 800 mA. It can handle somewhere between 7 to 12 V. You cannot power the devices operating at 3.3 V, as this regulator unable to generate as low as 3.3V.

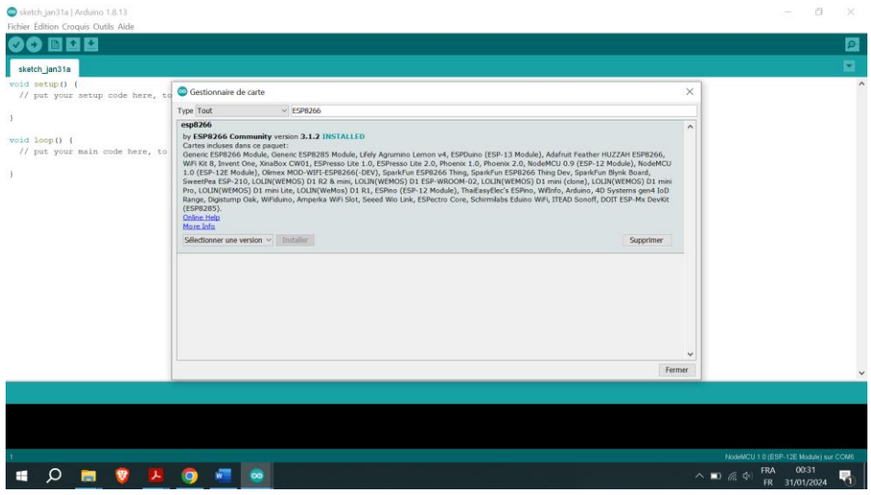
Step 1: Preparing the Arduino IDE

1. Install the Arduino IDE
2. open the Arduino IDE from the desktop icon
3. Click on File tab and then open preferences
4. In the additional Boards Manager URLs add the following link (https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json,http://arduino.esp8266.com/stable/package_esp8266com_index.json) and click OK
5. Goto Tools>Borads>Boards Manager
6. In the search field type esp8266 click the esp8266 by ESP8266 Community option and click Install





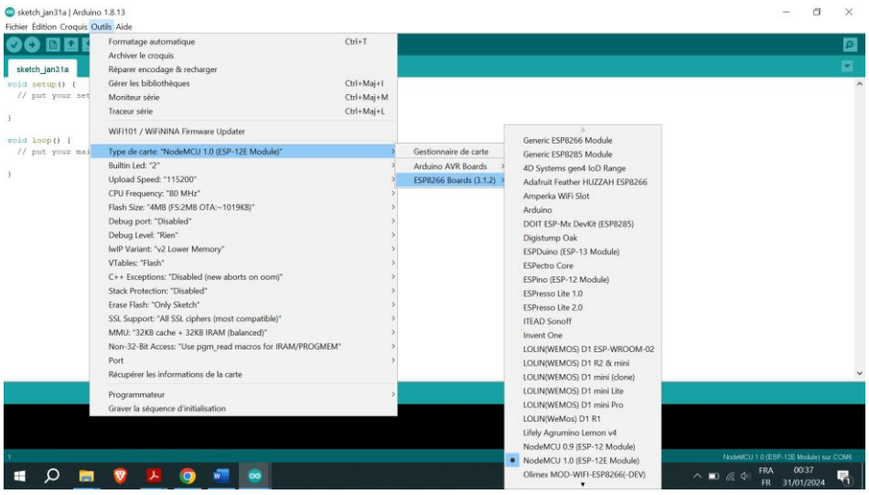

sketch_jan31a | Arduino 1.8.13
Fichier Édition Croquis Outils Aide
sketch_jan31a
Formatage automatique Ctrl+T
Archiver le croquis
Réparer encodage & recharger
Gérer les bibliothèques Ctrl+Maj+I
Moniteur série Ctrl+Maj+M
Traceur série Ctrl+Maj+L
WiFi101 / WiFiNINA Firmware Updater
Type de carte: "NodeMCU 1.0 (ESP-12E Module)"
Builtin Led: "2"
Upload Speed: "115200"
CPU Frequency: "80 MHz"
Flash Size: "4MB (FS:2MB OTA:~1019KB)"
Debug port: "Disabled"
Debug Level: "Rien"
lwIP Variant: "v2 Lower Memory"
VTables: "Flash"
C++ Exceptions: "Disabled (new aborts on oom)"
Stack Protection: "Disabled"
Erase Flash: "Only Sketch"
SSL Support: "All SSL ciphers (most compatible)"
MMU: "32KB cache + 32KB IRAM (balanced)"
Non-32-Bit Access: "Use pgm_read macros for IRAM/PROGMEM"
Port
Récupérer les informations de la carte
Programmateur
Graver la séquence d'initialisation
Gestionnaire de carte
Arduino AVR Boards
ESP8266 Boards (3.1.2)
NodeMCU 1.0 (ESP-12E Module) sur COM6
FRA
FR
00:30
31/01/2024


sketch_jan31a | Arduino 1.8.13
Fichier Édition Croquis Outils Aide
sketch_jan31a
void setup() {
// put your setup code here, to
}
void loop() {
// put your main code here, to
}
Gestionnaire de carte
Type Tout
ESP8266
esp8266
by ESP8266 Community version 3.1.2 INSTALLED
Cartes incluses dans ce paquet:
Generic ESP8266 Module, Generic ESP8285 Module, Lifely Agrumino Lemon v4, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, WiFi Kit 8, Invent One, XinaBox CW01, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SparkFun Blynk Board, SweetPea ESP-210, LOLIN(WEMOS) D1 R2 & mini, LOLIN(WEMOS) D1 ESP-WROOM-02, LOLIN(WEMOS) D1 mini (clone), LOLIN(WEMOS) D1 mini Pro, LOLIN(WEMOS) D1 mini Lite, LOLIN(WeMos) D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WiInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak, WiFiduino, Amperka WiFi Slot, Seeed Wio Link, ESPectro Core, Schirmlabs Eduino WiFi, ITEAD Sonoff, DOIT ESP-Mx DevKit (ESP8285).
Online Help
More Info
Sélectionner une version
Installer
Supprimer
Fermer
NodeMCU 1.0 (ESP-12E Module) sur COM6
FRA
FR
00:31
31/01/2024

Step 2: Code…

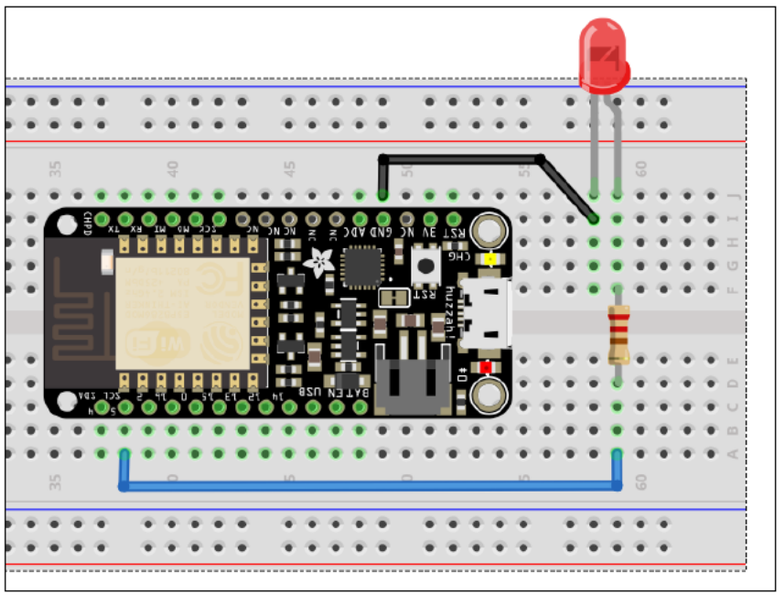
Now we can do whatever you want with your NodeMCU board

- In arduino IDE go to tools>Boards>select NODEMCU 1.0 (ESP - 12E Module)
- Again, go to tools and select port.
- Now click on Upload button to upload the following code.



TP0

L'objectif de ce code est de faire clignoter une LED connectée à une carte Arduino (ou compatible) à un intervalle régulier. La LED s'allume pendant une seconde, puis s'éteint pendant une seconde, et ce cycle se répète indéfiniment.



Code

```
// LED pin
int ledPin = 5;
void setup() {
pinMode(ledPin, OUTPUT);
}
void loop() {
// ON
digitalWrite (ledPin, HIGH);
delay(1000);
// OFF
digitalWrite (ledPin, LOW);
delay(1000);
}
```

Explications

int ledPin = 5;

Cette ligne déclare une variable entière nommée ledPin et lui attribue la valeur 5. Cette variable représente le numéro de la broche numérique à laquelle la LED est connectée. Ici, la LED est connectée à la broche numérique 5 de la carte ESP8288.

void setup() {

La fonction setup() est une fonction spéciale en Arduino qui est exécutée une seule fois au démarrage du programme. Elle est utilisée pour initialiser les paramètres et configurations nécessaires.

pinMode(ledPin, OUTPUT);

Cette ligne configure la broche numérique ledPin (broche 5) comme une sortie (OUTPUT). Cela signifie que la broche sera utilisée pour envoyer un signal (HIGH ou LOW) à un composant externe, comme une LED.

}

Cette accolade ferme la fonction setup().

void loop() {

La fonction loop() est une autre fonction spéciale en Arduino qui est exécutée en boucle après la fonction setup(). Tout le code à l'intérieur de cette fonction sera répété indéfiniment.

digitalWrite(ledPin, HIGH);

Cette ligne envoie un signal HIGH (5V) à la broche ledPin, ce qui allume la LED connectée à cette broche.

delay(1000);

Cette ligne introduit une pause de 1000 millisecondes (1 seconde) dans l'exécution du programme. Pendant cette pause, la LED reste allumée.

digitalWrite(ledPin, LOW);

Cette ligne envoie un signal LOW (0V) à la broche ledPin, ce qui éteint la LED connectée à cette broche.

delay(1000);

Cette ligne introduit une autre pause de 1000 millisecondes (1 seconde). Pendant cette pause, la LED reste éteinte.

}

Cette accolade ferme la fonction loop(). Après cette ligne, le programme revient au début de la fonction loop() et répète les étapes 6 à 9 indéfiniment.

TP1

L'objectif de ce code est de transformer un module NodeMCU ESP8266 en un point d'accès WiFi (AP) et un serveur web. Ce serveur web permet à un utilisateur de contrôler une LED connectée à la broche D7 du NodeMCU via une interface web accessible depuis un navigateur. L'utilisateur peut allumer ou éteindre la LED en cliquant sur des boutons dans une page web.

```
#include <ESP8266WebServer.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>

// Ces lignes incluent les bibliothèques nécessaires pour utiliser les fonctionnalités WiFi et serveur web de l'ESP8266.

ESP8266WebServer server(80);

// Un objet ESP8266WebServer est créé pour gérer les requêtes HTTP sur le port 80.

const char *ssid = "SITD";
const char *password = "SITD2025";

// Définition du nom du réseau WiFi (SSID) et du mot de passe associé pour le point d'accès.

#define LED D7
int statusLED = LOW;

// La broche D7 est définie comme la broche de sortie pour le contrôle du LED. statusLED est utilisé pour suivre l'état actuel du LED.

const String HtmlHtml = "<html><head>"
"<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\" /></head>";
const String HtmlTitle = "<h1>Control LED Using NodeMCU WiFi Access Point</h1><br/>\n";
const String HtmlLedStateLow = "<big>LED is now <b>OFF</b></big><br/><br/>\n";
const String HtmlLedStateHigh = "<big>LED is now <b>ON</b></big><br/><br/>\n";
const String HtmlButtons =
"<a href=\"LEDOn\"><button style=\"background-
color:green;color:white;width:20%;height:5%;\">ON </button></a>\n"
"<a href=\"LEDOff\"><button style=\"background-color:red;color:white;width:20%;height:5%;\">OFF
</button></a><br/>";
const String HtmlHtmlClose = "</html>";

// Ces constantes définissent le contenu HTML qui sera envoyé au client lorsque la page est demandée.

void handleLedOn() {
  statusLED = HIGH;
  digitalWrite(LED, statusLED);
  Serial.println("NodeMCU ESP8266 : LED is ON");
  response();
}

void handleLedOff() {
  statusLED = LOW;
  digitalWrite(LED, statusLED);
  Serial.println("NodeMCU ESP8266 : LED is OFF");
  response();
}

// Deux fonctions sont définies pour gérer les actions lorsque l'utilisateur clique sur les boutons pour
```

allumer ou éteindre la LED. Elles changent l'état de statusLED et appellent la fonction response().

```
void response(){
  String htmlRes = HtmlHtml + HtmlTitle;
  if(statusLED == LOW){
    htmlRes += HtmlLedStateLow;
  }else{
    htmlRes += HtmlLedStateHigh;
  }

  htmlRes += HtmlButtons;
  htmlRes += HtmlHtmlClose;

  server.send(200, "text/html", htmlRes);
}
```

// La fonction response() construit la réponse HTML qui sera envoyée au client, en fonction de l'état actuel du LED, puis envoie cette réponse.

```
void setup() {
    delay(1000);
    Serial.begin(115200);
    Serial.println();

    WiFi.softAP(ssid, password);

    IPAddress apip = WiFi.softAPIP();
    Serial.print("Connect your wifi laptop/mobile phone to this NodeMCU Access Point : ");
    Serial.println(ssid);
    Serial.print("Visit this IP : ");
    Serial.print(apip);
    Serial.println(" in your browser.");

    server.on("/", response);
    server.on("/LEDOn", handleLedOn);
    server.on("/LEDOff", handleLedOff);

    server.begin();
    Serial.println("HTTP server beginned");

    pinMode(LED, OUTPUT);
    digitalWrite(LED, statusLED);
}
```

// La fonction setup() est exécutée une fois au démarrage. Elle configure la connexion WiFi en tant que point d'accès, configure les gestionnaires de requêtes pour les différentes URL (/, /LEDOn, /LEDOff), démarre le serveur web et configure la broche du LED.

```
void loop() {
    server.handleClient();
}
```

// La fonction loop() est exécutée en boucle. Elle vérifie continuellement s'il y a des requêtes clients et les traite en appelant server.handleClient().

Explications

#include <ESP8266WebServer.h>

Inclut la bibliothèque ESP8266WebServer, qui permet de créer un serveur web sur l'ESP8266 pour gérer les requêtes HTTP.

#include <ESP8266WiFi.h>

Inclut la bibliothèque ESP8266WiFi, qui fournit les fonctionnalités WiFi pour le module ESP8266.

#include <WiFiClient.h>

Inclut la bibliothèque WiFiClient, qui permet de gérer les connexions clients pour le serveur web.

ESP8266WebServer server(80);

Crée un objet ESP8266WebServer qui écoute sur le port 80 (port standard pour HTTP).

const char *ssid = "SITD";

Définit le nom du réseau WiFi (SSID) que le NodeMCU va créer. Ici, le réseau s'appelle SITD.

const char *password = "SITD2025";

Définit le mot de passe du réseau WiFi créé par le NodeMCU. Ici, le mot de passe est SITD2024.

#define LED D7

Définit la broche D7 comme celle à laquelle la LED est connectée.

int statusLED = LOW;

Initialise une variable statusLED pour suivre l'état actuel de la LED (LOW signifie éteinte par défaut).

const String HtmlHtml = "<html><head>..."

Définit une chaîne de caractères contenant le code HTML de base pour la page web. Cela inclut des balises HTML, des styles CSS et des boutons pour contrôler la LED.

const String HtmlTitle = "<h1>Control LED Using NodeMCU WiFi Access Point</h1>
\n";

Définit le titre de la page web qui sera affiché dans le navigateur.

const String HtmlLedStateLow = "<big>LED is now OFF</big>

\n";

Définit un message HTML qui sera affiché lorsque la LED est éteinte.

const String HtmlLedStateHigh = "<big>LED is now ON</big>

\n";

Définit un message HTML qui sera affiché lorsque la LED est allumée.

const String HtmlButtons = ...

Définit les boutons HTML pour allumer (ON) et éteindre (OFF) la LED. Ces boutons sont liés à des URL (/LEDOn et /LEDOff).

const String HtmlHtmlClose = "</html>";

Ferme la balise HTML de la page web.

void handleLedOn() { ... }

Cette fonction est appelée lorsque l'utilisateur clique sur le bouton "ON". Elle allume la LED en mettant statusLED à HIGH et appelle la fonction response() pour mettre à jour la page web.

void handleLedOff() { ... }

Cette fonction est appelée lorsque l'utilisateur clique sur le bouton "OFF". Elle éteint la LED en mettant

statusLED à LOW et appelle la fonction response() pour mettre à jour la page web.

void response() { ... }

Cette fonction construit la réponse HTML en fonction de l'état actuel de la LED (statusLED) et envoie cette réponse au client via server.send().

void setup() { ... }

La fonction setup() est exécutée une fois au démarrage. Elle configure :

La communication série pour le débogage.

Le point d'accès WiFi avec le SSID et le mot de passe définis.

Les gestionnaires de requêtes pour les URL /, /LEDOn, et /LEDOff.

La broche LED en mode sortie (OUTPUT).

L'état initial de la LED (LOW).

void loop() { ... }

La fonction loop() est exécutée en boucle. Elle appelle server.handleClient() pour gérer les requêtes des clients connectés au serveur web.

TP2

```
#include "DHT.h"
#define DHTPIN 2 // what digital pin we're connected to
#define DHTTYPE DHT11 // DHT 11
DHT dht(DHTPIN, DHTTYPE);
void setup() {
Serial.begin(9600);
dht.begin();
}
void loop() {
// Wait a few seconds between measurements.
delay(2000);
// get humidity reading
float h = dht.readHumidity();
// get temperature reading in Celsius
float t = dht.readTemperature();
// get temperature reading in Fahrenheit
float f = dht.readTemperature(true);
// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t) || isnan(f)) {
Serial.println("Failed to read from DHT sensor!");
return;
}
// display data on serial monitor
Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" %\t");
Serial.print("Temperature: ");
Serial.print(t);
Serial.print(" *C ");
Serial.print(f);
Serial.println(" *F");
}
```

Explications

```
#include "DHT.h"
```

Cette ligne inclut la bibliothèque DHT.h, qui contient les fonctions nécessaires pour communiquer avec le capteur DHT11 ou DHT22. Cette bibliothèque simplifie la lecture des données de température et d'humidité.

```
#define DHTPIN 2
```

Définit la broche numérique 2 de l'Arduino/NodeMCU à laquelle le capteur DHT est connecté.

```
#define DHTTYPE DHT11
```

Spécifie le type de capteur utilisé (ici, un DHT11). Si vous utilisez un DHT22, remplacez par DHT22.

```
DHT dht(DHTPIN, DHTTYPE);
```

Crée un objet nommé dht qui représente le capteur. Cet objet utilise les paramètres DHTPIN et DHTTYPE définis précédemment pour savoir où et comment communiquer avec le capteur.

```
void setup() {

  Serial.begin(9600);   // Initialise la communication série à 9600 bauds

  dht.begin();          // Initialise le capteur DHT

}
```

Serial.begin(9600) : Configure la communication série entre l'Arduino/NodeMCU et l'ordinateur à une vitesse de 9600 bauds (pour afficher les données dans le "Moniteur Série" de l'IDE Arduino).
dht.begin() : Initialise le capteur DHT et prépare la communication.

```
void loop() {

  delay(2000);  // Attend 2 secondes entre chaque lecture
```

Introduit un délai de 2 secondes entre chaque lecture pour éviter de surcharger le capteur (le DHT11 a un temps de réponse minimal de 1 seconde).

```
float h = dht.readHumidity();  // Lit l'humidité en %
```

La fonction readHumidity() lit l'humidité relative en pourcentage (%) et la stocke dans la variable h.

```
float t = dht.readTemperature();  // Lit la température en °C
```

La fonction readTemperature() lit la température en degrés Celsius (°C) et la stocke dans la variable t.

```
float f = dht.readTemperature(true);  // Lit la température en °F (true = Fahrenheit)
```

Le paramètre true indique à la fonction readTemperature() de retourner la température en degrés Fahrenheit (°F). La valeur est stockée dans f.

```
if (isnan(h) || isnan(t) || isnan(f)) {

  Serial.println("Failed to read from DHT sensor!");

  return;

}
```

isnan() : Vérifie si les valeurs lues (h, t, f) ne sont pas des nombres (NaN = "Not a Number"). Cela peut arriver si le capteur est mal connecté ou défectueux.
Serial.println(...) : Affiche un message d'erreur si une lecture échoue, puis quitte la fonction loop() avec return pour réessayer au prochain cycle.

```
Serial.print("Humidity: ");

Serial.print(h);

Serial.print(" %\t");

Serial.print("Temperature: ");

Serial.print(t);
```

```
Serial.print(" *C ");

Serial.print(f);

Serial.println(" *F");
```

Serial.print() : Affiche les données dans le Moniteur Série sans saut de ligne.
Serial.println() : Affiche les données et ajoute un saut de ligne.