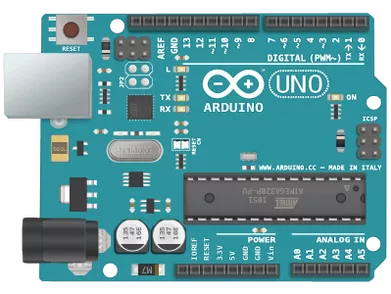
1. **Portas do Arduino**

****

***Power***

O grupo de pinos ***Power*** (visível na parte superior esquerda da imagem acima) possui todos os pinos de alimentação do arduino, tanto entradas de tensão quanto saídas, seguem abaixo descrições de cada uma destas portas: **IOREF:** Este pino fornece a referência de tensão com que o micro controlador opera. Um ***shield*** configurado corretamente pode ler a tensão do mesmo e selecionar a fonte de alimentação adequada ou habilitar tradutores de tensão nas saídas para trabalhar com a ***5V*** ou ***3.3V***. **RESET:** Esta porta realiza a reinicialização do Arduino uma vez que necessário durante a execução do programa, necessitando apenas enviar um pulso (negativo) na mesma. **3.3V:** Porta de saída de alimentação (positivo) 3.3 Volts.

**5V:** Porta de saída positivo direto, ou seja, não depende de comandos ou programação, injeta 5v continuamente enquanto o Arduino permanecer ligado. **GND:** Porta de alimentação ***negativa*** (entrada e/ou saída). **VIN:** Porta de entrada ***positivo***, ou seja, quando trabalharmos projetos móveis, como um robô ou um carrinho, o Arduino não estará conectado a USB, logo precisará de uma fonte de energia, nesta porta ligaremos o ***positivo*** de uma bateria, pilhas entre outros e o ***negativo*** ao ***GND*** citado anteriormente.

***Portas Analógicas* Portas A0 até A6:** Portas Analógicas, servem apenas para leitura de valores analógicos de ***0v*** até ***5v*** retornando um range de ***0*** até ***1023***.

***Portas Digitais* Portas 0 (RX) e 1 (TX)**: As Duas primeiras portas digitais do Arduino, relacionadas diretamente com a comunicação serial do processador (comunicação ***Serial***), por esse motivo **não é recomendado utilizar estas portas** uma vez que a utilização das mesmas pode gerar conflitos na comunicação padrão do Arduino.

**Portas de 2 até 13:** Portas ***Digitais*** do Arduino, podendo trabalhar valores lógicos como Desligado e Ligado (***LOW*** ou ***0*** que representa desligado e ***HIGH*** ou ***1*** que representa ligado)**,** servem basicamente para ligar e desligar ***atuadores*** como LEDs, Motores, entre outros, ou para receber a informação de ***sensores*** ou Shields envolvidos no seu projeto.

***Portas Digitais (PWM ~)***

Portas Digitais que comportam ***PWM*** (representadas na placa com um (~) til a frente), são portas capazes não só de enviar ou receber um pulso, como também de controlar este ciclo fazendo a “escrita analógica” de acordo com o valor passado (***0 a 255***), oscilando o pulso em determinado período de tempo. Através de ***PWM*** podemos controlar por exemplo, a intensidade de um LED, a velocidade de um Motor, emitir um tom específico para um Buzzer, entre outras aplicações.

***Obs***. Vale lembrar que ***PWM*** ou ***Pulse Width Modulation*** é a modulação do pulso, ou seja, faz a alternância entre ligado (***5v***) e desligado (***0v***) de acordo com a resolução passada ao pino PWM utilizado.

1. **Métodos padrão**

Um método é uma ação a ser executada no programa; No caso do Arduino, existem dois métodos padrão (***void setup()*** e ***void loop()***)que o sketch sempre executará, sem a necessidade do desenvolvedor o invocar, estes métodos são executados implicitamente. Acompanhe abaixo a definição de cada um destes:

**void setup()**

A instrução ***void*** define que o método não tem retorno, ou seja, executará as ações e não retorna valor algum, à ***setup***, é o nome do método, no caso é ele que o processador do Arduino invoca sempre que o mesmo é ligado.

Nos parênteses, seriam definidos os parâmetros que o método necessita para funcionar, no caso do exemplo de soma, suponha que precisaríamos retornar o resultado da soma de dois valores, o método ficaria:

float soma(float v1, float v2){

return v1 + v2;

}

No caso do método padrão ***setup()*** não há parâmetros, nestes casos os parênteses permanecem vazios.

**void loop()**

A função ***loop()*** é executada após a execução de ***setup()***, a função ***setup()*** como já vimos, é executada uma única vez quando o Arduino inicia, já a ***loop()*** é executada repetidamente até que o Arduino seja desligado, fazendo assim uma sequência de ações infinitamente, até que o Arduino seja desligado.

1. **Comandando Entradas/Saídas Digitais**

No Arduino, para comandar as portas digitais, utilizaremos as funções principais **pinMode(), digitalWrite()** e **digitalRead()**, da forma:

**pinMode()**

A função pinMode() é responsável por definir se a porta digital é saída (**OUTPUT**) ou entrada (**INPUT** ou **INPUT\_PULLUP**).

Definiremos a porta como **OUTPUT** sempre que desejarmos fazer a escrita digital na mesma, e ainda ler a porta (para sabermos se a porta encontra-se ligada ou desligada);

Definiremos a porta como **INPUT** sempre que desejarmos efetuar somente a LEITURA da porta digital, neste caso, não será possível efetuar a escrita;

Definiremos a porta como **INPUT\_PULLUP** quando desejarmos efetuar a leitura digital do pino, com o resistor pull-up interno do Arduino habilitado, veja mais no Apêndice - Resistor Pull-up interno;

**digitalWrite()**

A função **digitalWrite()** faz a escrita de valores digitais, ou seja, liga (5V) e desliga (0V) o pino em uma determinada porta, podendo receber os valores **HIGH** ou 1 para ligado e **LOW** ou 0 para desligado.

**digitalRead()**

A função **digitalRead()** faz a leitura de valores digitais, ou seja, retorna os valores **HIGH** ou **1** para **ligado** e **LOW** ou **0** para **desligado**.

**NOTA**: Vale salientar que os pinos de entrada analógica, podem ser utilizados como pinos digitais, para isso, referenciamos em **pinMode()** e **digitalWrite()** ou **digitalRead()** o pino analógico, exemplo: **digitalRead(A0)**;

1. **Comandando Entradas/Saídas Analógicas**

**analogRead()**

A função **analogRead()** será utilizada para fazer a leitura analógica de uma porta **Analog In (entrada analógica),** ela nos retorna um range de **0 a 1023** de acordo com a tensão recebida no pino analógico determinado, onde **0 equivale 0V e 1023 equivale a 5V**, os portas analógicas seguem destacadas abaixo:

1. **Tipos de Dados (Variáveis e retorno de métodos)**

Quando declaramos uma variável, precisamos definir o tipo de dado da mesma, caso a variável necessite armazenar números inteiros, o ideal seria ***int,*** ainda existem diversos tipos de dados, abaixo segue a lista da maioria deles:

**boolean**: valor verdadeiro (true) ou falso (false)

**char**: um único caractere

**byte**: um byte, ou sequência de 8 bits

**int**: número inteiro de 16 bits com sinal (-32768 a 32767)

**unsigned int**: número inteiro de 16 bits sem sinal (0 a 65535)

**long**: número inteiro de 16 bits com sinal (-2147483648 a 2147483647)

**unsigned long**: número inteiro de 16 bits sem sinal (0 a 4294967295)

**float**: número real de precisão simples (ponto flutuante)

**double**: número real de precisão dupla (ponto flutuante)

**String**: sequência de caracteres

**void**: utilizado em métodos, tipo vazio (sem retorno)

**Exemplos**:

boolean teste = v1 > v2; //Neste exemplo, estamos declarando a variável boolean e comparando se v1 é maior que v2, se sim a variável teste receberá o valor true se não, receberá o valor false.

char letra = ‘A’; //Neste exemplo, estamos declarando a variável char e atribuindo a letra A para a mesma;

int led = 13; // Neste exemplo, estamos declarando a variável int e declarando o valor 13 para a mesma;

float valor = 15.5 // Neste exemplo, estamos declarando a variável float valor e declarando o valor 15.5 (quinze e meio) para a mesma;

String nome = “Gabriel Pereira”; //Neste exemplo, estamos declarando a variável String e definindo o valor Gabriel Pereira para a mesma;

***Importante:*** Repare que utilizamos alguns símbolos como ***aspas (“”)*** na atribuição de ***String***, ***(‘’) apóstrofo*** na atribuição de ***char*** e ***ponto (.)*** na atribuição de ***números com casas decimais***, a utilização dos mesmos é obrigatório sempre que trabalharmos com estes tipos de dados, seja para atribuição de valores em variáveis, comparações, e qualquer outra situação que necessitarmos dos mesmos.

1. **Operadores, Variáveis, Constantes e Comentários**

**Constantes**

Constantes são estruturas que não podem ter seu valor alterado durante a execução. No Arduino, constantes podem ou não possuir tipos de dados, sendo declaradas da seguinte maneira:

const int led = 13; // Declarando uma constante utilizando a cláusula const do C e tipando a variável, ESTA FORMA DE DECLARAR CONSTANTES É ALTAMENTE NÃO RECOMENDADA

#define led 13 // Declarando constante utilizando a cláusula #define nativa do Arduino e ALTAMENTE RECOMENDADA

**Operadores**

**Operadores aritméticos**

Sempre que necessitarmos de uma operação aritmética, utilizaremos estes operadores: **+** : adição

**-** : subtração

**\*** : multiplicação

**/** : divisão

**%** : divisão modular (retorna o resto da divisão de inteiros)

**Operadores lógicos (comparação)**

**==** : igualdade ("igual a")

**!=** : desigualdade ("diferente de")

**!** : negação ("não")

**>** : "maior que"

**<** : "menor que"

**>=** : "maior ou igual a"

**<=** : "menor ou igual a"

**&&** : conjunção ("e")

**||** : disjunção ("ou")

***Importante: No caso de comparação de igualdade de Strings, utilizar o método equals() de String, exemplo:***

String nome = “Gabriel”; nome.equals(“Gabriel”); //true

**Operador Ternário**

O operador ternário é de extrema utilidade onde necessitamos de uma comparação (***if***) de respostas muito simples, em outras palavras, operador ternário é um ***if*** de uma única linha, veja exemplo:

idade >= 18 ? "Maior de Idade" : "Menor de Idade"

Na sintaxe do operador ternário, é utilizado interrogação (***?***) para separar o teste lógico do valor verdadeiro, neste caso “Maior de Idade“ e os dois pontos (***:***) para separar o valor verdadeiro do valor falso, neste caso “Menor de Idade”.

**Comentários**

As estruturas de comentários na Linguagem Arduino assim como no C++ são:

// Comentário de Linha

OU

/\*

Comentário de bloco

\*/

**7. Funções e Métodos Pré-Definidos de Conversão**

➢ ***String(valor) ->*** Converte um valor em String, seja ele qual for, por exemplo, se desejamos imprimir na serial “***Valor***” e um determinado valor (suponha float) de uma leitura analógica por exemplo, podemos fazer de duas formas:

**Serial**.print("Valor: "); **Serial**.println(v);

Ou de forma mais enxuta com método de conversão:

**Serial**.println("Valor: " + String(v));

➢ ***.charAt(int posição) ->*** Retorna um único caractere de acordo com a posição passada como parâmetro, exemplo:

String nome = "Gabriel Pereira"; **Serial**.println(nome.charAt(2)); // Imprime na Serial o caractere b

➢ ***.equals(String s) ->*** Compara a igualdade da String com a String passada como parâmetro de forma *sensitive case (diferencia maiúsculas de minúsculas)*, exemplo:

String nome = "Gabriel Pereira"; **Serial**.println(nome.equals("Gabriel pereira")); // Imprime na Serial false //por conta do “p” minúsculo

➢ ***.equalsIgnoreCase(String s) ->*** Compara a igualdade da String com a String passada como parâmetro de forma *insensitive case (não diferencia maiúsculas de minúsculas)*, exemplo:

String nome = "Gabriel Pereira"; **Serial**.println(nome.equalsIgnoreCase("gabriel pereira")); // Imprime na - //Serial true

➢ ***.length() ->*** Retorna um número inteiro, a contagem de caracteres desta String, por efetuar a contagem não recebe parâmetros, por exemplo:

String nome = "Gabriel Pereira"; **Serial**.println(nome.length()); // Imprime na Serial 15

***.substring(int posição inicial, int posição final) ->*** Retorna uma ***substring*** (um trecho) da String de acordo com as posições passadas como parâmetro,

***IMPORTANTE****: o método substring não retorna o caractere da posição final, retorna o caractere da posição inicial até uma antes do passado como posição final*, veja exemplo:

String nome = "Gabriel Pereira"; **Serial**.println(nome.substring(1, 3)); // Imprime na Serial ab

➢ ***.toInt() ->*** Converte uma String para número inteiro (se assim for possível), exemplo:

String idade = "22"; **Serial**.println(idade + 3); // Não compila

**Serial**.println(idade.toInt() + 3); // Imprime na Serial 25

➢ ***.toFloat() ->*** Trabalha exatamente igual ao anterior, porém retornando um número ***float.***

➢ ***.toLowerCase() ->*** Converte uma String inteira para lower case (minúsculas), veja exemplo:

String nome = "GaBriEl PeREirA"; **Serial**.println(nome.toLowerCase()); // Imprime na Serial gabriel pereira

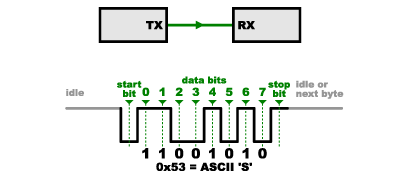
➢ ***.toUpperCase() ->*** Converte uma String inteira para upper case (maiúsculas), veja exemplo:

String nome = "GaBriEl PeREirA"; **Serial**.println(nome.toUpperCase()); // Imprime na Serial GABRIEL PEREIRA

*Obs. Para mais funções de conversão acesse a referência oficial:* https://www.arduino.cc/en/Reference/StringObject

**8. Comunicação Serial**

Comunicação serial nada mais é que a transferência de dados ***bit a bit*** (um de cada vez), alternando entre nível lógico alto e baixo, no caso do Arduino, 0V e 5V, como demonstrado na imagem abaixo.



É usada para a comunicação entre diversos dispositivos como ***Arduino***, ***computador***, ***displays de mensagem, teclado, mouse, modem, terminais***, entre muitos outros. Podemos gerar uma comunicação serial entre o ***Arduino*** e o ***computador***, ou ainda entre ***Arduino*** e um ***display de mensagem*** por exemplo.

É diferente da comunicação paralela na qual os dados são enviados simultaneamente, podendo ser usada na comunicação com impressoras, scanners, unidade de disco removível.

Cada bit representa uma condição de voltagem (ligado - on ou desligado - off) em um dos pinos da porta serial (***DB9*** ou ***DB25***), para envio ou recebimento de dados, representando ***0 (desligado)*** ou 1 ***(ligado).***

Existem muitos padrões de comunicação Serial, abaixo segue a lista de exemplo dos principais padrões:

▪ Código Morse;

▪ RS-232 (baixa velocidade, implementado em portas seriais);

▪ RS-423;

▪ RS-485;

▪ USB - Universal Serial Bus (velocidade moderada, para a conexão de periféricos de computador);

▪ FireWire;

▪ Ethernet;

▪ InfiniBand (velocidade muito alta, comparado em escopo com o PCI);

▪ MIDI (controle de instrumentos musicais);

▪ DMX512 (controle de iluminação teatral);

▪ Serial Attached SCSI;

▪ Serial ATA;

▪ PCI Express;

▪ SONET e SDH (telecomunicação de alta velocidade sobre fibra ótica).

A comunicação serial acontece na maioria das vezes com 5V mas ainda existem padrões que trabalham com maior tensão. Para que ocorra a comunicação, suponha a comunicação serial entre o computador e o Arduino via USB, temos de ligar o ***RX*** do computador ao ***TX*** do Arduino, e o ***TX*** do computador ao ***RX*** do Arduino, este esquema de ligação é conhecido popularmente como ligação em ***X*** pois para que ocorra a comunicação, o ***TX*** (Transmissor) tem de enviar o ***bit*** e do outro lado deve haver um ***RX*** (Receptor) para receber esta informação. Se ligássemos por exemplo, ***TX*** do computador ao ***TX*** do Arduino, ambos tentariam enviar informação e nenhum receberia...

**9. Arrays**

Um ***Array*** é uma coleção de um ou mais objetos, normalmente do mesmo tipo (altamente recomendado também), armazenados em endereços de memória. Cada objeto que compõe o ***Array*** é chamado de elemento do ***Array***. Da mesma forma que para variáveis simples, damos um nome ao ***Array***. O tamanho do ***Array*** é o seu número de elementos. A grosso modo, ***Arrays*** nada mais são que variáveis de várias posições (a variável seria uma gaveta que armazena determinado tipo de valor, ***Array*** será nosso gaveteiro).

Para criarmos um Array, iniciaremos como uma variável comum, passando tipo e nome do mesmo, em seguida, definiremos que isso é um ***Array*** utilizando de colchetes (***[]***), dentro dele, colocaremos o tamanho deste ***Array,*** após isso, para definirmos os valores presentes no mesmo, utilizaremos igual (***=***) e dentro de chaves (***{}***), passaremos os valores separados por vírgula. No projeto do Display de 7 segmentos, criaremos um Array simples que armazenará os pinos utilizados e outro (agora Array bidimensional) que definirá quais segmentos devem ser ligados e desligados para formar um número, fica assim:

byte pinos[7] = {2, 3, 4, 5, 6, 7, 8};

byte numeros[10][7] = {

{1,1,1,1,1,1,0}, // = Digito 0

{0,1,1,0,0,0,0}, // = Digito 1

{1,1,0,1,1,0,1}, // = Digito 2

{1,1,1,1,0,0,1}, // = Digito 3

{0,1,1,0,0,1,1}, // = Digito 4

{1,0,1,1,0,1,1}, // = Digito 5

{1,0,1,1,1,1,1}, // = Digito 6

{1,1,1,0,0,0,0}, // = Digito 7

{1,1,1,1,1,1,1}, // = Digito 8

{1,1,1,0,0,1,1} // = Digito 9

};

Podemos acessar os valores do Array utilizando, por exemplo: números[1][5]; // Neste caso 0

***Dicas****:*

• *Lembre-se que a indexação do Arduino se inicia em 0 portanto o Array byte pinos[7]... possui valores em endereços de 0 a 6.*

• *Podemos recuperar tamanhos de Arrays com o método* ***.length(),*** *como já vimos anteriormente.*

Para prosseguirmos, precisamos definir os pinos como ***OUTPUT*** no ***setup()*** porém perceba que são 7 pinos, faremos um a um manualmente? E se eu estivesse usando um Arduino MEGA e fosse definir os pinos de 2 até 50 como ***OUTPUT,*** vale a pena o trabalho braçal? Nestes casos, o mais rápido e inteligente de se fazer é utilizando um laço de repetição.

**9. Laços de repetição**

*while*

O comando while (enquanto) é o mais simples dos laços de repetição. Podemos entendê-lo como um repetidor de informação devido a uma expressão lógica.

***Exemplo***

void setup(){

**Serial**.begin(9600);

}

void loop(){

int i = 0;

while(i <= 10){

**Serial**.println("i:" + String(i));

i++;

}

}

Este exemplo, imprime números na serial de 0 até 10.

Nota: Repare que neste exemplo, o ***int i*** é declarado dentro de ***loop()*** isso torna o ***i*** uma variável de escopo local, ou seja, só pode ser utilizada dentro de ***loop(),*** caso tentarmos utilizá-la dentro de ***setup()*** por exemplo, o Arduino retornaria um erro de variável não declarada e não compilava o sketch. Quando precisamos usar a variável em todo o projeto, devemos declará-la acima de tudo (como já vimos anteriormente) o que torna a mesmo de escopo global (pode ser acessada em qualquer parte do projeto).

*do while*

A única diferença entre o ***while*** e o ***do while*** é a **primeira execução**. No caso do ***while***, somente será executado o comando caso a validação desde a primeira tentativa (execução) seja verdadeira. No ***do while***, o código é executado a primeira vez e posteriormente é validado.

***Exemplo***

void setup(){

**Serial**.begin(9600); }

void loop(){

int i = 0;

do{

**Serial**.println("i:" + String(i));

}while(i <= 10);

}

**for**

O comando ***for*** tem as mesmas utilidades de um comando ***while***, porém o incremento da variável cuja expressão lógica está vinculada é feito dentro do próprio comando, tornando-o muito mais prático na maioria das situações.

O mesmo exemplo, feito com ***for*** ficaria:

void setup(){

**Serial**.begin(9600);

}

void loop(){

for(int i = 0; i <= 10; i++){

**Serial**.println("i:" + String(i));

}

}