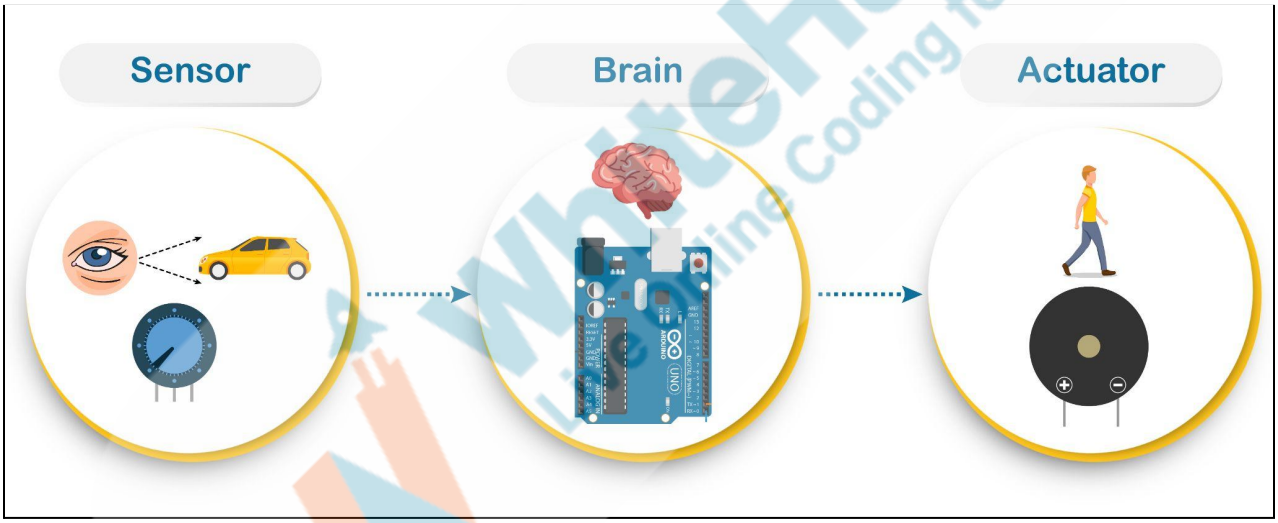
Topic	Serial communication	
Class Description	**Students will learn about a new microcontroller “Arduino”. They will also learn how 2 microcontrollers communicate with each other.**	
Class	**PRO C264**	
Class time	**50 mins**	
Goal	• Understanding the basics of Arduino. • Connecting 2 microcontrollers with each other. • Serial communication between controllers.	

Resources Required	• Teacher Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen
	  ○ Smartphone
	• Student Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen	

Class structure	**Warm-Up** **Teacher-Led Activity** **Student-Led Activity** **Wrap-Up**	**10 mins** **15 mins** **15 mins** **10 mins**

WARM-UP SESSION - 10 mins	
Teacher Action	**Student Action**
Hey <student's name>. How are you? It’s great to see you! Are you excited to learn something new today?	**ESR**: Hi, thanks! Yes, I am excited about it!

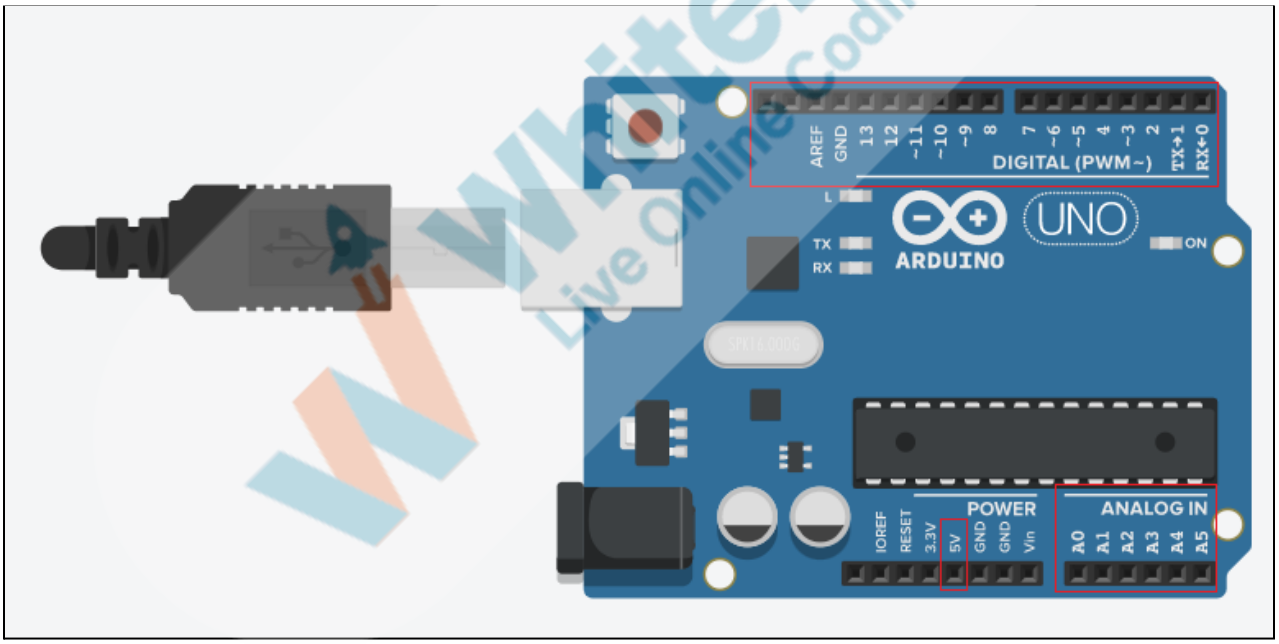
Following are the WARM-UP session deliverables: • Greet the student. • Revision of previous class activities. • Quizzes.	Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Activity Details **Following are the session deliverables:** • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
TEACHER-LED ACTIVITY 15 mins	
Teacher Initiates Screen Share	
• **Understanding the basics of Arduino.** • **Connecting 2 Arduinos with each other.** • **Sharing data.**	
Teacher Action	**Student Action**
Do you remember what we did in the last class? Great, let's do a quick revision. Which sensor did we use for **distance measurement**? Great, correct answer. Well done. Also, which **controller** did we use in the last class? Wow, you remember things pretty well.	**ESR** : Varied. **ESR** : Ultrasonic sensor. **ESR** : ESP32

Do you know how a human body works? Great, then tell me which organ of your body controls our **thoughts**, **speech**, **movement** and basically every function of the body? If you have guessed that the **brain** controls each and every aspect of our body, then you are right! Just like our **eyes, nose, tongue (all sensory organs)** and **limbs (hands, legs etc)** are controlled by our **brain**, in the similar manner the **ESP32** controls all the electronic components attached to it. It is like an **electronic brain**.	**ESR :** Yes **ESR** : Varied
	
Now we have extensively used ESP32 in the previous classes, and we know it's a very powerful electronic brain, but there are certain other **microcontrollers** or **processors** as well which can be used to control things like **Arduino**, **ESP8266**, **Raspberry pi,** etc. So the question comes, why use any other **controller** like **Arduino** if we have **ESP32**?	**ESR:** Varied

So there are certain reasons why we can use Arduino over ESP32,

a) Arduino is much simpler to use than ESP32.
b) ESP32 does not have a dedicated **5 volt** output pin, it can only give up to a maximum of **3.3 volts**, which means that you cannot directly connect those components with ESP32, which requires 5 volts to operate, whereas **Arduino** has a dedicated **5 volt** pin.
c) Also, ESP32 is more of a **microprocessor**, rather than a **microcontroller**.

Let's have a look at the Arduino architecture.

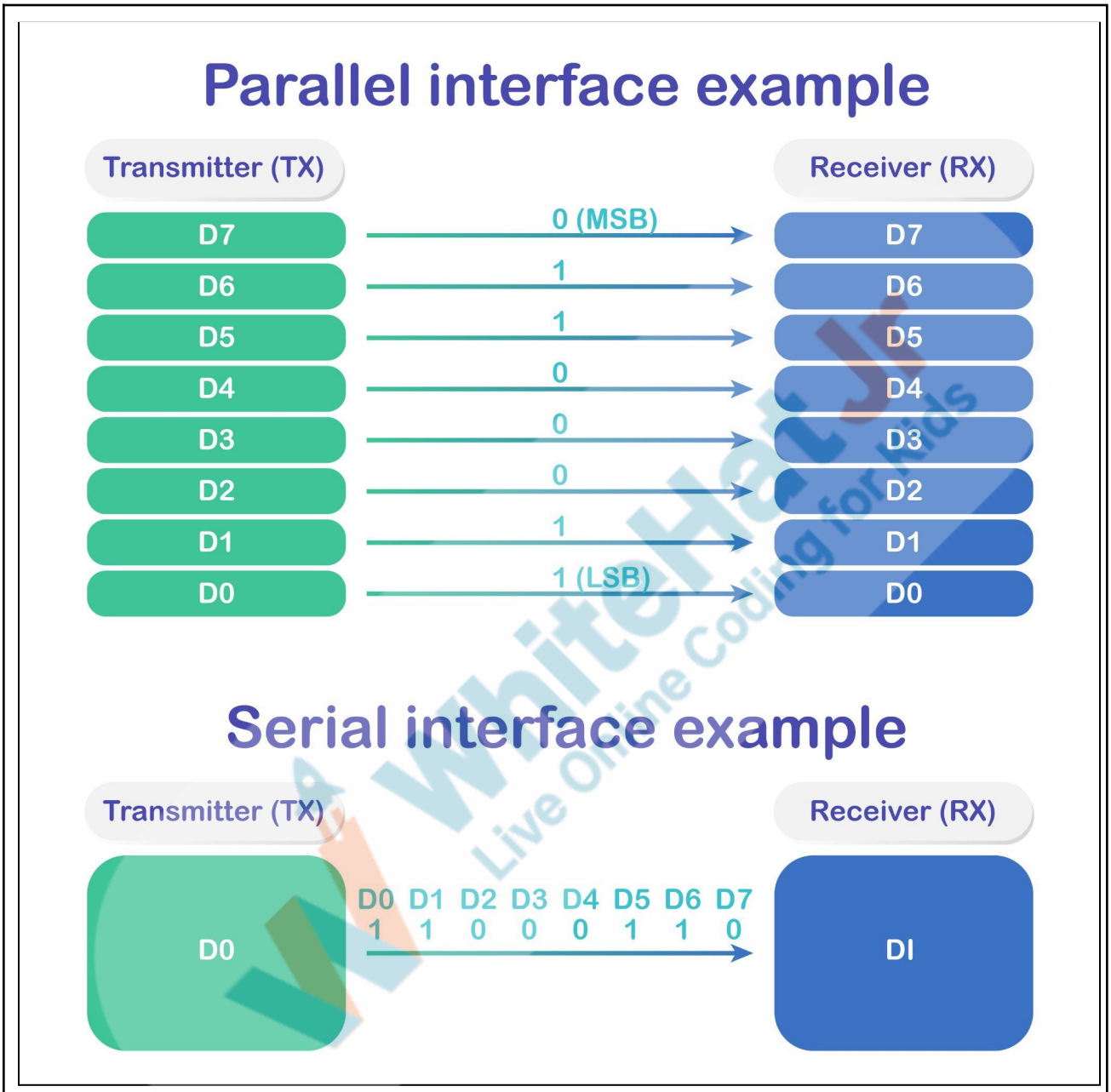


We can clearly see that Arduino has,

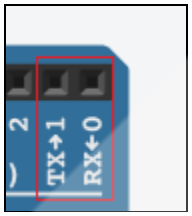
a) **Digital pins** (0 to 14)

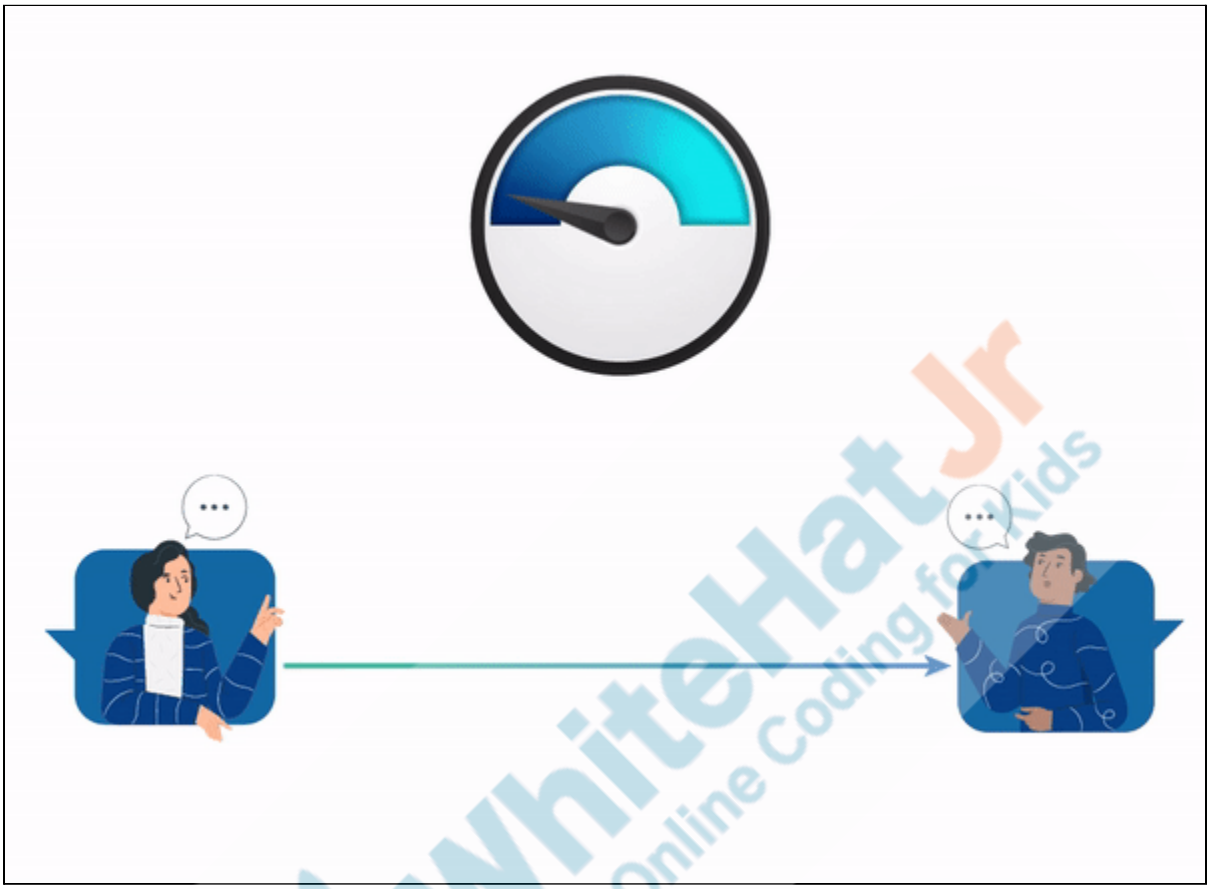
b) **Analog input pins** (A0 - A6) c) **5 volt** and **3.3 volt power pins.** Now you must have seen that there is a USB cable connected between your ESP32 and your laptop. Can you tell what USB stands for? It stands for **Universal Serial bus**. Do you remember the word **Serial** used in earlier classes? Great, do you recall the command **Serial.begin(9600)?** Can you tell me what exactly this line of code does? Great Answer! Now we have 2 important terms in the above given definition. a) **Serial communication.** b) **9600 bits per second.** Let's understand these terms one by one. Can you tell me what you understand by the term **Serial communication**? Serial communication is a **communication protocol** or a **set of rules**, with the help of which 2 machines can	**ESR:** Varied **ESR:** Yes **ESR:** Yes **ESR**: This command initializes the **Serial communication** between your **controller** and your **connected laptop through the USB,** at a speed of **9600 bits per second**. **ESR**: Varied

exchange data in a **serial manner**. Now by **serial**, we mean, all data bits travel one after another. There is only **1 data wire** between **transmitter** and **receiver**. There is a **Parallel communication** protocol as well, where **many data** bits can travel **simultaneously** through **multiple wires** connected between **transmitter** and **receiver**.	



Can you tell me the **sensory organs** which help us to communicate with other human beings?	**ESR:** **Ears** help us to listen. **Mouth** helps us to speak.

Great, just like we receive external sound with the help of our **ears**, the machine receives external signals with the help of the **receiver pin (RX pin 0).** Also, like we have a mouth to talk to, the machines have the **transmitter pin (TX pin 1)** to send data signals to other machines.	
	
Now let's try to understand the second term, which is **9600 bps(bits per second)** or the **baud rate.** Would you like to try and explain it to me? The baud rate is the rate at which **information** or **data** is transferred from one machine to another. Now **9600 bps** or **9600 baud rate** means that one machine can send or receive a **maximum** of **9600 data bits** in **one second.**	**ESR: Varied**



https://s3-whjr-curriculum-uploads.whjr.online/661ea912-d5b3-4c37-8927-360ec3217e58.gif

Now, enough with the theoretical part, let's do some hands-on coding so that we can understand how 2 **Arduinos** communicate with each other via **serial communication**. Are you excited? Great. Open the tinkercad simulator, sign in if required and click on Create new Circuit button.	**ESR:** Yes

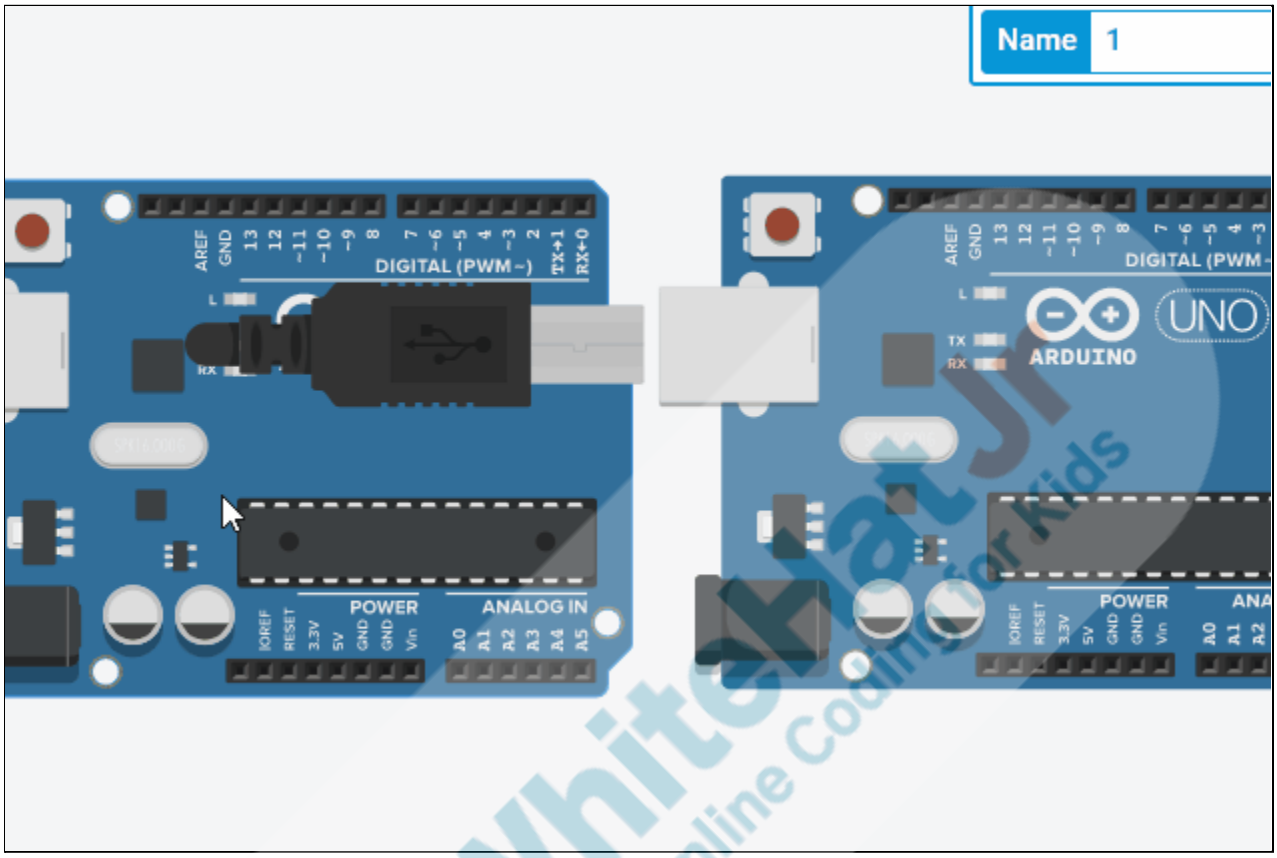
Note: We didn't use the wokwi simulator because we can't code 2 controllers simultaneously. Also, we have covered tinkercad in C242. You can refer to it.

Once you have created a new project,

- Drag out **2 Arduinos** into your project workspace.
- Rotate them using the **R key** on your keyboard, so that they get adjusted properly in your workspace.
- Do the following connections,

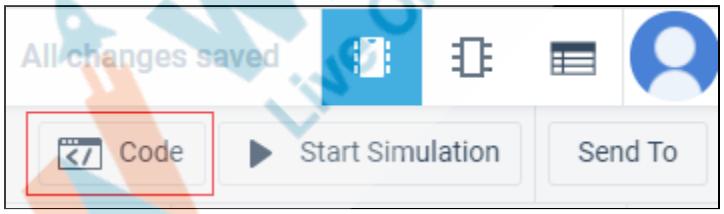
Arduino 1	Arduino 2
RX	TX
TX	RX
GND	GND

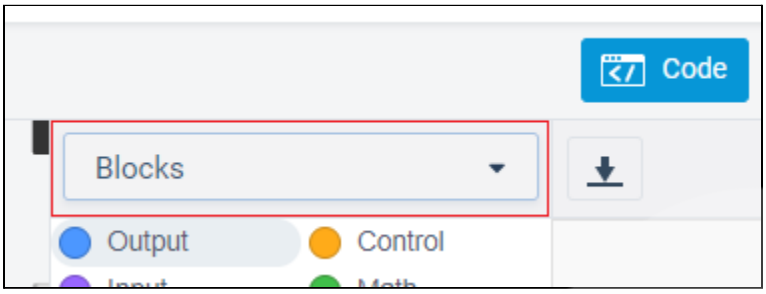
You can refer to the following gif.



S3 link:
https://s3-whjr-curriculum-uploads.whjr.online/feeaa55a-d3e2-4705-ae50-1773cb5434cd.gif

Would you like to try and explain, why did we connect the ground pin of both the Arduino together? Whenever we connect the **ground pins** of 2 or more electronic devices, we are trying to create a **common point of reference**. Let's try to understand this statement with the help of an example. Let's assume there are 2 people from different countries.	**ESR:** Varied

One is from the **US**, who can communicate only in **English**, and the other person is from **France**, who understands only **French**. Now, tell me if they both will be able to communicate with each other? If yes, then why yes and if no, then why no? True, for proper communication, both of them should have a common reference, which is the **"set of alphabets (A-Z)"** in our case. Just like humans need a common reference to communicate, machines also need a common reference to communicate and we provide them a common reference by connecting both their ground pins. Once the connections are made, let's start with the coding part as, Click on the **code** button in the top right corner.	**ESR:** No, because for communication there has to be a **common reference** or **common language**.
	
A code menu will appear which will let you code your Arduino using the **coding blocks**, but we need to code our Arduino in the **text mode**. For that, click on the **Blocks** option.	

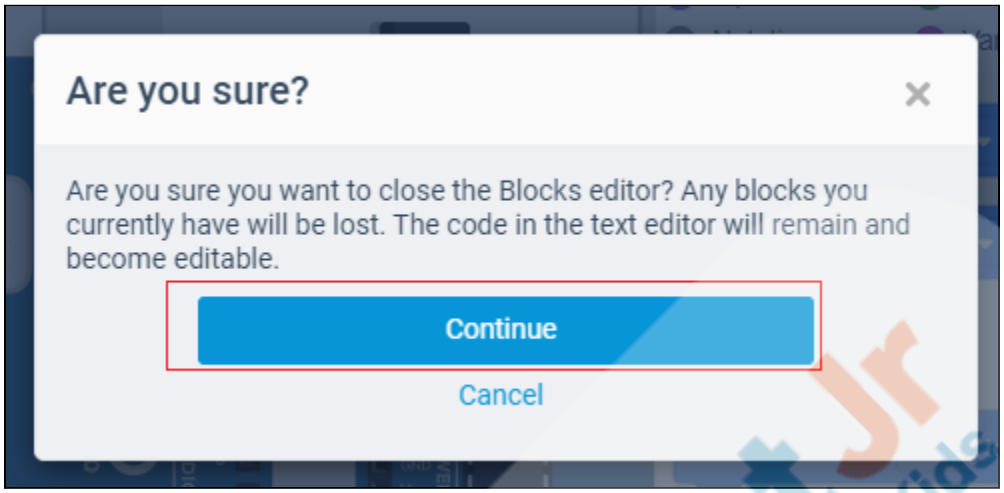


A **dropdown** menu will appear, which will let you select between **Blocks, Blocks + Text** and **Text mode**.

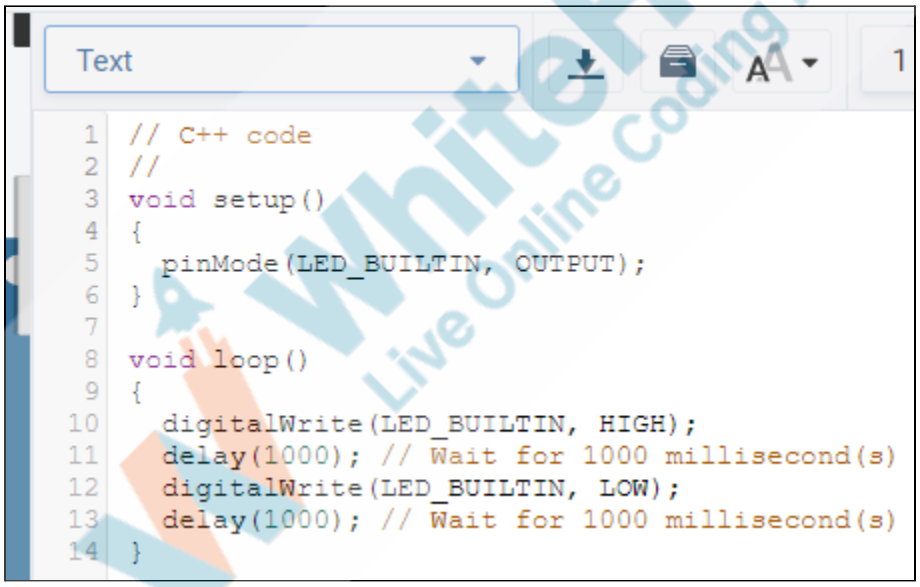
Click on the **Text** mode.

A pop up will appear which says, once you are in the text mode, you won't be able to back in the block mode.

Click on the **Continue** button.



The **Text** mode will open with some default code.



Remove all the code written within the **void setup()** and **void loop()** methods.

We need to make sure that the code which we will write, is for **Arduino 1**. For that you can check which Arduino is selected, using the toggle menu on the **right** corner.

Now once it's confirmed that we have selected Arduino 1, then let's begin with the coding part.

Within the void setup function, write the command, **Serial.begin(9600);**

This method will initialize the **serial communication** between both the **Arduinos** with a **baud rate** of **9600 bps (bits per second)**.

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

Now let's send a message from Arduino 1 to Arduino 2 continuously, at an interval of 1 second.

For that, write the code within the **void loop()** method as,

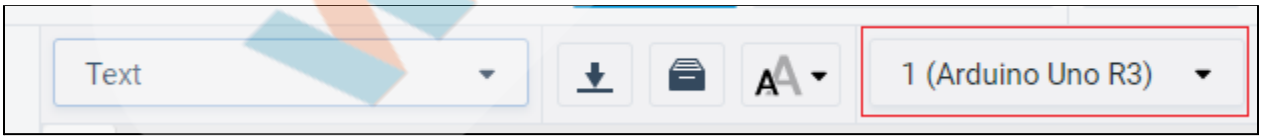
Serial.print("hello i am arduino 1");
delay(1000);

This piece of code will **send** the string **"hello i am Arduino 1"** towards **Arduino 2** at an interval of 1 second.

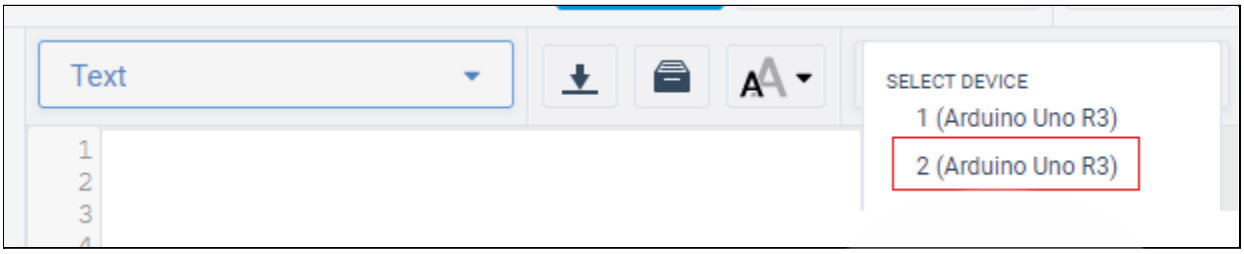
```
void loop()
{
  Serial.print("hello i am arduino 1");
  delay(1000);
}
```

Now we know that our Arduino 1 is transmitting a message towards our Arduino 2. So let's write some code for our Arduino 2 as well, where it can receive that message.

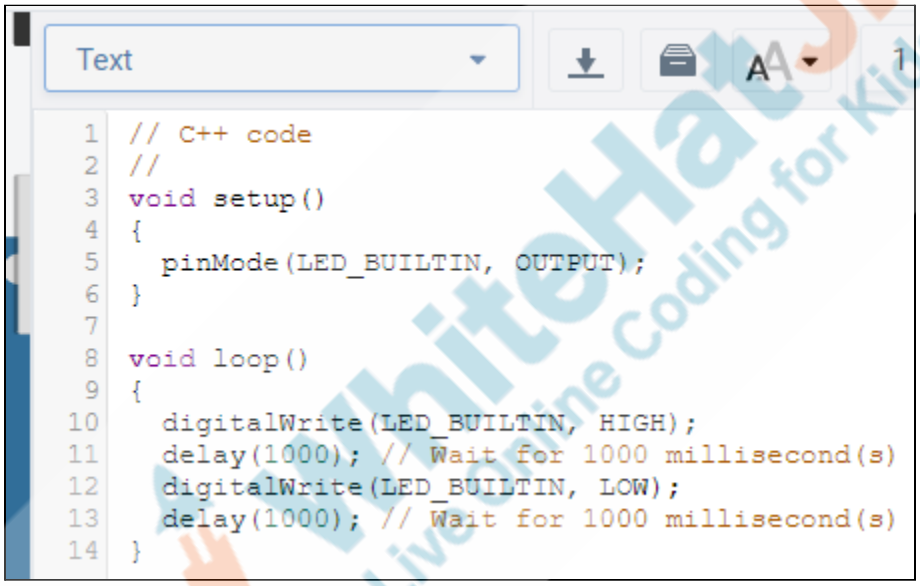
To do so, toggle the **controller** menu.



A dropdown menu will appear. Select **2 (Arduino Uno R3).**



The **Arduino 2** code window will open with some default code.



Remove all the code written within the **void setup()** and **void loop()** methods.

Within the **void setup()** method, write the code,

Serial.begin(9600);
Serial.setTimeout(100);

Note: For proper data exchange, make sure that the baud rate for both the Arduino should be the same.

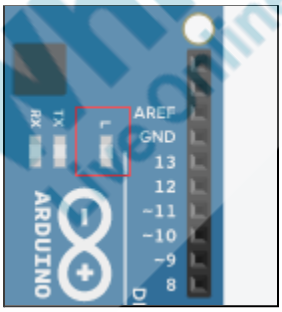
We will discuss the **setTimeout()** method later in the lesson.

```
void setup()
{
  Serial.begin(9600);
  Serial.setTimeout(100);
}
```

In the **void loop()** method check, if there is any data available to receive using the **.available()** method, and if the data exists, **read** it into a string variable using the

readString() method.	

<pre>void loop()
{
  if (Serial.available()){
  String data = Serial.readString();
  }
  }
}</pre>	

Now the question comes, how do we know that **Arduino 2** has received the data? Any ideas? *Note: Let the student try and answer.* We can glow the **inbuilt LED,** which is already connected internally with **pin 13** of Arduino.	**ESR:** Varied
	
To do so, first, we need to configure pin **13** as **output** within the **setup()** method. For that, use the **pinMode()** method as, **pinMode(13, OUTPUT);**	

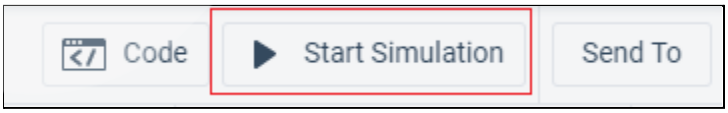
```
void setup()
{
  Serial.begin(9600);
  Serial.setTimeout(100);
  pinMode(13, OUTPUT);
}
```

Next, in the **loop** method, after receiving the message, check if that **message** is the same as the one sent by the Arduino 1. If that comes out to be **true**, turn on the LED connected to **pin 13**, using the **digitalWrite()** method as,

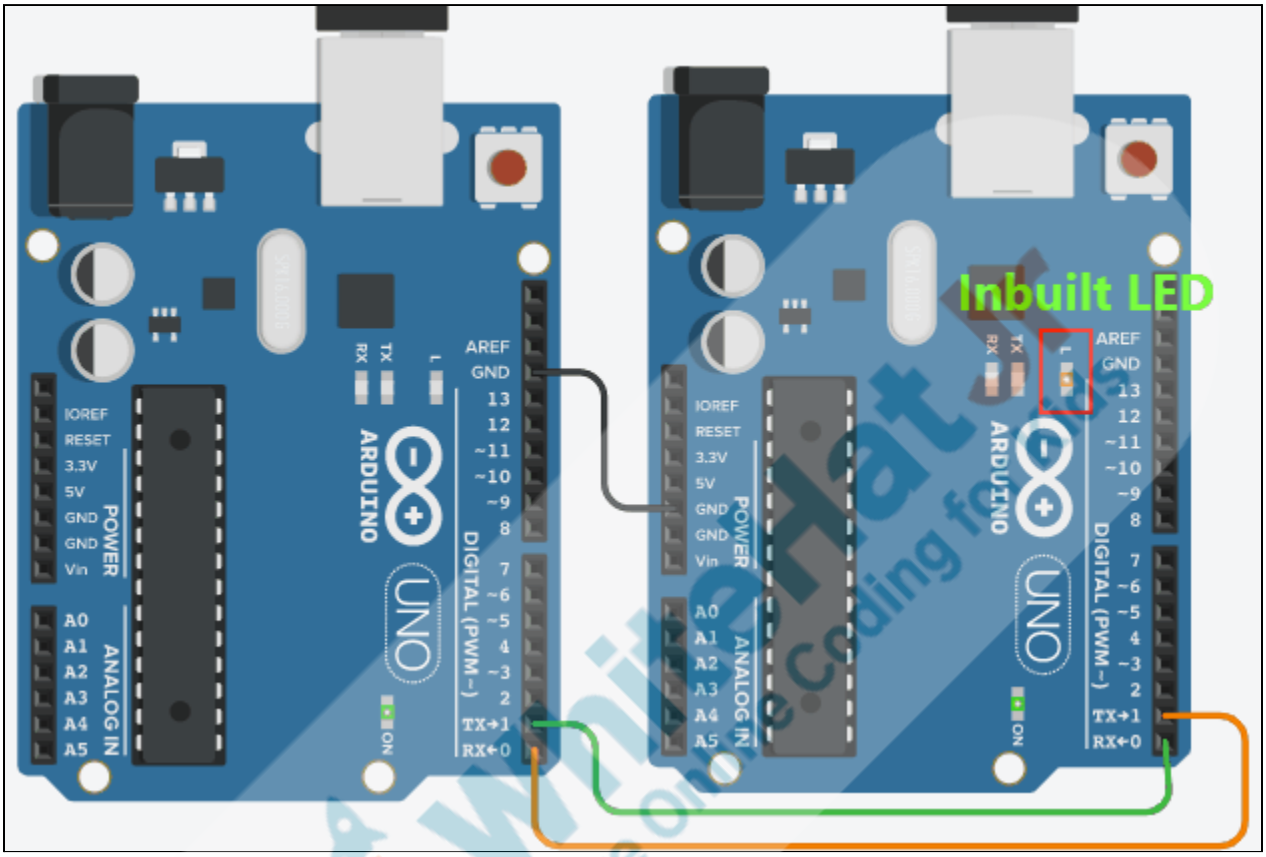
if (data == “hello i am arduino 1”){
digitalWrite(13 , HIGH);
delay(500);
digitalWrite(13 , LOW);
}

```
void loop()
{
  if (Serial.available()){
    String data = Serial.readString();
    if (data == "hello i am arduino 1"){
      digitalWrite(13,HIGH);
      delay(500);
      digitalWrite(13,LOW);
    }
  }
}
```

Click on the start simulation button, to see the output.



So now if Arduino 1 is transmitting the message, Arduino 2 is receiving that message properly, you will be able to see

the flashing of the inbuilt LED on Arduino 2.	
 S3 link : https://s3-whjr-curriculum-uploads.whjr.online/127c78ea-507d-49d0-88f9-4e4e242f0b39.gif	
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.	
Teacher Stops Screen Share	
STUDENT-LED ACTIVITY- 15 mins	
• **Ask the student to press the ESC key to come back to the panel.**	

<table>
<tr><td colspan="2">

- **Guide the student to start Screen Share.**
- **The teacher gets into Full Screen.**

</td></tr>
<tr><td colspan="2">Student Initiates Screen Share</td></tr>
<tr><td colspan="2">

ACTIVITY

- **Connecting 2 Arduinos with each other.**
- **Sharing sensor data from one Arduino to another.**
- **Serial communication.**

</td></tr>
<tr><td>Teacher Action</td><td>Student Action</td></tr>
<tr><td>

Now we have learned how data can be transferred from one Arduino to another with the help of **Serial communication**. Now let's use this principle to control the brightness of an **LED** that is connected to **Arduino 2**, using a **potentiometer**, which is connected with Arduino 1.

To do so, let's open the tinkercad link and create a new circuit.

Drag out the following components:

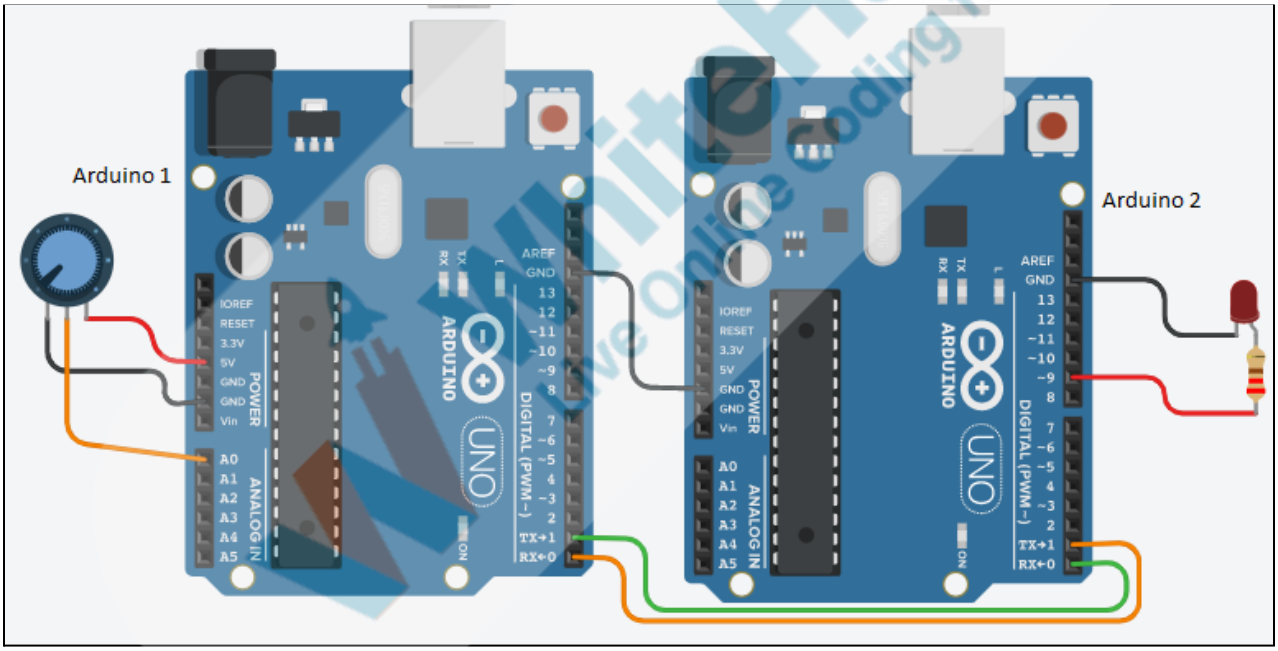
- 2 Arduinos
- 1 Potentiometer
- 1 Led
- 1 Resistor

Make the connections as per the below given table,

Note: Terminals with the same color should be connected together.

</td><td></td></tr>
</table>

Arduino 1	Arduino 2	Potentiometer	LED	Resistor
5 volts	Ground 1	Terminal 1	Cathode (negative leg)	Terminal 1
A0	RX (Pin 0)	Wiper (Middle pin)	Anode (positive led)	Terminal 2
Ground1	TX (Pin 1)	Terminal 2		
Ground 2	Ground 2			
TX (Pin 1)	Pin 9			
RX (Pin 0)				



Once the connections are made, let's define the functionality of both the Arduino

Arduino 1 will read the **potentiometer** data and send it to **Arduino 2** using **Serial communication**.

For that, go to the text coding mode and in the void setup() method, initialize the serial communication using the **Serial.begin(9600)** command.

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

In the **void loop()** method, read the status of **A0 pin** using the **analogRead(A0)** method.

It is very easy to handle **string** data while doing serial communication, so we have converted the output of **analogRead()** method to type string using the **String** keyword.

Finally, let's catch the resultant into a string variable named **pot_val**.

The code for the above would look like this,

String pot_val = String(analogRead(A0))

Now we have the state of our potentiometer in the string **pot_val**. Let's send this data towards Arduino 2 at an interval of **500 ms** using the **.print()** method as,

Serial.print(pot_val);
delay(500);

```
void loop()
{
  String pot_val = String(analogRead(A0));
  Serial.print(pot_val);
  delay(500);
}
```

Now we have written the complete code for Arduino 1. Let's write the functionality of Arduino 2. For that, toggle

the controller menu and select **2 (Arduino Uno R3)**.

Clear the prior code, and within the setup() method,

- Initialize the **Serial communication** using the **.begin()** method. Make sure the **baud rate** is the same as **9600**, for both the Arduino.
- Configure **pin 9** as **output** using the **pinMode()** method.
- Set the serial timeout as **100 ms** using the **.setTimeout()** method.

```
void setup()
{
  Serial.begin(9600);
  pinMode(9,OUTPUT);
  Serial.setTimeout(100);
}
```

In the loop() method,

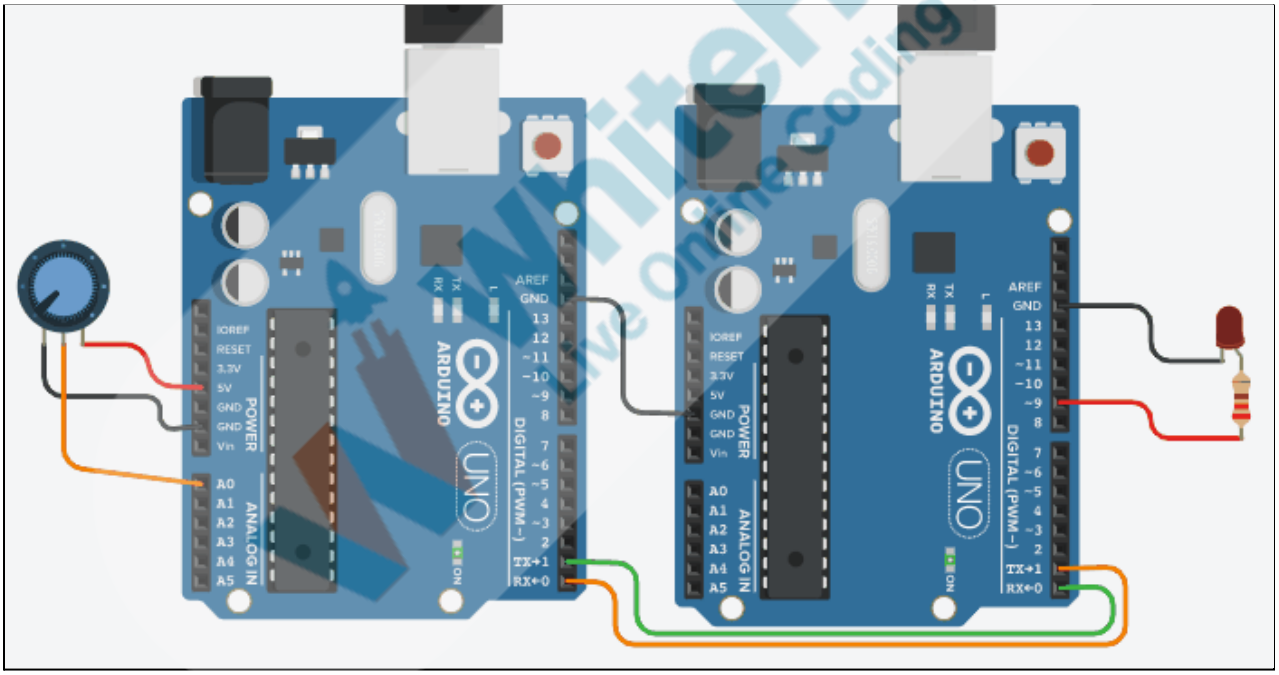
- Check if any data is available using the **.available()** method.
- If there is any data, read it using the **.readString()** method. The output would be a string, but we need to convert it to an integer. So use the **.toInt()** method for that. Finally, store the result in an integer variable.
- The output range of the potentiometer is from **0** to **1023**, but we need to convert it from **0** to **255**, as Arduino has an **8-bit PWM (0 - 255)**. So we will be using the **map()** method for that.
- Finally, let's control the LED using the **analogWrite()** method.

```
void loop()
{
  if (Serial.available()){
    int data = Serial.readString().toInt();  //  0-1023 range

    //  mapping data from 0-255
    data = map(data , 0 , 1023 , 0 , 255);
    analogWrite(9 , data);
  }
}
```

Click on the **Start Simulation** button and rotate the **potentiometer** to observe the change in the **LED brightness**.

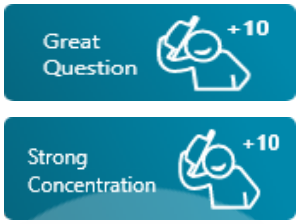
The output would look like,



S3 link :
https://s3-whjr-curriculum-uploads.whjr.online/3ae789e3-7084-42e3-8448-65798a28e257.gif

Great work. This all looks good.

Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 10 mins	
Activity details **Following are the WRAP-UP session deliverables:** ● Appreciate the student. ● Revise the current class activities. ● Discuss the quizzes.	
WRAP-UP QUIZ Click on In-Class Quiz	
Activity Details **Following are the session deliverables:** ● Explain the facts and trivia ● Next class challenge ● Project for the day ● Additional Activity (Optional)	
FEEDBACK ● **Appreciate and compliment the student for trying to learn a difficult concept.** ● **Get to know how they are feeling after the session.** ● **Review and check their understanding.**	
Teacher Action	**Student Action**
You get “hats-off” for your excellent work! In the next class, we will learn how to use an RTC module and create a smart clock using it.	*Make sure you have given at least 2 hats-off during the class for:* 

	
PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks 	
ADDITIONAL ACTIVITIES (Optional)	
Additional Activities	

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity 1	Simulator	Tinkercad
Teacher Reference 1	Teacher Activity Reference Code	https://github.com/procodingclass/PRO-C264-Teacher-Activity-Reference-Code.git
Teacher Reference 2	Student Activity Reference Code	https://github.com/procodingclass/PRO-C264-Student-Activity-Referen

		ce-Code.git
Teacher Reference 3	Project	https://docs.google.com/document/d/1ouovzXtwQcnKYeECqn6yzcbM4etWGeX155-MG_aadzY/edit?usp=sharing
Teacher Reference 4	Project Solution	https://github.com/procodingclass/PRO-C264-Project-Solution.git
Teacher Reference 5	In-Class Quiz	https://docs.google.com/document/d/171S7d46xVAhq87WZkeqB5BIrqIZhsNFceOGgckPcTr4/edit?usp=sharing
Student Activity 1	Simulator	Tinkercad