# Introdução à engenharia

## Engenharia de computação e software

Guilherme Augusto Barucke Marcondes - guilherme@inatel.br
Luan Patrick do Couto Siqueira – luan.siqueira@gec.inatel.br

# Capitulo 01

**Conceitos iniciais**

## 1.1 Introdução

Nesta disciplina iremos estudar os conceitos básicos de um curso de engenharia, mais especificamente engenharia de computação e de software, tais como:

- O que são microcontroladores?
- Introdução à programação embarcada;
- Desenvolvimento de raciocínio lógico para resolução de problemas;
- Desenvolvimento de ideias para projetos;
- Documentação de projetos.

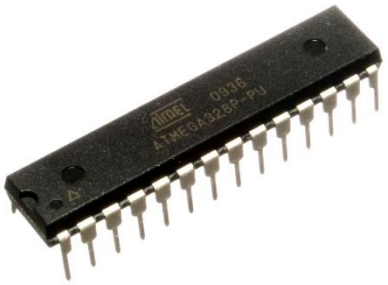
## 1.2 Os microcontroladores

Microcontroladores são pequenos computadores que já possuem memória, um núcleo de processamento e GPIO (entradas e saídas de periféricos) em um único circuito integrado.

São principalmente utilizados em aplicações embarcadas, nas quais é preciso uma comunicação com algum *hardware*.

O microcontrolador é o "cérebro" dos nossos projetos. Ele é a parte intermediária entre captar informações e executar as ações de acordo com a aplicação desejada. Essas duas tarefas são chamadas de entrada e saída, respectivamente, e são usados para isso, na prática, **sensores** e **atuadores**, seguindo um fluxo, como na figura a seguir:

Nesse curso iremos utilizar um microcontrolador em específico, o **ATMEL ATMEGA328**.




Para fins didáticos utilizaremos uma plataforma para facilitar o nosso aprendizado. Essa plataforma é simplesmente uma placa na qual poderemos fazer a comunicação do microcontrolador com uma interface mais "amigável" e uma programação de alto nível (sintaxe mais próxima dos idiomas). A plataforma será o **Arduino.**

## 1.3 O Arduino e os algoritmos

O Arduino é uma placa lançada em 2005 com o intuito de tornar mais acessível e simplificada a prototipagem de projetos eletrônicos, deixando mais fácil a conexão de sensores (entradas) e atuadores (saídas). Ele utiliza a linguagem Wiring, a qual se deriva das linguagens C e C++.


Como toda linguagem de programação, devemos sempre utilizar o raciocínio lógico para desenvolver os nossos códigos, quando estamos os escrevendo, nós estamos desenvolvendo uma rotina para que o microcontrolador possa executar as atividades da maneira e na ordem certa. A essa rotina chamamos de **Algoritmo**.

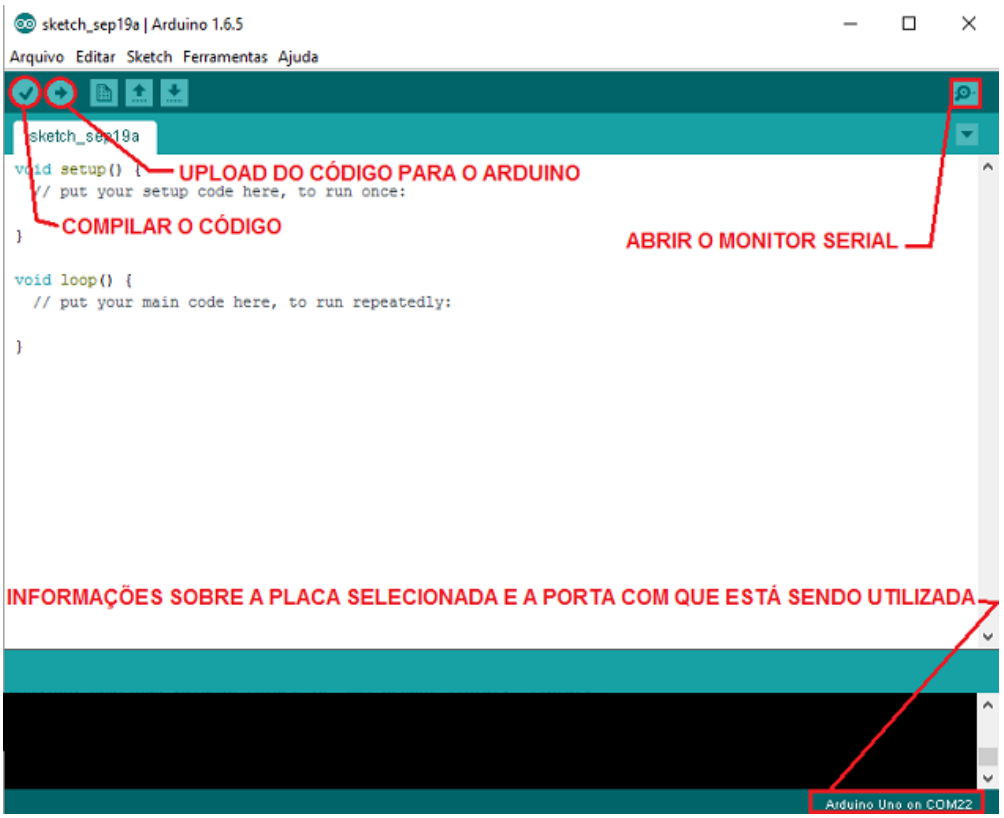
Sem percebermos estamos, no dia a dia, seguindo uma rotina para todas as nossas atividades. Vejamos um exemplo simples, como olhar as horas no celular:

```
1  void lerHoras()
2  {
3      Pegue seu celular;
4      Ligue a tela;
5      Leia qual é a hora;
6      Desligue a tela;
7      Guarde o celular;
8  }
```

Podemos ver que, em uma simples ação, nós executamos uma rotina (algoritmo). É como nos projetos com Arduino. Temos que fazer tudo em ordem, senão, não funcionará e nem fará muito sentido.

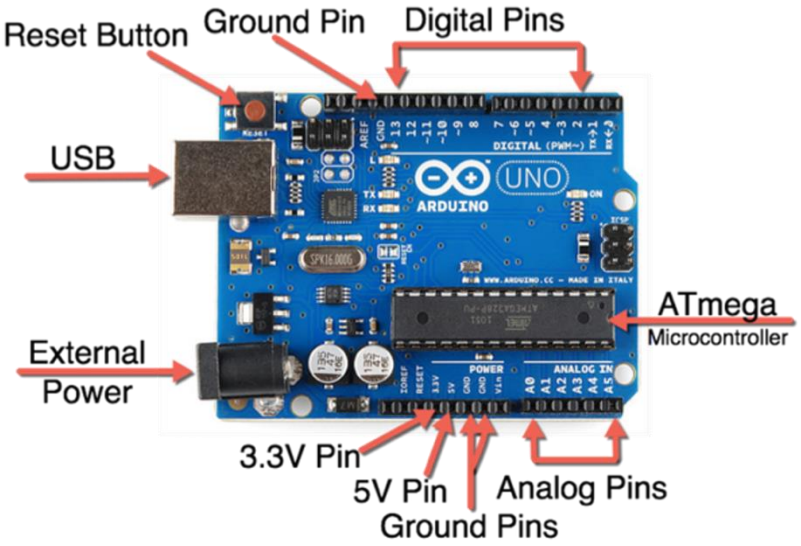
## 1.4 A IDE do Arduino

O IDE (*Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado) é basicamente a interface, o programa no computador (PC), por meio do qual iremos escrever, compilar e carregar o código para a placa. Ele também possui outras funções (como o monitor serial), que serão abordadas no curso.



## 1.5 Conhecendo o hardware do Arduino UNO

Vamos conhecer um pouco mais sobre o hardware do Arduino UNO, listando as principais partes.




- USB – Conector no qual plugamos o PC para o carregamento do código, também pode ser usado para energizar a placa;
- Fonte externa – Conector no qual podemos ligar uma fonte de 9V a 12V, para que não seja necessário a placa ficar plugada ao PC após o carregamento do programa.
- 3.3V e 5V – São os pinos de alimentação da placa, podem ser usados tamém como saída de tensão;
- GND's – São os pinos de terra da placa;
- Pinos analógicos – São usados como entradas analógicas ou saídas digitais;
- Microcontrolador – É o núcleo de processamento da placa;
- Pinos digitais – São usados como entradas ou saídas digitais;
- Botão de reset – É utilizado para reiniciar a placa, é equivalente a desligar e ligar a energia do Arduino.

**OBS:** Para saídas "analógicas" (Ex: controle de luminosidade de LED's) utilizaremos os pinos de PWM, estudaremos mais a frente.

## 1.6 Os pimeiros códigos

### 1.6.1 Void Setup e Loop

Na plataforma do Arduino, duas funções que são a base do nosso código, são elas:

#### 1.6.1.1 void setup()

No void setup escrevemos as configurações gerais do nosso código, como definir modos de pinos (entrada ou saída), inicializar variáveis e outros parâmetros, o void setup é executado apenas uma vez quando a placa é ligada.

#### 1.6.1.2 void loop()

No void loop escrevemos o nosso código em si, todos os comandos que nós precisamos que aconteça no nosso projeto, essa função começa a se repetir após o setup e é parada somente se resetarmos ou tirarmos a energia da placa.

```
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
```

### 1.6.2 pinMode e digitalWrite

#### 1.6.2.1 pinMode()

Para montarmos nossos projetos, precisamos definir o que são as entradas (sensores, botões) e saídas (LEDs, motores) do nosso sistema. Depois desta definição, é necessário configurar no código esses parâmetros, como já visto, as configurações são feitas no void setup().

A função para essa configuração se chama pinMode() e é escrita da seguinte forma: **pinMode(pino, MODO)**

- Pino – É o número do pino onde está conectado o nosso periférico;
- MODO – Aqui é declarado se o pino é um dispositivo de entrada ou saída, **INPUT** ou **OUTPUT**, respectivamente.

Vejamos um exemplo:

```
void setup()
{
  pinMode(6, INPUT); // Configuração do pino 6 como entrada
  pinMode(5, OUTPUT); // Configuração do pino 5 como saída
}
```

### 1.6.2.2 digitalWrite()

Para nós acionarmos um simples LED precisamos escrever uma função para que o pino receba energia, certo?

Essa função é o **digitalWrite(pino, ESTADO)**, na qual:

- Pino – É o número do pino onde está conectado o nosso periférico;
- ESTADO – Aqui é colocado se queremos ligar ou desligar o pino, **HIGH** ou **LOW**, respectivamente.

Vejamos um exemplo:

```
void setup()
{
  pinMode(5, OUTPUT); // Configuração do pino 5 como saída
}

void loop()
{
  digitalWrite(5, HIGH); // Liga o pino 5
  digitalWrite(5, LOW); // Desliga o pino 5
}
```

## 1.6.3 delay()

No item **1.7.1.2** é apresentada a função **void loop()**, que se repete até que desligamos o Arduino. Seguindo essa lógica e observando a imagem anterior, na qual tratamos de **digitalWrite()** se o nosso **void loop()** fica se repetindo, o nosso LED alternará seu estado entre ligado e desligado e, assim, ficará piscando, certo? **Errado!**

O **void loop()** se repete tão rápido que o olho humano não consegue perceber o LED piscar, mas como podemos resolver isso?

Com a função delay() nós pausamos o programa por um determinado perídodo e assim, nesse exemplo, conseguiríamos ver o LED piscar.

Para usarmos a função delay, usamos a seguinte sintaxe: **delay(tempo)**, em que:

tempo – É o período em **ms** que pausamos a execução do código, vejamos um exemplo:

```
1  void setup()
2  {
3    pinMode(5, OUTPUT); // Configuração do pino 5 como saída
4  }
5
6  void loop()
7  {
8    digitalWrite(5, HIGH); // Liga o pino 5
9    delay(1000); // Pausamos o código por 1 segundo
10   digitalWrite(5, LOW); // Desliga o pino 5
11   delay(1000); // Pausamos o código por 1 segundo
12 }
```

Agora sim veríamos o LED piscar, no caso, a cada 1 segundo ele trocaria o seu estado.

### 1.6.4 Alguns detalhes na escrita do código

Observando os códigos dados como exemplos, podemos observar dois detalhes que não foram mencionados ainda. São eles:

- Comentários (//) – Para ficar mais simples a interpretação do código, muitas vezes usamos os comentários para complementar informações relevantes no código (Ex: Fazer um comentário sobre o que determinada linha está fazendo). Sempre usamos duas barras "//" e depois o comentário desejado.

  **OBS:** Os comentários não são compilados para a execução, são apenas algo visual no IDE.

- Ponto e vírgula - Depois das linhas de todas as linhas de **comando** colocamos o ";" para sinalizar ao Arduino que aquela linha acaba naquele ponto.

```
6  void loop()
7  {
8    digitalWrite(5, HIGH); // Isso aqui é um comentário
9    delay(1000) // a IDE mostrará erro se compilarmos o código, pois não colocamos ";" nesse comando
10               // Isso aqui também é um comentário
11 }
```

### 1.6.5 digitalRead()

O **digitalRead()** é o oposto do **digitalWrite()**, com ele nós vamos **ler** alguma entrada, se nela tiver sinal ou não, que nos retornará informação correspondente. Vamos supor que no nosso pino 6 haja um botão conectado e se o apertarmos o Arduino recebe essa informação. Para escrevermos, colocamos: **digitalRead(PINO)**, em que PINO é a porta na qual queremos coletar a informação, no caso, é a porta 6.

```
1 void setup()
2 {
3   pinMode(6, INPUT); // Configuração do pino 6 como entrada
4 }
5
6 void loop()
7 {
8   digitalRead(6); // Lendo o pino 6
9 }
```

Porém, o digitalRead() não faz nada sozinho, veremos no próximo tópico como usá-lo de forma útil.

## 1.6.6 if/else

### 1.6.6.1 if()

Para verificarmos algo no nosso código (Ex: comparar valor de variável, receber informações de sensores ou botões, etc) utilizamos o comando **if()**. O bloco do if só será executado se a verificação for verdadeira. Vejamos um exemplo simples, o LED, conectado no pino 5, só será aceso **se** for apertado o botão que se encontra no pino 6.

```
1  void setup()
2  {
3    pinMode(6, INPUT); // Configuração do pino 6 como entrada
4    pinMode(5, OUTPUT); // Configuração do pino 5 como saída
5    digitalWrite(5, LOW); // Inicializamos o LED como desligado
6  }
7
8  void loop()
9  {
10   if (digitalRead(6) == HIGH) // Verificando a leitura do pino 6
11   {
12     digitalWrite(5, HIGH); // Acendemos o LED apenas SE o pino 6 receber energia do botão
13   }
14 }
```

### 1.6.6.2 else()

O comando **else()** é bloco executado somente se o **if** anterior não for verdadeiro. No nosso caso, o **else()** será executado somente se o nosso botão não for apertado. Vamos considerar que se não apertarmos o botão, o LED irá piscar a cada 1 segundo.

```
1  void setup()
2  {
3    pinMode(6, INPUT); // Configuração do pino 6 como entrada
4    pinMode(5, OUTPUT); // Configuração do pino 5 como saída
5    digitalWrite(5, LOW); // Inicializamos o LED como desligado
6  }
7
8  void loop()
9  {
10   if (digitalRead(6) == HIGH) // SE receber informação do botão, executamos:
11   {
12     digitalWrite(5, HIGH); // Acendemos o LED apenas SE o pino 6 receber energia do botão
13   }
14
15   else // SENÃO, executamos:
16   {
17      digitalWrite(5, HIGH); // Acendemos o LED
18      delay(1000); // Esperamos 1 segundo
19      digitalWrite(5, LOW); // Apagamos o LED
20      delay(1000);// Esperamos 1 segundo
21   }
22 }
```

Fazendo um apanhado sobre if/else, com pseudocódigo:

```
SE (CONDIÇÃO == VERDADEIRA)
{
        EXECUTE COMANDO_1
}
SENÃO
{
        EXECUTE COMANDO_2
}
```

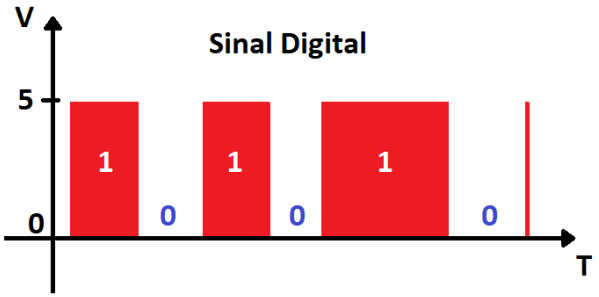
# Capítulo 02

## 2.1 Sinais Digitais & Analógicos

O Arduino UNO (modelo mostrado anteriormente) possui 14 portas digitais e 6 portas analógicas, mas o que siginifica isso?

### 2.1.1 Sinais Digitais

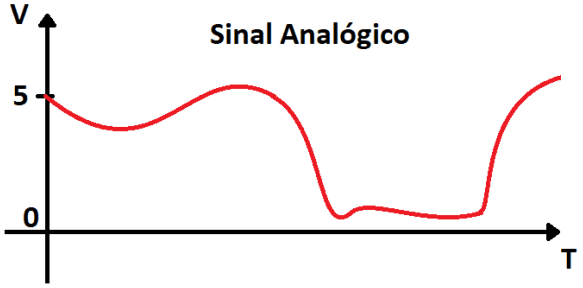
Os sinais digitais são caracterizados por serem discretos, ou seja, eles possuem valores finitos. No curso, vamos assumir que eles possuem apenas dois valores: "**0**" ou "**1**" que podem ser chamados de **nível lógico baixo** e **nível lógico alto**, respectivamente. No Arduino, o nível lógico baixo é representado com uma tensão próxima de **0 volts** tanto na saída, quanto na entrada. O nível lógico alto é representado com uma tensão próxima de **5 volts** também tanto na saída, como na entrada.




Um bom exemplo para lembrarmos dos sinais digitais é observarmos as lâmpadas das nossas casas acionadas por um interruptor, a lâmpada estará ligada ou desligada, não há possibilidade de a lâmpada ficar "meio apagada" ou "meio acesa"

### 2.1.2 Sinais Analógicos

Os sinais analógicos são caracterizados por terem infinitos valores, ou seja, podemos ajustar qualquer valor entre o máximo e o mínimo.



Um bom exemplo para lembrarmos dos sinais analógicos é observarmos os medidores de temperatura. Eles marcam números reais (com vírgula) como 27,8°C e 24,6°C. Se os termômetros marcassem apenas números inteiros (27°C e 24°C) seriam muito imprecisos, pois os valores à direita da vírgula fazem diferença. Se marcassem apenas dois valores como nos sinais digitais, esse sistema de medição não faria muito sentido.

## 2.2 Conversor A/D

Na tecnologia embarcada, é muito comum encontrarmos aplicações que utilizam o conversor analógico-digital, mas qual é a sua funcionalidade?

Vamos supor que estamos montando um projeto cuja finalidade é medir a distância entre o sensor e um objeto qualquer. Queremos medir com precisão essa distância e não somente se há ou não um objeto.

O conversor A/D é capaz de fornecer um valor proporcional de acordo com a tensão de entrada.

### 2.2.1 Resolução

O Arduino UNO possui uma resolução de entrada analógica de 10 bits, isso significa que a sua faixa de conversão é de 0 – 1023 ($2^n$, onde *n* é a resolução). Ou seja, possuímos 1024 valores para mapearmos as nossas entradas, vejamos um exemplo concreto para entendermos melhor:

**Exemplo:** Na entrada A0 há uma tensão de 2,5v, precisamos convertê-la para um valor digital para tratarmos no código. Vamos utilizar a seguinte equação:

$$V_{digital} = \frac{V_{entrada} * 1023}{V_{ref}}$$

**$V_{digital}$** = É um valor entre 0 – 1023 proporcional à tensão de entrada;
**$V_{entrada}$** = É a tensão na entrada analógica (saída de um sensor);
**$V_{ref}$** = É a tensão de referência do Arduino, por padrão é 5v.

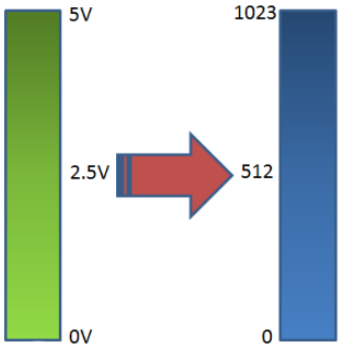
Resolvendo o nosso exemplo, vamos substituir os valores:

$$V_{digital} = \frac{2{,}5 * 1023}{5}$$

$$V_{digital} = 511{,}5$$

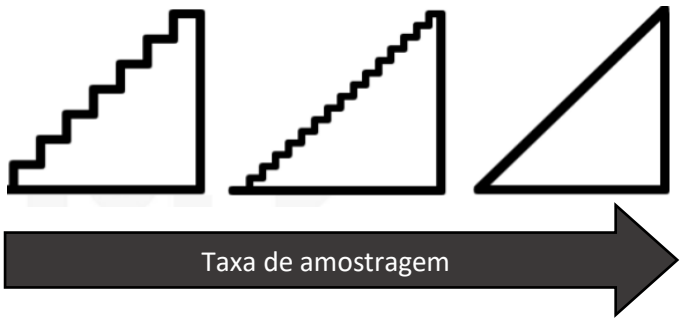
Podemos observar que, assim como **2,5v** é a metade da referência de tensão (**5v**), o valor **511,5** também é a metade da faixa do A/D (**1023**).

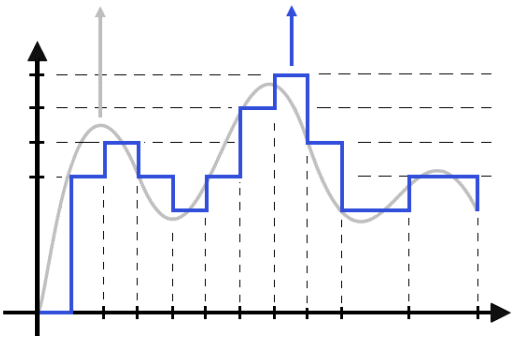
Assim comprovamos a proporcionalidade dos valores!




## 2.2.2 Taxa de amostragem

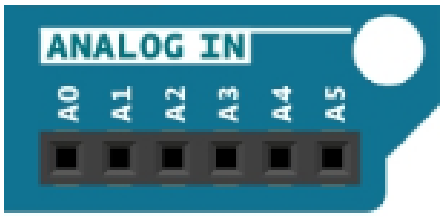
A taxa amostragem está diretamente ligada com a precisão da leitura. Quanto maior ela for, maior a resolução e assim os *gaps* de leitura (intervalo onde não é possível ler variação) são menores.



## 2.3 analogRead()

A função analogRead é utilizada para a leitura das seis portas analógicas (A0 até A5).




Como já visto anteriormente a resolução de entrada do Arduino é de 10 bits (0 - 1023). Aplicando uma tensão de 0v a 5v, por exemplo, no pino **A0**, utilizando a função **analogRead(PINO)** e atribuindo a uma variável, teremos:

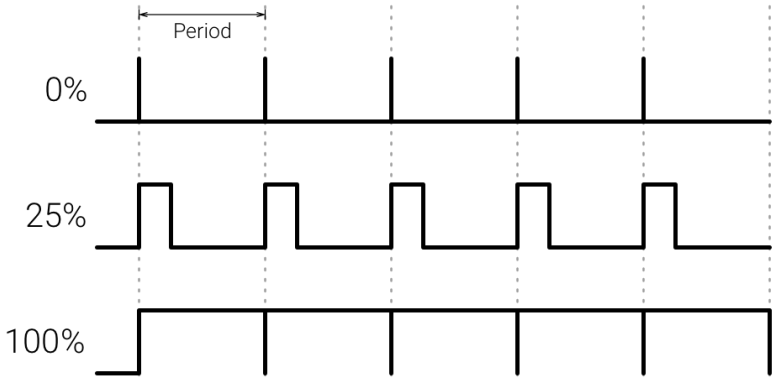
```
4 void loop()
5 {
6   int leitura; // Declarando uma variável do tipo inteira
7   leitura = analogRead(A0); // Atribuindo o valor da leitura
8 }
```

Se retornarmos o valor dessa variável, poderemos ver o conversor A/D atuando e adequando o valor de tensão de entrada para um valor digital, mantendo sempre a proporcionalidade.

## 2.4 PWM e o analogWrite()

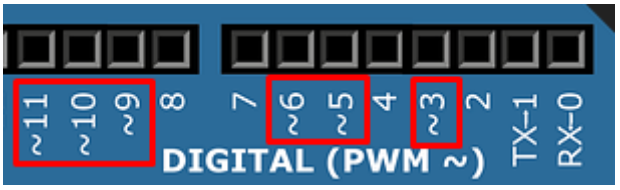
Até aqui aprendemos a como ligar e desligar saídas digitais e fazer leituras de entradas analógicas. Agora veremos como fazer uma saída analógica por meios digitais utilizando uma técnica chamada PWM (Pulse Width Modulation – Modulação por Largura de Pulso).

Essa técnica consiste na geração de uma onda quadrada em uma frequência muita alta em que pode ser controlada a porcentagem do tempo em que a onda permanece em nível lógico alto. Esse tempo é chamado de Duty Cycle e sua alteração provoca mudança no valor médio da onda, indo desde 0V (0% de Duty Cycle) a 5V (100% de Duty Cycle).




Sabendo como funcionam os parâmetros do PWM conseguimos perceber a sua utilidade. Basicamente ele serve para controle de nível de tensão na saída, não mais **0v** ou **5v**, agora podemos configurar qualquer tensão nessa faixa (Ex: Controlar a intensidade de um LED).

Os pinos PWM do Arduino UNO podem ser identificados através do símbolo ( ~ ) junto ao número do pino.




Já vimos que a resolução de entrada é de 10 bits. Vamos considerar os mesmos conceitos já vistos, agora para a saída. Porém, a resolução de saída do Arduino UNO é de 8 bits e sua faixa é de 0 – 255 ($2^n$, onde $n$ é a resolução).

Mas e na prática, como vamos aplicar o PWM? Com a função **analogWrite()**!

Essa função é muito simples de ser usada, precisamos passar dois parâmetros para ela.

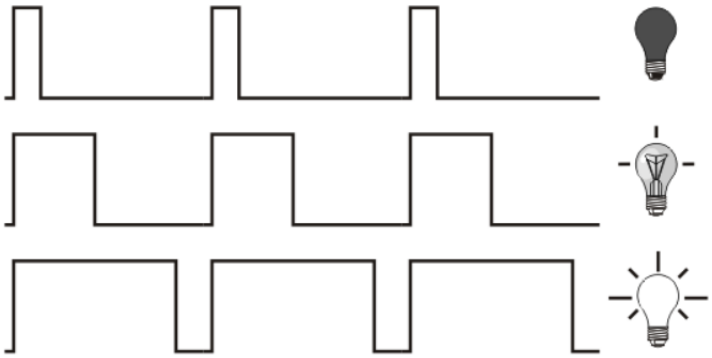
**analogWrite(PINO, VALOR)**, onde:

- PINO: Pino PWM a ser utilizado;
- VALOR: Saída que esteja na faixa de 0 – 255.

Vejamos um exemplo:

```
6  void loop() {
7    analogWrite(3, 0); // LED do pino 3 desligado
8    delay(500); // Pausa de 500ms
9    analogWrite(3, 127); // LED no pino 3 com 50% de luminosidade (50% de 255)
10   delay(500); // Pausa de 500ms
11   analogWrite(3, 191); // LED no pino 3 com 75% de luminosidade(75% de 255)
12   delay(500); // Pausa de 500ms
13 }
```

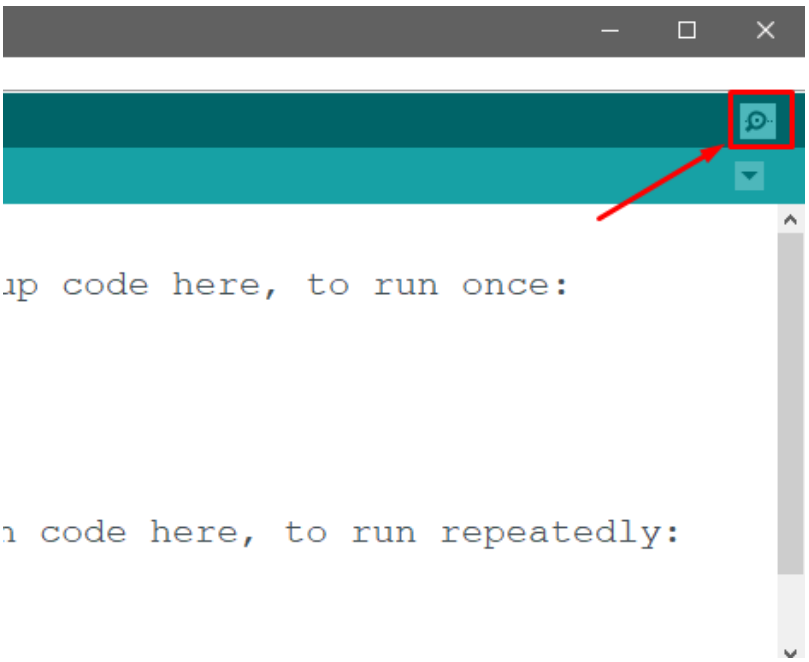
Resultado:



## 2.5 O monitor serial

O monitor serial é a ponte entre o PC e o Arduino. Com ele podemos enviar e receber informações na forma de texto, útil para depuração e também para controlar o Arduino pelo teclado do PC.

Para acessar o monitor serial corretamente devemos, primeiramente, deve conectar a placa, fazer upload do seu código e então clicar no símbolo da lupa que se encontra no canto superior direito na tela do IDE.




Para que o monitor serial funcione, precisamos fazer a sua configuração e escrever as linhas de comandos que imprime as informações desejadas.

A primeira coisa que precisamos fazer é configurar, no void setup() a velocidade de comunicação, utilizando o comando:

**Serial.begin(VELOCIDADE);**

**VELOCIDADE** é a taxa de comunicação em bits por segundo. Iremos utilizar o valor **9600** para a maioria das aplicações, mas há vários valores comuns como 300, 2400, 4800, 9600, 19200, 57600 e 115200.

Para imprimirmos dados no monitor, precisamos usar as funções **Serial.print(DADO)** ou **Serial.println(DADO)**, a única diferença das duas funções é a quebra de linha. **Serial.println(DADO)** coloca uma informação e depois adiciona uma nova linha.

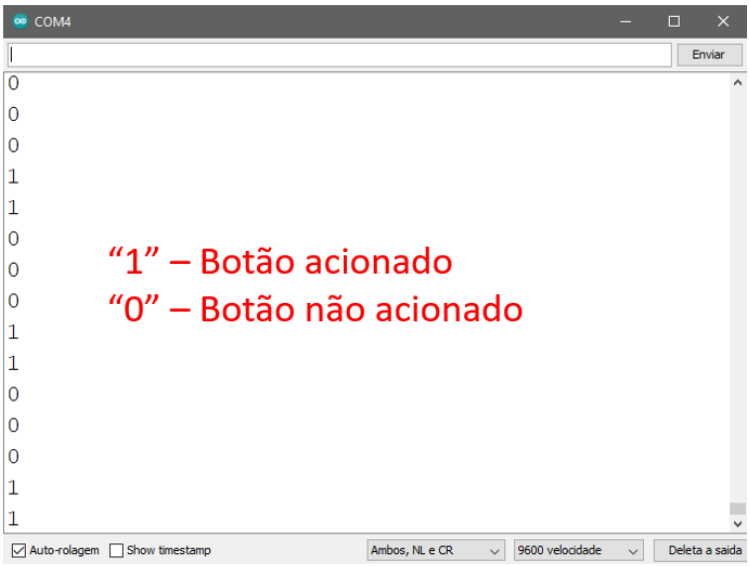
O **DADO** a ser mostrado pode ser uma variável, texto, leitura de pino, etc. Vejamos um exemplo do monitor serial mostrando a leitura de um pino.

**Código:**

```
void setup()
{
  // inicia a comunicação serial com 9600 bits por segundo:
  Serial.begin(9600);
  // declara o botão como pino de entrada:
  pinMode(3, INPUT);
}

void loop()
{
  // realiza a leitura do botão e guarda na variavel buttonState:
  int buttonState = digitalRead(3);
  // Imprime o valor guardado na variavel:
  Serial.println(buttonState);
  // delay entre as leituras para melhor estabilidade
  delay(1);
}
```

**Monitor serial:**



Também podemos utilizar o monitor serial como entrada de informações e comandos.

Vejamos um exemplo simples, a tecla **“l”** do teclado liga o LED e **“d”** desliga.

**OBS:** Devemos tomar cuidado com as letras maiúsculas e minúsculas, o arduino diferencia esses dois tipos de caractere.

```
1  void setup()
2  {
3    // inicia a comunicação serial com 9600 bits por segundo:
4    Serial.begin(9600);
5    // declara o pino 13 como saida
6    pinMode(13, OUTPUT);
7  }
8
9  void loop()
10 {
11   char comando;
12   if (Serial.available()) { // Verifica se há dados na serial
13     comando = Serial.read(); // Atribui o dado lida na variavel
14     if (comando == 'l') { // Verifica se é o comando "l"
15       digitalWrite(13, HIGH); // Liga o LED do pino 13
16     }
17     else if (comando == 'd') { // Verifica se é o comando "d"
18       digitalWrite(13, LOW); // desliga o LED do pino 13
19     }
20   }
21 }
```

O resultado é exatamente o que queríamos!