Arduino Programming

Commonly used functions

. Serial.print(value) or Serial.print(value, numberOfDigitsToShowAfterDecimalPoint) - prints value keeps cursor beside it

- Serial.println(value) or Serial.println(value, numberOfDigitsToShowAfterDecimalPoint)
  - ✓Prints the value to the Arduino IDE's Serial Monitor so you can view Arduino's output on your computer takes the cursor to a newline
- pinMode(pin, mode) INPUT or OUTPUT
  - ✓Configures a ~~digital~~ pin to read (input) or write (output) a digital value
- digitalRead(pin)
  - ✓Reads a digital value (HIGH or LOW) on a pin set for input
- digitalWrite(pin, value)
  - ✓Writes the digital value (HIGH or LOW) to a pin set for output

.analogRead(pin) - reads a analog value on a pin set for input
.analogWrite(pin, value) - writes the value on a pin set for output
.delay(value) - delays with value in miliseconds

A Typical Arduino Sketch

- Programs for Arduino are usually referred to as sketches.
- Sketches contain code—the instructions the board will carry out
- Code that needs to run only once (such as to set up the board for your application) must be placed in the setup function.
- Code to be run continuously after the initial setup has finished goes into the loop function

Using Simple Primitive Types (Variables)

- Arduino has different types of variables to efficiently represent values. You want to know how to select and use these Arduino data types.

Table 2-1. Arduino data types for 8-bit boards such as the Uno

Numeric types	Bytes	Range	Use
`int`	2	–32768 to 32767	Represents positive and negative integer values.
`unsigned int`	2	0 to 65535	Represents only positive values; otherwise, similar to `int`.
`long`	4	–2147483648 to 2147483647	Represents a very large range of positive and negative values.
`unsigned long`	4	4294967295	Represents a very large range of positive values.
`float`	4	3.4028235E+38 to –3.4028235E+38	Represents numbers with fractions; use to approximate real-world measurements.
`double`	4	Same as `float`	In Arduino, `double` is just another name for `float`.
`bool`	1	`false` (0) or `true` (1)	Represents true and false values.
`char`	1	–128 to 127	Represents a single character. Can also represent a signed numeric value between –128 and 127.
`byte`	1	0 to 255	Similar to `char`, but for unsigned values.

Other types	Use
`String`	Represents a sequence of characters typically used to contain text.
`void`	Used only in function declarations where no value is returned.

Using Simple Primitive Types (Variables)

Table 2-2. Arduino data types for 32-bit boards such as the Zero and 101

Numeric types	Bytes	Range	Use
`short int`	2	–32768 to 32767	Same as `int` on 8-bit boards.
`unsigned short int`	2	0 to 65535	Same as `unsigned int` on 8-bit boards.
`int`	4	–2147483648 to 2147483647	Represents positive and negative integer values.
`unsigned int`	4	0 to 4294967295	Represents only positive values; otherwise, similar to `int`.
`long`	4	–2147483648 to 2147483647	Same as `int`.
`unsigned long`	4	4294967295	Same as `unsigned int`.
`float`	4	±3.4028235E+38	Represents numbers with fractions; use to approximate real-world measurements.
`double`	8	±1.7976931348623158E+308	32-bit boards have much greater range and precision than 8-bit boards.
`bool`	1	`false` (0) or `true` (1)	Represents true and false values.
`char`	1	–128 to 127	Represents a single character. Can also represent a signed value between –128 and 127.
`byte`	1	0 to 255	Similar to `char`, but for unsigned values.

Other types	Use
`String`	Represents a sequence of characters typically used to contain text.
`void`	Used only in function declarations where no value is returned.

Using Simple Primitive Types (Variables)

- Variables declared using int will be suitable for numeric values if the values do not exceed the range.
- Choose a type that specifically suits your application. This is especially important if you are calling library functions that return values other than int.
- bool (boolean) types have two possible values: true or false.

Using Floating-Point Numbers

- Floating-point numbers are used for values expressed with decimal points (this is the way to represent fractional values).

```
float a = 123.456789;    // Up to 7 digits
float b = -45.6789123;   // Up to 7 digits

void setup() {

  Serial.begin(9600);

  Serial.print("a: ");
  Serial.println(a, 7);
  Serial.print("b: ");
  Serial.println(b, 7);}
```

```
void loop() {
  bool isAGreaterThanB;

    isAGreaterThanB = (a > b);
     a=a-50;
    if (isAGreaterThanB) {

    Serial.println("Variable 'a' is greater
than variable 'b'");
  }
else {
    Serial.println("Variable 'a' is not
greater than variable 'b'");
  }
  delay(2000);
}
```

Working with Groups of Values

- You want to create and use a group of values (called arrays). The arrays may be a simple list or they could have two or more dimensions.

```
int inputPins[] = {2, 3, 4, 5};
int ledPins[] = {10, 11, 12, 13};
void setup() {
  for (int i = 0; i < 4; i++)
{       pinMode(ledPins[i], OUTPUT);
    pinMode(inputPins[i], INPUT);
  }
}
void loop() {
  for (int i = 0; i < 4; i++) {
 int val;
    val = digitalRead(inputPins[i]);
    if (val == LOW) {
      digitalWrite(ledPins[i], HIGH);
      delay(1000);
    } else {
      digitalWrite(ledPins[i], LOW);
      delay(1000);
    }
  }
}
```

Using Arduino String Functionality

- You want to manipulate text. You need add bits together, and determine the number of characters.

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

  String text1 = "This text, ";
  String text2 = "has more words!";
    String concatenatedText = text1 + text2;
    int numberOfCharacters =
concatenatedText.length();
Serial.println("Text 1:" + text1);
  Serial.print("Number of characters in Text 1:
");
  Serial.println(text1.length());
  Serial.println("Text 2:" + text2);
  Serial.print("Number of characters in Text 2:
");
  Serial.println(text2.length());
  Serial.println("Concatenated Text: " +
concatenatedText);
  Serial.print("Number of Characters in
Concatenated Text: ");
  Serial.print(numberOfCharacters);
}

void loop() {
  // Nothing to do in the loop
}
```

a string class method that returns the number of char in that string

```
Text 1:This text,
Number of characters in Text 1: 11
Text 2:has more words!
Number of characters in Text 2: 15
Concatenated Text: This text, has more words!
Number of Characters in Concatenated Text: 26
```

Using Arduino String Functionality

Function	Description
strlen(str)	Returns the length of the string (excluding the null terminator).
strcpy(dest, src)	Copies the contents of the source string to the destination string.
strncpy(dest, src, n)	Copies at most n characters from the source string to the destination string.
strcat(dest, src)	Concatenates (appends) the source string to the destination string.
strncat(dest, src, n)	Concatenates (appends) at most n characters from the source string to the destination string.
strcmp(str1, str2)	Compares two strings lexicographically; returns 0 if equal, a positive value if str1 > str2, and a negative value if str1 < str2.
strncmp(str1, str2, n)	Compares at most n characters of two strings lexicographically.
strchr(str, c)	Returns a pointer to the first occurrence of character c in the string, or NULL if not found.
strstr(str, substr)	Returns a pointer to the first occurrence of substring substr in the string, or NULL if not found.
strtok(str, delim)	Breaks the string into a series of tokens using the specified delimiter characters.

Using C Character Strings

```
void setup() {
  Serial.begin(9600);
  char greeting[] = "Hello";
  char name[] = " Arduino";
  char combined[20];  // Assuming a size sufficient to hold both strings
  strcpy(combined, greeting);  // Copy the first string
  strcat(combined, name);      // Concatenate the second string
  Serial.println("Combined String: ");
  Serial.println(combined);
  int length = strlen(combined);
  Serial.print("Length of Combined String: ");
  Serial.println(length);
  char checkName[] = " Arduino";
  int compareResult = strcmp(name, checkName);
  if (compareResult == 0) {
    Serial.println("name and checkName are equal.");
  } else {
    Serial.println("name and checkName are not equal.");
  }
}
void loop() {}
```

Splitting Comma-Separated Text into Groups

- **Tokenization:** The process of dividing a string into tokens is called tokenization. This is often done by identifying and extracting substrings based on specified delimiters.

- **Token:** A token is a unit of information within a string. It can represent a word, a number, a symbol, or any other meaningful element.

- **Tokenizing Function:** In C and C++ programming, the ‘strtok’ function is commonly used for tokenization. It takes a string and a set of delimiters as arguments and returns pointers to the individual tokens within the string.

Splitting Comma-Separated Text into Groups

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

  // Example comma-separated text
  char inputText[] = "apple,orange,banana,grape";

  // Tokenize the input text using strtok
  char *token = strtok(inputText, ",");

  // Print each group
  Serial.println("Splitting into groups:");
  while (token != NULL) {
    Serial.println(token);
    token = strtok(NULL, ",");  // Get the next token
  }
}

void loop() {
  // Nothing to do in the loop
}
```

Converting a Number to a String

- The Arduino String class automatically converts numerical values when they are assigned to a String variable. You can combine (concatenate) numeric values at the end of a string using the concat function or the string + operator.

The following code results in `number` having a value of `13`:

```
int number = 12;
number += 1;
```

With a `String`, as shown here:

```
String textNumber = "12";
textNumber += 1;
```

`textNumber` is the text string `"121"`.

itoa and ltoa

- itoa and ltoa take three parameters:
  - the value to convert
  - the buffer that will hold the output string and
  - the number base (10 for a decimal number, 16 for hex, and 2 for binary).
- Buffers are used as temporary storage to hold data before it is processed, transferred, or manipulated.

```
void setup() {
  Serial.begin(9600);
  // Example numbers
  int intValue = 123;
  long longValue = 456789;
  // Buffers to store the converted strings
  char intBuffer[10];
  char longBuffer[20];
  // Convert integers to strings
  itoa(intValue, intBuffer, 10);
  ltoa(longValue, longBuffer, 10);
  // Print the converted strings
  Serial.print("Integer as String: ");
  Serial.println(intBuffer);

  Serial.print("Long Integer as String: ");
  Serial.println(longBuffer);
}
void loop() {
}
```

Converting a String to a Number

- You need to convert a string to a number. Perhaps you have received a value as a string over a communication link and you need to use this as an integer or floating-point value.
- Another approach to converting text strings representing numbers is to use the C language conversion function called atoi (for int variables) or atol (for long variables).
- Functions like ‘atoi’, ‘atol’, ‘atof’, ‘strtoXXX’ are used to convert strings to numbers.
- When using strtoXXX functions, know about the concept of base (e.g., 10 for decimal, 16 for hexadecimal).

Converting a String to a Number

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

  // Example string containing a number
  char numberString[] = "123";

  // Convert string to integer using atoi
  int convertedNumber = atoi(numberString);

  // Print the converted number
  Serial.print("Converted Number: ");
  Serial.println(convertedNumber);
}

void loop() {
  // Nothing to do in the loop
}
```

Error Handling During Conversion

If you need to handle errors during conversion, you might consider using ‘’strtol’’ for long integers or ‘’strtof’’ for floating-point numbers.

Pointers: A pointer is a variable that stores the memory address of another variable. It "points" to the location in memory where a value is stored.

- Declaration: int *ptr; \\declares a pointer to an integer.
- Initialization: int x = 10;
  int *ptr = &x; \\initializes the pointer ptr with the address of variable x.

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

  // Example string containing a number
  char numberString[] = "123abc";

  // Convert string to long integer using strtol with error checking
  char *endPtr;
  long convertedNumber = strtol(numberString, &endPtr, 10);

  if (*endPtr != '\0') {
    Serial.println("Error: Invalid characters in the string.");
  } else {
    Serial.print("Converted Number: ");
    Serial.println(convertedNumber);
  }
}
void loop() {
  // Nothing to do in the loop
}
```

Structuring Your Code into Functional Blocks

Function: A function is a program within a program, a defined function can be called from anywhere in the sketch and it contains its own variables and commands.

Example:

```
int ledPin=13;
int delayPeriod=250;
void setup()
{
pinMode(ledPin,OUTPUT);
}
void loop()
{
flash(20, delayPeriod);
delay(3000);
}
```

```
void flash(int numFlashes, int d)
{
for (int i=0; i<numFlashes; i++)
{
digitalWrite(ledPin, HIGH);
delay(d)
digitalWrite(ledPin, LOW);
delay(d);
}
}
```

Example

- Write down a sketch with a function that takes a parameter and returns a value. The parameter determines the length of the LED on and off times (in milliseconds). The function continues to flash the LED until a button is pressed, and the number of times the LED flashed is returned from the function.

```
const int ledPin = 13;     // Pin connected to the LED
const int buttonPin = 2;   // Pin connected to the button

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
  Serial.begin(9600);
}
```

Example

```
// Function to flash the LED based on the specified on/off times
int flashLED(int onTime, int offTime) {
  int flashCount = 0;

  while (digitalRead(buttonPin) == HIGH) {
    digitalWrite(ledPin, HIGH);  // Turn on the LED
    delay(onTime);
    digitalWrite(ledPin, LOW);   // Turn off the LED
    delay(offTime);

    flashCount++;
  }

  return flashCount;
}
```

```
void loop() {
  int onTime = 500;   // On time in milliseconds
  int offTime = 500;  // Off time in milliseconds

  Serial.println("Press the button to stop flashing.");

  int totalFlashes = flashLED(onTime, offTime);

  Serial.print("LED flashed ");
  Serial.print(totalFlashes);
  Serial.println(" times.");
delay(1000);  // Add a delay for visibility in the Serial Monitor
}
```

Taking Actions Based on Conditions

- You want to execute a block of code only if a particular condition is true. For example, you may want to light an LED if a switch is pressed or if an analog value is greater than some threshold.

```
const int switchPin = 2;    // Pin connected to the switch
const int analogPin = A0;   // Analog pin connected to a sensor
const int ledPin = 13;      // Pin connected to the LED
const int threshold = 500;  // Analog threshold value

void setup() {
  pinMode(switchPin, INPUT);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void
```

Taking Actions Based on Conditions

```
loop() {
  // Read the state of the switch
  int switchState = digitalRead(switchPin);

  // Read the analog sensor value
  int analogValue = analogRead(analogPin);

  // Check if the switch is pressed or analog value is greater than threshold
  if (switchState == LOW || analogValue > threshold) {
    // Turn on the LED
    digitalWrite(ledPin, HIGH);
    Serial.println("Switch is pressed or analog value is greater than threshold. LED is ON.");
  } else {
    // Turn off the LED
    digitalWrite(ledPin, LOW);
    Serial.println("Switch is not pressed and analog value is below threshold. LED is OFF.");
  }

  delay(100);  // Add a small delay for stability
}
```

In ON state, switches have LOW voltage

Repeating a Sequence of Statements

- You want to repeat a block of statements while an expression is true.

```
const int sensorPin = A0; // analog input 0

void setup()
{
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT); // enable LED pin as output
}

void loop()
{
  while(analogRead(sensorPin) > 100)
  {
    blink();    // call a function to turn an LED on and off
    Serial.print(".");
  }
  Serial.println(analogRead(sensorPin)); // this is not executed until after
}

void blink()
{
    digitalWrite(LED_BUILTIN, HIGH);
    delay(100);
    digitalWrite(LED_BUILTIN, LOW);
    delay(100);
}
```

Repeating Statements with a Counter

- You want to repeat one or more statements a certain number of times. The for loop is similar to the while loop, but you have more control over the starting and ending conditions.

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

void loop(){
  Serial.println("for(int i=0; i < 4; i++)");
  for(int i=0; i < 4; i++)
      {
        Serial.println(i);
      }
      delay(1000);
   }
```

Breaking Out of Loops

```
const int switchPin = 2; // digital input 2

void setup()
{
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT); // enable LED pin as output
  pinMode(switchPin, INPUT_PULLUP); // enable button pin as input
}

void loop()
{
  while(true) // endless loop
  {
    if(digitalRead(switchPin) == LOW)
    {
      break; // exit the loop if the switch is pressed
    }
    blink(); // call a function to turn an LED on and off
  }
}

void blink()
{
  digitalWrite(LED_BUILTIN, HIGH);
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);
  delay(100);
}
```

Taking a Variety of Actions Based on a Single Variable

```
void setup()
{
  Serial.begin(9600); // Initialize serial port to send and
                      // receive at 9600 baud
  pinMode(LED_BUILTIN, OUTPUT);
}
```

```
void blink()
{
  digitalWrite(LED_BUILTIN, HIGH);
  delay(500);
  digitalWrite(LED_BUILTIN, LOW);
  delay(500);
}
```

```
void loop()
{
  if ( Serial.available()) // Check to see if at least one
                           // character is available
  {
    char ch = Serial.read();
    switch(ch)
    {
    case '1':
      blink();
      break;
    case '2':
      blink();
      blink();
      break;
    case '+':
      digitalWrite(LED_BUILTIN, HIGH);
      break;
    case '-':
      digitalWrite(LED_BUILTIN, LOW);
      break;
    case '\n': // newline, safe to ignore
      break;
    case '\r': // carriage return, safe to ignore
      break;
    default:
      Serial.print(ch);
      Serial.println(" was received but not expected");
      break;
    }
  }
}
```

Comparing Character and Numeric Values

- You want to determine the relationship between values.

Table 2-5. Relational and equality operators

Operator	Test for	Example
==	Equal to	`2 == 3 // evaluates to false`
!=	Not equal to	`2 != 3 // evaluates to true`
>	Greater than	`2 > 3 // evaluates to false`
<	Less than	`2 < 3 // evaluates to true`
>=	Greater than or equal to	`2 >= 3 // evaluates to false`
<=	Less than or equal to	`2 <= 3 // evaluates to true`

```
int i = 1;  // some values to start with
int j = 2;

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

void loop(){
  Serial.print("i = ");
  Serial.print(i);
  Serial.print(" and j = ");
  Serial.println(j);

  if(i < j)
    Serial.println(" i is less than j");
  if(i <= j)
    Serial.println(" i is less than or equal to j");
  if(i != j)
    Serial.println(" i is not equal to j");
  if(i == j)
    Serial.println(" i is equal to j");
  if(i >= j)
    Serial.println(" i is greater than or equal to j");
  if(i > j)
    Serial.println(" i is greater than j");

  Serial.println();
  i = i + 1;
  if(i > j + 1)
  {
    delay(10000);  // long delay after i is no longer close to j
  }
  else
  {
    delay(1000);   // short delay
  }
}
```

Performing Logical Comparisons

Table 2-6. Logical operators

Symbol	Function	Comments
&&	Logical And	Evaluates as `true` if the conditions on both sides of the && operator are true
||	Logical Or	Evaluates as `true` if the condition on at least one side of the || operator is true
!	Not	Evaluates as `true` if the expression is false, and `false` if the expression is true

```
int a = 5;
int b = 10;
if (a > 0 && b > 0) {
  // This block will be executed if both a and b are greater than 0
  Serial.println("Both a and b are greater than 0");
}
```

```
int x = 7;
int y = 3;

if (x > 5 || y > 5) {
  // This block will be executed if either x or y is greater than 5
  Serial.println("Either x or y is greater than 5");
}
```

```
int value = 42;

if (!(value == 0)) {
  // This block will be executed if value is not equal to 0
  Serial.println("Value is not equal to 0");
}
```

Performing Bitwise Operations

Table 2-7. Bit operators

Symbol	Function	Result	Example
&	Bitwise And	Sets bits in each place to 1 if both bits are 1; otherwise, bits are set to 0.	`3 & 1` equals `1` (`0b11 & 0b01` equals `0b01`)
|	Bitwise Or	Sets bits in each place to 1 if either bit is 1.	`3 | 1` equals `3` (`0b11 | 0b01` equals `0b11`)
^	Bitwise Exclusive Or	Sets bits in each place to 1 only if one of the two bits is 1.	`3 ^ 1` equals `2` (`0b11 ^ 0b01` equals `0b10`)
~	Bitwise Negation	Inverts the value of each bit. The result depends on the number of bits in the data type.	`~1` equals `254` (`~00000001` equals `11111110`)

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

void loop(){
  Serial.print("3 & 1 equals "); // bitwise And 3 and 1
  Serial.print(3 & 1);          // print the result
  Serial.print(" decimal, or in binary: ");
  Serial.println(3 & 1 , BIN);   // print the binary representation of the result

  Serial.print("3 | 1 equals "); // bitwise Or 3 and 1
  Serial.print(3 | 1 );
  Serial.print(" decimal, or in binary: ");
  Serial.println(3 | 1 , BIN);   // print the binary representation of the result

  Serial.print("3 ^ 1 equals "); // bitwise exclusive or 3 and 1
  Serial.print(3 ^ 1);  Serial.print(" decimal, or in binary: ");
  Serial.println(3 ^ 1 , BIN);   // print the binary representation of the result


  byte byteVal = 1;
  int intVal = 1;

  byteVal = ~byteVal;  // do the bitwise negate
  intVal = ~intVal;

  Serial.print("~byteVal (1) equals "); // bitwise negate an 8-bit value
  Serial.println(byteVal, BIN);  // print the binary representation of the result
  Serial.print("~intVal (1) equals "); // bitwise negate a 16-bit value
  Serial.println(intVal, BIN);   // print the binary representation of the result

  delay(10000);
}
```

Combining Operations and Assignment

Table 2-11. Compound operators

Operator	Example	Equivalent expression
`+=`	`value += 5;`	`value = value + 5; // add 5 to value`
`-=`	`value -= 4;`	`value = value - 4; // subtract 4 from value`
`*=`	`value *= 3;`	`value = value * 3; // multiply value by 3`
`/=`	`value /= 2;`	`value = value / 2; // divide value by 2`
`>>=`	`value >>= 2;`	`value = value >> 2; // shift value right two places`
`<<=`	`value <<= 2;`	`value = value << 2; // shift value left two places`
`&=`	`mask &= 2;`	`mask = mask & 2; // binary-and mask with 2`
`|=`	`mask |= 2;`	`mask = mask | 2; // binary-or mask with 2`