

PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

PROFº DOUGLAS ROBERTO ROSA PEREIRA



LINGUAGEM DART

- A linguagem Dart é a linguagem de programação que a tecnologia Flutter utiliza.
- Site da linguagem: <https://dart.dev/>

DARTPAD

- O DartPad é um site que te permite testar códigos Dart sem ter que instalar nenhum aplicativo em seu computador.
- Para usar o DartPad acesse: <https://dartpad.dev/>

PRINCIPAIS CARACTERÍSTICAS DA LINGUAGEM DART

- Fortemente tipada;
- Orientada a objetos;
- Possui sintaxe baseada na linguagem C;
- Multiplataforma, entre outros.

CARACTERÍSTICAS DA LINGUAGEM DART

- A linguagem é *case sensitive*.
- Linhas terminam com ;
- Comentários são feitos com // (única linha) ou /* comentário */
- Atribuições são feitas com =
- Textos vão entre aspas duplas (") ou aspas simples (')
- Separador de casas decimais é o ponto. Ex: 9.99
- Blocos de código são delimitados por chaves { }

TIPOS DE VARIÁVEIS

- **int**: números inteiros
- **String**: textos
- **double**: números com ponto flutuante
- **bool**: valores booleanos **true** ou **false**
- **dynamic**: tipo especial que permite armazenar qualquer coisa

OPERADORES ARITMÉTICOS

- Usados para executar cálculos de valores:

Operação	Símbolo
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Divisão que retorna apenas a parte inteira	~/
Resto da divisão inteira (mod)	%

OPERADORES RELACIONAIS

- Usados para comparações:

Operação	Símbolo
Maior	>
Menor	<
Maior ou igual	>=
Menor ou igual	<=
Igual	==
Diferente	!=

OPERADORES LÓGICOS

- Usados com a lógico booleana:

Operação	Símbolo
E	&&
OU	
NÃO	!

ALGUNS COMANDOS BÁSICOS

- `print("texto")` : comando básico para imprimir em tela

COMANDOS BÁSICOS – IF

- Utiliza a mesma sintaxe da linguagem C

```
if (condição){  
    /*código se verdadeiro*/  
}  
else{  
    /*código se falso*/  
}
```

COMANDOS BÁSICOS – OPERADOR TERNÁRIO

- Segue a seguinte sintaxe:

Teste lógico ? Resultado se verdadeiro : Resultado se falso

- Em resumo temos uma condição a ser checada antes da interrogação, caso ela seja verdadeira será executado o código antes dos dois pontos, caso contrário o código após os dois pontos. Ex:

String situacao = nota >= 6 ? “Aprovado” : “Reprovado”

COMANDOS BÁSICOS – SWITCH CASE

```
switch(opção){  
    case valor_1:  
        /* código para valor 1*/  
        break;  
    case valor_2:  
        /* código para valor 2*/  
        break;  
    ...  
    default:  
        /* código para qualquer outro valor */  
}
```


COMANDOS BÁSICOS – WHILE

- O comando **while** tem a mesma sintaxe usada em C, Java....

```
while(condição){  
    /* código a ser repetido */  
}
```

COMANDOS BÁSICOS – DO WHILE

- A sintaxe do *loop* **do-while** é a seguinte:

```
do{  
    /* Código a ser repetido */  
} while (condição);
```

COMANDOS BÁSICOS – FOR

- A sintaxe do loop **for** é igual ao Java e várias outras linguagens:

```
for (início do iterador; condição de execução; incremento ou decremento){  
    /* Código a ser repetido */  
}
```

- O Dart permite declarar o valor do iterador dentro do **for**:

```
for (int i = 0; i < 10; i++){  
    print("Valor de i: $i");  
}
```

NOTA IMPORTANTE: INTERPOLAÇÃO

- Utilizamos o \$ antes da variável pois queremos que dentro daquela *String* seja exibido o valor de uma variável.

FUNÇÕES

- A sintaxe de uma função é:

```
tipo_de_retorno nome_da_função (tipo_de_dado parâmetros_se_existirem){  
    /* código da função */  
}
```

- Exemplos:

```
int soma (int num1, int num2){  
    return num1+num2;  
}
```


FUNÇÕES

```
void saudacoes (){  
    print("Olá, seja bem vindo!");  
}
```

```
String agora(){  
    DateTime agora = DateTime.now();  
    return agora.toString();  
}
```

RETORNO DE UMA FUNÇÃO

- Pode ser que você deseje usar no ***print***, que por si só já é uma função, o retorno de outra função.
- Veja como isso fica usando a função `agora()` criada anteriormente (slide anterior):

```
print("Data e hora atuais: ${agora()}");
```

- Note que além do `$` já visto anteriormente, quando temos um retorno de uma função sendo passado dentro de uma ***string***, é necessário o uso de chaves `{ }`

PARÂMETROS DE UMA FUNÇÃO

- Conforme demonstrado anteriormente os parâmetros de uma função são passados dentro dos parênteses.
- O uso de um parâmetros que foi recebido na função, para ser utilizado dentro da String requer que ele seja acompanhado por um \$ por conta da interpolação. Veja o exemplo:

```
void mensagem(String nome){  
    print("Olá $nome");  
}
```

PARÂMETROS OPCIONAIS DE FUNÇÕES

- É possível especificar alguns parâmetros de uma função como opcionais. Nesse caso é preciso especificar um valor padrão para caso ele não seja informado.
- Parâmetros opcionais devem vir dentro de colchetes.
- Veja o exemplo:

```
void mensagem(String nome, [String saudacao = "Olá" ]){  
    print("$saudacao $nome");  
}
```

PARÂMETROS OPCIONAIS DE FUNÇÕES

- Veja algumas saídas possíveis:

```
1 void main() {  
2     mensagem("Douglas");  
3     mensagem("Douglas", "Boa noite");  
4 }  
5 void mensagem(String nome, [String saudacao]) {  
6     print("$saudacao $nome");  
7 }  
8  
9  
10
```

Run

Console

Olá Douglas
Boa noite Douglas

PARÂMETROS OPCIONAIS DE FUNÇÕES

- Estes parâmetros que acabamos de ver são chamados de parâmetros opcionais posicionais, pois a sua posição é a única forma da função saber de qual parâmetro estamos nos referindo. Quando temos dois parâmetros opcionais deste tipo a função fica assim:

```
void mensagem(String nome, [String saudacao = "Olá" , String separador = "-"]){  
    print("$saudacao $nome");  
    print(separador * 20);  
}
```

PARÂMETROS OPCIONAIS POSICIONAIS

- Veja agora como ficam algumas saídas:

```
1 void main() {
2     mensagem("Douglas");
3     mensagem("Douglas", "Boa noite");
4     mensagem("Fatec", "Bom dia", "*");
5     mensagem("Fatec", "*");
6 }
7 void mensagem(String nome, [String saudacao = "Olá", String separador = "-" ]){
8     print("$saudacao $nome");
9     print(separador * 20);
10
11 }
12
13
```

Run

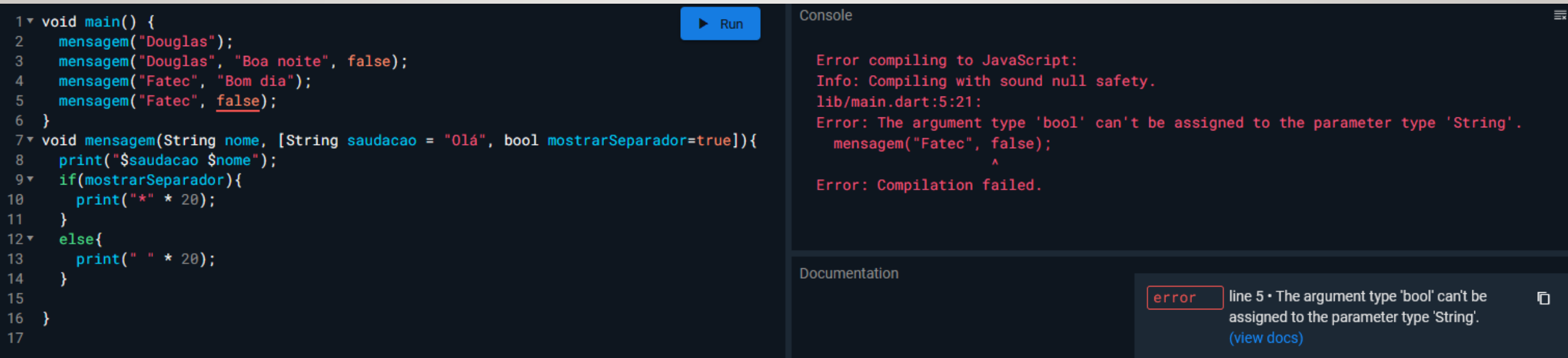
Console

```
Olá Douglas
-----
Boa noite Douglas
-----
Bom dia Fatec
*****
* Fatec
-----
```

- Note que no quarto uso da função, a linguagem não entende que eu queria o valor padrão para a saudação (Olá) e o asterisco para o separador.

PARÂMETROS OPCIONAIS POSICIONAIS

- Dependendo de como forem os parâmetros, pode-se até gerar um erro de compilação:



The screenshot displays an IDE interface with a code editor on the left and a console on the right. The code editor contains the following Dart code:

```
1 void main() {  
2   mensagem("Douglas");  
3   mensagem("Douglas", "Boa noite", false);  
4   mensagem("Fatec", "Bom dia");  
5   mensagem("Fatec", false);  
6 }  
7 void mensagem(String nome, [String saudacao = "Olá", bool mostrarSeparador=true]){  
8   print("$saudacao $nome");  
9   if(mostrarSeparador){  
10    print("*" * 20);  
11  }  
12  else{  
13    print(" " * 20);  
14  }  
15 }  
16 }  
17
```

The console on the right shows the following error message:

```
Error compiling to JavaScript:  
Info: Compiling with sound null safety.  
lib/main.dart:5:21:  
Error: The argument type 'bool' can't be assigned to the parameter type 'String'.  
    mensagem("Fatec", false);  
                      ^  
Error: Compilation failed.
```

Below the console, the documentation panel shows a detailed error message:

```
error line 5 • The argument type 'bool' can't be  
assigned to the parameter type 'String'.  
(view docs)
```

- Ainda na quarta função, como o segundo parâmetro passado foi do tipo *bool* e a função esperava uma *String*, foi gerado um erro.

PARÂMETROS NOMEADOS

- Para poder passar parâmetros em qualquer ordem podemos fazer uso dos parâmetros nomeados. Veja como fica o código anterior usando parâmetros nomeados:

```
1 void main() {
2     mensagem("Douglas");
3     mensagem("Douglas", saudacao: "Boa noite", mostrarSeparador: false);
4     mensagem("Fatec", mostrarSeparador: false);
5     mensagem("Fatec", saudacao: "Bom dia");
6
7 }
8
9 void mensagem(String nome, {String saudacao = "Olá", bool mostrarSeparador = true}) {
10    print("$saudacao $nome");
11    if (mostrarSeparador) {
12        print("*" * 20);
13    } else {
14        print(" " * 20);
15    }
16 }
17
```

Run

Console

```
Olá Douglas
*****
Boa noite Douglas

Olá Fatec

Bom dia Fatec
*****
```


PARÂMETROS NOMEADOS

- Agora ao invés de colchetes, usamos chaves para os parâmetros.
- Na hora de chamar a função, passamos os nomes do parâmetro, seguido de dois pontos e seu valor:

```
void main() {  
    mensagem("Douglas");  
    mensagem("Douglas", saudacao: "Boa noite", mostrarSeparador: false);  
    mensagem("Fatec", mostrarSeparador: false);  
    mensagem("Fatec", saudacao: "Bom dia");  
}
```


VARIÁVEIS *NULLABLE*

- A linguagem Dart tem o recurso *null safety*, ou seja ela impede que algum valor seja *null* a não ser que isso seja explicitamente desejado.
- Observe que no caso a seguir, normalmente a linguagem não deixa um parâmetro opcional ser *null*:

```
1 void main() {  
2   mensagem("Fatec");  
3   mensagem("Fatec", saudacao: "Boa noite");  
4   mensagem("Fatec", saudacao: "Bom dia");  
5  
6 }  
7  
8 void mensagem(String nome, {String saudacao = "Olá", String cliente}) {  
9   print("Boas vindas do (a) $nome");  
10  print("$saudacao $cliente");  
11  
12 }  
13
```

Run

Console

```
Error compiling to JavaScript:  
Info: Compiling with sound null safety.  
lib/main.dart:8:60:  
Error: The parameter 'cliente' can't have a value of 'null' because of its type 'String'  
void mensagem(String nome, {String saudacao = "Olá", String cliente}) {  
                        ^^^^^^^  
  
Error: Compilation failed.
```

Documentation

error

line 8 • The parameter 'cliente' can't have a value of 'null' because of its type, but the implicit default value is 'null'. [\(view docs\)](#)
Try adding either an explicit non-'null' default value or the 'required' modifier.

VARIÁVEIS *NULLABLE*

- Para que isso seja possível basta adicionar um ponto de interrogação após o tipo da variável:

```
1 void main() {  
2     mensagem("Fatec");  
3     mensagem("Fatec", saudacao: "Boa noite");  
4     mensagem("Fatec", cliente: "Jose");  
5  
6 }  
7  
8 void mensagem(String nome, {String saudacao = "Olá", String? cliente}) {  
9     print("Boas vindas do (a) $nome");  
10    print("$saudacao $cliente");  
11  
12 }
```

▶ Run

Console

```
Boas vindas do (a) Fatec  
Olá null  
Boas vindas do (a) Fatec  
Boa noite null  
Boas vindas do (a) Fatec  
Olá Jose
```

VARIÁVEIS *NULLABLE*

- A linguagem Dart impede (gerando um erro) que você tente fazer alguma operação com uma variável com valor *null*.
- Exemplos de códigos que geram erro:

```
int numero;
```

```
numero++;
```

- Ou

```
int? numero;
```

```
numero++;
```

TRANSFORMANDO UMA VARIÁVEL *NULLABLE* EM *NON-NULLABLE*

- Veja este exemplo onde tentamos colocar os textos em caixa alta:
- Um erro ocorre pois tentamos aplicar o método `.toUpperCase()` para a variável `cliente`, que pode ser *null*.

```
1 void main() {  
2   mensagem("Fatec");  
3   mensagem("Fatec", saudacao: "Boa noite");  
4   mensagem("Fatec", cliente: "Jose");  
5 }  
6  
7 void mensagem(String nome, {String saudacao = "Olá", String? cliente}) {  
8   print("Boas vindas do (a) ${nome.toUpperCase()}");  
9   print("$saudacao ${cliente.toUpperCase()}");  
10 }  
Console  
  
Error compiling to JavaScript:  
Info: Compiling with sound null safety.  
lib/main.dart:9:30:  
Error: Method 'toUpperCase' cannot be called on 'String?' because it is potentially null.  
    print("$saudacao ${cliente.toUpperCase()}");  
                          ^^^^^^^^^^^^^  
  
Error: Compilation failed.
```


TRANSFORMANDO UMA VARIÁVEL *NULLABLE* EM *NON-NULLABLE*

- Uma das formas de contornar isso é com um simples if:

```
1 void main() {  
2     mensagem("Fatec");  
3     mensagem("Fatec", saudacao: "Boa noite");  
4     mensagem("Fatec", cliente: "Jose");  
5 }  
6  
7 void mensagem(String nome, {String saudacao = "Olá", String? cliente}) {  
8     print("Boas vindas do(a) ${nome.toUpperCase()}");  
9     if (cliente != null) {  
10        print("$saudacao ${cliente.toUpperCase()}");  
11    }  
12 }  
13
```

▶ Run

Console

```
Boas vindas do(a) FATEC  
Boas vindas do(a) FATEC  
Boas vindas do(a) FATEC  
Olá JOSE
```

- Nesse caso, quando o nome do cliente for **null**, não é exibida a frase que poderia gerar um problema.

TRANSFORMANDO UMA VARIÁVEL *NULLABLE* EM *NON-NULLABLE*

- Outra forma de contornar o problema é utilizar o operador chamado *null aware*. Veja como fica:

```
1 void main() {  
2     mensagem("Fatec");  
3     mensagem("Fatec", saudacao: "Boa noite");  
4     mensagem("Fatec", cliente: "Jose");  
5 }  
6  
7 void mensagem(String nome, {String saudacao = "Olá", String? cliente}) {  
8     String c = cliente ?? "Nome desconhecido";  
9     print("Boas vindas do(a) ${nome.toUpperCase()}");  
10    print("${saudacao} ${c.toUpperCase()}");  
11 }
```

▶ Run

Console

```
Boas vindas do(a) FATEC  
Olá NOME DESCONHECIDO  
Boas vindas do(a) FATEC  
Boa noite NOME DESCONHECIDO  
Boas vindas do(a) FATEC  
Olá JOSE
```

- Atribuímos em uma nova variável **c** o conteúdo de cliente. Acrescentamos as duas interrogações (??) e colocamos um valor padrão para caso seja passado o valor *null*, o texto “Nome desconhecido” apareça em seu lugar.

PARÂMETROS NOMEADOS OBRIGATÓRIOS

- Os parâmetros nomeados que foram demonstrados até o momento eram facultativos, ou seja, ele não precisavam ser informados obrigatoriamente. Porém eles podem ser utilizados de forma obrigatório. Veja:

```
1 void main() {  
2     mensagem("Fatec", cliente: "Ana");  
3     mensagem("Fatec", cliente: "Luciana", saudacao: "Boa noite");  
4     mensagem("Fatec", cliente: "Jose");  
5 }  
6  
7 void mensagem(String nome, {String saudacao = "Olá", required String cliente}) {  
8     print("Boas vindas do(a) ${nome.toUpperCase()}");  
9     print("${saudacao} ${cliente.toUpperCase()}");  
10 }  
11
```

▶ Run

Console

```
Boas vindas do(a) FATEC  
Olá ANA  
Boas vindas do(a) FATEC  
Boa noite LUCIANA  
Boas vindas do(a) FATEC  
Olá JOSE
```

- A palavra reservada **required** obriga aquele parâmetro ser informado, caso contrário é gerado um erro.

PASSANDO FUNÇÕES COMO PARÂMETRO DE OUTRA FUNÇÃO

- É possível passar uma função como parâmetro em Dart:

```
1 void main() {  
2   mensagem("Fatec", cliente: "Ana");  
3   mensagem("Fatec", cliente: "Luciana", saudacao: boasVindas);  
4   mensagem("Fatec", cliente: "Jose");  
5 }  
6  
7 void mensagem(String nome, {Function? saudacao, required String cliente}) {  
8   print("\nBoas vindas da ${nome.toUpperCase()}");  
9   print(cliente);  
10  if (saudacao != null) {  
11    saudacao();  
12  }  
13 }  
14  
15 void boasVindas() {  
16   print("Seja bem vindo, sinta-se a vontade!");  
17 }
```

▶ Run

Console

Boas vindas da FATEC
Ana

Boas vindas da FATEC
Luciana
Seja bem vindo, sinta-se a vontade!

Boas vindas da FATEC
Jose

PASSANDO FUNÇÕES COMO PARÂMETRO DE OUTRA FUNÇÃO

- Veja outro exemplo agora passado uma função com parâmetros:

```
1 void main() {
2     mensagem("Fatec", cliente: "Ana");
3     mensagem("Fatec", cliente: "Luciana", saudacao: boasVindas);
4     mensagem("Fatec", cliente: "Jose");
5 }
6
7 void mensagem(String nome, {Function? saudacao, required String cliente}) {
8     print("\nBoas vindas da ${nome.toUpperCase()}");
9     print(cliente);
10    if (saudacao != null) {
11        saudacao("2023");
12    }
13 }
14
15 void boasVindas(String ano) {
16     print("Seja bem vindo aluno da turma de $ano");
17 }
18
19
```

Run

Console

Boas vindas da FATEC
Ana

Boas vindas da FATEC
Luciana
Seja bem vindo aluno da turma de 2023

Boas vindas da FATEC
Jose

Documentation

PASSANDO FUNÇÕES COMO PARÂMETRO DE OUTRA FUNÇÃO

- IMPORTANTE:
- Você jamais deve tentar passar uma função como parâmetro desta forma:

```
1 void main() {  
2     mensagem("Fatec", cliente: "Ana");  
3     mensagem("Fatec", cliente: "Luciana", saudacao: boasVindas("2022"));  
4     mensagem("Fatec", cliente: "Jose");  
5 }  
6
```

- Nesse caso o que está sendo feito é que você está tentando pegar o retorno da função boas-vindas. Como ela é do tipo *void* está dando um erro. Mas se ela tivesse um retorno, teríamos um resultado bem diferente do esperado.

FUNÇÕES ANÔNIMAS

- Assim como em outras linguagens uma função anônima é uma função sem nome que não pode ser novamente chamada pois você não consegue referencia-la (Lembre-se que chamamos as funções pelo seu nome, logo se ela não tem nome é impossível chamá-la).
- Geralmente essas funções são usadas para tarefas curtas que não serão necessárias novamente.
- No exemplo a seguir simplesmente pegamos a função anterior (saudação) e transformamos ela em anônima:

FUNÇÕES ANÔNIMAS

- Exemplo:

```
1 void main() {  
2     mensagem("Fatec", cliente: "Ana");  
3     mensagem("Fatec", cliente: "Luciana", saudacao: (String ano){  
4         print("Seja bem vindo aluno da turma de $ano");});  
5     mensagem("Fatec", cliente: "Jose");  
6 }  
7  
8 void mensagem(String nome, {Function? saudacao, required String cliente}) {  
9     print("\nBoas vindas da ${nome.toUpperCase()}");  
10    print(cliente);  
11    if (saudacao != null) {  
12        saudacao("2023");  
13    }  
14 }  
15
```

▶ Run

Console

Boas vindas da FATEC
Ana

Boas vindas da FATEC
Luciana

Seja bem vindo aluno da turma de 2023

Boas vindas da FATEC
Jose

RETORNO DE FUNÇÕES *NULLABLE*

- É possível especificar que uma função retorne ***null*** ao invés de seu tipo adicionando o ponto de interrogação logo após o tipo da função:

```
1 void main() {  
2     print(funcao(11));  
3     print(funcao(4));  
4 }  
5  
6 String? funcao(int x){  
7     if (x>10){  
8         return "Ola mundo";  
9     }  
10 }
```

Run

Console

Ola mundo
null

DECLARANDO VARIÁVEIS USANDO O VAR

- Utilizando a palavra reservada var é possível declarar uma variável sem especificar o seu tipo. Veja:

```
1 void main() {  
2     var nome = "Douglas";  
3     nome = "Roberto";  
4 }  
5
```

- Após ela receber um conteúdo, essa variável recebe um tipo e não pode mais ter seu tipo alterado.

```
1 void main() {  
2     var nome = "Douglas";  
3     nome = "Roberto";  
4     nome = 9;  
5 }  
6
```

error

line 4 • A value of type 'int' can't be assigned to a variable of type 'String'. ([view docs](#))

Try changing the type of the variable, or casting the right-hand type to 'String'.

VARIÁVEIS *DYNAMIC*

- O tipo *dynamic* é um tipo especial de dado que pode receber qualquer tipo de dado. Além disso o tipo de dado pode ser alterado a qualquer momento diferentemente do `var`:

```
1 void main() {  
2     dynamic variavel = "Douglas";  
3     variavel = 9;  
4     variavel = 2.75;  
5     variavel = 'c';  
6 }  
7
```

- Porém uma variável *dynamic* deve ser utilizada somente em situações muito específicas, pois não é uma boa prática de programação utiliza-la todo o momento.

VARIÁVEIS DO TIPO NUM

- O tipo **num** significa tipo **number**. Este tipo de variável pode receber tanto números inteiros quanto números com ponto flutuante.
- Em resumo, o Dart oferece três tipos para armazenar valores numéricos. O primeiro deles é **int**, utilizado para o armazenamento de qualquer número inteiro, seja ele negativo ou positivo. O segundo é **double**, que é utilizado para o armazenamento de números de pontos flutuantes. Ambos, **int** e **double**, são subtipos de num. Ao declarar uma variável como num ela pode ser tanto um inteiro quanto um número de ponto flutuante.
- Logo o tipo num é o terceiro tipo de variável numérica.

LENDO DADOS PELO TERMINAL

- Em um programa feito com o framework Flutter, não é comum a necessidade de entrada de dados em terminal.
- Nessa disciplina por exemplo, vamos trabalhar com dispositivos móveis, onde esse tipo de entrada de dados não é possível.
- Porém para a prática em aulas (exercícios) a entrada de dados no terminal facilita a e agiliza as necessárias resoluções.

LENDO DADOS PELO TERMINAL

- O primeiro passo é importar a biblioteca `dart.io`:

```
import 'dart:io';
```

- Agora podemos utilizar a função `stdin.readLineSync()` para a entrada de dados. Veja um exemplo:

```
print("Informe seu nome ");  
String? nome = stdin.readLineSync();  
print(nome);
```

LENDO DADOS PELO TERMINAL

- Tivemos que usar uma *String nullable* pois a função **stdin.readLineSync()** pode devolver um valor *null*.
- Outra forma de contornar isso é a seguinte:

```
💡 print("Informe seu nome ");  
String nome = stdin.readLineSync()!;  
print(nome);
```

- O operador **!** é usado para indicar que o resultado de **readLineSync()** não pode ser nulo (*null*). Isso é importante porque o método **readLineSync()** pode potencialmente retornar *null* se não houver mais entradas disponíveis.

LENDO DADOS PELO TERMINAL

- O método **`stdin.readLineSync()`** sempre nos retorna uma `String`, logo precisamos fazer as devidas conversões para as nossas variáveis.
- Alguns métodos de conversão:
 - ✓ `int.parse("conteúdo")`
 - ✓ `double.parse("conteúdo")`

EXERCÍCIOS

- 1 - Escreva uma função em Dart que receberá a duração de um evento expresso em segundos e exiba-o expresso em horas, minutos e segundos. Seu programa deverá exibir uma saída parecida com:

```
1 | Informe a duração do evento em segundos: 3712
2 | Duração do evento: 01:01:52
```

- 2 - Crie uma função que verifica se um número inteiro positivo passado como argumento é um número primo. A função deve retornar *true* se o número for primo e *false* caso contrário.

EXERCÍCIOS

- 3 - Crie uma função que calcula o preço final de um produto com desconto. A função deve receber o preço base e dois parâmetros opcionais posicionais: um desconto em valor fixo e um desconto percentual.
- 4 - Desenvolva uma função que gere senhas aleatórias. A função deve receber um parâmetro que determine o tamanho da senha gerada e um parâmetro opcional nomeado para permitir caracteres especiais.

EXERCÍCIOS

- 5 - Crie uma função que avalie as notas de um aluno. A função deve receber a nota do aluno como parâmetro e retornar uma mensagem como "Aprovado", "Reprovado" ou "Recuperação". Caso a nota seja nula, retorne "Nota não informada". Para ser aprovado a nota deve ser maior ou igual a 6.0, para ir para recuperação a nota deve ser maior ou igual a 4.0. Caso a nota seja inferior a 4.0 o aluno estará reprovado.

EXERCÍCIOS

- 6 - Um triângulo é uma forma geométrica (polígono) composta de três lados, sendo que cada lado é menor que a soma dos outros dois lados. Assim, para que um triângulo seja válido, é preciso que seus lados A, B e C obedecem à seguinte regra:
- $A < (B + C)$, $B < (A + C)$ e $C < (A + B)$.
- Escreva um programa Dart que leia os três lados de um triângulo e verifique se tais valores realmente formam um triângulo. Se o teste for satisfatório, informe se o triângulo é isósceles (dois lados iguais e um diferente), escaleno (todos os lados diferentes) ou equilátero (todos os lados iguais).
- Sua saída deverá ser parecida com:

```
1 | Informe o primeiro lado do triângulo: 30
2 | Informe o segundo lado do triângulo: 40
3 | Informe o terceiro lado do triângulo: 60
4 | O triângulo é escaleno
```


REFERÊNCIAS

- CIOLFI, Daniel; DUTRA, Ewerton; STARTTO.DEV. Curso de Criação de Apps Android/iOS/Web com Flutter - 5 cursos em 1. Disponível na plataforma Udemy.
- DEVMEDIA. **Sintaxe Dart:Tipos (não tão) primitivos**. 2019. Disponível em: <https://www.devmedia.com.br/sintaxe-dart-tipos-nao-tao-primitivos/40368>. Acesso em: 14 ago. 2023.