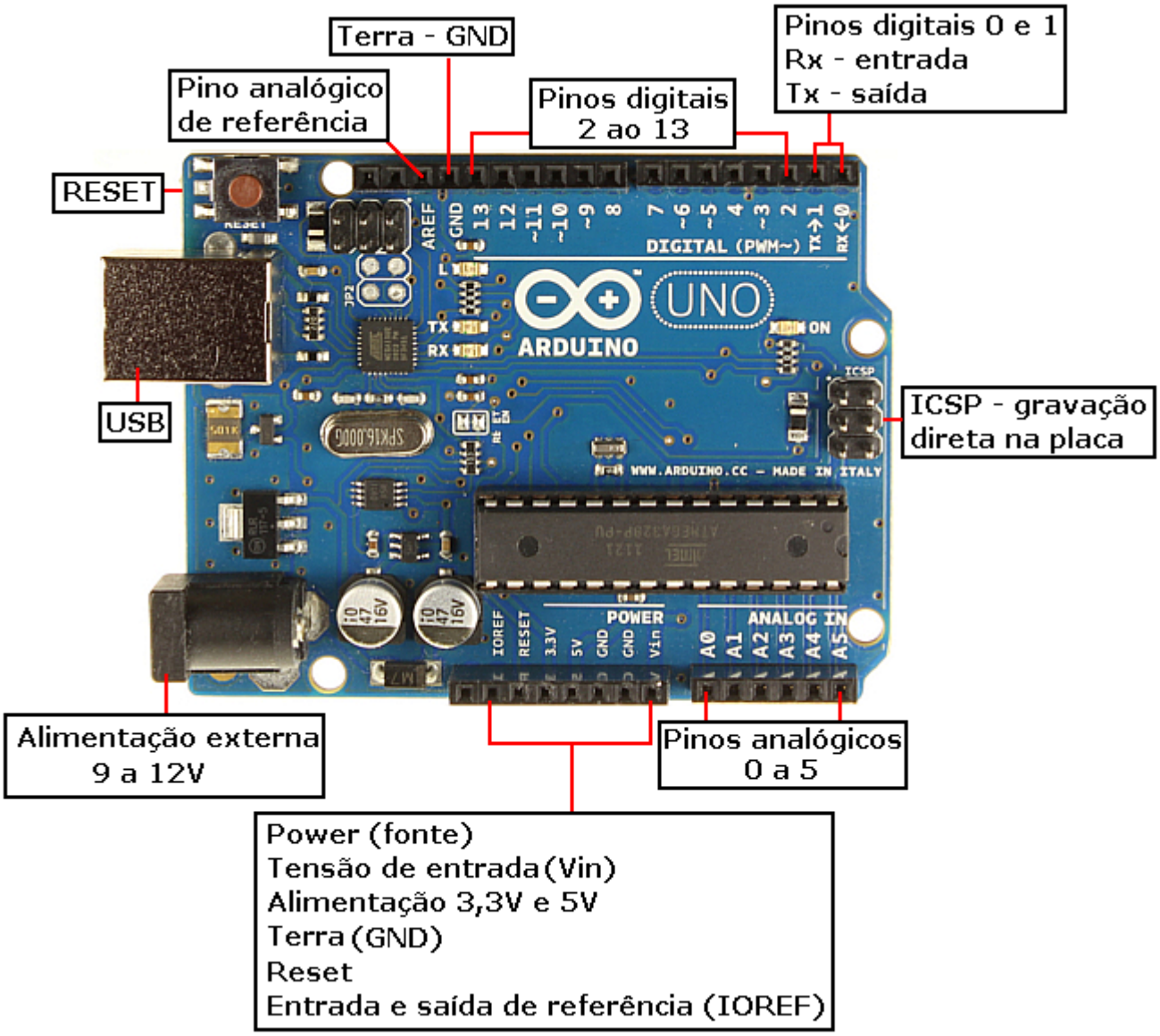
ARDUINO BÁSICO

PLACA DO ARDUINO UNO R3 – FRONT VIEW



O que é o ARDUINO?

É uma plataforma de computação física com um microcontrolador que permite um controle de entrada e de saída de sinais e dispositivos.

Podemos até dizer que Arduino tem o comportamento muito similar ao de um computador, pois podemos criar programas para manipular as entradas e saídas, como por exemplo, sensores, leds, relês, etc.

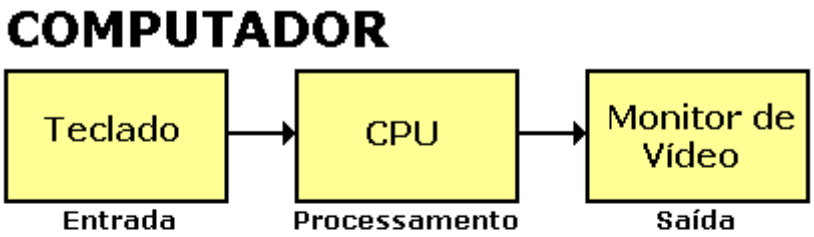
O ARDUINO surgiu na Itália em 2005, e permite que se crie vários projetos envolvendo a eletrônica, por exemplo, sem precisar dominar totalmente a eletrônica, abrindo as portas para pessoas criativas, permitindo assim o desenvolvimento de projetos muito interessantes.

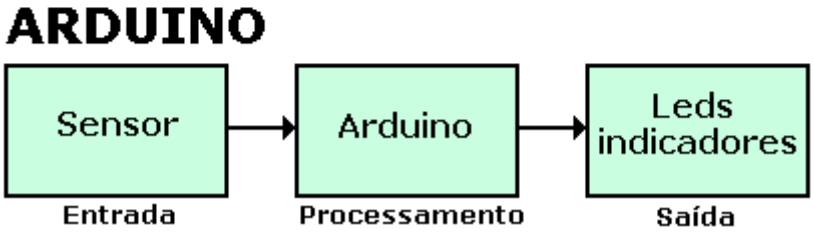
Devido ao seu custo barato, permite o acesso aos estudantes, professores e hobistas.

Qualquer dispositivo que emita dados, seja sinais analógicos ou digitais, ou que possa ser controlado, pode ser conectado ao Arduino.

Dentre esses dispositivos, podemos citar: leds, sensores de qualquer espécie, como por exemplo, temperatura, luz, pressão, umidade, módulos Ethernet, receptores GPS, etc.

DIAGRAMA DE BLOCOS BÁSICO – Comparando Arduino e Computador





Resumindo, o Arduino é um Hardware e Software de fonte aberta.

O projeto, fontes e esquemas podem ser usados de forma livre.

Você pode criar em casa sua própria placa Arduino baseado no diagrama de circuito do projeto original. Só não pode colocar o nome Arduino nessa placa.

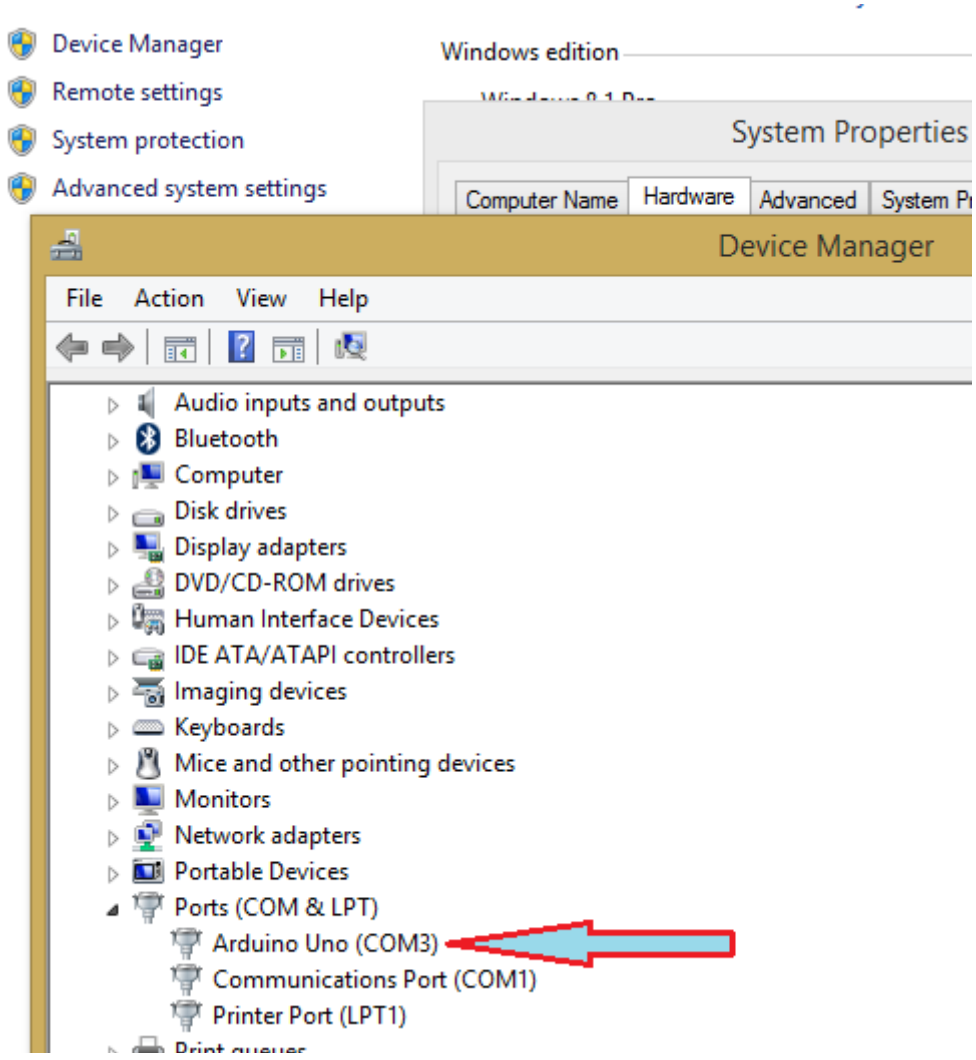
Instalação do programa:

O link abaixo acessa o site oficial do Arduino. Clicar na opção *Download* para selecionar o software que inclui o executável "Arduino IDE" e respectivo driver. O Arduino pode ser executado em plataforma Windows, Mac ou Linux.

https://www.arduino.cc/en/Main/Software#

Veja a seguir as opções de download, de acordo com o sistema operacional que será utilizado. A versão 1.6.7 foi atualizada em 21 de janeiro de 2016.

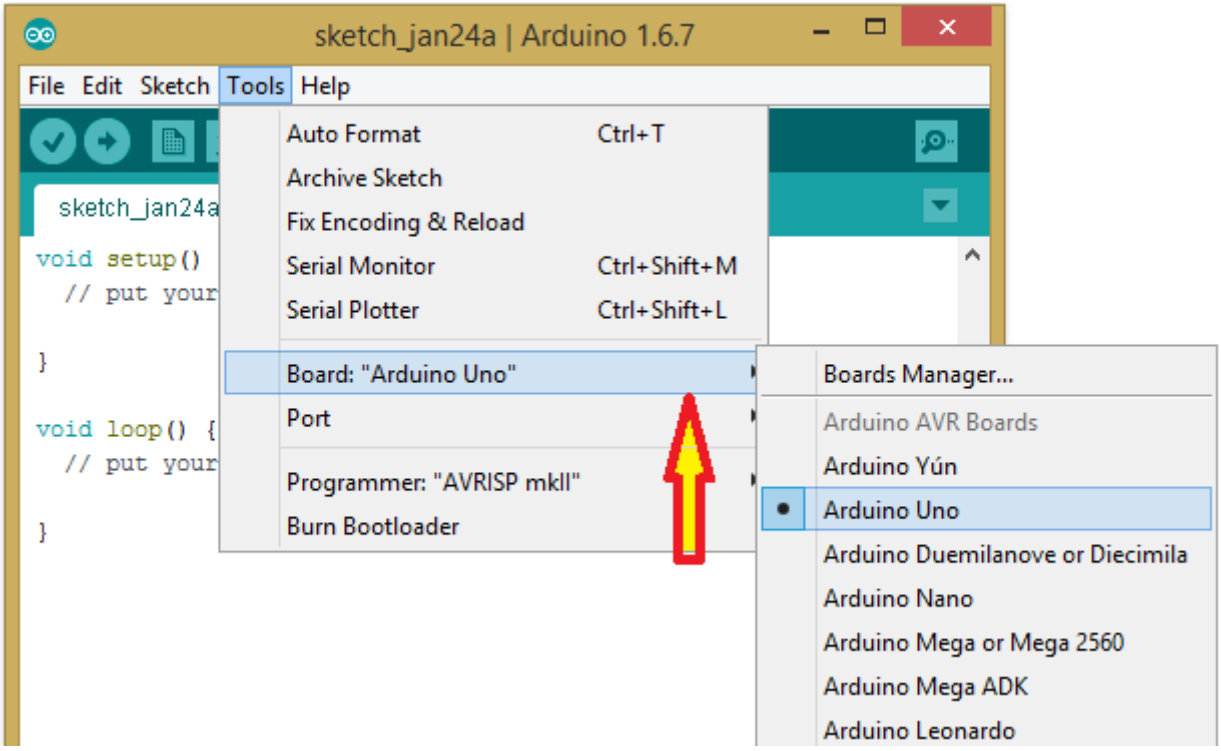
A figura a seguir mostra a placa Arduino sendo reconhecida pelo Windows 8.1 no gerenciador de Hardware.



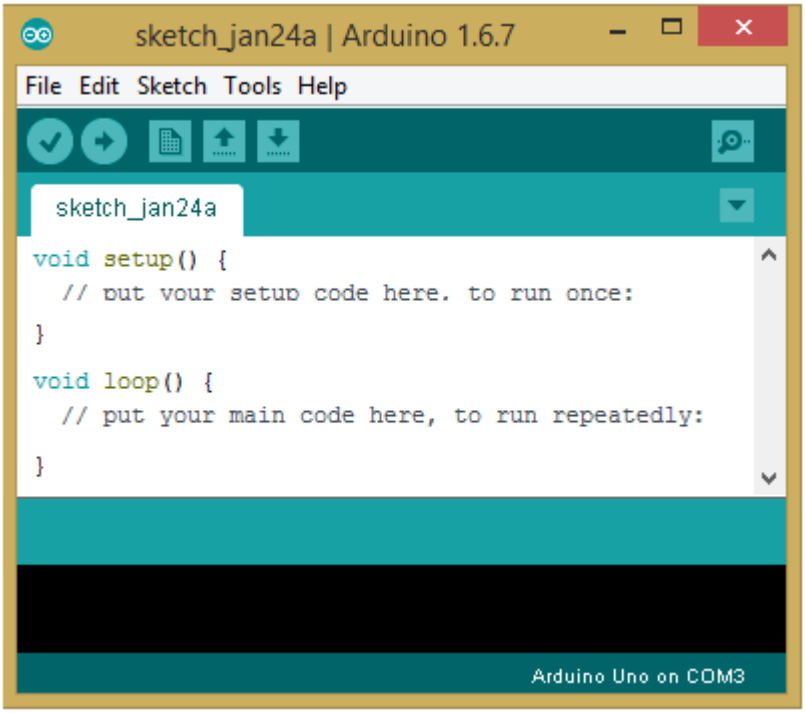
Observe no “sketch” do Arduino que a *Serial Port* deve ser coincidente com a porta detectada no gerenciador de dispositivos do Windows, caso contrário o Arduino não funcionará.

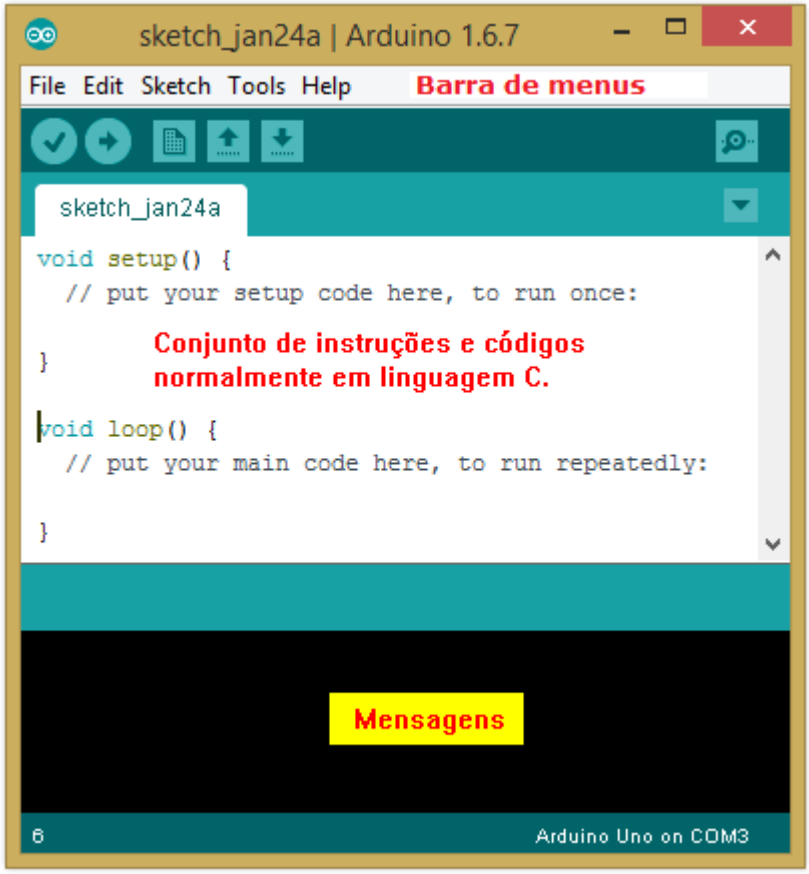
É recomendável baixar o software e driver do site oficial, pois lá são disponibilizadas as atualizações oficiais que incluem as versões mais atualizadas dos processadores da placa Arduino.

A figura a seguir mostra que a placa do Arduino UNO está devidamente reconhecida para iniciar o desenvolvimento dos projetos (Tools -> Board).



O IDE do Arduino: Depois de instalado, o IDE do Arduino deve ter o aspecto mostrado na figura a seguir:

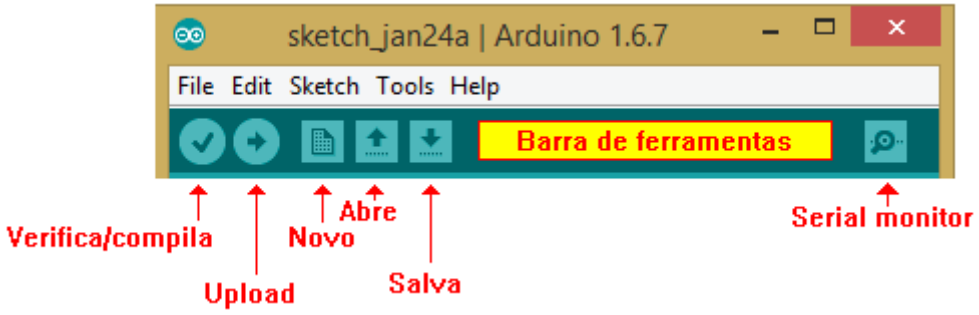




IDE (Integrated Development Environment), é um ambiente integrado para desenvolvimento de software

As mensagens no campo correspondente poderão indicar, além da normalidade quando da compilação, erros no *sketch* principalmente quando são digitados códigos errados ou incompletos.

Normalmente indicação de erro aparece em vermelho com a indicação da linha onde ocorreu o erro.



Verifica/compila (Verify) – verifica se há erros de código

Upload – faz o upload do sketch para a placa do Arduino

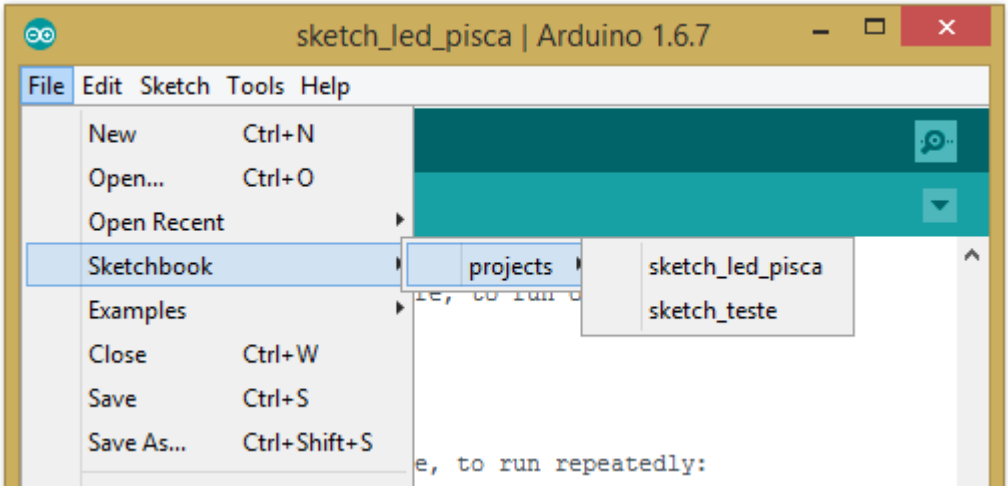
Novo (New)– cria um sketch em branco

Abre (Open) – mostra uma lista de sketches em seu sketchbook para abrir

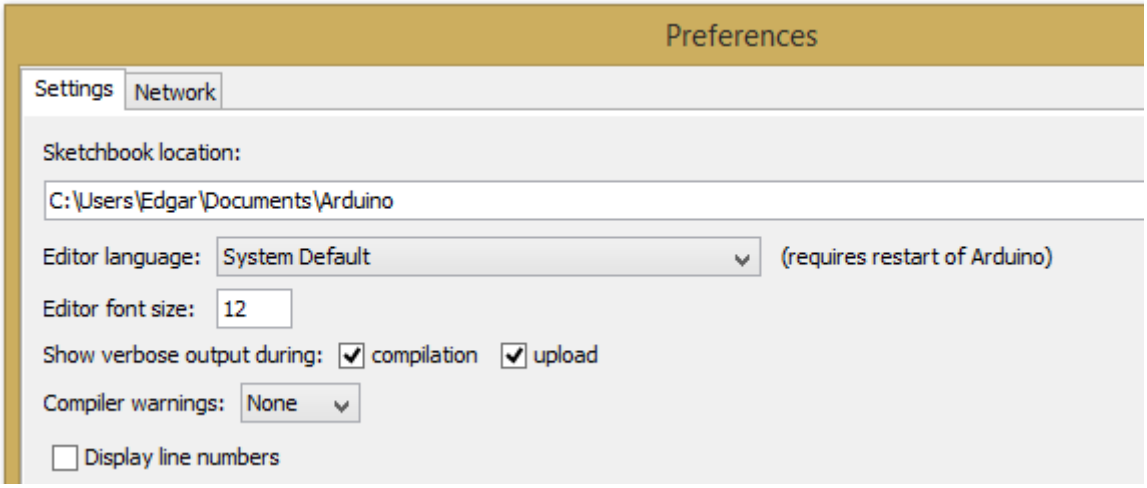
Salva (Save) – salva o sketch atual em seu sketchbook

Serial monitor – exibe os dados seriais enviados do Arduino.

A figura abaixo mostra a opção “Sketchbook” em File (barra de menus):



Resumindo, no Sketchbook ficam armazenados os seus projetos. A localização do Sketchbook pode ser determinada pelo usuário, conforme ilustra a figura a seguir.



OBS: Pode haver pequenas diferenças no IDE do Arduino, dependendo do sistema operacional utilizado, como por exemplo, o posicionamento dos botões.

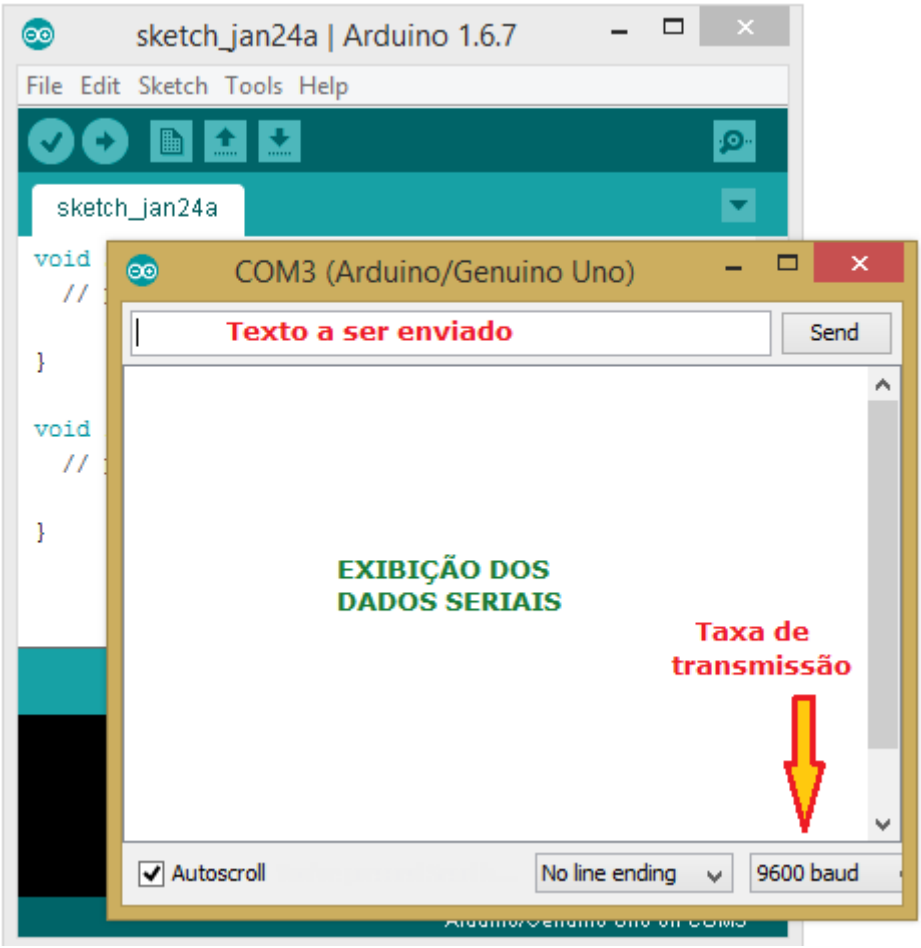
No entanto, o IDE permanece o mesmo.

Sobre o Serial monitor:

O serial monitor muito útil na depuração de código, pois exibe os dados enviados do Arduino.

Da mesma forma, essa ferramenta permite que sejam enviados dados de volta para o Arduino.

A figura a seguir mostra o aspecto da tela do Serial Monitor:

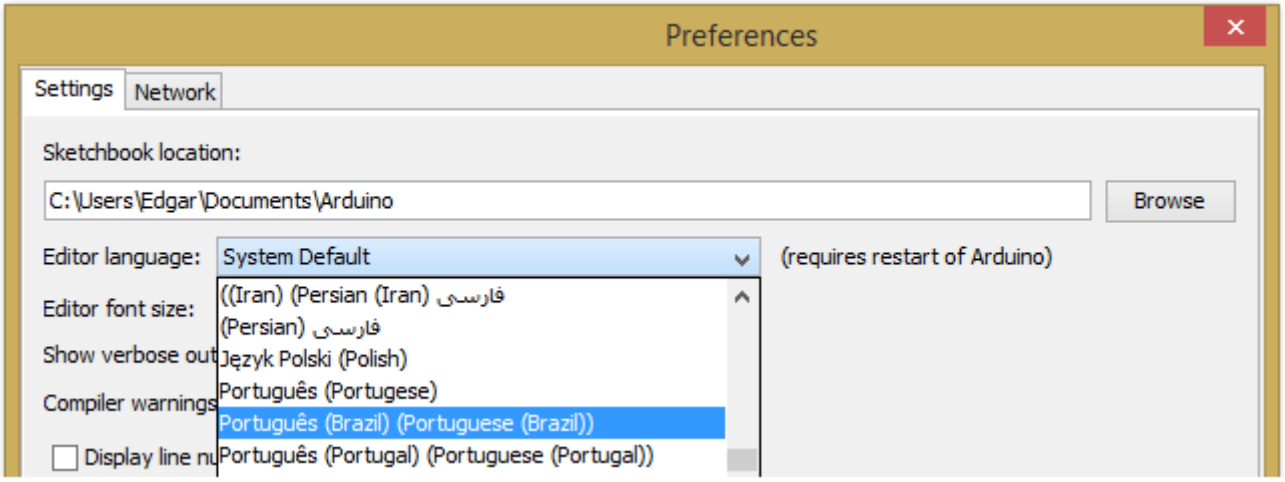


É importante salientar que para o Serial monitor receber dados do Arduino é necessário que no *sketch* tenha sido digitado o código correto para que isso aconteça, conforme veremos adiante.

Da mesma forma o Arduino não receberá nenhum dado, a não ser que o mesmo tenha sido codificado para tal.

Configurando o Arduino para idioma português do Brasil.

Para mudar o idioma do editor, basta clicar na barra de menu: File => Preferences => Editor language, conforme mostra a figura a seguir:



Códigos do Arduino (Language reference): A programação do Arduino divide-se basicamente em 3 partes:

1 – Estrutura (Structure)

2 – Valores (Values) onde estão incluídos *variáveis* (variables) e *constantes* (constants)

3 – Funções (functions)

Structure	Variables	Functions
setup() loop() **Control Structures** if if...else for switch case while do... while break continue return goto **Further Syntax** ; (semicolon) {} (curly braces) // (single line comment) /* */ (multi-line comment) #define #include **Arithmetic Operators** = (assignment operator) + (addition) - (subtraction) * (multiplication) / (division) % (modulo)	**Constants** HIGH | LOW INPUT | OUTPUT|INPUT_PULLUP true | false integer constants floating point constants **Data Types** void boolean char unsigned char byte int unsigned int word long unsigned long float double string – char array String – object array **Conversion** char() byte() int() word() long()	**Digital I/O** pinMode() digitalWrite() digitalRead() **Analog I/O** analogReference() analogRead() analogWrite() – *PWM* **Advanced I/O** tone() noTone() shiftOut() shiftIn() pulseIn() **Time** millis() micros() delay() delayMicroseconds() **Math** min() max() abs() constrain() map() pow()

Comparison Operators

== (equal to)
!= (not equal to)
< (less than)
> (greater than)
<= (less than or equal to)
>= (greater than or equal to)

Boolean Operators

&& (and)
|| (or)
! (not)

Pointer Access Operators

* dereference operator
& reference operator

Bitwise Operators

& (bitwise and)
| (bitwise or)
^ (bitwise xor)
~ (bitwise not)
<< (bitshift left)
>> (bitshift right)

Compound Operators

++ (increment)
-- (decrement)
+= (compound addition)
-= (compound subtraction)
*= (compound multiplication)
/= (compound division)
&= (compound bitwise and)
|= (compound bitwise or)

float()

Variable Scope & Qualifiers

variable scope
static
volatile
const

Utilities

sizeof()

sqrt()

Trigonometry

sin()
cos()
tan()

Random Numbers

randomSeed()
random()

Bits and Bytes

lowByte()
highByte()
bitRead()
bitWrite()
bitSet()
bitClear()
bit()

External Interrupts

attachInterrupt()
detachInterrupt()

Interrupts

interrupts()
noInterrupts()

Communication

Serial
Stream

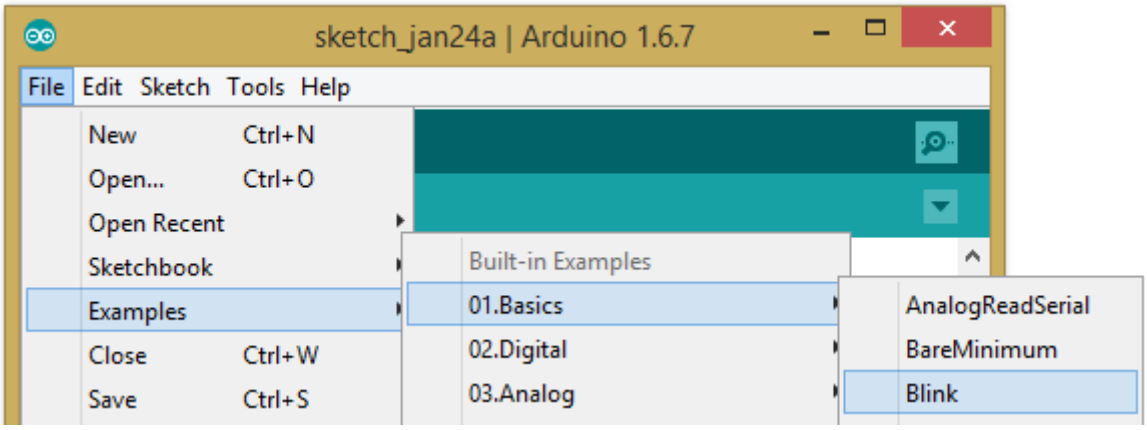
Leonardo Specific

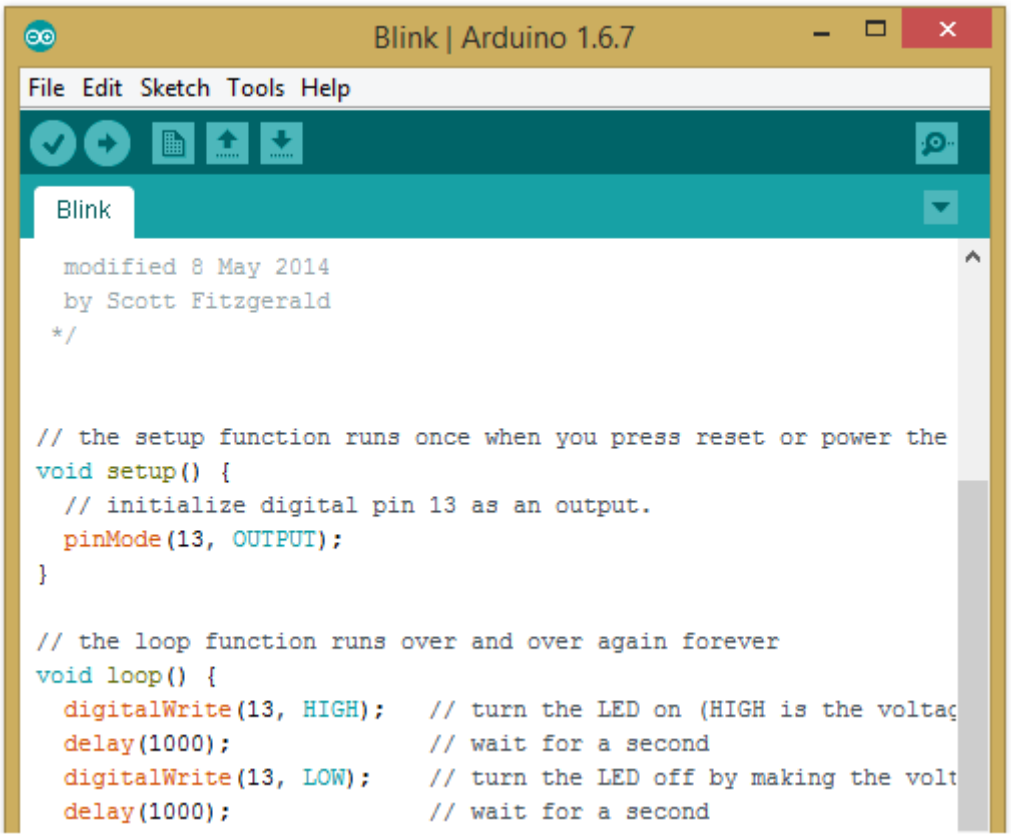
Keyboard
Mouse

Testando o Arduino:

Para verificar se a instalação do driver e do IDE Arduino estão corretos, podemos fazer um teste, fazendo um upload de um exemplo para a placa do Arduino. Escolheremos um exemplo bem simples, no caso “led piscante” apenas para testar se está tudo em ordem.

File ===> Examples ===> 01.Basics ===> Blink



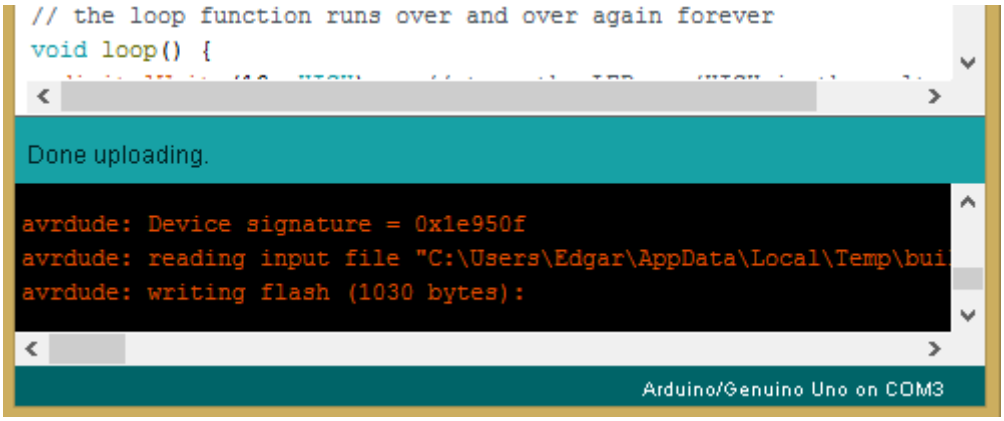


Ao abrir o exemplo, o *sketch* deverá ter a aparência mostrada na figura acima.

Os comentários contidos entre /*.....*/ são ignorados e da mesma forma, qualquer observação colocada após // em qualquer linha também será ignorada.

Após fazer a verificação, o próximo passo é fazer o upload para o Arduino.

A figura a seguir mostra que o upload ocorreu sem problemas.



Não é necessário ligar nenhum led para fazer o teste, pois no Arduino já existe ligado no pino 13 um micro led (SMD), de cor amarela que permite visualizar a experiência.

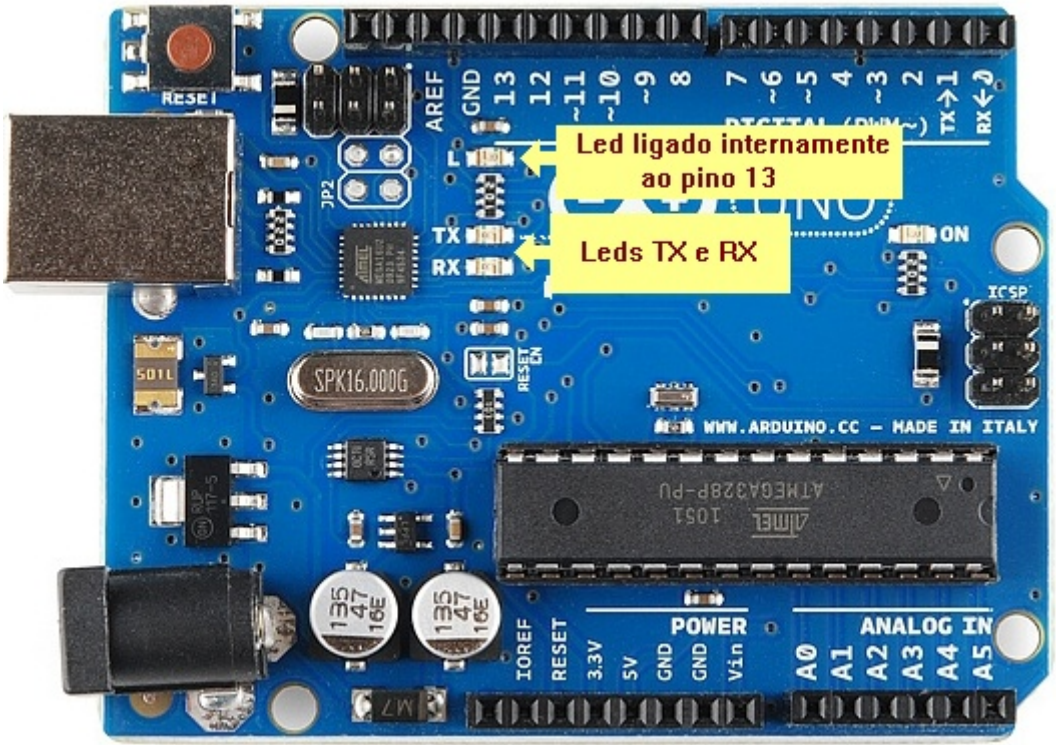
Durante o processo de upload, os micro leds TX e RX piscarão.

Depois de concluído o upload, na barra de status aparecerá a informação “Done uploading” e os micro leds TX e RX param de piscar.

Após alguns segundos o Arduino deverá estar funcionando, de acordo com os códigos inseridos no *sketch*, ou seja, o micro led deverá estar piscando.

A ligação do Arduino via USB assegura a alimentação de 5V necessária ao seu funcionamento, não havendo a necessidade de conectá-lo uma fonte externa.

A figura a seguir mostra a localização do led ligado ao pino 13 e dos leds TX e RX:



OBS: Embora o micro led da placa do Arduino esteja ligado internamente ao pino 13, este pino pode também ser utilizado para qualquer tipo de conexão externa.

Hardware, Software (Bootloader) do Arduino:

Como vimos anteriormente o Arduino é composto basicamente de duas partes:

Hardware: que é o conjunto de componentes eletrônicos montados em uma placa de circuito impresso, que se destina ao desenvolvimento de protótipos e projetos.

Bootloader: nada mais é do que um software ou aplicativo que fica residente na memória de programas do microcontrolador embarcado no Arduino.

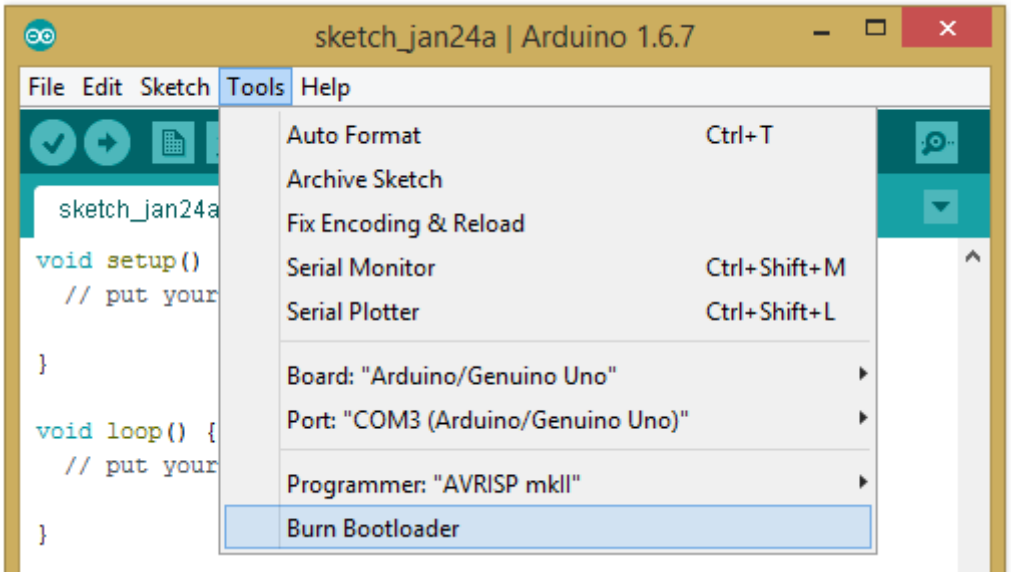
Existe também uma interface gráfica, ou seja um programa que roda num padrão PC ambiente Windows, ou até mesmo em uma plataforma Linux ou Mac OS.

É nessa interface gráfica que são criados os programas para posteriormente serem carregados para o hardware do Arduino.

Essa interface gráfica recebe o nome de IDE ou ambiente de desenvolvimento integrado. Conforme vimos anteriormente: ***I****ntegrated* ***D****evelopment* ***E****nvironment.*

O hardware do Arduino é baseado nos microcontroladores AVR da Atmel, particularmente o ATmega 328 (Arduino UNO) e o ATmega 1280 (Arduino MEGA).

Temos no menu Tools, a opção *Burn Bootloader* grava o Arduino Bootloader, que é um trecho de código do chip, para torná-lo compatível com o próprio IDE do Arduino.



Essa opção só é válida se for utilizado um programador AVR e caso tenha sido substituído o chip do Arduino.

É também válido caso tenham sido adquiridos chips em branco para utilização em projeto embarcado.

A menos que se deseje gravar vários chips para desenvolvimento de projetos embarcados, torna-se mais em conta adquirir um chip ATmega.

O que é um sistema embarcado?

Uma definição bastante simples e compreensível é dada a seguir:

*Um **sistema embarcado** (ou **sistema embutido**, ou **sistema embebido**) é um sistema microprocessado no qual o computador é completamente encapsulado ou dedicado ao dispositivo ou sistema que ele controla. Diferentemente de computadores de propósito geral, como o computador pessoal, um sistema embarcado realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos. Já que o sistema é dedicado a tarefas específicas, através de engenharia pode-se aperfeiçoar o projeto reduzindo tamanho, recursos computacionais e custo do produto.*

Em geral tais sistemas não podem ter sua funcionalidade alterada durante o uso. Caso queira-se modificar o propósito é necessário reprogramar todo o sistema.

Sistemas como PDAs são geralmente considerados sistemas embarcados pela natureza de seu hardware, apesar de serem muito mais flexíveis em termos de software. Fisicamente, os sistemas embarcados passam desde MP3 players a semáforos.

Quando tratamos de software na plataforma do Arduino, podemos referir-nos: ao ambiente de desenvolvimento integrado do Arduino e ao software desenvolvido por nós para enviar para a nossa placa.

O ambiente de desenvolvimento do Arduino é um compilador GCC (C e C++) que usa uma interface gráfica construída em Java.

Basicamente se resume a um programa IDE muito simples de se utilizar e de estender com bibliotecas que podem ser facilmente encontradas.

As funções da IDE do Arduino são basicamente duas: Permitir o desenvolvimento de um software e enviá-lo à placa para que possa ser executado.

Pinos do Arduino (pinos com funções especiais):

Existem pinos do Arduino que possuem características especiais, que podem ser usadas efetuando as configurações adequadas através da programação. São eles:

PWM (pinos 3, 5, 6, 9, 10 e 11): Tratado como saída analógica, na verdade é uma saída digital que gera um sinal alternado (0 e 1) onde o tempo que o pino fica em nível 1 (ligado) é controlado.

É usado para controlar velocidade de motores, ou gerar tensões com valores controlados pelo programa.

Porta serial USART (pino 0 - Rx recebe dados e pino 1 Tx envia dados): Podemos usar um pino para transmitir e um pino para receber dados no formato serial assíncrono (USART).

É possível conectar um módulo de transmissão de dados via bluetooth, por exemplo, e nos comunicarmos com o Arduino remotamente.

Comparador analógico (pinos 6 e 7): Podemos usar dois pinos para comparar duas tensões externas, sem precisar fazer um programa que leia essas tensões e as compare.

Essa é uma forma muito rápida de comparar tensões sendo feita pelo hardware sem envolver programação.

Interrupção externa (pinos 2 e 3): Podemos programar um pino para avisar o software sobre alguma mudança em seu estado.

Podemos ligar um botão a esse pino, por exemplo, e cada vez que alguém pressiona esse botão o programa rodando dentro da placa é desviado para um bloco que você escolheu. É usado para detectar eventos externos à placa.

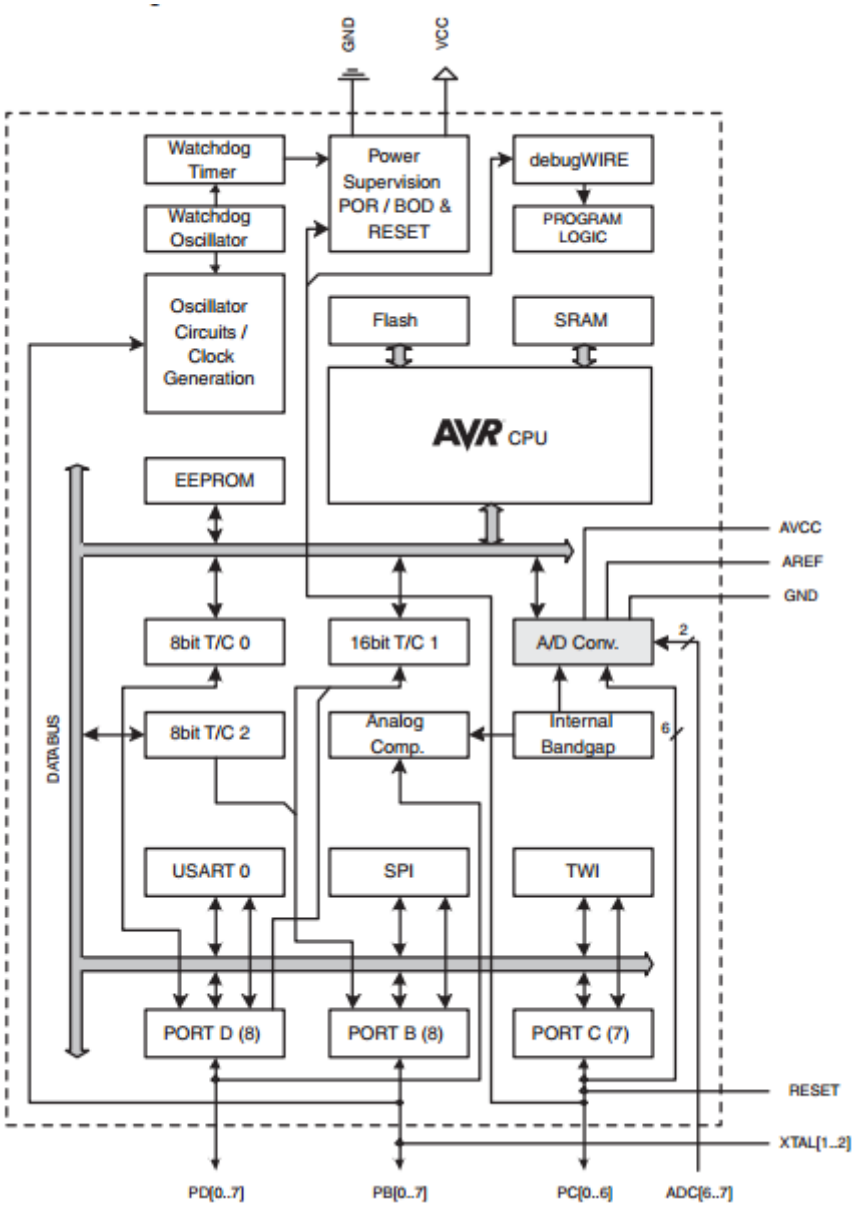
Porta SPI (Serial Peripheral Interface): É um padrão de comunicação serial síncrono, bem mais rápido que a USART. É nessa porta que conectamos, por exemplo, cartões de memória SD e muitos outros dispositivos similares.

Os pinos são:

10 (SS) – Slave Select
11 (MOSI) – Master Out Slave In
12 (MISO) – Master In Slave Out
13 (SCK) – Serial Clock

Slave Select: habilita ou desabilita dispositivos específicos
SCK – sincroniza a transmissão de dados geradas pelo mestre (master)
Miso – envia dados para o mestre (master)
Mosi – envia dados para os periféricos

A figura a seguir mostra o diagrama de blocos do microcontrolador ATmega 328P da Atmel.



ESTRUTURA DA LINGUAGEM DO ARDUINO

CONSTANTES: As constantes possuem valores pré-definidos, isto é, não mudam.

True / False: (verdadeiro/falso), são denominadas constantes booleanas que definem os estados lógicos 0 e 1 (álgebra de Boole).

True é tudo que for diferente de zero, enquanto que False é igual a zero.

High / Low: (alto/baixo), definem os níveis de tensão nos pinos do Arduino, onde High representa o nível de +5 volts (ligado) e Low representa o nível de terra ou GND (desligado).

Output / Input: (saída/entrada), são constantes que devem ser usadas com a função pinMode, para definir se o pino do Arduino será configurado como entrada ou saída.

VARIÁVEIS: As variáveis estão relacionadas com as posições de memória, e como o próprio nome sugere essas posições podem mudar durante a execução do programa.

Todas as variáveis devem ser declaradas no início do programa, devendo receber um nome, um tipo e algumas vezes um valor inicial.

Por exemplo, *int ledPin=9;* é uma variável que está sendo definida com o nome de ledPin recebendo um valor igual a 9. Essa variável é do tipo "int".

Características das variáveis:

1- Uma variável do tipo "int" (integer) armazena números inteiros, sem casa decimal, positivos ou negativos de até 2 bytes (espaço na memória RAM).
2- Uma variável do tipo "long" armazena números inteiros que ocupam até 4 bytes na memória RAM.
3- Uma variável do tipo "boolean" armazena até 1 byte.
4- Uma variável do tipo "float" tem a característica de armazenar valores em ponto flutuante de até 4 bytes.
5- Uma variável do tipo "char" (character) tem a extensão de 1 byte e armazena um caractere ASCII.

Um byte (Binary Term em inglês) é um dos tipos de dados integrais em computação. É usado com frequência para especificar o tamanho ou quantidade da memória ou da capacidade de armazenamento de um computador, independentemente do tipo de dados lá armazenados.

A quantidade padronizado de byte foi definido como sendo de 8 bits. O byte de 8 bits é, por vezes, também chamado de octeto, nomeadamente no contexto de redes de computadores e telecomunicações.

A uma metade de um byte, dá-se o nome de nibble ou semioctecto.

Bit (simplificação para dígito binário, "BInary digiT" em inglês) é a menor unidade de informação que pode ser armazenada ou transmitida. Usada na Computação e na Teoria da Informação. Um bit pode assumir somente 2 valores, por exemplo: 0 ou 1, falso ou verdadeiro.

FUNÇÕES: Podemos definir a função como sendo algo que relaciona um argumento a um valor numérico por meio de uma regra de associação ou fórmula.

A função é baseada portanto, em um conceito matemático.

Podemos dizer que funções são sub-rotinas ou procedimentos, agrupados em blocos formando assim o programa principal.

O *sketch* do Arduino (programa) é composto por duas partes principais:

função setup()

função loop()

Enquanto na linguagem "C" a função *main()* é a única obrigatória, no Arduino as funções *setup()* e *loop()* são obrigatórias.

Toda a função deve ter um nome e os procedimentos que ela vai executar, que devem estar contidos entre chaves ({....}).

Dentro dos parênteses do nome da função vem os parâmetros que a função deve receber, sempre finalizando com ponto e vírgula (;).

Por exemplo:

```
Void setup() {
pinMode(10, OUTPUT);}
```

Neste caso, essa função está determinando que o pino 10 do Arduino está sendo configurado como saída.

A função *loop()* é chamada logo após a função *setup(),* sendo executada repetidamente, daí então o nome de *loop.*

Portanto, a função *setup()* configura parâmetros sendo executada apenas uma vez, enquanto que a função *loop()* é executada repetidamente, até o desligamento da alimentação do Arduino.

A função *loop()* não recebe nenhum parâmetro e também não retorna nenhum valor, mas chama outras funções, como por exemplo:

```
void loop(){
digitalWrite(10,HIGH);
delay(1000);}
```

Neste caso, *digitalWrite* está colocando o pino 10 do Arduino em nível alto, ou seja, se houver um led nesse pino o mesmo acenderá e um delay de 1 segundo (1000ms) é utilizado para executar a próxima instrução.

Vejamos a seguir as funções mais utilizadas no Arduino:

pinMode: É um canal físico de comunicação externa do Arduino.

O Arduino tem 20 desses canais, assim distribuídos: 14 digitais 0 a 13) e 6 analógicos (A0 a A5), sendo que estes últimos podem também ser configurados como canais digitais.

O *pinMode* pode ser configurado como entrada (INPUT) ou saída (OUTPUT).

digitalRead: Lê o estado lógico de um pino que foi previamente definido na entrada, sendo usado para ler qualquer elemento que retorne 1 ou 0 (true/false, ON/OFF).

int chave = digitalRead(9)

Neste caso, a variável chave do tipo "int" vai armazenar o estado lógico lido no pino 9 do Arduino.

digitalWrite: Envia para o pino do Arduino um nível lógico alto ou baixo (HIGH, LOW), devendo ser configurado como saída pelo pinMode.

```
pinMode(8, OUTPUT);
digitalWrite(8, HIGH);
```

analogRead: Lê o valor de um dos 6 pinos analógicos do Arduino e retorna um valor inteiro entre 0 e 1023, representando assim um valor de tensão analógica entre 0 e 5 volts. Abaixo temos uma configuração para uma variável a qual denominamos "sensor".

int sensor = analogRead(A0);

Neste caso, a variável sensor guarda um valor para um inteiro convertido entre 0 e 1023 da tensão analógica do pino A0.

OBS: Lembrar que os pinos analógicos do Arduino são identificados de A0 até A5 (o que equivale aos pinos 14 até 19). Desta forma a variável "sensor" pode também ser assim escrita:

int sensor = analogRead(14);

analogWrite: O Arduino que tem como microcontrolador o ATmega328 (caso do Arduino UNO R3), possui 6 pinos (3,5,6,9,10,11) que também podem gerar saídas com PWM (Pulse Width Modulation).

PWM é uma técnica para gerar tensões analógicas a partir de sinais digitais através da função *analogWrite*, que gera uma onda quadrada de frequência fixa no pino analógico previamente especificado.

Quando for definido um valor digital entre 0 a 255 (valor máximo que pode ser atribuído é 255), será gerada uma tensão de 0 a 5 volts.

Assim, na função abaixo exemplificada, será gerada uma tensão analógica de aproximadamente 2,5 volts no pino 10 do Arduino.

analogWrite(10,128);

delay: interrompe o programa que está sendo executado por um período de tempo, que é especificado em milissegundos (ms). Assim, para um delay de 1 segundo escreve-se 1000.

delay(1000);

millis: Não usa nenhum parâmetro, como no caso do "delay".

Como retorna somente o número de milissegundos desde que o programa foi iniciado e não suspende as atividades, é uma função muito usada para medir o tempo entre eventos.

Por armazenar períodos de tempo, requer muita memória e por isso é uma função do tipo *long.*

long timer = millis();

random: A função random retorna um valor aleatório entre mínimo (min) e máximo (max) passados como parâmetro.

int numRand = random(50,100);

Neste caso, *numRand* vai armazenar um número inteiro entre 50 e 100.

Serial.begin: Todo o Arduino tem no mínimo uma porta de comunicação serial do tipo RS-232 ou USB, utilizando a USART interna do microcontrolador.

No Arduino UNO R3 esses pinos são 0 e 1, RX (recepção) e TX (transmissão) respectivamente.

Assim, abre-se um canal de comunicação entre o Arduino e o PC, cuja taxa de transferência em bits por segundo (baud) pode variar entre 300 e 115200.

O valor default (padrão) é 9600bps.

Como esses pinos são exclusivos para comunicação serial, raramente são usados para entrada ou controle de dados externos.

Serial.println: Transfere caracteres ASCII do Arduino para o PC. Mais adiante veremos como monitorar essa transmissão de dados.

COMANDOS:

if: É um comando de seleção ou condicional. Serve para testar uma expressão para determinar se o seu resultado é falso ou verdadeiro.

Verdadeiro é sempre um resultado diferente de zero, mesmo que negativo. Falso é sempre igual a 0.

Com base nesse teste, *if* executa os comandos entre as chaves somente se este for verdadeiro.

if (expressão) {
bloco de comandos 1;}

else

{bloco de comandos 2;}

Desta forma, se (if) o que está entre as chaves for verdadeiro, ocorrerá a execução, senão (else) não ocorrerá a execução.

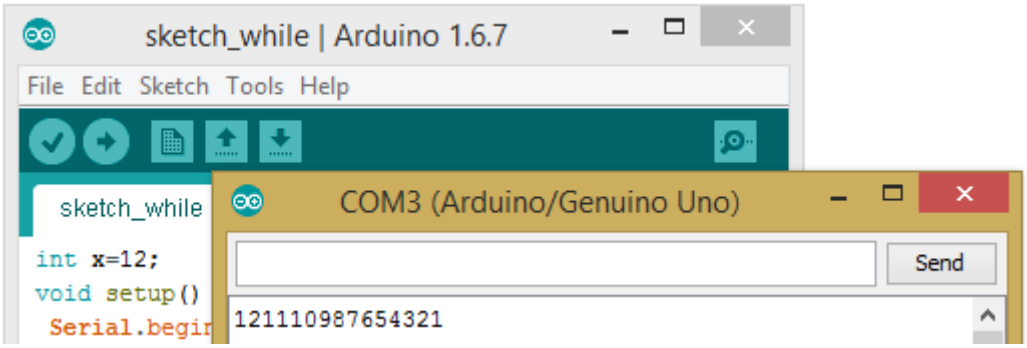
while: O comando *while* vai executar o bloco de comandos entre as chaves, quando a condição for verdadeira ou diferente de zero.

While (condição) {
Bloco de comandos;}

Vamos testar esse comando no Arduino, digitando no seu *sketch* o código a seguir:

```
int x=12; //ao “x” é atribuído o valor 12
void setup(){
 Serial.begin(9600); //a porta serial é configurada em 9600 bauds
}
void loop(){
 while(x<=12&&x>0) //verifica se” x” está entre os valores 0 e 12{
  Serial.print(x); //se for verdadeiro, transmite o valor de “ x”
  delay(1000); //pausa de 1 segundo para transmitir cada valor entre 0 e 12
  x--; //decrementa “x” e retorna ao teste;  se “x” for maior do que 12 (x>12), o teste para.
 }
}
```

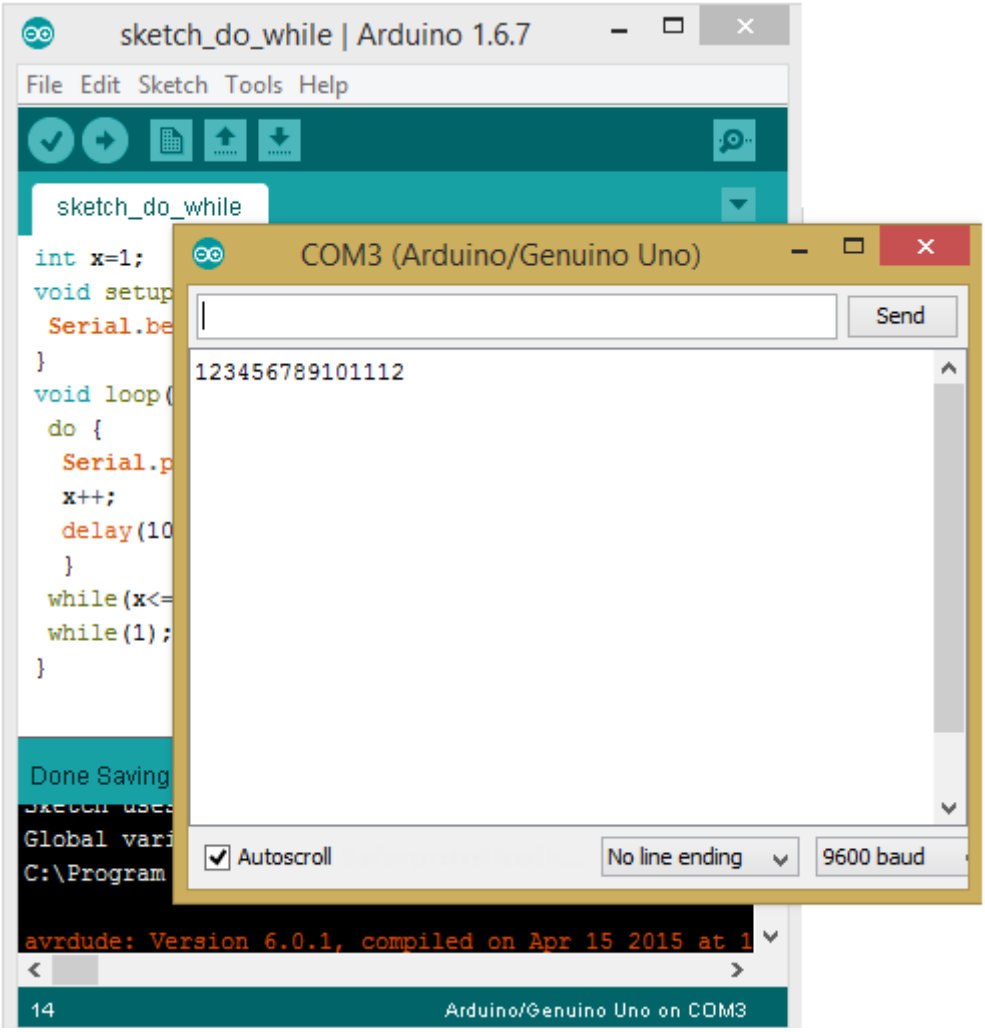
Podemos visualizar toda a operação clicando no botão Serial Monitor.



Do...while: similar ao comando anterior, porém a condição é testada no final do bloco de comandos e não no início.

No exemplo a seguir veremos que haverá agora um incremento do valor atribuído ao "x".

```
int x = 1;
void setup() {
 Serial.begin(9600);
}
void loop() {
 do {
  Serial.println(x);
  x++; // incrementa "x"
  delay(1000);
 }
  while(x<=12); //quando x<12, executa o comando e se x=12,o programa trava
  while(1);{ }
 }
```



for: É usado para repetir um bloco de comando certa quantidade de vezes determinada pelo programa. Veja um exemplo de sintaxe:

for (int x=0; x<3; x++) *int x* é a inicialização; *x<3* é a condição; *x++* é o incremento. Neste caso o led acenderá e apagará 3 vezes.

```
{
  digitalWrite(ledPin, HIGH);
  delay(150);
  digitalWrite(ledPin, LOW);
  delay(100);
 }
```

OPERADORES: No Arduino os operadores são classificados em quatro grupos distintos:

I – Aritméticos: retornam o resultado de uma operação aritmética com dois operandos. Os operandos são:

Soma (+) – exemplo: a = x + 2
Subtração (-) – exemplo: b = y-3
Produto (*) – exemplo: i = j*6
Divisão (/) – exemplo: y = k/5

II – Lógicos: comparam duas expressões e retornam 1 ou 0 (verdadeiro ou falso)

AND (&&) – exemplo: x>0 && y<5; (retorna 1 se a expressão for verdadeira)
OR (||) – exemplo: x>0 || y3; (retorna 1 se a expressão for verdadeira)
NOT (!) – exemplo: ! x > 0; (retorna 1 se a expressão for falsa)

III – Comparação: comparam duas variáveis ou constantes e retornam 1 ou zero (verdadeiro ou falso)

Comparação (==) – exemplo: x == y (retorna 1 se x for igual a y)
Diferença (!=) – exemplo: x != y (retorna 1 se x for diferente de y)
Menor que (<) – exemplo: x<y (retorna 1 se x for menor que y)
Maior que (>) – exemplo: x>y (retorna 1 se x for maior que y)
Menor ou igual (<=) – exemplo: x<=y (retorna 1 se x for menor ou igual a y)
Maior ou igual (>=) – exemplo: x>=y (retorna 1 se x for maior ou igual a y)

IV – Compostos: comparam um operador aritmético com um operador de atribuição:

Soma (+=) – exemplo: x += y (equivale a $x = x+y$)
Subtração (-=) – exemplo: x -= y (equivale a $x = x-y$)
Produto (*=) – exemplo: x *= y (equivale a $x = x*y$)
Divisão (/=) – exemplo: x /= y (equivale a $x = x/y$)
Incremento (++) – exemplo: x++ (equivale a $x = x+1$)
Decremento (--) – exemplo x--(equivale a $x = x-1$)

TENSÕES DIGITAIS E ANALÓGICAS NO ARDUINO

Como visto anteriormente, os pinos do Arduino podem ser configurados como entrada ou saída, operando como analógicos ou digitais.

Como o Arduino trata os sinais analógicos e digitais?

Vamos dar ênfase aos pinos PWM que são: 3, 5, 6, 9, 10 e 11 os quais podem também ser configurados como entrada e saída digitais, no entanto o inverso não vale para os pinos digitais, que não podem ser configurados como PWM.

As entradas analógicas podem distinguir qual é a tensão presente no pino, em outras palavras, permitem dividir um valor de 0-5V em 1024 níveis, neste caso, 1024 bits (0 a 1023) com base em um conversor AD de 10 bits.

Assim, o valor mínimo corresponde a 0V e o valor máximo a 4,955V (arredondando para 5V).

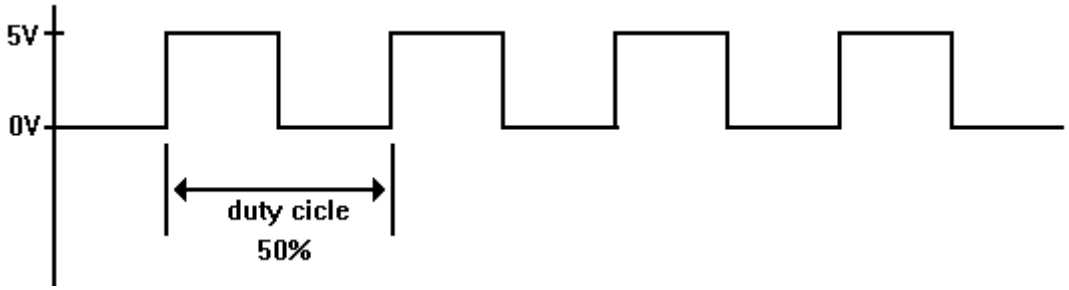
Assim, uma tensão de 2,5V vai permitir que uma variável guarde um valor inteiro igual a 512. Por outro lado as entradas digitais distinguem apenas dois níveis: 0 ou 5V.

Nas saídas digitais então, com PWM (Pulse Width Modulation), podemos controlar a tensão que sai das portas através de uma onda quadrada, em que somente a largura dos pulsos é alterada, mantendo fixa a frequência (*duty cycle*).

Em outras palavras, podemos controlar a tensão de saída nesses pinos, ou seja, gerar uma tensão analógica a partir de uma onda quadrada.

A figura a seguir mostra uma onda quadrada com *duty cycle* = 50%, correspondendo ao código *analog Write (127).*

Duty cycle de 50% significa que durante um certo período T, o tempo em que a onda quadrada permanece em 0V e em 5V é exatamente igual.

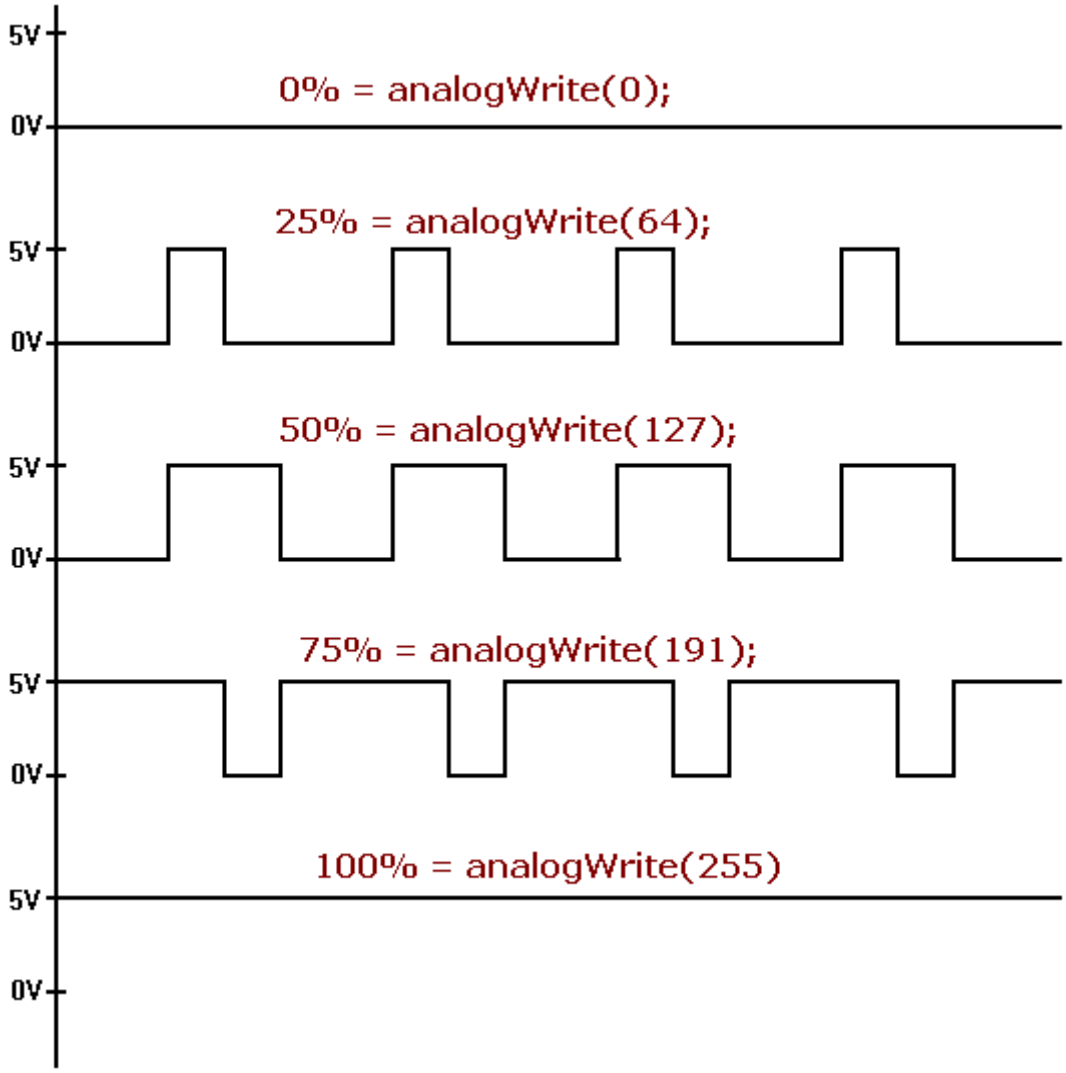


Desta forma essa saída PWM permite controlar rotação de motores, sensores, brilhos de leds, etc.

O valor obtido nas saídas PWM é o resultado da relação 1024/256 = 4, isto é, o máximo valor obtido na saída refere-se a 1024/4 o que nos dá um escopo de 0 a 255.

É bom lembrar que valores acima de 255 não serão reconhecidos por dispositivos associados aos pinos PWM.

A figura a seguir mostra a correspondência entre *analogWrite* e *duty cycle*, para 0%, 25%, 50%, 75% e 100%.



CONCLUINDO:

analogWrite(0 a 255);

digitalWrite(0 ou 1);