# หน่วยการเรียนรู้ที่ 1 พื้นฐานอินเทอร์เน็ตในทุกสรรพสิ่ง และการเขียนโปรแกรมไมโครคอนโทรเลอร์พื้นฐาน


นายธงชัย ชาบุดศรี

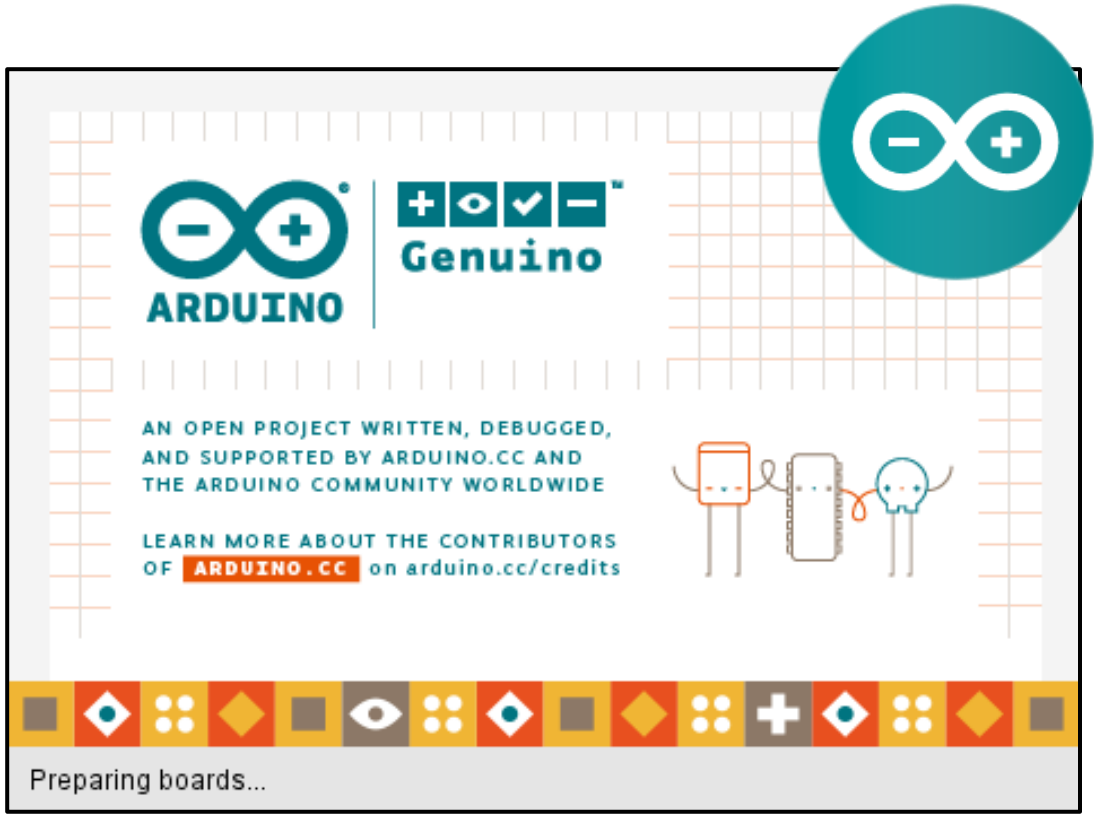
แผนกวิชาเทคโนโลยีสารสนเทศ

วิทยาลัยเทคนิคชลบุรี

# จุดประสงค์การเรียนรู้

1. เรียนรู้ความหมาย และความสำคัญของ Internet of Things
2. เรียนรู้โครงสร้าง และการใช้งานโปรแกรม Arduino IDE
3. เรียนรู้การใช้งานบอร์ดที่ใช้ชิป ESP32 และ ESP8266
4. เรียนรู้การทำงาน Input/Output ของพอร์ตแอนะล็อกและดิจิตอล
5. เรียนรู้การสื่อสารในรูปแบบอนุกรม (Serial Monitor)

# ใบงานที่ 1.1 พื้นฐานอินเทอร์เน็ตในทุกสรรพสิ่ง และการใช้งาน Arduino IDE



# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## แนวคิดของ Internet of Things

แนวคิด **Internet of Things** นั้นมาจาก **Kevin Ashton** ในปี ค.ศ.1999 โดยได้เสนอแนวคิดว่า

**“การนำสิ่งของต่างๆ ไม่ว่าจะเป็นคอมพิวเตอร์, เครื่องจักร และอุปกรณ์ตรวจจับมาเชื่อมต่อกับเครือข่ายอินเทอร์เน็ต เพื่อรายงานสถานะการทำงาน สถานะข้อมูล และรับรู้คำสั่งควบคุม”**

โดยในช่วงเวลานั้นโลกเพิ่งรู้จักอินเทอร์เน็ตได้ไม่นานแต่ก็มีคนนำแนวคิด IoT ไปสานต่อ และมีชื่อเรียกที่แตกต่างไป เช่น

- **Machine to Machine (M2M)**
- **Ubiquitous Computing**
- **Embedded Computing**
- **Smart Service**
- **Industrial Internet**


**Kevin Ashton**

**ผู้เสนอแนวคิด IoT**

# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## ความหมายของ Internet of Things (IoT)

Internet of Things ( IoT ) หรือ “อินเตอร์เน็ตในทุกสิ่ง” หมายถึง การที่สิ่งต่างๆ ถูกเชื่อมโยงทุกสิ่งทุกอย่างเข้าสู่โลกอินเตอร์เน็ต ทำให้มนุษย์สามารถติดตาม และควบคุมสั่งการอุปกรณ์ต่างๆ ทางเครือข่ายอินเตอร์เน็ต เช่น การสั่งเปิด-ปิด อุปกรณ์เครื่องใช้ไฟฟ้า เครื่องมือสื่อสาร เครื่องใช้สำนักงาน เครื่องมือทางการเกษตร เครื่องจักรในโรงงานอุตสาหกรรม เครื่องใช้ในชีวิตประจำวันต่างๆ

### ข้อดี ของ IoT

1. เพิ่มความสะดวกสบายในการทำงานและการดำเนินชีวิต
2. เพิ่มความรวดเร็ว และประสิทธิภาพในการทำงาน
3. ลดต้นทุนในด้านต่างๆ ลงได้จากการใช้ IoT

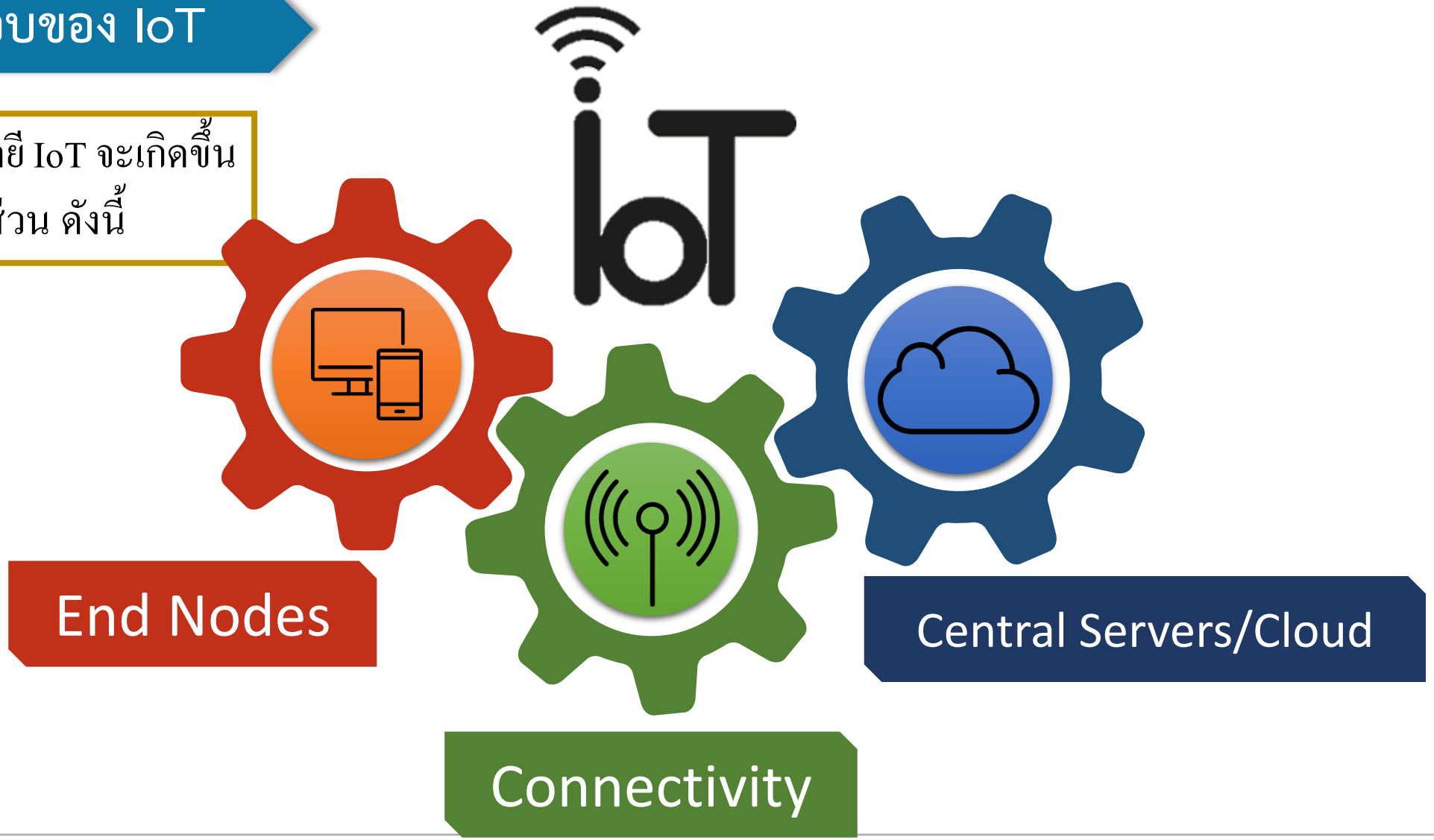
### ข้อเสีย ของ IoT

1. ต้องอาศัยระบบ Internet ตลอดเวลา
2. ระบบความปลอดภัย และความปลอดภัยของข้อมูล
3. ความผิดพลาดจากการประมวลผลของอุปกรณ์ต่างๆ

# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## องค์ประกอบของ IoT

ระบบหรือเทคโนโลยี IoT จะเกิดขึ้นได้ต้องมีองค์ประกอบ 3 ส่วน ดังนี้

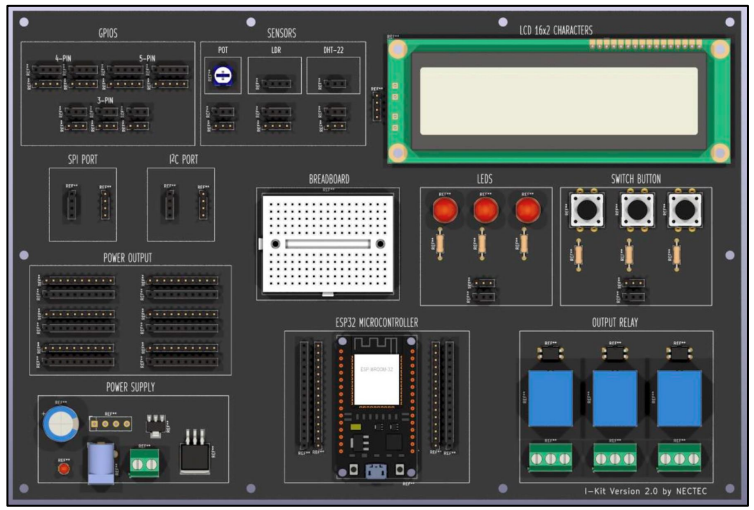


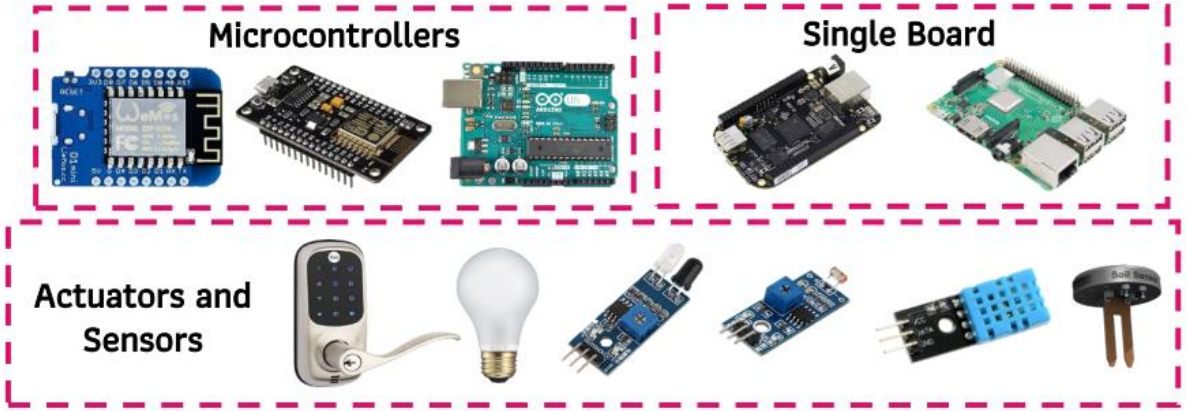
# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## องค์ประกอบของ IoT

### End Nodes

**ส่วนของสิ่งของ** (หลอดไฟ, มอเตอร์, เซนเซอร์) และ**อุปกรณ์ที่ถูกนำไปใช้ในการควบคุมสิ่งต่างๆ หรือตรวจจับ** ตรวจวัด ค่าพารามิเตอร์ต่างๆ ที่ต้องการ เช่น อุณหภูมิ ความชื้น ค่าแสงสว่าง เป็นต้น โดยสิ่งของสำหรับ IoT จะเป็นอะไรก็ได้ เพียงแต่ จะต้องติดตั้งอุปกรณ์ระบบสมองกลฝังตัวไว้ เพื่อทำหน้าที่เชื่อมต่อกับเครือข่ายอินเตอร์เน็ต โดยอุปกรณ์ที่นิยมปัจจุบันได้แก่ Microcontroller, Single Board เป็นต้น



# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## องค์ประกอบของ IoT

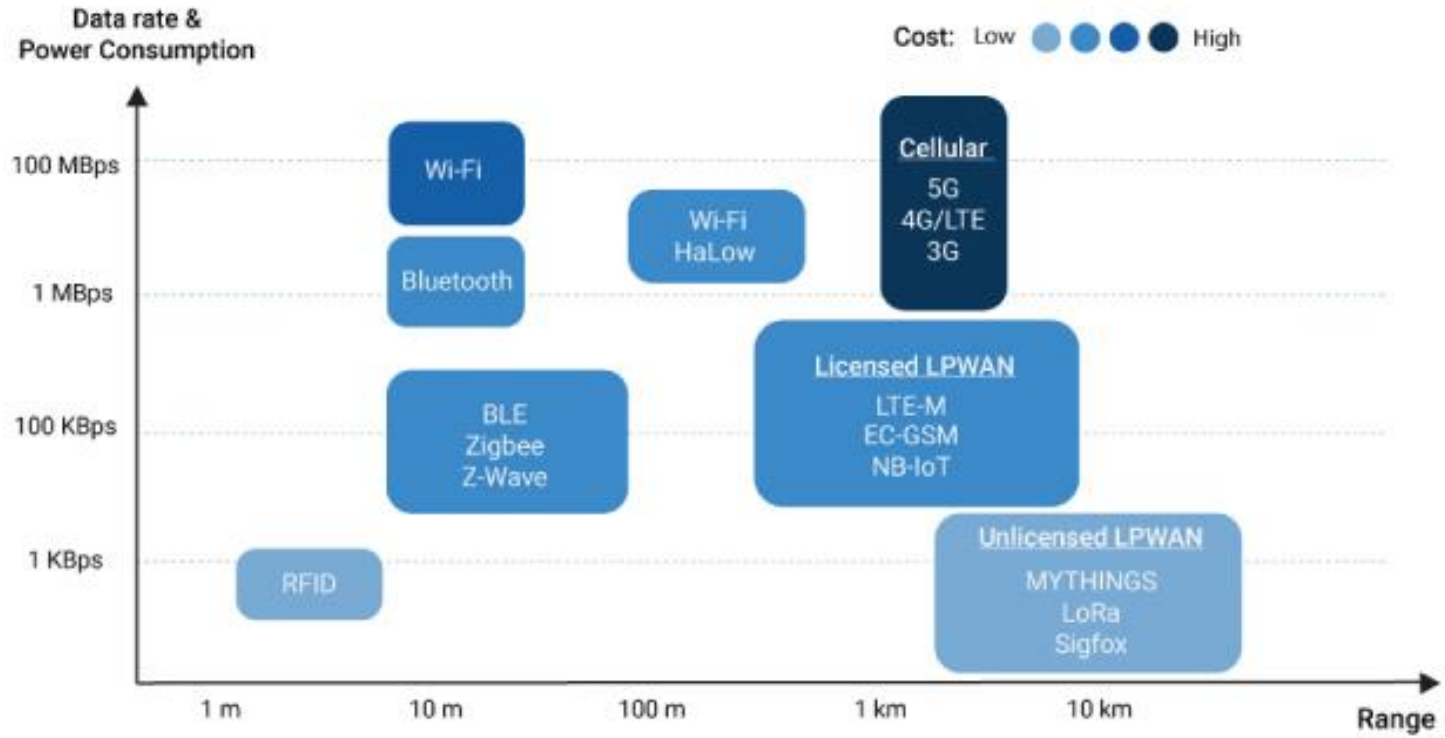
### Connectivity

Connectivity หรือ **ระบบเชื่อมต่ออินเตอร์เน็ต** คือ **ระบบที่ทำให้อุปกรณ์และสิ่งของสามารถเชื่อมต่อกับระบบอินเทอร์เน็ต** เพื่อทำการรับและส่งข้อมูลระหว่างอุปกรณ์ไปยัง Cloud โดยระบบเชื่อมต่ออินเตอร์เน็ตนั้นมีหลากหลายประเภท จะเป็นแบบมีสายหรือไร้สายก็ได้ ขึ้นอยู่กับความเหมาะสมในการใช้งาน

# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## องค์ประกอบของ IoT

### เปรียบเทียบ Data Rate และ Range ของแต่ละ Connectivity



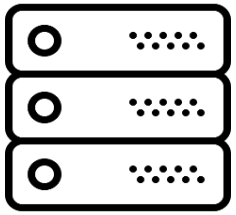
# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## องค์ประกอบของ IoT

### Central Servers/Cloud

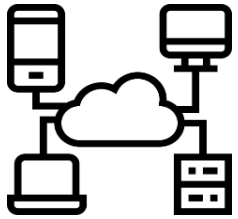
**Central Servers/Cloud** คือ พื้นที่จัดเก็บข้อมูลเพื่อนำมาสร้างเป็น Web Server, Application Server เป็นต้น โดยปัจจุบันสำหรับระบบ IoT นิยมใช้ Cloud เป็นพื้นที่จัดเก็บข้อมูลโดยที่ Server และ Cloud มีความแตกต่างกันคือ

**Server**



คือ เครื่องหรือโปรแกรมคอมพิวเตอร์ซึ่งทำงานให้บริการในระบบเครือข่ายแก่ลูกข่าย เครื่องคอมพิวเตอร์ที่ทำหน้าที่เป็นเซิร์ฟเวอร์นี้ควรมีประสิทธิภาพสูง มีความเสถียร สามารถให้บริการแก่ผู้ใช้ได้เป็นจำนวนมาก

**Cloud**



คือ การนำ Server หลายๆเครื่องมาทำงานด้วยกันโดยมีหน่วยประมวลผลระดับสูง โดย Cloud สามารถสร้าง Service ขึ้นมาทำงานได้หลากหลาย ในปัจจุบัน Cloud มีผู้ให้บริการหลากหลายทั้งแบบมีและไม่มีค่าใช้จ่าย

# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## ตัวอย่างการใช้งาน IoT

**Smart Farm**


# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## ตัวอย่างการใช้งาน IoT

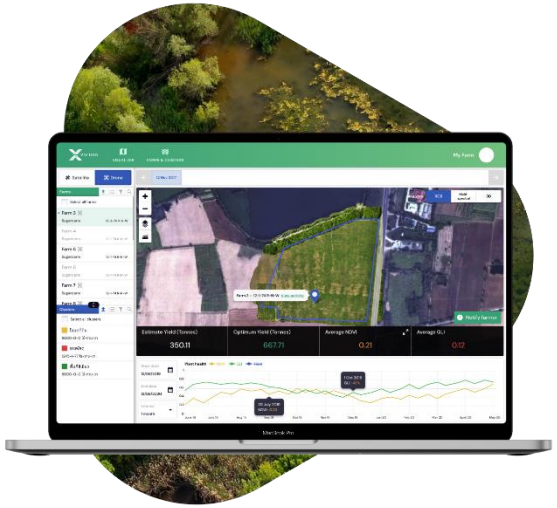
Smart Farm

### Handy Sense

"HandySense" หรือ "ระบบเกษตรแม่นยำฟาร์มอัจฉริยะ" นำเทคโนโลยีเซนเซอร์ (sensor) ผนวกอุปกรณ์ไอโอที (Internet of Things) สู่อุปกรณ์ตรวจวัดและควบคุมสภาพแวดล้อมที่เป็นปัจจัยต่อการเจริญเติบโตของพืชผล


### Varuna

Varuna เกิดจากการต่อยอดเทคโนโลยีอากาศยานไร้คนขับ หรือ โดรนและปัญญาประดิษฐ์ (Artificial Intelligence) เป็นเครื่องมือช่วยแก้ปัญหาการเกษตร โดยใช้เทคโนโลยีภาคการเกษตรเพื่อเพิ่มประสิทธิภาพการเพาะปลูก



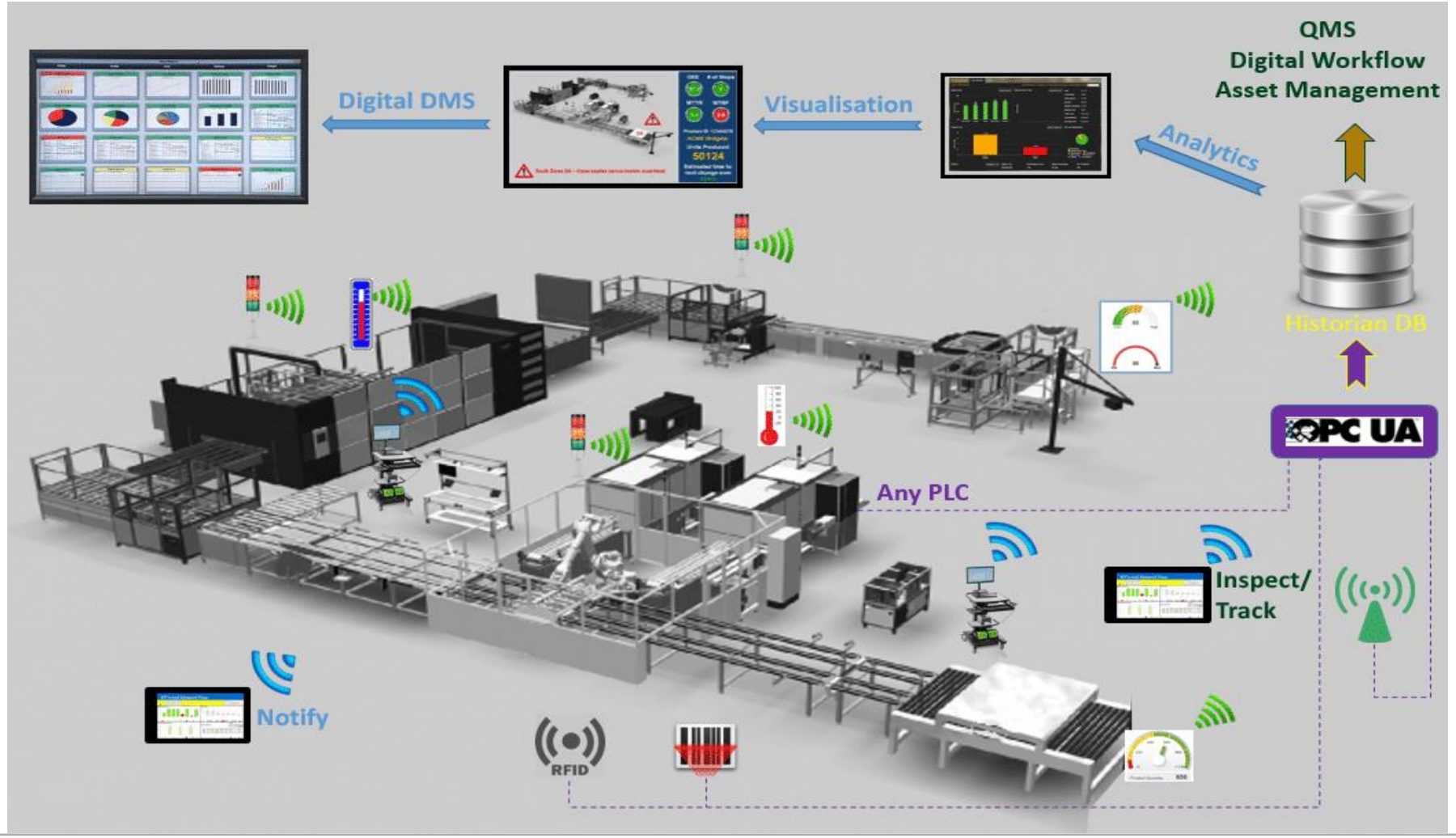
### Aqua-IoT

เป็นระบบที่จะช่วยในการเฝ้าระวัง Aqua IoT ประกอบไปด้วย **4 เทคโนโลยีย่อย** 1.ระบบตรวจวัดและติดตามสภาพทางกายภาพ GROW FiT System 2.ระบบตรวจวัดและติดตามสภาพทางกายภาพ MuEye System 3.ระบบตรวจวัดและติดตามสภาพทางเคมี ChemEye System 4.ระบบตรวจวัดและติดตามสภาพทางชีวภาพ Minimal Lab System

# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## ตัวอย่างการใช้งาน IoT

**Smart Factory**



# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## Smart Factory

### บริษัท นิเด็ค ชิบาอุระ อิเล็กทรอนิกส์ (ประเทศไทย)

การนำ IoT มาใช้ในโรงงานอุตสาหกรรม เริ่มตั้งแต่ส่วนของสิ่งอำนวยความสะดวกไปยังส่วนของการผลิต เช่น การแสดงสถานะห้องไปชุม แอพลิเคชั่นเปิด/ปิดไฟ ระบบควบคุมอุณหภูมิในห้องตรวจสอบคุณภาพ ไปจนถึงการมอนิเตอร์ค่าๆต่างของเครื่องจักร และการติดตามผลของฝ่ายการผลิต


### โครงการ “IDA NECTEC”

เป็นแพลตฟอร์มที่สามารถเชื่อมโยงข้อมูลจากจากอุปกรณ์ IoT (Internet of Things) ตรวจจับสัญญาณต่าง ๆ จากเครื่องจักรในกระบวนการผลิตสู่การวิเคราะห์และบูรณาการข้อมูล ทำให้ทราบสถานภาพของเครื่องจักร นำไปสู่การบริหารจัดการการผลิตอย่างมีประสิทธิภาพและอนุรักษ์พลังงาน


### โครงการ “UNAI”

ระบบ UNAI จัดอยู่ในกลุ่มของเทคโนโลยีอินเทอร์เน็ตสรรพสิ่ง (Internet of Things) ประกอบด้วยอุปกรณ์ 3 ส่วนหลัก ได้แก่ 1. อุปกรณ์ส่งสัญญาณไร้สาย 2. อุปกรณ์รับสัญญาณไร้สาย 3. ระบบสื่อสารสำหรับส่งข้อมูลไปยังเซิร์ฟเวอร์

# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## ตัวอย่างการใช้งาน IoT

### Smart Home


**Smart home**

คือ เป็นระบบบ้านยุคใหม่ที่มีความทันสมัยที่สามารถควบคุมอุปกรณ์ และ ระบบภายในบ้านได้จากภายนอกหรือทุกที่มี internet ผ่าน Application ใน Smartphone เพื่อตอบสนอง Lifestyle ในปัจจุบัน และ ยังสามารถเชื่อมต่อระบบ security เพื่อเพิ่มความปลอดภัยภายในบ้านได้

# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## ตัวอย่างการใช้งาน IoT

### Smart Home Product



# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## ตัวอย่างการใช้งาน IoT

### Smart City

เมืองอัจฉริยะ คือ เป็นรูปแบบการประยุกต์เทคโนโลยีดิจิทัล หรือข้อมูลสารสนเทศและการสื่อสารใน การเพิ่มประสิทธิภาพและคุณภาพของบริการชุมชนเพื่อช่วยในการลดต้นทุน และการบริโภคของประชากร โดยยังคงเพิ่มประสิทธิภาพให้ประชาชนสามารถอาศัยได้ในคุณภาพชีวิตที่ดีขึ้น ซึ่งพัฒนาให้เข้ากับยุค 4.0 โดยการเอาเทคโนโลยีมาผสานกับการใช้ชีวิตของประชาชน

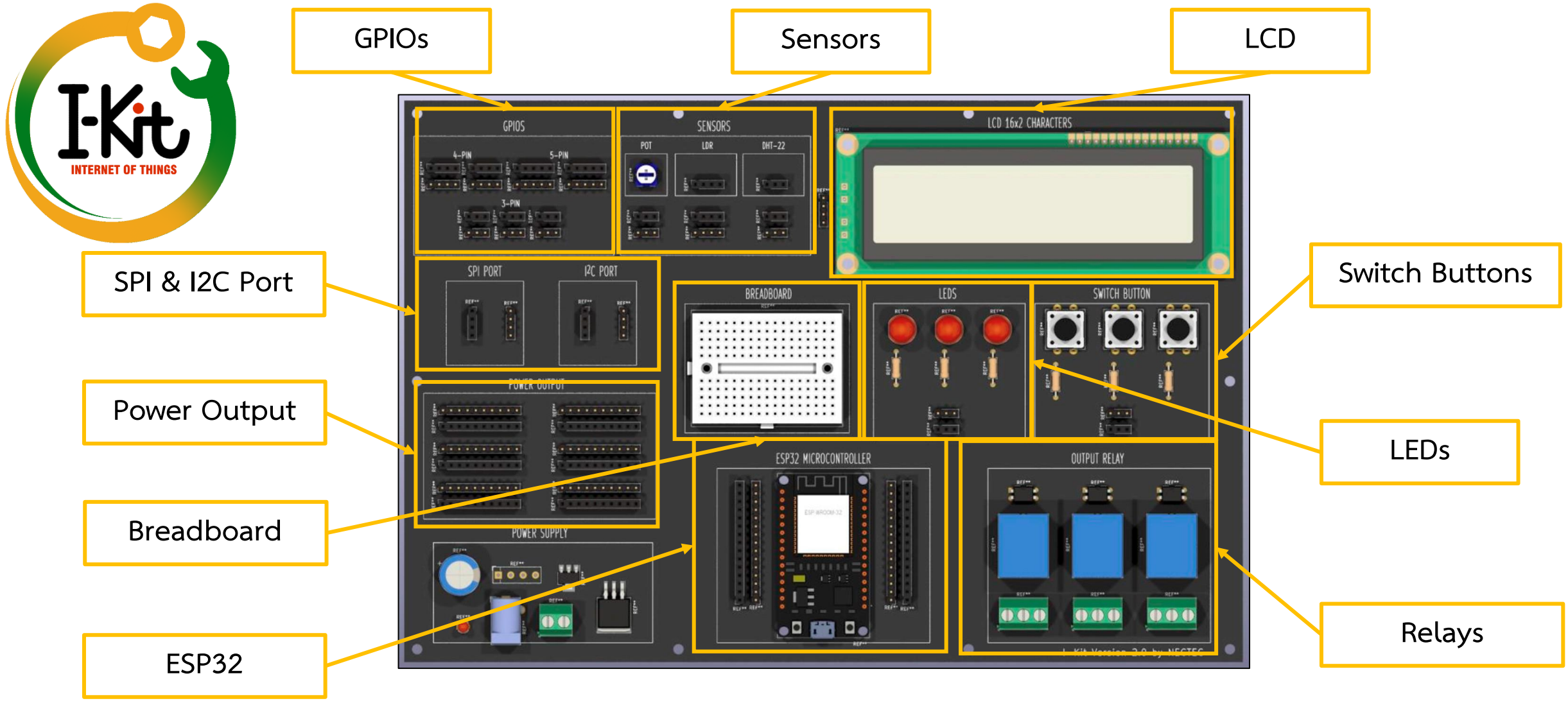

# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## ตัวอย่างการใช้งาน IoT

### Smart health

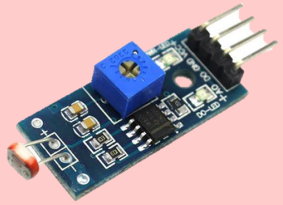

ตัวอย่าง m-Health & Realtime monitoring หรือการตรวจตราและเฝ้าระวังสุขภาพผ่านโทรศัพท์มือถือ แอปพลิเคชัน และอุปกรณ์สวมใส่ต่างๆ (Wearable) แบบ Real time ซึ่งสามารถ เชื่อมต่อซึ่งกันและกันระหว่างผู้ป่วยและบุคลากรทางการแพทย์

# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น



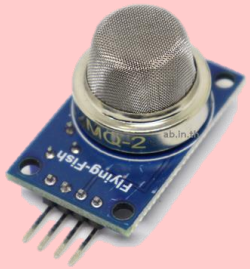
# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## Example End nodes in IoT



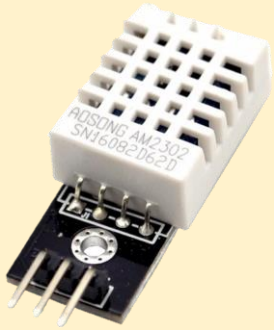
LDR Sensor Module

หน้าที่: ใช้ตรวจวัดความเข้มแสง



MQ2 Sensor Module

หน้าที่:โมดูลตรวจวัดแก๊ส ที่ไวต่อแก๊สไวไฟ รวมไปถึงควันไฟที่เกิดจากการเผาไหม้ด้วย นิยมนำมาใช้ในการตรวจจับการรั่วของแก๊สต่างๆ



DHT Sensor Module

หน้าที่: ใช้ตรวจวัดอุณหภูมิและความชื้น โดยรับค่าเป็น analog signal จากนั้น chip ในตัวเซ็นเซอร์จะแปลงเป็น digital signal

PIR Sensor

หน้าที่: โมดูลตรวจจับการเคลื่อนไหวผ่านความร้อน

Push bottom switch

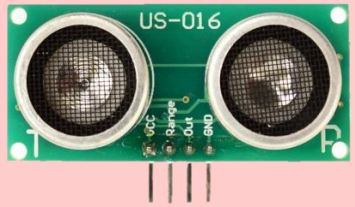
หน้าที่: ใช้ตัดและต่อวงจรไฟฟ้าได้ สวิตช์ทั่วไปนั้นจะเป็นประเภทสวิตช์กดติดปล่อยดับ (Tact Switch) ซึ่งในการเขียนโปรแกรมนั้นสามารถประยุกต์ได้


Potentiometer

หน้าที่: ตัวต้านทานปรับค่าได้ สามารถนำไปประยุกต์ใช้งานได้หลายรูปแบบ เช่น ปรับความสว่างของ LED

# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## Example End nodes in IoT




Ultrasonic Sensor Module

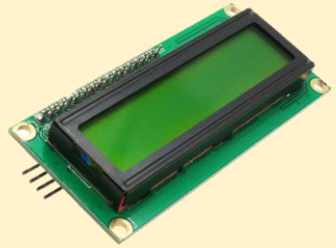
หน้าที่: ใช้ตรวจวัดระยะทาง ด้วยคลื่น Ultrasonic

RFID Module

หน้าที่: เป็นโมดูลที่ใช้อ่านบัตร Key Card


จอแสดงผล OLED

หน้าที่:จอแสดงผลที่สร้างจากวัสดุสารกึ่งตัวนำอินทรีย์ (Organic Semiconductor) ที่มีลักษณะเป็นชั้นสารกึ่ง-ตัวนำบาง ๆ อยู่ระหว่างขั้วบวก (Anode) และขั้วลบ (Cathode) และสามารถเปล่งแสงได้เมื่อมีกระแสไฟฟ้าไหลผ่าน

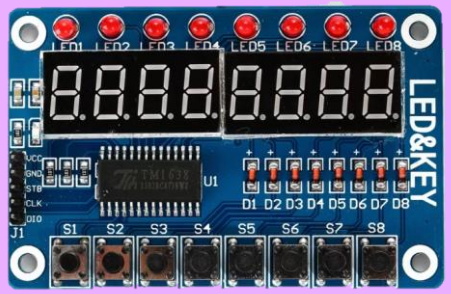


จอแสดงผล LCD

Liquid Crystal Display ซึ่งเป็นจอที่ทำมาจากผลึกคริสตอลเหลว หลักการคือด้านหลังจอจะมีไฟส่องสว่างหรือที่เรียกว่า Backlight อยู่ เมื่อมีการปล่อยกระแสไฟฟ้าเข้าไปกระตุ้นที่ผลึก ก็จะทำให้ผลึกโปร่งแสง


Relay Module

หน้าที่: ใช้ทำหน้าที่ตัดต่อวงจรแบบเดียวกับสวิตช์ โดยควบคุมการทำงานด้วยไฟฟ้า




7 Segments Board

หน้าที่: หน้าจอแสดงผลตัวเลข - ตัวอักษรได้บางตัว เมื่อทำให้หลอด LED แต่ละดวงติดพร้อมกัน ก็จะทำให้แสดงออกมาเป็นตัวเลขทรงเหลี่ยมได้

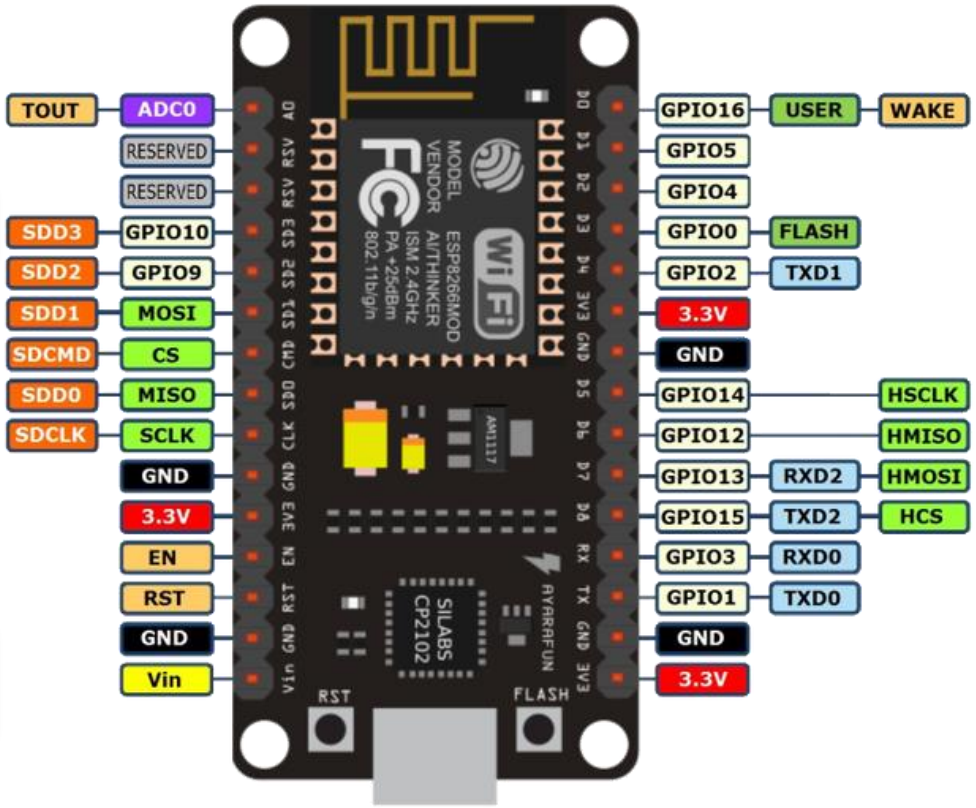
# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## NodeMCU (ESP8266)

NodeMCU-12E หรือ V3 หรือ Development Kit V1.0 (ชื่อเรียกนั้นขึ้นกับผู้ผลิต) เป็นการนำ ESP8266-12E หรือ 12F มาต่อร่วมกับชิปแปลงสัญญาณ USB เป็น UART มีสวิตซ์เพื่อเข้าสู่โหมดโปรแกรมเฟิร์มแวร์ รวมอยู่บนแผงวงจรขนาดเล็กที่ออกแบบมาให้ติดตั้งลงบนแผงต่อวงจร

### คุณสมบัติทางเทคนิคที่สำคัญ

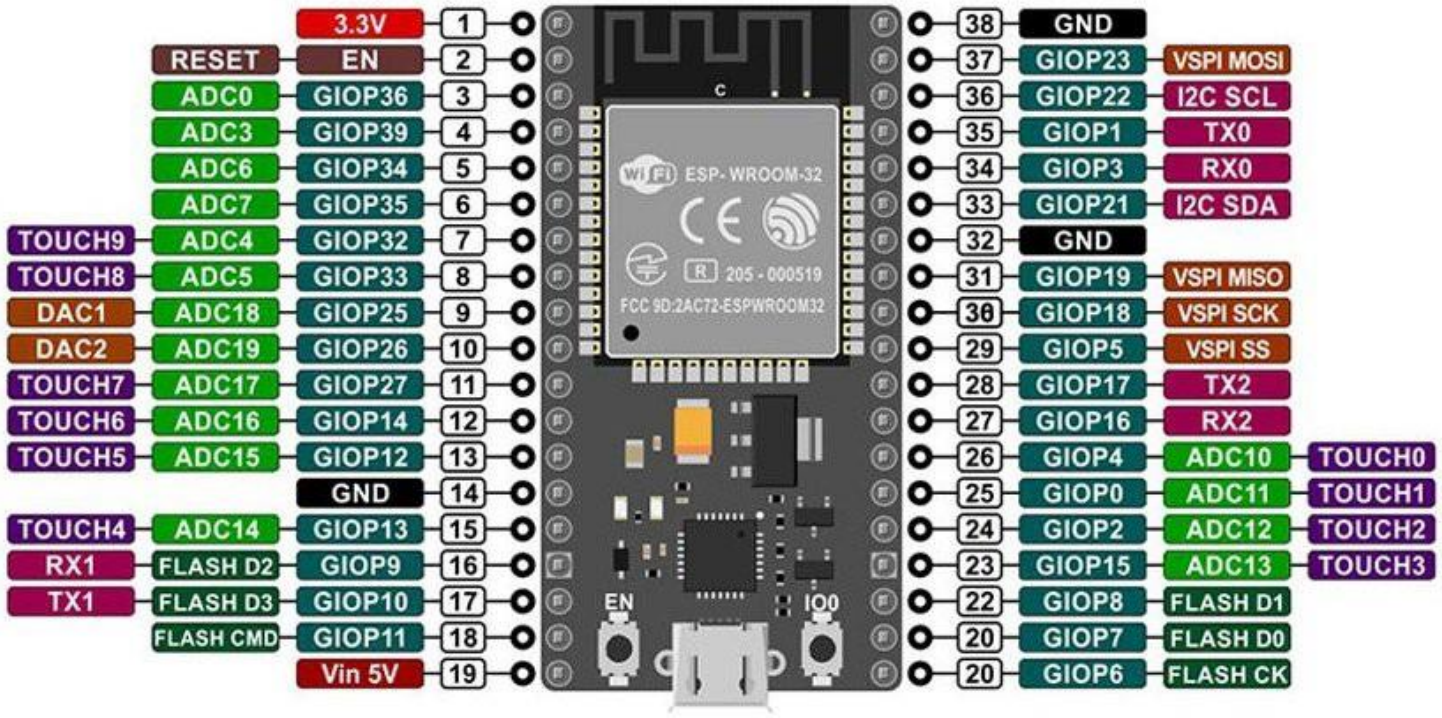
1. ใช้ไฟเลี้ยงภายนอก +5V มีวงจรควบคุมแรงดันไฟเลี้ยงสำหรับอุปกรณ์ 3.3V กระแสไฟฟ้าสูงสุด 80 mA
2. มีขาพอร์ต SPI สำหรับติดต่อกับ SD การ์ด
3. มีสวิตช์ RESET และ FLASH สำหรับโปรแกรมเฟิร์มแวร์ใหม่
4. มีอินพุตเอาต์พุตดิจิตอล (ลอจิก 3.3V) รวม 16 ขา
5. มีอินพุตแอนาล็อก 1 ช่อง รับแรงดันไฟตรง 0 ถึง +3.3Vdc เข้าสู่วงจรแปลงสัญญาณแอนาล็อกเป็นดิจิตอล ความละเอียด 10 บิต



# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## ESP32 DevKit

จากความสำเร็จของ ESP8266 Wi-Fi ที่ทำให้อุปกรณ์หลาย ๆ อย่าง สามารถเชื่อมกับโลกของ Internet จึงเกิดชิป ESP32 ซึ่งเป็นรุ่นต่อจาก ESP8266 มีความเร็ว แรง และฟังก์ชันเยอะกว่ารุ่น ESP8266 และยังรองรับการเขียนโปรแกรมบน Arduino IDE อีกด้วย



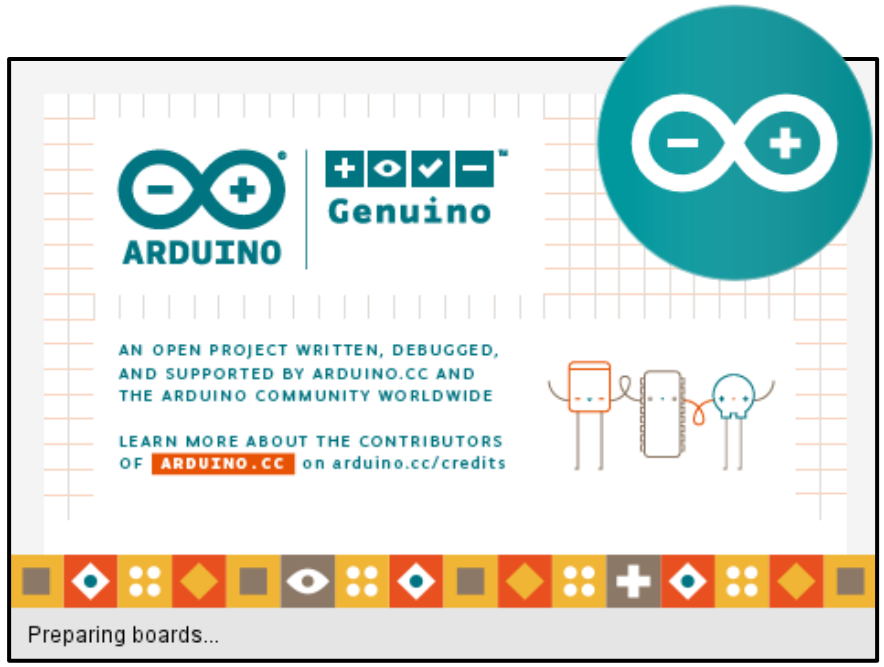

## คุณสมบัติทางเทคนิคที่สำคัญ

1. ทำงานแบบ Dual Core
2. รองรับการเครือข่ายสัญญาณไร้สาย WIFI และ Bluetooth 4.0
3. มีการทำงานแบบ 32 Bit
4. ความถี่ Clock ความเร็วสูงสุดถึง 240 MHz
5. หน่วยความจำ RAM 512 kB
6. มีความสามารถอีกหลายหลาย เช่น Capacitive Touch , Hall Sensor, ADCs , DAC , UART , SPI ,I2C และอื่น ๆ

# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## การเขียนโปรแกรมบน I-Kit

การเขียนโปรแกรมควบคุมการทำงาน I-Kit สามารถใช้โปรแกรม Sketch ของ Arduino IDE, PlatformIO หรือ Visual Studio Code แต่ในการอบรมนี้เราจะใช้ Arduino IDE ซึ่งเป็นโปรแกรมใช้งานง่าย เขียนด้วยภาษา C และเป็น Open Source ทำให้ใช้งานง่ายโดยไม่มีค่าใช้จ่าย ได้รับความนิยมสูง มีแหล่งข้อมูลให้ศึกษาค้นคว้ามากมาย

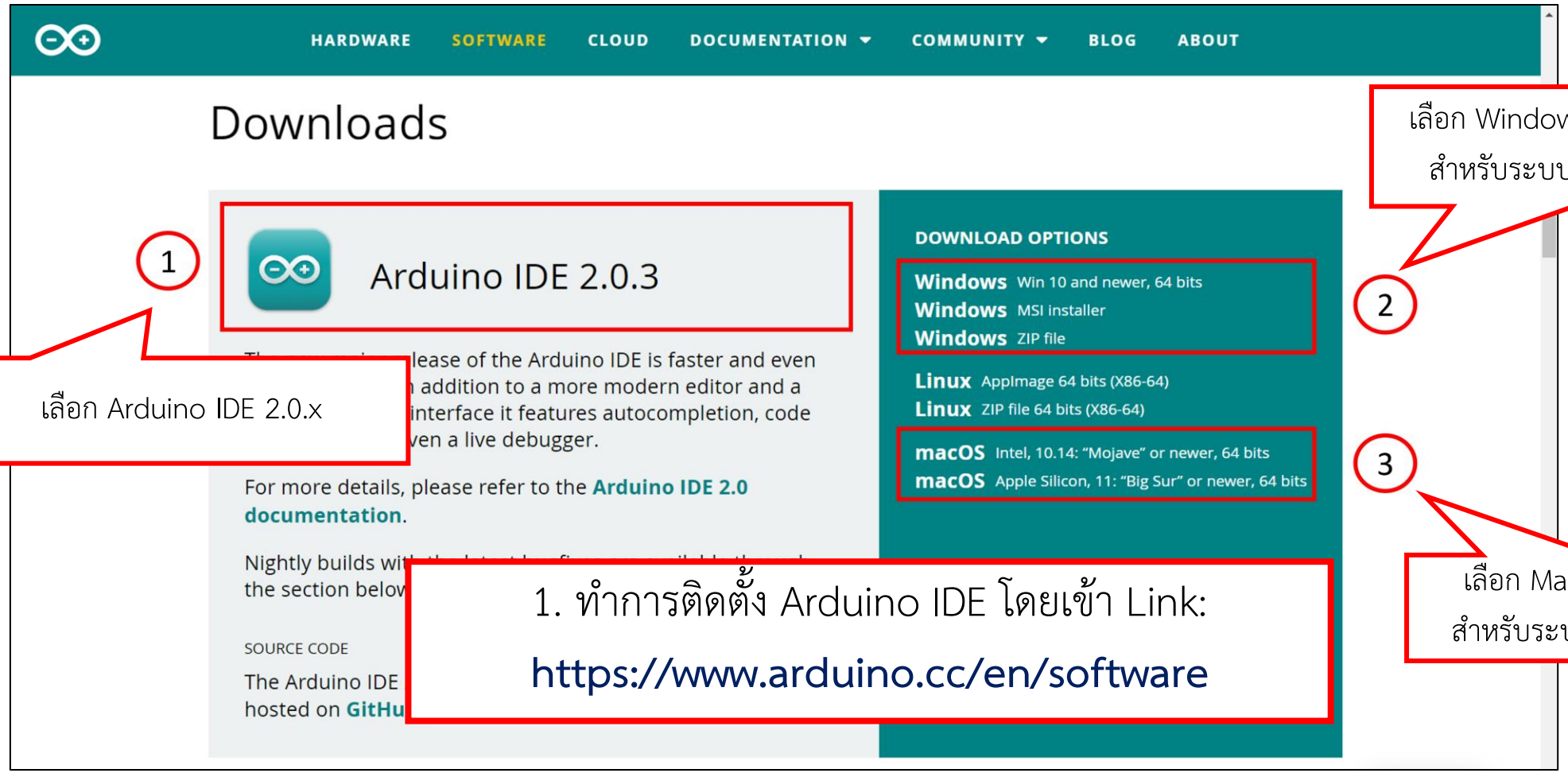



PlatformIO

Visual Studio Code

# ใบงานที่ 1.1 ขั้นตอนการทดลอง

## การทดลองที่ 1 การติดตั้งโปรแกรม Arduino IDE




เลือก Arduino IDE 2.0.x

เลือก Windows win 10 and newer สำหรับระบบปฏิบัติการ Window

เลือก Mac OS ตาม Processor สำหรับระบบปฏิบัติการ Mac OS

1. ทำการติดตั้ง Arduino IDE โดยเข้า Link:

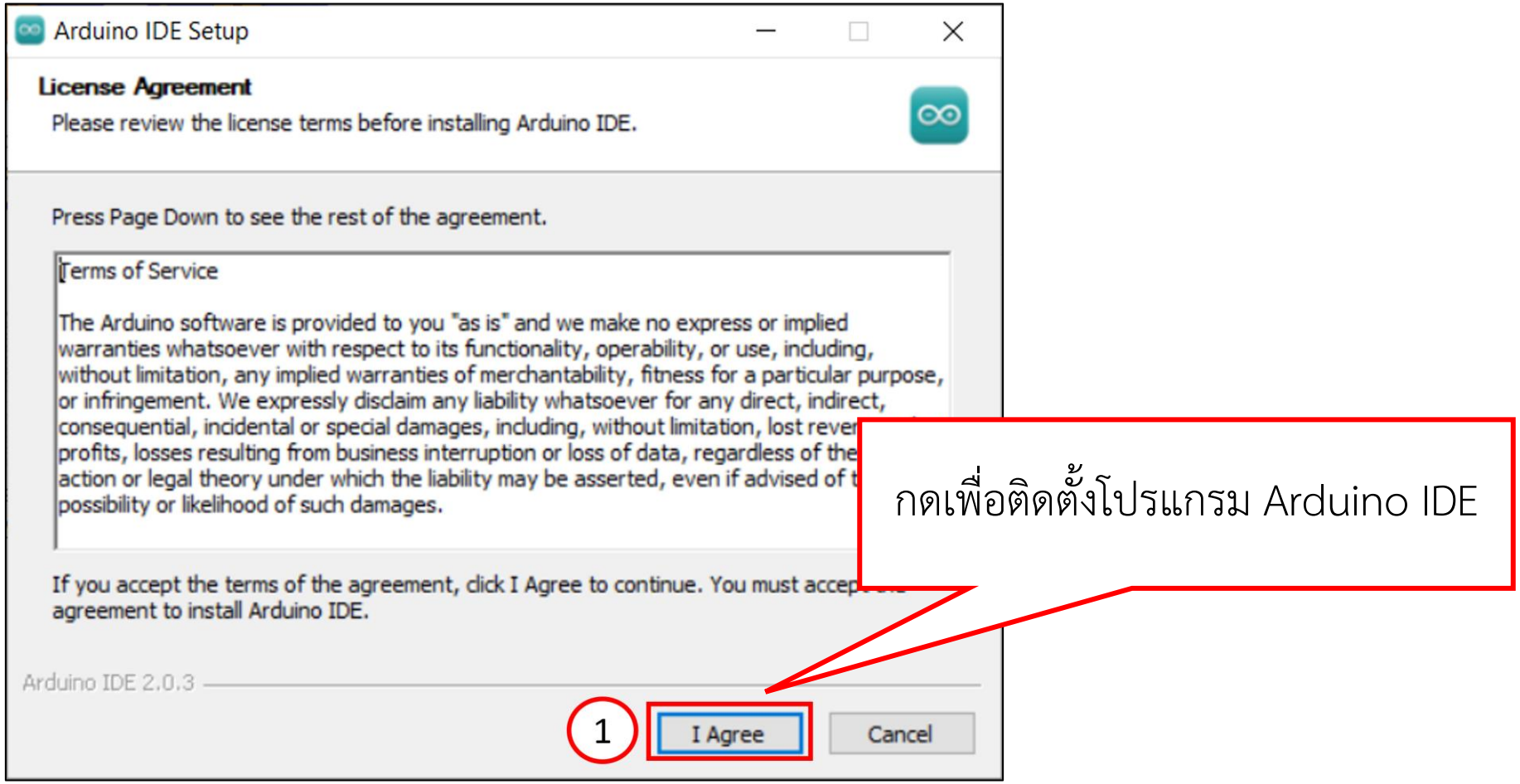
https://www.arduino.cc/en/software

# ใบงานที่ 1.1 ขั้นตอนการทดลอง

## การทดลองที่ 1 การติดตั้งโปรแกรม Arduino IDE

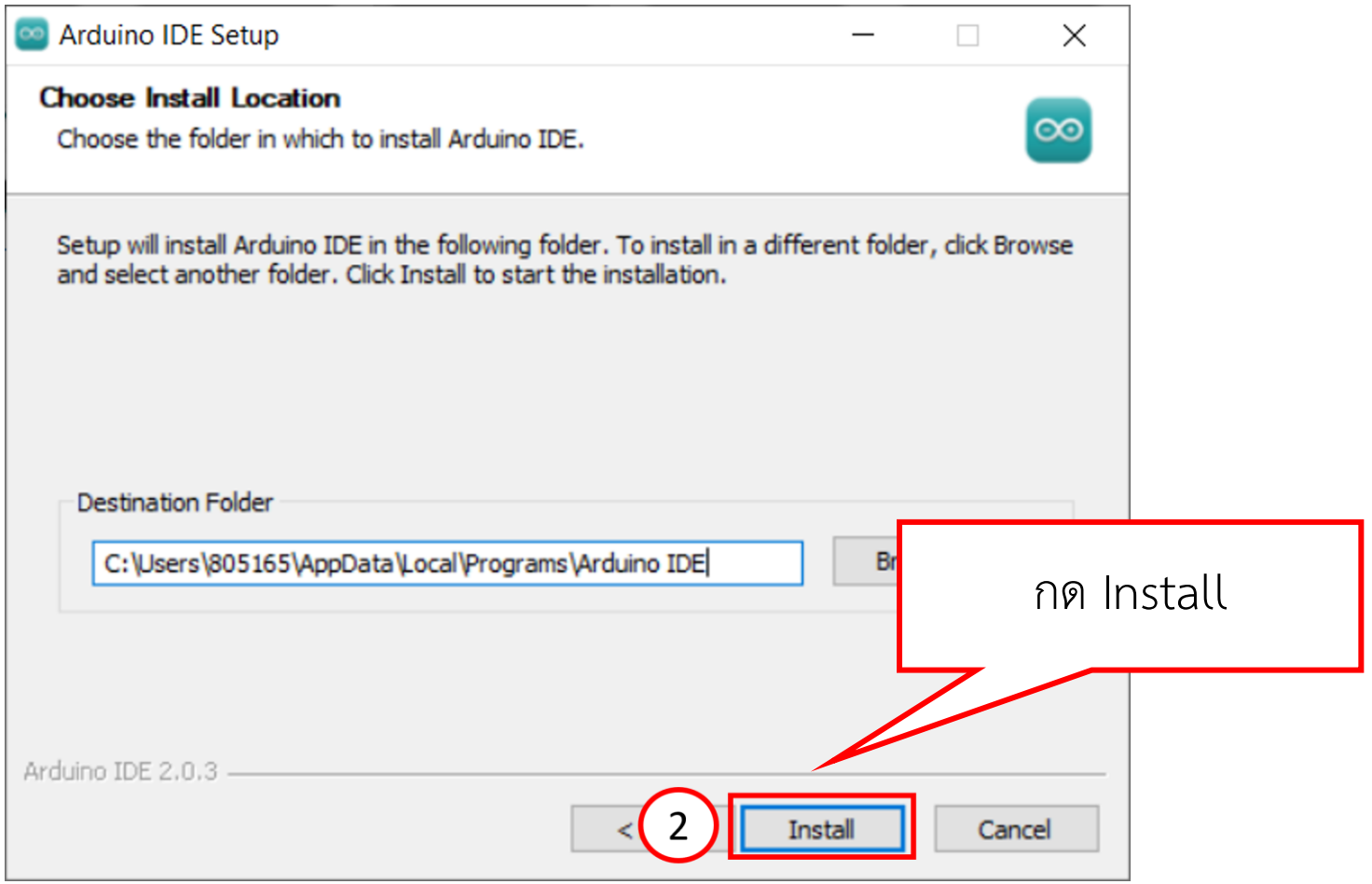
# ใบงานที่ 1.1 ขั้นตอนการทดลอง

## การทดลองที่ 1 การติดตั้งโปรแกรม Arduino IDE



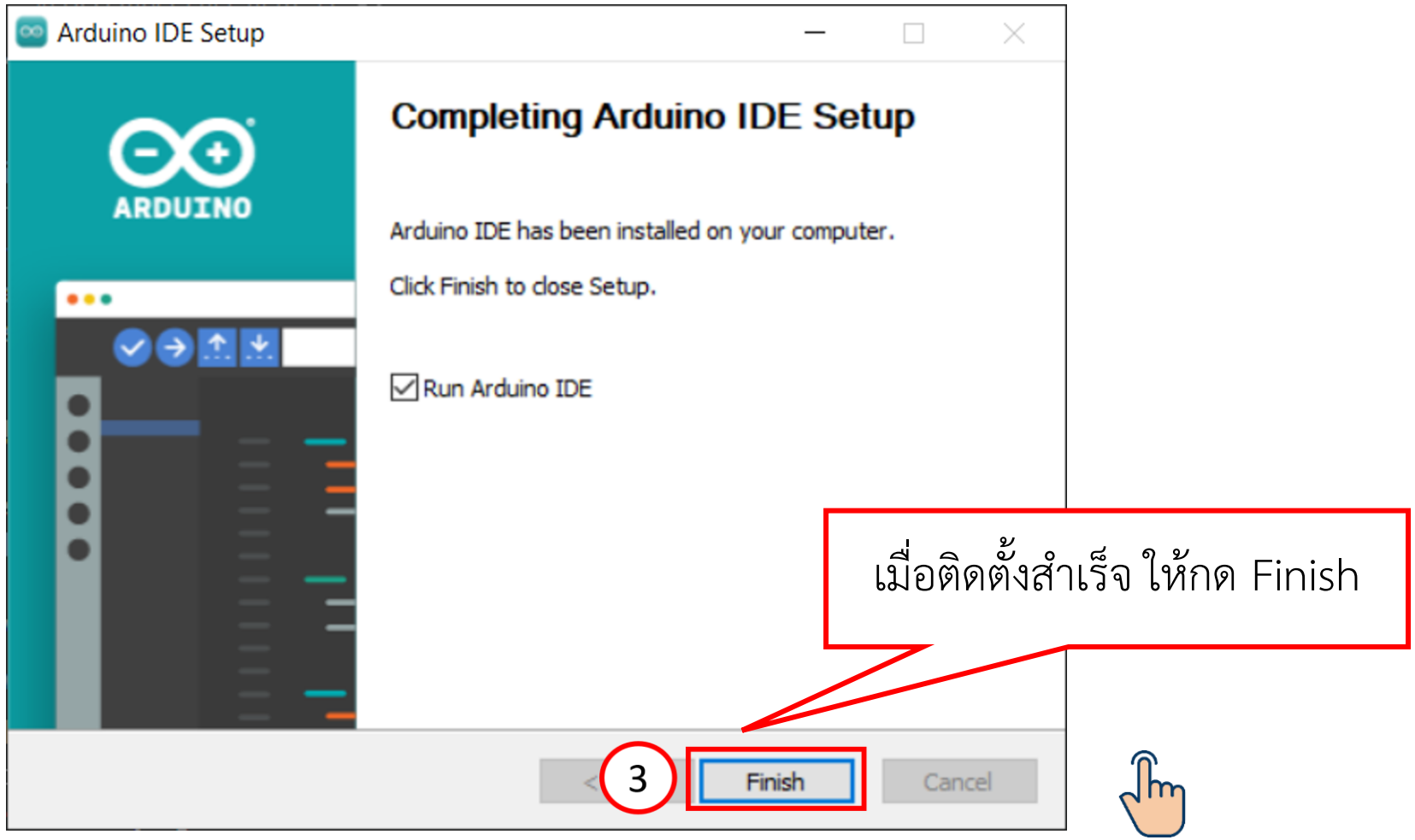
# ใบงานที่ 1.1 ขั้นตอนการทดลอง

## การทดลองที่ 1 การติดตั้งโปรแกรม Arduino IDE



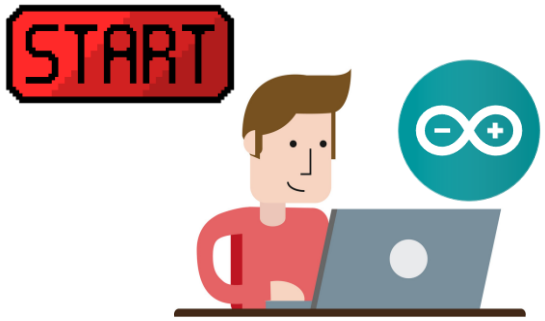
# ใบงานที่ 1.1 ขั้นตอนการทดลอง

## การทดลองที่ 1 การติดตั้งโปรแกรม Arduino IDE



# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## การทำงานของ Arduino




เขียนโค้ดโปรแกรม (Sketch) บน Arduino IDE

ARDUINO IDE

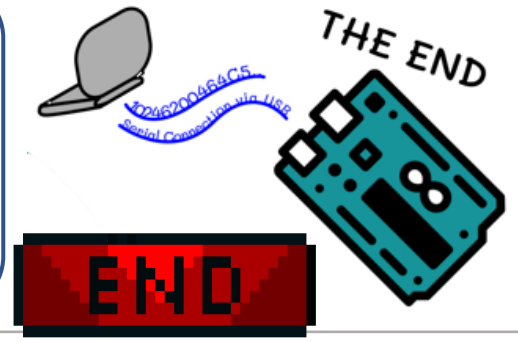
Sketch → C++

Arduino IDE แปลงโค้ดที่เราเขียน (Sketch) ให้กลายเป็นภาษา C++


Compiler AVR-GCC ของ Arduino IDE จะแปลงโค้ดให้กลายเป็นภาษาที่ microcontroller (.hex) เข้าใจ


.hex คือไฟล์รหัสภาษาเครื่อง (machine code) เป็นไฟล์ต้นฉบับเลขฐานสิบหกที่มักใช้โดยอุปกรณ์ลอจิกที่โปรแกรมได้เช่นไมโครคอนโทรลเลอร์ มีการตั้งค่าข้อมูลการกำหนดค่าหรือข้อมูลอื่น ๆ ที่บันทึกในรูปแบบเลขฐานสิบหก

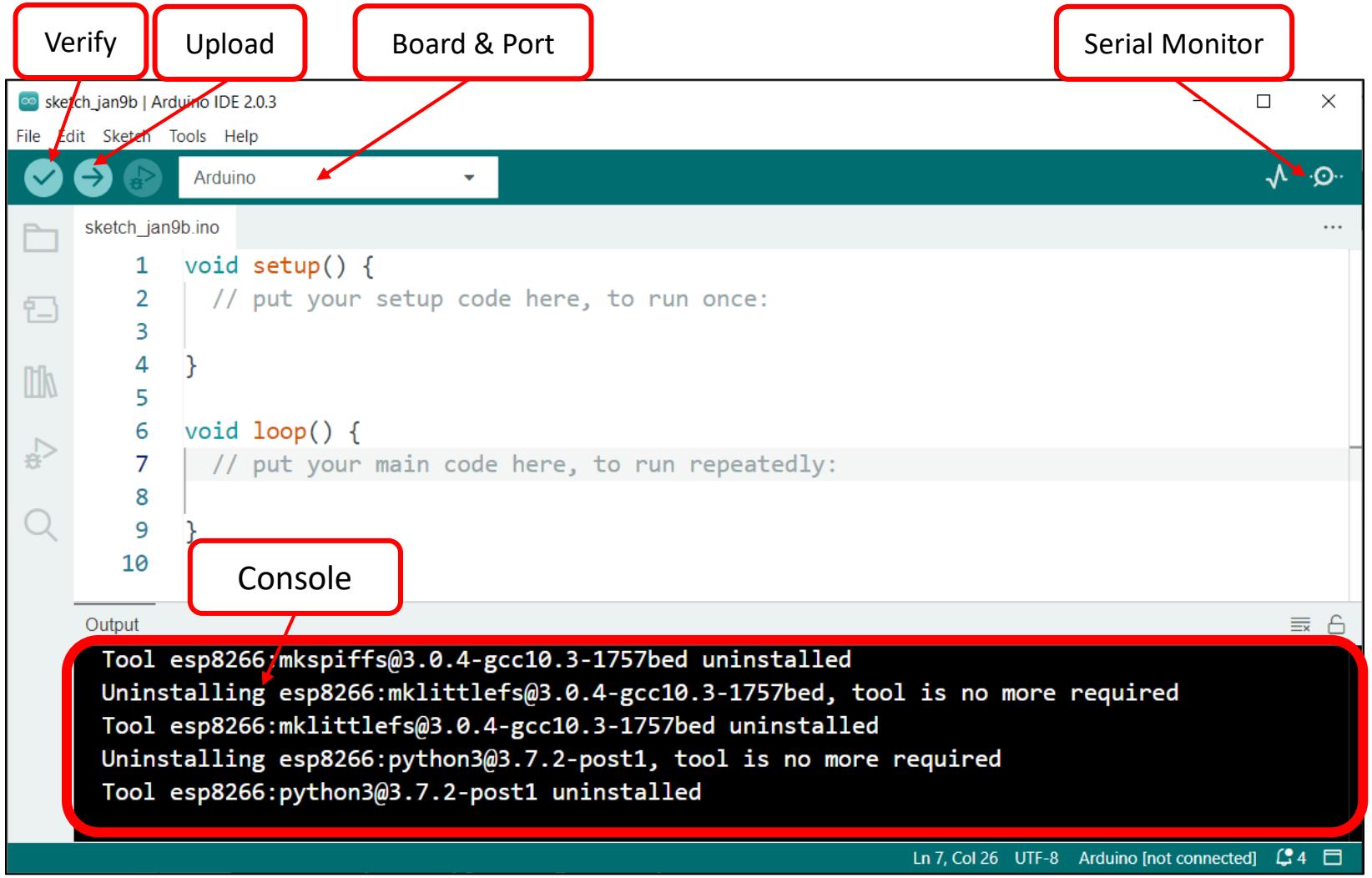
เขียนและส่งโปรแกรมที่ผ่านการแปลงเป็นภาษาที่ Microcontroller เข้าใจ ผ่าน USB (Serial Connection) และ bootloader ที่ติดตั้งอยู่บนตัวบอร์ด



# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## Arduino IDE

**เมนูต่างๆของโปรแกรม Arduino IDE**



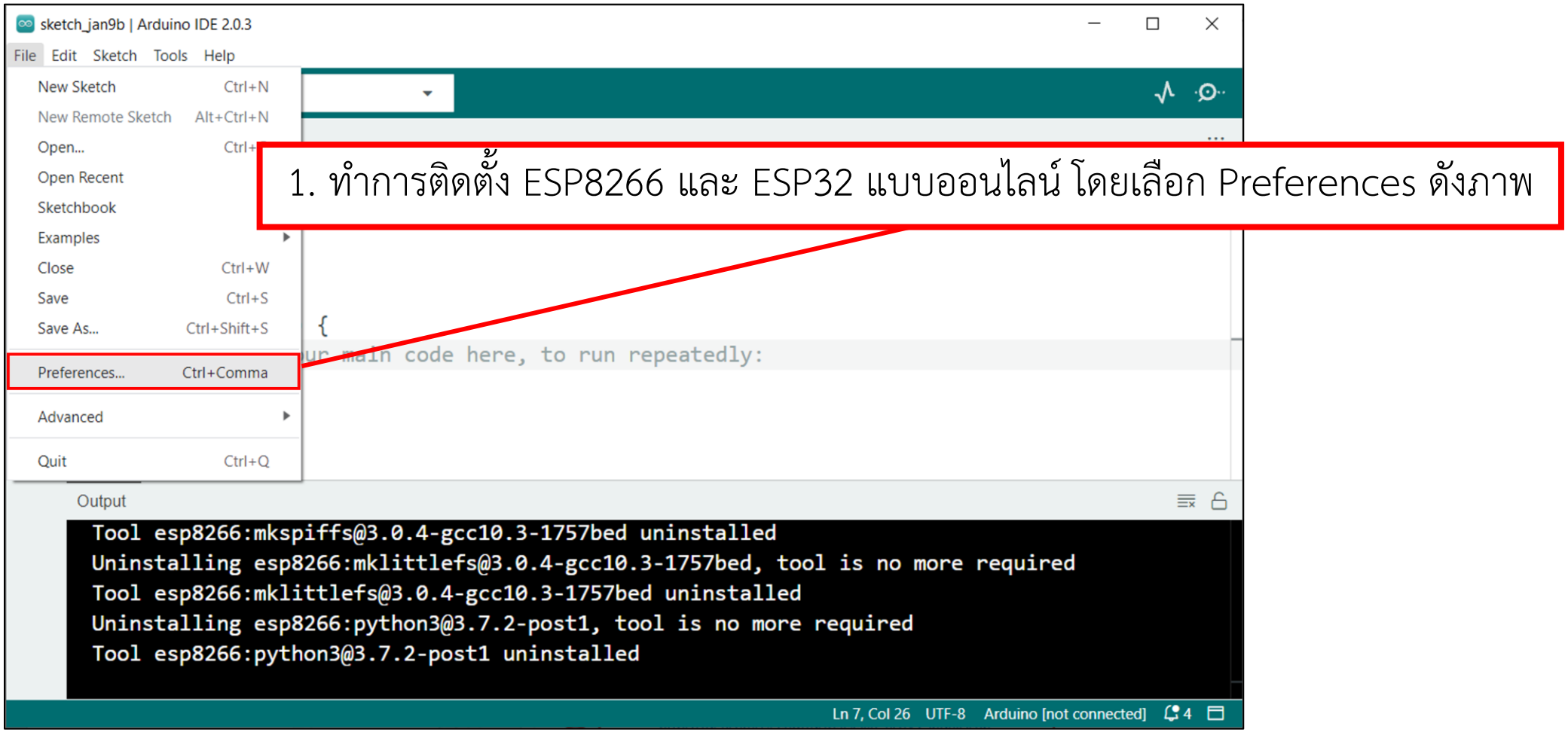
# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## Arduino IDE

**เมนูต่างๆของโปรแกรม Arduino IDE**

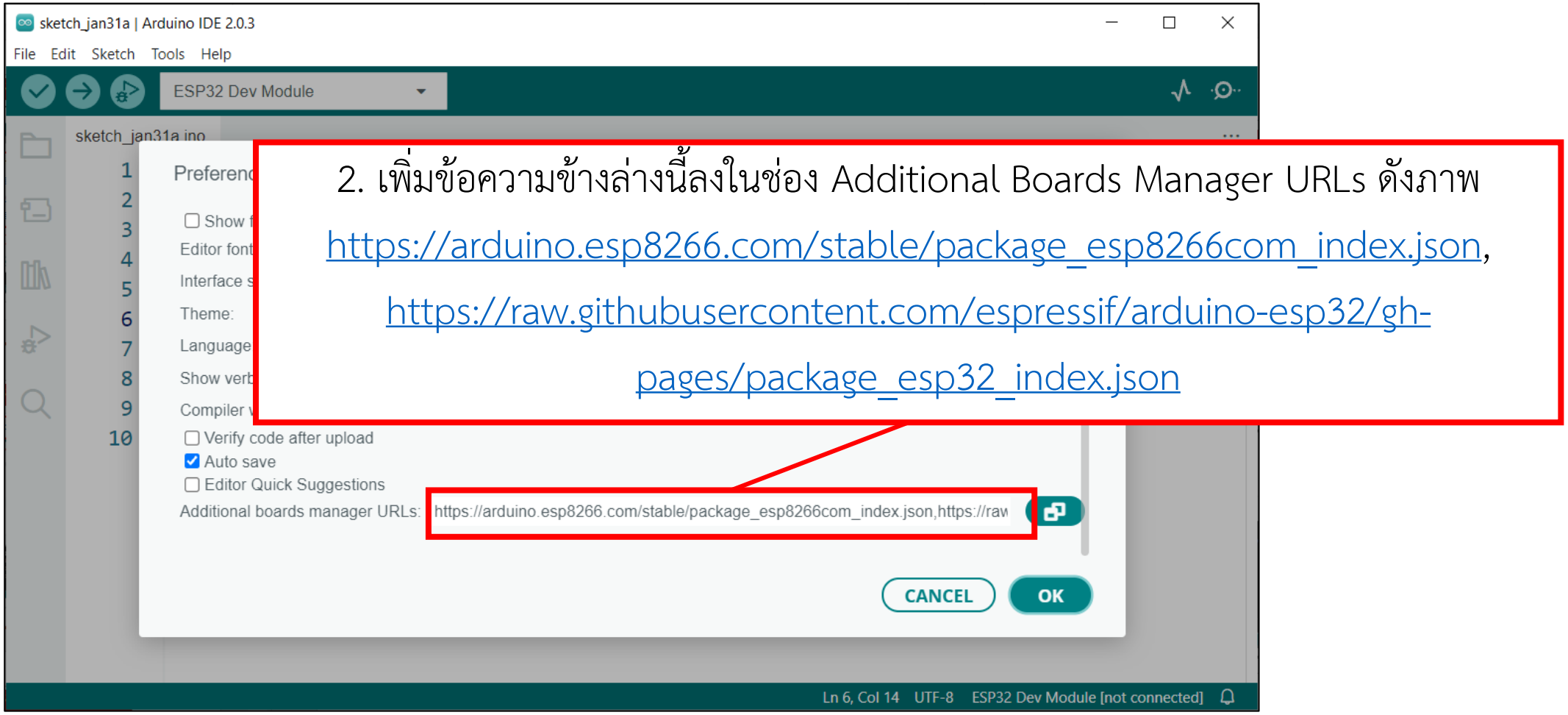
# ใบงานที่ 1.1 ขั้นตอนการทดลอง

## การทดลองที่ 2 การติดตั้งบอร์ด ESP8266 และ ESP32 บน Arduino IDE



# ใบงานที่ 1.1 ขั้นตอนการทดลอง

## การทดลองที่ 2 การติดตั้งบอร์ด ESP8266 และ ESP32 บน Arduino IDE



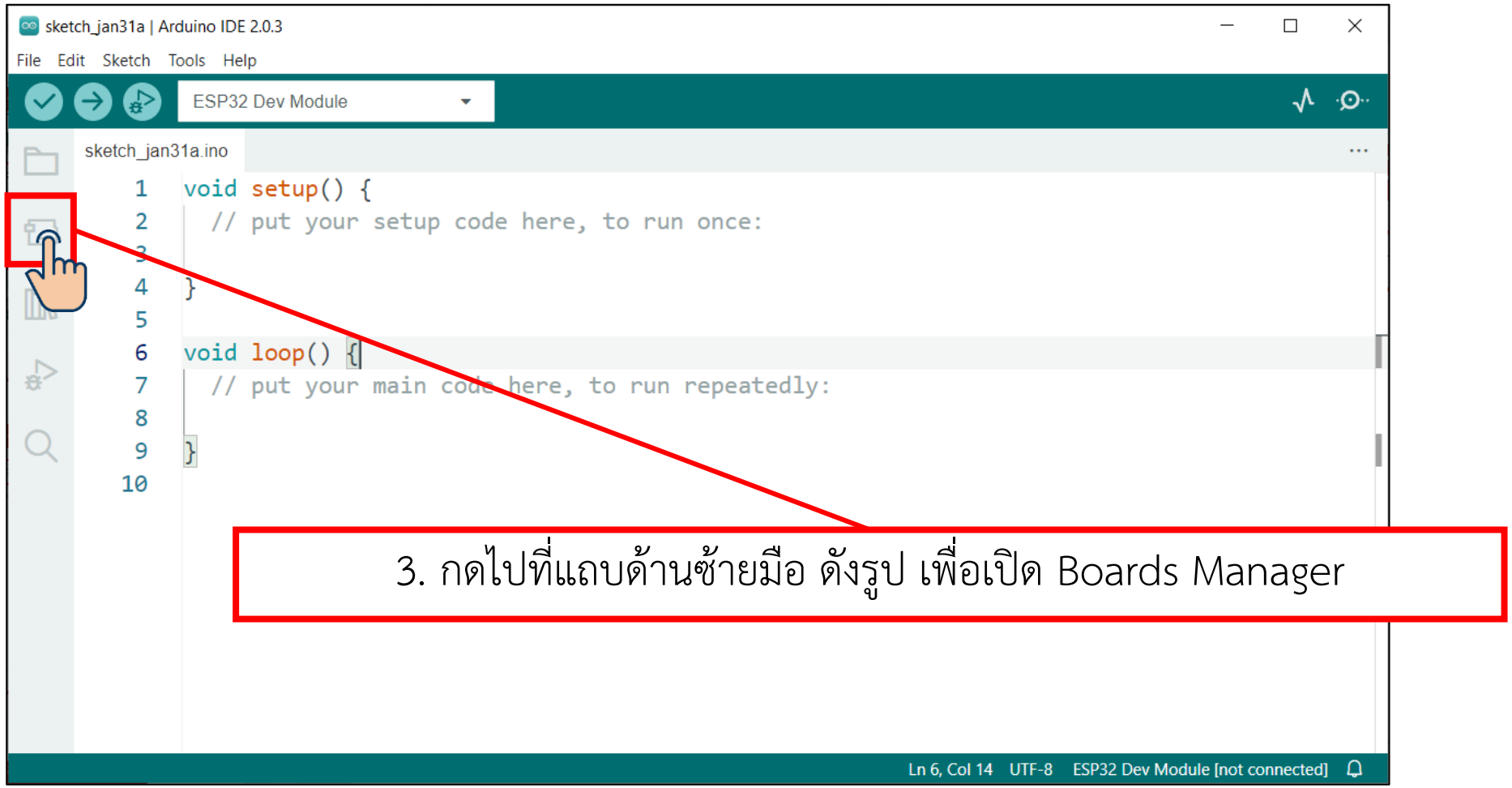

2. เพิ่มข้อความข้างล่างนี้ลงในช่อง Additional Boards Manager URLs ดังภาพ

https://arduino.esp8266.com/stable/package_esp8266com_index.json,

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

# ใบงานที่ 1.1 ขั้นตอนการทดลอง

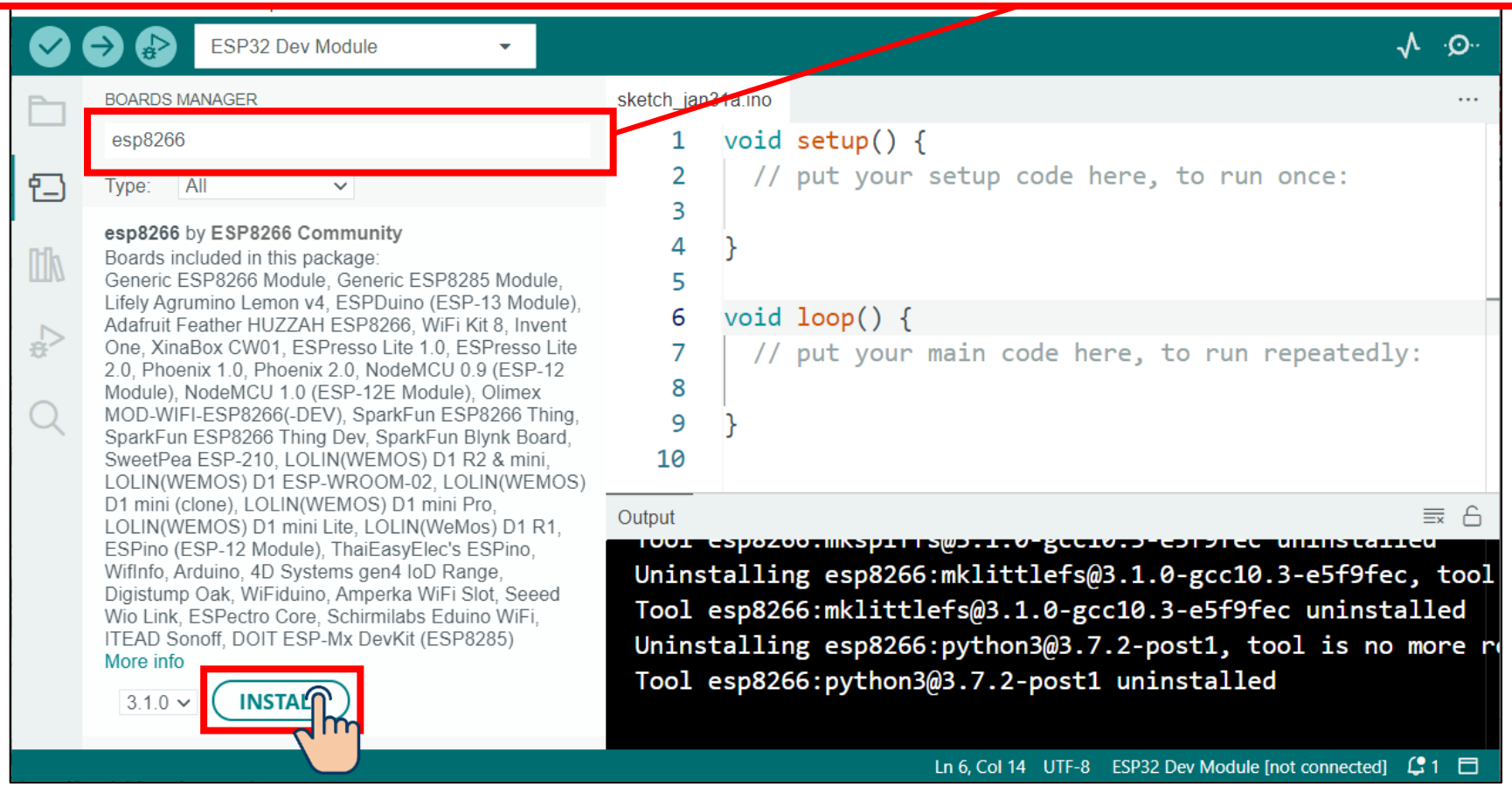
## การทดลองที่ 2 การติดตั้งบอร์ด ESP8266 และ ESP32 บน Arduino IDE



# ใบงานที่ 1.1 ขั้นตอนการทดลอง

## การทดลองที่ 2 การติดตั้งบอร์ด ESP8266 และ ESP32 บน Arduino IDE

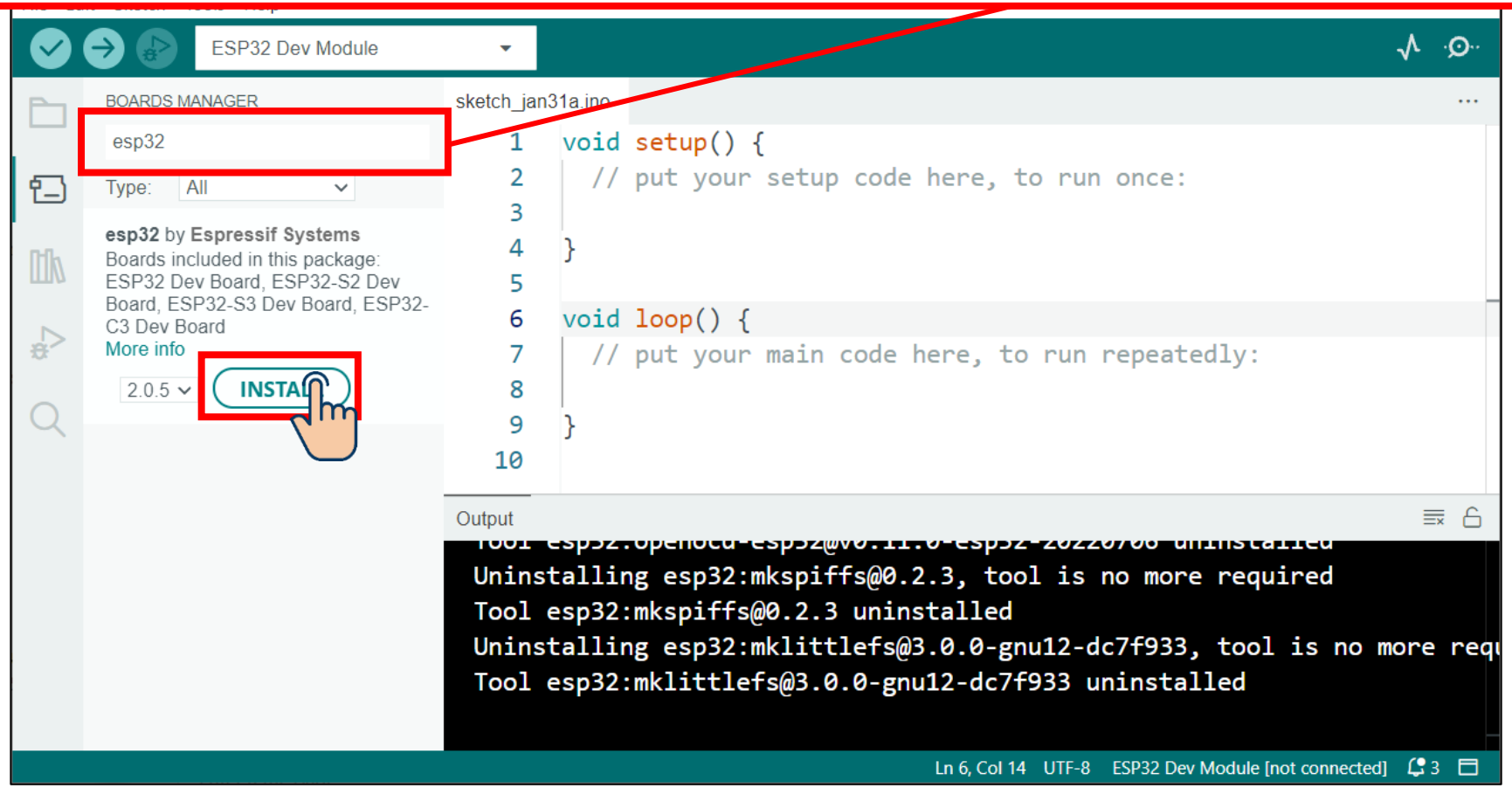
4. ให้คลิกที่ช่องค้นหาแล้วพิมพ์คำว่า “esp8266” จะปรากฏการติดตั้ง esp8266 ให้ทำการติดตั้งด้วยการคลิก Install



# ใบงานที่ 1.1 ขั้นตอนการทดลอง

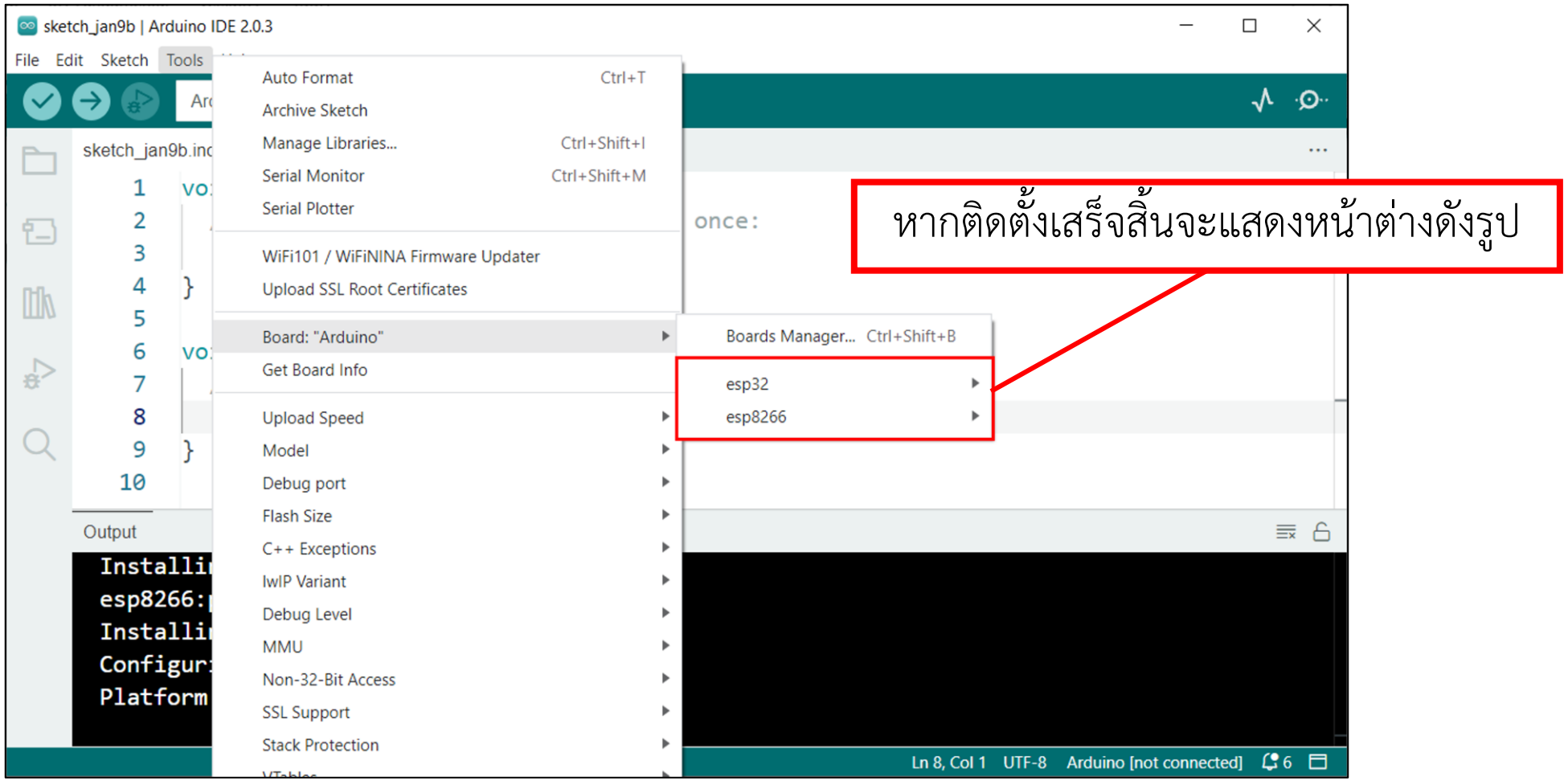
## การทดลองที่ 2 การติดตั้งบอร์ด ESP8266 และ ESP32 บน Arduino IDE

5. ให้คลิกที่ช่องค้นหาแล้วพิมพ์คำว่า “esp32” จะปรากฏการติดตั้ง esp32 ให้ทำการติดตั้งด้วยการคลิก Install



# ใบงานที่ 1.1 ขั้นตอนการทดลอง

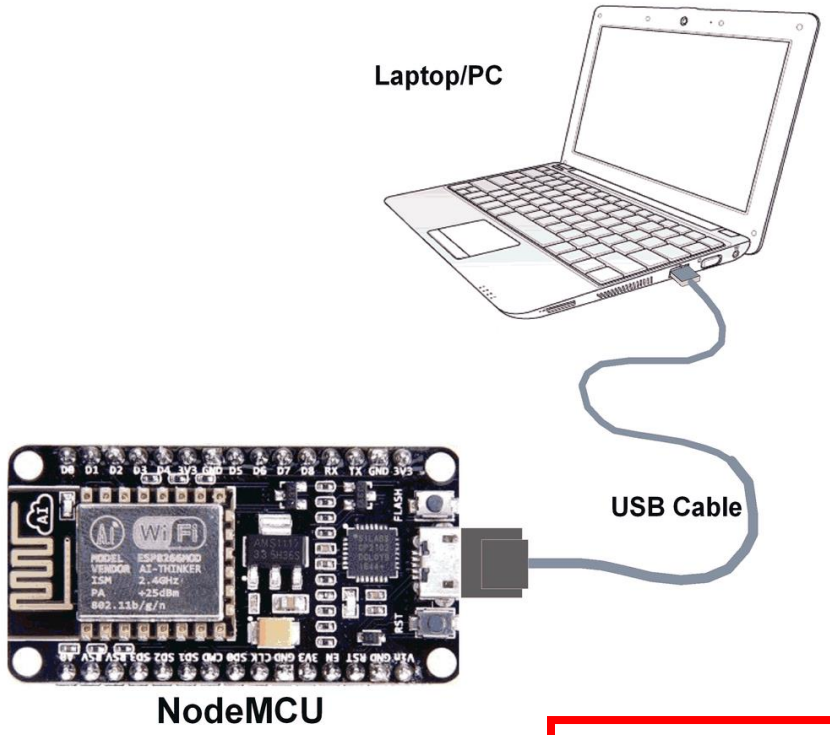
## การทดลองที่ 2 การติดตั้งบอร์ด ESP8266 และ ESP32 บน Arduino IDE



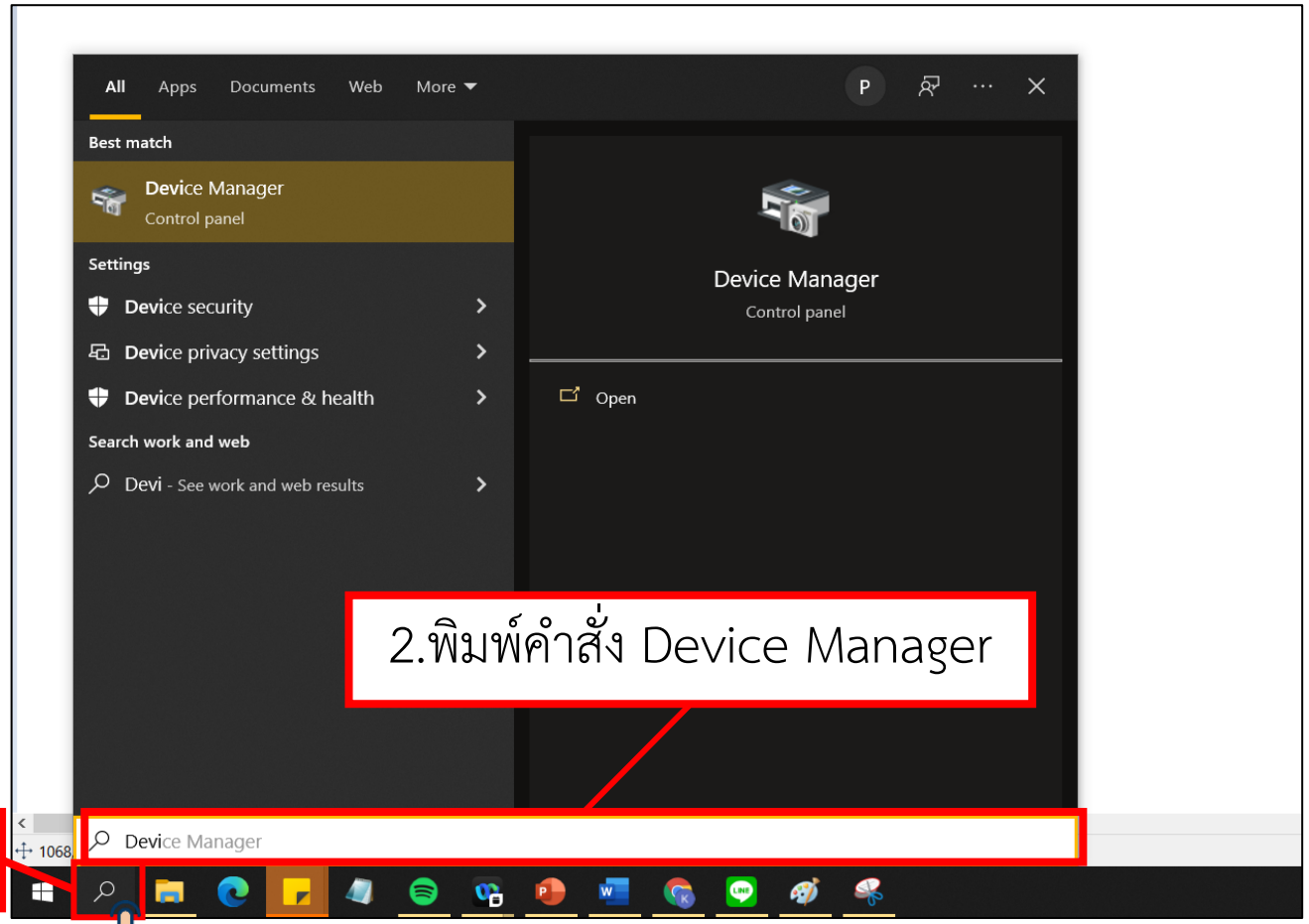
# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## การตรวจสอบพอร์ทที่ใช้สำหรับรับส่งข้อมูลสำหรับ Windows

1. เสียบสายเชื่อมต่อระหว่าง USB กับบอร์ด



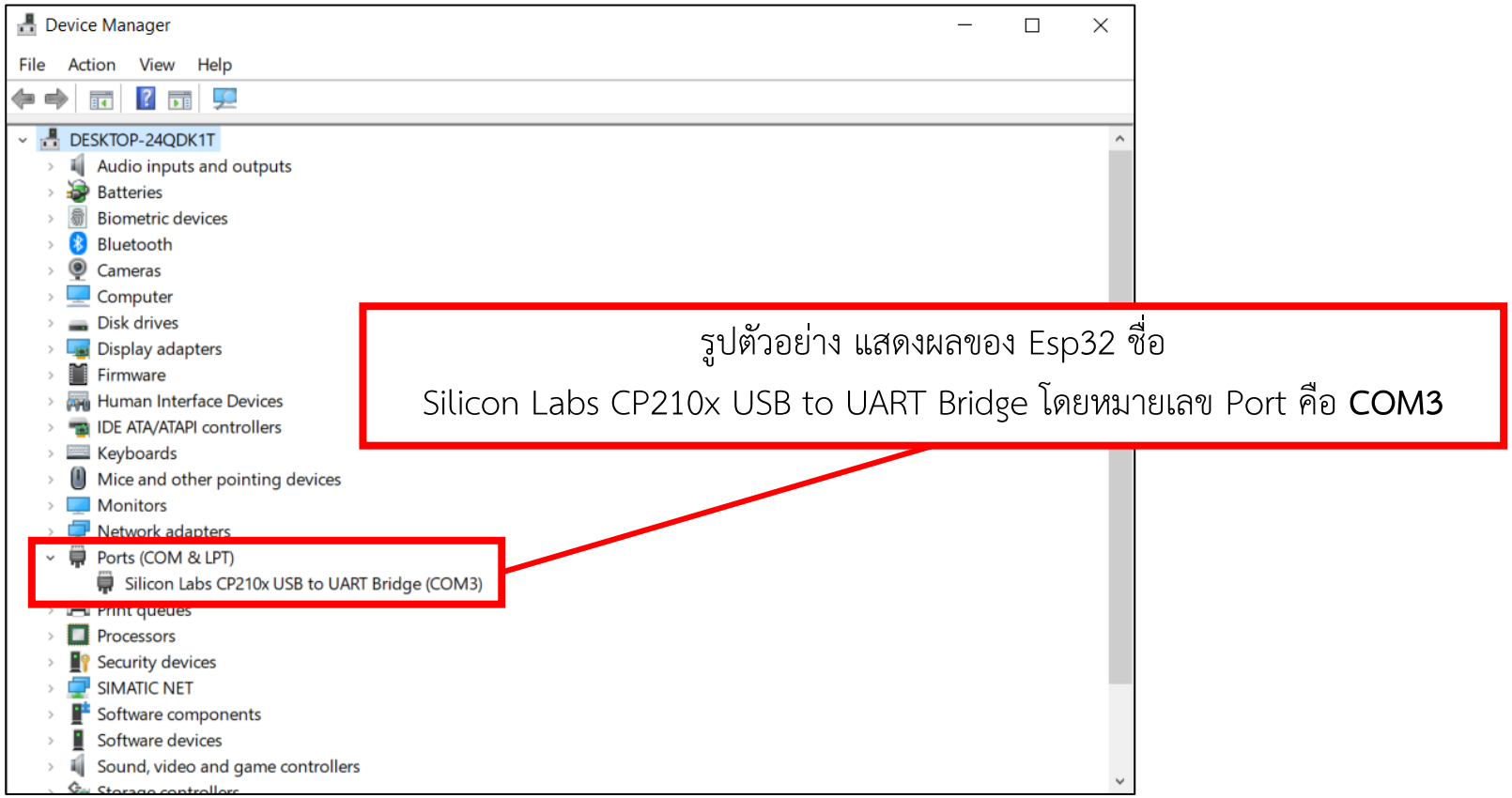

2. เปิด Window Device Manager



# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

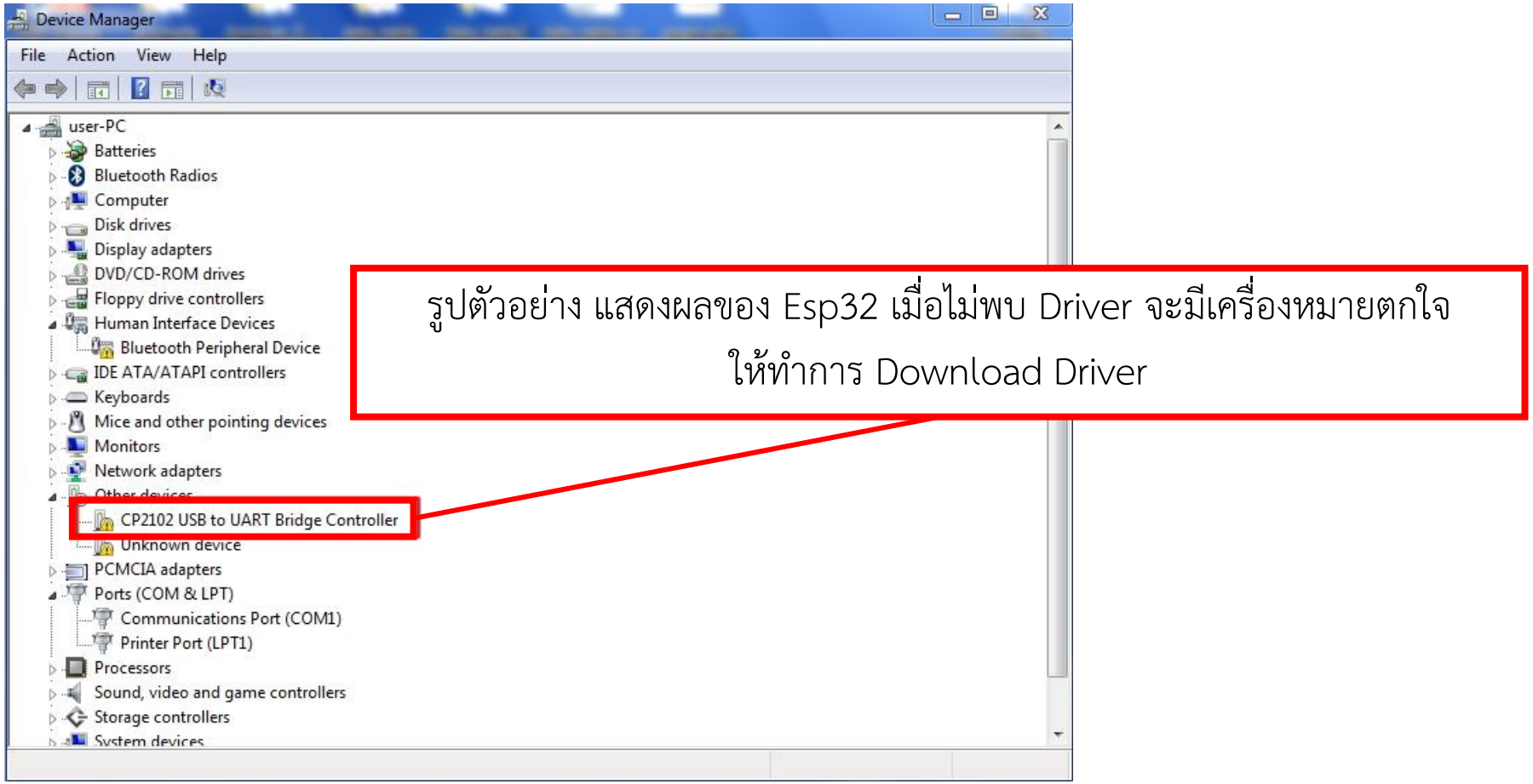
## การตรวจสอบพอร์ทที่ใช้สำหรับรับส่งข้อมูลสำหรับ Windows

3. ในหน้าต่าง Device Manager เลื่อนลงมาจนถึงส่วนของ Port ให้คลิกขยาย เพื่อตรวจสอบ Port



# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

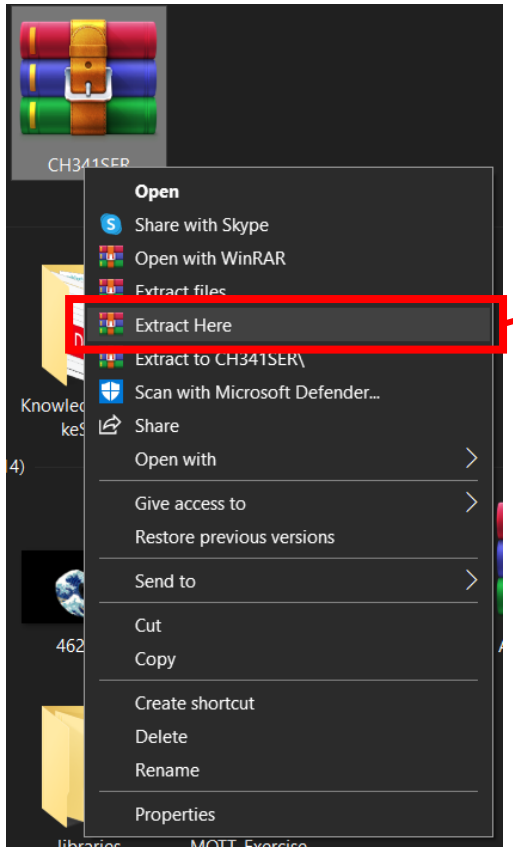
## การตรวจสอบพอร์ทที่ใช้สำหรับรับส่งข้อมูลสำหรับ Windows



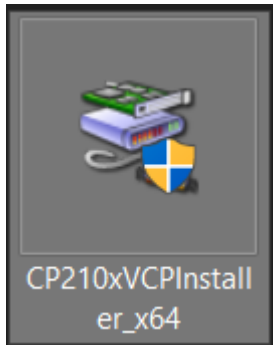
# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## การติดตั้ง Driver CH210 (ESP32)

1. Download Driver CH210 link: https://bit.ly/3KUls5b



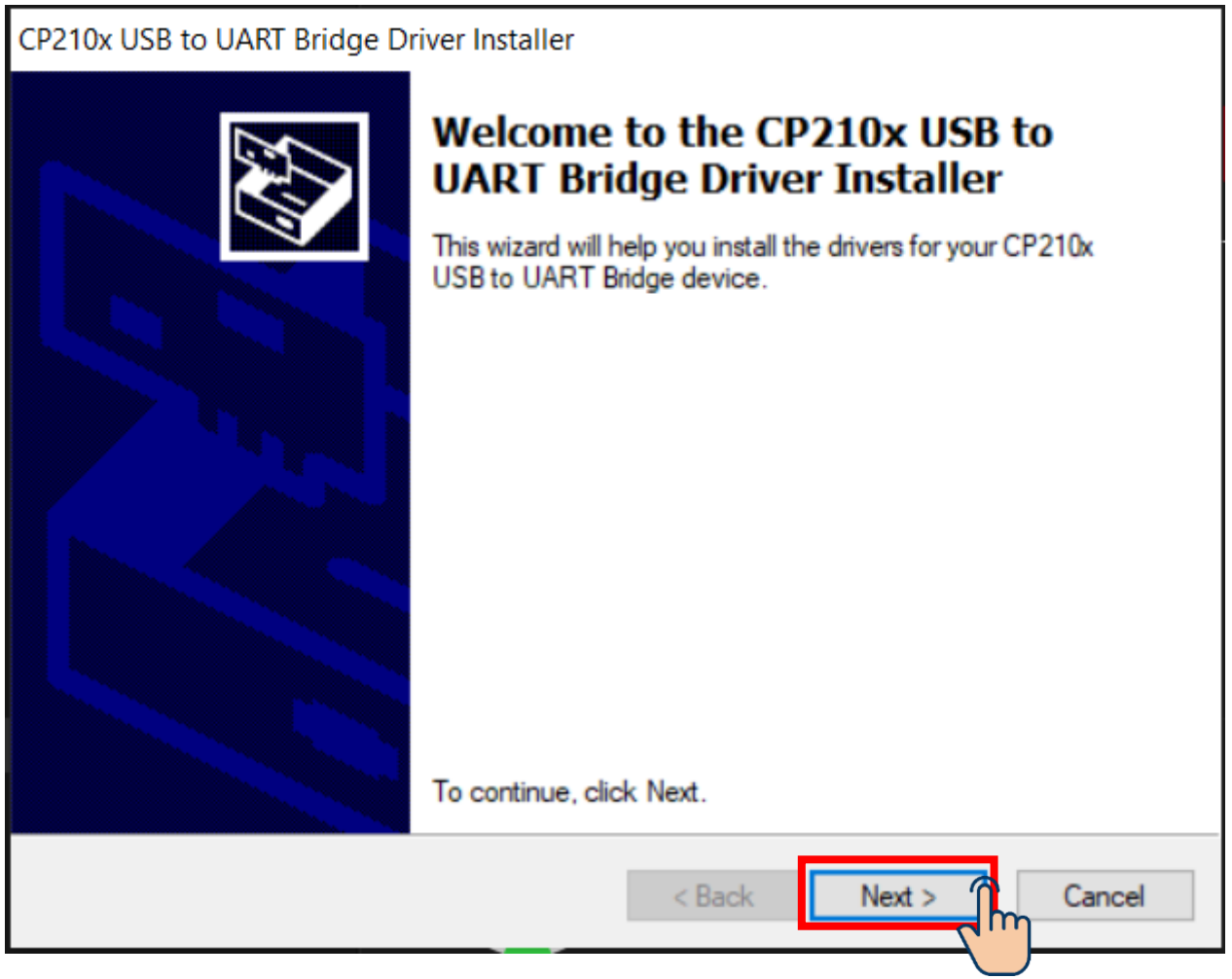

2. ให้ทำการแตกไฟล์




3. เมื่อแตกไฟล์แล้วให้ทำการเปิดไฟล์ชื่อ CP210xVCInstaller_x64

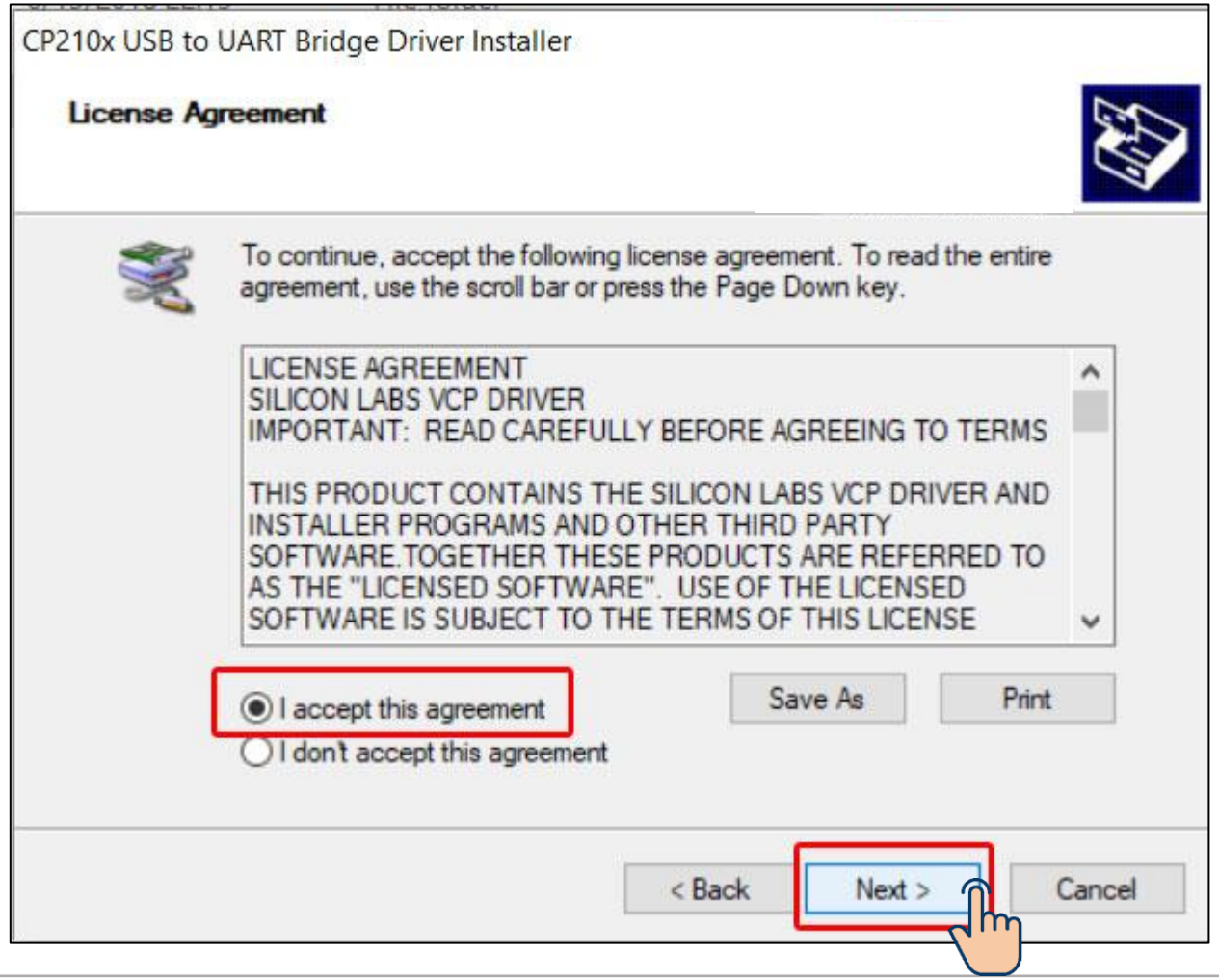
# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## การติดตั้ง Driver CH210 (ESP32)



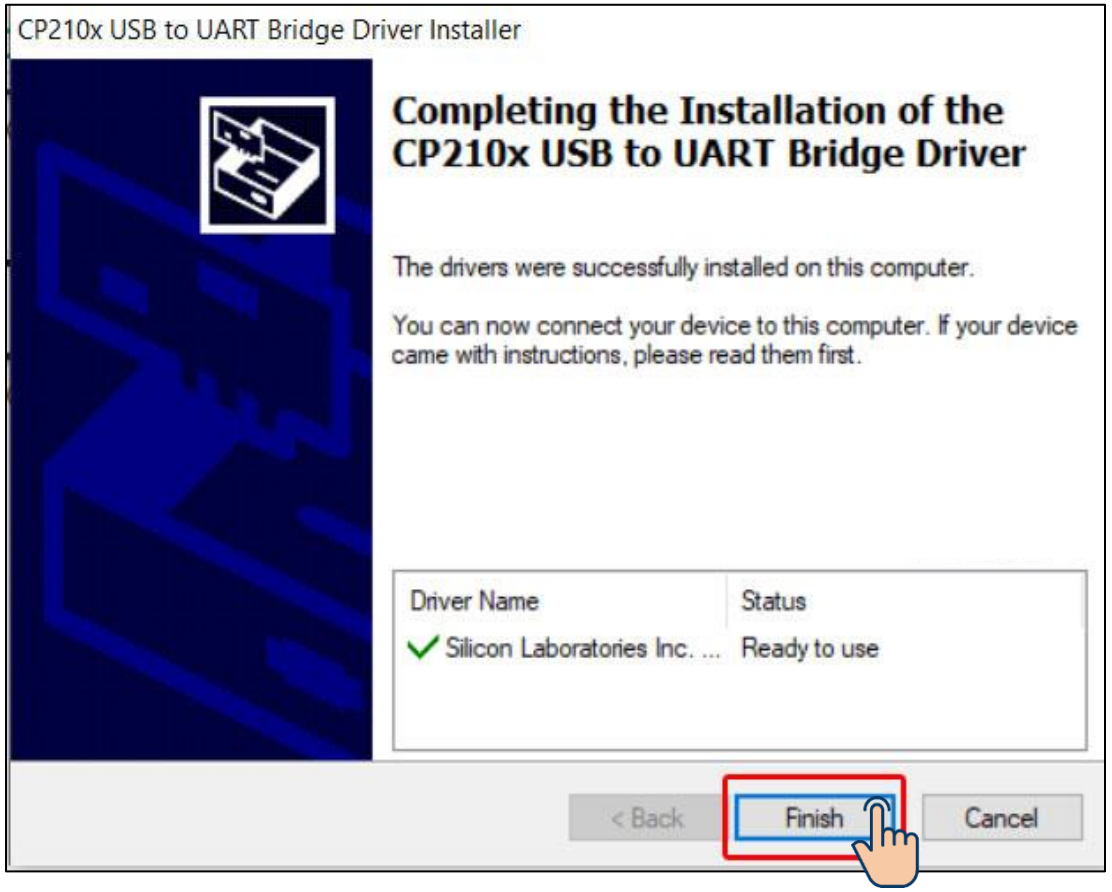
# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

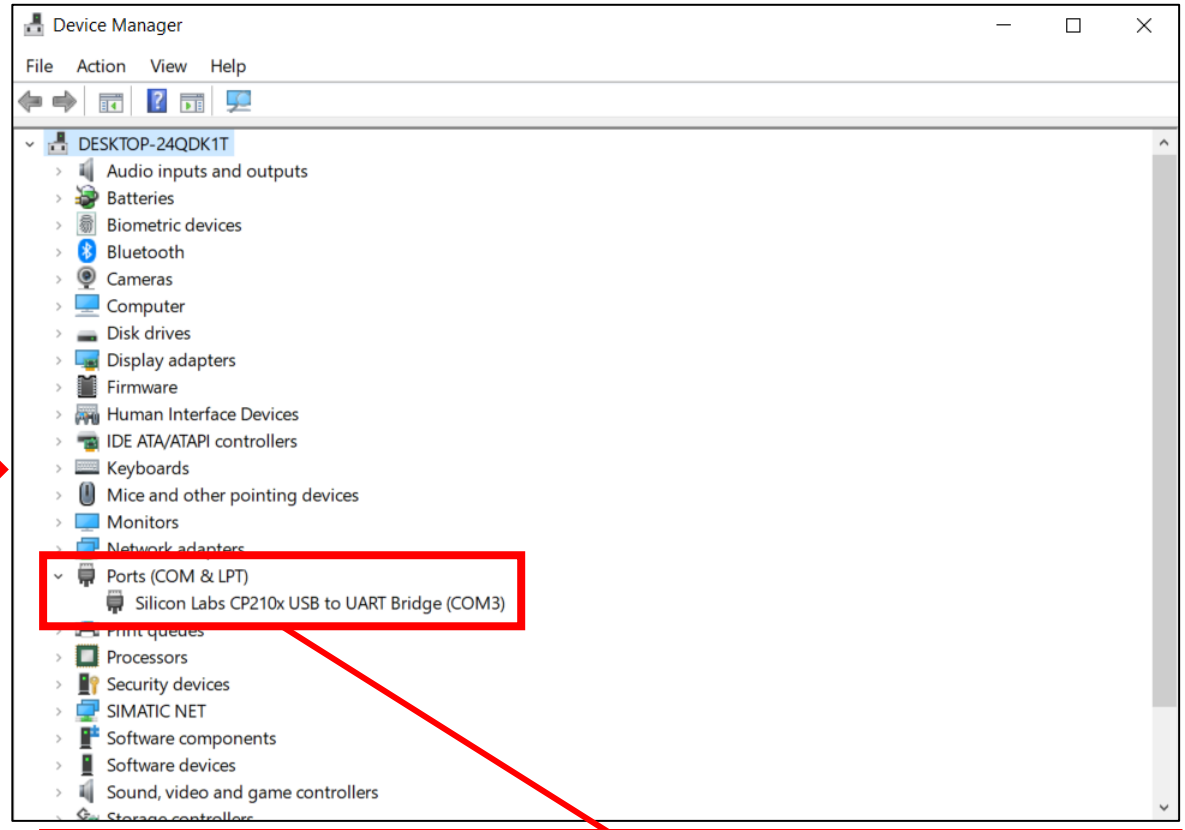
## การติดตั้ง Driver CH210 (ESP32)



# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

## การติดตั้ง Driver CH210 (ESP32)



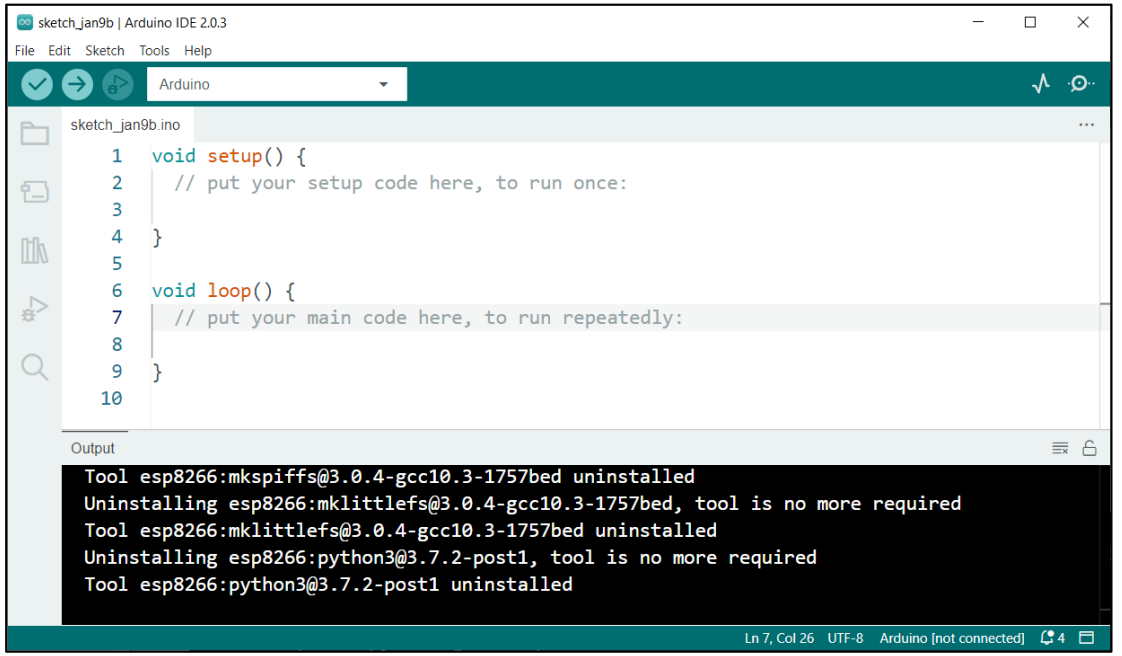

4. ให้ทำการเสียบสาย USB อีกครั้ง จากนั้นเช็คพอร์ท ก็จะพบว่าติดตั้ง Driver เรียบร้อยแล้ว

# ใบงานที่ 1.1 ทฤษฎีเบื้องต้น

ในการเขียนโปรแกรมภาษา C/C++ สำหรับ ESP32 โดยใช้ภาษาของ Arduino ซึ่งส่วนประกอบของโปรแกรมนั้นแบ่งได้เป็น 2 ส่วนหลักคือ

1. โครงสร้างภาษา ตัวแปร และค่าคงที่
2. ฟังก์ชั่น (Function)

ภาษาของ Arduino จะอ้างอิงตามภาษา C/C++ โดยเรียกใช้ฟังก์ชั่นและไลบรารีที่ทาง Arduino ได้เตรียมไว้ให้ ซึ่งสะดวก และทำให้ผู้ที่ไม่มีความรู้ด้านไมโครคอนโทรเลอร์อย่างลึกซึ้งสามารถเขียนโปรแกรมสั่งงานได้ โปรแกรมของ Arduino แบ่งได้เป็น 2 ส่วนคือ




## void setup()

ฟังก์ชั่น setup() เมื่อโปรแกรมทำงานจะทำคำสั่งของฟังก์ชั่นนี้เพียงครั้งเดียว จึงนิยมใช้ในการกำหนดค่าเริ่มต้นของการทำงาน โดยปกติใช้กำหนดโหมดการทำงานของขาต่างๆ

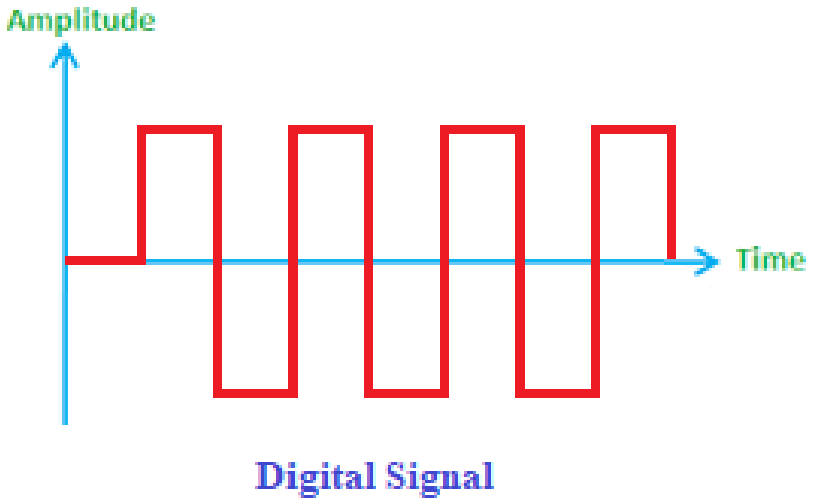
## void loop()

ฟังก์ชั่น loop() เป็นส่วนทำงาน โปรแกรมจะทำคำสั่งในฟังก์ชั่นนี้ต่อเนื่องกันตลอดเวลา เช่น อ่านค่าอินพุต ประมวลผล สั่งงานเอาต์พุต

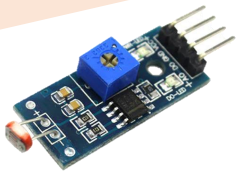
## เพิ่มเติม

สำหรับการกำหนดค่าเริ่มต้น เช่น ตัวแปร จะต้องเขียนที่ส่วนหัวของโปรแกรม ก่อนถึงตัวฟังก์ชั่น นอกจากนั้น ยังต้องคำนึงถึงตัวพิมพ์เล็ก-ใหญ่ของตัวแปรและชื่อฟังก์ชั่นให้ถูกต้องด้วย

# ใบงานที่ 1.2 การควบคุมสัญญาณดิจิตอล




Analog Signal


LDR Sensor


MQ2 Sensor

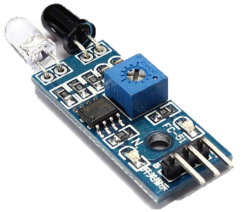
Ultrasonic Sensor

PTM

Digital Signal


Push bottom switch


PIR Sensor

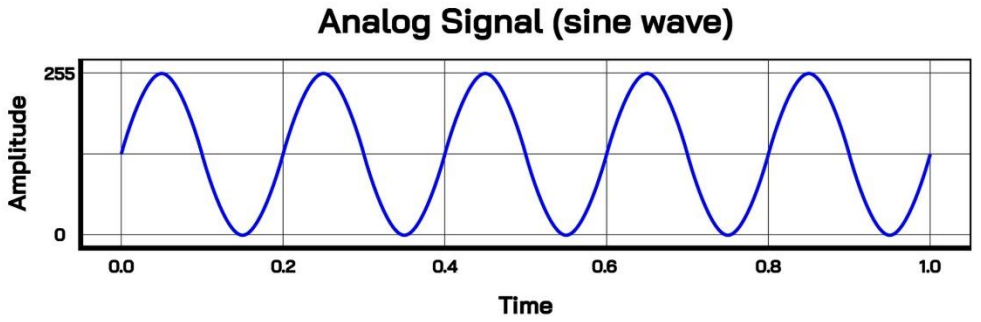

Infrared Sensor

# ใบงานที่ 1.2 ทฤษฎีเบื้องต้น

## สัญญาณ Digital และ Analog

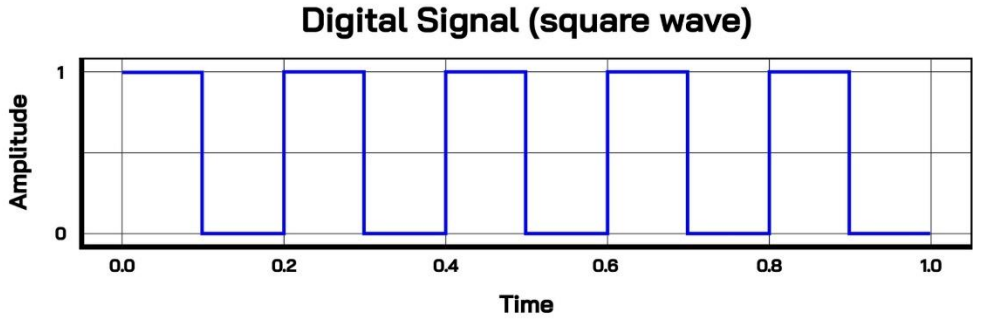
### Analog Pin

สัญญาณAnalogเป็นสัญญาณที่มีการเปลี่ยนแปลงหรือการเคลื่อนที่ของข้อมูลแบบต่อเนื่อง โดยสัญญาณจะมีขนาดไม่คงที่ สามารถส่งข้อมูลได้ไกล



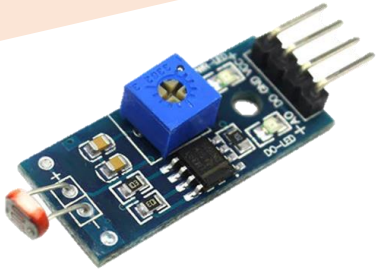

### Digital Pin

สัญญาณ Digital เป็นสัญญาณที่ไม่มีความต่อเนื่องทางเวลา โดยมีค่า 0 กับ 1 หรือ Low กับ High เท่านั้น

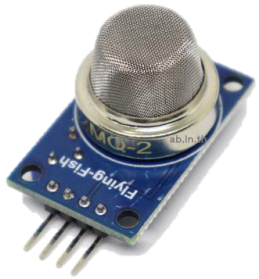


# ใบงานที่ 1.2 ทฤษฎีเบื้องต้น

## Analog Signal



LDR Sensor



MQ2 Sensor

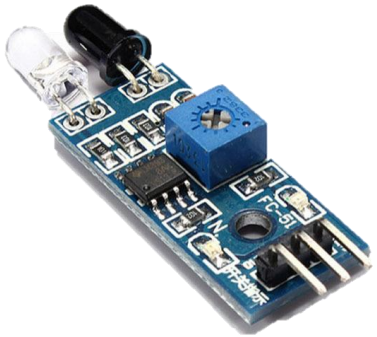

Ultrasonic Sensor


PTM

## Digital Signal

Push bottom switch

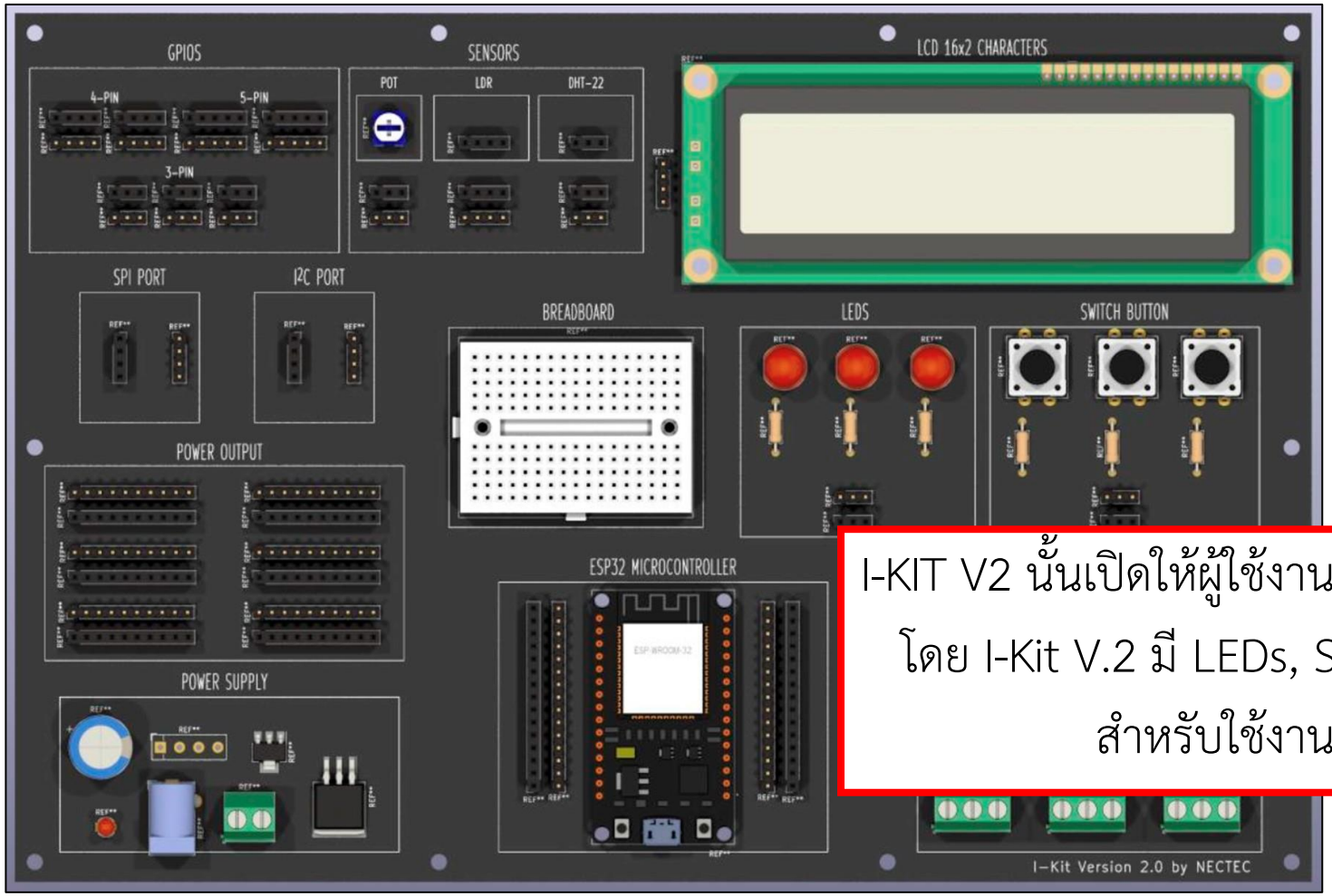
PIR Sensor



Infrared Sensor

# ใบงานที่ 1.2 ทฤษฎีเบื้องต้น

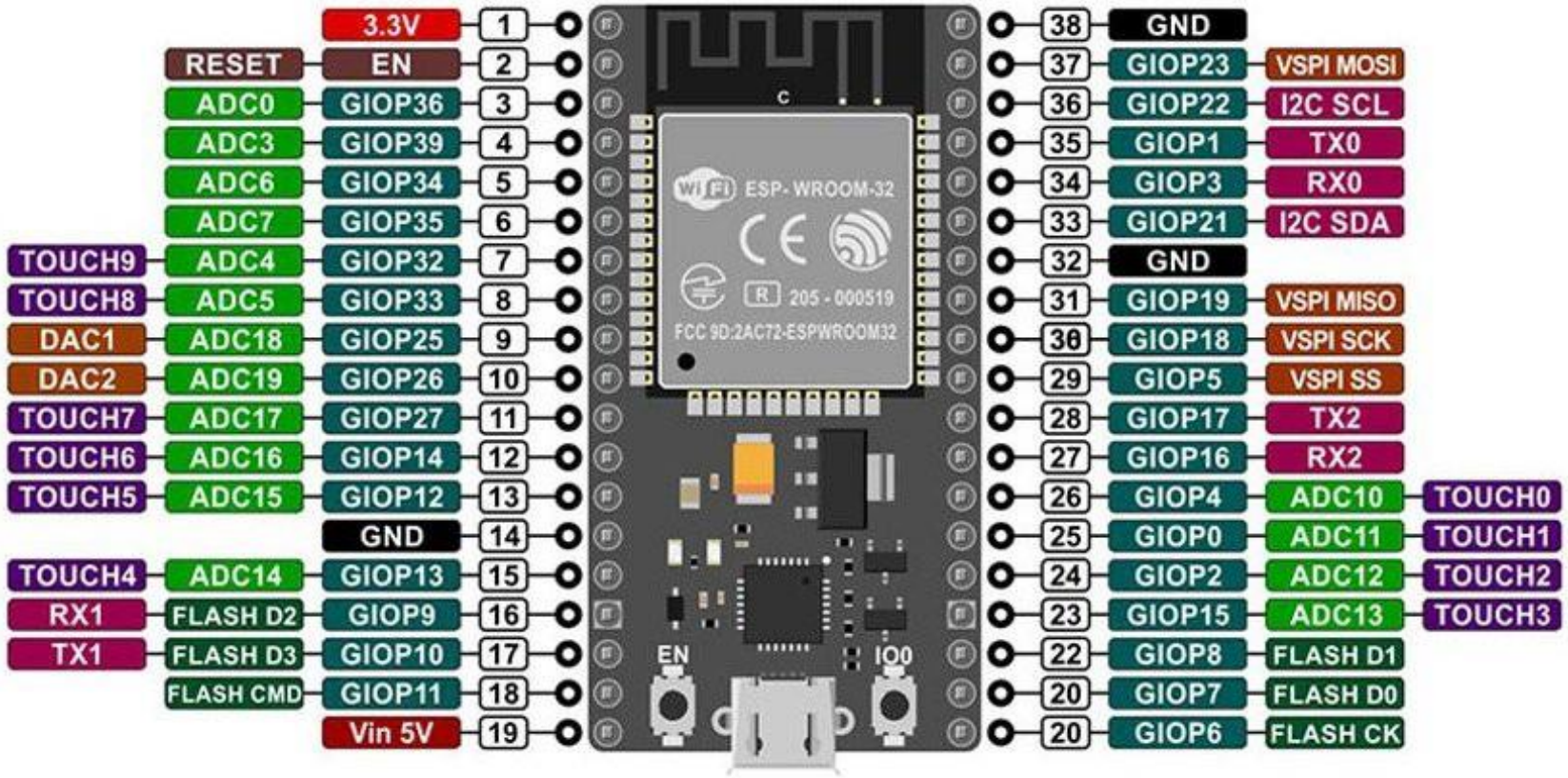
## I-KIT V.2




I-KIT V2 นั้นเปิดให้ผู้ใช้งานสามารถเลือกการใช้ GPIO ได้ตามที่ต้องการ โดย I-Kit V.2 มี LEDs, Switch Buttons, Relays, LCD หรือ ช่องสำหรับใช้งาน Sensors ต่างๆ บางส่วนไว้ให้

# ใบงานที่ 1.2 ทฤษฎีเบื้องต้น

## ESP32s Pinout



# ใบงานที่ 1.2 ทฤษฎีเบื้องต้น

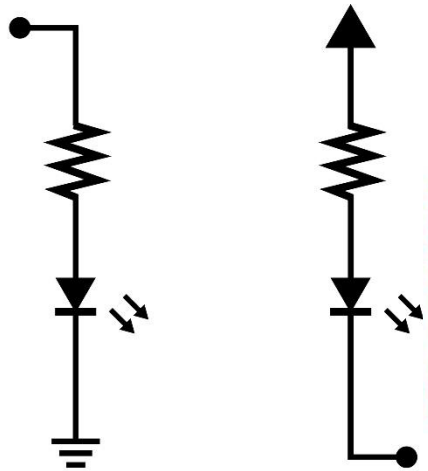
## การติดต่อ Digital Output

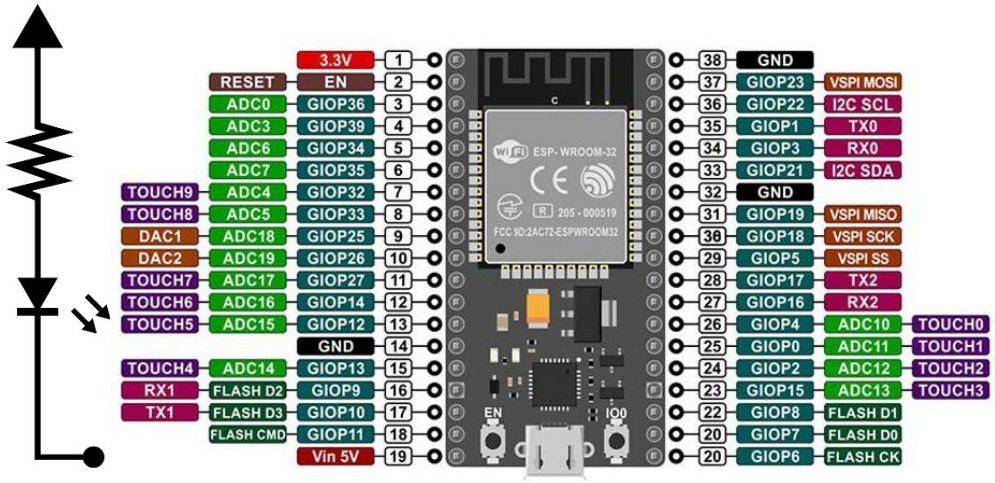
### อุปกรณ์ Output

Light Emitting Diode (LED) สัญญาณแบบ Digital OUTPUT

ในการเปิด/ปิด LED มีอยู่ด้วยกันสองวิธี คือ

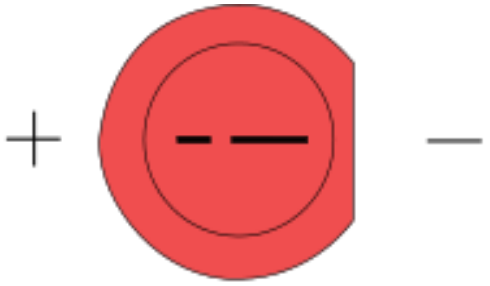
1. Active High เมื่อส่งลอจิก HIGH หรือ 1 จาก NodeMCU หลอด LED จะติด และหากส่ง LOW หรือ 0 หลอด LED จะดับ
2. Active Low เมื่อส่งลอจิก LOW หรือ 0 จากบอร์ดหลอด LED จะติด และหากส่ง HIGH หรือ 1 หลอด LED จะดับ



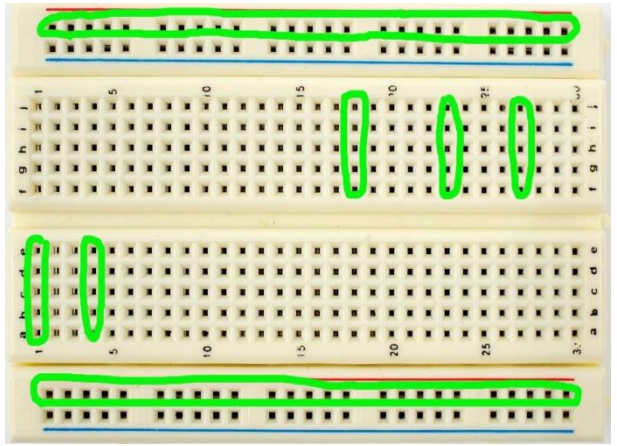



**Active Hight** **Active Low**

LED สีแดง, เขียว, เหลือง



วงจร LED

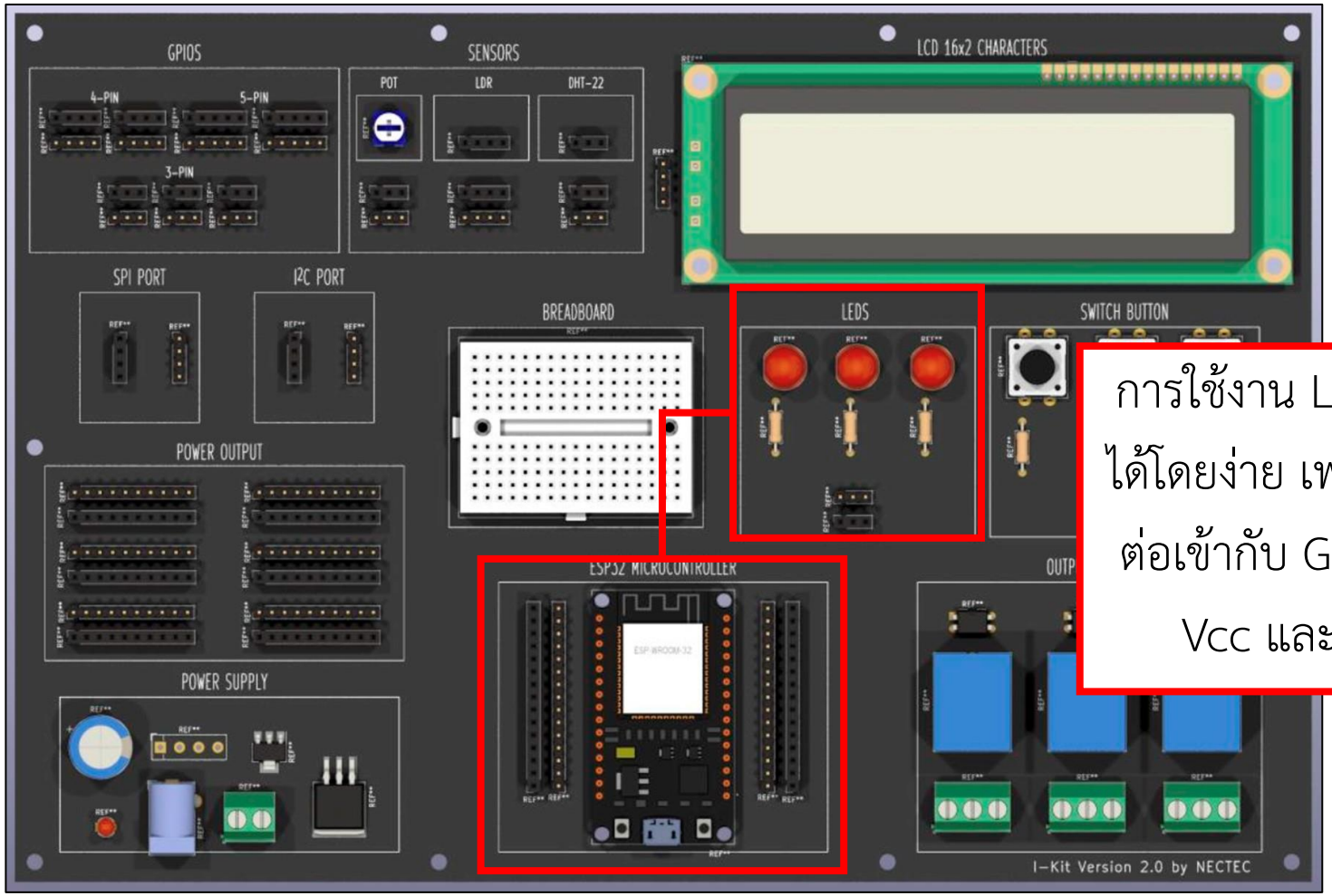


แผงวงจรของ Breadboard

ESP32 มี LED ติดมาด้วยอยู่แล้ว 1 ดวงที่ขา GPIO2 โดยต่อแบบ Active High

# ใบงานที่ 1.2 ทฤษฎีเบื้องต้น

## การใช้งาน LED




การใช้งาน LED บน I-KIT V.2 นั้น สามารถใช้งานได้โดยง่าย เพราะในช่อง LEDs มีพอร์ทสำหรับการต่อเข้ากับ GPIO ของ ESP32 โดยมีการต่อเข้ากับ Vcc และ GND ให้แล้วแบบ Active LOW

# ใบงานที่ 1.2 ทฤษฎีเบื้องต้น

## Function และคำสั่ง ในการติดต่อ Digital Output

### #define name value/pin

การกำหนด #define ปกติแล้วเป็นการกำหนดค่าหรือขาของบอร์ดต่างๆให้มีค่าคงที่ตามที่กำหนด เช่น **#define D0 16**

### pinMode(pin ,mode)

pinMode เป็นฟังก์ชันที่ใช้ในการตั้งค่าโหมดการทำงานให้กับขาของบอร์ด ไม่ว่าจะเป็น OUTPUT, INPUT, INPUT_PULLUP เช่น **pinMode(D0,OUTPUT);**

### digitalWrite(pin ,logic)

digitalWrite(pin, logic) เป็นฟังก์ชันในการเขียนค่าไปที่ขาของบอร์ด เช่น **digitalWrite(D0,HIGH);**

### delay(ms)

delay(ms) เป็นการหน่วงเวลาการทำงานของโปรแกรมโดยมีหน่วยเป็น millisecond เช่น **delay(500);**

# ใบงานที่ 1.2 ทฤษฎีเบื้องต้น

## ตัวอย่างการควบคุมพอร์ต Digital Output

โดยให้เดินสายไฟจาก GPIO5 ไปยัง LED1

การสั่งเปิด LED

```
#define LED1 5
void setup()
{
  pinMode(LED1, OUTPUT);
}
void loop()
{
  digitalWrite(LED1, LOW);
}
```

การสั่งปิด LED

```
#define LED1 5
void setup()
{
  pinMode(LED1,OUTPUT);
}
void loop()
{
  digitalWrite(LED1, HIGH);
}
```

# ใบงานที่ 1.2 ขั้นตอนการทดลอง

## การทดลองที่ 1 การสั่งให้ LED ภายนอกที่เชื่อมต่อ ESP32 กระพริบ

การสั่งไฟกระพริบ

```
#define LED1 5
void setup()
{
  pinMode(LED1, OUTPUT);
}
void loop()
{
  digitalWrite(LED1, LOW);
  delay(1000);
  digitalWrite(LED1, HIGH);
  delay(1000);
}
```

การสั่ง LED บน ESP32 กระพริบ (เสริม)

```
#define LED 2
void setup()
{
  pinMode(LED, OUTPUT);
}
void loop()
{
  digitalWrite(LED, HIGH);
  delay(1000);
  digitalWrite(LED, LOW);
  delay(1000);
}
```

# ใบงานที่ 1.2 ขั้นตอนการทดลอง

## การทดลองที่ 2 การสั่งให้ LED ภายนอกและภายในที่เชื่อมต่อ ESP8266 กระพริบ

การสั่ง LED ภายนอก และบน ESP32 กระพริบ

```
#define LED1 5
#define LED 2
void setup()
{
  pinMode(LED1, OUTPUT);
  pinMode(LED, OUTPUT);
}
void loop()
{
  digitalWrite(LED1, LOW);
  delay(1000);
  digitalWrite(LED1, HIGH);
  delay(1000);
  digitalWrite(LED, HIGH);
  delay(1000);
  digitalWrite(LED, LOW);
  delay(1000);
}
```

# ใบงานที่ 1.2 ขั้นตอนการทดลอง

## การทดลองที่ 3 การสั่งให้ LED ภายนอกที่เชื่อมต่อ ESP32 กระพริบ

การสั่ง LED ภายนอกด้วย ESP32

```
#define LED1 5
#define LED2 18
#define LED3 19

void setup()
{
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
}

void loop()
{
digitalWrite(LED1, LOW);
  delay(1000);
  digitalWrite(LED1, HIGH);
  delay(1000);
  digitalWrite(LED2, LOW);
  delay(1000);
  digitalWrite(LED2, HIGH);
  delay(1000);
  digitalWrite(LED3, LOW);
  delay(1000);
  digitalWrite(LED3, HIGH);
  delay(1000);
}
```

เมื่อเขียน Code เสร็จเรียบร้อย ให้ทำการเปลี่ยน Board: ESP32 Dev Module และ Port ก่อนจะ Upload

# ใบงานที่ 1.2 คำถามท้ายหน่วยการเรียนรู้

**คำถามท้ายหน่วยการเรียนรู้ที่ 3**

จงเขียนโปรแกรมควบคุม LED บน ESP32 DEVKIT โดยมีลักษณะการทำงานดังนี้

1. เริ่มต้นโปรแกรม LED1, LED2 และ LED3 จะต้องกระพริบพร้อมกันจำนวน 1 ครั้ง
2. หลังจากกระพริบพร้อมกันแล้วให้ไฟติดเรียงกันจาก LED1, LED2 และ LED3
3. หลังจากติดเรียงแล้วให้ไฟดับเรียงกันจาก LED3, LED2 และ LED1
4. หลังจากนั้นให้ LED1, LED2 และ LED3 กระพริบพร้อมกันจำนวน 2 ครั้ง
5. และเมื่อสิ้นสุดข้อที่ 4 ให้โปรแกรมกลับไปทำงานข้อที่ 1

(โดยกำหนด delay ของโปรแกรมเป็นเวลา 0.5 วินาที)

**คำถามท้ายหน่วยการเรียนรู้ที่ 4**

จงเขียนโปรแกรมควบคุม LED บน ESP32 DEVKIT โดยมีลักษณะการทำงานดังนี้

1. เริ่มต้นโปรแกรม LED1 จะต้องกระพริบจำนวน 3 ครั้ง
2. หลังจากนั้นให้ LED2 จะต้องกระพริบจำนวน 5 ครั้ง
3. หลังจากนั้นให้ LED3 จะต้องกระพริบจำนวน 2 ครั้ง
4. แล้วให้ LED1, LED2 และ LED3 จะต้องกระพริบพร้อมกันจำนวน 3 ครั้ง
5. และเมื่อสิ้นสุดข้อที่ 4 ให้โปรแกรมกลับไปทำงานข้อที่ 1

(โดยกำหนด delay ของโปรแกรมเป็นเวลา 0.5 วินาที)

จาก Exercise 2 จะสังเกตได้ว่าเป็นการเขียนโปรแกรมที่ค่อนข้างลำบากเพราะใช้จำนวนบรรทัดในการเขียนจำนวนมาก เป็นไปได้ไหมที่จะมีวิธีที่อำนวยความสะดวกตรงนี้???

# ใบงานที่ 1.3 การเขียนโปรแกรมวนลูป (For Loop)

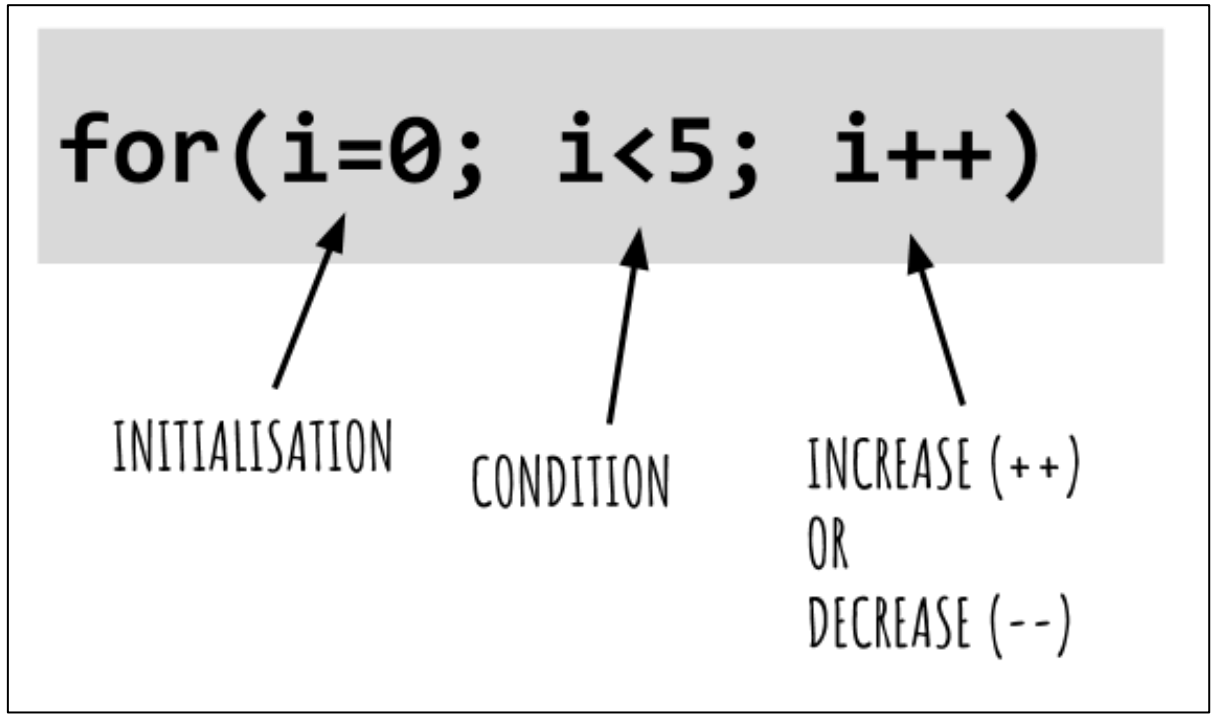
sketch_jan31a | Arduino IDE 2.0.3

File Edit Sketch Tools Help

ESP32 Dev Module

sketch_jan31a.ino

```
1  void setup() {
2    Serial.begin(9600);
3    for (int i = 0; i < 10; i++){
4      Serial.println("Value of i is: " + String(i));
5    }
6
7  }
8
9  void loop() {
10   // put your main code here, to run repeatedly:
11
12 }
```



# ใบงานที่ 1.3 ทฤษฎีเบื้องต้น

## สรุปเบื้องต้นในการเขียนโปรแกรม

### 1. Preprocessor Directives

โดยปกติแล้วเกือบทุกโปรแกรมต้องมี โดยส่วนนี้จะเป็นส่วนที่คอมไพลเลอร์จะมีการประมวลผลและทำตามคำสั่งก่อนที่จะมีการคอมไพล์โปรแกรม ซึ่งจะเริ่มต้นด้วยเครื่องหมายไดเร็กทีฟ (directive) หรือเครื่องหมายสี่เหลี่ยม # แล้วจึงตามด้วยชื่อคำสั่งที่ต้องการเรียกใช้ หรือกำหนด โดยปกติแล้วส่วนนี้จะอยู่ในส่วนบนสุด หรือส่วนหัวของโปรแกรม และต้องอยู่นอกฟังก์ชั่นหลักใดๆก็ตาม เช่น

```
#include <Time.h>

#include “myFunction.h”

#define LEDPIN 13
```

**การอ้างอิงไฟล์จากภายใน หรือการอ้างอิงไฟล์ไลบารี่ที่มีอยู่แล้วใน Arduino หรือเป็นไลบารี่ที่เพิ่มเข้าไปเอง จะใช้เครื่องหมาย <> ในการคร่อมชื่อไฟล์ไว้ เพื่อให้โปรแกรมคอมไพลเลอร์เข้าใจว่าควรไปหาไฟล์เหล่านี้จากในโฟลเดอร์ไลบารี่ แต่หากต้องการอ้างอิงไฟล์ที่อยู่ในโฟลเดอร์โปรเจค จะต้องใช้เครื่อหมาย "" คร่อมแทน ซึ่งคอมไพล์เลอร์จะวิ้งไปหาไฟล์นี้โดยอ้างอิงจากไฟล์โปรแกรมที่คอมไพล์เลอร์อยู่**

# ใบงานที่ 1.3 ทฤษฎีเบื้องต้น

## สรุปเบื้องต้นในการเขียนโปรแกรม

### 2. ส่วนของการกำหนดค่า (Global Declarations)

ส่วนนี้จะเป็นส่วนที่ใช้ในการกำหนดชนิดตัวแปรแบบนอกฟังก์ชัน หรือประกาศฟังก์ชัน เพื่อให้ฟังก์ชันที่ประกาศสามารถกำหนด หรือเรียกใช้ได้จากทุกส่วนของโปรแกรม

```
int pin = 13;
```

### 3. ฟังก์ชัน setup() และฟังก์ชัน loop()

ฟังก์ชั่น setup() และฟังก์ชั่น loop() เป็นคำสั่งที่ถูกบังคับให้ต้องมีในทุกโปรแกรม โดยฟังก์ชั่น setup() จะเป็นฟังก์ชั่นแรกที่ถูกเรียกใช้ นิยมใช้กำหนดค่า หรือเริ่มต้นใช้งานไลบารี่ต่างๆ เช่น ในฟังก์ชั่น setup() จะมีคำสั่ง pinMode() เพื่อกำหนดให้ขาใดๆก็ตามเป็นดิจิตอลอินพุต หรือเอาต์พุต ส่วนฟังก์ชั่น loop() จะเป็นฟังก์ชั่นที่ทำงานหลังจากฟังก์ชั่น setup() ได้ทำงานเสร็จสิ้นไปแล้ว และมีการวนรอบแบบไม่รู้จบ เมื่อฟังก์ชั่น loop() งานครบตามคำสั่งแล้ว ฟังก์ชั่น loop() ก็จะถูกเรียกขึ้นมาใช้อีก

# ใบงานที่ 1.3 ทฤษฎีเบื้องต้น

## สรุปเบื้องต้นในการเขียนโปรแกรม

### 4. การสร้างฟังก์ชัน และการใช้งานฟังก์ชัน (Users-defined function)

ในการสร้างฟังก์ชั่นขึ้นมา คำสั่งต่างๆที่อยู่ภายในฟังก์ชั่น ต้องอยู่ภายใต้เครื่องหมายปีกกาเปิด { และปีกกาปิด } เท่านั้น ภายใต้เครื่องหมาย {} เราสามารถนำฟังก์ชั่นหรือคำสั่งใดๆก็ได้มาใส่ไว้ แต่จะต้องคั่นแต่ละคำสั่งด้วยเครื่องหมาย ; โดยจะนำคำสั่งทั้งหมดไว้บรรทัดเดียวกันเลย หรือแยกบรรทัดกันก็ได้เพื่อความสวยงามของโค้ด (ไม่มีผลกับขนาดของโปรแกรมหลังคอมไพล์)

**ตัวอย่างการสร้างฟังก์ชัน**

```
void Mode(int pin) {
pinMode(pin,OUTPUT); }
void setup(){
Mode(13) }
```

### 5. ส่วนอธิบายโปรแกรม (Program Comments)

ส่วนอธิบายโปรแกรม หรือการคอมเม้นโปรแกรมเป็นส่วนที่สำคัญอย่างมากที่จะช่วยให้ผู้ที่ไม่ได้เขียนโปรแกรมหรือเป็นผู้เขียนโปรแกรมเข้าใจโปรแกรมได้ง่ายขึ้นโดยอ่านจากคอมเม้น แทนการทำความเข้าใจโปรแกรมโดยอ่านแต่ละฟังก์ชั่น ส่วนอธิบายโปรแกรม หรือส่วนคอมเม้นนี้ จะไม่มีผลใดๆกับขนาดของโปรแกรมหลังคอมไพล์ เนื่องจากส่วนนี้จะถูกตัดทิ้งทั้งหมดเนื่องจากไม่ได้ถูกนำไปใช้งาน มีผลเพียงแค่ว่าไฟล์โค้ดโปรแกรมจะใหญ่ขึ้นมา หากมีการคอมเม้นโค้ดเยอะๆ แต่ขนาดก็จะเพิ่มขึ้นตามตัวอักษร ดังนั้นการคอมเม้นโค้ดจึงไม่คิดพื้นที่มากนัก

**ตัวอย่างการ Comments**

แบบที่ 1

```
/*
This code by NECTEC
*/
```

แบบที่ 2

```
// Set  pin 13 to output
```

# ใบงานที่ 1.3 ทฤษฎีเบื้องต้น

## ชนิดของข้อมูล (Data Type)

ในการเขียนโปรแกรมหนึ่งๆ มีข้อมูลต่างๆ เข้ามาเกี่ยวข้องเช่น การนับจำนวนรอบ ของการทำงานโดยใช้ข้อมูลชนิดจำนวนเต็ม หรือการแสดงข้อความ โดยใช้ข้อมูลชนิดอักษร จะเห็นว่าข้อมูลต่างๆ ถูกแบ่งออกเป็นหลายชนิดตามจุดประสงค์ของการใช้งาน นอกจากนี้ข้อมูลแต่ละชนิด ยังใช้เนื้อที่หน่วยความจำ ไม่เท่ากันจึงมีการแบ่งชนิดข้อมูล

| Data type | Range | Meaning |
|---|---|---|
| boolean | 0 และ 1 | เป็น logic นั่นคือ 0 (False) และ 1(True) |
| char | -128 ถึง 127 | เก็บข้อมูลชนิดอักขระ |
| int | -32768 ถึง 32767 | เก็บข้อมูลชนิดจำนวนเต็ม |
| unsigned int | 0 ถึง 655535 | เก็บข้อมูลชนิดจำนวนเต็ม ไม่คิดเครื่องหมาย |
| long | -2147483648 ถึง 2147483647 | เก็บข้อมูลชนิดจำนวนเต็มแบบยาว |
| float | $3.4e^{-38}$ ถึง $3.4e^{38}$ | เก็บข้อมูลชนิดเลขทศนิยม |

# ใบงานที่ 1.3 ทฤษฎีเบื้องต้น

## ตัวอย่างการประกาศตัวแปรใน Arduino IDE แบบกำหนดค่า

#define variable_name value

```
#define LED2 19
#define LED3 18
```

int variable_name = value;

```
int LED2 = 19;
int LED3 = 18;
int LED4 = 5;
```

bool variable_name = true/false;

```
bool is = true;
bool am = false;
```

float variable_name = value;

```
float X = 10.5;
float Y = 1.23456;
float pi = 3.145926;
```

char variable_name[X] = “XXX”;

```
char ch1 = ‘A’;
char nickname[4] = “Ing”;
```


ข้อควรระวัง: การประกาศ #define ต้องกำหนดไว้ที่ส่วนหัวของโปรแกรมเท่านั้น

# ใบงานที่ 1.3 ทฤษฎีเบื้องต้น

## ตัวอย่างการประกาศตัวแปรใน Arduino IDE แบบไม่กำหนดค่า

### int variable_name;

```
Int age;
Int X;
Int i;
```

### float variable_name;

```
float X;
float Y;
float Z;
```

### bool variable_name;

```
bool X;
bool Y;
bool Z;
```

### char variable_name;

```
char X;
char Y;
char Z;
```

# ใบงานที่ 1.3 ทฤษฎีเบื้องต้น

## ตัวดำเนินการ (Operators)

ตัวดำเนินการ คือ เครื่องหมายที่ใช้แสดงถึงการกระทำต่างๆ ในโปรแกรมที่มีผลต่อข้อมูล หรือตัวแปร โดยสามารถใช้ตัวดำเนินการประเภทต่างๆร่วมกันได้ในภาษา C

| ตัวดำเนินการ | การกระทำ | ตัวอย่าง | ตัวดำเนินการ | การกระทำ | ตัวอย่าง |
|---|---|---|---|---|---|
| + | การบวก | a + b | == | เท่ากับ | a == b |
| - | การลบ | a – b | < | น้อยกว่า | a < b |
| * | การคูณ | a * b | <= | น้อยกว่าเท่ากับ | a <= b |
| / | การหาร | a / b | > | มากกว่า | a > b |
| % | การหารค่าเอาเฉพาะเศษ | a % b | >= | มากกว่าเท่ากับ | a >= b |
| ++ | การเพิ่มค่า | a++ | != | ไม่เท่ากับ | a != b |
| -- | การลดค่า | a-- | && | และ (AND) | a && b |
| += | การผสม | x += y | \|\| | หรือ (OR) | a \|\| b |
| ~ | การกลับบิทจาก 0 เป็น 1 | ~a | ! | ไม่ใช่ | !(a > b) |

# ใบงานที่ 1.3 ทฤษฎีเบื้องต้น

## ฟังก์ชันวนซ้ำ หรือ วนลูป (Loop)

ฟังก์ชันวนซ้ำหรือวนลูป (loop Statements) คือ การควบคุมโปรแกรมเพื่อให้โปรแกรมทำบางส่วนของโค้ดซ้ำๆ ตามเงื่อนไขต่างๆ โดยคำสั่งวนซ้ำมีหลายประเภทที่สามารถใช้ได้ในภาษา C ดังนี้

- **while loop**

  เป็นการทำงานวนทำซ้ำไปเรื่อยๆ จนกว่าเงื่อนไขจะไม่เป็นจริง

```
while (เงื่อนไข)
{
   คำสั่งภายใน loop
}
```

- **do-while loop**

  เป็นการตรวจสอบเงื่อนไขหากเป็นจริงจะวนมาทำคำสั่งภายใน loop แล้วตรวจสอบเงื่อนไขใหม่ ถ้าไม่เป็นจริงจะทำคำสั่งถัดไป

```
do
{
  คำสั่งภายใน loop
}
while (เงื่อนไข)
```

- **for loop**

  เป็นการทำงานวนซ้ำ โดยมีตัวแปรตรวจสอบเงื่อนไข และสามารถเปลี่ยนแปลงค่าของตัวแปร

```
for ( ตัวแปร;เงื่อนไข;ตัวแปร)
{
  คำสั่งภายใน loop
}
```

# ใบงานที่ 1.3 ขั้นตอนการทดลอง

## การทดลองที่ 1 การเขียนโปรแกรมวนลูปเพื่อควบคุมการทำงานของ LED

```
#define LED1 5
#define LED2 18
#define LED3 19

void setup()
{
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
}

void loop()
{
  for (int i = 0; i < 2; i++){
    digitalWrite(LED1, LOW);
    delay(1000);
    digitalWrite(LED1, HIGH);
    delay(1000);
  }
  for (int i = 0; i < 3; i++){
    digitalWrite(LED2, LOW);
    delay(1000);
    digitalWrite(LED2, HIGH);
    delay(1000);
  }
  for (int i = 0; i < 4; i++){
    digitalWrite(LED3, LOW);
    delay(1000);
    digitalWrite(LED3, HIGH);
    delay(1000);
  }
}
```

การใช้ Loop ในการควบคุม LED

# ใบงานที่ 1.3 คำถามท้ายหน่วยการเรียนรู้

**คำถามท้ายหน่วยการเรียนรู้ที่ 5**

จงเขียนโปรแกรมโดยใช้ Loop เพื่อควบคุม LED บน ESP32 DEVKIT โดยมีลักษณะการทำงานดังนี้

1. เริ่มต้นโปรแกรม LED1 จะต้องกระพริบจำนวน 3 ครั้ง
2. หลังจากนั้นให้ LED2 จะต้องกระพริบจำนวน 5 ครั้ง
3. หลังจากนั้นให้ LED3 จะต้องกระพริบจำนวน 2 ครั้ง
4. แล้วให้ LED1, LED2 และ LED3 จะต้องกระพริบพร้อมกันจำนวน 3 ครั้ง
5. และเมื่อสิ้นสุดข้อที่ 4 ให้โปรแกรมกลับไปทำงานข้อที่ 1

(โดยกำหนด delay ของโปรแกรมเป็นเวลา 0.5 วินาที)

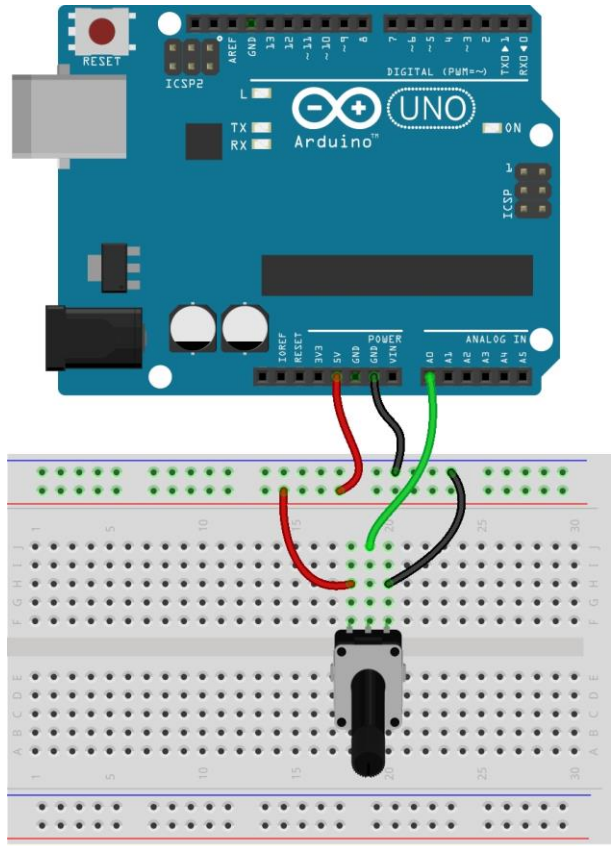
**คำถามท้ายหน่วยการเรียนรู้ที่ 6**

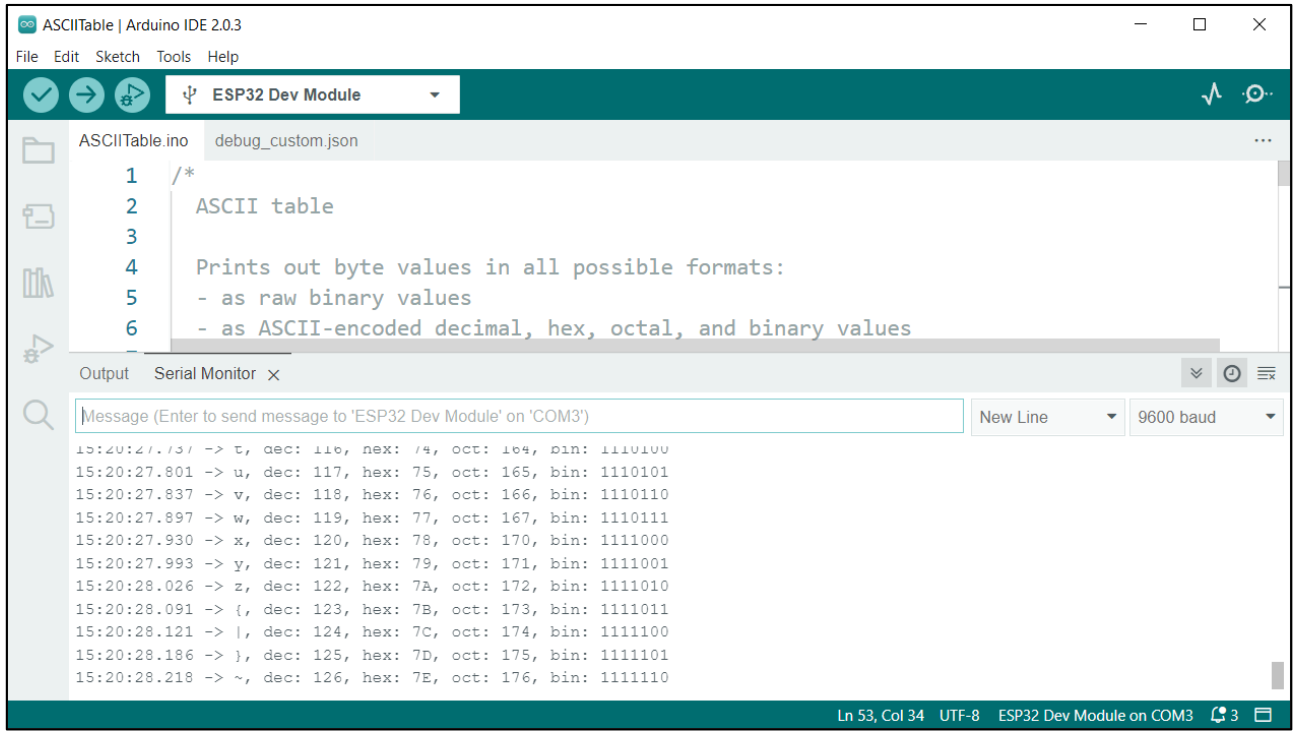
จงเขียนโปรแกรมโดยใช้ Loop เพื่อควบคุม LED บน ESP32 DEVKIT โดยมีลักษณะการทำงานดังนี้

1. เริ่มต้นให้ LED1, LED2 และ LED3 จะต้องกระพริบพร้อมกันจำนวน 5 ครั้ง
2. หลังจากนั้นให้ไฟติดเรียงจาก LED1 ไปยัง LED3 และดับเรียงจาก LED3 ไปยัง LED1 จำนวน 3 ครั้ง
3. หลังจากนั้นให้ LED1 และ LED2 กระพริบพร้อมกันจำนวน 3 ครั้ง
4. หลังจากนั้นให้ LED1 และ LED3 กระพริบพร้อมกันจำนวน 5 ครั้ง
5. แล้วให้ LED2 และ LED3 จะต้องกระพริบพร้อมกันจำนวน 4 ครั้ง
6. และเมื่อสิ้นสุดข้อที่ 4 ให้โปรแกรมกลับไปทำงานข้อที่ 1

(โดยกำหนด delay ของโปรแกรมเป็นเวลา 0.5 วินาที)

# ใบงานที่ 1.4 การรับค่าแอนะล็อก (Analog) และการสื่อสารในรูปแบบอนุกรม (Serial Monitor)



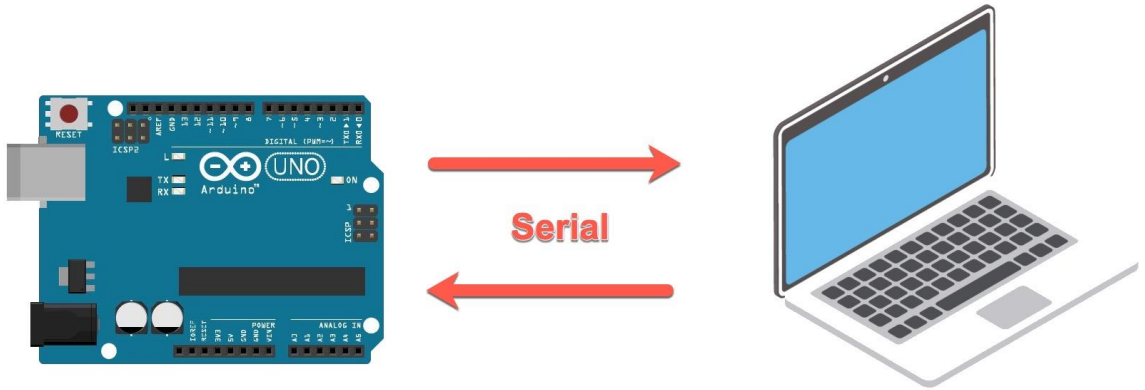



# ใบงานที่ 1.4 ทฤษฎีเบื้องต้น

## Serial Communication

Serial Communication คือ การสื่อสารแบบอนุกรม เป็นการรับส่งข้อมูลแบบต่อเนื่องในเส้นทางทางเดียวกัน โดยเราจะต้องตั้งค่าอัตราความเร็วในการส่ง (Baud Rate) ให้เท่ากัน เพื่อให้อุปกรณ์ทั้งตัวรับและตัวส่ง สามารถสื่อสารกันได้

โดยปกติค่ามาตรฐาน ได้แก่ 9600 และ 115200 ซึ่งในโปรแกรม Arduino IDE เราจะกำหนดค่า Baud Rate ไว้ในโค้ด Serial.begin();



# ใบงานที่ 1.4 ทฤษฎีเบื้องต้น

## การสื่อสารบน NodeMCU และ DEVKIT

### Serial.begin(speed)

Serial.begin เป็นฟังก์ชั่นที่ใช้ในการตั้งค่าความเร็วในการส่งข้อมูลของพอร์ต Serial ที่เชื่อมระหว่าง NodeMCU และ คอมพิวเตอร์ โดยที่ speed คือ ความเร็วในการรับส่งข้อมูล หน่วยเป็นบิตต่อวินาที (baud)

### Serial.print()

Serial.print เป็นฟังก์ชั่นเพื่อให้บอร์ดส่งค่าข้อมูลออกไปพิมพ์ทางพอร์ต Serial มีอยู่ 2 รูปแบบ

1. **Serial.print(val);** val เป็นค่าข้อมูลที่ต้องการพิมพ์ เป็นตัวแปรชนิดใดก็ได้
2. **Serial.print(val, format);** format ใช้ในตัวเลข และต้องการระบุฐานของตัวเลข หรือจำนวนทศนิยม

### Serial.println()

Serial.println เป็นฟังก์ชั่นที่ทำงานเหมือนกัน Serial.println ต่างกันตรงที่เมื่อใช้คำสั่ง Serial.println จะขึ้นบรรทัดใหม่เมื่อพิมพ์ค่าข้อมูลเสร็จ

# ใบงานที่ 1.4 ขั้นตอนการทดลอง

## การทดลองที่ 1 การแสดงผลข้อความผ่านการสื่อสารอนุกรม

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println("Training IoT Day1");
  delay(2000);
  Serial.println("Training by I-Kit");
  delay(2000);
}
```

การใช้คำสั่ง Print เพื่อแสดงผลใน Serial Monitor

```
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  for (int i = 0; i < 2; i++){
    Serial.print("Hello World 1");
    delay(1000);
  }
  for (int i = 0; i < 3; i++){
    Serial.println("Hello World 2");
    delay(1000);
  }
}
```

การใช้คำสั่ง Print ร่วมกับการวนลูป (เสริม)

# ใบงานที่ 1.4 ทฤษฎีเบื้องต้น

## การติดต่อ Analog Input

### อุปกรณ์ Input

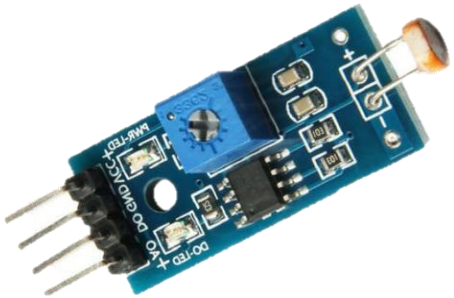
**LDR Photoresistor Sensor Module**

**สัญญาณแบบ Analog input**

LDR (Light Dependent Resistor) คือ ตัวต้านทานชนิดที่ไวต่อแสง หรือตัวต้านทานที่สามารถเปลี่ยนสภาพทางความนำไฟฟ้าเมื่อมีแสงมาตกกระทบ บางครั้งเรียกว่า Photo Conductor เป็นตัวต้านทานที่ทำมาจากสารกึ่งตัวนำ

โดยหลักการทำงานของ LDR Sensor จะทำการเปลี่ยนความต้านทานไปตามปริมาณแสงที่ตกกระทบ ทำให้เกิดความนำไฟฟ้าที่เปลี่ยนไป

LDR Photoresistor Sensor Module



## Function และคำสั่ง ในการติดต่อ Analog Input

### analogRead(pin)

analogRead เป็นฟังก์ชันที่ใช้ในการอ่านค่าสัญญาณแอนะล็อกของขาที่ต้องการ สำหรับ NodeMCU คือขา A0 ผลลัพธ์ที่ได้หลังจากสัญญาณแอนะล็อกถูกแปลงด้วย ADC จะเป็นตัวเลขจำนวนเต็ม 0-1023

### Serial.begin(speed)

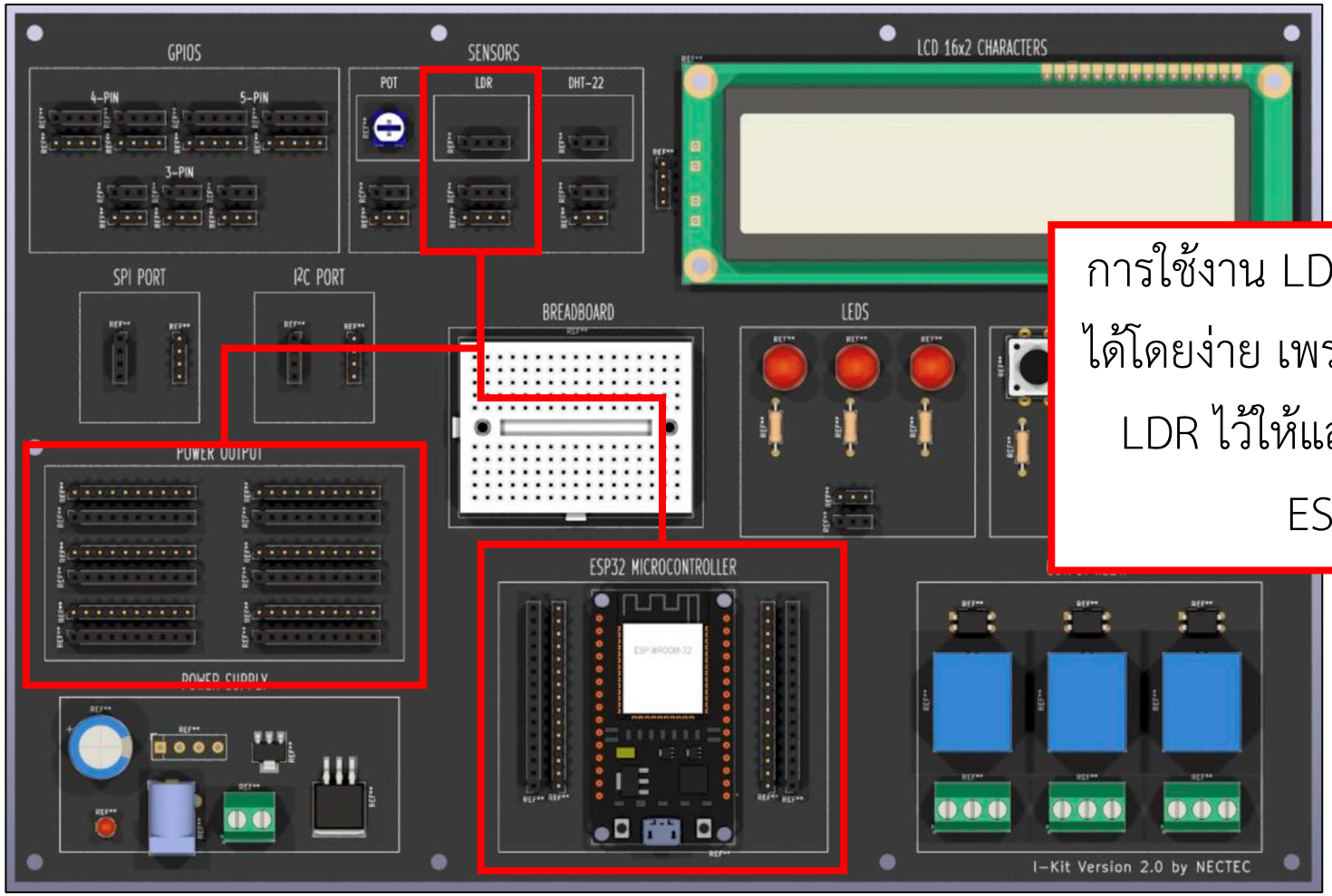
Serial.begin เป็นฟังก์ชั่นที่ใช้ในการตั้งค่าความเร็วในการส่งข้อมูลของพอร์ต Serial ที่เชื่อมระหว่าง NodeMCU และ คอมพิวเตอร์ โดยที่ speed คือ ความเร็วในการรับส่งข้อมูล หน่วยเป็นบิตต่อวินาที (baud)

### Serial.print() หรือ Serial.println()

Serial.print เป็นฟังก์ชั่นเพื่อให้บอร์ดส่งค่าข้อมูลออกไปพิมพ์ทางพอร์ต Serial

# ใบงานที่ 1.4 ทฤษฎีเบื้องต้น

## การใช้งาน LDR




การใช้งาน LDR บน I-KIT V.2 นั้น สามารถใช้งานได้โดยง่าย เพราะในช่อง Sensors มีพอร์ทสำหรับ LDR ไว้ให้แล้ว เพียงต่อสายเข้ากับ GPIO ของ ESP32 และต่อ Vcc, GND

# ใบงานที่ 1.4 ขั้นตอนการทดลอง

## การทดลองที่ 2 การทดสอบรับสัญญาณจาก LDR Photoelectric Sensor

```
#define ldr 39
void setup()
{
  Serial.begin(9600);
  pinMode(ldr, INPUT);
}
void loop()
{
  int ldr_data = analogRead(ldr);
  Serial.print("LDR Sensor Value = ");
  Serial.println(ldr_data);
  delay(1000);
}
```

ทดสอบใช้งาน LDR Sensor

# ใบงานที่ 1.4 ทฤษฎีเบื้องต้น

## การติดต่อ Analog Input

### อุปกรณ์ Input

**ตัวต้านทานปรับค่าได้ (Potentiometer 10 KΩ)**

**ส่งสัญญาณ Analog**

ในตัวอย่างที่ 13 เป็นการจำลองสัญญาณอนาลอกด้วยแรงดันไฟฟ้าจากตัวต้านทานปรับค่าได้ ซึ่งค่าแรงดันไฟฟ้ามีค่าท่ากับ 5V เป็นสัญญาณอนาลอก ดังนั้นค่าที่ ESP32 แสดงผลจะมีค่าเป็น 4095 ดังนั้นจึงได้สมการแปลงค่าแรงดันไฟฟ้าคือ

ได้สมการแปลงค่าแรงดันไฟฟ้าคือ

volts = 5*sensorValue/4095

# ใบงานที่ 1.4 ขั้นตอนการทดลอง

## การทดลองที่ 3 การทดสอบรับสัญญาณและแปลงข้อมูลจาก ตัวต้านทานปรับค่าได้

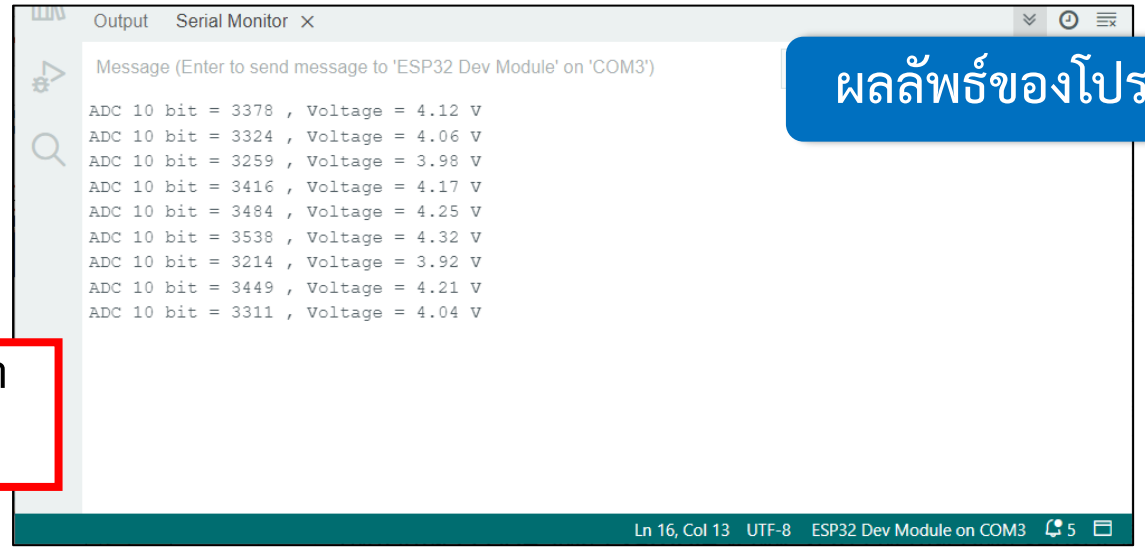
```
#define PTM 39

void setup()
{
  Serial.begin(9600);
  pinMode(PTM, INPUT);
}

void loop()
{
  int sensorValue = analogRead(PTM);
  Serial.print("ADC 12 bit = ");
  Serial.print(sensorValue);
  float volts = 3.3*sensorValue /4095.00;
  Serial.print(" , Voltage = ");
  Serial.print(volts,2);
  Serial.println(" V");
  delay(1000);
}
```

สมการแปลงค่าแรงดันไฟฟ้า

การจำลองสัญญาณ Analog ด้วยแรงดันไฟฟ้าจากตัวต้านทานปรับค่าได้จากทางขา GPIO39 ของ ESP32 ซึ่งค่าแรงดันไฟฟ้ามีค่าท่ากับ 5V เป็นสัญญาณ Analog ดังนั้นค่าที่ ESP32 แสดงผลจะมีค่าเป็น 4095 ดังนั้นจึงได้สมการแปลงค่าแรงดันไฟฟ้าคือ

volts = 5*sensor Value/4095

ผลลัพธ์ของโปรแกรม



# ใบงานที่ 1.4 ขั้นตอนการทดลอง

## การทดลองที่ 4 การทดสอบรับสัญญาณและแปลงข้อมูลจาก LDR Photoelectric Sensor

```
#define ldr 39
float ADC_value = 0.0048828125;
void setup()
{
  Serial.begin(9600);
  pinMode(ldr, INPUT);
}
void loop()
{
  float ldr_data = analogRead(ldr);
  int lux = int((250.000000/(ADC_value*ldr_data))-50.000000);
  Serial.print("Light Value = ");
  Serial.print(lux);
  Serial.println(" Lux unit");
  delay(1000);
}
```

ทดสอบแปลงค่าจากเซนเซอร์ LDR ให้เป็นค่าปริมาณแสงในหน่วย LUX

การแปลงค่าจากเซนเซอร์ LDR ที่ ESP32 ให้เป็นหน่วย Lux สามารถแปลงได้ตามสมการ

Lux = (250/(0.0048828125*LDR)) - 50

# ใบงานที่ 1.4 ทฤษฎีเบื้องต้น

## ฟังก์ชันเงื่อนไข (Condition Statements)

เป็นฟังก์ชันที่ใช้ในการควบคุมทิศทางการทำงานของโปรแกรม โดยคำสั่งเลือกทำตามเงื่อนไข จะทำให้โปรแกรมเดียวกันมีการทำงานที่แตกต่างกันไปตามค่าของตัวแปรต่างๆ ที่นำมาใช้กำหนดค่าในเงื่อนไข คำสั่งเลือกทำตามเงื่อนไข มีอยู่ 3 ประเภท คือ

- if

เป็นฟังก์ชันที่ใช้ในการตรวจสอบว่าเงื่อนไขเป็นจริงหรือไม่ โดยการใช้ตัวดำเนินการต่างๆ ในการสร้างเงื่อนไข ถ้าผลลัพธ์เป็นจริง โปรแกรมจะทำงานในบล็อคคำสั่ง if

```
if (เงื่อนไข) {
   คำสั่งการทำงาน
}
```

- if-else

เป็นฟังก์ชันมีการเพิ่มเงื่อนไข else if เข้ามาสำหรับการสร้างเงื่อนไขแบบหลายทางเลือก

```
if (เงื่อนไข) {
   คำสั่งการทำงาน
}
else if {
คำสั่งการทำงาน
}
```

- switch case

เป็นฟังก์ชันเงื่อนไขที่ทำงานคล้ายกับ if แต่ส่วนมากใช้สำหรับการเปรียบเทียบกับค่าคงที่

```
switch (เงื่อนไข)
{
   case 1: คำสั่งการทำงาน
   case 2: คำสั่งการทำงาน
}
```

# ใบงานที่ 1.4 ขั้นตอนการทดลอง

## การทดลองที่ 5 การทดสอบเขียนโปรแกรมเงื่อนไขร่วมกับเซนเซอร์ LDR Photoelectric Sensor

```
#define ldr 39
float ADC_value = 0.0048828125;

void setup()
{
  Serial.begin(9600);
  pinMode(ldr, INPUT);
}

void loop()
{
  float ldr_data = analogRead(ldr);
  int lux = int((250.000000/(ADC_value*ldr_data))-50.000000);
  Serial.print("Light Value = ");
  Serial.print(lux);
  Serial.println(" Lux unit");
  delay(1000);
  if (lux < 100){
    Serial.println("turn off");
  }
  else if (lux >= 100) {
    Serial.println("turn on");
  }
}
```

ทดสอบการใช้เงื่อนไขร่วมกับ LDR Sensor

# ใบงานที่ 1.4 คำถามท้ายหน่วยการเรียนรู้

**คำถามท้ายหน่วยการเรียนรู้ที่ 7**

จงเขียนโปรแกรมควบคุม LED1 ที่เชื่อมต่อกับ ESP32 โดยมีลักษณะการทำงานดังนี้

1. เริ่มต้นโปรแกรม LED1 จะต้องกระพริบจำนวน 3 ครั้ง (โดยทำแค่ครั้งเดียวเท่านั้น)
2. LED จะติดเมื่อมีค่าของแสงสว่างจากเซนเซอร์ LDR มากกว่าเท่ากับ 150 ขึ้นไป
3. LED จะดับเมื่อมีค่าของแสงสว่างจากเซนเซอร์ LDR น้อยกว่า 150 ลงไป

**คำถามท้ายหน่วยการเรียนรู้ที่ 8**

จงเขียนโปรแกรมควบคุม LED1 และ LED บน ESP32 โดยมีลักษณะการทำงานดังนี้

1. เริ่มต้นโปรแกรม LED1 และ LED บน ESP32 จะต้องกระพริบพร้อมกันจำนวน 3 ครั้ง (ครั้งเดียวเท่านั้น)
2. LED1 จะทำหน้าที่ไฟกระพริบเพื่อเป็นสัญญาณบอกว่าอุปกรณ์ยังทำงานปกติ โดยมีอัตราการกระพริบ 2 s
3. LED บน ESP32 จะติดเมื่อมีค่าของแสงสว่างจากเซนเซอร์ LDR มากกว่าเท่ากับ 150 ขึ้นไป
4. LED บน ESP32 จะดับเมื่อมีค่าของแสงสว่างจากเซนเซอร์ LDR น้อยกว่า 150 ลงไป

โดยโปรแกรมจะต้องทำงานกันอย่างต่อเนื่อง ไม่มีการหน่วงเวลาเกิดขึ้น

# ใบงานที่ 1.5 การประยุกต์ใช้งานฟังก์ชั่นเงื่อนไขกับสวิตช์

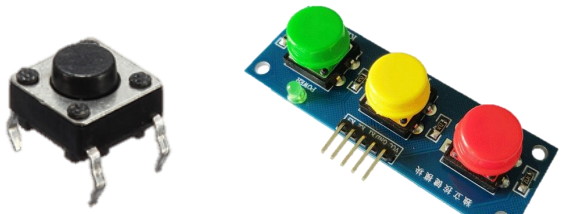
สวิตช์และ สวิทซ์โมดูล

# ใบงานที่ 1.5 ทฤษฎีเบื้องต้น

## การใช้งาน Switch

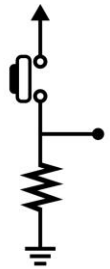
**Switch**

สวิตช์เป็นอุปกรณ์ประเภทอินพุตซึ่งสามารถหาซื้อได้ง่ายตามร้านค้าอุปกรณ์อิเล็กทรอนิกส์ทั่วไป โดยที่สวิตช์ทั่วไปนั้นจะเป็นประเภทสวิตช์กดติดปล่อยดับ (Tact Switch) ซึ่งในการเขียนโปรแกรมนั้นสามารถประยุกต์จากสวิตช์กดติดปล่อยดับเป็นสวิตช์กดติดกดดับ (Trigger Switch)



สวิตช์และ สวิทช์โมดูล

Pull Up resistor circuit



Pull Down resistor circuit

การต่อวงจรอินพุตแบบดิจิทัล

## การต่อวงจรอินพุตแบบดิจิทัล

การต่อวงจรอินพุตแบบดิจิทัล (Digital Input) มี 2 ลักษณะ คือ

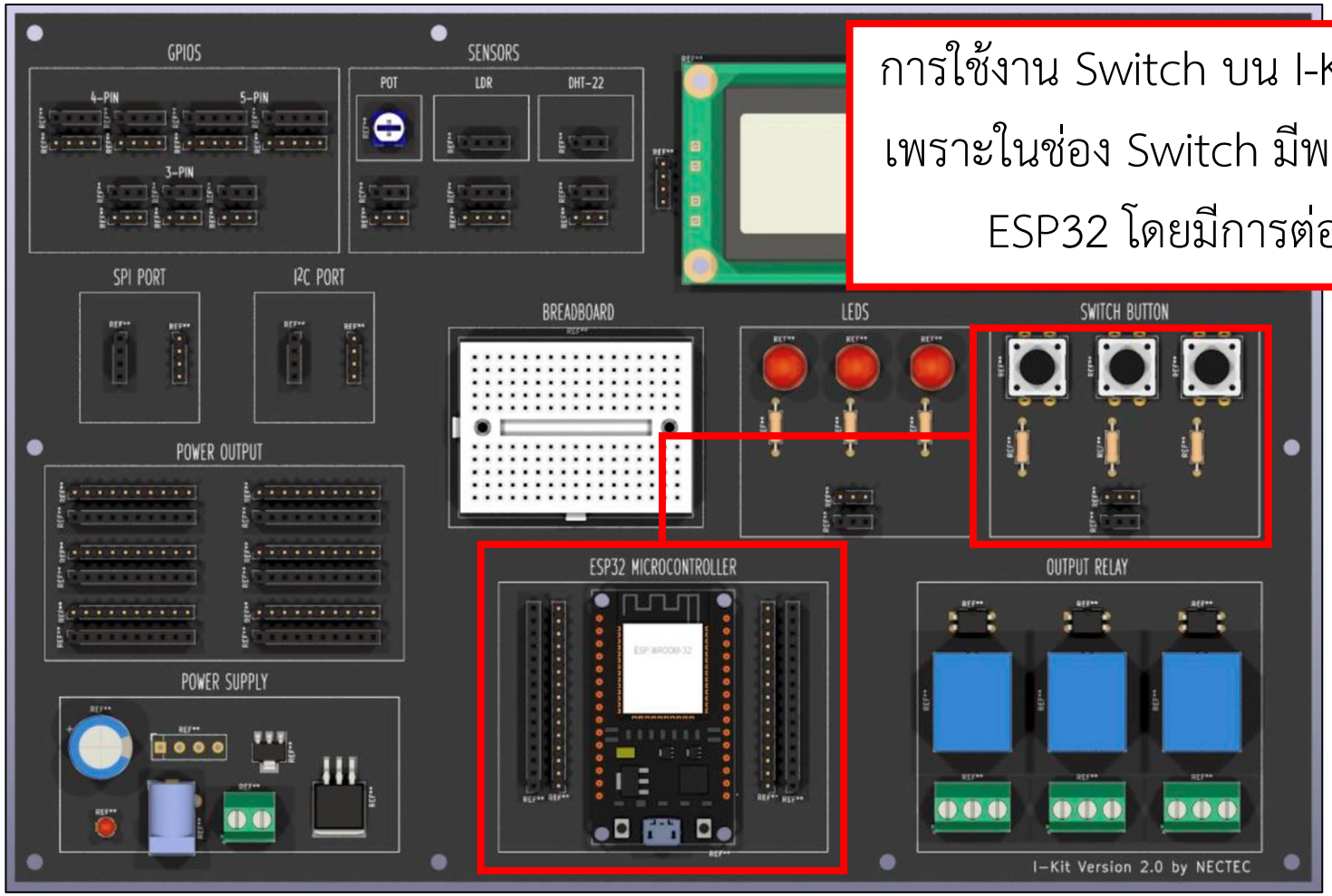
1. การต่อแบบ Pull Up ในการต่อแบบนี้นั้นจะทำให้ในเวลาปกตินั้นบอร์ดอ่านค่าแรงดันไฟฟ้า 3.3 V หรือลอจิก HIGH (1) แต่เมื่อมีสิ่งตอบสนองแรงดันไฟฟ้าจะลงมาที่ 0 V หรือลอจิก LOW (0) ซึ่งการทำงานภายใต้การต่อแบบ Pull Up จะถูกเรียกว่า Active Low
2. การต่อแบบ Pull Down มีผลลัพธ์สลับกันกับการต่อแบบ Pull Up กล่าวคือ ปกติบอร์ดอ่านค่าลอจิก LOW (0) และเมื่อมีสิ่งตอบสนองจะอ่านค่า HIGH (1) ซึ่งเรียกการทำงานแบบนี้ว่า Active High

## การต่อสวิทซ์เข้ากับ NodeMCU

เนื่องจาก GPIO ของ ESP8266 เกือบทั้งหมดนั้นมีตัวต้านทาน Pull up ภายในอยู่แล้ว ดังนั้นการต่อสวิตช์เข้ากับบอร์ดจึงไม่จำเป็นต้องต่อตัวต้านทานภายนอกเพิ่มเติมอีก ต่อเพียงสวิตช์อย่างเดียวเท่านั้นดังรูป **การต่อสวิตช์กับ** NodeMCU

# ใบงานที่ 1.5 ทฤษฎีเบื้องต้น

## การใช้งาน Switch



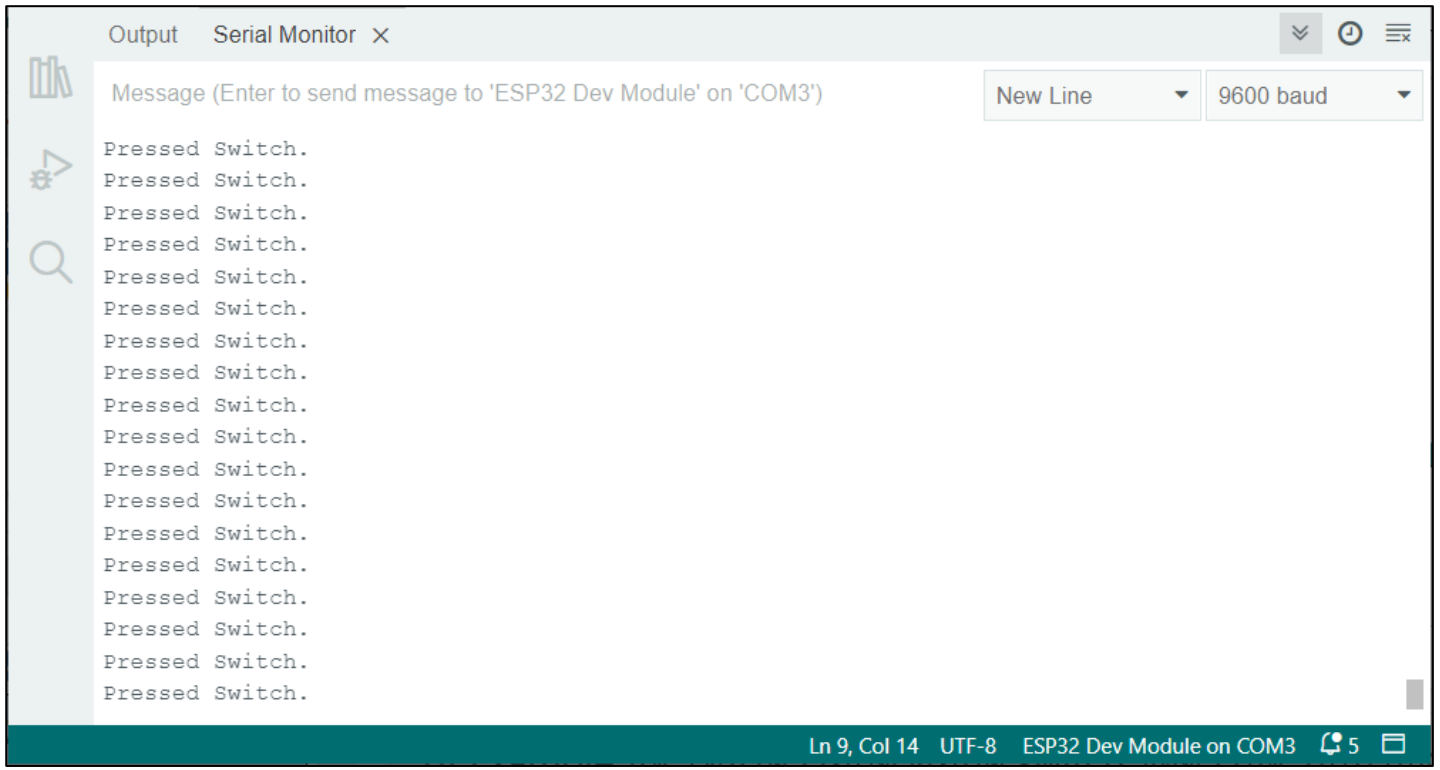

การใช้งาน Switch บน I-KIT V.2 นั้น สามารถใช้งานได้โดยง่าย เพราะในช่อง Switch มีพอร์ทสำหรับการต่อเข้ากับ GPIO ของ ESP32 โดยมีการต่อเข้ากับ Vcc และ GND ให้แล้ว

# ใบงานที่ 1.5 ขั้นตอนการทดลอง

## การทดลองที่ 1 การเขียนโปรแกรมสวิตช์กดติดปล่อยดับ (Tact Switch)

```
#define button 16
void setup()
{
  Serial.begin(9600);
  pinMode(button ,INPUT_PULLUP);
}

void loop()
{
  bool ReadSwitch = digitalRead(button);
  if(ReadSwitch == 0)
  {
    Serial.println("Pressed Switch.");
  }
}
```

**ผลลัพธ์ของโปรแกรม**



# ใบงานที่ 1.5 ขั้นตอนการทดลอง

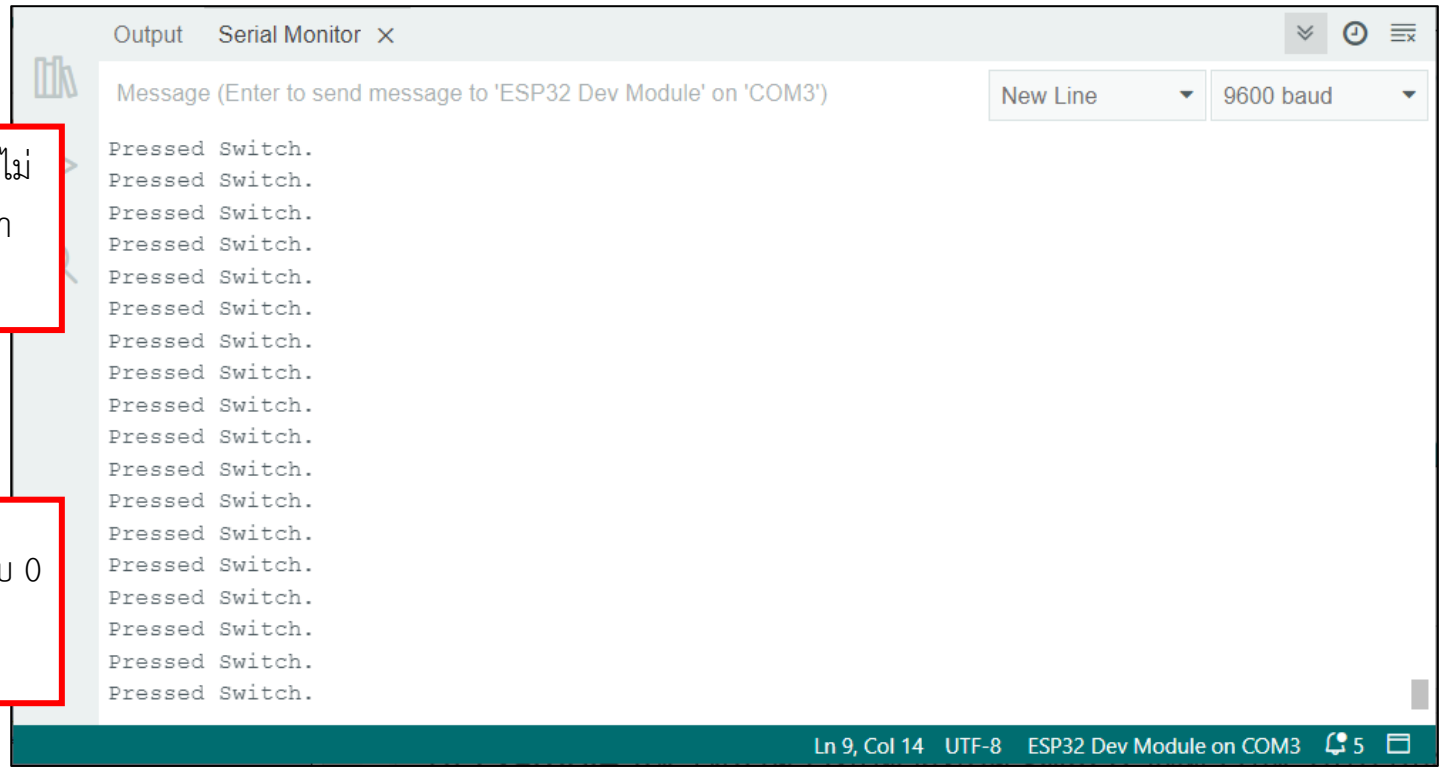
## การทดลองที่ 2 การเขียนโปรแกรมสวิตช์กดติดกดดับ (Trigger Switch)

```
#define button 16
int CheckButton;

void setup()
{
  Serial.begin(9600);
  pinMode(button ,INPUT_PULLUP);
}

void loop()
{
  bool ReadSwitch = digitalRead(button);
  if (CheckButton != ReadSwitch)
  {
    if(ReadSwitch == 0)
    {
      Serial.println("Pressed Switch.");
    }
    CheckButton = ReadSwitch;
    delay(100);
  }
}
```

เงื่อนไข คือ ถ้า checkBotton ไม่เท่ากับค่า Readswitch ให้เข้าเงื่อนไข

เงื่อนไข คือ Readswitch เท่ากับ 0 ให้แสดง Pressed Switch

**ผลลัพธ์ของโปรแกรม**



# ใบงานที่ 1.5 ขั้นตอนการทดลอง

## การทดลองที่ 3 การเขียนโปรแกรมควบคุม LED โดยใช้ Switch

```
#define led1 5
#define led 2
#define sw1 16
int checksw1;
int i=0;

void setup() {
  Serial.begin(9600);
  pinMode(led1, OUTPUT);
  pinMode(led, OUTPUT);
  pinMode(sw1, INPUT_PULLUP);
  digitalWrite(led1, 1);
  digitalWrite(led, 0);
}

void loop() {
  bool readsw1 = digitalRead(sw1);
  if ( checksw1 != readsw1 ){
    if (readsw1 == 0){
      i++;
    }
    checksw1 = readsw1;
    delay(1);
  }
  if (i==1){
    digitalWrite(led1 ,0);
  }
  else if (i == 2){
    digitalWrite(led1, 1);
  }
  else if (i == 3){
    digitalWrite(led, 1);
  }
  else if (i == 4){
    digitalWrite(led, 0);
  }
  else if (i == 5){
    digitalWrite(led1, 0);
    digitalWrite(led, 1);
  }
  else if (i == 6){
    digitalWrite(led1, 1);
    digitalWrite(led, 0);
  }
  else if (i == 7){
    i=1;
  }
}
```

# ใบงานที่ 1.5 ขั้นตอนการทดลอง

## การทดลองที่ 3 การเขียนโปรแกรมควบคุม LED โดยใช้ Switch

โปรแกรมควบคุม LED1 และ LED ที่อยู่บน ESP32 โดยมีลักษณะการทำงานคือ

เมื่อกด SW1 ครั้งที่ 1 ให้ LED1 ติดค้างไว้

เมื่อกด SW1 ครั้งที่ 2 ให้ LED1 ดับ

เมื่อกด SW1 ครั้งที่ 3 ให้ LED บน ESP32 ติดค้างไว้

เมื่อกด SW1 ครั้งที่ 4 ให้ LED บน ESP32 ดับค้างไว้

เมื่อกด SW1 ครั้งที่ 5 ให้ LED1 และ LED บน ESP32 ติดค้างไว้

เมื่อกด SW1 ครั้งที่ 6 ให้ LED1 และ LED บน ESP32 ดับ

เมื่อกด SW1 ครั้งที่ 7 ให้เริ่มกลับไปทำงานเหมือนครั้งที่ 1

**ผลการทดลองที่ 3**

# ใบงานที่ 1.5 ขั้นตอนการทดลอง

## การทดลองเสริม

```
#define ldr 39
#define led1 5
#define sw1 16
long lasttime = 0;
int checksw1;
int i;

float ADC_value = 0.0048828125;

void setup() {
  Serial.begin(9600);
  pinMode(led1, OUTPUT);
  pinMode(ldr, INPUT);
  pinMode(sw1, INPUT_PULLUP);
  digitalWrite(led1, 1);
}

void loop() {
  float ldr_data = analogRead(ldr);
  int lux =
int((250.000000/(ADC_value*ldr_d
ata))-50.000000);
  long now = millis();
  if (now - lasttime > 5000) {
    lasttime = now;
    Serial.print("Light Value = ");
    Serial.print(lux);
    Serial.println(" Lux unit");
  delay(500);
}
bool readsw1 = digitalRead(sw1);
if (checksw1 != readsw1){
  if (readsw1 == 0){
    i++;
  }
  checksw1 = readsw1;
  delay(1);
}
if (i == 1){
  digitalWrite(led1, 0);
}
if (i == 2){
  digitalWrite(led1, 1);
}
if (i == 3){
  i=1;
}
delay(1);
}
```

# ใบงานที่ 1.5 ขั้นตอนการทดลอง (เสริม)

## การทดลองเสริม

โปรแกรมควบคุม LDR Sensor โดยมีลักษณะการทำงานคือ แสดงค่าแสงทุกๆ 5 วินาที
เมื่อกด SW1 ครั้งที่ 1 ให้ LED1 ติดค้างไว้
เมื่อกด SW1 ครั้งที่ 2 ให้ LED1 ดับ
เมื่อกด SW1 ครั้งที่ 3 ให้ทำงานเหมือนครั้งที่ 1

**ผลการทดลองเสริม**

# ใบงานที่ 1.5 ทฤษฎีเบื้องต้น

## การใช้งาน delay() และ millis()

### การใช้งาน delay()

เป็นฟังก์ชั่นที่ใช้หยุดโปรแกรมชั่วคราวตามระยะเวลา (เป็นมิลลิวินาที) ที่ระบุเป็นพารามิเตอร์ไว้ (มี 1,000 มิลลิวินาทีในหนึ่งวินาที)

### Example Code

```
void setup() {
  Serial.begin(9600);
}
```

```
void loop() {
  Serial.println("Start timer");
  delay(2000);
  Serial.println("Time up");
}
```

### การใช้งาน millis()

เป็นฟังก์ชันสำหรับนับเวลาในหน่วยมิลลิวินาที โดยจะนับตั้งแต่บอร์ดเริ่มทำงานจนกว่าจำนวนนับจะเกิดการ overflow แล้วกลับไปเริ่มนับที่ 0 ใหม่อีกครั้ง

### Example Code

```
long lastMsg = 0;
void setup() {
  Serial.begin(9600);
}
voide loop(){
long now = millis();
```

```
if (now - lastMsg > 2000) {
    lastMsg = now;
    Serial.println(“Time up”);
}
delay(1);
}
```

# ใบงานที่ 1.5 คำถามท้ายหน่วยการเรียนรู้

## คำถามท้ายหน่วยการเรียนรู้ที่ 9

จงเขียนโปรแกรมควบคุม LED1, LED2 และ LED3 ที่อยู่บน ESP32 โดยมีลักษณะการทำงานคือ

เมื่อกด SW2 ครั้งที่ 1 ให้ LED1 ติดค้างไว้

เมื่อกด SW2 ครั้งที่ 2 ให้ LED1 ดับ

เมื่อกด SW2 ครั้งที่ 3 ให้เริ่มกลับไปทำงานเหมือนครั้งที่ 1

เมื่อกด SW3 ครั้งที่ 1 ให้ LED2 ติดค้างไว้

เมื่อกด SW3 ครั้งที่ 2 ให้ LED2 ดับ

เมื่อกด SW3 ครั้งที่ 3 ให้ LED3 ติดค้างไว้

เมื่อกด SW3 ครั้งที่ 4 ให้ LED3 ดับ

เมื่อกด SW3 ครั้งที่ 5 ให้ LED2 และ LED3 ติดค้างไว้

เมื่อกด SW3 ครั้งที่ 6 ให้ LED2 และ LED3 ดับ

เมื่อกด SW3 ครั้งที่ 7 ให้เริ่มกลับไปทำงานเหมือนครั้งที่ 1

และเขียนแสดงผลค่าของตัวต้านทานปรับค่าได้ทุกๆ 2 วินาที บน Serial Monitor (โดย millis)