LECTURE V

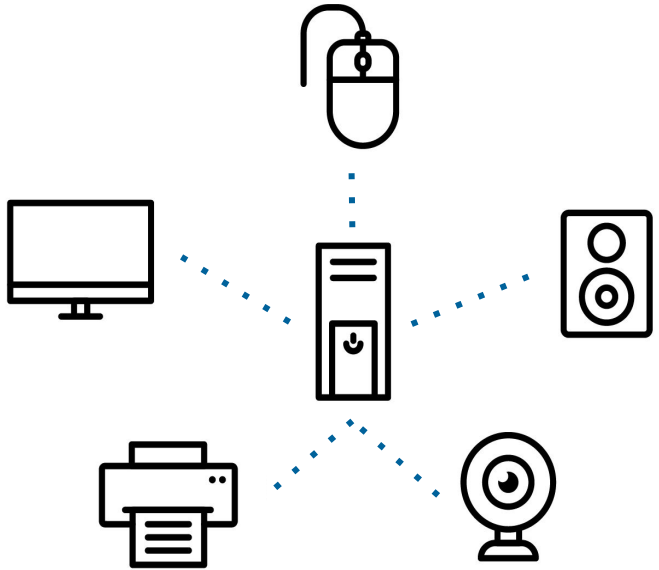
Communication Protocols I

SECTION I

What are Communication Protocols?

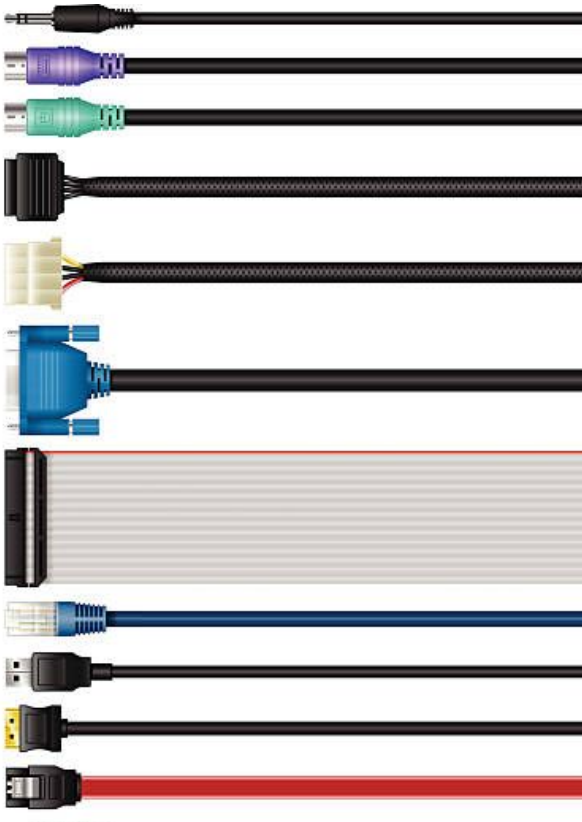
Communication

- Devices, like your home computer, **communicate** with other devices
  - i.e. mouse, monitor, printer, webcam, speakers
- Just like your computer, the Arduino board must communicate with devices too

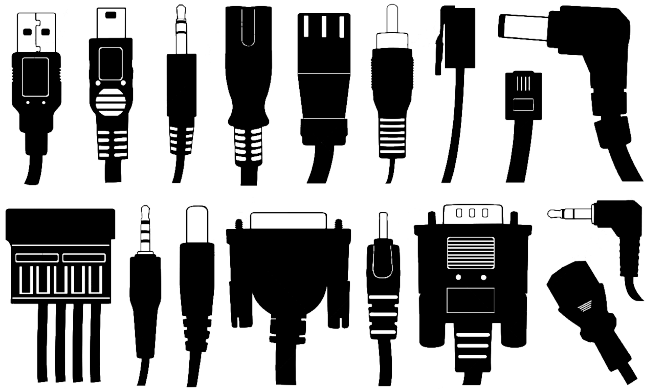


Communication Media

- Devices communicate over different **media**
  - i.e. copper wire, coaxial cable, radio, fiber optic wire, etc.
  - Some media are **wired** while others are **wireless**
- Devices communicate by **transmitting** and **receiving** electrical signals over a shared medium
  - Signals may be **digital** (1s and 0s) or **analog** (i.e. AM/FM radio)



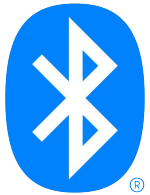
Communication Protocols

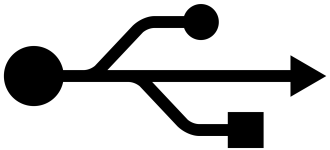


- Communicating devices **must share a common “language”** to understand messages transmitted between them
- The rules for how we send, receive, and interpret these signals are defined by **protocols**
- You already know of some protocols…

Communication Protocols (Cont'd)

- Your smartphone communicates with your wireless earbuds over a **Bluetooth** protocol
- All the laptops in the lecture hall connect to a wireless router using a **WiFi** protocol (a.k.a **IEEE 802.11x**)
- Your wired mouse connects to your computer using a **Universal Serial Bus** (**USB**) protocol

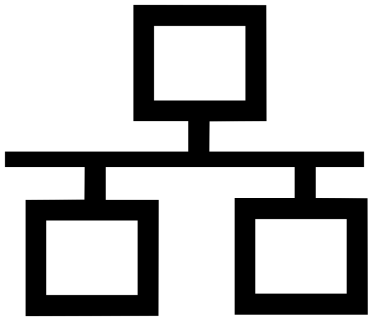


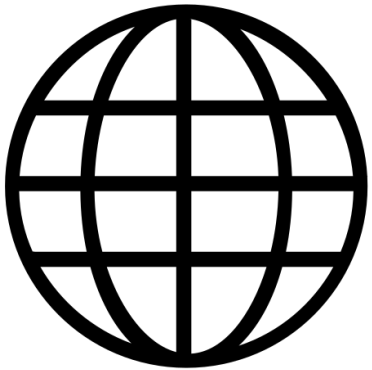


Communication Protocols (Cont'd)

I/A

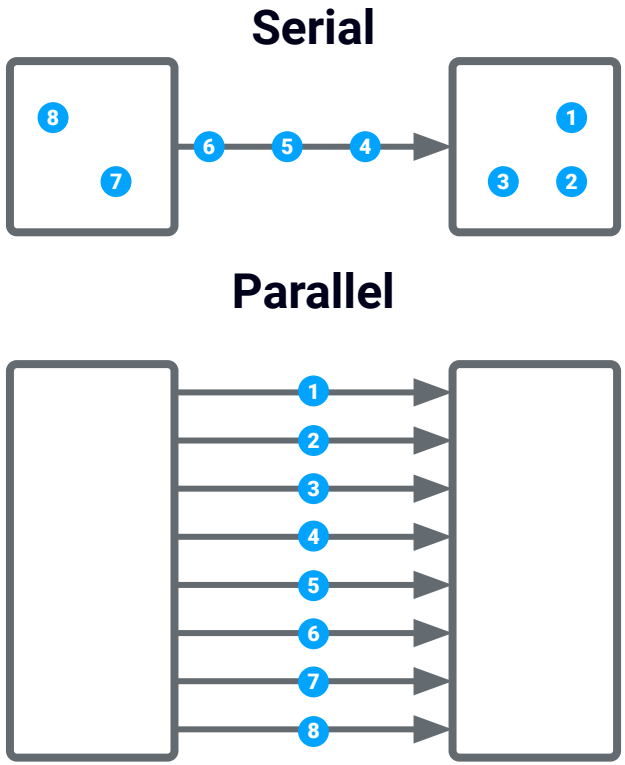
What are some more **examples** of communication protocols?





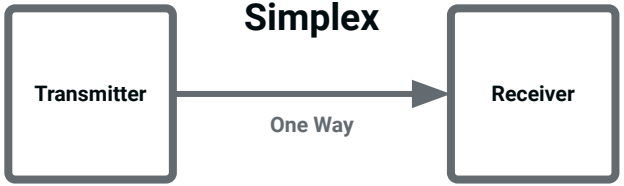
Serial vs Parallel Protocols

- **Serial** - bits are sent over a connection one by one to a device
  - Bits are sent in a specific order
  - The most common protocols are serial
- **Parallel** - multiple bits (often one byte) are sent simultaneously over a connection
  - This connection requires more wires, which takes up more space
  - Higher transmission rate

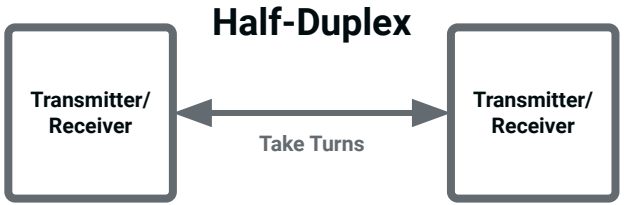


Transmission Modes

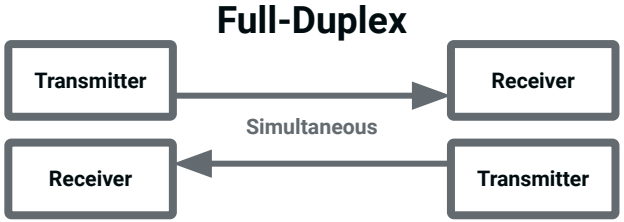
- **Simplex** protocols allow communication in only one direction



- **Half-Duplex** protocols allow communication in both directions but only in one direction at a time



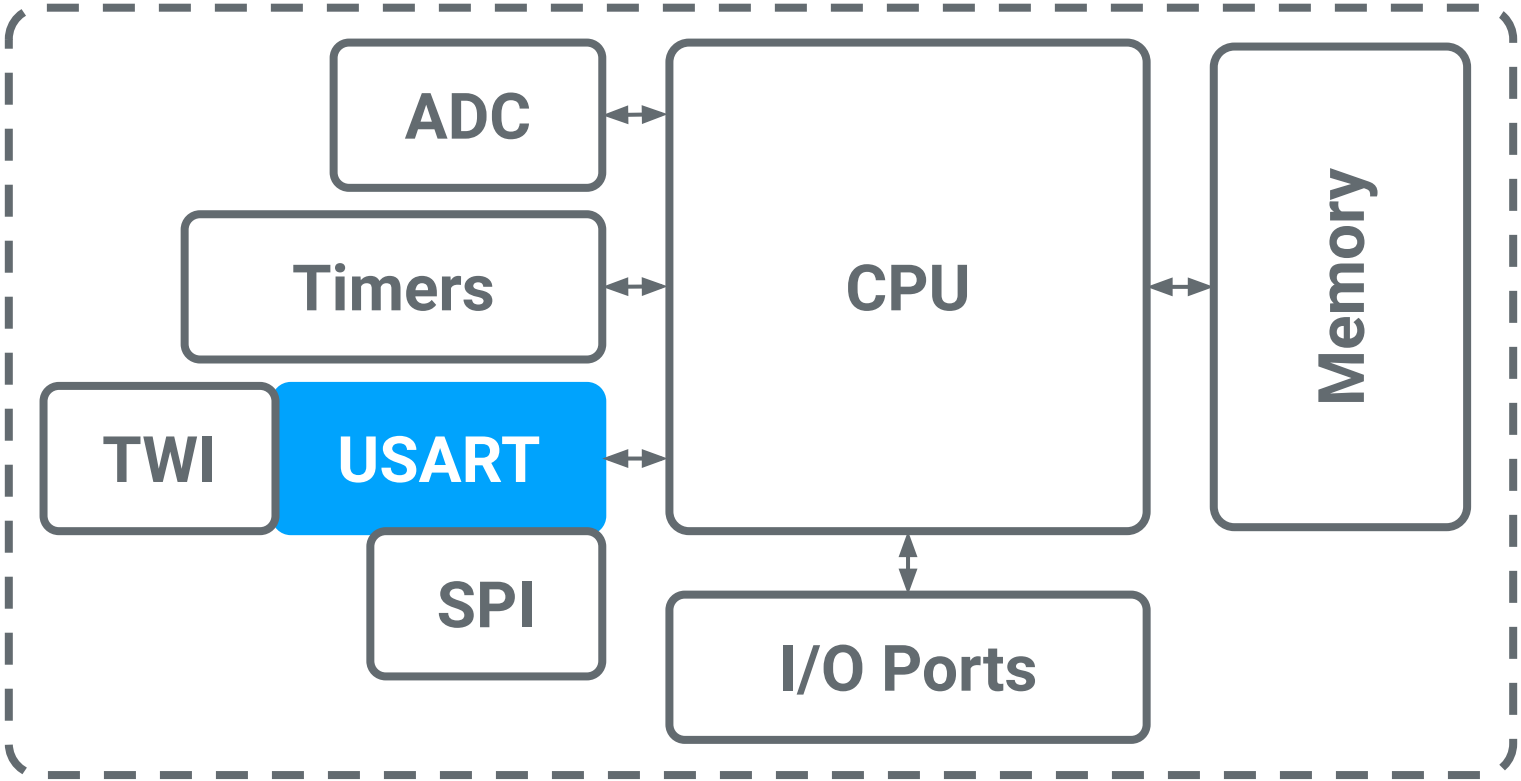
- **Full-Duplex** protocols allow communication in both directions simultaneously



SECTION II

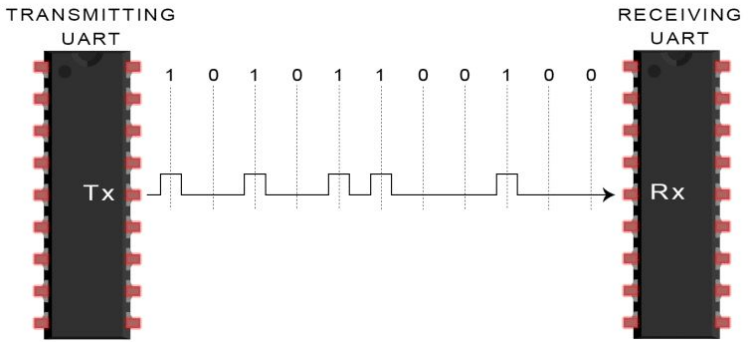
USART/UART

AVR Architecture



USART

- The **Universal Synchronous/Asynchronous Transmitter/Receiver** or **USART** is a hardware device that **serially communicates** with other USARTs
  - It is **integrated into microcontrollers** (as shown in the AVR Architecture slide)
  - In synchronous mode, the USARTs use a *shared* clock signal to time the transmission



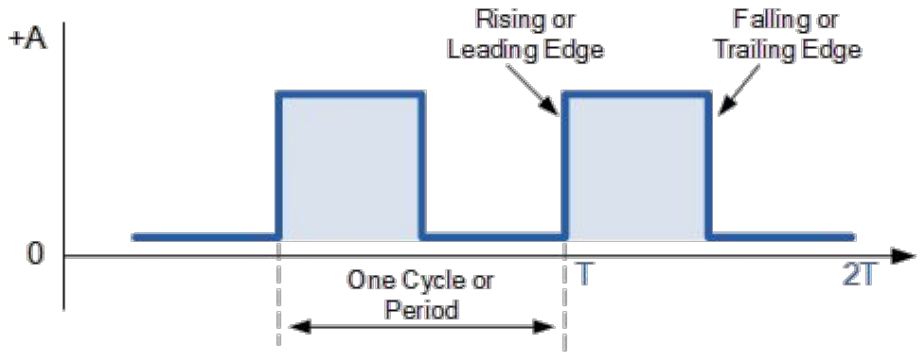
  - In **asynchronous mode**, there is **no shared clock signal**. This is functionally identical to a **UART** device (which is how we will use it)

UART Protocol

- The USART uses **UART protocol** when in asynchronous mode
  - Remember UART is a **communication protocol and a physical device**. It has rules for data transmission
- UART can be configured to **full-duplex, half-duplex,** and **simplex modes**
- There are only **two lines** (electrical connections)
  - There is **no clock signal**/line hence the “asynchronous” part of the title
- The data transmission speed is determined by a **baud rate**
  - It is quite **slow compared to other protocols**

Clock Signals

- **Digital signals** are timed to a **clock signal** - most often, a square wave with a 50% duty cycle
  - For example, in a serial signal, one bit of data is sent per clock cycle
    - The bit might be timed to the rising or falling edge of the clock signal
- **Clock speed** determines the rate at which data is transmitted
  - measured as **frequency (Hz)**
  - inversely proportional to cycle (or period)

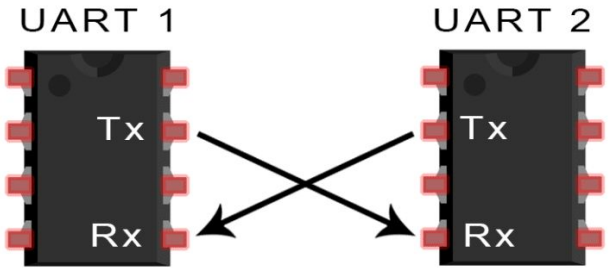


UART Baud Rate

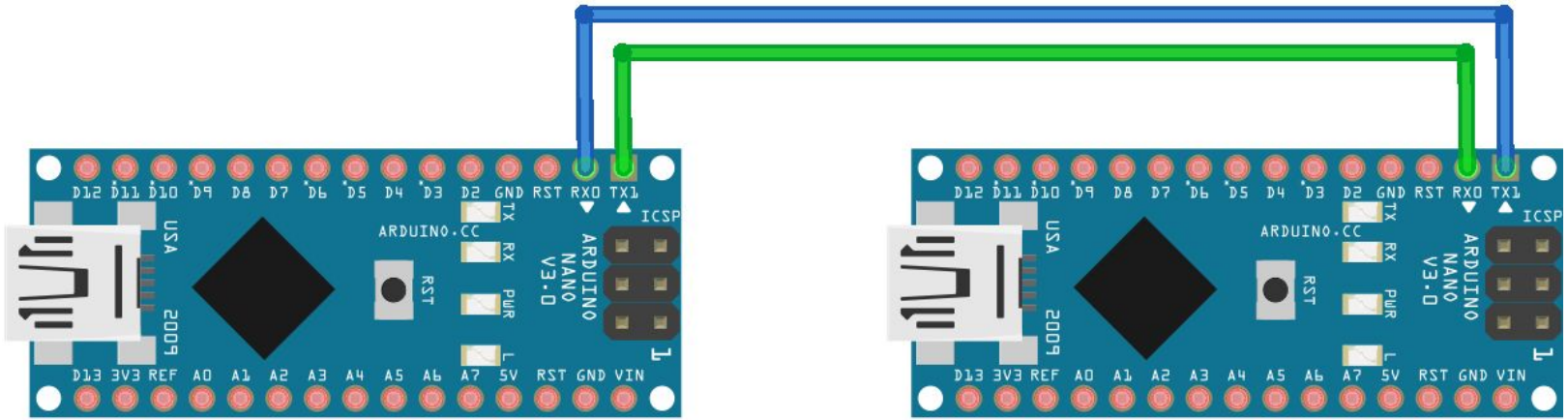
- A UART sends and receives transmissions at the speed of the **baud rate**
  - It is configured by the programmer on each UART
  - Measured in **bits per second**
- The baud rate **must be equal** on the two communicating UARTs to successfully send and receive the packets
  - Otherwise, messages are read at the wrong speed and become garbled

UART Layout

- Two UARTs may be connected as pictured
  - The **transmitter** (**TX**) pin of one UART is connected to the **receiver** (**RX**) pin of the other
  - The are two data lines, which **enables full-duplex communication**
- Data is sent from the transmitter of one UART to the receiver of the other UART



UART Layout (Cont'd)



Two Arduino Nanos connected via UART pins (TX/RX)

UART Frame Format

- Data is transmitted as segments of bits called **packets**
  - This data is sent one bit at a time (a.k.a **serially**) between UARTs
- Without modification, the UART packet follows this format:

UART Packet

UART Frame Format (Cont'd)

- A **start bit** indicates when the packet begins
  - The data transmission line is brought from a default HIGH voltage to a LOW voltage by the transmitting UART for one clock cycle
- The **data frame**, containing the data, follows the start bit
  - If the parity bit is used, it can be 5 to 8 bits. If the parity bit is not used, it can be 9 bits.

UART Packet

UART Frame Format (Cont'd)

- The **parity bit** is an **optional** bit comes after the data frame and is used by the receiving UART to check if the data is free of errors
  - If the parity bit is 0 (**even parity**), then the number of 1s in the data frame should be even
  - If the parity bit is 1 (**odd parity**), then the number of 1s in the data frame should be odd

UART Packet

UART Frame Format (Cont'd)

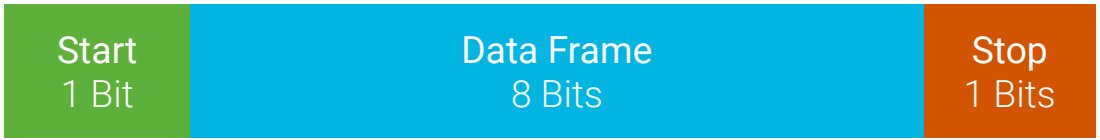
- Lastly, a **stop bit** indicates the end of a packet
  - The transmitting UART pulls the transmission line from a LOW to HIGH voltage for 1 or 2 clock cycles
- A USART frame looks identical to a UART frame
  - Remember that the only difference is that USART has a *shared* clock line

UART Packet

Arduino UART Frame Format

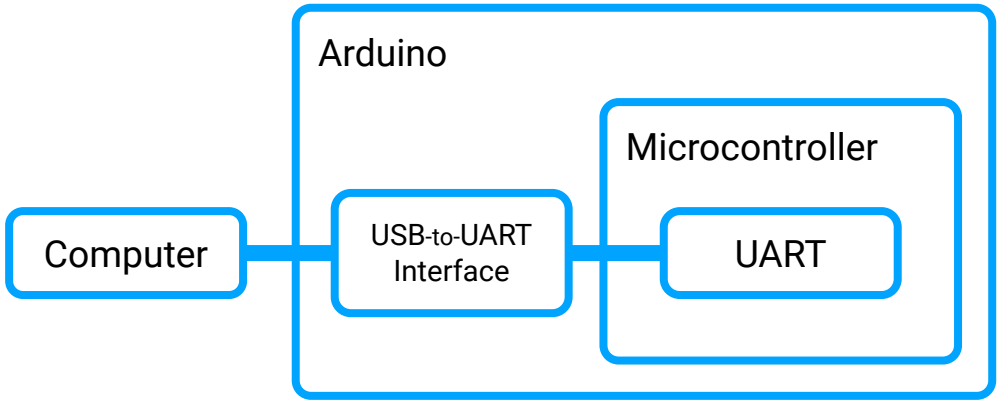
- Arduino's default UART frame is 10 bits in size by default
- The *optional* parity bit is removed
- The data frame is restricted to 8 bits (the most common size)
    - Consider that ASCII characters are 8 bits

UART Packet (Arduino)



USB on the Arduino

- Data is transmitted between the computer and Arduino Nano through a chip that converts USB signals to UART signals
    - Those converted signals are transmitted to and from the UART
- The Arduino's USB port is hardwired to the TX and RX pins of the board
- **Do not use the UART while simultaneously using USB**



SECTION III

Arduino Serial Library

Arduino Serial Library

- We can use the USART in our Arduino Nano to transmit data over UART protocol
  - To do so, we will enlist the help of the Serial library, which is already installed in the Arduino IDE
- The Arduino UNO and Nano have **only one USART**
  - One workaround is to use the SoftwareSerial library and emulate a second UART programmatically
  - Another workaround is to use a different protocol (to be discussed next lecture)

Serial.begin

- `Serial.begin(int baud_rate)`
  - `Serial` is an object that facilitates communication between the Arduino board and other devices over UART protocol
  - Pass `9600` bits per second as the argument for this member function
    - This is the default baud rate

```
void setup()
{
    pinMode(LED, OUTPUT);
    pinMode(RECEIVER, INPUT);
    Serial.begin(9600);
}
```

Serial.write

- `Serial.write` is used to transmit from the UART in the Arduino board

Common Variations

- `Serial.write(val)` - Sends `val`, a single byte of data
- `Serial.write(str)` - Sends `str`, a series of bytes represented by a string

```
Serial.write(62); // Sends 62
Serial.write("Hello"); // Sends “Hello” over multiple frames
```

Serial.available

- `Serial.available` returns true if the Arduino's UART has received any data
- The incoming data is stored in a sort of "mailbox" called a **receive buffer**. The buffer is ostensibly an array that stores the data
    - If the buffer is full, then the incoming data is dropped and lost forever

```
if (Serial.available())
{
    // If the buffer has data to be read, the program will enter this
    // block
}
```

Serial.read

- `Serial.read` returns the first byte of incoming serial data available (or -1 if no data is available
  - The data is returned as an `int`
  - Each time this function is called, the byte returned by it is dropped from the receive buffer

```
int incomingByte = Serial.read(); // Stores the incoming byte
```

SECTION III

Arduino UART Exercise

Arduino Echo Communication

I/A

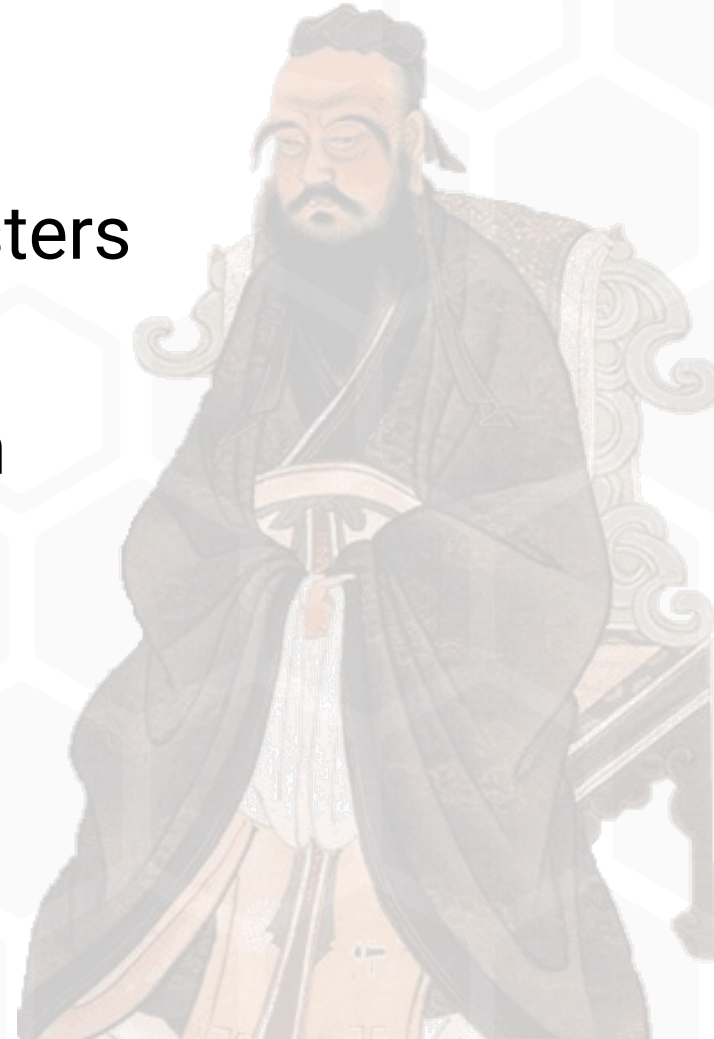
Write a program that satisfies the following requirements:

- You must write a program in which the **Arduino board “echoes” back any byte (0-255) sent to it over UART.**
- When the user enters a byte value (0-255) into the Arduino IDE Serial Monitor from their computer, the value is sent to the Arduino board. The board must send the same value back to the computer.
- The Arduino will **only “echo” back a value sent to it once**. It should not resend the value multiple times.
- The program must **run in an infinite loop**.

“The wise programmer who **masters** the **art of UART** shall establish a connection that **transcends** both distance and **baud rate**.”

Confucius (circa 551 BC)

Famous Misquotes



FAIR USE DISCLAIMER

Copyright Disclaimer under section 107 of the Copyright Act 1976, allowance is made for “fair use” for purposes such as criticism, comment, news reporting, teaching, scholarship, education and research.

Fair use is a use permitted by copyright statute that might otherwise be infringing.

Non-profit, educational or personal use tips the balance in favor of fair use.