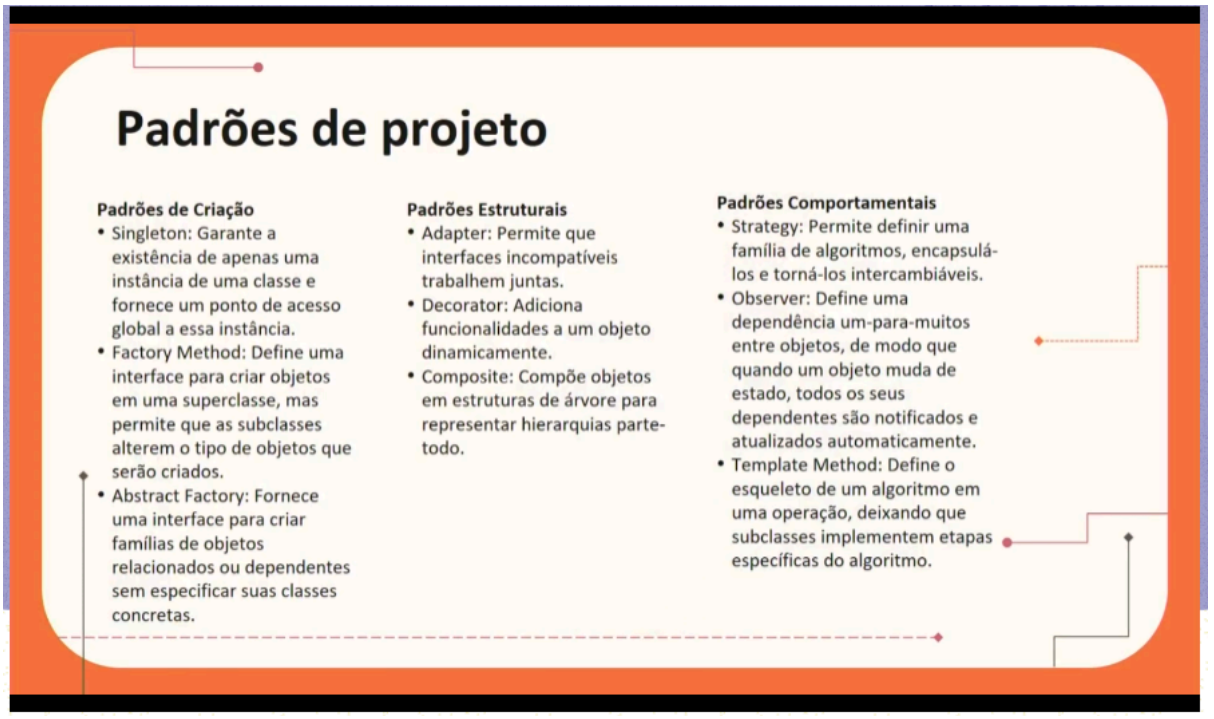

- Analista de dados: responsável por analisar e preparar os dados, transformando-os em informações úteis para a empresa. Deve ter habilidades em programação, estatística e visualização de dados, além de ser capaz de criar narrativas com base nos dados.
- Engenheiro de dados: responsável por lidar com grandes volumes de dados, extrair e armazenar diferentes tipos de dados em bancos de dados, e disponibilizá-los para a equipe de ciência de dados e para a empresa. Suas

habilidades incluem programação, manipulação de dados e engenharia de software.
- Cientista de dados: trabalha próximo ao negócio, compreendendo suas necessidades e desenvolvendo modelos preditivos por meio de aprendizado de máquina e estatística. Deve possuir habilidades em programação, estatística, *machine learning* e visualização de dados, além de colaborar com os engenheiros de dados e *machine learning*.
- Engenheiro de *machine learning*: responsável por implementar os modelos preditivos em ambiente produtivo, automatizando sua atualização e integrando suas respostas à aplicação de negócio. Suas habilidades incluem programação, *machine learning* e engenharia de software.

Paradigma de programação é um meio de se classificar as linguagens de programação baseado em suas funcionalidades. As linguagens podem ser classificadas em vários paradigmas
Um paradigma de programação fornece e determina a visão que o programador possui sobre a estruturação e execução do programa
Os mais tradicionais são:
- Programação estruturada
- Programação Orientada a Objetos

A **programação estruturada** (PE) é um paradigma de programação de software que enfatiza sequência, decisão e iteração, usando sub-rotinas, laços de repetição, condicionais e estruturas em bloco
Surgiu junto com as linguagens ALGOL 58 e ALGOL 60 no final da década de 1950
Baseado nas vantagens práticas que oferece, assim como no 'teorema do programa estruturado' (ou teorema do Bohm-Jacopini) e na carta aberta de Dijkstra GoTo Statement Considered Harmful'
Foi o padrão dominante até a ascensão da Programação Orientada a Objetos

A **programação Orientada a Objetos** (POO), é um paradigma de programação que aumenta a produtividade através da reutilização de código
Ela oferece uma forma mais fácil de adaptar-se a mudanças, pois permite isolar partes do código em classes
A POO permite que a estrutura do código (classes) se torne um objeto, que contém dados e funções. Uma função ou método é um bloco de código que executa uma determinada tarefa ou ação específica
Isso possibilita aos programadores criar relacionamentos entre objetos
Neste ponto, vale diferenciar POO e PE. Enquanto a PE se concentra em estruturas de controle de alto nível e conceitos topdown, a POO se baseia no conceito de objetos com atributos e métodos
A idéia central é tratar os dados como se fossem objetos reais e as funções a maneira como interagimos com os objetos

A programação Orientada a Objetos (POO) se baseia em quatro pilares fundamentais que ajudam a organizar e estruturar o código de forma mais eficiente e reutilizável:

- Abstração
    - É o cerne da Programação Orientada a Objetos
    - Refere-se a capacidade de representamos objetos da vida real dentro de um algoritmo, imaginando suas identidades únicas, suas propriedades (características essenciais) e as ações que executarão, chamadas de métodos
- Encapsulamento
    - O encapsulamento crucial para a segurança do código, pois permite ocultar propriedades e métodos de cada classe, impedindo modificações indiscriminadas
    - Apesar de conhecermos as funcionalidades de um módulo, ocultamos detalhes de sua implementação para terceiros, como no exemplo da classe Conexao em que o usuário sabe a função, mas não os detalhes de como é feita
- Herança
    - Herança é um recurso que estabelece a relação entre classes distintas, garantindo o reúso de código uma das principais vantagens de POO
    - Permite criar novas classes que herdam atributos e métodos da classe pai, mas podem ter alterações para se adequarem a novas características
- Polimorfismo
    - Polimorfismo refere-se à capacidade de assumir diferentes formas
    - Na POO, isso significa que a linguagem de programação pode processar objetos de maneiras diferentes dependendo do seu tipo de dado ou classe
    - Uma vantagem é a possibilidade de ter métodos com o mesmo nome, mas implementação de maneiras distintas em classes herdeiras

Como a POO se relaciona com Big Data

- Inicialmente o conceito de big data envolve o volume, a velocidade e a variedade de dados produzidos. Empresas que não se adaptaram e não souberem lidar com big data correm o risco de enfrentar sérias dificuldades
- No entanto, soluções como Apache Spark, um framework de código aberto, têm contribuído para reduzir os custos de armazenamento e processamento
- O apache Spark é um motor de computação unificado e um conjunto de bibliotecas para big data, com componentes-chave
- Unificação: O spark unifica tarefas de análise de dados e desenvolvimento de software, tornando-as mais fáceis e eficientes devido à variedades de APIs disponíveis

- Motor de Computação: O Spark limita seu escopo a um motor, permitindo o uso de diversos sistemas de armazenamento, como Apache Hadoop, Azure e Amazon S3
- Bibliotecas: O spark inclui bibliotecas para SQL e dados estruturados, machine learning (MLlib) e análises gráficas de dados (GraphX)

1. **Sintaxe simples:** *Python* prioriza a legibilidade, permitindo que os programadores compreendam facilmente o código, direcionando seus esforços para o essencial.
2. **Indentação:** a indentação em *Python* é fundamental para a estrutura do código, com níveis de indentação indicando diferentes blocos de código.
3. **Linguagem interpretada:** *Python* é uma linguagem interpretada, o que significa que suas ações podem ser executadas rapidamente em seu ambiente operacional, oferecendo resultados imediatos.
4. **Linguagem de alto nível:** *Python* é uma linguagem de alto nível, mais próxima da linguagem humana, facilitando o processo de programação em comparação com linguagens de baixo nível, mais próximas da linguagem de máquina.
5. **Linguagem de propósito geral:** *Python* possui uma ampla gama de aplicações, podendo ser utilizada para diversos fins, como criação de amostras de dados, programas de matemática e engenharia de dados, manipulação de linguagem de marcação (XML), interação com bancos de dados e desenvolvimento de aplicações com interface de usuário.

Listas

Em *Python*, as listas são conjuntos ordenados e mutáveis de dados, criados a partir da utilização de colchetes, com os elementos separados por vírgula. Elas podem conter dados de diferentes tipos, ou seja, são estruturas heterogêneas.
Os elementos de uma lista são acessados por meio de seu índice, que começa em zero. Podemos acessar um elemento utilizando seu índice dentro de colchetes. Além disso, é possível acessar os elementos da lista na ordem inversa, utilizando índices negativos.
Para acessar um intervalo de índices de uma lista, utilizamos a sintaxe: **nome_da_lista[índice_inicial:índice_final]**. Se o índice inicial for omitido, o intervalo começará do índice zero. Se o índice final for omitido, o intervalo terminará no último elemento da lista.
Existem métodos disponíveis para editar listas, como **append()**, que adiciona um objeto ao final da lista, e **pop()**, que remove o último objeto da lista.

Tuplas

A próxima estrutura de dados que vamos abordar é a tupla. Assim como as listas, as tuplas são conjuntos ordenados de elementos acessíveis pelo seu índice. A principal diferença entre elas é que a tupla é imutável, ou seja, não pode ser alterada, adicionada ou removida após a sua criação.
As tuplas são criadas com o uso de parênteses, separando seus elementos por vírgula. Embora em alguns casos o uso de parênteses seja opcional, torna-se

obrigatório quando a tupla é utilizada como parte de outra estrutura em *Python*, como em uma lista ou em um argumento de uma função. Assim como nas listas, os elementos de uma tupla podem ser acessados por meio de seus índices.

Dicionários

A última estrutura de dados que vamos abordar são os dicionários. Dicionários são coleções de objetos armazenados em pares chave-valor, nos quais cada chave está associada a um valor específico por meio de dois pontos (:). Os pares chave-valor são separados por vírgulas. Para criar um dicionário, utilizamos chaves.
Os dicionários são especialmente úteis para buscar objetos de forma rápida, bastando buscar pela chave associada ao objeto desejado. Além disso, os dicionários são estruturas mutáveis, o que significa que podemos alterar o valor de uma chave existente, bem como adicionar ou remover pares chave-valor.
Existem alguns métodos comuns que podem ser aplicados aos dicionários. Os mais utilizados são (DATA, 2024):

- **keys()**: Retorna todas as chaves do dicionário.
- **values()**: Retorna todos os valores associados às chaves do dicionário.
- **items()**: Retorna todos os pares chave-valor armazenados no dicionário, representados como tuplas.

Pandas

De acordo com McKinney (2018), criador do p*andas*, essa biblioteca foi projetada como uma alternativa para se trabalhar com dados tabulares ou estruturados.
Por convenção, é correto seguirmos alguns padrões definidos pela comunidade desenvolvedora, e um deles é sempre importar a biblioteca *pandas* com o apelido (*alias*) “pd”.

Além das estruturas de dados, o *pandas* oferece uma ampla gama de ferramentas para manipulação e análise de dados, incluindo funcionalidades para limpeza de dados, manipulação de datas, agregação estatística, visualização de dados e integração com outras bibliotecas de análise e visualização, como *Matplotlib* e *Seaborn*.

Series

A *Series* é uma estrutura de dados unidimensional que armazena uma sequência de valores e possui um índice representado por um *array* de rótulos (*labels*) (MCKINNEY, 2018). Ao criar uma *Series*, geralmente não especificamos o *array* de rótulos. O *pandas* cria automaticamente um índice representado por uma sequência numérica, começando em zero.

Por ser uma classe, a *Series* possui métodos e atributos que nos ajudam a explorar e a manipular os dados. A seguir, veremos alguns métodos e atributos relacionados a essa classe do *pandas*.

***class pandas*.Series(*data=None*, *index=None*, *dtype=None*, *name=None*, *copy=None*, *fastpath=_NoDefault.no_default*)**

- *Index*: esse atributo guarda o índice da *Series*. Esse atributo guarda: um objeto do tipo *RangeIndex*. Esse objeto é muito parecido com a função *built-in range*(), pois cria uma sequência de números.

- *dtypes*: esse atributo guarda o tipo dos dados presentes na *Series*.
- *count*(): retorna o número de observações não-NA/nulas na Série.
- *sum*(): retorna a soma dos valores no eixo solicitado.
- *min*(): retorna o mínimo dos valores no eixo solicitado.
- *max*(): retorna o máximo dos valores no eixo solicitado.

DataFrame

Um *DataFrame* é uma tabela de dados composta por linhas e colunas. Todos os dados de uma mesma coluna precisam ser do mesmo tipo (*int, float, string*, etc.), mas em uma tabela com diferentes colunas, podemos ter colunas de tipos distintos de dados em um mesmo *DataFrame* (MCKINNEY, 2018). Ao contrário das *Series*, um *DataFrame* é uma estrutura bidimensional de dados, semelhante a uma planilha.

***class* pandas.DataFrame(*data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=None*)**

Além de possuir o atributo *index*, assim como as *Series*, um *DataFrame* tem um atributo chamado *columns*, no qual fica guardado o nome das colunas. Veja outros métodos:

- *head*(): retorna as primeiras n linhas.
- *tail*(): retorna as últimas n linhas.
- *info*(): imprime um sumário de um *DataFrame*.
- *describe*(): retorna algumas estatísticas básicas para cada coluna do *DataFrame.*

Abaixo estão algumas das principais bibliotecas já estabelecidas que fazem parte do cotidiano de um profissional de dados:

- **NumPy**: uma biblioteca para processamento de matrizes, contendo um amplo conjunto de funções matemáticas e métodos para operações com esses objetos.
- **Scikit-learn**: fornece algoritmos para construção de modelos de machine learning.
- **Matplotlib**: muito utilizada para visualização de gráficos e diagramas.
- Seaborn: oferece recursos para criação de visualizações, incluindo séries temporais.
- **Plotly**: popular pela facilidade de criar gráficos dinâmicos e interativos.
- **XGBoost**: uma biblioteca dedicada à implementação do algoritmo de machine learning gradient boosting, amplamente utilizado.

Matplotlib

A biblioteca *Matplotlib* é uma das ferramentas mais populares e amplamente utilizadas para visualização de dados em *Python*. Desenvolvida com base nos princípios da programação orientada a objetos (POO), ela oferece uma ampla gama

de funcionalidades para criar gráficos estáticos, interativos e personalizados, permitindo que os usuários visualizem e comuniquem efetivamente informações a partir de seus dados.
Com a *Matplotlib*, os desenvolvedores podem criar uma variedade de gráficos, incluindo gráficos de linha, barra, dispersão, histogramas, entre outros, com controle preciso sobre todos os aspectos visuais, como cores, marcadores, escalas e legendas. Isso é possível graças à abordagem orientada a objetos da Matplotlib, que permite aos usuários manipularem cada elemento do gráfico como um objeto separado, tornando a customização e a criação de gráficos complexos mais acessíveis.
Além disso, a *Matplotlib* é altamente integrada com outras bibliotecas do ecossistema *Python*, como *NumPy* e *pandas*, facilitando a visualização de dados provenientes dessas ferramentas.

```
import matplotlib.pyplot as plt

import numpy as np



xpoints = np.array([0, 6])

ypoints = np.array([0, 250])



plt.plot(xpoints, ypoints)

plt.show()
```

NumPy

A biblioteca *NumPy* é uma das ferramentas fundamentais para computação científica em *Python.* Baseada nos princípios da *POO,* ela oferece suporte para arrays multidimensionais e funções matemáticas de alto nível, tornando-a uma escolha popular para manipulação eficiente de dados numéricos.
Além disso, a *NumPy* fornece uma ampla gama de funções matemáticas e estatísticas, como média, desvio padrão, produto escalar, entre outras, que são essenciais para análise de dados e a computação científica. Sua integração estreita com outras bibliotecas populares, como *pandas*, *SciPy* e *Matplotlib*, permite que os usuários construam pipelines de análise de dados completos e eficientes em *Python*.

```
import numpy as np


# Criando um array de números

valores = np.array([1, 2, 3, 4, 5])


# Calculando a média dos valores

media = np.mean(valores)

print("Média:", media)
```