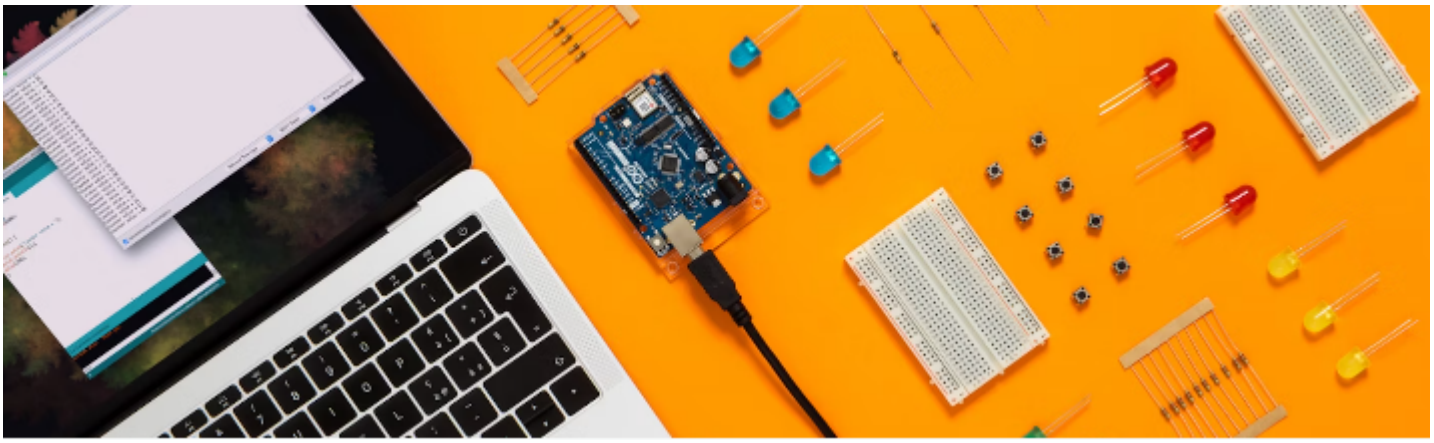
Lesson 6: Serial to Board



SERIAL TO BOARD

We are going to explore serial communications further by sending data in the opposite direction, that is, from the computer to the Arduino board. We will see how this data is used by the Arduino board to control the behavior of the components.

ESTIMATED TIME: **45 min**

YOU WILL LEARN ABOUT:

- ARRAY
- VARIABLE TYPES
- COUNT ALGORITHM
- REPRESENTING DATA IN DIFFERENT WAYS
- READING DATA FROM VARIOUS INPUTS
- DESIGNING PROGRAMS BASED ON SPECIFICATIONS

Recap of previous lessons

Question 1

Identify which commands would generate which of the following outputs in the Serial Monitor in a properly functioning program:

Question 2

Identify which symbols/operators would result in the following actions:

- assignment of value to a variable
- comparison of values/variables
- logical AND
- logical OR
- incrementing a variable by 1
- decrementing a variable by 1
- declaring a text as a string
- setting the follow-up text as a comment
- closing a command

ANSWERS AT THE END

Introduction

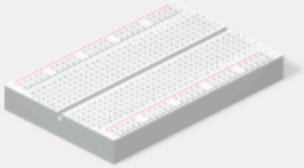
In this lesson, we are going to explore serial communications further by sending data in the opposite direction, that is, from the computer to the Arduino board. We will see how this data is used by the Arduino board to control the behavior of the components. We will also build a two-way communication channel that will allow us to send data back and forth between the board and the computer.

Today we will learn

- new programming concepts that are used to send information from the computer to the Arduino board,
- how information is stored and transmitted in digital devices,
- how to read and process the bytes of information received on the Arduino board, and
- how to create a two way communication channel.

Let’s prepare the materials

1 ARDUINO® UNO WIFI REV2

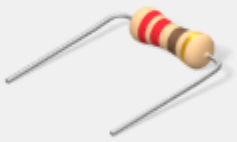


1 BREADBOARD

1 USB CABLE

1 5MM RED LED

1 5MM GREEN LED

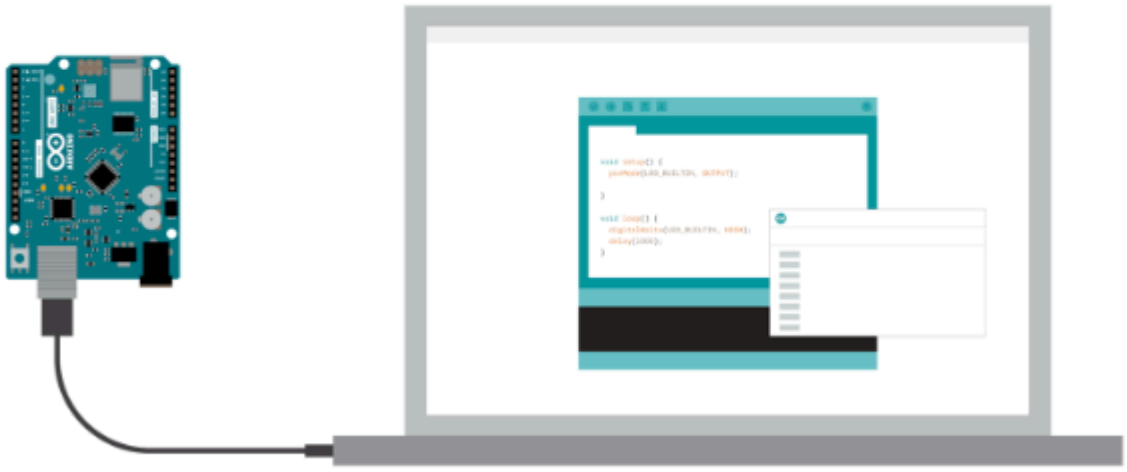


2 220Ω RESISTOR

10 MALE-MALE JUMPER WIRE

Different ways to display data

In this activity, we will learn how to store and represent the data we send from the computer to the board in different ways.



Sending data to the board

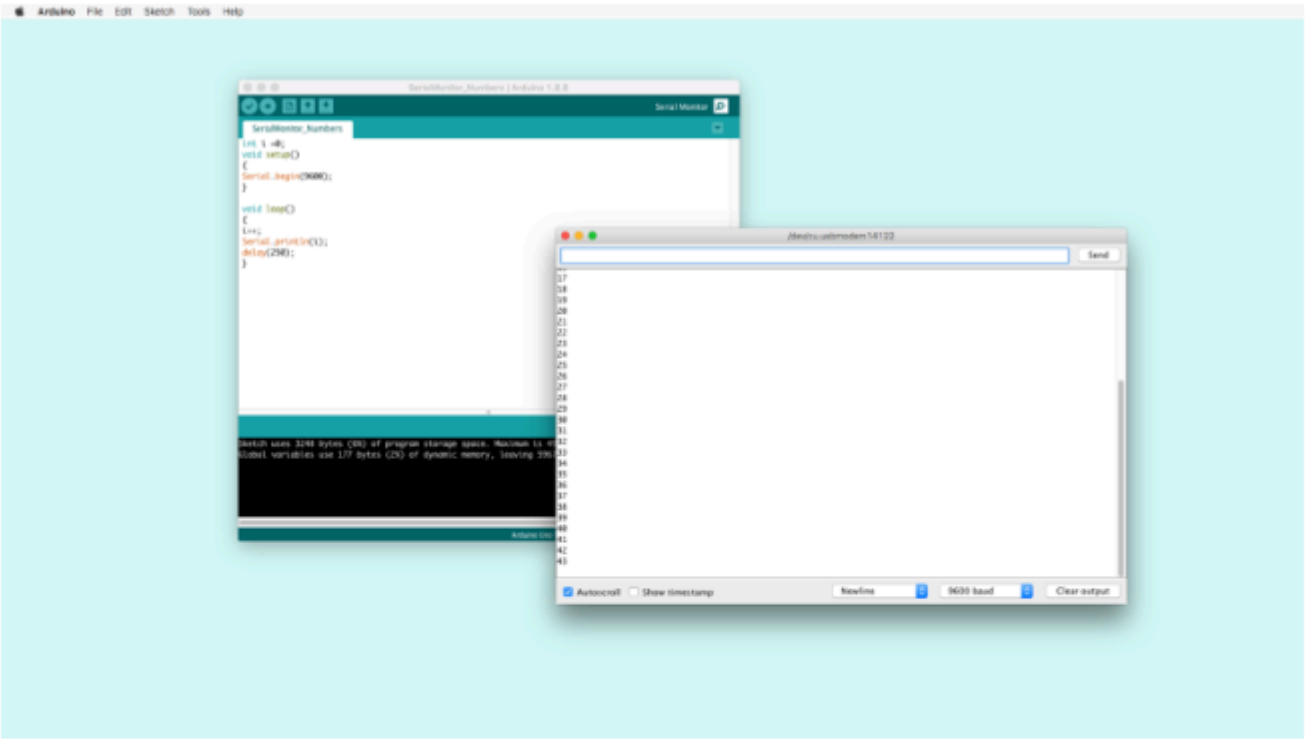
As we know from the previous lesson, when data flows through serial communication, it is flowing in binary code. There are different ways to work with the data that arrives through serial communication to the board. We can store it using an `int` variable - then we will see a number that codifies the character arriving at the board - or we can use a new type of variable called `char` - and then we will see what the character is that has arrived at the board.

Data	Type
Numbers (0, 1, 2, 3)	int
Characters (a, b, c, +, *, (,))	char

Programming the board

Let's program the board to test the differences between storing data on a `char` variable and on an `int` variable. To do this, open the code in **File > Examples > CTC GO CORE > Lessons > SerialToBoard > SerialToBoard_Activity1** and upload it to the board.

Now that we have programmed the board, we need to open the Serial Monitor and configure it as follows:



The **no line ending** will configure the Serial Monitor in order not to send extra characters when we press the ENTER key.

This program just shows in the Serial Monitor the data we sent to the board. It does this by showing the differences between storing the data as `int` or `char`.

Try out sending different characters (it is possible to try this using capital letters as well) and numbers through the Serial Monitor and see what the number is that codifies it.

Not working?

- Check that the correct board is selected.
- Check that the correct port is selected.
- Check that the number in the Serial Monitor matches the one written in the `Serial.begin()` function.
- Check that the Serial Monitor configuration matches with the configuration shown in the **Programming the board** section.

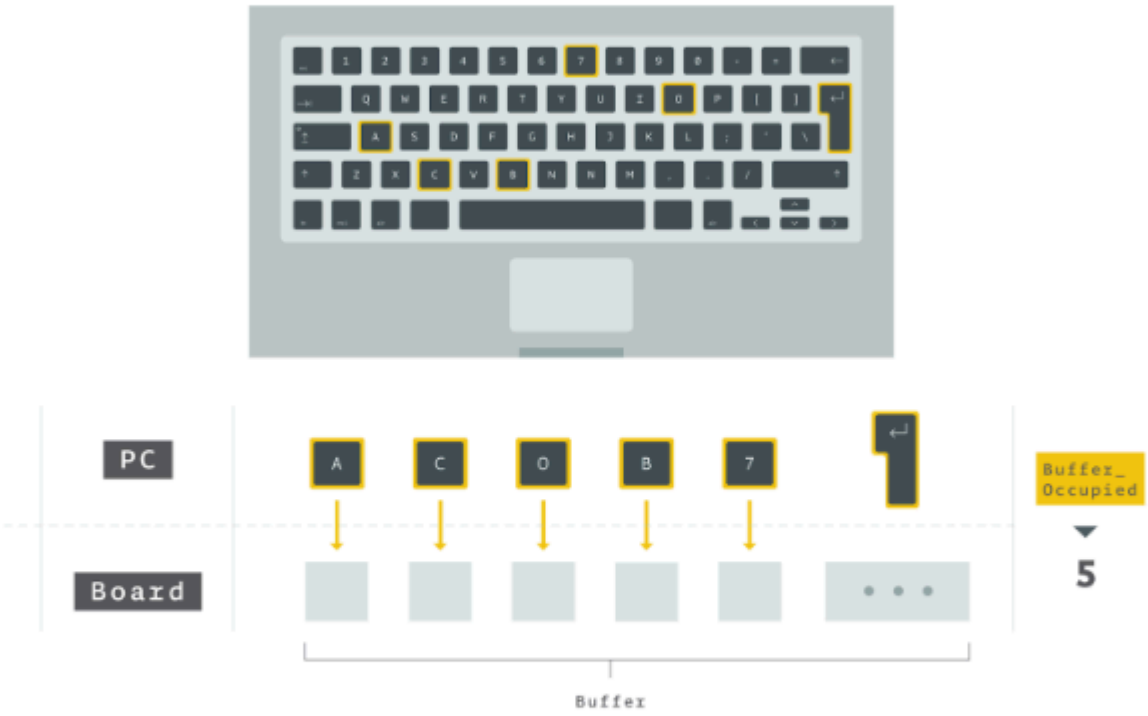
Understanding the program

In the first part of the code, we initialize the variables and serial communication by selecting the communication speed of `9600` bits/sec (bauds).

```
int incomingNum = 0;
char incomingChar;

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

When we send data from the computer to the board, it is stored in a *buffer*. A buffer is a space in the memory in which, temporarily, some **data is stored waiting to be read**. To know if there is data waiting to be read in the buffer, we use the function `Serial.available()`, which checks how occupied that buffer is. It means, how many characters are stored in it.



Inside the loop, we are using:

```
Serial.available() > 0
```

With it, we are checking whether there is any data arriving through serial communication.

```
void loop()
{
  if (Serial.available() > 0)
  {
```

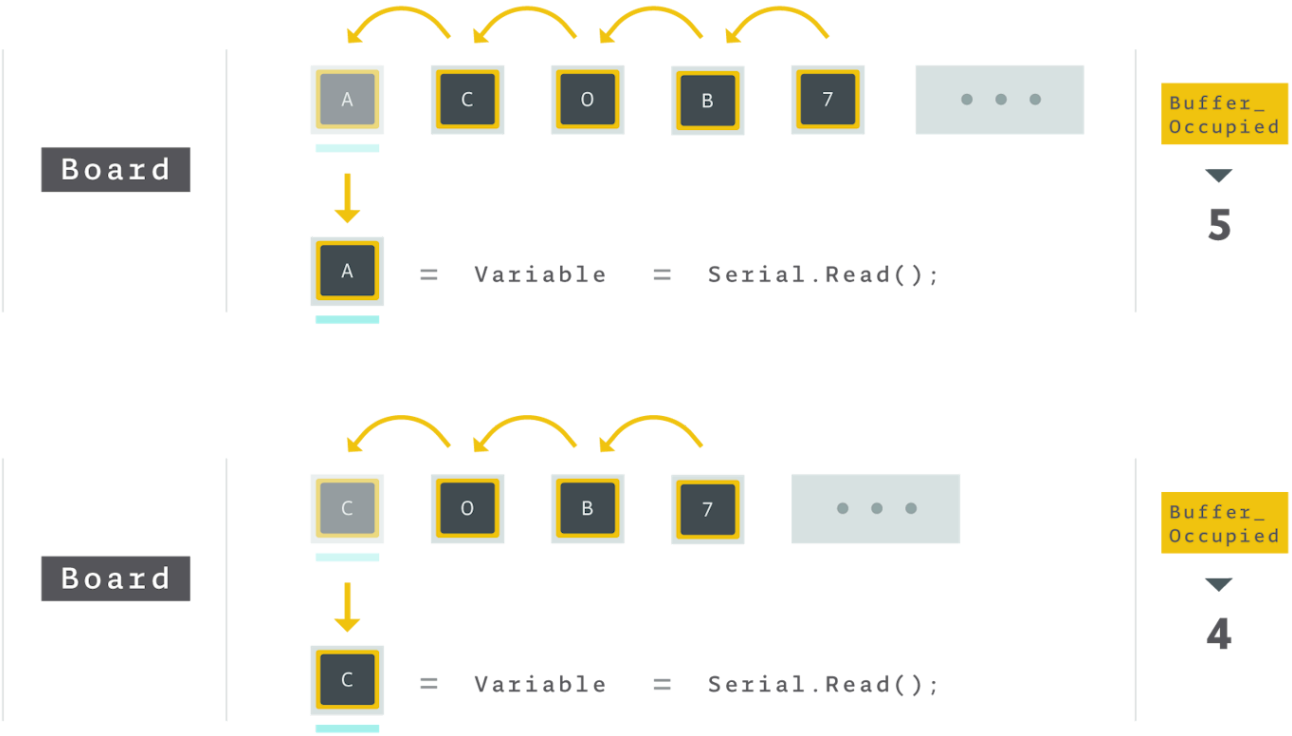
The data that arrive at the buffer is organized following first in, first out (FIFO) method, this means that when we want to retrieve this data from the buffer, we are going to start taking the first character came into the buffer.

Once we know that there is data in the buffer, we are going to retrieve it. To do this, we need a function that takes the data out of the buffer and stores it in a different variable. By doing this, we are able to **use the data that has arrived through serial communication.**

The function we are going to use to do this is:

`Serial.read()`

Every time a character is taken from the buffer, the amount of occupied space on it decreases. In our program, it continues until there is not data stored in the buffer.



Then, by using:

`incomingNum = Serial.read();`

We are taking the data arriving through serial communication and saving it in a variable.

In this case, we first store the incoming data in `incomingNum` and then save it in `incomingChar`, which is a different type of variable.

```
    incomingNum = Serial.read();
    incomingChar = incomingNum;
```

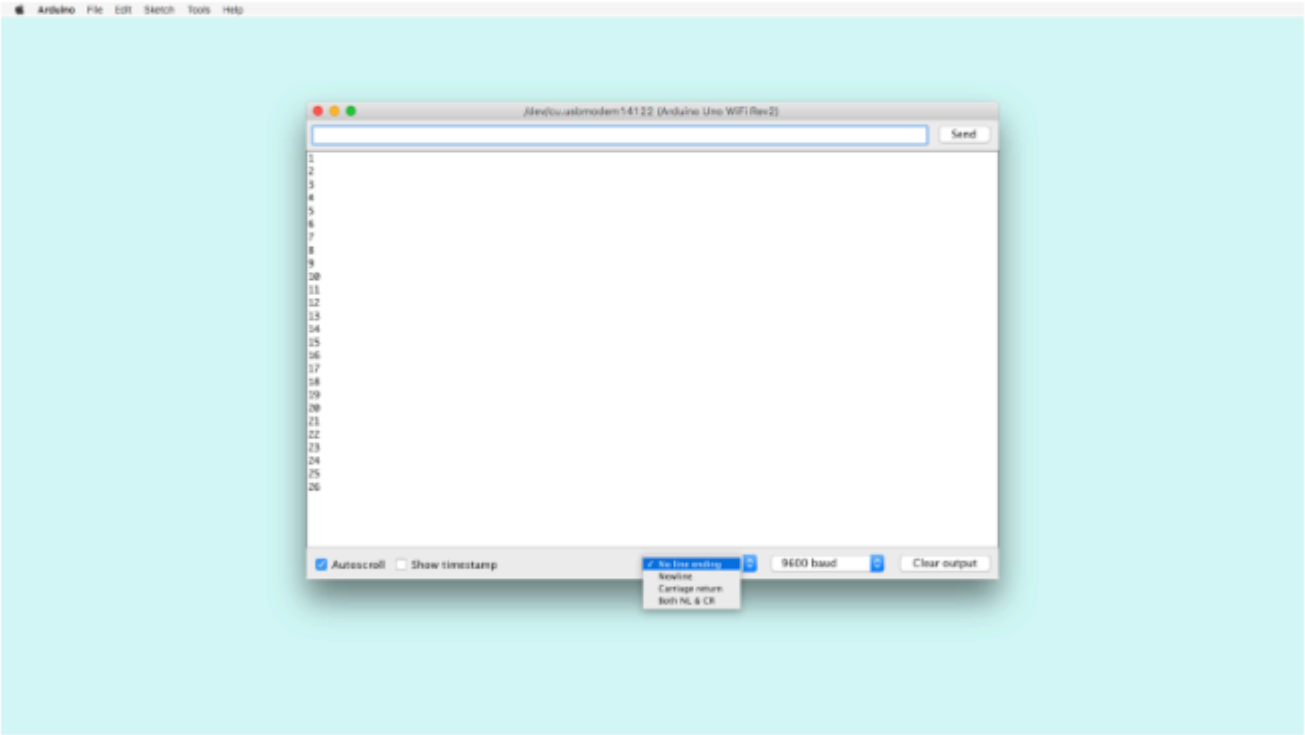
Once all is saved, we represent it in the Serial Monitor.

First, we print a row with the information we are going to print and then we print the variables we want to see: `incomingChar` and `incomingNum`.

```
    Serial.println("Character Code");
    Serial.print(incomingChar);
    Serial.print(" ");
    Serial.println(incomingNum);
  }
}
```

Experiment

- Try changing the Serial Monitor configuration and observe how it affects the data we send to the board.



- To see how the buffer size changes, let's modify the program to save the buffer size in a variable and then let's print its value after reading each character. To save the buffer size, we need to use the following command:

```
bufferSize = Serial.available();
```

- All codes for the different characters are presented according to the ASCII code; if you are interested in it, you can investigate it further.

TEACHER NOTES

The Serial Monitor should generally be set to *No line ending*.

Char variables can store numbers (as characters) as well, but we cannot make mathematical operations with them as characters properly.

You can demonstrate the effect with an example code, as shown below:

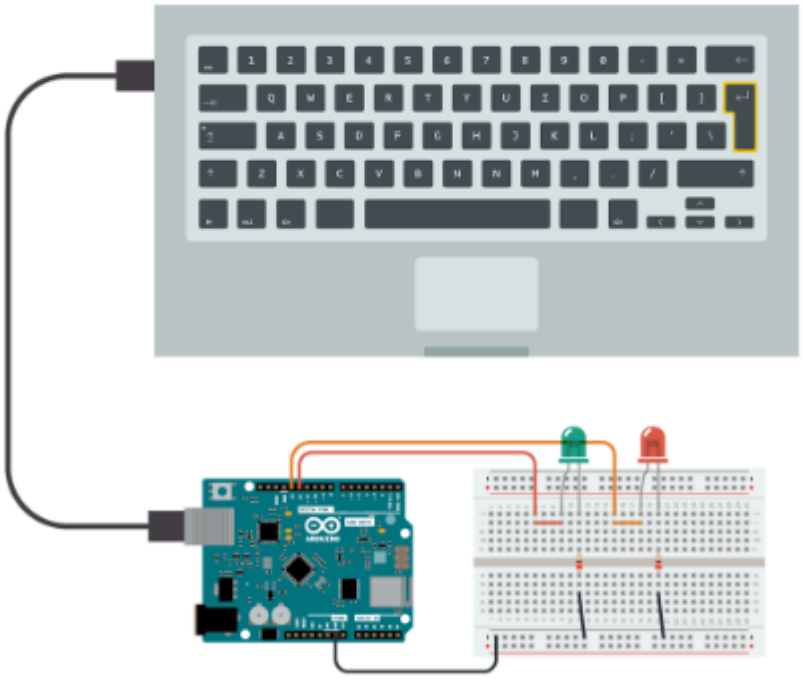
```
/*


*/
char firstNumber = '1';
char secondNumber = '2';
char newNumber = ' ';
void setup() {
 Serial.begin(9600);
}
void loop() {
 Serial.println("Adding characters in the print function:");
 Serial.println(firstNumber + secondNumber);
 Serial.println();
 Serial.println("Adding characters together in another char variable:");
 newNumber = firstNumber + secondNumber;
 Serial.println(newNumber);
 Serial.println();
 delay(2000);
}
```

Ask students to guess why you might get these outputs.

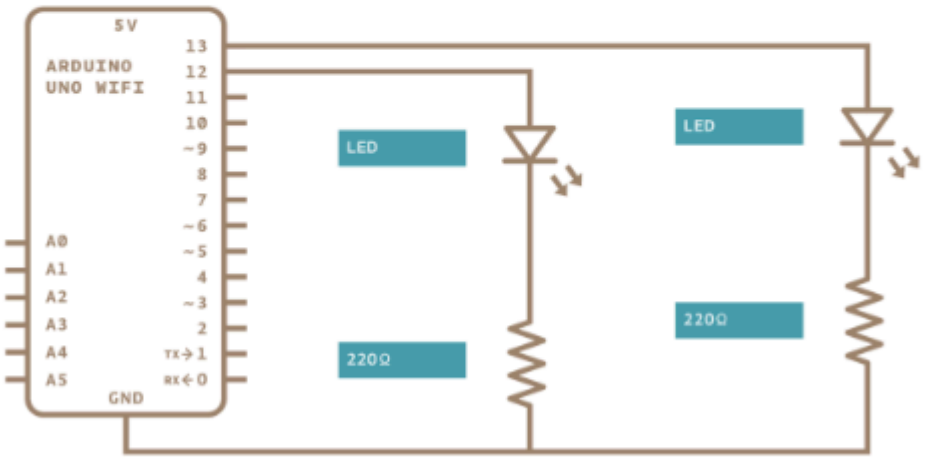
Vowels detector

In this activity, we will learn how we can control the state of some LEDs by sending data to the board through the serial communication.



Building the circuit

For this activity, we will need a circuit with two LEDs: a green LED connected to pin 13 and a red one connected to pin 12.



Programming the board

Let's program the board to count the number of vowels a word has. Every time the board detects a vowel, it will turn ON the green LED for half a second; however, if the board detects a consonant, it will turn ON the red LED for half a second. To do this, open the code in **File > Examples > CTC GO CORE > Lessons > SerialToBoard > SerialToBoard_Activity2** and upload it to the board.

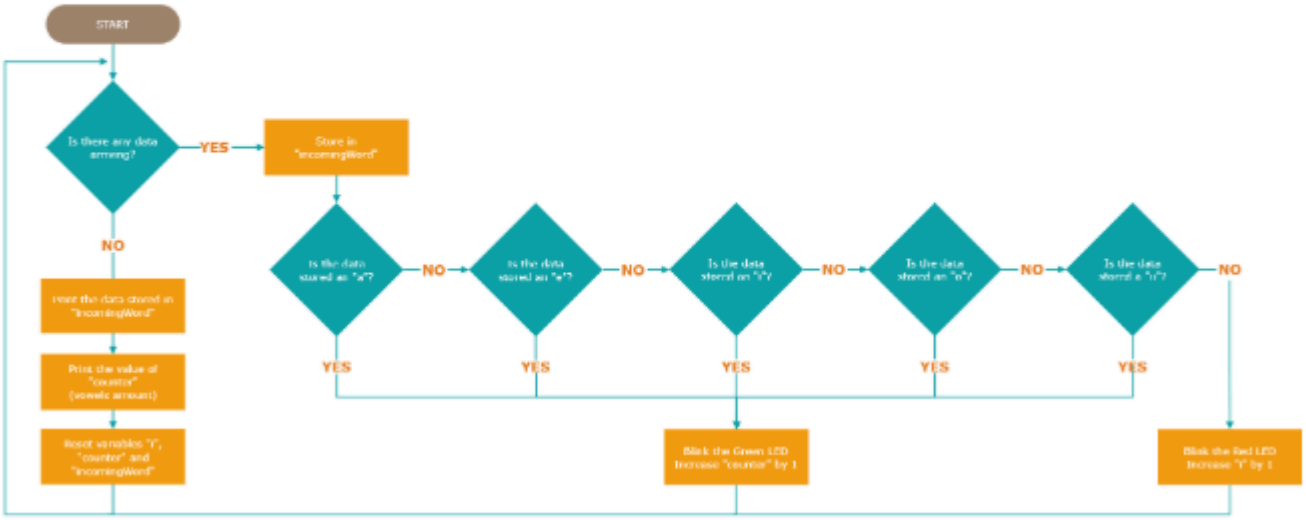
Now, to check if the program is working fine, we need to write a 10-letter word (i.e. a word that is 10 letters long, maximum).

Not working?

- Check that the selected board is correct.
- Check that the selected port is correct.
- Check that the speeds of the program and the Serial Monitor match.
- Check that the Serial Monitor configuration matches the configuration from Activity 1.
- Check that the connections are correct.

Understanding the program

The following flowchart shows the program behavior:



In the first part of the code, we initialize the variables.

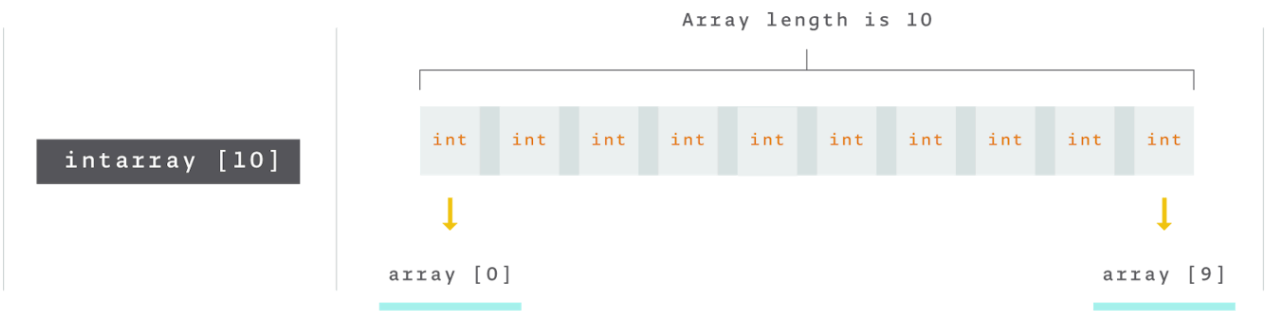
```
char incomingWord[10];
int counter = 0;
int i = 0;

int redLED = 13;
int greenLED = 12;
```

In this exercise, we are using a new type of variable called an *array*:

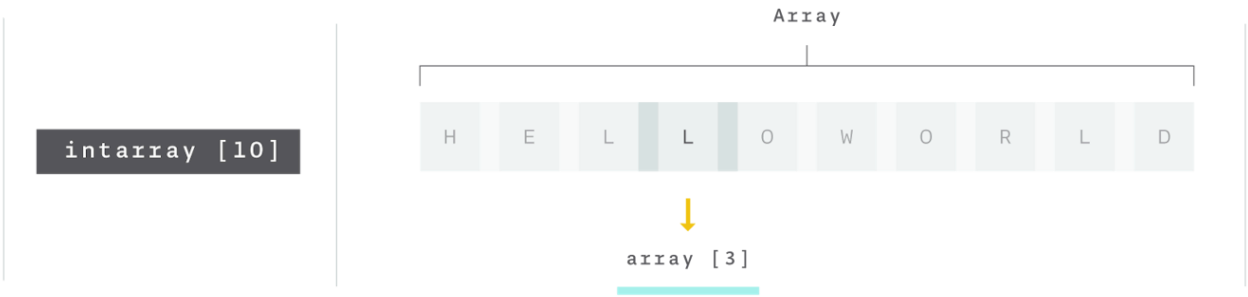
```
char incomingWord[10];
```

An array is a group of same type variables stored together. The number between curly braces indicates the number of elements that the array can store, in this case 10.



The **buffer** we saw in the previous activity is an **array**.

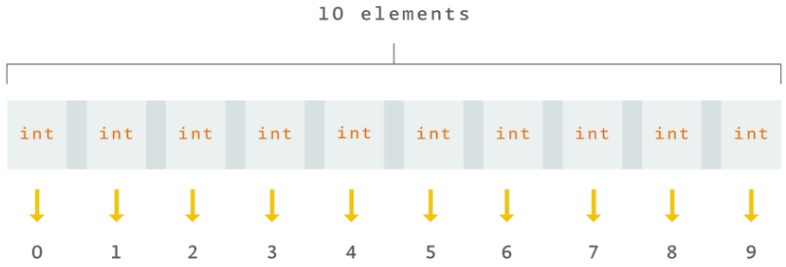
We can access an array element simply by referring to the array name and specifying the particular element we want to access.



We need to pay special attention when working with arrays. When we work with arrays, we **start counting from the number 0**, so the **first element of an array is**:

```
array[0]
```

```
intarray [10]
```


10 elements
int
int
int
int
int
int
int
int
int
int
0
1
2
3
4
5
6
7
8
9

In the setup, we initialize serial communication and the pins to which we connected the LEDs.

```
void setup()
{
  pinMode(redLED, OUTPUT);
  pinMode(greenLED, OUTPUT);
  Serial.begin(9600);
}
```

In the loop, we check to see if there is any data coming.

In case there is data coming, we are going to store it in the `incomingWord` array element that is indicated by the variable `i` that starts at `0`.

```
void loop()
{
  if (Serial.available() > 0)
  {
    incomingWord[i] = Serial.read();
```

Using `char` variables introduces changes to the way we are used to checking the value of a variable:

When we check the type `int` value, **we compare it with a number**:

```
incomingWord == 0
```

When we check the type `char` value, **we need to compare it with a character.** To do so, we need to place it between `' '`:

```
incomingWord == '0'
```

Then, we are going to check if the letter that has arrived is a vowel or not; in case it is a vowel: We will blink the green LED once for half a second, and then we will increase 1 unit in the counter variable.

```
    if (incomingWord[i] == 'a')
    {
      digitalWrite(greenLED, HIGH);
      delay(500);
      digitalWrite(greenLED, LOW);
      delay(500);
      counter += 1;
    }
```

We will repeat the same process for each vowel: a, e, i, o, and u.

```
    else if (incomingWord[i] == 'e')
    {
      digitalWrite(greenLED, HIGH);
      delay(500);
      digitalWrite(greenLED, LOW);
      delay(500);
      counter += 1;
    }
    else if (incomingWord[i] == 'i')
    {
      digitalWrite(greenLED, HIGH);
      delay(500);
      digitalWrite(greenLED, LOW);
      delay(500);
      counter += 1;
    }
    else if (incomingWord[i] == 'o')
    {
      digitalWrite(greenLED, HIGH);
      delay(500);
      digitalWrite(greenLED, LOW);
      delay(500);
      counter += 1;
    }
    else if (incomingWord[i] == 'u')
    {
      digitalWrite(greenLED, HIGH);
      delay(500);
      digitalWrite(greenLED, LOW);
      delay(500);
      counter += 1;
```

Once we know whether the letter that has arrived is a vowel or not, we will increase by 1 the value of the variable `i` in order to store the next letter arriving through serial communication.

```
    i += 1;
  }
```

TEACHER NOTES

Students may come up with the idea to simplify the code by swapping multiple conditionals with a single expression that combines conditions with `OR`, or `||`.

If students don't volunteer it, you can ask them if they can think of such a simplification.

To print the `incomingWord` array elements, we need to use a loop that is called a `for()` loop.

The `for()` loop executes the commands between the curly braces a certain number of times, which are indicated between the parentheses next to `for()` in the following way:

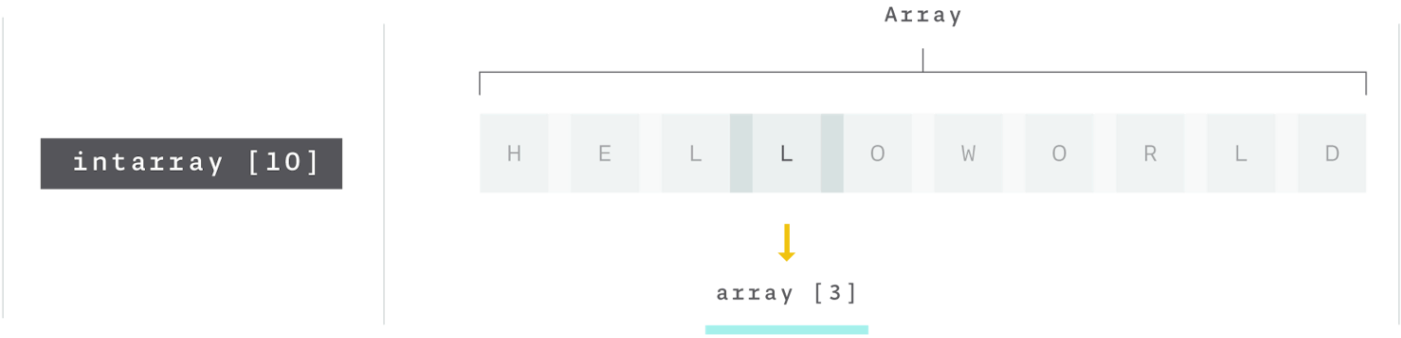
- Since `i` value is `0`, and while `i` value is smaller than `10`, it will execute the commands between the curly braces.
- Once all the commands between the curly braces have been **executed once**, it will increase the value of `i` and repeat the process.
- When `i` value is bigger than or equal to `10`, **go out of the loop.**

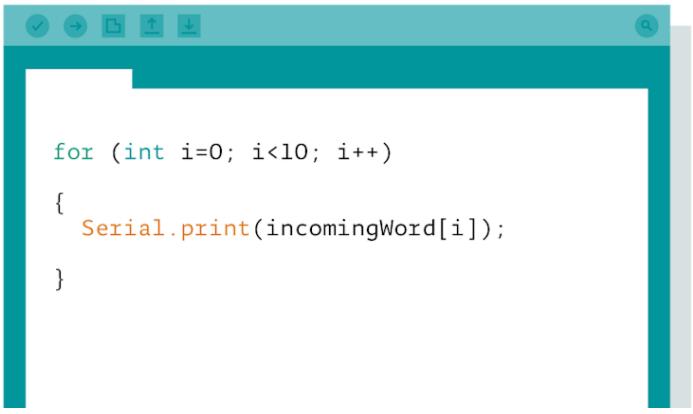
```
for (int i = 0; i < 10; i++)
{


}
```

To better understand the `for()` loop, let's see a simulation of the execution of the next `for()` loop code:

```
for (int i = 0; i < 10; i++)
{
  Serial.print(incomingWord[i]);
}
```





As you can see, the `for()` loop is going to be **repeated 10 times**, since the value of variable `i` is `0`. It will do so until the variable `i` value is `9`.

The way we apply this in our program is: if there is no more data arriving, let's print the data we have: First, we will print the word that has arrived, Second, we will print the number of vowels we have found in the word that has arrived, Third, we will wait 5 seconds.

```
  else
  {
    Serial.print("The word introduced is: ");
    for (int i = 0; i < 10; i++)
    {
      Serial.print(incomingWord[i]);
    }
    Serial.println("");
    Serial.print("It has: ");
    Serial.print (counter);
    Serial.println(" Vowels");
    delay(5000);
```

And the last step of the program is to reset all the variables in order to get them ready for the next piece of data arriving.

We use a `for()` loop in this step to put a `0` in all the elements of the incomingWord array.

```
    counter = 0;
    i = 0;
    for (int i = 0; i < 10; i++)
    {
      incomingWord[i] = 0;
    }
  }
}
```

Experiment

Modify the code, using a for loop, to blink the red LED 3 times whenever the program detects a consonant.

Guessing word

In this activity, we will learn how to store data in arrays, how to modify it, and how to check it while we build a game.

Programming the board

Let's program the board to build a game, where one of the group members chooses a secret word and the other ones have to guess it by introducing letters through the Serial Monitor. The basic configuration allows the players three mistaken letters. To do this, open the code in **File > Example > CTC GO CORE > Lessons > SerialToBoard > SerialToBoard_Activity3** and upload it to the board.

Once the program is open, we need to find the array called `selectedWord` and to change the word between `" "` with the one we want the others to guess. Then, we upload the sketch to the board and open the Serial Monitor.

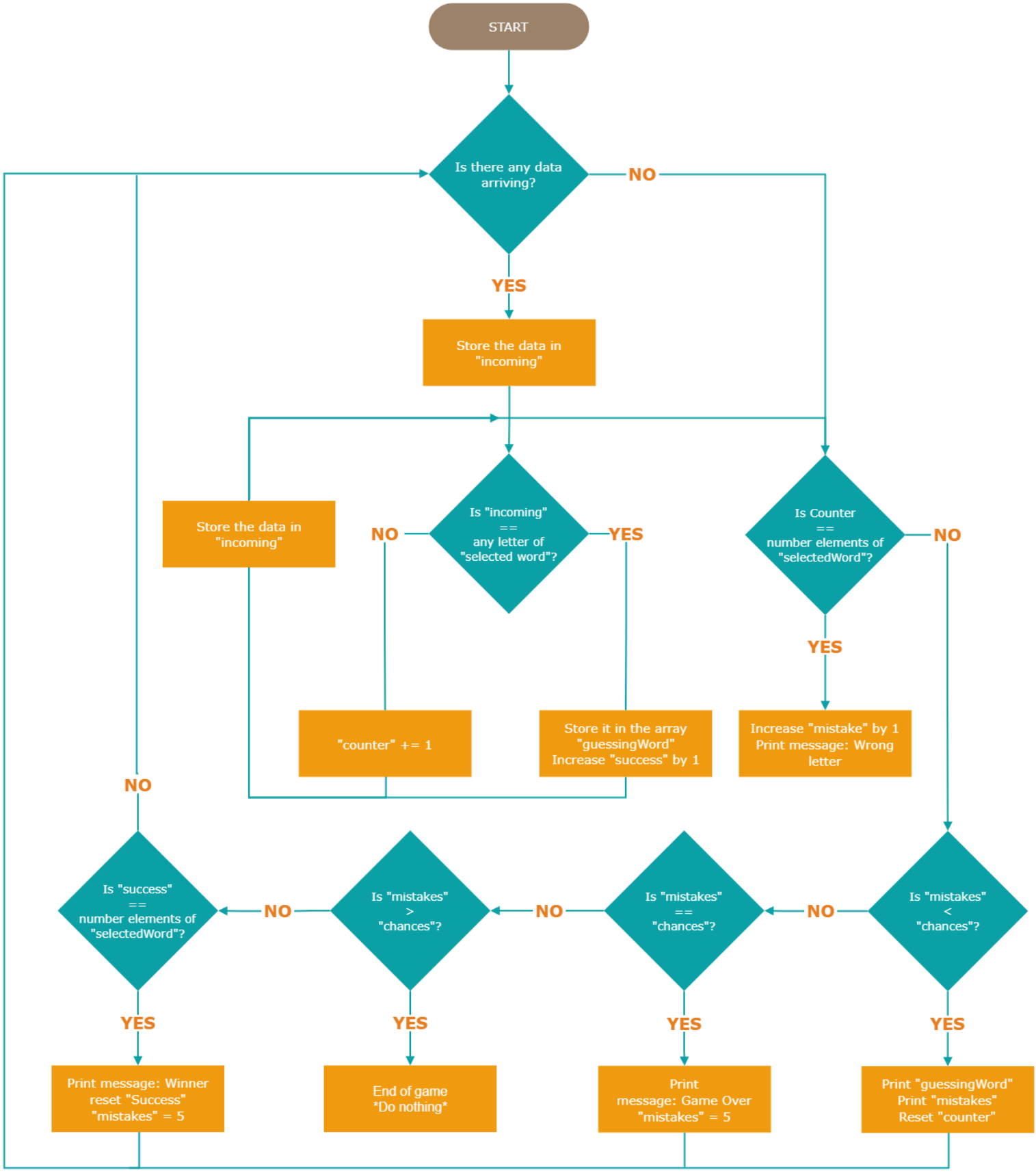
To restart the game, we have to close the Serial Monitor and reopen it.

Not working?

- Check that the selected board is correct.
- Check that the selected port is correct.
- Check that the speeds of the program and the Serial Monitor match.
- Check that the Serial Monitor configuration matches the configuration from Activity 1.

Understanding the program

The following flowchart shows how the code works:


START
Is there any data arriving?
NO
YES
Store the data in "incoming"
Store the data in "incoming"
Is "incoming" == any letter of "selected word"?
NO
YES
Is Counter == number elements of "selectedWord"?
NO
YES
"counter" += 1
Store it in the array "guessingWord" Increase "success" by 1
Increase "mistake" by 1 Print message: Wrong letter
NO
Is "success" == number elements of "selectedWord"?
NO
Is "mistakes" > "chances"?
NO
Is "mistakes" == "chances"?
NO
Is "mistakes" < "chances"?
YES
YES
YES
YES
Print message: Winner reset "Success" "mistakes" = 5
End of game *Do nothing*
Print message: Game Over "mistakes" = 5
Print "guessingWord" Print "mistakes" Reset "counter"

First, let's initialize the variables and arrays that we are going to use.

```
char selectedWord[] = {"HelloWorld"};
char guessingWord[sizeof(selectedWord)];
char incoming = 0;

int counter = 0;
int mistakes = 0;
int success = 0;
int chances = 3;
```

When we want to initialize an array with specific data, **we have to introduce this data between curly braces** `{ }`.

In case the data consists of characters, they should be between `" "` as well, same as in `Serial.print()`:

```
char selectedWord[] = {"HelloWorld"};
```

In this case, we have not indicated the array size with a number between the curly braces. The array size is the amount of letters we have added between the `" "` + 1. When we initialize an array with characters, a symbol that tells the microcontroller that the array finishes there is added automatically just after the last letter.

The `selectedWord` array length is 5 (letters) + 1 = 6.

In case the data consists of numbers, we can add them between curly braces:

```
selectedNumber[] = {1021312};
```

In this case, the size of the array is the amount of numbers that we added between the curly braces. The `selectedNumber` array length = 7.

When we are working with arrays that could change their size and we need to know the array size, to do this, we can use the function:

```
sizeof(array);
```

It will give us the number of elements that **the array has between the parentheses.**

In the following example, the array `guessingWord` always has the same number of elements (letters) as the `selectedWord`.

```
char guessingWord[sizeof(selectedWord)];
```

In the setup, we initialize serial communication.

Then, we fill the `guessingWord` array with an underscore, `'_'`, which will allow the players to know the position of the achieved letters.

```
void setup()
{
  Serial.begin(9600);
  for (int i = 0; i < sizeof(selectedWord) - 1; i++)
  {
    guessingWord[i] = '_';
  }
  Serial.print("The Word you are looking for is: ");
  Serial.println(guessingWord);
}
```

In the loop, the first thing we do, if there is data arriving, is to store it in the `incoming` variable.

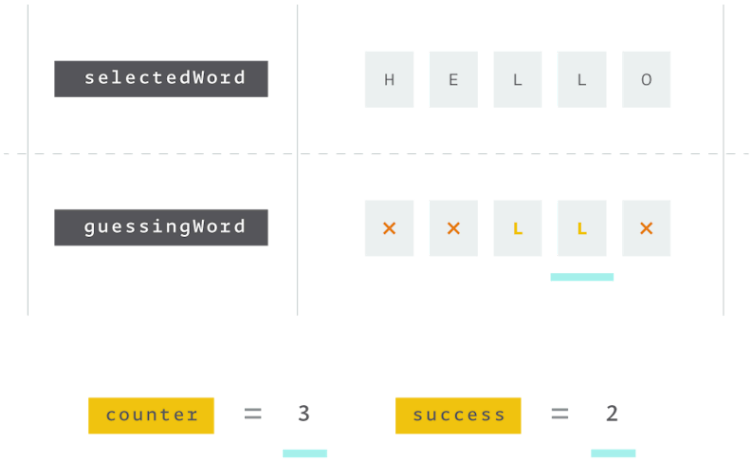
Then, we will go through the `selectedWord` array, by using a `for()` loop, checking if `incoming` value is the same as any of the elements of the `selectedWord` array.

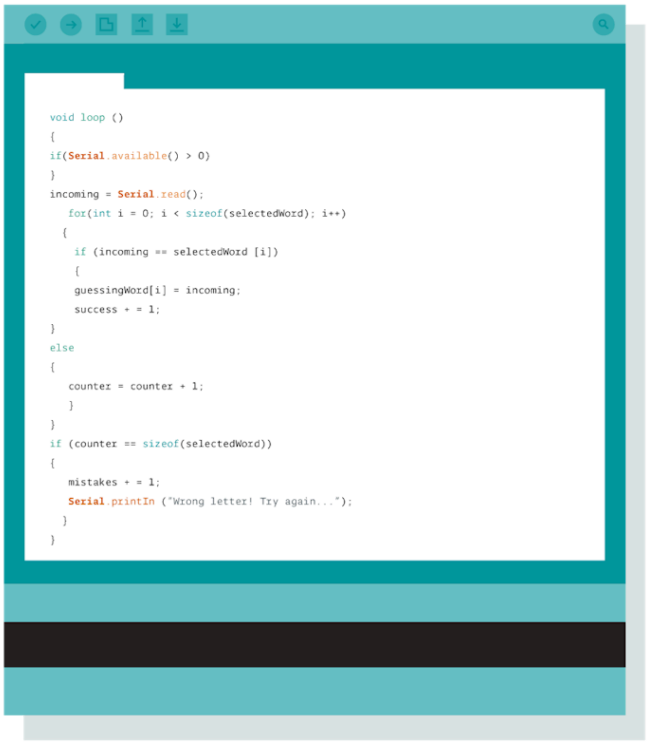
In case we want to find an element with the same value as `incoming`, we store `incoming` in the same element number of the `guessingWord` and increase the `success` variable by one.

In case we do not find any element of `selectedWord` with the same value as `incoming`, we just increase the value of the `counter` variable by one and print a message.

```
void loop()
{

  if (Serial.available() > 0)
  {
    incoming = Serial.read();
    for (int i = 0; i < sizeof(selectedWord); i++)
    {
      if (incoming == selectedWord[i])
      {
        guessingWord[i] = incoming;
        success += 1;
      }
      else
      {
        counter = counter + 1;
      }
    }
```





```
void loop ()
{
if(Serial.available() > 0)
}
incoming = Serial.read();
   for(int i = 0; i < sizeof(selectedWord); i++)
  {
    if (incoming == selectedWord [i])
    {
    guessingWord[i] = incoming;
    success + = 1;
}
else
{
   counter = counter + 1;
   }
}
if (counter == sizeof(selectedWord))
{
   mistakes + = 1;
   Serial.printIn ("Wrong letter! Try again...");
  }
}
```

After checking the incoming data, we check the value of counter.

If `counter` **value is equal to the number of elements** that the `selectedWord` has, it means that there is no match between the letter that has arrived and the `selectedWord`, which means that the player made a mistake.

In that case, we increase the `mistakes` variable by one and print a message.

```
    if (counter == sizeof(selectedWord))
    {
      mistakes += 1;
      Serial.println("Wrong letter!! Try again...");
    }
  }
```

When there is **no data arriving** and the value of the `mistakes` value is **smaller than that of the chances variable**, we show the player the **game state**.

To do this, we print `guessingWord` array that shows what letters the player has guessed and the number of mistakes made.

We also reset the counter to have it ready for the next letter.

```
  if (mistakes < chances)
  {
    Serial.print("The Word you are looking for is:  ");
    for (int i = 0; i < sizeof(selectedWord); i++)
    {
      Serial.print(guessingWord[i]);
    }
    Serial.print("      Mistakes: ");
    Serial.println(mistakes);
    counter = 0;
    delay(1000);
  }
```

When the **number of mistakes is equal to the number of allowed chances**, the game finishes.

When the variable `mistakes` is bigger than `chances`, the program enters a certain state, in it, the **program is stuck doing nothing.**

```
  else if (mistakes == chances)
  {
    Serial.println();
    Serial.println("Game Over");
    mistakes = 5;
  }

  else
  {}
```

If the `success` value is equal to the number of `selectedWord` elements without the end of array character, it means that all the letters are guessed and that the **player won the game.**

```
  if (success == sizeof(selectedWord) - 1)
  {
    Serial.println();
    Serial.print("WINEER!!");
    success = 0;
    mistakes = 5;
  }
}
```

Experiment

Modify the number of mistaken letters that the game allows the players in order to make the game more difficult.

Challenge

Modify the code from Activity 3 in order to:

- Add a green LED that will blink once for each correct letter that is introduced through the Serial Monitor and that will stay ON when the player wins the game.
- Add a red LED that will blink once for each mistaken letter that is introduced through the Serial Monitor and that will stay ON when the game is over.

COMPLETE PROGRAM

C/C++

```
/*
 CTC GO! CORE MODULE
 LESSON 06 - Serial to Board

 This sketch is written to accompany Activity 3 in Lesson 06 of the CTC GO! core module
*/

char selectedWord[] = {"HelloWorld"};
char guessingWord[sizeof(selectedWord)];
char incoming = 0;

int counter = 0;
int mistakes = 0;
int success = 0;
int chances = 3;

//Add the pinout
int greenLED = 13;
int redLED = 12;

void setup()
{
 //Set up the LEDs
 pinMode(greenLED , OUTPUT);
 pinMode(redLED, OUTPUT);

 Serial.begin(9600);
```

```
 for (int i = 0; i < sizeof(selectedWord) - 1; i++)
 {
   guessingWord[i] = '_';
 }
 Serial.print("The Word you are looking for is: ");
 Serial.println(guessingWord);
}

void loop()
{

 if (Serial.available() > 0)
 {
   incoming = Serial.read();
   for (int i = 0; i < sizeof(selectedWord); i++)
   {
     if (incoming == selectedWord[i])
     {
       guessingWord[i] = incoming;
       success += 1;

       //green LED blink
       digitalWrite(greenLED , HIGH);
       delay(500);
       digitalWrite(greenLED , LOW);
     }
     else
     {
       counter = counter + 1;
     }
   }

   if (counter == sizeof(selectedWord))
   {
     mistakes += 1;
     Serial.println("Wrong letter!! Try again...");

     //red LED Blink
     digitalWrite(redLED , HIGH);
     delay(500);
     digitalWrite(redLED , LOW);

   }
 }
```

```
 if (mistakes < chances)
 {
   Serial.print("The Word you are looking for is:  ");
   for (int i = 0; i < sizeof(selectedWord); i++)
   {
     Serial.print(guessingWord[i]);
   }
   Serial.print("      Mistakes: ");
   Serial.println(mistakes);
   counter = 0;
   delay(1000);
 }

 else if (mistakes == chances)
 {
   Serial.println();
   Serial.println("Game Over");
   mistakes = 5;

   //red LED ON
   digitalWrite(redLED , HIGH);
 }

 else
 {
   //green LED ON
   digitalWrite(greenLED ,HIGH);}

 if (success == sizeof(selectedWord) - 1)
 {
   Serial.println();
   Serial.print("WINEER!!");
   success = 0;
   mistakes = 5;


 }

}
```

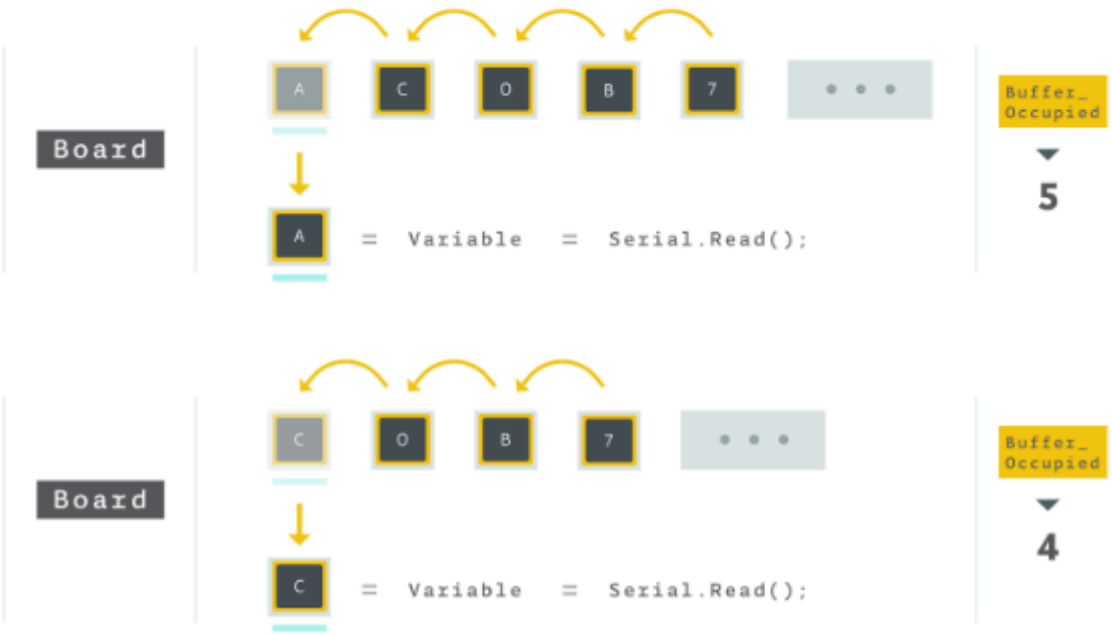
Conclusions

In this lesson, we learned:

- about the different variables that allow us to store various types of data,

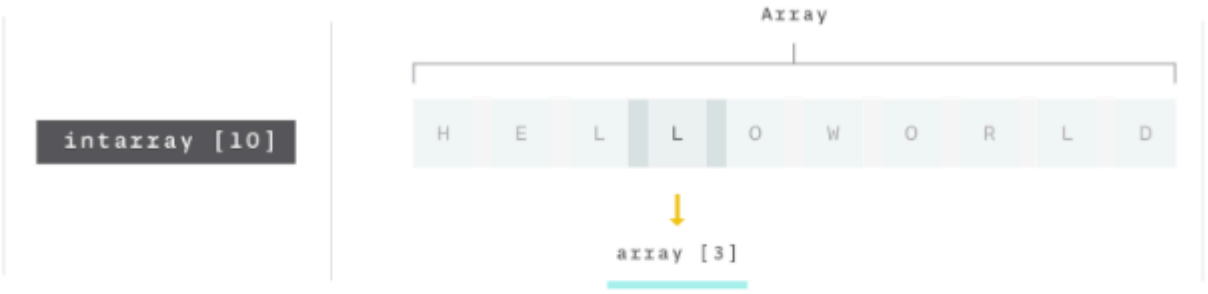
Data	Type
Numbers (0, 1, 2, 3)	int
Characters (a, b, c, +, *, (,))	char

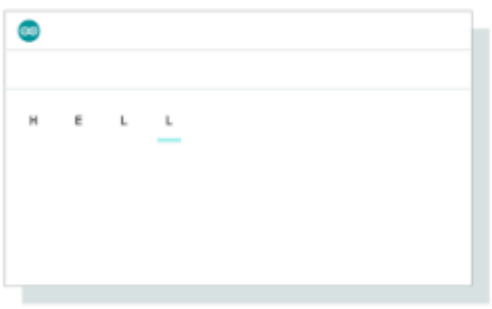
- what is a buffer and how to read data from the Arduino board buffer,

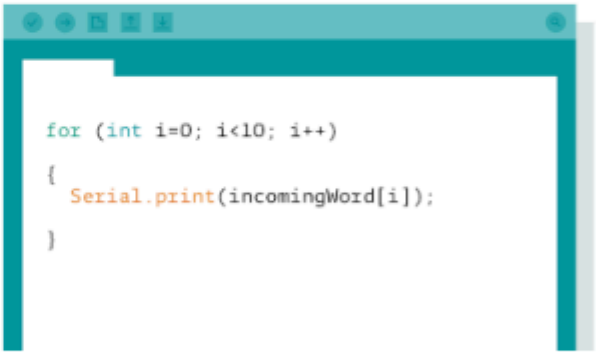


- how to use the data that has been stored in the buffer through the concept of arrays,

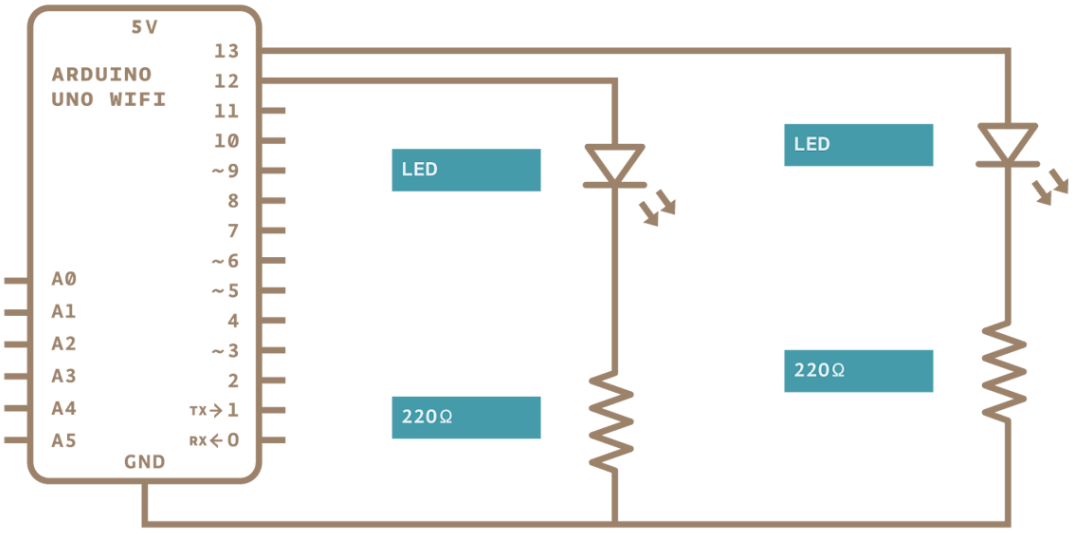
- accessing the elements of the array using a special programming concept called for-loops,and







- sending and receiving data between an arduino board and a computer creating a two way communication channel.



Wrapping up

At the end of the class, all materials should be organized in their respective boxes to make sure that they are ready for use in the next lesson.

Before leaving, make sure that:

- the battery is disconnected from the battery holder,
- there are no components on the floor, and
- that broken components (if any) are reported to the teacher.

Afterclass

Since you might have created some really clever solutions for home automation for this lesson, it would be a shame not to do some tinkering in connection with it.

That wonderful invention most probably happens to be quite complicated, so try thinking of a simpler approach for now and make a plan for how you could make use of what you have learned thus far.

Create an algorithm and a schematic circuit built with Arduino for a basic solution to the tool suggested in the last Afterclass. You can have just one function for the product.

In case you do not have an idea or do not want to work on your own invention, there is one here that you can use:

A Smart Fridge

Main functions

- Automatic reorder
  - Makes a list of different food products and assigns a value to each of the products that represents the ideal amount of that product
  - Orders food if the amount of it drops below a set value and if automatic ordering is turned on for the product.
  - The amount of order is x and it is calculated as x= ideal+1-current
- Tracks expiry dates
  - Notifies about expired food.

Pro

- Convenience
  - It is not necessary to bother thinking about shopping every day.

Contra

- Too much waste
- Probably buys more than what is needed.
- Stocks everything we set, so we might not be able to eat everything before it expires.

Additional costs over a simple product

- Barcode scanner.
- Information panel.
- Internet connection for ordering, purchase option.
- Database with all products, where to buy them, when they expire, how much they cost.

SOLUTIONS TO RECAPPED LESSON 5

QUESTION 1

TEACHER NOTES

The matching commands for the outcomes in order are:

- Serial.print("number");
- Serial.println("number");
- Serial.print(number);
- Serial.println(number);

QUESTION 2

TEACHER NOTES

The matching symbols for the actions are:

assignment of value to a variable	=
comparison of values/variables	==
logical AND	&&
logical OR	||
incrementing a variable by 1	+=
decrementing a variable by 1	-=
declaring a text as a string	""
setting the follow-up text as a comment	//
closing a command	;