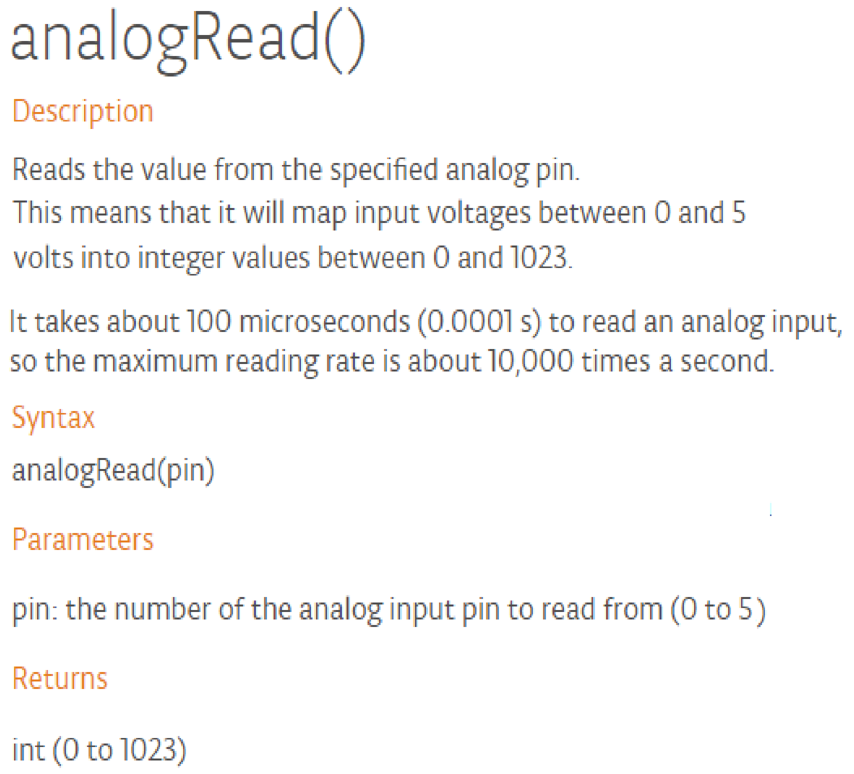
CSE423: Embedded System
Summer-2020
Understanding Arduino Code/Sketch (2)

Todays Lecture

- *Understanding Arduino Code (Sketch) step by step*
- *Implementation of code*

Understanding Codes: Analog I/O

- **1. AnalogRead( )**



analogRead()

Description

Reads the value from the specified analog pin.
This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023.

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

Syntax

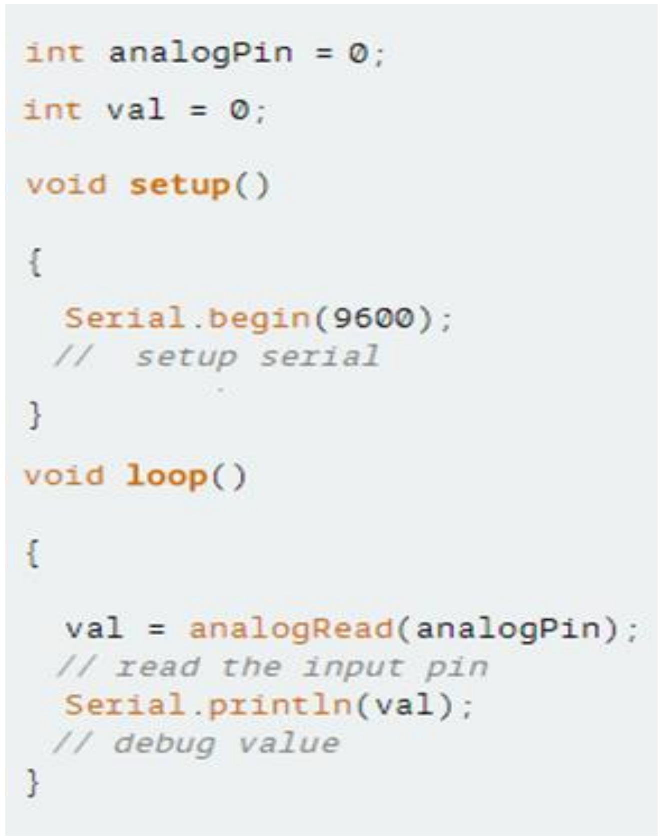
analogRead(pin)

Parameters

pin: the number of the analog input pin to read from (0 to 5)

Returns

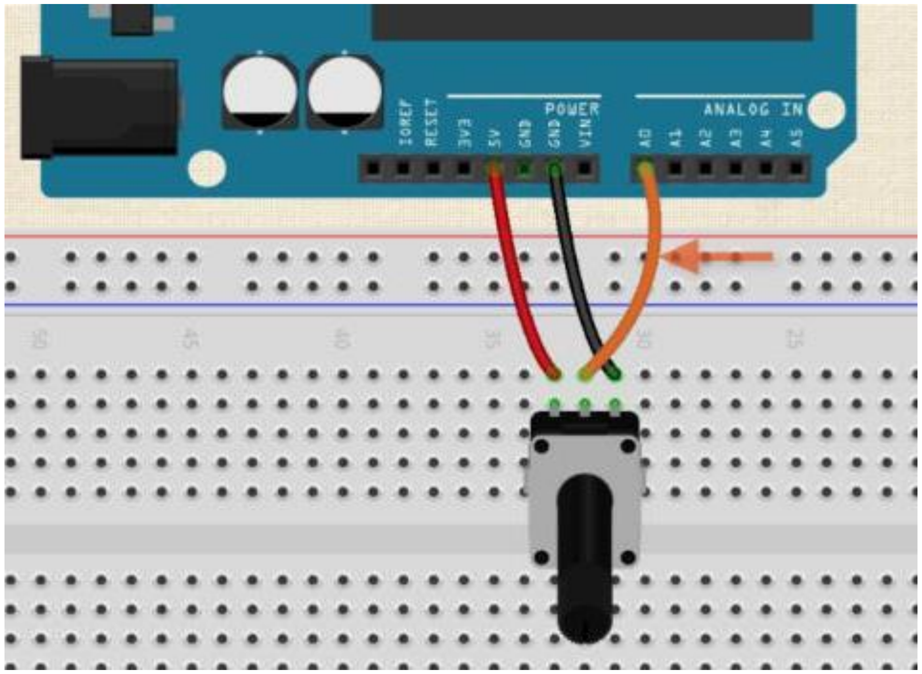
int (0 to 1023)

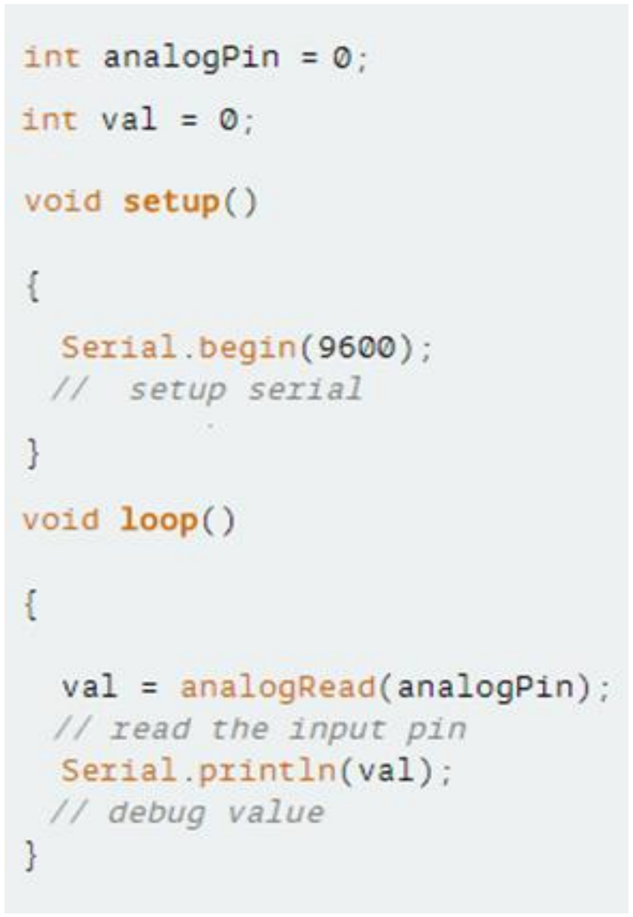


```
int analogPin = 0;
int val = 0;

void setup()

{
   Serial.begin(9600);
  //  setup serial

}
void loop()

{

   val = analogRead(analogPin);
  // read the input pin
   Serial.println(val);
  // debug value
}
```

Understanding Codes: Analog I/O

- 1. AnalogRead( )

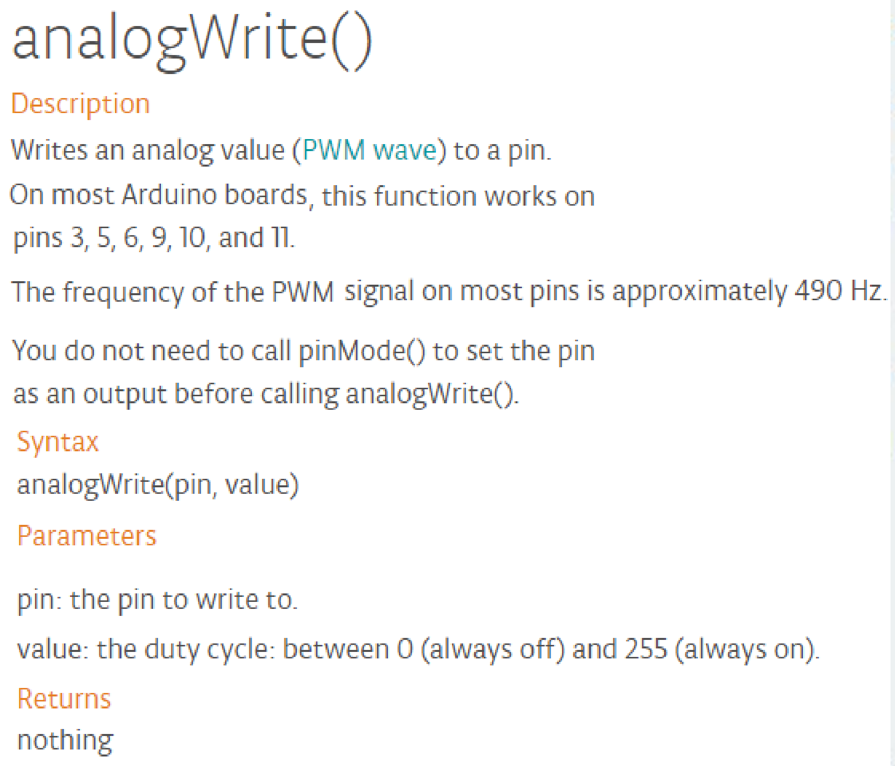




```
int analogPin = 0;
int val = 0;

void setup()

{
  Serial.begin(9600);
 //  setup serial

}
void loop()

{

  val = analogRead(analogPin);
 // read the input pin
  Serial.println(val);
 // debug value
}
```

Understanding Codes: Analog I/O

- 2. AnalogWrite( ) -PWM



analogWrite()

Description

Writes an analog value (PWM wave) to a pin.

On most Arduino boards, this function works on pins 3, 5, 6, 9, 10, and 11.

The frequency of the PWM signal on most pins is approximately 490 Hz.

You do not need to call pinMode() to set the pin as an output before calling analogWrite().

Syntax

analogWrite(pin, value)

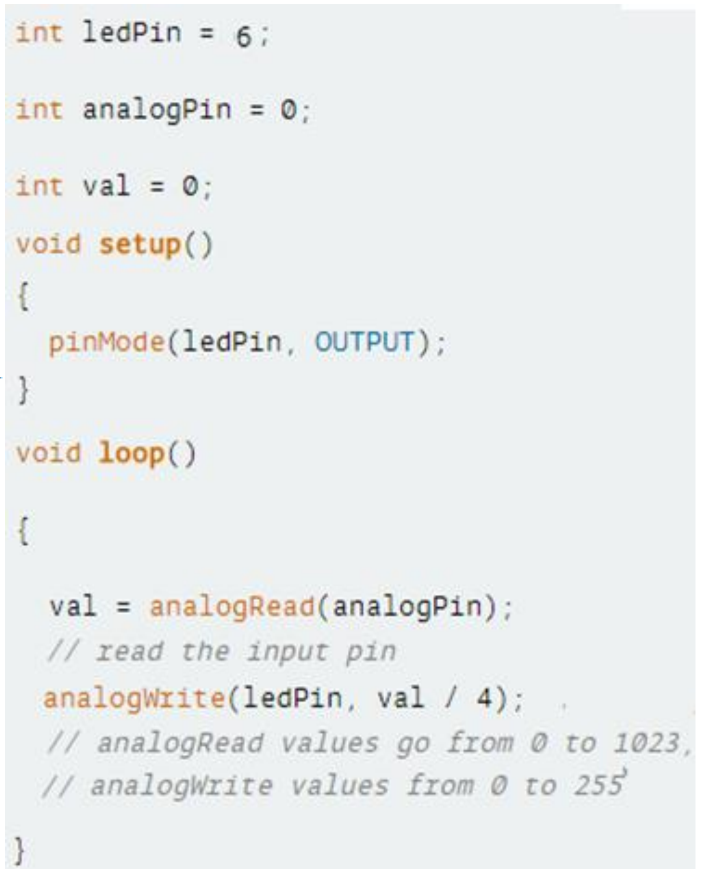
Parameters

pin: the pin to write to.

value: the duty cycle: between 0 (always off) and 255 (always on).

Returns

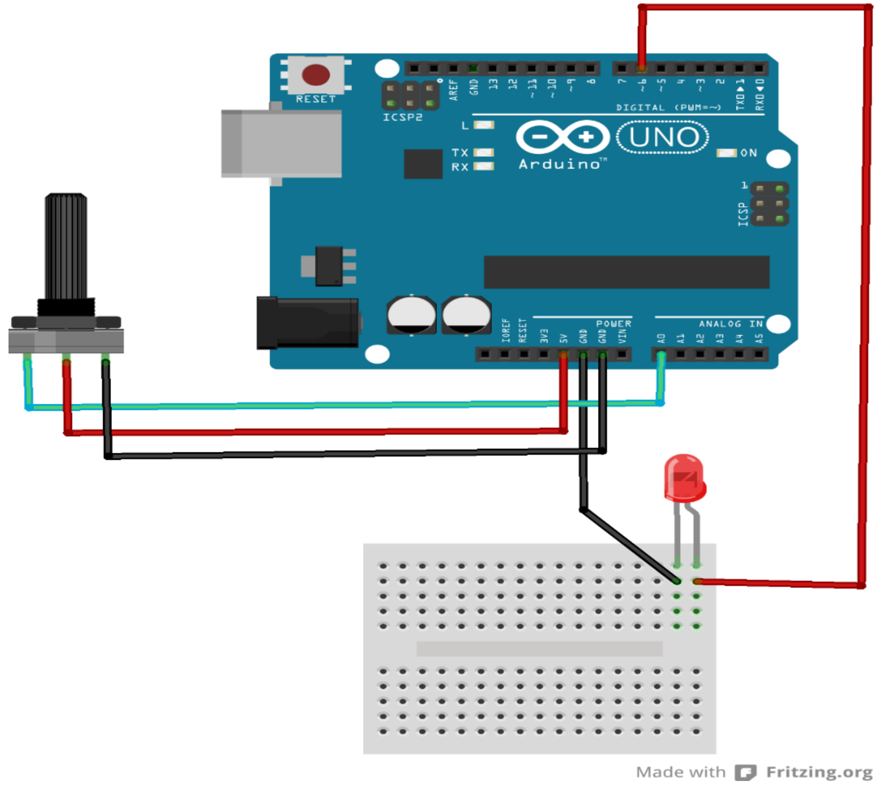
nothing

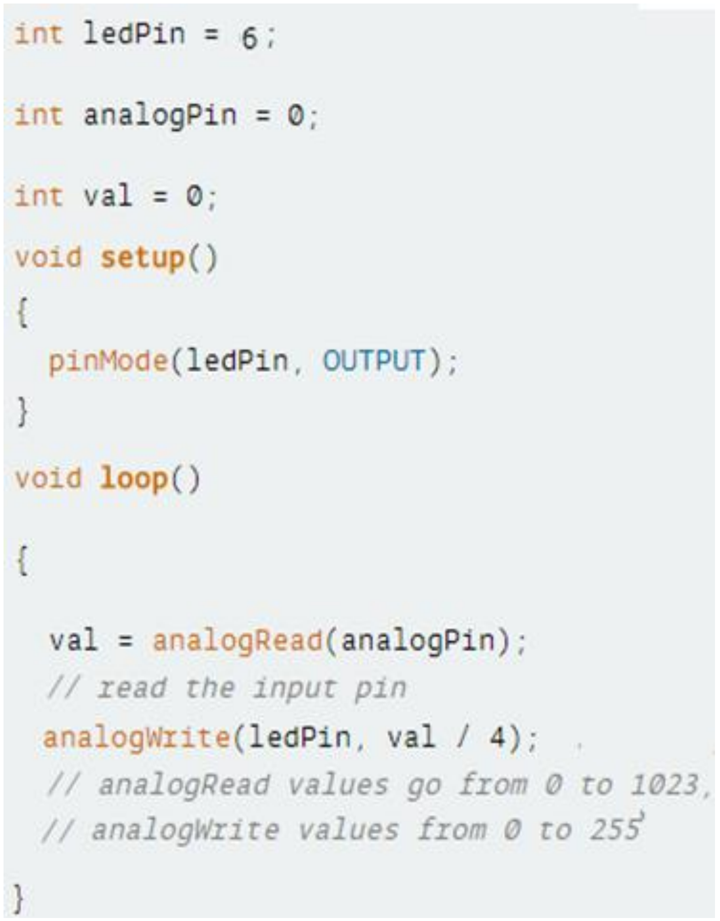


```
int ledPin = 6;

int analogPin = 0;

int val = 0;
void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()

{

  val = analogRead(analogPin);
  // read the input pin
  analogWrite(ledPin, val / 4);
  // analogRead values go from 0 to 1023,
  // analogWrite values from 0 to 255

}
```

Understanding Codes: Analog I/O

- 2. AnalogWrite( ) -PWM

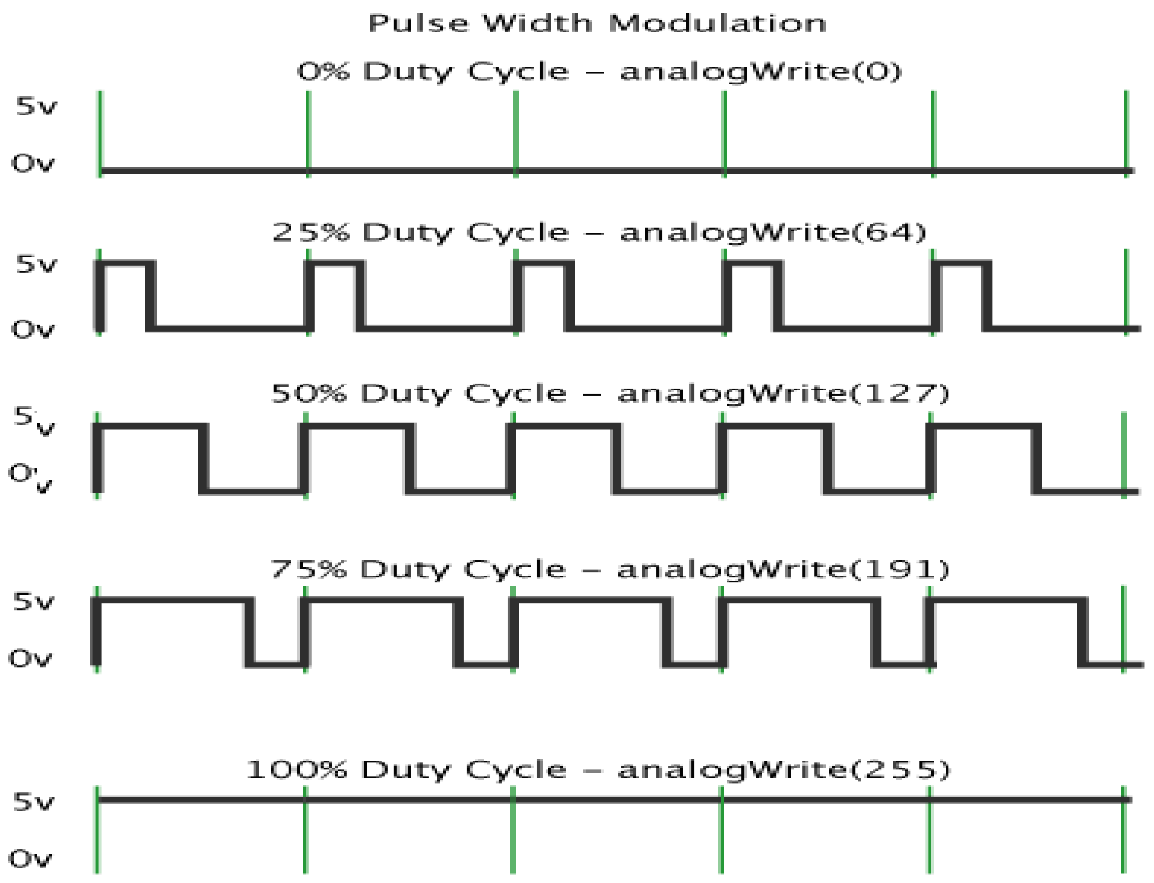


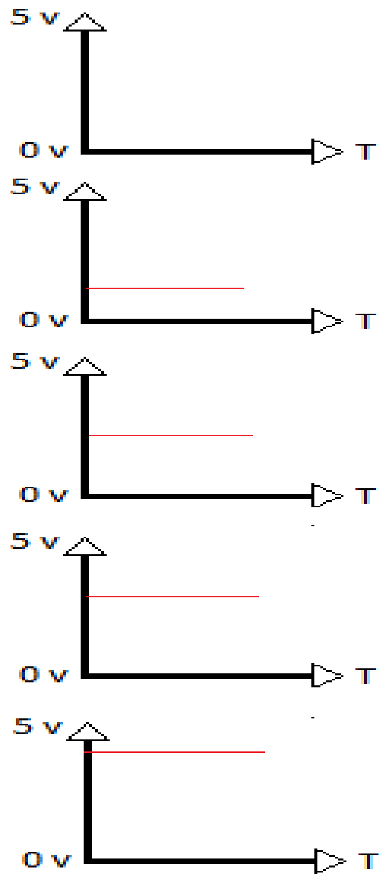


```
int ledPin = 6;

int analogPin = 0;

int val = 0;
void setup()
{
  pinMode(ledPin, OUTPUT);
}
void loop()

{

  val = analogRead(analogPin);
  // read the input pin
  analogWrite(ledPin, val / 4);
  // analogRead values go from 0 to 1023,
  // analogWrite values from 0 to 255
}
```

Understanding Codes: Analog I/O

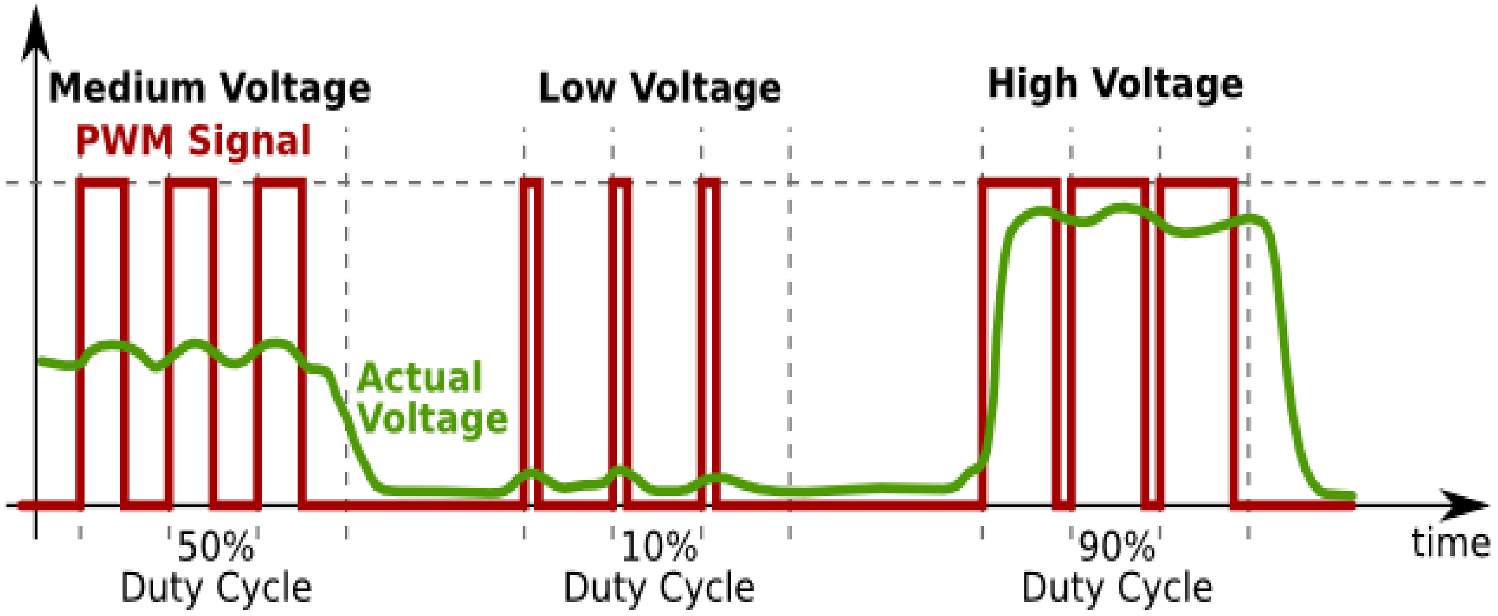
- 2. AnalogWrite( ) -PWM


Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
5v
0v
25% Duty Cycle – analogWrite(64)
5v
0v
50% Duty Cycle – analogWrite(127)
5v
0v
75% Duty Cycle – analogWrite(191)
5v
0v
100% Duty Cycle – analogWrite(255)
5v
0v

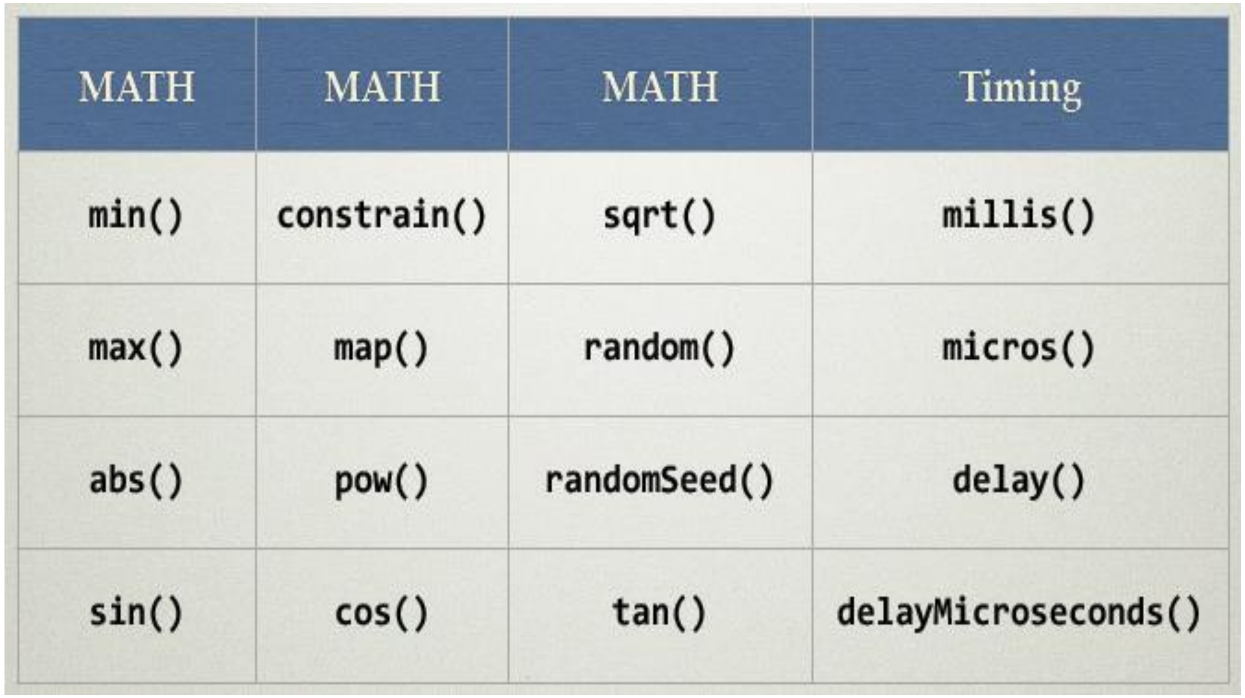


Understanding Codes: Analog I/O

- 2. AnalogWrite( ) -PWM



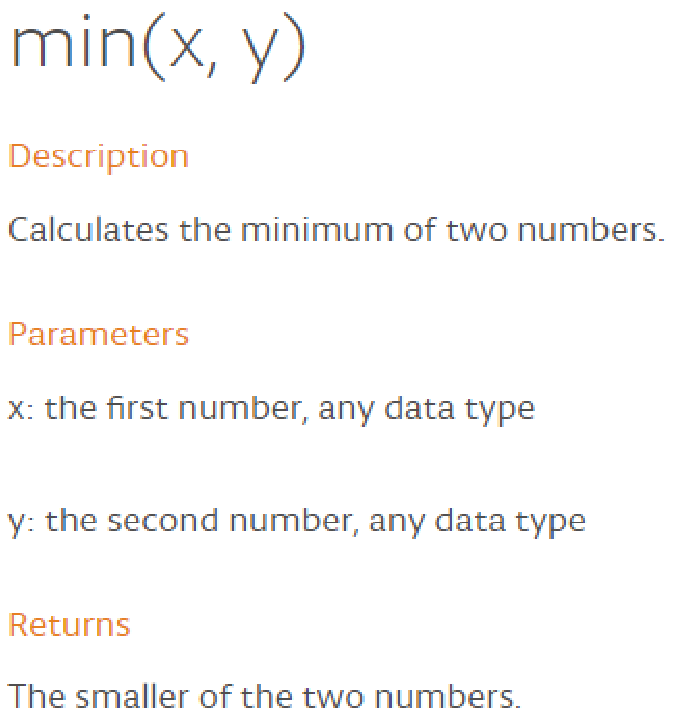
Understanding Codes: Functions



MATH	MATH	MATH	Timing
min()	constrain()	sqrt()	millis()
max()	map()	random()	micros()
abs()	pow()	randomSeed()	delay()
sin()	cos()	tan()	delayMicroseconds()

Understanding Codes: Functions

- **1. min( )**



min(x, y)

Description

Calculates the minimum of two numbers.

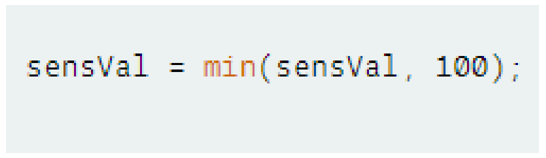
Parameters

x: the first number, any data type

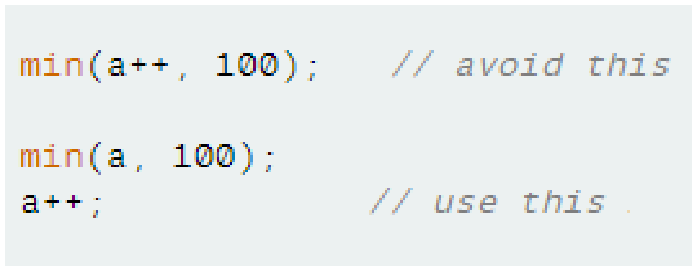
y: the second number, any data type

Returns

The smaller of the two numbers.



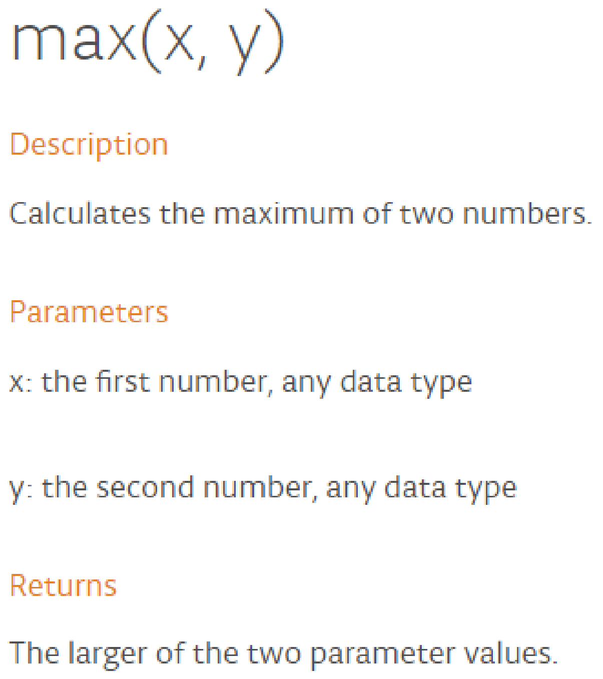
```
sensVal = min(sensVal, 100);
```



```
min(a++, 100);    // avoid this

min(a, 100);
a++;              // use this
```

Understanding Codes: Functions

- 2. max( )



max(x, y)

Description

Calculates the maximum of two numbers.

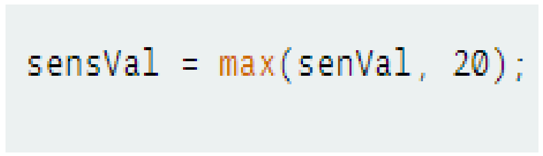
Parameters

x: the first number, any data type

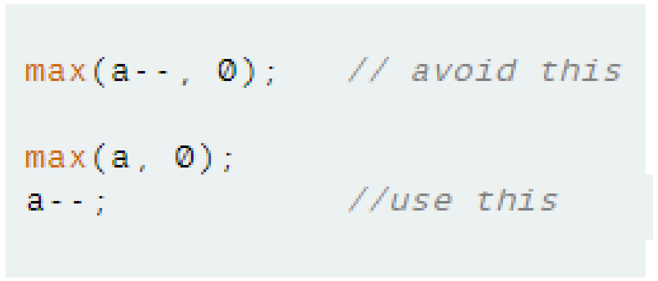
y: the second number, any data type

Returns

The larger of the two parameter values.



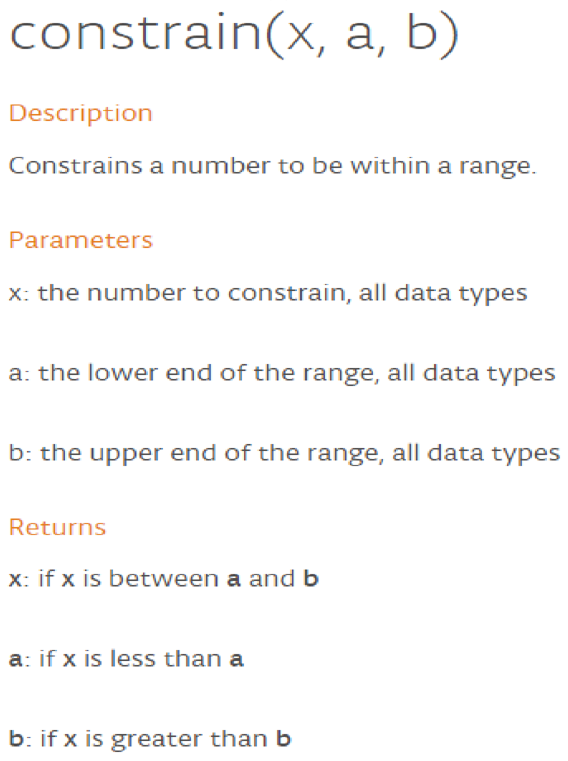
```
sensVal = max(senVal, 20);
```



```
max(a--, 0);   // avoid this

max(a, 0);
a--;           //use this
```

Understanding Codes: Functions

- **3. constrain ( )**



constrain(x, a, b)

Description

Constrains a number to be within a range.

Parameters

x: the number to constrain, all data types

a: the lower end of the range, all data types

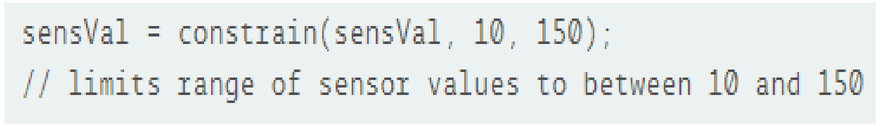
b: the upper end of the range, all data types

Returns

x: if x is between **a** and **b**

a: if x is less than **a**

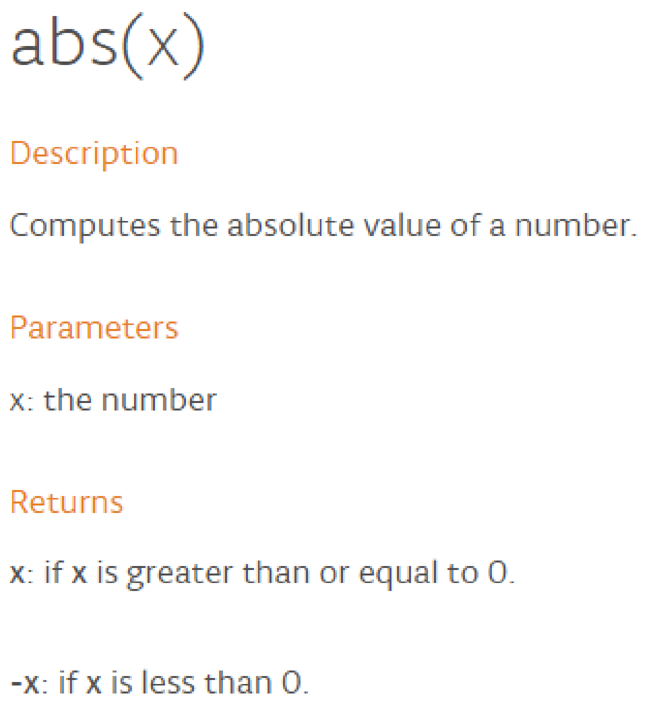
b: if x is greater than **b**



```
sensVal = constrain(sensVal, 10, 150);
// limits range of sensor values to between 10 and 150
```

Understanding Codes: Functions

- **4. abs( )**



abs(x)

Description

Computes the absolute value of a number.

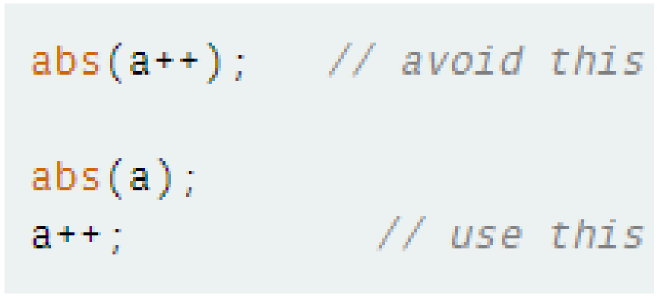
Parameters

x: the number

Returns

x: if x is greater than or equal to 0.

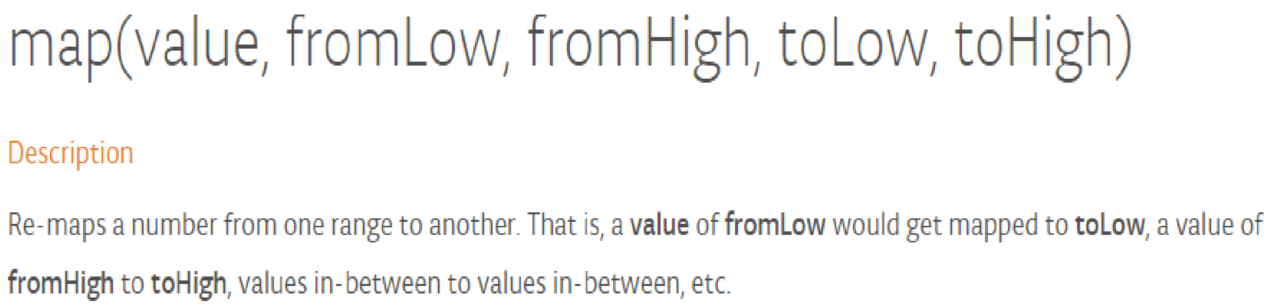
-x: if x is less than 0.



```
abs(a++);    // avoid this

abs(a);
a++;         // use this
```

Understanding Codes: Functions

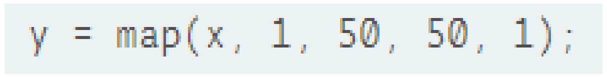
- **5. map( )**



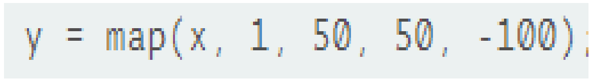
map(value, fromLow, fromHigh, toLow, toHigh)

Description

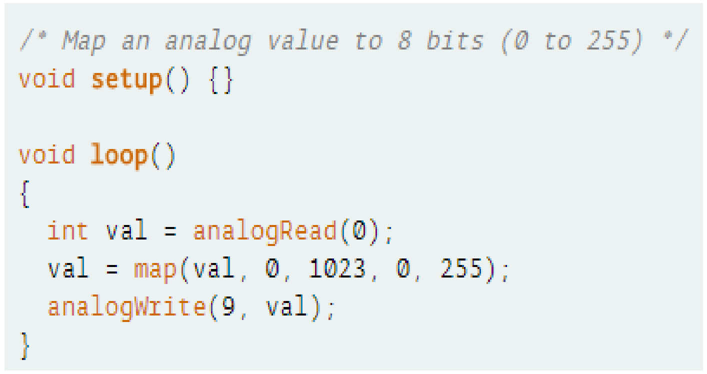
Re-maps a number from one range to another. That is, a **value** of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc.



```
y = map(x, 1, 50, 50, 1);
```



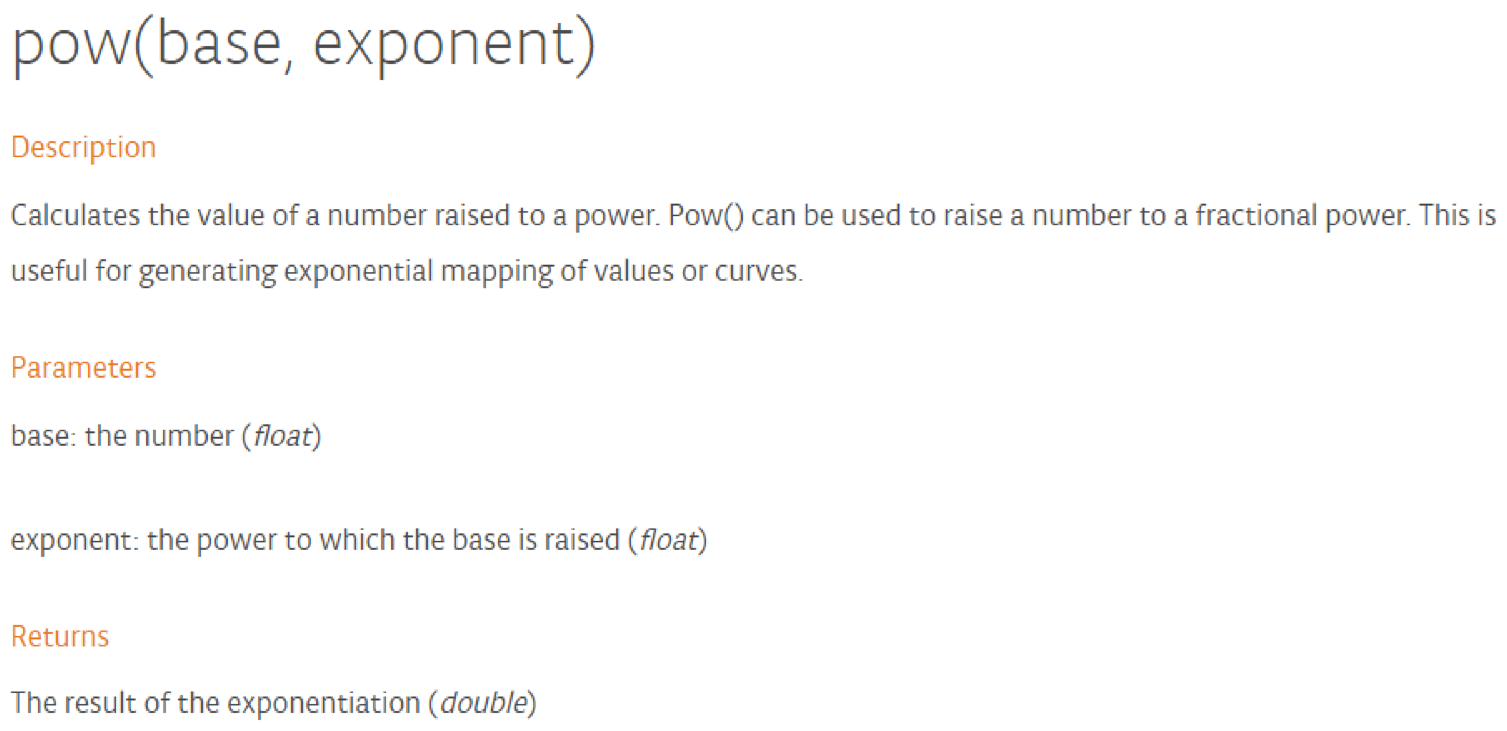
```
y = map(x, 1, 50, 50, -100);
```



```
/* Map an analog value to 8 bits (0 to 255) */
void setup() {}

void loop()
{
  int val = analogRead(0);
  val = map(val, 0, 1023, 0, 255);
  analogWrite(9, val);
}
```

Understanding Codes: Functions

- **6. pow( )**



pow(base, exponent)

Description

Calculates the value of a number raised to a power. Pow() can be used to raise a number to a fractional power. This is useful for generating exponential mapping of values or curves.

Parameters

base: the number (*float*)

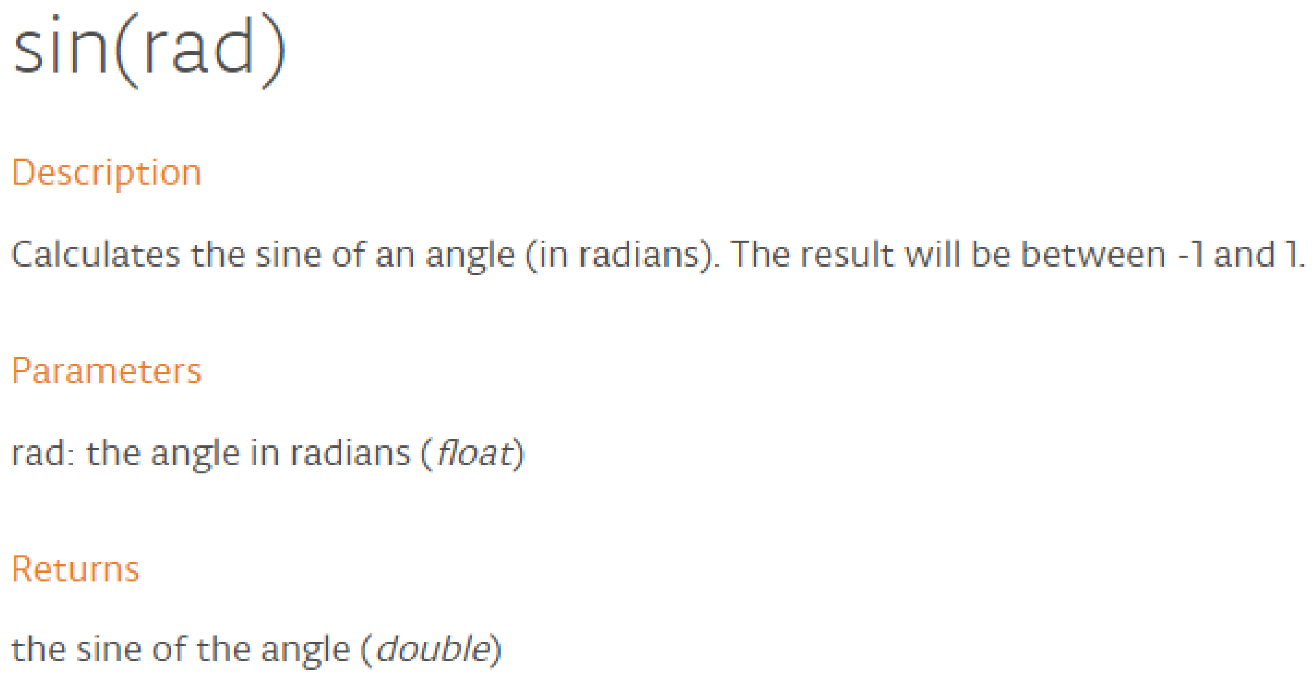
exponent: the power to which the base is raised (*float*)

Returns

The result of the exponentiation (*double*)

Understanding Codes: Functions

- **7. sin( )**



sin(rad)

Description

Calculates the sine of an angle (in radians). The result will be between -1 and 1.

Parameters

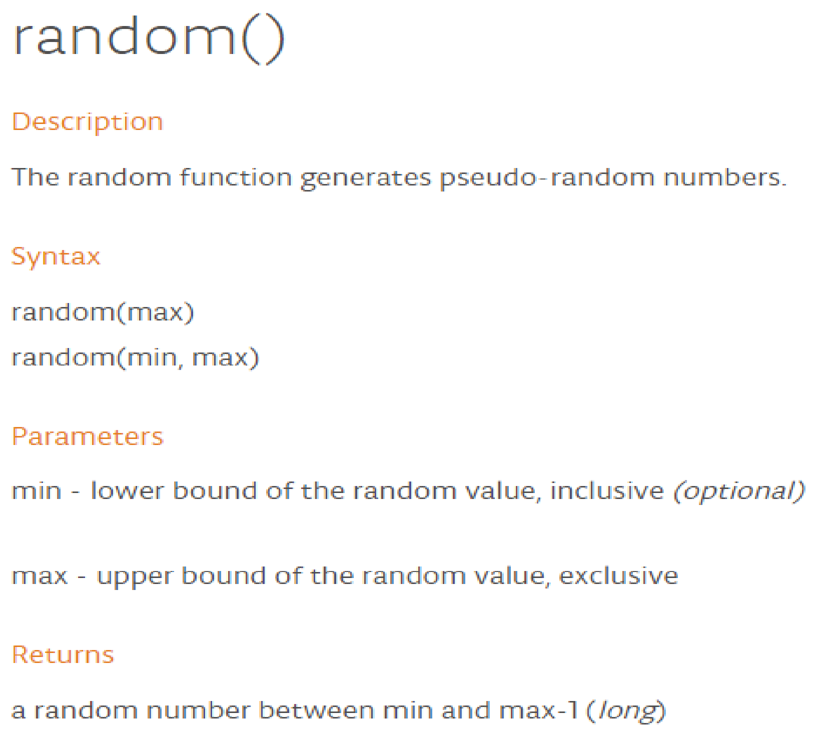
rad: the angle in radians (*float*)

Returns

the sine of the angle (*double*)

Understanding Codes: Functions

- **8. random( )**



random()

Description

The random function generates pseudo-random numbers.

Syntax

random(max)

random(min, max)

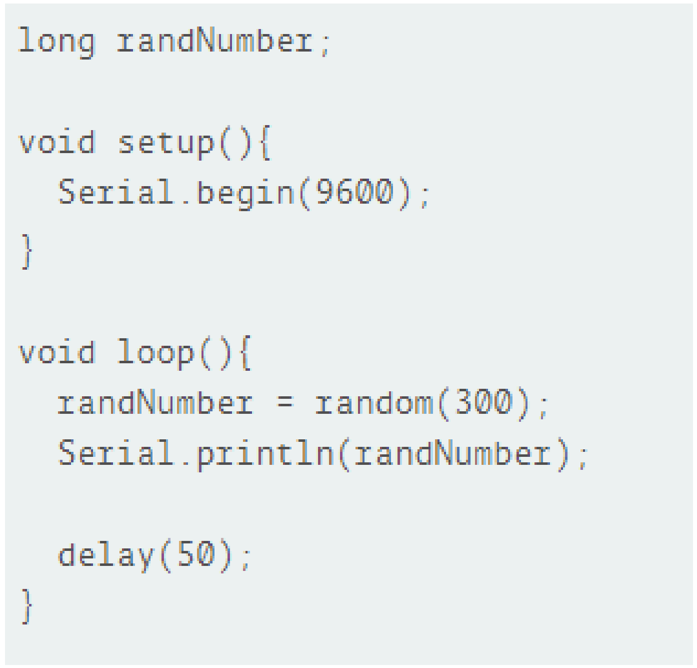
Parameters

min - lower bound of the random value, inclusive *(optional)*

max - upper bound of the random value, exclusive

Returns

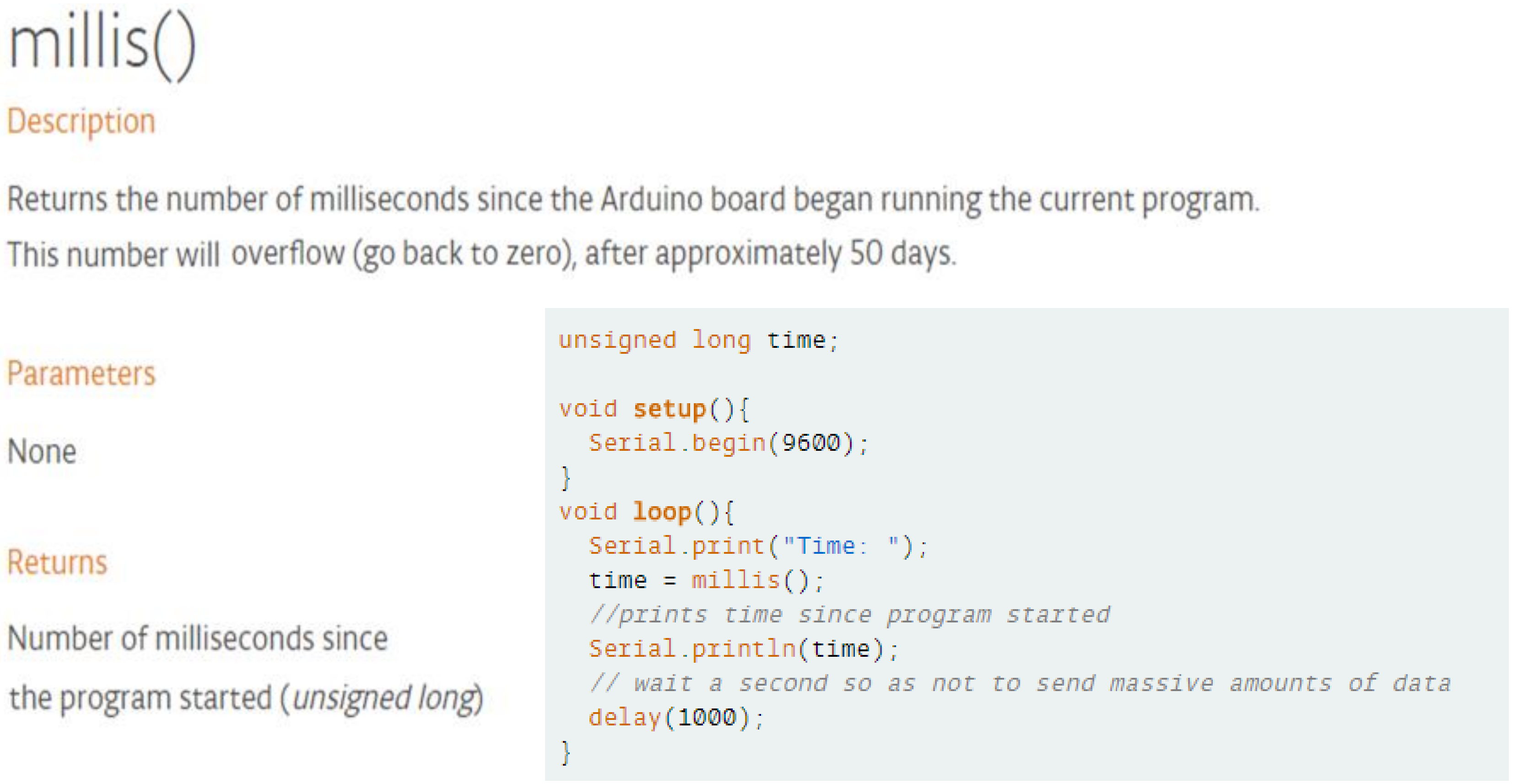
a random number between min and max-1 (*long*)



```
long randNumber;

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

void loop(){
  randNumber = random(300);
  Serial.println(randNumber);

  delay(50);
}
```

Understanding Codes: Functions

- 9. millis( )



millis()

Description

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.

Parameters

None

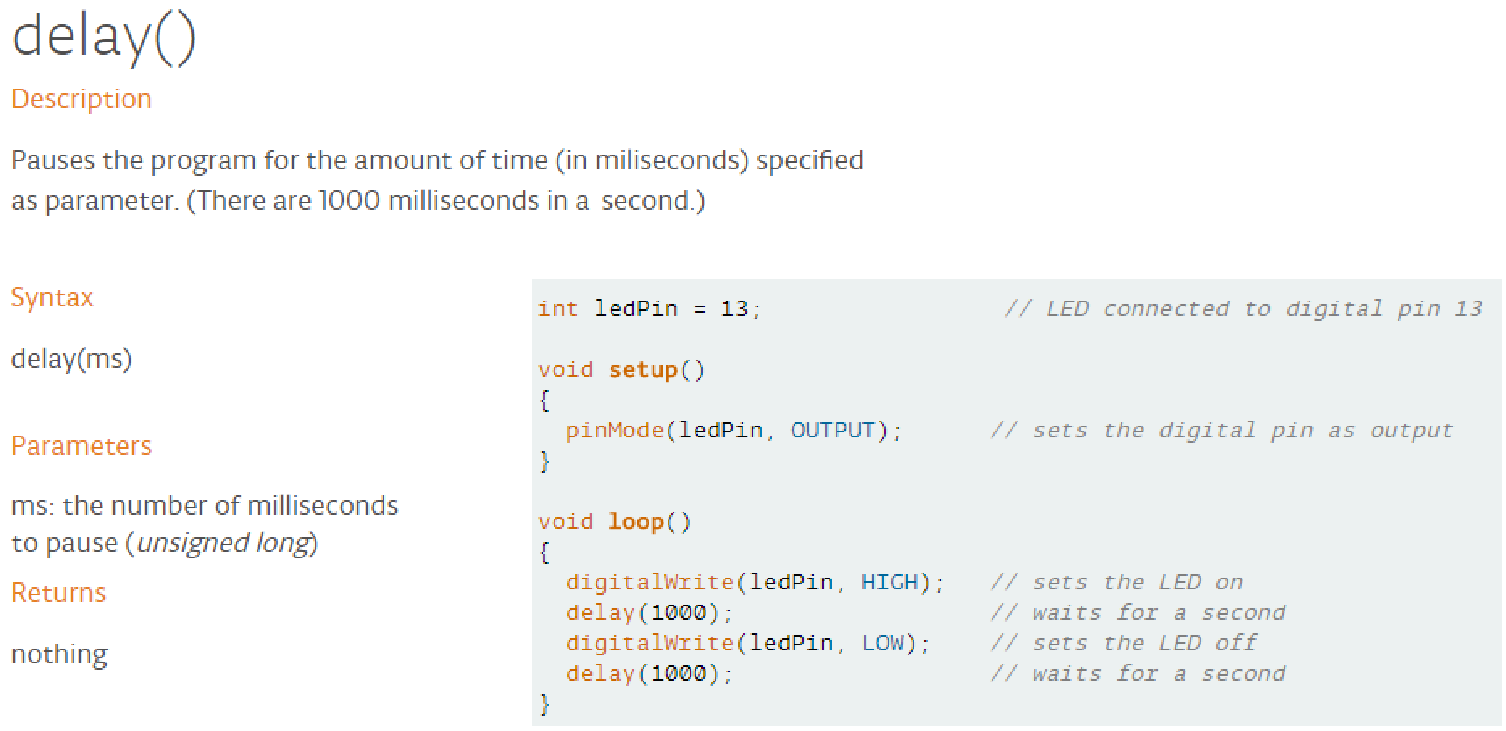
Returns

Number of milliseconds since the program started (*unsigned long*)

```
unsigned long time;

void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.print("Time: ");
  time = millis();
  //prints time since program started
  Serial.println(time);
  // wait a second so as not to send massive amounts of data
  delay(1000);
}
```

Understanding Codes: Functions

- **10. delay( )**



delay()

Description

Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

Syntax

delay(ms)

Parameters

ms: the number of milliseconds to pause (*unsigned long*)

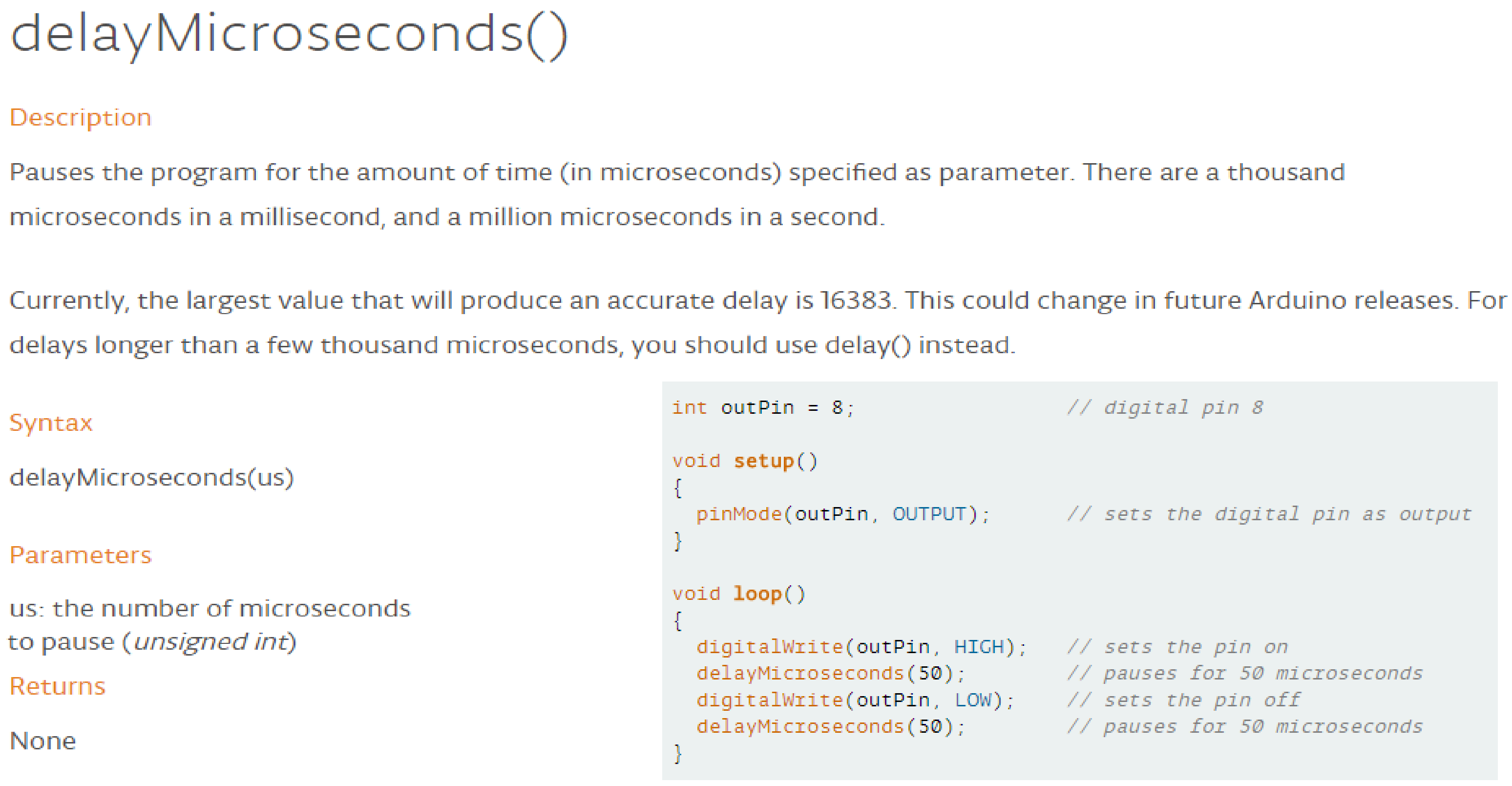
Returns

nothing

```
int ledPin = 13;                 // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

Understanding Codes: Functions

- 11. delayMicroseconds( )



delayMicroseconds()

Description

Pauses the program for the amount of time (in microseconds) specified as parameter. There are a thousand microseconds in a millisecond, and a million microseconds in a second.

Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use delay() instead.

Syntax

delayMicroseconds(us)

Parameters

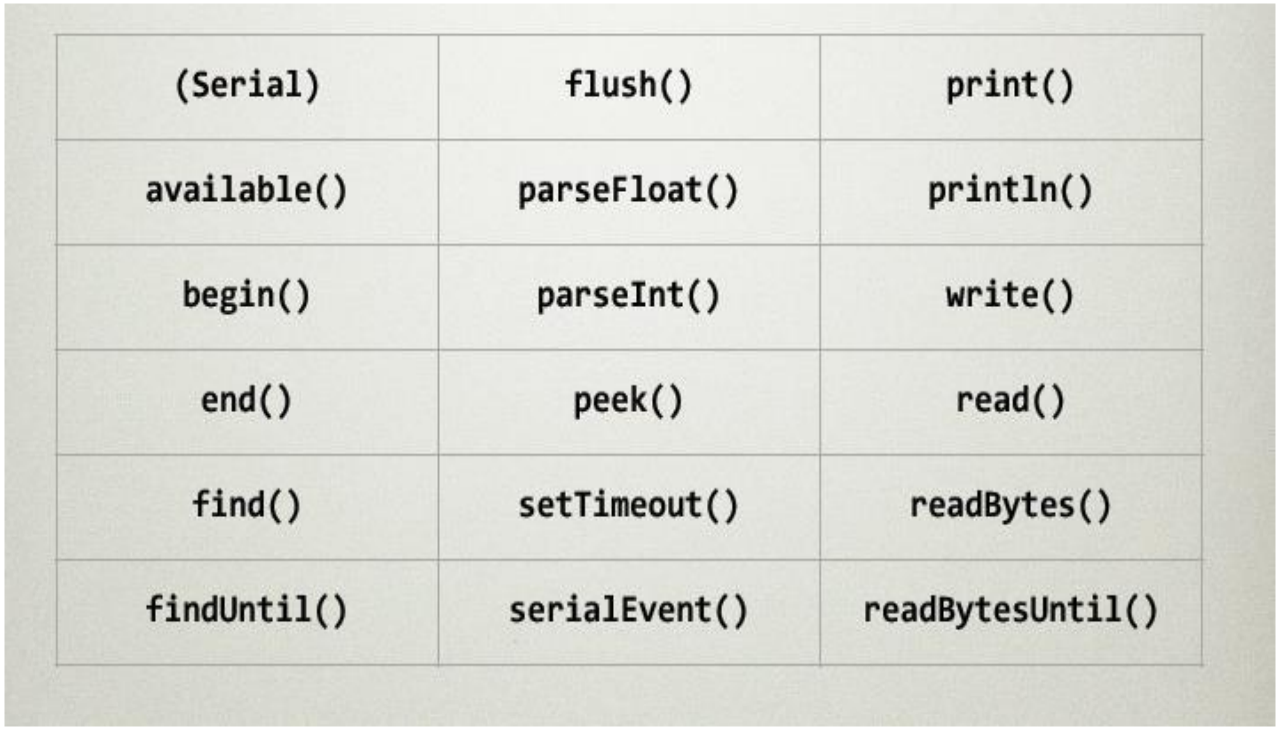
us: the number of microseconds to pause (*unsigned int*)

Returns

None

```
int outPin = 8;                 // digital pin 8

void setup()
{
  pinMode(outPin, OUTPUT);      // sets the digital pin as output
}

void loop()
{
  digitalWrite(outPin, HIGH);   // sets the pin on
  delayMicroseconds(50);        // pauses for 50 microseconds
  digitalWrite(outPin, LOW);    // sets the pin off
  delayMicroseconds(50);        // pauses for 50 microseconds
}
```

Understanding Codes: Serial.xxx



(Serial)	flush()	print()
available()	parseFloat()	println()
begin()	parseInt()	write()
end()	peek()	read()
find()	setTimeout()	readBytes()
findUntil()	serialEvent()	readBytesUntil()