

NA ESTRADA DO DESENVOLVIMENTO

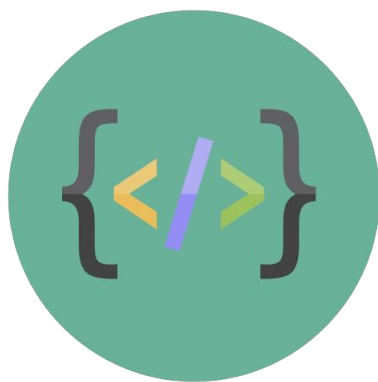
Navegando pelos Paradigmas da Programação



ARTHUR DOS SANTOS

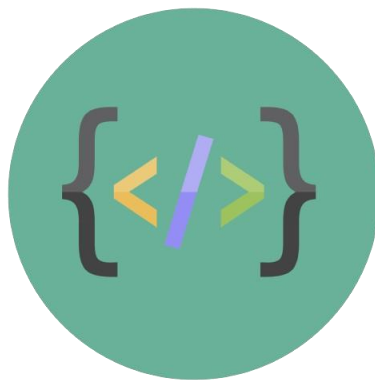
INTRODUÇÃO

Neste ebook, vamos explorar os diferentes paradigmas de programação, entendendo seus principais pontos e vendo exemplos simples de códigos. Esta abordagem ajudará você a compreender como cada paradigma funciona e onde ele pode ser mais eficaz.



PARADIGMA IMPERATIVO

0 Passo a Passo do Código



O paradigma imperativo é o mais tradicional e comum. Ele se baseia em comandos sequenciais para modificar o estado do programa. Isso significa que você escreve instruções que dizem ao computador o que fazer, passo a passo. Utiliza-se amplamente loops e condicionais para controlar o fluxo de execução do programa.

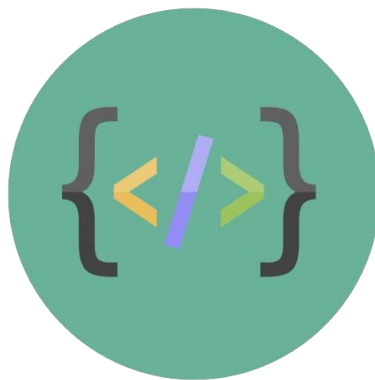


```
# Soma de números de 1 a 10
soma = 0
for i in range(1, 11):
    soma += i
print(soma)
```

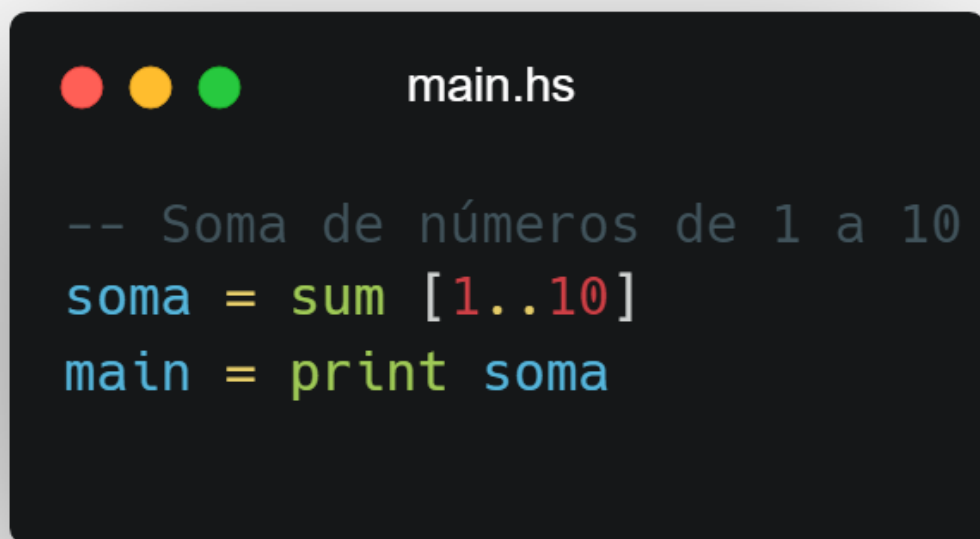
Neste exemplo, temos um loop que itera de 1 a 10, somando os valores em uma variável acumuladora chamada soma.

PARADIGMA FUNCIONAL

Funções Puras e Imutabilidade



O paradigma funcional foca no uso de funções matemáticas e evita mudanças de estado e dados mutáveis. Isso significa que as funções não têm efeitos colaterais e sempre produzem o mesmo resultado para os mesmos argumentos. Este paradigma é ideal para programas que requerem alta confiabilidade e fácil paralelismo.

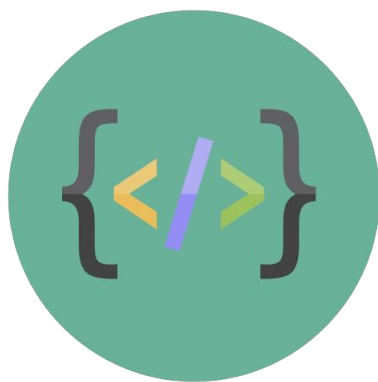


```
-- Soma de números de 1 a 10
soma = sum [1..10]
main = print soma
```

Aqui, a função `sum` recebe uma lista de números de 1 a 10 e retorna a soma deles. O código é conciso e expressa claramente a intenção do programador.

PARADIGMA ORIENTADO A OBJETO

Organizando o Código em Objetos



O paradigma orientado a objetos (OOP) organiza o código em objetos que encapsulam estado e comportamento. Cada objeto é uma instância de uma classe, e as classes podem herdar características de outras classes, promovendo a reutilização de código e a modularidade.



```
main.py

class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

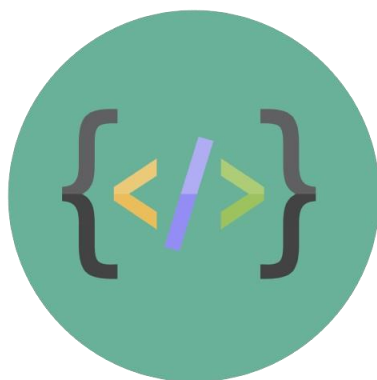
    def cumprimentar(self):
        print(f"Olá, meu nome é {self.nome} e eu tenho {self.idade} anos.")

pessoa = Pessoa("João", 30)
pessoa.cumprimentar()
```

Neste exemplo, criamos uma classe Pessoa com atributos nome e idade, e um método cumprimentar que imprime uma saudação. Criamos uma instância de Pessoa e chamamos o método cumprimentar.

PARADIGMA DECLARATIVO

Diga o Que Fazer, Não como Fazer



No paradigma declarativo, o foco está em declarar o que o programa deve fazer, em vez de como fazê-lo. Exemplos incluem linguagens de consulta e programação lógica. Este paradigma é útil quando se quer simplificar a expressão de lógica complexa sem se preocupar com a implementação detalhada.



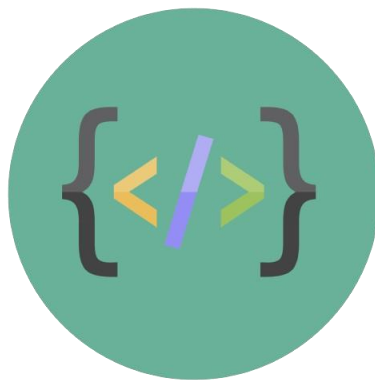
main.sql

```
-- Selecionar todos os nomes de uma tabela de usuários  
SELECT nome FROM usuarios;
```

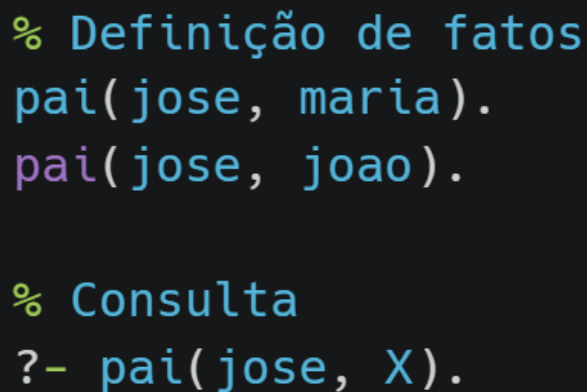
Aqui, declaramos que queremos selecionar todos os nomes da tabela usuários. Não precisamos nos preocupar com o como essa seleção será executada pelo sistema de gerenciamento de banco de dados.

PARADIGMA LÓGICO

Programação Baseadas em
Regras



A programação lógica se baseia em lógica formal e regras de inferência. Definimos fatos e regras, e o sistema de programação lógica infere novas informações a partir dessas regras. É amplamente usado em inteligência artificial e sistemas de conhecimento.



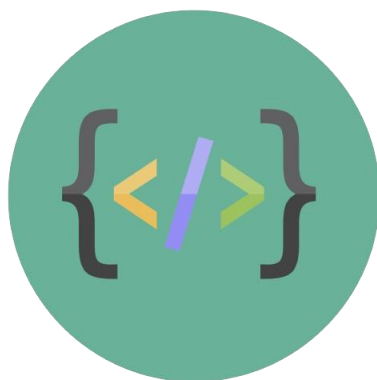
```
% Definição de fatos
pai(jose, maria).
pai(jose, joao).

% Consulta
?- pai(jose, X).
```


Neste exemplo, definimos que José é pai de Maria e João. A consulta pergunta quem são os filhos de José, e o sistema retorna Maria e João.

PARADIGMA FUNCIONAL-REATIVO

Fluxos de Dados Dinâmicos



O paradigma funcional-reativo combina programação funcional e reativa, manipulando fluxos de dados e a propagação de mudanças de forma dinâmica. É muito usado em interfaces de usuário e sistemas que lidam com eventos.



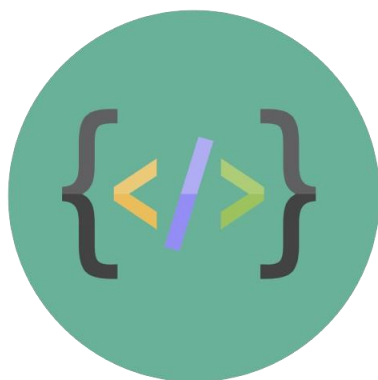
```
// Cria um Observable que emite valores de 1 a 5
const { of } = require('rxjs');
const { map } = require('rxjs/operators');

of(1, 2, 3, 4, 5).pipe(
  map(x => x * 2)
).subscribe(console.log);
```

Aqui, criamos um Observable que emite valores de 1 a 5, e usamos o operador map para dobrar cada valor antes de emitir. Os resultados são então exibidos no console.

PARADIGMA DE PROGRAMAÇÃO CONCORRENTE

Executando Tarefas Simultaneamente



A programação concorrente lida com a execução simultânea de processos e threads. É essencial para aplicativos que precisam realizar várias tarefas ao mesmo tempo, como servidores web e sistemas de tempo real.



```
package main

import (
    "fmt"
    "time"
)

func diga(s string) {
    for i := 0; i < 5; i++ {
        time.Sleep(100 * time.Millisecond)
        fmt.Println(s)
    }
}

func main() {
    go diga("Olá")
    diga("Mundo")
}
```

Neste exemplo, a função `diga` imprime uma mensagem cinco vezes com uma pausa de 100 milissegundos entre cada impressão. No `main`, a função `diga` é executada concorrentemente com a palavra "Olá", enquanto a palavra "Mundo" é impressa sequencialmente.

AGRADECIMENTOS

Esse Ebook foi gerado por IA, e diagramado por humano.

O passo a passo se encontra no meu Github.

Esse conteúdo foi gerado com fins de realizar o desafio do BOOTCAMP "Santander 2024 - Fundamentos de IA para Devs" promovido pela Digital Innovation One (DIO). Não foi realizada uma validação cuidadosa humana no conteúdo e pode conter erros gerados por uma IA.

Link do repositório com passo a passo

