

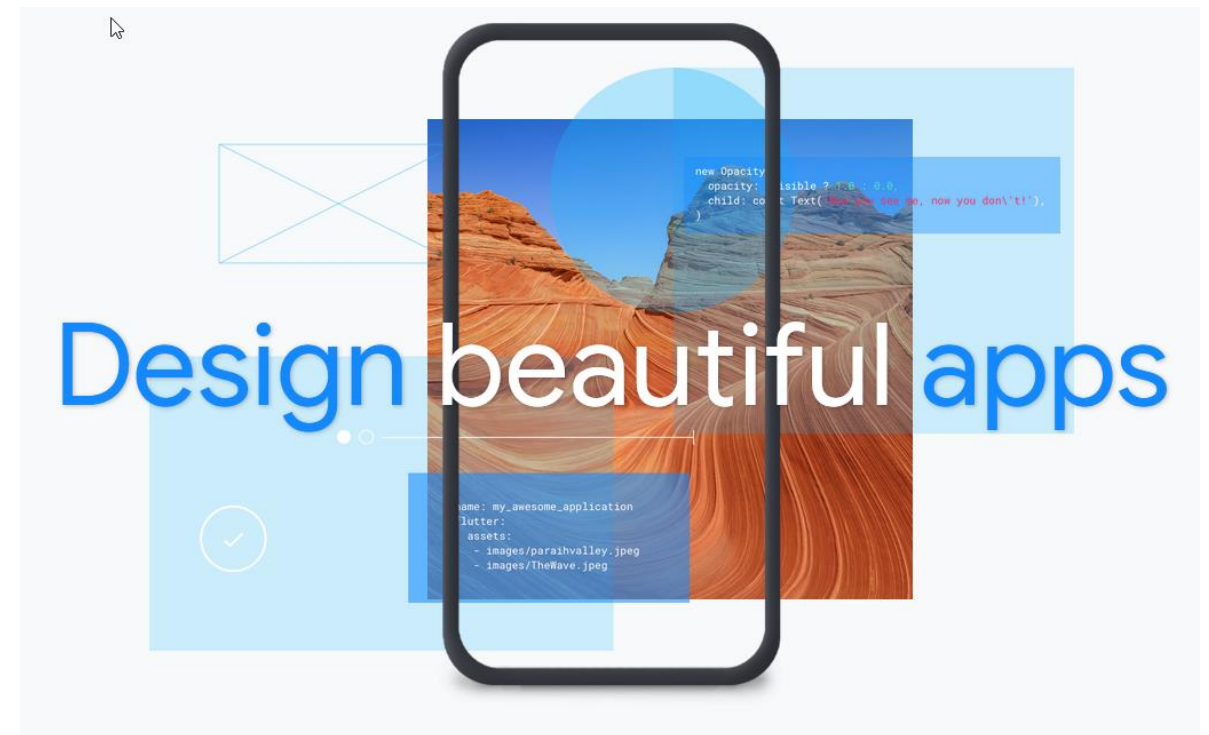


Transformando o futuro das pessoas
e as pessoas para o futuro.

#Senacfaz75



Desenvolvimento Mobile: Flutter



O que é o Flutter?

Flutter é um kit de desenvolvimento de interface de usuário (UI toolkit), de código aberto, criado pelo Google, que possibilita a criação de aplicativos compilados nativamente. Atualmente pode compilar para Android, iOS, Windows, Mac, Linux, Google *Fuchsia e Web.

O que é o Dart?

Os aplicativos Flutter são escritos na linguagem de programação Dart e fazem uso de muitos dos recursos mais avançados da linguagem.

No Windows, macOS e Linux, por meio do projeto semi-oficial Flutter Desktop Embedding, o Flutter é executado na máquina virtual Dart, que possui um mecanismo de compilação que ocorre em tempo de execução. Ao escrever e depurar um aplicativo, o Flutter usa a compilação JIT, permitindo o "hot reload", com a qual as modificações nos arquivos de origem podem ser injetadas em um aplicativo em execução. O Flutter estende isso com suporte para hot reload de widgets stateful, onde na maioria dos casos as alterações no código-fonte podem ser refletidas imediatamente no aplicativo em execução, sem a necessidade de uma reinicialização ou perda do Estado.

As versões de lançamento dos aplicativos Flutter são compiladas com a compilação antecipada (AOT) no Android e no iOS, possibilitando o alto desempenho do Flutter em dispositivos móveis.

*Fuchsia é um sistema operacional atualmente sendo desenvolvido pelo Google. Ao contrário de sistemas operacionais anteriores desenvolvidos pelo Google, como o Chrome OS e o Android, que são baseados no kernel Linux, Fuchsia é baseado em um novo microkernel chamado Zircon (o nome anterior era Magenta), derivado do Little Kernel, que foi destinado para sistemas embarcados e é principalmente escrito em C. Fuchsia foi projetado para ser executado em uma infinidade de dispositivos, incluindo telefones celulares e computadores pessoais.



Instalações:

Java SE

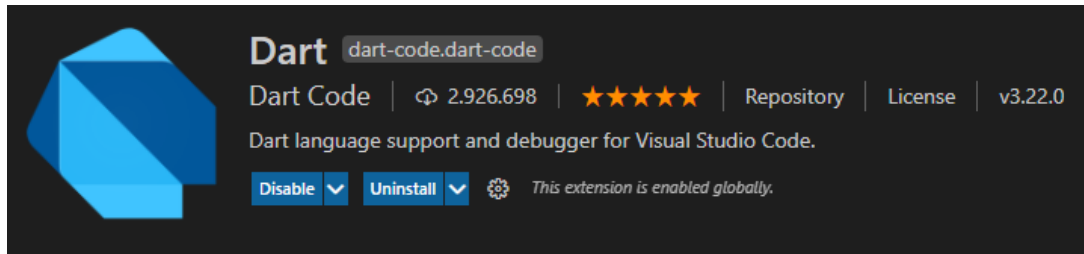
<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>

Flutter

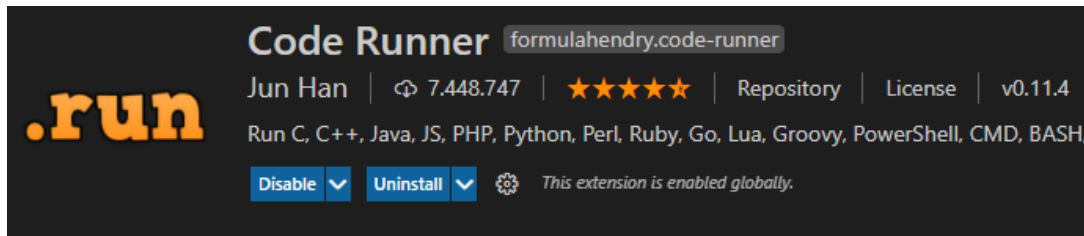
<https://flutter.dev/docs/get-started/install/windows>

OBS: Colocar a pasta bin do Flutter no Path do Windows

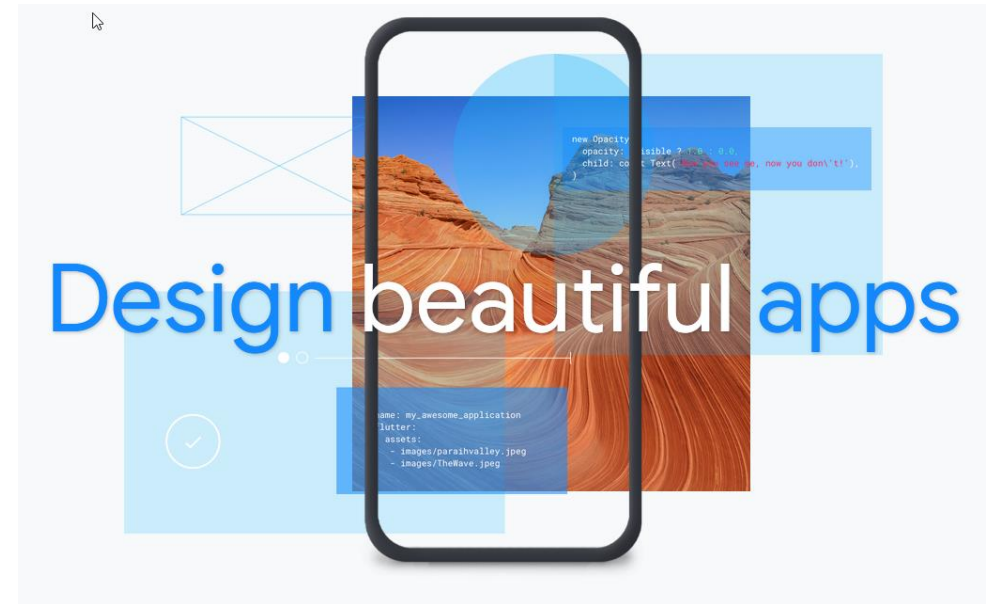
Extensões VS Code



Dart `dart-code.dart-code`
Dart Code | 2.926.698 | ★★★★★ | Repository | License | v3.22.0
Dart language support and debugger for Visual Studio Code.
Disable | Uninstall | ⚙️ This extension is enabled globally.



Code Runner `formulahendry.code-runner`
Jun Han | 7.448.747 | ★★★★★ | Repository | License | v0.11.4
Run C, C++, Java, JS, PHP, Python, Perl, Ruby, Go, Lua, Groovy, PowerShell, CMD, BASH.
Disable | Uninstall | ⚙️ This extension is enabled globally.



Null Safety

A partir do Dart 2.12, temos a possibilidade de usar o Null Safety para facilitar nossas aplicações, dificultando erros de valores nulos. Aprenderemos a identificar erros de valores nulos, aplicar no nosso projeto do ByteBank, atualizar o Dart para a nova versão e migrar um projeto inteiro! Assim podemos criar códigos complexos com mais facilidade!

Erros de valor Null

Existem infinitos erros que podem acontecer por conta de um valor nulo:

- Você pode estar esperando um dado do seu back-end, mas ele não existe ainda...
- Você pode criar uma lista que muda de tamanho de acordo com a quantidade de produtos...

Fonte: Kako(Caio couto Moreira). Alura. Disponível em: <https://www.alura.com.br/artigos/flutter-null-safety>



Aula 1 – Hello dart e tipos de variáveis!

```
hellodart.dart > ...
1 //É preciso criar uma classe main para rodar o código Dart
  Run | Debug
2 main() {
3   //print é o comando de saída em Dart
4   print('Hello dart!');
5   print('Como vai?');
6
7   //Tipos de variáveis em Dart
8   //Dart faz inferência de tipo (Genérico)
9   var nome = 'John Doe';
10  //nome é do tipo String
11  //Se eu tentar colocar outro valo nessa variável que não seja String
12  //vai dar erro
13  //var nome = 10; //Tipo já foi definido como String (ERRO)
14
15  /**
16   * Podemos também definir uma variável
17   * já com o seu tipo
18   */
19
20  //Variável do tipo String
21  String novoNome = 'Jane Doe'; //Estou tipando minha variável
22  String CPF = '999.999.999-99';
23
24  //Definindo um tipo numérico inteiro
25  int ano = 1970;
26
```

```
27 //Definindo um número decimal
28 double altura = 1.77;
29
30 //Definindo um tipo booleano (Verdadeiro ou falso)
31 bool vf = true;
32 bool fv = false;
33
34 //Saída Interpolada, ao contrário do javascript
35 //Não é preciso as {} e `` para interpolar valores
36 print('Seu nome: $novoNome'); //Lembra PHP
37 print('Nascimento: $ano');
38 print('Sua altura: $altura');
39 print('Resultado VF: $vf');
40 print('Resultado FV: $fv');
41 }
```

```
Hello dart!
Como vai?
Seu nome: Jane Doe
Nascimento: 1970
Sua altura: 1.77
Resultado VF: true
Resultado FV: false
```

```
[Done] exited with code=0 in 1.359 seconds
```


Aula 1 – Estruturas de Dados Arrays (arrays.dart)

```
Run | Debug
1 void main() {
2     //Arrays são estruturas que armazenam mais de um valor
3     //a uma variável
4
5     //Criando um array
6     var compras = ['Macarrão', 'Feijão', 'Pão', 'Manteiga'];
7
8     //Exibindo o array
9     print('Lista de compras: $compras');
10
11    //Exibindo os itens pelo índice
12    print('Nome do primeiro produto: ${compras[0]}');
13    print('Nome do segundo produto: ${compras[1]}');
14    print('Nome do terceiro produto: ${compras[3]}');
15
16    //Item out of range
17    //print('Nome do primeiro produto: ${compras[5]}');
18
19    //Acessando e alterando um valor do array
20    compras[0] = 'Arroz';
21    //Exibindo os itens pelo índice
22    print('Nome do primeiro produto: ${compras[0]}');
23
24    //Criando array numérico
25    var pares = [0, 2, 4, 6];
26
27    //Exibindo o array
28    print('Lista de números pares $pares');
29    print('-----');
30
31    //Métodos utilizados em arrays
32    /**
33     * first() : Retorna o primeiro elemento do Array
34     * last() : Retorna o último elemento do Array
35     * isEmpty() : Retorna true se a lista está vazia, caso contrário, false.
36     * length() : Retorna o tamanho do Array
37     */
38
39    var listaNomes = ['José Maria', 'Pedro da Silva', 'Cristina Pereira'];
40
41    print('Primeiro nome: ${listaNomes.first}');
42    print('Último nome: ${listaNomes.last}');
43    print(
44        | 'O Array está vazio? ${listaNomes.isEmpty}');
45    print('Tamanho do Array: ${listaNomes.length}');
46 }
```

```
Lista de compras: [Macarrão, Feijão, Pão, Manteiga]
Nome do primeiro produto: Macarrão
Nome do segundo produto: Feijão
Nome do terceiro produto: Manteiga
Nome do primeiro produto: Arroz
Lista de números pares [0, 2, 4, 6]
```

```
-----
Primeiro nome: José Maria
Último nome: Cristina Pereira
O Array está vazio? false
Tamanho do Array: 3
Exited
```

Aula 1 – Operadores Aritméticos (oparitimeticos.dart)

```
Run | Debug
1 void main() {
2     //Operadores aritméticos
3     // + - * / %
4
5     int a = 20;
6     int b = 5;
7
8     //Operador de Soma
9     var soma = a + b;
10
11    //Operador de Subtração
12    var subt = a - b;
13
14    //Operador de Multiplicação/Produto
15    var produto = a * b;
16
17    //Operador de divisão
18    var divisao = a / b;
19
20    //operador resto da divisão
21    var restoDiv = a % b;
22
23    //Saída
24    print('Soma de $a + $b = $soma');
25    print('Subtração de $a - $b = $subt');
26    print('Produto de $a * $b = $produto');
27    print('Divisão de $a / $b = $divisao');
28    print('Resto da Divisão de $a % $b = $restoDiv');
29}
```

```
30 //Operador resumido
31 int num = 100;
32 num += 20;
33 num -= 10; //Pode ser * /
34
35 //Saída
36 print('Número: $num');
37
38 //Incremento e decremento
39 num++;
40 num--;
41
42 //Saída
43 print('Número: $num');
44 }
```

```
Soma de 20 + 5 = 25
Subtração de 20 - 5 = 15
Produto de 20 * 5 = 100
Divisão de 20 / 5 = 4.0
Resto da Divisão de 20 % 5 = 0
Número: 110
Número: 110

[Done] exited with code=0 in 1.345 seconds
```


Aula 1 – Operadores Relacionais (oprelaconais.dart)

```
Run | Debug
1 void main() {
2     //Operdaores Relacionais
3     /**
4      * ==, != (Igual e diferente)
5      * >, < (Maior e menor)
6      * >=, <= (Maio igual e Menor igual)
7      */
8
9     //Declarando variáveis
10    int a = 20;
11    int b = 5;
12
13    //Verificando as variáveis
14    print('$a = $b? Resultado: ${a == b}');
15    print('$a ≠ $b? Resultado: ${a != b}');
16    print('$a > $b? Resultado: ${a > b}');
17    print('$a < $b? Resultado: ${a < b}');
18    print('$a ≥ $b? Resultado: ${a >= b}');
19    print('$a ≤ $b? Resultado: ${a <= b}');
20
21    //Podemos atribuir esses resultados a uma variável
22    bool igual = a == b;
23    print('Verificação de igualdade: $igual');
24 }
25
```

```
20 = 5? Resultado: false
20 ≠ 5? Resultado: true
20 > 5? Resultado: true
20 < 5? Resultado: false
20 ≥ 5? Resultado: true
20 ≤ 5? Resultado: false
Verificação de igualdade: false
```

```
[Done] exited with code=0 in 1.508 seconds
```

Aula 1 – Operadores Lógicos (oplogicos.dart)

```
Run | Debug
1 void main() {
2     //Operadores Lógicos
3     /**
4      * && (E) --> V && V = V
5      * || (OU) --> F || F = F
6      * ! Negação
7      */
8
9     //Declarando as variáveis
10    int a = 20;
11    int b = 5;
12    int c = 7;
13
14    //Verificando Verdadeiro
15    bool proposicao1 = a > b;
16    bool proposicao2 = b < c;
17
18    //Saída
19    //      V      V
20    print('$a > $b && $b < $c - Resposta: ${proposicao1 && proposicao2}');
21    //      V      F
22    print('$a > $b || $b > $c - Resposta: ${proposicao1 || proposicao2}');
23 }
```

```
24 //Verificando o Falso
25 bool proposicao3 = a < b;
26 bool proposicao4 = b > c;
27
28 //Saída
29 print('$a < $b && $b > $c - Resposta: ${proposicao3 && proposicao3}');
30 print('$a > $b || $b > $c - Resposta: ${proposicao4 || proposicao4}');
31
32 //Negar um valor
33 bool v = true;
34 bool f = false;
35
36 //Saída
37 print('Negando o V: ${!v}');
38 print('Negando o F: ${!f}');
39 }
```

```
20 > 5 && 5 < 7 - Resposta: true
20 > 5 || 5 > 7 - Resposta: true
20 < 5 && 5 > 7 - Resposta: false
20 > 5 || 5 > 7 - Resposta: false
Negando o V: false
Negando o F: true
```

```
[Done] exited with code=0 in 1.404 seconds
```

Aula 1 – Condicional If Else (condicionais.dart)

```
Run | Debug
1 void main() {
2     //Condicionais
3     //Declaração de variáveis
4     int a = 10;
5     int b = 5;
6
7     //Condicional simples, Else opcional
8     if (a > b) {
9         print('Informação Verdadeira!');
10    } else {
11        print('Informação Falsa!');
12    }
13
14    //Declarando Variável
15    String nome = 'john';
16
17    //Condicional
18    if (nome != 'Jane') {
19        print('Os nomes são diferentes!');
20    } else {
21        print('Os nomes são iguais!');
22    }
23
24    //Condicionais encadeadas
25    //Testando mais de uma condição
26    //Declarando variável
27    double media = 4;
28
29    //Condicional
30    if (media >= 7.5 && media <= 10) {
31        print('O aluno passou de ano!');
32    } else if (media >= 5 && media < 7.5) {
33        print('Aluno em recuperação!');
34    } else {
35        print('Aluno Reprovado!');
36    }
37
38 }
```

```
Informação Verdadeira!
Os nomes são diferentes!
Aluno Reprovado!
Exited
```

Aula 1 – Condicional Switch (condswitch.dart)

```
Run | Debug
1 void main() {
2   print('1 - Numero par');
3   print('2 - Maior número');
4   print('3 - Sair');
5
6   int opcao = 2;
7
8   switch (opcao) {
9     case 1:
10      //Declaração
11      int numero = 10;
12
13      //Condicional
14      if (numero % 2 == 0) {
15        print('O número $numero é par!');
16      } else {
17        print('O número $numero é ímpar!');
18      }
19      break;
20
21     case 2:
22      //Declaração
23      int a = 3;
24      int b = 3;
25
26      //Condicional
27      if (a > b) {
28        print('O número $a é maior que o número $b!');
29      } else if (a < b) {
30        print('O número $a é menor que o número $b!');
31      } else {
32        print('Os números são iguais!');
33      }
34      break;
35
36     default:
37       print('Fora do intervalo de opções!');
38   }
39 }
```

```
1 - Numero par
2 - Maior número
3 - Sair
Os números são iguais!
Exited
```

Aula 1 – Loop While (while.dart)

```
1 //Loop While
  Run | Debug
2 /**
3  * O loop while executa as instruções cada vez que a
4  * condição especificada é avaliada como verdadeira. Em outras palavras,
5  * o loop avalia a condição antes que o bloco de código seja executado
6  */
7 void main() {
8     //Declarando uma Flag
9     int contador = 0;
10
11     while (contador <= 20) {
12         print('Número: $contador');
13
14         //Incrementando o contaador para o loop não ficar infinito
15         contador++;
16
17         //condicional para quebra o Loop
18         //Podemos usar também o continue
19         if (contador == 10) {
20             print('Loop interrompido!');
21             break;
22         }
23     }
24 }
```

```
Número: 0
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
Número: 6
Número: 7
Número: 8
Número: 9
Loop interrompido!
Exited
```

Aula 1 – Loop For (for.dart)

```
1 //Loop For
  Run | Debug
2 /**
3  * O laço For é uma implementação de um loop definido.
4  * O loop for executa o bloco de código por um determinado número de vezes.
5  * Ele pode ser usado para iterar sobre um conjunto fixo de valores,
6  * como uma matriz.
7  */
8
9 void main() {
10 //Declaração
11 int contador = 20;
12
13 for (var i = 0; i < contador; i++) {
14   print('Número: $i');
15   if (i == 10) {
16     print('Loop interrompido!');
17     break;
18     //Podemos usar o continue também
19   }
20 }
21 print('-----');
22
23 //Varrendo array
24 //Definindo o Array
25 var nomes = ['Bete', 'Ana', 'Pedro', 'João', 'Maria'];
26
27 for (var i = 0; i < nomes.length; i++) {
28   print('Nome: ${nomes[i]}');
29 }
30 print('-----');
31
32 //Simulando Post de Filmes
33 var filmes = [
34   'Matrix',
35   'Uma vida iluminada',
36   'Teoria de Tudo',
37   'Divertidamente'
38 ];
39
40 for (var filme in filmes) {
41   print(filme);
42 }
43 }
```

```
Número: 0
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
Número: 6
Número: 7
Número: 8
Número: 9
Número: 10
Loop interrompido!
```

```
-----
Nome: Bete
Nome: Ana
Nome: Pedro
Nome: João
Nome: Maria
```

```
-----
Matrix
Uma vida iluminada
Teoria de Tudo
Divertidamente
```

```
[Done] exited with code=0 in 2.107 seconds
```


Aula 2 – Funções (funcao_void.dart)

```
1 //Main() é uma função de inicialização de código dart
2 //Essa função é obrigatória
3 //Essa função é void, não retorna valor
4
5 //Exemplos
6 void mostrarNome() {
7     print('John Doe');
8 }
9
10 void linha() {
11     print('-----');
12 }
13
14 //Função com parâmetro
15 void calcularValores(String operador, double a, double b) {
16     if (operador == '+') {
17         double soma = a + b;
18         print('Operador \"$operador\" a soma de $a + $b = $soma');
19     } else if (operador == '*') {
20         double produto = a * b;
21         print('Operador \"$operador\" o produto de $a x $b = $produto');
22     } else if (operador == '-') {
23         double subt = a - b;
24         print('Operador \"$operador\" a subtração de $a - $b = $subt');
25     } else if (operador == '/') {
26         double divisao = a / b;
27         if (b == 0) {
28             print('Divisão inválida!');
29         } else {
30             print('Operador \"$operador\" o produto de $a / $b = $divisao');
31         }
32     }
33 }
34
```

```
35 void main() {
36     //Chamando a função
37     mostrarNome();
38     linha();
39
40     calcularValores('+', 10, 5);
41     linha();
42
43     calcularValores('-', 3, 5);
44     linha();
45
46     calcularValores('*', 30, 2);
47     linha();
48
49     calcularValores('/', 5, 10);
50     linha();
51 }
52
```

```
John Doe
-----
Operador "+" a soma de 10.0 + 5.0 = 15.0
-----
Operador "-" a subtração de 3.0 - 5.0 = -2.0
-----
Operador "*" o produto de 30.0 x 2.0 = 60.0
-----
Operador "/" o produto de 5.0 / 10.0 = 0.5
-----

[Done] exited with code=0 in 1.336 seconds
```

Aula 2 – Funções (funcao_retorno.dart)

```
1 //Funções com retorno
2
3 //Declaração
4 //Foi importado assim que eu chamei a função linha()
5 import 'funcao_void.dart';
6
7 double calcularMedia(double n1, double n2, double n3, double n4) {
8     double media = (n1 + n2 + n3 + n4) / 4;
9     return media;
10 }
11
12 double calcularDivisao(double n1, double n2) {
13     double div = (n1 / n2);
14     return div;
15 }
16
17 //Função Resumida
18 double calcularPorcentagem(percent, valor) => (percent * valor) / 100;
19
20 Run | Debug
21 void main() {
22     double media = calcularMedia(10, 10, 10, 10);
23     double divisao = calcularDivisao(10, 5);
24
25     //Saída
26     print('Média Aritmética');
27     print('A média é $media');
28     linha();
29     print('A divisão é $divisao');
30     linha();
31
32     //Declaração de variáveis
33     double percent = 5;
34     double valor = 100;
35
36     //Executa a função e guarda o valor em uma variável
37     double percentual = calcularPorcentagem(percent, valor);
38
39     //Imprime 10% de 100
40     print('$percent% de $valor = $percentual');
41 }
42 }
```

```
Média Aritmética
A média é 10.0
-----
A divisão é 2.0
-----
5.0% de 100.0 = 5.0
Exited
```

Aula 2 – Funções (funcao_param_opcional.dart)

```
1 //Parâmetro opcional
2
3 import 'dart:math';
4
5 void main() {
6   //declarar limite
7   int numero = sortearNumero(1000);
8   //Saída
9   print('O número sorteado foi: $numero');
10
11   ///////////////////////////////////////////////////
12   int valor1 = 10;
13   int valor2 = 20;
14
15   //Segundo parâmetro é opcional
16   int soma = somarValores(valor1, valor2);
17
18   //Saída
19   print('Resultado dos valores somados: $soma');
20 }
21
22 //Utilizamos os colchetes para determinar o parâmetro opcional
23 int sortearNumero([int limite = 3]) {
24   //Se nenhum valor limite for passado
25   //Será assumido o 3
26   return Random().nextInt(limite);
27 }
28
29 //1 parâmetro obrigatório e 1 opcional
30 int somarValores(int v1, [int v2 = 0]) {
31   //Se não informar o valor 2
32   //o parâmetro assume 0
33   print('Valor 1: $v1');
34   print('Valor 2: $v2');
35   return v1 + v2;
36 }
```

```
O número sorteado foi: 778
Valor 1: 10
Valor 2: 20
Resultado dos valores somados: 30
Exited
```

Aula 2 - Funções (funcao_param_nomeado_1.dart)

```
1 //Parâmetros nomeados em função
2 //ajuda na clareza do código
3 //Evitando confusão com os parâmetros posicionais
4
5 Run | Debug
6 void main(){
7   //chamando a função com parâmetros nomeados
8   //exibirCadastro(salario: 1500, funcionario: 'John Doe', funcao: 'Gerente');
9 }
10
11 //Funções
12 exibirCadastro(
13   //Utilizamos required por conta da nova funcionalidade do Dart
14   //Assunto complexo: non-null
15   //{required String funcao,
16   //  required double salario,
17   //  required String funcionario}){
18   //print('Nome do Funcionário: $funcionario');
19   //print('Função: $funcao');
20   //print('Salário: R$${salario}');
21   //return 'Tudo ok!';
22 }
```

```
Nome do Funcionário: John Doe
Função: Gerente
Salário: R$1500.0
Exited
```

Aula 2 - Funções (funcao_param_nomeada_2.dart)

```
1 //Função anônima e parâmetros opcionais
2 //Função com parâmetros opcionais
3 void avaliarFilme(String nomeFilme, {categoria, nota}) {
4   ..
5   ..//Verificar se o valor está nulo
6   ..var n = nota ?? 0;
7   ..var cat = categoria ?? 'Sem categoria';
8   ..
9   ..//Imprime os Dados
10  ..print('Nome do Filme: $nomeFilme');
11  ..print('Categoria: $cat');
12  ..print('Nota: $n');
13  ..
14 }
15
16 Run | Debug
17 void main() { //função obrigatória
18   ..
19   ..//Executar a função avaliarFilme()
20   ..//avaliarFilme('Matrix');
21   ..avaliarFilme('Matrix', categoria: 'Ficção', nota: 10);
22 }
```

```
Nome do Filme: Matrix
Categoria: Ficção
Nota: 10
Exited
```

Aula 2 – Funções (funcao_anonima.dart)

```
1 //Funções anônimas: Funções que podem ser
2 //armazenadas dentro de variáveis
3
4 import 'dart:math';
5
6 Run | Debug
7 void main() {
8   //Criando e definindo o tipo
9
10  //Minha função é do tipo int
11  int Function(int, int) soma = somarFunction;
12
13  //Minha função é do tipo int/double/String/Dynamic
14  String Function(String, String) mostraTexto = mostrarTexto;
15
16  //Uma outra forma de criar a função anônima
17  //Minha função é do tipo int/double/String/Dynamic
18  dynamic Function(double, double) potencia = (base, expoente) {
19    return pow(base, expoente);
20  }; //Não esquecer
```

```
21 //mais uma forma de criar esse tipo de função
22 var produto = (double a, int b) {
23   return a * b;
24 }; //Não esquecer
25
26 //Saída
27 print('A soma dos valores é ${soma(10, 10)}');
28 print('Frase: ${mostraTexto("Olá", "Mundo!")}');
29 print('Resultado da potência é: ${potencia(2, 3)}');
30 print('Resultado da multiplicação é: ${produto(10, 3)}');
31 }
32
33 //Funções
34 int somarFunction(int a, int b) {
35   return a + b;
36 }
37
38 String mostrarTexto(String a, String b) {
39   return a + b;
40 }
```

```
A soma dos valores é 20
Frase: Olá Mundo!
Resultado da potência é: 8.0
Resultado da multiplicação é: 30.0
Exited
```


Aula 2 – Funções (funcao_arrow.dart)

```
1 //Arrow-function é uma forma reduzida de representar
2 //um função. Contra: são limitadas, não podem ser utilizadas
3 //em todas as situações
4
5 Run | Debug
6 void main() {
7   //Função anônima
8   var produto = (int a, int b) {
9     return a * b;
10   };
11
12   //Arrow Function
13   //1 sentença de código apenas
14   var divisao = (double a, double b) => a / b;
15   var modulo = (double c, double d) => c % d;
16   var media = (double n1, double n2, double n3) => (n1 + n2 + n3) / 3;
17
18   //Saída
19   print('O valor do produto é: ${produto(10, 5)}');
20   print('O valor da divisão é: ${divisao(50, 2)}');
21   print('O valor da divisão é: ${modulo(5, 3)}');
22   print('A média calculada é: ${media(4, 5, 6)}');
23 }
```

```
O valor do produto é: 50
O valor da divisão é: 25.0
O valor da divisão é: 2.0
A média calculada é: 5.0
Exited
```

Aula 2 – Funções (funcao_tipo_dinamico.dart)

```
1 //Dart é altamente tipado
2 //Cuidado ao criar suas variáveis
  Run | Debug
3 void main() {
4
5   //Criando tipos dinâmicos
6   dynamic a = 1000;
7   dynamic b = 'Mundo!';
8
9   //Chamada da função e atribuição em uma variável
10  dynamic uniao = concatenar(a, b);
11
12  //Saída
13  print('A união dos valores \"$a\" e \"$b\" é: $uniao');
14 }
15
16 //Função para juntar os valores de a e b
17 dynamic concatenar(param1, param2) {
18
19   //método toString faz casting
20   print(param1.toString() + param2.toString());
21
22   //Retornando os valores
23   return param1.toString() + param2.toString();
24 }
```

1000 Mundo!

A união dos valores "1000" e " Mundo!" é: 1000 Mundo!

Exited

Aula 2 – Funções (funcao_param1.dart)

```
1 //As funções pode também receber como parâmetros outras funções
2
3 //Importando a biblioteca matemática
4 import 'dart:math';
5
6 Run | Debug
7 void main() {
8   //Calcular par ou ímpar
9   var par = () => print('Número Par');
10  var impar = () => print('Número Ímpar');
11
12  //Chamando a função calcularParImpar
13  calcularParImpar(par, impar);
14 }
15
16 //Funções
17 void calcularParImpar(Function calcularPar, Function calcularImpar) {
18   //Criando número randômico
19   //O VsCode importa da biblioteca dart:math automaticamente
20   var numero = Random().nextInt(50);
21   print('Número randômico: $numero');
22   //Verificando
23   if (numero % 2 == 0) {
24     calcularPar();
25   } else {
26     calcularImpar();
27   }
28 }
```

```
Número randômico: 1
Número Ímpar
Exited
```

Aula 2 – Funções (funcao_param2.dart)

```
1 //Importando a biblioteca matemática
2 import 'dart:math';
3
4 //Criando uma função para encontrar o Delta
5 double calcularDelta(double a, double b, double c) {
6   //Fórmula do Delta =  $b^2 - 4ac$ 
7   double delta = (b * b) - (4 * a * c);
8   return delta;
9 }
10
11 //Criando uma função para encontrar os valores de X1 e X2
12 //Passo os valores das incógnitas e a função do Delta
13 dynamic calcularEquacao(double a, double b, double c, Function calcularDelta) {
14   //Lista para a solução
15   var solucao = [];
16
17   //Calculando as raízes
18   var x1 = ((-b) + sqrt(calcularDelta(a, b, c))) / (2 * a);
19   var x2 = ((-b) - sqrt(calcularDelta(a, b, c))) / (2 * a);
20
21   //Inserindo os valores na lista
22   solucao.add(x1);
23   solucao.add(x2);
24   return solucao;
25 }
26
27 Run | Debug
28 void main() {
29   //Declarando as incógnitas
30   double a = 1;
31   double b = 2;
32   double c = -15;
33
34   //Chamando a função com parâmetros misturados
35   var equacao = calcularEquacao(a, b, c, calcularDelta);
36
37   //Saída
38   print(equacao);
39 }
```

```
[3.0, -5.0]
Exited
```

Aula 2 – Funções (funcao_ret_funcao.dart)

```
1 //Com uma função retornando outra função
2 //podemos fazer com que alguma parte do nosso código
3 //seja executada em um outro momento
4 double Function(double) calcularJuros(double a) {
5
6     //Função interna
7     return (double b) { //Retornando uma função anônima
8         return a * b;
9     };
10 }
11
12
13 Run | Debug
14 void main() {
15     //Passando os dois valores, a e b
16     print(calcularJuros(2)(20));
17
18     //Colocando a função em uma variável para otimizar o processo
19     //passando 1 valor
20     var juros = calcularJuros(.10);
21
22     //Saída passando o valor de b
23     print(juros(1500));
24     print(juros(1000));
25     print(juros(10000));
26 }
```

```
40.0
150.0
100.0
1000.0
Exited
```

Aula 2 – Funções (lista_2.dart)

```
1 void main() {  
2   //Criando uma lista  
3   var minhaLista = [];  
4  
5   //Método para inserir valor na lista  
6   minhaLista.add('Dart');  
7   minhaLista.add('Java');  
8   minhaLista.add('Python');  
9   minhaLista.add('Ruby');  
10  minhaLista.add('Php');  
11  minhaLista.add('JavaScript');  
12  
13  //Método para retornar o tamanho da lista  
14  int tamanhoLista = minhaLista.length;  
15  
16  //Método para remover elementos da lista  
17  minhaLista.remove('Php');  
18  
19  //Método para remover elementos da lista pelo índice  
20  minhaLista.removeAt(2);  
21  
22  //Método para remover o último elemento da lista  
23  minhaLista.removeLast();  
24  
25  //Método para remover os elementos pelo intervalo  
26  minhaLista.removeRange(1, minhaLista.length);  
27  
28  //Imprimindo a lista  
29  print('Minha lista: $minhaLista');  
30  print('Tamanho da lista: $tamanhoLista');  
31 }
```

Minha lista: [Dart]

Tamanho da lista: 6

Exited

Aula 2 – Funções (filtragem_1.dart)

```
1 //Exemplo de filtragem sem o uso do método dart
2 //Verrendo os elemento de uma lista e colocando em outra
Run | Debug
3 void main() {
4
5     //Criando a lista de valores
6     var valores = [5.5, 7.0, 6.3, 9.7, 10.0, 3.5];
7
8     //Criando uma lista para guardar as valores maiores que 7
9     var valoresAcimaMedia = [];
10
11     //for para varrer a lista e pegar as valores
12     for (var valor in valores) {
13         if (valor >= 7) {
14
15             //Método add() para guardar as valores em uma lista nova
16             valoresAcimaMedia.add(valor);
17         }
18     }
19
20     //Imprimindo da lista
21     print(valores);
22     print(valoresAcimaMedia);
23 }
```

[5.5, 7.0, 6.3, 9.7, 10.0, 3.5]

[7.0, 9.7, 10.0]

Exited

Aula 2 – Funções (filtragem_2.dart)

```
1 //Melhorando a filtragem de dados
2 //Agora vamos usar o método Dart para fazer a filtragem
Run | Debug
3 void main() {
4   //Criando a lista de valores
5   var valores = [5.5, 7.0, 6.3, 9.7, 10.0, 3.5];
6
7   //Criar uma função para pegar as valores
8   //Veja que o retorno de arrow function é um boolean
9   //É desta forma que o método where funciona
10  bool Function(double) valoresAcimaMedia = (double valor) => valor >= 6;
11
12  //Usando o Método Where
13  //De todas as valores pegue somente as acima da média
14  var valoresNovos = valores.where(valoresAcimaMedia);
15  print('Primeira lista: $valores');
16  print('Lista nova: $valoresNovos');
17 }
```

```
Primeiro lista: [5.5, 7.0, 6.3, 9.7, 10.0, 3.5]
Lista nova: (7.0, 6.3, 9.7, 10.0)
Exited
```

Aula 2 – Funções (filtragem_3.dart)

```
1 //Simulando o a implementação Where
2 //Para filtrarValores valores de forma genérica
3
4 //Criar uma função
5 //funcao pode ter qualquer nome
6 List<double> filtrarValores(List<double> lista, bool Function(double) funcao) {
7   List<double> listaFiltrada = [];
8
9   //Usar um laço para inserir os elementos na lista
10  //de acordo com a condição passada para função filtrarValores
11  for (double elemento in lista) {
12    if (funcao(elemento)) {
13      //Inserindo o valor na lista
14      listaFiltrada.add(elemento);
15    }
16  }
17  return listaFiltrada;
18 }
19
20
21 Run | Debug
22 void main() {
23   //Criar um array Double
24   var valores = [10.5, 9.4, 15.7, 6.3, 7.0];
25
26   //Criar uma função para montar a condição de varredura
27   //Só preenche com valores maiores iguais a 8.0
28   var valoresAcimaMedia = (double valor) => valor >= 8.0;
29
30   //Chama a função que vai filtrarValores nossos valores
31   var executaFiltragem = filtrarValores(valores, valoresAcimaMedia);
32
33   //Saída
34   print('Lista original: $valores');
35   print('Lista nova: $executaFiltragem');
```

```
Lista original: [10.5, 9.4, 15.7, 6.3, 7.0]
Lista nova: [10.5, 9.4, 15.7]
Exited
```

Aula 2 – Funções (funcao_map.dart)

```
1 //Map: uma lista mapeada sempre terá o mesmo número de elementos
2 //Ao contrário do where, onde podemos obter listas de diferentes tamanhos
3 //Map serve para mapear um elemento em outro elemento
4 //Exemplo: em uma lista de nomes podemos obter outra lista com
5 //o tamanho (em caracteres) de cada nome.
6 import 'funcao_void.dart';
7
8 Run | Debug
9 void main() {
10   //Criando um Map
11   var mediaAlunos = [
12     {'nome': 'John', 'nota': 8},
13     {'nome': 'Jane', 'nota': 7},
14     {'nome': 'Carol', 'nota': 8},
15     {'nome': 'Mike', 'nota': 6},
16   ];
17
18   //Uma função que pega os alunos pelo nome
19   //Map aqui é somente um tipo, assim como List
20   String Function(Map) capturaAluno = (aluno) => aluno['nome'];
21
22   //Método map() para mapear o resultado da função
23   //Retornar para um variável
24   var listaResultado = mediaAlunos.map(capturaAluno);
25
26   //Veja que o resultado tem o mesmo tamanho de nosso Map
27   print('Lista de Nomes: $listaResultado');
28   print('O Map tem tamanho ${mediaAlunos.length}');
29
30   //Outro exemplo no uso de Map
31   //Para mapear uma lista
32   List<int> numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
33
34   //Recuperar uma lista com o dobro de cada valor
35   var resultadoDobro = numeros.map((numero) => numero * 2);
36
37   linha();
38   print(resultadoDobro);
39 }
```

Lista de Nomes: (John, Jane, Carol, Mike)

O Map tem tamanho 4

(2, 4, 6, 8, 10, 12, 14, 16, 18, 20)

Exited

Aula 2 – Funções (funcao_reduce.dart)

```
1 import 'funcao_void.dart';
2
3 Run | Debug
4 void main() {
5     //Uma forma noraml para calcular a média
6     var notasAlunos = [10, 6, 8, 7];
7
8     //Preciso colocar 0.0 para a inferência funcionar
9     //Caso contrário, devo explicitar o tipo (double)
10    var totalNotas = 0.0;
11
12    //percorrendo a lista com for
13    //Pacada cada nota em notas de alunos, some as notas
14    for (var nota in notasAlunos) {
15        totalNotas += nota;
16    }
17
18    //Média
19    double media = totalNotas / 4;
20
21    //Saída
22    linha();
23    print('MÉTODO CONVENCINAL');
24    print('=====');
25    print('Valor total das notas: $totalNotas');
26    print('Média de notas: $media');
27    linha();
28    print('MÉTODO REDUCE()');
```

```
29 //////////////////////////////////////////////////
30 //MÉTODO UTILIZANDO O REDUCE
31 /**
32  * Reduz uma coleção a um único valor combinando iterativamente os
33  * elementos da coleção usando a função fornecida.
34  * O iterável deve ter pelo menos um elemento. Se tiver apenas um elemento,
35  * esse elemento é retornado.
36  * Caso contrário, este método começa com o primeiro elemento do iterador e,
37  * em seguida, combina-o com os elementos restantes na ordem de iteração.
38  */
39
40 //Declarando uma lista de valores double
41 var precoPlacaVideo = [15000.99, 23000.99, 16000.99, 19000.99];
42
43 //O reduce() é mais interessante que o Map, pois podemos transformar
44 //no lista no que quisermos. O reduce() pede uma função para determinar
45 //o que vais fazer
46 var precoTotal = precoPlacaVideo.reduce(funcaoSomarValores);
47
48 //Imprimindo o resultado
49
50 print('=====');
51 print('Soma dos valores: $precoTotal');
52
53 ///OUTRO EXEMPLO
54
55 //Declarando minha lista
56 var listaAlunos = ['Maria', 'José', 'Pedro'];
57
58 //usar reduce() para juntar os nomes
59 String alunosCombinados =
60     listaAlunos.reduce((valor, elemento) => "$valor | $elemento");
61
62 print('Lista combinada: $alunosCombinados');
63
64
```

Aula 2 – Funções (funcao_reduce.dart)

```
64
65 //Criar função para o reduce
66 //Veja que ele pede uma função:
67 //double reduce(double Function(double, double) combine)
68 double funcaoSomarValores(double a, double b) {
69   //O que acontece
70   print('Valor de $a + $b');
71
72   //Retorno
73   return a + b;
74 }
```

MÉTODO CONVENCIONAL

=====

Valor total das notas: 31.0

Média de notas: 7.75

MÉTODO REDUCE()

Valor de 15000.99 + 23000.99

Valor de 38001.98 + 16000.99

Valor de 54002.97 + 19000.99

=====

Soma dos valores: 73003.96

Lista combinada: Maria | José | Pedro

Exited

Aula 2 – Funções (funcao_reduce.dart)

```
1 //Exemplo na utilização do .map(), .reduce(), .where()
2
3 import 'funcao_void.dart';
4
5 Run | Debug
6 void main() {
7   //Declarar uma lista de nomes e valores
8   var listaNomeValor = [
9     {'nome': 'John', 'valor': 100.5},
10    {'nome': 'Jane', 'valor': 200.5},
11    {'nome': 'Carol', 'valor': 300.5},
12    {'nome': 'Mike', 'valor': 400.5},
13  ];
14
15  //Pega Valores
16  var ValorFinal = listaNomeValor
17    .map((nome) => nome['valor']) //Cria uma lista de valores
18    .cast<double>() //Faz um casting no valor
19    .map((valor) => (valor as double).roundToDouble())
20    .where((valor) => valor >= 200.0); //Cria uma lista com critério
21
22  //Soma os valores da lista filtrada pelo Where
23  var total = ValorFinal.reduce((valor, elemento) => valor + elemento);
24
25  //Calcula a média
26  var media = total / ValorFinal.length;
27
28  //Saída
29  linha();
30  print('A média de valores é: $media');
31  linha();
32 }
```

A média de valores é: 301.0

Exited

Programação Orientada a Objetos



Aula 3 – Orientação a objetos

Programação Orientada a Objetos (também conhecida pela sua sigla POO) é um modelo de análise, projeto e programação de software baseado na composição e interação entre diversas unidades chamadas de 'objetos'. A POO é um dos 4 principais paradigmas de programação (as outras são programação imperativa, funcional e lógica). Os objetos são operados com o conceito de 'this' (isto) ou 'self' (si), de forma que seus métodos (muitas vezes) modifiquem os dados da própria instância.

Os programas são arquitetados através de objetos que interagem entre si. Dentre as várias abordagens da POO, as baseadas em classes são as mais comuns: objetos são instâncias de classes, o que em geral também define o tipo do objeto.

Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos.

A alternativa mais usual ao uso de classes é o uso de protótipos. Neste caso, objetos são cópias de outros objetos, não instâncias de classes. Javascript e Lua são exemplos de linguagens cuja POO é realizada por protótipos. A diferença prática mais evidente é que na POO baseada em protótipos apenas a herança simples é implementada pela cópia do objeto. Assim, na POO, implementa-se um conjunto de classes passíveis de serem instanciadas como objetos, e.g. Python e C++ (ou objetos protótipos que são copiados e alterados, e.g. JavaScript e VimL).

Aula 3 – Orientação a objetos

Críticas à POO

As críticas mais recorrentes à POO podem ser resumidas em:

- 1) não satisfazer os objetivos de reusabilidade e modularidade, e
- 2) a ênfase demasiada em design e modelamento de software em detrimento de outros aspectos (computabilidade/algoritmos).
- 3) Não há consenso de que a POO realmente seja útil para modelar a realidade, tampouco se modelar a realidade é um objetivo pertinente. A premissa de que 'a POO é apropriada para modelar sistemas complexos com comportamentos complexos', contrasta com o renomado princípio KISS (mantenha simples). Linguagens naturais não compartilham da estratégia usual na POO: 'priorizar coisas ao invés de ações', o que leva muitas vezes a representações contorcidas.
- 4) A POO muitas vezes é realizada sem uma representação transparente do fluxo de controle (ordem das instruções), o que tem se tornado uma desvantagem com a relevância, acentuada nos últimos anos, do processamento paralelo. Outras críticas usuais incluem: a POO é 'intrinsecamente menos eficiente' que o código procedural, em geral demora mais para compilar, e tende a ser extremamente complexa

Críticas de programadores notórios

Joe Armstrong: "O problema com linguagens orientadas a objetos é que você tem todo esse ambiente implícito que elas levam consigo. Você queria uma banana mas o que você conseguiu foi um gorila segurando a banana e uma selva inteira."

Alexander Stepanov: "Dizer que tudo é um objeto é simplesmente dizer nada."

Steve Yegge: "Por que alguém iria tão longe para colocar um só aspecto da fala [substantivos] em um pedestal?"

Eric Steven Raymond: "A POO encoraja programas com muitas camadas e destrói a transparência."

Rob Pike: "A POO constitui os números romanos da computação."

Aula 3 – Linguagens de Programação

Procedural:

O termo Programação procedural (ou programação procedimental) é às vezes utilizado como sinônimo de Programação imperativa (Paradigma de programação que especifica os passos que um programa deve seguir para alcançar um estado desejado), mas pode se referir (como neste artigo) a um paradigma de programação baseado no conceito de chamadas a procedimento (en: procedure call). Os Procedimentos, também conhecidos como rotinas, subrotinas, métodos, ou funções (que não devem ser confundidas com funções matemáticas, mas são similares àquelas usadas na programação funcional) simplesmente contêm um conjunto de passos computacionais a serem executados. Um dado procedimento pode ser chamado a qualquer hora durante a execução de um programa, inclusive por outros procedimentos ou por si mesmo. Exemplo: PHP, Visual Basic, Delphi...

Imperativa:

Na Ciência da Computação, programação imperativa é um paradigma de programação que descreve a computação como ações, enunciados ou comandos que mudam o estado (variáveis) de um programa. Muito parecido com o comportamento imperativo das linguagens naturais que expressam ordens, programas imperativos são uma sequência de comandos para o computador executar. O nome do paradigma, Imperativo, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo... Este paradigma de programação se destaca pela simplicidade, uma vez que todo ser humano, ao se programar, o faz imperativamente, baseado na ideia de ações e estados, quase como um programa de computador. Exemplo: Bava, Basic, Cobol, Fortran...

Funcional:

Em ciência da computação, programação funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Ela enfatiza a aplicação de funções, em contraste da programação imperativa, que enfatiza mudanças no estado do programa. Enfatizando as expressões ao invés de comandos, as expressões são utilizados para cálculo de valores com dados imutáveis. Exemplo: Lisp, Haskell, Elixir, *Dart...

Lógica:

É um paradigma de programação que faz uso da lógica matemática. John McCarthy [1958] foi o primeiro a publicar uma proposta de uso da lógica matemática para programação. A primeira linguagem de programação lógica foi a Planner, a qual permitia a invocação orientada a padrões de planos procedimentais de asserções e de objetivos. Com a necessidade de adaptação aos sistemas de memória muito limitada, que eram disponíveis quando ela foi desenvolvida. A linguagem Planner usava estruturas de controle de backtracking, de tal forma que apenas um único caminho computacional tinha que ser armazenado por vez. Em seguida, o Prolog foi desenvolvido como uma simplificação do Planner que permitia a invocação orientada a padrões apenas a partir de objetivos (também baseado em backtracking).

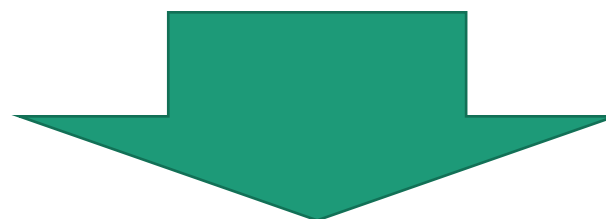
OBJETOS OU INSTÂNCIAS
CASA 1... CASA 2...

MÉTODOS
ABRIR...
FECHAR...

CLASSE



**MOLDE
MODELO**



ATRIBUTOS
CASA AMARELA
CASA MARROM
QUANTIDADE JANELAS

Aula 3 – Linguagens de Programação

Classes:

As classes de programação são receitas de um objeto, aonde têm características e comportamentos, permitindo assim armazenar propriedades e métodos dentro dela. Para construir uma classe é preciso utilizar o pilar da abstração. Uma classe geralmente representa um substantivo, por exemplo: uma pessoa, um lugar, algo que seja “abstrato”.

Uma classe é um tipo definido pelo usuário que contém a “receita”, a especificação para os objetos, algo mais ou menos como o tipo inteiro contém o molde para as variáveis declaradas como inteiros. A classe envolve, associa, funções e dados, controlando o acesso a estes, defini-la implica em especificar os seus atributos (dados) e seus métodos (funções).

Um programa que utiliza uma interface controladora de um motor elétrico provavelmente definiria a classe motor. Os atributos desta classe seriam: temperatura, velocidade, tensão aplicada. Estes provavelmente seriam representados na classe por tipos como int ou float. Os métodos desta classe seriam funções para alterar a velocidade, ler a temperatura, etc.

Objetos:

Objetos (computacionais) são caracterizados por atributos e métodos. Atributos são as propriedades de um objeto. Métodos são as ações que um objeto pode realizar. Os objetos são características definidas pelas classes. Neles é permitido instanciar objetos da classe para inicializar os atributos e invocar os métodos.

Atributos:

Atributos são as características de um objeto, essas características também são conhecidas como variáveis, utilizando o exemplo das casas, temos alguns atributos, tais como: cor, quantidadeJanelas, area.

Métodos:

Métodos são as ações que os objetos podem exercer quando solicitados, onde podem interagir e se comunicarem com outros objetos, utilizando o exemplo das casas, temos alguns exemplos: abrirPorta, fecharJanela, acenderLuz.

Construtores

Construtores são basicamente funções de inicialização de uma classe, as quais são invocadas no momento em que objetos desta classe são criadas. Eles permitem inicializar campos internos da classe e alocar recursos que um objeto da classe possa demandar, tais como memória, arquivos, semáforos, soquetes, etc.

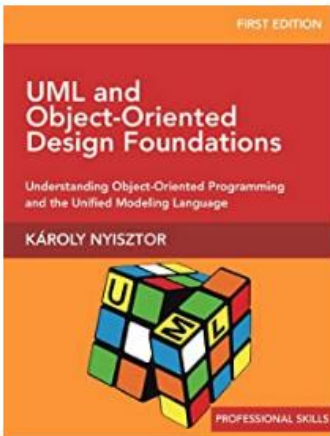
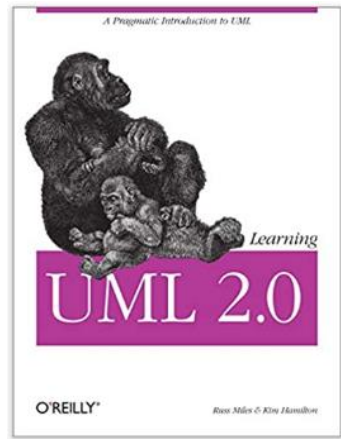
A função construtor de uma classe é reconhecida por ter o mesmo nome da classe. Por exemplo, um construtor para uma classe chamada Rectangle seria a função Rectangle(). É possível definir-se mais de um construtor; em outras palavras, a função membro construtor, assim como qualquer outra função membro de uma classe, pode ser sobrecarregada.

Aula 3 - Linguagens de Programação

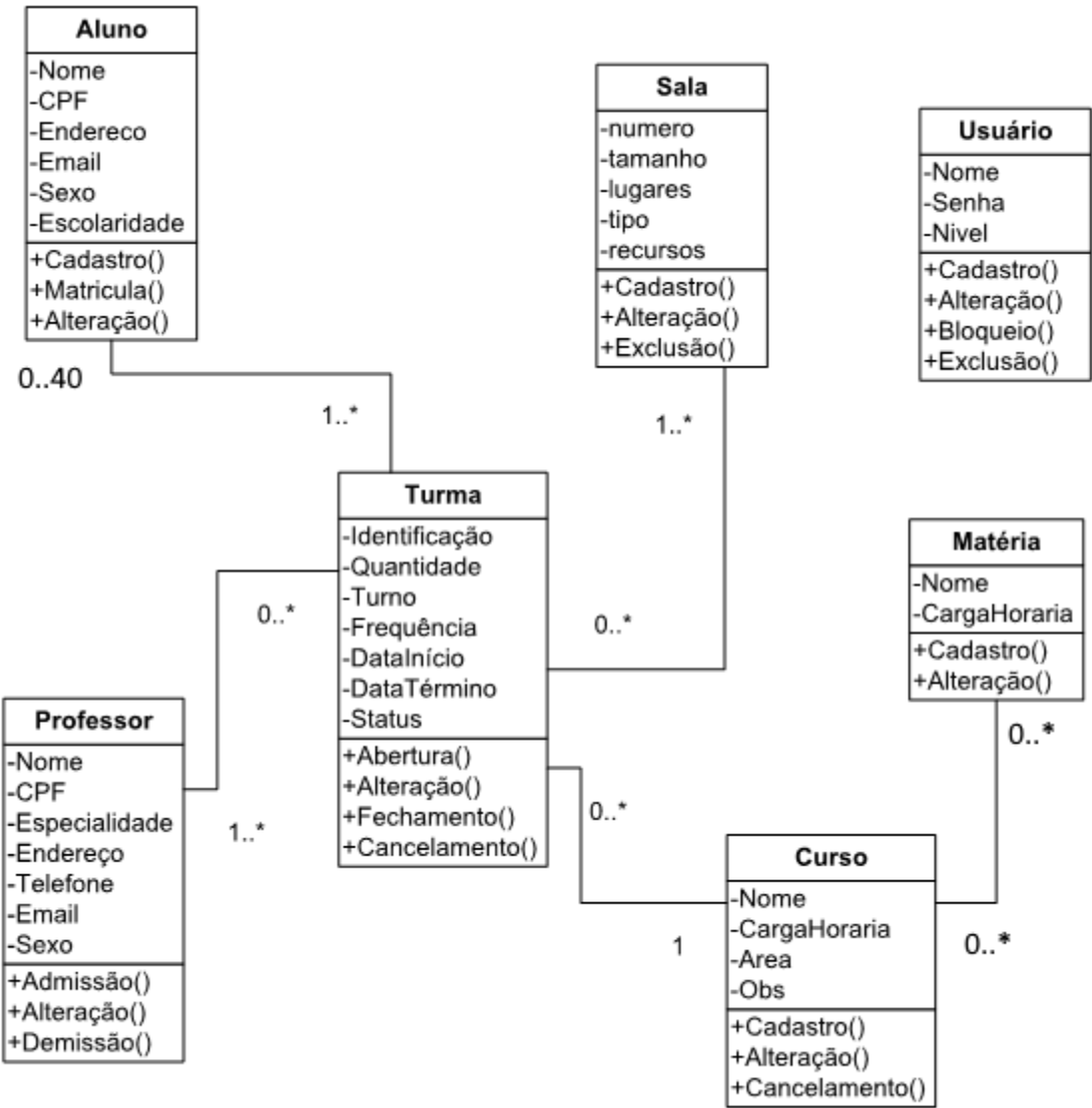
Veja ao lado um exemplo de um **diagrama de classe** de um controle de alunos com as seguintes classes, propriedades e métodos.

UML, abreviação de Unified Modeling Language, é uma linguagem de modelagem padronizada que consiste em um conjunto integrado de diagramas, desenvolvido para ajudar os desenvolvedores de sistema e software a especificar, visualizar, construir e documentar os artefatos de sistemas de software, bem como para modelagem de negócios e outros sistemas que não sejam de software.

A UML representa uma coleção das melhores práticas de engenharia que provaram ser bem-sucedidas na modelagem de sistemas grandes e complexos. A UML é uma parte muito importante do desenvolvimento de software orientado a objetos e do processo de desenvolvimento de software. A UML usa principalmente notações gráficas para expressar o design de projetos de software.



Softwares para UML
Lucidchart for Enterprise
Visio
Draw.IO
Astah



Aula 3 – Classes: 01-classes.dart

```
1 //Para criar uma classe usamos inicial maiúscula
2 //Em caso de palavra composta, exemplo: CasaDaJane
3 class Casa {
4     //Criando meus atributos
5     //Atributos definem minha classe
6     String cor = '';
7     int quantidadeJanelas = 0;
8     int numero = 0;
9
10    //Criando métodos
11    //Métodos são ações que pode ser realizadas com nossa classe
12    void abrirJanela(String posicao) {
13        print('Abrindo a janela...${posicao}');
14    }
15
16    void fecharJanela(String posicao) {
17        print('Fechando a janela...${posicao}');
18    }
19 }
20
21 Run | Debug
22 void main() {
23     //Definido minha classe
24     //A casa do John é uma instância de Casa
25     //Ou seja uma cópia do modelo original Casa().
26     //Assim cadaDoJohn é um objeto de Casa().
27     Casa casaDoJohn = new Casa();
28
29     //Tendo construído um objeto casaDoJohn
30     //Podemos colocar valores em seus atributos
31     casaDoJohn.cor = 'Marrom';
32     casaDoJohn.quantidadeJanelas = 5;
33     casaDoJohn.numero = 1970;
```

```
34 //Acessando o método abrirJanela()
35 //É preciso o objeto para fazer uso não só de atributos, mas de métodos também
36 casaDoJohn.abrirJanela('Frente!');
37 print('A casa do John é: ${casaDoJohn.cor}');
38 print('A casa do John tem: ${casaDoJohn.quantidadeJanelas} janelas');
39 print('O número da casa do John é: ${casaDoJohn.numero}');
40 //Acessando o método fecharJanela()
41 casaDoJohn.fecharJanela('Fundos!');
42
43 //A vantagem de utilizar classes é que podemos ter várias instâncias
44 //dessa mesma classe
45 //então agora posso criar a casa da Jane também, aí ninguém briga :)
46
47 //Criando a instância da casa da Jane
48 Casa casaDaJane = new Casa();
49
50 casaDaJane.cor = 'Verde';
51 casaDaJane.quantidadeJanelas = 10;
52 casaDaJane.numero = 2000;
53
54 //Saída para os dados da Casa da Jane
55 print('-----');
56 casaDoJohn.abrirJanela('Frente!');
57 print('A casa da Jane é: ${casaDaJane.cor}');
58 print('A casa da Jane tem: ${casaDaJane.quantidadeJanelas} janelas');
59 print('O número da casa da Jane é: ${casaDaJane.numero}');
60 casaDoJohn.fecharJanela('Fundos!');
61 }
```

Aula 3 – Classes: 02-classes.dart

```
1 //Outro exemplo
2 class Data {
3   ..//Colocamos o LATE para dizer ao dart que o atributo vai ser
4   ..//inicializado futuramente -- NÃO É LEGAL, POIS VOCÊ PODE NÃO INICIÁ-LO
5   ..late int dia;
6   ..late String mes;
7   ..late int ano;
8 }
9
10 Run | Debug
11 void main() {
12   ..//Criando uma data (3 INSTÂNCIAS)
13   ..Data data = new Data();
14   ..Data dataPedido = new Data();
15   ..Data dataSaida = new Data();
16
17   ..//Inicializando uma data
18   ..data.dia = 3;
19   ..data.mes = 'Janeiro';
20   ..data.ano = 1970;
21
22   ..//Inicializando a data
23   ..dataPedido.dia = 20;
24   ..dataPedido.mes = 'Março';
25   ..dataPedido.ano = 2015;
26
27   ..//Inicializando a data
28   ..dataSaida.dia = 29;
29   ..dataSaida.mes = 'Dezembro';
30   ..dataSaida.ano = 2020;
31
32   ..//Imprimindo as Datas
33   ..print('Mostrar data: ${data.dia}/${data.mes}/${data.ano}');
34   ..print('Mostrar data: ${dataPedido.dia}/${dataPedido.mes}/${dataPedido.ano}');
35   ..print('Mostrar data: ${dataSaida.dia}/${dataSaida.mes}/${dataSaida.ano}');
36 }
```

01-classes.dart

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
			Abrindo a janela...Frente! A casa do John é: Marrom A casa do John tem: 5 janelas O número da casa do John é: 1970 Fechando a janela...Fundos! ----- Abrindo a janela...Frente! A casa da Jane é: Verde A casa da Jane tem: 10 janelas O número da casa da Jane é: 2000 Fechando a janela...Fundos! Exited

02-classes.dart

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
			Mostrar data: 3/Janeiro/1970 Mostrar data: 20/Março/2015 Mostrar data: 29/Dezembro/2020 Exited

Aula 3 – Classes

03-metodos.dart

```
1 class Casa-{
2   ..//Atributos definem minha classe
3   ..String cor = '';
4   ..int quantidadeJanelas = 0;
5   ..int numero = 0;
6
7   ..//Métodos são ações que pode ser realizadas com nossa classe
8   ..void abrirJanela(String posicao) { ..//Método
9     ..print('Abrindo a janela...${posicao}');
10  }
11
12  ..void fecharJanela(String posicao) { ..//Método
13    ..print('Fechando a janela...${posicao}');
14  }
15
16  ..//Métodos dentro de métodos
17  ..//This serve para fazer referência a atributos e métodos
18  ..//que estão dentro da mesma classe
19  ..void checarCasa() { ..//Método
20    ..this.abrirJanela('Frente!');
21    ..this.fecharJanela('Fundos!');
22  }
23 }
```

Run | Debug

```
24 void main() {
25   ..//Criando instância da Classe Casa
26   ..Casa casaDoJohn = new Casa();
27
28   ..//Atribuindo valores
29   ..casaDoJohn.cor = 'Marrom';
30   ..casaDoJohn.quantidadeJanelas = 5;
31   ..casaDoJohn.numero = 1970;
32   ..//Checando a casa do John
33   ..casaDoJohn.checarCasa();
34 }
```

04-metodos.dart

```
1 class Data-{
2   ..//Colocamos o LATE para dizer ao dart que o atributo vai ser
3   ..//inicializado futuramente -- NÃO É LEGAL, POIS VOCÊ PODE NÃO INICIÁ-LO
4   ..late int dia;
5   ..late String mes;
6   ..late int ano;
7
8   ..//Criando Métodos
9   ..String exibirData() {
10     ..//Exibir data formatada
11     ..//Essa colocação é inútil, pois não podemos manipular a data
12     ..return '$dia/$mes/$ano';
13   }
14
15   ..//Usando o método toString para retornar a exibição da data
16   ..String toString() {
17     ..//Vai retornar nosso método anterior
18     ..//convertendo em String
19     ..return exibirData();
20   }
21 }
22 }
```

Run | Debug

```
23 void main() {
24   ..//Criando uma data (3 INSTÂNCIAS)
25   ..Data data = new Data();
26   ..Data dataPedido = new Data();
27   ..Data dataSaida = new Data();
28
29   ..//Inicializando uma data
30   ..data.dia = 3;
31   ..data.mes = 'Janeiro';
32   ..data.ano = 1970;
33 }
```

Aula 3 – Classes

04-metodos.dart (cont)

```
34  ..//Inicializando a data
35  ..dataPedido.dia = 20;
36  ..dataPedido.mes = 'Março';
37  ..dataPedido.ano = 2015;
38
39  ..//Inicializando a data
40  ..dataSaida.dia = 29;
41  ..dataSaida.mes = 'Dezembro';
42  ..dataSaida.ano = 2020;
43
44  ../*Imprimindo as Datas
45  ..print('Mostrar data: ${data.dia}/${data.mes}/${data.ano}');
46  ..print('Mostrar data: ${dataPedido.dia}/${dataPedido.mes}/${dataPedido.ano}');
47  ..print('Mostrar data: ${dataSaida.dia}/${dataSaida.mes}/${dataSaida.ano}');
48  ..*/
49
50  ..//Nova Saída
51  ..//Cada uma das instâncias vai receber o comportamento do método exibirData
52  ..//Muito Prático
53  ..String data1 = data.exibirData();
54  ..String data2 = dataPedido.exibirData();
55  ..String data3 = dataSaida.exibirData();
56
57  ..print('Data atual: $data1');
58  ..print('Data da realização Pedido: $data2');
59  ..print('Data da Saída do Pedido: $data3');
60
61  ..//Saída pelo método toString
62  ..print('-----');
63  ..//Imprimindo diretamente o objeto!
64  ..print(data);
65  ..print(dataPedido);
66  ..print(dataSaida);
67 }
```

03-metodos.dart

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Abrindo a janela...Frente!
Fechando a janela...Fundos!
Exited
```

04-metodos.dart

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Data atual: 3/Janeiro/1970
Data da realização Pedido: 20/Março/2015
Data da Saída do Pedido: 29/Dezembro/2020
-----
3/Janeiro/1970
20/Março/2015
29/Dezembro/2020
Exited
```

Aula 3 – Classes: 05-construtor.dart

```
1 class Data {
2   ..//Colocamos o LATE para dizer ao dart que o atributo vai ser
3   ..//inicializado futuramente -- NÃO É LEGAL, POIS VOCÊ PODE NÃO INICIÁ-LO
4   ..late int dia;
5   ..late String mes;
6   ..late int ano;
7
8   ..//Criando um construtor
9   ..//Se não criar, a linguagem vai criar um padrão
10  ..//Ele tem o mesmo nome da Classe
11  ..//Só posso ter um com o mesmo nome da Classe
12  ..//Colocamos parâmetros para receber os valores
13  Data(int diaCorrente, String mesCorrent, int anoCorrent) { //Construtor
14    ..//Atribuimos os valores aos nosso atributos
15    ..dia = diaCorrente;
16    ..mes = mesCorrent;
17    ..ano = anoCorrent;
18  }
19
20  String exbirData() { //Método
21    ..return '$dia/$mes/$ano';
22  }
23
24  String toString() { //Método
25    ..return exbirData();
26  }
27 }
```

```
28 void main() {
29   ..//Com o construtor criado
30   ..//Posso agora colocar os valores diretamente na criação do objeto
31   Data data = new Data(3, 'Janeiro', 1970);
32   Data dataPedido = new Data(20, 'Março', 2015);
33   Data dataSaida = new Data(29, 'Dezembro', 2020);
34
35   ..//Cada uma das instâncias vai receber o omportamento do método exibirData()
36   ..//Muito Prático
37   String data1 = data.exbirData();
38   String data2 = dataPedido.exbirData();
39   String data3 = dataSaida.exbirData();
40
41   print('Data atual: $data1');
42   print('Data da realização Pedido: $data2');
43   print('Data da Saída do Pedido: $data3');
44
45   ..//Saída pelo método toString
46   print('-----');
47   ..//Imprimindo diretamente o objeto!
48   print(data);
49   print(dataPedido);
50   print(dataSaida);
51 }
```

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
			Data atual: 3/Janeiro/1970 Data da realização Pedido: 20/Março/2015 Data da Saída do Pedido: 29/Dezembro/2020 ----- 3/Janeiro/1970 20/Março/2015 29/Dezembro/2020 Exited



Siga o Senac em Minas nas Redes Sociais:

