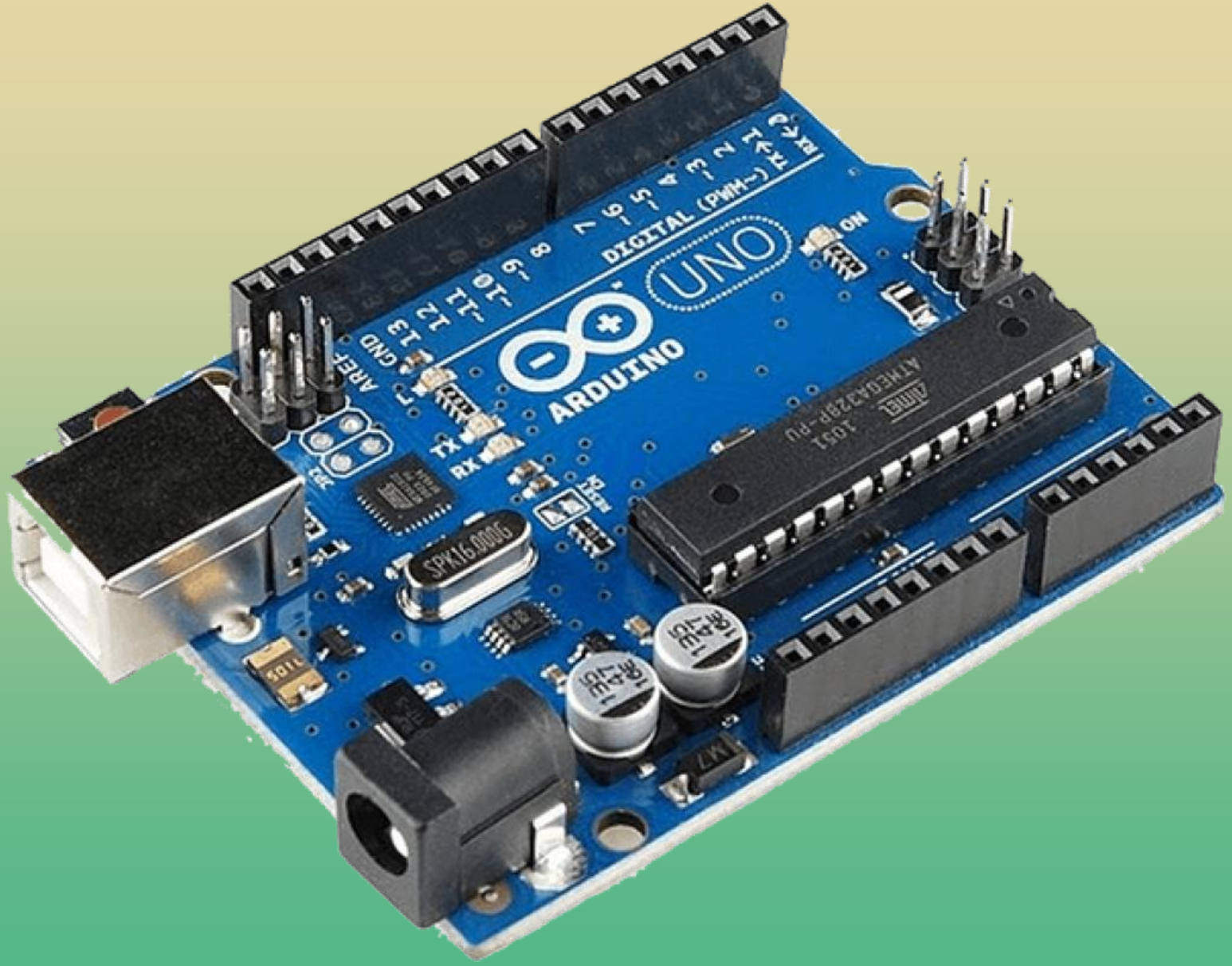
Programming with Arduino Uno

Unit - III

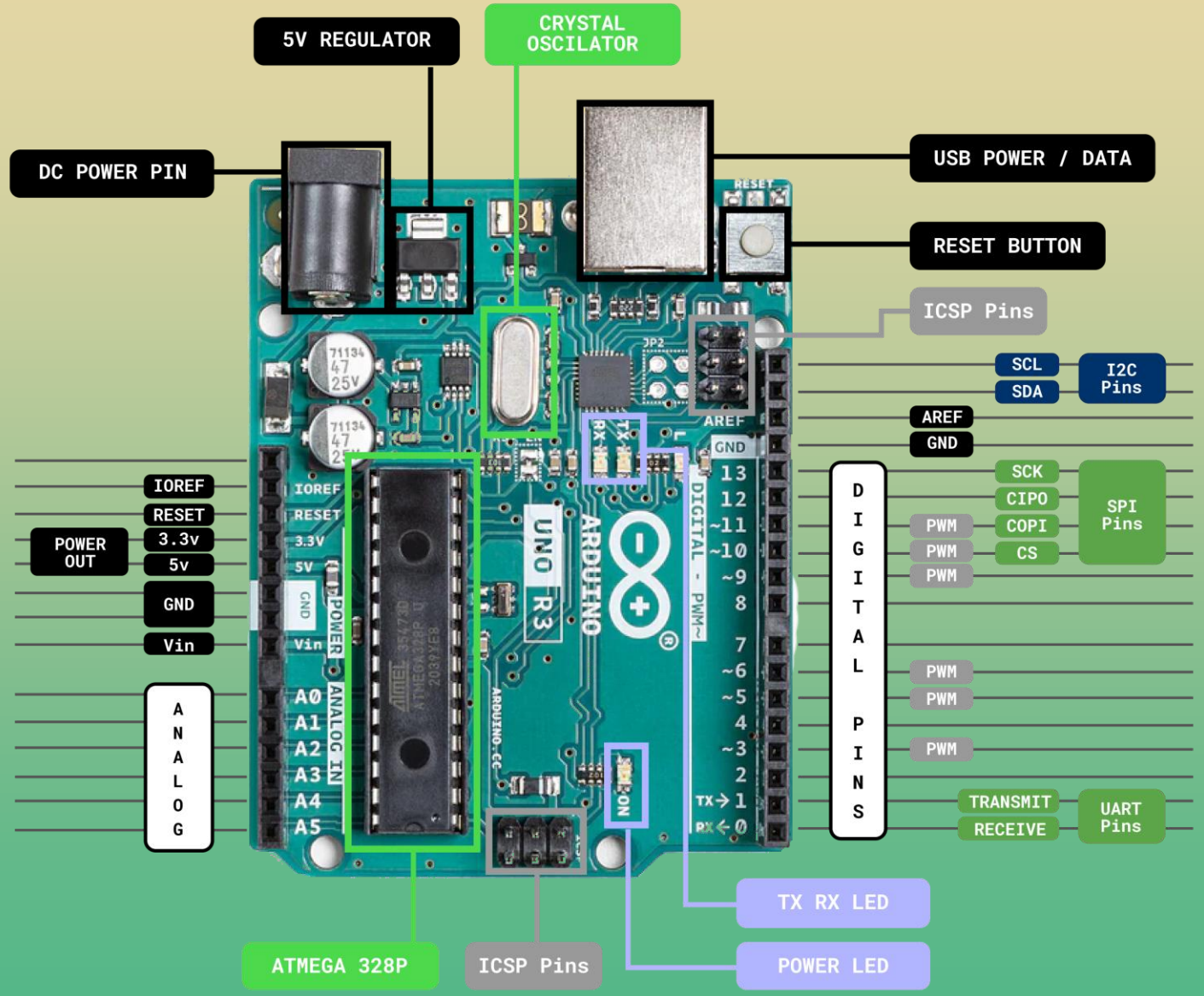
Topics to be covered

- Block Diagram of Arduino Uno
- Sketch Structure
- Data types & Built-in Constants
- Operators
- Control Statements & Loops
- Functions & Library Functions
  - User Defined Functions
  - Library Functions
- Basic LED Blinking program in Arduino Uno
- Serial Communication functions

Block Diagram of Arduino Uno



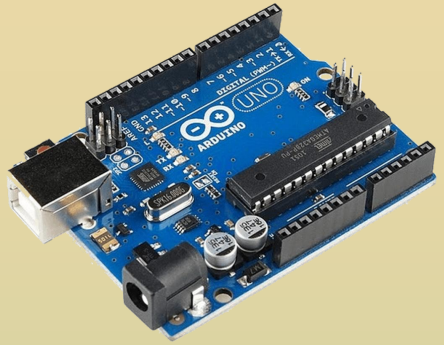
Block Diagram of Arduino Uno



Arduino Uno Pins

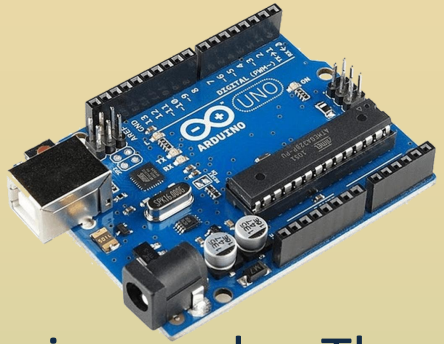
- **Power Connections:**
  - Arduino Uno board can be powered using USB type-B as well as Barrel Jack pins available at the top of the board
  - These provide power connection to turn on the board. Also USB port can be used to feed program to Arduino board.
- **Power Pins:**
  - Arduino board is available with built in Power pins for 3.3 V and 5 V, that can be directly provided to some devices directly connected with Arduino board.
  - Arduino is able provide power of up to 500mA for 5V and 50mA for 3.3V pins.
  - Also GND pins are available to close the circuit.
  - Vin pin is used to take 7-12 V as an Unregulated Voltage input, Vin is connected to the voltage regulator which convert this unregulated voltage into a regulated 5V DC for components to work properly.
- **IORef:**
  - Pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V

Arduino Uno Pins



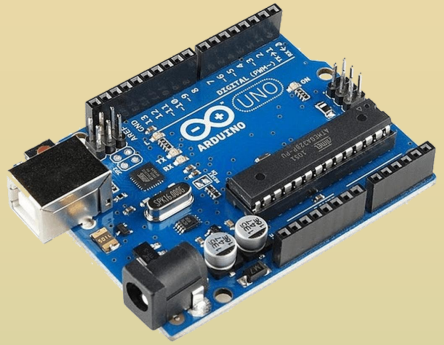
- **AtMega 328P:**
  - A Micro controller,
    - 8-bit Word Size
    - 16 MHz Clock
    - 32 KB Flash memory
    - 1 KB EEPROM
    - 2 KB SRAM
- **Reset Button:**
  - to reset the program.
- **16MHz Crystal Oscillator:**
  - To generate and supply common clock to all the connected devices.

Arduino Uno Pins



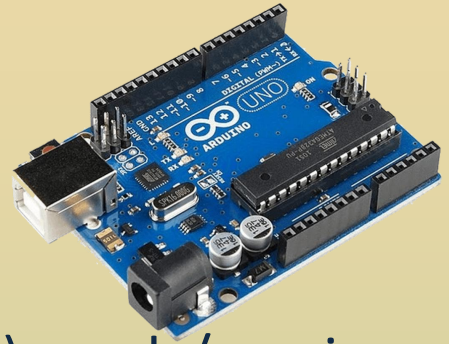
- **Memory:**
  - The SAM3X has 512 KB (2 blocks of 256 KB) of flash memory for storing code. The bootloader is preburned in factory from Atmel and is stored in a dedicated ROM memory. The available SRAM is 96 KB in two contiguous bank of 64 Kb and 32 KB. All the available memory (Flash, RAM and ROM) can be accessed directly as a flat addressing space
- **ICSP for USB Controller and AtMega328:**
  - In-Circuit-Serial-Programming, used to connect several Arduino boards in Serial to work together.
- **Aref (Analogue REFerence):**
  - Allows us to feed the Arduino a reference voltage from an external power supply. For example, if we want to measure voltages with a maximum range of 3.3V, we would feed a nice smooth 3.3V into the AREF pin – perhaps from a voltage regulator IC.

Arduino Uno Pins



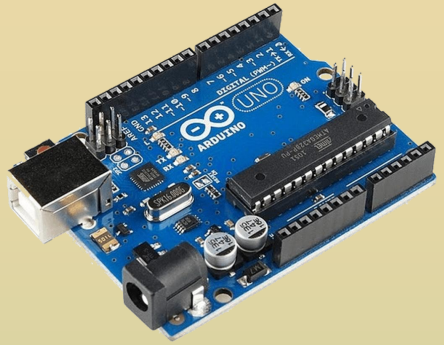
- **SPI (Serial Peripheral Interface) Pins:**
  - Only used when multi board communication is needed.
  - **SCK (Serial Clock):** The clock pulses which synchronize data transmission generated by the Controller and one line specific for every device
  - **CIPO (Controller In Peripheral Out):** The Peripheral line for sending data to the Controller
  - **COPI (Controller Out Peripheral In):** The Controller line for sending data to the peripherals.
  - **CS (Chip Select):** the pin on each device that the Controller can use to enable and disable specific devices. When a device's Chip Select pin is low, it communicates with the Controller. When it's high, it ignores the Controller. This allows you to have multiple SPI devices sharing the same CIPO, COPI, and SCK lines.

Arduino Uno Pins



- **TX/RX Pins:**
  - Used for Digital Serial Communication. TX (Transmitter) and RX (Receiver) sends/receives 1-byte of Digital Data at a time.
  - The UART (Universal Asynchronous Receiver/Transmitter) protocol converts the byte into a serial stream of 8 bits + additional framing bits - start, stop & parity.
- **6x Analog In or Digital IN/OUT:**
  - For providing interfacing with Analog devices, Arduino has A0 to A5 (6-Pins) with built-in 10-bit ADC, to convert an analog signal into a 10-bit digital signal.
- **Digital Pins:**
  - Pins 0-13 (14 pins) are used to send/receiver digital information (Binary).
  - Where Pins 3, 5, 6, 9, 10, 11 can also be used as PWM (Pulse Width Modulation) Pins, where we can modify duty cycle of the Digital signal to provide varying voltage in Output.

Arduino Uno Pins



- **Built-in LED:**
  - To check basic communication Arduino Uno board comes with a Built-in LED which is connected to Pin-13.
- **TX/RX LED:**
  - Blinks when digital TX/RX is underway.
- **Power LED:**
  - Turns-On when the board is Powered ON.

Sketch Structure

- For Arduino programming C++ code is used, however .ino or .pde extention is used for the Arduino program.
- The program of Arduino is called a “Sketch”.
- 3 main parts
  - **Structure**
  - **Values (variables and constants)**
  - **Functions**
    - **User Defined Functions**
      - Some other functions needed to be defined by the user.
    - **void setup()**
      - The setup() function is called when a sketch starts.
      - Use it to initialize the variables, pin modes, start using libraries, etc.
      - The setup function will only run once, after each power up or reset of the Arduino board.
    - **void loop()**
      - After creating a setup() function, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

Data Types

- All C++ data types are available for programming.
  - int
  - char
  - boolean
  - float
  - short
  - long
  - unsigned int
  - unsigned long
  - unsigned char
  - double
  - Array
  - String (char*)

Constants in Arduino

- 2-types
  - User-defined Constants:
    - Can be defined as pre-processor macro using **#DEFINE**
    - Also using **const** keyword, a constant can be defined.
  - Built-in Constants:
    - Some built-in constants are available in Arduino to help with the programming
    - Logical level constants
      - True
      - False
    - Pin level constants
      - HIGH – indicates high signal, 1 logic level
      - LOW – indicates low signal, 0 logic level
    - Led constant:
      - LED_BUILTIN – indicating built-in LED Pin, which is 13.
    - Some more constants may be available, when we use different libraries.

Operators: Arithmetic

Operator	Description	Example
+	Addition	`int result = 5 + 3;`
-	Subtraction	`int result = 7 - 2;`
*	Multiplication	`int result = 4 * 6;`
/	Division	`int result = 8 / 2;`
%	Modulus	`int result = 10 % 3;`
++	Increment	`int num = 5; num++;`
--	Decrement	`int num = 5; num--;`

Operators: Boolean

Operator	Description	Example
&&	Logical AND	`if (condition1 && condition2)` `{ /* Code if both conditions are true */ }`
||	Logical OR	`if (condition1 || condition2)` `{ /* Code if atleast one condition is true */ }`
!	Complement	`if (!condition)` `{ /* Code if the condition is false */ }`

Operators: Comparison

Operator	Description	Example
Equal to (==)	Checks if two values are equal	`If (temperature == 25)` `{ /* Code if temperature is 25 */ }`
Not equal to (!=)	Checks if two values are not equal	`If (humidity != 70)` `{ /* Code if humidity is not 70 */ }`
Less than (<)	Checks if the left value is less than the right value	`If (lightIntensity < 500)` `{ /* Code if light intensity is less than 500 */ }`
Greater than (>)	Checks if the left value is greater than the right value	`If (pressure > 1000)` `{ /* Code if pressure is greater than 1000 */ }`
Less than or equal to (<=)	Checks if the left value is less than or equal to the right value	`If (distance <= 50)` `{ /* Code if distance is less than or equal to 50 */ }`
Greater than or equal to (>=)	Checks if the left value is greater than or equal to the right value	`If (speed >= 60)` `{ /* Code if speed is greater than or equal to 60 */ }`

Operators: Compound

Operator	Description	Example
+=	Add and assign	`variable += 5`
-=	Subtract and assign	`variable -= 3`
*=	Multiply and assign	`variable *= 2`
/=	Divide and assign	`variable /= 4`
%=	Modulus and assign	`variable %= 3`

Operators: Comparision

Operator	Description	Example
Equal to (==)	Checks if two values are equal	`If (temperature == 25)` `{ /* Code if temperature is 25 */ }`
Not equal to (!=)	Checks if two values are not equal	`If (humidity != 70)` `{ /* Code if humidity is not 70 */ }`
Less than (<)	Checks if the left value is less than the right value	`If (lightIntensity < 500)` `{ /* Code if light intensity is less than 500 */ }`
Greater than (>)	Checks if the left value is greater than the right value	`If (pressure > 1000)` `{ /* Code if pressure is greater than 1000 */ }`
Less than or equal to (<=)	Checks if the left value is less than or equal to the right value	`If (distance <= 50)` `{ /* Code if distance is less than or equal to 50 */ }`
Greater than or equal to (>=)	Checks if the left value is greater than or equal to the right value	`If (speed >= 60)` `{ /* Code if speed is greater than or equal to 60 */ }`

Operators: Bitwise

Operator	Description	Example
&	Bitwise AND	`result = variable1 & variable2`
|	Bitwise OR	`result = variable1 | variable2`
^	Bitwise XOR	`result = variable1 ^ variable2`
~	Bitwise NOT	`result = ~variable`
<<	Left Shift	`result = variable << 2`

Control Statements & Loops

- Same as C++
- Control Structures are:
  - if
  - if … else
  - Nested if … else
  - else if Ladder
  - Switch case
  - For loop
  - While loop
  - Do … while loop

Control Statements & Loops

- if(expression)
  {//Statements if true}

- if(expression)
  {//Statements if true}
  else
  {//Statements if false}

- if(expression)
  {//Statements if true
  if(expression)
  {//Statements if true}
  }

- if(expression1)
  {//Statements if true}
  else if(expressio2)
  {//Statements if true}
  else
  {//Statements if false}

Control Statements & Loops

- for(init; expressions; post_statemets)
  {//Statements until expression is false}

- while(expression)
  {//Statements until expression is false}

- do
  {//Statements before checking expression} while(expression};

User defined Functions

- If the user needs to performs specific tasks repeatedly at several places, they can define a UDF in the sketch itself.
- Syntax is same as C++.

```
<return-type> <function-name> (<param1>, <param2>)
    {//Functions Statements.}
```

Library Functions: I/O Functions

- Pins can be configured to take input or to give output and data can be received/transmitted from those pins respectively.
- For that some functions are available.
- To set the pin mode

  ```
  pinMode (pin, mode)
  ```

  - pin – the number of the pin whose mode you wish to set
  - mode – INPUT, OUTPUT, or INPUT_PULLUP.
- To read Digital value from the pin

  ```
  digitalRead (pin)
  ```

  - pin – the number of the pin whose mode you wish to set
  - Returns the value from the pin as a return value.

Library Functions: I/O Functions

- To write digital value to the pin

  ```
  digitalWrite (pin, value)
  ```

  - pin – the number of the pin whose mode you wish to set
  - value – HIGH, or LOW.
- To read analog value from the pin

  ```
  analogRead (pin)
  ```

  - pin – the number of the analog input pin to read from.
  - Returns converted value from analog to digital using ADC.
- To write analog (PWM) value to the pin

  ```
  analogWrite (pin, value)
  ```

  - pin – the number of the pin whose mode you wish to set
  - value – 0 to 255

Library Functions: I/O Functions

- To Configure the reference voltage used for analog input (i.e. the value used as the top of the input range). *type* can be given as below:

```
analogReference (type)
```

  - *DEFAULT:* the default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)
  - *INTERNAL:* a built-in reference, equal to 1.1 volts on the ATmega168 or ATmega328P and 2.56 volts on the ATmega32U4 and ATmega8
  - *INTERNAL1V1:* a built-in 1.1V reference (Arduino Mega only)
  - *INTERNAL2V56:* a built-in 2.56V reference (Arduino Mega only)
  - *EXTERNAL:* the voltage applied to the AREF pin (0 to 5V only) is used as the reference.

Library Functions: Character Functions

Prototype	Description
`int isalpha( int c )`	Returns 1 if c is a letter and 0 otherwise.
`int isalnum( int c )`	Returns 1 if c is a digit or a letter and 0 otherwise.
`int isdigit( int c )`	Returns 1 if c is a digit and 0 otherwise.
`int isxdigit( int c )`	Returns 1 if c is a hexadecimal digit character and 0 otherwise.
`int isspace( int c )`	Returns 1 if c is a white-space character—newline ('\n'), space (' '), form feed ('\f'), carriage return ('\r'), horizontal tab ('\t'), or vertical tab ('\v')—and 0 otherwise.
`int isupper( int c )`	Returns 1 if c is an uppercase letter; 0 otherwise.
`int islower( int c )`	Returns 1 if c is an lowercase letter; 0 otherwise.

Library Functions: Math Functions

- Various Math functions are defined under math.h header file.

Prototype	Description
`double abs (double x)`	Returns absolute value from the given value.
`double max (double x, double y)`	Returns maximum of given 2 values.
`double min (double x, double y)`	Returns minimum of given 2 values.
`double pow (double x, double y)`	Returns the power of x^y.
`double sqrt (double x)`	Returns the Square root of given value.

Library Functions: constrain()

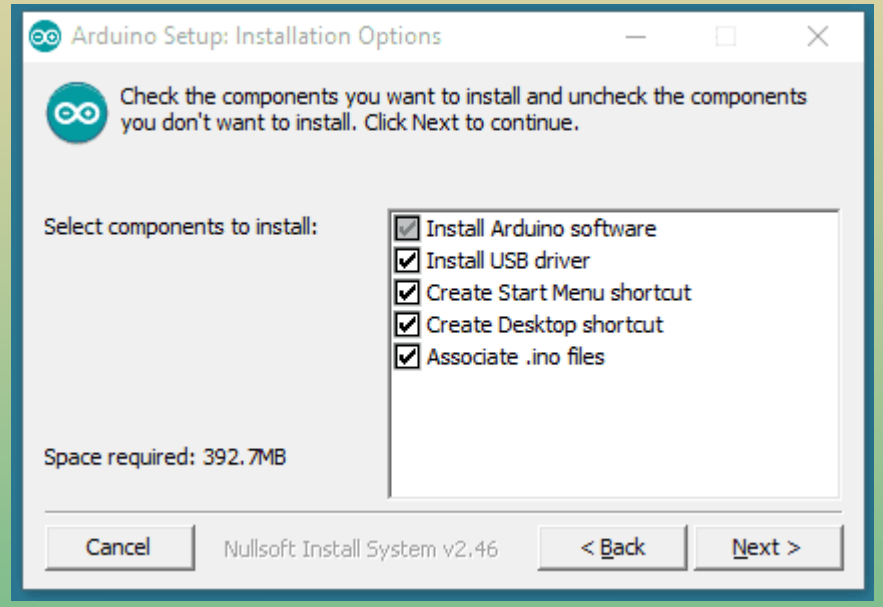
- Constrain function is a very useful function while trying to work with sensors which can give range of values.
- This function helps in constraining the value got into a given range.

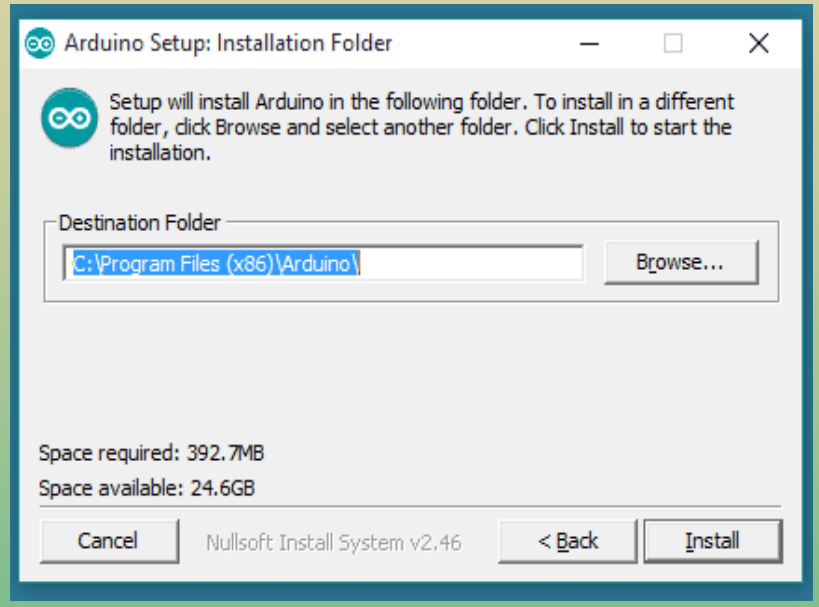
```
constrain(x, a, b)
```

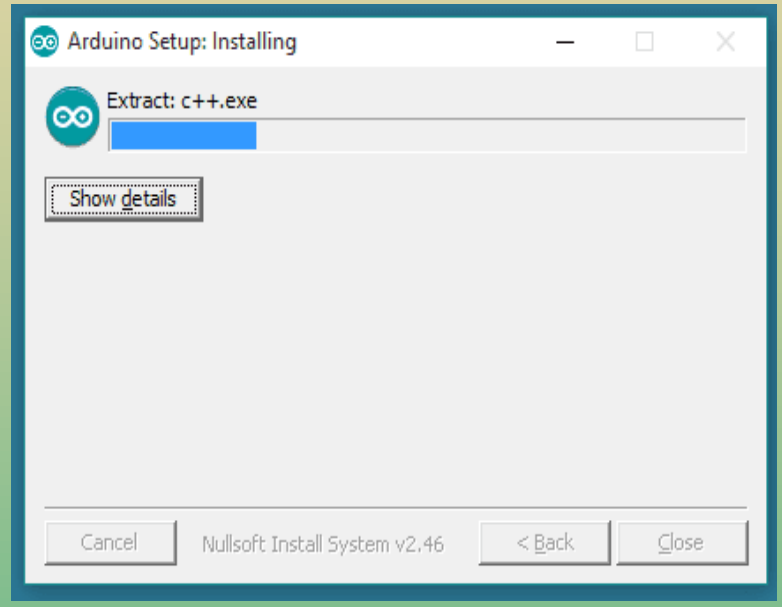
  - x: the number to constrain Allowed data types: all data types.
  - a: the lower end of the range. Allowed data types: all data types.
  - b: the upper end of the range. Allowed data types: all data types.
- Returns:
  - x: if x is between a and b.
  - a: if x is less than a.
  - b: if x is greater than b.

Arduino Programming Setup

- Official IDE of Arduino is available at: https://www.arduino.cc/en/software
- Download and install the setup, which is available for free.



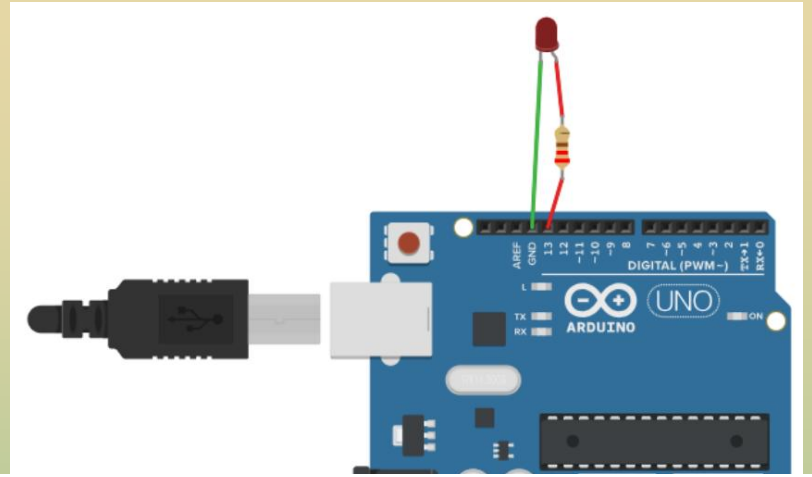


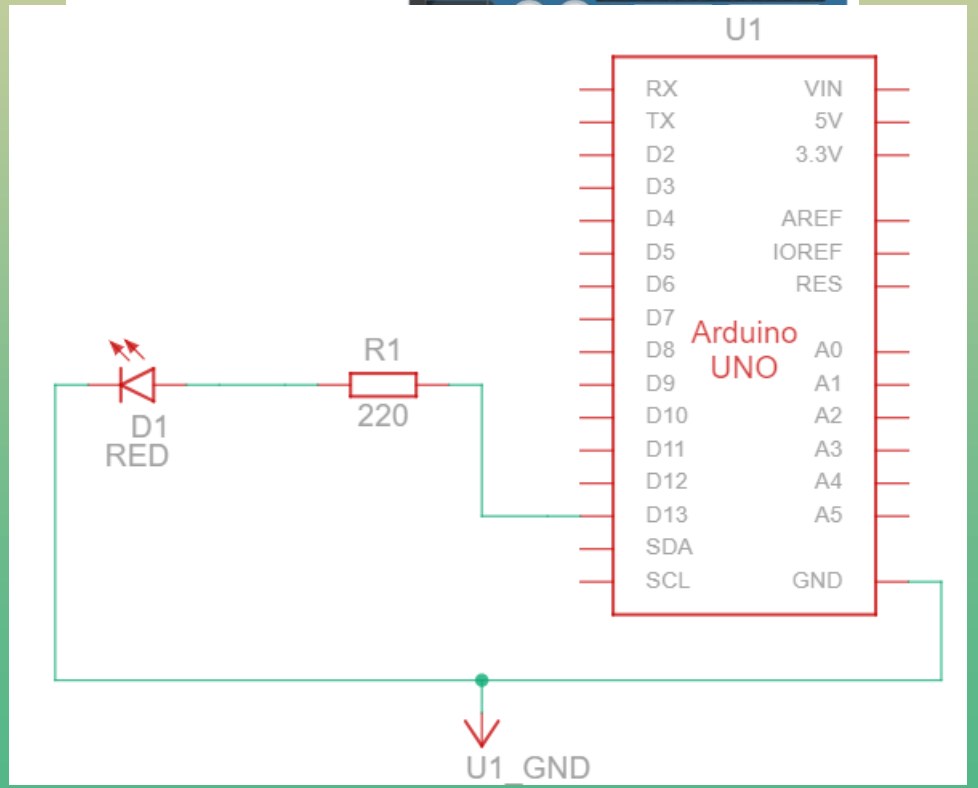


- Arduino sketch extension is ***".ino"***
- After compilation a ***".hex"*** file is generated, which is to be uploaded in Arduino board for execution.

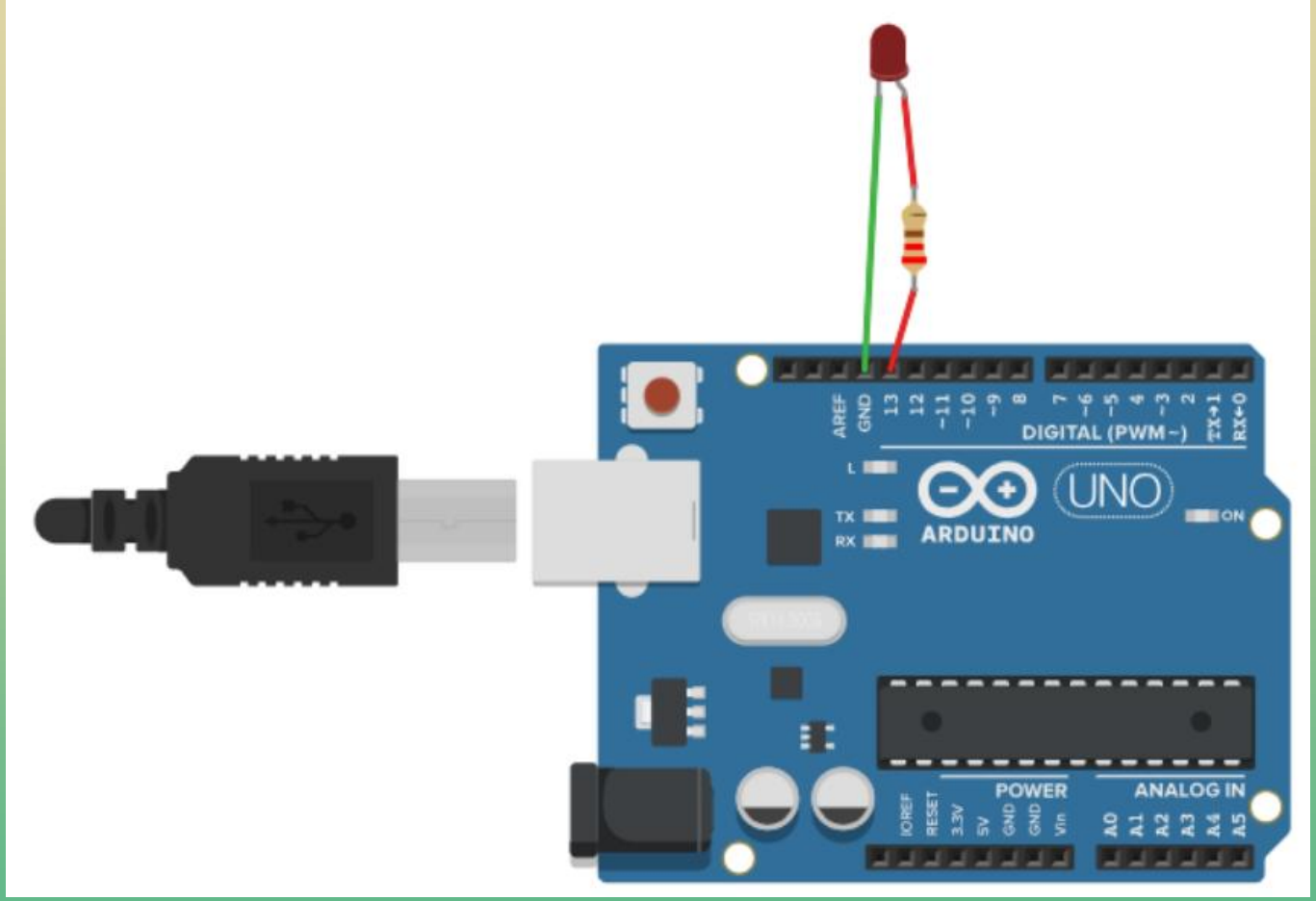
LED Blinking Using Arduino Uno

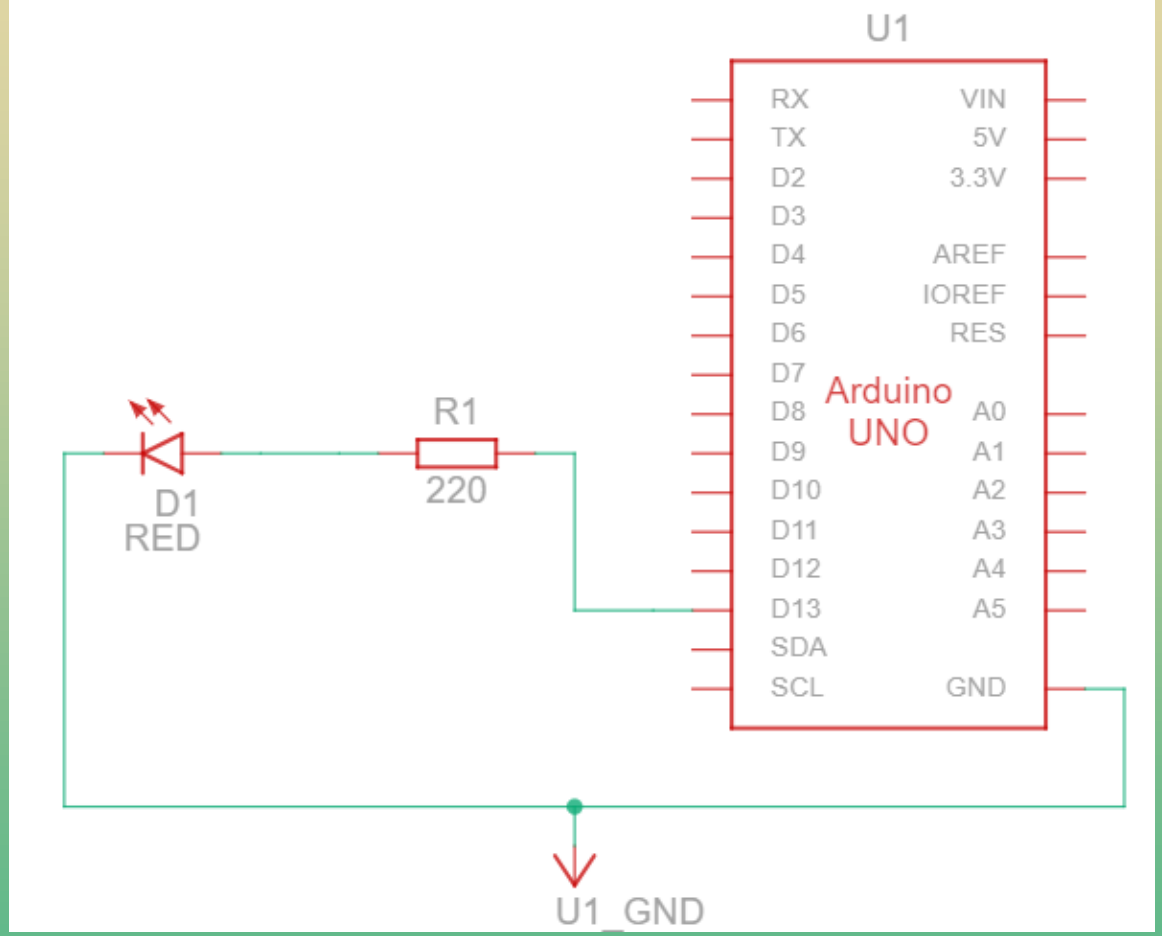
- Equipment needed:
  - Arduino Uno Board – 1U
  - LED - 1U
  - Resistor (~200 Ω) – 1U
- Develop LED Blinking sketch in IDE and compile it.
- Connect Arduino board in Upload mode
- Upload it into Arduino Uno board
- Restart the board in execution mode
- Observe the output





LED Blinking Using Arduino Uno





LED Blinking Using Arduino Uno

```
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT); // LED_BUILTIN = pin no 13.
}

void loop()
{
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000); // Wait for 1000 millisecond(s)
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
}
```

Serial Communication

- A flexible way of communication from Arduino board to a serial port, usually connected to a workstation/computer.
- Before beginning the communication, data rate of communication, baud rate, needs to be negotiated (because of serial communication). Generally its 9600 bps.
- There are other functions to transmit/receive data between Arduino board and Serial terminal.

Serial Communication

- **`if(Serial):`**
  - Used to check availability of the serial communication device.
  - Returns true if the specified serial port is available.
- **`Serial.available():`**
  - Gets the number of bytes available for read in serial communication.
  - Returns the number of bytes available to read
- **`Serial.begin(baud-rate):`**
  - Sets the data rate in bits per second (baud) for serial data transmission.
- **`Serial.end():`**
  - Disables serial communication, allowing the RX and TX pins to be used for general input and output. To re-enable serial communication, call Serial.begin().

Serial Communication

- **`Serial.print(value):`**
  - Prints data (characters) to the serial port as human-readable ASCII text.
  - Returns the number of bytes written
- **`Serial.println(value):`**
  - Prints data (characters) to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n').
  - Returns the number of bytes written
- **`Serial.write(val) or Serial.write(str) or Serial.val(buf, len):`**
  - Writes binary data to the serial port. This data is sent as a byte or series of bytes.
  - Returns the number of bytes written.
- **`Serial.read():`**
  - Reads incoming serial data
  - Returns The first byte of incoming serial data available (or -1 if no data is available).

Serial Communication

- **`Serial.readBytes(buf, len):`**
  - Reads characters from the serial port into a buffer. The function terminates if the determined length has been read, or it times out.
  - Returns the number of bytes placed in the buffer.
- **`Serial.readString():`**
  - Reads characters from the serial buffer into a String. The function terminates if it times out.
  - A String read from the serial buffer.