Téc em Desenvolvimento
de Sistemas Bilíngue

Desenvolver Código

Orientado a Objetos

UC4 | Prof. Leonardo de Souza

Estruturas de Dados

Imagine que você tem uma prateleira cheia de livros desorganizados. Quando precisa encontrar um livro específico, é como procurar uma agulha no palheiro, certo?

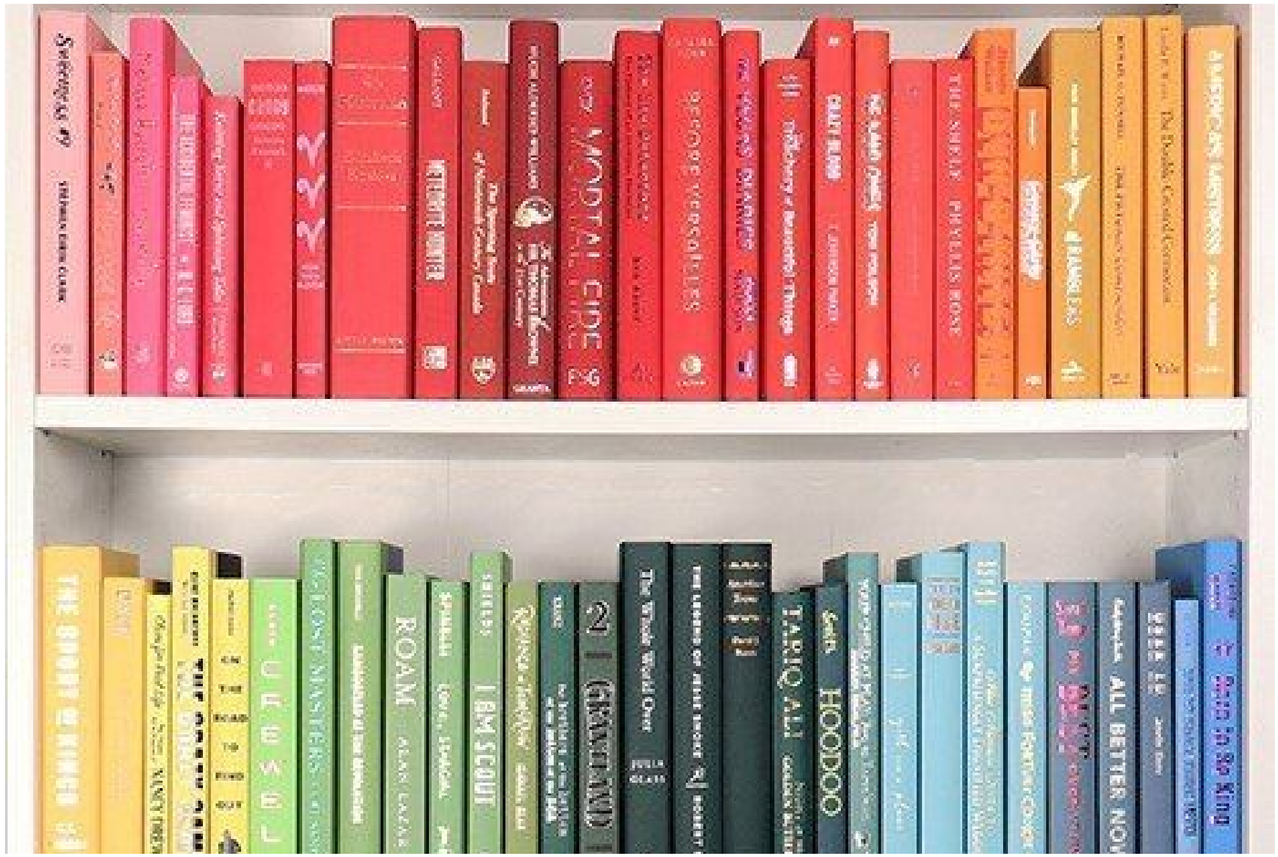
Estruturas de Dados

Difícil achar algo aqui, não?

Estruturas de Dados

Agora, pense em uma maneira de organizar esses livros para que você possa encontrá-los facilmente quando quiser ler ou estudar.

Estruturas de Dados



Muito melhor!

Estruturas de Dados

Pergunta: como podemos organizar os livros de uma biblioteca?

Estruturas de Dados

Estruturas de dados são como diferentes maneiras de organizar seus livros: você pode classificá-los por gênero, autor ou até mesmo por ordem alfabética.

Estruturas de Dados

Cada estrutura de dados tem suas próprias vantagens e desvantagens, e escolher a certa pode tornar sua vida muito mais fácil quando se trata de encontrar e gerenciar seus livros.

Estruturas de Dados

Existe um recurso que nos permite percorrer cada livro em nossa estante, nos ajudando a encontrar e ler cada um deles de forma eficiente.

É aí que entra o...

Estruturas de Dados

For

For

Vamos pensar na nossa prateleira de livros novamente. Digamos que você queira verificar todos os livros para ver quais deles são de ficção científica. Uma maneira de fazer isso é olhar um por um, certo? Bem, o loop for é como um assistente eficiente que faz isso de maneira automática.

For

Com ele, podemos criar uma instrução que diz algo como:

'Pegue o primeiro livro, veja se é de ficção científica. Se sim, ótimo, se não, pegue o próximo livro.'

E ele continua fazendo isso até verificar todos os livros na prateleira.

```
for (inicialização; condição; incremento/decremento) {
    // bloco de código a ser repetido
}
```

For

Inicialização: É onde você inicializa o contador ou variável de controle do loop. Geralmente, você define uma variável e atribui um valor inicial a ela. Esta parte é executada apenas uma vez, no início do loop.

For

Condição: É a condição que determina se o loop deve continuar ou não. Enquanto essa condição for verdadeira, o bloco de código dentro do loop será executado. Se a condição for falsa, o loop termina e a execução continua após o loop.

For

Incremento/Decremento: É onde você altera o valor da variável de controle do loop. Isso geralmente envolve aumentar ou diminuir o valor da variável. Essa parte é executada após cada iteração do loop.

For

Bloco de código: É o conjunto de instruções que serão executadas a cada iteração do loop. Essas instruções podem ser qualquer coisa que você queira repetir várias vezes, como manipulação de dados, chamadas de função, etc.

For

Importante: **ITERACÃO** é o nome que se dá a cada “volta” que o loop dá!

```typescript
// Lista de livros
let livros: string[] = ["Fundação", "Neuromancer", "Orgulho e Preconceito", "Duna"];

// Verificar se cada livro é de ficção científica
for (let i = 0; i < livros.length; i++) {
    if (livros[i] === "Fundação" || livros[i] === "Neuromancer" || livros[i] === "Duna") {
        console.log(`${livros[i]} é um livro de ficção científica.`);
    } else {
        console.log(`${livros[i]} não é um livro de ficção científica.`);
    }
}
```

For

Executa uma iteração com início e término determinados.

Não precisa estar ligada a um array.

O iterador pode ser de qualquer tipo.

Estruturas de Dados

For In

For In

Vamos pensar novamente na nossa biblioteca de livros.
Digamos que você queira achar um determinado autor dentro de um determinado gênero.
Uma maneira de fazer isso é olhar cada livro individualmente, certo?

For In

Com o loop for...in, podemos criar uma instrução que diz algo como: 'Pegue o primeiro livro EM ficção científica, veja quem é o autor.
Depois, pegue o próximo livro.' E ele continua fazendo isso até verificar todos os livros de ficção.

```javascript
// Definindo um objeto representando os livros de ficção com seus respectivos autores
let ficcao = {
    "Fundação": "Isaac Asimov",
    "Duna": "Frank Herbert"
};

// Iterando sobre os livros de ficção e exibindo os autores
for (let livro in ficcao) {
    console.log(`Autor do livro "${livro}": ${ficcao[livro]}`);
}
```

For In

- O for-in itera sobre as propriedades de um objeto.
- É usado principalmente para iterar sobre as chaves de um objeto.
- Não é recomendado para iterar sobre arrays, pois pode incluir propriedades herdadas e não numéricas.

For In

Não é recomendado utilizar o forIn para percorrer arrays

```
const arr = [1, 2, 3];

for (const index in arr) {
    console.log(typeof index); // Imprime '0', '1', '2'
}
```

Quando usar

- É útil quando você precisa acessar as chaves e os valores associados de um objeto durante a iteração.

Porque usar

- Sintaxe simples
- Fornece tanto as chaves quanto os valores

Estruturas de Dados

For Of

For Of

Ainda no exemplo da nossa biblioteca, nós podemos percorrer nossos livros de outra forma, bem mais simples.

```javascript
// Array representando os títulos dos livros na prateleira
let meusLivros = ["Fundação", "Duna", "Orgulho e Preconceito"];

// Iterando sobre os títulos dos livros e exibindo-os
for (let livro of meusLivros) {
    console.log(livro);
}
```

For In

- O for-of é uma maneira de iterar sobre os valores de um objeto iterável, como arrays, strings, mapas, conjuntos, etc.
- É mais intuitivo e limpo do que for-in, pois itera apenas sobre os valores e não sobre as chaves.
- Não é possível usar for-of em objetos não-iteráveis.

Quando usar

- Quando você precisa apenas iterar sobre os valores dos elementos.
- Quando você precisa apenas dos valores e não precisa se preocupar com o índice ou com a referência ao array original.

Porque usar

- Fornece uma sintaxe mais limpa e concisa.

Estrutura de Dados

For Each

For Each

Agora imagine que queremos ler todos os livros de um determinado autor. Existe um modo de percorrer (ler) todos os livros desejados e executar uma ação para cada um através do forEach.

```
let tolkienLivros = ['Senhor dos Aneis', 'O Hobbit', 'Contos Inacabados'];

tolkienLivros.forEach(livro => {
    console.log(`Eu já li ${livro}`);
});
```

```
let tolkienLivros = ['Senhor dos Aneis', 'O Hobbit', 'Contos Inacabados'];

tolkienLivros.forEach(function(livro) {
    console.log(`Eu já li ${livro}`);
});
```

For Each

- Executa uma determinada função para cada elemento do array.
- É uma maneira mais conveniente e segura de iterar sobre os elementos de um array em comparação com for-in.
- Não é possível usar o forEach em objetos não-iteráveis (ou seja, só se usa em arrays). Pode ser usado em objetos mas não retorna valor dos atributos.

For Each

Não utiliza um iterador como contador, mas sim passa cada elemento do array para a função de callback (uma função que é passada como argumento para outra função)

For Each

Essa função de callback aceita até 3 parâmetros:

- element: O valor do elemento atual sendo processado no array. (ex: "carro")
- index: O índice do elemento atual sendo processado no array.("ex:0")
- array: O array original sobre o qual forEach() foi chamado.(ex:["carro", "moto", "bicicleta"])

```
const arr = ["carro", "moto", "bicicleta"];

arr.forEach((element, index, array) => {
    console.log(`Elemento ${element} no índice ${index}`);
    console.log('Array original:', array);
});
```

Quando usar

- É usado quando você precisa iterar sobre os valores dos elementos e também precisa acessar o índice ou o array original durante a iteração.
- É útil quando você precisa realizar uma operação em cada elemento do array e precisa acessar o índice ou o array original durante essa operação.

Porque usar

- Fornece uma maneira conveniente de fazer algo com cada elemento do array.
- Permite passar uma função de callback que pode receber até três argumentos: o valor do elemento, o índice do elemento e o array original.

Estrutura de Dados

RESUMO

forIn

- Use quando estiver iterando sobre as propriedades e valores de um objeto.
- Evite usar em arrays.

forOf

- Use quando estiver iterando sobre os valores de uma estrutura de dados iterável, como um array.
- Ideal quando você só precisa dos valores dos elementos e não precisa se preocupar com o índice ou com a referência ao array original.

forEach

- Use quando estiver iterando sobre os valores de uma estrutura de dados iterável, como um array, mas precisa realizar uma operação em cada elemento do array e precisa acessar o índice ou o array original durante essa operação.

Estrutura de Dados

ENUMS

O que são

- Enum é uma abreviação de "enumeration", que em português seria "enumeração". Basicamente, é uma maneira de nomear um conjunto de constantes relacionadas. Em TypeScript, um enum é uma maneira de declarar um conjunto fixo de valores que têm um significado específico em um contexto.

O que são

- São como uma lista de palavras-chave que você pode usar para representar valores específicos em seu código. Cada palavra-chave tem um valor associado a ela.

Estrutura de Dados

Características

Características

- Palavras-chave Descritivas: Os enumeradores permitem usar palavras-chave significativas para representar valores específicos.

Características

- Valores Automáticos ou Personalizados: Os valores dos enumeradores podem ser atribuídos automaticamente começando de 0 ou podem ser definidos manualmente para cada palavra-chave

Características

- Suporte a Valores Numéricos e de Texto: Os enumeradores podem conter tanto valores numéricos quanto de texto.

Características

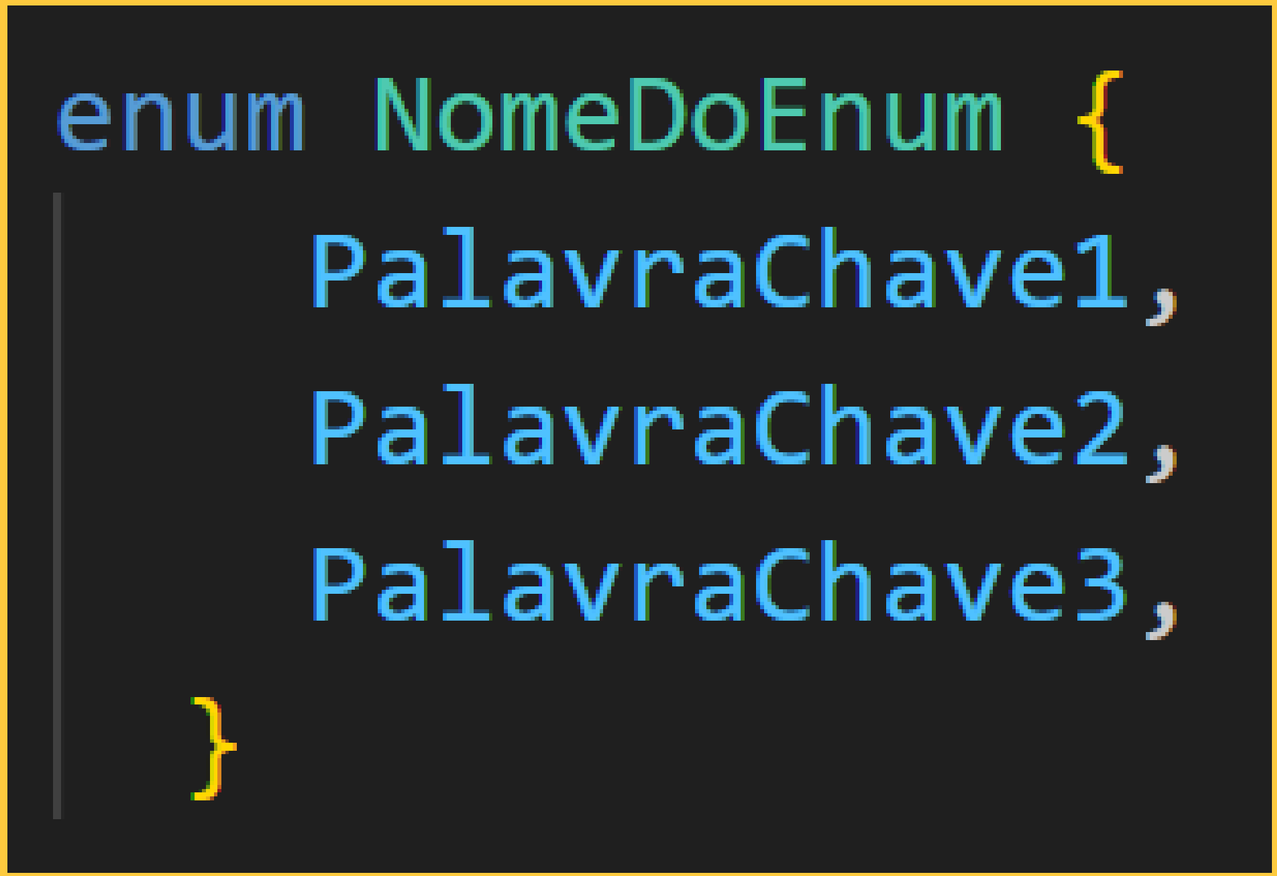
- Fácil de Iterar: Você pode facilmente percorrer todas as palavras-chave de um enumerador usando loops ou métodos de iteração.

Estrutura de Dados

Sintaxe

Sintaxe

- Para criar um enumerador em TypeScript, você usa a palavra-chave enum seguida por uma lista de palavras-chave separadas por vírgulas.


enum NomeDoEnum {
PalavraChave1,
PalavraChave2,
PalavraChave3,
}

Estrutura de Dados

Exemplos

Exemplos

- Vamos ver alguns exemplos para entender melhor como os enumeradores funcionam:

```
enum DiasDaSemana {
    Segunda,
    Terça,
    Quarta,
    Quinta,
    Sexta,
    Sábado,
    Domingo,
  }

  console.log(DiasDaSemana.Segunda); // Saída: 0
```

```typescript
enum Cores {
    Vermelho = "#FF0000",
    Verde = "#00FF00",
    Azul = "#0000FF",
}

console.log(Cores.Verde); // Saída: "#00FF00"
```

```
enum Direcoes {
    Norte,
    Sul,
    Leste,
    Oeste,
}

for (let direcao in Direcoes) {
    console.log(direcao);
}
```

Atenção

- Quando criamos um enum, damos um nome a ele, e as palavras-chave são as constantes. Elas nunca mudam de valor, por isso chamamos elas de constante. Elas também devem ter algum tipo de significado em comum para a aplicação.

Atenção

- Os valores das constantes podem ser declarados das seguintes formas:

```
//Numérico e implícito:
export enum Level {
BLUE, // 0
YELLOW, // 1
ORANGE, // 2
RED // 3
}
```

```
//Numérico e explicito
export enum Level {
BLUE = 0,
YELLOW = 10,
ORANGE = 20,
RED = 30
}
```

```
//Numérico e explicito incremental
export enum Level {
BLUE = 1, // 1
YELLOW, // 2
ORANGE, // 3
RED // 4
}
```

```
//String:
enum Level {
BLUE ="Blue",
YELLOW ="Yellow",
ORANGE ="Orange",
RED ="Red"
}
```

Quando usar ENUMS?

- Você pode usar enums sempre que tiver um conjunto fixo e limitado de valores relacionados que precisam ser representados de forma clara e concisa no seu código. Eles são úteis em situações como status, opções de menu, dias da semana, meses, e assim por diante.

```
// Definição do enum Level
export enum Level {
    BLUE = 1,// Valor atribuído: 1
    YELLOW,// O valor será automaticamente incrementado para 2
    ORANGE,// O valor será automaticamente incrementado para 3
    RED// O valor será automaticamente incrementado para 4
  }
```

```typescript
import { Level } from "./Level";

export class Survivor {
  protected level: Level;   // Propriedade level do tipo Level

  constructor(level: Level = Level.BLUE) {
    this.level = level;
  }

  // Método para obter o nível do sobrevivente
  getLevel(): Level {
    return this.level;
  }
```

```
// Método para subir de nível
levelUp(): Level | null {
  // Obtém o próximo nível baseado no nível atual
  const nextLevel = this.getNextLevel();

  // Verifica se o próximo nível existe
  if (nextLevel !== null) {
    this.level = nextLevel;   // Define o próximo nível como o novo nível
    return this.level;        // Retorna o novo nível
  } else {
    console.log("Não há próximo nível disponível.");
    return null;
  }
}
```

```
  // Método auxiliar para obter o próximo nível
  private getNextLevel(): Level | null {
    switch (this.level) {
      case Level.BLUE:
        return Level.YELLOW;
      case Level.YELLOW:
        return Level.RED;
      case Level.RED:
        return Level.GREEN;
      case Level.GREEN:
        return null; // Não há próximo nível após o nível verde
    }
  }
}
```

```
  // Método auxiliar para obter o próximo nível
  private getNextLevel(): Level | null {
    switch (this.level) {
      case Level.BLUE:
        return Level.YELLOW;
      case Level.YELLOW:
        return Level.RED;
      case Level.RED:
        return Level.GREEN;
      case Level.GREEN:
        return null; // Não há próximo nível após o nível verde
    }
  }
}
```

Enum Level

Aqui, temos um enum chamado Level, que representa os diferentes níveis de um jogo.

Os valores numéricos podem ser atribuídos explicitamente a cada constante do enum. Por exemplo, BLUE tem o valor 1.

Se não for atribuído um valor para uma constante do enum, ele automaticamente será incrementado em relação ao valor do anterior. Portanto, YELLOW tem o valor 2, ORANGE tem o valor 3 e RED tem o valor 4.

Enum Level

Classe Survivor:

-A classe Survivor é definida com uma propriedade protegida chamada level, que é do tipo Level, e que armazena o nível do sobrevivente.

-Isso significa que o nível do sobrevivente é representado por uma das constantes definidas no enum Level.

-O método getLevel() retorna o nível atual do sobrevivente.

-O método levelUp() permite que o sobrevivente suba de nível.

Enum Level

- Quando um objeto da classe Survivor é instanciado, o nível inicial é definido como um dos valores do enum Level. Por exemplo, podemos criar um sobrevivente de nível BLUE da seguinte maneira:

```
const survivor = new Survivor();
console.log(survivor.getLevel()); // Saída: Level.BLUE
```

Enum Level

Da mesma forma, o método levelUp() permite que o sobrevivente suba de nível. Por exemplo:

```
survivor.levelUp(); // Sobrevivente sobe de nível
console.log(survivor.getLevel()); // Saída: Level.YELLOW
```

Após chamar levelUp(), o nível do sobrevivente é incrementado para YELLOW, que é o próximo valor no enum Level. Assim, a classe Survivor utiliza o enum Level para representar e gerenciar os níveis do sobrevivente.

Enum Level

E o que podemos perceber com isso?

Que além de criar o conjunto de constantes, estamos também criando um novo TIPO de dado. Isso significa que podemos usar esse enum para declarar variáveis, parâmetros de função, retornos de função e muito mais.

Estrutura de Dados

Exercícios

Enum Level

Desenvolva um sistema para uma pizzaria em TypeScript.
Você deve definir um enum chamado SaborPizza que representará os sabores disponíveis de pizzas.
Crie uma classe chamada Pizza que vai ter os parâmetros sabor, tamanho e preço. Crie nesta classe um método chamado descrição que retorna uma string contendo a descrição da pizza no formato "Pizza [sabor], Tamanho: [tamanho], Preço: R$ [preco]".

Crie três instâncias de pizzas com diferentes sabores, tamanhos e preços e exiba no console.

Estrutura de Dados

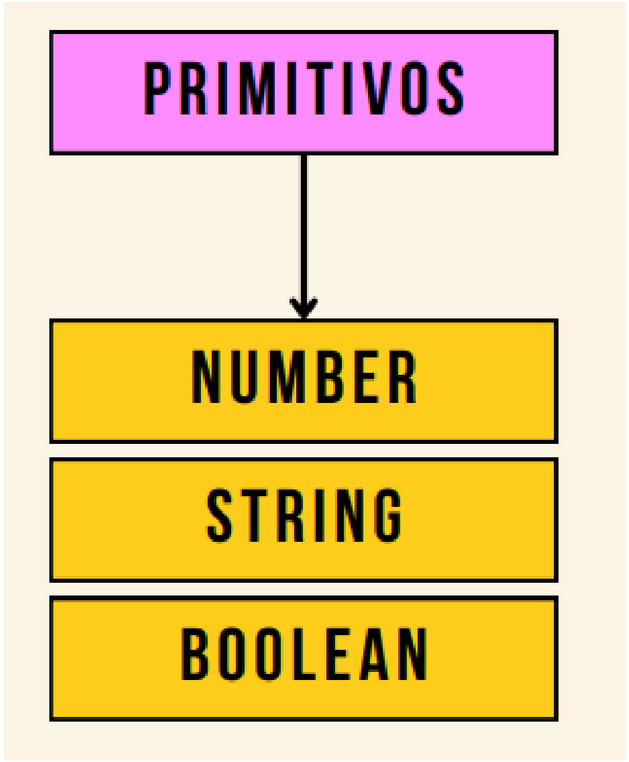
Estruturas de Dados

Estruturas de dados

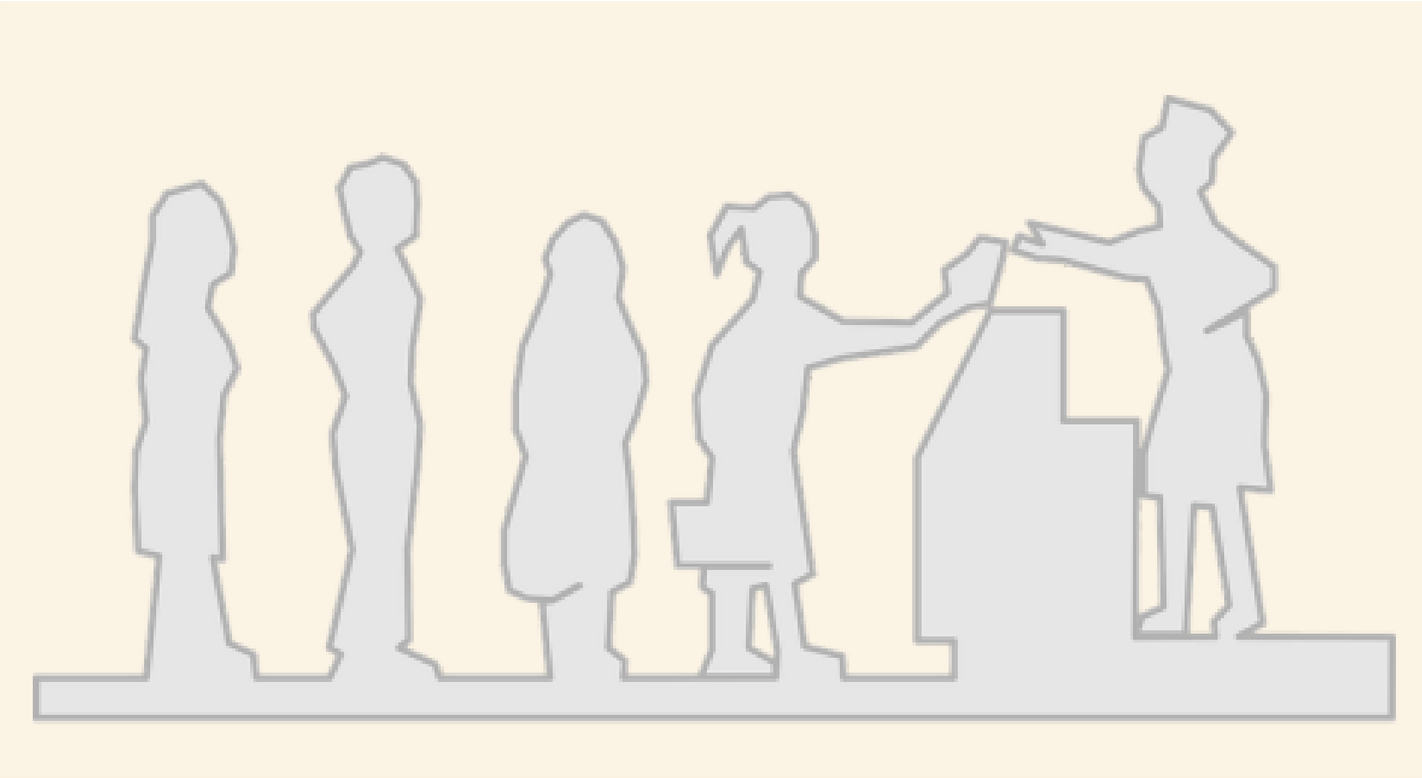
Estruturas de dados são formas de organizar e armazenar dados de maneira eficiente para que possam ser manipulados e acessados de forma conveniente.

Elas fornecem um meio de representar e organizar informações em computadores de forma que permitam operações eficientes, como inserção, remoção, pesquisa e ordenação.

Tipos de dados



Tipos de dados



Tipos de dados

Tipos de dados

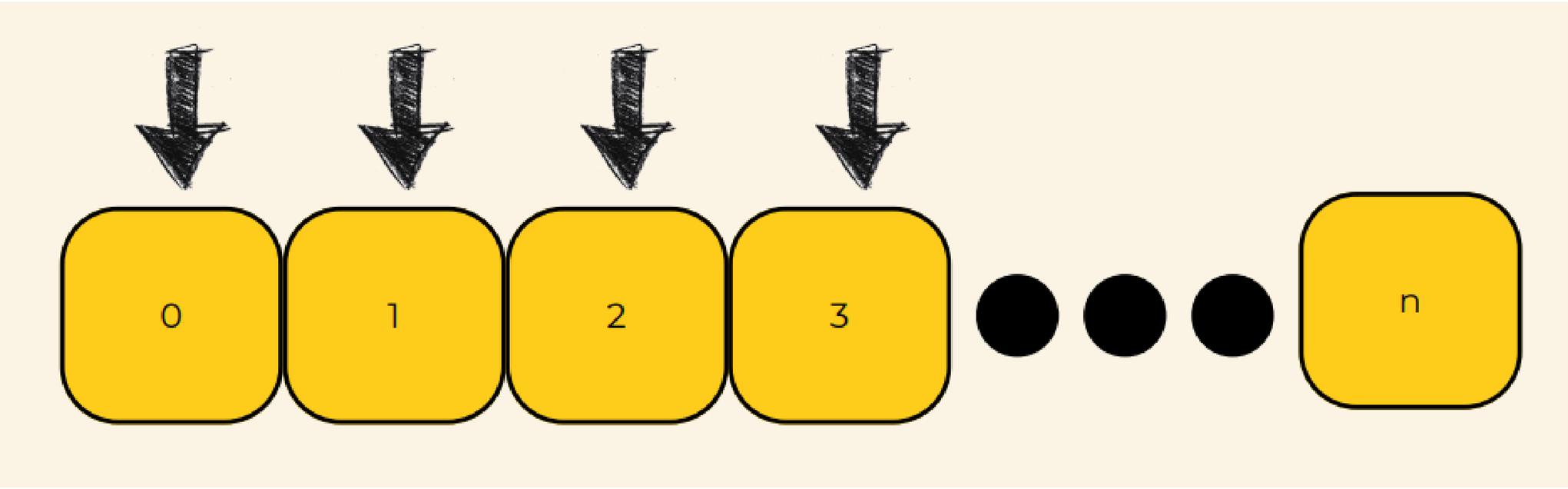
Estruturas de dados

Tipos não-primitivos de organização de dados para atender aos diferentes requisitos de processamento

Estrutura de Dados

Arrays

Array



Array

Uma coleção de elementos do mesmo tipo, organizados em uma sequência indexada. Os arrays oferecem acesso rápido aos elementos por meio de índices, mas têm tamanho fixo.

Coleção

Os elementos de um array são organizados em uma sequência ordenada, o que significa que cada elemento tem uma posição específica dentro do array. Esta ordem é mantida quando os elementos são inseridos, removidos ou acessados.

Índice

Cada elemento em um array é identificado por um índice, que é um número inteiro que representa sua posição no array. Os índices geralmente começam em 0 (em muitas linguagens de programação), o que significa que o primeiro elemento está no índice 0, o segundo está no índice 1 e assim por diante.

Acesso Rápido

Os arrays oferecem acesso rápido aos elementos com base no índice. Dado um índice, você pode acessar o elemento correspondente diretamente, o que significa que o tempo de acesso é constante e independente do tamanho do array.

```ts
let numeros: Array<number> = [1, 2, 3, 4, 5];

let nomes: Array<string> = ["Alice", "Bob", "Carol"];

let valores: Array<any> = [1, "dois", true];
```

```
//Acessando elementos
let numeros2: Array<number> = [10, 20, 30, 40, 50];
console.log(numeros[0]); // Saída: 10
console.log(numeros[2]); // Saída: 30
```

```
//Iterando
let nomes2: Array<string> = ["Alice", "Bob", "Carol"];
for (let i = 0; i < nomes.length; i++) {
    console.log(nomes[i]);
}
```

```
//Métodos
let numeros3: Array<number> = [1, 2, 3, 4, 5];
numeros.push(6); // Adiciona um elemento no final da lista
console.log(numeros); // Saída: [1, 2, 3, 4, 5, 6]

numeros.pop(); // Remove o último elemento da lista
console.log(numeros); // Saída: [1, 2, 3, 4, 5]
```

Resumo

Posições bem definidas

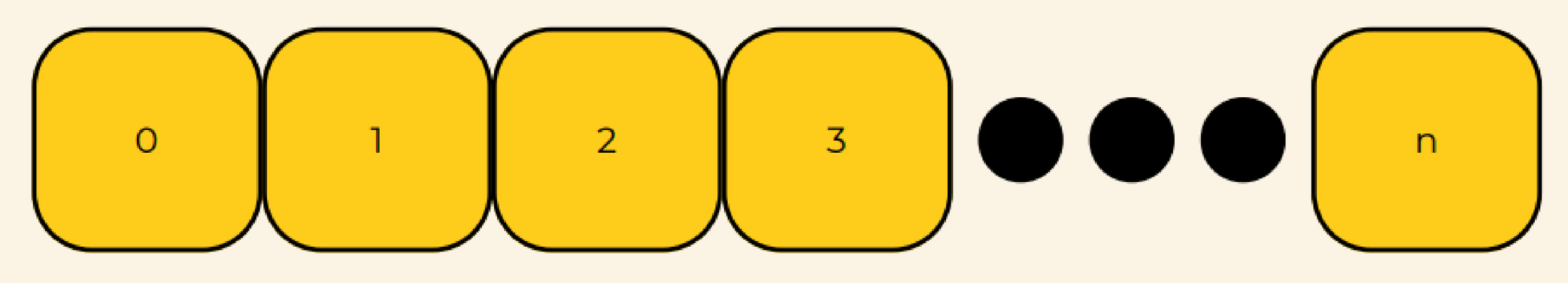
Tamanho limitado (nem sempre)

Itens podem ser adicionados em qualquer posição

Estrutura de Dados

Listas (um array de luxo)

Listas



Listas

Uma sequência de elementos que podem crescer ou encolher dinamicamente e permitem inserções e remoções eficientes em qualquer ponto da lista.

As listas ligadas são uma estrutura de dados onde cada elemento, além de armazenar o valor, também armazena uma referência para o próximo elemento na lista. Essa característica de "ligação" entre os elementos é o que dá o nome a essa estrutura.

Inserção e Remoção

Em listas, tanto a adição quanto a remoção dos itens pode ser feita em qualquer posição.

Inserção e Remoção

Operações Principais:

- Inserção no início: Adiciona um elemento no início da lista.
- Inserção no final: Adiciona um elemento no final da lista.
- Inserção em uma posição específica: Adiciona um elemento em uma posição específica da lista.
- Remoção no início: Remove o primeiro elemento da lista.
- Remoção no final: Remove o último elemento da lista.
- Remoção em uma posição específica: Remove o elemento em uma posição específica da lista.

Inserção e Remoção

Operações Principais:

- Acessar elementos:
- Obter elemento por índice: Retorna o elemento em uma posição específica da lista.
- Verificar se a lista está vazia:
- Verificar se a lista está vazia: Retorna verdadeiro se a lista estiver vazia, caso contrário, retorna falso.
- Obter o tamanho total da lista:
- Obter o tamanho da lista: Retorna o número total de elementos na lista.

Tamanho Dinâmico

Posições bem definidas

Tamanho se adapta à posição ocupada de maior índice

Itens podem ser adicionados em qualquer posição

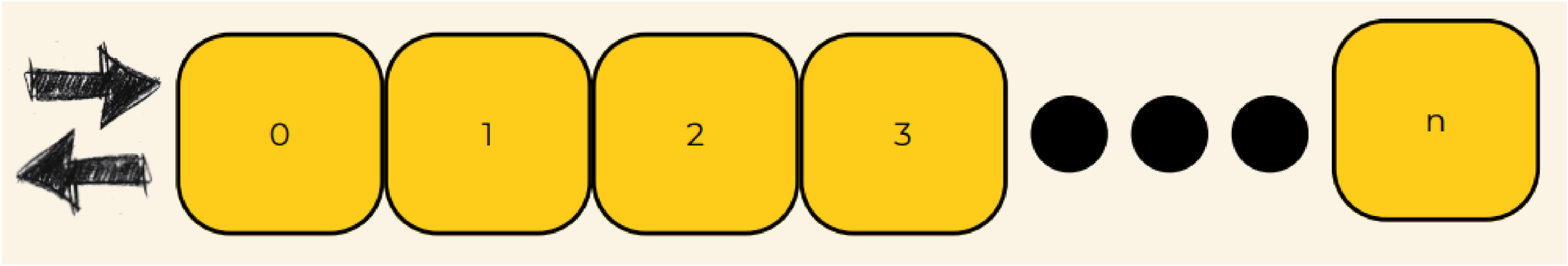
Array de "luxo"

Não possui ordem de saída definida

Estrutura de Dados

Pilhas (stacks)

Pilhas (stacks)



Pilhas (stacks)

Uma pilha é uma estrutura de dados que segue o princípio de "último a entrar, primeiro a sair" (LIFO - Last In, First Out).

Isso significa que o último elemento inserido na pilha é o primeiro a ser removido.

É como uma pilha de pratos, onde você sempre adiciona um novo prato no topo da pilha e remove o prato mais recente que foi adicionado.

Pilhas (stacks)

Operações PrincipaiPush (Empurrar): Adiciona um novo elemento no topo da pilha.

- Pop (Remover): Remove o elemento mais recentemente adicionado à pilha, que está no topo da pilha.
- Peek (Espiar): Retorna o elemento no topo da pilha sem removê-lo.
- isEmpty (Está vazia): Verifica se a pilha está vazia.
- Size (Tamanho): Retorna o número de elementos na pilha.

Pilhas (stacks)

Posições bem definidas

Tamanho se adapta à posição ocupada de maior índice

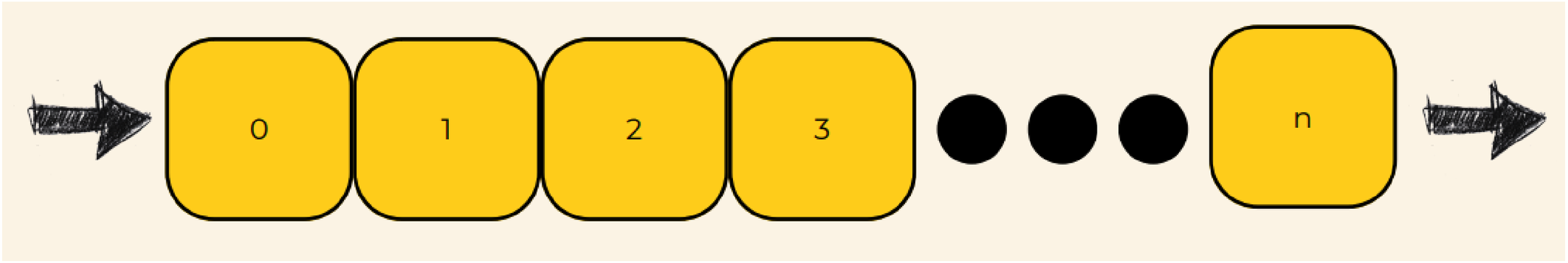
Itens podem ser adicionados apenas no menor índice (zero)

Saída deve ser feita pelo menor índice ocupado

Estrutura de Dados

Filas (queues) Primeiro a Chegar Primeiro a Sair (PCPS) First In, First Out (FIFO)

Filas (queues)



Filas (queues)

Uma fila é uma estrutura de dados que segue o princípio de "primeiro a entrar, primeiro a sair" (FIFO - First In, First Out). Isso significa que o primeiro elemento inserido na fila será o primeiro a ser removido. Pense em uma fila de pessoas esperando em um caixa de supermercado: a pessoa que chegou primeiro é a primeira a ser atendida e sair.

Filas (queues)

Operações Principais:

- Enqueue: Adiciona um elemento ao final da fila.
- Dequeue: Remove e retorna o elemento do início da fila.
- Front: Retorna o elemento no início da fila sem removê-lo.
- Size: Retorna o número de elementos na fila.
- isEmpty: Verifica se a fila está vazia.

Filas (queues)

Aplicações:

- As filas são comumente usadas em situações onde a ordem de chegada é importante, como simulações, sistemas de processamento de pedidos, algoritmos de busca em largura (BFS), entre outros.
- Elas são usadas em muitas situações do mundo real, como processamento de tarefas em sistemas operacionais, gerenciamento de solicitações em servidores da web, etc.

Estrutura de Dados

Desafio

Desafio 1