




# Dart Básico - Aula 3



Uma matriz é uma coleção de variáveis de mesmo tipo, acessíveis com um único nome e armazenados contiguamente na memória.

A individualização de cada variável de um vetor é feita através do uso de índices.

Os Vetores são matrizes de uma só dimensão.



```
// Criando uma lista dentro da outra
List<List<int>> tecladoTelefone = [];

// Populando as listas da minha matriz
tecladoTelefone.add([7, 8, 9]);
tecladoTelefone.add([4, 5, 6]);
tecladoTelefone.add([1, 2, 3]);

// Percorrendo a matriz
for (int i = 0; i < tecladoTelefone.length; i++) {
    for (int y = 0; y < tecladoTelefone[i].length; y++) {
        print(tecladoTelefone[i][y]);
    }
}
```



É possível, no dart, nós criarmos os nosso próprios tipos de dados customizados, veja:

```
// declarando novos tipos
typedef MatrizInt = List<List<int>>;

typedef BoolList = List<bool>;

// Testando os tipos criados
void main() {
  MatrizInt matriz = [];

  print(matriz);
}
```

Podemos criar também tipos de funções, para a criação de eventos e passagem por parâmetro



```
// Criando um tipo de função
typedef OnPercorreListaItem = void Function(int index, int itemAtual);

// meu forEach personalizado
void meuForEach(List<int> lista, OnPercorreListaItem evento) {
    for (int i = 0; i < lista.length; i++)
        evento(i, lista[i]);
}

// Testando os métodos e tipos criados
void main() {
    List<int> listaTeste = [7, 14, 33, 331];


    meuForEach(listaTeste, (index, itemAtual) => print("[${index}] ${itemAtual}"));
}
```

Aproveitando a possibilidade de utilizarmos matrizes, vamos criar um jogo da velha. Os jogadores (X e O) serão gerados por valores numéricos.

O que a aplicação deverá controlar é:

- A exibição do tabuleiro
- A vez de cada jogador
- Quando um jogador ganhou, completando uma coluna, linha ou diagonal
- E quando não existirem mais jogadas disponíveis, deve encerrar o jogo





Uma exceção é um sinal que indica que algum tipo de condição excepcional ocorreu durante a execução do programa. Assim, exceções estão associadas a condições de erro que não tinham como ser verificadas durante a compilação do programa.

As duas atividades associadas à manipulação de uma exceção são:

- **Geração**  
A sinalização de que uma condição excepcional (por exemplo, um erro) ocorreu, e
- **Captura**  
A manipulação (tratamento) da situação excepcional, onde as ações necessárias para a recuperação da situação de erro são definidas.




### Disparando uma exceção em Dart

```
// Lançando uma exceção  
throw FormatedException("Teste")  
  
// Lançando uma exceção  
throw "Teste só de string";
```



## Capturando uma exceção gerada



```
try {
    rodarProcesso();
} catch (e) {
    print("Erro ao executar o processo. Erro: $e");
}
```

```
try {
    rodarProcesso();
} catch (e) {
    gravarLog(e);


    rethrow;
}
```

## Principais exceções padrões no dart



Exceção	Descrição
DeferredLoadException	Ele é lançado quando uma biblioteca adiada falha ao carregar.
FormatException	É a exceção que é lançada quando uma string ou algum outro dado não tem um formato esperado
IntegerDivisionByZeroException	É lançado quando o número é dividido por zero.
IOException	É a classe base das exceções relacionadas a entradas e saídas.
IsolateSpawnException	Ele é lançado quando um isolado não pode ser criado.
TimeoutException	É acionado quando ocorre um tempo limite programado durante a espera por um resultado assíncrono.

## Um outro exemplo de captura



```
try {
    breedMoreLlamas();

} on OutOfLlamasException {
    // A specific exception
    buyMoreLlamas();

} on Exception catch (e) {
    // Anything else that is an exception
    print('Unknown exception: $e');

} catch (e) {
    // No specified type, handles all
    print('Something really unknown: $e');
}
```



Executando um código mesmo após a geração de uma exceção

```
try {  
    rodarProcesso();  
} finally {  
    // mesmo que tenha ocorrido um erro, vai vir para cá  
    fecharPortas();  
}
```

## Exercício 15



No nosso primeiro exercício de exceções, vamos criar um mecanismo de conta corrente. No nosso método `main`, vamos guardar o saldo atual da conta em uma variável.

Sempre que quisermos efetuar um depósito ou um saque, utilizaremos uma função, que receberá o saldo atual e o valor que se deseja sacar ou depositar. Essas funções retornarão o novo saldo atual da conta.

Porém, não deverá ser possível sacar um valor maior que o saldo disponível, nem tentar sacar ou depositar um valor zerado ou negativo.

Nesses casos, uma exceção deve ser gerada.





Vamos criar uma lista de contatos. Todos os contatos serão armazenados dentro de uma lista de maps, guardando o nome e o telefone do contato cadastrado.

Os campos de nome e telefone são obrigatórios, caso algum deles não tenha sido preenchido, uma exceção deve ser disparada. Ex: O campo nome é obrigatório. O campo telefone é obrigatório.

Você deverá preparar a rotina principal, para que, caso ocorra alguma exceção, a mensagem de erro seja tratada e exibida de maneira amigável ao usuário.

Ex:

Erro ao cadastrar um novo contato: O campo nome é obrigatório.





Em uma lista de CEPs, vamos criar um método que busca um CEP na lista e o remove. Caso o CEP não exista na lista, gere uma exceção.

Porém, o usuário quer que, após executar a rotina de exclusão, a lista de CEPs seja exibida no console, mesmo que uma exceção ocorra.





Além de utilizar as exceções já existentes no dart, é possível criar as nossa próprias exceções.

Abaixo um exemplo:

```
class MinhaException implements Exception {  
  String mensagem;  
  MinhaException(this.mensagem);  
  
  @override  
  String toString() {  
    return "Erro especial: " + this.mensagem;  
  }  
}
```





Para testarmos a criação de exceções personalizadas, vamos criar uma exceção que é gerada sempre que um número par é gerado randomicamente.

Após criar o tipo de exceção, no método main, gere um valor aleatório, caso esse valor seja par.

Realize também o tratamento da exceção, para que caso a exceção gerada seja do tipo da exceção personalizada, a exceção não seja disparada para o usuário, somente a mensagem seja exibida no console.

