

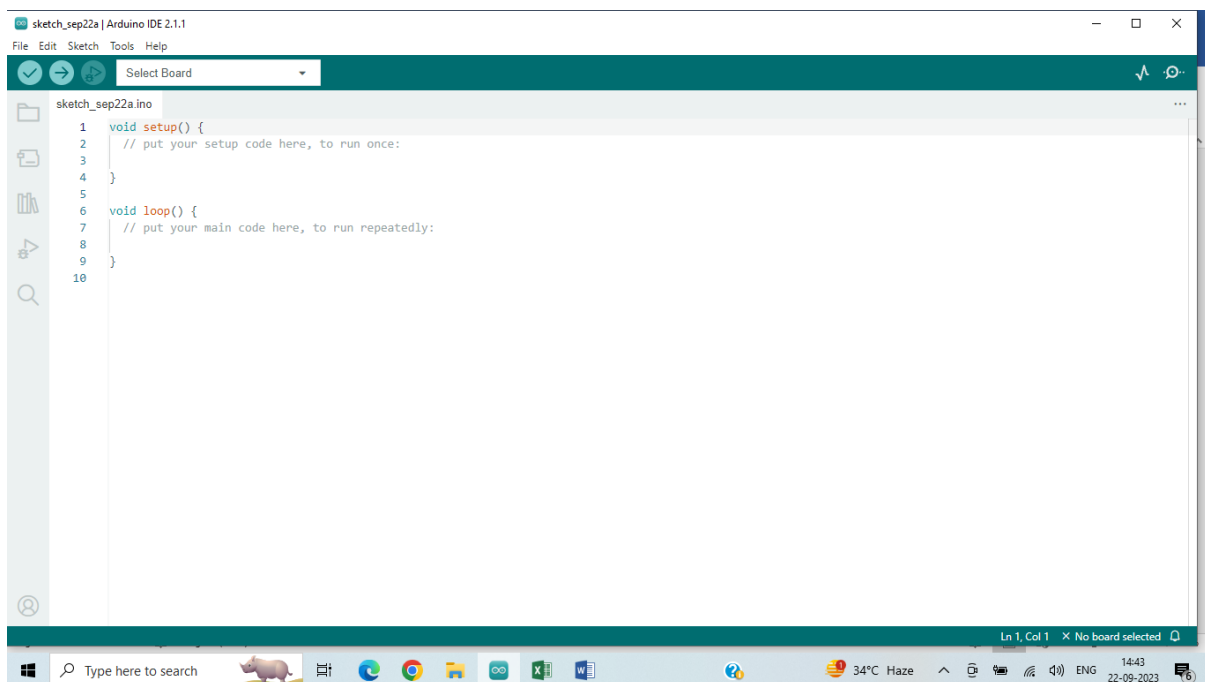
Experiment No. 2(A)

Objective: Arduino IDE and Operators in IDE

Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to Arduino hardware to upload programs and communicate with them.

Writing Sketches


Programs written using Arduino Software (IDE) are called **sketches**. These sketches are written in text editor and are saved with file extension. ino. Editor has features for cutting/pasting and for searching/replacing text. Message area gives feedback while saving and exporting and also displays errors. Console displays text output by Arduino Software (IDE), including complete error messages and other information. Bottom right hand corner of the window displays the configured board and serial port. Toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open serial monitor.




NB: Versions of the Arduino Software (IDE) prior to 1.0 saved sketches with the extension .pde. It is possible to open these files with version 1.0, you will be prompted to save the sketch with the. ino extension on save.




Verify Checks your code for errors compiling it.

 **Upload** Compiles your code and uploads it to the configured board. If you are using an external programmer with your board, you can hold down the "shift" key on your computer when using this icon. The text will change to "Upload using Programmer"

 **New** Creates a new sketch.

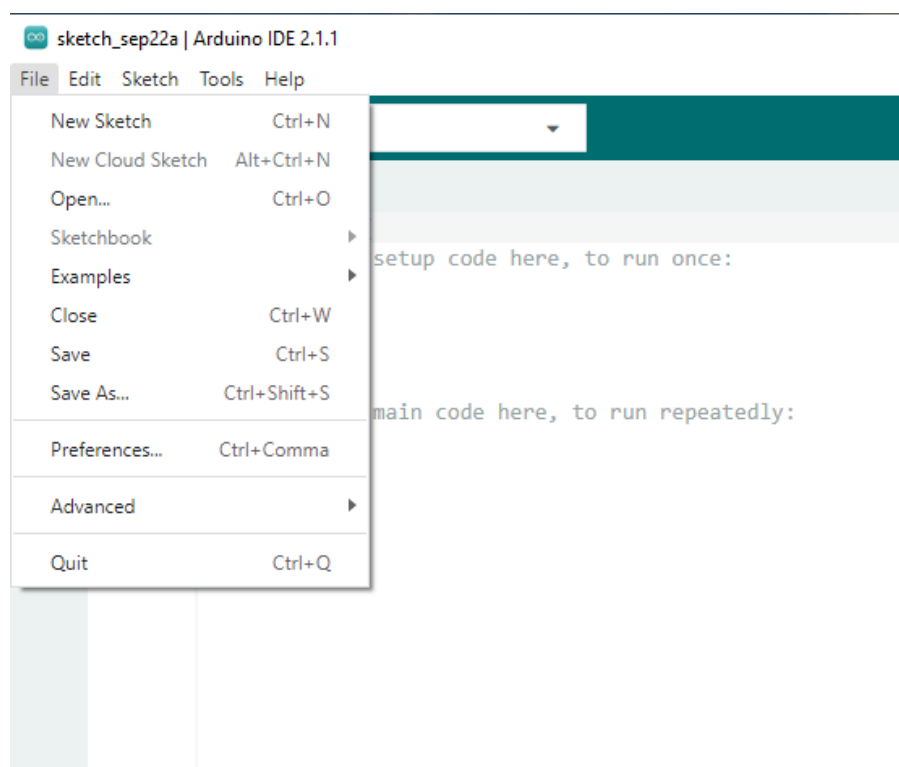
 **Open** Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window overwriting its content.

Note: due to a bug in Java, this menu doesn't scroll; if you need to open a sketch late in the list, use the **File | Sketchbook** menu instead.

 **Save** Saves your sketch.

 **Serial Monitor** Opens the serial monitor.

Additional commands are found within the five menus: **File**, **Edit**, **Sketch**, **Tools**, **Help**. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.



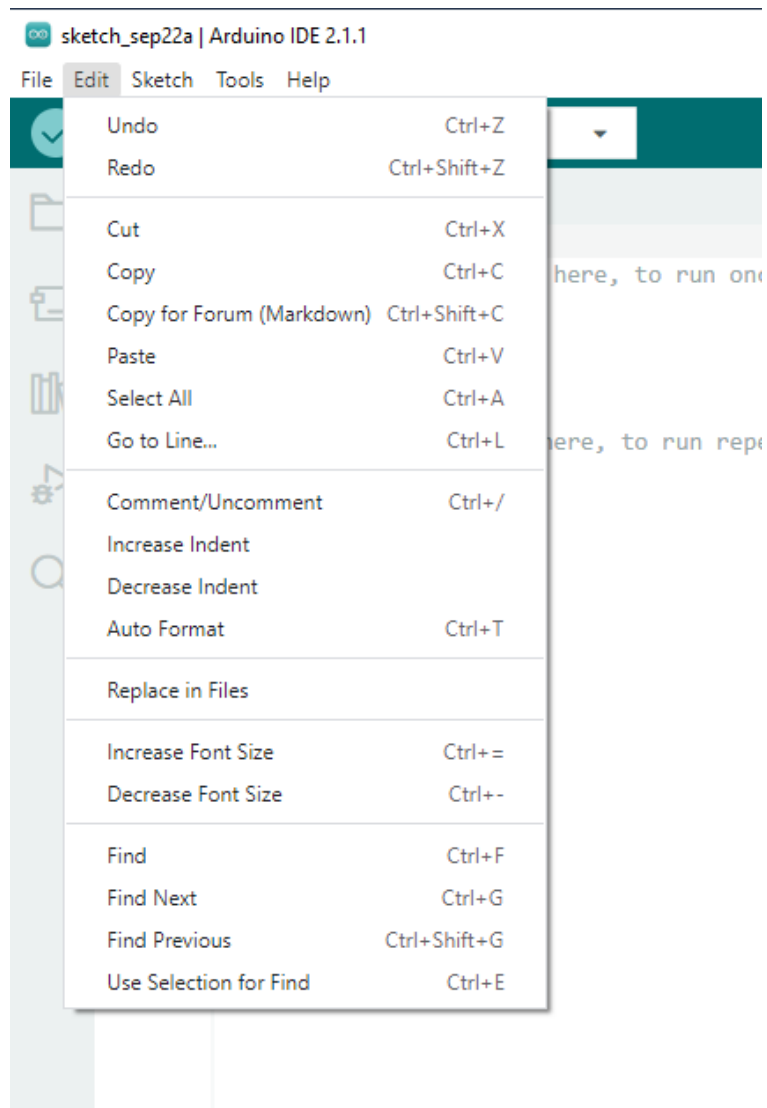
File

- *New* Creates a new instance of the editor, with the bare minimum structure of a sketch already in place.
- *Open* Allows to load a sketch file browsing through the computer drives and folders.
- *Open Recent* Provides a short list of the most recent sketches, ready to be opened.
- *Sketchbook* Shows the current sketches within the sketchbook folder structure; clicking on any name opens the corresponding sketch in a new editor instance.

- *Examples* Any example provided by the Arduino Software (IDE) or library shows up in this menu item. All the examples are structured in a tree that allows easy access by topic or library.
- *Close* Closes the instance of the Arduino Software from which it is clicked.
- *Save* Saves the sketch with the current name. If the file hasn't been named before, a name will be provided in a "Save as.." window.
- *Save as...* Allows to save the current sketch with a different name.
- *Page Setup* It shows the Page Setup window for printing.
- *Print* Sends the current sketch to the printer according to the settings defined in Page Setup.
- *Preferences* Opens the Preferences window where some settings of the IDE may be customized, as the language of the IDE interface.
- *Quit* Closes all IDE windows. The same sketches open when Quit was chosen will be automatically reopened the next time you start the IDE.

Edit

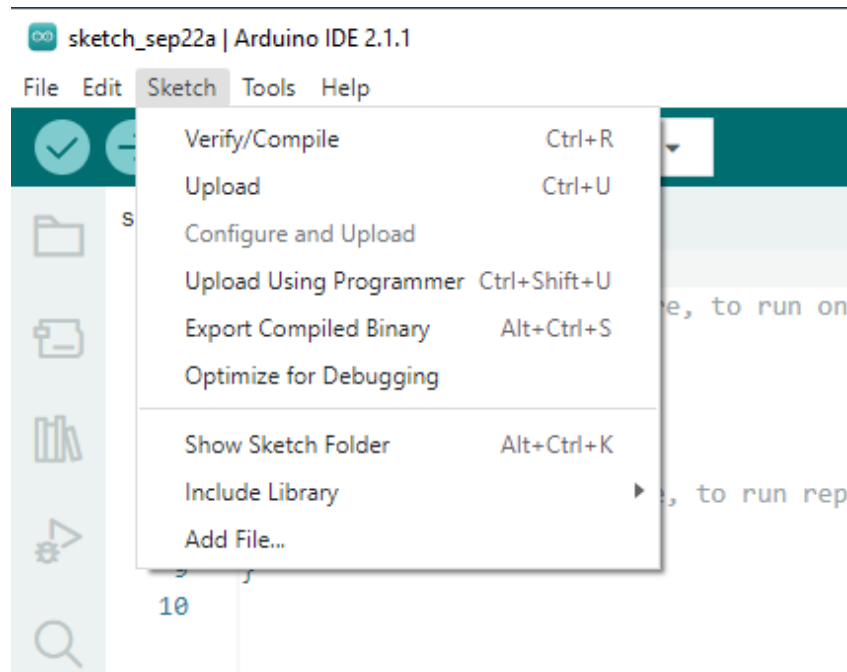
- *Undo/Redo* Goes back of one or more steps you did while editing; when you go back, you may go forward with Redo.
- *Cut* Removes the selected text from the editor and places it into the clipboard.
- *Copy* Duplicates the selected text in the editor and places it into the clipboard.
- *Copy for Forum* Copies the code of your sketch to the clipboard in a form suitable for posting to the forum, complete with syntax coloring.
- *Copy as HTML* Copies the code of your sketch to the clipboard as HTML, suitable for embedding in web pages.
- *Paste* Puts the contents of the clipboard at the cursor position, in the editor.
- *Select All* Selects and highlights the whole content of the editor.
- *Comment/Uncomment* Puts or removes the // comment marker at the beginning of each selected line.
- *Increase/Decrease Indent* Adds or subtracts a space at the beginning of each selected line, moving the text one space on the right or eliminating a space at the beginning.
- *Find* Opens the Find and Replace window where you can specify text to search inside the current sketch according to several options.
- *Find Next* Highlights the next occurrence - if any - of the string specified as the search item in the Find window, relative to the cursor position.
- *Find Previous* Highlights the previous occurrence - if any - of the string specified as the search item in the Find window relative to the cursor position.



Sketch

- *Verify/Compile* Checks your sketch for errors compiling it; it will report memory usage for code and variables in the console area.
- *Upload* Compiles and loads the binary file onto the configured board through the configured Port.
- *Upload Using Programmer* This will overwrite the bootloader on the board; you will need to use Tools > Burn Bootloader to restore it and be able to Upload to USB serial port again. However, it allows you to use the full capacity of the Flash memory for your sketch. Please note that this command will NOT burn the fuses. To do so *Tools* -> *Burn Bootloader* command must be executed.
- *Export Compiled Binary* Saves a .hex file that may be kept as archive or sent to the board using other tools.
- *Show Sketch Folder* Opens the current sketch folder.

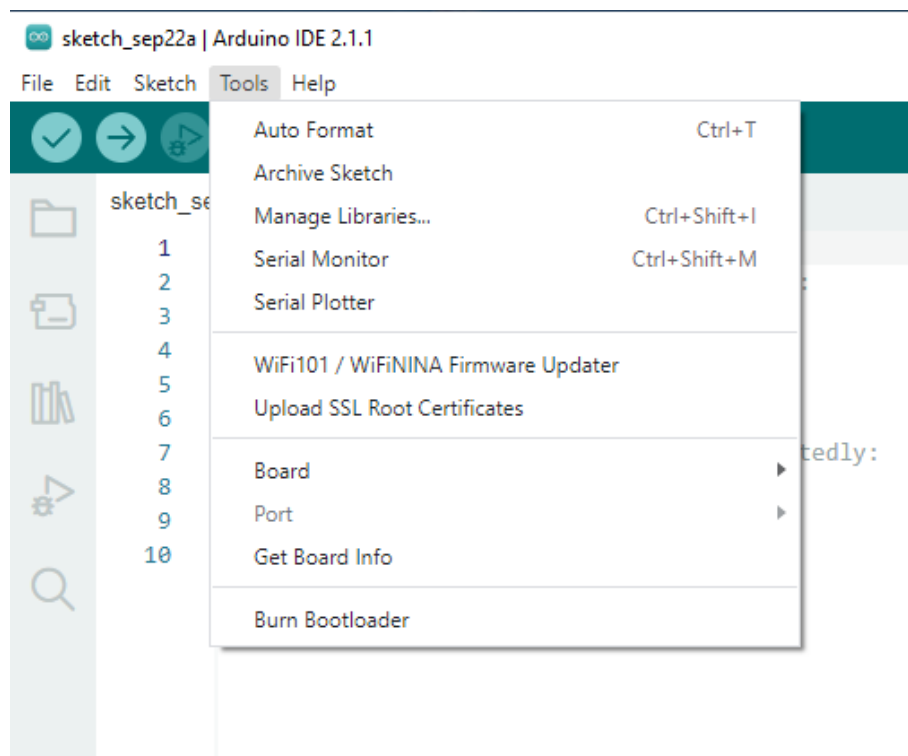
- *Include Library* Adds a library to your sketch by inserting `#include` statements at the start of your code. For more details, see libraries below. Additionally, from this menu item you can access the Library Manager and import new libraries from .zip files.
- *Add File* Adds a supplemental file to the sketch (it will be copied from its current location). The file is saved to the data subfolder of the sketch, which is intended for assets such as documentation. The contents of the data folder are not compiled, so they do not become part of the sketch program.



Tools

- *Auto Format* This formats your code nicely: i.e. indents it so that opening and closing curly braces line up, and that the statements inside curly braces are indented more.
- *Archive Sketch* Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.
- *Fix Encoding & Reload* Fixes possible discrepancies between the editor char map encoding and other operating systems char maps.
- *Serial Monitor* Opens the serial monitor window and initiates the exchange of data with any connected board on the currently selected Port. This usually resets the board, if the board supports Reset over serial port opening.
- *Board* Select the board that you're using. See below for descriptions of the various boards.
- *Port* This menu contains all the serial devices (real or virtual) on your machine. It should automatically refresh every time you open the top-level tools menu.

- *Programmer* For selecting a hardware programmer when programming a board or chip and not using the onboard USB-serial connection. Normally you won't need this, but if you're burning a bootloader to a new microcontroller, you will use this.
- *Burn Bootloader* The items in this menu allow you to burn a bootloader onto the microcontroller on an Arduino board. This is not required for normal use of an Arduino board but is useful if you purchase a new ATmega microcontroller (which normally come without a bootloader). Ensure that you've selected the correct board from the **Boards** menu before burning the bootloader on the target board. This command also set the right fuses.



Help

Here you find easy access to a number of documents that come with the Arduino Software (IDE). You have access to Getting Started, Reference, this guide to the IDE and other documents locally, without an internet connection. The documents are a local copy of the online ones and may link back to our online website.

- *Find in Reference* This is the only interactive function of the Help menu: it directly selects the relevant page in the local copy of the Reference for the function or command under the cursor.

Sketchbook

The Arduino Software (IDE) uses the concept of a sketchbook: a standard place to store your programs (or sketches). The sketches in your sketchbook can be opened from the **File** >

Sketchbook menu or from the **Open** button on the toolbar. The first time you run the Arduino software, it will automatically create a directory for your sketchbook. You can view or change the location of the sketchbook location from with the **Preferences** dialog. Beginning with version 1.0, files are saved with a .ino file extension. Previous versions use the .pde extension. You may still open .pde named files in version 1.0 and later, the software will automatically rename the extension to .ino.

Tabs, Multiple Files, and Compilation

Allows you to manage sketches with more than one file (each of which appears in its own tab). These can be normal Arduino code files (no visible extension), C files (.c extension), C++ files (.cpp), or header files (.h).

Libraries

Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from the **Sketch > Import Library** menu. This will insert one or more **#include** statements at the top of the sketch and compile the library with your sketch.

Serial Monitor

This displays serial sent from the Arduino board over USB or serial connector. To send data to the board, enter text and click on the "send" button or press enter. Choose the baud rate from the drop-down menu that matches the rate passed to **Serial.begin** in your sketch.

The types of Operators classified in Arduino are:

1. Arithmetic Operators
2. Compound Operators
3. Boolean Operators
4. Comparison Operators
5. Bitwise Operators

Arithmetic Operators

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

Assignment Operator (=): Used to set variable's value. It is quite different from equal symbol (=) normally used in mathematics.

Addition (+): Used for addition of two numbers. For example, $P + Q$.

Subtraction (-): Used to subtract one value from the another. For example, $P - Q$.

Multiplication (*): Used to multiply two numbers. For example, $P * Q$.

Division (/): Used to determine result of one number divided with another. For example, P/Q .

Modulo (%): Used to calculate remainder after division of one number by another number.

Code 1: Add two numbers and print their result on serial monitor

```
int b;  
void setup ( )  
{  
  Serial.begin( 9600 );  
}  
void loop ( )  
{  
  b = 5 + 2;  
  Serial.println(b);  
}
```

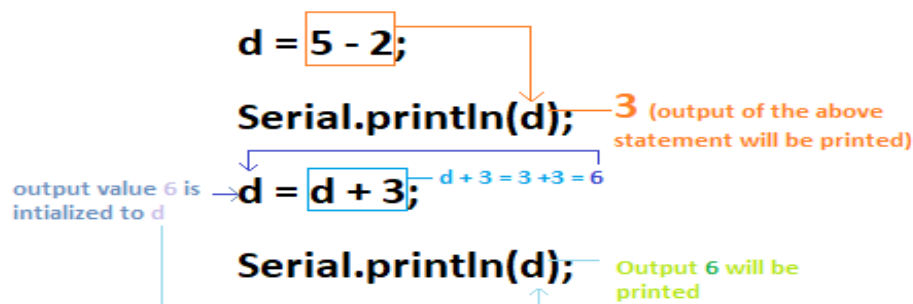
Result: 7

Code 2: Run the below code to check working of arithmetic operators

```
int d;  
void setup ( )  
{  
  Serial.begin( 9600 );  
}  
void loop ( )  
{  
  d = 5 - 2;  
  Serial.println(d);  
  d = d + 3;  
  Serial.println(d);  
}
```

Result: 3

6



If we want decimal values to be printed, we need to use the **float** instead of **int**.

Code 3: Use float datatype and divide operator to write a code.

```
float b;
```



```

void setup ( )
{
  Serial.begin( 9600 );
}
void loop ( )
{
  b = 20.0 / 3; // decimal value is used to force the compiler to print decimal
                value.
  Serial.println(b);
}

```

Code 4: Write a code to verify BODMAS

```

int c;
void setup ( )
{
  Serial.begin( 9600 );
}
void loop ( )
{
  c = 2 * 3 / (2 + 1) + 4;
  Serial.println(c);
}

```

Output: 6

the number inside parentheses will be
calculated first

$$b = 2 * 3 / (2 + 1) + 4$$

$$b = 2 * 3 / 3 + 4$$

As discussed above, from left to right calculations will be performed i.e.
first multiplication and then division

$$b = 6 / 3 + 4$$

$$b = 2 + 4$$

$$b = 6$$

Compound Operators

Compound operators perform two or more calculations at once. Result of right operand is assigned to left operand. Same condition will apply to all the compound operators, which are listed below:

Let's consider a variable **b**.

- **b ++**

Here, $b = b + 1$. It is called **increment operator**.

- **b +=**

For example, $b += 4$. It means, $b = b + 4$.

- **b --**

Here, $b = b - 1$. It is called as the **decrement operator**.

- **$b -=$**

For example, $b -= 3$. It means, $b = b - 3$.

- **$b *=$**

For example, $b *= 6$. It means, $b = b * 6$.

- **$b /=$**

For example, $b /= 5$. It means, $b = b / 5$.

- **$b \% =$**

For example, $b \% = 2$. It means, $b = b \% 2$.

Now, let's use the above operators with two variables, b and c .

- $b += c$ ($b = b + c$)
- $b -= c$ ($b = b - c$)
- $b *= c$ ($b = b * c$)
- $b /= c$ ($b = b / c$)
- $b \% = c$ ($b = b \% c$)

Boolean Operators

The Boolean Operators are **NOT (!)**, **Logical AND (& &)**, and **Logical OR (| |)**.

- **Logical AND (& &):** The result of the condition is true if both the operands in the condition are true.
 - **if** ($a == b \ \& \ \& \ b == c$)

Above statement is true if both conditions are true. If any of conditions is false, statement will be false.

- **Logical OR (| |):** Result of condition is true, if either of variables in condition is true.
 - **if** ($a > 0 \ || \ b > 0$)

The above statement is true, if either of above condition ($a > 0$ or $b > 0$) is true.

- **NOT (!):** It is used to reverse the logical state of the operand.
 - For example, $a! = 2$.

The NOT operator returns value 1 or TRUE when specified operand is FALSE. It also reverses the value of the specified expression.

Comparison Operators

The comparison operators are used to compare the value of one variable with the other. The comparison operators are listed below:

- **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.

Example: Consider the below code.

```
1. int b;
2. int c ;
3. void setup ( )
4. {
5.   Serial.begin( 9600 );
6. }
7. void loop ( )
8. {
9.   b = 3;
10.  c = 5;
11.  if ( b < 4 )
12.    Serial.println(b);
13.  if ( c < 4)
14.    Serial.println( c);
15. }
```

Output: 3

In above code, if any of two statements is correct, corresponding value of variable will be printed. Here, only first condition is correct. Hence, value of b will be printed.

- **greater than (>):** Less than operator checks that value of left side of a statement is greater than right side. Statement is true if the condition is satisfied. For example, $a > b$. If a is greater than b, the condition is true, else false.
- **equal to (==):** It checks the value of two operands. If the values are equal, the condition is satisfied. For example, $a = b$. The above statement is used to check if the value of a is equal to b or not.
- **not equal to (!=):** It checks the value of two specified variables. If the values are not equal, the condition will be correct and satisfied. For example, $a \neq b$.
- **less than or equal to (<=):** The less or equal than operator checks that the value of left side of a statement is less or equal to the value on right side. The statement is true if either of the condition is satisfied. For example, $a \leq b$. It checks the value of a is less or equal than b.
- **greater than or equal to (>=):** The greater or equal than operator checks that the value of the left side of a statement is greater or equal to the value on the right side of that statement. The statement is true if the condition is satisfied. For example, $a \geq b$. It checks the value of a is greater or equal than b. If either of the conditions satisfies, the statement is true.

Bitwise Operators

The Bitwise operators operate at the **binary level**. These operators are quite easy to use. There are various bitwise operators. Some of the popular operators are listed below:

1. **bitwise NOT (~):** The bitwise NOT operator acts as a complement for reversing the bits. For example, if $b = 1$, the NOT operator will make the value of $b = 0$.
 - 0 0 1 1 // Input or operand 1 (decimal value 3)
 - 1 1 0 0 // Output (reverses the input bits) decimal value is 12
2. **bitwise XOR (^):** The output is 0 if both the inputs are same, and it is 1 if the two input bits are different. For example,
 1. 1 0 0 1 // input 1 or operand 1
 2. 0 1 0 1 // input 2
 3. 1 1 0 0 // Output (resultant - XOR)
3. **bitwise OR (|):** Output is 0 if both of the inputs in the OR operation are 0. Otherwise, the output is 1. The two input patterns are of 4 bits. For example,
 1. 1 1 0 0 // input 1 or operand 1
 2. 0 0 0 1 // input 2
 3. 1 1 0 1 // Output (resultant - OR)
4. **bitwise AND (&):** Output is 1 if both the inputs in the AND operation are 1. Otherwise, the output is 0. The two input patterns are of 4 bits. For example,
 1. 1 1 0 0 // input 1 or operand 1
 2. 0 1 0 1 // input 2
 3. 0 1 0 0 // Output (resultant - AND)
5. **bitwise left shift (<<):** The left operator is shifted by the number of bits defined by the right operator.
6. **bitwise right shift (>>):** The right operator is shifted by the number of bits defined by the left operator.

Experiment No. 2(B)

Aim: Frequently used functions in Arduino IDE.

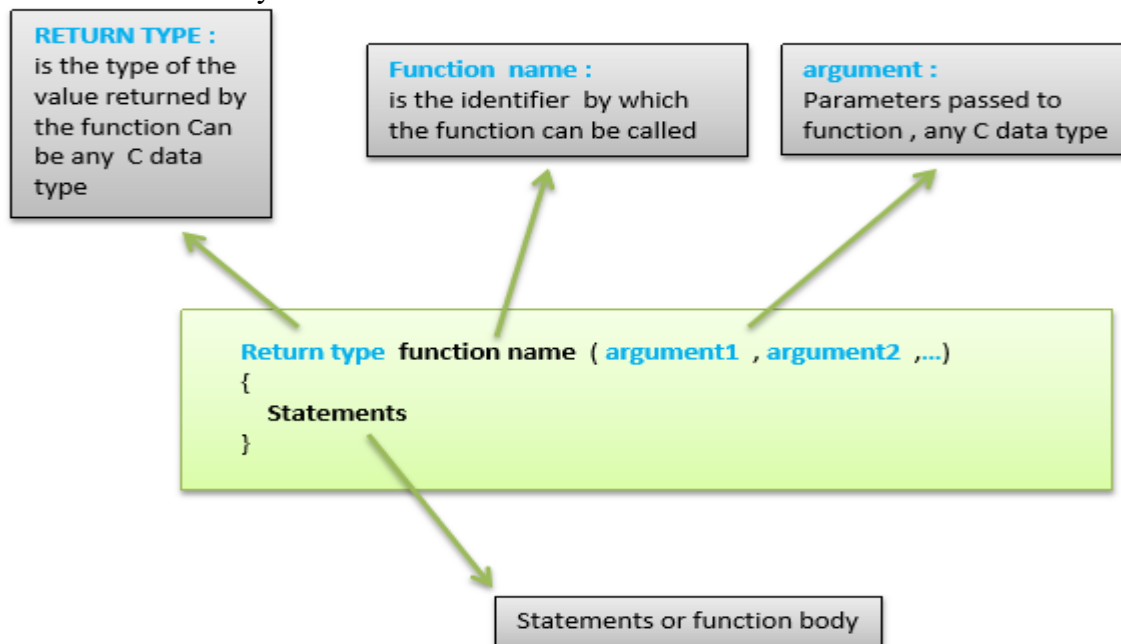
Functions allow structuring the programs in segments of code to perform individual tasks. The typical case for creating a function is when one needs to perform the same action multiple times in a program.

Standardizing code fragments into functions has several advantages –

- Functions help the programmer stay organized. Often this helps to conceptualize the program.
- Functions codify one action in one place so that the function only has to be thought about and debugged once.
- This also reduces chances for errors in modification, if the code needs to be changed.
- Functions make the whole sketch smaller and more compact because sections of code are reused many times.
- They make it easier to reuse code in other programs by making it modular, and using functions often makes the code more readable.

There are two required functions in an Arduino sketch or a program i.e. setup () and loop(). Other functions must be created outside the brackets of these two functions.

The most common syntax to define a function is –



Function Declaration

A function is declared outside any other functions, above or below the loop function.

We can declare the function in two different ways –

The first way is just writing the part of the function called **a function prototype** above the loop function, which consists of –

- Function return type
- Function name
- Function argument type, no need to write the argument name

Function prototype must be followed by a semicolon (;).

The following example shows the demonstration of the function declaration using the first method.

Arduino IDE offers a wide range of functions and libraries to help you program your Arduino microcontroller effectively. Here are some frequently used functions along with coding examples:

1. **pinMode()**: Configures a pin as INPUT or OUTPUT.

```
pinMode(ledPin, OUTPUT);
```

Example: // LED connected to pin 13

```
const int ledPin = 13;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  digitalWrite(ledPin, HIGH); // Turn on the LED
  delay(1000);                // Wait for 1 second
  digitalWrite(ledPin, LOW);  // Turn off the LED
  delay(1000);                // Wait for 1 second
}
```

2. **digitalWrite()**: Writes a HIGH or LOW value to a digital pin.

```
digitalWrite(ledPin, HIGH);
```

Example: // LED connected to pin 13

```
const int ledPin = 13;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  digitalWrite(ledPin, HIGH); // Turn on the LED
  delay(1000);                // Wait for 1 second
  digitalWrite(ledPin, LOW);  // Turn off the LED
  delay(1000);                // Wait for 1 second
}
```

3. **digitalRead()**: Reads the value from a digital pin (HIGH or LOW).

```
int buttonState = digitalRead(buttonPin);
```

Example: // Button connected to pin 2

```
const int buttonPin = 2;
void setup() {
  pinMode(buttonPin, INPUT_PULLUP);
  Serial.begin(9600);
}
void loop() {
  int buttonState = digitalRead(buttonPin);
  Serial.println(buttonState);
  delay(1000);
}
```

4. **analogWrite()**: Writes an analog value (PWM) to a pin.

```
analogWrite(ledPin, brightness);
```

5. **analogRead()**: Reads the analog value from an analog pin (0-1023).

```
int sensorValue = analogRead(A0);
```

Example: // Analog sensor connected to analog pin 0

```
const int sensorPin = A0;
void setup() {
  Serial.begin(9600);
}
void loop() {
  int sensorValue = analogRead(sensorPin);
  Serial.println(sensorValue);
  delay(1000);
}
```

6. **delay()**: Pauses the program for a specified number of milliseconds.

```
delay(1000); // Pause for 1 second
```

Example:

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000); // Wait for 1 second
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000); // Wait for 1 second
}
```

7. **Serial.begin()**: Initializes serial communication.

```
Serial.begin(9600); // Initialize serial communication at 9600 bps
```

8. **Serial.println()**: Prints data to the Serial Monitor.

```
Serial.println("Hello, Arduino!");
```

Example:

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.print("Hello, World!");
  delay(1000);
}
```

9. **if() statement**: Conditional execution based on a condition.

```
if (temperature > 25) { // Code to execute when temperature is above 25°C }
```

10. **for() loop**: Repeats a block of code a specified number of times.

```
for (int i = 0; i < 5; i++) { // Code to repeat 5 times }
```

11. **while() loop:** Repeats a block of code as long as a condition is true.

```
while (buttonState == HIGH) { // Code to execute while the button is pressed }
```

12. **Functions:** Custom functions allow you to organize and reuse code.

```
void blinkLED(int pin, int duration)
{
    digitalWrite(pin, HIGH);
    delay(duration);
    digitalWrite(pin, LOW);
    delay(duration);
}
```