

# Référence rapide Dart

Types, fonctions, classes, async, null safety, collections

## Bases

### Hello World

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

### Variables

```
var name = 'Dart'; // type inféré  
String lang = 'Dart'; // type explicite  
final pi = 3.14; // constante d'exécution  
const max = 100; // constante de compilation
```

### Interpolation de chaînes

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

## Types

### Types intégrés

<b>int</b>	Entier 64 bits
<b>double</b>	Virgule flottante 64 bits
<b>num</b>	Supertype de int et double
<b>String</b>	Chaîne UTF-16
<b>bool</b>	true ou false
<b>List</b>	Collection ordonnée (tableau)
<b>Set</b>	Collection unique non ordonnée
<b>Map</b>	Paires clé-valeur
<b>dynamic</b>	Tout type, désactive la vérification statique
<b>void</b>	Pas de valeur de retour

### Vérifications de type et casts

```
if (obj is String) print(obj.length);  
var s = obj as String; // cast  
print(obj.runtimeType); // type à l'exécution
```

## Fonctions

### Syntaxe des fonctions

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

### Paramètres

<b>int f(int a)</b>	Paramètre positionnel requis
<b>int f([int a = 0])</b>	Positionnel optionnel avec valeur par défaut
<b>f({required int a})</b>	Paramètre nommé requis
<b>f({int a = 0})</b>	Nommé optionnel avec valeur par défaut

### Fermetures et tearoffs

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

## Flux de contrôle

### Conditions

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

## Boucles

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

### Switch et correspondance de motifs

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

## Classes

### Définition d'une 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));  
}
```

### Constructeurs nommés et 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']);  
}
```

### Héritage

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

## Mixins et extensions

### Mixins

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

### Méthodes d'extension

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

### Abstrait et 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(uri);  
  return res.body;  
}
```

### Streams

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

### Gestion des erreurs

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

## Collections

### Opérations sur les listes

```
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();
```

### Opérations sur les maps

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

### Spread et collection if/for

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

## Null safety

### Types nullables

<b>int?</b>	int nullable (peut être null)
<b>int</b>	int non nullable (jamais null)
<b>!</b>	Opérateur d'assertion non-null
<b>?.</b>	Accès null-aware
<b>??</b>	Valeur par défaut si null
<b>??=</b>	Assigner si null
<b>late</b>	Initialisation différée

### Exemples de null safety

```
String? name; // nullable  
int len = name?.length ?? 0;  
late final String title; // à définir avant usage  
name ??= 'default'; // assigner si null
```

# Référence rapide Dart

---

## Patterns courants

---

### Enum avec valeurs

---

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

### Records et destructuration

---

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

### Classes sealed

---

```
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); }
```