


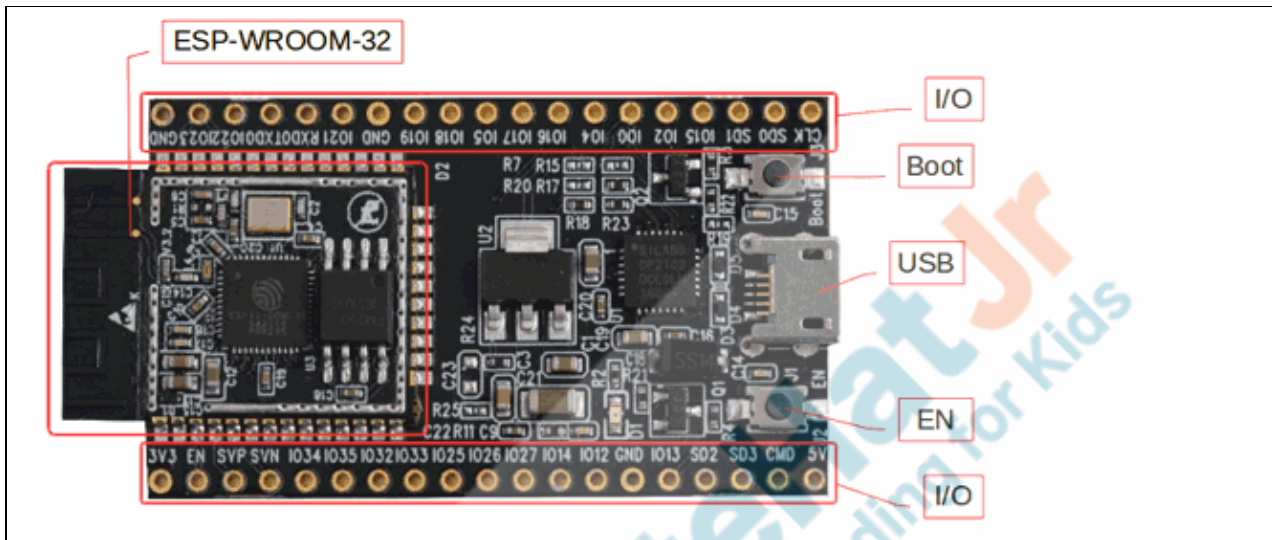
Topic	Blinking Banners	
Class Description	Students will be introduced to the ESP32 microcontroller and will learn how to make patterns of light using an embedded programming language.	
Class	PRO C243	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> <li>• Learn about electronics</li> <li>• Learn about breadboard</li> <li>• Learn about Tinkercad</li> <li>• Learn about Voltage and Current</li> </ul>	
Resources Required	<ul style="list-style-type: none"> <li>• Teacher Resources:               <ul style="list-style-type: none"> <li>○ Laptop with internet connectivity</li> <li>○ Earphones with mic</li> <li>○ Notebook and pen</li> <li>○ Smartphone</li> <li>○ WHJR IoT kit</li> <li>○ Components Requirement:</li> </ul> </li> <li>• Student Resources:               <ul style="list-style-type: none"> <li>○ Laptop with internet connectivity</li> <li>○ Earphones with mic</li> <li>○ Notebook and pen</li> <li>○ WHJR IoT Kit</li> <li>○ Components Requirement:</li> </ul> </li> </ul>	
Class structure	<b>Warm-Up</b> <b>Teacher-led Activity 1</b> <b>Student-led Activity 1</b> <b>Teacher-led Activity 2</b>	<b>10 mins</b> <b>15mins</b> <b>20 mins</b> <b>05 mins</b>

	<b>Student-led Activity 2 Wrap-Up</b>	
<b>WARM-UP SESSION - 10 mins</b>		
<b>Teacher Action</b>		<b>Student Action</b>
<p>Hey &lt;student's name&gt;. How are you? It's great to see you! Are you excited to learn something new today?</p> <p><b>Following are the WARM-UP session deliverables:</b></p> <ul style="list-style-type: none"> <li>Greet the student.</li> <li>Revision of previous class activities.</li> <li>Quizzes.</li> </ul>		<p><b>ESR:</b> Hi, thanks! Yes, I am excited about it!</p> <p>Click on the slide show tab and present the slides.</p>
<p><b>WARM-UP QUIZ</b> Click on In-Class Quiz</p>		
<p><b>Continue WARM-UP Session</b> Slide 6-8</p> 		
<p><b>Activity Details</b></p> <p><b>Following are the session deliverables:</b></p> <ul style="list-style-type: none"> <li>Appreciate the student.</li> <li>Narrate the story by using hand gestures and voice modulation methods to make the students feel engaged.</li> </ul>		
<b>TEACHER-LED ACTIVITY - 10 mins</b>		
<b>Teacher Initiates Screen Share</b>		
<p><b><u>ACTIVITY</u></b></p> <ul style="list-style-type: none"> <li><b>Teacher Activity description(in bullet points)</b></li> </ul>		

Teacher Action	Student Action
<p>In the last two sessions we were learning about IoT and then we discussed circuits used in electronics.</p> <p>Do you remember that we used the example of a bulb to understand circuits?</p> <p>Let's take the same example again. But this time we are talking about a smart bulb. It can switch on and off depending on the brightness and darkness.</p> <p>Now I have a question for you. How does this device act as a smart device?</p> <p>Okay, I have another question now. Which part of your body controls the entire body?</p> <p>Correct, it's the brain that regulates all body systems properly and that makes you smart, intelligent.</p> <p>Now, let's come back to smart devices. To make devices smart, there should be a brain right?</p> <p>Do you know what we call the brain of smart devices? It is called a <b>controller</b>.</p> <p>So today our focus is on the controllers. <b>What are microcontrollers?</b></p> <p><b><i>"A microcontroller is a compact integrated circuit (IC) that includes a processor, memory, and input/output (I/O) ports on a single chip"</i></b></p>	<p><b>ESR</b> Yes!</p> <p><b>ESR</b> Varied!</p> <p><b>ESR</b> Brain!</p> <p><b>ESR</b> Varied!</p>

<p>We have different types of controllers in the world but today we are focusing on the <b>ESP32</b> microcontroller.</p> <p><i>The teacher will guide the students to check the WHJR-IoT kit and observe the ESP32 controller.</i></p> <p>We need to explore ESP32. We can see a lot of pins and even something like IO wrote on it. Each pin has its own use.</p> <p>Let's focus on the main components of ESP32.</p>	
<p><b>Micro-USB Jack:</b> The micro USB jack is used to connect the ESP32 to the computer through a USB cable.</p> <p><b>EN Button:</b> The EN button is the reset button of the ESP module. Pressing this button will reset the code running on the ESP module.</p> <p><b>Boot Button:</b> This button is used to upload the Program from Arduino to the ESP module. It has to be pressed after clicking on the upload icon on the Arduino IDE. When the Boot button is pressed along with the EN button, ESP enters into firmware uploading mode. Do not play with this mode unless you know what you are doing.</p> <p><b>Red LED:</b> The red LED on the board is used to indicate the power supply. It glows red when the board is powered.</p> <p><b>Blue LED:</b> The blue LED on the board is connected to the GPIO pin. It can be turned on or off through programming.</p> <p><b>I/O Pins:</b> This is where the development takes place. These pins are used as inputs and outputs. We'll delve into the details regarding these pins, a little later on.</p>	

**ESP-WROOM-32:** This is the heart of the ESP32 module. It is a 32-bit microprocessor.



Now the next thing is how this controller understands what to do and how to do it. And how does only one type of controller do a lot of things to make different smart devices.

Let's understand this. What's the use of programming languages?

Programming languages help to develop different applications.

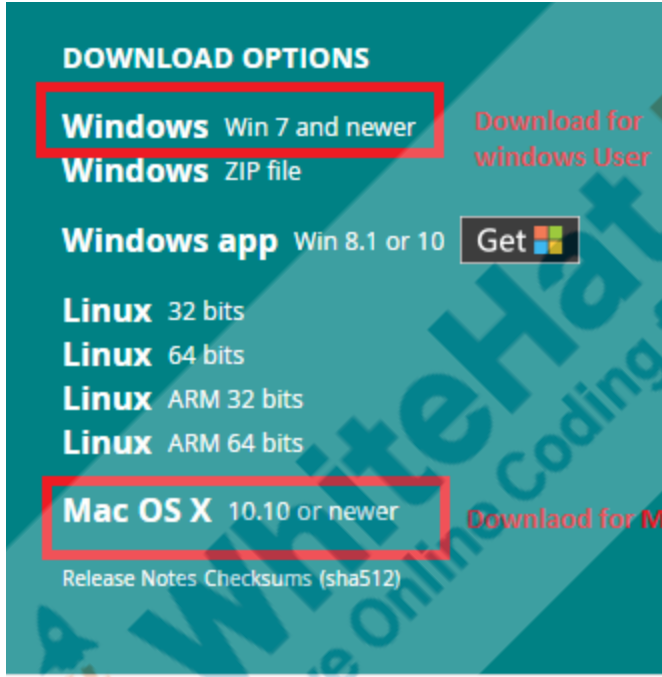
Similarly, if you want to make different applications on the microcontroller you must know the microcontroller language.

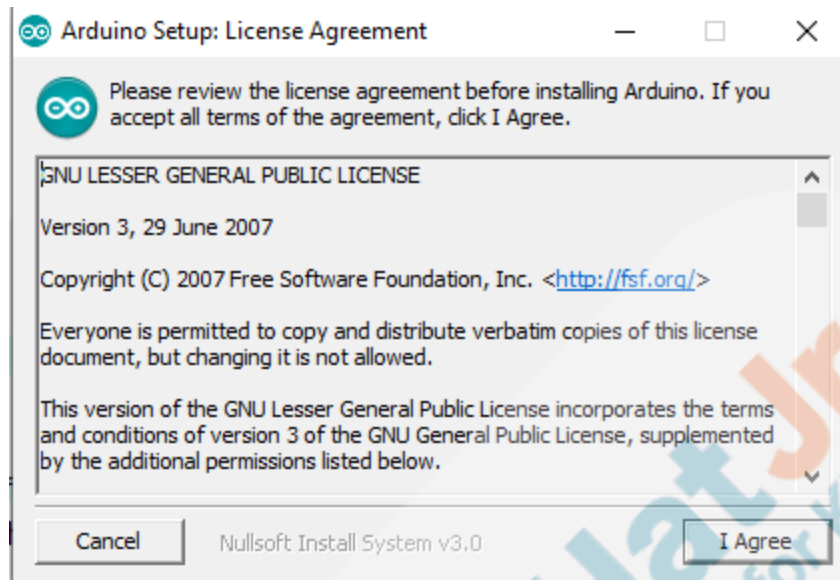
For microcontrollers, we use an embedded language that can be written in the C language, and to run that C program we need software.

So let's install the software and learn a new language

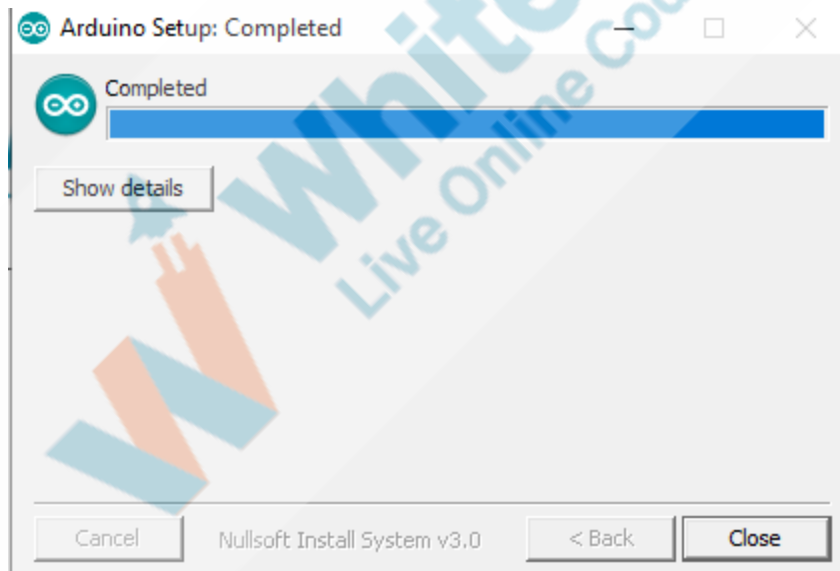
## ESR

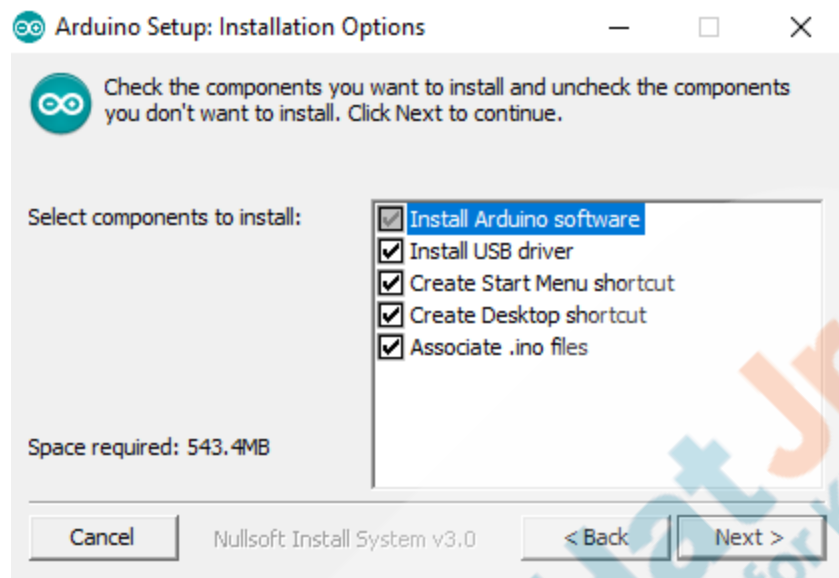
To make different applications

called embedded language.	
Click on <a href="#">Teacher Activity 1</a>	Student clicks on <a href="#">Student Activity 1</a>
Download Arduino Software	
 <p>The screenshot shows the 'DOWNLOAD OPTIONS' for Arduino. It lists several options: 'Windows Win 7 and newer' (highlighted with a red box), 'Windows ZIP file', 'Windows app Win 8.1 or 10' (with a 'Get' button), 'Linux 32 bits', 'Linux 64 bits', 'Linux ARM 32 bits', 'Linux ARM 64 bits', and 'Mac OS X 10.10 or newer' (highlighted with a red box). There are also links for 'Release Notes' and 'Checksums (sha512)'.</p>	
Go to <b>Downloads</b> <ul style="list-style-type: none"> <li>• Open <b>Software</b></li> <li>• Click on <b>I Agree</b></li> </ul>	

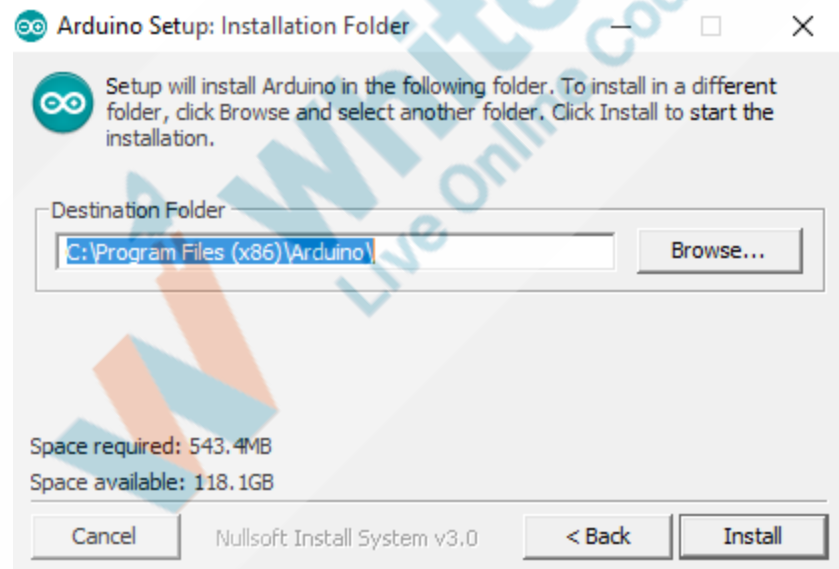


Click on **Next**



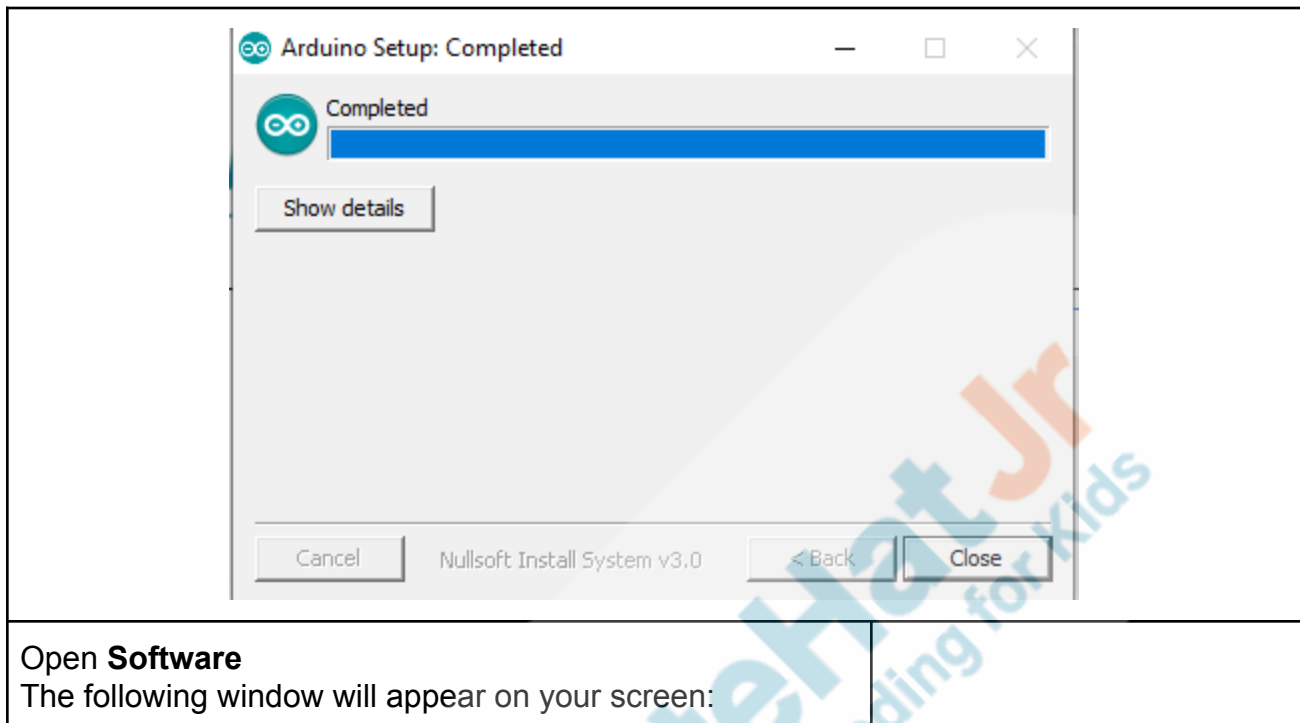


Click on **Install**



Click on **Close**

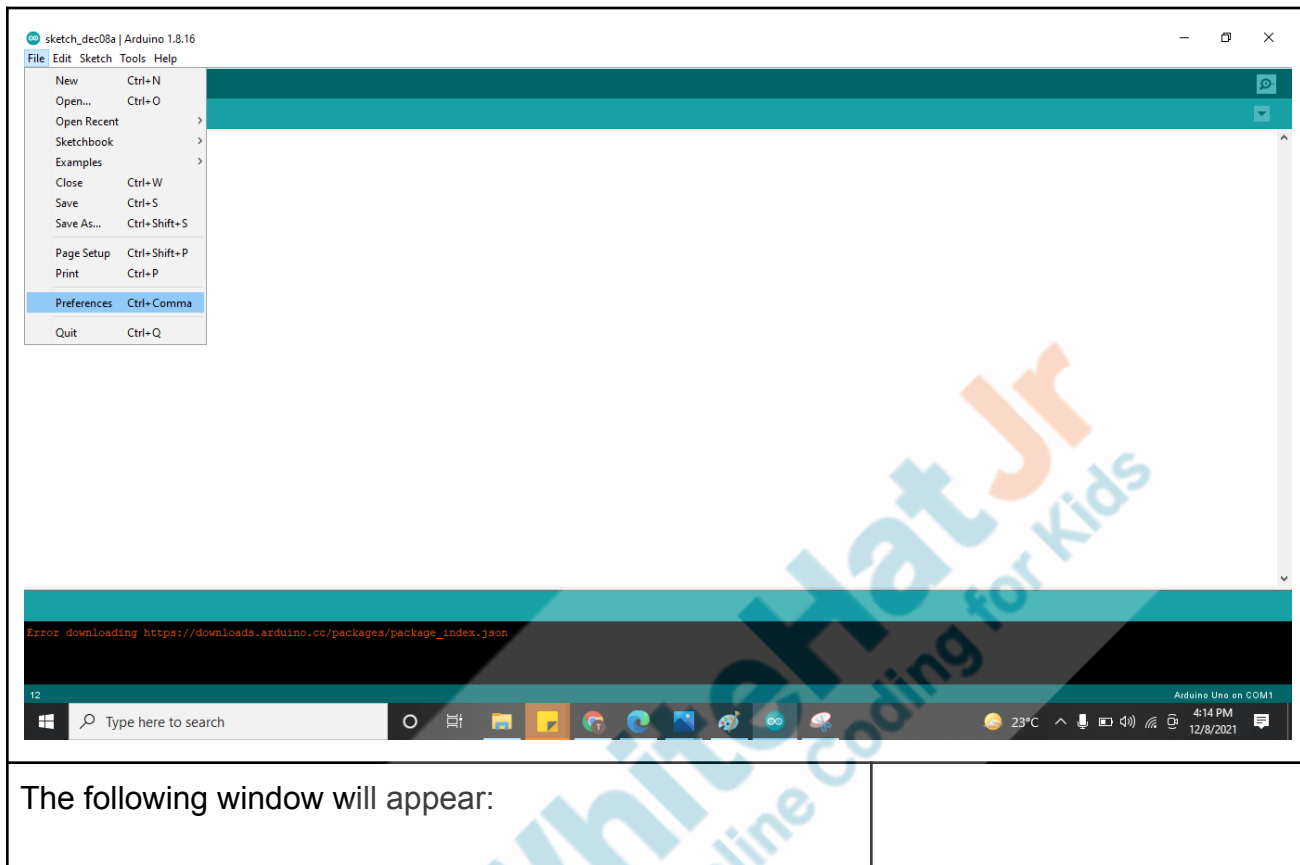


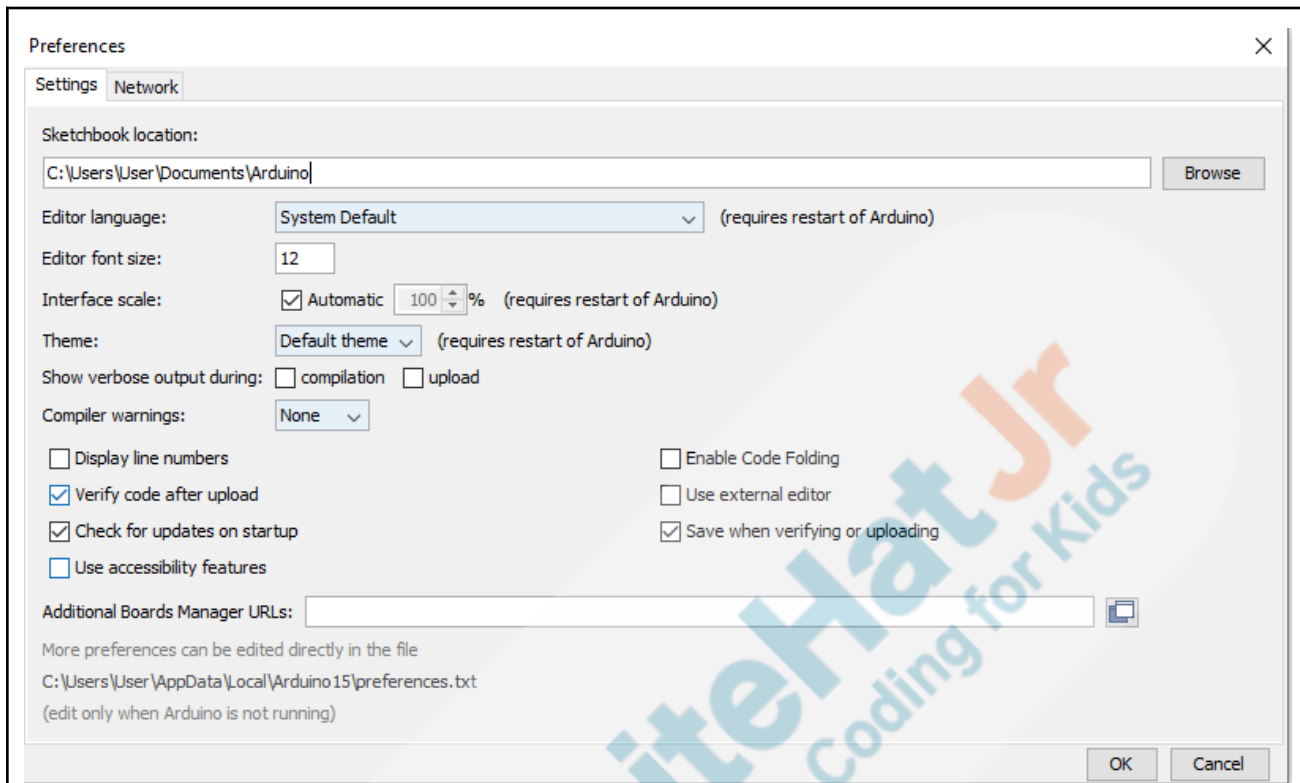




To make it compatible with ESP 32 we need to adjust the settings as follows:

Click on **File> Preferences**





Now copy the following link and paste the copied text into **Additional Boards Manager URLs**:

[https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json)

Click on **OK**.

Preferences

Settings Network

Sketchbook location:  
C:\Users\User\Documents\Arduino Browse

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

Interface scale: ☒ Automatic 100% (requires restart of Arduino)

Theme: Default theme (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload


Compiler warnings: None

☐ Display line numbers ☐ Enable Code Folding

☒ Verify code after upload ☐ Use external editor

☒ Check for updates on startup ☒ Save when verifying or uploading

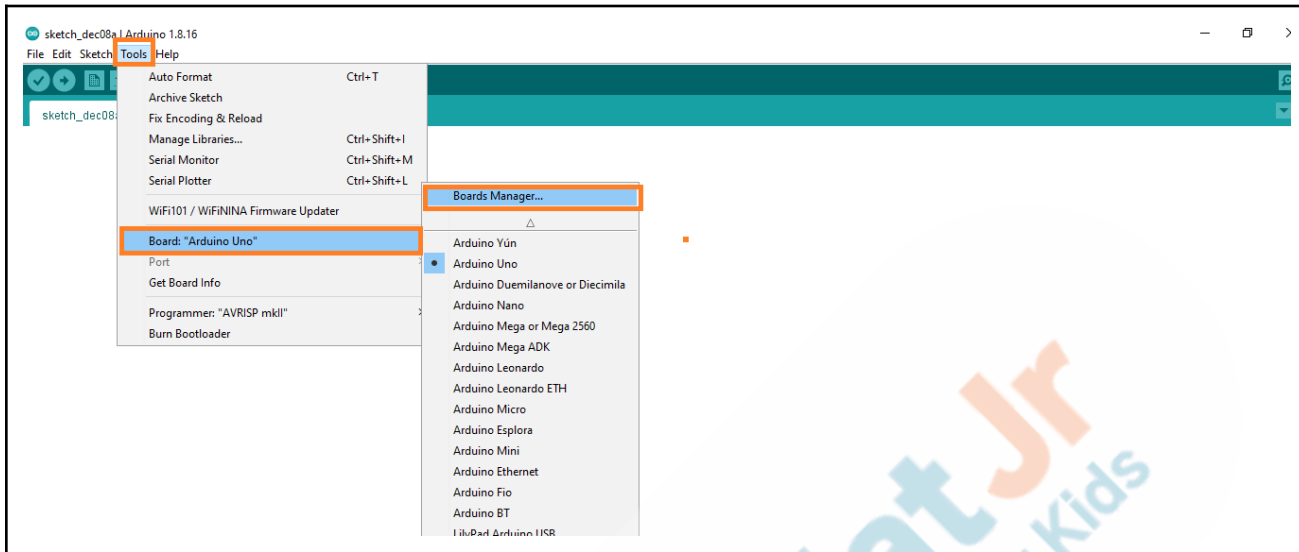
☐ Use accessibility features

Additional Boards Manager URLs: [https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json) 

More preferences can be edited directly in the file  
C:\Users\User\AppData\Local\Arduino15\preferences.txt  
(edit only when Arduino is not running)

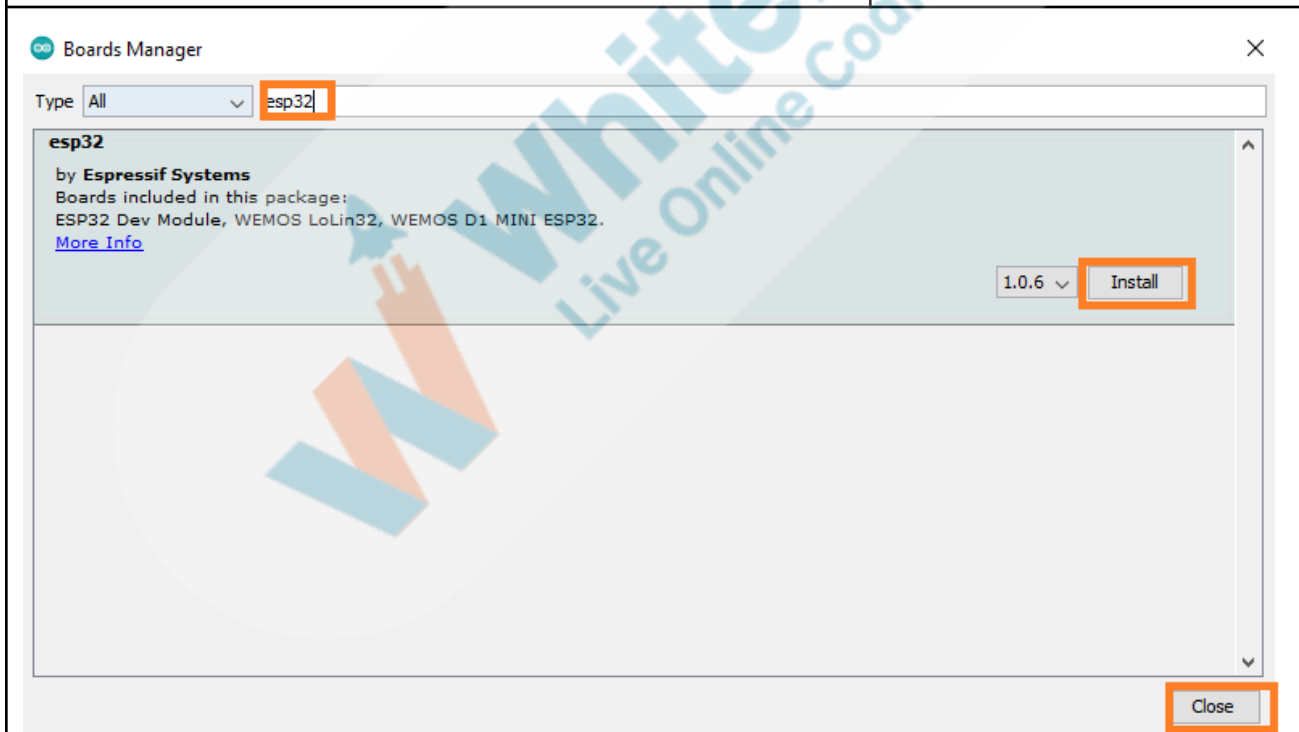
OK Cancel

Now go to the **Tools >Board: “Arduino Uno”> Boards Manager**.



Type **esp32** and then click on **Install**.

Click on **Close**.



The software installation is done, but now we need to understand how to write a program.

The concepts of programming will remain the same for all languages, whether it is C, Java, Python, or Block language. We simply need to learn about the syntax.

By default, the Arduino IDE generates empty setup and loop functions for you when you open a new program.

Let's understand what this means.

**void setup()** and **void loop()** are two mandatory functions in Arduino.

In a C/C++ program, we have to write a **main** function. This **main** function will be called first and from there we will call other functions and execute the functionalities of the program. But in Arduino IDE instead of one, we have two functions: **setup()** and **loop()**.

### **void setup()**

The **void setup()** function is called only once at the very beginning of the program.

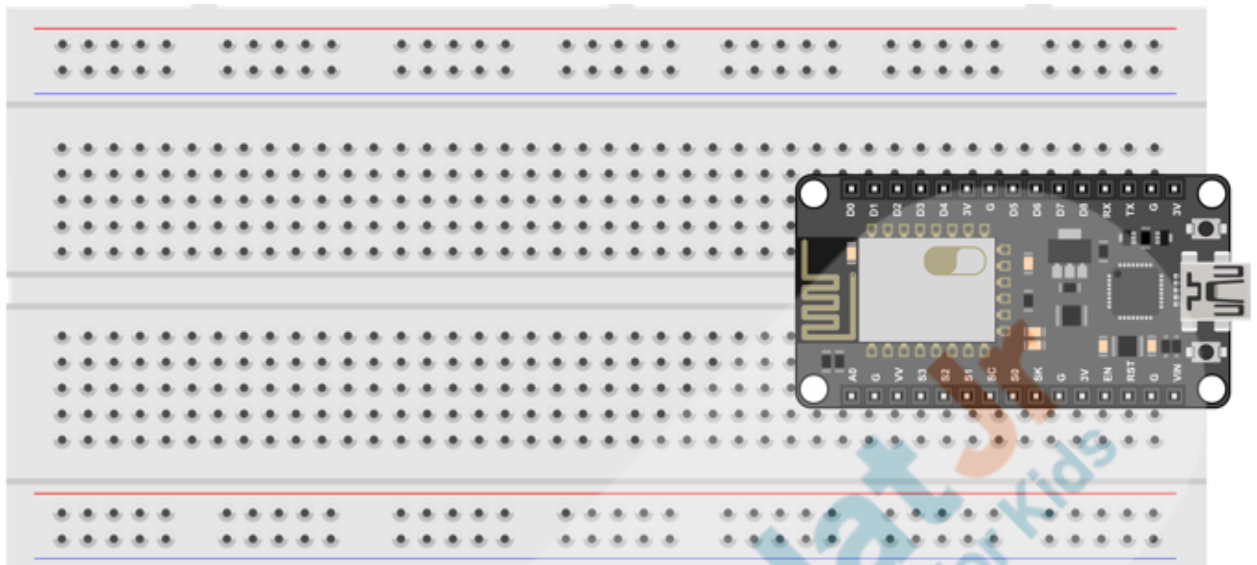
Uses of **setup()** function:

- To initialize variables and their values
- To set up communications (eg: serial)
- To set up modes for digital pins (input/output)
- To initialize any hardware component that is plugged into the Arduino

The **setup()** function, as its name suggests, is made for you to do any setup required at the beginning of the program. Don't write the core functionalities here, just the

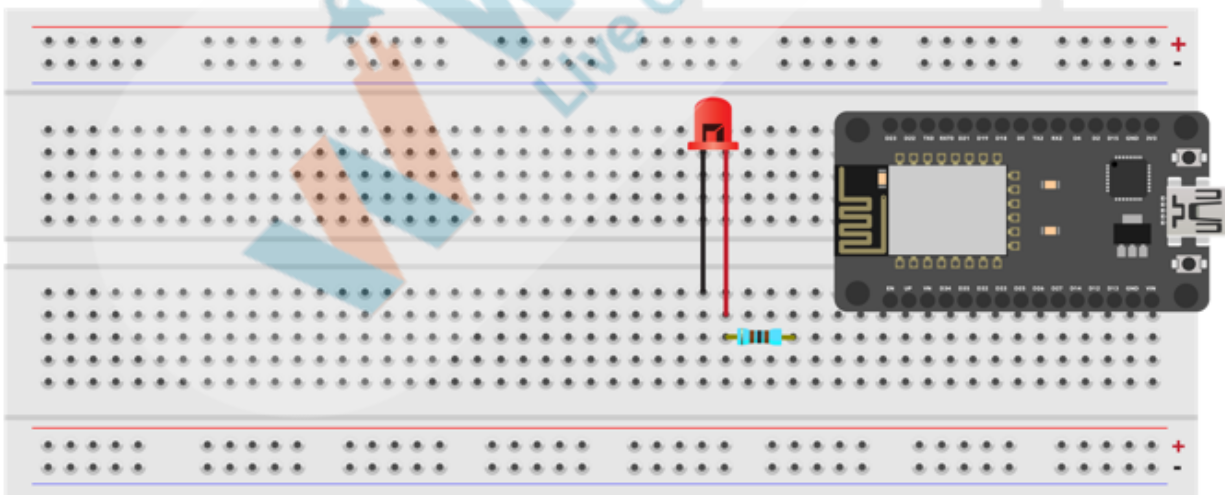
<p>initialization code.</p> <p><b>void loop()</b></p> <p>In the <b>loop()</b> function, we will write our main program, knowing that the initialization is already done. In this function, always keep in mind that the last line is followed by the first line.</p> <ul style="list-style-type: none"> <li>• There's no need to write all the code directly in the function.</li> <li>• We can create as many other functions as we want and call those functions in the <b>loop()</b> function.</li> </ul>	
<p>Now, you must be eager to run something on an Arduino IDE using ESP32.</p> <p>But what will be the application?</p> <p>Let's try to make an LED blink using the ESP32 controller and Arduino IDE.</p> <p>Let's gather the following material from the WHJR-IoT kit:</p> <ol style="list-style-type: none"> <li>1. 1 x ESP32</li> <li>2. 1 x USB Cable</li> <li>3. 1 x LED</li> <li>4. 1 x 330-ohm Resistors</li> <li>5. 1 x Jumper Wires</li> <li>6. 1 x Battery</li> <li>7. 1 x Battery Socket</li> </ol>	<p><b>ESR</b> <b>Varied!</b></p>
<p>The first and most important thing to do is to mount the ESP32 on the breadboard across both sides of the middle breakers.</p>	





## Mount Components

- **Mount LED & Resistor**
  - Insert the LED into the breadboard.
  - Connect the longer leg of the LED to one end of the resistor as shown below:

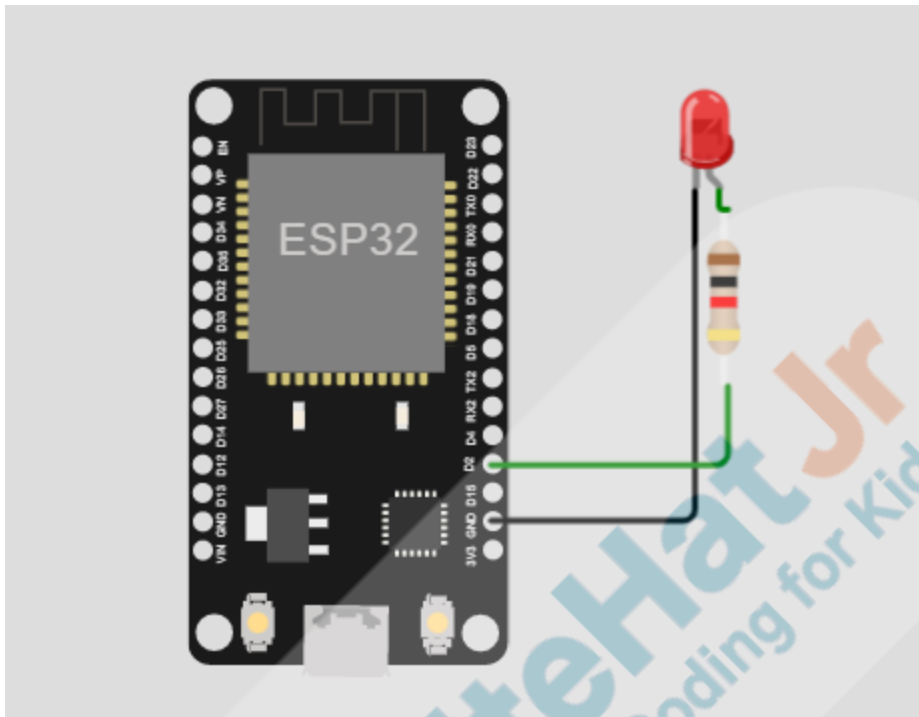


The actual circuit diagram looks like this:

© 2021 - WhiteHat Education Technology Private Limited.

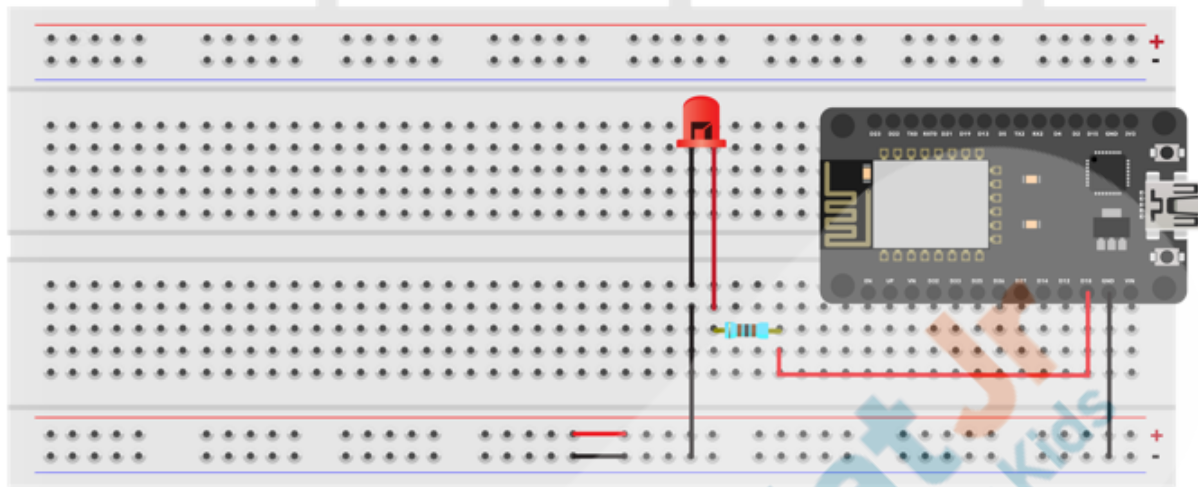
Note: This document is the original copyright of WhiteHat Education Technology Private Limited.

Please don't share, download or copy this file without permission.



**Wiring Connections:**

- Connect the other end of the resistor with **ESP 32** GPIO **pin no 18**.
- Connect the shorter leg of the LED with the **GND** of the ESP32.



Open the Arduino IDE software on your computer. Code in the Arduino language will control your circuit

The first and most important thing is to initialize the variables:

- Let's initialize the **LED**.
- Declare the **LED** before **void setup()**

Can you tell me which LED terminal is used to control the LED on or off?

**ESR**  
**Anode(+)**

Right. This means we need to configure this LED pin with ESP 18.

We must initialize the PIN number. In ESP32, the LED on the board is connected to pin number 18. However, we can choose any number on the breadboard.

The next task is to configure the pin. To configure the pin we use the **pinMode()** function.

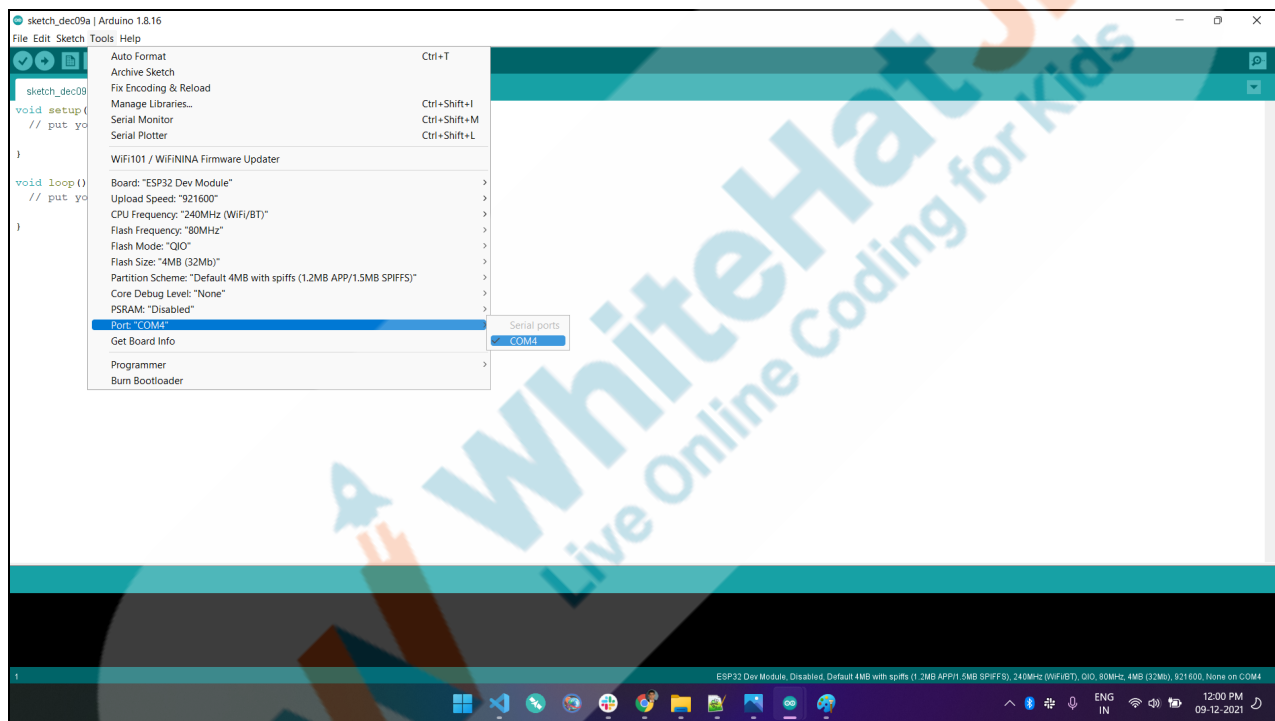
<p><b>pinMode()</b> configures the specified pin to behave either as input or output. Since we want this pin for output, we are writing <b>OUTPUT</b> here.</p> <p><b>Syntax:</b> <i>pinMode(pin, mode)</i></p> <p><b>pin:</b> The pin do we need to set  <b>mode:</b> Set the mode INPUT, OUTPUT</p>	
<pre>#define LED 18  void setup() {   pinMode(LED, OUTPUT); }</pre>	
<p>The pin needs to be programmed to be either ON or OFF, that is, we can command it to be ON (output 5 volts), or OFF (output 0 volts).</p> <p>To switch it on and off, we need to use a function called <b>digitalWrite()</b>.</p>	
<pre>void loop() {   digitalWrite(LED, HIGH);   delay(500);   digitalWrite(LED, LOW);   delay(500); }</pre>	
<p>Let's upload the program to check if we are able to program our ESP32 module.</p> <p>For uploading, we need a USB cable.</p> <p>Take the USB cable, from WHJR-IoT kit.</p> <p>Insert one end of the USB cable into the ESP32 USB port</p>	

and another end in the computer's USB port.

Now select the port for communication:

1. Go to **Tools**.
2. Select the Serial Port.
3. Select the Port Number.

**Note:** We can find the port number by accessing the **device manager** on Windows.



Everything is ready now. The Arduino board is now ready to communicate with your PC. Instructions will be sent to the ESP32 board from your PC.

Click on the **tick** to verify the **syntax**.

OR

Click on **Sketch >>Verify/Compile**.

When you click on the tick button, the program you have written in Arduino IDE will be compiled for any errors.



```

Blink_Led | Arduino 1.8.16
File Edit Sketch Tools Help

Blink_Led $
int LED = 10;

void setup() {
  pinMode(LED, OUTPUT);
}

void loop() {
  digitalWrite(LED, HIGH);
  delay(500);
  digitalWrite(LED, LOW);
  delay(500);
}

```

Now, the next thing is to upload the program,

Click on the **arrow** to upload the program

OR


Click on **Sketch >>Upload**

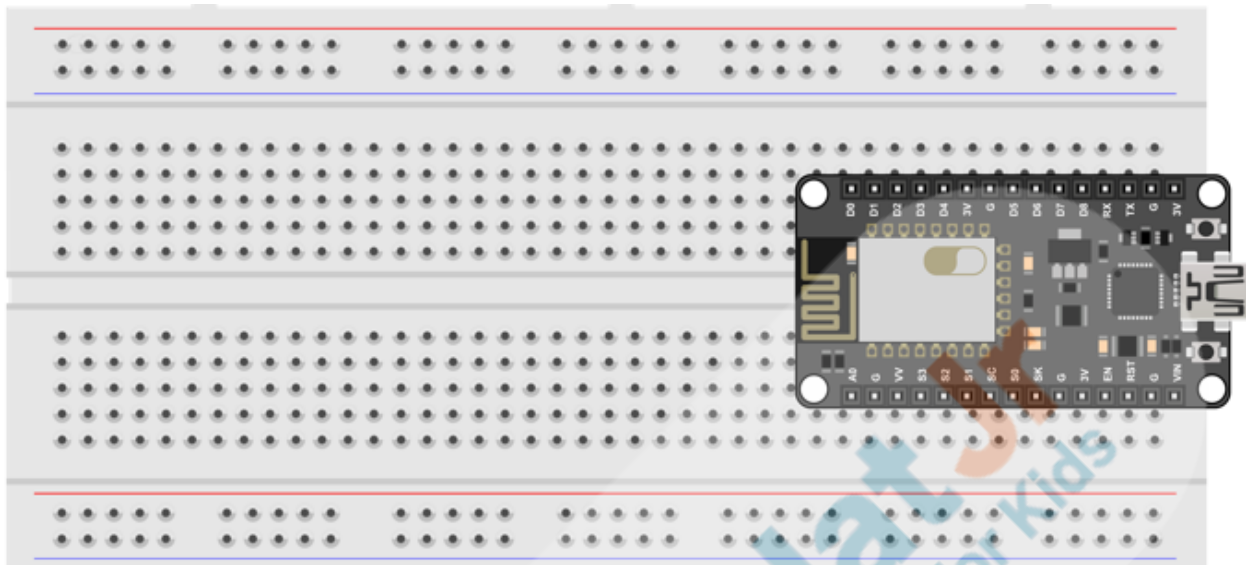


If you look at your ESP32, you can see the 2 LEDs near Tx and Rx blinking. This is an indication of successful communication between your PC and the ESP32 board. If the program has been uploaded successfully, you will see a message like **Done Uploading**. If the uploading process was not successful, you will see an error message accordingly.

Now, you will see the LED will blink on and off after 10 seconds. This is how LEDs can be made to blink using a microcontroller.

Now it's your turn to do that. But this time we need to take two or three LEDs.

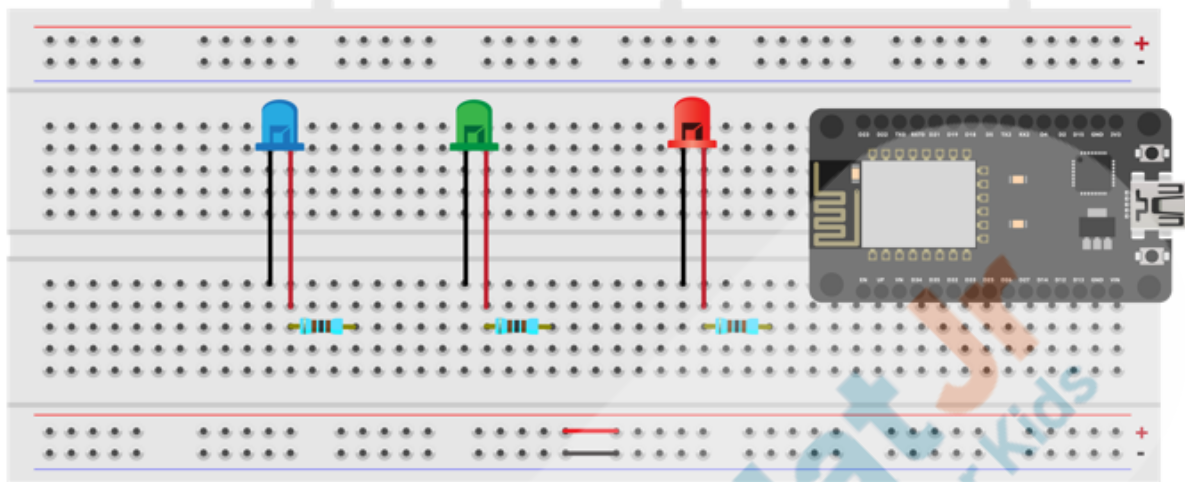
Teacher Stops Screen Share	
So now it's your turn. Please share your screen with me.	
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.	
<div>  </div> <b>Teacher Ends Slideshow</b>	
STUDENT-LED ACTIVITY - 20 mins	
<ul style="list-style-type: none"> <li>• Ask the student to press the ESC key to come back to the panel.</li> <li>• Guide the student to start Screen Share.</li> <li>• The teacher gets into Full screen.</li> </ul>	
ACTIVITY	
<ul style="list-style-type: none"> <li>• Student Activity description (in bullet points).</li> </ul>	
Teacher Action	Student Action
Gather the following materials from the WHJR-IoT kit: <ul style="list-style-type: none"> <li>1 x ESP32</li> <li>1 x USB Cable</li> <li>3 x LED</li> <li>1 x 330-ohm Resistor</li> <li>1 x Jumper Wires</li> <li>1 x battery</li> <li>1 x Battery Socket</li> </ul>	
The first and foremost thing to do is to mount the ESP32 on the breadboard across both the sides of the middle breakers.	



### Mounting the Components

- **Mount LED's & Resistor:**
  - Insert three LEDs into the breadboard & three resistors.
  - Connect the longer leg of the LED to one end of the resistor as shown below.
  - Do the same for other three LEDs.





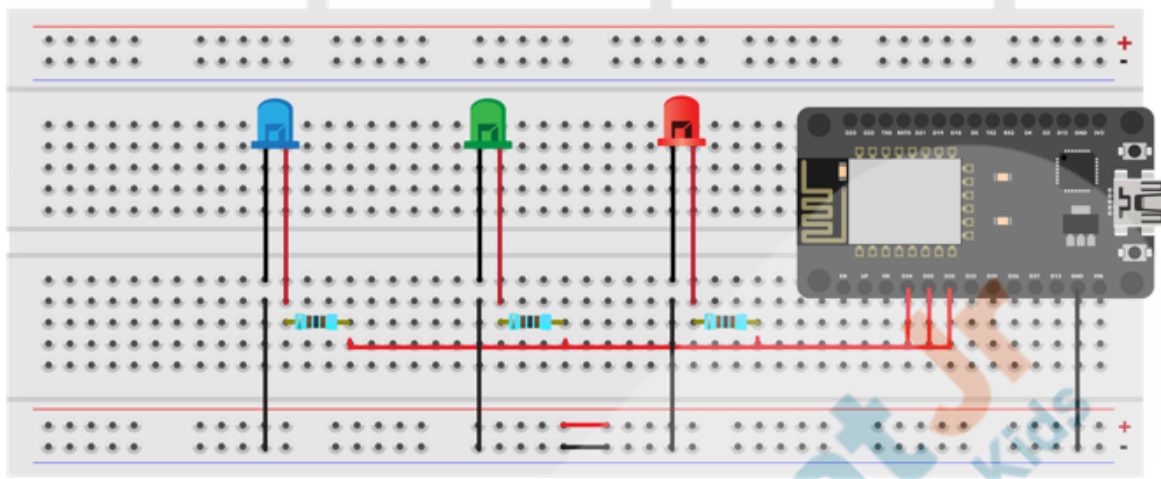
**Step-3:** The third step is to wire everything up:

Provide +ve (5V) and GND (-ve) to the LED and the resistor respectively.

- Connect the other end of the resistor with ESP32 pins **D25, D26, D27, D31, D32** respectively. Just below the resistor pin, insert the male jumper wire.

*Note: All the pins in the breadboard component rails are internally connected to each other*

- Connect the negative part (shorter leg) of the LED to the GND terminal of the ESP32. Take a male jumper wire and connect it just below the shorter leg of the LED and drag it to the GND (-ve) terminal of the ESP32. Do the same for the other four LEDs too.



*Note: The above diagram representation is for three LEDs; the student can try this for up to five LEDs.*

Now, it's time for programming.

Open the Arduino IDE software on your computer. Code in the Arduino language will control your circuit.

**Step-1** The first and most important thing is to declare the global variables

- Define variables along with their data types.
- **int, const int** are called datatypes, data type **int** is used to store integer value.
- As we have a total of five LED's, define an array of LEDs named **arr\_pin**.
- We must initialize the PIN numbers. In ESP32, the LED on the board is connected to pin numbers **25, 26, 27, 32 and 33**. However, we can choose any number on the breadboard.
- **i** and **j** will be used for the **for** loop.

```
int delay_ms = 200;

int arr_pin[5]={32,33,25,26,27};

int i, j;
```

**Step-2:** The next task is to initialize under **setup()** function

- Range-based for loop for array of LED's. For loop is used for iteration , its same as we did in other programming language like python, java etc
- To configure these pins, we use the **pinMode()** function.
- **PinMode()** configures the specified pin(arr\_pin) to behave either as an input or an output. As we want to act as output we are writing **OUTPUT** here
- **Syntax:** *pinMode(pin, mode)*
- Pin: Which pin do we need to set
- mode: Set the mode INPUT, OUTPUT,
- Set a delay of **200ms**.

```
void setup() {
    for (int i=0; i<5; i++) {
        pinMode(arr_pin[i], OUTPUT);
    }
    delay(200);
}
```

**Step-3:** Execution of the main task. Blink an LED from higher to lower

All the main processes need to be set under **void loop()**.

<ul style="list-style-type: none"> <li>• For loop used to on and off LED'S pin</li> <li>• The pin needs to be programmed to be either ON or OFF, we can command it to be ON (output 5 volts), or OFF (output 0 volts).</li> <li>• To make it on and off we need to use a function called <b>digitalWrite()</b>.</li> </ul>	
<pre>void loop() {   // loop from the lowest pin to the highest:   for (i=0; i&lt;5; i++) {     digitalWrite(arr_pin[i], HIGH);     delay(delay_ms);     digitalWrite(arr_pin[i], LOW);     //delay(delay_ms);   } }</pre>	
<p><b>Step-4 Blink an LED from highest pin to the lowest</b></p> <ul style="list-style-type: none"> <li>• for loop used to on and off LED'S pin</li> <li>• The pin needs to be programmed to be either ON or OFF, we can command it to be ON (output 5 volts), or OFF (output 0 volts).</li> <li>• To make it on and off we need to use a function called <b>digitalWrite()</b></li> <li>• Set up a delay</li> </ul>	
<pre>// loop from the highest pin to the lowest: for (j=4; j&gt;=0; j--) {   digitalWrite(arr_pin[j], HIGH);   delay(delay_ms);   digitalWrite(arr_pin[j], LOW);   //delay(delay_ms); } }</pre>	
<p>Let's upload the program to check if we are able to program our ESP32 module.</p> <p>For uploading, we need a USB cable.</p>	

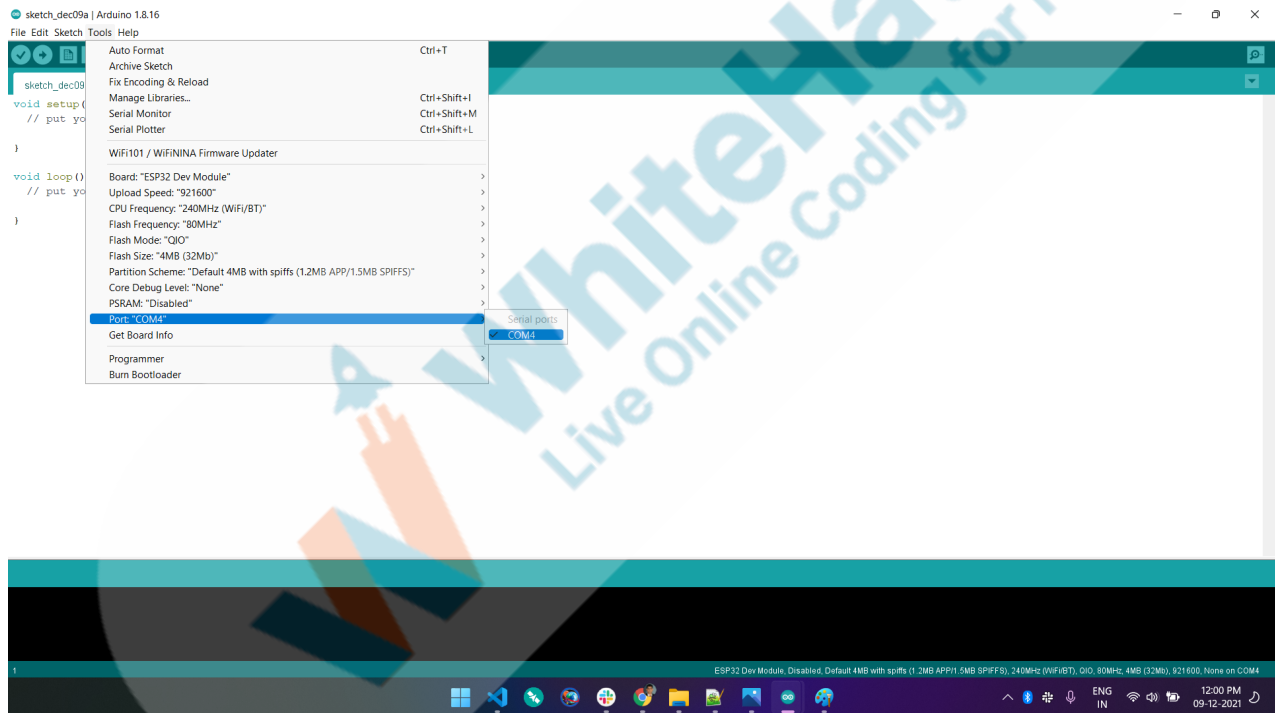
Take the USB cable, from WHJR-IoT kit.

Insert one end of the USB cables in the ESP32 USB port and another end in the Computer's USB port.

Now select the port for communication:

4. Go to **Tools**.
5. Select the Serial Port
6. Select the Port No

**Note:** We can find the port number by accessing the **device manager** on Windows.



Everything is ready now. The Arduino board is now ready to communicate with your PC. Instructions will be sent to the ESP32 board from your PC.

Click on the tick to verify the syntax.

OR

Click on Sketch >>Verify/Compile

When you hit the tick button, the program you have written in the Arduino IDE will be compiled for any errors.



```

C3_Knight_Rider | Arduino 1.8.16
File Edit Sketch Tools Help

C3_Knight_Rider$
int i, j;

void setup() {
  for (int i=0; i<5; i++) {
    pinMode(arr_pin[i], OUTPUT);
  }
  delay(200);
}

void loop() {
  // loop from the lowest pin to the highest:
  for (i=0; i<5; i++) {
    digitalWrite(arr_pin[i], HIGH);
    delay(delay_ms);
    digitalWrite(arr_pin[i], LOW);
    //delay(delay_ms);
  }

  // loop from the highest pin to the lowest:
  for (j=4; j>=0; j--) {
    digitalWrite(arr_pin[j], HIGH);
    delay(delay_ms);
    digitalWrite(arr_pin[j], LOW);
    //delay(delay_ms);
  }
}
  
```

Done Saving.

Now, the next thing is to upload the program,

Click on the arrow to upload the program.




OR

Click on **Sketch >>Upload**



If you look at your **ESP32**, you can see the 2 LEDs near Tx and Rx blinking. This is an indication of successful communication between your PC and ESP32 board. If the program has been uploaded successfully, you will see a message like **Done Uploading**. If the uploading process was not successful, you will see an error message accordingly.

<p>Now You will see the LED blinking pattern from lower to highest and higher to lower.</p> <p>This is how we can create blinking patterns with an LED and a microcontroller.</p>	
<b>WRAP-UP SESSION - 05 mins</b>	
<p><b>Activity details</b></p> <p><b>Following are the WRAP-UP session deliverables:</b></p> <ul style="list-style-type: none"> <li>• Appreciate the student.</li> <li>• Revise the current class activities.</li> <li>• Discuss the quizzes.</li> </ul>	
<p style="text-align: center;"><b>WRAP-UP QUIZ</b> Click on In-Class Quiz</p>	
<p><b>Activity Details</b></p> <p><b>Following are the session deliverables:</b></p> <ul style="list-style-type: none"> <li>• Explain the facts and trivia</li> <li>• Next class challenge</li> <li>• Project for the day</li> <li>• Additional Activity (optional)</li> </ul>	
<p style="text-align: center;"><b><u>FEEDBACK</u></b></p> <ul style="list-style-type: none"> <li>• <b>Appreciate and compliment the student for trying to learn a difficult concept.</b></li> <li>• <b>Get to know how they are feeling after the session.</b></li> <li>• <b>Review and check their understanding.</b></li> </ul>	
<p style="text-align: center;"><b>Teacher Action</b></p>	<p style="text-align: center;"><b>Student Action</b></p>
<p>You get hats-off for your excellent work!</p> <p>In the next class, _____</p>	<p><i>Make sure you have given at least 2 hats-off during the class for:</i></p>

	<div>Creatively Solved Activities  +10</div> <div>Great Question  +10</div> <div>Strong Concentration  +10</div>
<b>PROJECT OVERVIEW DISCUSSION</b> Refer the document below in Activity Links Sections	
Teacher Clicks	<div>✕ End Class</div>
<b>ADDITIONAL ACTIVITIES</b> (Optional)	
<ul style="list-style-type: none"> <li><b>Additional Activities</b></li> </ul>	

ACTIVITY LINKS		
Activity Name	Description	Links
Teacher Activity 1	Blink LED Reference Code	<a href="https://github.com/procodin/gclass/PRO-C243-Teacher-Activity">https://github.com/procodin/gclass/PRO-C243-Teacher-Activity</a>
Teacher Reference 1	Project	<a href="https://s3-whjr-curriculum-uploads.whjr.online/4e49727d-9025-4edb-b2b0-8ceba4d294cb.docx">https://s3-whjr-curriculum-uploads.whjr.online/4e49727d-9025-4edb-b2b0-8ceba4d294cb.docx</a>
Teacher Reference 2	Project Solution	



		<a href="https://github.com/procodingclass/PRO-C243-Project-Solution">https://github.com/procodingclass/PRO-C243-Project-Solution</a>
Teacher Reference 3	In_Class Quiz	<a href="https://s3-whjr-curriculum-uploads.whjr.online/50f5b87d-6bf3-478c-abfe-5aa67354c2b3.docx">https://s3-whjr-curriculum-uploads.whjr.online/50f5b87d-6bf3-478c-abfe-5aa67354c2b3.docx</a>
Student Activity 1	Reference Code	<a href="https://github.com/procodingclass/PRO-C243-Student-Activity">https://github.com/procodingclass/PRO-C243-Student-Activity</a>