# Introduction to Technical Programming

# Topic 3:Concepts of programming for single board microprocessor or microcontrollers

**3.1 Introduction to IO on single board computing**

After you have completed this module, you should be able to :

* Expand the term GPIO.
* Compare and contrast some of the major advantages of python compared to other programming languages.
* Explain and identify where the GPIO pins are located.
* Differentiate between a compiler and an interpreter.
* Explain the purpose of the GPIO pins.
* Discuss the major characteristics of the python programming language as an interpreted one.
* Define the term physical computing.
* Differentiate between a shell and an IDE.
* Read and interpret a Pi GPIO Pin guide.
* Define the term physical computing.
* Explain what the terms 3V3, 5V, GND GP2 means on the GPIO board.
* Discuss how Arduino is used to enable physical computing.

**3.2 Visual Programming and solution development**

* Construct (code/write) using the visual tool, debug and run simple programs incorporating: Declaration of variables of different types, use and assignment of values to variables, incorporating program constructs with sequence, selection and iteration structures. Expose and apply various programming concepts as part of the coded solution such as:
  + retrieving remainders: modulus
  + differentiate between real value division and integer division
  + comparison operators and performing logical comparisons
  + incorporate and write code constructs to perform basic calculations such as area, volume, VAT and simple formulae, typical calculations done in other subjects
  + include conditional constructs [if and if then-else] (up to a maximum of two nested levels)
  + Include iteration (looping) structures [fixed counter loop]
  + incorporate a combination of iteration and condition structures as part of the solution (i.e. program code)
* Write code which applies programming language tools and constructs to draw various shapes (turtle type commands) on an output screen/window. Reinforce concepts such as:
  + - Sequence
    - Selection
    - Iteration
    - Creation of objects and shapes
* Design a coding solution to a problem incorporating a combination of different programming constructs which include:
  + Sequence
  + Selection
  + Iteration
* Design and develop solutions for specific problems that include computational thinking and applying software engineering principles.
* Explore lists/arrays (storing and accessing a list of numbers and strings) and containers. (Range: Manipulating lists/arrays such as adding, deleting, replacing, inserting items.)

### 3.1 Introduction to IO on single board computing

**Introduction**

In Level 2, we learnt to program using Python language on a single board computer. We made use of the Raspberry Pi as our single board computing hardware. We then moved to visual programming using Scratch to describe processes through illustrations. In this level, we are still going to use single board computing and our hardware will be Arduino boards and the programming language we are going to use is C++.

Raspberry Pi and Arduino are two very popular examples of single board computing hardware commonly used in among programming and electronics. However, the two boards are quite different. Raspberry Pi serves as a learning tool for computer programming, whereas Arduino is designed for rapid programming and circuit prototyping. There is one major difference between them: Arduino is a microcontroller board, while Raspberry Pi is a microprocessor-based minicomputer. Each board has unique benefits and drawback. We have already learnt the common features of a Raspberry Pi. We will proceed to discuss the features of an Arduino board.

**VOCABULARY**

**A single-board** computer is a fully functional computer with a microprocessor, I/O operations, memory, and other features all integrated onto a single circuit board. There are no expansion slots for peripherals.

### 3.1.1 Exploring the Arduino board

Arduino is an open-source electronics prototyping platform that contains both hardware and software and was founded in 2005 by Massimo Banzi and David Cuartielles. So, to be very clear, Arduino consists of hardware and software. There are two components to Arduino: a physical circuit board (also called a microcontroller) and software, which is a simplified version of the C++ IDE (Integrated Development Environment). The software allows you to develop and upload computer code to the board on your computer. You can load new code into the Arduino using a USB-B cable, as opposed to a separate piece of hardware as with earlier programmable circuit boards. Most users prefer Arduino because they are inexpensive, cross-platform, have a simple and clear programming environment, and are built on open-source and extensible hardware and software.

**Types of Arduino boards**

Numerous Arduino boards have been built throughout the years for thousands of projects, ranging from simple everyday objects to complex scientific apparatus. The list of Arduino boards includes the following:Arduino Uno (R3)

* Arduino Nano
* Arduino Micro
* Arduino Due
* Arduino Mega (R3) Board
* Arduino Robot

The features of different types of the common Arduino boards are listed in Table 3.1.

Table 3. 1: General specifications of common types of Arduino boards

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name of Arduino board** | **Processor type** | **Memory** | **Digital I/O** | **Analogue I/O** |
| Arduino Uno | 16Mhz ATmega328 | 2KB SRAM, 32KB flash | 14 | 6 input, 0 output |
| Due | 84MHz AT91SAM3X8E | 96KB SRAM, 512KB flash | 54 | 12 input, 2 output |
| Mega | 16MHz ATmega2560 | 8KB SRAM, 256KB flash | 54 | 16 input, 0 output |
| Leonardo | 16MHz ATmega32u4 | 2.5KB SRAM, 32KB flash | 20 | 12 input, 0 output |

We are going to discuss the Arduino Uno board as this is one of the more common boards among many users.

**Features of Arduino Uno Board**

Figure 3.1 illustrates an Arduino Uno pin structure.

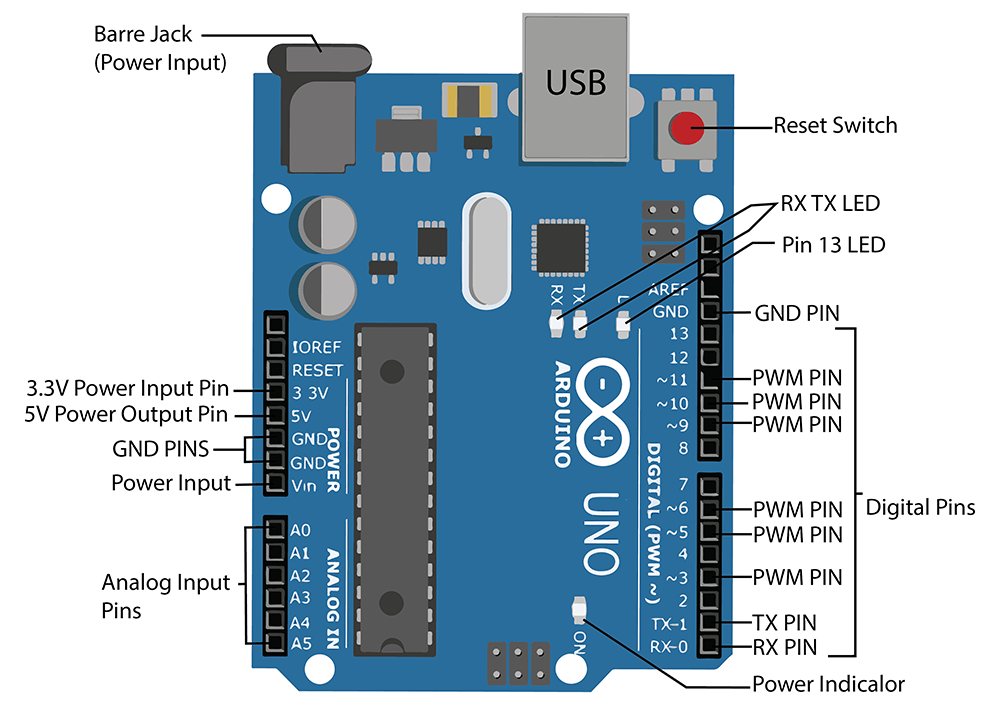


Figure 3. 1: Arduino Uno Pin Diagram

**Power:** A method for connecting to a power supply is required for every Arduino board. There are three ways to power the Arduino UNO and these are: Barrel Jack VIN Pin and USB cable. Use a power supply no higher than 12 Volts to avoid overpowering your Arduino and ultimately destroying it. For most Arduino models, a voltage between 6 and 12 volts is advised.

**PWM Pins:** These pins function normally as digital pins but can also be used for pulse-width modulation. (PWM). They are used as analog output (like fading an LED in and out). PWM pins are of an Arduino are 3, 5, 6, 9, 10, & 11, and gives an output of an 8-bit PWM with the function analog Write ().

**RX – TX**: These pins are used for serial communication and can connect to both computers and other Arduino boards.

**Reset Button:** The code loaded on the Arduino can be restarted using this button.

**Power Indicator LED:** Every time you plug your Arduino into a power source, this LED ought to turn on. There is a good possibility that something is amiss if this light does not turn on.

**Pin 13 LED:** An LED included into the Arduino Uno is wired to digital pin 13. LED turns on whenever the pin is HIGH and turns off whenever it is LOW.

**RX – TX LEDs:** When our Arduino is sending or receiving data on the RX TX Pins, these LEDs will provide us with some excellent visual cues.

**Memory:** The memory of this Atmega328 Arduino microcontroller includes flash memory-32 KB for storing code, SRAM-2 KB EEPROM-1 KB.

**Install the Arduino Desktop IDE and Setup**

As we have earlier mentioned, apart from the Arduino board, to start working with the microcontroller, you will need to install the IDE. Depending on your operating system, choose one of the following links to get step-by-step instructions.

**STEPS**

Step 1: Navigate to the following URL <https://www.arduino.cc/en/software>

Select the operating system of choice. For the purpose of this module, we are going to select Windows.

Step 2: Navigate to the location where the downloaded executable file is and double click it to start the installation process. Accept the licence agreement by clicking on “I Agree”.

Step 3: Select the user from the screen shown on Figure 3.2.

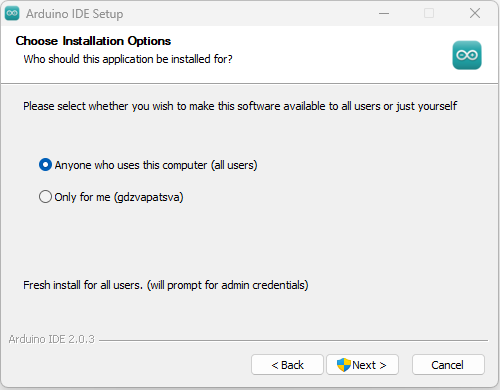


Figure 3. 2: Installation options

Step 4: Select the folder in which you want to install the IDE. The default one is C:\Program Files\Arduino IDE and click next.

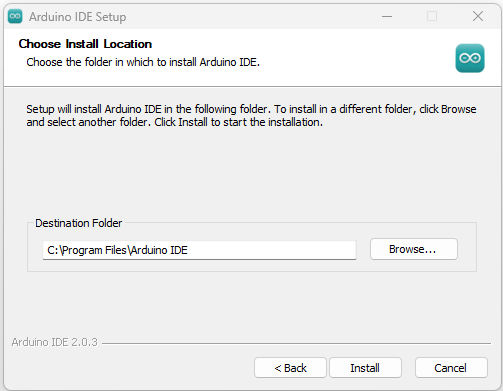


Figure 3. 3: Installation path

Step 5: Installation process goes on and when done click finish on the window as shown on Figure 3.4.

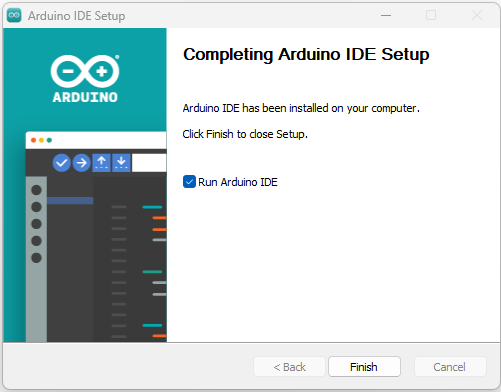


Figure 3. 4: Arduino IDE Setup completion

Immediately, you will see the Arduino IDE as shown in Figure 3.5.

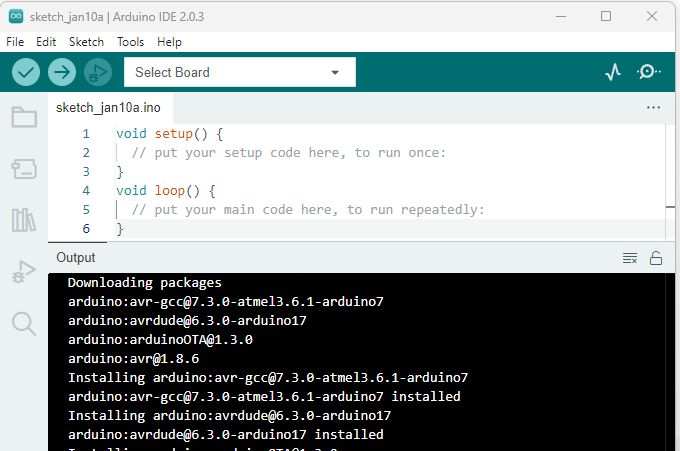
****

Figure 3. 5: Arduino IDE

As seen on Figure 3.5, there are two functions: void setup(){} and void loop(){}

**void setup(){}-** The commands put inside the curly brackets will be executed once. For example the pins used in the program.

**void loop(){}-** The commands that you want to be repeated over and over are put in the void loop. For example, blinking the LED or printing specific output repeatedly.

**VOCABULARY**

A function is a block of organized, reusable code that is used to perform a single, related action.

### 3.1.2 Expand the term GPIO

The term "GPIO" stands for General-Purpose Input/Output. GPIO allows the board to be connected to external components such as sensors. A key feature of the Arduino board is its row of GPIO pins along the top and bottom edges. An Arduino board is physically connected to the outside world via these pins. The simplest way to think about them is as switches, which you can turn on or off (input) or the Arduino can turn on or off (output).

### 3.1.3 Compare and contrast some of the major advantages of python compared to other programming languages

In this section, we are going to compare C++ and Python. Level 2 taught you Python programming, which we can easily refer to as we carry out or compare tasks. Both C++ and Python are popular and widely used **high level** programming languages that offer a wide variety of programming capabilities in addition to being **versatile** and **object-oriented**. Both C++ and Python allow single and multiple inheritance. In terms of syntax, usability, simplicity, and overall programming philosophy, the two programming languages are significantly dissimilar. Python is an interpreted language while C++ is a compiled language . Python is less expensive compared to programs like Java, which is why for most projects it’s the preferred choice. Python is well suited for trending technologies such as **artificial intelligence** and **machine learning**. Python has an extensive **AI/ML libraries**.

Table 3.2 shows the differences between Python and C++.

Table 3. 2: Comparison of C++ and Python

|  |  |  |
| --- | --- | --- |
| **Parameter** | **C++** | **Python** |
| **Extension** | C++ program are saved with .cpp extension. | Python programs are saved with .py extension. |
| **Code** | Tends to have long lines of code. | Python has fewer lines of code. |
| **Structure** | Use curly brackets | Use indentation |
| **Compilation** | C++ is precompiled. | Python is interpreted. |
| **Speed** | C++ is faster once compiled as compared to python. | Python is slower since it uses interpreter |
| **Nature of variables** | C++ is statically typed. You can not declare a variable without a data type. | Python is dynamically typed. You do not need to indicate data type on declaration |
| **Variable scope** | In C++, the scope of variables is limited within the loops. | Accessible even outside the loop. |
| **Memory Management** | C++ does not support automatic memory management (no garbage collector but can be implemented manually) | Python offers automatic memory management (garbage collector) |
| **Functions** | C++ accepts and returns a predefined type of value according to the definition. | There is no limitation on the type of the argument and the type of its return value. |
| **Common Usage** | Emmbedded systems, compilers, Databases eg MySQL, Operating Systems eg Window, Linux, Search Engines, Banking Apps etc | Data Analysis and Machine learning, Web development, Software testing |

The choice of the language depends on the programmer’s level of appreciation for programming. For beginners, Python is usually the most appropriate. However, if you are more worried about speed, C++ is the best. To compensate for Python's fewer desirable features, developers often combine C++ modules with Python. Additionally, calling C++ from Python leads to low-level capabilities.

### 3.1.4 Explain and identify where the GPIO pins are located

GPIO pins are located on the edges of the Arduino board as can be seen in Figure 3.6. We must configure the switch as input if we wish to read the state, sensor data, etc., from it. And we must set it up as an output if we want to control the LED brightness, motor rotation, display of text, etc. The 14 digital IO pins (pin 0 -pin 13) of the Arduino Uno board can be utilized for input and output devices. The analog pins (A0 to A5) can be used as digital IO.

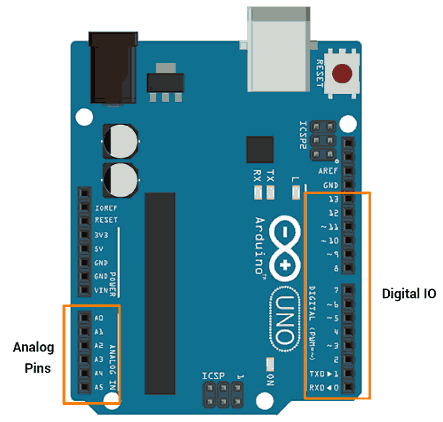


Figure 3. 6: Uno GPIO pin layout

### 3.1.5 Differentiate between a compiler and an interpreter

Compilers and interpreters convert high level languages (source codes) into machine code that computers can understand. Computer programs are usually written in high-level languages that can be understood by humans. In this process, compilers and interpreters are used to translate these source codes into machine language. A compiler translates the entire source code in a single run whereas an interpreter translates the entire source code line by line. A detailed explanation of compilers and interpreters is given in sections 4.2.1 and 4.2.4, respectively.

### 3.1.6 Explain the purpose of the GPIO pins

GPIO stands for "general-purpose input/output." The GPIO pins allow the microcontroller such as Arduino boards, and microprocessor, such as Raspberry Pis to control and monitor the outside world by being connected to electronic circuits. These pins are often used to access sensors, motors, LEDs, etc The number of pins differs with the device and model.

Digital pins on an Arduino can be set up as outputs to power output devices. These pins need to be set up to be used as output. The **pinMode()** function, which determines whether a pin is an input or an output, is used to configure these pins.

Syntax

* **pinMode(pin no, Mode)**
* **pinMode**: This function is used to configure GPIO pin as input or output.
* **pin no**: number of pin whose mode we want to set.
* **Mode**:INPUT, OUTPUT or INPUT\_PULLUP

**e.g.**pinMode (13, OUTPUT)  //set pin 13 as output

Despite the Arduino pins being capable of supplying and sinking current up to 40 mA, it is not sufficient to drive motors, relays, etc.

**NOTE**

Use resistors when connecting devices to Arduino output pins. The Arduino pin or IC will be damaged if any connected device draws more than 40 mA from the Arduino.

Depending on the voltage level, these pins produce HIGH (5V or 3.3V) or LOW (0V) outputs. Using the **digitalWrite ()** function, we can set these pins to output.

* **digitalWrite (pin no, Output value)**

With the digitalWrite() function, we can change the output mode of a pin to HIGH (5V) or LOW (0V).

Output value HIGH or LOW

**E.g**., **digitalWrite (13, HIGH)**

Let us also discuss how we can read data from a device. Digital pins must be set up as inputs to read data from sensors or from any other type of device or circuit. Arduino pins are set to be digital inputs by default. Therefore, there is no need to set the pin as an input. Alternatively, the **pinMode()** function is utilized to set a pin as a digital input. The **digitalRead()** function can be used to read data from GPIO pins.

### 3.1.7 Discuss the major characteristics of the python programming language as an interpreted one

Python is a high-level, interpreted, dynamic programming language that is open source. In addition to object-oriented programming, it also supports procedural programming. Python is a dynamically typed language, so we don't need to declare variable types. This basically means that the type of a variable is determined during execution rather than beforehand. Python is highly portable across platforms. Furthermore, because Python is written in C, it allows the execution of code written in other programming languages such as Java, C, and C# by default, making it one of the most adaptable in the industry.

### 3.1.8 Define the term physical computing

**VOCABULARY**

Physical computing is the interaction of hardware devices and software programs with the real world.

Physical computing also provides developers an opportunity to explore the hardware aspects (electronics) along with the software aspects (computational skills). Among the well-known physical computing platforms are Arduino and Raspberry Pi. The beauty of physical computing is that it allows us to interact with the real world by attaching a microcontroller or microprocessor to sensors and motors.

Several physical computing systems appear to be automated. Picture an automated door. When a sensor detects motion in front of the door, it opens. We might overlook the fact that the software in between controls the door and the sensor. This software may alter the sensor's sensitivity; for instance, it may ignore or deactivate the sensor during off-peak hours, functioning as a lock. Remember the projects that we did in level 2 using the Raspberry Pi? For instance, with the blinking light, we were able to change the delay for the blinking LED.

### 3.1.9 Differentiate between a shell and an IDE

**VOCABULARY**

Shell – Is the program that avails the command line to enable users to interact with hard ware by typing in commands on the keyboard; exposes the services of an operating system to a human user or other programs.

A shell processes commands entered through a command-line interface, allowing people to communicate with computers by giving them instructions. Examples of shells include, but are not limited to, MS-DOS Shell (command.com), csh, ksh, PowerShell, sh, and tcsh. Figure 3.7 illustrates the structure of a computer shell. A computer user makes a request to the operating system through the shell. The shell then checks for the validity and correctness of the instruction before passing it to the kernel to be executed. In simpler terms, the shell acts as a command interpreter.

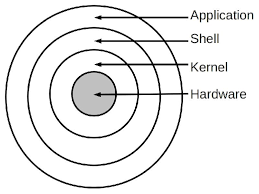


Figure 3. 7: Computer shell

To start a shell in Windows, type the word shell on the search area and select the option Windows shell. You will see a screen as shown in Figure 3.8. Any of the Window’s commands which we covered in level 2 works well on the PowerShell.

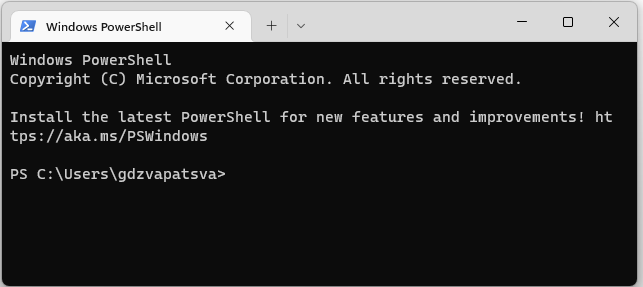


Figure 3. 8: Windows PowerShell

Test the following command to refresh your memory and explain to your friend what each does.

* cd
* mkdir
* cls
* dir
* date
* echo : | TIME

**VOCABULARY**

IDE is a software application that provides a comprehensive facility to computer programmers for software development. IDE normally consists of:

* compiler
* Source code editor
* A debugger

IDEs come with an integrated user interface that combines everything a developer needs to write code conveniently. IDEs are designed to encompass all programming tasks in one application. One of the main benefits of an IDE is that it offers a central interface with all the tools a developer needs, including a code editor, compiler, and debugger. Because several tools don't need to be manually configured and integrated as part of the setup process, an IDE enables developers to begin developing new apps rapidly. Some of the common examples of IDEs are VSCode, PyCharm, Atom, Eclipse, NetBeans, Komodo, etc. A detailed discussion on IDEs is later explained in Section 4.3.2.

### 3.1.10 Read and interpret a Pi GPIO Pin guide

**Arduino Digital pins**

As we've already discussed, the Arduino Uno is the most common board on the market and best suited for beginners. The board can be directly connected to the computer using a USB cable, which serves as both a serial port and a power source. As can be seen in Figure 3.9, there are 14 digital pins on an Arduino Uno board. They are easily recognizable, from 0 to 13 on the circuit board on the right-hand side. As highlighted earlier, digital pins are used to read data from sensors and write data to other components (actuators). A digital pin can only be in one of two states: HIGH or LOW. You may think of them as binary pins. LOW denotes a pin with a voltage of zero volts, and HIGH refers to the Arduino Uno's 5V (volts) supply.

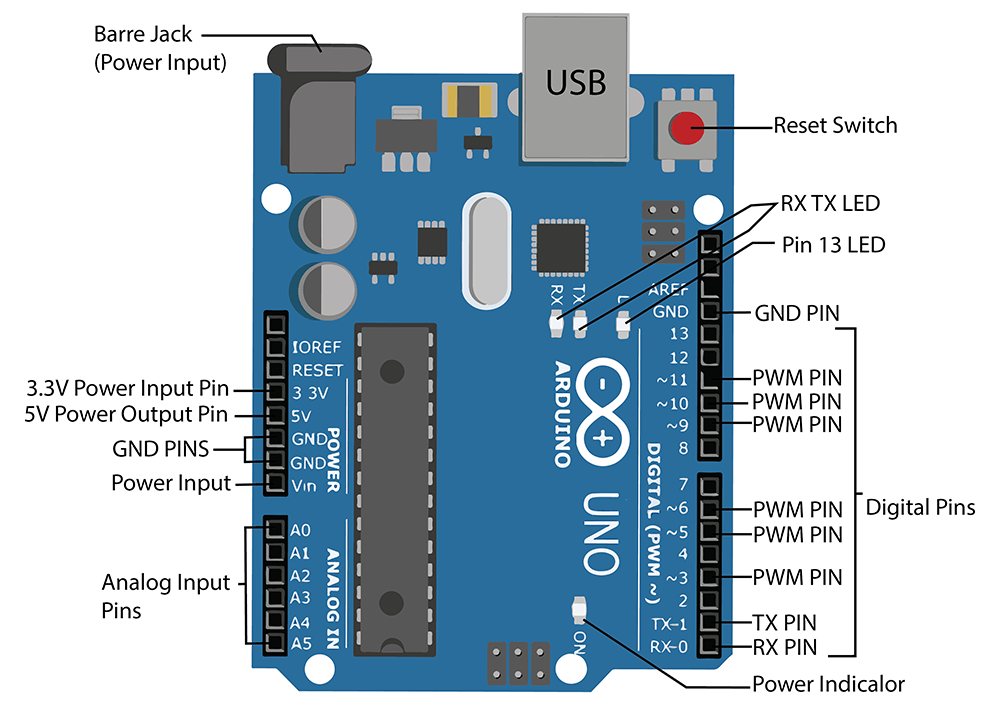


Figure 3. 9: Arduino Pin Guide

**Pulse Width Modulation (PWM) Pins**:. With a PWM pin, you can get a specific voltage (ex: 4.1V) with only HIGH/LOW (5V/0V) states. On the Arduino board they are also easily recognised with a “~” next to their number for example on pin 3,5,6,9,10,and 11

**Arduino Analog pins**

The Arduino Uno has 6 analog pins which are located near the power pins and marked A0-A5. Analog pins are useful when reading values other than 0 or 1 for instance for instance a potentiometer. All the analog input pins provide a 10-bit resolution Analog to Digital Converter (ADC) feature, which can be read using analogRead() function.

* analogRead (pin)-This function is used to read analog value from specified analog pin.

pin - number of analog pin which we want to read

returns - digital value 0 – 1023

To use an analog pin as a digital pin, you simply have to set the mode for the pin, as you would do for digital pins in the setup() function of your Arduino program. Then, you can use the digitalWrite() and digitalRead() functions and it will work perfectly.

**VOCABULARY**

Potentiometer-A position sensor that can measure displacement in any direction.

**WARNING**

When you connect your Arduino board to your computer, be sure not to let it touch any metal surface as it could shot out the Arduino and render it useless.

So far, we've talked about the Arduino board and a little bit about the IDE. It’s now time to connect our Arduino board and start our programming. Now, we want to hard code the program. We are going to start with a simple program to turn ON the onboard LED.

**Turning LED ON Project**

You will need:

• Arduino Uno R3 board.

**NOTE**

If you bought, the Arduino starter pack, it comes with lots of components but for our first project we will use the Arduino board only with no components. Another important point is that pin 13 comes preconfigured as a built-in LED. Once we connect the board, you will see it blinking.

**INSTRUCTIONS**

1. Let us start our IDE by typing Arduino in the search area from the taskbar. Immediately you will see a screen as shown in Figure 3.10. I have already renamed my project to Exercise1\_LED.ino.

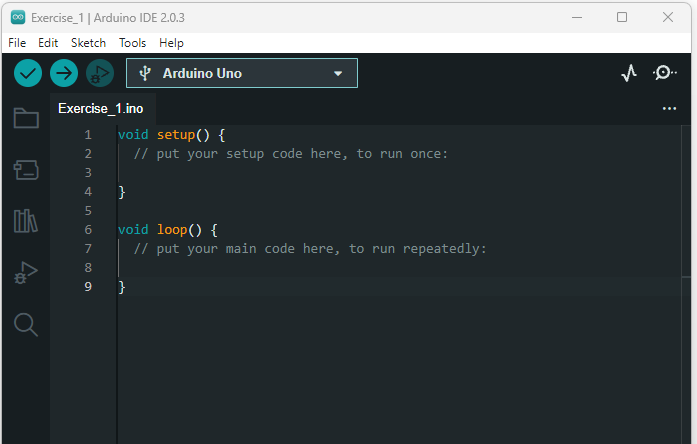


Figure 3. 10: Arduino IDE

2. Take the Arduino board and the USB-B cable and connect it to the computer. Immediately you should hear a sound to show that the board is responding. A green light will appear to show that the board is on.

3. Now let’s make sure that our IDE is talking to the Arduino board. Click on Tools select the port . You should see a port selected in my case it shows COM4 as shown in Figure 3.11. Port names might differ.

Graphical user interface, application

Description automatically generated

Figure 3. 11: :Port selection

1. Select Tools from the menubar and select board. Under board choose Arduino AVR boards and select the specific board which you are using. In our case we select Arduino Uno. Immediately the board name and port will reflect as in Figure 3.12.

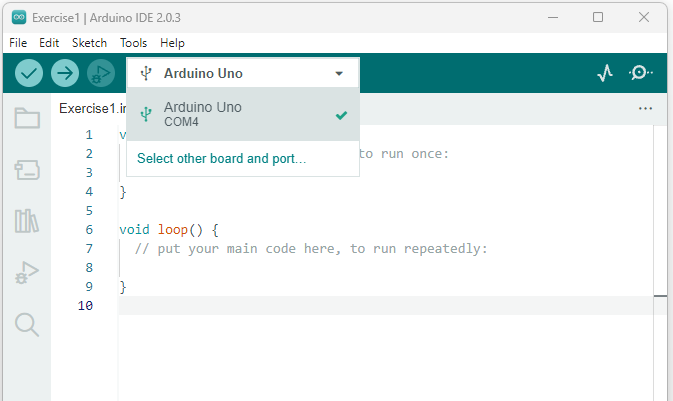


Figure 3. 12: Board type and port name

1. Setup the pin that will be used for this program inside the void setup() function using pinMode. We mentioned that we are going to use pin 13, which is hooked up to the onboard LED. The code will look like this:

void setup() {

  // put your setup code here, to run once:

  pinMode(13, OUTPUT);

}

1. Now let us go to the void loop() function and turn on the LED by setting it to HIGH.

The code looks as follows:

void loop() {

  // put your main code here, to run repeatedly:

digitalWrite(13, HIGH);

}

1. Now our code is still in our IDE We need to send it to the Arduino board. The full code looks as follows:

void setup() {

  // put your setup code here, to run once:

  pinMode(13, OUTPUT);

}

void loop() {

  // put your main code here, to run repeatedly:

digitalWrite(13, HIGH);

}

1. Click upload arrow below Edit tab. Alternatively, press the control key and U, or go to Sketch on the menu bar and select Upload.
2. In the output section of the IDE, you will see the compilation status and immediately the LED light on the board Turns ON. Congratulations on successfully running your first program on the Arduino Uno R3 board. Let us analyse our code.

void setup() {

  // put your setup code here, to run once:

  pinMode(13, OUTPUT);

}

We already know that the void setup() function runs only once at the beginning of our code. Within the function, inside the curly braces, we used a command called pinMode(), which takes two parameters, i.e., the pin number and mode. So, we can tell our program that we are using the pin for OUTPUT or INPUT. OUTPUT means that this pin will produce a result like turning on or off an LED. INPUT is used to prepare the pin to receive information from a connected device, like a sensor. So, we used pin 13 and set it to OUTPUT.  Take note that the word OUTPUT is in uppercase. The statement is terminated with a semi-colon to indicate the end of the statement. Missing a semi-colon will result in an error which will appear in the output section as follows:

Compilation error: expected ';' before '}' token

exit status 1

**Curly braces{}:** Are used to define the beginning and the end of functions and certain statements.

**//**- represent line comment. Any text next to // is not executed. It is a good practice to tell other people what each piece of code is doing.

void loop() {

  // put your main code here, to run repeatedly:

digitalWrite(13, HIGH);

}

Within the curly braces of the void loop() we used one statement: digitalWrite(), and it takes two parameters, i.e. pin and mode.

**digitalWrite()** changes the status of a pin by either writing a 5V (high) or 0V (low): In our case, we set pin 13 to HIGH to activate the LED that is connected to it.

**DID YOU KNOW**

You can change the theme of the IDE to one of several options by going to File, selecting Preferences, and then selecting your preferred theme under Settings. I changed my IDE theme to Dark( Arduino).

As we do our programming on Arduino, there are common programming principles that we need to understand such as variables and operators. While these are repeated again in Topic 6, it will be good reinforcement. Let us start by defining variables.

Variables are memory location that hold data that can be changed during project execution important. Variables assist in the assignment of a data type which determines how much space they occupy in the storage and how the bit pattern stored is interpreted. Memory is set aside for storing the variable and the variable is given a name that allows us to access it in the sketch at a given time.

Here is an example of a variable declaration:

int x =13;

int is the data type, x is the variable name and holds a value 13.

**NOTE**

Values of variables can change.

**Rules for naming variables**

* Variables can consist of both uppercase (A-Z) and lowercase(a-z) letters.
* Variables can contain numbers 0 to 9, but cannot start with a number.
* Variables may not have the same names as Arduino language keywords, e.g. you cannot have a variable named float.
* Variables must have unique names, i.e. you cannot have two variables with the same name.
* Variable names are case sensitive, so **Count** and **count** are two *different* variables.
* Variables may not contain any special characters, except the underscore (\_)

**Arduino Data Types**

There are different data types which can be used in Arduino programming, and these are:

* void Data Type
* int Data Type- Store integer numbers
* Char Data Type- stores any number of characters set. e.g 52
* Float Data Type- A floating-point number is one that has both a fractional and a decimal component e.g., 5.6.
* Double Data Type- The double data type is also used for handling the decimal or floating-point numbers.
* Unsigned int Data Type-Stores only positive numbers
* short Data Type- A short is a 16-bit data-type.
* long Data Type-
* Unsigned long Data Type
* byte data type- stores values from 0 to 255.
* word data type- It is regarded as an unsigned 16-bit or 2-byte number that may hold values between 0 and 65535.

Table 3.3 summarises the different data types as used in Arduino programming.

Table 3. 3: Data Types in Arduino

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size in Bytes** | **Explanation** |
| int | 2 Bytes | Stores a 2 byte(16 bits) signed integer value that is in range of -32,768 to 32,767. |
| char | 1 Byte | The char datatype can store any number of character set. The literals are written inside a single quote like ‘a’, ‘#’ etc and their ASCII numerical is stored at corresponding variable location. |
| Float | 4 Bytes | Stores a signed 4-byte(32-bit) value that is integer or a value with decimal point (say 12.15) that is in range of -3.4028235E+38 to 3.4028235E+38. |
| Double | 4 Bytes | The double data type is also used for handling the decimal or floating-point numbers. |
| Unsigned int | 2 Bytes | Stores an unsigned integer value of 2 bytes(16 bits) that is in range of 0 to 65,536 |
| byte | 8 bits | It is considered as an unsigned number, which stores values from 0 to 255. |
| short | 2 bytes | The short is an integer data type that stores two bytes or 16-bit of data. |
| long | 4 Bytes | Stores a 4 byte (32 bit) signed integer value that is in range of -2,147,483,648 to 2,147,483,647 |
| word | 2 bytes, | It is considered as an unsigned number which stores values from 0 to 65535. |
| void |  | The void keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called. |

Let us do a practical on data types. We will declare the common data types and print them. First, we are going to introduce two key commands:

* Serial.begin()
* Serial.print()

**Serial.begin(9600): This function initiates serial communication between the microcontroller (Arduino board) and an additional device.** The most common reason to use Serial.begin() is when you want to output some information from your Arduino to your computer screen. Majority of the time, you’ll be putting the Serial.begin() function inside of the setup() function. 9600 is called the **baud rate but can be changed.**

Serial.print(): This command is used to display text on the computer screen from the Arduino board. Serial.print() and Serial.println() are the same. The only difference is that with println(), anything new that is printed gets printed in the next line, i.e., a new line is formed. Here is an example of code and the output shown in Example 3.1.

**EXAMPLE 3.1**

void setup() {

  Serial.begin(9600);

// using print function

Serial.print("Arduino");

Serial.print("Programming");

Serial.println();

Serial.println("Arduino");

Serial.println("Programming");

}

void loop() {

  // put your main code here, to run repeatedly:

}

OUTPUT

ArduinoProgramming

Arduino

Programming

The first two print functions resulted in the text being printed in one line. The first println() function caused the cursor to jump to the next line and the words "Arduino" and "Programming" have been printed in two separate lines.

 Here is another example of printing number continuously :

**EXAMPLE 3.2**

int counter=1;

int wait=5000;

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

}

void loop() {

  // put your main code here, to run repeatedly:

  Serial.print("Counter = ");

  Serial.println(counter);

  counter=counter +1;

  delay(wait);

}

In the above code, you will notice how we implemented the use of Serial.print() and Serial.println().

Also of importance, one might wonder how to see the output. This is accomplished by compiling and uploading the project to Arduino, going to the Tools menu bar and selecting Serial Monitor. Your output will appear in the output section of the IDE.

**EXAMPLE 3.3**

int a = 3; //declaring a integer

float age=18; //declaring a float

char myvariable= 66; //declaring a character

byte number1 = 20;

long idnumber=1235478; //declaring a double

bool running=false;

void setup( ) {

 Serial.begin(9600);

 Serial.println(a);

 Serial.println(age);

 Serial.println(myvariable);

  Serial.println(number1);

 Serial.println(idnumber);

  Serial.println(running);

}

void loop ( )

{

  }

OUTPUT

3

18.00

B

20

1235478

0

**NOTE**

We declared variable age and assigned value18, but the printout was 18.00 because float will use decimal point. Also, for Boolean, the value of running is false, but the printout is 0. 0 is false and 1 is true. myvariable printed the equivalent ASCII code.

**Scoping**

A very important subject we need to make sure, you fully understand is called the variable scope. A scope is an area where the variable is declared and accessible. We are going to discuss two scoping techniques which are:

* Local variables
* Global variables

**Local Variables**- these are declared inside a function or block. They can be used only by the statements that are inside that function or block of code for example 3.4:

**EXAMPLE 3.4**

void setup() {

  Serial.begin(9600);

// using print function

int x=5;

int y=10;

Serial.println(x);

Serial.println(y);

}

void loop() {

}

OUTPUT

5

10

If we modify the program above and put the println() commands in the void loop() function as below, we will get an error.

void setup() {

  Serial.begin(9600);

// using print function

int x=5;

int y=10;

}

void loop() {

Serial.println(x);

Serial.println(y);

}

Error Message

Compilation error: 'x' was not declared in this scope

**Global Variables**

Global variables are ones whose influence is throughout the program. Global variables are defined outside of all the functions, usually at the top of the program. The global variables will hold their value throughout the life-time of your program. Global variables are declared before the void setup() function and can be used throughout program execution.

**Constant**

**VOCABULARY**

A constant is a predefined expression whose value remains the same.

To declare a constant use keyword const, datatype and value as follows:

const int z=7;

This will render z as read-only. Any attempt to re-assign a value will result in the following error message.

Compilation error: assignment of read-only variable 'z'

Instead of using the word const, we can do the same by using # define.

Syntax

#define z 7

When using #define, you do not need to add semi-colon. Also, #define does not take up any memory space on the chip. Because the const keyword adheres to the variable scope's restrictions, it is preferred over the #define keyword.

**Note**

We can either use const or #define in the case of strings and numeric constants. But we can only use const for arrays.

**Operators**

An operator is a symbol used in the evaluation of operations such arithmetic, logical boolean e.t.c. Arduino has five categories of operators:

* Arithmetic operators
* Comparison operators
* Boolean operators
* Compound operators
* Bitwise operators

Arithmetic Operators- There are six basic operators responsible for performing mathematical operations in Arduino, which are listed below:

**Assignment operator** – Uses the = sign. This is not an equal to operator.

**Example**

int a = 3;

int b= 7;

a=b;

The new value of a when printed will be 7.

We will continue to assume that a=3 and b=7 for the rest of the arithmetic operator examples.

**Addition (+)-**Adds two operands

Example

int c=a+b;

The value of c is 10.

Subtraction(-): Subtracts second operand from the first.

 int c=a-b;

The result is -4.

Multiplication (\*)- Multiply both operands

 int c=a\*b;

The value of c is 21.

Division (/)-Divide numerator by denominator

 int c=a/b;

The output is 0 because 3/7 is 0.428…

Even if we change the data type c to float, the answer will still be 0.00.

Modulo (%): The Modulo operator is used to calculate the remainder after the division

int c=a % b;

The value of c will be 3 if the data type was int or 3.00 if the data type is set to float.

Here is the sequence of actions the Arduino considers when completing calculations:

1. Parentheses ( )
2. Multiplication, division, and modulo
3. Addition and subtraction

**Example 3.5**

int a=2;

int b=4;

int c=6;

int z;

void setup ( )

{

Serial.begin( 9600 );

z=  b \* c / (b + a)  + 8;

Serial.println(z);

}

void loop ( )

{

}

Let us break it step by step.

Step 1:

Z=4 \* 6 / (4 +2 ) +8

Step 2

Z=4 \* 6 / 6+8

Step 3

Z=24/6 +8

Step 4

Z=4+8

Z=12

**Comparison Operators**

As the name implies, comparison operators are used to compare operands.

Types of comparison operators

1. **equal to** (==):It checks the value of two operands. If the values are equal, the condition is satisfied.

void setup ( )

{

  int num1=25, num2=33;

Serial.begin( 9600 ).

Serial.println(num1==num2);

}

The output is 0 meaning its false.

1. **not equal to** (!=):Checks whether the values of two operands are equal; if the values are not equal, the condition is true.
2. .

  int num1=25;

  int num2=33;

Serial.begin( 9600 ).

Serial.println(num1!=num2);

Output

1

It is true that 25 is not equal to 33

1. **less than(<):** The less than operator checks that the value of the left operand is less than the right operand. The statement is true if the condition is satisfied.

int num1=25;

int num2=33;

Serial.begin( 9600 );

Serial.println(num1<num2);

The output is 1 meaning its true that 25 is less than 33.

1. **Greater than (>):** The greater than operator checks that the value of the left side of a statement is greater than the right side. The statement is true if the condition is satisfied.

  int num1=25;

  int num2=33;

Serial.begin( 9600 );

Serial.println(num1>num2);

The output is 0 meaning it is false that 25 is greater than 33.

1. **less than or equal to ( < = ):** Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

  int num1=25;

  int num2=33;

Serial.begin( 9600 );

Serial.println(num1<=num2);

The output is still a 1 because 25 is les than 33.

1. **greater than (>=):**Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.

int num1=25;

int num2=33;

Serial.begin( 9600 );

Serial.println(num1>=num2);

The result for the above snippet is false hence it prints a 0.

**Boolean Operators**

In a search, Boolean operators are symbols used in combine expressions or excluding outcome from another expression.

Types of Boolean operators are:

* **NOT ( ! )**
* **Logical AND ( & & )**
* **Logical OR ( | | )**.

1. **Not (!):** Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make the result false.

int num1=0;

Serial.begin(9600);

Serial.println(!num1);

The output is 1 since not 0 is 1

1. **And (&&):** If both the operands are non-zero then then condition becomes true.

  int num1=3;

  int num2=7;

  Serial.begin(9600);

 Serial.println(num1&&num2);

Output is a 1 which equates to true. If we change the value of any of the operands to 0, the output will be 0 which is false.

1. **Logical Or (||)** : The result of the condition is true, if either of the variables in the condition is true.

  int num1=0;

  int num2=1;

  Serial.begin(9600);

 Serial.println(num1|| num2);

0 or 1 is 1. So, in this case the output is 1. If we change the value of num1 say to value 5, the result is still 1. For more understanding, revisit truth tables on the following link:

**e-Link**

<https://www.techtarget.com/whatis/definition/truth-table>

**Compound Operators**

The compound operators perform two or more calculations at once with the result of the right operand being assigned to the left operand. Let’s assume we have two variables num1 and num2 with assigned values of 5 and 1 respectively. We will use these values in our explanation of the different types of compound operators.

1. Increment : Increment operator, increases integer value by one

 int num1=5;

  int num2=1;

num1++;

  Serial.println(num1);

}

The output is a 6.

1. **decrement operator**: Decrement operator, decreases integer value by one.

int num1=5;

int num2=1;

Serial.begin(9600);

num1--;

Serial.println(num1);

The output of the above program is 4.

1. **compound addition**- It adds right operand to the left operand and assign the result to left operand. num2 += num1 is equivalent to num2 = num2+ num1.

int num1=5;

int num2=1;

Serial.begin(9600);

num2+=num1;

Serial.println(num2);

The output of the above code snippet is 6.

1. **compound subtraction**: . It subtracts right operand from the left operand and assign the result to left operand, num2 += num1 is equivalent to num2 = num2- num1.

int num1=5;

int num2=1;

Serial.begin(9600);

num2-=num1;

Serial.println(num2);

The output is -4.

1. **compound multiplication**: It multiplies right operand with the left operand and assign the result to left operand. num2\*= num1 is equivalent to num2 = num2\* num1.

int num1=5;

int num2=1;

Serial.begin(9600);

num2\*=num1;

Serial.println(num2);

The output of 5 x 1 is 5.

1. **compound division**: It divides left operand with the right operand and assign the result to left operand. num2/= num1 is equivalent to num2 = num2/ num1.

int num1=5;

int num2=1;

Serial.begin(9600);

num2/=num1;

Serial.println(num2);

The output is 0 since 1/5 =0.2. But because the / is for integer division, it takes the integer part only.

1. **compound modulo**-  It takes modulus using two operands and assign the result to left operand. num2%= num1 is equivalent to num2 = num2% num1.

int num1=5;

int num2=1;

Serial.begin(9600);

num2%=num1;

Serial.println(num2);

Take note of the answer here which is 1. 1%5 is 0 remainder 1.

**Bitwise Operators**

Bitwise operations entail dealing with bits, small data units. The binary value of each bit is either 0 or 1. There are various bitwise operators. Some of the popular operators are listed below:

1. **bitwise NOT ( ~ ):**

  int a = 60;

Serial.begin(9600);

Serial.println(~a);

The answer is -61

1. **bitwise and (&):** If the bit is present in both operands, the binary AND operator duplicates it to the result. Let’s assume we have two operands a and b with the following values 34 and 15 respectively.

34 in binary is 100010

15 in binary is 001111

int a = 34,b = 15;

int c;

Serial.begin(9600);

  c=a&b;

Serial.println(c);

The output is 2 equivalents to 000010.

1. **Bitwise Or (|):** If a bit is present in either operand, the binary OR operator copies it. Let us look at the code snippet below:

  int a = 34,b = 15;

 Serial.begin(9600);

 Serial.println(a|b);

The result in binary is 101111.The answer in the output section is 47.

1. **bitwise left shift (<<):** The left operator is shifted by the number of bits defined by the right operator.

int a = 34;

Serial.begin(9600);

Serial.println(a<<2);

34= 1000102

The output is 136. 136=100010002

1. **bitwise right shift:** The right operator is shifted by the number of bits defined by the left operator.

int a = 34;

 Serial.begin(9600);

 Serial.println(a>>2);

34= 1000102

The output is 8. 8 =10002

If we change to a>>3, the output would be 4.

So now we have discussed variables and data types. You will notice that we have been using hard coded data. How about using input from the keyboard? Accepting input entails three things: we enter a message to accept, wait, and accept the input. This is demonstrated in Example 3.6.

**EXAMPLE 3.6**

//declaring the three variables

int num1;

int num2;

String message2=" Num1 is";

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

}

void loop() {

  // put your main code here, to run repeatedly:

Serial.println("Please enter first number: ");

while ( Serial.available()==0){}

num1=Serial.parseInt();

Serial.print("Num1 is ");

Serial.println( num1);

Serial.println("Enter second number ");

while ( Serial.available()==0){}

num2=Serial.parseInt();

Serial.print("Num2 is ");

Serial.println( num2);

delay(500);

while (Serial.available()==0){}

}

00:41:31.464 -> Please enter first number:

00:41:37.293 -> Num1 is 6

00:41:37.331 -> Enter second number

00:41:40.542 -> Num2 is 7

We are going to do more practice in section 3.2.

**Blinking LED**

We want to set our LED in such a way that it blinks by turning ON and OFF repeatedly. For this exercise we will introduce the delay() command. You will notice that in the previous example, if we just alternate the digitalWrite() to HIGH and LOW, it will blink so fast that we won’t even notice it. The solution is to introduce a delay after every state.

**delay():** The next statement's execution is postponed by the specified amount of milliseconds using the delay() function. For instance,  delay(1000) means delay the execution of the next statement by 1000 milliseconds or 1 second.

We will also need to set the digitalWrite() command in the void loop() function to LOW. Our code will loop as in example 3.7:

**EXAMPLE 3.7**

void setup() {

  // put your setup code here, to run once:

  pinMode(13, OUTPUT);

}

void loop() {

  // put your main code here, to run repeatedly:

digitalWrite(13, HIGH); //Turning the LED ON

delay(3000); //Delaying by 3 seconds

digitalWrite(13, LOW); //Turning the LED OFF

delay(3000); //Delaying by 3 seconds

}

### 3.1.11 Explain what the terms 3.3V, 5V, GND GP2 means on the GPIO board

**Vin:**This is the input voltage pin of the Arduino board used to provide input supply from an external power source. It is located just above the analog pins.

**GND stand for Ground.** Arduino Uno R3 board which we are using for this book has 2 GND pins. This pin of the board is used to ground the Arduino board. To prevent user contact with excessive voltages, electrical equipment is grounded by connecting it to the ground.

**5V:**This pin of the Arduino board is used as a regulated power supply voltage, and it is used to give supply to the board as well as onboard components.

**3.3V:** This pin of the board is used to provide a supply of 3.3V which is generated from a voltage regulator on the board.

### 3.1.12 Discuss how Arduino is used to enable physical computing

A physical computing environment is a very effective way to introduce or reinforce programming concepts. It also allows students to explore how the concept can be applied to applications that are not based on a screen. However, it has also been used outside of education by artists, engineers for instance to test water quality and so on. In our case, to connect the Arduino to the outside environment or components, we will need to hook it up through a **solderless** breadboard. You will recall how we used a breadboard with a Raspberry Pi in level 2 programming. On a solderless breadboard, metal strips are enclosed in plastic with a grid of holes called tie-points. We can build circuits quickly without soldering the components permanently (for example). Figure 3.13 shows a picture of a breadboard.

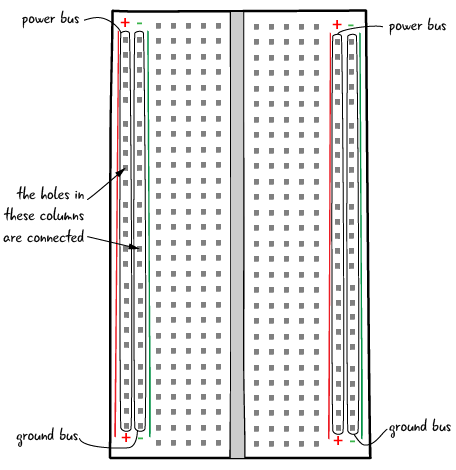


Figure 3. 13: Breadboard

To refresh your memory on using a breadboard, go through the tutorial in the link below:

**e-Link**

<https://www.youtube.com/embed/CfdaJ4z4u4w>

**Connecting an external LED to the Arduino**

In this exercise, we need to connect an LED on to the breadboard and link it with our Arduino and make it blink.

You will need:

* Arduino Uno R3 board
* Breadboard
* Two jumper cables. (I recommend using different colours)
* An LED
* 220 Ohm resistor

**INSTRUCTIONS**

* 1. First make sure that the Arduino is powered off (no USB cable plugged to anything).
  2. Check the LED, you will see that one of the leg is shorter than the other one. This is the cathode and it connects to the ground GND via the jumper cable (black cable preferred for GND). (Use roe 6 all in column E)
  3. Connect the longer leg of the LED (Anode) to a different hole (row 7 on the breadboard), on a different and independent line of the breadboard.
  4. Plug a 220 Ohm resistor between this longer leg and a digital pin of the Arduino, using an additional coloured wire (no red, no black) for convenience. The anode is linked to pin 12 on our breadboard.
  5. Now let us program our Arduino to communicate with the IDE. Your code will look as in example 3.8:

**EXAMPLE 3.8**

//Code to blink an external LED connected on the breadboard

void setup()

{

  pinMode(12, OUTPUT); // Initialising pin 12 as our output

  }

void loop() {

  digitalWrite(12, HIGH); //turning on the LED

  delay(1000);  //delaying the output in HIGH state

  digitalWrite(12, LOW); //Turning off the LED

   delay(1000); // Delaying the state of LED in OFF state

}

* 1. Click on verify to compile the code. Load the code to the Arduino. You will see the LED blinking ON/OFF/ON/OFF.

Figure 3.14 illustrates the connection for the project.

**CONNECTIONS**

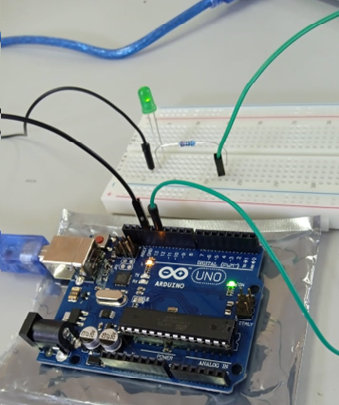


Figure 3. 14: Blinking LED

We can modify our code slightly by using a #define outside the void setup as follows.

#define LED1 12

Take note that the statement does not have a semi-colon. In this instance, we created a #define so we can use the pin number later on in the code without having to hard-code it. Ultimately, our code will then look like the one in Example 3.9:

**EXAMPLE 3.9**

#define LED1 12

//Code to blink an external LED connected on the breadboard

void setup()

{

  pinMode(LED1, OUTPUT); // Initialising pin 12 as our output

}

void loop() {

  digitalWrite(LED1, HIGH); //turning on the LED

  delay(1000);  //delaying the output in HIGH state

  digitalWrite(LED1, LOW); //Turning off the LED

   delay(1000); // Delaying the state of LED in OFF state

}

**NOTE**

As done before using Raspberry Pi, we can connect more than one LED or components to our Arduino board.

**Arduino Traffic Light Project**

We have done this project before using Pictoblox. We are now going to hardcode the program on the IDE. Roads without any supervision or guidance in areas with heavy traffic flows can lead to traffic accidents or congestion. Traffic signals are required for an orderly flow of traffic. A traffic signal is used as an instructing device that indicates the road user to act as per the displayed LED. There are different colours in traffic lights and each colour informs the driver on action to take.

* Red light ON- A driver should stop.
* Amber (Orange) light ON- A driver must slow down and be ready to stop.
* Greenlight ON- A driver can go or proceed to pass the intersection.

You are required to create a project using Arduino to simulate traffic lights.

You will need:

* Arduino Uno
* Breadboard
* LEDs ( Red, Yellow, Green)
* 3 Resistors( 220 Ohm)
* Jumper cables

Figure 3.15 illustrates the connection diagram for the project.

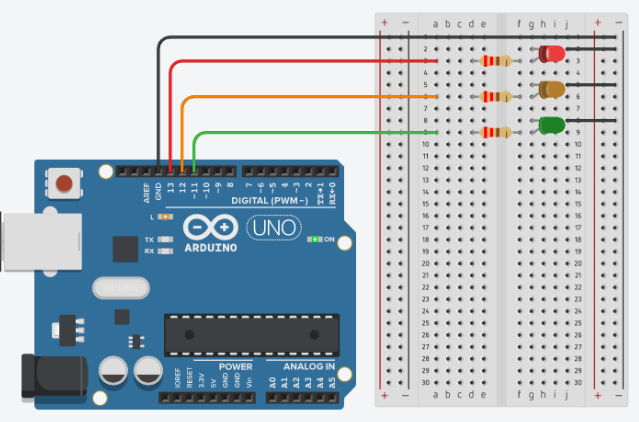


Figure 3. 15: Arduino Traffic Lights circuit

SOLUTION: TRAFFIC SIGNALS

For this project, I used pin 11, 12 and 13.

* Pin 13 is for the Red LED
* Pin 12 is for the Amber (Orange) LED
* Pin 11 is for the Green LED

**EXAMPLE 3.10**

#define LED\_PIN\_R 13 // Pin for the Red LED

#define LED\_PIN\_O 12 //Pin for the Amber LED

#define LED\_PIN\_G 11 //Pin for the Green LED

//Code to blink an external LED connected on the breadboard

void setup()

{

  //Intialising the 3 pins for OUTPUT mode

 pinMode(LED\_PIN\_R, OUTPUT);

 pinMode(LED\_PIN\_O, OUTPUT);

 pinMode(LED\_PIN\_G, OUTPUT);

}

void loop() {

  //Turn RED LED ON and SET Amber and Green to OFF

  digitalWrite(LED\_PIN\_R, HIGH);

  digitalWrite(LED\_PIN\_O, LOW);

  digitalWrite(LED\_PIN\_G, LOW);

  // Delay for 3 seconds

  delay(3000);

   //Turn Amber LED OF and SET RED and Green to OFF

  digitalWrite(LED\_PIN\_R, LOW);

  digitalWrite(LED\_PIN\_O, HIGH);

  digitalWrite(LED\_PIN\_G, LOW);

  // Delay for 1.5 seconds while Amber is ON

  delay(1500);

   //Turn RED and Amber OFF and set Green to ON

  digitalWrite(LED\_PIN\_R, LOW);

  digitalWrite(LED\_PIN\_O, LOW);

  digitalWrite(LED\_PIN\_G, HIGH);

   // Delay for 3 seconds while Green LED is ON

  delay(3000);

}

Figure 3.16 presents a completed project on the traffic light system.

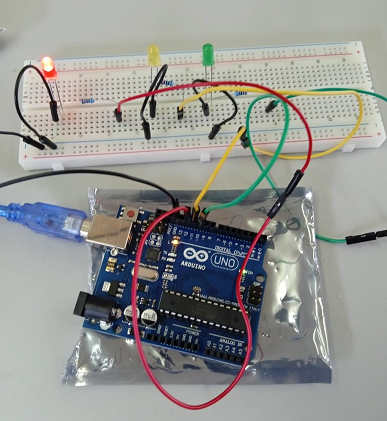


Figure 3. 16: Arduino Traffic Lights connections

# FORMATIVE ASSESSMENT 3.1 INDIVIDUAL TASK

3.1 Define the term single-board computing. (2)

3.2 List 3 types of Arduino Boards (3)

3.3 What does PWM stands for in Arduino. (2)

3.4 Complete table 3.4 to differentiate C++ and Python. (12)

Table 3. 4: Comparing C++ and Python

|  |  |  |
| --- | --- | --- |
| **Parameter** | **C++** | **Python** |
| Extension |  |  |
| Compilation |  |  |
| Speed |  |  |
| Nature of variables |  |  |
| Memory Management |  |  |
| Functions |  |  |

3.5 What is the difference between an interpreter and a compiler. (2)

3.6 Building and LED Counter

You are expected to build an LED Binary Counter with four inputs (4 LEDs) to reflect the output as shown on the following table.

Table 3. 5: LED Binary Counter

|  |  |  |  |
| --- | --- | --- | --- |
| **LED1** | **LED2** | **LED3** | **LED4** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 |

**You will need:**

* LEDs
* Jumper cables x 4
* Small jumper cables for connecting to ground x 4
* 1 long jumper cable for GND to the breadboard
* 4 Resistors( 220 Ohm)
* Breadboard

Code the IDE so that the lights turn ON and OFF to reflect the Binary Counter shown in table 3.4.

Here is the circuit diagram which you can follow.

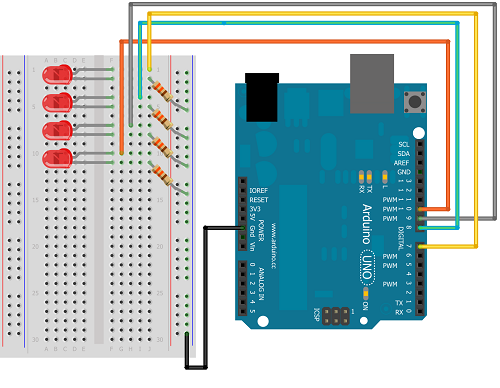


Figure 3. 17: Binary LED Counter

**(20)**

**[Total =41 marks]**

## 3.2 Visual Programming and solution development

In level 2, we used Scratch with Python for visual programming on Raspberry Pis. The four common graphical programming environments are Scratch, Ardublock, mBlock and Minibloq. For the purposes of continuity, we decided to remain with the Scratch dimension. We are going to introduce the same concept using Pictoblox to build on the concepts of Scratch. So why Pictoblox instead of Scratch 3.0? If you go to Scratch under extensions, you only find Microbit instead of Arduino. So for us to have the same functionality, we use Pictoblox. PictoBlox is visual programming language extended from Scratch 3.0 to allow coding with Arduino Just like Scratch, Pictoblox uses the same concepts of drag and drop and has a user friendly interface too. One can learn to code, make interactive animations and games, work on interesting projects based on the Internet of Things, program robot actions, and much more!

**VOCABULARY**

A visual programming language (VPL) is a program that develops applications using graphical components and figures. The graphical components are coded so that when actions such as hover, click mouseover, they trigger some events to occur.

PictoBlox is a an example of a visual programming language. It also has hardware, robotics, artificial intelligence, and machine learning extensions. Bluetooth connections allow for real-time control of a broad variety of generic sensors, actuators, and microcontrollers, such as the microbit, Arduino Uno, Mega, esp32, and many others. You can also use upload mode to upload your code to the hardware. You can also record and/or capture stage output with the PictoBlox.

**Pictoblox Installation**

Step 1:Visit the following link: [https://thestempedia.com/product/pictoblox/download- pictoblox/](https://thestempedia.com/product/pictoblox/download-%09pictoblox/)

Step 2: Select your operation system. In my case I have selected Windows Installer 64-Bit V6.0.1 and run the .exe file

The interface looks just like Scratch but has an option of board and connect for linking Arduino board.

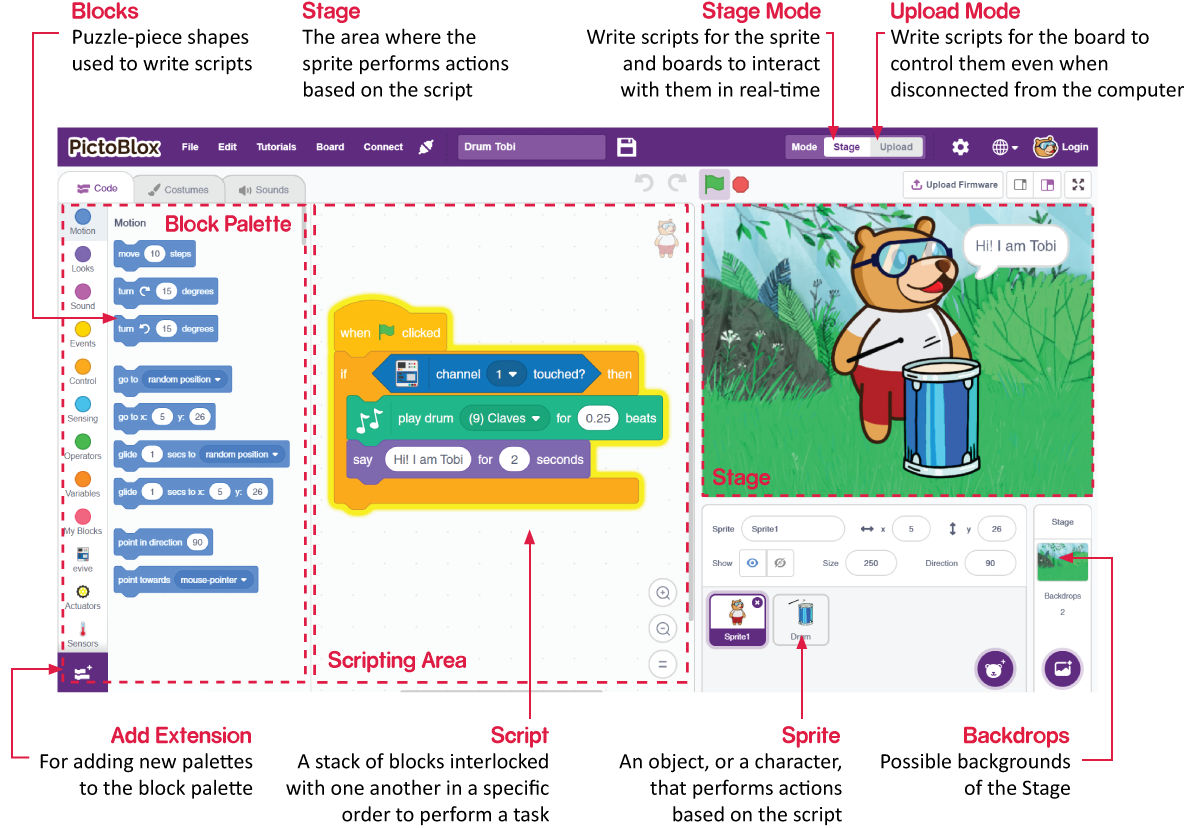


Figure 3. 18: Arduino Interface (Source:https://thestempedia.com/tutorials/program-arduino-board-with- pictoblox)

On Figure 3.18, you will also notice that it has a Python tab to indicate that you can program it with Python.

**How to link Arduino board to Pictoblox**

**Steps**

Step 1: Connect the Arduino board to the computer using the USB. Click on board from the menu bar and select the type of Arduino board. For this lesson, I have an Arduino Uno so I select Uno.

Step 2: Click on connect and select the port. Once connected you will see an image like one in Figure 3.19.

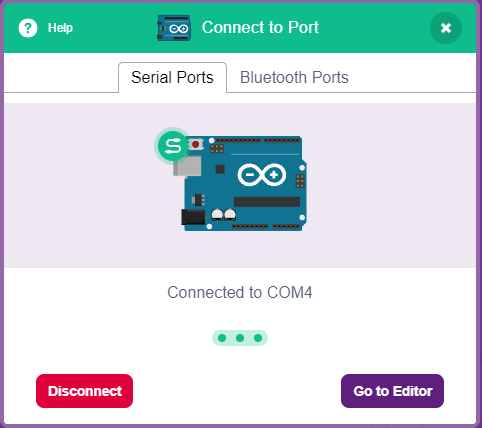


Figure 3. 19: Connecting to the port

Step 3: Notice that Pictoblox comes with Arduino extension as one of the blocks. Under the Arduino Block there are:

* Arduino Uno blocks
* Actuators blocks.
* Arduino Motor Driver Shield
* Sensor
* Display Module
* Dabble blocks, LED brightness blocks, Terminal module blocks, Game Module blocks, IOT , Music Module
* Oscilloscope
* Colour Detector Module
* Camera Module
* Phone Sensors Module
* Inputs Module

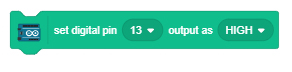
**Mode**

Well, this is like the new dish on the platter. Unlike Scratch, you can work in two modes in PictoBlox:

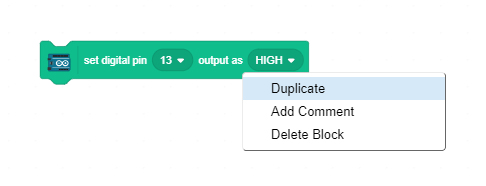
1. **Stage Mode:** In this mode, you can program boards like the Arduino Uno, Nano, Mega, and eVive to interact in real time with sprites using scripts. You can no longer interact in PictoBlox if the board is disconnected.
2. **Upload Mode**: In this mode, you can create scripts and upload them to the chosen board, enabling you to utilize it even when it is not connected to your computer.

**Lighting an LED using Arduino and Pictoblox**

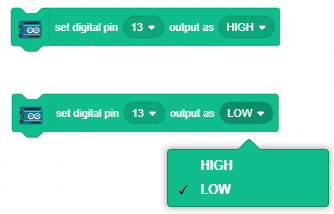
**Step 1:** To control the LED connected to pin 13, we’re going to use the set digital pin () output block.



**Step 2:** We need two such blocks to make the LED blink. So, duplicate it by right-clicking on it.



**Step 3:** Set the output in this block to LOW.



**Step 4:** Now, to control the speed of blinking, we’re going to use the wait block. Go to the Control palette and drag and drop one below each set digital pin () output block.

**Step 5:** Now let’s stack all the blocks together.

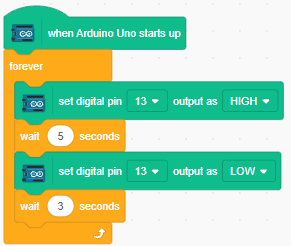


Figure 3. 20: Blink LED

**NOTE: Make sure to click on the upload mode**

Step 6: Tu run the project, click on upload code. Also notice the source code available

{

pinMode(13, OUTPUT); // sets the digital pin 13 as output

}

void loop()

{

digitalWrite(13, HIGH); // sets the digital pin 13 on

delay(1000); // waits for a second

digitalWrite(13, LOW); // sets the digital pin 13 off

delay(1000); // waits for a second

}

You will see the LED blinking. If we remove the forever block, the code will not repeat blinking as it is executed following **sequential construct**. For the process to be repeated several times, use a forever block which causes iteration of the blinking.

**Selection Program**

We will use selection if we want to make a choice. We want to enter a number and if the number entered is greater than 5 onboard LED turns off, and if the number is less than 5, it doesn’t turn on. Use Pictoblox as shown in Figure 3.21.

Solution

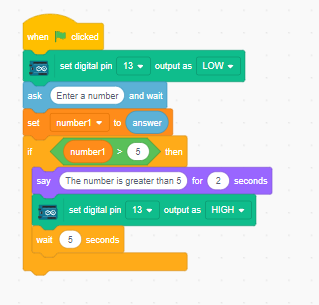


Figure 3. 21: Comparing numbers

**NOTE**

Unlike in previous program, we did not use the “when Arduino Uno starts up” block because we want this to happen only after we enter our own number.

We could display a message if the number entered is below 5. This is done by adding an if … then else block. The script will then look as in Figure 3.22:

**Graphical user interface, text, application, chat or text message

Description automatically generated**

Figure 3. 22: Comparing numbers using if else

In the above script, if a number less than 5 is entered, a message will be displayed but the onboard LED will not turn on because we never instructed it to.

Sometimes we want to execute certain steps repeatedly; we will have to use a forever block. Whatever is in the forever block will be executed repeatedly. A practical example is the traffic lights which will have to run continuously.

**NOTE**

The upload mode shows the full code as you would apply when typing in Arduino IDE.

### 3.2.1 Construct (code/write) using the visual tool

#### The purpose of this section is to introduce you to visual tools for writing code. You will also learn concepts from a visual perspective as well as from a hard-coding perspective using Arduino IDE.

#### 3.2.1.1 retrieving remainders: modulus

The modulo operator, denoted by%, is an arithmetic operator that produces the remainder of an integer division, as detailed in Section 3.2.

**Syntax**

Answer= num1 % num2

The above syntax will produce the remainder when num1 is divided by num2. If num2 completely divides num1, the result of the expression is 0.

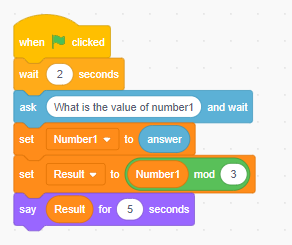


Figure 3. 23: Retrieving remainders

The following example explains how to compute modulus with the Arduino Uno and display the results on the serial monitor. Take note that the principle is the same as the output we get when we use visual programming.

**EXAMPLE 3.11**

//declaring variables

int num1, answer;

int num2 =5;

void setup() {

  // put your setup code here, to run once:

Serial.begin(9600);

}

void loop() {

  // Capturing the value of num1

Serial.println("Please enter value of num1: ");

while ( Serial.available()==0){}

num1=Serial.parseInt();

Serial.print("The value of num1 is : ");

 Serial.println(num1);

//calculating the remainder (modulus)

answer = num1 % num2;

// Displaying output

Serial.print("The answer is ");

Serial.println(answer);

delay(5000);

//incrementing the dividend

num1++;

}

Sample Output

23:01:56.394 -> Please enter value of num1:

23:02:00.933 -> The value of num1 is : 26

23:02:00.978 -> The answer is 1

23:02:05.937 -> Please enter value of num1:

23:02:15.136 -> The value of num1 is : 10

23:02:15.136 -> The answer is 0

23:02:20.140 -> Please enter value of num1:

 As discussed, the sample output confirms what we highlighted. When the value of num1 is 10, the output is 0. Output is only greater than 1 and less than 5 (divisor) if the remainder is not 0.

#### 3.2.1.2 differentiate between real value division and integer division

Because of rational and irrational numbers, real numbers can include fractions, whereas integer division splits two numbers and delivers the integer component of the result. Division is performed in C++ using the arithmetic operator /. The operator returns the division of the left operand by the right operand based on two operands:

*Result= operand\_1/operand\_2*

The dividend is operand \_1, the divisor is operand\_2, and the result is the quotient. It is best to think of the division operator as having two different "modes". The division operator executes floating point division if either operand is a floating point value. Floating point division yields a floating point value and keeps the fraction. As an example:

Serial.begin(9600);

integer num1=45.7;

int num2=5;

int result;

 result=45.7/5;

Serial.print(result);

delay(50000);

The output is 9.14.  Integer division drops any fractions and returns an integer value. For example: 45/5 will give 9.

Serial.print(45/5);

Output

9

#### 3.2.1.3 comparison operators and performing logical comparisons .

We introduced various operators in Section 3.2, including comparison operators. Two values (or variables) can be compared using comparison operators. We need this to make decisions and find answers in programming. The output is Boolean. If the result is true, the output will be 1 and if the result is false, the output will be 0. Of course, when we do the same with PictoBlox, you won’t see the 0s and 1s, but using Arduino will show them. We are first going to depict a scenario of comparing two numbers using Pictoblox, as shown in Figure 3.24.

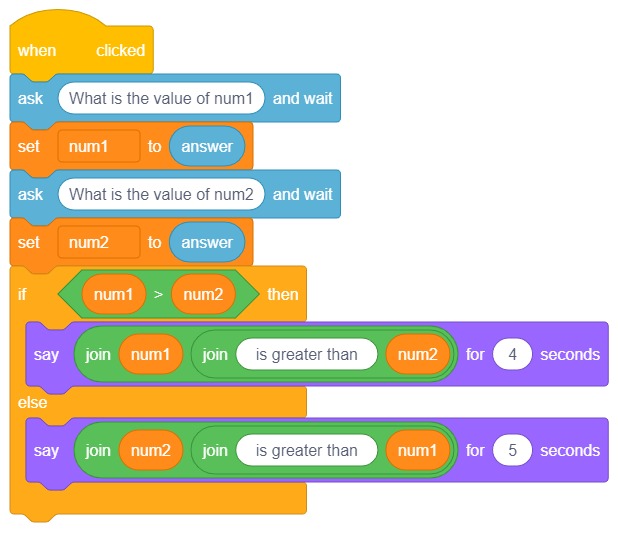


Figure 3. 24: Comparison Operators

As stated earlier, if the num1 is greater, the binary output is a 1 meaning its true and the statement *num1 “is greater than “ num2* will be displayed in the staging area. If the second number is greater then it means the output of the condition is false which is a zero. In this instance, our output will be *num2 “is greater than “ num1.*

Example 12 demonstrates how you could do it using Arduino. Take note that in the example with Pictoblox, we accepted input from the user.

**EXAMPLE 3.12**

//variables

int num1=56;

int num2=44;

void setup() {

  // put your setup code here, to run once:

Serial.begin(9600);

//logical operators

Serial.println(num1>num2);

Serial.println(num1<num2);

Serial.println(num1>=num2);

Serial.println(num1<=num2);

Serial.println(num1==num2);

Serial.println(num1!=num2);

delay(50000);

 }

void loop() {

  // put your main code here, to run repeatedly:

}

Output

21:22:10.020 -> 1

21:22:10.020 -> 0

21:22:10.020 -> 1

21:22:10.020 -> 0

21:22:10.020 -> 0

21:22:10.020 -> 1

#### 3.2.1.4 incorporate and write code constructs to perform basic calculations

Now that we have learned about variables and operators, we will make use of the Arduino to solve some mathematical problems. We are going to start by calculating the area of a circle with a predefined radius. First, we use Pictoblox to represent the solution.

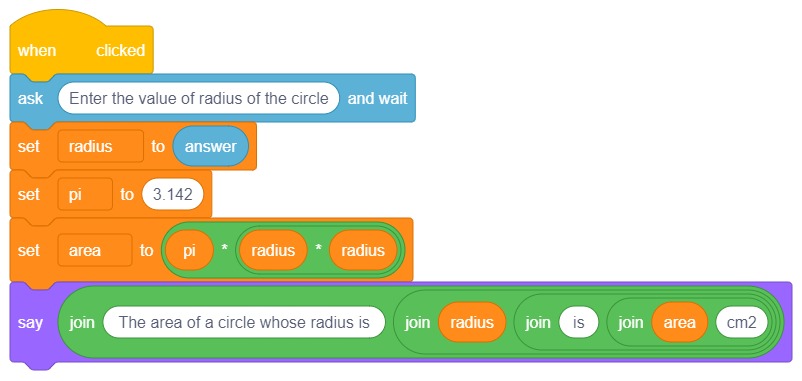


Figure 3. 25: Area of a circle

Example 3.13 illustrates how you would perform the same calculation using Arduino.

**Example 3.13**

float pi=3.142;

float radius=7.00;

float area;

long wait=5000;

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

  }

void loop() {

  // put your main code here, to run repeatedly:

  area=pi  \* radius\*radius;

  Serial.print("A circle with a radius of: ");

  Serial.print(radius );

  Serial.print(" has an area of ");

  Serial.print(area);

  Serial.println("cm2");

  delay(wait);

}

Sample Output

A circle with a radius of: 7.00has an area of 153.96cm2

In the above example, we used float as the data type because our values and the result will have decimal points. I also avoided hardcoding the value for the delay; instead, we declared a variable for that. This is considered a good practice. Also, to avoid seeing the same calculation, you can modify the code by incrementing the radius after the last delay statement as follows:

  radius=radius +.5;

**CLASS EXERCISE**

Use an Arduino to calculate area of a rectangle whose length=8 and width=5 is entered by the user.

**SOLUTION**

Example 3.14 shows how to calculate area of a rectangle whose sides are 8 and 5 respectively for length and width. (using Arduino)

**EXAMPLE 3.14**

int length=0;

int width=0;

float area;

long wait=5000;

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

  }

void loop() {

Serial.println("A rectangle with a length of  ");

Serial.println("Please enter the length of the rectangle: ");

while ( Serial.available()==0){}

length=Serial.parseInt();

Serial.println(length );

Serial.println("Please enter the width of the rectangle: ");

while ( Serial.available()==0){}

width=Serial.parseInt();

Serial.println(width );

area=length \* width;

  Serial.print("A rectangle with a length of  ");

  Serial.print(length );

  Serial.print(" and width of " );

  Serial.print(width );

  Serial.print(" has an area of ");

  Serial.print(area);

  Serial.println("cm2");

delay(5000);

while ( Serial.available()==0){}

}

Sample output

A rectangle with a length of 8 and width of 5 has an area of 40.00cm2

Adding two numbers keyed in through the serial monitor. Now that we have done a lot of examples using graphical representation, adding numbers should be pretty easy. Try it out with your classmate. Here is Arduino version of the same task shown in example 3.15.

**Example 3.15**

//declaring the three variables

int num1;

int num2;

int answer;

String message2=" Num1 is";

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

}

void loop() {

  // put your main code here, to run repeatedly:

Serial.println("Please enter first number: ");

while ( Serial.available()==0){}

num1=Serial.parseInt();

Serial.print("Num1 is ");

Serial.println( num1);

Serial.println("Enter second number ");

while ( Serial.available()==0){}

num2=Serial.parseInt();

Serial.print("Num2 is ");

Serial.println( num2);

answer=num1+num2;

Serial.print("The answer is ");

Serial.println(answer);

delay(5000000);

}

When getting input from the user through the serial monitor, keep in mind the steps for accepting user input with Arduino: ask, wait, and accept input. In the above example, the ask part is when you write the following statement:

Serial.println("Please enter first number: ");

Wait:

while ( Serial.available()==0){}

Accept input:

num1=Serial.parseInt();

**Calculating Volume**

Volume refers to how much space an object fills in three dimensions. It is often measured in cubic units. Arithmetic formulae may be used to readily compute the volume of three-dimensional mathematical forms such as the cube, cuboid, cylinder, prism, and cone, among others.

**EXAMPLE**

You are required to find the volume of a cylinder. To find the volume of a cylinder we first need to find the circular area of the base. The formula for calculating the area of a circle is:

*Area=πr2*

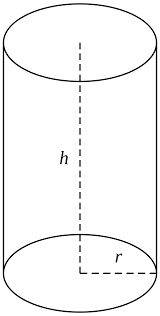
**

Figure 3. 26:Image of a cylinder

We then multiply the area of the circular base by the height (or length) of the cylinder. The formula for the volume of a cylinder is:

*Volume=πr2h*

Find the volume of this cylinder whose radius and perpendicular height  are entered by the user. Use Pictoblox and then show the same solution using Arduino IDE.

Figure 3.27 illustrates the graphical solution for calculating volume of a cylinder.

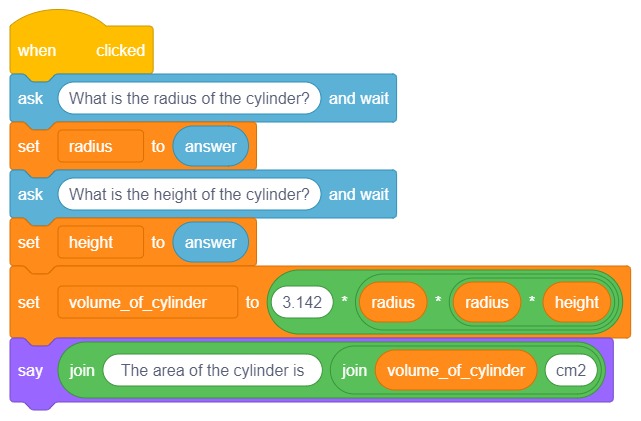


Figure 3. 27: Volume of a Cylinder

Example 3.16 shows how to calculate volume of a cylinder on an Arduino.

**EXAMPLE 3.16**

//declaring variables

float radius, height, volume\_of\_cylinder;

const float pi= 3.14;

void setup() {

  // put your setup code here, to run once:

Serial.begin(9600);

}

void loop() {

  // Accepting the inputs

Serial.println("Please enter radius of the cylinder: ");

while ( Serial.available()==0){}

radius=Serial.parseInt();

Serial.print("The radius is ");

Serial.println( radius);

Serial.println("Please enter height of the cylinder: ");

while ( Serial.available()==0){}

height=Serial.parseInt();

Serial.print("The height of the cylinder is ");

Serial.println( height);

//processing

volume\_of\_cylinder=pi\* radius\* radius \* height;

//Displaying the output

Serial.print("The volume of the cylinder is :");

Serial.println( volume\_of\_cylinder);

delay(5000000);

}

Output

Please enter radius of the cylinder:

The radius is 7.00

Please enter height of the cylinder:

The height of the cylinder is 10.00

The volume of the cylinder is :1538.60cm3

**Value Added Tax (VAT)**

SARS permits businesses to reclaim the VAT component of accepted invoices. This is known as VAT input, and it is a company asset. If a product or service is sold inclusive of VAT, it indicates that the price you see on the label includes VAT (15%). It is the ultimate price of the goods or services that the consumer will pay after VAT is deducted.

**EXAMPLE 3.17**

Minentle bought a cellphone from Abdulla for R4 999.00 inclusive of VAT. How much VAT did Minentle pay? The formula to use is VAT amount= tax fraction x amount incl. VAT.

//declaring variables

float total\_price, vat\_amount;

float VAT=15.00/115;

float answer;

void setup() {

  // put your setup code here, to run once:

Serial.begin(9600);

}

void loop() {

// Accepting the inputs

Serial.println("Please enter total price inclusive of VAT: ");

while ( Serial.available()==0){}

total\_price=Serial.parseFloat();

//processing

Serial.print("The total price including VAT is ");

Serial.println(total\_price);

vat\_amount=VAT \* total\_price;

//Displaying the output

Serial.print("The amount of VAT paid is :R");

Serial.println(vat\_amount);

delay(5000);

while ( Serial.available()==0){}

}

Output

23:59:12.365 -> Please enter total price inclusive of VAT:

23:59:13.835 -> The total price including VAT is 4999.00

23:59:13.900 -> The amount of VAT paid is :R652.04

Therefore, VAT amount= 15/115\*4999 incl VAT.

VAT Amount = R652.04

If an item price is labelled VAT **exclusive** what does it mean? It means that the marked price you see excludes VAT (15%) and this 15% VAT still has to be added on to the selling price. It is also called the base price.

**Task 3.1**

Meagan bought a Raspberry Pi from [takealot.com](https://www.google.com/search?rlz=1C1CHBD_en-GB&sxsrf=AJOqlzU-W9WaGPMGKCZSpb-76ox77buQ8A:1674641791663&q=takealot.com&spell=1&sa=X&ved=2ahUKEwiQ1PjovuL8AhXKVsAKHVaRC7AQBSgAegQICRAB) for R2 385.09 exclusive of VAT. Calculate the VAT paid by Meagan. What was the new total price. Use Arduino IDE to present your solution.

**Task 3.2**

Quadratic equations are widely used in everyday life since they are critical to human survival. Quadratic equations must be employed directly or indirectly in every sector where speed, area, or profit must be calculated. Before beginning a job, construction personnel employ quadratic equations to compute the area.

The Quadratic Formula: x = −b ± √b²-4ac/2a

Solve: 5x2 + 6x + 1 = 0 Use Arduino to calculate the value of x1, x2.

**Programming Constructs**

Programming constructs are typical building blocks used in the creation of programs. A program's foundation is made up of these programming structures. Three programming constructs exist:

* Sequence
* Selection/Conditional construct
* Iteration/Looping/Repetition construct

**Sequence** - is the sequence in which instructions are received and processed. All the programs we have been running so far follow sequential constructs since statements are executed in the order they are given or typed.

**Selection/Conditional** construct- specifies the path that a program follows when running depending on a condition met/not met. The most common selection statements are the if statements, if then statements and switch statements. These are explained in detail in Topic 8.

**Iteration**- is the repeated execution of a segment of code until a condition is met or while a condition is met. The most common statements are for loops, and while loop.

#### 3.2.1.5 include conditional constructs [if and if..then-else]

The [**if()**](https://www.arduino.cc/reference/en/language/structure/control-structure/if/?_gl=1*5hl86d*_ga*NTEwNjQ1MTMwLjE2NzMyNTI0Nzc.*_ga_NEXN8H46L5*MTY3NDY0OTU5MS4yOS4xLjE2NzQ2NDk2MTEuMC4wLjA.) statement is the most basic of all programming control structures. It allows you to make something happen or not, depending on whether a given condition is true or not. It looks like this:

if (someCondition) {

// do stuff if the condition is true

}

Let us consider a situation where you want to compare two numbers and display which of the two numbers is greater. In such a scenario, you can implement the if statement as in example 3.18:

**EXAMPLE 3.18**

//declaring variables

int a=9;

int b= 6;

void setup() {

 // put your setup code here, to run once:

Serial.begin(9600);

}

void loop() {

//processing

if (a>b){

Serial.print(a);

Serial.print("  is greater than : ");

Serial.print( b);

}

delay(5000);

while ( Serial.available()==0){}

}

Output

14:33:19.517 -> 9 is greater than : 6

**NOTE**

An if statement must have a condition and in the example the condition is:

(a>b). This is a simple if statement. The statement is only printed if the condition is true. If value of a was smaller than b, then nothing will be printed out.

To avoid the program from just saying nothing when the condition is not met, we implement.

**if….else…** statement. The code on comparing two numbers can then be modified as example 3.19:

**EXAMPLE 3.19**

//declaring variables

int a=3;

int b= 6;

void setup() {

 // put your setup code here, to run once:

Serial.begin(9600);

}

void loop() {

//processing

if (a>b){

Serial.print(a);

Serial.print("  is greater than : ");

Serial.print( b);

}

else{

  Serial.print(b);

Serial.print("  is greater than : ");

Serial.print( a);

}

delay(5000);

while ( Serial.available()==0){}

}

Output

6 is greater than : 3

**if-else if-else Statement**

The "if...else if...else" or "if-else if-else" statement is used when we need to execute the required block of code based on multiple conditions. That is, we require multiple "if-else" statement when we need to test many circumstances and wish to run a single block of code. The typical format is as follows:

if (someCondition) {

// do stuff if the condition is true

}

else if (someCondition){

do stuff if the condition is true

}

else if (someCondition){

do stuff if the condition is true

}

**Task 3.3**

Consider the following the college results in table 3.6 used to manually grade the students based on their average mark. You are required to create a system for the school to avoid the manual grading system which has been found to have potential of errors.

Table 3. 6: Grading table

|  |  |
| --- | --- |
| **Mark** | **Grade** |
| 90-100 | A+ |
| 80-89 | A |
| 70-79 | B+ |
| 60-69 | B |
| 50-59 | C |
| 40-49 | D |
| 30-39 | E |
| 0-29 | F |

**Introducing Arduino components**

When purchasing Arduino starter pack, it comes with lots of components such as actuators, sensors, display modules and dabble. In this section we are going to use one type of actuators called potentiometer. An actuator is a device that converts energy in movement for example potentiometer.

**VOCABULARY**

Potentiometers are three-terminal resistors with sliding or rotating contacts that provide a voltage divider that can be adjusted.

A potentiometer acts as an adjustable voltage divider. We are going to use a potentiometer to get varying resistance by turning the knob on the component. The resistance will range from 0 to 1023 depending with position of knob.

**Hardware Required**

* Arduino Uno
* 10k Potentiometer
* Breadboard
* Jumper wires

Connections

Step 1: Plug the potentiometer along column j on the breadboard.

Step 2: Connect the centre of the pin on the breadboard to the analog pin A0 on the Arduino

Step 3: Connect the ground (GND) on to the first leg of the potentiometer on the breadboard.

Step 4: Connect the other outside leg of the potentiometer to 5V on the Arduino.

Step 5: Connect the USB-B cable to the Arduino and to the computer port.

Step 6: Code the program on the Arduino IDE

Step 7: Run the program and change position of the knob. In the example below, we made use of the **if … else …. Statement**

You are going to see varying resistance displayed in the output area based on the position of the knob.

**EXAMPLE 3.20**

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

    // put your main code here, to run repeatedly:.

  int sensorValue=analogRead(A0); //read the input on analog pin 0

 Serial.println(sensorValue); // print out the value you read

  if (sensorValue<512){

    Serial.print("Resistance is on the lower half");

  }

  else if (sensorValue>512){

    Serial.print("Resistance is now on the upper half");

  }

  Serial.println("");

  delay(1000);

  while (Serial.available()==0){}

}

void loop() {

}

Output

21:54:26.339 -> 1023

21:54:26.339 -> Resistance is now on the upper half

21:54:41.669 -> 0

21:54:41.669 -> Resistance is on the lower half

21:54:53.985 -> 247

21:54:53.985 -> Resistance is on the lower half

21:55:08.837 -> 521

21:55:08.870 -> Resistance is now on the upper half

The application additionally generates statements that indicate the knob has been turned from the lower to higher half depending on reading. If the reading is less than 512, the resistance will be displayed in the lower half; otherwise, it will be displayed in the top half.

**Sensors**

The Arduino Uno starter kit comes with many sensors. There are different types of sensors such as light sensors, temperature sensors,  humidity sensors, sound sensors and many others. We are going to discuss the temperature sensors.

**VOCABULARY**

Sensors are devices that change a physical quantity—like temperature or light intensity—into an electrical quantity.

A good example of a sensor is an LM35 sensor IC, which is shipped together with the Arduino Uno starter kit. The LM35 has just 3 pins: 2 for the power supply and one for the analog output. The output pin provides an analog voltage output that is linearly proportional to the temperature in degrees C. An output of 0 V corresponds to a temperature of 0 degrees C, and the output increases by 10 mV for every degree increase in the temperature. To convert the output voltage to temperature, you need only to divide the output voltage in mV by 10. For example, if the output value equals 315 mV (0.315 V), the temperature is 31.5°C.

Temperature sensors can be applied to devices such as fridges and geysers.

**CLASS EXERCISE**

You are required to design a program using Arduino to implement a temperature sensor. If the temperature sensor reads temperature greater than 300C, turn on a red LED and turn off a green LED else turn the green LED on and red LED off.

**Required**

* Breadboard
* Arduino Uno
* Jumper wires
* x 470-ohm resistors
* x LEDs (Green and Red)
* LM35 Temperature Sensor

Conversion Formula:

Read temperature from the analogue input and multiply by 0.48828125

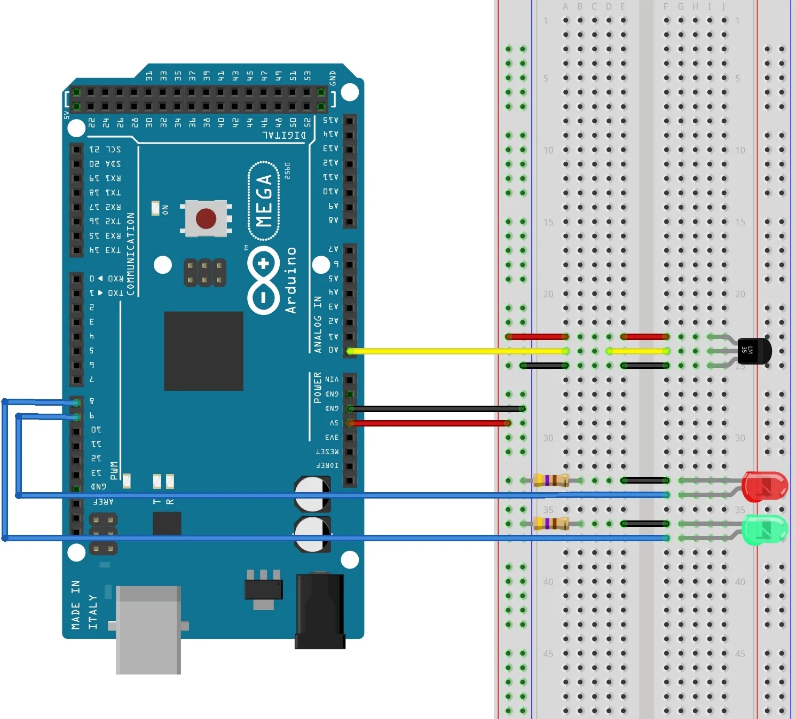


Figure 3. 28: Heat Sensor Diagram

**EXAMPLE 3.21**

const int temp\_read = 0 ; //naming pin 0 of analog input side as temp\_read

const int high = 8 ; // For turning on and off yellow LED connected to pin 8

const int low = 9; // For turning on and off Green LED connected to pin 9

void setup()

{

Serial.begin(9600) ; //Starting serial Communication at baud rate of 9600

pinMode(high,OUTPUT); //declaring LED pins as OUTPUT

pinMode(low,OUTPUT);

}

void loop()

{

int temp\_read = analogRead(0) ; //reading analog voltage

temp\_read = temp\_read \* 0.48828125; //converting temperature read into Celsius

//Displaying current temperature

Serial.print("Current temperature = ");

Serial.print(temp\_read);

Serial.print("\*C");

Serial.println();

//Set delay to 1 sec

delay(1000);

//check if temp read is greater than 30

if(temp\_read>30)

{

//if greater than 30 turn the red light

digitalWrite(high,HIGH) ;

digitalWrite(low,LOW) ;

}

else

{

  //if less than 30 turn the green light

digitalWrite(high,LOW) ;

digitalWrite(low,HIGH) ;

}

}

Output

14:55:49.416 -> Current temperature = 26\*C

14:55:50.399 -> Current temperature = 27\*C

14:55:51.401 -> Current temperature = 28\*C

14:55:52.415 -> Current temperature = 29\*C

14:55:53.427 -> Current temperature = 30\*C

14:55:54.413 -> Current temperature = 31\*C

14:55:55.430 -> Current temperature = 33\*C

Connections

**Steps:**

Step 1: Connect the Green and Red LEDs to the breadboard. Add two resistors to the shorter legs of the two LEDs

Step 2: Connect pin 9 to the longer leg of the Green LED

Step 3: Connect pin 8 to the longer leg of the Red LED

Step4: Plug the LM35 Temperature Sensor on to the breadboard

Step 5: Connect A0 to the middle leg of the sensor

Step 6: Connect the one leg of the sensor to the negative row on the breadboard and the other leg on to the positive row.

Step 7: Connect the GND to the negative row on the breadboard to link with the resistor

Step 8: Connect the 5V from the Arduino to the positive row on the breadboard

**Sample connections**

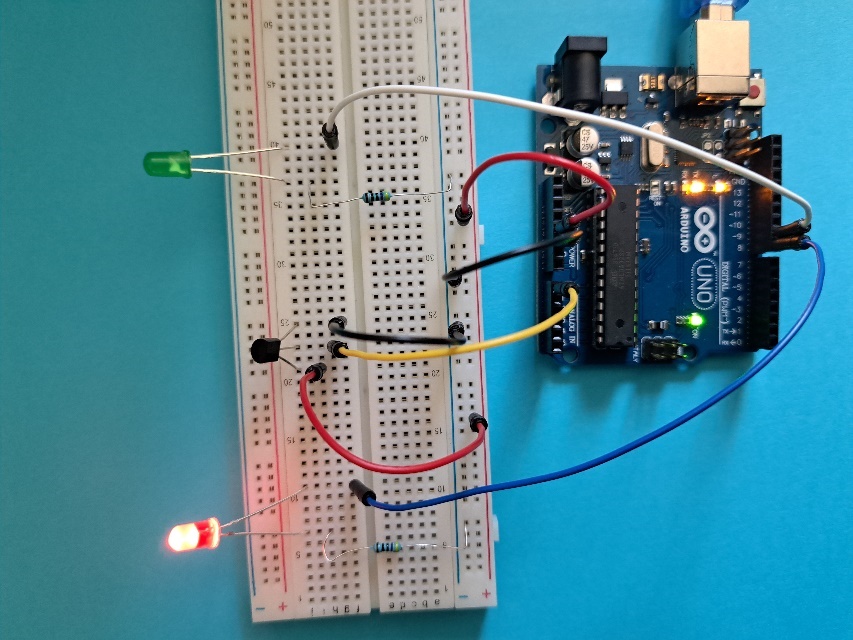


Figure 3. 29: Using Temperature sensor

Run the program you will see the green LED turning on. Try raising the temperature above 30 °C (you can do this by blowing hot air into the temperature sensor with a hair dryer) and observing the output.

**CLASS EXERCISE**

**Control Pin 13 LED of Uno**

You are required to control Pin 13 LED by the signals received from the IR sensor. Use Pictoblox to turn the sensor ON when we bring our hand close to the IR Sensor and OFF when we remove your hand close to the IR sensor. If the hand does not work use any remote control

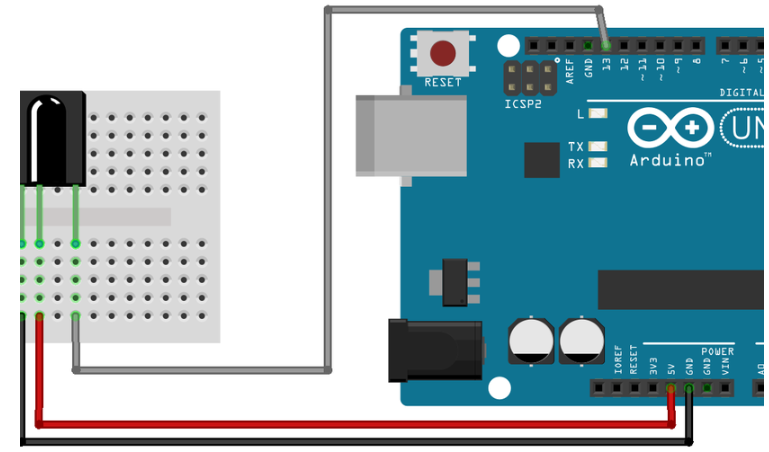


Figure 3. 30: Circuit diagram for IR Sensor connection

You need:

* IR sensor
* x jumper cables
* Breadboard
* Arduino Uno



* A=Signal
* B=GND
* C=VCC (Supply Voltage 5V)

**Connections**

Steps

Step 1: Connect the output pin of the IR sensor to pin 2 (Signal)

Step 2: Connect the middle leg of the sensor GND

Step 3: Connect the outer leg to the 5V on the breadboard (VCC)

Step 4: Add When *Arduino Uno* starts up hat block

Step 5: Add a *forever* block and an *if… else* block inside the *forever* block

Step 6: Drag and drop the *read digital sensor* block inside the hexagonal space of the *if- else* block and select IR form the drop-down menu.

Step 7: Duplicate this block and drop it below the *else* arm. Then, set the output to HIGH.

Step 8: Run the program by clicking the upload code

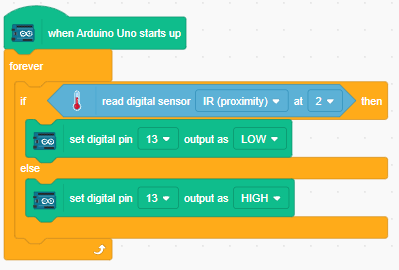


Figure 3. 31: Using IR Sensor

Uno’s Pin 13 LED turns ON when you bring your hand close to the IR sensor and it turns OFF as you take your hand away. However, the best bet would use a remote control which emits infrared signal.

**Application**

This sensor can be used for security systems at home. For example, if you want to use beams for home security and have an alarm system activated when an object passes through the beams.

#### 3.2.1.6 Include iteration (looping) structures [fixed counter loop]

Programs use loops to control their flow. A loop repeats the same block of code over and over. In the loop, each cycle is referred to as an "iteration." It is possible to control whether the program enters the loop by defining certain conditions in the code. There are two functions in the Arduino code structure: void setup() and void loop(). The loop will repeat whatever is inside several times. You see the output being printed continuously because of this. An example of a loop would be running around the athletics ground five times. Another example of a loop is printing numbers from 0 to 10. We have three types of loops in C++:

* For Loop
* While Loop
* Do While Loop

**For Loop**

A for loop verifies the condition stated before entering the loop block, making it an **entry-controlled loop**. A loop variable is used in the For loop to control it. This variable is first initialized to a certain value, after which it is checked by comparison with the counter variable before the loop variable is updated.

**Syntax of a for loop**

for (initialization expression; test expression; update expression) {  
  *// code block to be executed*  
}

The line of code in Arduino would look like below:

**for** (**int** i=0; i<=10; i++) Serial.println(i);

Example 3.22 demonstrates code for a fixed counter loop using Arduino IDE.

**EXAMPLE 3.22**

// declaring the loop variable

int x=0;

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

}

void loop() {

  //setting up the condition and incrementing by 1

 for (x=0; x<11; x++){

   Serial.println(x);

   delay(2000);

 }

 while (Serial.available()==0){}

}

Output

16:02:45.643 -> 0

16:02:47.636 -> 1

16:02:49.646 -> 2

16:02:51.680 -> 3

16:02:53.651 -> 4

16:02:55.650 -> 5

16:02:57.691 -> 6

16:02:59.636 -> 7

16:03:01.668 -> 8

16:03:03.639 -> 9

16:03:05.679 -> 10

In Example 3.22, x is the loop variable and is initialized to 0. The step value is 1, and the stopping value is 10. The output can be controlled by the update expression. For example, if you want to print numbers in the range of 10,9,8,7,6,5,4,3,2,1, Here is the code snippet:

for (int i=10; i>=1; i--){}

The full code will be as below:

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

}

void loop() {

 Serial.println("Arduino count down for loop");

   for (int i=10; i>=1; i--) {

      Serial.print("i is : ");

      Serial.println(i);

   }

delay(500);

while (Serial.available()==0){}

}

OUTPUT

11:58:01.061 -> i is : 10

11:58:01.094 -> i is : 9

11:58:01.094 -> i is : 8

11:58:01.094 -> i is : 7

11:58:01.127 -> i is : 6

11:58:01.127 -> i is : 5

11:58:01.127 -> i is : 4

11:58:01.159 -> i is : 3

11:58:01.159 -> i is : 2

11:58:01.159 -> i is : 1

On the above countdown loop we changed the starting point of our loop to 10. We also used a decrement our iterator expression.

**TASK 3.4**

Use an Arduino IDE to print multiples of 3 between 30 and 10.

We will work more on loops as we go along. Let us move on to the while loop().

**While Loop**

The while loop iterates through a block of code as long as a specified condition is true. When we are unsure of the exact number of times the body of the loop needs to run beforehand, we utilize while loops.

Syntax

while (condition) {  
*// code block to be executed*  
}

We are going to repeat the same task of printing values from 1 to 10 in example 3.23.

**EXAMPLE 3.23**

int x =0;

void setup() {

  // put your setup code here, to run once:

Serial.begin(9600);

}

void loop() {

  // put your main code here, to run repeatedly:

while (x<=10){

Serial.println(x);

x=x+1;

delay(1000);

  }

 while (Serial.available()==0){}

}

18:34:55.339 -> 0

18:34:56.332 -> 1

18:34:57.350 -> 2

18:34:58.352 -> 3

18:34:59.341 -> 4

18:35:00.333 -> 5

18:35:01.356 -> 6

18:35:02.351 -> 7

18:35:03.353 -> 8

18:35:04.345 -> 9

18:35:05.345 -> 10

While loop is a pretest loop meaning that the condition is checked before

**Do While Loop**

Do while loop is similar to while loop except that the condition is checked after executing at least once regardless of the condition.

**Syntax**

do

{

// statements to execute in the loop body

update\_expression;

} while (test\_expression);

Example 3.24 demonstrates the do while loop to print first 10 natural numbers.

**EXAMPLE 3.24**

int x =0;

void setup() {

  // put your setup code here, to run once:

Serial.begin(9600);

}

void loop() {

  // put your main code here, to run repeatedly:

do{

Serial.println(x);

x=x+1;

delay(1000);

  }

  while (x<=11);

 while (Serial.available()==0){}

}

Output

19:00:08.400 -> 0

19:00:09.384 -> 1

19:00:10.388 -> 2

19:00:11.392 -> 3

19:00:12.390 -> 4

19:00:13.384 -> 5

19:00:14.417 -> 6

19:00:14.417 -> 6

19:00:15.393 -> 7

19:00:16.411 -> 8

19:00:17.399 -> 9

19:00:18.407 -> 10

**CLASS EXERCISE**

Use a for loop to light up 5 LEDs in sequence and in reverse. Use digital pins 2-6 as outputs for the 6

Step 1: Connect 5 LEDs to the breadboard

Step 2: Connect GND to the negative rail on the breadboard

Step 3: Connect each cathode of each LED to the negative rail through a 220-ohm resistor

Step 4: Connect the anode of each LED to the digital output ports (from pin 2 to pin 6)

Step 5: Code the program on the Arduino IDE. Run the program and you should see the LED turning

Source code in example 3.25 shows how to light up 5 LEDS in sequence and in reverse.

**EXAMPLE 3.25**

int timer = 100;  // The higher the number, the slower the timing.

void setup() {

  // use a for loop to initialize each pin as an output:

  for (int output\_pin = 2; output\_pin < 7; output\_pin++) {

    pinMode(output\_pin, OUTPUT);

  }

}

void loop() {

// loop from the low (pin2) to high (pin6)

  for (int output\_pin = 2; output\_pin < 7; output\_pin++) {

  // turn the pin on:

    digitalWrite(output\_pin, HIGH);

    delay(timer);

    // turn the pin off:

    digitalWrite(output\_pin, LOW);

  }

  // iterate from high (pin6) to low (pin2)

  for (int output\_pin = 6; output\_pin >= 2; output\_pin--) {

    // turn the pin on:

    digitalWrite(output\_pin, HIGH);

    delay(timer);

    // turn the pin off:

    digitalWrite(output\_pin, LOW);

  }

}

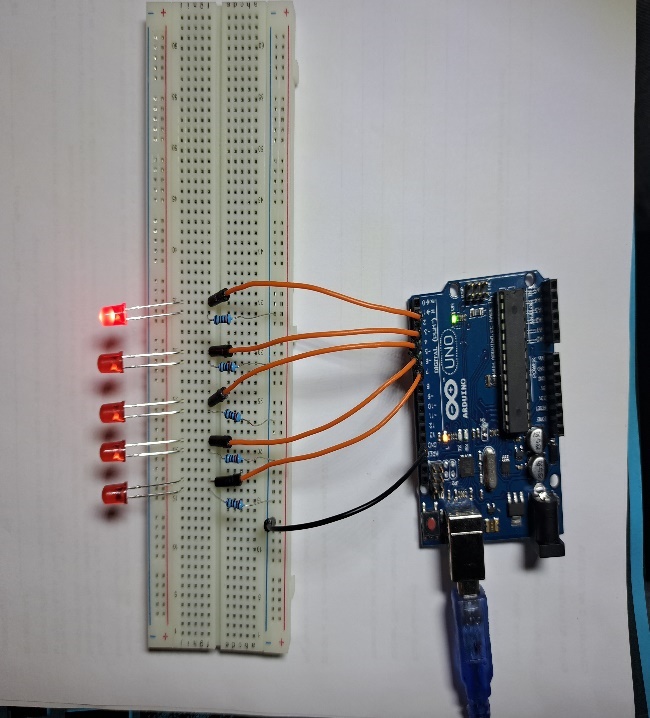


Figure 3. 32: Lighting LEDs in sequence and reverse

**TASK 3.5 Lighting LEDs in sequence and reverse**

Modify code in example 3.25 use a while loop instead of a for loop to power the 5 LEDs in sequence.

#### 3.2.1.7 incorporate a combination of iteration and condition structures

Keep in mind that we use iteration when we require the computer to repeat actions, and conditionals structures when we need the machine to decide. Therefore, we mix these elements when we must frequently decide on something or when the outcome of a decision necessitates repeatedly performing an action. We'll examine each of these individually.

Conditional execution may happen inside of a loop , and this is common. When we are iterating across a list of items, we often need to inspect each item and then decide about what to do with that item. We can also make use of Break and Continue statements inside our conditions.

**Break**

In C++, the break statement is used to end a loop. The loop iterations cease when the break statement is reached from within a loop, and control is instantly transferred from the loop to the first statement following the loop.

**Syntax**

for (initialization expression; test expression; update expression) {

if (someCondition) {

break;

// do stuff if the condition is true

}

  // code block to be executed  
}

**Scenario**

Let us assume we can have a loop to print numbers from 1 to 10 and decide that if along the process, the program encounters a digit 3 then it should exit the loop and print “I have encountered a block!! I am exiting” else print- “the value of I is I”. In such cases, you will need to make use of break commands as shown in example 3.26

**EXAMPLE 3.26**

int  i;

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

}

void loop() {

  //for loop to print integers from 1 to 10

  for(i=0; i<10; i++) {

    //if the value of i is 3, the loop must stop

    if(i == 3){

    Serial.println("I have encountered a block!! I am exiting");

    break;

    }

    Serial.print("The value of i is ");

    Serial.println(i);

}

Serial.println("End of the program");

//stopping the program from execution

while (Serial.available()==0){}

}

Output

13:05:48.424 -> The value of i is 0

13:05:48.424 -> The value of i is 1

13:05:48.457 -> The value of i is 2

13:05:48.490 -> I have encountered a block!! I am exiting.

13:05:48.523 -> End of the program

Notice that our program only executes the *for…. loop* statements as long as the value of I is not equal to 3. Once this happens, the program stops. We can add some electronics and either a buzzer that rings or an LED that turns on when that condition is met. Try it out with your classmate.

**Buzzer**

There are two types of buzzers used with Arduino: active buzzers and passive buzzers. Active buzzers are called active because they only need a DC voltage to produce sound. Passive buzzers need an AC voltage to produce sound. For instance, if you take an active buzzer and attach the positive leg to the Arduino on the 5V and the negative leg to the GND, the buzzer will start making a steady noise right away. TRY IT OUT.

**NOTE**

An active buzzer will always have a sticker on top and it also shows the positive sign.

**Pair Assessment**

Modify the program with for loop and if statements and implement a buzzer when the value of i is 3.

Steps

Step 1: Connect the buzzer on to the breadboard (Positive on to pin 8 and negative to GND).

Code: Make sure you declare the buzzer on pin 8.

**EXAMPLE 3.27**

//declare the variables

int  i;

int buzzPin=8;

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

  pinMode(buzzPin, OUTPUT);

}

void loop() {

  //for loop to print integers from 1 to 10

  for(i=0; i<10; i++) {

    //if the value of i is 3, the loop must stop

    if(i == 3){

    Serial.println("I have encountered a block!! I am exiting");

    digitalWrite(buzzPin, HIGH);

    delay(2000);

    digitalWrite(buzzPin, LOW);

    break;

    }

    Serial.print("The value of i is ");

    Serial.println(i);

}

Serial.println("End of the program");

//stopping the program from execution

while (Serial.available()==0){}

}

Output

14:33:48.956 -> The value of i is 0

14:33:50.511 -> The value of i is 1

14:33:50.543 -> The value of i is 2

14:33:50.575 -> I have encountered a block!! I am going to buzz

14:33:54.566 -> End of the program

**CONNECTIONS**

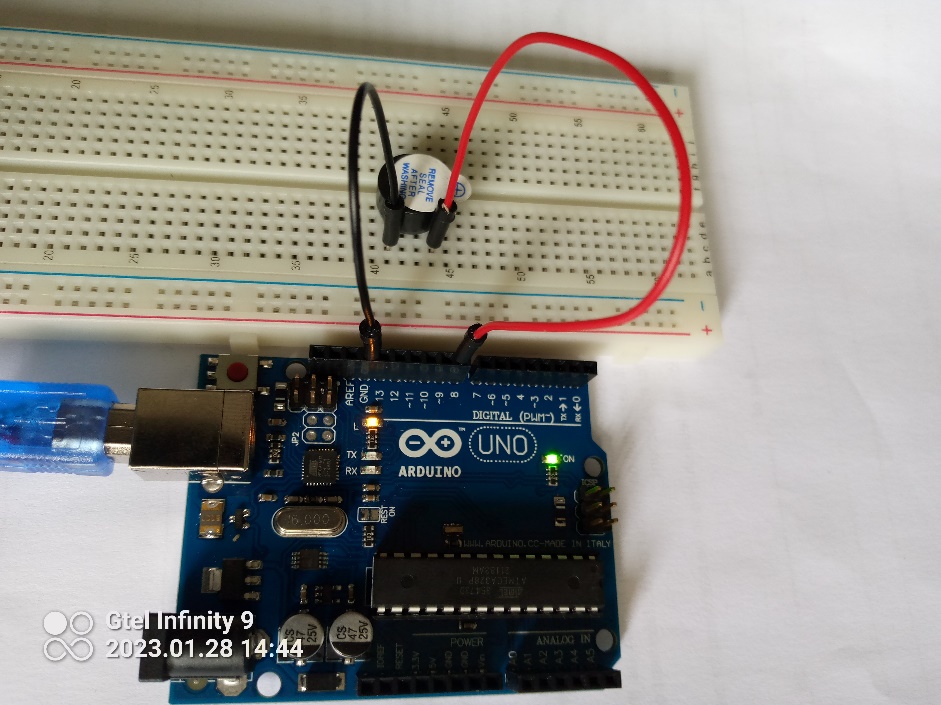


Figure 3. 33: Implementing a buzzer

**CONTINUE COMMAND**

**Continue Command**

The continue keyword allows you additional control over how loops behave. When the program is prompted to continue, the current iteration of the loop is terminated and a new cycle is started.

We are going to modify the program to print numbers from 0 to 10. When it reaches value 3, it will skip printing but continue afterward to the end of the loop.

**EXAMPLE 3.28**

//declare the variables

int  i;

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

  //set the digital pin 8 to OUTPUT

 }

void loop() {

     //for loop to print integers from 1 to 10

  for(i=0; i<10; i++) {

    //if the value of i is 3, the loop must stop

    if(i == 3){

    Serial.println("I have encountered a block!! I am going to buzz");

    //Turn the buzzer on

       delay(2000);

    delay(2000);

   // turn the buzzer off

       continue;

    }

    Serial.print("The value of i is ");

    Serial.println(i);

}

Serial.println("End of the program");

//stopping the program from execution

while (Serial.available()==0){}

}

12:03:43.170 -> The value of i is 0

12:03:43.202 -> The value of i is 1

12:03:43.202 -> The value of i is 2

12:03:43.234 -> I have encountered a block!! I am going to buzz

12:03:47.244 -> The value of i is 4

12:03:47.244 -> The value of i is 5

12:03:47.276 -> The value of i is 6

12:03:47.309 -> The value of i is 7

12:03:47.309 -> The value of i is 8

12:03:47.341 -> The value of i is 9

12:03:47.381 -> End of the program

### 3.2.2 Applying programming language tools and constructs to draw various shapes

Common building pieces, known as programming constructs, are used to create programs. All programs are built on the foundation of these programming structures. We have already identified the three programming constructs in thistopic which are: sequence, selection and iteration.

#### 3.2.2.1 Sequence

The order in which instructions arise and are processed is referred to as the "sequence." There are some problems that require the sequential execution of statements to achieve the desired result. For instance, if we want to develop a program to calculate the total cost of ten loaves of bread, we will need to enter the cost per unit and multiply by the quantity in that order, then display the total cost. It is very simple to design a sequence construct using block-based programming. Students have the chance to perceive a programming aspect from a different perspective thanks to visual representations. Instead of impatiently awaiting the print statement after your program runs, you may see the concept itself in a visual format. This is especially helpful for ideas like recursion and iteration that are abstract in nature.

In this tutorial we are going to draw a square using Pictoblox to demonstrate the sequence construct. We make use of the pen extension in Pictoblox.

**Steps**

Step 1: Add the *When…. clicked* block

Step 2: Choose the + sign under blocks to add  *pen* extension. You will need to type pen in the search area.

Step 3: Add the *pen up* block

Step 4: Add an *erase all* blocks in case there is something on th

Step 5: Add a *pen down* block

Now we want to start drawing, so we need to add a new block. Under blocks pallette click on *my block* and make a *new block*. Name it square and a define square will appear on the script area.

Step 6: Add the square block under the *pen down* block. Your code will look as follows:

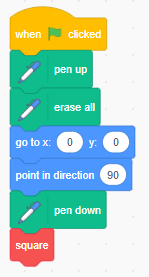
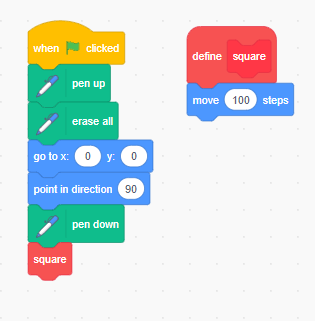


Figure 3. 34: Drawing a square

Now we want to start drawing the shapes. We will start with a straight line. Add a move 0 steps below the define square block. Change the value to 100. As of now your code will look as follows:



Step 7: We want to make a square. So add a turn 90 degrees block

Step 8: Add the move … steps blocks and type in 100. Add another turn 90 degrees block

Step 9: Repeat step 7 and 8 again.

Step 10: Add a erase all block anywhere on the Script area to allow deleting everything when done. . Run the script and you will see your square on stage area.

Graphical user interface

Description automatically generated

Figure 3. 35: Script to draw a square

If you want to clear everything, click on erase all.

#### 3.2.2.2 Selection

Selection or Branching is computer programming construct for choosing whether. a line of code or a group of lines of code should be executed (run) based on the condition met/or not met. We are going to use Pictoblox to check whether a person can vote or not.

Steps

Step 1:Add a *when..clicked* block from events

Step 2:Select the *ask….*and *wait* from the sensing blocks. Edit the text to read: please enter your age.

Step 3:Go to *variables* and click make a variable. Name the variable age.

Step 4: Add the *set.. to* block. Edit the block to set *age* to *answer*

Step 5: Add the *if… else….* block*.* Add a greater than operator and edit it to read: age >17

Step 6:Add a *say…* block from *Looks* category. Edit the text to read: “Congratulations!!!. You are now eligible to vote”.

Step 7: In the else section, add another *say…* block. Edit the text to read: “Unfortunately. You are not eligible to vote yet”

Step 8: Run your code and enter age as a integer.

Your script will look as the one in Figure 3.36.



Figure 3. 36: Selection construct

#### 3.2.2.3 Iteration

Iteration refers to the concept of repeating commands or a group of lines. In computer programming, it is also known as a "loop." When the code executes (repeats) once, this repetition is referred to as one iteration. There is no limit to the number of iterations the program can run. You can even develop infinite loops; however, these are not recommended as they have a negative impact on memory. We can use the Pictoblox application to demonstrate  iteration by drawing a circle.

**Drawing a Circle**

Icon

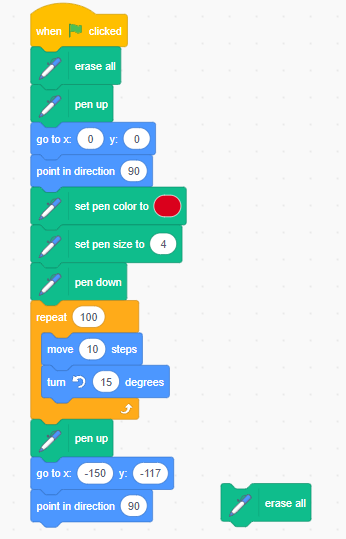
Description automatically generated

Figure 3. 37:Script for drawing a circle

**Group Assessment**

We are going to draw a beautiful flower. Here is the script which you can follow:

**Steps**

Step 1: Add : *When .. clicked* block

Step 2: Add *erase all* then pen up block

Step 3: Add the *go to x: 0 and y: 0* to set the sprite

Step 4: Add *point in direction 90* block

Step 5: Add the *set pen colour*, set pen size to. For this demonstration, I set it to 2.

Step 6: Add a *pen down* block

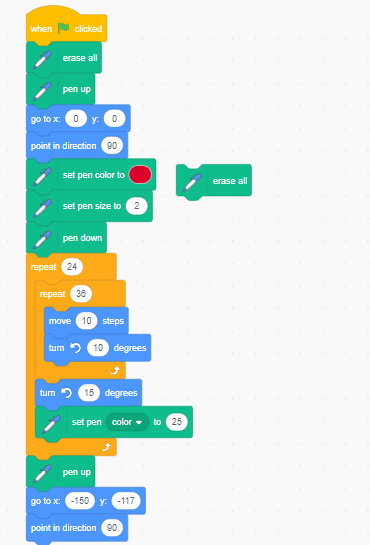
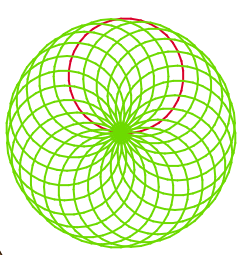
Step 7: Add a *repeat* block and change the value to 24. This is so because on the turn degrees block I used 15 degrees. i.e 24 x25=360

Step 8: Add an inner *repeat* block inside the one in Step 7. Configure the repeat to 36 and the move … steps to 10. Add a turn 10 degrees.

Step 9: The *set pen colour* is to change the colour and make it look nice.

Step 10: After drawing the shape we want our sprite to move away from our shape hence we add a *pen up* block, go to x: y: and set coordinates as -150 and -117

Step 11: Add the point in direction so that the sprite stands upright.



Try to implement more colours to produce a nice flower

**TASK 3.6**

Use Pictoblox to draw a hexagon as the one shown in Figure 3.38.

Shape, polygon

Description automatically generated

Figure 3. 38: Hexagon

**TASK 3.7**

Draw a Triangle using Pictoblox as the one shown in Figure 3.39.

A picture containing shape

Description automatically generated

Figure 3. 39: Triangle

#### 3.2.2.4 Creation of objects and shapes

In this section we are going to use Pictoblox to draw a square filled with any colour. Here are the steps:

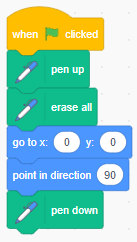
Step 1: Add a *when…clicked* block

Step 2: Add the pen module extension on the list of Blocks. Select pen up and add below the *when..clicked* event block

Step 3: Add the *goto x:…y…*

Step 4: Add the *point in direction 90*

Step 5: Add a *pen down* block. At this point your code will be as follows:

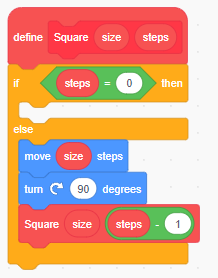


Step 7: Make a new block called square. Add an input called size and another called steps.

Graphical user interface, application

Description automatically generated Your new block or object will look like this:

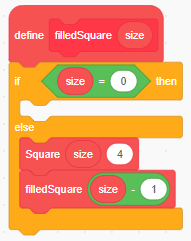
Step 8: Add an *if….else* block from the controls and with the else, add few blocks such that the new script looks as follows



Step 9: make one more block called filledSquare and add an input called size. Add an if… else from controls.

Step 10: Add an = operator and put size = 0.

Step 11: Add the square block in the else segment. The new block script will look as follows:



Step 11: Add the filled Square on the first script we made just below the pen down and modify the value to 100.

The full script loop as follows:

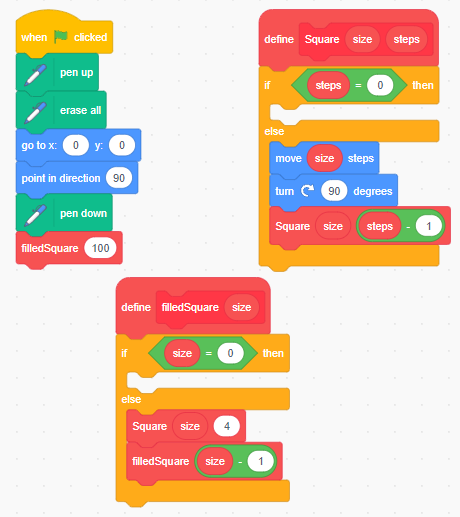


Figure 3. 40: Script for a shaded square

Run your script you will see the square being drawn.



Figure 3. 41: The shaded Square

### 3.2.3 Design a coding solution to a problem incorporating a combination of different programming constructs which include:

#### 3.2.3.1 Sequence

With no conditions, the sequence follows predetermined stages. For smaller programs, sequence constructs are simple to develop. The utilization of sequence structures becomes increasingly difficult as the problems become more complicated. When used in complicated applications, sequence designs are more prone to errors. Here's a small program where sequence designs are used.

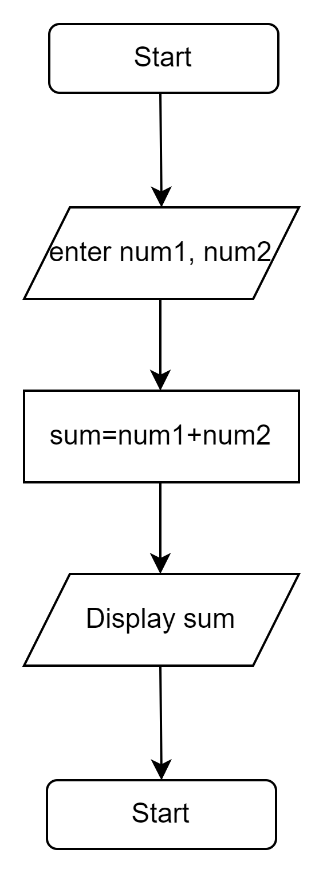
Accept two numbers and add them together. A flowchart depicts a comparable design.

Figure 3. 42: Sequence construct

#### 3.2.3.2 Selection

As discussed earlier, selection requires specific condition to be met for specific statements to be executed. We are going to implement some design techniques to achieve a solution. Consider the following problem:

**Write a program to compute the sum of the two given integer values. If the two values are the same, then return triple their sum.**

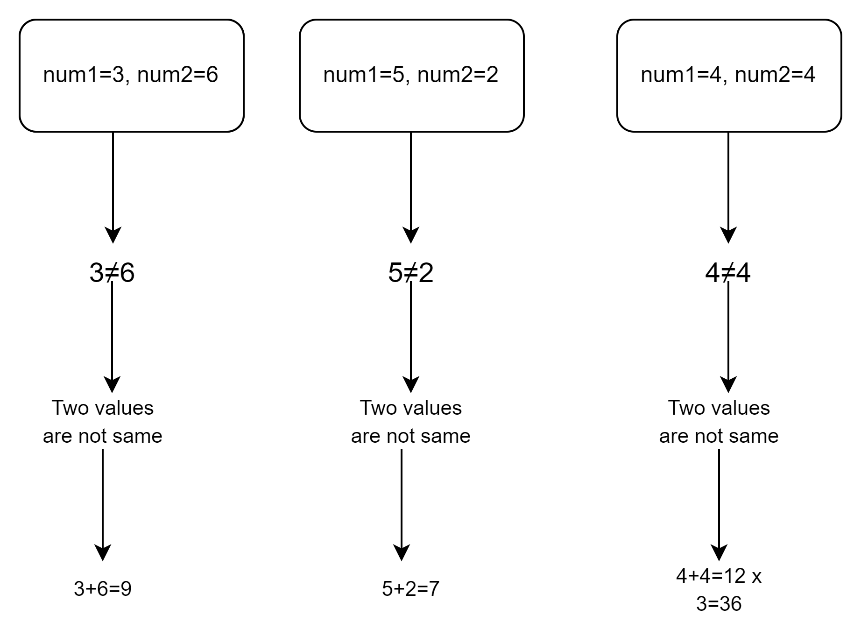


Figure 3. 43:Selection construct design

The selection statement is represented in flowchart form below. The user is required to enter two integer numbers (num1, num2). If the two inputs are equal, the sum is multiplied by three; otherwise, the sum is added.

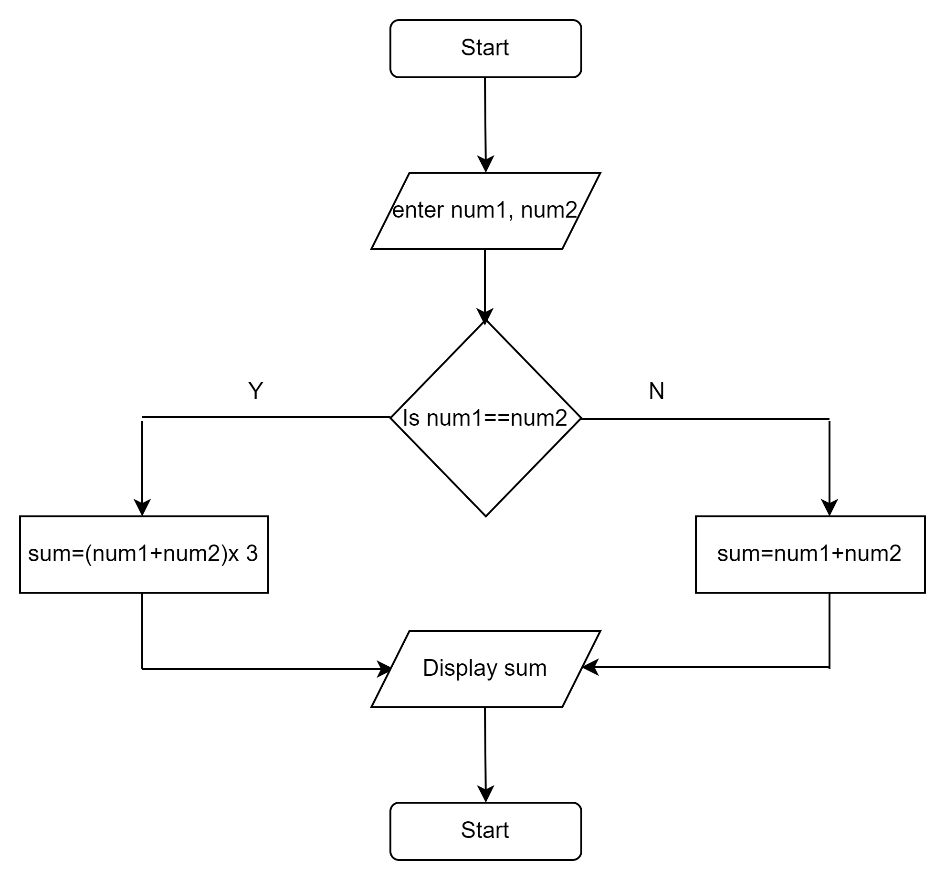


Figure 3. 44: Selection Flowchart

#### 3.2.3.3 Iteration

Iteration allows us to simplify our algorithm by indicating that certain processes will be repeated unless we are instructed differently. This makes algorithm design faster and easier because it eliminates the need for several unneeded steps. There are cases when we have same action which needs to be repeated and instead of writing several lines of the same statement in sequential way, iteration will quicken the process. Here is an example of a problem to be solved:

Write a program to find the first 10 natural numbers. What we know is that the numbers a natural number is a positive integer. So, from our problem, the program is supposed to print

1,2,3,4,5,6,7,8,9,10.

Imagine if we had to write the following statements:

Serial.printl(1)

Serial.printl(2)

Serial.printl(3)

Serial.printl(4)

Serial.printl(5)

Serial.printl(6)

Serial.printl(7)

Serial.printl(8)

Serial.printl(9)

Serial.printl(10)

Already you can see that there are statements which we keep repeating. The only thing changing is value. Here iteration becomes handy in such situation. Figure 3.45 is a flowchart diagram to represent an iteration to print first 10 natural numbers.

A picture containing shape

Description automatically generated

Figure 3. 45: Looping construct flowchart

Even if we were to use sequence construct to represent the above scenario, we will need more than two pages perhaps.- So in essence, iteration saves resources.

While one can adopt a specific construct in design and implementation, complex problems will require a mix of all the constructs. Some programs will start with a sequence construct, have some selection and iteration, and end with a sequence structure. There is no pre-set design to be followed when solving problems as solution implementation differs depending on the decomposition of the problem.

### 3.2.4 Design and develop solutions for specific problems that include computational thinking and applying software engineering principles.

To develop predictions using computational thinking, we must first specify four steps connected to the problem and its solution:

* Decomposition: Creating smaller, more manageable parts of a complex problem or system. In some texts, they use the phrase "problem specification." We begin by assessing the problem, stating it precisely, and establishing the solution criteria.
* Pattern recognition: looking for similarities among and within problems.

Abstraction :An abstraction is a way of focusing only on the most important information, ignoring irrelevant finer details, and using one solution to solve several problems at once.

* Algorithms : developing a step-by-step solution to the problem, or the rules to follow to solve the problem.

**Scenario for computational thinking**

Computational thinking can help you figure out what to tell the computer to do. Suppose you want to advise a friend on which course to take after school. Your first step would be to look at all the courses you know about. You then use the information you've gathered about your friend's interests and goals. Eliminate all other courses that may be out of reach due to cost or entry requirements. Consider the available courses and which one is "better"—this could be the quickest or the easiest course or one that is on demand. Outline the steps needed to complete the application. This is more like an algorithmic phase. Then you'd follow the step-by-step instructions to assist your friend in making an application for the selected course. In this context, planning is akin to computational thinking, whereas following the application process is analogous to programming.

### 3.2.5 Explore lists/arrays (storing and accessing a list of numbers and strings) and containers.

In programming, various collections are used for storing items that have to be stored together.  These are either implemented as arrays or lists. Imagine if we want to store 10 numbers, how do we do it? While variables can be used, you will need 10 variables for the 10 numbers, which has an impact on storage.

**VOCABULARY**

An array can be defined as a group or collection of similar kinds of elements or data items that are stored together in contiguous memory spaces.

To declare an array, define the variable type, specify the name of the array followed by **square brackets** and specify the number of elements it should store.

**Syntax**

*datatype arrayname[size]*

int marks[10];

Arrays store multiple values in a single variable, instead of declaring separate variables for each value.

To insert predefined values into the array, we can use an array literal - place the values in a comma-separated list, inside curly braces:

int marks[5]= {56,89,64,97,34 }.

**Accessing elements of the array**

Each element of an array in C++ has a corresponding number. The value is referred to as an array index. These indices allow us to access the items of an array.

Consider the array mark which we declared before presented in table 3.7.

Table 3. 7: Array Index

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| mark[0] | mark[1] | mark[2] | mark[3] | mark[4] |
| 56 | 89 | 64 | 97 | 34 |
| 0 | 1 | 2 | 3 | 4 |

The indices start at position 0. So, the number of indices is always 1 less than the size of the array. For example, on the small table to access value 64, we would write:

mark[2];

**EXAMPLE 3.29**

//declaring the array

  int marks[5]={56,89,64,97,34 };

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

  Serial.print(marks[2]);

}

void loop() {  }

Output

15:32:38.973 -> 64

Another method to initialize array during declaration:

//declaring the array

  int marks[]={56,89,64,97,34};

On the above declaration, you notice there in no size of the array.

**Displaying all elements from the array**

Iterating through the array elements is necessary to display all elements in the array. For the purposes of this lesson, we are going to use a for ...loop. The code is presented in Example 3.30.

**EXAMPLE 3.30**

//declaring an array

int marks[]={56,89,64,97,34 };

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

  //iterating through the array

  for (int i=0 ; i<5 ; i++ )

  {

    //displaying element at index i

   Serial.println(marks[i]);

  }

}

void loop() {

}

**Replacing an element from the array**

There are times when we want to replace elements of the array. Take, for example, if we discover that we have made an error in capturing our marks and possibly want to replace them with a known mark.

**Syntax**

*Arrayname[index]=value;*

Consider the array marks:

int marks[]={56,89,64,97,34 };

For example, we are going to replace 64 with 44 as shown below.

marks[2]=44;

Let us try to represent collections of items using a visual application. In Pictoblox, there is no special category for lists, but we can go to the very bottom under the variables and click on "Make a list" and name your new list. You will see a lot of blocks associated with the list. Also on the stage, you will see the list name.

**Steps**

Step 1: Add the *when clicked* block.

Step 2: Click on variables and select *make a list*. Type in the word marks as the list name.

Step 3: Click on *sensing* and take the *ask… and wait* block.

Step 4: From variables select the *add… to marks* blocks.

Step 5: Inside the *add.. to marks* block, insert the *answer* block. Type in the first value of the list 56 and click the *when clicked* block. The code will look as follows:



Figure 3. 46:Lists

You can repeat the process and add the other elements to the list. However, lists show a count of the elements, which is why it is displaying 1 and 2. Let's complete the process and store all five numbers as declared in marks. The final list will look as follows:Chart, bar chart

Description automatically generated

**Inserting an element into the list**

It is possible with Scratch to insert a mark at a specific position. Let’s insert 55 after value 64 in our list. Here is the code and the final output after execution.

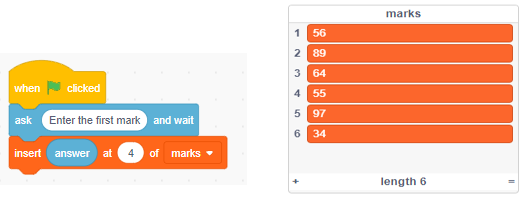


Figure 3. 47:Inserting elements in a list

Figure 3.47 shows the value 55 inserted after 64.

**Deleting an element from the list**

To delete an item from the list, you simply must use the *delete …. of ….* arrayname block. For example to delete value 55 add the following block:



**Replacing an element from the list**

Sometimes, when working with lists, you might want to replace an element. This does not reduce or increase the size of a list. We are going to replace the value 89 at position 2 with 99. So, we use the *replace item… of … with ….*



**NOTE**

When replacing elements, you do not type in the element you want to replace but rather the position.

Sometimes you may want to check the number of elements in the list. This is achieved by using *length of …* block as shown below.

# FORMATIVE ASSESSMENT 3.2 INDIVIDUAL TASK

3.2.1 Define the term visual programming language? (2)

3.2.2 List any FOUR common examples of visual programming languages. (4)

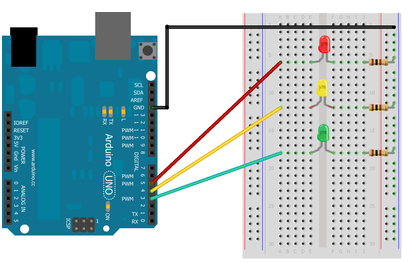
3.2.3 **Arduino Traffic Light Project**

We want do design traffic lights. The sequence of traffic lights is Greed, Amber and then Red. Use Pictoblox to depict the scenario and Arduino Uno.

HINT

The concept is same as the way we designed a traffic light using Raspberry Pi.

**Sample connections**



(10)

3.2.4 Define the term variable. (2)

3.2.5 Differentiate between real value division and integer division. (4)

3.2.6 List THREE types of programming constructs. (3)

**[Total =25 marks]**

# SUMMATIVE ACTIVITY 3.3 INDIVIDUAL TASK

3.3.1 What is the difference between an Arduino and a raspberry Pi. (2)

3.3.2 List the three main types of pins on an Arduino board. (3)

3.3.3 What are the four key aspects of computational thinking? (4)

3.3.4 Arduino has two default functions i.e void setup() and void loop(). Explain purpose of each of them. (4)

3.3.5 Explain the use of inbuilt function pinMode() with examples. (4)

3.3.6 What’s the difference between digitalWrite() and analogWrite() in Arduino? (4)

3.3.7 Explain in brief how you can upload code to an Arduino board. (2)

3.3.8 Consider the code below and answer the question which follows.

void setup() {

  // put your setup code here, to run once:

  Serial.begin(9600);

}

void loop() {

 Serial.println("Arduino count down for loop");

   for (int i=10; i>=1; i--) {

      Serial.print("i is : ");

      Serial.println(i);

   }

delay(500);

}

After compiling the code and uploading it to the Arduino board, the output continuously display in a looping mode. Rewrite the code to stop printing when the loop reaches its limit. (2)

3.3.9 Write a script in using Pictoblox to display the decimal equivalent of a binary number entered through the keyboard. (5)

3.3.10 Use an Pictoblox to calculate area of a rectangle whose length and width is entered by the user. (4)

**[Total = 34 marks]**

# References

<https://docs.arduino.cc/tutorials/>

<https://toptechboy.com/arduino-lessons/>

<https://thestempedia.com/tutorial-hub/pictoblox-tutorial/>