REFERÊNCIA RÁPIDA DART

Tipos, funções, classes, async, null safety, coleções

Básico

Hello World

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

Variáveis

```
var name = 'Dart';        // tipo inferido
String lang = 'Dart';     // tipo explícito
final pi = 3.14;          // constante em tempo de execução
const max = 100;          // constante em tempo de compilação
```

Interpolação de String

```
var name = 'World';
print('Hello, $name!');
print('1 + 1 = ${1 + 1}');
```

Tipos

Tipos Embutidos

`int`	Inteiro de 64 bits
`double`	Ponto flutuante de 64 bits
`num`	Supertipo de int e double
`String`	String UTF-16
`bool`	true ou false
`List`	Coleção ordenada (array)
`Set`	Coleção única não ordenada
`Map`	Pares chave-valor
`dynamic`	Qualquer tipo, desativa verificação estática
`void`	Sem valor de retorno

Verificações e Conversões de Tipo

```
if (obj is String) print(obj.length);
var s = obj as String;   // cast
print(obj.runtimeType);  // tipo em tempo de execução
```

Funções

Sintaxe de Função

```
int add(int a, int b) => a + b;
void greet({required String name}) {
  print('Hello, $name');
}
```

Parâmetros

`int f(int a)`	Parâmetro posicional obrigatório
`int f([int a = 0])`	Posicional opcional com padrão
`f({required int a})`	Parâmetro nomeado obrigatório
`f({int a = 0})`	Nomeado opcional com padrão

Closures e Tearoffs

```
var square = (int n) => n * n;
[1, 2, 3].map((e) => e * 2);
[1, 2, 3].forEach(print);  // tearoff
```

Controle de Fluxo

Condicionais

```
if (x > 0) { print('pos'); }
else if (x == 0) { print('zero'); }
else { print('neg'); }
var result = x > 0 ? 'pos' : 'neg';
```

Laços

```
for (var i = 0; i < 5; i++) { }
for (var item in list) { }
while (x > 0) { x--; }
do { x--; } while (x > 0);
```

Switch e Pattern Matching

```
switch (color) {
  case 'red':   print('R'); break;
  case 'blue':  print('B'); break;
  default:      print('?');
}
```

Classes

Definição de Classe

```
class Point {
  final double x, y;
  Point(this.x, this.y);
  double distanceTo(Point p) =>
    sqrt(pow(x - p.x, 2) + pow(y - p.y, 2));
}
```

Construtores Nomeados e Factory

```
class Point {
  double x, y;
  Point(this.x, this.y);
  Point.origin() : x = 0, y = 0;
  factory Point.fromJson(Map j) =>
    Point(j['x'], j['y']);
}
```

Herança

```
class Animal { void speak() {} }
class Dog extends Animal {
  @override
  void speak() => print('Woof');
}
```

Mixins e Extensions

Mixins

```
mixin Flyable {
  void fly() => print('Flying');
}
class Bird with Flyable {}
```

Extension Methods

```
extension StringX on String {
  String capitalize() =>
    '${this[0].toUpperCase()}${substring(1)}';
}
print('hello'.capitalize()); // Hello
```

Abstract e Implements

```
abstract class Shape {
  double area();
}
class Circle implements Shape {
  final double r;
  Circle(this.r);
  @override double area() => pi * r * r;
}
```

Async/Await

Futures

```
Future<String> fetchData() async {
  var res = await http.get(url);
  return res.body;
}
```

Streams

```
Stream<int> count(int n) async* {
  for (var i = 0; i < n; i++) {
    yield i;
  }
}
```

Tratamento de Erros

```
try {
  var data = await fetchData();
} on HttpException catch (e) {
  print('HTTP error: $e');
} catch (e) {
  print('Error: $e');
}
```

Coleções

Operações em List

```
var nums = [1, 2, 3];
nums.add(4);
nums.where((n) => n > 2);     // [3, 4]
nums.map((n) => n * 2);        // [2,4,6,8]
var sorted = nums..sort();
```

Operações em Map

```
var m = {'a': 1, 'b': 2};
m['c'] = 3;
m.containsKey('a');  // true
m.entries.map((e) => '${e.key}=${e.value}');
```

Spread e Collection If/For

```
var all = [0, ...nums];
var nav = ['Home', if (isAdmin) 'Admin'];
var sq = [for (var i in nums) i * i];
```

Null Safety

Tipos Nullable

`int?`	int nullable (pode ser null)
`int`	int não-nullable (nunca null)
`!`	Operador de asserção nula
`?.`	Acesso null-aware
`??`	Padrão se null
`??=`	Atribuir se null
`late`	Inicialização diferida

Exemplos de Null Safety

```
String? name;            // nullable
int len = name?.length ?? 0;
late final String title; // definir antes de usar
name ??= 'default';      // atribuir se null
```

Padrões Comuns

Enum com Valores

```
enum Color {
  red('FF0000'), green('00FF00');
  final String hex;
  const Color(this.hex);
}
```

Records e Desestruturação

```
(String, int) userInfo() => ('Alice', 30);
var (name, age) = userInfo();
print('$name is $age');
```

Sealed Classes

```
sealed class Shape {}
class Circle extends Shape { final double r; Circle(this.r); }
class Rect extends Shape { final double w, h; Rect(this.w,
this.h); }
```