

FIAP GRADUAÇÃO

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Disruptive Architectures: IA & IoT

PROF. ANTONIO SELVATICI

SHORT BIO



É engenheiro eletrônico formado pelo ITA, com mestrado e doutorado pela Escola Politécnica (USP), e passagem pela Georgia Institute of Technology em Atlanta (EUA). Desde 2002, atua na indústria em projetos nas áreas de robótica, visão computacional e internet das coisas, aliando teoria e prática no desenvolvimento de soluções baseadas em Machine Learning, processamento paralelo e modelos probabilísticos. Desenvolveu projetos para Avibrás, Rede Globo, IPT, CESP e Systax.

PROF. ANTONIO SELVATICI

profantonio.selvatici@fiap.com.br

1. DISPOSITIVOS DE IoT

ARDUÍNO – PORTA SERIAL

Como ler e escrever na porta serial do Arduino

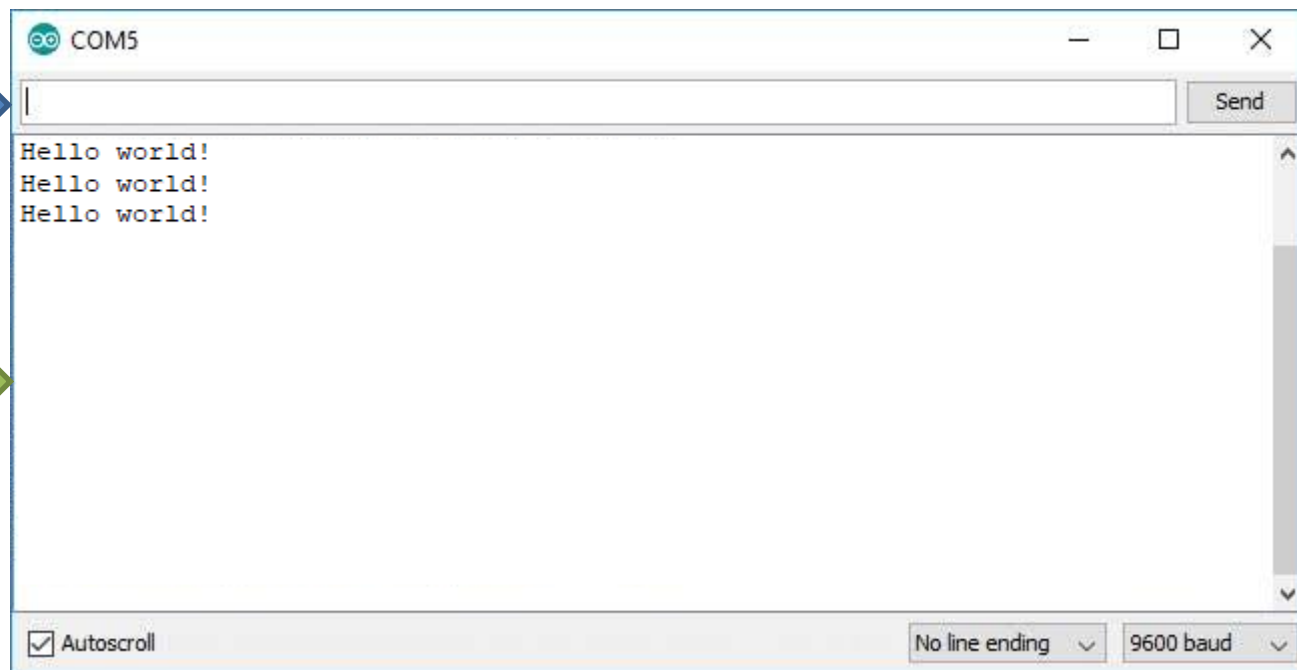
- Da mesma forma como podemos escrever dados na porta serial do Arduino, enviando dados para o computador, podemos também ler os dados que a placa recebe pela mesma porta
- Qual é a vantagem?
 - Podemos receber comandos do computador para executar alguma ação, permitindo o controle externo
- No exemplo a seguir, vamos ler o valor do nível do brilho do LED a partir da porta serial (de 0 a 255)
- Para auxiliar na recuperação dos dados da porta serial, usamos a classe String do Arduino
 - <https://www.arduino.cc/reference/pt/language/variables/data-types/stringobject/>

MONITOR SERIAL

Campo de envio de mensagens para o Arduino



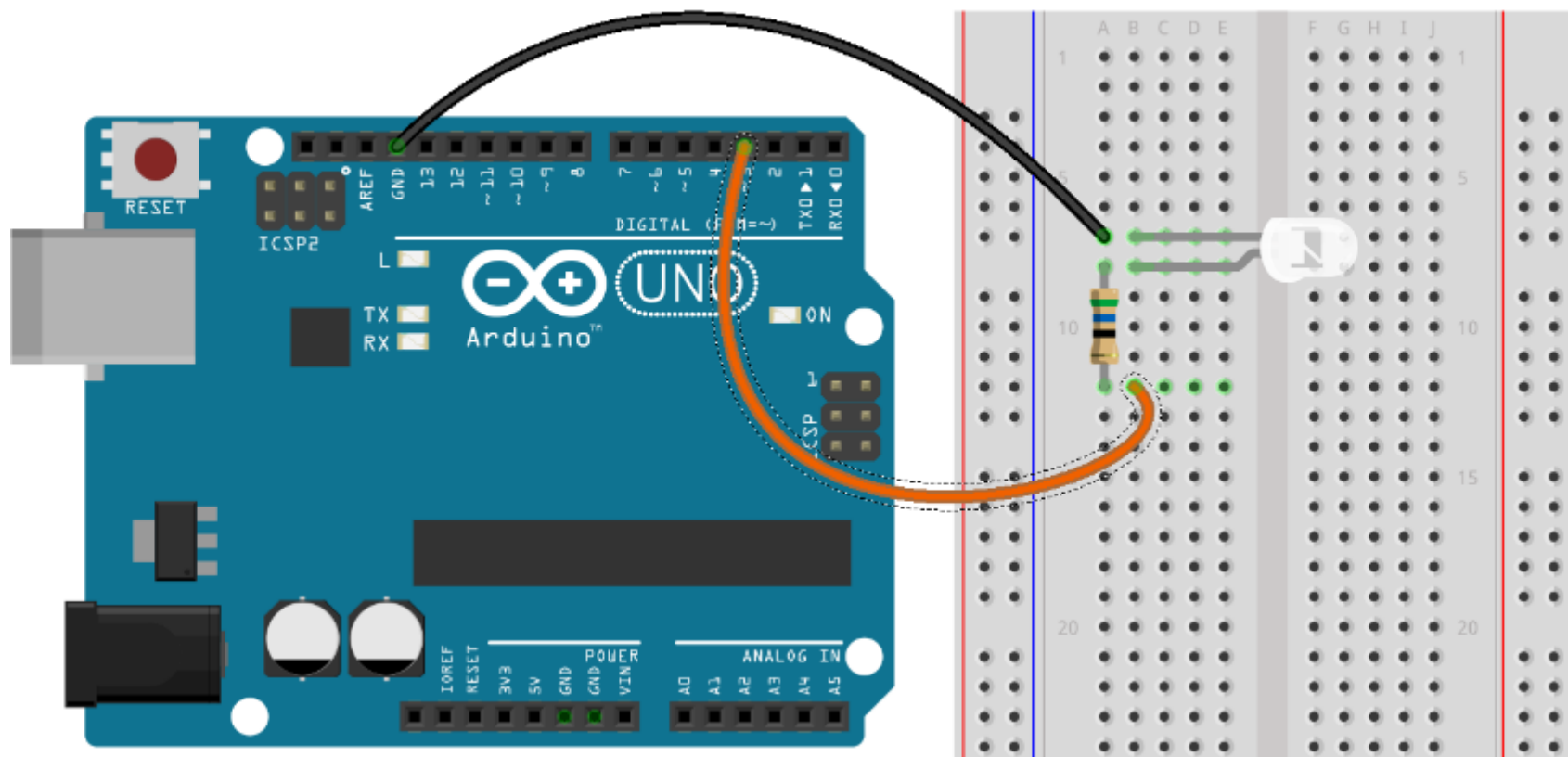
Área de recepção de mensagens vindas do Arduino



EXPERIMENTO 9: CONTROLE EXTERNO DA LUMINOSIDADE DO LED

- O objetivo é fazer com que a luz do LED seja controlada através de comandos externos, recebidos pela porta serial. O comando pode ser enviado através do monitor serial.
- O comando a ser enviado deve obedecer ao padrão: “B000E”
 - “000” representa uma sequência de dígitos (0 a 9)
 - Exmplo: B120E
- Materiais:
 - 1 Arduino Uno
 - 1 Protoboard
 - 1 LED de alto brilho
 - 1 Resistor de 56 ohm
 - Cabinhos tipo jumper

EXPERIMENTO 9: CIRCUITO



EXPERIMENTO 9: PROGRAMAÇÃO

```
const int LED = 3;
char nextChar = 0, lendo = 0;
String valor;
void setup() {
    Serial.begin(9600);
    pinMode(LED,OUTPUT);
}
void loop() {
    if (Serial.available() > 0) {
        // lê o byte disponível na porta serial:
        nextChar = Serial.read();
        if(!lendo && (nextChar == 'B' || nextChar == 'b')) {
            lendo = 1; //lendo <- true
            valor = "";
        } else if(lendo &&(nextChar == 'E' || nextChar == 'e')) {
            lendo = 0; //lendo <- false
            analogWrite(LED,valor.toInt());
            Serial.println(String("Potencia do LED: ") + valor);
        } else if(lendo && nextChar >= '0' && nextChar <= '9') {
            valor += nextChar;
        }
    }
}
```

EXPERIMENTO 9: PASSOS DO PROGRAMA

- `Serial.available()` indica a quantidade de caracteres que estão aguardando a leitura na porta serial. Assim, o programa só lê a porta se houver pelo menos um caractere.
- `Serial.read()` lê um caractere da porta serial, que é armazenado em `nextChar`
- Caso a leitura do valor numérico não tenha sido iniciada (`lendo=0`) e o caractere seja 'B', inicia a leitura com `lendo=1` e esvaziando a String `valor`
- Caso a leitura do valor já tenha sido iniciada (`lendo=1`) e o caractere seja um dígito (caractere ASCII entre '0' e '9'), concatena esse caractere à String `valor`, que irá formar a representação textual do número
- Caso a leitura do valor já tenha sido iniciada e o caractere seja 'E', transforma a String `valor` em inteiro e usa para definir o brilho do LED, finalizando a leitura do valor numérico com `lendo=0`

USANDO A STRING NO ARDUÍNO

- A forma tradicional de representar strings em C é através de arrays de caracteres
 - `char string_do_c[256] = "Ola, isto eh uma string";`
- No entanto, a API do Arduino fornece a classe `String`, que é bem mais flexível:
 - `String string_do_Arduino = "Isto eh uma string do Arduino";`
- Criando uma `String` a partir da concatenação de valores
 - `String outraString = String("Valor: ") + 128;`
- Anexando valores:
 - `outraString += ", outro valor:"; outraString += 256;`
- Interpretando valores numéricos, retornando zero no caso de erro
 - `int numero = minhaString.toInt();`
 - `float numFloat = minhaString.toFloat();`

Decodificando números e recebendo linhas de texto

- A API do Arduino possui funções que consomem os caracteres disponíveis na porta Serial de acordo com alguma regra
- As funções **`Serial.parseInt()`** e **`Serial.parseFloat()`** leem a porta serial em busca de um número inteiro/ponto flutuante
 - Caso caracteres não numéricos sejam encontrados, eles são pulados
 - Essas funções aguardam por um tempo pré-especificado por mais caracteres até retornarem o valor definitivo. Esse tempo é 1000 ms por padrão, e pode ser configurado através de **`Serial.setTimeout(int milliseg)`**
 - Caso não seja possível decodificar um número, o resultado é 0 (zero)! Isso pode gerar confusões no programa, então fique atento!
- A função **`Serial.readBytesUntil()`** lê caracteres vindo da porta serial, escrevendo em um vetor de caracteres.
 - A função termina quando o caractere de terminação for detectado, o tamanho máximo for lido ou o tempo de espera por mais texto terminou
 - **`Serial.readBytesUntil(char terminador, char buffer[], int tamanho)`**

Arduino – lendo número inteiro diretamente da porta serial

```
const int LED = 3;
char nextChar = 0;
void setup() {
    Serial.begin(9600);
    pinMode(LED,OUTPUT);
}
void loop() {
    if (Serial.available() > 0) {
        // lê o byte disponível na porta serial:
        nextChar = Serial.read();
        if(nextChar == 'B') {
            //Lê o próximo inteiro vindo da serial
            int valor = Serial.parseInt();
            //Atenção: em caso de erro o valor lido será 0
            analogWrite(LED,valor);
            Serial.println(String("Potencia do LED: ") + valor);
        }
    }
}
```

PADRONIZAÇÃO DAS MENSAGENS

- Até agora, as mensagens que enviamos ao Arduino obedecem ao seguinte padrão:
 - B000E
- Esse padrão foi imaginado especificamente para esta aula, e não segue o padrão adotado por nenhuma empresa ou governo
- O ideal seria que o Arduino conversasse com programas no computador através de um padrão de mensagens que pudesse ser compreendido por diferentes programas
- Para formatar as mensagens e garantir que uma ampla gama de programas consigam se comunicar com o Arduino, vamos usar o formato JSON.
- Outro formato bastante popular, o XML é mais complexo e exigiria um processamento maior para sua geração e interpretação

■ JSON – JavaScript Object Notation

- Do próprio site **json.org**:
 - JSON é um formato leve de troca de dados (serialização), de fácil leitura e escrita por humanos e máquinas
 - É parte da especificação de 1999 do JavaScript, que codifica e decodifica JSON nativamente
 - JSON é um formato de texto completamente independente de linguagem
- Exemplo de estrutura JSON:
 - `{"nome": "João", "idade": 23, "mulher": false, "filhos": ["Pedro", "Artur"] }`
 - A quebra de linha é opcional, porém facilita a visualização humana
- Podemos validar um JSON através do site <http://jsonlint.com/>

I Valores em JSON

- Um valor escrito em JSON pode assumir um dos seguintes formatos:
 - **String:**
 - texto unicode não formatado, escrito sempre entre aspas (como em Java)
 - Exemplos: "José", "Marçäl", "opa123", etc.
 - **Número:**
 - sequência de dígitos com separador decimal (ponto) e notação científica, como em Java, mas aceita apenas números decimais
 - Exemplos: 12, 15.01, 1.35e-24
 - **Objeto (object):**
 - Pares do tipo chave:valor contidos entre chaves ({ }) e separados por vírgula
 - Exemplo: {"idade":23, "peso":53.5}
 - **Vetor (array):**
 - Conjunto ordenado de valores contidos entre colchetes ([]) e separados por vírgula
 - Exemplo: [1, 2.5, "três", [4], {" próx": 5}]
 - **true, false:** constantes lógicas representando verdadeiro e falso, respectivamente
 - **null:** constante indicando um valor nulo

Objeto do JSON

- É a estrutura de dados mais emblemática do JSON
- É composto por um conjunto de pares do tipo `chave:valor` separados por vírgula
 - `chave` deve ser uma **string**, que serve de rótulo para o valor
 - `valor` é qualquer valor válido do JSON, incluindo um array ou ainda outro objeto
- Alguns objetos válidos
 - `{ }` : objeto vazio
 - `{ "chave " : "valor " }`
 - `{ "nome" : "Alberto", "idade" : 54, " pais" : [" José", "Maria"] }`

Exemplo de um valor JSON válido

```
[
  {
    "id": 100,
    "nome": "Astolfo",
    "sobrenome":
    "Silva",
    "endereco": {
      "rua": "Rua das
    Orquideas",
      "no": 23
    }
  },
  {
    "id": 101,
    "nome": "Maria",
    "sobrenome":
    "Teresa",
    "idade": 49
  }
]
```

EXERCÍCIO: CRIAÇÃO DE DOCUMENTO JSON

- Acesse o site `jsonlint.org`
- No campo de texto, criar um objeto Json contendo os seguintes campos (crie seus próprios valores):
 - `nome` (texto)
 - `idade` (número)
 - `corintiano` (booleano)
 - `endereço`: objeto com os campos
 - `rua` (string)
 - `número` (número)
 - `cep` (string)
 - `telefones`: array contendo 3 números de telefone

I JSON no Arduino

- Há várias bibliotecas em C++ para a codificação e a decodificação de JSON, porém nem todas são otimizadas para rodar no Arduino
- A biblioteca que vamos adotar aqui é a **ArduinoJson** (<https://github.com/bblanchon/ArduinoJson>), que relaciona objetos JSON com a estrutura de dados de dicionário do C++
- Os dicionários do C++ também relacionam uma chave (ou índice) a um rótulo, acrescentando dinamicamente elementos
 - `meuDic["nome"] = "Pedro Henrique"; //Acrescenta o elemento 'nome'`
 - `long valor = meuDic["idade"]; // Lê o elemento idade`
- Para usar a API, a primeira providência é importar o seu cabeçalho no código, trazendo na primeira linha do programa:
 - `#include <ArduinoJson.h>`

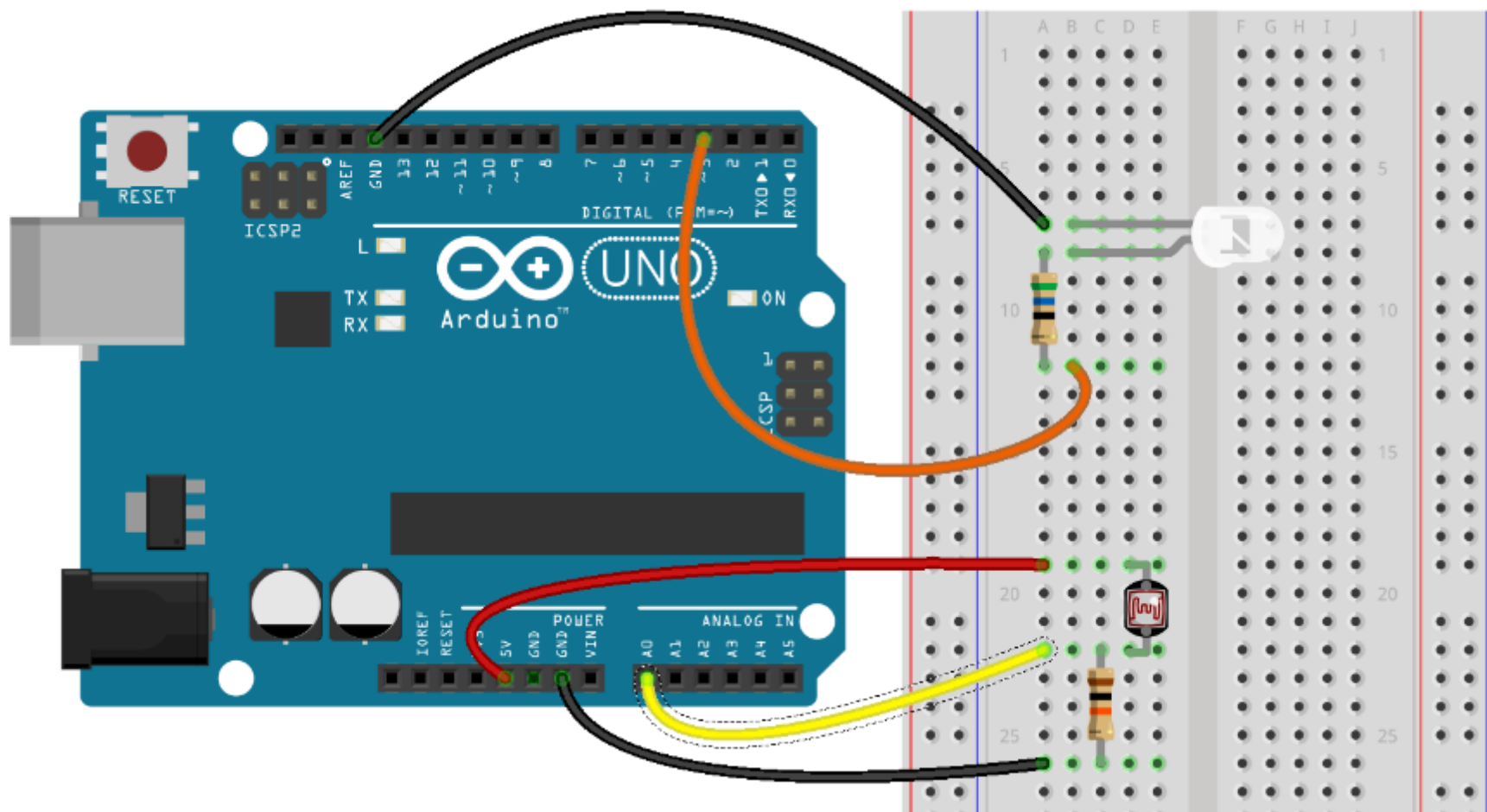
■ Criando e imprimindo um objeto JSON

- Primeiramente, devemos reservar memória para a criação do objeto (aqui é Arduino, não esqueçam)
 - `StaticJsonDocument<200> json; //Reserva 200 bytes`
- Agora podemos criar os elementos, com formatos identificados automaticamente
 - `json["sensor"] = "gps";`
 - `json["time"] = 1351824120;`
 - `json["pi"] = 3.141592;`
- Para acrescentar um array ou outro objeto, é necessário usar um método especial
 - `JsonArray array = json.createNestedArray("meu_array");`
 - `array.add("José"); array.add(48.756080);`
 - `JsonObject obj = json.createNestedObject("obj");`
- Finalmente, imprimimos a string JSON resultante na porta serial ou em uma string do C
 - `serializeJson(json, Serial); //manda o resultado pela porta serial`
 - `char txt[200]; serializeJson(json, txt); //escreve o resultado na string do C 'txt'`

EXPERIMENTO 10: ARDUINO MANDANDO DADOS DE LUMINOSIDADE

- O objetivo é permitir que o Arduino envie dados de luminosidade ao computador no formato JSON
- Materiais (deixe montado para o próximo experimento):
 - 1 Arduino Uno
 - 1 Protoboard
 - 1 LDR (fotorresistor)
 - 1 LED de alto brilho
 - 1 Resistor de 10 kilo-ohm
 - 1 Resistor de 56 ohm
 - Cabinhos tipo jumper

EXPERIMENTO 10: CIRCUITO



EXPERIMENTO 10: PROGRAMAÇÃO

```
#include <ArduinoJson.h>
const int LUZ = A1;
const int TAMANHO = 200;
void setup() {
    Serial.begin(9600);
}
void loop() {
    StaticJsonDocument<TAMANHO> json;
    json["luz"] = analogRead(LUZ);
    serializeJson(json, Serial);
    Serial.println();
    delay(1000);
}
```

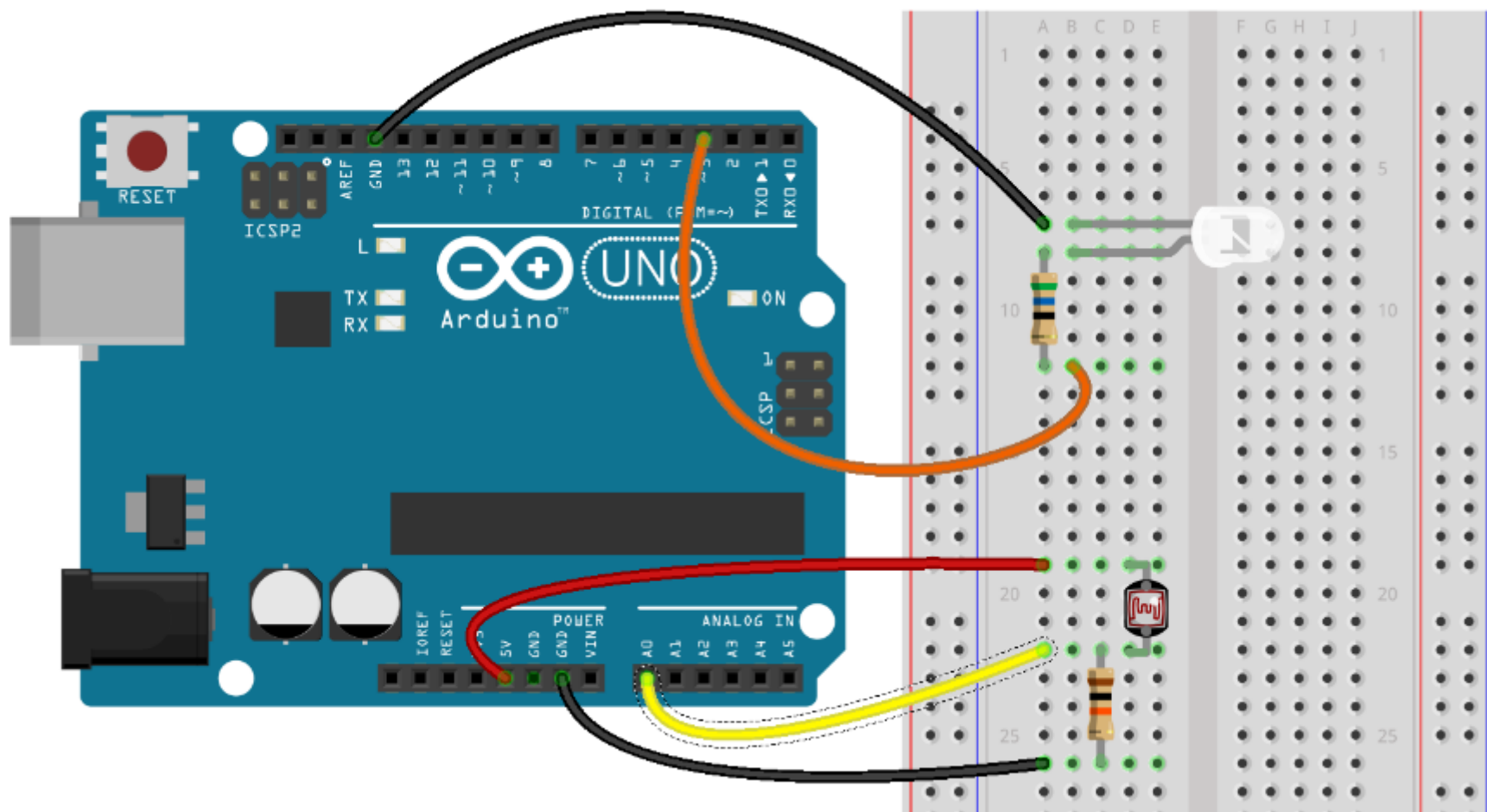

Lendo um JSON no Arduino

- Para decodificar um JSON é ainda mais fácil. Basta reservar a memória, checar os campos e resgatar os valores desejados
- Primeiro, vamos transformar o texto JSON armazenado em uma variável ou vindo da porta Serial em um objeto
 - `char texto[] = "{ \"sensor\": \"gps\", \"time\": 1351824120, \"data\": [48.756080, 2.302038]}\";`
 - `StaticJsonDocument<200> json;`
 - `deserializeJson(json, texto);`
 - `if (json.containsKey(\"time\")) {...} //verifica se o campo \"time\" foi capturado corretamente`
- Podemos desserializar diretamente da porta serial
 - `StaticJsonDocument<200> json;`
 - `deserializeJson(json, Serial);`
- Capturando os valores:
 - `const char* sensor = raiz[\"sensor\"];`
 - `long time = raiz[\"time\"];`
 - `double latitude = raiz[\"data\"][0];`
 - `double longitude = raiz[\"data\"][1];`

EXPERIMENTO 11: ARDUINO RECEBENDO COMANDOS DO LED NO CAMPO "led"

- O objetivo é permitir que o Arduino receba os comando do led (número de 0 a 255) dentro do campo "led" do JSON
- O comando que deve ser enviado é:
 - {"led": 127}
- Configurar o monitor serial para inserir quebra de linha após o comando
- Materiais:
 - 1 Arduino Uno
 - 1 Protoboard
 - 1 LDR (fotorresistor)
 - 1 LED de alto brilho
 - 1 Resistor de 10 kilo-ohm
 - 1 Resistor de 56 ohm
 - Cabinhos tipo jumper

EXPERIMENTO 11: CIRCUITO



EXPERIMENTO 11: PROGRAMAÇÃO

```
#include <ArduinoJson.h>
const int LED = 3;
const int TAMANHO = 200;
void setup() {
    Serial.begin(9600);
    //O valor padrão de 1000ms é muito tempo
    Serial.setTimeout(10);
    pinMode(LED, OUTPUT);
}
void loop() {
    if (Serial.available() > 0) {
        //Lê o JSON disponível na porta serial:
        StaticJsonDocument<TAMANHO> json;
        deserializeJson(json, Serial);
        if(json.containsKey("led")) {
            int valor = json["led"];
            analogWrite(LED, valor);
        }
    }
    delay(300);
}
```

EXERCÍCIO

- Construir um programa do Arduino que unifica as funcionalidades do Experimento 10 do Experimento 11, ou seja:
 - Deve enviar valores de luminosidade no campo `luz` do JSON para a porta serial
 - Receber comando para o LED enviados como JSON no campo `led`
 - Usar pausa de 300 ms para ambas as atividades
- Dica: no C++, variáveis e objetos que são declaradas dentro de um bloco de código (`if`, `for`, `while`, etc.) estão isoladas de variáveis e objetos declarados em um escopo mais geral
 - Assim, se você declarar um documento JSON dentro e fora de um bloco, eles serão objetos diferentes
 - Quando o programa sai do escopo (bloco) de um objeto automático, ele é apagado (libera memória)

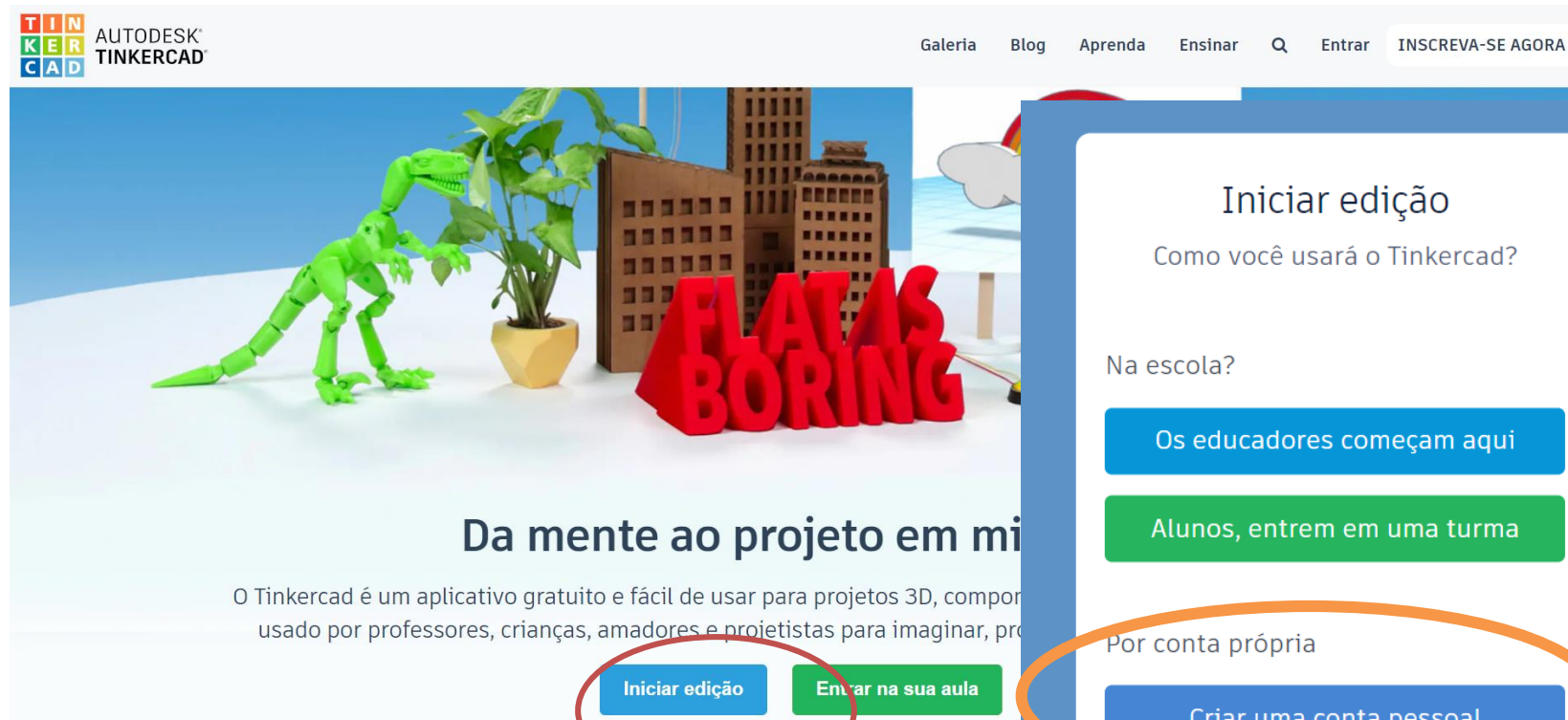
O SIMULADOR TINKERCAD

- O simulador de circuitos Tinkercad permite a simulação online de circuitos eletrônicos contendo diversos componentes, inclusive sensores, atuadores e alguns microcontroladores
 - Com ele é possível inclusive programar o Arduino simulado, contando inclusive com toda a API do Arduino mais algumas bibliotecas extras
 - Spoiler: a biblioteca ArduinoJson está presente, porém sem estar listada e em versão antiga... Vai dar um pouco de trabalho, mas conseguiremos utilizá-la!

USANDO O SIMULADOR TINKERCAD

- Acesse o site tinkercad.com
- Escolha a opção “Iniciar Edição” e se inscreva ou faça o login em uma conta pessoal.
- Quando chegar ao dashboard, escolher a opção “Circuits”
- Para criar um novo circuito, clicar em “Criar um novo circuito”

ENTRANDO NA OPÇÃO DE CRIAR CONTA OU LOGAR



The image shows the Autodesk Tinkercad website. At the top, there's a navigation bar with links: Galeria, Blog, Aprenda, Ensinar, Q, Entrar, and a button INSCREVA-SE AGORA. The main header features the Tinkercad logo and a large 3D scene with a green dinosaur, a potted plant, and a city skyline. Below the scene, the text reads "Da mente ao projeto em minutos" and "O Tinkercad é um aplicativo gratuito e fácil de usar para projetos 3D, composto por peças e ferramentas digitais que podem ser usadas por professores, crianças, amadores e projetistas para imaginar, projetar e imprimir em 3D". At the bottom of the main content area, there are two buttons: "Iniciar edição" (highlighted with a red circle) and "Entrar na sua aula". A large orange arrow points from the "Iniciar edição" button to a blue-bordered box on the right.

INICIAR EDIÇÃO

Como você usará o Tinkercad?

Na escola?

Os educadores começam aqui

Alunos, entrem em uma turma

Por conta própria


Criar uma conta pessoal

OU


Já tem uma conta?


Entrar

Opção “Circuits”


**AUTODESK
TINKERCAD**

Classes
Galeria
Blog
Aprenda
Ensinar


100


antoniohps

Pesquisar projetos...

Projetos 3D

Circuits


Blocos de código
NOVO

Lições

Projetos


+ Criar projeto

Tweets
Seguir


Tinkercad
@tinkercad

In this video, Rob Morrill guides you

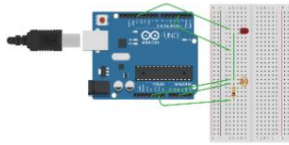
<https://www.tinkercad.com/things/1XirWAZwM-funky-trug>


antoniohps

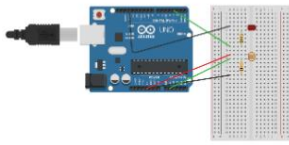
Circuits

Criar novo Circuito

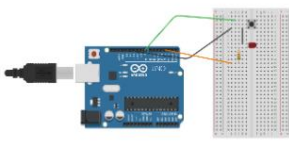
☒ Select



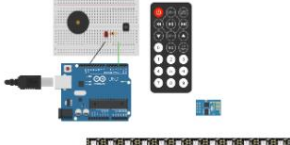
Surprising Albar
há uma hora
Privado



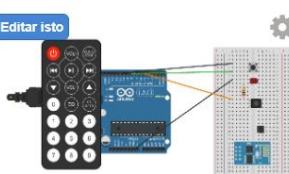
Brilliant Crift
há 6 dias
Público




Stunning Wolt
há 7 meses
Privado




Frantic Gogo-Bombul
há 9 meses
Privado



Tremendous Fulffy-Lappi
há um ano



Mighty Inari
há um ano



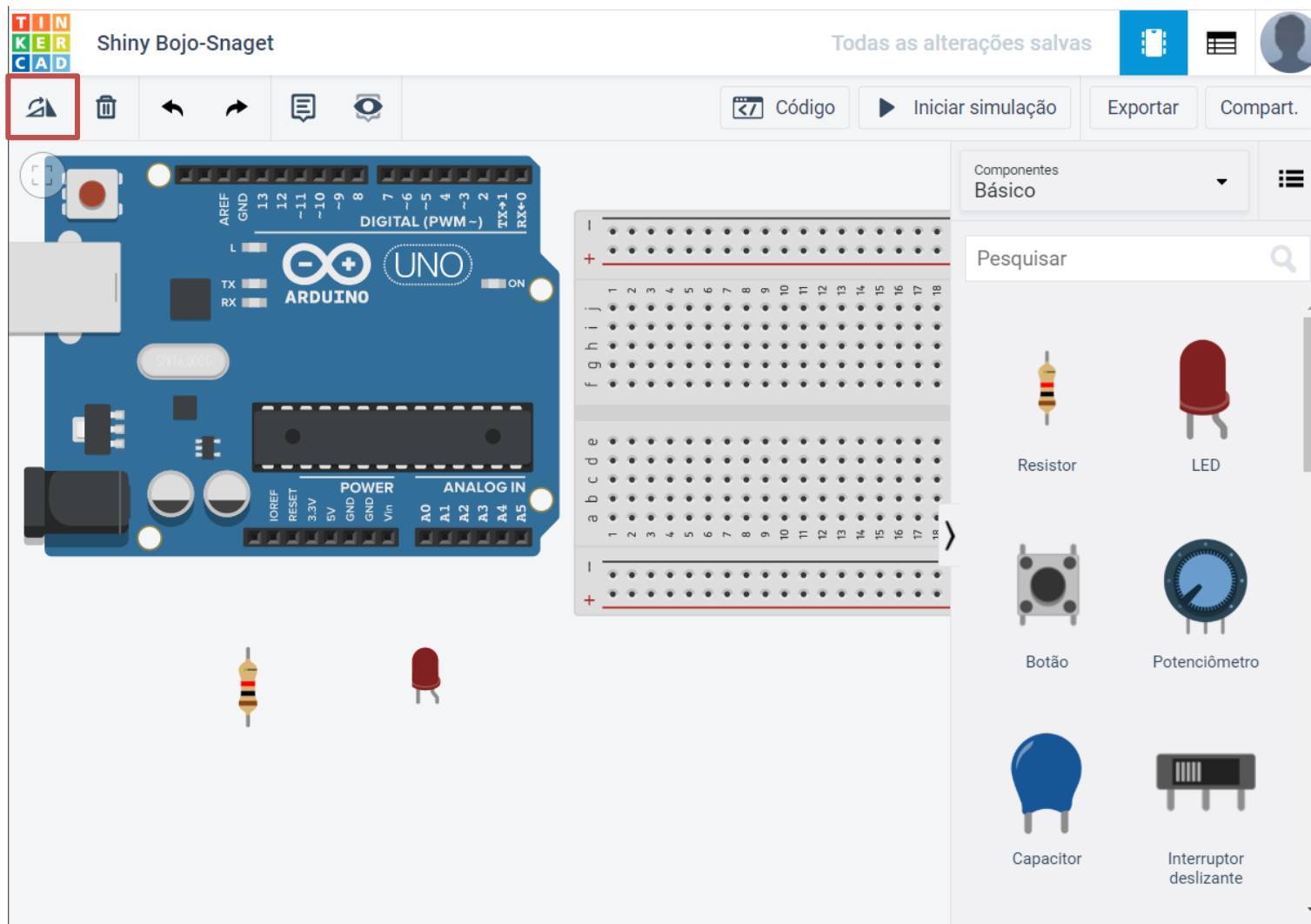
Super Kasi-Duup
há um ano

I CRIANDO UM CIRCUITO

- Ao iniciar a tela de criação de circuito, o Tinkercad atribui um nome espalhafatoso, que pode ser trocado ao se clicar sobre ele
- À esquerda da tela temos um menu de opções de componentes, incluídos o Arduino Uno, protoboard (chamado de placa de ensaio), LED, resistor, etc.
- Para importar um componente, basta clicar sobre ele e trazer o ponteiro do mouse para a tela de projeto

Exemplo de tela de projeto com resistor, LED, protoboard e Arduino Uno

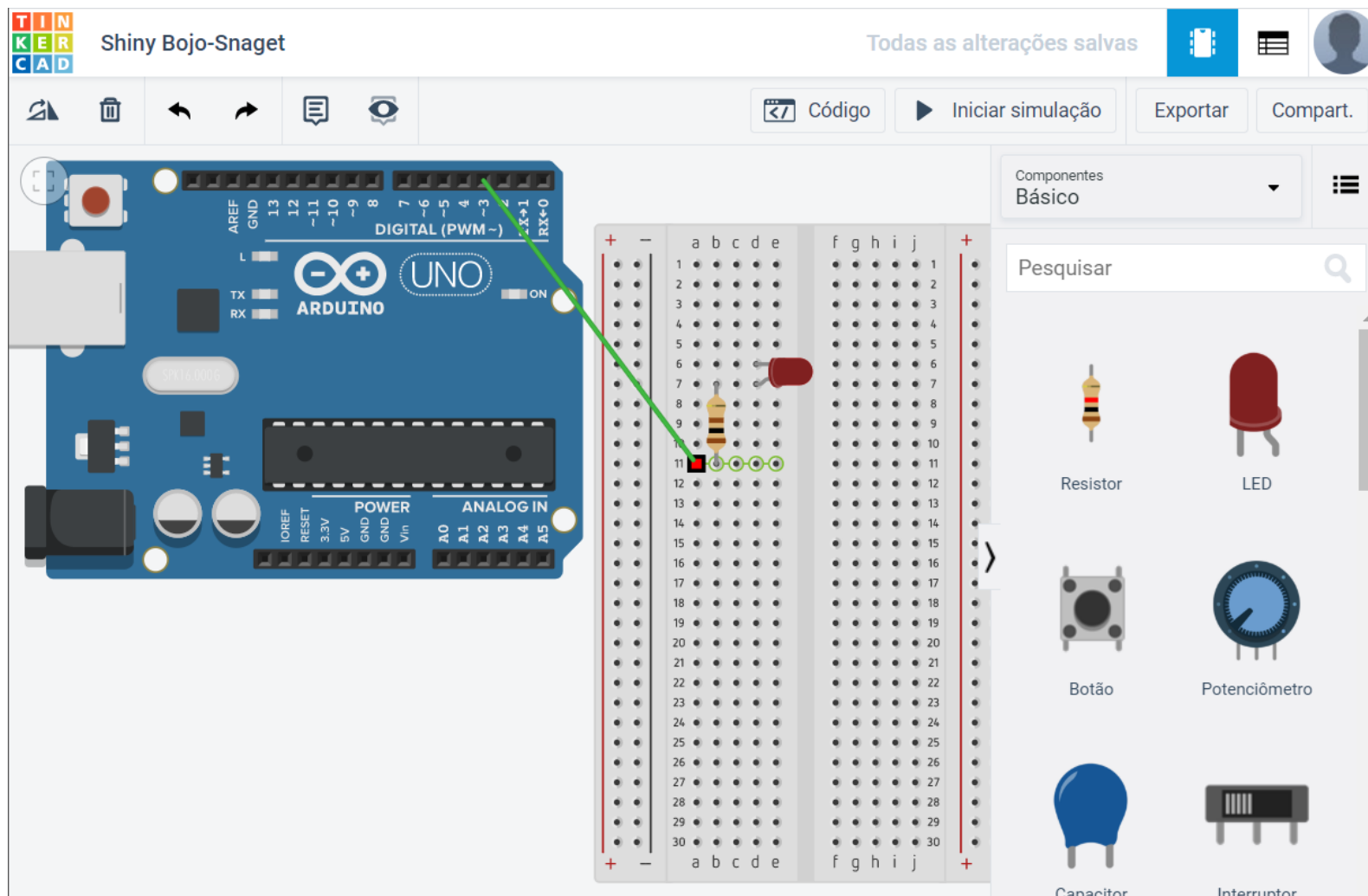
Girar



CONECTANDO OS COMPONENTES

- Para conectar os componentes ao protoboard, basta arrastá-los sobre ele e ajustar as pernas para coincidir com os conectores do protoboard
- Para girar um componente, o selecionamos e clicamos sobre o botão de girar, ou apertar a tecla R
 - Temos também a opção de mudar o nome do componente e outras propriedades específicas, como as cores do LED e do fio, bem como a resistência do resistor
- Para conectar trilhas do protoboard e portas do Arduino, basta clicar sobre elas que um fio será automaticamente criado. Ele deve ser arrastado até o próximo terminal ou trilha

CONECTANDO COMPONENTES



PROGRAMANDO O ARDUINO SIMULADO

- Para programar o Arduino, escolhemos a opção “Código” ao lado direito da tela
 - Aparecerá uma tela com o código na forma de blocos (linguagem sketch), o que deverá ser mudado para texto no menu seletor apropriado
 - Podemos ver também o monitor serial na parte de baixo da tela de código
- A programação do Arduino é feita exatamente como na IDE

PROGRAMAÇÃO DO ARDUINO

The image shows the Tinkercad Arduino IDE interface. On the left is a 3D model of an Arduino Uno R3 board. In the center is the code editor with the following code:

```

6 void loop()
7 {
8   digitalWrite(13, HIGH);
9   delay(1000); // Wait for 1000 millisecond(s)
10  digitalWrite(13, LOW);
11  delay(1000); // Wait for 1000 millisecond(s)
12 }

```

Annotations with boxes point to specific features:

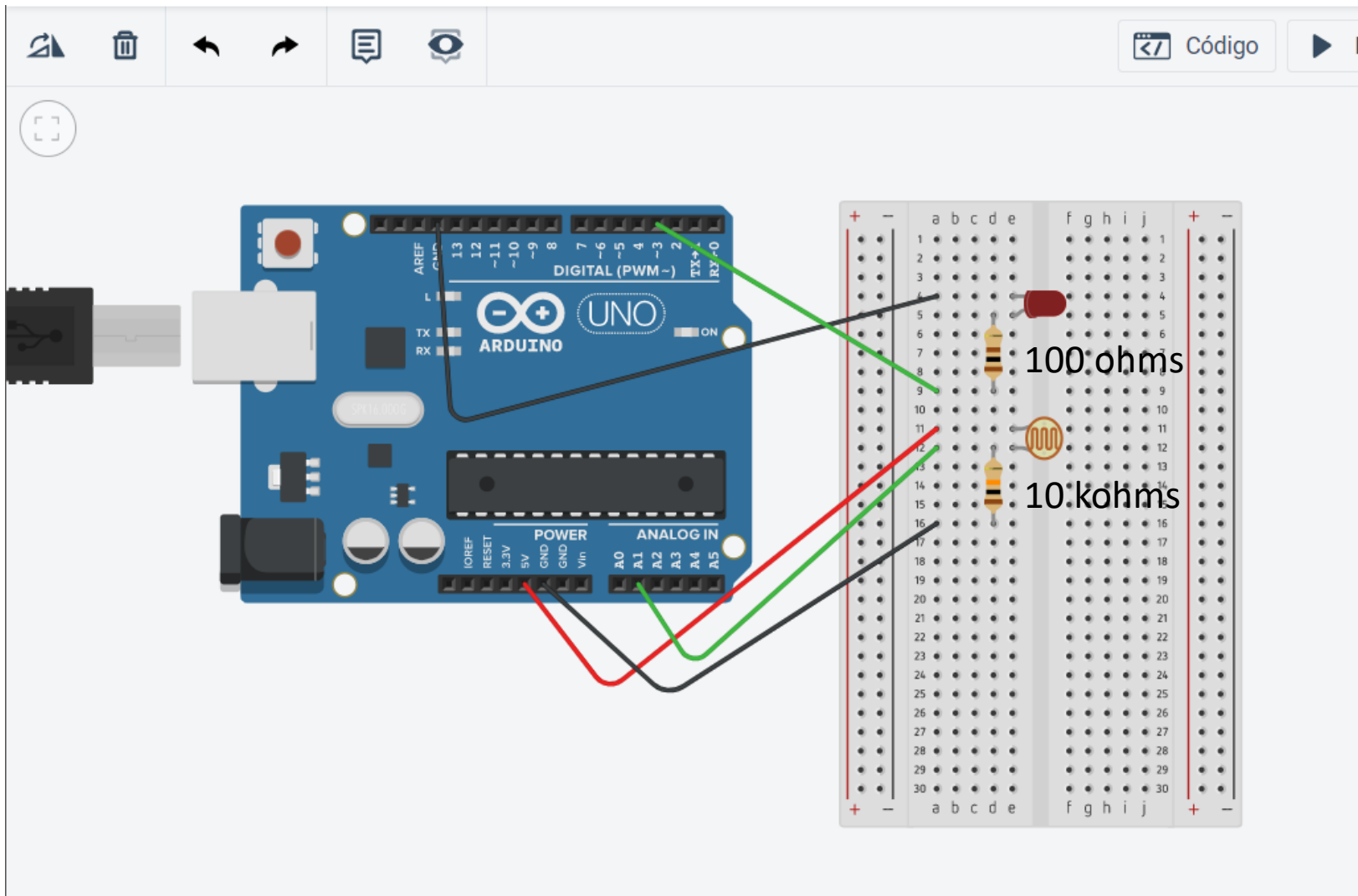
- Mostra a tela de código**: Points to the "Código" button in the top right toolbar.
- Menu seletor de linguagem**: Points to the "Texto" button in the block palette on the left.
- Tela de codificação**: Points to the code editor area.
- Mostra o monitor serial**: Points to the "Monitor serial" button at the bottom of the interface.

Other visible elements include the "Shiny Bojo-Snaget" project name, the "Iniciar simulação" button, and the "Exportar" and "Compart." buttons.

EXPERIMENTO 12: CONTROLE DA LUMINOSIDADE NO TINKERCAD

- O objetivo é fazer com que a luz do LED seja controlada pela luminosidade detectada pelo LDR. Assim, quanto mais claro o ambiente, menor a luz do LED.
- O código é igual ao do experimento 8.
- Para mudar a luminosidade
- Materiais do Tinkercad:
 - 1 Arduino Uno
 - 1 Protoboard pequeno
 - 1 LDR (fotorresistor)
 - 1 LED
 - 1 Resistor de 10 kilo-ohm
 - 1 Resistor de 100 ohm
 - Cabinhos tipo jumper

EXPERIMENTO 12: CIRCUITO



USANDO ARDUINOSON NO TINKERCAD

- Quanto ao ArduinoJson, temos uma notícia boa e uma ruim...
 - A boa é que a biblioteca já está pré-instalada!
 - A ruim é que a versão é antiga, e por isso a sintaxe é mais complexa
- Assim, para utilizarmos a sintaxe moderna do ArduinoJson, ou uma parte dela, é necessário construir um código de adaptação
- No próximo slide temos um código de adaptação para o uso que fizemos da biblioteca até agora:
 - Classe `StaticJsonDocument<N>`
 - Função `serializeJson()` para a porta serial
 - Função `deserializeJson()` a partir da porta serial
- Esse código deve ser inserido logo após a inclusão do cabeçalho `ArduinoJson.h`

Código de Adaptação

```

////////// INÍCIO DO CODIGO DE ADAPTAÇÃO //////////
template <int N> class StaticJsonDocument;
template <int N> void serializeJson(const StaticJsonDocument<N>& json, Stream& stream);
template <int N> void deserializeJson(StaticJsonDocument<N>& json, Stream& stream);

template <int N> class StaticJsonDocument
{
    friend void serializeJson<N>(const StaticJsonDocument<N>& json, Stream& stream);
    friend void deserializeJson<N>(StaticJsonDocument<N>& json, Stream& stream);
public:
    StaticJsonDocument()
    { objptr = &buffer.createObject(); }
    JsonObjectSubscript<const char*> operator[] (const char* key)
    { return (*objptr)[key]; }
    bool containsKey(const char* key) const
    { return objptr->containsKey(key); }
private:
    StaticJsonBuffer<N> buffer;
    JsonObject* objptr;
};

template<int N> void serializeJson(
    const StaticJsonDocument<N>& json,
    Stream& stream)
{ json.objptr->printTo(stream); }

template<int N> void deserializeJson(
    StaticJsonDocument<N>& json,
    Stream& stream)
{
    char buff[N];
    stream.readBytesUntil('\n', buff, N);
    json.objptr = &json.buffer.parseObject(buff);
}
////////// FIM DO CODIGO DE ADAPTAÇÃO //////////

```

EXERCÍCIO

- Construir um programa do Arduino no Tinkercad que unifica as funcionalidades do Experimento 10 do Experimento 11, ou seja:
 - Deve enviar valores de luminosidade no campo `lu` do JSON para a porta serial
 - Receber comando para o LED enviados como JSON no campo `led`
 - Usar pausa de 300 ms para ambas as atividades
- Dica: usar o mesmo programa do exercício do slide 29, porém use o circuito do experimento 12 acrescentando o código de adaptação do slide 43.

REFERÊNCIAS



1. Arduino Team. **Arduino Reference: String Object.** url: <https://www.arduino.cc/en/Reference/StringObject>
2. Arduino Team. **Arduino Reference: Serial Port.** url: <https://www.arduino.cc/reference/en/language/functions/communication/serial/>
3. http://www.telecom.uff.br/pet/petws/downloads/tutoriais/arduino/Tut_Arduino.pdf
4. Benoît Blanchon. **ArduinoJson Documentation.** url: <https://arduinojson.org/v6/doc/>
5. Autodesk. **Tinkercad.** url: <https://tinkercad.com>



Copyright © 2020 Prof. Antonio Henrique Pinto Selvatici

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).