Major Functions on Pin, Delay, and Interrupts

pinMode	Configures the specified pin to behave either as an input or an output.	
	pinMode(pin, mode)	**pin:** the number of the pin whose mode you wish to set **mode**: INPUT, OUTPUT, or INPUT_PULLUP.
digitalRead	Reads the value from a specified digital pin, either HIGH or LOW.	
	digitalRead(pin)	**pin:** the number of the digital pin you want to read Return the value HIGH or LOW.
digitalWrite	Write a HIGH or a LOW value to a digital pin.	
	digitalWrite(pin, value)	**pin:** the pin number **value:** HIGH or LOW
analogRead	Reads the value from the specified analog pin.	
	analogRead(pin)	**pin:** the number of the analog input pin to read from (0 to 5) Return the value (as an integer)
analogReference	Configures the reference voltage used for analog input	
	analogReference(type)	**type:** (DEFAULT, EXTERNAL).
analogWrite	Writes an analog value (PWM wave) to a pin.	
	analogWrite(pin, value)	**pin:** the pin to write to. Allowed data types: int. **value**: the duty cycle: between 0 (always off) and 255 (always on). Allowed data types: int
delay	Pauses the program for the amount of time (in milliseconds) specified as parameter.	
	delay(ms)	**ms:** the number of milliseconds to pause (unsigned long)
millis	Returns the number of milliseconds since the Arduino board began running the current program.	
	Unsigned long millis()	Return a unsigned long value
attachInterrupt	Set digital pin for external interrupt	
	attachInterrupt (digitalPinToInterrupt(pin), ISR, mode);	**pin:** the pin number (0, 1, 2, 3, 7) **ISR:** the ISR to call when the interrupt occurs; **mode**: defines when the interrupt should be triggered. Four constants are predefined as valid values: • **LOW** to trigger the interrupt whenever the pin is low. • **CHANGE** to trigger the interrupt whenever the pin changes value • **RISING** to trigger when the pin goes from low to high, • **FALLING** for when the pin goes from high to low.
Interrupts()	Re-enables interrupts after they've been disabled by noInterrupts()	
	interrupts()	
noInterrupts()	Disables interrupts	
	noInterrupts()	

Major Functions for Serial Monitor

Serial	Indicate if the specified Serial port is ready. Return true if the specified serial port is available, else return false	
Serial.available	Get the number of bytes (characters) available for reading from the serial port.	
	Serial.available()	Return the number of bytes
Serial.begin	Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200.	
	Serial.begin(speed)	**speed:** in bits per second (baud)
Serial.read	Reads incoming serial data.	
	Serial.read()	Return the first byte of incoming serial data available (or -1 if no data is available)
Serial.write	Writes binary data to the serial port. This data is sent as a byte or series of bytes;	
	Serial.write(val) Serial.write(str)	**val:** a value to send as a single byte **str:** a string to send as a series of bytes
Serial.println	Prints data to the serial port as human-readable ASCII text.	
	Serial.println(val)	**val:** the value to print - any data type
Serial.end	Disables serial communication	
	Serial.end()	

Functions (.begin, .read, .write, .println) are also defined similarly in the library SoftwareSerial.h

Major Functions for String

length()	Returns the length of the String, in characters.	
	str.length()	**str**: a variable of type String
substring()	Get a substring of a String, specified by a starting index upto the end of the String. The starting index is inclusive, i.e. the corresponding character is included in the substring).	
	str.substring(from)	**str**: a variable of type String **from**: the index to start the substring at Return the substring
toInt()	Converts a valid String to an integer. The input string should start with an integer number. If the string contains non-integer numbers, the function will stop performing the conversion.	
	str.toInt()	**str**: a variable of type String Return the integer value. If invalid, return 0
indexOf()	Locates a character or String within another String	
	str.indexOf(val)	**str**: a variable of type String **val**: the value to search for; it can be char or a String Return the index of val within the String, or -1 if not found.
equals()	Compares two strings for equality. The comparison is case-sensitive.	
	str.equals(str2)	**str**, **str2**: variables of type String Return true: if str equals str2 Return false: otherwise

Major Functions in Timer.h

update	Update the timer; always needed when timer is used.	
	t.update()	**t**: a variable of type Timer
every	Run the 'callback' procedure every 'period' milliseconds.	
	t.every(period, callback) OR t.every(period, callback, repeatCount)	**t**: a variable of type Timer **period**: in millisecond **callback**: procedure name to be called **repeatCount**: total number of times to repeat Return the ID of timer event (as an integer)
oscillate	Toggle the state of the digital output 'pin' every 'period' milliseconds.	
	t.oscillate(pin, period, startValue) OR t.oscillate(pin, period, startValue, repeatCount)	**t**: a variable of type Timer **pin**: digital output pin **period:** in milliseconds **startValue**: initial value of the pin **repeatCount**: total number of times to repeat Return the ID of timer event (as an integer)
pulse	Toggle the state of the digital output 'pin' just once after 'period' milliseconds.	
	t.pulse(pin, period, startValue)	**t**: a variable of type Timer **pin**: digital output pin **period**: in milliseconds **startValue**: initial value of the pin Return the ID of timer event (as an integer)
stop	Stop the timer event running	
	t.stop(id)	**t**: a variable of type Timer **id**: timer event id to be stopped
after	Run the 'callback' once after 'period' milliseconds	
	t.after(duration, callback)	**t**: a variable of type Timer **duration**: long int **callback**: procedure name to be called Return the ID of timer event (as an integer)