



Introdução ao Dart



Hello, World



```
void main() {  
    print('Hello, World!');  
}
```

Variáveis



```
// Criando uma variável e inicializando
```

```
var name = 'Bob';
```

```
//O tipo da variável name pode ser inferida, mas você pode  
especificar de forma explícita
```

```
String name = 'Bob';
```

```
// Se um objeto não for definido por um único tipo, use o  
tipo Object (ou dynamic se necessário)
```

```
Object name = 'Bob';
```

Strings



```
var s1 = 'Single quotes work well for string literals.';  
var s2 = "Double quotes work just as well.";   
var s3 = 'It\'s easy to escape the string delimiter.';  
var s4 = "It's even easier to use the other delimiter.";
```

Strings

```
void main() {  
    String name = 'John';  
    String message = 'Hello $name';  
    print(message);  
}
```

```
// Output  
// Hello John
```

Números

```
// Integers são números sem ponto decimal.
```

```
int x = 1;
```

```
// Se um número incluir um valor decimal, será um double.
```

```
double y = 1.1;
```

```
// Você também pode declarar uma variável como um número.  
// A variável poderá ter tanto um valor integer quanto um  
valor double.
```

```
num z = 1;
```

```
// Integer criados de maneira literal são automaticamente  
convertidos para double quando necessário
```

```
double a = 1; // Equivalente a double a = 1.0.
```

List



```
var list = [1, 2, 3];
```



```
var list = [  
    'Car',  
    'Boat',  
    'Plane',  
];
```

List



```
var list = [1, 2, 3];  
assert(list.length == 3);  
assert(list[1] == 2);
```

```
list[1] = 1;  
assert(list[1] == 1)
```




const
x
final

const


```
void main() {  
    double distanceMile = 1;  
    const toKm = 1.609;  
    double distanceKm = distanceMile * toKm;  
  
    print('$distanceMile Mile = $distanceKm Km');  
  
    // error  
    toKm = 1.6;  
}
```

const



```
void main() {  
    const currentTime = DateTime.now();  
    print(currentTime);  
}
```


final



```
void main() {  
    final DateTime currentTime;  
    currentTime = DateTime.now();  
  
    // error  
    currentTime = DateTime.utc(2022, 12, 31);  
  
    print(currentTime);  
}
```



Executando Funções



```
void greet() {  
    print('Hi there');  
}  
  
void main() {  
    greet();  
}
```

Passando informações para funções

```
void greet(String name) {  
    print('Hi $name!');  
}
```

```
void main() {  
    greet('Dash');  
}
```

Retornando valores

```
String greet(String name) {  
    return 'Hi $name!';  
}  
  
void main() {  
    String name = 'Dash';  
    String greeting = greet(name);  
    print(greeting);  
}
```



Sintaxe abreviada



```
String greet(String name) => 'Hi $name!';
```




Null Safety



Null safety

- O Dart oferece segurança para valores nulos, o que significa que os valores não podem ser nulos, a menos que você diga que podem ser
- Pode protegê-lo de exceções (null exceptions) em tempo de execução por meio da análise de código estático.



```

// Sem null safety:
bool isEmpty(String string) ⇒ string.length == 0;

main() {
  isEmpty(null);
}
```



Declarando Variáveis Não Nulas



Variáveis não nulas devem sempre ser inicializadas com valores não nulos.



```
void main() {  
    int age; // non-nullable  
    age = null;  
}
```



```
void main() {  
    int age; // non-nullable  
    age = 18;  
}
```

Declarando Variáveis Nulas



```
String? name; //inicializado como nulo por padrão
```

```
int? age = 36; // inicializado com valor não nulo
```

```
age = null; // pode ser reatribuído para nulo
```

Assertion operator



```
int? maybeValue = 42;  
int value = maybeValue!; // valid, value is non-nullable
```



```
String? name;  
print(name!); // NoSuchMethodError: '<Unexpected Null Value>'  
print(null!); // NoSuchMethodError: '<Unexpected Null Value>'
```

Flow Analysis

```
int absoluteValue(int? value) {  
    if (value == null) {  
        return 0;  
    }  
    // if we reach this point, value is non-null  
    return value.abs();  
}
```

Exercício

No código abaixo, corrija o erro para retornar o tamanho (length) da string ou null se for nulo:

```
int? stringLength(String? nullableString) {  
    return nullableString.length;  
}
```



Null-aware Operator

```
int? first(List<int>? items) {  
    return items != null ? items[0] : null;  
}
```

```
int? first(List<int>? items) {  
    return items?[0];  
}
```



Null-coalescing operators

```
// Ambos irão printar 'alternate' se nullableString for null  
print(nullableString ?? 'alternate');  
  
print(nullableString != null ? nullableString : 'alternate');
```



Parâmetros



Named parameters



```
/// Sets the [bold] and [hidden] flags ...  
void enableFlags({bool? bold, bool? hidden}) {...}
```



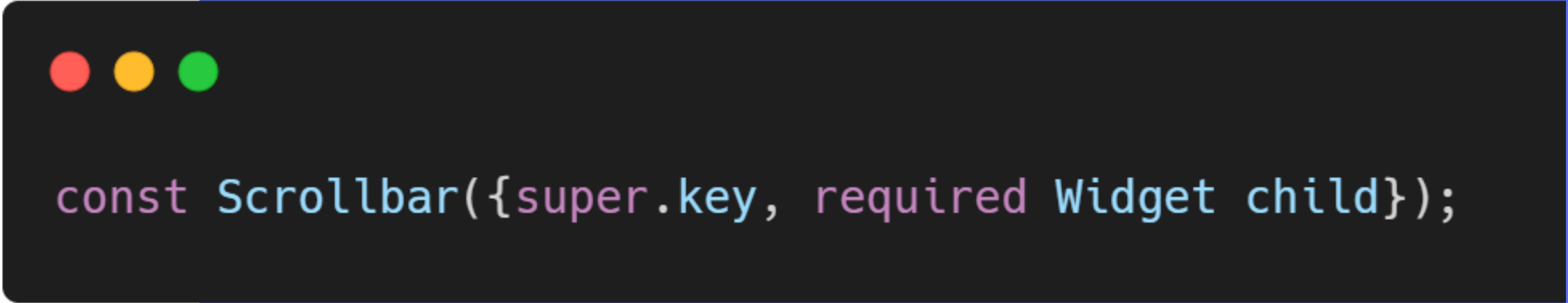
```
enableFlags(bold: true, hidden: false);
```

Default value

```
/// Sets the [bold] and [hidden] flags ...  
void enableFlags({bool bold = false, bool hidden = false}) {...}  
  
// bold will be true; hidden will be false.  
enableFlags(bold: true);
```



Required



```
const Scrollbar({super.key, required Widget child});
```

Optional positional parameters

```
String say(String from, String msg, [String? device]) {  
    var result = '$from says $msg';  
    if (device != null) {  
        result = '$result with a $device';  
    }  
    return result;  
}
```



Exercício

Escreva uma função que receba como primeiro parâmetro posicional uma lista de inteiros (x) e um segundo parâmetro nomeado e obrigatório, um valor inteiro (y).
x pode ser nulo.
y não pode ser nulo.

Retorne a quantidade de vezes que o valor (x) aparece na lista (y),
se a lista (x) for nula retorne 0;
No final, escreva (print) com a seguinte mensagem:

A quantidade de vezes que o valor (x) aparece na lista é de:
(número de vezes)



Orientação a Objetos



Definindo uma Classe

```
// By convention, the class names follow the  
PascalCase naming convention  
class MyClass {  
  
}  
  
class Point {  
}
```

Criando objetos

```
class Point {  
  
}  
  
void main() {  
    var p1 = Point();  
}
```

Adicionando propiedades

```
class Point {  
    int x = 0;  
    int y = 0;  
}  
  
void main() {  
    var p1 = Point();  
    p1.x = 10;  
    p1.y = 20;  
}
```

Adicionando métodos

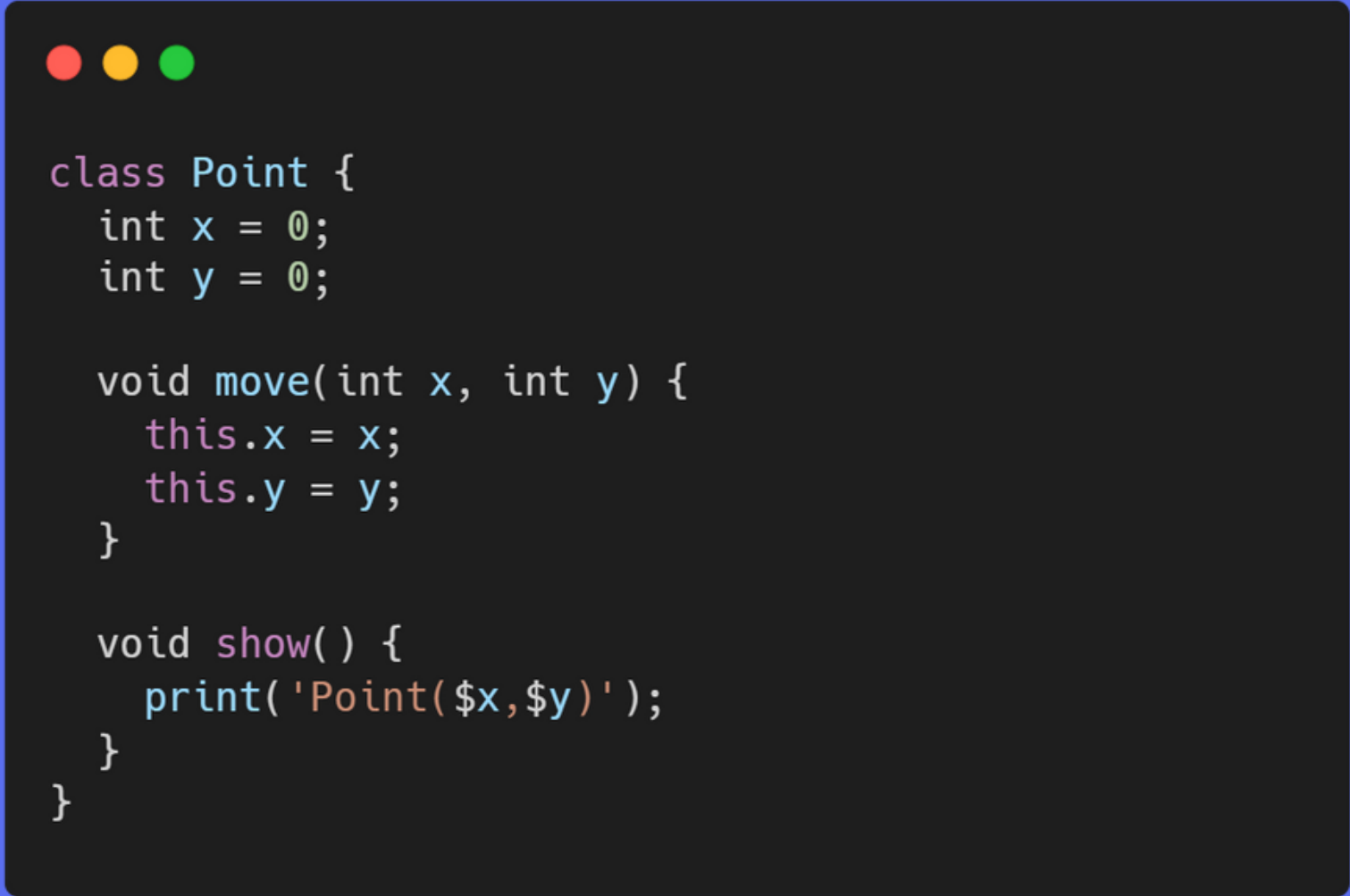
```
class Point {  
    int x = 0;  
    int y = 0;  
  
    void move(int x1, int y1) {  
        x = x1;  
        y = y1;  
    }  
  
    void show() {  
        print('Point($x,$y)');  
    }  
}  
  
void main() {  
    var p1 = Point();  
    p1.x = 10;  
    p1.y = 20;  
  
    p1.move(100, 200);  
    p1.show();  
}
```

private fields

```
class Point {  
    int _x = 0;  
    int _y = 0;  
  
    Point({int x = 0, int y = 0}) {  
        this._x = x;  
        this._y = y;  
    }  
    show() {  
        print('Point(x=$_x,y=$_y)');  
    }  
}
```



this keyword



```
class Point {  
    int x = 0;  
    int y = 0;  
  
    void move(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    void show() {  
        print('Point($x,$y)');  
    }  
}
```

Construtores



Default constructor

```
class Point {  
    int x = 0;  
    int y = 0;  
  
    Point() {  
        print('Constructor was called.');    }  
}  
  
void main() {  
    var p1 = Point();  
}
```

Custom constructor

```
class Point {  
    int x = 0;  
    int y = 0;  
  
    Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
void main() {  
    var p1 = Point(10, 20);  
}
```

short-form constructor

```
class Point {  
    int x = 0;  
    int y = 0;  
  
    Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
void main() {  
    var p1 = Point(10, 20);  
}
```

Named constructor

```
class Point {  
    int x = 0;  
    int y = 0;  
  
    Point(this.x, this.y);  
  
    Point.origin() {  
        this.x = 0;  
        this.y = 0;  
    }  
}  
  
void main() {  
    var p1 = Point.origin();  
}
```

Forwarding constructors

```
class Point {  
    int x = 0;  
    int y = 0;  
  
    Point(this.x, this.y);  
  
    Point.origin() : this(0, 0);  
}  
  
void main() {  
    var p1 = Point.origin();  
}
```

Const Constructor

```
class Text {  
    final String content;  
    Text({this.content = ''});  
}  
  
void main() {  
    const text = Text(content: 'Hello');  
    print(text.content);  
}
```

Const Constructor



```
// error  
text.content = 'Bye';
```

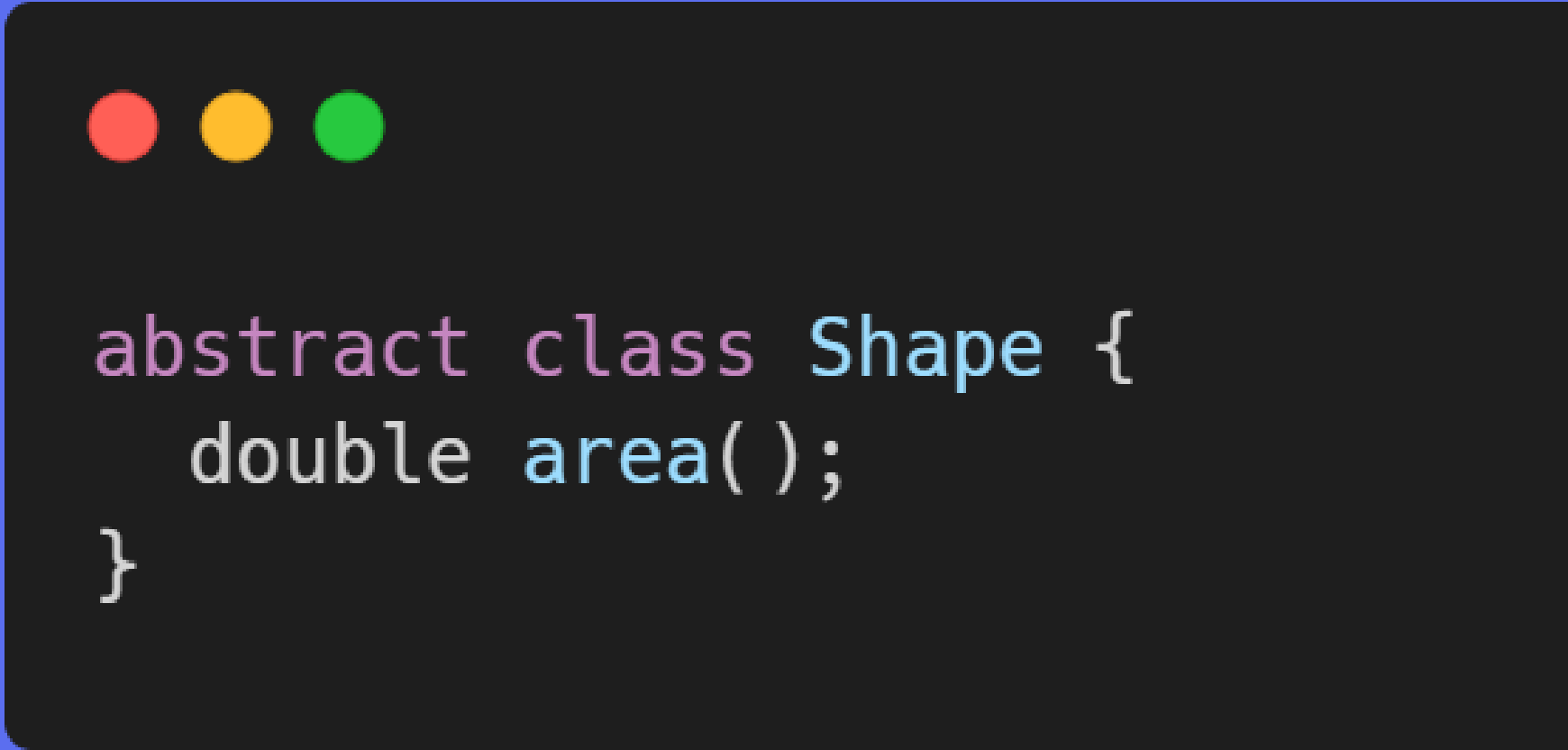
'content' can't be used as a setter because it's final.

Const Constructor

```
class Text {  
    final String content;  
    const Text({this.content = ''});  
}  
  
void main() {  
    var text1 = const Text(content: 'Hello');  
    var text2 = const Text(content: 'Hello');  
    print(identical(text1, text2)); // true  
}
```


A white rectangular window with rounded corners and a title bar at the top containing three small circles (black, blue, grey).

abstract classes

A dark rectangular window with rounded corners and a title bar at the top containing three small circles (red, yellow, green).

```
abstract class Shape {  
    double area();  
}
```

abstract classes

```
class Circle extends Shape {  
  double radius;  
  Circle({this.radius = 0});  
  
  @override  
  double area() => 3.14 * radius * radius;  
}
```

```
class Square extends Shape {  
  double length;  
  Square({this.length = 0});  
  
  @override  
  double area() => length * length;  
}
```

Exercício

Crie um construtor nomeado "black" que seta todas as 3 propriedades para 0.

```
class Color {  
    int red;  
    int green;  
    int blue;  
  
    Color(this.red, this.green, this.blue);  
  
    // Crie um construtor nomeado chamado "black" aqui:  
}
```



Exercício

Transforme a class Color em um objeto imutável e utilize const para os construtores

```
class Color {  
    int red;  
    int green;  
    int blue;  
  
    Color(this.red, this.green, this.blue);  
  
    // Crie um construtor nomeado chamado "black" aqui:  
}
```





**Obrigado pela
presença!**