PMR 3100 – Introdução à Engenharia Mecatrônica

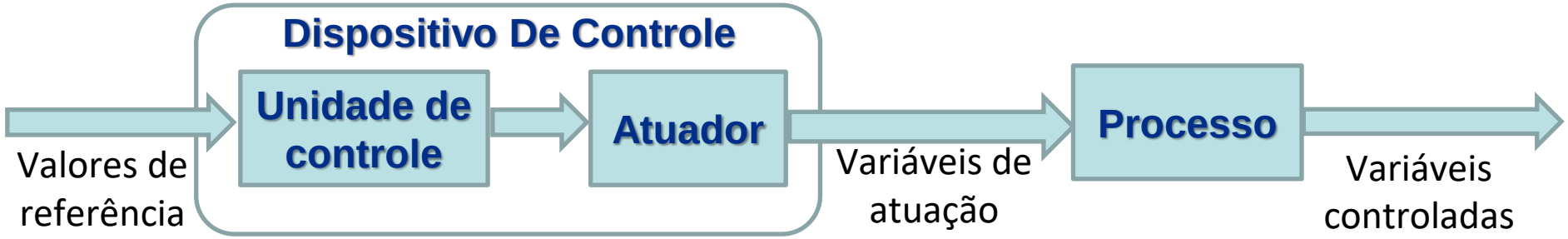
Módulo 04 – Meu Primeiro Robô

Aula 04 – Arduino, TinkerCAD, datasheets

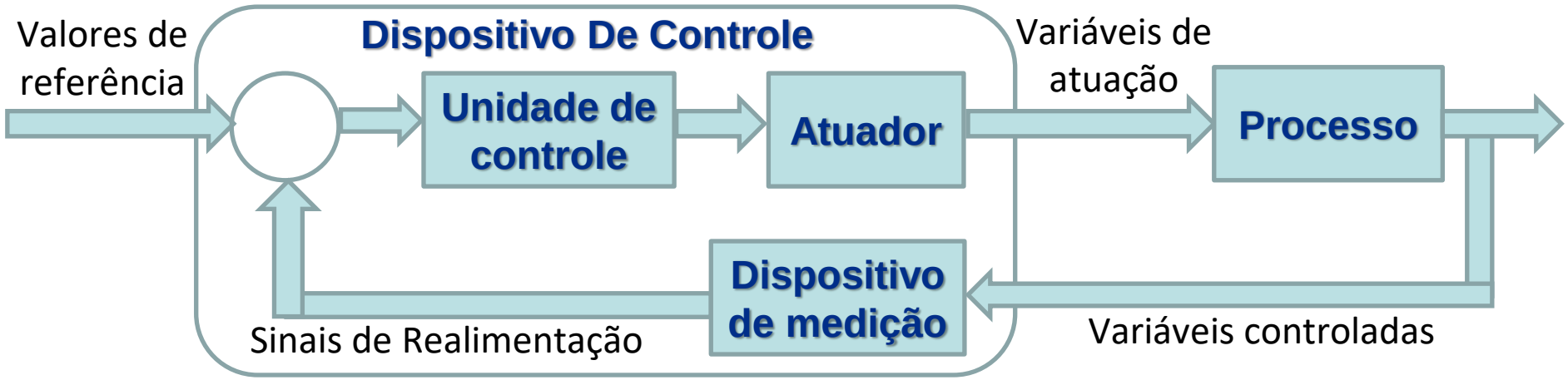
Prof. Dr. Rafael Traldi Moura

Controle – malha aberta e fechada

- Sistema mecatrônico com controle de malha aberta:



- Sistema mecatrônico com controle de malha fechada:



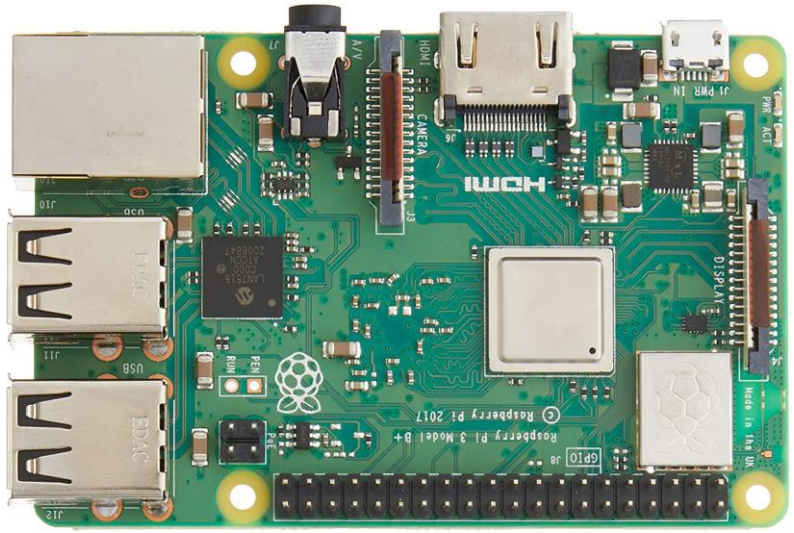
Microcontrolador x Microprocessador

- *Mas professor... Eu não sei se eu uso um Arduino ou um Raspberry Pi no meu projeto....*

Microcontrolador
ATmega328p

Microcontroller development board

- Carrega um *firmware*, um único *programa*;
- Memória em KB ou MB;
- Clock de MHz;



SBC - Single board Computer

- Roda um SO, com vários programas;
- Memória em GB ou TB;
- Clock de GHz.

Microcontrolador x Microprocessador

Microcontrolador

Microprocessador

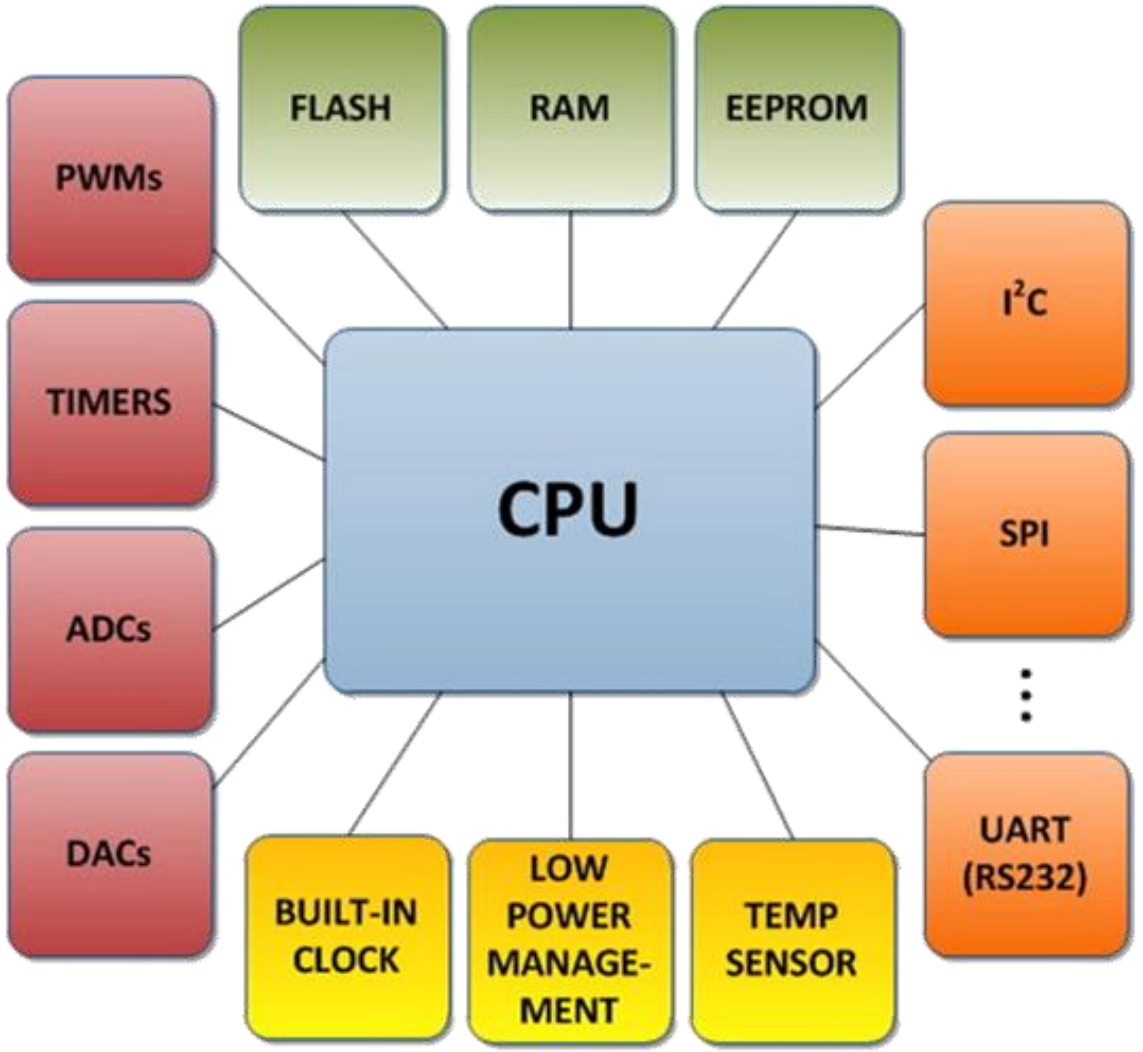
2+?=4

Arduino uno – 42mA e 7V = **0,29Watt**

Raspberry Pi 3 – 500mA a 1A e 5V = **2,5 a 5Watt**

Microcontrolador

- Um microcrontolador, MCU, é um SoC (System-on-a-Chip), pois é um CI (circuito integrado) que contém uma CPU, memória, entradas e saídas, protocolos de comunicação, conversores analógico digital e digital analógico, timers e PWMs.
- Exemplos de microcontroladores:
  - Atmel (AVR);
  - Microchip (PIC);
  - ST (ARM);
  - Raspbarry Pi Pico (ARM) – RP2040.



Datasheets – tecnologias, processador e pinos

ATmega328p

The ATmega48A/PA/88A/PA/168A/PA/328/P is a low power, CMOS 8-bit microcontrollers based on the AVR® enhanced RISC architecture.

4/8/16/32KBytes of In-System Self-Programmable Flash program memory

28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF

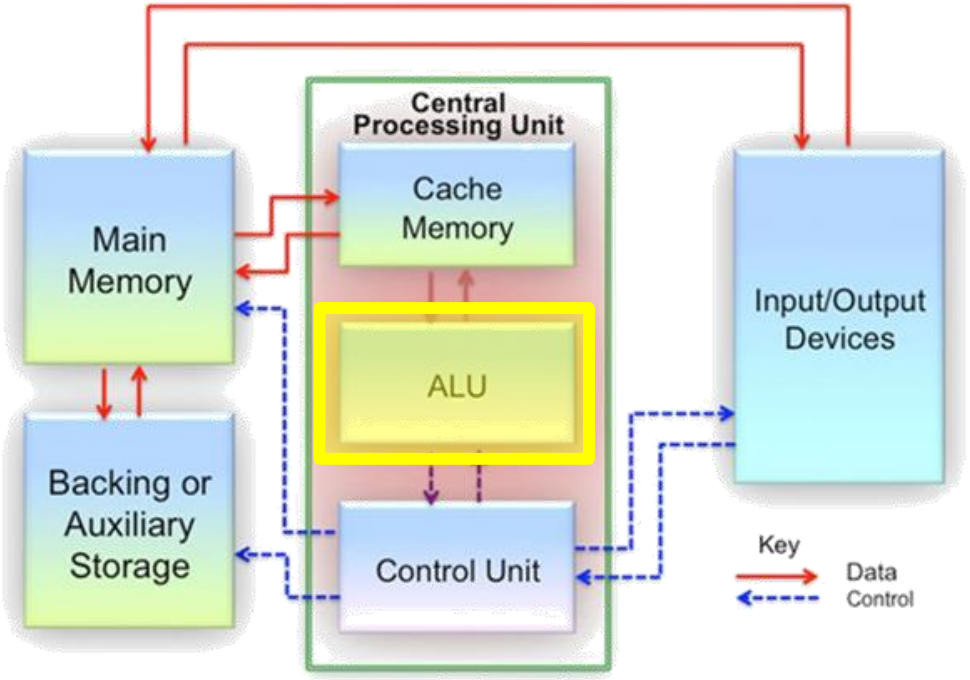
23 Programmable I/O Lines

PIC16F886

28/40/44-Pin Flash-Based, 8-Bit CMOS Microcontrollers

High-Performance RISC CPU

24/35 I/O Pins with Individual Direction Control



Datasheets – performance e consumo

ATmega328p

- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier

- Power Consumption at 1MHz, 1.8V, 25°C
  - Active Mode: 0.2mA
  - Power-down Mode: 0.1µA
    - Power-save Mode: 0.75µA (Including 32kHz RTC)

PIC16F886

High-Performance RISC CPU:

- Only 35 Instructions to Learn:
  - All single-cycle instructions except branches
- Operating Speed:
  - DC – 20 MHz oscillator/clock input
  - DC – 200 ns instruction cycle
- Interrupt Capability
- 8-Level Deep Hardware Stack
- Direct, Indirect and Relative Addressing modes

Low-Power Features:

- Standby Current:
  - 50 nA @ 2.0V, typical
- Operating Current:
  - 11 µA @ 32 kHz, 2.0V, typical
  - 220 µA @ 4 MHz, 2.0V, typical

Datasheets – memórias

ATmega328p

Device	Flash	EEPROM	RAM
ATmega48A	4KBytes	256Bytes	512Bytes
ATmega48PA	4KBytes	256Bytes	512Bytes
ATmega88A	8KBytes	512Bytes	1KBytes
ATmega88PA	8KBytes	512Bytes	1KBytes
ATmega168A	16KBytes	512Bytes	1KBytes
ATmega168PA	16KBytes	512Bytes	1KBytes
ATmega328	32KBytes	1KBytes	2KBytes
ATmega328P	32KBytes	1KBytes	2KBytes

- High Endurance Non-volatile Memory Segments
  - 4/8/16/32KBytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1KBytes EEPROM
  - 512/1K/1K/2KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)

PIC16F886

Device	Program Memory	Data Memory	
	Flash *(words)	SRAM (bytes)	EEPROM (bytes)
PIC16F882	2048	128	128
PIC16F883	4096	256	256
PIC16F884	4096	256	256
PIC16F886	8192	368	256
PIC16F887	8192	368	256

High Endurance Flash/EEPROM Cell:

- 100,000 write Flash endurance
- 1,000,000 write EEPROM endurance
- Flash/Data EEPROM retention: > 40 years

*word => 14-bits

Datasheets – recursos

ATmega328p

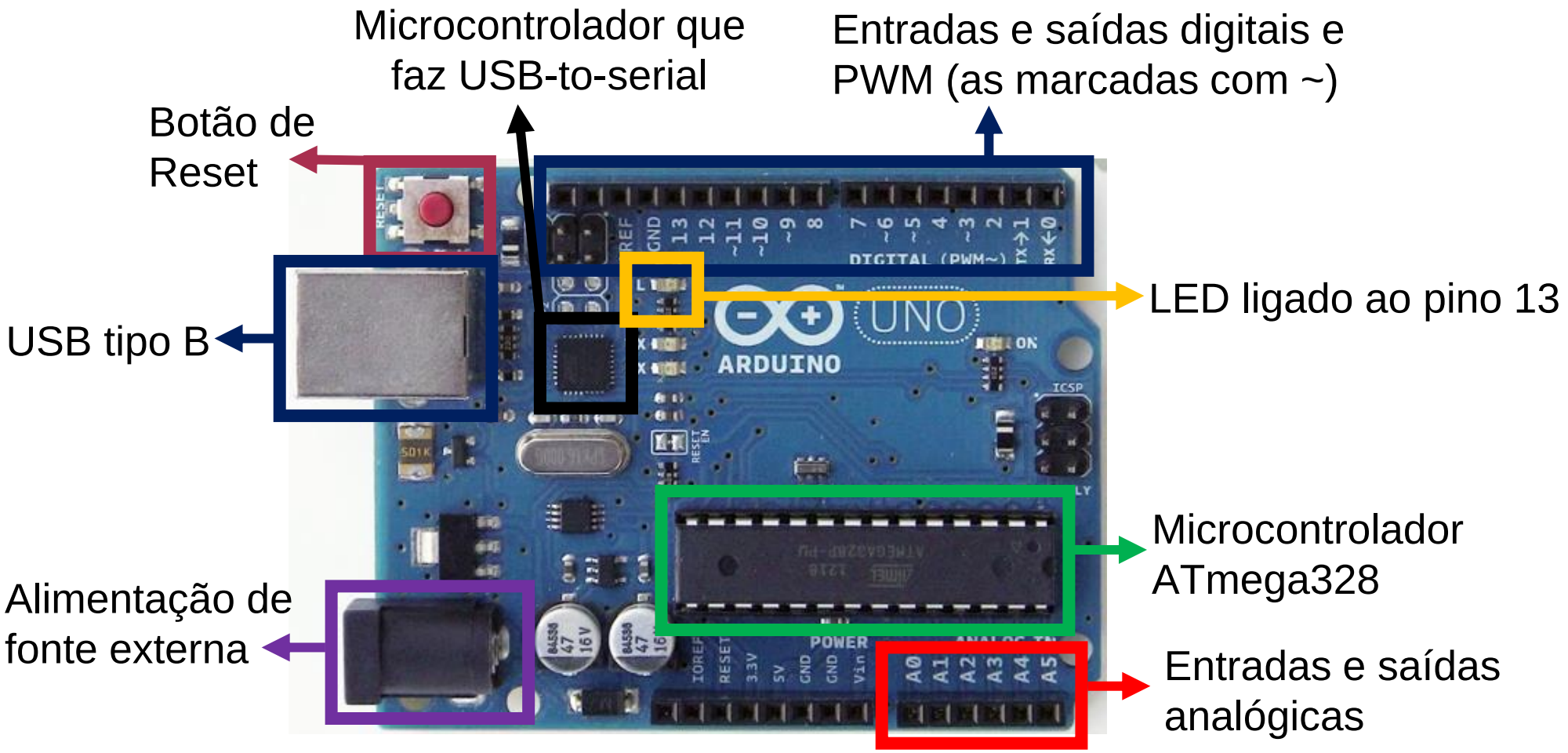
The ATmega48A/PA/88A/PA/168A/PA/328/P provides the following features: 4K/8Kbytes of In-System Programmable Flash with Read-While-Write capabilities, 256/512/512/1Kbytes EEPROM, 512/1K/1K/2Kbytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented 2-wire Serial Interface, an SPI serial port, a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages), a programmable Watchdog Timer with internal Oscillator, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, USART, 2-wire Serial Interface, SPI port, and interrupt system to continue functioning.

PIC16F886

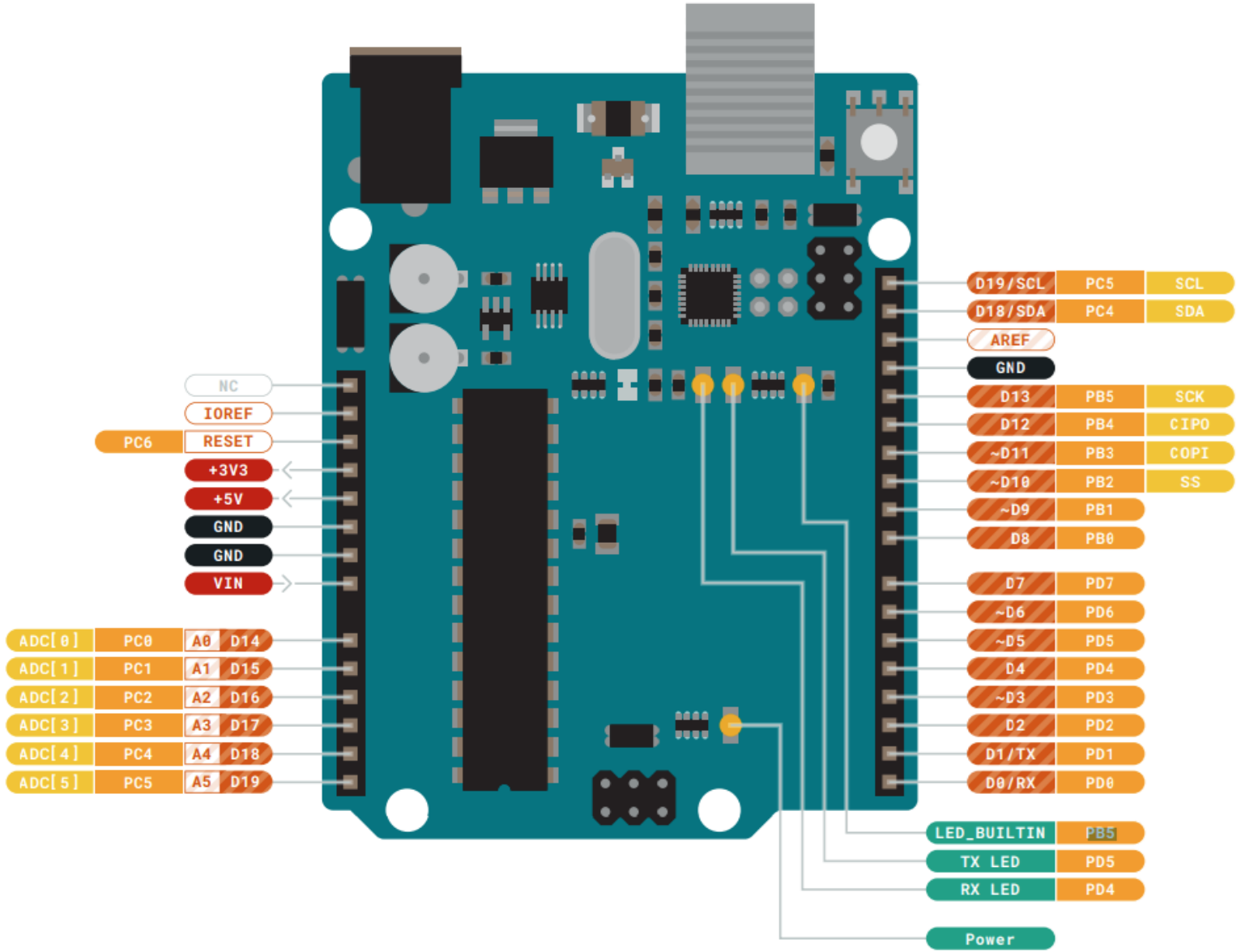
Device	Program Memory	Data Memory		I/O	10-bit A/D (ch)	ECCP/ CCP	EUSART	MSSP	Comparators	Timers 8/16-bit
	Flash (words)	SRAM (bytes)	EEPROM (bytes)							
PIC16F882	2048	128	128	24	11	1/1	1	1	2	2/1
PIC16F883	4096	256	256	24	11	1/1	1	1	2	2/1
PIC16F884	4096	256	256	35	14	1/1	1	1	2	2/1
PIC16F886	8192	368	256	24	11	1/1	1	1	2	2/1
PIC16F887	8192	368	256	35	14	1/1	1	1	2	2/1

- A/D Converter:
  - 10-bit resolution and 11/14 channels
- Timer0: 8-bit Timer/Counter with 8-bit Programmable Prescaler
- Enhanced Timer1:
  - 16-bit timer/counter with prescaler
  - External Gate Input mode
  - Dedicated low-power 32 kHz oscillator
- Timer2: 8-bit Timer/Counter with 8-bit Period Register, Prescaler and Postscaler
- Enhanced Capture, Compare, PWM+ Module:
  - 16-bit Capture, max. resolution 12.5 ns
  - Compare, max. resolution 200 ns
  - 10-bit PWM with 1, 2 or 4 output channels, programmable “dead time”, max. frequency 20 kHz
  - PWM output steering control
- Capture, Compare, PWM Module:
  - 16-bit Capture, max. resolution 12.5 ns
  - 16-bit Compare, max. resolution 200 ns
  - 10-bit PWM, max. frequency 20 kHz
- Enhanced USART Module:
  - Supports RS-485, RS-232, and LIN 2.0
  - Auto-Baud Detect
  - Auto-Wake-Up on Start bit
- In-Circuit Serial Programming™ (ICSP™) via Two Pins
- Master Synchronous Serial Port (MSSP) Module supporting 3-wire SPI (all 4 modes) and I^2C™ Master and Slave Modes with I^2C Address Mask

Placa de desenvolvimento



Placa de desenvolvimento

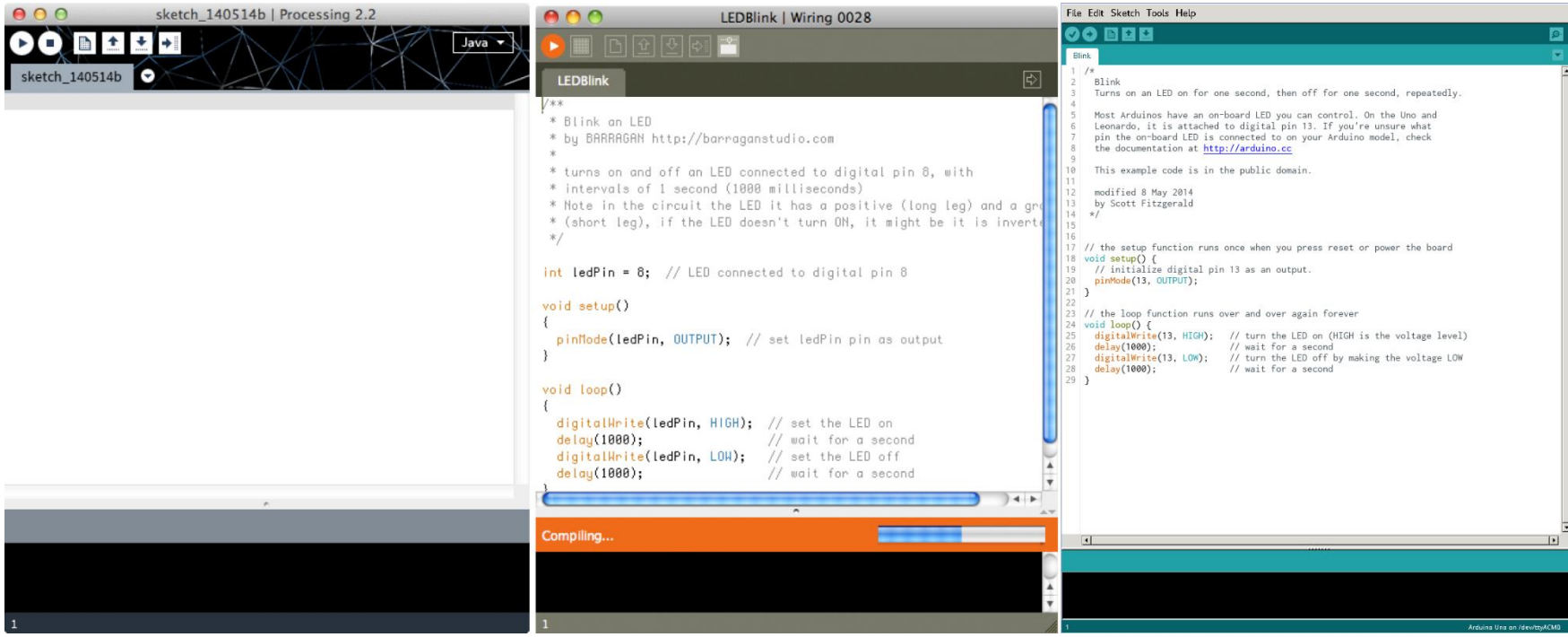


- Ground
- Power
- LED
- Internal Pin
- SWD Pin
- Digital Pin
- Analog Pin
- Other Pin
- Microcontroller's Port
- Default

Então porque o Arduino deu tão certo?

- Vimos que a placa Arduino Uno não passa de uma placa de desenvolvimento baseada no microcontrolador ATmega328p. Então porque outras placas de desenvolvimento não fizeram tão sucesso anteriormente?

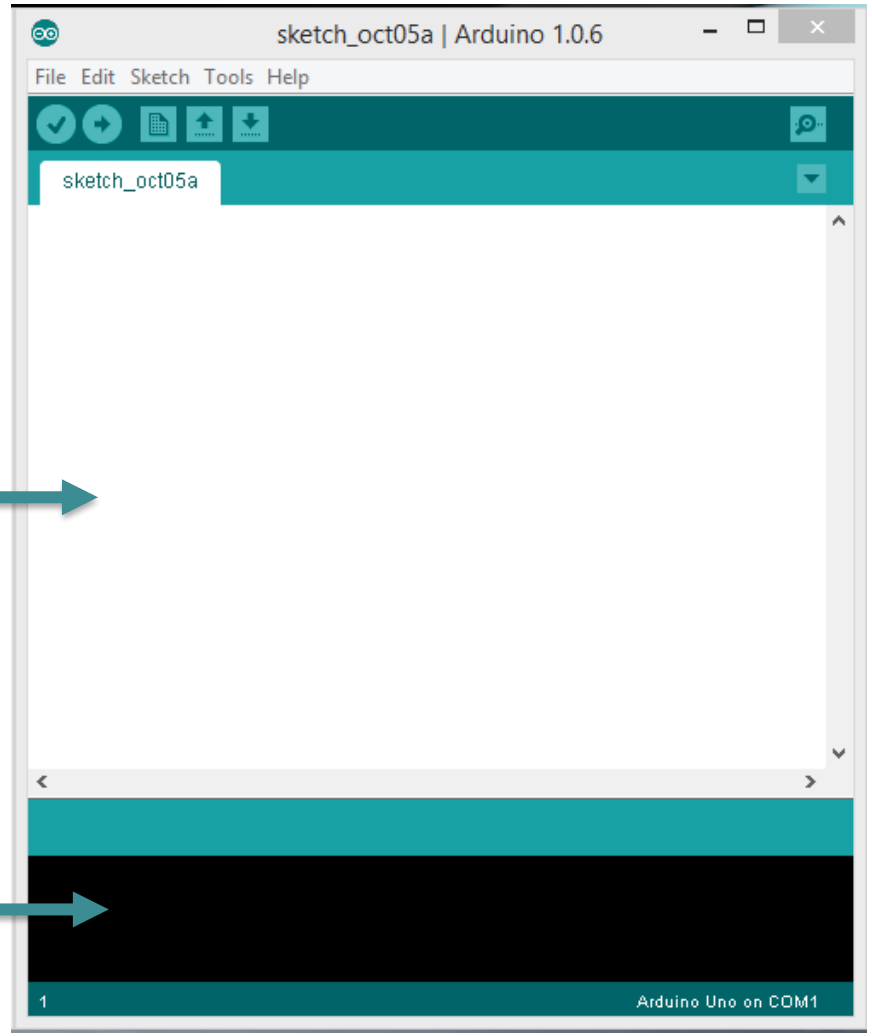
- A resposta está baseada em 2 fatores:
  - IDE Arduino, baseada no Processing;
  - Linguagem de programação baseada em Wiring.



- Para mais detalhes, leia https://arduinohistory.github.io/ .

Arduino IDE

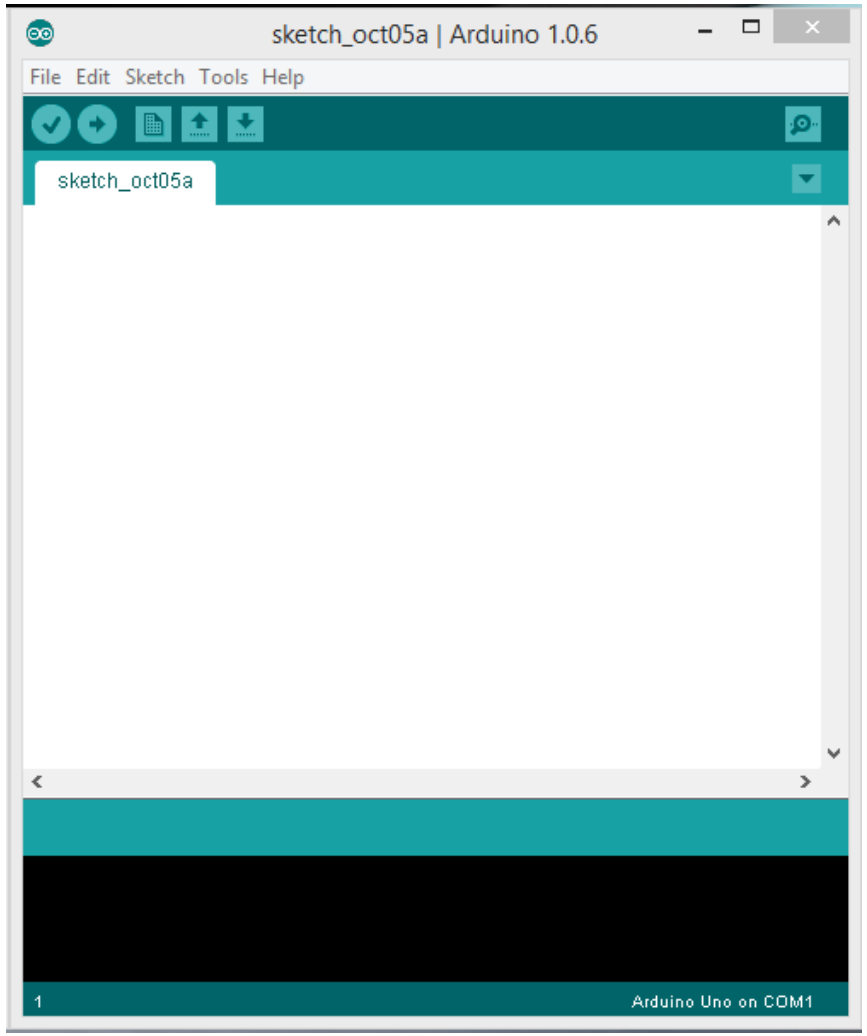
- Uma das vantagens que tornou o Arduino popular é sua IDE (Integrated Development Environment) e deve ser baixada do site: https://www.arduino.cc/en/software;
- A IDE possuir diversos elementos. O maior e mais visível é a área de **editor de texto**, parte branca, no qual o código é digitado.
- Logo abaixo, encontra-se o console, em preto. No console serão mostradas **mensagens de erro de compilação e durante a execução**.



Arduino IDE

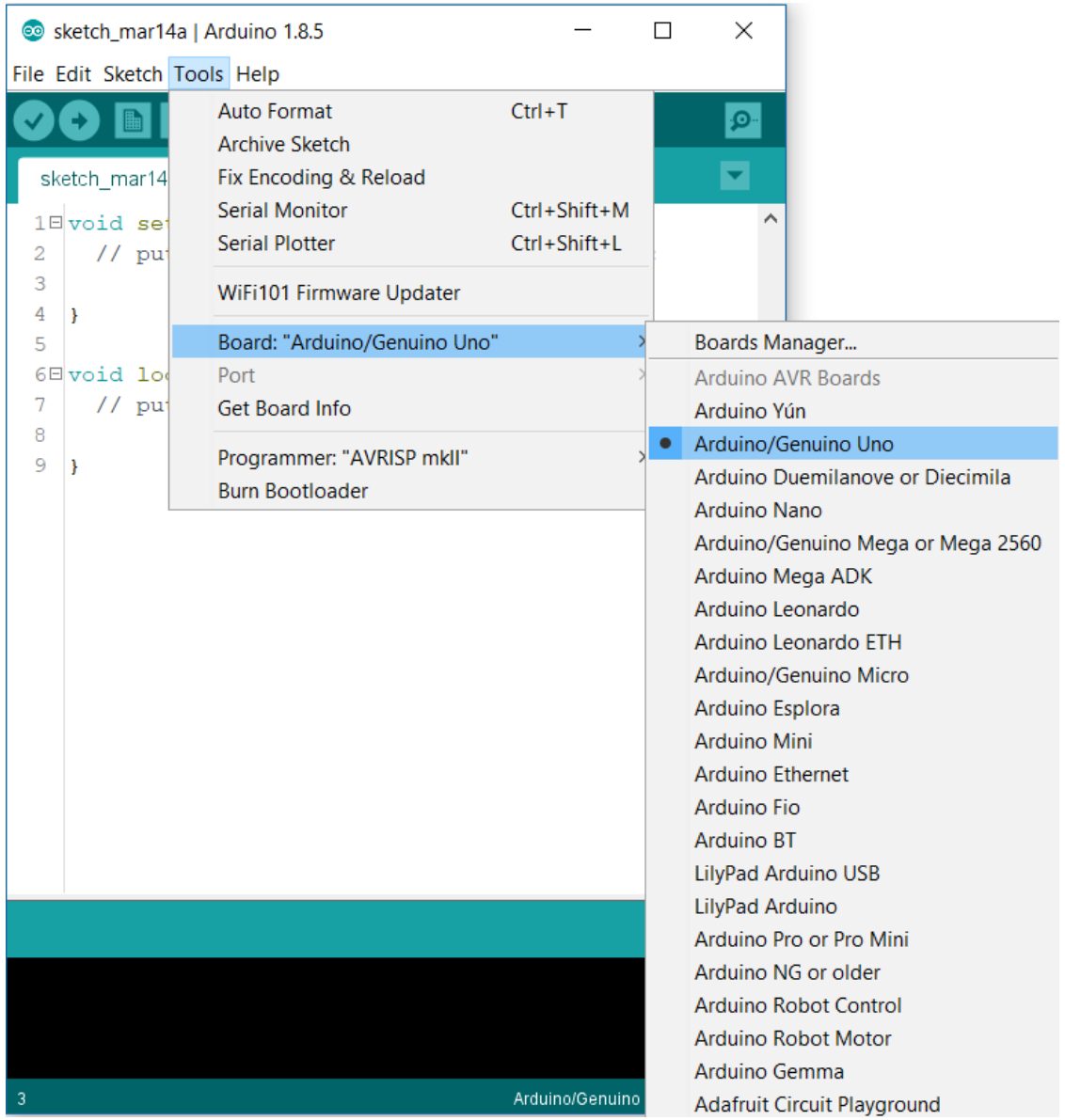
As funções dos botões de comando são:

- verify;
- upload
- new;
- open;
- save;
- serial monitor.



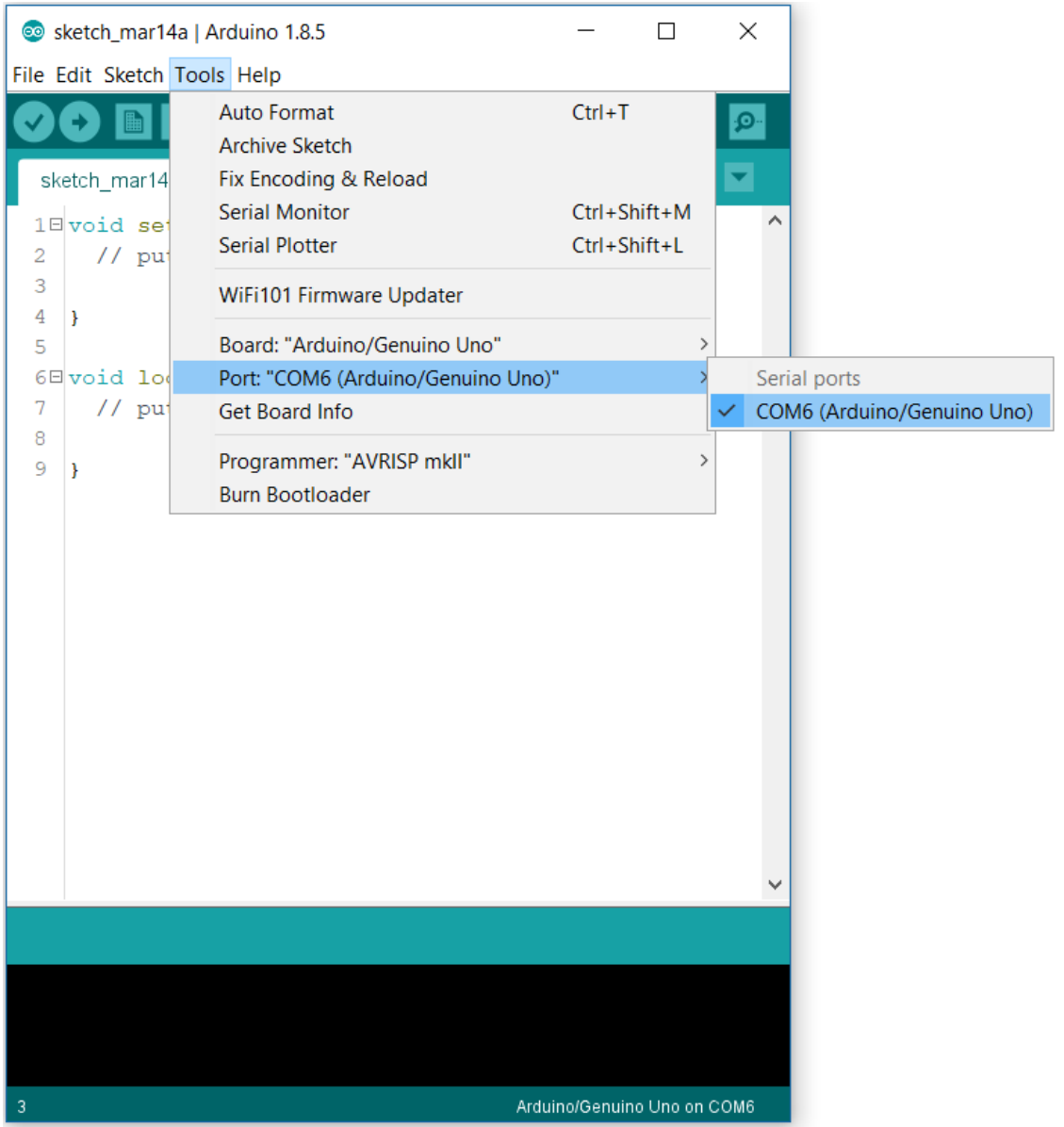
Arduino IDE – selecionando a placa correta

- Para que a IDE consiga transformar seu código em um conjunto de instruções chamada *firmware* corretamente, a mesma precisa saber qual microcontrolador está sendo utilizado;
- Para selecionar o microcontrolador, devemos marcar o modelo correto de Arduino.
- Clique em *Tools*, depois em *board* e selecione *Arduino/Genuino Uno*.



Arduino IDE – selecionando a porta de comunicação COM

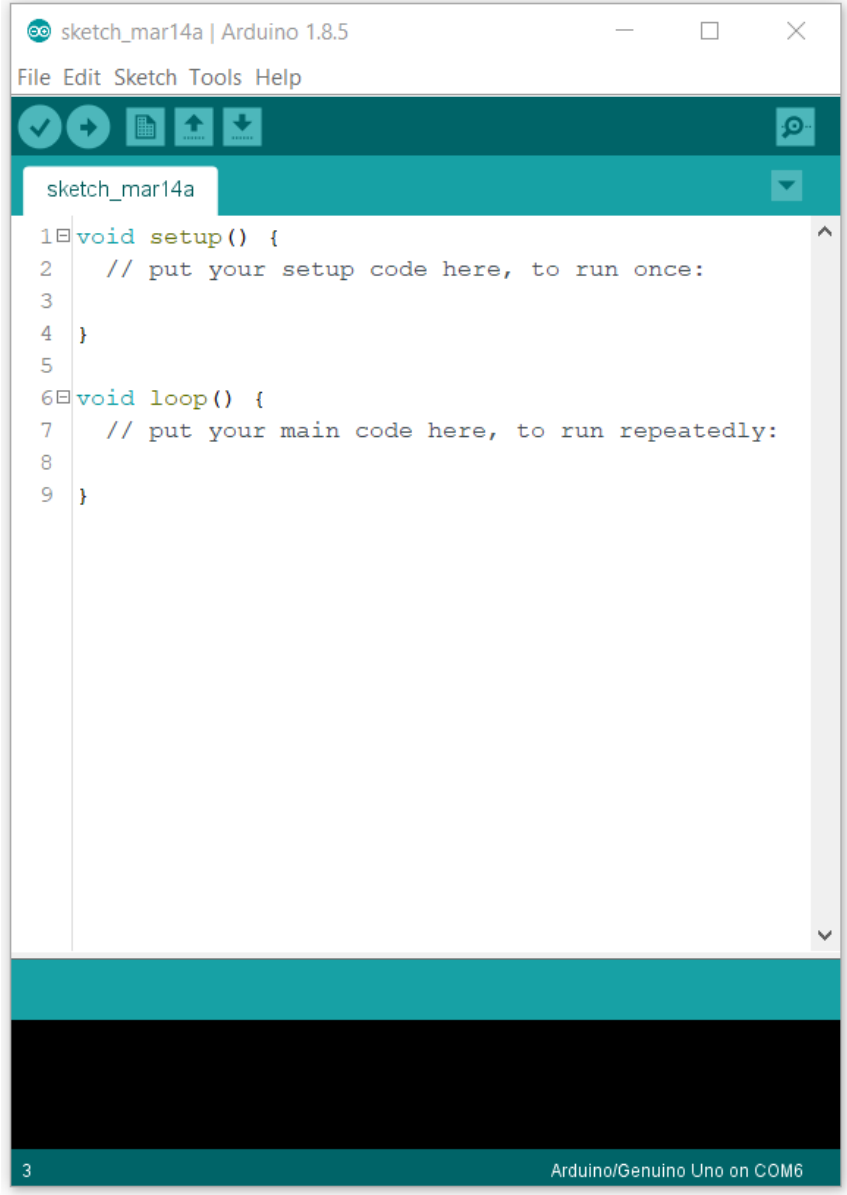
- Como você pode ter conectado mais de um Arduino ao seu computador, temos que informar à IDE em qual porta de comunicação COM está o Arduino Uno que desejamos usar;
- Clique em *Tools*, depois em *Port* e selecione a COM que estiver indicado o *Arduino/Genuino Uno*.



Arduino IDE – estrutura do sketch

- O código de um programa em Arduino é chamado de sketch. A estrutura básica de um sketch é dividida em 3 partes: o escopo de variáveis, a função **setup()** e a função **loop()**;

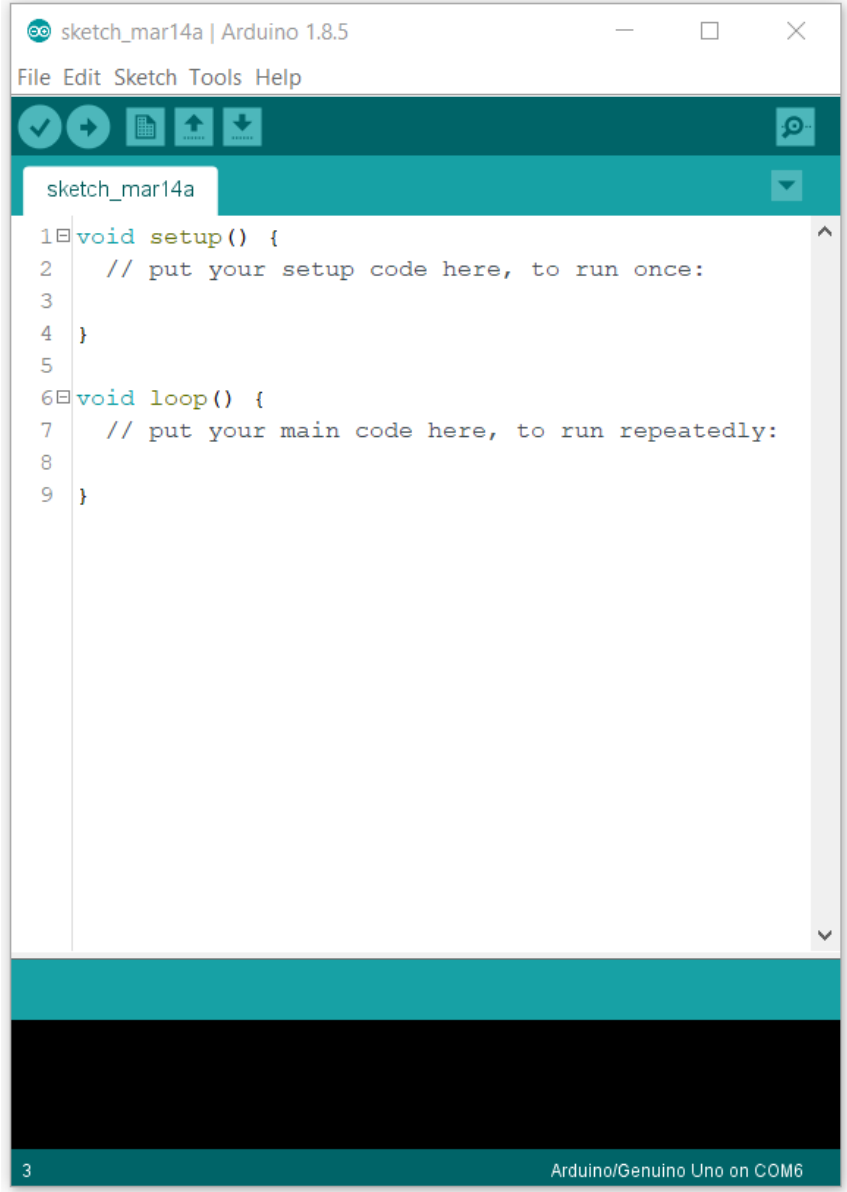
- Escopo de variáveis: É a parte na qual são declaradas as **variáveis globais** do sketch, acessíveis de qualquer função do seu código, assim como as bibliotecas utilizadas. Os tipos de dados comumente utilizados são: byte, int, float, double, boolean e char.



Arduino IDE – estrutura do sketch

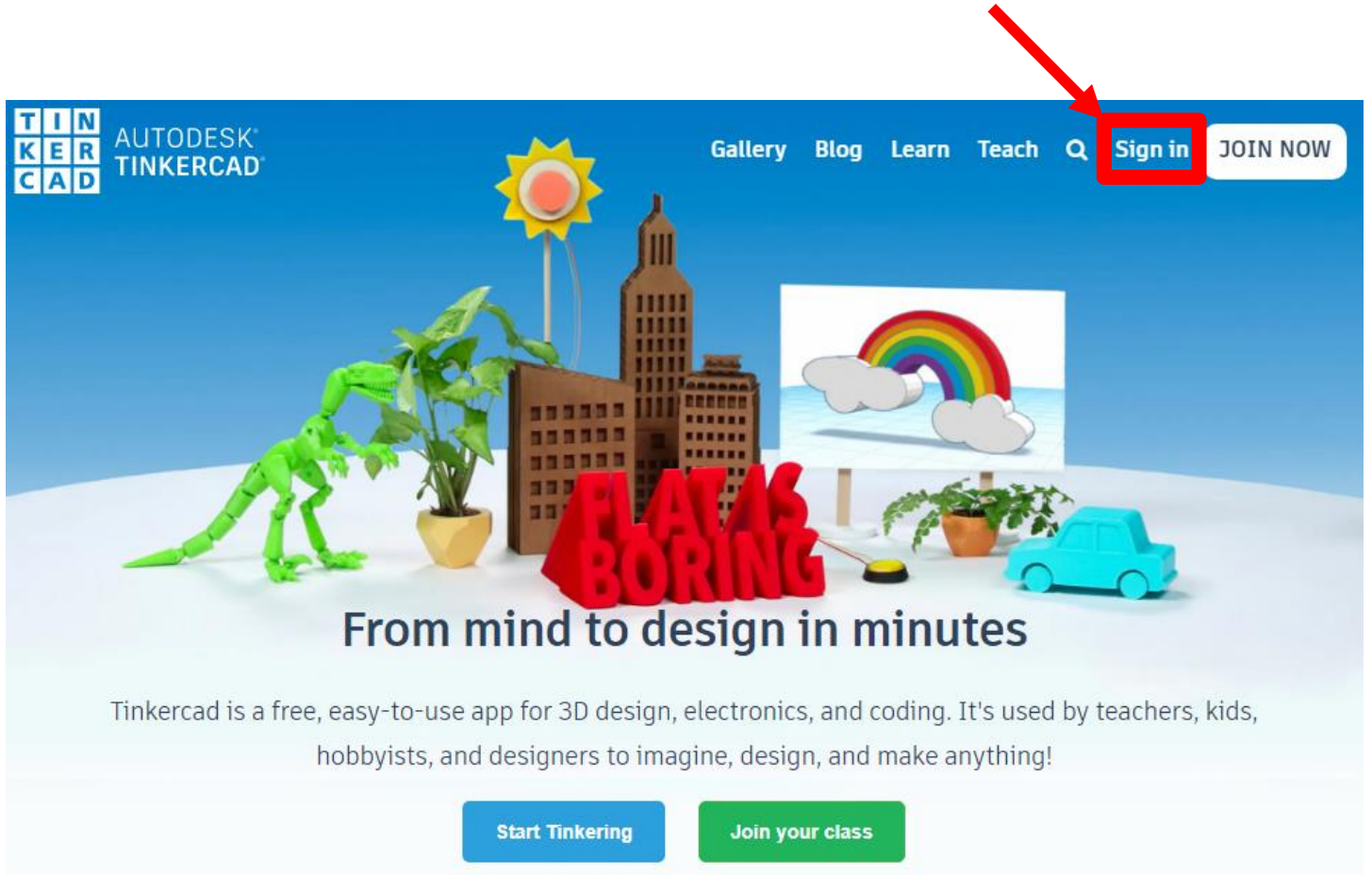
- Função **setup()**: é chamada uma vez apenas na execução do programa, normalmente no início da execução. É onde se especifica, por exemplo, se um pino é de entrada ou saída e são inicializadas algumas variáveis.

- **loop()**: a função será executada em loop até que o programa seja terminado.

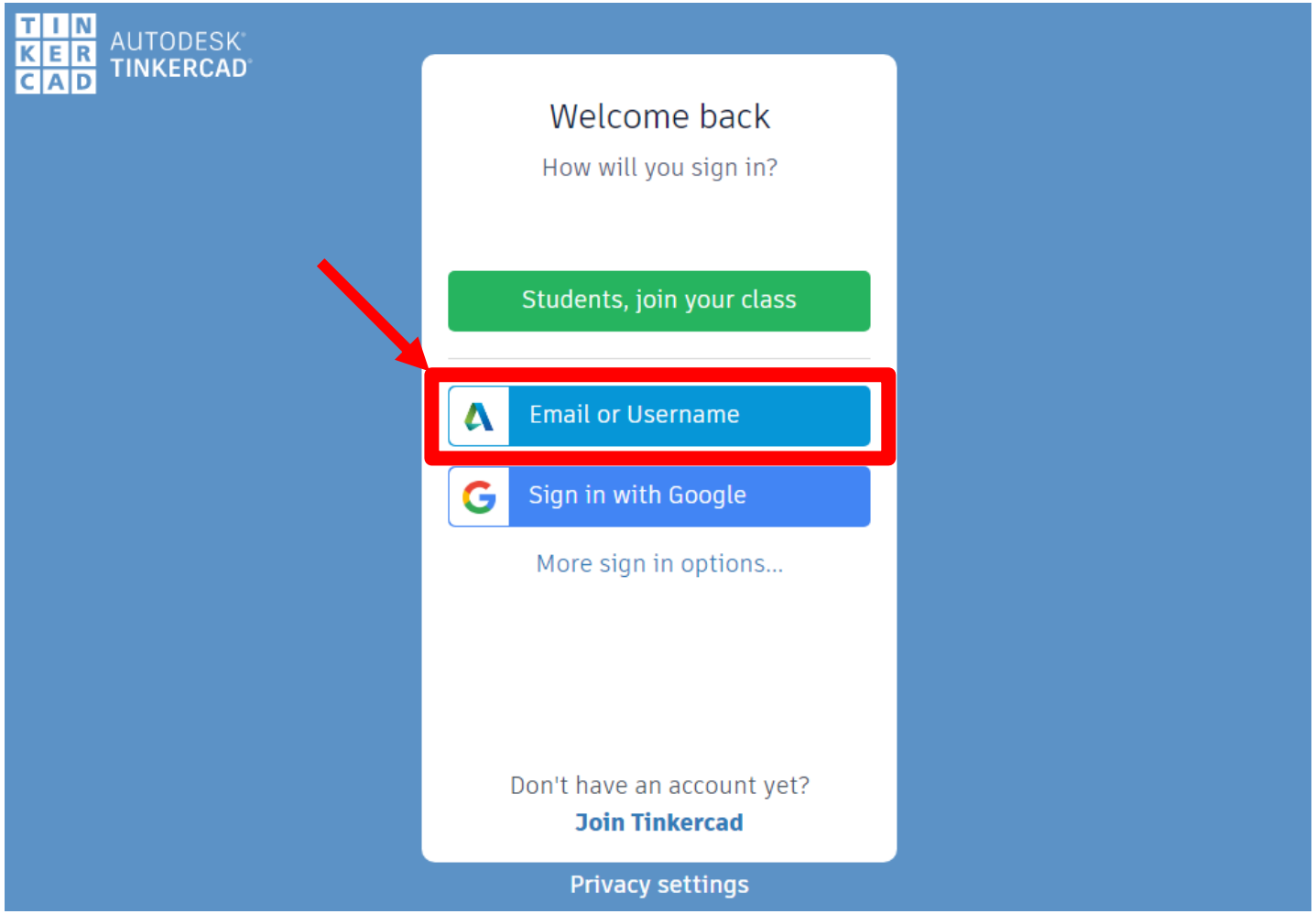


TinkerCAD

Entre em https://www.tinkercad.com/ e clique em **Sign in**, conforme figura ao lado.

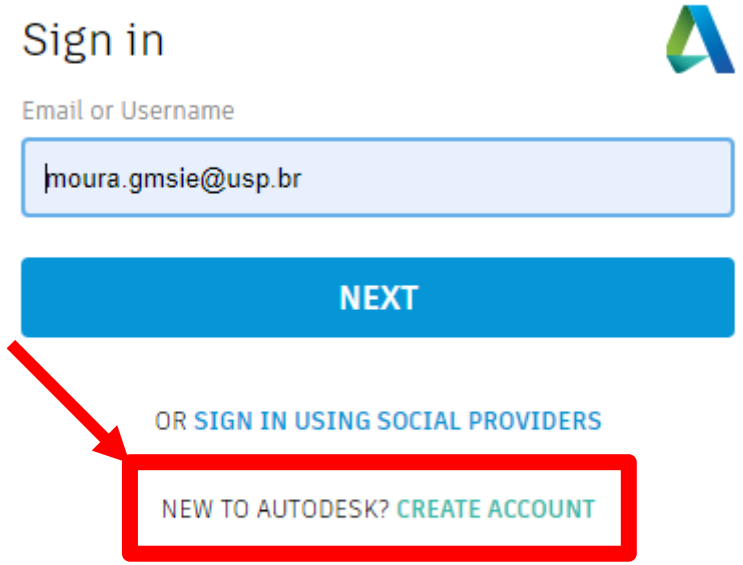


Clique em **Email or Username**, conforme figura ao lado.

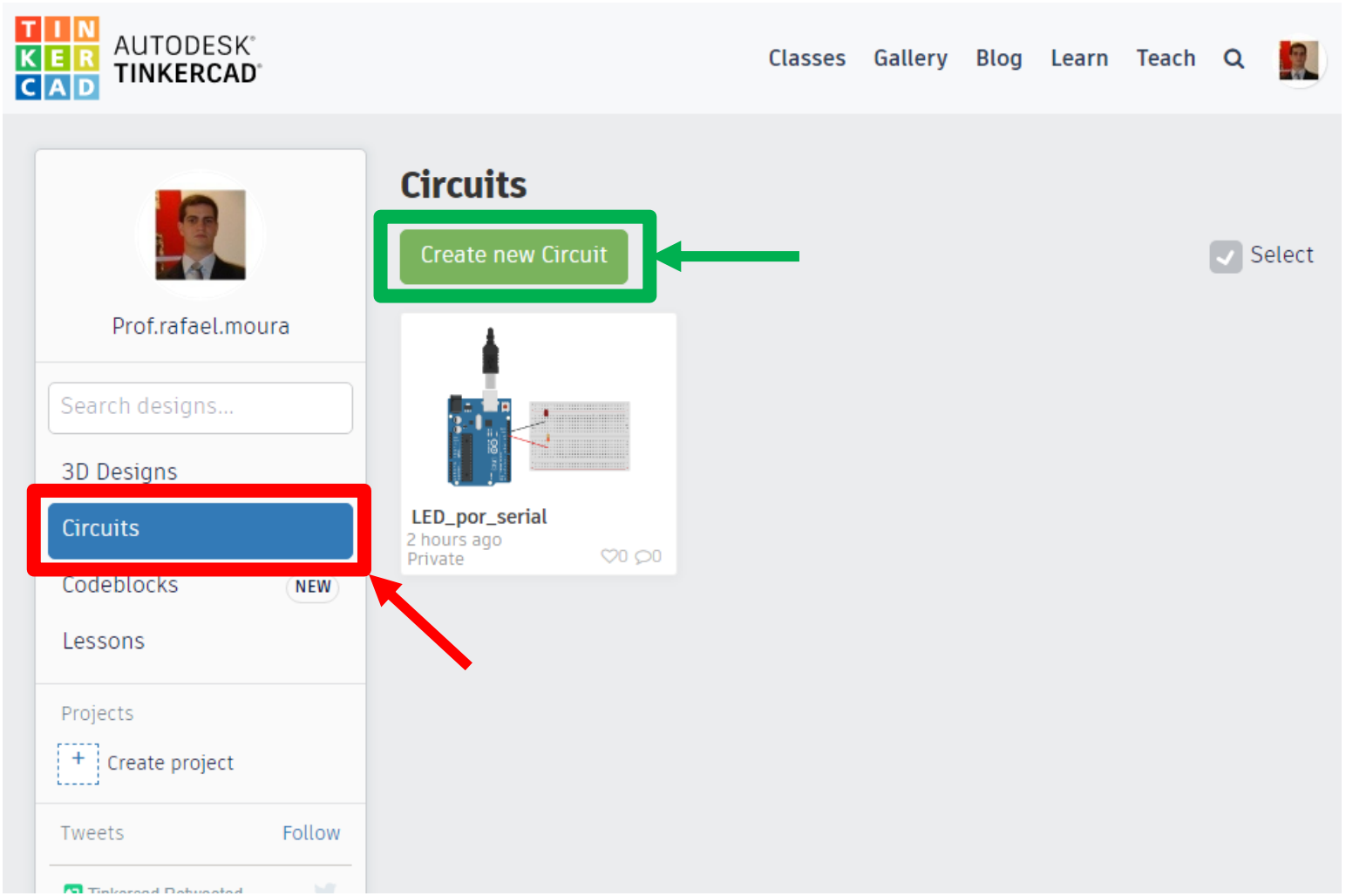


Caso já tenha feito uma conta, faça o login normalmente.

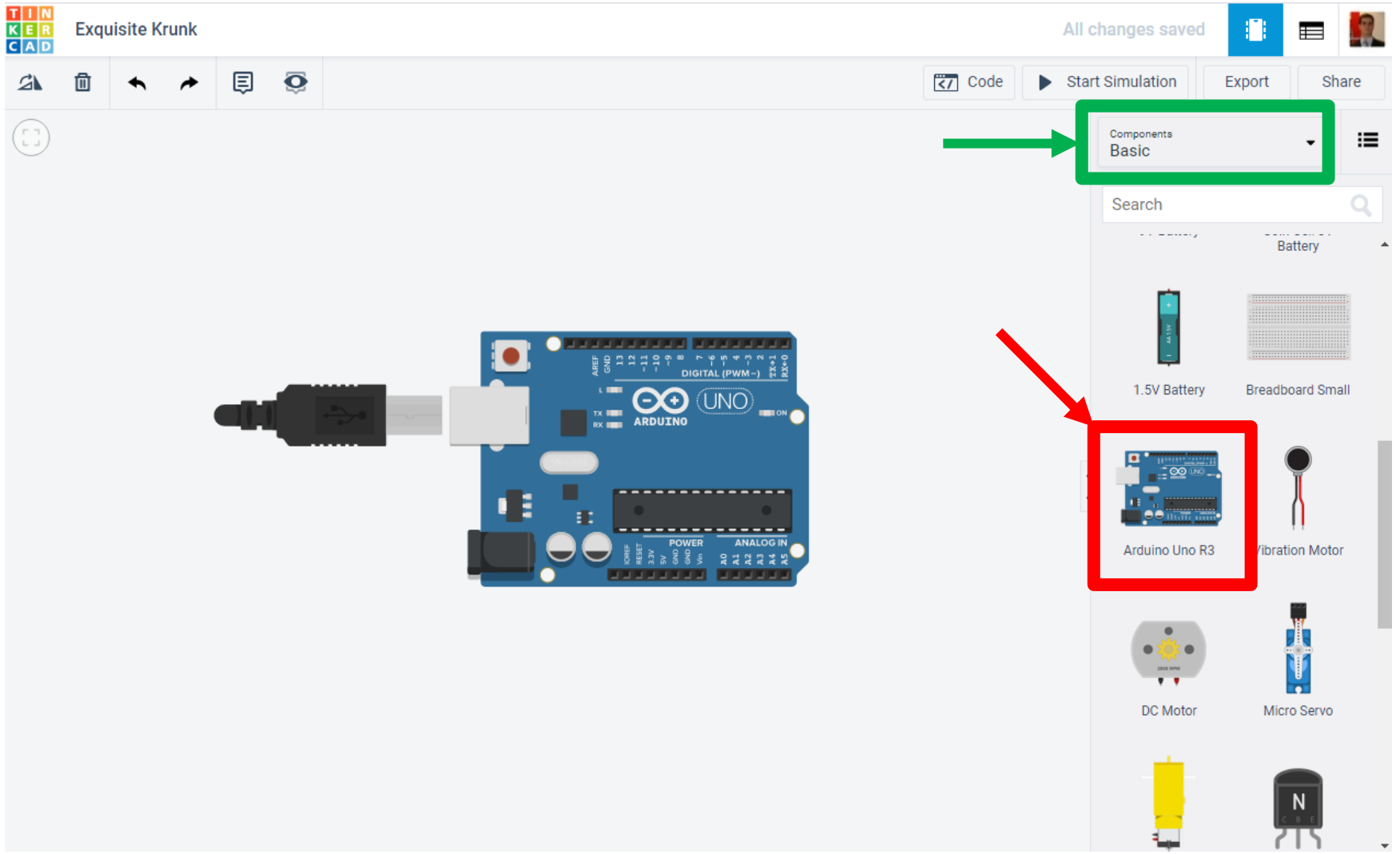
Caso não tenha criado uma conta antes, clique em **Create Account**, conforme figura ao lado. Então refaça os passos anteriores e faça o Login.



Uma vez logado, clicar em **Circuits** e em **Create new Circuit**.

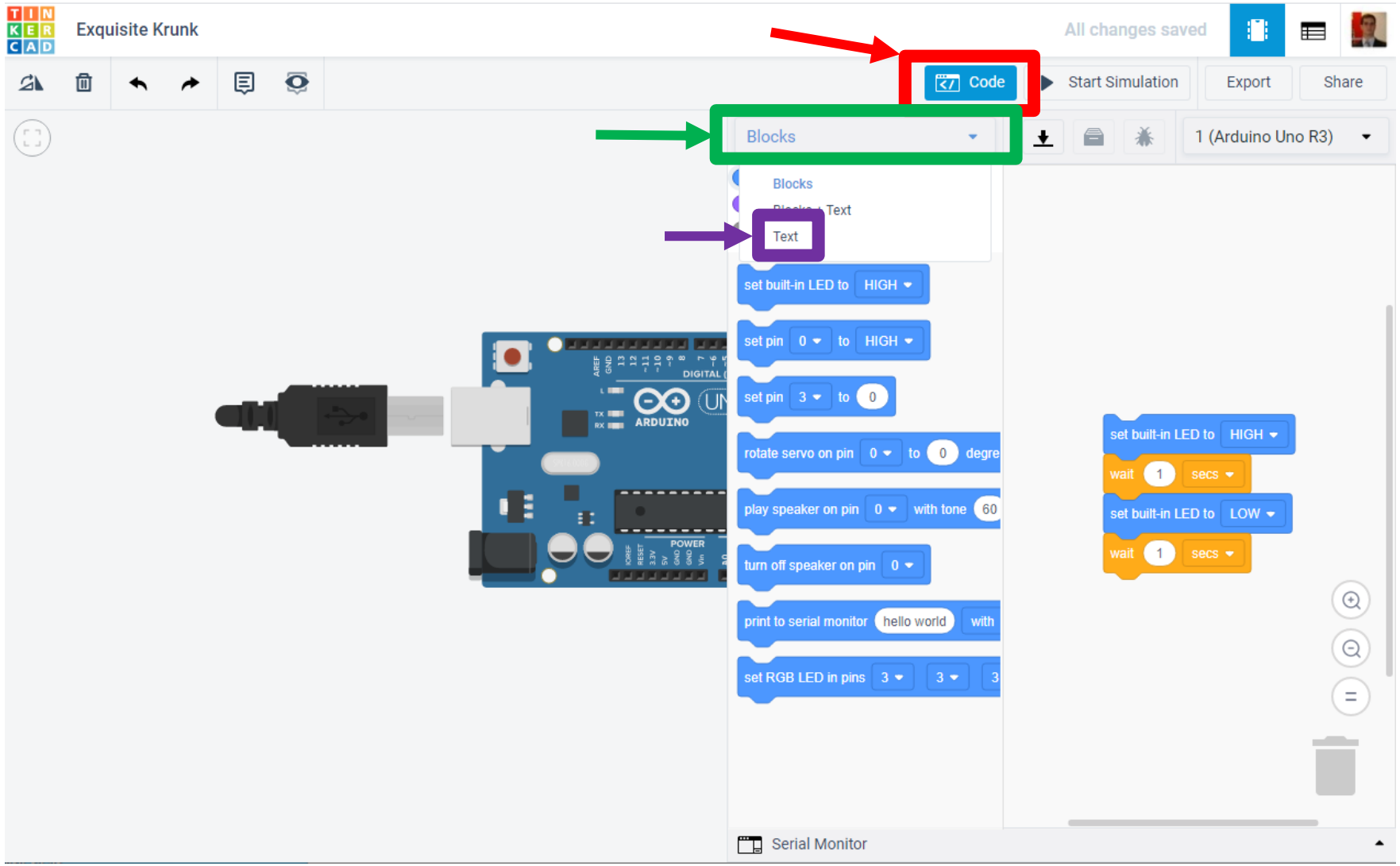


Arraste um **Arduino Uno R3** dos itens de **Components Basics** para a área na esquerda.



Clique em **Code**, depois em **Blocks** e selecione **Text**.

Surgirá um pop-up. Nele, selecione **Continue**.



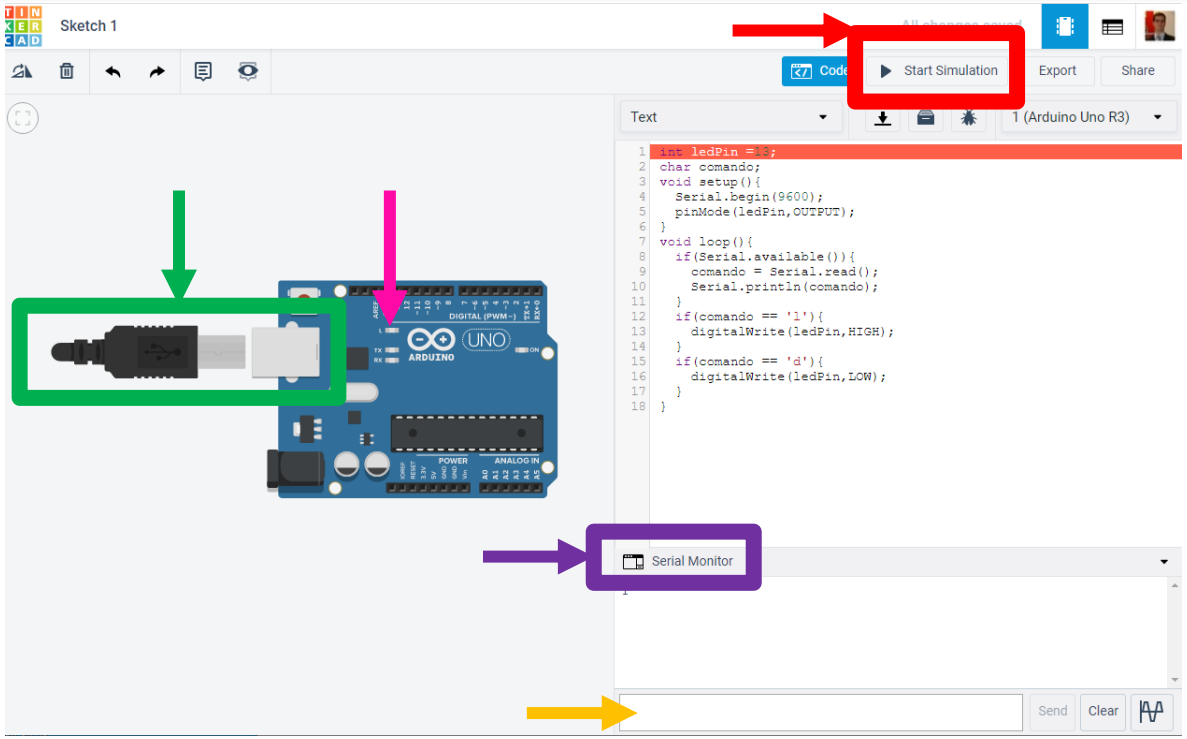
Exemplo 01

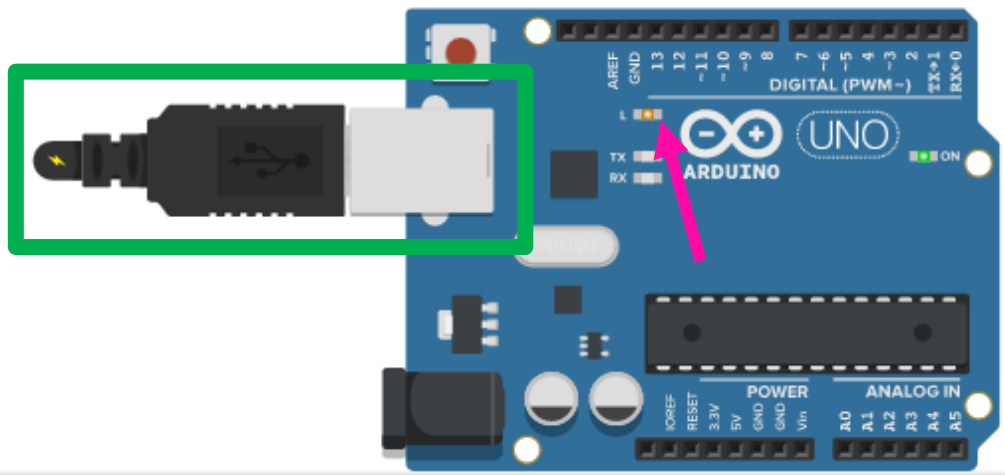
Digite o código e clique em **Start Simulation.**

Note que a **USB** é conectada ao Arduino e o botão se torna **Stop Simulation**.

Clique então em **Serial Monitor**. **Na parte mais abaixo**, clique, digite ***l*** (de liga), de enter e veja o **LED** do pino 13 acender. Digite ***d*** (de desliga), de enter e veja o **LED** do pino 13 apagar.

```
int ledPin =13;
char comando;
void setup() {
  Serial.begin(9600);
  pinMode(ledPin,OUTPUT);
}
void loop() {
    if(Serial.available()){
    comando = Serial.read();
    Serial.println(comando);
  }
  if(comando == 'l'){
    digitalWrite(ledPin,HIGH);
  }
  if(comando == 'd'){
    digitalWrite(ledPin,LOW);
  }
}
```



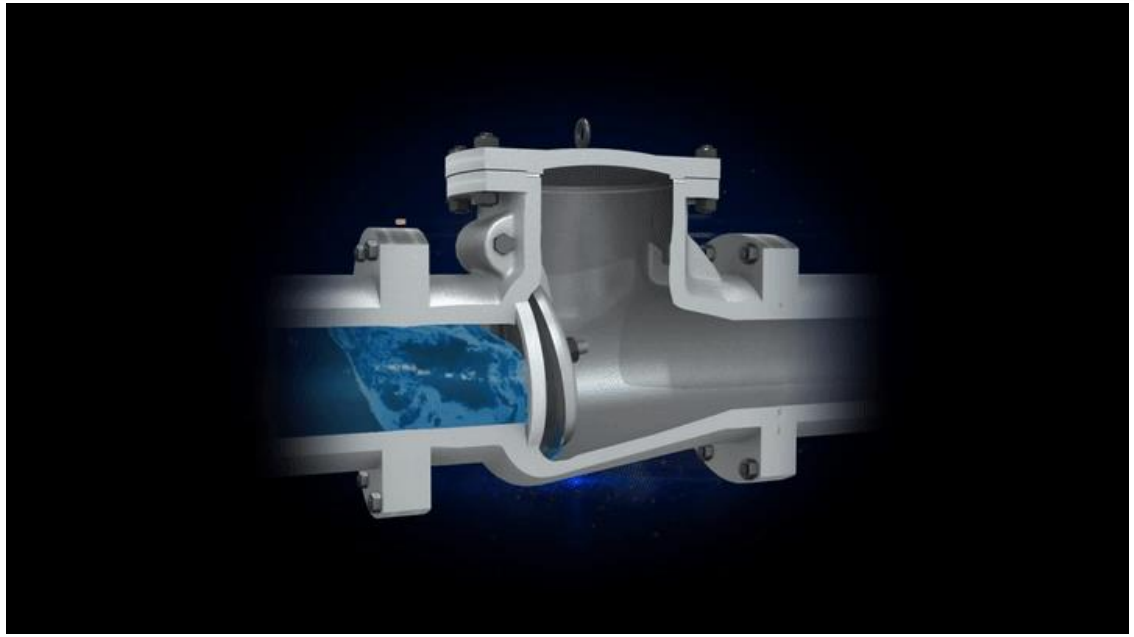


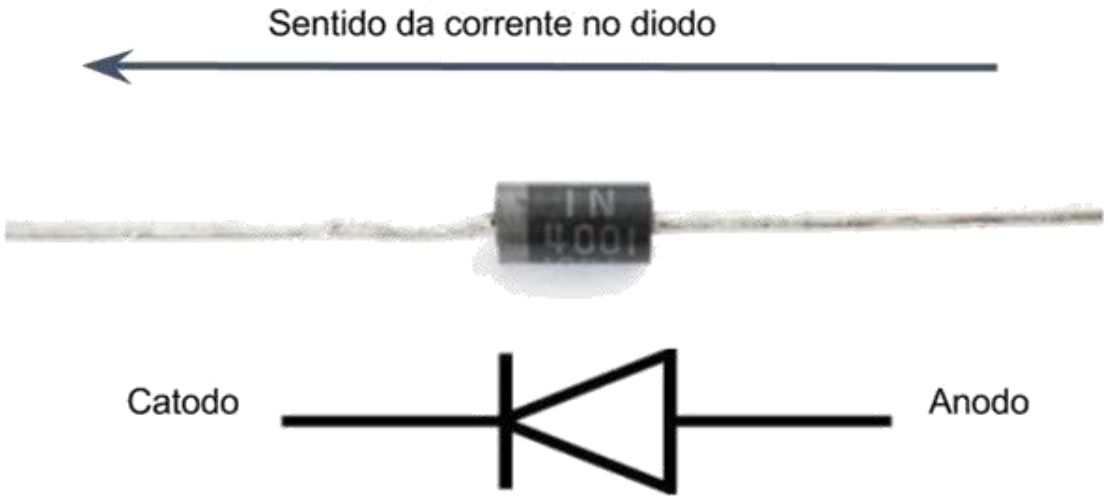
TinkerCAD – Comunicação Console

- Para que o Arduino seja capaz de executar a comunicação serial, esta deve ser inicializada dentro de `setup()`, através de `Serial.begin(9600)`, com 9600 sendo a velocidade da conexão em bits por segundo (baud rate).
- `Serial.available()` Retorna a quantidades de bytes disponíveis para leitura no **buffer** de leitura. Essa função auxilia em loops onde a leitura dos dados só e realizada quando há dados disponível. A quantidade máxima de bytes no buffer é 64.
- `pinMode()` Configura um pino especifico para se comportar tanto como entrada ou saída.
- `Serial.read()` Lê o byte mais recente apontado no buffer de entrada da serial.
- `Serial.print()` Escreve na serial texto em formato ASCII.
- `Serial.println()` Como a função Serial.print(), a diferença é que esta função acrescenta ao fim da mensagem o caractere de retorno de carro e o caractere de nova linha.
- `digitalWrite()` Escreve um valor HIGH ou um LOW em um pino digital. Se o pino foi configurado como uma saída (output), sua voltagem será determinada como 5V (ou 3.3V nas placas de 3.3V) para HIGH, 0V (terra) para LOW.

Eletrônica Básica – Diodo

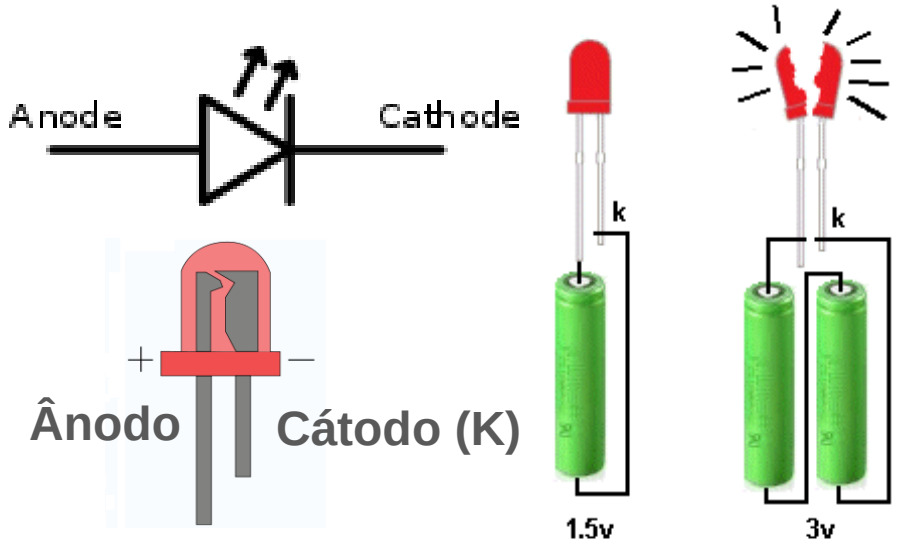
- O diodo é um componente eletrônico que permite corrente em uma direção, mas a impede de fluir na outra direção.
- No diodo, existe uma queda de voltagem constante. O mesmo possui uma potência máxima, e, consequentemente, uma corrente máxima. O encapsulamento do diodo está diretamente relacionado com a capacidade de dissipação térmica e também a corrente máxima.
- Desta forma, podemos colocar um resistor em série com o diodo para reduzir a corrente.

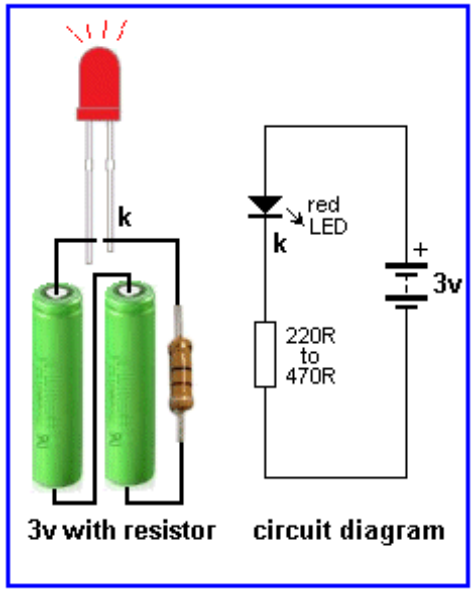




Eletrônica Básica – LED

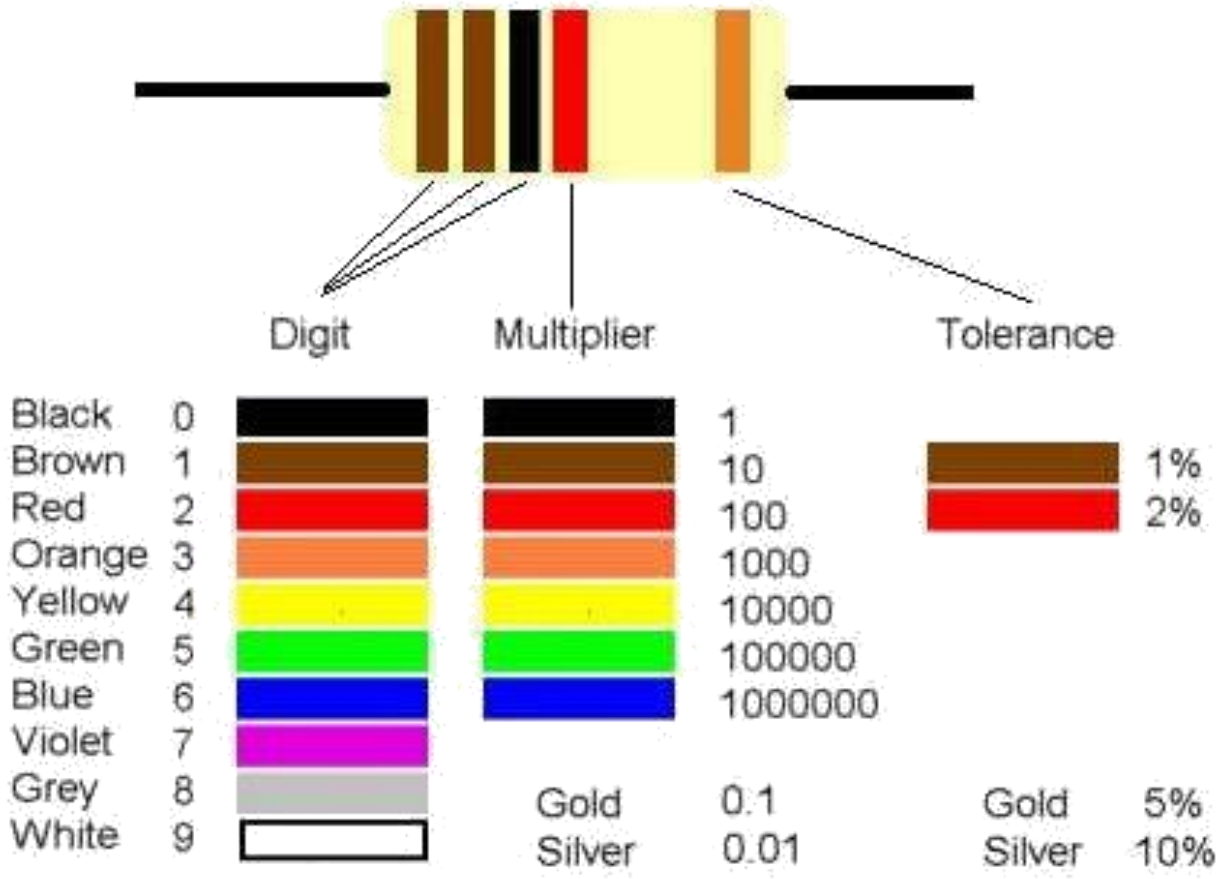
- LED significa Light Emitting Diode. Ou seja, ele é um diodo que emite uma luz quando há passagem de corrente. Assim como nos diodos, podemos considerar que há uma queda de voltagem constante no LED, independente da corrente.
- O LED possui uma potência máxima suportada. Acima, o mesmo queima. Como a queda de voltagem é constante, ele possui uma corrente máxima. Desta forma, basta colocar um resistor em série para baixar a corrente!
- Se ligar sem resistor numa fonte de tensão, ele vai acender por algum tempo e depois queimar!





Eletrônica Básica – Resistores

- Existem diversos tipos de resistores, sempre com o valor de sua resistência descrita por faixas ao redor do mesmo.
- Uma dessas faixas é responsável por informar a tolerância do valor nominal da resistência.
- Assim como no diodo, o encapsulamento está diretamente relacionado com a capacidade de dissipação térmica e, consequentemente, com a potência e conseguinte corrente máximas.

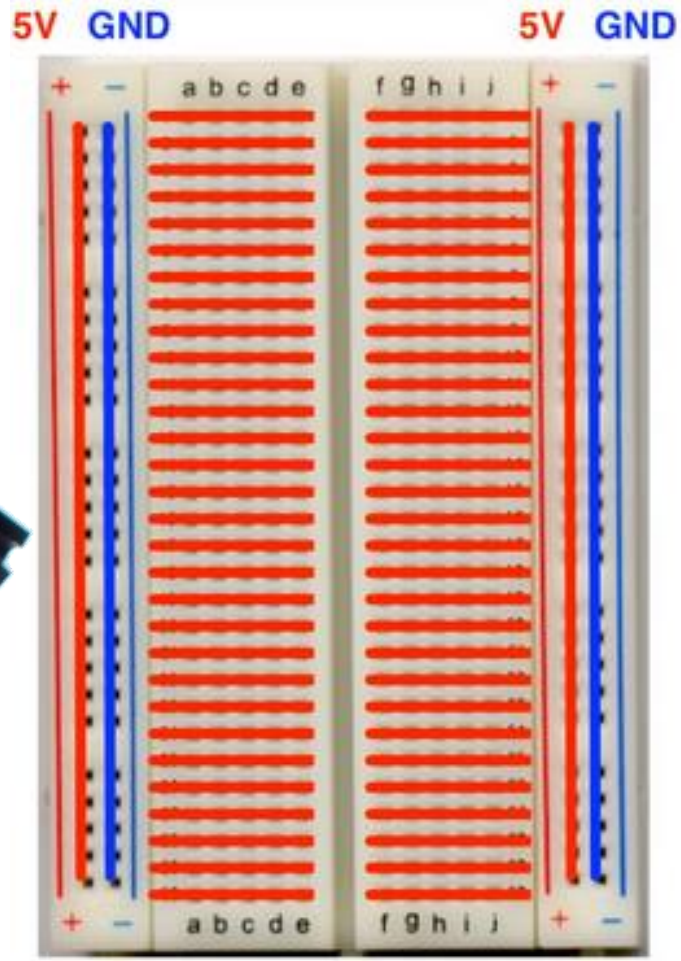


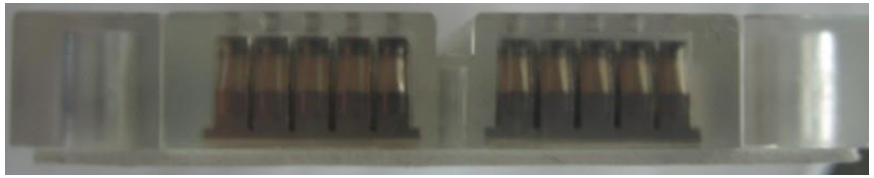
Eletrônica Básica – Calculado resistores em série com LEDs

- Cada LED de cor diferente tem uma queda de potencial diferente.
- Se temos uma bateria de 9V e um LED vermelho com queda de potencial de 2V, teremos uma voltagem de 7V na resistência. Como sabemos a corrente máxima de 20mA, basta dividir a voltagem de 7V pela corrente de 20mA para obtermos uma resistência de 350 Ω;
- Para LEDs verde, amarelo e laranja temos uma corrente de i = 20 a 25 mA e uma voltagem V = 2,1 Volts. Para LEDs azul e branco temos uma corrente de i = 15 a 30 mA e uma voltagem V = 2,8 a 4,2 Volts;
- Uma regra prática é sempre usar um resistor entre 470 Ω e 1k Ω.

Eletrônica Básica – Protoboard (breadboard)

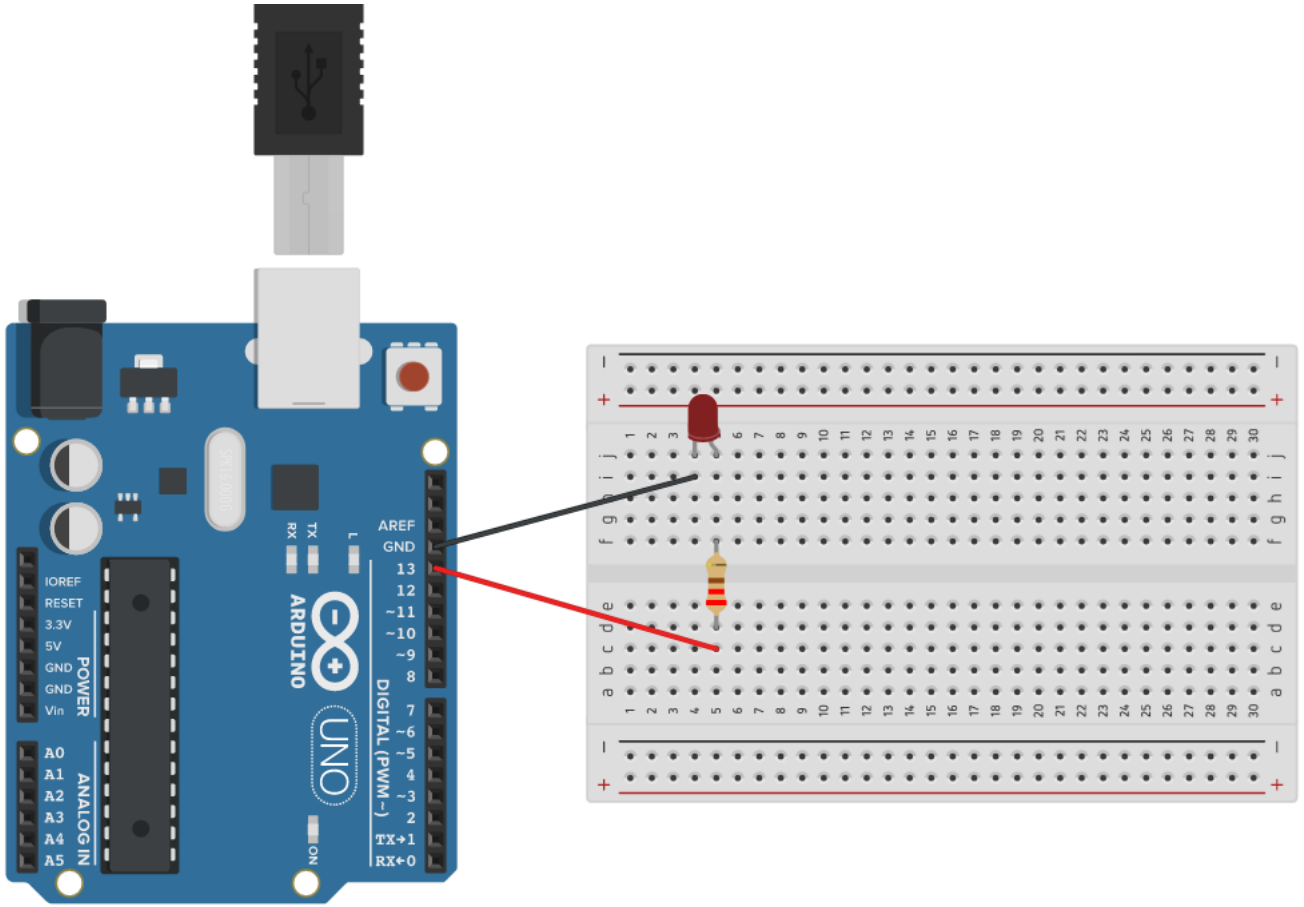
- Mas como podemos interligar os componentes de maneira que seja prático alterar e testar circuitos, ou seja, sem soldar os componentes?
- Para tal utilizamos tanto a protoboard como cabos jumper. A protoboard tem furos conectados entre si conforme a imagem ao lado.
- Já os cabos jumper servem para facilitar a utilização da protoboard, com pontas de conexão *normalizadas*, evitando mal contato.





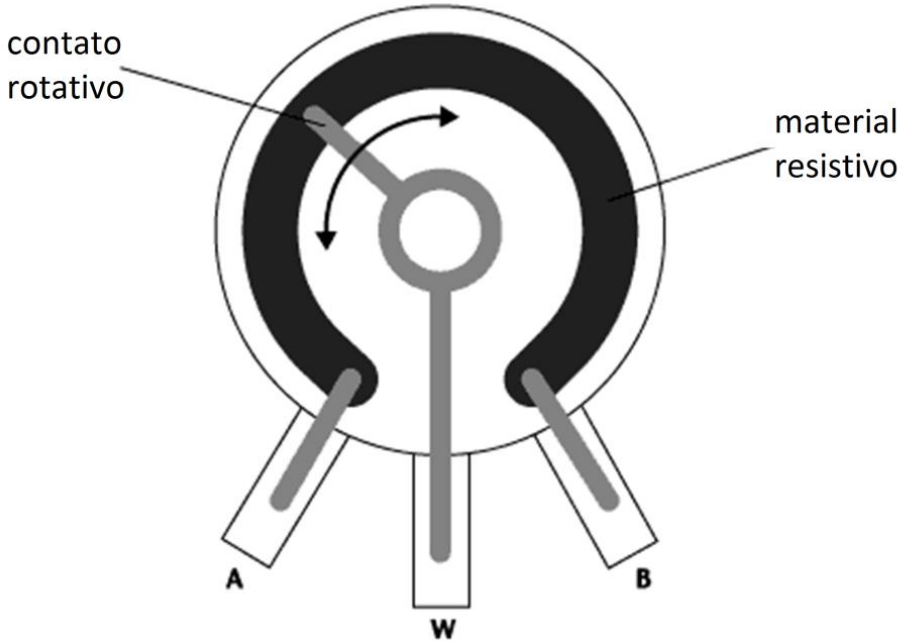
Exemplo 02

- Monta o circuito ao lado com um LED vermelho em série com um resistor de 220 Ohm;
- Teste o mesmo programa carregado anteriormente no arduino.



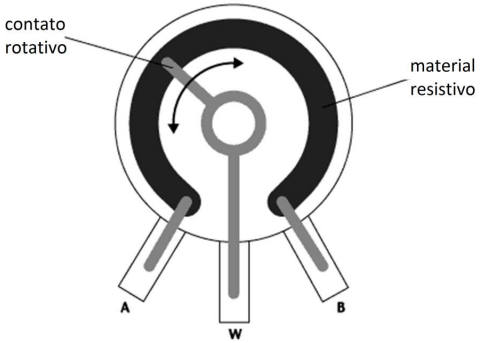
Eletrônica Básica – Potenciômetro

- Um potenciômetro é um simples botão giratório que fornece uma resistência variável e que pode ser lida pelo Arduino como um valor analógico.
- Possui internamente uma trilha resistiva (de níquel-cromo ou de carbono), sobre a qual desliza um cursor , que altera a resistência elétrica entre seu conector central e um dos dois laterais (normalmente são três conectores).
- Para lermos o valor a posição do potenciômetro no Arduino, usamos um conversor analogico-digital. No arduino, como o ADC é de 10 bits, temos 1024 níveis lógicos: de 0 a 1023.



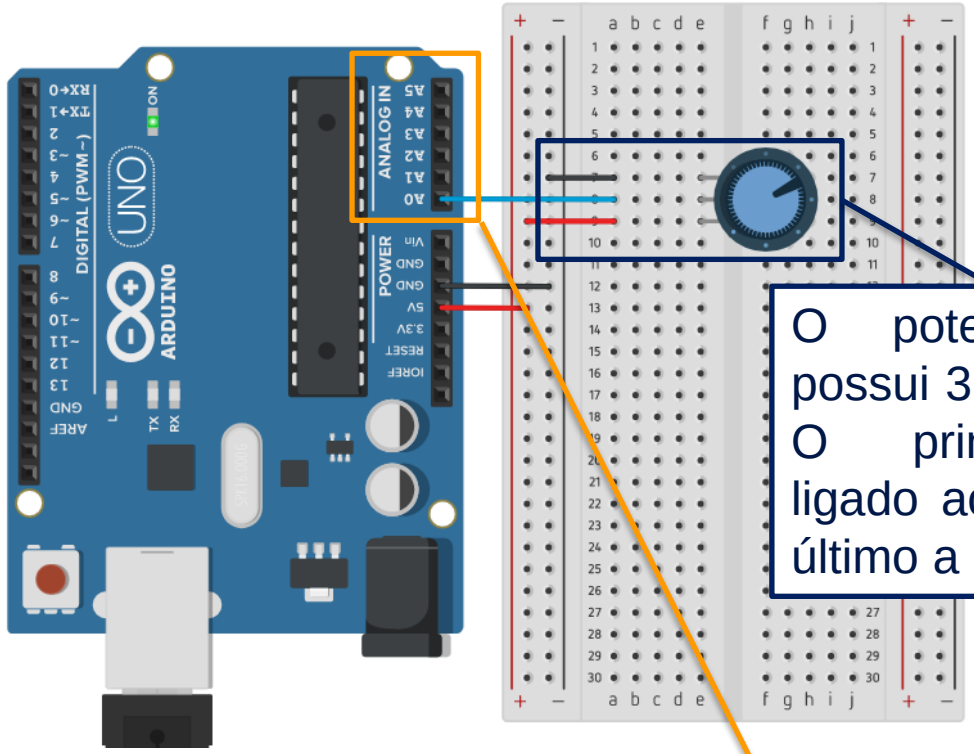
Eletrônica Básica – Potenciômetro

- Se girarmos o cursor do potenciômetro, alteramos a resistência em cada lado do contato elétrico que vai conectado ao terminal central do botão. Isso provoca a mudança na proximidade do terminal central aos 5 volts ou ao terra, o que implica numa mudança no valor analógico de entrada. Quando o cursor for levado até o final da escala, teremos por exemplo zero volts a ser fornecido ao pino de entrada do Arduino e, assim, ao lê-lo obteremos 0. Quando giramos o cursor até o outro extremo da escala, haverá 5 volts a ser fornecido ao pino do Arduino e, ao lê-lo, teremos 1023. Em qualquer posição intermediária do cursor, teremos um valor entre 0 e 1023, que será proporcional à tensão elétrica sendo aplicada ao pino do Arduino.



Exemplo 03

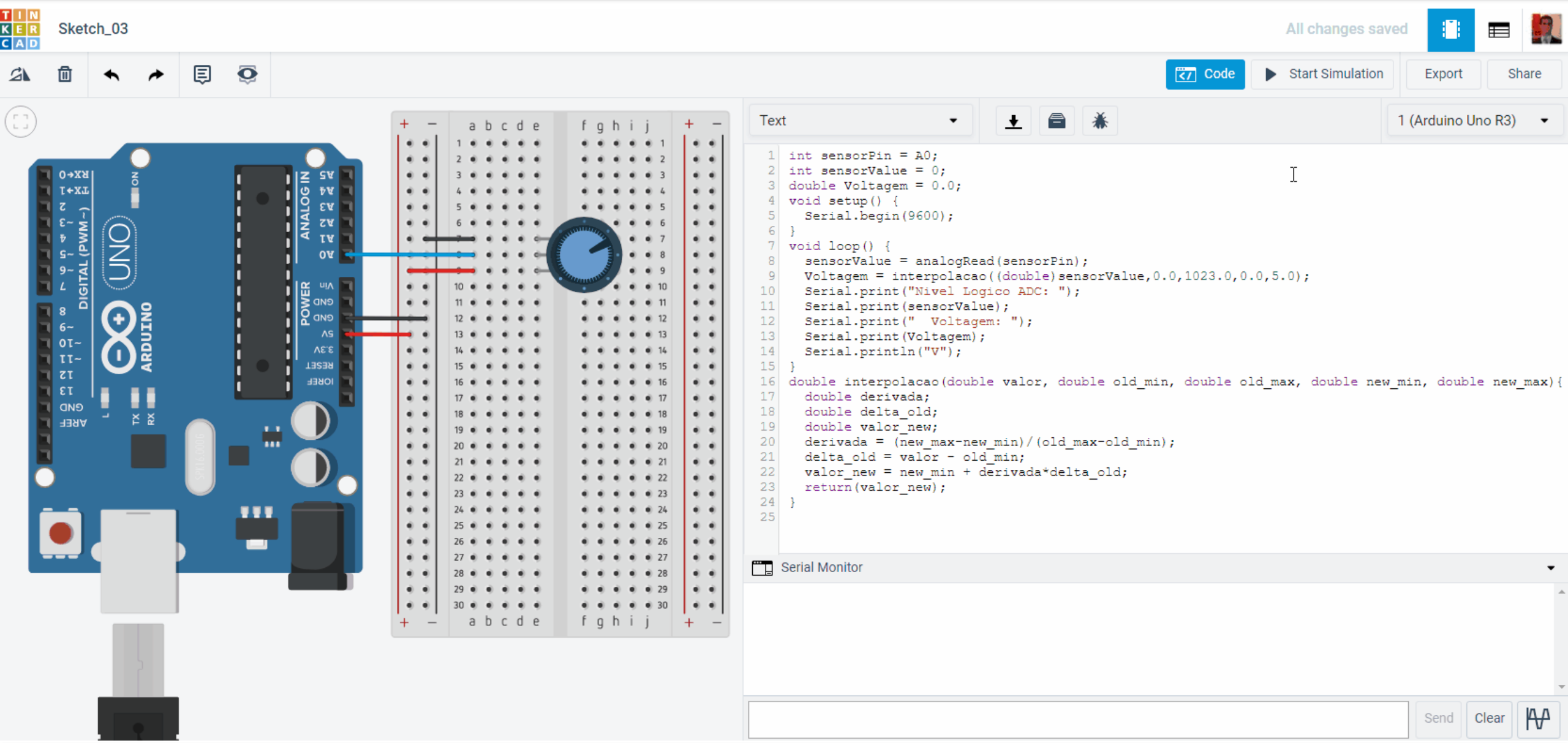
```
int sensorPin = A0;
int sensorValue = 0;
double Voltagem = 0.0;
 void setup() {
 Serial.begin(9600);
}
void loop() {
  sensorValue = analogRead(sensorPin);
  Voltagem = interpolacao((double)sensorValue,0.0,1023.0,0.0,5.0);
 Serial.print("Nivel Logico ADC: ");
 Serial.print(sensorValue);
 Serial.print("  Voltagem: ");
 Serial.print(Voltagem);
 Serial.println("V");
}
double interpolacao(double valor, double old_min, double old_max, double new_min, double new_max){
  double derivada;
  double delta_old;
  double valor_new;
  derivada = (new_max-new_min)/(old_max-old_min);
  delta_old = valor - old_min;
  valor_new = new_min + derivada*delta_old;
  return(valor_new);
}
```



O potenciômetro possui 3 terminais. O primeiro é ligado ao GND. O último a 5 volts.

O terminal central do potenciômetro deve ser ligado ao pino analógico, que pode assumir valores diversos, e não apenas 0 e 1 como o caso do pino digital. *Internamente, existe um conversor analógico-digital, que converte os 5V em 1024 níveis, indo de 0 a 1023.*

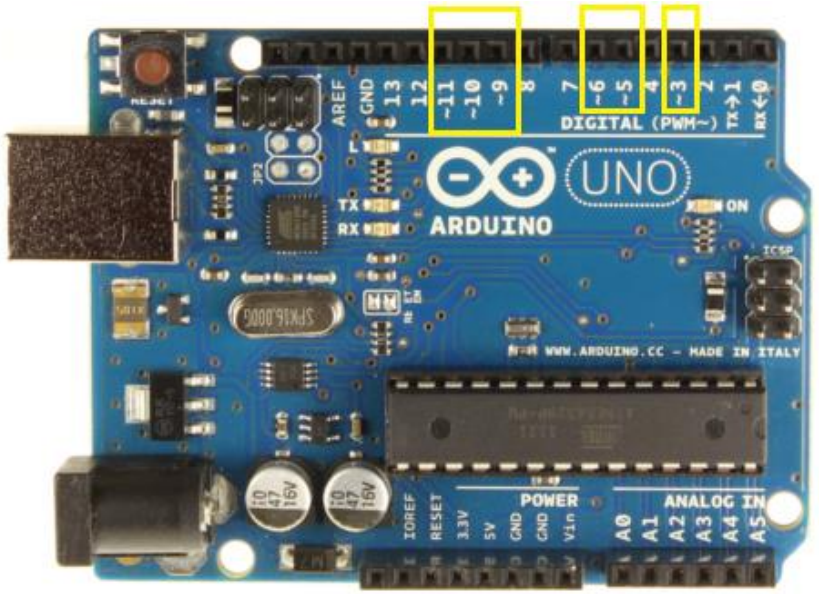
Exemplo 03


Sketch_03
All changes saved
Code
Start Simulation
Export
Share
Text
1 (Arduino Uno R3)
int sensorPin = A0;
int sensorValue = 0;
double Voltagem = 0.0;
void setup() {
Serial.begin(9600);
}
void loop() {
sensorValue = analogRead(sensorPin);
Voltagem = interpolacao((double)sensorValue,0.0,1023.0,0.0,5.0);
Serial.print("Nivel Logico ADC: ");
Serial.print(sensorValue);
Serial.print(" Voltagem: ");
Serial.print(Voltagem);
Serial.println("V");
}
double interpolacao(double valor, double old_min, double old_max, double new_min, double new_max){
double derivada;
double delta_old;
double valor_new;
derivada = (new_max-new_min)/(old_max-old_min);
delta_old = valor - old_min;
valor_new = new_min + derivada*delta_old;
return(valor_new);
}
Serial Monitor
Send
Clear

Acionamentos – PWM

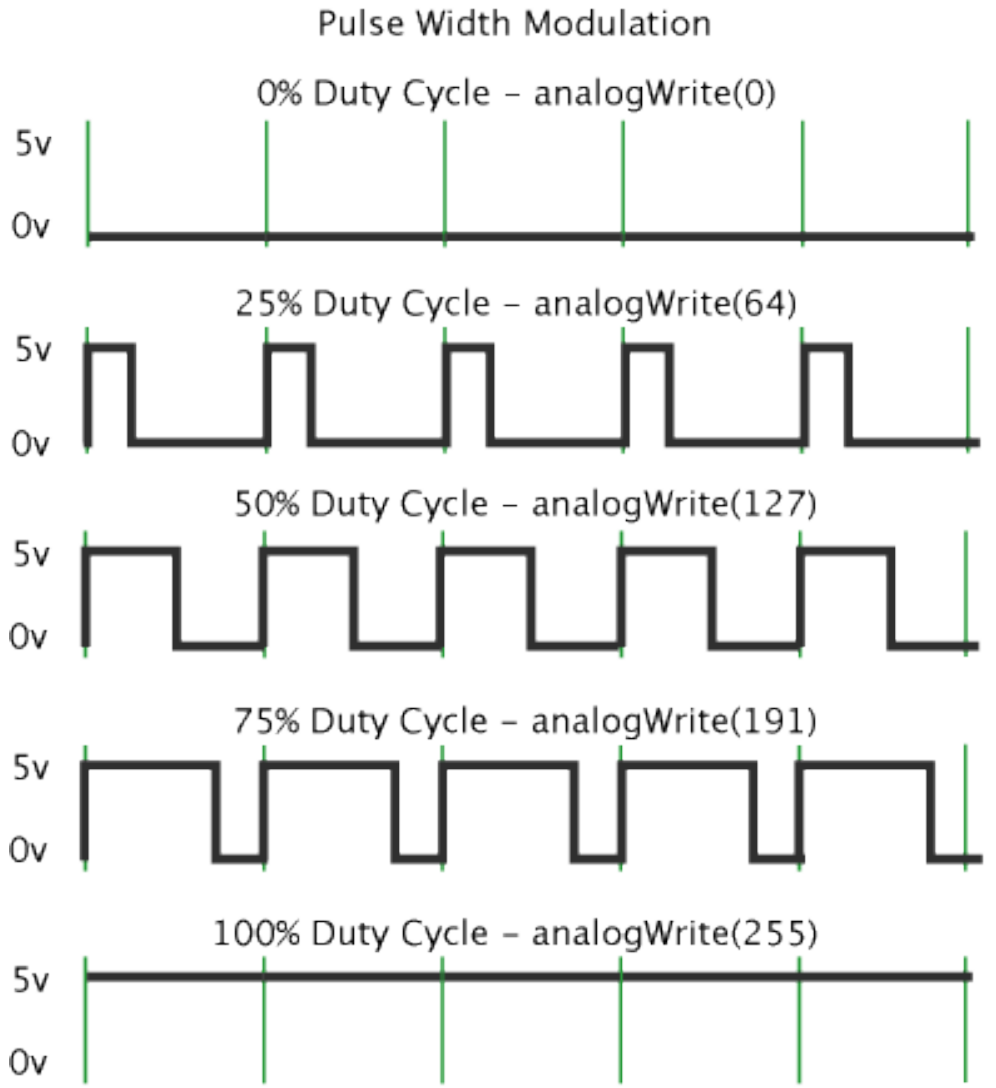
- PWM é a modulação da largura de ondas quadradas.
- PWM, do inglês Pulse Width Modulation, é uma técnica utilizada por sistemas digitais para variação do valor médio da voltagem, enquanto só se aplica na saída voltagens nula ou máxima. A técnica consiste em manter a frequência de uma onda quadrada fixa e variar o tempo que o sinal fica em nível lógico alto. Esse tempo é chamado de duty cycle.

A frequência do PWM no Arduino é:
- 1KHz nos pinos 5 e 6
- 500 Hz nos pinos 3,9,10,11



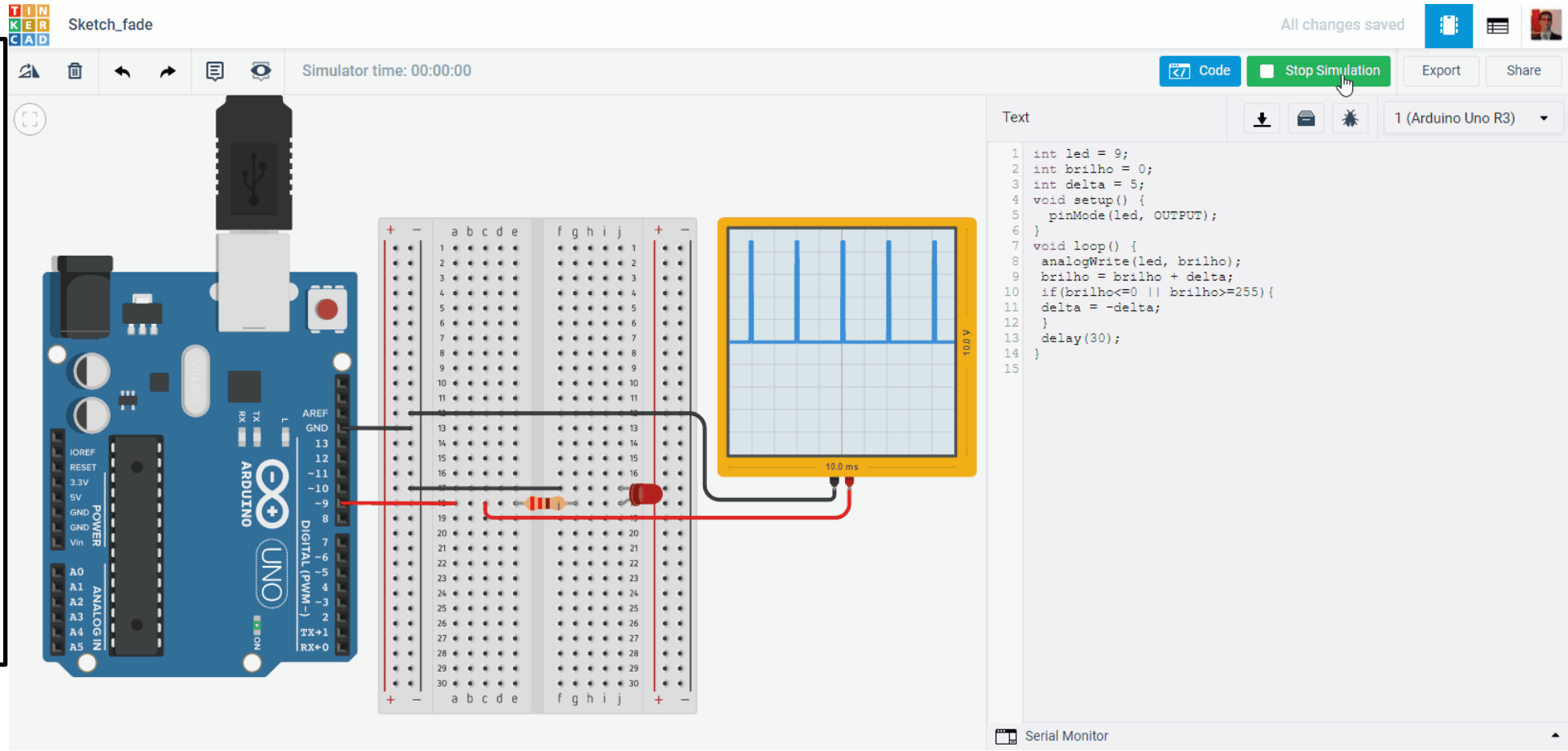
Acionamentos – PWM

- O gráfico ao lado ilusta a influência da variação do duty cycle no valor analógico médio.
- No Arduino, conforme explicado anteriormente, somente os pinos com ~ possuem PWM por Hardware, sendo controlados pela função `analogWrite(pino,valor)`, no qual o pino corresponde ao pino que será gerado o sinal PWM (3, 5, 6, 9, 10 ou 11) e o valor corresponde ao duty cycle, variando de 0 a 255. Se for 0, a saída permanece sempre em nível baixo (0V) e 255 a saída permanece sempre em nível alto (5V).

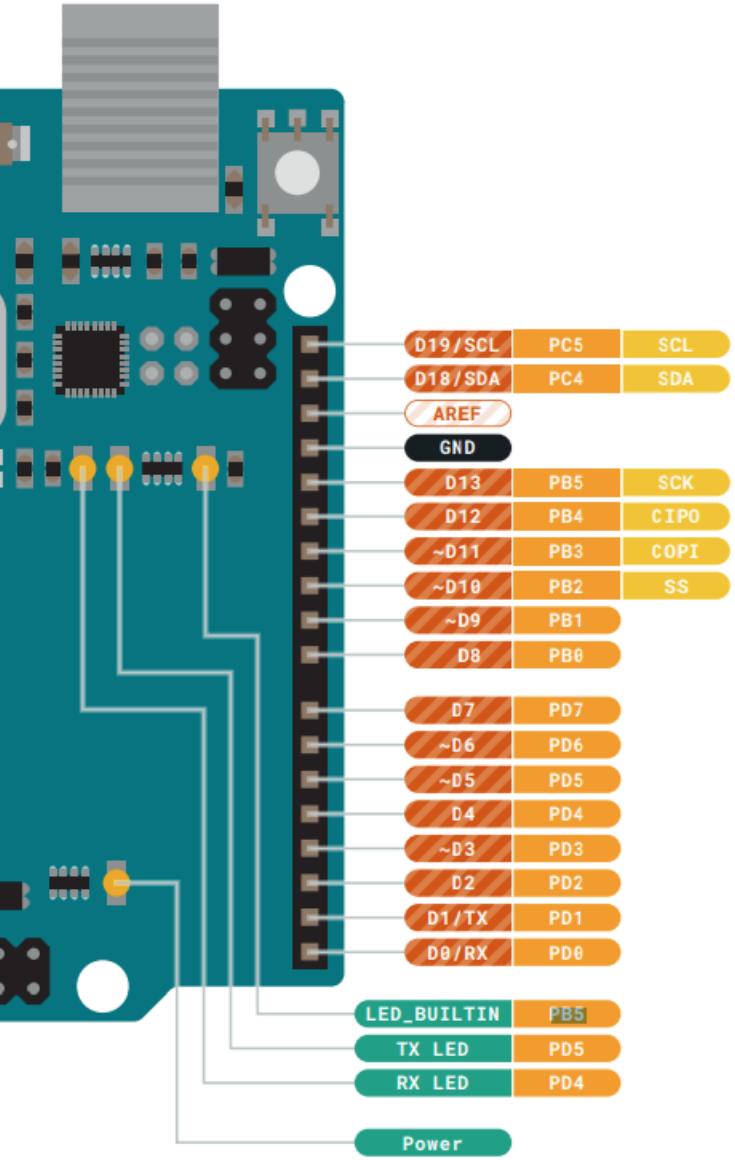


Exemplo 04

```
int led = 9;
int brilho = 0;
int delta = 5;
void setup() {
  pinMode(led, OUTPUT);
}
void loop() {
 analogWrite(led, brilho);
 brilho = brilho + delta;
 if(brilho<=0 || brilho>=255){
 delta = -delta;
 }
 delay(30);
}
```



Usando os registradores



14.2.1 Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in "Register Description" on page 100, the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.

The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

If PORTxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written logic zero or the pin has to be configured as an output pin. The port pins are tri-stated when reset condition becomes active, even if no clocks are running.

If PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

14.4.9 DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

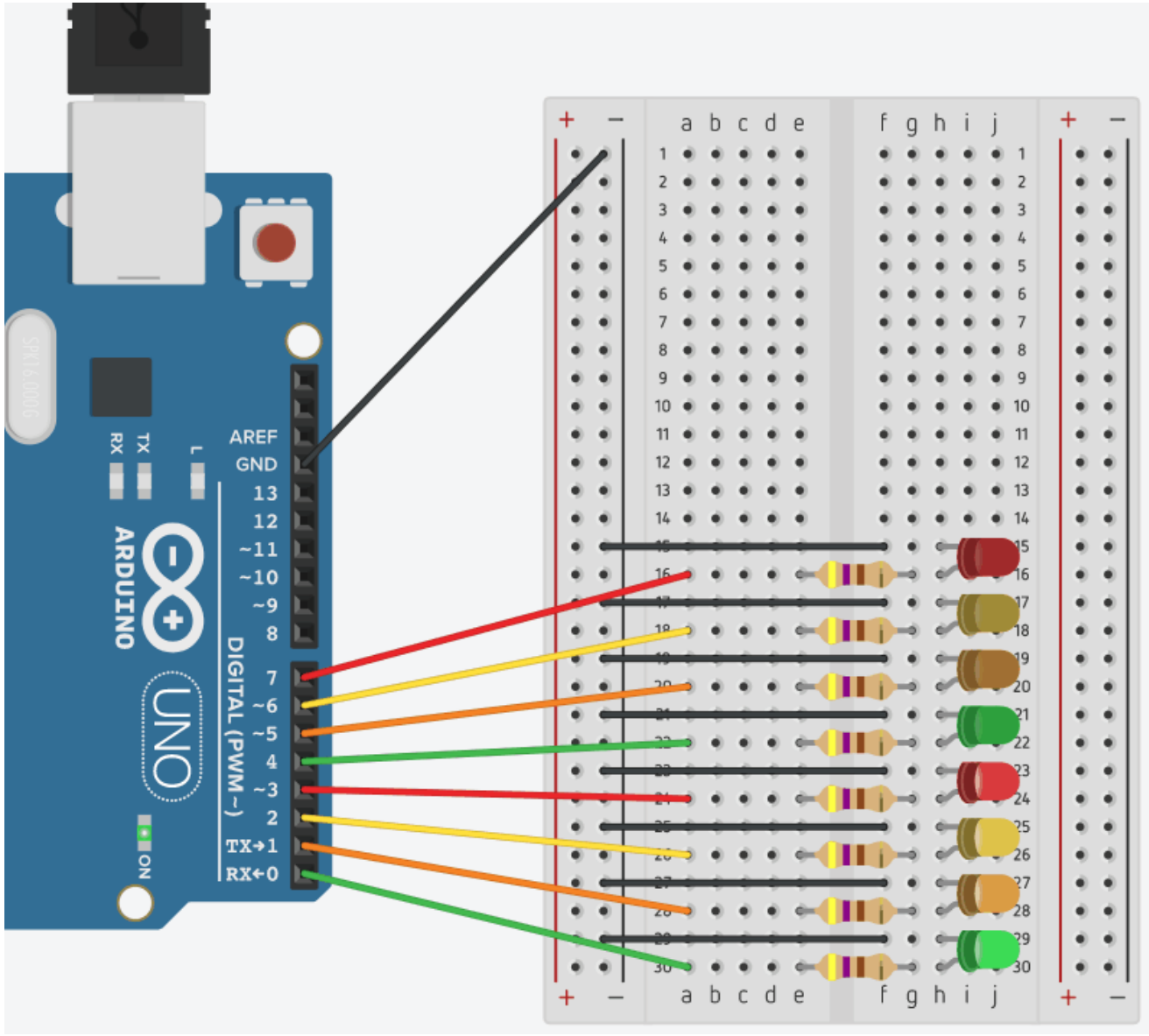
14.4.8 PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Exemplo 05

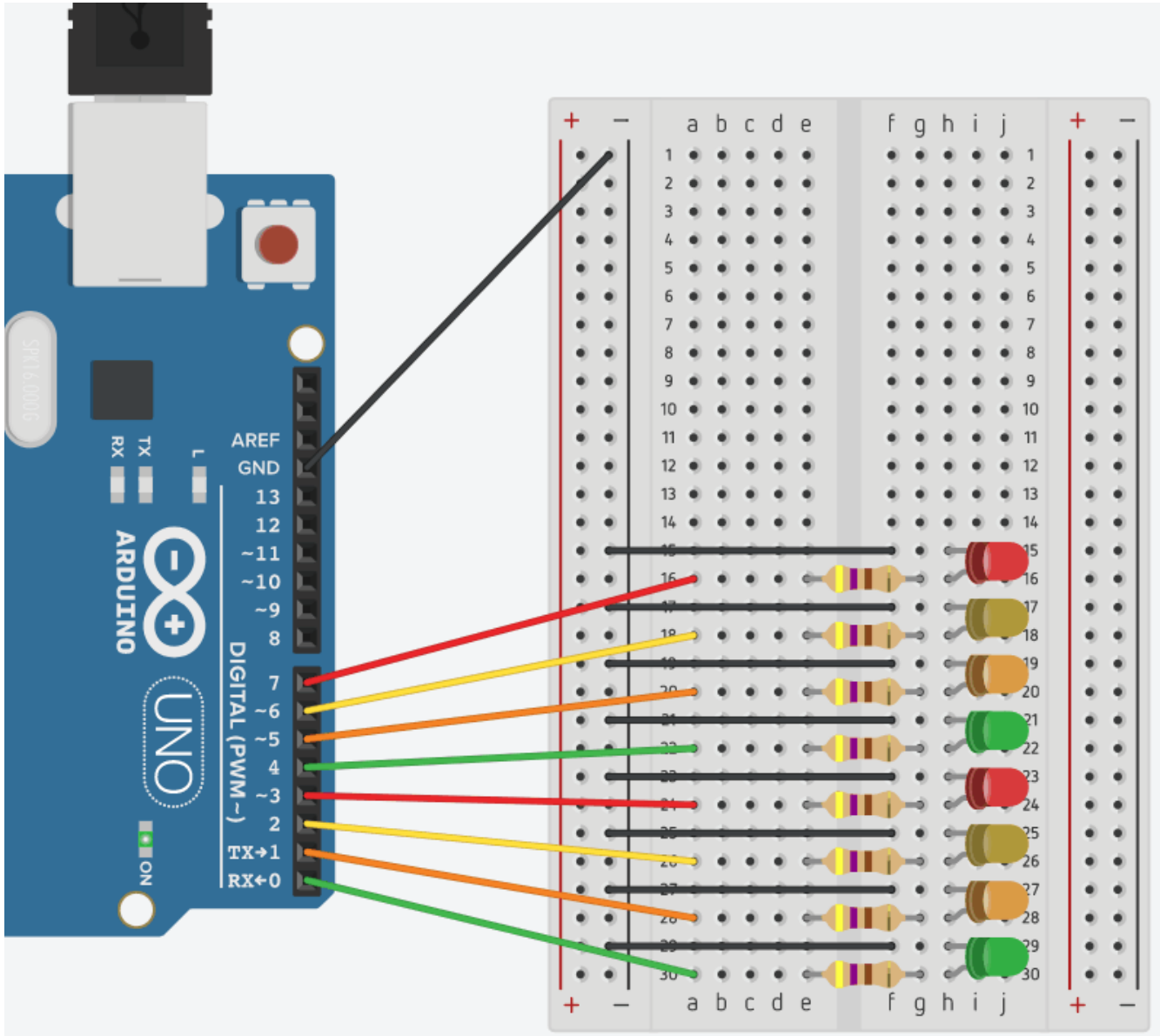
```
void setup() {
   DDRD = 0xFF;
}
void loop() {
   DDRD = 0x0F;
   delay(500);
   DDRD = 0xF0;
   delay(1500);

}
```



Para dia 05/07

1. Faça no TinkerCAD um circuito e programação que controle a intensidade de brilho de um LED através de um Potenciômetro.
2. Faça no TinkerCAD um circuito e programação que controle a velocidade com que um LED pisca através de um Potenciômetro.
3. Altere o último exemplo para que os LEDs ascendam alternadamente;
4. Com base no último exemplo, faça um sketch que pisque o LED do pino 13;



Como programamos o blink do LED no pino 13

```
1  // C++ code
2  //
3
4  #define portBArduinoValue *((volatile byte*) 0x25)
5  #define portBArduinoDirection *((volatile byte*) 0x24)
6
7  void setup()
8  {
9    portBArduinoDirection = 32 ;
10 }
11
12 void loop()
13 {
14   portBArduinoValue = 32;
15   _delay_ms(500); // Wait for 1000 millisecond(s)
16   portBArduinoValue = 0;
17   _delay_ms(2000); // Wait for 1000 millisecond(s)
18 }
```