FIAP GRADUAÇÃO

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Arquiteturas Disruptivas, IoT, Big Data e IA

PROF. ANTONIO SELVATICI

Versão 4– Fevereiro/2018

SHORT BIO

É engenheiro eletrônico formado pelo Instituto Tecnológico de Aeronáutica (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, IPT, CESP e Systax.

PROF. ANTONIO SELVATICI
profantonio.selvatici@fiap.com.br

INTERNET DAS COISAS

USANDO BILIOTECAS

Sensor DHT-11

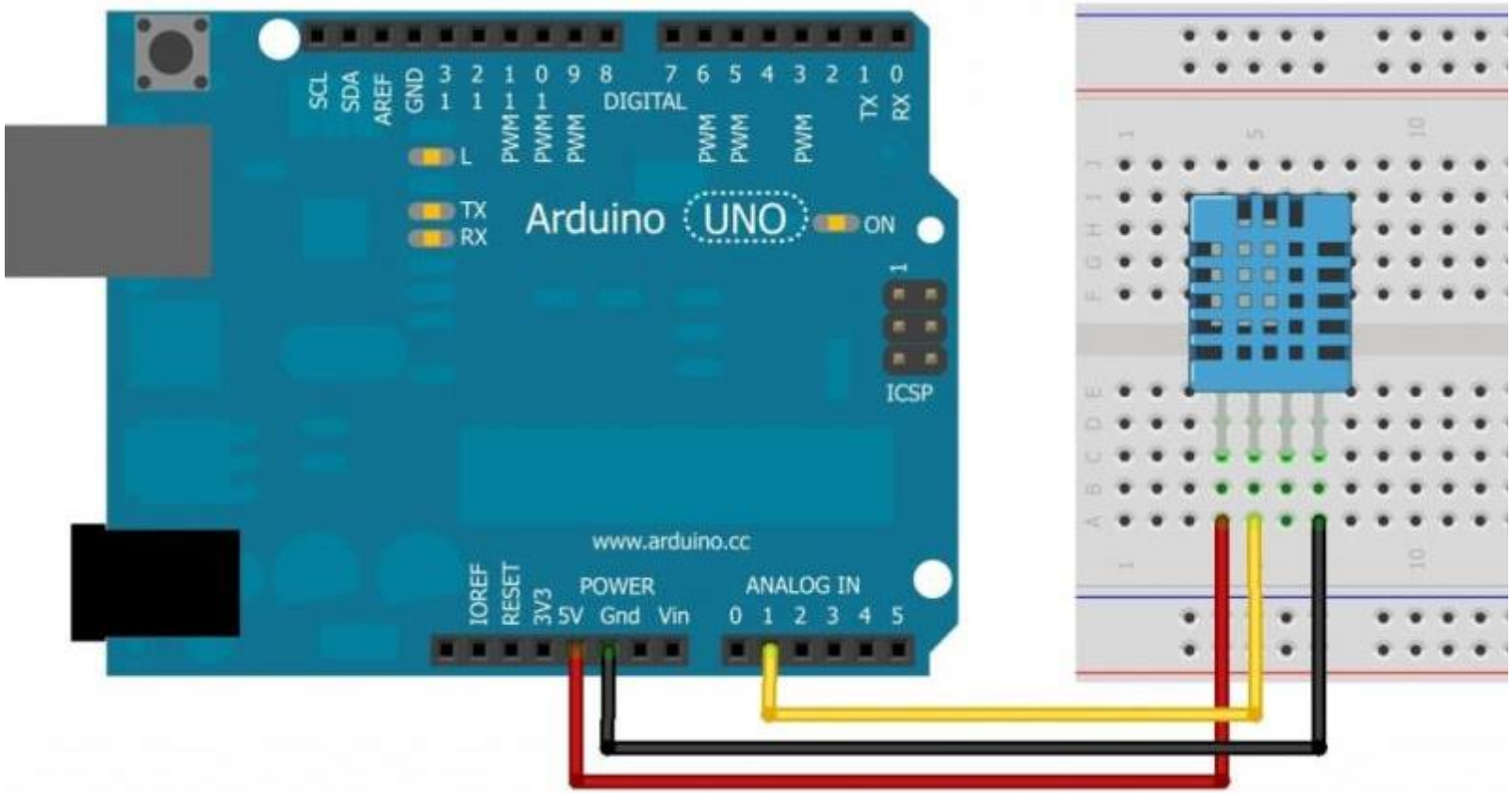
- Além da API padrão do Arduino, o Arduino IDE permite que importemos diversas bibliotecas que facilitam a vida do desenvolvedor
- Há bibliotecas que já vêm instaladas no Arduino IDE, e basta, ser invocadas através da diretiva #include<...>, mas outras precisam ser instaladas antes.
- Há duas formas de instalar novas bibliotcas:
  - Através do gerenciador de bibliotecas:
  - `Sketch → Import Libraries → Library Manager...`
  - Através da importação de arquivos compactados
  - `Sketch → Import Libraries → Add Library...`
  - É o modo como deve ser feito no dia da prova
- As bibliotecas possuem definição de classes e podem conter exemplos de utilização

O SENSOR DHT-11

- O sensor DHT11 fornece tanto temperatura quanto umidade do ar instantaneamente e de forma muito fácil.
- A comunicação com o Arduino é feita através de um protocolo serial próprio
- Verificar no datasheet:
  - http://robocraft.ru/files/datasheet/DHT11.pdf
- A biblioteca DHT cuida de todas as funções necessárias, restando para nós apenas acessar aos dados.
- Instalando a biblioteca
  - O arquivo DHT.zip da biblioteca está fornecida no portal de apostilas
  - No gerenciador de bibliotecas, instalar a versão 1.1.0 da biblioteca da Adafruit

SENSOR DHT-11

Lendo os dados através da porta A1



LENDO O SENSOR DHT-11

```
#include "DHT.h"
#define DHTPIN A1 // pino que estamos conectado
#define DHTTYPE DHT11 // DHT 11
DHT dht(DHTPIN, DHTTYPE); //Instanciação do objeto do sensor
void setup() {
  Serial.begin(9600);
  dht.begin();
}
void loop() {
  // A leitura da temperatura e umidade pode levar 250ms!
  float h = dht.readHumidity();//Valor da umidade
  float t = dht.readTemperature(); //Valor da temperatura
  if (isnan(t) || isnan(h)) {
    Serial.println("Erro ao ler do DHT");
  } else {
    Serial.print("Umidade: ");
    Serial.print(h); Serial.print(" %\t");
    Serial.print("Temperatura: ");
    Serial.print(t); Serial.println(" °C");
  }
}
```

INTERRUPÇÕES NO ARDUINO

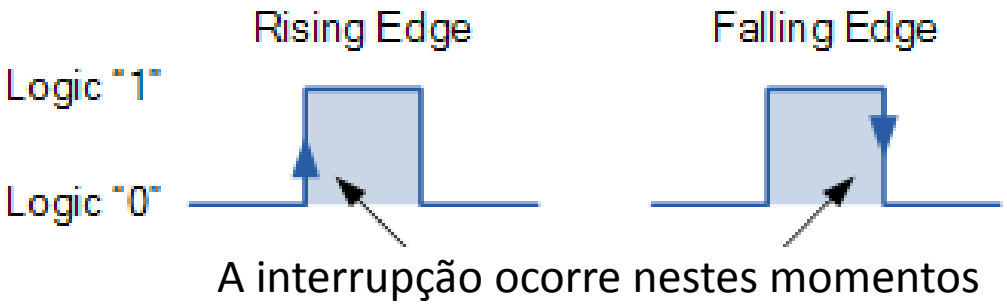
Realizando operações críticas em tempo real

- Imagine o seguinte cenário:
  - Uma fábrica precisa monitorar a pressão de 100 tanques de fabricação de amônia
  - Para ler a pressão de cada tanque o controlador demora 250ms
  - Se algum tanque estiver com sobrepressão, há um tempo hábil de um segundo para abrir a válvula de escape
- Se o controlador tivesse que ler a pressão de cada tanque para eventualmente tomar a decisão de abrir a válvula de algum deles, poderíamos ter uma explosão na fábrica, já que ele demoraria 25 segundos para ler todos os tanques
- Para essas situações, microcontroladores e microprocessadores possibilitam as chamadas **interrupções**, que são eventos que servem de gatilhos para ações especiais a serem executadas
- Assim, um sensor especial de sobrepressão poderia estar ligado a uma porta do controlador, que lançaria uma interrupção quando um tanque estivesse nessa situação

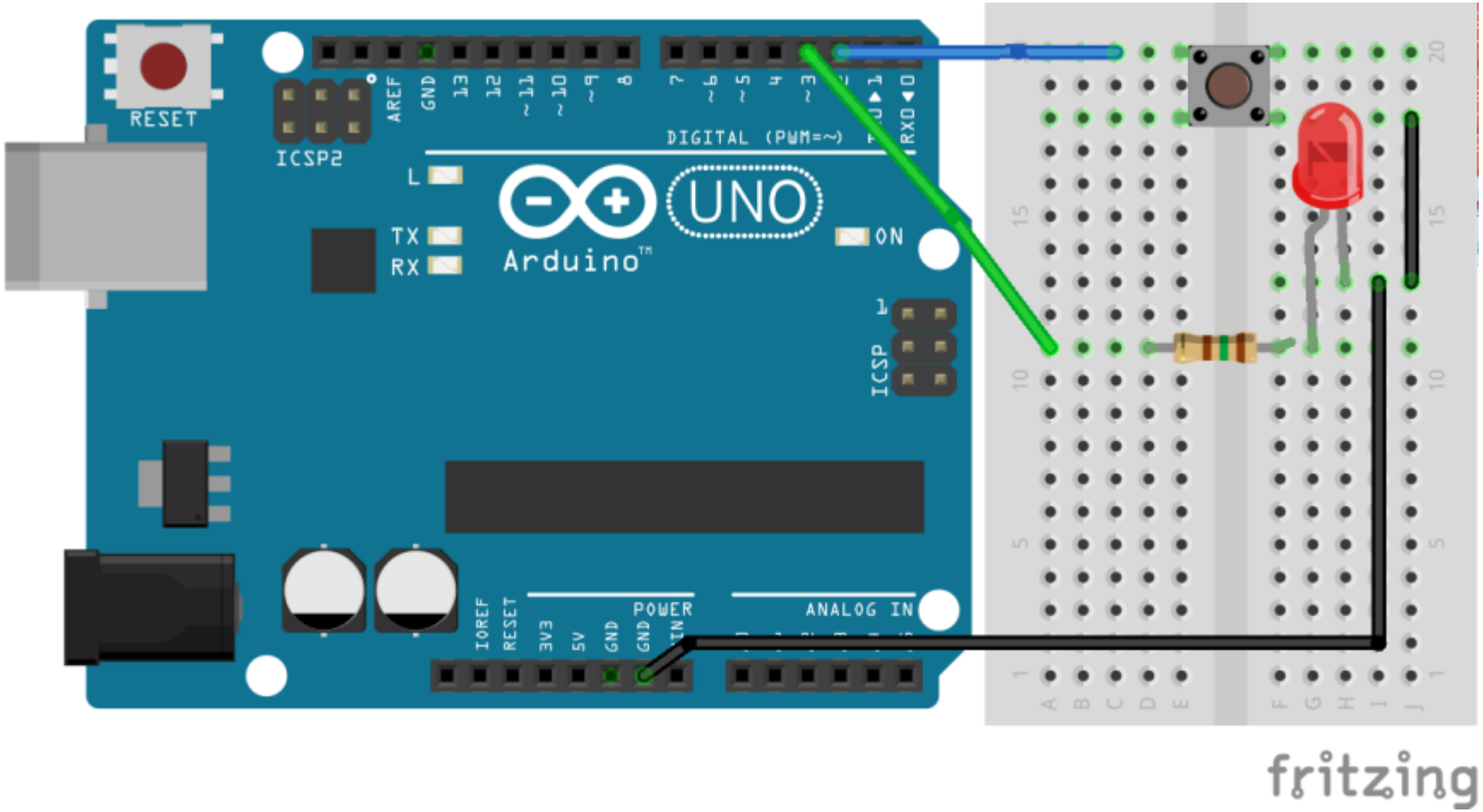
INTERRUPÇÕES NO ARDUINO

Realizando operações críticas em tempo real

- A ação a ser executada numa interrupção é executada na forma de uma **ISR** (*Interrupt Service Routine*), uma função com certas limitações que é invocada assim que ocorre a interrupção, interrompendo o processamento sendo executado no momento
- O gatilho pode ser gerado internamente, através de um temporizador chegando a zero, por exemplo, ou...
- Pode ser um gatilho externo, como o valor de uma porta de entrada sendo escrito
- Mais comumente o gatilho externo é de dois tipos
  - *Rising Edge*: interrupção com valor indo de LOW para HIGH
  - *Falling Edge*: interrupção com valor indo de HIGH para LOW



INTERRUPÇÕES NO ARDUINO



EXEMPLO DE INTERRUPÇÃO - ARDUINO

Ligar a porta D2 em GND usando o botão

```
int led = 3; //Porta do LED
//porta da interrupção:
//O Arduino Uno só aceita as portas 2 e 3
int interruptPort = 2;

//Variáveis modificadas por interrupções devem ser volatile
volatile int state = LOW;

void setup() {
  pinMode(led, OUTPUT);
  pinMode(interruptPort, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPort), toggle, CHANGE);
}
void loop() {
   // Qualquer processamento mais longo...
}
void toggle() {
  state = !state;
  digitalWrite(led,state);
}
```

INTERRUPÇÕES NO ARDUINO

Como funcionam

- A definição de uma **ISR** no Arduino é dada por uma função sem argumentos e sem valor de retorno (void).
- Durante a execução de uma ISR, as interrupções estão desabilitadas
  - Por isso devem ser de rápida execução
  - Funções de E/S podem não funcionar durante sua execução
- Para ativar uma interrupção, invocamos a função:
  - `attachInterrupt(NUMERO, ISR, MODO);`
  - `NUMERO:` identificação da interrupção, a depender da porta usada (ver em http://arduino.cc/en/Reference/attachInterrupt)
    - Para traduzir a porta usada para o número da interrupção, usar a função `digitalPinToInterrupt(pin)`
  - `ISR:` função a ser invocada durante a interrupção
  - `MODO:` o tipo do gatilho, podendo ser
    - LOW: dispara a interrupção quando a porta estiver em LOW
    - CHANGE: dispara quando há mudança de valor
    - RISING: dispara quando ocorre uma Rising Edge na porta
    - FALLING: dispara quando ocorre uma Falling Edge na porta

ARDUÍNO – PORTA SERIAL

Como ler a 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
- Podemos receber comandos do computador para executar alguma ação
- No exemplo a seguir, vamos ler o valor da potência do LED a partir da porta serial (de 0 a 255)

- Protocolo **Firmata**: controla o Arduino via porta serial através de um programa de computador externo. Assim, podemos mudar a programação sem ter que reprogramar o Arduino (não usaremos agora...)

- Para auxiliar na recuperação dos dados da porta serial, usamos a classe String do Arduino

ARDUÍNO – STRING

Representando strings no Arduino
Mais métodos de String em https://www.arduino.cc/en/Reference/StringObject

- 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();`

ARDUINO - LENDO A PORTA SERIAL

```
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(nextChar == 'B') {
      lendo = 1; //lendo <- true
      valor = "";
    } else if(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;
    }
  }
}
```

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 milisseg)**
  - 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 a porta Serial (outro modo)

```
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);
    }
  }
}
```

LENDO A PORTA SERIAL

Escrevendo na porta serial

- Podemos escrever na porta serial do Windows de uma forma bem simples através do Serial Monitor do Arduino (barra de texto superior)

```
B127E
```

- O ideal é termos um programa no computador que se comunicasse com o Arduino via porta serial, e que expusesse um serviço para que possa ser controlado remotamente, através de algum protocolo para troca de dados de sensores e comandos para os atuadores.

- Para formatar as mensagens e garantir que uma ampla gama de programas consigam se comunicar com o Arduino, vamos usar o formato JSON.

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/`

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
  }
]
```

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 um elemento
  - 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 um objeto JSON

- Primeiramente, devemos reservar memória para a criação do objeto (aqui é Arduino, não esqueçam)
  - `StaticJsonBuffer<200> jsonBuffer;  //Reserva 200 bytes`
- Então devemos alocar a árvore JSON na memória
  - `JsonObject& raiz = jsonBuffer.createObject();`
  - Podemos pensar em um objeto ou array JSON como sendo uma árvore porque cada elemento é considerado um filho, que por sua vez podem ser estruturas contendo outros objetos ou arrays, e assim por diante
- Agora podemos criar os elementos, com formatos identificados automaticamente
  - `raiz["sensor"] = "gps";`
  - `raiz["time"] = 1351824120;`
- Valores decimais devem ser especificados com o número de casas decimais desejadas, caso contrário são usadas apenas 2 casas. O exemplo abaixo usa 4 casas:
  - `raiz["pi"] = double_with_n_digits(3.1415, 4);`
  - Para acrescentar um array ou outro objeto, é necessário usar um método especial
  - `JsonArray& vetor = raiz.createNestedArray("vetor");`
  - `vetor.add("José"); vetor.add(48.756080, 6); //Usa 6 casas`
  - `JsonObject& obj = raiz.createNestedObject("obj");`
- Finalmente, imprimimos a string JSON resultante na porta serial ou em uma string do C
  - `raiz.printTo(Serial); //manda o resultado pela porta serial`

Lendo um JSON

- Para decodificar um JSON é ainda mais fácil. Basta reservar a memória, checar possíveis erros e resgatar os valores desejados
  - ```
    char json[] = "{
    \"sensor\":\"gps\",\"time\":1351824120,
    \"data\":[48.756080,2.302038]}";
    ```
  - ```
    StaticJsonBuffer<200> jsonBuffer;
    ```
  - ```
    JsonObject& raiz =
    jsonBuffer.parseObject(json);
    ```
  - ```
    if (!raiz.success()) {/* Tratar o erro*/}
    ```
- Capturando os valores:
  - ```
    const char* sensor = raiz["sensor"];
    ```
  - ```
    long time = raiz["time"];
    ```
  - ```
    double latitude = raiz["data"][0];
    ```
  - ```
    double longitude = raiz["data"][1];
    ```
- Exercício: criar um programa que manda a luminosidade, temperatura e umidade no formato JSON pela porta serial, enquanto aceita comandos para o LED usando JSON. Escolha o nome dos campos a serem preenchidos.

Lendo o JSON da porta Serial e mandando a luminosidade

```
#include <ArduinoJson.h>
const int LED = 3;
const int LUZ = A1;
const int TAMANHO = 200;
void setup() {
  Serial.begin(9600);
  Serial.setTimeout(10); //1000ms é muito tempo
  pinMode(LED,OUTPUT);
}
void loop() {
  if (Serial.available() > 0) {
    //Lê o texto disponível na porta serial:
    char texto[TAMANHO];
    Serial.readBytesUntil('\n', texto, TAMANHO);
    //Grava o texto recebido como JSON
    StaticJsonBuffer<TAMANHO> jsonBuffer;
    JsonObject& json = jsonBuffer.parseObject(texto);
    if(json.success() && json.containsKey("led")) {
      analogWrite(LED, json["led"]);
    }
  }
  StaticJsonBuffer<TAMANHO> jsonBuffer;
  JsonObject& json = jsonBuffer.createObject();
  json["luz"] = analogRead(LUZ);
  json.printTo(Serial); Serial.println();
  delay(1000);
}
```

REFERÊNCIAS

- http://www.telecom.uff.br/pet/petws/downloads/tutoriais/arduino/Tut_Arduino.pdf
- http://arduino.cc/en/Reference/HomePage
- https://www.arduino.cc/en/Reference/StringObject
- http://json.org

FIAP