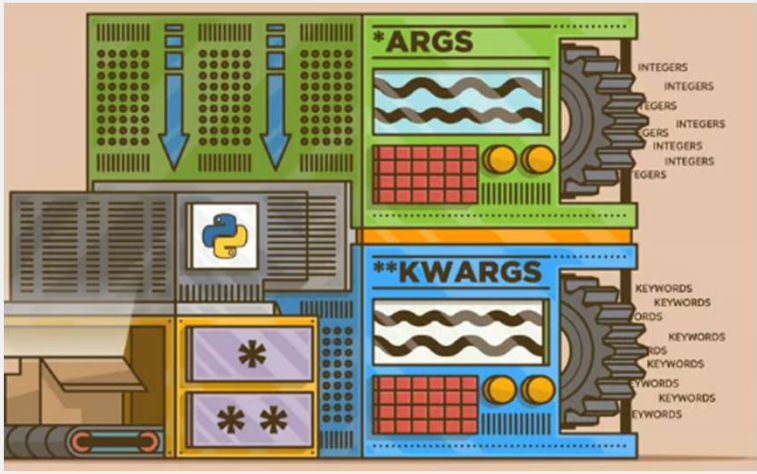

SENAI
Programação em Python para Data Science

Python - Avançando na linguagem

*Args e **Kwargs



*Args e **Kwargs são representações de parâmetros que são indefinidos em uma função. Quando criamos uma função usando esses parâmetros, estamos informando ao Python que nossa função pode receber quantidade indefinida de parâmetros.
Nessa construção, o parâmetro ***args** é uma tupla contendo os valores.

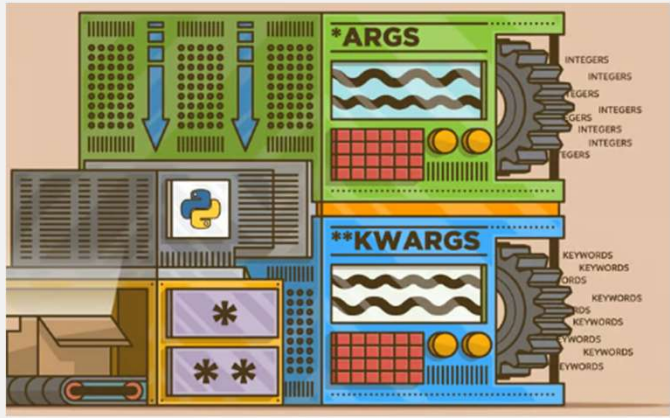
```python
def somar(*args):
    s = 0
    for i in args:
        s += i
    print(s)
```

```
somar(45,100)
somar(10,50,0,45,78,12,1)
```

```
145
196
```

Python - Avançando na linguagem

****Kwargs –** Diferentemente de *args, o **Kwargs tem uma estrutura de dicionário (chave:valor) e dessa forma conseguimos "nomear" os valores informados via função:




```python
def imprimir_cadastro(**kwargs):
    for c,v in kwargs.items():
        print(f"{c:7} | {v}")
    print("\n")
```

```python
imprimir_cadastro(nome="Luiza",idade=25,
                  email="luiza@gmail.com")

imprimir_cadastro(nome="João",altura=1.78,
                  peso=92,idade=32)
```

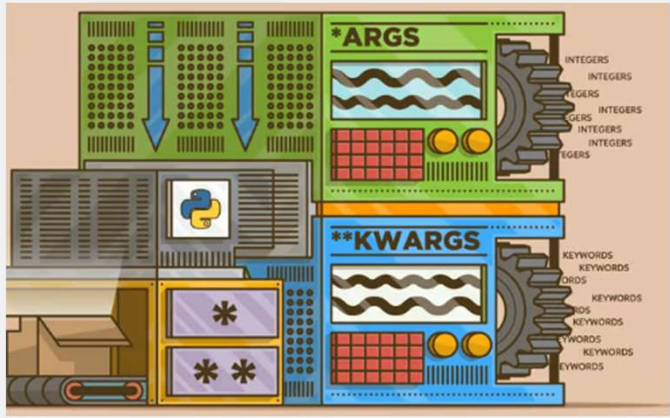
```
nome    | Luiza
idade   | 25
email   | luiza@gmail.com
```

```
nome    | João
altura  | 1.78
peso    | 92
idade   | 32
```

Python - Avançando na linguagem

SENAI

Não precisamos chamar esses parâmetros obrigatoriamente como *args ou **kwargs. Esses nomes são somente uma convenção da comunidade Python.



Quando formos utilizar essa estrutura precisamos apenas usar * (para args) e ** (para kwargs) e o nome do parâmetro segue a regra geral conforme nossa lógica:

```python
def imprimir_nomes(*nomes):
    print(f"Qtd de nomes enviados: {len(nomes)}")
    print("-------------------------------")
    for nome in nomes:
        print(nome)

imprimir_nomes("Ana","Diego","João","Larissa")
```

```
Qtd de nomes enviados: 4
-------------------------
Ana
Diego
João
Larissa
```

python SOFTWARE FOUNDATION

Python - Avançando na linguagem

Atividade:

Crie um programa em Python que receba as notas de um aluno no semestre. A quantidade de provas aplicadas pode variar conforme a matéria. Faça a entrada de dados de forma dinâmica, perguntando previamente ao usuário quantas notas serão digitadas.

Crie uma função que receba um valor indefinido de notas, faça sua média aritmética e mostre o resultado da média na tela.

A função deve exibir também se o aluno foi aprovado ou não. Regra: Média deve ser maior ou igual a 7.0 para que um aluno seja considerado aprovado.

Usar conceito de *args em Python na função.

Python - Orientação a objetos

SENAI

Até agora aprendemos a programação chamada procedural, que é uma programação sequencial que no máximo nos possibilita criar funções e chamar métodos.

Apesar desse paradigma ser muito poderoso e ter possibilitado inclusive a criação de sistemas operacionais completos, ele possui muitas limitações. Uma das formas de suprir essas limitações foi criar o paradigma da **programação orientada a objetos (POO)**.

Esse paradigma tem como base trazer para a programação objetos e entidades do mundo real. São as chamadas **Classes e objetos**.

As classes seriam como um conjunto de características e comportamentos que definem e servem de molde para algo. Os objetos (ou instâncias) são variáveis que trarão o contexto abstrato da classe para o "mundo real" e que possibilitarão armazenarmos dados/valores em nosso programa.

Toda classe é composta por atributos e comportamentos, ou seja, funções. Vejamos um exemplo:

python
SOFTWARE FOUNDATION

Python - Orientação a objetos

Criando uma classe e definindo suas características, ou seja, **seus atributos:**

```python
class Pessoa: # Criando uma classe
    # Construtor da classe com seus atributos:
    def __init__(self,nome,idade,altura):
        self.nome = nome
        self.idade = idade
        self.altura = altura
```

Python - Orientação a objetos

Conceito principal: Dados e funcionalidades (comportamentos) devem andar juntos!
Por definição, na POO os parâmetros são chamados de **atributos** e as funções de **métodos**.

SENAI

```python
class Pessoa: # Criando uma classe:
    # Construtor da classe com seus atributos:
    def __init__(self,nome,idade,altura):
        self.nome = nome
        self.idade = idade
        self.altura = altura
    # métodos:
    def andar(self):
        print(f"{self.nome} está andando...")
    def falar(self,frase):
        print(f"{self.nome} está dizendo: {frase}")
    def dormir(self):
        print(f"{self.nome} está dormindo agora.")
```

Neste exemplo, criamos a classe Pessoa com os seguinte atributos: nome, idade e altura.

Os métodos, ou seja, as ações que essa classe pode realizar são: andar, falar e dormir.

Python - Orientação a objetos

Testando nossa classe Pessoa:

```
# Importando classe Pessoa:
from pessoa import Pessoa
# Instanciando (criando) objeto da classe pessoa:
pessoa = Pessoa("João",32,1.78)
# Usando os métodos da classe:
pessoa.andar()
pessoa.falar("Boa noite!")
pessoa.dormir()
```

```
João está andando...
João disse: Boa noite!
João está dormindo agora.
```

SENAI

python
SOFTWARE FOUNDATION

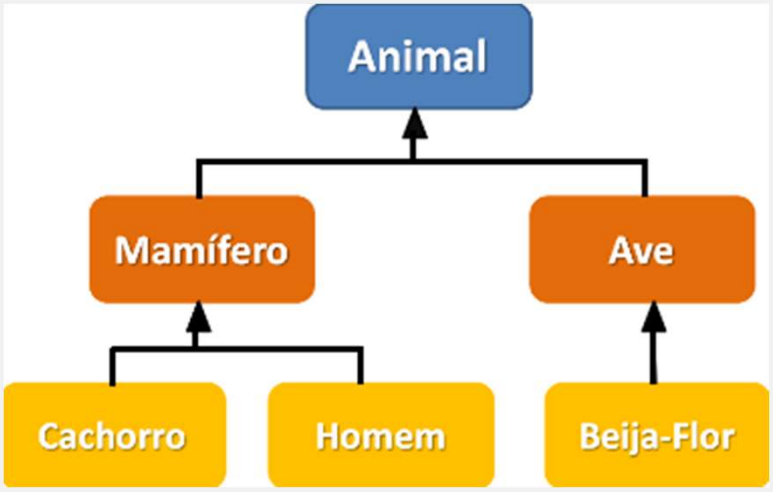
Python - Orientação a objetos

Conceito de herança na orientação a objetos

Entendemos herança como algo que um antepassado deixou de legado para seus sucessores (bens materiais, genética, etc).

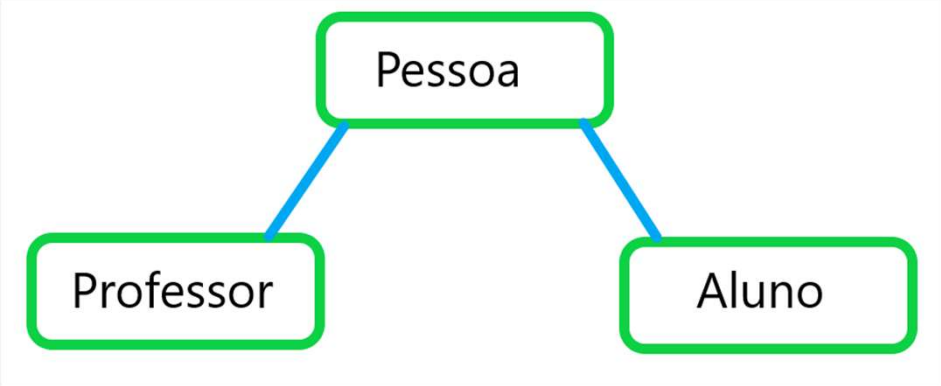
Na POO o conceito é muito parecido mas é usado para conceituar classes que herdam atributos e métodos de uma classe superior.

Isso evita repetição de códigos quando temos classes que são semelhantes e possuem atributos e métodos em comum, além do código ficar muito mais limpo, padronizado e organizado.



Python - Orientação a objetos

Herança



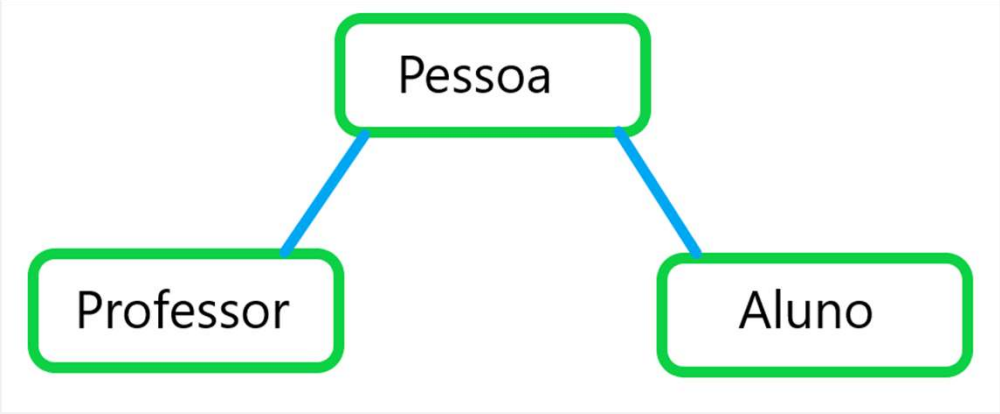

```python
# criando classe Aluno que herda os métodos e atributos
# da classe Pessoa:
class Aluno(Pessoa):
    def __init__(self,nome,idade,altura,celular,email):
        # Inicializando construtor da classe "pai"
        # através da função super()
        super().__init__(nome,idade,altura)
        # Definindo atributos inerentes à classe Aluno:
        self.celular = celular
        self.email = email
```

SENAI

python
SOFTWARE FOUNDATION

Python - Orientação a objetos

Herança




```python
class Professor(Pessoa):
    def __init__(self,nome,idade,altura,area):
        super().__init__(nome,idade,altura)
        self.area = area
    def apresentarProfessor(self):
        print(f"\nProfessor: {self.nome}\n"
              f"Idade: {self.idade}\n"
              f"Área: {self.area}")
```

Python - Orientação a objetos

Herança

As classes Professor e Aluno herdaram os atributos e funcionalidades da Classe Pessoa, portanto um objeto da classe Professor teria os seguintes atributos e métodos:

Atributos: nome, idade, altura, area

Métodos: andar(), falar(), dormir() e mostrarProfessor():

```
professor = Professor("Diego Silva",58,1.72,"Programação")
professor.andar()
professor.falar("Bom dia pessoal!")
professor.apresentarProfessor()
```

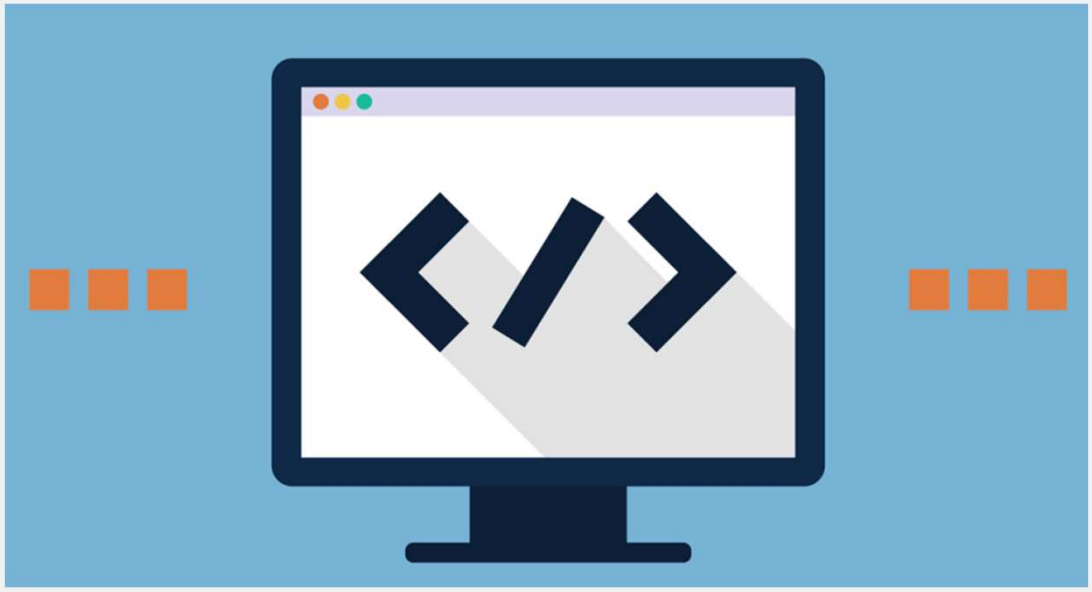
```
Diego Silva está andando...
Diego Silva disse: Bom dia pessoal!

Professor: Diego Silva
Idade: 58
Área: Programação
```

Python - Orientação a objetos

Vamos praticar!

Use sua criatividade e crie uma classe pai com dois atributos e dois métodos. Crie duas classes filhas que herdem os atributos e funções da classe pai e implemente pelo menos mais um método e atributo para cada classe filha.

Instancie objetos das classes filhas e faça alguns testes.



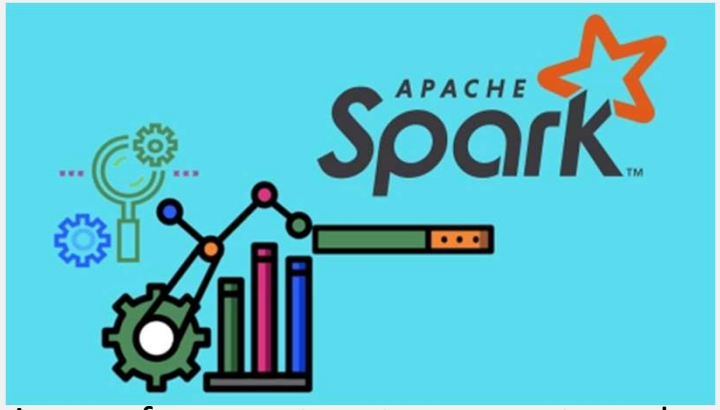
Apache Spark - Definição

O Apache Spark é um framework gratuito e de código aberto que possibilita o processamento de grandes conjuntos de dados em escala.

Isso significa que o processamento dos dados pode ser distribuído em clusters, tecnologia que possibilita a junção de vários computadores (servidores) para executarem uma mesma tarefa.

Além de ter uma tecnologia de processamento distribuído (entre vários nós), o Spark possui várias APIs com as principais linguagens de análise e manipulação de dados:

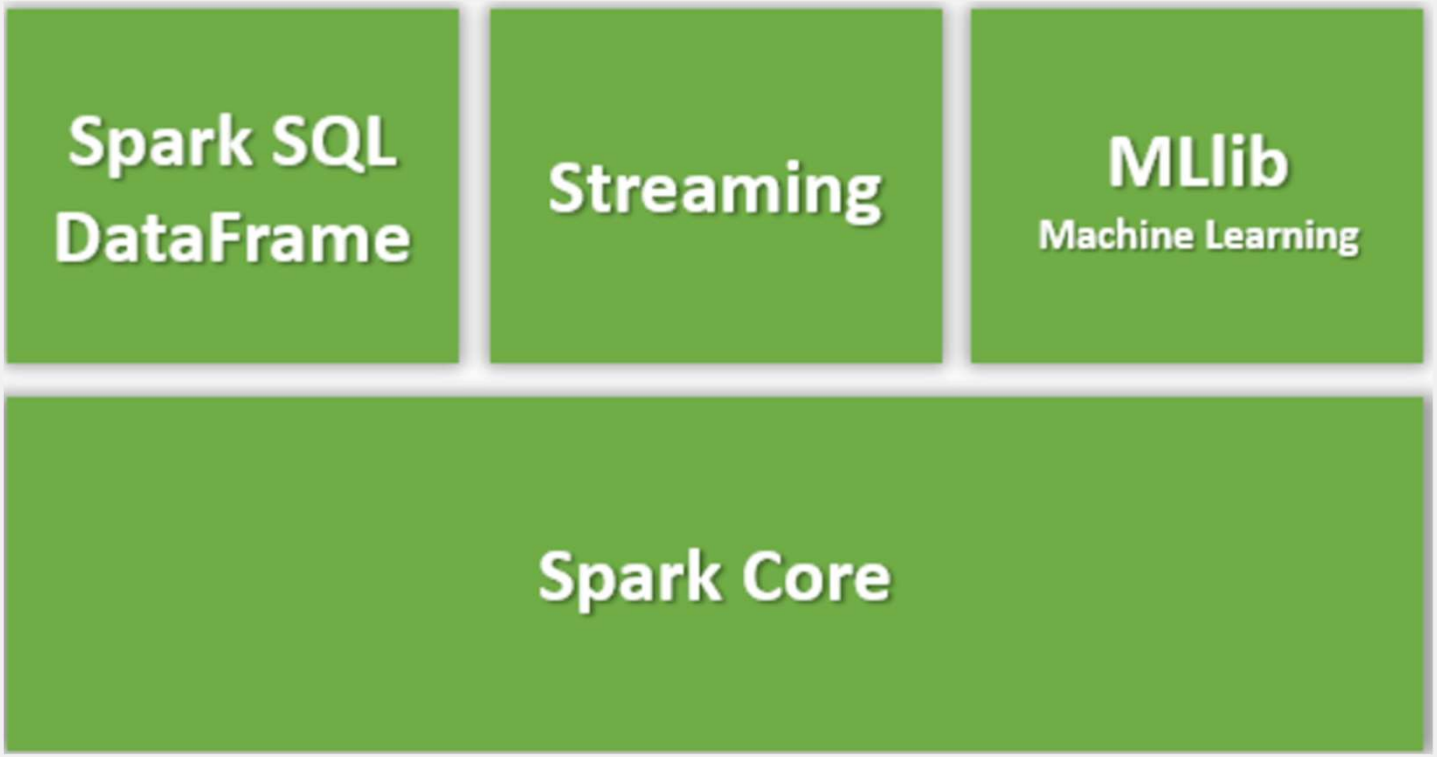
- Java
- Python
- Scala
- SQL
- R



Isso permite que o Spark seja uma ferramenta extremamente poderosa nas áreas de engenharia de dados, análise de dados, *machine learning*, ciência de dados e streaming de dados.

Apache Spark - Definição

Principais recursos do Spark:



Apache Spark - Definição

Spark SQL e DataFrame
Spark SQL é um módulo Spark para processamento de dados estruturados. Ele fornece uma abstração de programação chamada DataFrame e também pode atuar como mecanismo de consulta SQL distribuído.

Spark Streaming
Executando em cima do Spark, o recurso de streaming no Apache Spark possibilita o uso de poderosas aplicações interativas e analíticas em streaming e dados históricos, enquanto herda a facilidade de uso do Spark e as características de tolerância a falhas.

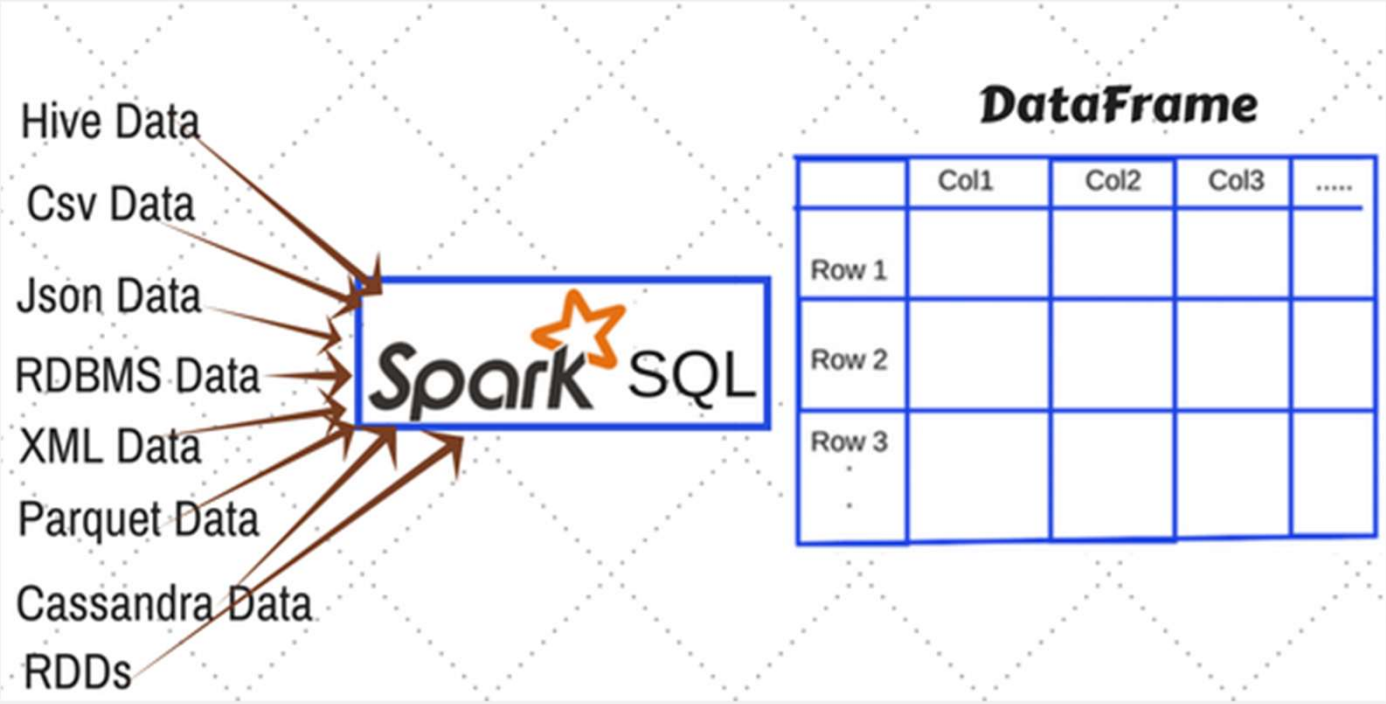
Spark MLlib
Construído sobre o Spark, MLlib é uma biblioteca de aprendizado de máquina escalonável que fornece um conjunto uniforme de APIs de alto nível que ajudam os usuários a criar e ajustar pipelines de aprendizado de máquina práticos.

Spark Core
Spark Core é o mecanismo de execução geral subjacente para a plataforma Spark sobre o qual todas as outras funcionalidades são construídas. Ele fornece um RDD (Resilient Distributed Dataset) e recursos de computação na memória.

Apache Spark - Definição

Uma das principais características do Spark é possibilitar o deploy e manipulação dos dados diretamente na memória RAM. Dessa forma a manipulação das informações torna-se muito mais rápida se comparada com manipulações feitas em disco.

Uma das estruturas mais usadas no Spark é o DataFrame que nada mais é que alocar diretamente na memória, em formato tabelado ou colunar, dados que estavam em disco.



PySpark --> Python + Apache Spark

O PySpark é uma API Python para se trabalhar no ambiente Spark.

Essa API permite a manipulação e tratamento de grandes volume de dados com a lógica, sintaxe e comandos em Python.

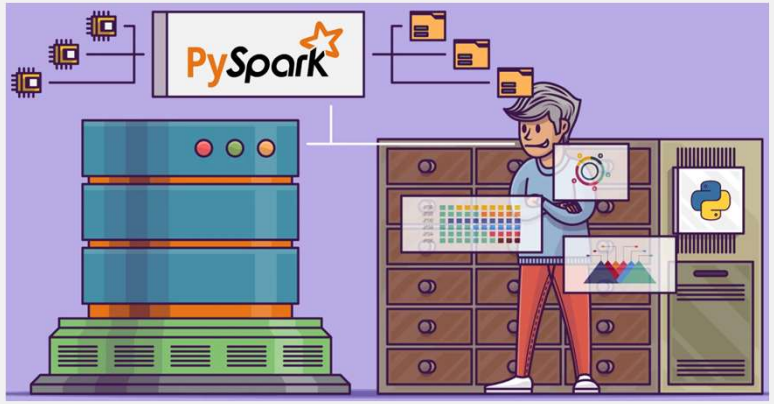
[1]O PySpark combina a capacidade de aprendizado e a facilidade de uso do Python com o poder do Apache Spark para permitir o processamento e a análise de dados em qualquer tamanho para todos os familiarizados com o Python.

O PySpark oferece suporte a todos os recursos do Spark, como Spark SQL, DataFrames, streaming estruturado, aprendizado de máquina (MLlib) e Spark Core.

[1]https://spark.apache.org/docs/latest/api/python/

PySpark --> Python + Apache Spark

SENAI



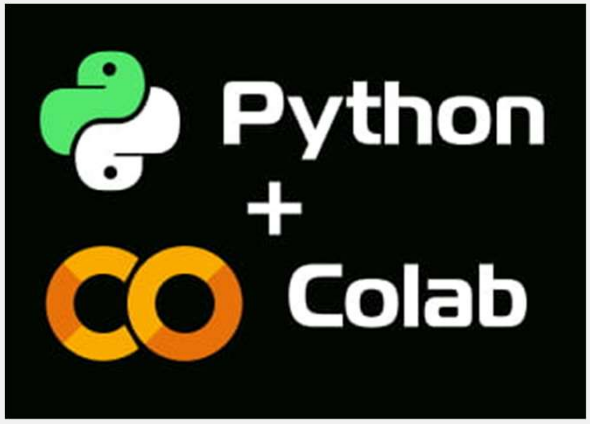
O PySpark é uma biblioteca do Python. Para utilizá-lo precisamos instalá-lo através do gerenciador de pacotes "pip".

Existem ambientes de desenvolvimento mais adequados para a análise de dados, tais como o Jupyter e o Google Colab.

Nesse curso iremos trabalhar no ambiente em nuvem do Google Colab.

Projeto Jupyter: https://jupyter.org/

Google Colab: https://colab.research.google.com/

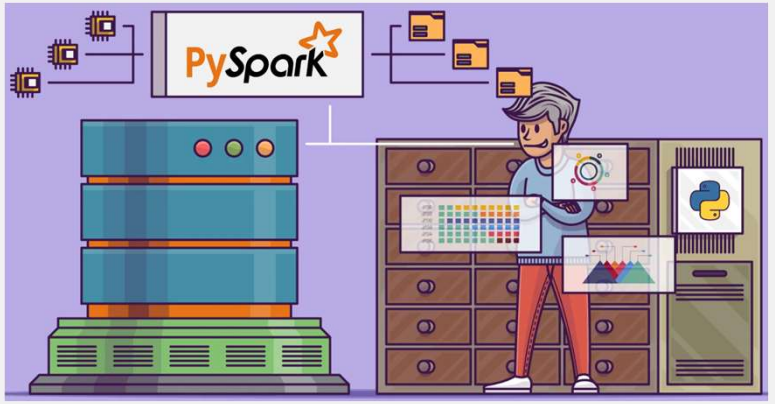


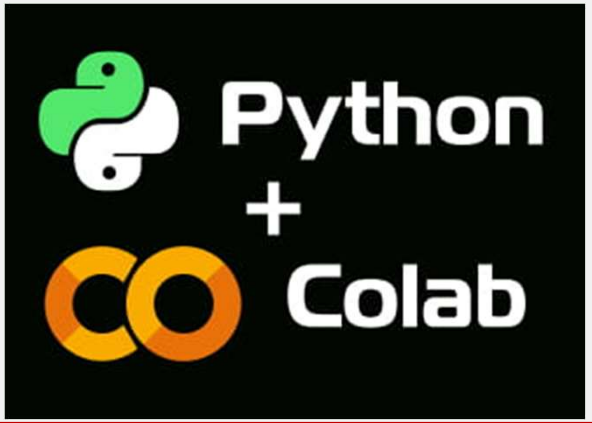
PySpark --> Python + Apache Spark

SENAI

Com as ferramentas PySpark e Google Colab aprenderemos a:

- Criar uma sessão Spark no Colab
- Criar DataFrames manualmente ou via importação de arquivos
- Tratamento e adequação de informações
- Conversão de dados
- Consultas e pesquisas dos dados em memória
- Salvamento dos dados da memória RAM para arquivos em disco.



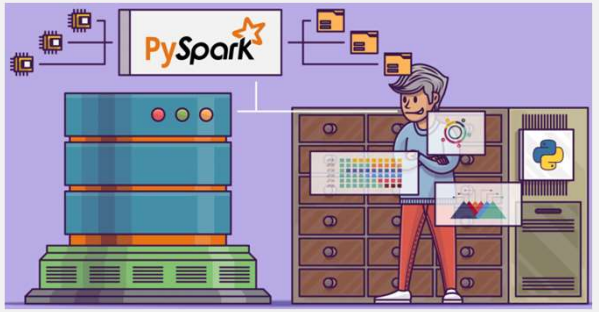


PySpark - Criando ambiente no Colab

```
# Comandos para inicializarmos um ambiente Spark no Google Colab
# Instalando o PySpark
!pip install pyspark
# Instalando o FindSpark para auxílio na criação do ambiente
!pip install findspark
```

```
# "Encontrando" Spark:
import findspark
findspark.init()
# Importando a classe SparkSession que criará
# efetivamente o ambiente Spark:
from pyspark.sql import SparkSession
# criando variável que receberá o ambiente da
# classe SparkSession:
spark = SparkSession.builder.appName("SENAI-SPARK").getOrCreate()
```

SENAI



Python + Colab

PySpark - Criando ambiente no Colab

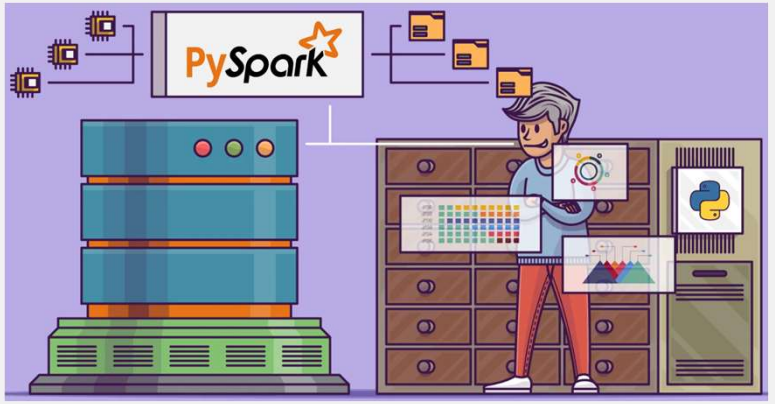
```
# Testando ambiente Spark
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
    v3.4.0
Master
    local[*]
AppName
    SENAI-SPARK



Python + Colab

Python - PySpark

Vamos praticar!

Faça as seguintes alterações no Dataset "empresas" usando o ambiente Spark no Google Colab:

1 - Mude o nome da coluna "razao_social" para "nome_empresa" e da coluna "capital_social" para "valor_investido".

2 - Verifique quantas empresas estão em recuperação judicial (essa informação está no nome da empresa, de forma padronizada).

3 - Crie um novo DataFrame somente com as empresas que possuem valor_investido > 500.000 e <= 1.000.000. Quantas empresas estão nessa regra?

4 - Crie um novo DataFrame com algum tipo de filtro que você considera interessante.

SQL + Spark

A biblioteca PySpark também nos possibilita executarmos comandos SQL avançados em nossos DataFrames.

Os DataFrames se comportam como se fossem tabelas em um banco de dados e conseguimos realizar consultas com extrema facilidade.

O domínio do SQL ajuda muito o analista de dados em seu processo de filtragem, análise e criação de outros DataFrames.

Para executarmos os comandos SQL, precisamos criar uma "view" do DataFrame em memória. Não podemos usar diretamente o DataFrame com o SQL. Função utilizada: objeto_spark.createOrReplaceTempView():

```
# Criando view de um DataFrame no PySpark:
empresas.createOrReplaceTempView("v_empresas")
```

SQL + Spark

```
spark.sql("""
SELECT *
FROM V_EMPRESAS AS V
WHERE V.RAZAO_SOCIAL LIKE '%ESCOLA DE%'
ORDER BY V.RAZAO_SOCIAL
""").show(10,False)
```

```
+--------+-----------------------------------------------------------------------
|cnpj    |razao_social
+--------+-----------------------------------------------------------------------
|36026995|"ASSOCIACAO ESCOLA COMUNIDADE ESCOLA DE 1.GRAU ""CAROLINA P.GAIGHER"
|66657271|"CLUBE ESCOLA DE PARA-QUEDISMO ""NINHO DAS AGUIAS"""
|74102971|"ESCOLA DE DATILOGRAFIA ""RUI BARBOSA DE SOUZA"" S/C LTDA"
|69285146|"ESCOLA DE ED INFANTIL ""A-VENTURA DO CRESCER"" S/C LTDA"
```

SENAI

Python + Colab

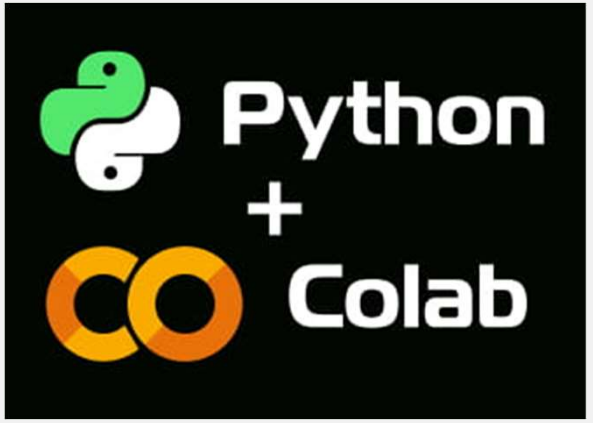
APACHE PySpark

SQL + Spark

```
spark.sql("""
SELECT V.CNPJ, V.RAZAO_SOCIAL AS HOSPITAL,
       V.CAPITAL_SOCIAL AS VALOR_INVESTIDO
FROM V_EMPRESAS AS V
WHERE
  V.CAPITAL_SOCIAL >= 1000000 AND
  V.CAPITAL_SOCIAL <= 150000000 AND
  V.RAZAO_SOCIAL LIKE 'HOSPITAL%'

""").limit(100).toPandas()
```

CNPJ	HOSPITAL	VALOR_INVESTIDO
35359145	HOSPITAL SAO FRANCISCO DE BARREIRAS LTDA	26667000.0
17239624	HOSPITAL ORION LTDA	10000000.0
23443518	HOSPITAL OTOCLINICA LTDA	1000000.0
10623179	HOSPITAL HCC DE ARIQUEMES LTDA	7186782.0

SENAI



SQL + Spark

Criando um DataFrame personalizado somente com empresas que são hospitais e com capital social entre 1.000.000 e 150.000.000:

```
hospitais = spark.sql("""
SELECT V.CNPJ, V.RAZAO_SOCIAL AS HOSPITAL,
       V.CAPITAL_SOCIAL AS VALOR_INVESTIDO
FROM V_EMPRESAS AS V
WHERE
  V.CAPITAL_SOCIAL >= 1000000 AND
  V.CAPITAL_SOCIAL <= 150000000 AND
  V.RAZAO_SOCIAL LIKE 'HOSPITAL%'

""")
```

```
hospitais
```

```
DataFrame[CNPJ: string, HOSPITAL: string, VALOR_INVESTIDO: float]
```

SENAI

Python + Colab

APACHE PySpark

SQL + Spark

Criando um DataFrame personalizado somente com empresas que são hospitais e com capital social entre 1.000.000 e 150.000.000:

```
hospitais.limit(5).toPandas()
```

	CNPJ	HOSPITAL	VALOR_INVESTIDO
0	35359145	HOSPITAL SAO FRANCISCO DE BARREIRAS LTDA	26667000.0
1	17239624	HOSPITAL ORION LTDA	10000000.0
2	23443518	HOSPITAL OTOCLINICA LTDA	1000000.0
3	10623179	HOSPITAL HCC DE ARIQUEMES LTDA	7186782.0
4	4290944	HOSPITAL DO CORACAO DO PARA LTDA	12000000.0

SENAI

Python + Colab

APACHE PySpark

SQL + Spark

Usando funções de agregação:

```
spark.sql("""
SELECT 'TOTAL_INVESTIDO',
       SUM(V.VALOR_INVESTIDO) AS TOTAL
FROM V_HOSPITAIS AS V
""").toPandas()
```

	TOTAL_INVESTIDO	TOTAL
0	TOTAL_INVESTIDO	382772996.0

SENAI

Python + Colab

APACHE PySpark

SQL + Spark

Usando funções de agregação:

```
spark.sql("""
SELECT V.natureza_juridica, SUM(V.CAPITAL_SOCIAL) TOTAL
FROM V_EMPRESAS AS V
WHERE V.CAPITAL_SOCIAL <> 0
GROUP BY V.natureza_juridica
""").toPandas()
```

	natureza_juridica	TOTAL
0	2305	8.132683e+10
1	3999	2.963225e+06
2	2046	7.877859e+10
3	2330	1.722200e+05

SENAI

Python + Colab

PySpark

SQL + Spark

Fazendo junções entre tabelas e criando um único DataFrame com informações de 2 views:

```
# Criando um join entre as views de empresas e
# estabelecimentos:
join_empresas = spark.sql("""
SELECT *
FROM V_EMPRESAS
INNER JOIN V_ESTABS ON V_ESTABS.CNPJ_BASICO = V_EMPRESAS.CNPJ
""")
```

```
join_empresas.count()
```

```
4836221
```

```
join_empresas
```

```
DataFrame[cnpj: string, razao_social: string, natureza_juridica: string, qualif
float, porte: string, ente_federativo: string, cnpj_basico: string, cnpj_ordem:
```

SENAI

Python + Colab

APACHE PySpark

SQL + Spark

Consulta personalizada:

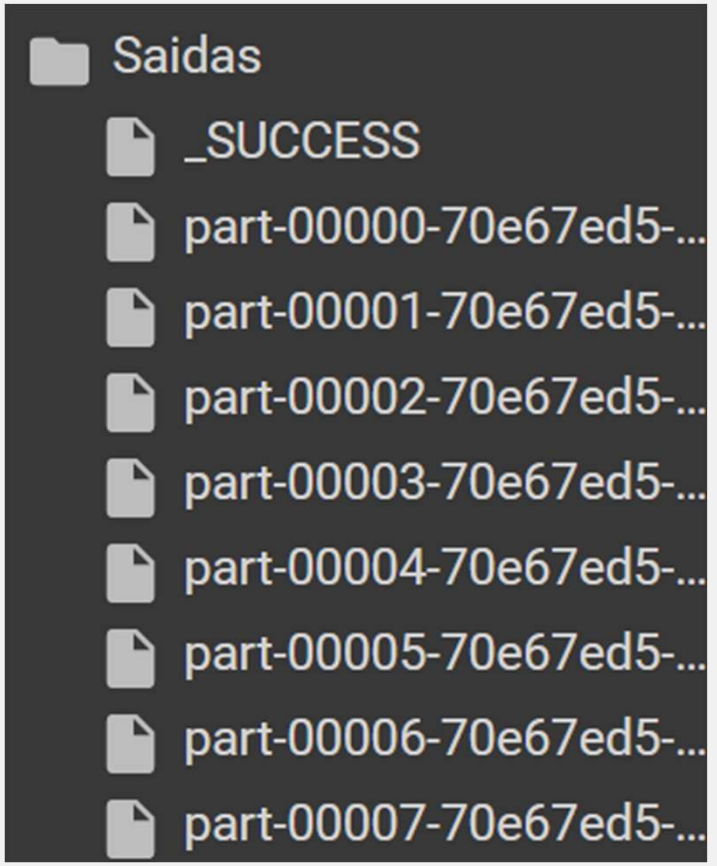
```
spark.sql("""
SELECT V.CNPJ, V.CNPJ_ORDEM LOJA, V.NOME_FANTASIA,
       V.MUNICIPIO, V.TIPO_DE_LOGRADOURO AS TIPO,
       V.LOGRADOURO, V.BAIRRO
FROM V_JOIN_EMPRESAS AS V
WHERE V.NOME_FANTASIA IS NOT NULL AND
      V.CAPITAL_SOCIAL >= 50000 AND
      V.CAPITAL_SOCIAL <= 200000
ORDER BY V.CAPITAL_SOCIAL DESC
""").show(15,False)
```

```
+--------+----+-----------------------------------+---------+-------+----------------------------+--------------------+
|CNPJ    |LOJA|NOME_FANTASIA                      |MUNICIPIO|TIPO   |LOGRADOURO                  |BAIRRO              |
+--------+----+-----------------------------------+---------+-------+----------------------------+--------------------+
|10215358|1   |JAGUAR CALCADOS                    |0139     |AVENIDA|GETULIO VARGAS              |CENTRO              |
|1020753 |1   |HUMAITA TUR                        |0770     |AVENIDA|JOSE BENTO RIBEIRO DANTAS   |RASA                |
|10220426|1   |TEREZA IMOVEIS                     |7535     |RUA    |PROFESSOR FABIO DE SOUZA    |PORTAO              |
|10276553|1   |CONVIVERE EMPREENDIMENTOS IMOBILIARIOS|6579     |AVENIDA|DR. OTAVIANO PEREIRA MENDES |CENTRO              |
|10235842|1   |JCLL WITCH SERVICOS                |5839     |RUA    |RUA DA AMELIA REIS          |COROA GRANDE        |
|1020844 |1   |SCHWANCK & LEFFA                   |8901     |RODOVIA|RS-118                      |BOA VISTA           |
|10289565|1   |MAR FORT                           |6001     |RUA    |AGUAPE                      |PARADA DE LUCAS     |
|10202811|1   |MAIS VOCE SUPERMERCADO             |9143     |RUA    |MOHAMED ALLE                |CENTRO              |
```

Salvando DataFrames em arquivos

Após realizar as análises e manipulação dos dados, podemos salvar o DataFrame em um arquivo. Por padrão, o pyspark irá dividir os arquivos conforme o volume de dados:

```
join_empresas.write.csv(
    "/content/drive/MyDrive/Colab Notebooks/Saidas/",
    sep=";",
    mode = "overwrite",
    header=True
)
```



Salvando DataFrames em arquivos

Se quisermos salvar nossos dados em um único arquivo, podemos usar a função coalesce(1). Salvando em formato .CSV:

```
join_empresas.coalesce(1).write.csv(
    "/content/drive/MyDrive/Colab Notebooks/Saidas/",
    sep=";",
    mode = "overwrite",
    header=True
)
```

Salvando em formato ORC:

```
join_empresas.coalesce(1).write.orc(
    "/content/drive/MyDrive/Colab Notebooks/Saidas/",
    mode = "overwrite",
)
```

Formato de arquivo ORC

Os arquivos CSV embora poderosos e muito utilizados no armazenamento de dados textuais e tabelados, tem suas limitações.

Quando estamos tratando com registros de Gigabytes, o CSV começa a apresentar lentidão.

Pensando na melhoria no desempenho, pesquisadores desenvolveram alguns formatos de arquivos que não trabalham no padrão "linha x coluna" mas no padrão inteiramente colunar.

Alguns desses formatos são o PARQUET e o ORC.

Podemos testar na prática a utilização desses formatos em nosso ambiente Spark, comparando o desempenho de um DataFrame importado de um CSV x um DataFrame importado de um arquivo ORC.

Para saber mais: https://orc.apache.org/talks/

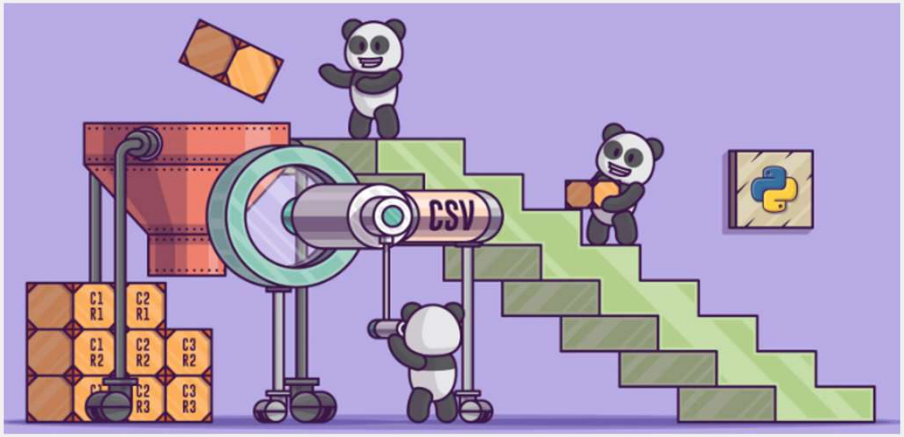
Biblioteca Pandas

É uma das bibliotecas mais utilizadas para análise de dados e *machine learning*.

Possui funções e ferramentas de tratamento, agregação, alteração e plotagem gráfica de elementos.

Com essa biblioteca podemos facilmente importar arquivos e manipulá-los, realizar consultas, agregações e gerar gráficos.

Por se tratar de uma biblioteca já instalada na biblioteca padrão do Python, podemos utilizá-la fazendo diretamente sua importação:




```python
# Importando biblioteca pandas:
import pandas as pd
# Lendo um arquivo .csv:
df = pd.read_csv(
    "caminho.csv",
    sep=";"
)
```