

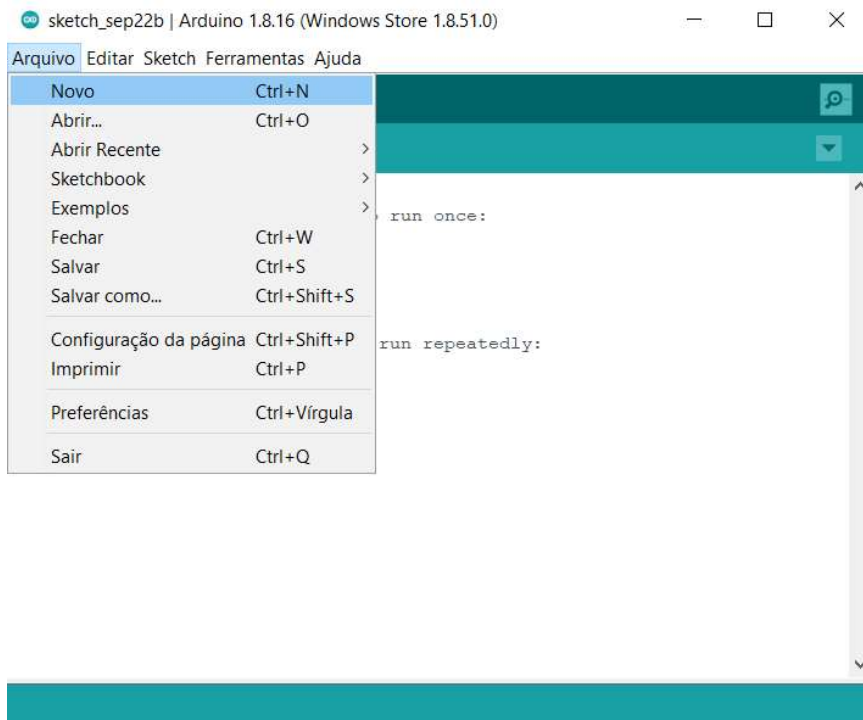
# TUTORIAL

## PROGRAMAÇÃO DO IDE

Este tutorial deve ser lido com calma! **Tente não pular as etapas e siga os passos sugeridos um a um**, para que você compreenda melhor o conteúdo desta aula e não precise reler o texto novamente.

Nesta aula tentaremos entender os conceitos fundamentais de uma programação para o IDE do Arduino. Estes conceitos se aplicam, de maneira em geral, a outras linguagens de programação pois todas estão baseadas em conceitos de lógica matemática e tomadas de decisão.

Assim, vamos começar iniciando um novo projeto na janela do Arduino IDE. Para isto inicie o aplicativo IDE e clique em **ARQUIVO > NOVO**. Salve este arquivo com o nome que desejar para sua análise futura.



Veja que a nova janela que se abriu representa justamente o Ambiente IDE aonde você colocará a programação que será gravada no Arduino. De uma maneira em geral, a programação envolve dois componentes principais: (1) uma função chamada “setup” e outra função chamada “loop” e ambas estão destacadas na figura a seguir.

```
teste §  
void setup() {  
  // put your setup code here, to run once:  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
}
```

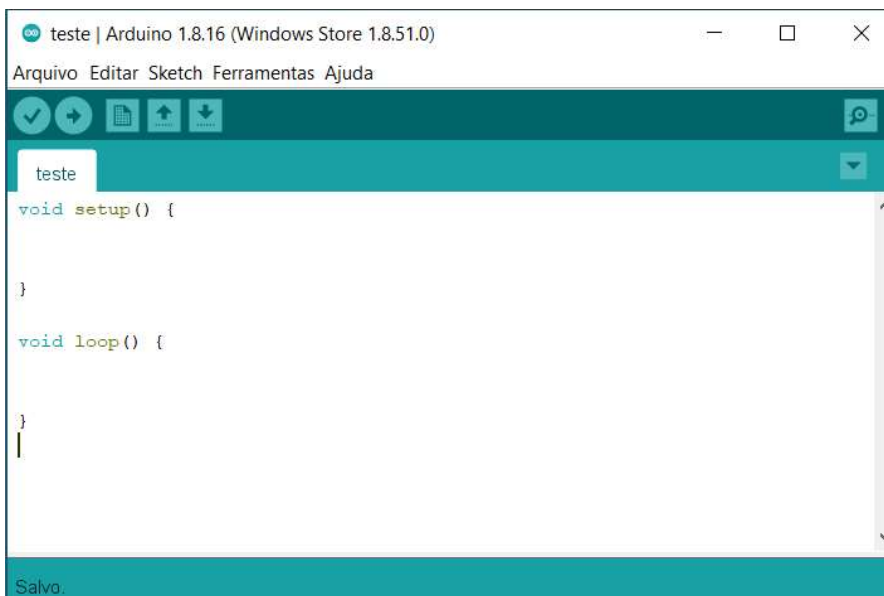
Salvo.

Notem que estas funções são escritas utilizando sempre o comando “void” seguido do nome da função com dois parênteses “()” em seguida. Veremos em outras ocasiões que é possível criar outras funções além de “setup()” e “loop()”. Além disto, observe atentamente que o conteúdo de cada uma destas funções é tudo o que fica entre as duas chaves “{}”; cada função tem seu conjunto de chaves.

No Arduino, a função “setup()” tem este nome porque ela é executada APENAS 1 VEZ, quando o equipamento é ligado. Já a função “loop()” é executada REPETIDAMENTE SEM PARAR (até desligar o equipamento); uma vez que os comandos desta função tenham sido executados até o fim, os comandos são novamente iniciados a partir do começo (por isto o nome “loop”).

É comum utilizar na função “setup()”: (1) comandos de inicialização de dispositivos; (2) comandos que definem valores iniciais de variáveis; (3) comandos em geral que devem ser executados uma única vez tão logo se inicie o dispositivo. Os demais comandos são normalmente colocados na função “loop()”.

Vale notar que no exemplo acima, nenhuma ação é executada pelo programa, já que todas as linhas do código de programação que são iniciadas por “//” são linhas de comentários e servem apenas para lembrar ao programador o que ele está fazendo no seu código de programação. Assim, o código acima é exatamente igual ao código abaixo; e a função “setup()” será executada 1 única vez e a função “loop()” repetidamente.



The image shows a screenshot of the Arduino IDE interface. The title bar reads "teste | Arduino 1.8.16 (Windows Store 1.8.51.0)". The menu bar includes "Arquivo", "Editar", "Sketch", "Ferramentas", and "Ajuda". The toolbar contains icons for saving, undo, redo, and other functions. The main editor area shows a sketch named "teste" with the following code:

```
void setup() {  
  
}  
  
void loop() {  
  
}
```

The status bar at the bottom indicates "Salvo." (Saved).

Já vimos em aulas anteriores como ligar, por exemplo, um LED. Vocês se lembram? Quando fizemos isto, utilizamos essencialmente 3 comandos. Vocês se lembram? Anote os comandos (códigos) que você lembra abaixo:

Usamos um comando chamado “pinMode()”, lembram? Este comando não funciona sozinho, pois precisava receber 2 parâmetros que são colocados dentro dos parênteses e separados por vírgula (“,”); de tal forma que o verdadeiro comando era “pinMode(param1, param2)”. Enquanto o parâmetro 1 (param1) poderia receber qualquer número inteiro e que este número representaria o pino do Arduino, o parâmetro 2 (param2) deveria receber ou o texto “OUTPUT” ou “INPUT”. Este parâmetro então definiria se o pino selecionado com o parâmetro 1 seria um pino de saída (que enviaria “energia/sinal” para um dispositivo) ou um pino de entrada (que receberia “energia/sinal” de um dispositivo qualquer). Vejam, portanto, que se este pino escolhido permanecer sempre, durante toda a programação, como um pino de saída por exemplo, seria razoável programar este pino uma única vez. Isto seria feito na função “setup()”, concordam?



```
void setup() {  
  pinMode(6, OUTPUT);  
  pinMode(7, INPUT);  
}  
  
void loop() {  
  
}
```

Notem que na figura acima eu defini o pino 6 do Arduino como sendo um pino de saída e o pino 7 como um pino de entrada.

Deve-se destacar que é **muito comum escrever cada comando da sua programação em uma linha a parte, e separar cada comando com um ponto-e-vírgula (“;”)**.

Vocês conseguem imaginar o que este programa faria quando carregado no Arduino? Nada, certo? Ele apenas **definiu** os pinos como sendo de saída e entrada.

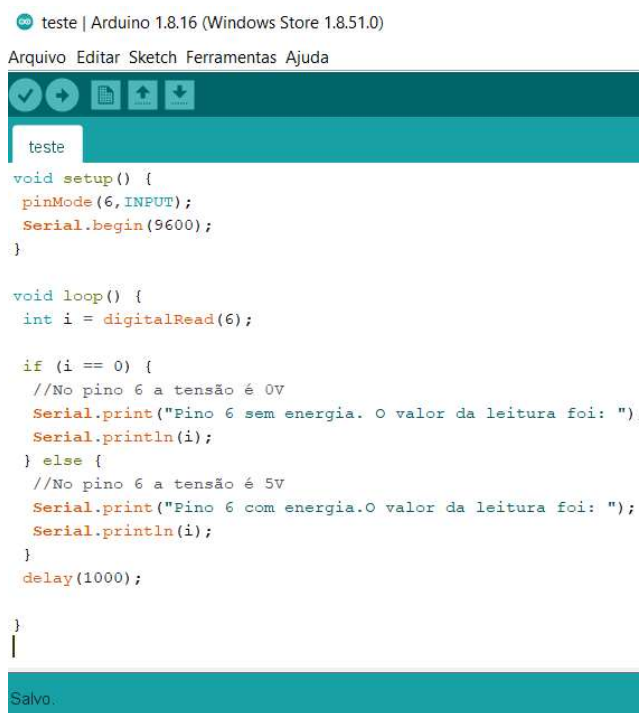
Bom, mas vocês aprenderam um outro comando: “digitalWrite(param1, param2)” que também recebe dois parâmetros: o parâmetro 1 (param1) é o número do pino, enquanto o param2 pode ser “HIGH” ou “LOW” apenas! Vejam que este comando é o comando que pede ao Arduino para “colocar” ou “tirar” a energia/sinal naquele pino em específico. Se você utilizar o comando “digitalWrite(7, HIGH)”, a princípio, você pede ao Arduino para que ele coloque energia/sinal (colocar uma voltagem/tensão de 5V) naquele pino 7! Mas atenção, veja o programa abaixo. Será que ele vai funcionar?



```
void setup() {  
  pinMode(6, OUTPUT);  
  pinMode(7, INPUT);  
}  
  
void loop() {  
  digitalWrite(7, HIGH);  
}
```

Se você respondeu que NÃO vai funcionar, você acertou! Mas por quê? Por que ainda que você tenha escrito o comando certo para “energizar” o pino 7 dentro da função “loop()”; o pino 7 foi definido como pino de entrada dentro da função “setup()”. Logo, não há como o pino 7 funcionar como pino de saída! Como você pode corrigir isto? Escreva sua solução abaixo.

Um outro comando associado ao pino pode ser o comando “digitalRead(param1)” que aceita apenas o parâmetro 1 (param1) que deve ser o pino que será feita a leitura. Se não houver nenhuma tensão (5V) chegando ao pino 6, o comando “digitalRead(6);” dará o resultado igual a 0; por outro lado se houver 5V chegando ao pino 6, o comando “digitalRead(6);” resultará num valor igual a 1. Veja no exemplo abaixo a utilização do comando “digitalRead()”.



```
teste | Arduino 1.8.16 (Windows Store 1.8.51.0)
Arquivo Editar Sketch Ferramentas Ajuda
teste
void setup() {
  pinMode(6, INPUT);
  Serial.begin(9600);
}

void loop() {
  int i = digitalRead(6);

  if (i == 0) {
    //No pino 6 a tensão é 0V
    Serial.print("Pino 6 sem energia. O valor da leitura foi: ");
    Serial.println(i);
  } else {
    //No pino 6 a tensão é 5V
    Serial.print("Pino 6 com energia.O valor da leitura foi: ");
    Serial.println(i);
  }
  delay(1000);
}

void setup() {
  pinMode(6, INPUT);
  Serial.begin(9600);
}

void loop() {
  int i = digitalRead(6);

  if (i == 0) {
    //No pino 6 a tensão é 0V
    Serial.print("Pino 6 sem energia. O valor da leitura foi: ");
    Serial.println(i);
  } else {
    //No pino 6 a tensão é 5V
    Serial.print("Pino 6 com energia.O valor da leitura foi: ");
    Serial.println(i);
  }
  delay(1000);
}
```

Carregue este programa no Arduino IDE e depois grave-o no Arduino. Vamos testar seu funcionamento ligando e desligando o pino 6 ao pino de 5V do Arduino. Vamos colocar energia no pino 6: o que deve acontecer? Anote a seguir sua expectativa.

O exemplo acima nos mostra outras funções que vocês já viram nas aulas anteriores. Uma destas funções foi utilizar a Porta Serial do Arduino (aquele conector que está conectado ao computador pela fio USB) para que o Arduino possa enviar informações para a tela do computador através do Monitor Serial. Vocês se lembram disto.

A primeira coisa que precisamos fazer para utilizar esta função é garantir que o Arduino tenha sua Porta Serial disponível para conexão. Alguns Arduinos possuem esta porta (pinos RX e TX) já adaptados para conectores tipo USB; o que torna a comunicação trivial. Basta “plugar” os conectores no Arduino e no Computador e começar a programação para transferir informações por este cabo.

A segunda coisa que devemos fazer é inserir um comando que vai dizer para o Arduino como se comunicar com o computador. Este comando é chamado de “Serial.begin(param1);” e o comando aceita apenas um parâmetro (param1) que é a velocidade da transmissão dos dados (usamos 9600 bits/segundo)! Vocês devem se lembrar que este valor deve ser o mesmo valor selecionado no Monitor Serial para que a comunicação entre Arduino e Computador seja perfeita. Vocês concordam que este comando deve ser inserido apenas 1 vez, dentro da função “setup()”; afinal de contas só nos interessa realizar esta definição 1 ÚNICA VEZ.

Vocês devem perceber que o comando “Serial.begin(9600);” não faz nada por si só! Ele apenas define uma forma de comunicação. Por outro lado, o comando “Serial.print(param1);” ou “Serial.println(param1);” são os comandos que efetivamente enviam alguma coisa do Arduino para o Computador (Monitor Serial). Qual a diferença entre eles? O parâmetro 1 neste caso pode ser qualquer variável que esteja na programação do Arduino bem como qualquer tipo de texto. Quando enviarmos texto, o parâmetro de envio deve estar entre aspas. Se enviarmos uma variável do sistema, ela deve ser escrita sem aspas! Para entender melhor isto, vejam os com’andos que estão no programa. Por exemplo:

“Serial.print(i);” e “Serial.print(“i”);” enviariam a mesma informação para o Monitor Serial?

Anote abaixo sua resposta e sua explicação!

Notem ainda no mesmo exemplo que utilizamos o comando “delay(1000);” cuja função é colocar o processador em espera por 1000 milisegundos (ou seja, 1 segundo!). Se alterarmos o parâmetro interno deste comando “delay(param1);” o processador permanecerá em espera pelo tempo determinado pelo parâmetro 1 (param1).

Mas temos também dois outros pontos importantes e diferentes neste programa:

1) Olhem a linha “int i = digitalRead(6);” Vamos interpretá-lo pois ele é muito importante em todo tipo de programação. Bom, vocês já sabem o que “digitalRead(6);” faz, certo? Ele lê o sinal que está no pino 6. Se o sinal lá for de 5V, então “digitalRead(6);” vale 1! Contudo, se não tiver energia no pino 6, “digitalRead(6);” vale 0. Notem que o valor do comando “digitalRead(6);” varia com o tempo e com a programação; hora ele vale uma coisa, hora ele vale outra coisa! Portanto esta leitura é variável! E para que o programa possa guardar o valor desta leitura, é necessário que criemos uma variável na minha programação. Esta variável que eu dei o nome de “i”, mas poderia chamá-la de “variável” ou ainda “minhaleitura” é apenas um nome que serve para guardar o valor das minhas leituras. Como minhas leituras são 0 ou 1, o tipo de variável que eu devo criar é do tipo inteiro ou int. Portanto, o comando para criar uma variável no meu programa é “int i = digitalRead(6);” como poderia ser “int myvar = digitalRead(6);”.

2) Outra linha de programação muito importante aqui é a seguinte:

**if (statement) {action} end {action}**

Sua versão simplificada é a seguinte:

**if (statement) {action}**

Vamos detalhá-la com calma na aula.