Texto de Revisão da Disciplina
FORMAÇÃO PROFISSIONAL EM COMPUTAÇÃO

Nesta disciplina, revisamos alguns dos conhecimentos adquiridos no curso de Ciência ou Engenharia da Computação. A primeira dúvida que surge para os estudantes da computação é como se beneficiarão dos conhecimentos de linguagem de programação. Para sanar essa dúvida, discutimos sobre a importância do pensamento computacional, o qual nos proporciona a capacidade de sistematizar, representar, analisar e resolver problemas. No decorrer das semanas, reforçamos e acessamos conhecimentos advindos de diversas áreas da computação, conceitos estes importantes para conduzir a uma solução computacional de problemas. O intuito desse caminho percorrido é que você tenha uma visão geral do curso e compreenda como todo o conhecimento adquirido se encaixa na sua futura vida profissional. Espera-se que você compreenda como sair de uma ideia, utilizando o raciocínio lógico com pensamento computacional e a lógica de programação, criar uma solução, usando requisitos, projeto e uma linguagem de programação. Para validar os resultados alcançados, foi proposto utilizar os conhecimentos da aula e gerar códigos para resolver os problemas atuais como geração de WebService, ciência de dados e IOT.

1. Pensamento Computacional, Evolução da Computação, Conceitos de Paradigmas de Computação

Apesar do termo **Pensamento Computacional** estar sendo muito utilizado nos últimos anos, a ideia remonta, pelo menos, à década de 1950. O termo "pensamento computacional" foi usado pela primeira vez por Seymour Papert em 1980 em seu livro *Mindstorms: Children, computers, and powerful ideas*, e posteriormente no artigo "*An Exploration in the Space of Mathematics Educations*" publicado em 1996. Os conceitos de pensamento computacional podem suportar a resolução algorítmica de problemas complexos e é frequentemente usado para realizar melhorias na eficiência. Segundo Seymour, o pensamento computacional é a "Capacidade de sistematizar, representar, analisar e resolver problemas".

O pensamento computacional se baseia em quatro pilares que orientam o processo de solução de problemas:

- **Decomposição:** processo que divide os problemas em partes menores para facilitar a resolução, desenvolvimento e gerenciamento.
- **Reconhecimento de Padrões:** os padrões são similaridades ou características que alguns problemas compartilham. Quanto mais padrões encontrarmos, mais fácil e rápida será a nossa tarefa geral de solução de problemas.

- **Abstração:** filtragem e classificação dos dados, criando mecanismos que permitam separar apenas os elementos essenciais em determinado problema, ignorando detalhes irrelevantes. Permite criar uma representação (ideia) do que está se tentando resolver.
- **Algoritmo:** é uma sequência finita de etapas (passos), cada qual executável em um tempo finito, por um agente computacional, natural (humano) ou sintético (computador). Um algoritmo é um plano, uma estratégia ou um conjunto de instruções ordenadas para a solução de um problema ou execução de uma tarefa.

Mais recentemente, Jeannette Wing em 2010 publicou o artigo "*Computational thinking*" no *Communications of the ACM*, neste estudo ela define o pensamento computacional como "processos de pensamento envolvidos na formulação de um problema e que expressam sua solução ou soluções eficazmente, de tal forma que uma máquina ou uma pessoa possa realizar".

A proposta de pensamento computacional não foi pensada unicamente para pessoas da área da computação, é mais um desafio para que todas as pessoas apliquem técnicas e processos da computação para o dia a dia. No livro *lógica de programação para iniciantes*, o autor fez a seguinte afirmação:

> Aprender programação ajuda muito a forma como as pessoas pensam, principalmente por desenvolver a disciplina da lógica, que é um campo da filosofia criado por Aristóteles que cuida das regras do bem pensar, ou do pensar correto, sendo, portanto, um instrumento do pensar. A lógica guia o raciocínio humano através de argumentos para chegar a conclusões de verdade (ALVES, p. 03).

A internet é uma plataforma que pode sustentar os pilares de pensamento computacional. O surgimento da internet é um marco histórico e é defendida por muitos como a evolução da computação. A internet desde sua idealização em 1969 vem passando por grandes mudanças. Porém, no início, o objetivo era de cunho militar. Os Estados Unidos a projetaram como uma ferramenta para descentralização com o Pentágono para evitar possíveis ataques e perdas de documentos pelo governo. Esse cenário é muito diferente da utilização dos dias atuais.

Em 1962, já se falava na criação de uma rede intergaláctica de computadores pelo engenheiro do instituto Tecnológico de Massachusetts (MIT) Joseph Licklider; mas, apenas sete anos depois que se deu realmente início ao nascimento da internet. Nos anos 70, Net Vilton Cerf e Bob Kahn propuseram os protocolos e padrões "TCP/IP". O uso do termo

internet se deu apenas em 1974 com a primeira publicação do TCP, ela sendo assinada na Universidade de Stanford por Vinton Cerf, Yogem Dalal e Carl Sunshine.

Nos anos 80, surgiram as redes baseadas em TCP/IP, assim, todos os computadores que ainda faziam utilização da rede antiga optaram pelo sistema de pacotes da nova rede. Com a criação do Protocolo de Internet, todas as redes com o mesmo endereço de IP poderiam estar conectando para troca de arquivos e para envio de mensagens. Mas a grande popularização da internet se deu apenas em 1988, quando houve a abertura das redes para fins comerciais. No final dos anos 80, os Estados Unidos resolveram comercializar utilizando a internet.

Criado por Tim Bernes-Lee, o World Wide Web (WWW ou Web) surgiu com o intuito de ajudar a Organização Europeia em suas investigações nucleares, assim várias pessoas poderiam estar trabalhando juntas no mesmo documento. Após a primeira publicação do uso do WWW, o interesse mundial em torno da internet cresceu muito rapidamente, incentivando o interesse por equipamentos pessoais (*Personal Computers*). Nos anos 2000, começou a grande novidade da web com a facilidade da aquisição de computadores e com a internet para o público em geral, assim, a tecnologia foi evoluindo e tendo grandes avanços significativos, da internet discada à banda larga à criação da rede sem fio, 3G,4G e agora 5G.

Hoje em dia, a internet está em todos os objetos e lugares e podemos afirmar que é uma necessidade diária tanto para o uso empresarial, quanto para o uso doméstico. Do ponto de vista de profissionais da área da computação, é necessário além de ter conhecimento das infraestruturas e dos protocolos de comunicação, dominar ferramentas e linguagens de programação que permitam desenvolver as soluções que o mundo moderno demanda.

Para isso nesta disciplina, vimos os seguintes conceitos básicos da computação, principalmente na área de desenvolvimento:

i. **Linguagens de programação**: são linguagens usadas para a comunicação com o computador.
ii. **Desenvolvimento de software**: é o ato de elaborar e implementar um sistema computacional.
iii. **Ciência de dados**: é uma área interdisciplinar voltada para o estudo e a análise de dados econômicos, financeiros e sociais, estruturados e não-estruturados, que visa a extração de conhecimento, detecção de padrões e/ou obtenção de *insights* para possíveis tomadas de decisão.

iv. **Programação Orientada a Objetos**: é um modelo de análise, projeto e programação de software. baseado na composição e interação entre diversas unidades chamadas de 'objetos'.

Compreendemos que aprender linguagens de programação das diversas categorias, aumenta nossa capacidade de usar diferentes construções ao escrevermos programas. Permite-nos escolher linguagens para projetos de forma mais adequada e inteligente e facilita o aprendizado de novas linguagens. O projeto e a avaliação de uma linguagem de programação são altamente dependentes das variáveis do projeto e do domínio. Entre os critérios mais importantes para a avaliação de linguagens estão a legibilidade, a facilidade de escrita, a confiabilidade e o custo.

O objetivo do conteúdo visto foi compreender o processo de resolução de problemas baseado nos conceitos do Pensamento Computacional e ter as ferramentas necessárias para uma avaliação crítica de linguagens de programação existentes e linguagens a serem propostas no futuro.

2. Introdução ao Google Colab para implementação em Python e Ciência de Dados

A decisão sobre a escolha da linguagem de programação a ser utilizada em um projeto leva em conta diversos aspectos, um deles diz respeito ao tempo de execução de um programa. Python é uma linguagem de programação de alto nível, interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Python é uma linguagem de propósito geral de alto nível e que suporta diversos paradigmas. Depois de aprender conceitos de programação em outras linguagens, Phyton é uma excelente adição aos seus conhecimentos.

Além do Python, utilizamos também um ambiente de desenvolvimento, IDE para Python. Um IDE é um pacote de software que consolida as ferramentas básicas necessárias para escrever e testar softwares. Os IDEs aumentam a produtividade reduzindo o tempo de configuração de ambiente, aumentando a velocidade das tarefas de desenvolvimento, mantendo os desenvolvedores atualizados e padronizando o processo de desenvolvimento.

Vimos também dois projetos de ambientes de desenvolvimento on-line que proporcionam aos usuários conjunto de bibliotecas e ferramentas para facilitar a integração e desenvolvimento em Python. Em 2001, é disponibilizado o interpretador interativo IPython com o qual é possível administrar computação paralela usando comunicação assíncrona e/ou MPI e permite customização e flexibilidade para executar diretamente

códigos Python. O IPython também oferece uma interface chamada Notebook (NB). Notebook é formada por dois componentes, o primeiro baseado em JSON para compartilhar códigos Python e o segundo em RTF (*Rich Text Format*) para publicar os códigos on-line. O notebook tem a proposta de trazer uma interface limpa e acessível para não desenvolvedores.

Baseado no IPython, foi desenvolvido o projeto Jupyter, criado para desenvolver software de código aberto, padrões abertos e serviços para computação interativa em dezenas de linguagens de programação O Projeto Jupyter suporta ambientes de execução em dezenas de linguagens de programação. O nome do projeto é uma referência às três principais linguagens de programação suportadas por Jupyter, Julia, Python e R.

Outro projeto baseado nos conceitos do IPython é Google Colab, ou Google Collaboratory. O Colab é um serviço de armazenamento em nuvem de notebooks voltados à criação e execução de códigos em Python, diretamente em um navegador, sem a necessidade de nenhum tipo de instalação de software em uma máquina. O Google Colab é hospedado pelo Jupyter Notebook, e possibilita o comportamento de código + anotações e *markdown* do Colab. A diferença entre utilizar o Jupyter Notebook, instalado em sua máquina, e utilizar o Google Colab é que, com o Colab, você não irá necessitar realizar configurações no seu próprio computador: basta acessá-lo e começar a programar. Colab também permite que seus usuários utilizem notebooks Jupyter em seu ambiente.

Com o Google Colab, é possível ler, desenvolver e rodar códigos e Rich Texts em documentos interativos que agrupam células de códigos (notebooks), compartilhá-los com outros programadores, modificá-los a qualquer momento e mantê-los salvos de maneira totalmente on-line. Todo o poder computacional utilizado para executar o software que você escrever é fornecido pela nuvem de computadores da Google, possibilitando processar uma grande quantidade de dados. Desde sua rápida e extremamente compreensível interface até a potência de seus ambientes de execução, esta ferramenta ainda facilita o compartilhamento de recursos e o trabalho em conjunto com outros programadores praticamente em tempo real. As principais vantagens do uso do Colab são:

- Nenhuma configuração: o Google Colab não exige absolutamente nenhuma configuração externa para o seu uso.
- Acesso gratuito a GPUs: o hardware padrão da máquina criada pelo Colab para um notebook é o CPU, mas é possível acessar gratuitamente um GPU com uma simples configuração interna. A GPU, para alguns processamentos, é muito mais eficiente

que a CPU, principalmente em cálculos avançados de *deep learning*, entretanto, é sempre bom conhecermos estas opções e possibilidades.

- Bibliotecas pré-instaladas: um dos maiores atrativos do Python é a grande quantidade de bibliotecas e *frameworks* disponíveis para se trabalhar com essa linguagem, e no Google Colab, esta realidade não é diferente. Para tornar sua utilização ainda mais simples, o Colab já conta com centenas de bibliotecas pré-instaladas, o que reduz seu tempo de importação e facilita o aproveitamento de seus recursos.

3. Projeto de Desenvolvimento de Software, Algoritmo e Implementação

Desde o início dos tempos, no mundo todo, trabalhos grandiosos na área da construção principalmente são realizados. O gerenciamento de projetos não é algo novo, em toda história, podemos encontrar grandes feitos que só foram possíveis devido a um trabalho conjunto, envolvendo muitas pessoas. No dicionário, existem diversos significados para essa palavra, entre eles, que projeto é uma vontade, desejo ou intenção de realizar algo. É muito comum que relacionemos com algo futuro como, por exemplo, uma viagem ou a compra de um imóvel.

Segundo o PMBOK, um projeto é um esforço temporário que tem como finalidade um resultado único e possui recursos delimitados. PMBOK, é o guia mais utilizado quando o assunto é gestão de projetos. Ele foi criado pelo PMI (*Project Management Institute*), uma organização sem fins lucrativos que tem o objetivo de disseminar as melhores práticas da gestão de projetos pelo mundo todo.

Um projeto deve sempre entregar algo novo. Um produto, serviço ou resultado exclusivo, algo que a organização não possua e deseje alcançar. Para isso, precisamos dividir o nosso objetivo final em objetivos menores, que possam ser realizados em períodos aceitáveis. Um projeto é temporário, pois deve ter datas de início e fim definidas é iniciado com a intenção de criar um produto ou serviço, ou apenas aprimorar algo já existente. Deve haver um escopo inicial, isto é, a descrição (mesmo que parcial) do trabalho que precisa ser feito. Estipular custos e recursos no início do projeto como força de trabalho, materiais que serão utilizados, infraestrutura, verbas e prazos.

Antes de se desenvolver um software, deve-se compreender o "sistema", em que ele residirá, e não o pensar de forma isolada. Deve-se determinar o objetivo geral do sistema e os papéis do hardware, software, pessoal, bases de dados e outros elementos. Requisitos operacionais precisam ser obtidos, analisados, especificados, modelados, validados e

gerenciados. Software está em quase tudo, nos transportes, na área de saúde, telecomunicações, na área militar, industrial, entre outros.

A maioria do **software** é feita sob medida/encomenda ao invés de ser montada a partir de módulos/componentes de programação existentes. O software é desenvolvido ou projetado por engenharia que por não ser um sistema físico, não é fabricado no sentido clássico, ele é desenvolvido ou passa por um processo de engenharia.

Quando um componente de hardware se desgasta, é substituído por uma peça sobressalente; em contrapartida, no software, não há peças sobressalentes. Falhas de software indicam erros no projeto ou na codificação; por isso, a manutenção de software é muito mais complexa que a manutenção de hardware.

A **engenharia de software** é abordagem sistemática, disciplinada e capaz de ser medida ao longo de um processo de construção de um software:

- Tipo e a ordem de execução de atividades na construção do software;
- Modelos gráficos descritivos do software;
- Diretrizes com base em boas práticas de projeto.

Processos vem sendo propostos pela indústria e a academia de diferentes países. Para todos os programas construídos, há a necessidade de se entender os requisitos e o processo de negócio do contratante. Os elementos do mundo real envolvidos no desenvolvimento e manutenção de um produto de software incluem os recursos, ferramentas, atividades, artefatos e organização.

Processo de software é um conjunto de atividades para especificação, projeto, implementação e teste de software (Sommerville).

Modelo de Processo de Software é uma representação abstrata de um processo**.** É a descrição de um processo, a partir de uma perspectiva particular.

O **ciclo de vida** de desenvolvimento é um conjunto de fases voltadas para o desenvolvimento do produto, serviço ou resultado e podem ser classificadas como:

- Preditivo: Neste modelo, o escopo, o prazo e o custo do projeto são determinados nas fases iniciais e qualquer mudança necessária deve ser cuidadosamente avaliada para que não se distancie do previsto inicialmente.
- Interativo: Recomendável para projetos que já possuam a definição do que será entregue de maneira clara, porém, com prazo e custos ainda desconhecidos. Cada

volta no ciclo de vida chamamos de iteração. A cada iteração, a equipe compreende melhor o problema, e as estimativas de prazo e custos são atualizadas.

- Incremental: A entrega é realizada por partes e pouco a pouco o produto final vai sendo formado. A cada novo conjunto de funcionalidades que é criado, um incremento é entregue. O ciclo de vida incremental também utiliza iterações para marcar cada entrega de um incremento. O produto, serviço ou resultado só pode ser considerado completo após a última iteração.
- Adaptativos (ágeis): Este modelo de ciclo de vida utiliza características dos Iterativos e Incrementais somados a conceitos de metodologias ágeis, obtendo, assim, resultados melhores em ambientes com muitas incertezas. O detalhamento do escopo é definido e aprovado antes do início de cada iteração.
- Híbrido: É a combinação do ciclo de vida preditivo com o adaptativo. Enquanto o preditivo espera um ambiente completamente conhecido e o adaptativo um ambiente totalmente incerto, o modelo híbrido busca encontrar um equilíbrio entre os dois.

Para cada situação de projeto, devem ser utilizados tipos diferentes de ciclo de vida, pois cada projeto possui características específicas, cabendo à equipe de gerenciamento definir qual modelo de ciclo de vida se encaixará melhor ao projeto. Durante essa decisão, deve-se levar em conta o grau de flexibilidade admitido para cada projeto, evitando surpresas, como, por exemplo, a variação de fatores externos.

A Gerência de Projetos envolve o planejamento, monitoramento e controle das pessoas e processos que ocorrem conforme evolui o desenvolvimento do software (produto). O gerente que esquece que o trabalho de engenharia de software está intensamente relacionado com as **pessoas** nunca terá sucesso na gerência de projetos.

O gerente que não incentiva uma compreensiva comunicação com o cliente, logo no início do projeto, corre o risco de construir uma solução muito boa para um problema errado. O gerente que dá pouca atenção ao **processo** corre o risco de inserir no vácuo, métodos e ferramentas competentes. O gerente que trabalha sem um sólido plano de **projeto** arrisca o sucesso do produto.

Gerência de projeto é uma atividade intensivamente relacionada com as pessoas e por isso desenvolvedores competentes nem sempre se tornam bons líderes de equipe. Um líder de equipe deve ter uma mistura de habilidades:

- Deve estimular as habilidades dos integrantes da equipe, para assim desenvolver confiança e aperfeiçoar as formas de comunicação;

- É responsável por intermediar a comunicação, buscando sempre proteger a equipe e mantê-la unida com um verdadeiro time;
- Capacidade de Liderança;
- Eficiência na comunicação;
- Poder de negociação;
- Disciplina e organização;
- Persistência;
- Visão a logo prazo;
- Tomada de decisão;
- Desenvolver a equipe;
- Proatividade;
- Motivação e otimismo.

Temos que tomar muito cuidado em não confundir projeto com processo. **Processo** é tudo aquilo que tem atividades repetitivas, que possui um esforço contínuo sem tempo determinado e que não possui um resultado único. Por exemplo: uma dona de casa organiza e limpa sua casa em dias intercalados. Podemos chamar isso de processo, pois as tarefas que ela faz são as mesmas, não mudam.

Supondo que ela decida fazer uma reforma na casa que terá duração de dois meses. Neste caso estamos falando de um **projeto**, pois será algo temporário e com um resultado exclusivo. E depois da reforma, quando a casa já estiver pronta, a dona de casa poderá voltar a fazer normalmente os seus processos de limpeza como antes.

4. Padrões da Web, HTML, JSON e Python

Os padrões web são documentos que definem e regulamentam as tecnologias em uso na web, com o objetivo de informar aos desenvolvedores as melhores práticas na criação de sites e sistemas, garantindo aos usuários acessibilidade e compatibilidade. As especificações e guias técnicos são desenvolvidos e propostos pelo W3C (World Wide Web Consortium).

Estes padrões são criados considerando aspectos de acessibilidade, privacidade, segurança e internacionalização. Eles refletem as visões de diferentes indústrias e os interesses de diversos setores ao redor do mundo, equilibrando velocidade, prestação de contas e qualidade. Os padrões são revisados e testados extensivamente por grupos de dentro e de fora do W3C, e são disponibilizados gratuitamente sob licenças livres de *royalties*.

Quando seguidos, os padrões permitem que os benefícios da web estejam disponíveis para todos, independentemente dos sistemas e dispositivos que utilizam, da infraestrutura de rede, do idioma, da cultura, da localização geográfica, ou de limitações físicas ou cognitivas.

Para construir sites, precisa-se saber sobre HTML, que é a tecnologia fundamental usada para definir a estrutura de uma página da web. HTML é usado para especificar se o conteúdo da web deve ser reconhecido como um parágrafo, lista, título, *link*, imagem, reprodutor de multimídia, formulário ou um dos muitos outros elementos disponíveis ou até mesmo um novo elemento que você definir. A força da plataforma web se baseia em muito mais do que apenas HTML e CSS: há várias outras tecnologias que o W3C e seus parceiros estão criando e aprimorando constantemente, elas são:

- HTML - a tecnologia fundamental usada para definir a estrutura de uma página da web. HTML é usado para especificar se o conteúdo da web deve ser reconhecido como um parágrafo, lista, título, link, imagem, reprodutor de multimídia, formulário ou um dos muitos outros elementos disponíveis ou até mesmo um novo elemento que você definir.
- *Cascading Stylesheets* - ou CSS - é a primeira tecnologia que você deve começar a aprender depois do HTML. Enquanto o HTML é usado para definir a estrutura e a semântica do seu conteúdo, o CSS é usado para estilizá-lo e defini-lo. Por exemplo, você pode usar CSS para alterar a fonte, cor, tamanho e espaçamento de seu conteúdo, dividi-lo em várias colunas ou adicionar animações e outros recursos decorativos.
- JavaScript é uma linguagem de programação que permite implementar coisas complexas em páginas da web. Cada vez que uma página da web faz mais do que apenas ficar lá e exibir informações estáticas para você ver - exibindo atualizações de conteúdo oportunas, mapas interativos, gráficos 2D/3D animados, jukeboxes de vídeo de rolagem ou mais - você pode apostar que provavelmente o JavaScript está envolvido.
- Ajax é o uso metodológico de tecnologias como Javascript e XML, providas por navegadores, para tornar páginas Web mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações.
- XML é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais. Seu propósito principal é a facilidade de compartilhamento de informações por intermédio da internet.

- JSON (JavaScript Object Notation - Notação de Objetos JavaScript) é uma formatação leve de troca de dados. Para seres humanos, é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar. Está baseado em um subconjunto da linguagem de programação JavaScript, Standard ECMA-262 3a Edição - Dezembro - 1999. JSON é em formato texto e completamente independente de linguagem, pois usa convenções que são familiares às linguagens C e familiares, incluindo C++, C#, Java, JavaScript, Perl, Python e muitas outras. Estas propriedades fazem com que JSON seja um formato ideal de troca de dados.

Em diferentes localidades da web, você pode observar que existe algum site ou aplicação utilizando *frameworks* JavaScripts e bibliotecas da linguagem. Uma biblioteca JavaScript é um "pedaço de código" reutilizável que pode servir de base para a implementação de outros códigos. A ideia de utilização de uma biblioteca é a de reutilizar códigos existentes para não perder tempo escrevendo códigos que outro programador já escreveu. Em outras palavras, é um arquivo de JavaScript que contém um monte de funções, e essas funções realizam alguma tarefa útil para sua aplicação web.

Um framework JavaScript é uma abstração que combina códigos comuns para serem reutilizados. Ele tem como principal objetivo resolver problemas recorrentes com uma abordagem genérica. Assim, o desenvolvedor pode-se concentrar em resolver seus problemas ao invés de ficar reescrevendo linhas de código.

jQuery é, de longe, a principal biblioteca de JavaScript que existe atualmente. Foi lançado em 2006 por John Resig e tem sido usado em sites em todo o mundo, mais de 50% de todos os sites ativos usam jQuery.

Quando se fala em frameworks e bibliotecas JavaScript, é impossível não citar o React. Isso acontece por inúmeros fatores. O primeiro é o fato dele pertencer a uma família muito conhecida por todos. Criado pela equipe do Instagram, hoje ele é mantido pelo grupo Facebook. Além disso, o React é conhecido por ser uma biblioteca para criar interfaces eficientes. Dessa forma, ele se posiciona como uma ferramenta para o *front-end*. Isso se traduz, para o desenvolvedor, como uma maior facilidade na hora de ter um código limpo e reutilizável.

O Bootstrap é um dos principais frameworks JavaScript web e *open-source* do mercado e é capaz de oferecer padrões para o desenvolvimento HTML, CSS e JavaScript. Foi desenvolvido em 2010 e lançado oficialmente em 2011 pelos profissionais do Twitter, Mark Otto e Jacob Thornton.

O AngularJS é um dos frameworks JavaScript mais importantes quando falamos de desenvolvimento web mantido pela Google. O Angular (sem JS no nome) é um *framework* multiplataforma também mantido e desenvolvido pela Google. Ele é uma reescritura completa do antigo AngularJS, citado anteriormente, e foi escrito usando TypeScript. O estilo da arquitetura mudou para ser baseado em componentes e, devido ao TypeScript, o Angular 2 recebeu um novo compilador incorporado.

O Node.js, é a opção mais potente de todas. Isso acontece porque o Node.js foi criado para ser escalável. Isso significa que toda sua composição de código foi estruturada para proporcionar ótima performance, mesmo com uma utilização pesada e com muitos acessos ao mesmo tempo.

Nas duas tabelas a seguir, tem-se uma lista de vantagens e de desvantagens dos *frameworks* respectivamente.

Jquery	React	BootStramp	AngularJS
Utiliza seletores CSS para localizar elementos da estrutura de marcação HTML	Facilidade para criar aplicações web dinâmicas	Uma estrutura consistente que suporta grande parte dos navegadores	Os componentes podem ser reutilizados em diferentes partes da aplicação
Vasta quantidade de plug-ins	Componentes reutilizáveis	Estruturas e estilos responsivos	Prototipagem mais rápida de aplicativos
Ocupa pouco espaço em disco.	Melhorias na performance por conta do virtual DOM	Foco na experiência do usuário	Reduz a carga das CPUs do servidor

Jquery	React	BootStramp	AngularJS
O código-fonte não é tão protegido	O alto ritmo de desenvolvimento	HTML não é nativamente compatível	Dependência do JavaScript
As aplicações precisam de um servidor para gerenciar sessões	Abrange apenas as camadas da interface do usuário	Os estilos podem gerar muitos resultados desnecessários	Aplicações complexas apresentam atrasos
Podem ocorrer conflitos entre scripts dificultando a depuração	Pouca documentação	O JavaScript está vinculado ao jQuery	Pouca documentação

Em uma pesquisa realizada por Anne Coifman em 2020, com base na aceitação e utilização pelos profissionais e gestores da área de desenvolvimento de software entre os

seguintes dez *frameworks* mais importantes estão *frameworks* JavaScript já mencionados para desenvolvimento Web: Bootstrap, Angular, React e JQuery Mobile.

5. Framework Python, APIs e Web Services

Como vemos a programação para Web ganha cada vez mais espaço nesse imenso mercado virtual. As linguagens de programação estão mais poderosas e ao mesmo tempo produtivas e com excelente desempenho. E como vimos os *frameworks* são importantíssimos, pois seu objetivo é fornecer ferramentas para facilitar o desenvolvimento do projeto em uma base estável.

Acompanhando o mercado, o Python também oferece ótimas opções para a programação Web. E é claro que existem vários *frameworks* disponíveis em Python para desenvolvimento. Alguns que podemos mencionar são:

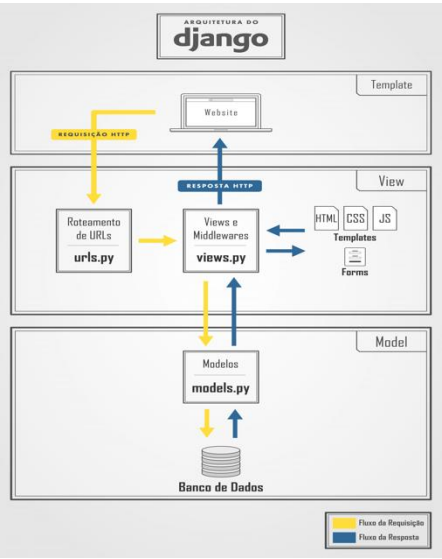
- O **Django** é um *framework* de código aberto e escrito em Python que permite aos desenvolvedores implementar aplicativos complexos de maneira rápida e eficiente. Ele apresenta uma estrutura de alto nível que otimiza o desenvolvimento de aplicativos, possuindo uma variedade de bibliotecas para as quais exige uma menor necessidade de codificação e uma grande reutilização de componentes, o que garantirá consistência a projetos de médio e grande porte.
- O **Flask** também é um *framework* web escrito em Python, baseado nas bibliotecas WSGI Werkzeug e Jinja2, e inspirado na estrutura Sinatra Ruby. O principal objetivo do framework é prover um modelo simples para desenvolvimento web, ao mesmo tempo que tem a flexibilidade no uso da linguagem Python. É chamado de micro-*framework* por manter um núcleo simples, mas estendível.
- O **web2py**, publicado em 2007 sob a licença GNU, é um *framework* de código aberto e escrito em Python, que permite aos desenvolvedores programarem conteúdo web dinâmico, reduzindo tarefas tediosas, como, por exemplo, o desenvolvimento de formulários web do zero, mesmo possibilitando a construção do zero se necessário. O projeto do web2py foi inspirado nos *frameworks* Ruby on Rails e Django, sendo originalmente projetado como uma ferramenta com ênfase na facilidade de uso.
- O **CherryPy**, publicado em 2002 sob a licença BSD, é um *framework* de código aberto que incorpora um servidor de web com pool de threads, estrutura de instalação e módulos. Sua estrutura permite implementar aplicativos complexos de maneira rápida e eficiente, possuindo para tanto uma variedade de bibliotecas, exigindo uma menor codificação e uma grande reutilização de componentes, garantindo assim uma

consistência nos projetos. É um framework para desenvolvimento ágil de aplicações web, orientados a objetos, permitindo o uso de qualquer tipo de tecnologia para acesso a dados, modelos etc.

- O **Bottle**, publicado em 2009 sob a licença MIT, é um micro-framework WSGI rápido, simples e leve projetado e distribuído como um módulo de arquivo único e sem dependências além da biblioteca padrão do Python.

A primeira opção dos desenvolvedores Python é o Django pela facilidade de desenvolvimento e manutenção de grandes sites, como, por exemplo, Pinterest, Instagram, Bitbucket, Mozilla, Disqus e The Washington Times. O Django é um *framework* gratuito e *open source* para desenvolvimento web tão alto nível que já traz para programadores uma solução ORM, simplificando ao máximo o acesso a dados no banco de dados. Outro diferencial é a interface de administração por ele fornecida, a Django Admin.

O Django apresenta uma estrutura de alto nível que otimiza o desenvolvimento de aplicativos, possuindo uma variedade de bibliotecas para as quais exige uma menor necessidade de codificação e uma grande reutilização de componentes, o que garantirá consistência a projetos de médio e grande porte. Alguns de seus principais recursos são: o mecanismo de autenticação e autorização, interface administrativa, URLs amigáveis, sistema de templates, cache integrado ao memcached, o roteamento de URLs, suporte para aplicações multi-idioma (internacionalização).



A arquitetura do Django é relativamente simples. Basicamente, um projeto Django possui como padrão de projeto o MTV (Model, Template, View), que servem para:

- Model: Mapeamento do banco de dados para o projeto;
- Template: Páginas para visualização de dados. Normalmente, é aqui que fica o HTML que será renderizado nos navegadores;
- View: Lógica de negócio. É aqui que determinamos o que irá acontecer em nosso projeto.

Além de implementar os modelos MTV (Model-Template-View) visando segmentar as funcionalidades da aplicação e o ORM (*Object Relational Mapper*) para o mapeamento de seus objetos para tabelas de bancos de dados. A estrutura do Django sustenta vários bancos de dados como PostgreSQL, MySQL e Oracle, e mesmo suportando oficialmente bancos NoSQL, há uma série de projetos e *forks* que permitem o seu funcionamento.

O Django possui uma extensão indicada para a criação de web services RESTful: o Django REST. Com ele podemos tratar com facilidade os verbos e códigos de status do protocolo HTTP, manipular dados no formato JSON e XML, além de outros recursos que aceleram o desenvolvimento desse tipo de aplicação. Uma API, ou Interface de Programação de Aplicativo, é um conjunto de definições e protocolos que permitem que um aplicativo se comunique com outro aplicativo. Em geral, quando falamos sobre APIs, provavelmente estamos falando sobre APIs da web [APIs que são acessíveis pela internet]. Este nem sempre é o caso, porém. As APIs podem ser expostas por meio de arquivos locais (como um arquivo JAR em um programa Java, arquivo .h em programas C / C ++, etc.) para permitir que dois aplicativos locais se comuniquem entre si. Isso não requer uma rede, pois os dois aplicativos estão se comunicando em um único dispositivo.

Um serviço da Web é uma forma de duas máquinas se comunicarem em uma rede. Um servidor da Web em execução em um computador atende a solicitações de outros computadores. Quando uma solicitação de outro computador é recebida, em uma rede, o serviço da Web retorna os recursos solicitados. Este recurso pode ser JSON, XML, um arquivo HTML, imagens, arquivos de áudio etc. É importante observar o requisito da solicitação ser feito por uma rede.

Nem todas as APIs podem ser acessadas pela Internet (uma rede), enquanto os Web Services devem sempre ser acessados por meio de uma rede. Essa é a principal diferença entre as duas tecnologias. Por tudo isso podemos afirmar que "Todo Web Service pode ser considerado API, MAS não toda API é um Web Service."

6. Internet das Coisas (IoT) e os protocolos MQTT e REST

Na década de 70 e 80, a Internet era basicamente uma forma de conectar computadores. Os anos 90 e 2000, marcaram a Internet como a ferramenta de conexão entre as pessoas. Agora, a ênfase está mudando para pensar e projetar a forma de conectar tudo a Internet. A internet das coisas (ou em Inglês: *Internet of Things* – IoT) é um conceito que pretende conectar objetos físicos que possuam tecnologia embarcada com sensores e conexão para coletar dados, transmiti-los, receber dados e enfim fornecer serviços. Esse conceito tem se tornado cada vez mais possível, devido aos avanços de barateamento e popularização das tecnologias para sistemas embarcados e microeletrônica.

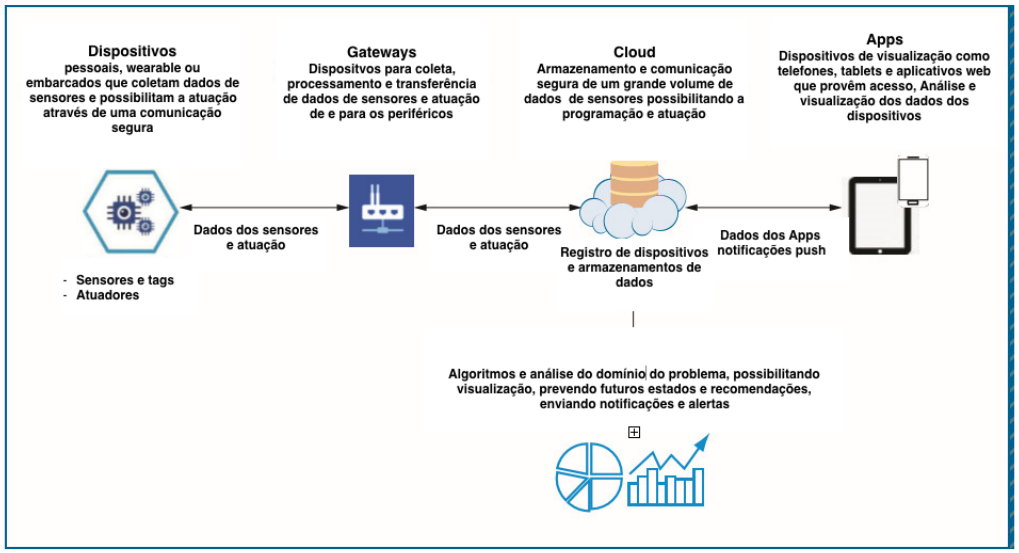
Grande parte da pesquisa atual de IoT é destinada aos problemas de aquisição de dados, análise destes dados em tempo real e offline, aprendizado de máquina, visualização de dados e outros tópicos importantes para *big data*. Este enfoque não nos surpreende, dado ao enorme potencial de negócio que emerge da habilidade de coletar dados de milhões de dispositivos e sensores, processá-los e combiná-los de forma a fornecer novas informações sobre o comportamento da população e diversos outros fenômenos do mundo real. Nesse contexto, as definições de alguns termos é:

- **Coisa:** é um objeto do nosso dia a dia colocado em nosso ambiente diário. A coisa pode ser um carro, uma geladeira, mas também pode ser abstraído a uma casa ou uma cidade, dependendo do caso de uso;
- **Dispositivo:** um sensor, atuador ou tag. Geralmente o dispositivo é parte de uma coisa. A coisa processa informações de um contexto e informa sobre os dados coletados para outras coisas. Além disso, a coisa pode passar ações para atuadores.

A evolução do hardware e o fácil acesso a chips integrados poderosos e de baixo custo, possibilita embarcar conectividade, máquinas virtuais e ambientes de execução de linguagens dinâmicas em todos os dispositivos. Desta forma, os dispositivos do nosso dia a dia como lâmpadas, maçanetas, sistemas de ar-condicionado, regadores de jardim, aspiradores, escova de dentes e a pia da cozinha se tornarão conectadas e programáveis de forma dinâmica. O termo "*Internet of Things*" (Internet das Coisas) não é novo. Nos últimos 20 anos, professores do MIT descrevem um mundo onde as coisas (dispositivos ou sensores) estão conectados e compartilham dados. No princípio, os conceitos de IoT surgiram baseados em tecnologias como RFID e redes de sensores *wireless*, e agora se expandiram

bluetooth, sigbi, 3G, 5G. Além disso, recentemente IoT se expandiu para vários domínios, incluindo saúde, transportes, energia, varejo e automação de processos.

O que é interessante em relação à grande gama de plataformas de IoT disponíveis é sua similaridade conceitual. Apesar de uma aparente diversidade e um grande número de fornecedores focados no mercado de IoT, uma arquitetura *end-to-end* comum está emergindo das soluções de IoT, pois um grande número de elementos é comum a todos os sistemas. Dispositivos (ou periféricos) são elementos de hardware que coletam dados de sensores ou realizam determinadas ações e que possuem capacidade de se comunicar de forma a transmitir os dados coletados ao ecossistema de IoT. Funcionalmente, dispositivos são sensores ou atuadores. Os sensores fornecem informações sobre a entidade física que monitoram. Esta informação pode variar desde a identidade da entidade física até variáveis mensuráveis, tais como temperatura, umidade, pressão, luminosidade, nível de som, fluxo de fluido, vibração e abrasão. Sensores cujo único propósito é facilitar um processo de identificação são chamados de *tags*. Os atuadores utilizam energia, geralmente transportada por ar, eletricidade ou líquido, e a convertem em uma mudança de estado, modificando assim uma ou mais entidades físicas. Os *gateways* (ou *hubs*) coletam, pré-processam e transferem dados de dispositivos IoT e seus sensores, empregando diferentes protocolos de comunicação (geralmente sem fio), como Wi-Fi ou Bluetooth Smart. Os *gateways* fornecem tradução segura de protocolo entre dispositivos e a nuvem, e podem suportar tarefas como armazenamento e pré-processamento de dados, descoberta de serviços, geolocalização, verificação e cobrança. Os gateways também entregam os pedidos de atuação da nuvem para os dispositivos. Em alguns sistemas, os próprios dispositivos podem carregar dados de detecção diretamente na nuvem e receber solicitações de atuação diretamente da nuvem, por exemplo, por meio de redes Wi-Fi, 3G ou 4G e 5G. Essas soluções não exigem *gateways* dedicados. Além disso, os próprios dispositivos geralmente atuam como *gateways* para outros dispositivos, possivelmente formando topologias de rede *peer-to-peer* ou *mesh* por meio da conectividade da rede local. A *cloud* tem armazenamento e análise baseadas em nuvem que são fundamentais para a maioria das plataformas IoT. Na arquitetura genérica de *end-to-end* (ponta a ponta), a nuvem desempenha 3 papéis principais. O primeiro é a aquisição de dados, armazenamento e acesso. O segundo papel é a análise de dados. O terceiro papel é o suporte de atuação. Além disso, uma solução em nuvem inclui funções administrativas, como gerenciamento de dispositivos, gerenciamento de contas de usuários, registro de uso, monitoramento de status do servidor e recursos de relatórios.



Fonte: A. Taivalsaari and T. Mikkonen 2017
https://doi.ieeecomputersociety.org/10.1109/MS.2017.26

Pode-se abstrair dessa arquitetura 3 partes principais:

- Pontos de extremidade que incluem um ou mais sensores;
- *Gateways* que agregam dados de vários pontos de extremidade do sensor e os encaminham para os serviços de *back-end*;
- Serviços de *back-end* em que os dados do sensor são analisados, armazenados, roteados e exibidos.

O mundo do IoT – ou Internet das Coisas – está revolucionando a forma como realizamos a interação com os dispositivos no nosso dia a dia, principalmente quando se percebe a capacidade de conexão com a rede mundial de computadores que permite também a troca de informações de forma constante e em tempo real e os protocolos utilizados para esta comunicação são igualmente importantes para o sucesso no desenvolvimento da tecnologia. O IoT e a Indústria 4.0 "andam de mãos dadas" na evolução da computação e destas novas tecnologias, principalmente, quando se consideram os sistemas embarcados e na evolução dos protocolos. A indústria 4.0 é chamada de Quarta Revolução Industrial e tem desenvolvido o ambiente industrial em prol da automatização, inteligência artificial, modularização e outros pilares ligados a inovação e tecnologia. Nesse contexto, MQTT e REST são protocolos de comunicação que evoluíram, a partir dos protocolos industriais como o Modbus e Fieldbus. O MQTT (*Message Queue Telemetry Transport*) e REST (*Representational State Transfer*) são os dois protocolos que se sobressaem em IoT, ambos com pontos positivos e negativos e diversas aplicações. REST tem princípios e regras bem definidos MQTT tem leveza e facilidade. O MQTT e o REST possuem benefícios e

características positivas que são inerentes ao seu desenvolvimento e ambos são utilizados no mundo IoT.

O protocolo MQTT é descrito como um protocolo de comunicação IoT *Machine-to-Machine* (M2M), ou máquina a máquina. Na sua descrição, destaca-se a leveza no projeto de transporte de mensagens por meio da publicação e assinatura de tópicos, o que é muito útil em sistemas de comunicação que possuem restrições de largura de banda e alta latência na transmissão dos dados. O conceito básico do protocolo MQTT é o seu modelo de publicação e assinatura que consiste em dois tipos básicos de entidades: um *broker* de mensagens e os clientes. O *broker* é um grande servidor que recebe todas as mensagens dos seus clientes e envia essas mensagens aos clientes de destino – e como exemplo, alguns clientes podem ser desde sensores IoT ou até uma aplicação que recebe os dados dos sensores e os processa.

O protocolo REST (ou *Representational State Transfer* que em português é Transferência de Estado Representacional) trata-se de uma abstração da arquitetura da Web com os seus princípios, regras e limitações para permitir a criação do projeto com as interfaces de transmissão de dados de uma forma bem definida, com as operações mais importantes como POST, GET, PUT e DELETE. O REST é um protocolo de comunicação mais recente que surgiu com o objetivo de simplificar o acesso aos Web services. Este baseia-se no protocolo HTTP e permite utilizar vários formatos para representação de dados, como JSON (um dos mais utilizados), XML, RSS, entre outros. Os Web Services permitem uma ampla interação entre os sistemas conectados por meio da internet e são uma ótima forma de conectar aplicações com um desempenho rápido, confiabilidade e habilidade de escalar o serviço.

O REST permite a existência de recursos que são utilizados por meio de um identificador global e que manipulam estes recursos por meio dos componentes da rede, ou os clientes e servidores. A comunicação por meio da interface HTTP possibilita a troca de informação através das operações comuns neste tipo de conexão. A conexão de equipamentos no mundo IoT por meio do protocolo REST é facilitada com a utilização da interface HTTP e permite a utilização da Internet para efetuar essa comunicação. Os dispositivos inteligentes podem se conectar com a internet e se fazem utilizáveis também por outros dispositivos, permitindo uma troca de informações de fato de análise inteligente dos dados por meio de diferentes sistemas conectados.

REST é uma abstração da arquitetura da Web. Resumidamente, o REST consiste em princípios/regras/c*onstraints* que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas.

Já o MQTT, é um protocolo *lightweight*, criado originalmente para suprir as necessidades de conectividade de dispositivos IoT com algumas vantagens sobre outras arquiteturas.

A arquitetura HTTP REST é uma conexão unidirecional e a conexão com o servidor é intermitente. O cliente se conecta ao servidor quando necessário para enviar dados do cliente e envia os dados para o cliente. O servidor precisa esperar que os clientes se conectem para enviar os dados destinados ao cliente. Isso faz com que o usuário pretenda agir para aguardar a conexão do cliente. A maioria dos provedores de soluções permite que seu servidor de borda ou seus gateways se conectem a cada 1 minuto ou mais para que o servidor não seja carregado.

A arquitetura MQTT permite que o cliente seja conectado sempre fornecendo uma comunicação bidirecional entre o cliente e o servidor. Isso permite que o servidor envie a mensagem para o dispositivo, fazendo com que este responda ao seu comando instantaneamente, conforme esperado pelos clientes.

7. Ciência de dados e Python

Data Science é o estudo disciplinado dos dados e das informações inerentes ao negócio e todas as visões que podem cercar um determinado assunto. É uma ciência que estuda as informações, seu processo de captura, transformação, geração e, posteriormente, análise de dados. A ciência de dados envolve diversas disciplinas:

- Computação;
- Estatística;
- Matemática;
- Conhecimento do Negócio.

Pode-se afirmar que o grande volume de dados disponíveis atualmente é o que está mudando o mundo, a indústria e a forma como consumidores e empresas se relacionam. *Data Science, Data Analytics* e *Big Data* são áreas de conhecimento e atuação que trabalham com o mesmo objeto: dados. Mas cada uma tem suas especificidades quando entramos no dia a dia de seus profissionais. Para saber qual destas vertentes é a melhor para um projeto, é preciso conhecer a diferença entre as áreas de dados.

Há uma gigantesca transformação na forma como decisões são tomadas, mas, para isso, é preciso saber coletar, extrair, analisar, classificar, examinar e comparar todos os dados a nossa disposição. E isso não é uma tarefa simples.

As três áreas examinam um conjunto de dados brutos para extrair informações de valor.

Mas esse conhecimento será aplicado de diferentes formas, enquanto para um analista de dados é indispensável a habilidade com estatísticas descritivas e inferenciais, para um analista de *big data* é preciso estar por dentro de "*crunching* numérico", ou seja, saber processar dados numéricos em larga escala.

O cientista de dados lida com análises e métodos mais complexos. Por isso, é imprescindível que ele conheça linguagens de programação. Já para quem trabalha com *Big Data*, tudo isso é amplificado por um enorme volume de dados que precisam ser tratados.

Os conhecimentos aplicados e necessários para um cientista de dados são mais específicos e técnicos, como a importância de conhecer linguagens como Python, R, SAS, Java, Perl, C/C++, além de plataformas como Hadoop e SQL.

Em resumo, o cientista de dados é quem define o algoritmo que entregará a resposta da sua pesquisa no Google em uma fração de segundos e o alcance de anúncios em sites e redes sociais. Ele ainda auxilia na experiência do usuário na hora de encontrar um produto ou serviço específico em um oceano de ofertas. Esses são só alguns dos exemplos das muitas possíveis formas de aplicar Data Science no dia a dia de uma empresa ou de um negócio.

A definição simples de *Data Analytics* pode ser respondida como a ciência de examinar dados brutos para poder extrair conclusões e informações de valores a respeito daquele dado. Cientistas e pesquisadores, por exemplo, utilizam *data analytics* para verificar ou desacreditar modelos e hipóteses. Já empresas a utilizam para validar tomadas de decisões. O analista de dados pode utilizar ferramentas como SAS e R para extrair dados e procurar informações de valor, mas não é algo mandatório no dia a dia do profissional. Seu trabalho envolve responder questões de rotina, geralmente, determinadas pela empresa.

Resumindo, para analistas de dados, a habilidade de conseguir transformar os dados em algo que possa ser facilmente visualizado ou comunicar adequadamente as informações importantes faz toda a diferença no momento de apresentar resultados. Por isso, é fundamental que um analista não seja só focado na parte exata, mas também consiga exercer criatividade.

Fontes:

Evolução dos Computadores.
https://medium.com/@amarcoscrf/evolu%C3%A7%C3%A3o-dos-computadores-3b54c5a116bd

Conheça a história da Internet, sua finalidade e qual o cenário atual.
https://rockcontent.com/br/blog/historia-da-internet/

A evolução da web: http://www.evolutionoftheweb.com/#/evolution/night

The birth of the Web: https://home.cern/science/computing/birth-web

Conheça 10 frameworks que tornam mais rápido o desenvolvimento de softwares
https://blog.cronapp.io/frameworks-para-desenvolvimento-de-softwares/

Entendendo o MTV do Django
https://www.treinaweb.com.br/blog/entendendo-o-mtv-do-django/

Google Colab- Guia do Iniciante
https://medium.com/machina-sapiens/google-colab-guia-do-iniciante-334d70aad531

O que é o Google Colab?
https://kenzie.com.br/blog/google-colab/#:~:text=Google%20Colab%2C%20ou%20Google%20Collaboratory,de%20software%20em%20uma%20m%C3%A1quina.

O que é REST API? Entenda mais sobre integração de sistemas: https://blog.vindi.com.br/o-que-e-rest/

REST Tutorial: https://www.devmedia.com.br/rest-tutorial/28912

API vs Web Service: What's the Difference?
https://rapidapi.com/blog/api-vs-web-service/#:~:text=APIs%20are%20application%20interfaces%2C%20meaning,web%20APIs%20using%20HTTP%20methods.

EXEMPLO DE USO DO MQTT COM JAVA.
https://www.paulocollares.com.br/programacao/exemplo-de-mqtt-com-java/

O que é API? REST e RESTful? Conheça as definições e diferenças!
https://becode.com.br/o-que-e-api-rest-e-restful/

O que é MQTT? Minha automação residencial ou projeto de IoT precisa dele?
https://www.youtube.com/watch?v=qRV8Jusu-go

Deep Learning Book.
https://www.deeplearningbook.com.br/

Internet das Coisas em prol das pessoas.
https://www.fortknox.com.br/blog/seguranca-empresarial/internet-das-coisas-em-prol-das-pessoas/#

O que é Ciência de Dados? https://www.oracle.com/br/data-science/what-is-data-science/