Programação Funcional - Analise de Dados

LAMFO

AGENDA

- Paradigmas De Programação;
- Paradigma Estruturado;
- Paradigma Orientado A Objetos (OO);
- Paradigma Funcional;
- Linguagens Funcionais;
- Linguagens Funcionais: LISP e Haskell;
- Linguagens Funcionais: Lua;
- Linguagens Funcionais: Elixir;
- Linguagens Funcionais: Scala;
- SPARK.

PARADIGMAS DE PROGRAMAÇÃO

- **O que é?**

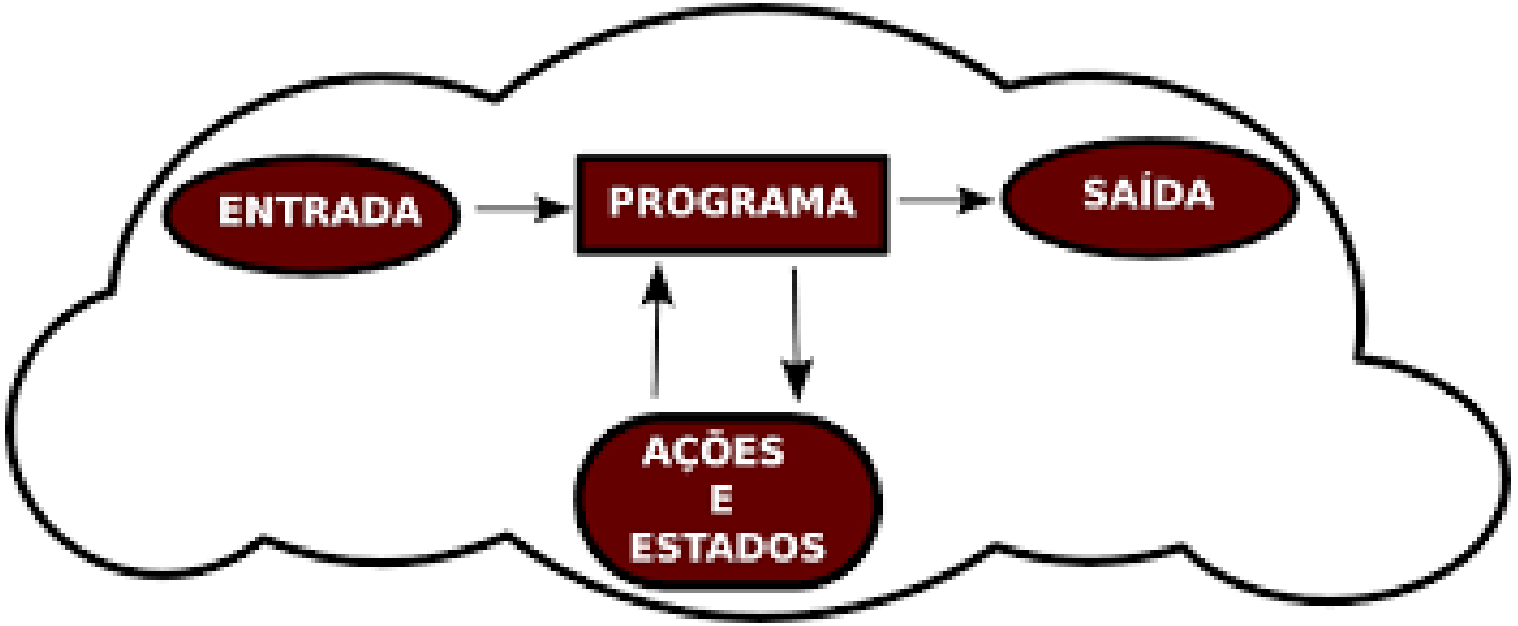
Um estilo ou forma de programar.

Meios de se classificar as linguagens de programação através de suas funcionalidades. Fornece ao desenvolvedor o raciocínio a ser seguido para a estruturação e execução do programa. Cada paradigma tenta organizar como o programa será executado.

PARADIGMA ESTRUTURADO

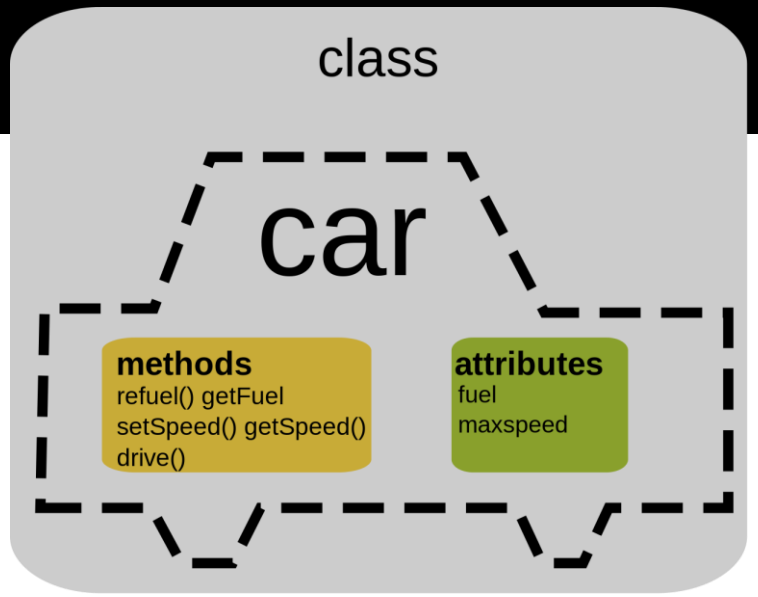
Tipo de programação imperativa onde o controle do fluxo é definido por loops aninhados, condicionais e subrotinas mais do que utilizando goto. As variáveis são normalmente locais aos blocos.:

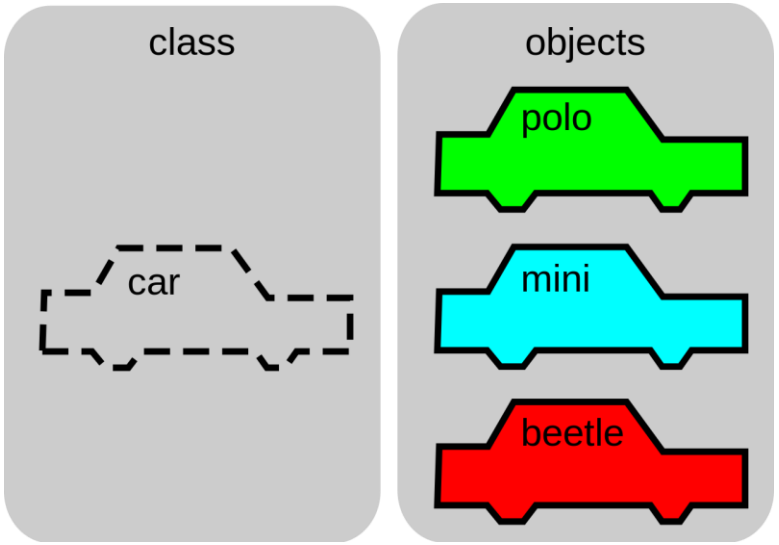
1 - Estrutura de sequência: na qual um comando é executado após o outro, de forma linear, e, portanto sem o uso de goto;
2 - Estrutura de decisão: na qual trechos do programa podem ser executados dependendo do resultado de um teste lógico;
3 - Estrutura de iteração: na qual determinado trecho de código é repetido por um número finito de vezes, enquanto um teste lógico for verdadeiro.



PARADIGMA ESTRUTURADO - EXEMPLO

```
#Exemplo Python:
    ##(Loop que faz a requisição dos dados na API)##

lista_coins = ['bitcoin','ethereum','xrp','btc_cash','eos']
lista_var_json = []
# Looping through the dict
for coins in lista_coins:
    # Making the request for each coin
    print('### Retrieving data for',coins)

    # Looping through markets
    for market in globals()[coins].keys():
        # return the URL for the market
        print('\n####Requesting data for the exchange',market)

        # Requesting the data
        req = requests.get(globals()[coins][market], headers={'User-Agent':'LAMFO (leolacerdagaller@live.com)'} )

        # Creating the name for the variable that will stores the data
        var_name = coins + '_' + market + '_data'

        # Salvando os nomes de variáveis em lista para utiliza-los no passo de normalização do JSON
        lista_var_json.append(var_name)

        print('\n        File name is ',var_name)

        # Passing the data to the variable
        vars()[var_name] = req.json()

        print('\n    End of request for the exchange',market)

time.sleep(1)    # Seconds
print('\nEnd retrieving data for', coins)
```





ORIENTAÇÃO A OBJETOS (OO)

Baseado na troca de mensagens entre objetos. Ou seja, vários objetos diferentes trocam informações durante a execução do programa. Composto de 4 princípios:

- **1 - Encapsulamento:** É alcançado quando cada objeto mantem seu estado oculto dos outros objetos. Não existe acesso direto, o acesso é feito através da chamada de métodos.;
- **2 - Abstração:** Aplicar abstração significa que cada objeto deveria expor apenas uma estrutura básica ou comum de funcionamento. Pequeno conjunto de métodos públicos que outros objetos podem chamar sem saber como ele funciona;
- **3 - Herança:** Objetos costumam ter muitos itens similares mas não são totalmente iguais, é possível herdar todos os campos e métodos da classe pai e implementar seus próprios métodos;
- **4 - Polimorfismo:** Definir uma estrutura geral que pode ser utilizada para criar todo tipo de objeto novo garantindo métodos iguais a todos os filhos;

PARADIGMA FUNCIONAL

As funções em Python são funções de primeira classe, ou seja tratam funções como dados. Isso permite que as funções sejam passadas para outras funções ou retornadas de outras funções.

Funções anônimas: Lambda Funções de uma linha que aceitam qualquer quantidade de parâmetros

Por que usar:

- Otimização de Compilador: Não a necessidade de passar para uma variável ou de definir antes de usar.
  Ex: (lambda x, y: x + y)(5, 3)
- Lexical closure: na definição da função anônima já pode ser definido um valor padrão de variável que sera usado na função.

Ex:

```
def make_adder(n):
    return lambda x: x + n

plus_3 = make_adder(3) # Definindo o valor da variavel N
plus_5 = make_adder(5) # Definindo o valor da variavel N

print(plus_3(4))
#7
print(plus_5(4))
#9
```

- Fácil de testar
- Pode ser executado em paralelo

MapReduce

Map: função built-in aceita uma função como argumento e uma quantidade ilimitada de argumentos(No python 3, no python 2 não, pois ele gera uma lista de resultados, e argumentos infinitos se tornariam grandes demais).

A função será aplicada a todos os argumentos. No python 3 map não é mais built-in, e sim uma classe. No python 3 é gerado um iterador, e não uma lista. (Lazy evaluation).
Ex:

```
dict_a = [{'name': 'python', 'points': 10}, {'name': 'java', 'points': 8}]

print(list(map(lambda x : x['name'], dict_a))) # Output: ['python', 'java']

print(list(map(lambda x : x['points']*10,  dict_a))) # Output: [100, 80]

print(list(map(lambda x : x['name'] == "python", dict_a))) # Output: [True, False]
```

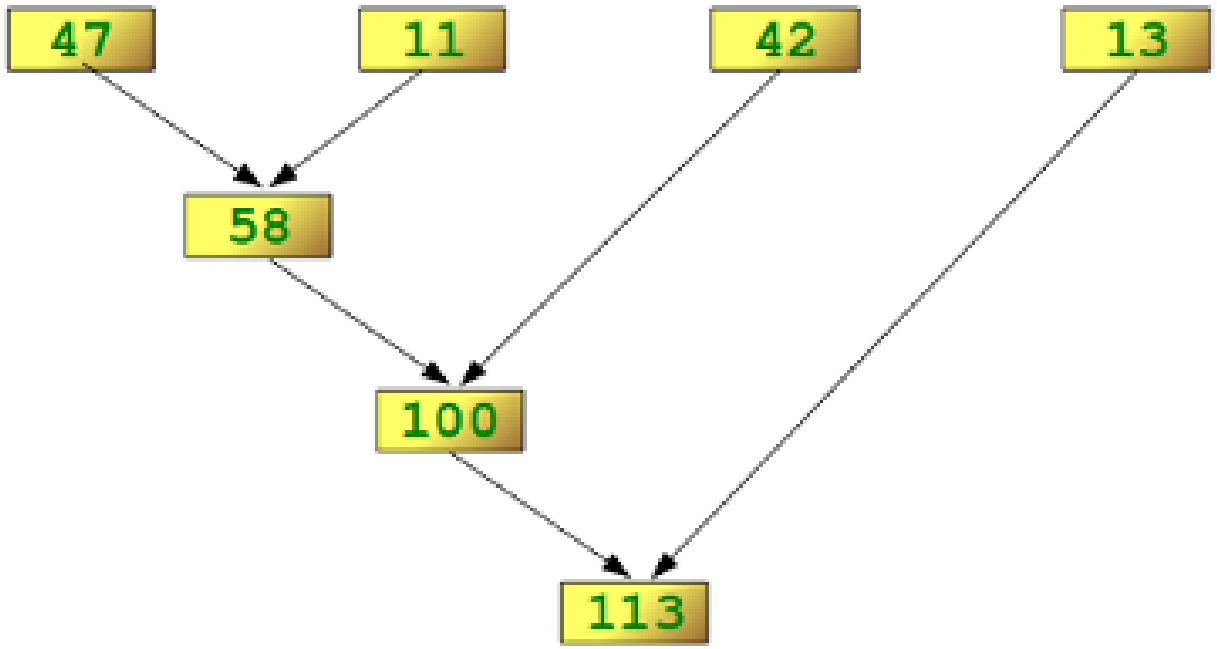
Reduce: Aceita uma funçao e e uma sequencia de parametros. Pega os dois primeiros itens e aplica a função, o terceiro item é calculado junto ao resultado do calculo anterior.
Ex:

```
import functools as ft
ft.reduce(lambda x,y: x+y, [47,11,42,13])
```

MapReduce

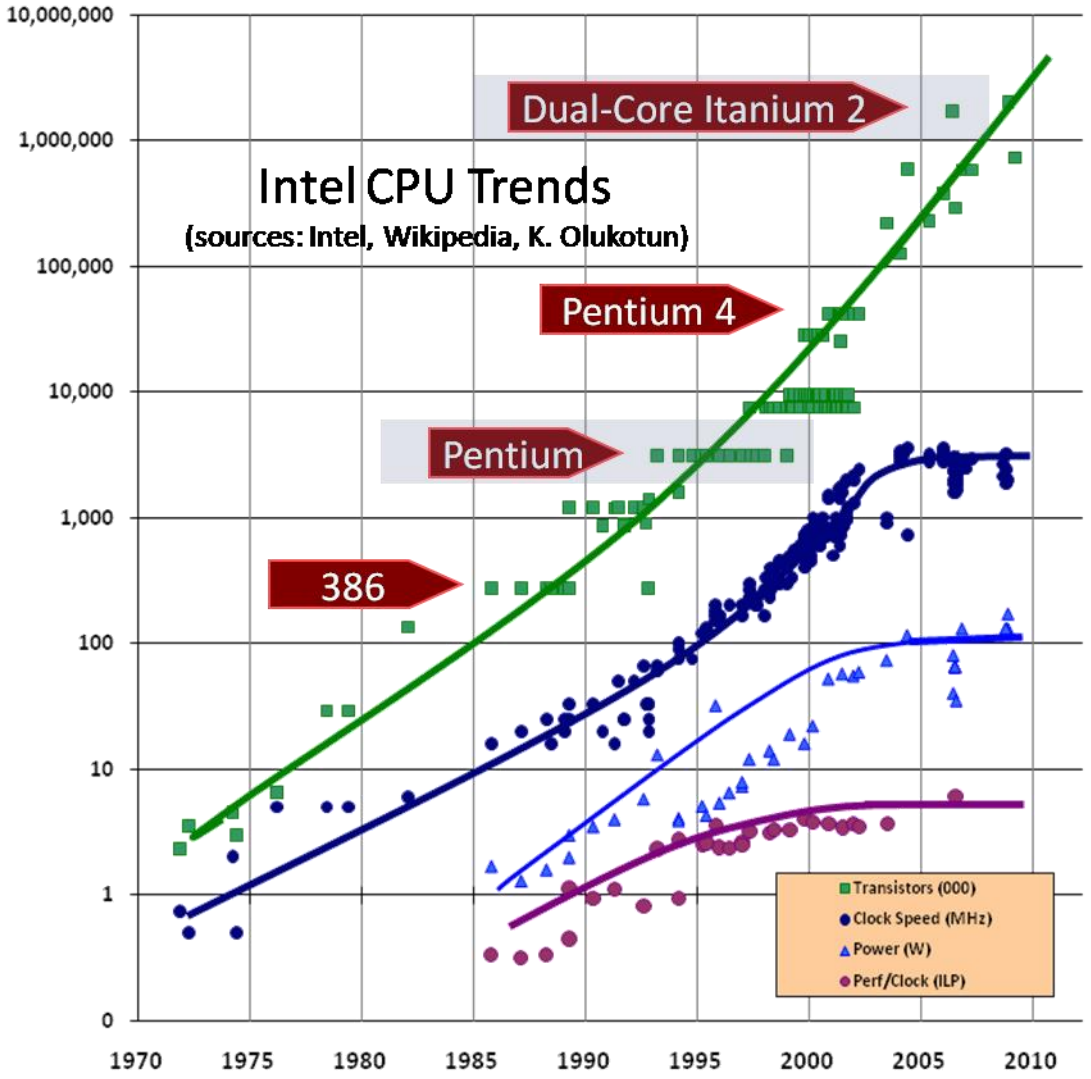
```
>>> reduce(lambda x,y: x+y, [47,11,42,13])
113
```

will be computed as follows:



THE FREE LUNCH IS OVER

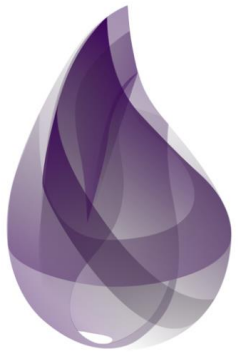
A Fundamental Turn Toward Concurrency in Software

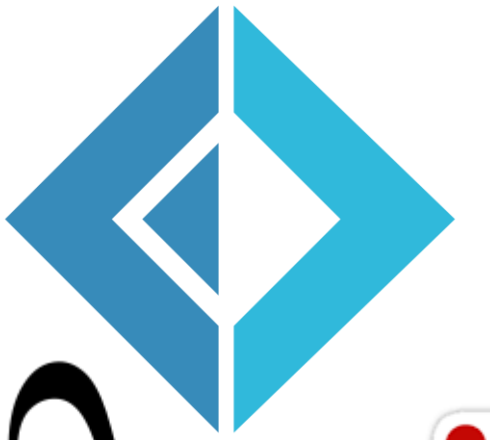


http://www.gotw.ca/publications/concurrency-ddj.htm

LINGUAGENS FUNCIONAIS

- LISP
- Haskell
- Ocaml
- Scala
- Clojure
- F#
- Elixir
- Lua
- Scheme





LINGUAGENS FUNCIONAIS: LISP e Haskell

LINGUAGENS FUNCIONAIS: LUA

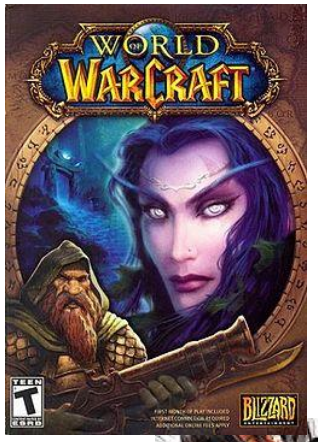
Linguagem Brasileira – PUC – RJ (1993)

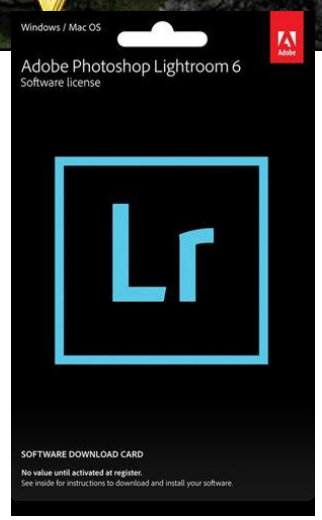
Multiparadigma – Linguagem Extensível

É tipada dinamicamente, interpretada muita usada em jogos e prototipagem.

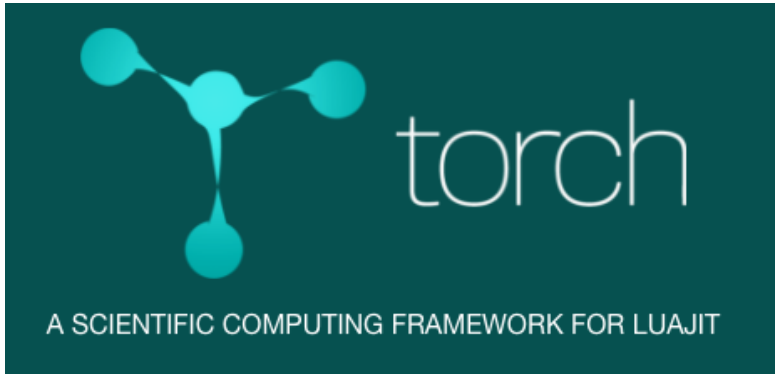
Código Aberto – MIT

Exemplos de uso:





https://www.lua.org/portugues.html



LINGUAGENS FUNCIONAIS: LUA

Torch is a scientific computing framework with wide support for machine learning algorithms that puts GPUs first. It is easy to use and efficient, thanks to an easy and fast scripting language, LuaJIT, and an underlying C/CUDA implementation.

LINGUAGENS FUNCIONAIS: LUA

≠

PYTORCH

LINGUAGENS FUNCIONAIS: ELIXIR

Linguagem Brasileira – José Valim (2012)

Multiparadigma – Funcional

Inspirada na tecnologia Erlang, criada em 1986 pela Ericsson e a base do WhatsApp, a proposta do Elixir é ser escalável, para atender os sistemas com alta performance e em tempo real. De modo simples, a linguagem permite que os sistemas recebam milhares de acesso ao mesmo tempo sem que haja instabilidade para os usuários.

Licença: Apache License 2.0

Exemplos de uso:

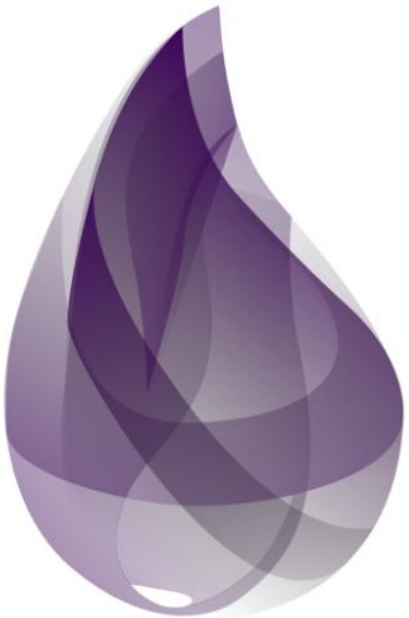
TELNYX

Pinterest

GLOBOSAT

GoPro

Zappos.com



https://elixir-lang.org/

https://hipsters.tech/elixir-a-linguagem-hipster-hipsters-48/

LINGUAGENS FUNCIONAIS: SCALA

Linguagem Brasileira – José Valim (2012)

Multiparadigma – OO + Funcional

Incorpora recursos de linguagens orientadas a objetos e funcionais. Roda na JVM.

Licença: Licença BSD

Exemplos de uso:

LinkedIn

FOURSQUARE

twitter

tumblr.

SIEMENS

Scala

www.scala-lang.org

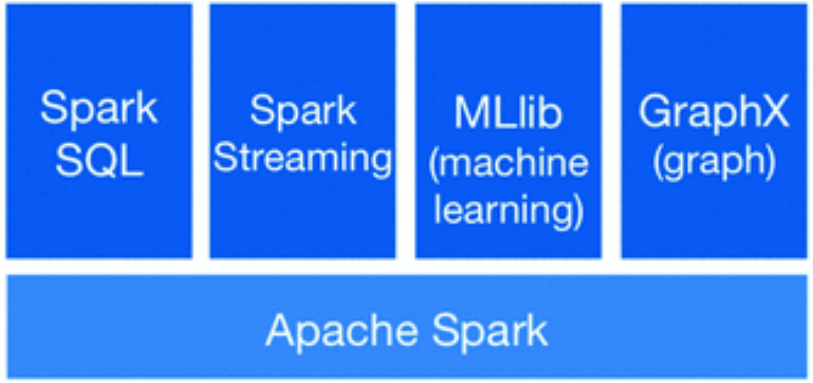
SPARK

SPARK

(Scala, Java, SQL, Python, R)

Apache Spark é um framework de código fonte aberto para computação distribuída.Foi desenvolvido no AMPLab da Universidade da Califórnia e posteriormente repassado para a Apache Software Foundation que o mantém desde então. Spark provê uma interface para programação de clusters com paralelismo e tolerância a falhas.

Licença: Apache License 2.0

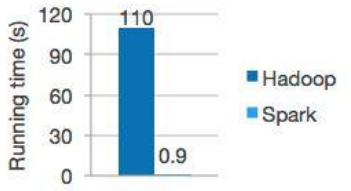


https://spark.apache.org/

Speed

Run workloads 100x faster.

Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.



Logistic regression in Hadoop and Spark

Ease of Use

Write applications quickly in Java, Scala, Python, R, and SQL.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala, Python, R, and SQL shells.

```
df = spark.read.json("logs.json")
df.where("age > 21")
  .select("name.first").show()
```

Spark's Python DataFrame API
Read JSON files with automatic schema inference

SPARK - BENEFICIOS

LAMFO

Leonardo Lacerda Galler

Patrick Henrique Azevedo Gomes