**Rafiqa Maharani Putri Siregar – L200 154 010**

**Kelas A**

**Pemogramman Sistem Embedded**

**TUGAS 1**

**Language reference for arduino.**

The program of Arduino can be divided in three main parts:

**FIRST PART - STRUCTURE**

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Information** |
| Structures | setup() | It is called in the beginning of program and only run once (after each powerup or when the board of Arduino resetting).  It initializes and sets initial values (variables, pin modes, start using library, and else). |
| loop() | Loop the instruction that is written inside the loop() function breckets {}.  The instruction will loops continued, to support your program to change and respond |
|  |  |  |
| Control Structures | if | To tests whether a particular condition is reached or not during the instruction execution.  Use in conjuction with a comparison operator. |
| If/else | Give more control to the instructions than previous if statement by give support to do multiple tests just in a single group.  In shortway, if/else give more than one possible conditions, and each condition do different instruction. |
| for | To repeat code block inside the brecket, continuously that as many as you want.  Uses counter increament or decreament to terminate the loop.  Not only using counter increament. Sometimes it uses arrays to operate the data/pins |
| Switch…case | Compares the value of variable to values specified in each case statements, when it found whose value matches that of variable, the code in that case statement is run.  It’s like the if statement, but in this term, each case statements is placed in one breckets. |
| While loops | Loop continuously and infinitely until the condition or expression inside parenthesis() becomes false.  You must put something in your code to changed variable as control to stop the while loop, the loop will never end. It can be incremented variable or external condition like sensor. |
| Do-while | Works like while loop, but it tested the condition whether the loop will be happened or not, after the block code in Do segment has been done,  It will make at least one loop. |
| Break | To exit from the do, for, while, or switch statement. It bypassing the normal loop condition. |
| Continue | It will skips the rest of the current iteration of the loop (do, for, while).  It will check do conditional expression of the loop and continue the next iteration. |
| return | To terminate the function and return the value of the function to the calling function if desired. |
| goto | To go to labeled point in the program.  In wise way, the use of goto can simplify certain programs. It better use goto when it really-really needed, like to exit from deeply nested for loops.  Because the use of goto is not really discouraged. |
|  |  |  |
| Further Syntax | ; (semicolon) | Put in the end of each statement |
| {} (curly braces) | Very important in C language. Instructions is written inside the curly braces |
| // for single line comment | Compiler will ignores the lines that is started by “//” |
| /\*\*/ for multiline comments | It is ok to make single line comment in multiline comment. But it can not do vice versa. |
| #define | Give support for programmer to give name to a constant value (constanta) before the program is run.  Each constant name will be replaced with the constant value that was written before during the compilation.  There is no need add semicolon in the end of #define statements |
| #include | To make outside libraries are included in your sketch. To access a large group of standard C libraries and Arduuino libraries  Need not to use semicolon in the end of statement. |
|  |  |
| Arithmetic Operators | = (assignment operator) | Every value in the right of assignment sign will be stored in the variable to the left of the equal sign  Example:  A = 3  So in microcontroller, 3 will be stored in A.  Everytime A is written, it’s mean you write 3. |
| +, -, \*, / | To calculate addition, substraction, multiplication, and division. |
| % (modulo) | Calculates the reminder when one integer is divided by other integer. |
|  |  |
| Comparisons  Operator  (usually is used in conditional term) | == | Equal to |
| != | Not equal to |
| < | Less than |
| > | Bigger than |
| <= | Less than or equal to |
| >= | Bigger than or equal to |
|  |  |  |
| Boolean Operators | && | And (both of condition should be reached) |
| || | Or (both or just one of the condition should be reached) |
| ! | Not (negation of the condition) |
|  |  |  |
| Bitwise Operators  (Performs calculation in bit level of variables) | & (bitwise AND) | Used between two other integer expressions. Operates on each bit position of the surrounding expressions idependently by implement the rule of AND Logic |
| | (bitwise OR) | Used between two other integer expressions. Operates on each bit position of the surrounding expressions idependently by implement the rule of OR Logic |
| ^ (bitwise XOR) | Exclusive OR, only give output 1 if the input is different |
| ~ (bitwise NOT) | Is applied as single operand to its right. Changes 0 to 1 and vice versa.  As another interesting fact, for any integer x, ~x is the same as -x-1 |
| << (bitshift left) | Shift the position of number that specified from the right to the LEFT.  In simple way to think it, multiplies the number that in LEFT to operator with power 2 of the number in the RIGHT to operator  Example  1<< 0 == 1  1<< 2 == 4  1<< 3 == 8  2 >> 2 == 8 |
| >> (bitshift right) | To shift the position of bit to the RIGHT.  If the data is integer, so the higher bit as the sign whether it is positive or negative.  If it negative, the higher one is bit one. Then the shifting will produced 1 from the left to the right.  If you want to produce zero, the just declare it or avoid using sign extension. And also you can use right-shift operator to divide number in the left operator by powers of 2 the number in right.  Example:  Int A = 1000;  Int b = A >> 3; // b will be equal to 1000/3 power 2, that is 125 |
|  |  |  |
| Compound Operators | [++](https://www.arduino.cc/en/Reference/Increment) (increment) | x++; // increment x by one and returns the old value of x  ++x; // increment x by one and returns the new value of x |
| [--](https://www.arduino.cc/en/Reference/Increment) (decrement) | x-- ; // decrement x by one and returns the old value of x  --x ; // decrement x by one and returns the new value of x |
| [+=](https://www.arduino.cc/en/Reference/IncrementCompound) (compound addition) | x += y; // equivalent to the expression x = x + y; |
| [-=](https://www.arduino.cc/en/Reference/IncrementCompound) (compound subtraction) | x -= y; // equivalent to the expression x = x - y; |
| [\*=](https://www.arduino.cc/en/Reference/IncrementCompound) (compound multiplication) | x \*= y; // equivalent to the expression x = x \* y; |
| [/=](https://www.arduino.cc/en/Reference/IncrementCompound) (compound division) | x /= y; // equivalent to the expression x = x / y; |
| [%=](https://www.arduino.cc/en/Reference/IncrementCompound) (compound modulo) | x %= y; // equivalent to the expression x = x % y; |
| [&=](https://www.arduino.cc/en/Reference/BitwiseCompoundAnd) (compound bitwise and) | Is often used with a variable and a constant to force particular bits in a variable to the LOW state (zero).  Referred to “resetting” bits. |
| [|=](https://www.arduino.cc/en/Reference/BitwiseCompoundOr) (compound bitwise or) | Is often used with a variable and a constant to “set” to 1 particular bits in a variable. |

**SECOND PART - VARIABLES**

|  |  |  |
| --- | --- | --- |
| Constants | [HIGH](https://www.arduino.cc/en/Reference/Constants) | [LOW](https://www.arduino.cc/en/Reference/Constants) | Both are possible values of the pin.  HIGH, when pin is configured as a INPUT will give report that:   * A voltage bigger than 3.0V (in 5.0V board) * A voltage bigger than 2.0V (in 3.3V board)   HIGH, when pin is configured as OUTPUT:   * The pin is at 5V (5V board) * The pin is at 3.3V(3.3V board) |
| LOW, when pin as INPUT:   * voltage less than 1.5V is present at the pin (5V boards); * a voltage less than 1.0V (Approx) is present at the pin (3.3V boards);   LOW, when pin as OUTPUT:  the pin is at 0 volts (both 5V and 3.3V boards) |
| [INPUT](https://www.arduino.cc/en/Reference/Constants) | pins configured as **INPUT** with pinMode() are said to be in a high-impedance state.  If you have your pin configured as an INPUT, and are reading a switch, when the switch is in the open state the input pin will be "floating", resulting in unpredictable results. In order to assure a proper reading when the switch is open, a pull-up or pull-down resistor must be used |
| [INPUT\_PULLUP](https://www.arduino.cc/en/Reference/Constants) | resistors that connect to power internally.  If you prefer to use these instead of external pull-up resistors, you can use the **INPUT\_PULLUP** argument in pinMode().  Pins configured as inputs with either INPUT or INPUT\_PULLUP can be damaged or destroyed if they are connected to voltages below ground (negative voltages) or above the positive power rail (5V or 3V). |
| [OUTPUT](https://www.arduino.cc/en/Reference/Constants) | Pins configured as **OUTPUT** with pinMode() are said to be in a low-impedance state.  So, they can support a substantial amount of current to other circuits. Atmega pins can source (provide current) or sink (absorb current) up to 40 mA (milliamps) of current to other devices/circuits. This makes them useful for powering LEDs because LEDs typically use less than 40 mA. Loads greater than 40 mA (e.g. motors) will require a transistor or other interface circuitry.  Pins configured as outputs can be damaged or destroyed if they are connected to either the ground or positive power rails. |
| [LED\_BUILTIN](https://www.arduino.cc/en/Reference/Constants) | The number of the pin to which the on-board LED is connected. Usually in pin 13. |
| [true](https://www.arduino.cc/en/Reference/Constants) | [false](https://www.arduino.cc/en/Reference/Constants) | Boolean Constants  true is often defined as 1 but true also in term of any integer which is non-zero (-1, -2, -234, 4, 200, etc)  false is defined as 0 (zero) |
| [integer constants](https://www.arduino.cc/en/Reference/IntegerConstants) | Are numbers used directly in sketch, like 123 (base 10-decimal) or other bases that leading by formatter.  Binary is lead by B  Octal is lead by 0  Hexa is lead by 0x  By default it is treated as integer, but it can change with U and L modifiers.  U to force the constant into unsigned data format: 33u  L make the data into long data format: 100000L  UL to force into an unsigned long constant: 35436333ul |
| [floating point constants](https://www.arduino.cc/en/Reference/Fpconstants) | Swap the constants during compile time for the value to which the expression evaluates.  “e” or “E” both accepted as valid exponent indicators  10.0 evaluates to 10  2.3e5 evaluates to 2.34 \* 10^5 or 234000  67e-12 evaluates to 67.0 \* 10^-12 0r 0.000000000067 |
|  |  |  |
| Data Types | [void](https://www.arduino.cc/en/Reference/Void) | Only use in function declarations to indicates that the function is expected to not return information to the particular function. |
| [boolean](https://www.arduino.cc/en/Reference/BooleanVariables) | Separate data to two values, true or false, occupied one byte of memory. |
| [char](https://www.arduino.cc/en/Reference/Char) | Character literals written in single quotes: ‘A’  Multiple characters – string- is written in double quotes: “ABC”  Stores in 1 byte of memory.  Saved as ascii code  The char datatype is a signed type, so it encodes numbers from -128 to 127    char myChar = ‘A’;  char myChar = 65 ; // both are equivalent |
| [unsigned char](https://www.arduino.cc/en/Reference/UnsignedChar) | Encodes number from 0 to 255 |
| [byte](https://www.arduino.cc/en/Reference/Byte) | A byte stores in 8-bit unsigned number from 0 to 255  Example  Byte B = B10010; // equal to 18 decimal |
| [int](https://www.arduino.cc/en/Reference/Int) | Is stored in 16-bit part of memory.  It is the primary data-type for number storage. |
| [unsigned int](https://www.arduino.cc/en/Reference/UnsignedInt) | It is also stored in 16-bit, but only store positive value. |
| [word](https://www.arduino.cc/en/Reference/Word) | On the Uno and other ATMEGA based boards, a word stores a 16-bit unsigned number. On the Due and Zero instead it stores a 32-bit unsigned number. |
| [long](https://www.arduino.cc/en/Reference/Long) | It give support to store extended size variables for number = 32 bits. |
| [unsigned long](https://www.arduino.cc/en/Reference/UnsignedLong) | Like long, but just store positive values |
| [short](https://www.arduino.cc/en/Reference/Short) | Store in 16-bit value (for number) |
| [float](https://www.arduino.cc/en/Reference/Float) | A number that has decimal point and the value is very little. Is often used to approximate and continuous values. Stores in 32-bit part of memory.  Only have 6-7 decimal digits of precision  The calculation also slower than int calculation. You used to add decimal point to make float, if not the number will be treated as integer. |
| [double](https://www.arduino.cc/en/Reference/Double) | It precision float number , saved in 8 bytes. |
| [string](https://www.arduino.cc/en/Reference/String) - char array | Make string out of an array of type char and null-terminate it.  Note that it's possible to have a string without a final null character  Character arrays are referred to as strings with a small s.  constant strings, specified in "double quotes" are treated as char arrays, not instances of the String class. |
| [String](https://www.arduino.cc/en/Reference/StringObject) - object | String as object give you more things to do. You can concatenate Strings, append to them, search for and replace substrings, and more. It takes more memory than a simple character array, but it is also more useful.  instances of the String class are referred to as Strings with a capital S |
| [array](https://www.arduino.cc/en/Reference/Array) | An array is a collection of variables that are accessed with an index number.  Arrays are **zero indexed**, that is, referring to the array initialization above, the first element of the array is at index 0 |
|  |  |  |
| Changes | [char()](https://www.arduino.cc/en/Reference/CharCast) | Converts a value to the [char](https://www.arduino.cc/en/Reference/Char) data type. |
| [byte()](https://www.arduino.cc/en/Reference/ByteCast) | Converts a value to the [byte](https://www.arduino.cc/en/Reference/Byte) data type. |
| [int()](https://www.arduino.cc/en/Reference/IntCast) | Converts a value to the [int](https://www.arduino.cc/en/Reference/Int)eger data type. |
| [word()](https://www.arduino.cc/en/Reference/WordCast) | Convert a value to the [word](https://www.arduino.cc/en/Reference/Word) data type or create a word from two bytes. |
| [long()](https://www.arduino.cc/en/Reference/LongCast) | Converts a value to the [long](https://www.arduino.cc/en/Reference/Long) data type. |
| [float()](https://www.arduino.cc/en/Reference/FloatCast) | Converts a value to the [float](https://www.arduino.cc/en/Reference/Float) data type. |
|  |  |  |
| Variable Scope and Qualifiers | [variable scope](https://www.arduino.cc/en/Reference/Scope) | A property, to declares local variable(s), so it will only can be used by function in which they are declared.  It is also handy to use local variable(s) inside a for loop, these variable can only be accessed inside the for-loop. |
| [static](https://www.arduino.cc/en/Reference/Static) | If variable is declared using the “static”, the variable will always store the last value that was given to it.  And can only be seen by one function where it was created. |
| [volatile](https://www.arduino.cc/en/Reference/Volatile) | When the value of variable(s) can be changed by something beynd the control of the code section when it appears, e.g. interrupts, those variable(s) should be declared as volatile |
| [const](https://www.arduino.cc/en/Reference/Const) | Stands for “constant”.  This variable still can be used like other variable, but the value can not be changed. Like variable  Conts float pi = 3.14  The value is always 3.14, never changes |
|  |  |  |
| Utilities | Sizeof() | Is used to determined the size – in bytes- of variable(s) or data type or sizeof bytes in an array |
| PROGMEM | Variable modifier that is used to put the particular information into flash memory instead of into SRAM, where it normally go.  Please note that variables must be either globally defined, OR defined with the **static** keyword, in order to work with PROGMEM.  Ps: it should be used only with the datatypes defined in pgmspace.h library. |

**THIRD PART - FUNCTIONS**

|  |  |  |
| --- | --- | --- |
| Digital I/O | pinMode() | Configurates pin to behave either as input or output.  pinMode(pin, mode)  #pin= number of pin  #mode= input, input\_pullup, or output.  This is syntax that was made for Arduino |
| digitalWrite() | To set the value for digital pin = HIGH or LOW.  it’s important to set the pinMode() as INPUT or OUTPUT, if not, the digitalWrite() will have enabled the internal pull-up resistor to the particular pin. which acts like a large current-limiting resistor. |
| digitalRead() | Reads the value from a specified digital pin, either [HIGH](https://www.arduino.cc/en/Reference/Constants) or [LOW](https://www.arduino.cc/en/Reference/Constants).  If the pin isn't connected to anything, digitalRead() can return either HIGH or LOW (and this can change randomly). |
|  |  |  |
| Analog I/O | [analogReference](https://www.arduino.cc/en/Reference/AnalogReference)() | Is used to declares the voltage for analog input.  There are several options:   * DEFAULT: the default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards) * INTERNAL: an built-in reference, equal to 1.1 volts on the ATmega168 or ATmega328 and 2.56 volts on the ATmega8 (*not available on the Arduino Mega*) * INTERNAL1V1: a built-in 1.1V reference (*Arduino Mega only*) * INTERNAL2V56: a built-in 2.56V reference (*Arduino Mega only*) * EXTERNAL: the voltage applied to the AREF pin (0 to 5V only) is used as the reference.   The first reading from analogRead() will be not accurate at the first time when the analog reference is changed.  Remember to not uses anything less than 0V or more than 5V for external reference, it would be damaging the microcontroller on the board. |
| [analogRead](https://www.arduino.cc/en/Reference/AnalogRead)() | Reads value from the specified analog pin. It will map input from 0 and 1023 (integer) or between 0 to 5 Volts.  If the analog input pin not connected to anything, the value that will be returned by the analogRead() will fluctuate based on factors like value of other analog input, how close your hand to the board and else. |
| [analogWrite](https://www.arduino.cc/en/Reference/AnalogWrite)() - PWM | Write analog value (PWMM wave) to a pin. It can be used to power on LED in various brightnesses or drive a motor at various speeds.  Maximum PWM signal in UNO in pin 5 and 6 is 980Hz.  Need not to call pinMode() to set the pin as an output before calling analogWrite() |
|  |  |  |
| Due & Zero only | [analogReadResolution](https://www.arduino.cc/en/Reference/AnalogReadResolution)() | It is extension of the Analog API for Arduino Due and Zero.  It will sets size in bits of the value that will be returned by analogRead().  The defaults to 10 bits (value 0-1023) for backward compability with AVR based boards. |
| [analogWriteResolution](https://www.arduino.cc/en/Reference/AnalogWriteResolution)() | It sets the resolution of the analogWrite() function. The defaults to 8 bits (values 0-255) |
|  |  |  |
| Advanced I/O | [tone](https://www.arduino.cc/en/Reference/Tone)() | To play tones and can connected to piezo buzzer or other speaker to play tones.  Only one tone can be generated at a time. If a tone is already played in defferent pin, the call to tone() will have no effect to the current pin. |
| [noTone](https://www.arduino.cc/en/Reference/NoTone)() | Stops the square wave that was triggered by tone().  You need to call noTone() to play different pitches on multiple pins. |
| [shiftOut](https://www.arduino.cc/en/Reference/ShiftOut)() | Shifts out a byte of data one bit at a time. Starts from either the leftmost or rightmost significant bit.  Each bit is written in turn to a data pin, after which a clock pin is pulsed (taken high, then low) to indicate that the bit is available.  Note: if you're interfacing with a device that's clocked by rising edges, you'll need to make sure that the clock pin is low before the call to shiftOut(), e.g. with a call to digitalWrite(clockPin, LOW). |
| [shiftIn](https://www.arduino.cc/en/Reference/ShiftIn)() | Shifts in a byte of data one bit at a time. |
| [pulseIn](https://www.arduino.cc/en/Reference/PulseIn)() | To reads pulse on a pin, either HIGH or LOW, if value is HIGH pulseIn() waits the pin go HIGH and starts timing. Then waits for the pin to go LOW |
|  |  |  |
| Time | [millis](https://www.arduino.cc/en/Reference/Millis)() | Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days. |
| [micros](https://www.arduino.cc/en/Reference/Micros)() | Counts the number of microseconds since the Arduino starts running the current program.  The number will go back to zero after approximately 70 minutes. |
| [delay](https://www.arduino.cc/en/Reference/Delay)() | Pauses the program for amount of time (in miliseconds) specified as parameter.  (1 second = 1000 miliseconds) |
| [delayMicroseconds](https://www.arduino.cc/en/Reference/DelayMicroseconds)() | Pauses the program for the amount of time (in microseconds) specified as parameter.  (1 milisecond = 1000 microseconds  1 second = 1.000.000 microseconds) |
|  |  |  |
| Math | [min](https://www.arduino.cc/en/Reference/Min)() | Calculates the minimum of two numbers.  min (x, y) |
| [max](https://www.arduino.cc/en/Reference/Max)() | Calculates the maximum of two numbers.  max (x, y) |
| [abs](https://www.arduino.cc/en/Reference/Abs)() | Computes the absolute value of a number. |
| [constrain](https://www.arduino.cc/en/Reference/Constrain)() | Constrains a number to be within a range.  constrain(x, y) |
| [map](https://www.arduino.cc/en/Reference/Map)() | Re-maps a number from one range to another. That is, a **value** of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc.  The function also handles negative numbers well |
| [pow](https://www.arduino.cc/en/Reference/Pow)() | Computes the power of number. It also useful to raise a number to fractional power. |
| [sqrt](https://www.arduino.cc/en/Reference/Sqrt)() | Calculates the square root of a number |
|  |  |  |
| Trigonometry | [sin](https://www.arduino.cc/en/Reference/Sin)() | Calculates the sine of an angle (in radians). The result will be between -1 and 1 |
| [cos](https://www.arduino.cc/en/Reference/Cos)() | Calculates the cos of an angle (in radians). The result will be between -1 and 1 |
| [tan](https://www.arduino.cc/en/Reference/Tan)() | Calculates the tangent of an angle (in radians). The result will be between negative infinity and infinity. |
|  |  |  |
| Characters | [isAlphaNumeric](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is alphanumeric. |
| [isAlpha](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is is alpha. |
| [isAscii](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is ASCII. |
| [isWhitespace](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is a white space. |
| [isControl](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is a control character. |
| [isDigit](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is a digit |
| [isGraph](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is a printable character. |
| [isLowerCase](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is a lower case character. |
| [isPrintable](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is a printable character. |
| [isPunct](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is punctuation character. |
| [isSpace](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is a space character. |
| [isUpperCase](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is an upper case character. |
| [isHexadecimalDigit](https://www.arduino.cc/en/Reference/CharacterAnalysis)() | Analyse if a char is a valid hexadecimal digit.. |
|  |  |  |
| Random Numbers | [randomSeed](https://www.arduino.cc/en/Reference/RandomSeed)() | Makes pseudo-random number generator, that very long, random but always have same pattern. |
| [random](https://www.arduino.cc/en/Reference/Random)() | The random function generates pseudo-random numbers.  Syntax:  random(max) random(min, max) |
|  |  |  |
| Bits and Bytes | [lowByte](https://www.arduino.cc/en/Reference/LowByte)() | Extracts the low-order (rightmost) byte of a variable (e.g. a word). |
| [highByte](https://www.arduino.cc/en/Reference/HighByte)() | Extracts the high-order (leftmost) byte of a word (or the second lowest byte of a larger data type). |
| [bitRead](https://www.arduino.cc/en/Reference/BitRead)() | Reads a bit of a number. |
| [bitWrite](https://www.arduino.cc/en/Reference/BitWrite)() | Writes a bit of a numeric variable. |
| [bitSet](https://www.arduino.cc/en/Reference/BitSet)() | Sets (writes a 1 to) a bit of a numeric variable. |
| [bitClear](https://www.arduino.cc/en/Reference/BitClear)() | Clears (writes a 0 to) a bit of a numeric variable. |
| [bit](https://www.arduino.cc/en/Reference/Bit)() | Computes the value of the specified bit (bit 0 is 1, bit 1 is 2, bit 2 is 4, etc.). |
|  |  |  |
| External Interrupts | [attachInterrupt](https://www.arduino.cc/en/Reference/AttachInterrupt)() | Interrupts are useful for making things happen automatically in microcontroller programs, and can help solve timing problems. Good tasks for using an interrupt may include reading a rotary encoder, or monitoring user input.  Delay() will not work, also milis() will not increment.  Should use digitalPinToInterrupt(pin) to translate the actual digital pin to the specific interrupt number. For example, if you connect to pin 3, use digitalPinToInterrupt(3) as the first parameter to attachInterrupt. |
| [detachInterrupt](https://www.arduino.cc/en/Reference/DetachInterrupt)() | Turns off the given interrupt. |
|  |  |  |
| Interrupts | [interrupts](https://www.arduino.cc/en/Reference/Interrupts)() | Interrupts allow certain important tasks to happen in the background and are enabled by default.  And some function will not work while interrupts are disabled. |
| [noInterrupts](https://www.arduino.cc/en/Reference/NoInterrupts)() | Disables interrupts (you can re-enable them with interrupts()) |
|  |  |  |
| Communication | Serial | Serial is used for communication between the Arduino board and a computer or other devices.  Serial communication on pins TX/RX uses TTL logic levels (5V or 3.3V depending on the board). Don't connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage your Arduino board. |
| Stream | Stream defines the reading functions in Arduino. When using any core functionality that uses a read() or similar method, you can safely assume it calls on the Stream class  Stream is the base class for character and binary based streams. It is not called directly, but invoked whenever you use a function that relies on it. |
|  |  |  |
| USB (32u4 based boards and Due/Zero only) | Keyboard | The keyboard functions enable a Leonardo, Micro, or Due to send keystrokes to an attached computer.  When Keyboard library, it may be best to test your output first using Serial.print() |
| Mouse | The mouse functions enable a Leonardo, Micro, or Due to control cursor movement on a connected computer. When updating the cursor position, it is always relative to the cursor's previous location. |