

CSA Practical Communications

Sorry the photos are of low quality... it's all my camera can do

In this practical we create an Arduino project to help us learn how serial communications work

We will securely transmit data using a white led and receiving it using a photoresistor

We will be emulating how data is sent via Fiber Optic Cable

Hardware Needed

We will need the breadboard to place the project components on

For receiving...

- One 100K ohm resistor
- One Photoresistor



For transmission...

- One 220-ohm resistor
- One white LED

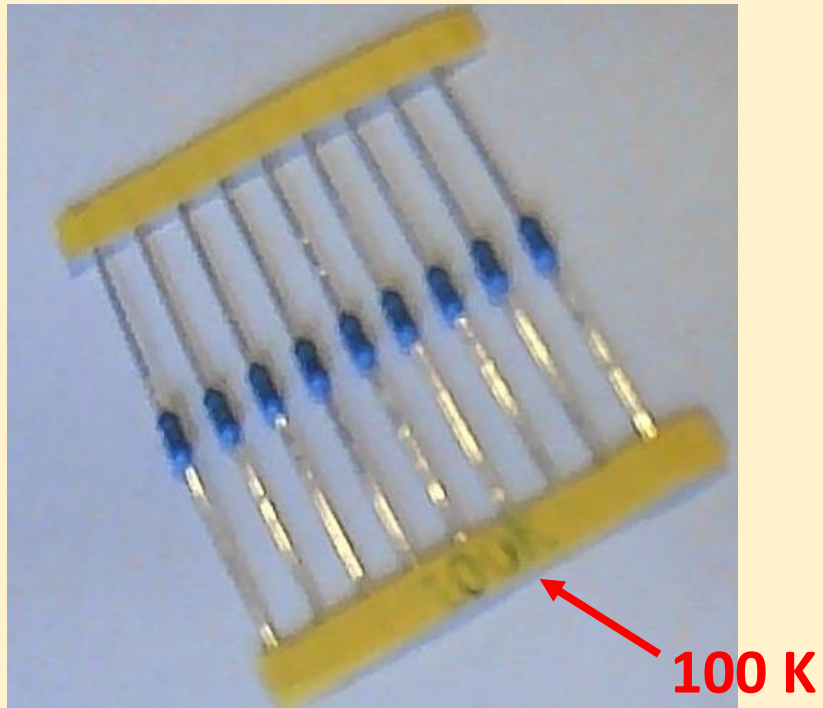


Make sure it's
the two-legged
LEDs and not
the four-
legged ones

You will also need some of the cables

Receiving

We are going to start playing with the receiving end



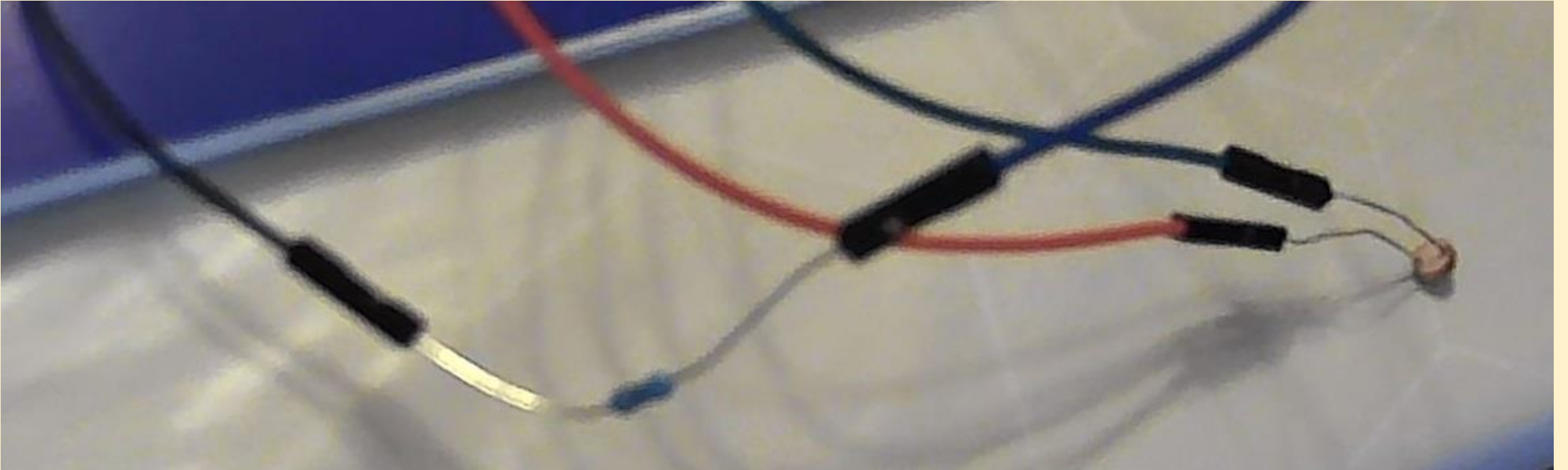
It has a flat top
with a lined
pattern on it



Inserting the photoresistor and 100 K resistor into the breadboard is difficult as their legs will just bend

To solve this, take four of the Male-Female cables; red, green, blue and black

Place the red and green cables on the photoresistor

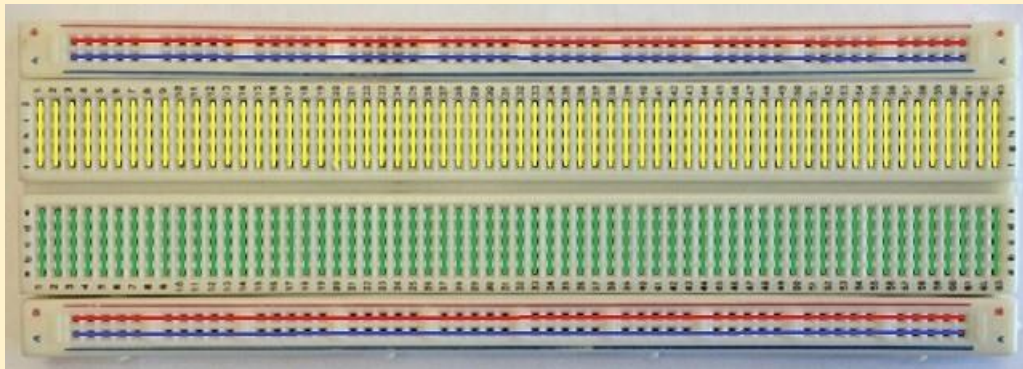


Place the blue and black cables on the 100K resistor

We will need to attach these to breadboard

But first we need to think about how the breadboard works... it's just a board with internal wiring that you can plug components into

The internal wiring is like this...



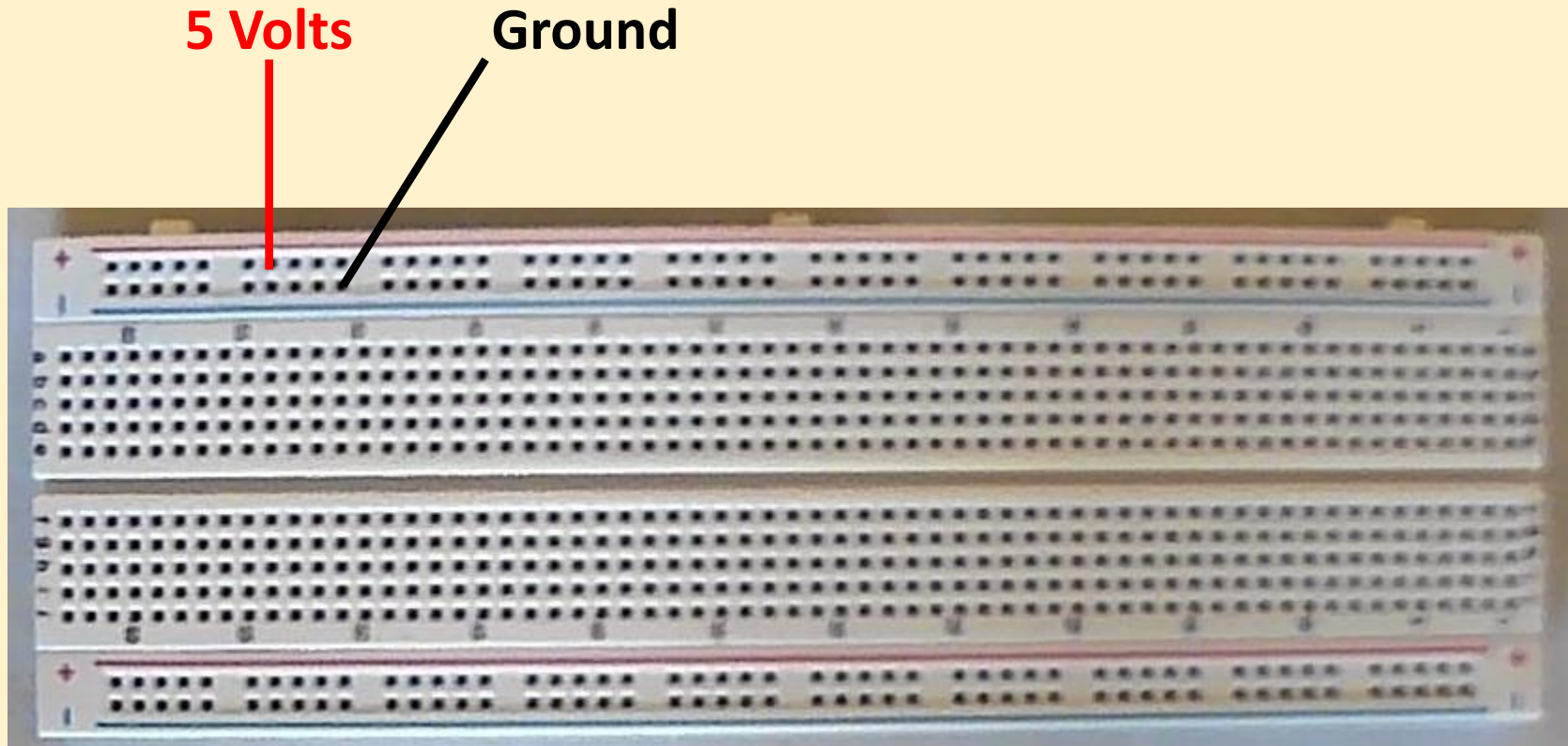
We attach 5 volts to the red line

We attach ground (0 volts) to the blue line

The yellow and greens show where groups of 5 holes are connected together

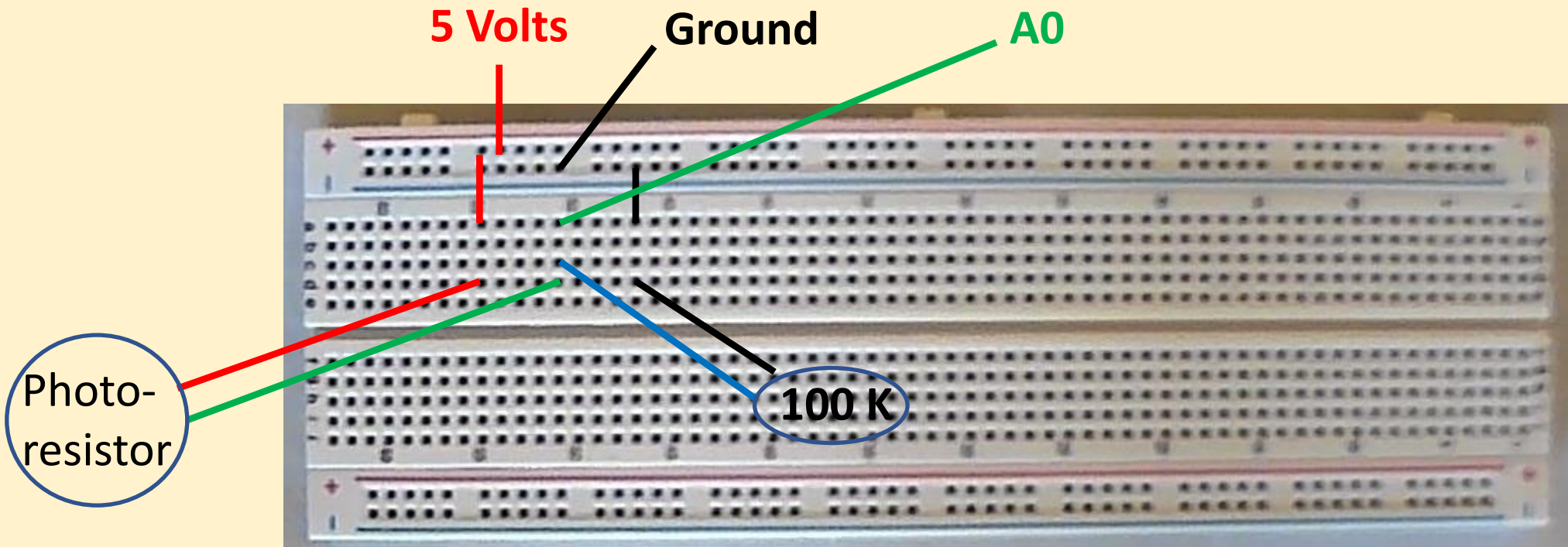
Steps

1. Connect a red M-M wire to the first (red) row on the breadboard to the 5 volts on the Arduino board
2. Connect a black M-M wire to the second (blue) row on the breadboard to the ground on the Arduino board



Steps

1. Connect a red M-M wire from the 5-volt rail to one of the bread board (BB) columns
2. Connect a black M-M wire from the ground rail to a different BB column
3. Connect a green M-M wire to A0 on the Arduino and yet another BB column
4. Connect the photoresistor to the red and green columns
5. Connect the 100K resistor to the green and black columns



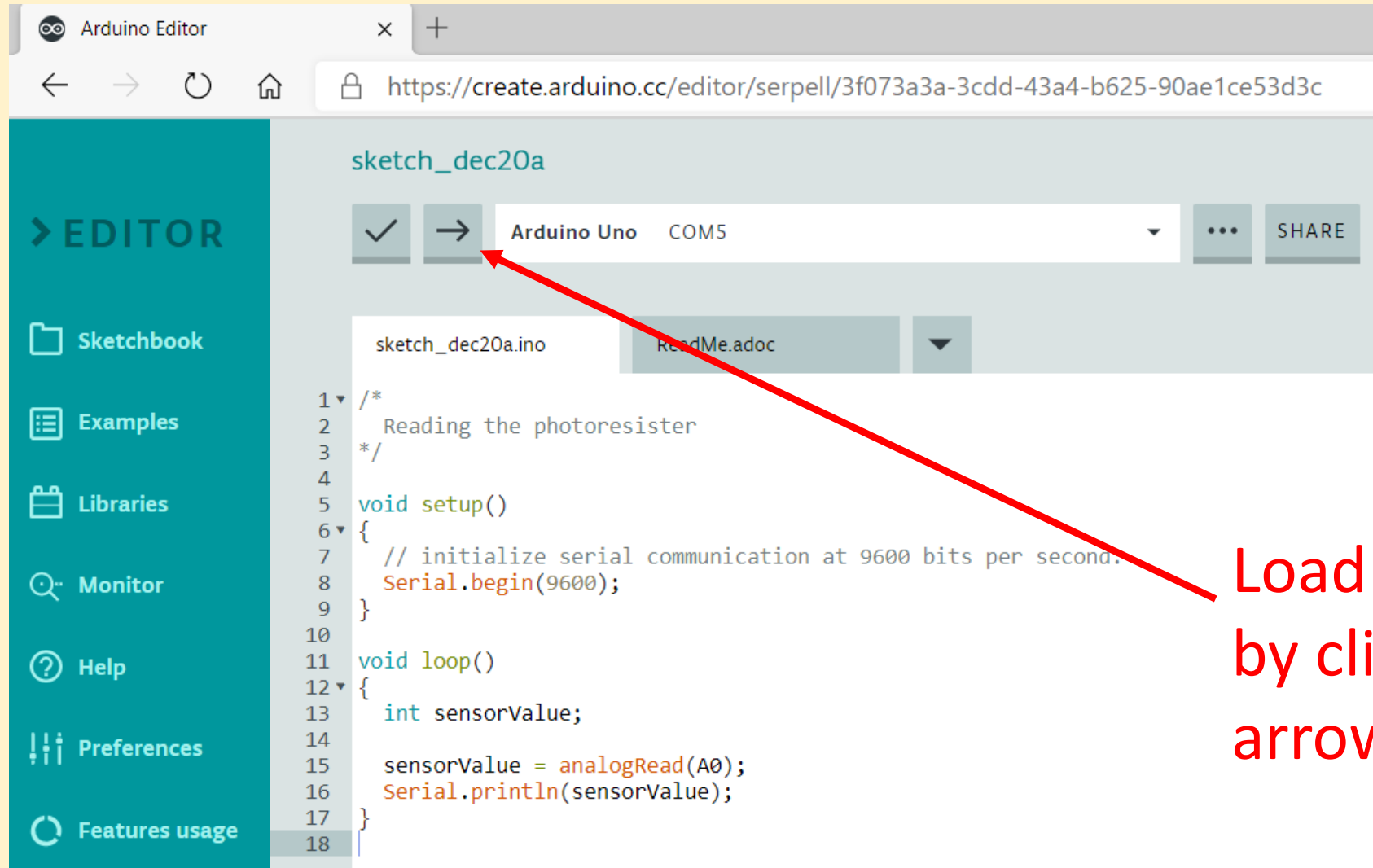
Arduino project setup

initial steps and Reading the photoresistor

1. Connect your Arduino to your Laptop/PC
2. Open the Arduino IDE in your browser
3. Create a new program (Sketch in Arduino talk)
4. Enter the code on the next slide...

```
/*  
  Reading the photoresistor  
*/  
  
void setup()  
{  
  // initialize serial communication at 9600 bits per second:  
  Serial.begin(9600);  
}  
  
void loop()  
{  
  int sensorValue;  
  
  sensorValue = analogRead(A0);  
  Serial.println(sensorValue);  
}
```

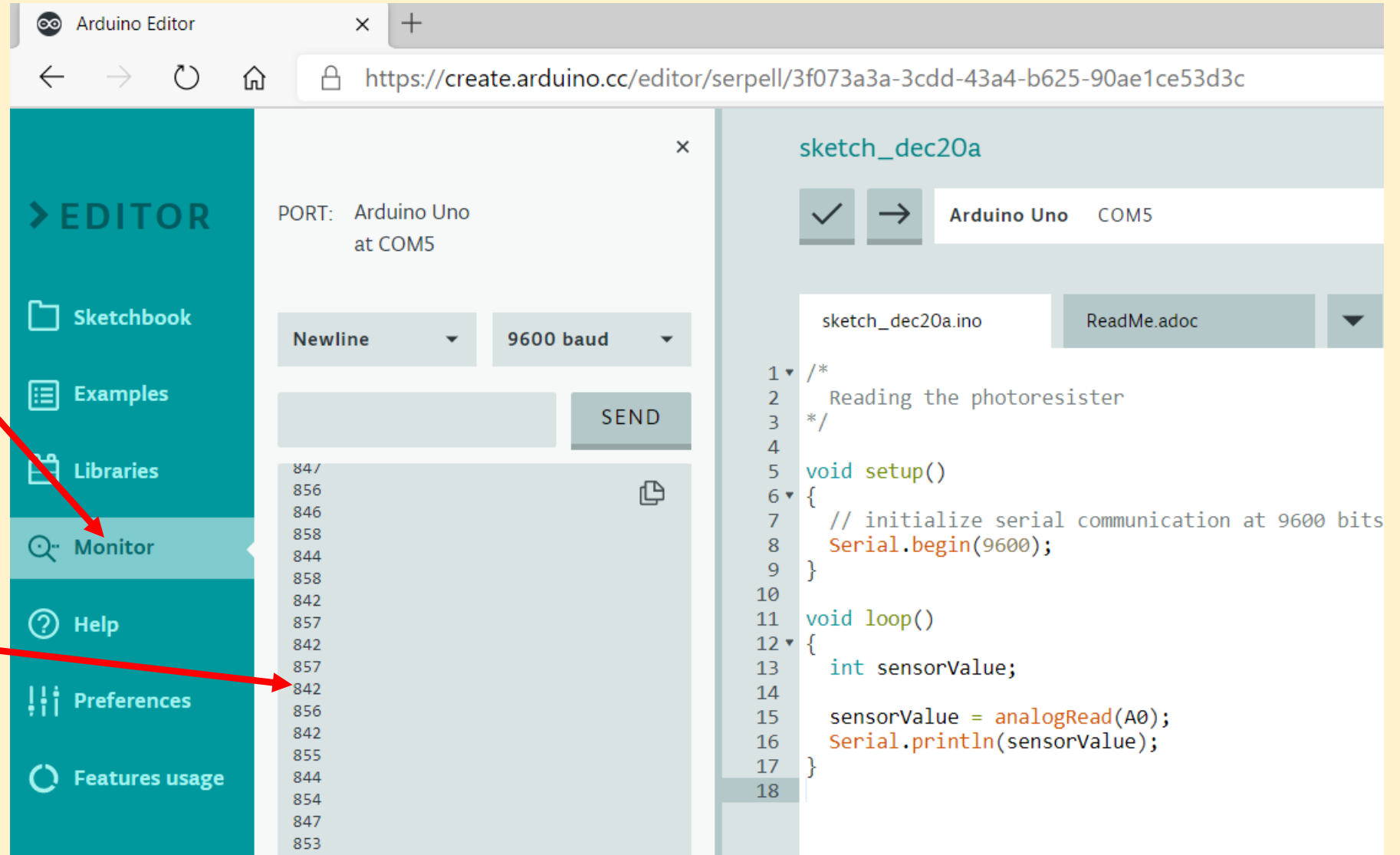
It should look like this...



Load and run
by clicking this
arrow

1) Click on monitor

2) Watch the photoresistor readings appear in real-time



If you put your finger over the top of the photoresistor
you should see the values change in the monitor

For me (in my dining room)

- Uncovered 800s
- Covered 600s

For you it might be different

Exercise 1

Record the range of photoresistor values on your system when...

- Uncovered
- Covered

Remember in the Arduino we actually have...

```
void main()
{
    setup(); /* hardware and software */

    while (true) /* Background loop runs forever */
    {
        loop (); /* Do stuff */
    }
}
```

Well, we also have hidden code handling the monitor for us...

```
void main()
{
    setup(); /* hardware and software */

    while (true) /* Background loop runs forever */
    {
        loop (); /* Do stuff */
        handle_monitor();
    }
}
```

Providing the monitor facility for us is very helpful but it comes at a price...

...it takes time to execute!

And this lost time is hidden from us, but we need to keep it in the back of our minds as it means that chunks of time disappear from our processing without us being able to easily see them

Transmitting

Let's look at the transmitting end



220



Two legged and not
four legged

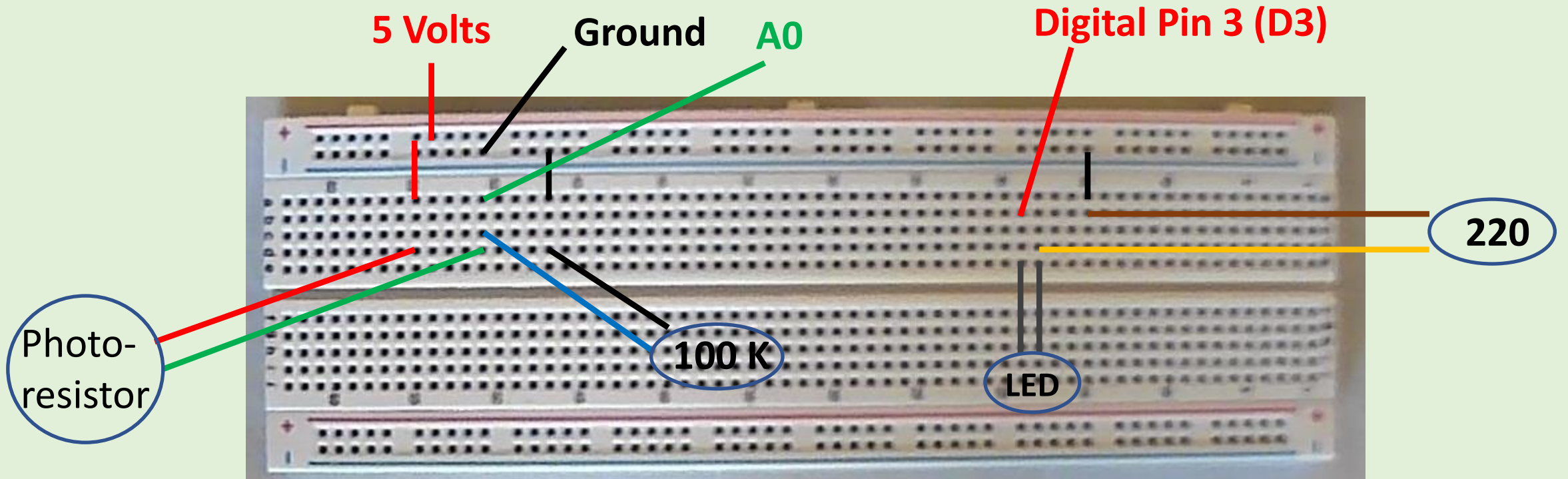
The white LED has sturdy legs, and they can be placed directly into the breadboard...

However, the resistor can't and so please take the brown and yellow M-F wires and connect them to each end of the 220-ohm resistor

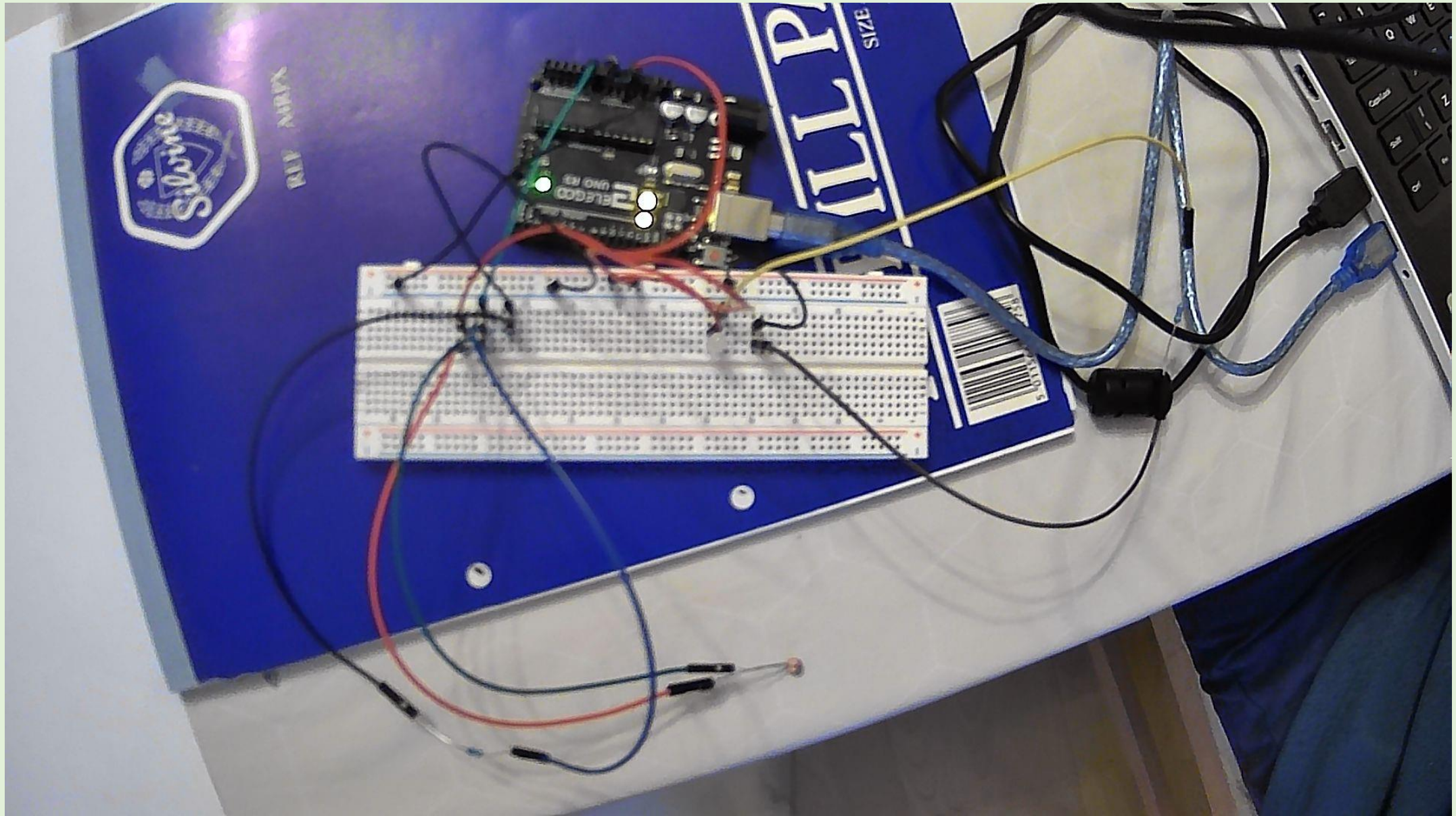


Steps (don't worry if you make the odd mistake... nothing should break)

1. Connect a black M-M wire from the ground rail to a new different BB column
2. Connect a red M-M wire to D3 on the Arduino and a different BB column
3. Place the White LED in the red and adjacent columns
4. Connect the 220 resistor to the black and adjacent columns



It should look like this...



Time for a new Program - Flash the LED

1. In the Arduino IDE create a new program (Sketch)
2. Add the code on the two next slide...

```
/*  
  Flash the LED  
*/
```

```
void setup()  
{  
  // initialize serial communication, for the monitor, at 9600 bits per second:  
  Serial.begin(9600);  
}
```

```
const long txInterval = 1000;      // interval at which to tx bit (milliseconds)  
unsigned long previousTxMillis = 0; // will store last time LED was updated  
int ledState = LOW;                // ledState used to set the LED
```

```
void loop()
{
    unsigned long currentTxMillis = millis();

    if (currentTxMillis - previousTxMillis >= txInterval)
    {
        // save the last time you blinked the LED
        previousTxMillis = currentTxMillis;

        if (ledState == LOW)
        {
            ledState = HIGH;
        }
        else
        {
            ledState = LOW;
        }
        digitalWrite(3, ledState);
    }
}
```

It should look like this...

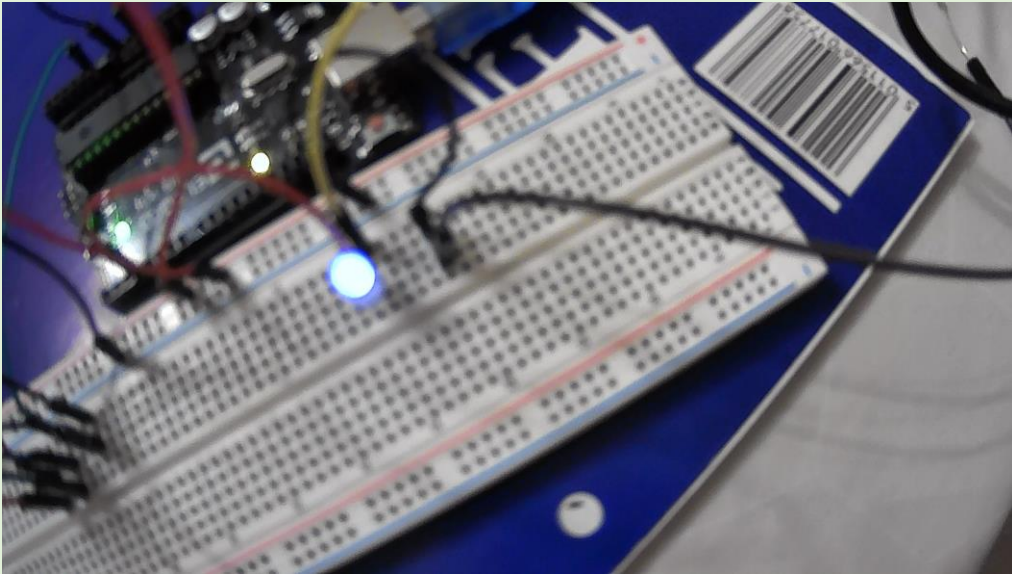


The screenshot shows the Arduino IDE interface. On the left, the 'Sketchbook' panel displays a list of files: 'sketch_dec20b' (selected), 'sketch_dec20a', 'ReadAnalogVoltage_copy', and 'sketch_dec16a'. The main editor area shows the code for 'sketch_dec20b.ino'. The code is a C++ sketch for flashing an LED. It includes a comment at the top: '/* Flash the LED */'. The 'setup()' function initializes serial communication at 9600 baud. The 'loop()' function checks if a certain interval (1000 ms) has passed since the last LED update. If so, it toggles the LED state between LOW and HIGH. The code is as follows:

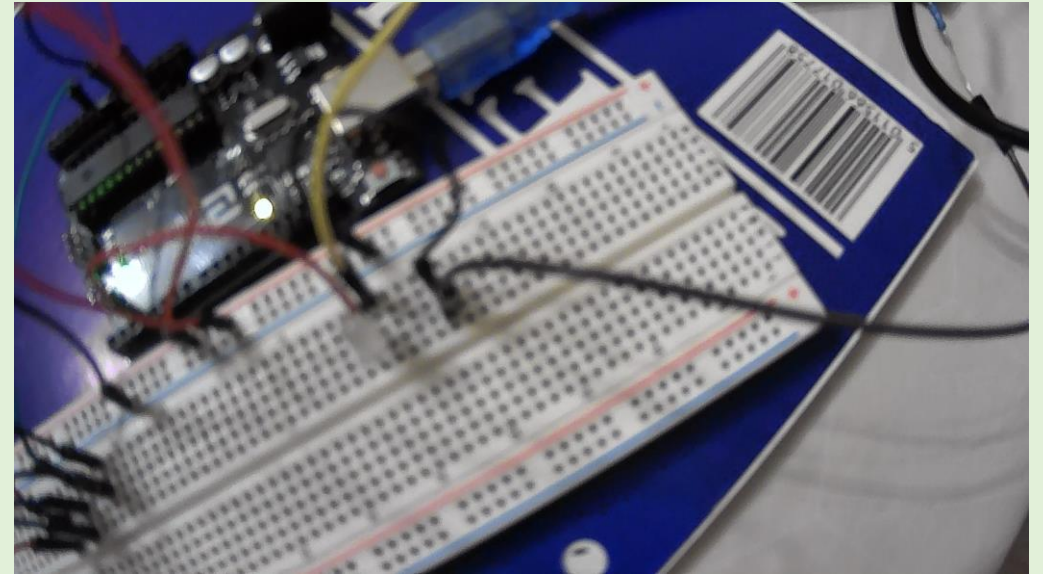
```
1  /*
2   Flash the LED
3  */
4
5  void setup()
6  {
7      // initialize serial communication, for the monitor, at 9600 bits per second:
8      Serial.begin(9600);
9  }
10
11
12  const long txInterval = 1000;           // interval at which to tx bit (milliseconds)
13  unsigned long previousTxMillis = 0;     // will store last time LED was updated
14  int ledState = LOW;                    // ledState used to set the LED
15
16  void loop()
17  {
18      unsigned long currentTxMillis = millis();
19
20      if (currentTxMillis - previousTxMillis >= txInterval)
21      {
22          // save the last time you blinked the LED
23          previousTxMillis = currentTxMillis;
24
25          if (ledState == LOW)
26          {
27              ledState = HIGH;
28          }
29          else
30          {
```

...load and run by clicking the arrow symbol

The white LED should flash; one second on and one second off



ON



OFF

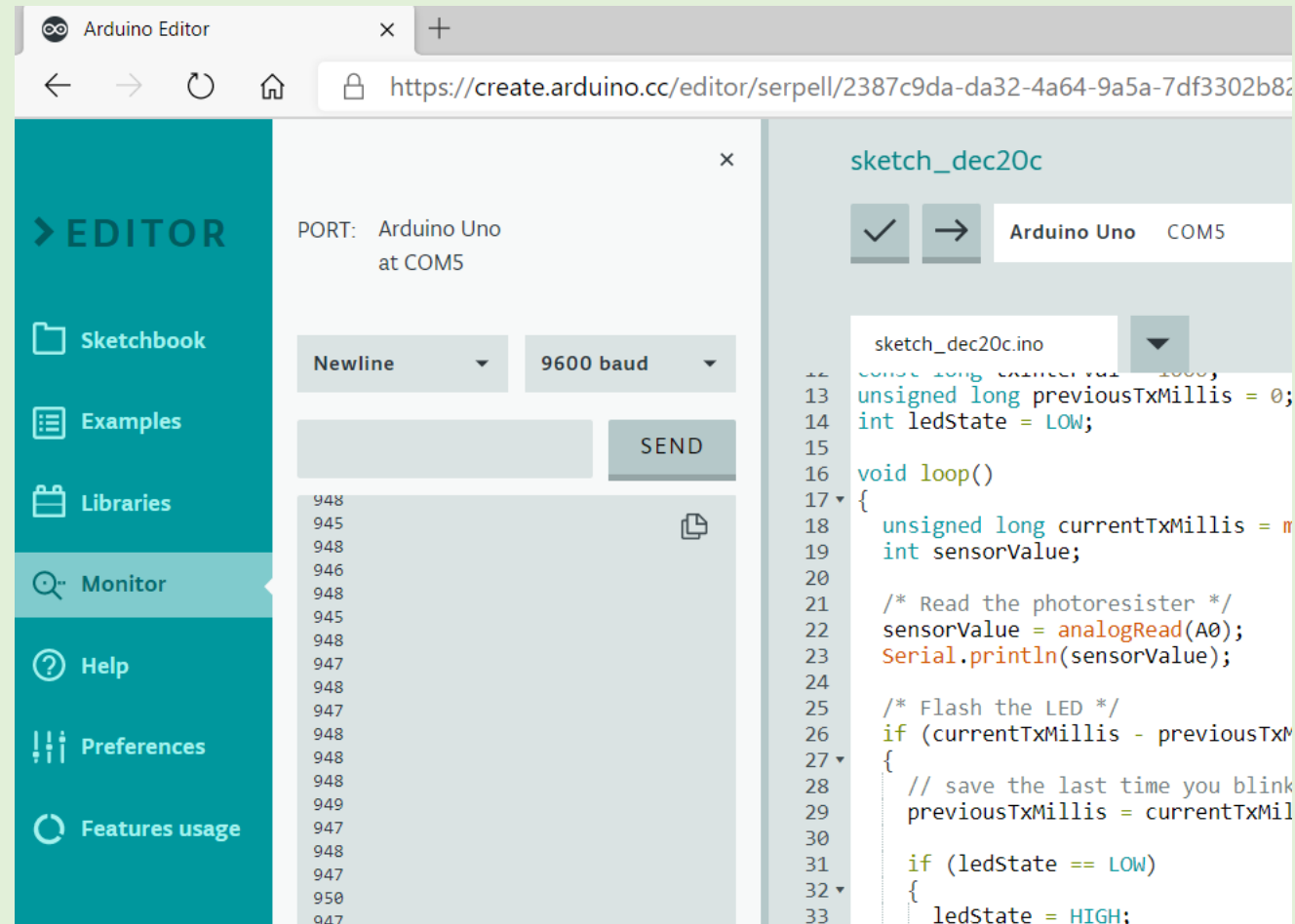
Let's have some fun and combine the two programs

1. Create a new program (sketch) in the IDE
2. And add the code in 'combined.txt' on Blackboard

It should look like...

```
sketch_dec20c.ino
12 const long txInterval = 1000; // interval at which to blink (milliseconds)
13 unsigned long previousTxMillis = 0; // will store last time LED was updated
14 int ledState = LOW; // ledState used to set the LED
15
16 void loop()
17 {
18     unsigned long currentTxMillis = millis();
19     int sensorValue;
20
21     /* Read the photoresistor */
22     sensorValue = analogRead(A0);
23     Serial.println(sensorValue);
24
25     /* Flash the LED */
26     if (currentTxMillis - previousTxMillis >= txInterval)
27     {
28         // save the last time you blinked the LED
29         previousTxMillis = currentTxMillis;
30
31         if (ledState == LOW)
32         {
33             ledState = HIGH;
34         }
35         else
36         {
37             ledState = LOW;
38         }
39         digitalWrite(3, ledState);
40     }
41 }
```


Place the photoresistor up close to the LED and use the Monitor to read the value in real-time



For me, the readings are...

- 800s Off
- 900s On (though yesterday it was >1000)

The light level in your room has an affect

Your readings might be a little different

Exercise 2

Record the range of photoresistor values on your system when...

- LED Off
- LED On

We can now communicate

Currently we can't communicate much but let's see how far we can take this...

Characters are represented as numbers in computers, in fact there is something called ASCII which provides a representation of our common letters and symbols

ASCII Table

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	Start of Header	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	Start of Text	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	End of Text	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	End of Transmission	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	Enquiry	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	Acknowledgment	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	Bell	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	Backspace	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	Horizontal Tab	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	Line feed	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	Vertical Tab	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	Form feed	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	Carriage return	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	Shift Out	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	Shift In	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	Data Link Escape	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	Device Control 1	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	Device Control 2	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	Device Control 3	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	Device Control 4	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	Negative Ack.	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	Synchronous idle	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	End of Trans. Block	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	Cancel	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	End of Medium	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	Substitute	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	Escape	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	File Separator	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	Group Separator	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	Record Separator	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	Unit Separator	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		Del

So, how would we go about transmitting a character, say the character 'H'?

Well, 'H' is 0x48 in hexadecimal and 01001000 in binary

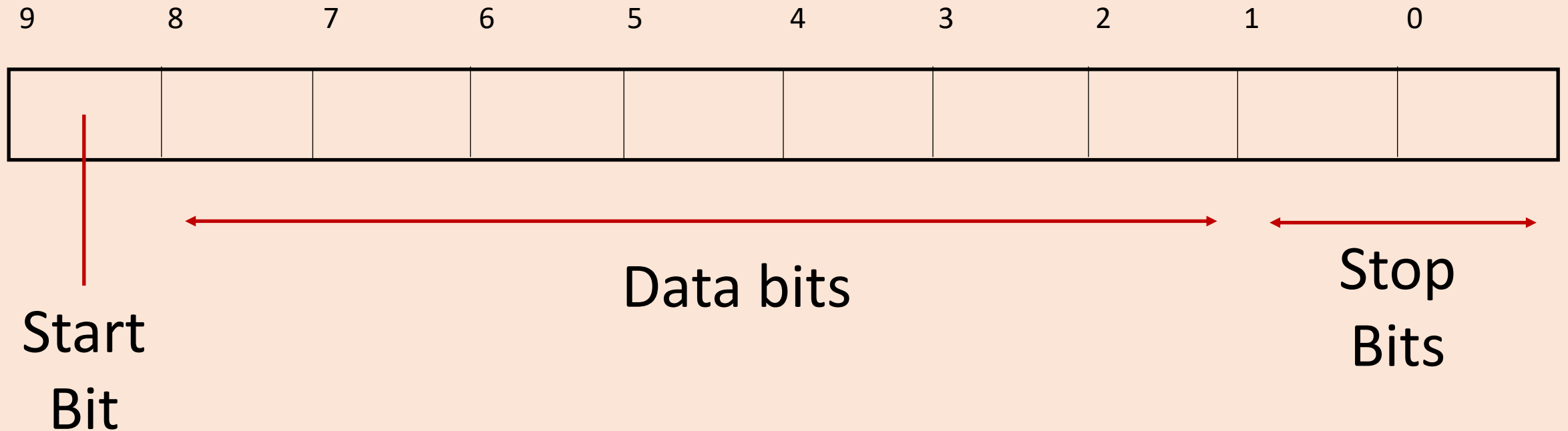
We can transmit the 0s and 1s of binary by turning the white LED off and on.

We can flash the LED to indicate 0s and 1s but how do we indicate the start of a character?

The NULL character is all zeros so we would never see it as the LED would never be turned on

We get around this by adding start and stop bits

So, our character looks like this...



...we have one start bit, seven data bits and two stop bits

The bits...

- The start bit is set to 1 and indicates start of character
- The seven data bits are set to 0s and 1s representing the binary ASCII value of the character
- The two stop bits are set to 1 and 0 to indicate the end of the character

This means that we need to send 10 bits to transmit a 7-bit ASCII character (ASCII only uses 7 bits)

Transmitting a character is easy enough... we just flash the
LED

Receiving the character is a little harder... we need to
monitor the photoresistor more frequently than the LED
flashes

Typically, we need to read at twice the rate of flashing
(Nyquist period) but to be safe we are going to do it ten
times faster

Remember earlier we saw that the monitor on the Arduino uses up a lot of the Arduino's CPU time... well this is where we are going to have to take this lost time into account

We aren't going to be able to transmit and receive at a high data rate... we are going to have to start slow

Time for a new program- Communication

1. In your IDE, make sure that the monitor is turned off as it interferes with the loading of new programs
2. Create a new program (sketch) on the Arduino
3. Put the **Communications v1** code that you will find on Blackboard into the new program

The program should start like this...

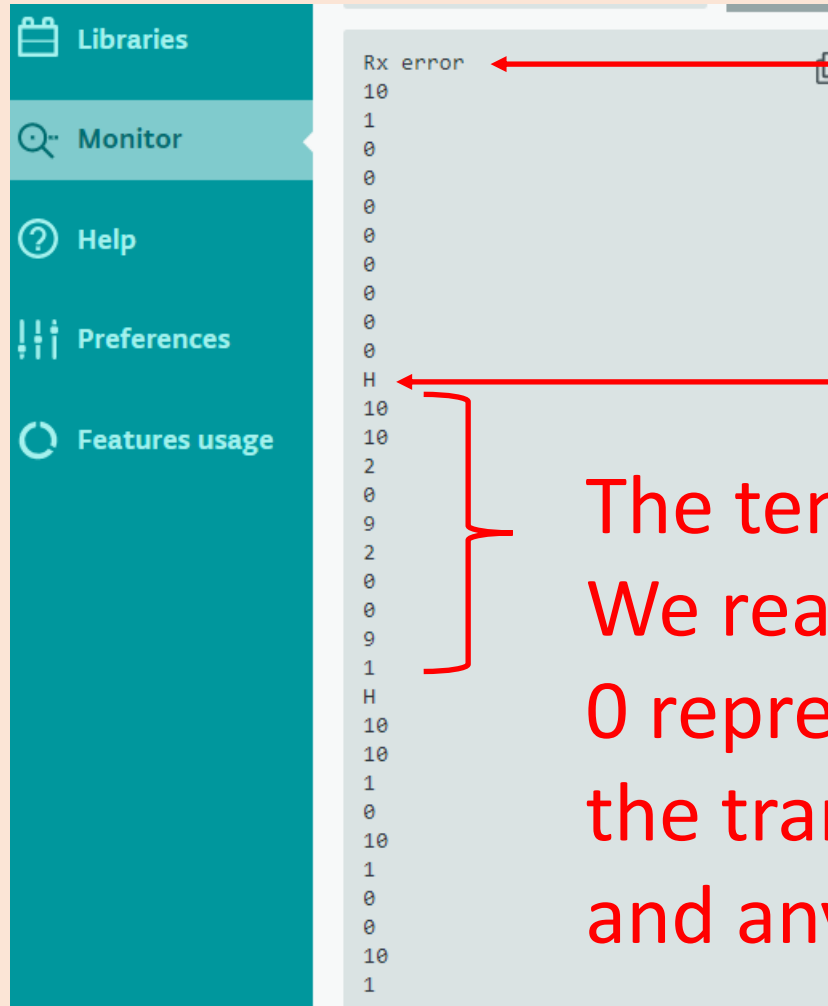
```
sketch_dec20d.ino
1  /*
2   Communications
3
4   Transmits data using a white LED and recieves it using a photoresistor
5
6   */
7
8
9  int ledState = LOW;           // ledState used to set the LED
10
11
12 // the setup routine runs once when you press reset:
13 void setup()
14 {
15     // set the digital pin as output:
16     pinMode(3, OUTPUT);
17     // initialize serial communication at 9600 bits per second:
18     Serial.begin(9600);
19 }
20
21
22 const long txInterval = 200;   // interval at which to tx bit (milliseconds)
23 int tx_state = 0;
24 char chr = 'H';
25 unsigned long previousTxMillis = 0;    // will store last time LED was updated
26
27 void txChar()
28 {
29     unsigned long currentTxMillis = millis();
30
```

This program sends and receives data at 5 bits per second (bps)

Not fast by Broadband standards where you would expect at least 5 million bits per second (bps)

But it's a start and we all have to start somewhere

Run the program and turn the monitor on, when you place the photoresistor next to the LED you should get a reading...



Placed the photoresistor next to the LED half-way through sending the character

H

The ten transmitted bits – 1 start, 7 data, 2 stop
We read ten times per bit so 10 represents 1 and 0 represents 0... but we also see some noise on the transmission and so anything above 3 is a 1 and anything below is a 0

Exercise 3

Modify the line...

```
char chr = 'H';
```

So that it sends a different character

Noise in Communications

Noise is expected during communications but there are often things we can do to mitigate the problem

Here we are already sampling the signal ten times faster than it is sent and using a count to determine if we have received a 1 or a 0

There are other things that we can do...

Let's think about how we are reading the character in

Look at the function **rxChar()**

It contains the code...

```
unsigned long currentRxMillis = millis();  
int sensorValue;  
int i;  
  
if (currentRxMillis - previousRxMillis >= rxInterval)  
{  
    // save the last time you read the analogue input  
    previousRxMillis = currentRxMillis;
```

The function **millis()** returns the number of milliseconds this program has been running for

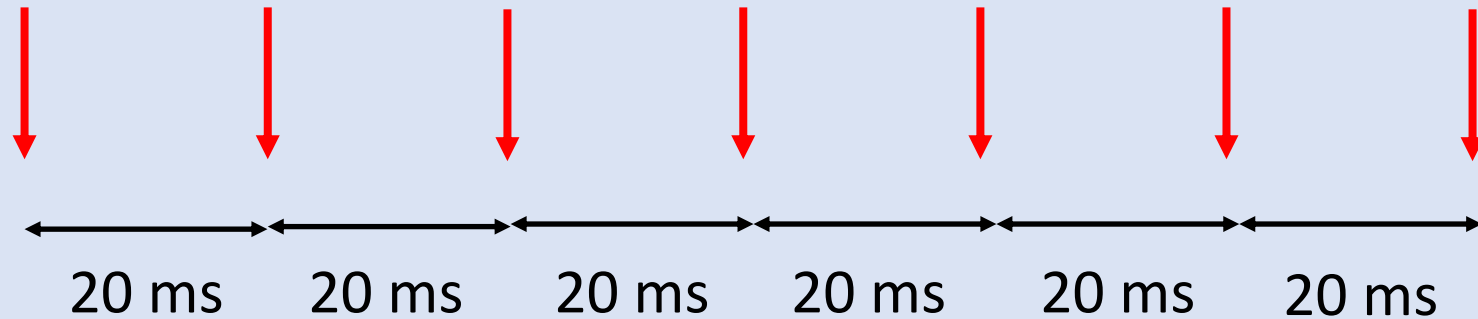
The constant **rxInterval** is set to 20 milliseconds

This means that anything inside this **if** statement should run every 20 milliseconds...

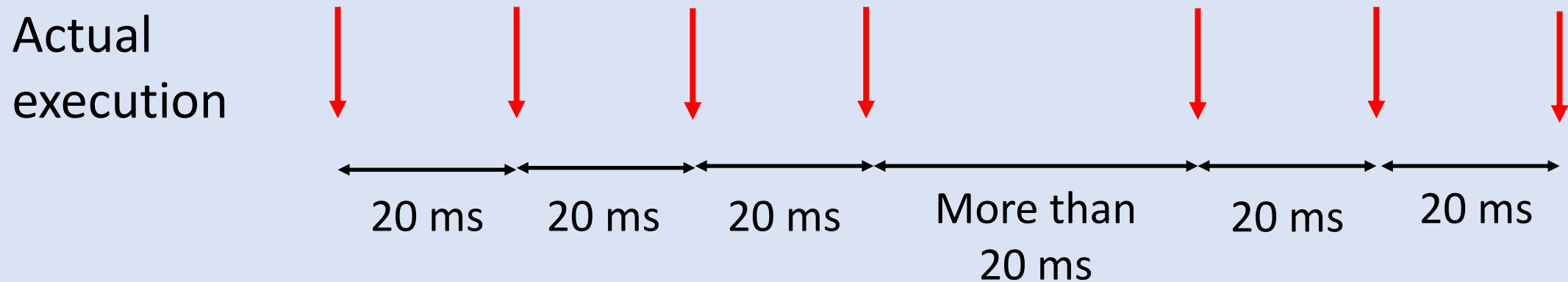
```
if (currentRxMillis - previousRxMillis >= rxInterval)
{
}
```

And when it does run the variable **previousRxMillis** gets updated to ensure that the code inside the **if** statement will not run until another 20 milliseconds elapse

Expected
execution



But we have already seen that the nice people at Arduino have added extra code like the monitor into the system that we can see... so, what problems could this cause?



And so **rxChar()** may not be running when we expect!

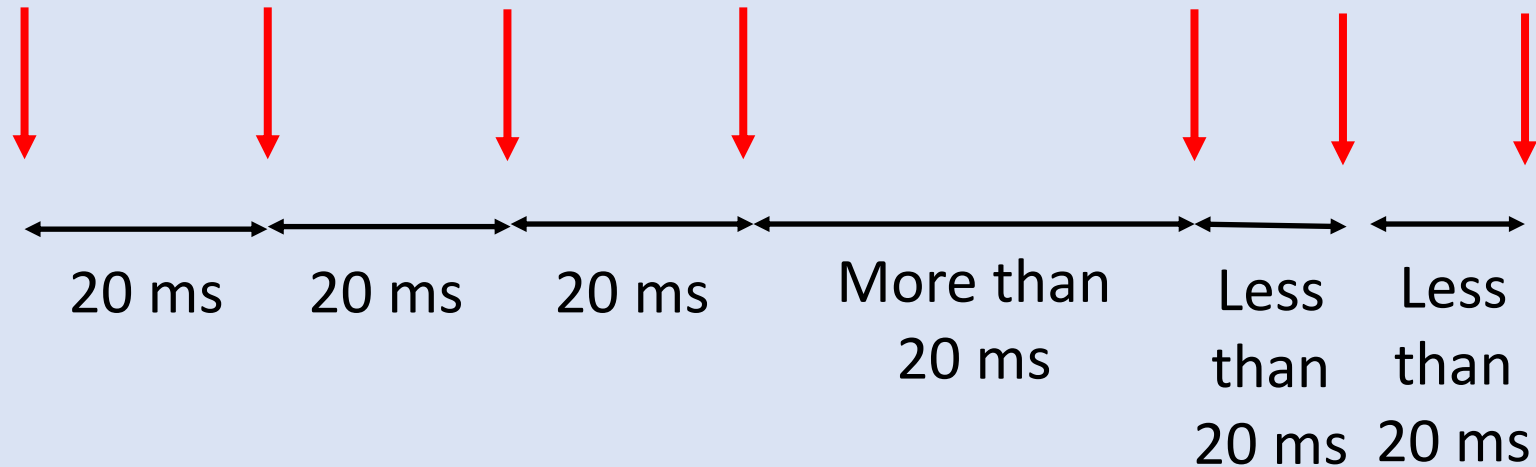
A fix for this (though not perfect) is to change...

```
previousRxMillis = currentRxMillis;
```

To...

```
previousRxMillis = previousRxMillis + rxInterval;
```

Improved
execution



Eventually the intervals will recover and go back to 20 ms

Exercise 4

The monitor output contains the 'H' (or other character) followed by the ten bits represented by the numbers 0 to 10

Where the numbers are not 0 or 10 an error has occurred

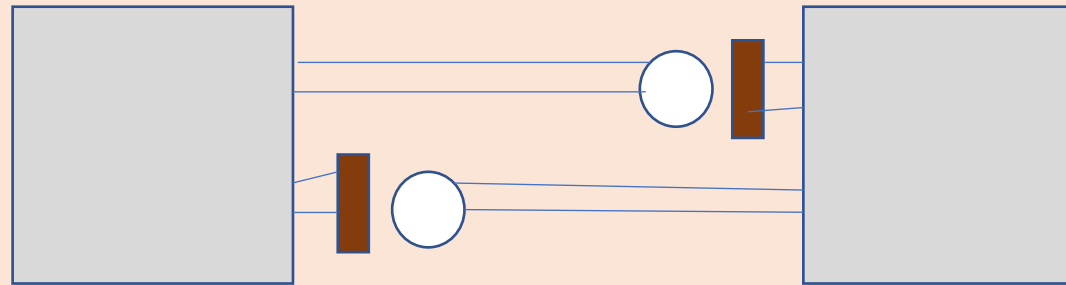
Record the number of errors

- Before the code change
- After the code change

Exercise 5

Only do this exercise if we no longer have social distancing

Place two Arduino boards next to each other with the LED of one next to the Photoresistor of the other



You should see characters transmitted between Arduinos

Improving the Code

Let's examine how the code works

```
void loop()  
{  
    txChar();  
    rxChar();  
}
```

We permanently sit in a loop running these two functions

These two functions must be written to be **non-blocking**

This means that they must not hog CPU time

txChar() could have been written...

```
output(start bit);  
delay (200);  
output (bit 7);  
delay (200);  
etc...
```

But if we had done this then we would have finished transmitting the character before we started to read it... and this would not work!

Hence, we need **non-blocking** code

It is this that makes **txChar()** non-blocking...

```
unsigned long currentTxMillis = millis();

if (currentTxMillis - previousTxMillis >= txInterval)
{
    // save the last time you blinked the LED (improved version)
    previousTxMillis = currentTxMillis;
}
```

The function runs many times but only does anything every 200 ms (when it executes what's inside the **if**).

Exercise 6

Replace the cases 1 to 7 with a single case whose code that uses a shift and mask, e.g.

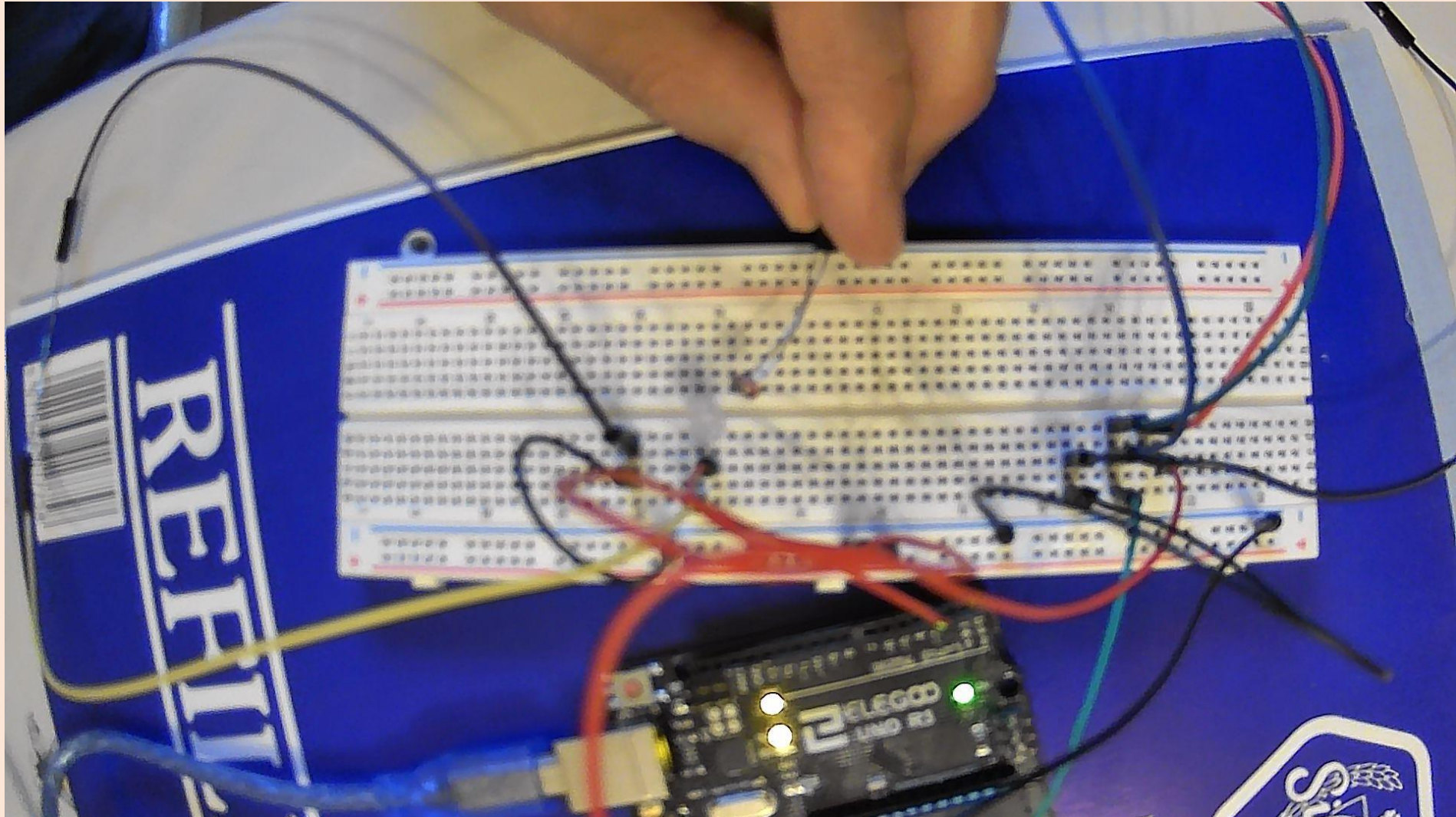
```
case 1:
    if ((chr & 0x40) != 0) /* Mask Transmit Bit */
    {
        digitalWrite(3, HIGH);
    }
    else
    {
        digitalWrite(3, LOW);
    }
    chr = chr << 1; /* shift chr left by one bit */
    tx_state++;
    break;
```

Let's see how you did

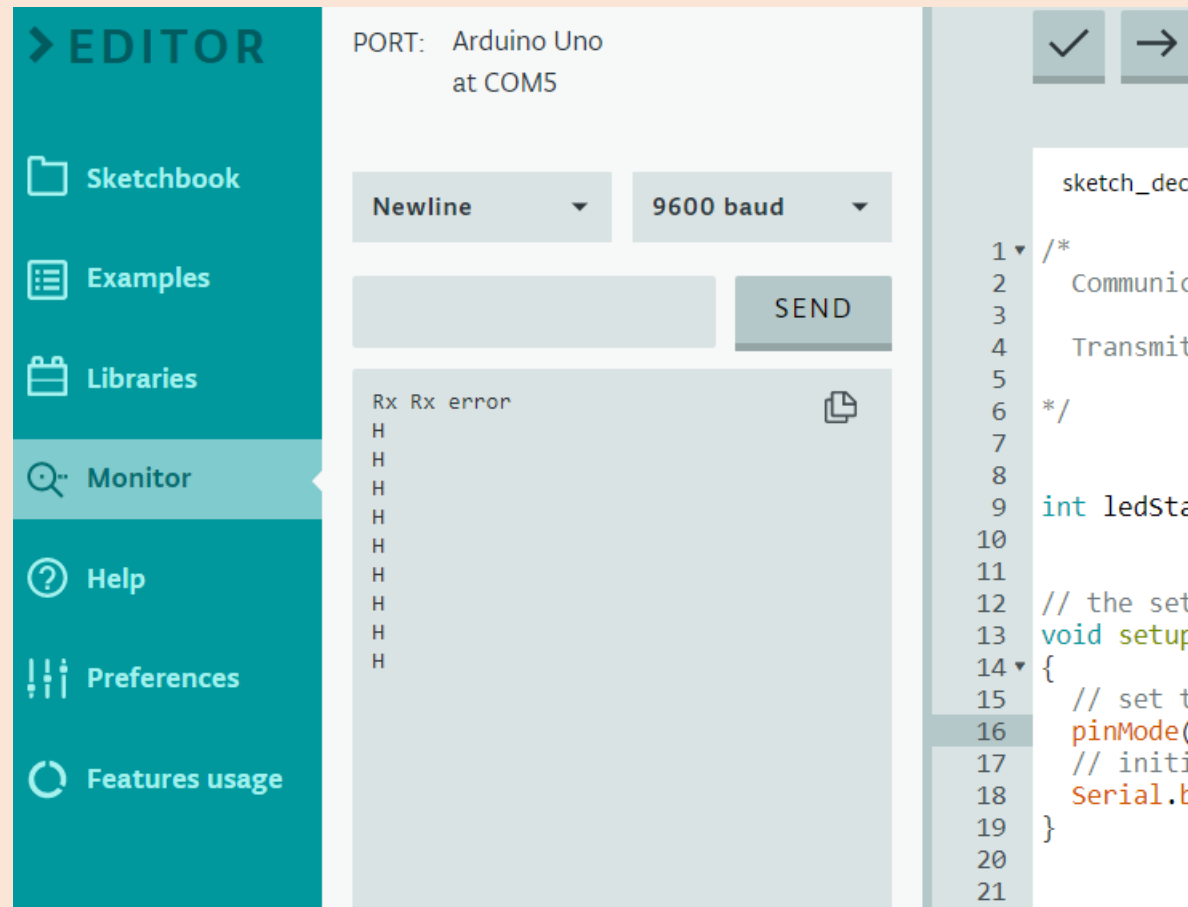
Create a new project (sketch) and put the
Communications v2 code in it.

Load and run (**don't forget to turn the monitor off first**)...

Remember to hold the photoresistor very close to the LED



You should get monitor output like this...



Exercise 7

Please look at the changes made to the code in the
txChar() and rxChar() functions

Can you see how the txChar() function now uses shift to
output each data bit in turn?

Communication Speed

We are sending data at 5 bits per second (it takes us two seconds to send a character)

Data rates are also measured in **baud**

Baud rate measures the rate at which useful data is transmitted

Remember we had to add a start bit and two stop bits to the seven data bits that we were sending? Well, the baud rate only counts the seven data bits

Therefore, we are sending 7 bits every 2 seconds and therefore we have a baud rate of 3.5

Do you remember the **setup()** function?

```
void setup()  
{  
  // set the digital pin as output:  
  pinMode(3, OUTPUT);  
  // initialize serial communication at 9600 bits per second:  
  Serial.begin(9600);  
}
```

Well, the line...

```
Serial.begin(9600);
```

Sets up the communications with the monitor on your PC/Laptop and it sets the **baud rate** to 9600

It is sending data as 8-bits (we sent it as 7-bits) so the monitor can receive 1200 characters per second

The monitor communication is serial like ours, but it is an electrical signal that goes over wire instead of using light

This type of serial communications uses a clock signal to make it more efficient... but in this case that is being simulated as the signal is actually going over a USB connection which sends data over packets.

Sending a Text Message

Sending a single character isn't that interesting so let's send a whole string of characters.

Remember that we had to have start and stop bits when sending a single character to indicate where it started and finished?

Well, we need start and stop characters to indicate when our message starts and finishes

Characters are represented by numbers and we are using the ASCII standard in this exercise and it just so happens that ASCII defines extra non-printable characters for this purpose...

ASCII Table

ASCII represents
characters as numbers

Some of these numbers
represent special non-
printable characters

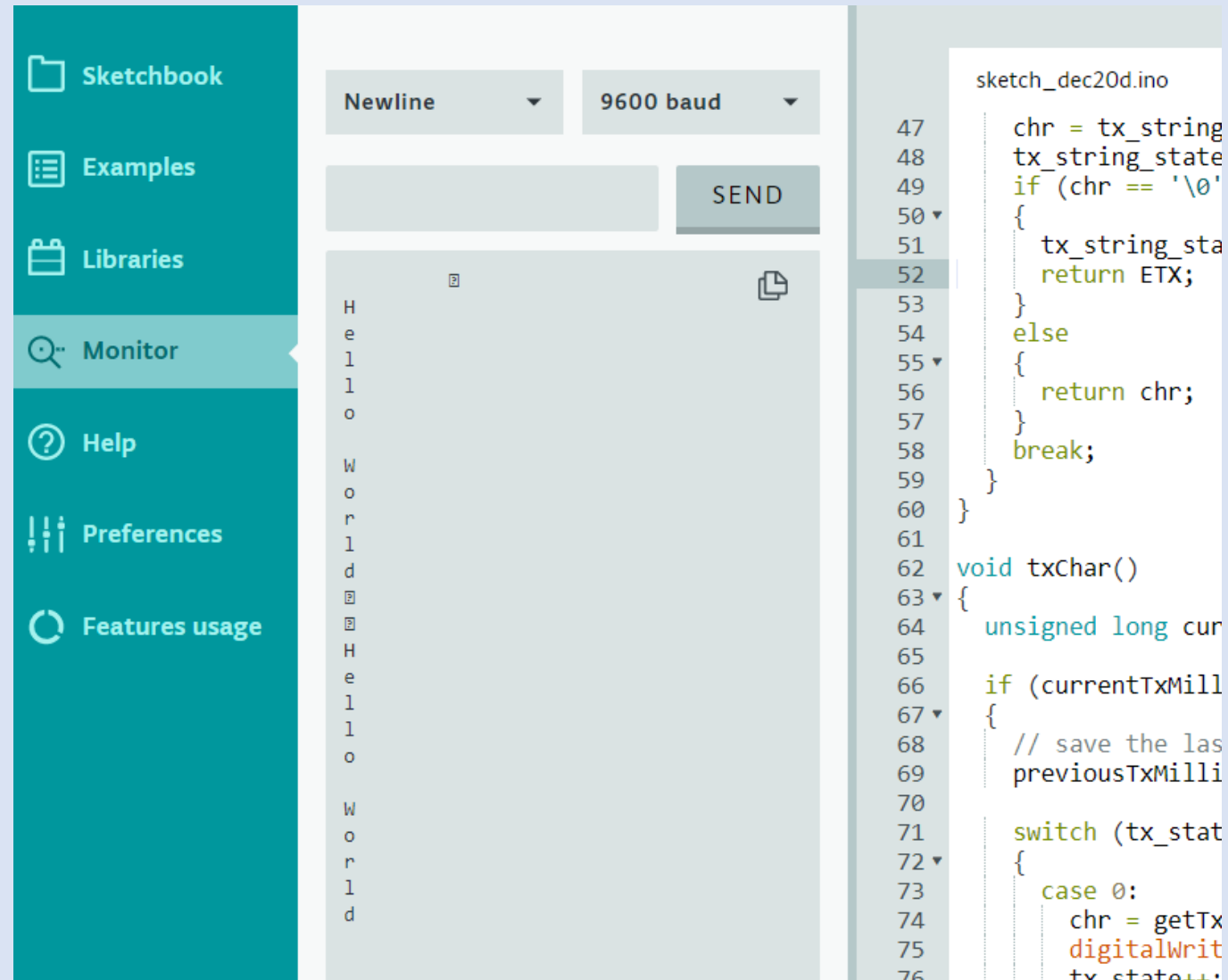
We need to use **STX** and
ETX to indicate the start
and end of our character
string

Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT End of transmission	0x24	36	\$	0x44	68	D	0x64	100	d
0x05	5	ENQ Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS Backspace	0x28	40	(0x48	72	H	0x68	104	h
0x09	9	TAB Horizontal tab	0x29	41)	0x49	73	I	0x69	105	i
0x0A	10	LF New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1 Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2 Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3 Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4 Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC Escape	0x3B	59	;	0x5B	91	[0x7B	123	{
0x1C	28	FS File separator	0x3C	60	<	0x5C	92	\	0x7C	124	
0x1D	29	GS Group separator	0x3D	61	=	0x5D	93]	0x7D	125	}
0x1E	30	RS Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Create a new project (sketch) and put the
Communications v3 code in it.

Load and run (**don't forget to turn the monitor off
first**)...

Holding the photoresistor close to the LED should get you this...



The screenshot displays the Arduino IDE interface. On the left, a teal sidebar contains navigation options: Sketchbook, Examples, Libraries, Monitor (selected), Help, Preferences, and Features usage. The central workspace is divided into three panes. The top pane shows 'Newline' and '9600 baud' settings, along with a 'SEND' button. The middle pane displays the output of the serial monitor, showing the text 'Hello World' on two lines. The right pane shows the source code for 'sketch_dec20d.ino'. The code includes a loop that checks for a null character and returns ETX, and a function 'txChar()' that handles character transmission and state management.

```
sketch_dec20d.ino
47   chr = tx_string
48   tx_string_state
49   if (chr == '\0'
50   {
51       tx_string_sta
52       return ETX;
53   }
54   else
55   {
56       return chr;
57   }
58   break;
59 }
60 }
61
62 void txChar()
63 {
64     unsigned long cur
65
66     if (currentTxMill
67     {
68         // save the las
69         previousTxMilli
70
71         switch (tx_stat
72         {
73             case 0:
74                 chr = getTx
75                 digitalWrit
76                 tx_state++
```

Exercise 8

Send a different message to “Hello World”

Sketchbook

Examples

Libraries

Monitor

Help

Preferences

Features usage

Newline 9600 baud

SEND

⏏

⏏

H
e
l
l
o

W
o
r
l
d

H
e
l
l
o

W
o
r
l
d

sketch_dec20d.ino

47 chr = tx_string
48 tx_string_state
49 if (chr == '\0'
50 {
51 tx_string_sta
52 return ETX;
53 }
54 else
55 {
56 return chr;
57 }
58 break;
59 }
60 }
61
62 void txChar()
63 {
64 unsigned long cur
65
66 if (currentTxMill
67 {
68 // save the las
69 previousTxMilli
70
71 switch (tx_stat
72 {
73 case 0:
74 chr = getTx
75 digitalwrit
76 tx_state++

These are the non-printable characters STX and ETX

Exercise 9

Please update rxChar() so that it prints a space when it receives a non-printable character

Secure Communications

As we know communications can be intercepted

To prevent a third party reading our messages we need to
add encryption

Create a new project (sketch) and put the
Communications v4 code in it.

Load and run (**don't forget to turn the monitor off
first**)...

You will find two new routines, added for secure communications

```
char encrypt(char in_char)
{
    char out_char;

    out_char = in_char;

    return out_char;
}
```

```
char decrypt(char in_char)
{
    char out_char;

    out_char = in_char;

    return out_char;
}
```

Exercise 10

These new functions are called in txChar() and rxChar(),
find them and think about what they are doing

Exercise 11

Update encrypt() and decrypt() to implement substitution encryption using the following substitution table

A -> L	I -> M	Q -> ,	Y -> O
B -> K	J -> -	R -> A	Z -> T
C ->	K -> E	S -> ?	-> R
D -> Y	L -> D	T -> F	. -> U
E -> G	M -> P	U -> !	, -> .
F -> Z	N -> X	V -> H	- -> S
G -> I	O -> V	W -> B	? -> Q
H -> W	P -> N	X -> C	! -> J

Exercise 12

The previous substitution encryption did not work on the whole character set...

Update `encrypt()` and `decrypt()` to implement substitution encryption using your own substitution table that covers the whole ASCII character set

Exercise 13

Update encrypt() and decrypt() to implement XOR encryption using the character 'a' as the key

Then use other keys

If we are not in lockdown and it is safe send messages between different Arduinos

Having the message printed down the screen is not as readable as having it printed across the screen



Exercise 14

It's not convenient writing "Hello World" down the monitor page

Rewrite rxChar() to store the incoming message in a receive buffer and then print out the whole message only when it has been completely received

Hint: You need to use **STX** and **ETX**

The End