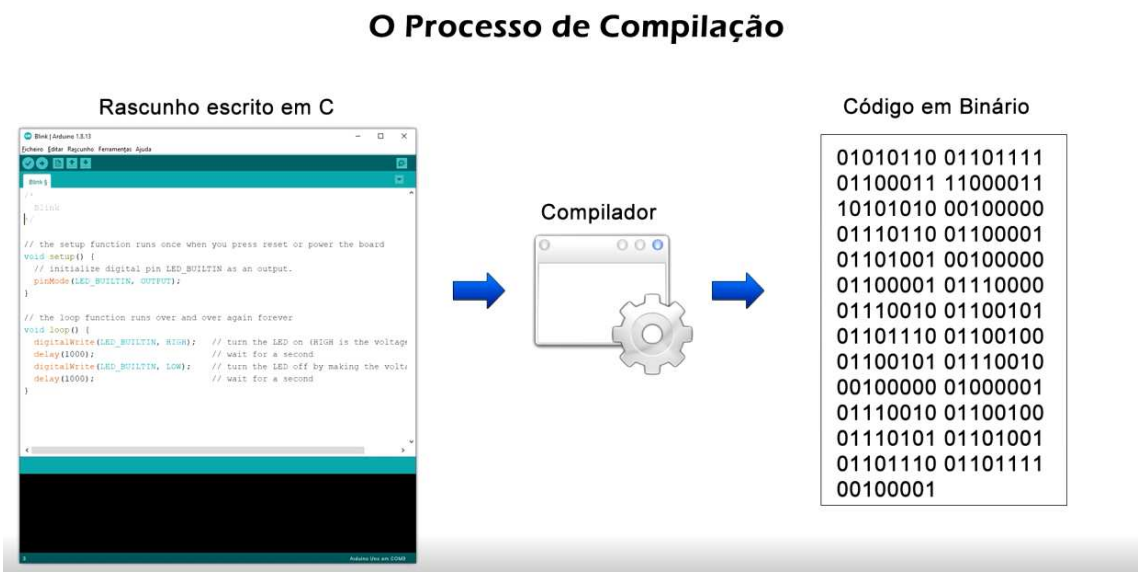
# Processo de Compilação

CELISTA

## Compilação

Todo rascunho, ou sketch do arduino, é escrito em C. Existem várias linguagens como PHP, Java, Pascal, C#, C++, e muitas outras, e o ambiente de desenvolvimento do Arduino utiliza a linguagem C como você já sabe. O problema é que a linguagem C é de nível médio, isto quer dizer que eu e você entendemos essa linguagem porque utilizamos palavras para programar, só que o computador utiliza linguagem de nível baixo, que é código de máquina.

O computador só entende 0 e 1 que é código binário e é aí que entra o compilador, ele faz a tradução do nosso código em C para código binário em linguagem de máquina para que o processador entenda o que deve ser feito:




## Mas o que é a Sintaxe?

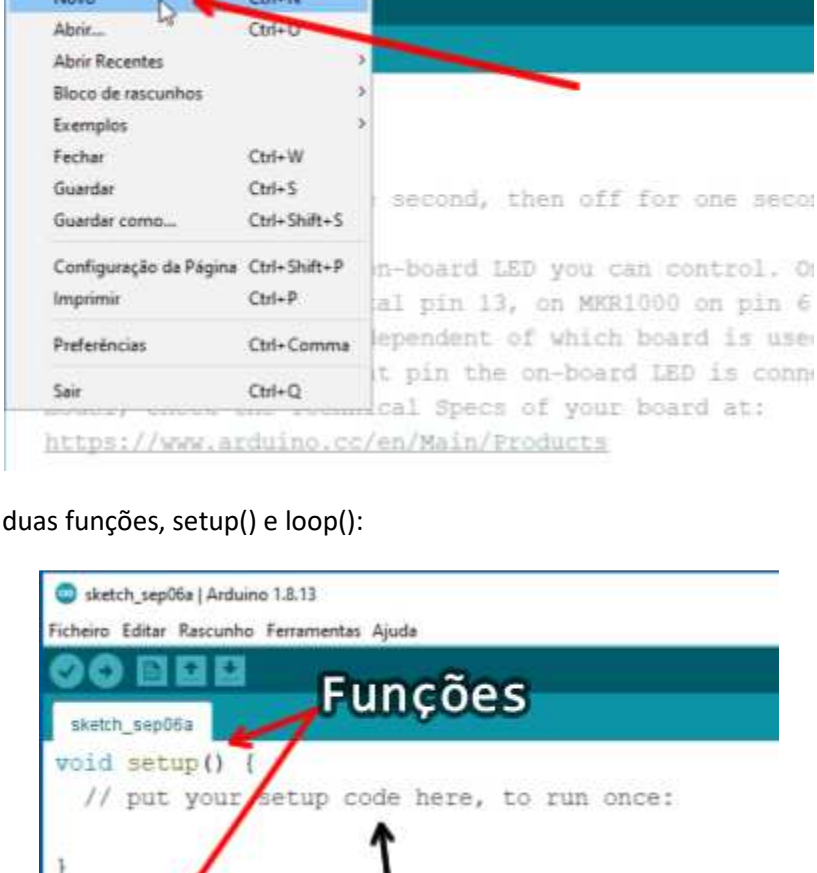
Assim como o Português, a linguagem C possui algumas regras que devemos obedecer ao escrever nossos códigos e a essa forma de se escrever damos o nome de sintaxe. Se você ocultar alguma palavra numa frase essa frase fica sem sentido e pode provocar um mau entendido entre a pessoa que está falando e a outra que está ouvindo.

Por exemplo na frase: O ar condicionado está sem gás.

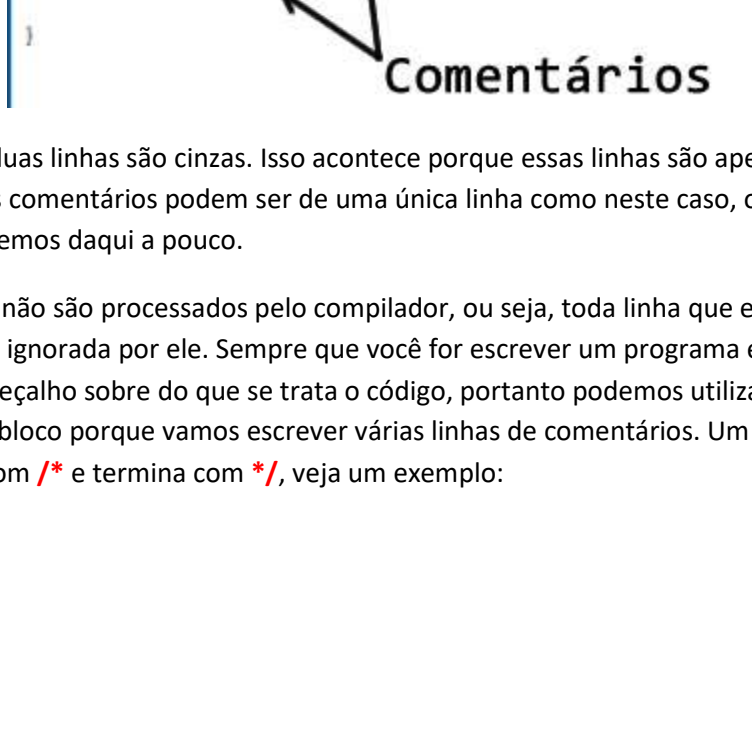
Se nós removermos a palavra SEM o cliente não vai entender o sentido da frase porque não está estruturada da forma adequada, o mesmo se aplica a linguagem C, se você esquecer de fechar um bloco de código ou escrever algum comando com uma letra errada o compilador não vai entender o seu código e vai gerar um erro como aconteceu na aula passada.

## Funções e Comentários

Você vai entender agora como funciona a lógica sequencial do nosso código. Vamos abrir a IDE e um novo rascunho, clique em Arquivo... novo..

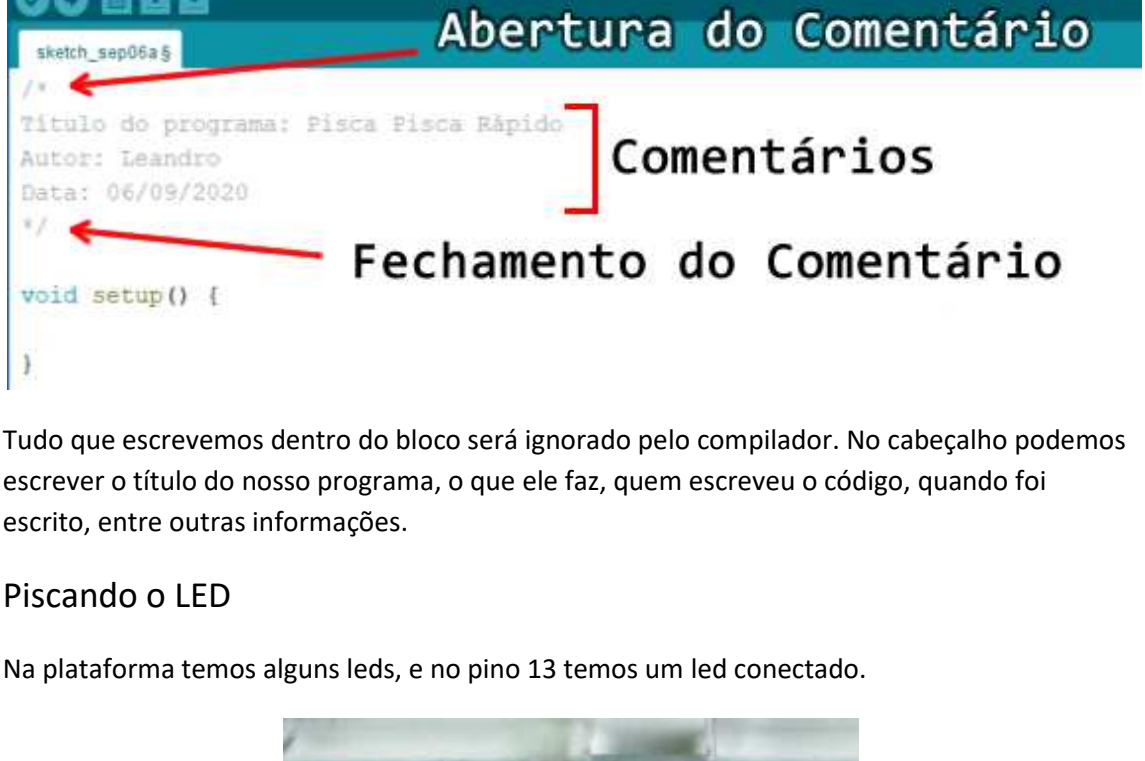



Temos então duas funções, setup() e loop():




Veja que estas duas linhas são cinzas. Isso acontece porque essas linhas são apenas comentários. Os comentários podem ser de uma única linha como neste caso, ou de múltiplas linhas como veremos daqui a pouco.

Os comentários não são processados pelo compilador, ou seja, toda linha que estiver comentada será ignorada por ele. Sempre que você for escrever um programa é interessante informar no cabeçalho sobre do que se trata o código, portanto podemos utilizar um comentário em bloco porque vamos escrever várias linhas de comentários. Um comentário em bloco começa com **/*** e termina com ***/**, veja um exemplo:




Tudo que escrevemos dentro do bloco será ignorado pelo compilador. No cabeçalho podemos escrever o título do nosso programa, o que ele faz, quem escreveu o código, quando foi escrito, entre outras informações.

## Piscando o LED

Na plataforma temos alguns leds, e no pino 13 temos um led conectado.


Vamos fazer esse led piscar. Dentro do SETUP vamos adicionar a função pinMode() que vai definir como vai se comportar o pino que vamos selecionar para piscar o led. Como o led está no pino 13 nós vamos dizer então pra função que vamos modificar o estado do pino 13 e que esse pino vai funcionar como saída, *output*:

```
/*
Titulo do programa: Pisca Pisca Rápido
Autor: Leandro
Data: 06/09/2020
*/

void setup() {
  // Aqui vou colocar meu codigo para ser executado uma unica vez
  pinMode(13,OUTPUT); // Define o pino 13 como saida

}
```

Lembre-se de que nosso objetivo é acender o led por isso a tensão deve saír do pino.

Mais uma informação importante, a linguagem C é case sensitive, isso que dizer que existe diferença entre você digitar pin**m**ode() e pin**M**ode(), veja que quando digitamos uma função ou comando corretamente a cor muda. No final de cada comando precisamos colocar um ponto e vírgula para dizer ao compilador que a instrução termina ali.

Quando começamos a programar em C é normal acontecer erros de compilação por esquecermos às vezes do ponto e vírgula.

Escrever comentários é uma boa prática em programação por isso coloque ao lado do comando pinMode(13,OUTPUT): *// Define o pino 13 como saída,* como ilustrado na imagem anterior. Você pode colocar um comentário tanto na linha anterior ao código como ao lado, você que define.

Conecte o cabo USB no Arduino e clique em Enviar para enviar o programa:




Como é a primeira vez que vamos compilar a IDE pede pra que a gente salve o rascunho, coloque um nome para o rascunho, pode ser *blink2* e clique salvar. Será realizada a compilação do programa e então o envio para a plataforma. Veja que o led permanece apagado, isso aconteceu porque o programa anterior que estava na plataforma foi apagado e por padrão quando definimos um pino como saída o nível lógico fica definido como baixo e como não temos mais nenhum comando na função loop o led permanece assim mesmo. Vamos continuar com a escrita do programa.

Agora dentro da função loop vamos adicionar mais uma função chamada digitalWrite();
Essa função vai escrever ou definir em uma porta digital um nível lógico que pode ser HIGH ou LOW, Alto e Baixo respectivamente. Como nós queremos acender o led vamos definir como HIGH mas a função precisa saber em qual pino vai ser definido esse nível lógico alto, e o pino como você já sabe é o 13. Então fica digitalWrite(13,HIGH) e finaliza com ponto e virgula:

```
void setup() {
  // Aqui vou colocar meu codigo para ser executado uma unica vez
  pinMode(13,OUTPUT); // Define o pino 13 como saida

}

void loop() {
  // Aqui vou colocar meu codigo principal que sera executado varias vezes repetidamente
  digitalWrite(13,HIGH);
}
```

Muito bem... agora queremos que o led apague não é verdade? Porque o objetivo é fazer ele piscar então na linha abaixo colocamos digitalWrite(13... só que agora é LOW porque vamos colocar o pino  em nível lógico baixo, sem tensão alguma:

```
void loop() {
  // Aqui vou colocar meu codigo principal que sera executado varias vezes repetidamente
  digitalWrite(13,HIGH);
  digitalWrite(13,LOW);
}
```

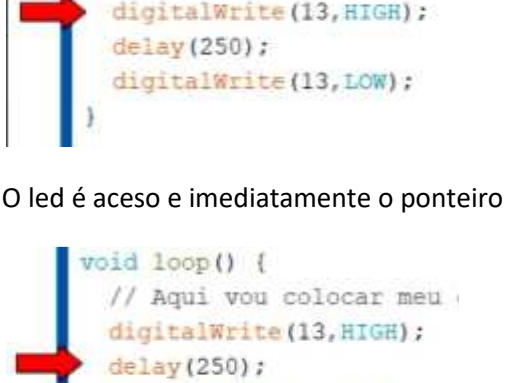
Vamos ver como ficou? Clique em Enviar... Ué Leandro mas o led não piscou e ficou meio apagado porque isso aconteceu? Lembra que na primeira aula nós vimos que o Arduino tem um cristal de 16Mhz que executa 16 milhoes de ciclos por segundo? Pois é... o Arduino está ligando e desligando o led tão rápido que assim que ele acende logo a instrução seguinte é executada e ele se apaga e isso acontece muito rápido e é por isso que vamos colocar logo aqui abaixo, assim que o led acende, a função delay() que vai fazer com que o micro aguarde alguns milisegundos antes de executar a instrução seguinte. Então vamos colocar... delay(250):

```
void loop() {
  // Aqui vou colocar meu codigo principal
  digitalWrite(13,HIGH);
  delay(250);
  digitalWrite(13,LOW);
}
```

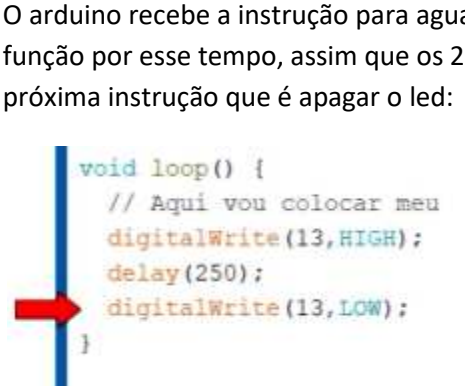
E envie o programa novamente.

Veja que agora o LED ficou aceso direto, porque? Bom agora chegamos no ponto mais importante da aula que é justamente entender a sequencia de instruções.

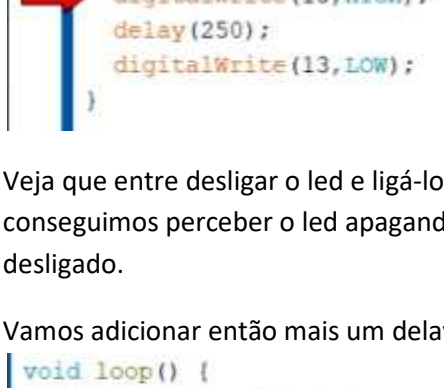
As instruções são executadas linha por linha de cima pra baixo. Quando o arduino é energizado a primeira instrução é o conteúdo da função setup porque o que tem acima é só comentário e é ignorado pelo compilador. Dentro da função setup temos outra função pra definir o pino 13 como saída. Como essa é a função setup ela é executa uma única vez então a seta continua pra baixo. Quando o ponteiro chega na função loop ele continua e a primeira instrução é para ligar o led:




O led é aceso e imediatamente o ponteiro vai para a próxima linha:




O arduino recebe a instrução para aguardar 250 milisegundos e o ponteiro fica preso nessa função por esse tempo, assim que os 250 milisegundos se passam o ponteiro vai para a próxima instrução que é apagar o led:




Como estamos na função loop o ponteiro chegou no final do bloco então ele volta para o topo e liga o led de novo:




Veja que entre desligar o led e ligá-lo novamente não houve uma interrupção por isso não conseguimos perceber o led apagando porque ele volta a ligar imediatamente assim que é desligado.

Vamos adicionar então mais um delay de 250ms logo abaixo da função como indicado:

E enviamos o programa novamente, veja agora que agora o led pisca normalmente.