ECOLE DES MINES DE L'INDUSTRIE
ET DE LA GÉOLOGIE

Développement Mobile

2025-2026

Chapitre 3 : Programmation Dart

Flutter

1. Qu'est-ce que Dart?

Dart est un **langage orienté objet** très **puissant** et **polyvalent**, qui a été développé chez Google au départ pour combler les défauts du **JavaScript** et avait pour objectif de le remplacer en tant que langage phare du web.

Date de première version : 10 octobre 2011



- C'est un langauge de programmation open-source, basé sur des classes et éventuellement typé.
- Aujourd'hui avec le langage Dart et Flutter, nous pouvons créer des **applications mobiles**, des **sites web** et des **logiciels de bureau**.

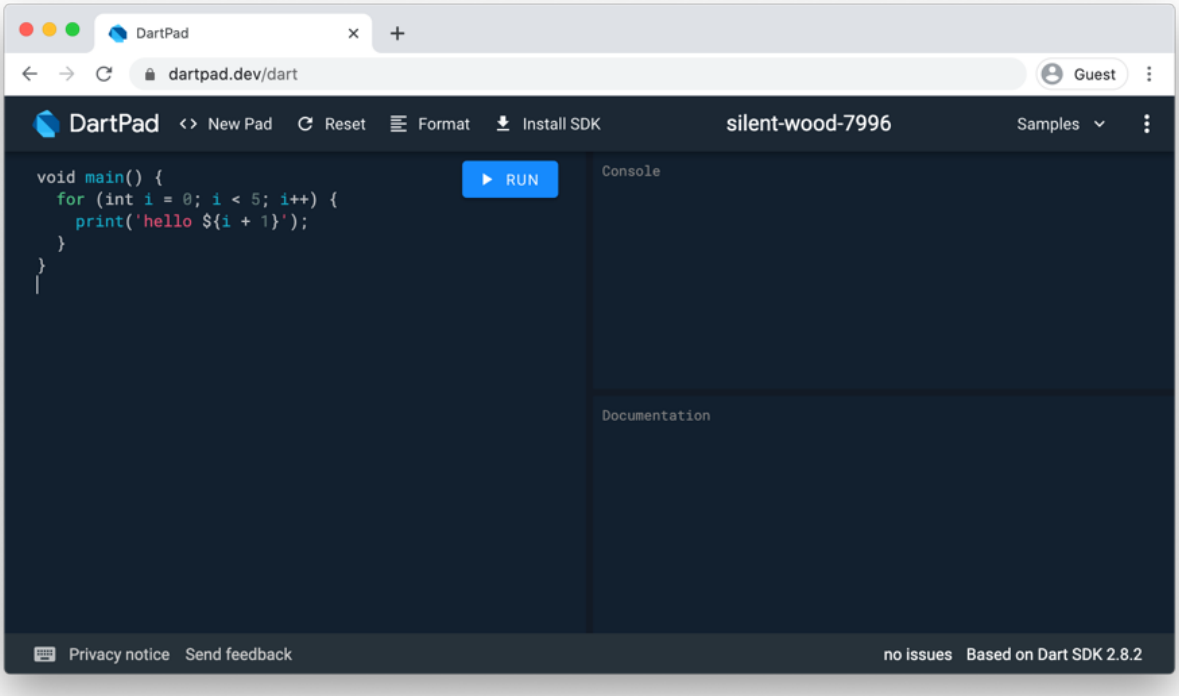
2. Notions de base Dart

- ❑ Environnement Dart
- ❑ Les variables en Dart
- ❑ Les lists et les maps
- ❑ Les fonctions
- ❑ Les structures de contrôle
- ❑ Les classes
- ❑ Les imports et les exports
- ❑ L'asynchrone
- ❑ Commentaires
- ❑ Les operateurs

3. Environnement Dart

Vous avez plusieurs moyens d'exécuter du Dart :

- ❑ Exécuter du Dart localement(via un IDE ou Editeur de code)
- ❑ Utiliser le Dart Pad(https://dartpad.dev/)



- ❑ En Dart tout est un objet, tout ce que vous placez dans une variable est un objet.
- ❑ Dart est un langage typé comme par exemple **Java ou Typescript**. Mais comme pour Typescript, les types peuvent être inférés (c'est-à-dire détectés automatiquement) dans certains cas.

✓ **Obtenir le SDK Dart**

- Documentation officielle : https://dart.dev/get-dart

✓ **Hello World**

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

4. Les variables

❑ Les variables stockent des références à des objets

❑ Dart supporte un certain nombre de types comme:

- Nombres,
- Les chaînes de caractères,
- Les booléens,
- Les collections comme les listes simples ou les listes fonctionnant avec un couple clé/valeur.

.

Var: permet de déclarer une variable sans déclarer son type.

```
var name = 'Voyager I';
var year = 1977;
var antennaDiameter = 3.7;
var flybyObjects = ['Jupiter', 'Saturn', 'Uranus', 'Neptune'];
var image = {
  'tags': ['saturn'],
  'url': '//path/to/saturn.jpg',
};
```

✓ Définir une variable typée

Nous pourrions cependant typer la variable directement. Pour cela il faut déclarer un des types Dart :

- **numbers,**
- **strings,**
- **booleans,**
- **lists,**
- **maps,**
- **Etc.**

```
main() {
        // Les deux types de nombres :
        int x = 100;
        double y = 22.15;
        // Les chaînes de caractères
        String s = 'Bonjour ! Je suis une chaîne de caractères';
        // Les booléens
        bool isValid = false;
}
```

✓ Interpolation de variable

Comme en JavaScript, vous pouvez interpoler des variables avec **$var ou ${var}** :

```
main() {
 // Interpolation
 String interpolation1 = 'La valeur de x est $x';
 String interpolation2 = 'La valeur de s en lowercase est ${s.toLowerCase()}';
 print(interpolation1);
 print(interpolation2);
}
```

✓ Concaténation de chaînes de caractères et les chaînes multilignes

```
main() {
        // Il existe deux moyens de concaténer des chaines de caractère
  // 1. séparées par des espaces ou des sauts de ligne:
 var s1 = 'La concaténation '
 'de chaînes de caractères'
 " fonctionnent avec des sauts de ligne et les deux types de guillements. \n";
 // 2. Le second est d'utiliser le signe + comme en JavaScript :
 var s2 = 'La concaténation' + 'en utilisant ' + "des plus. \n";
 // Il est possible d'écrire des chaînes de caractères sur plusieurs lignes
 // en utilisant des triples guillements '''
 var s3 = '''Une chaîne de caractères peut s'écrire sur plusieurs lignes sans aucun problème en Dart, grâce aux triples guillements'’’;
}
```

5. Les constantes

Il existe deux types de constantes: **const** et **final**

- ✓ Les **const** sont des constantes qui doivent etre définies au moment de la compilation.
- ✓ Les **final** sont des constantes, qui ne peuvent donc etre initialisées qu'une seule fois, mais qui peuvent etre assignées pendant l'exécution.

Exemple:

```
main() {
 // Deux moyens de déclarer des const :
 const pi = 3.14;
 const double pi2 = 3.13;
 // Ne compilera pas :
 pi = 24;
 // Même chose pour les finals :
 final x = 1;
 final int y = 2;
 // La différence entre les deux :
 const t = DateTime.now(); // ne compilera pas !
 final t = DateTime.now(); // valeur connue lors de l'exécution mais initialisation fixe
```

6. Les liste et les maps

Les lists : sont comme des tableaux en JavaScript.

- les lists commencent à l'index 0.
- Les lists ont une propriété length pour connaître la longueur de celle-ci(Comme en JavaScript).
- Le dernier élément d'une list est donc à l'index list.length - 1.

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

maListe.add('Hello !'); // Error: The argument type 'String'

// can't be assigned to the parameter type 'int'.
```

✓ Accéder à un élément d'une list

Pour accéder à un élément d'une list, il suffit de faire list[index].

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

list[1]; // 2
```

✓ L'opérateur spread (...)

Comme en JavaScript depuis ES6, Dart permet d'utiliser l'opérateur spread pour copier facilement une list dans une autre list:

```
main() {
 var maListe = [1,2,3];
 var maListe1 = [...maListe, 4, 5, 6];
print(maListe1);
}
```

✓ Quelques méthodes

Il existe environ 50 méthodes en Dart pour les lists.

add(), addAll(), clear(), indexOf(), contains(), remove() et removeAt().

```
main() {
 var list = [1,2,1];
 // Ajouter une valeur à la fin de la List :
 list.add(2);
 // Enlever la première occurence d'une valeur spécifique d'une List :
 // Retourne true si un élément a été enlevé, false sinon.
 list.remove(1);
 print(list);
 // Enlever l'élément à l'index spécifié :
 list.removeAt(0);
 print(list);
 // Enlever tous les éléments de la List :
 list.clear();
 print(list);
 // Ajouter tous les éléments passés dans une list à la fin de la List :
 list.addAll([1,2,3]);
 print(list);
 // Retourne l'index de l'élément :
 print(list.indexOf(1));
 // Retourne un booléen suivant que l'élément est dans la List ou non :
 print(list.contains(22));
}
```

7. Les maps

- ❑ Un map est un objet qui associe des clés et des valeurs.
- ❑ Les clés et les valeurs peuvent avoir tout type d'objet.
- ❑ Les clés doivent être uniques mais pas les valeurs.

✓ **Définir un map:**

```
var map = {
 'clé1': 'valeur1',
  'clé2': 'valeur2'
};
```

```
Map<String, String> map1 = {'clé1': 'valeur1'};
Map<int, String> map2 = {1: 'valeur1'};
Map<String, double> map3 = {'clé3': 45.22};
```

✓ **Accéder à un élément d'un map :**

```
Map<String, String> map1 = {'clé1': 'valeur1'};
Map<int, String> map2 = {1: 'valeur1'};
Map<String, double> map3 = {'clé3': 45.22};
```

✓ **La propriété length:** fonctionne de la même manière que pour les lists.

✓ **Quelques autres propriétés: values, keys et entries**

```
main() {
  var map1 = {
    'clé1': 'valeur1',
1: 'valeur2'
  };
  print(map1.entries);   // La propriété entries permet d'obternir les paires clé/valeur
  print(map1.values);   // La propriété values permet d'obtenir les valeurs
  print(map1.keys.runtimeType);   // La propriété keys permet d'obtenir les clés
}
```

✓ Opérateur spread

```
main() {
  var map1 = {
    'clé1': 'valeur1',
    1: 'valeur2'
  };
  var map2 = {
    ...map1,
    'clé3': 44.23
  };
  print(map2);
}
```

✓ Quelques méthodes

- containsKey,
- containsValue
- remove

```
main() {
  var map1 = {
    'clé1': 'valeur1',
    1: 'valeur2'
  };
  // containsKey() permet de savoir si une clé existe :
  print(map1.containsKey('clé1'));
  // containsValue() permet de savoir si une valeur existe :
  print(map1.containsValue(1));
  // remove() permet de supprimer une paire clé/valeur :
  map1.remove('clé1');
  print(map1);
}
```

8. Les fonctions

Comme le reste en Dart, les fonctions sont des objets.

Elles peuvent donc être assignées à des variables et passées comme arguments à d'autres fonctions.

En Dart il est recommandé de typer le retour des fonctions, par exemple :

```
void bonjour() {
  print('Bonjour');
}
bonjour();
```

8.1 Les fonctions fléchées

En Dart il existe aussi des fonctions fléchées qui fonctionnent exactement comme en JavaScript :=> expression équivaut à { return expr; }.

Par exemple :

```
void direBonjour() => print('Bonjour');
direBonjour();
```

8.2 Les paramètres

En Dart il vaut mieux également typer les paramètres.

```
main() {
  var nomComplet = getFullName('Ibrahim', 'Oumarou');
  print(nomComplet);
}
String getFullName(String prenom, String nom) {
  return "$prenom $nom";
}
```

8.3 Paramètres nommés et null safety

En Dart, il est également possible d'utiliser des paramètres nommés pour cela il suffit de faire lors de la définition :

```
nomFonction({param1, param2}) {}
```

Et lors de l'utilisation :

```
nomFonction(param1: val1, param2: val2);
```

Remarque: Les paramètres nommés sont optionnels sauf si vous utilisez required devant le nom du paramètre.

```
main() {
  var nomComplet = getFullName(nom: 'Oumarou', prenom: 'Ibrahim');
  print(nomComplet);
}
String getFullName({String? prenom, String? nom}) {
  return "$prenom $nom";
}
```

✓ Paramètres optionnels ou requis

En Dart, les paramètres peuvent être optionnels ou obligatoires. Par défaut les paramètres positionnels sont obligatoires :

```
main() {
  var nomComplet = getFullName('Ibrahim'); // Error: Too few positional arguments:
  // 2 required, 1 given.
  print(nomComplet);
}
String getFullName(String prenom, String nom) {
```

✓ **Pour rendre un paramètre optionnel il faut utiliser [param]** :

```
main() {
  var nomComplet = getFullName('Jean');
  print(nomComplet);
}
String getFullName(String prenom, [String? nom]) {
  return "$prenom $nom"; // Jean null
}
```

✓ **Paramètres par défaut**

Il est également possible de définir la valeur par défaut des paramètres en Dart.

```
main() {
  var nomComplet = getFullName();
  print(nomComplet);
}
// Fonction avec deux paramètres optionnels et dont
// le premier paramère a une valeur par défaut :
String getFullName([String prenom = 'Inconnu', String? nom]) {
  return "$prenom $nom";
}
```

✓ **Même chose avec les paramètres nommés :**

```
main() {
  var nomComplet = getFullName(prenom: 'Jean');
  print(nomComplet);
}
String getFullName({String? prenom, String nom = 'Inconnu'}) {
  return "$prenom $nom"; // Jean Inconnu
}
```

8.4 Les fonctions anonymes

Comme en JavaScript, il est possible d'utiliser des fonctions anonymes appelées également lambdas :

```
([[Type] param1[, …]]) {
 codeBlock;
};
```

```
main() {
 List<int> list = [1,2,3,4];

 list.forEach((item) {
  print(item);
 });
  // Ou encore avec une fonction fléchée anonyme :
 list.forEach((item) => print(item));
}
```

8.5 Passer des fonctions comme argument á d'autres fonctions

Comme en JavaScript, les fonctions peuvent être passées en argument à d'autre fonction.

```
main() {
 List<int> list = [1,2,3,4];
 list.forEach(print);
}
```

```
main() {
 List<String> list = ['je', 'suis', 'en', 'lowercase'];
 var listUpper = list.map(toUpperCase);
 print(listUpper);
}
String toUpperCase(String elem) => elem.toUpperCase();
```

9. Les structures de contrôle

9.1 Les blocs conditionnels

Comme dans la plupart des langages nous pouvons utiliser if, else et else if :

```
main() {

        bool test1 = false;
        bool test2 = true;

        if (test1) {
         print('Par là !');
        } else if (test2) {
         print('Ici !');
        } else {
         print('Là !');
        }

}
```

9.2 L'opérateur ternaire

```
main() {
  int age = 17;
  bool isMajeur = age > 18 ? true : false;
  print(isMajeur);
}
```

9.3 Les boucles

✓ **Les boucles for**

```
void main() {
  for (int i = 0; i < 5; i++) {
    print('Hello ${i + 1}');
  }
}
```

✓ **Les boucles forEach**

```
void main() {
  List<int> liste = [1, 2, 3];
  int result = 0;
  liste.forEach((elem) => result = result + elem);
  print(result);
}
```

✓ **for … in:**

```
void main() {
  List<int> liste = [1, 2, 3];
  int result = 0;
  for (var e in liste) {
    result = result + e;
  }
  print(result);
}
```

✓ **Do / while**

```
void main() {
  List<int> liste = [1, 2, 3];
  do {
    print(liste);
    liste.removeLast();
  } while (liste.length != 0);

  print(liste);
}
```

9.4 Utilisation de break et continue

- **break:** permet d'arrêter la boucle si une condition survient.
- **continue:** permet de sauter une itération suivant une condition.

```
void main() {
  List<int> liste = [1, 2, 3];
  while (liste.length != 0) {
    // Si la liste n'a plus qu'un élément nous stoppons la boucle :
    if (liste.length == 1) break;
    print(liste);
   liste.removeLast();
  }
  for (var i = 2; i < 10; i++) {
    // Nous sautons l'itération si i est pair :
    if (i % 2 == 0) continue;
    // Donc seuls les éléments impairs sont ajoutés :
    liste.add(i);
  }
}
```

9.5 Les switch / case

```
void main() {
        const int a = 3;
        switch (a) {
                case 1:
                 print('Ici');
                 break;
                case 2:
                 print('là');
                 break;
                default:
                 print('LA !');
                 break;
        }
}
```

10. Les classes

10.1 Généralités sur les classes en Dart

- En Dart tous les objets sont les instances d'une classe, et toutes les classes descendent de la classe Object.

- En Dart, toutes les classes ont exactement une classe parente, appelée superclass, sauf la classe Object.

✓ Créer une classe

```
class Point {
}
```

✓ Pour obtenir une instance d'une classe il suffit de faire :

```
var instance1 = new Point();
// ou car new est optionnel en Dart :
var instance2 = Point();
```

✓ Le contenu des classes

Les classes peuvent contenir des fonctions, appelées méthodes, et des données, appelées variables d'instance.

```
class Point {
  // Déclaration d'une variable d'instance x de type num et initialisation à null :
  num? x;
  // Déclaration d'une variable d'instance y de type num et initialisation à 0 :
  num y = 0;
}
```

10.2 Les constructeurs

En Dart les constructeurs de base ont le même nom que la classe auxquels ils appartiennent et ce sont de simples fonctions :

```
class Point {
  num? x, y;
  Point(num x, num y) {
    this.x = x;
    this.y = y;
  }
}
```

```
class Point {
  num? x, y;
  Point(this.x, this.y);
}
```

10.3 Les méthodes d'instance

```
void main() {
  // Nous créons deux instances de la classe Point :
  Point point1 = Point(10, 2);
  Point point2 = Point(22, 11);
  num distance = point1.distanceTo(point2);
  print(distance);
}
class Point {
  num x, y;
  Point(this.x, this.y);
  num distanceTo(Point autrePoint) {
    num dx = x - autrePoint.x;
    num dy = y - autrePoint.y;
    return sqrt(dx * dx + dy * dy);
  }
}
```

10.4 Les getters et les setters

Les getters et les setters sont des méthodes spéciales qui permettent de définir comment des propriétés d'instance sont lus et / ou écrites.

Par défaut les propriétés d'instance ont un getter et un setter implicites :

11. Les imports et les exports

11.1 Importer une librairie

- Importer une librairie la rend accessible dans le fichier où elle est importée.

- Pour importer une librairie il suffit de faire, par exemple pour une librairie native :

```
import 'dart:math'
```

- Une fois une librairie importée vous pouvez utiliser toutes ses fonctions :

```
import 'dart:math';
 void main() {
   print("La racine carrée de 16 est : ${sqrt(16)}");
 }
```

11.2 Exporter une librairie

- Vous pouvez exporter une librairie depuis une librairie en utilisant export.
- Cela peut être utile pour créer des index par exemple :

```
// soit un fichier tests.dart
export 'tests/test1.dart';
export 'tests/test2.dart';
export 'tests/test3.dart';
```

```
import 'tests.dart';
```

12. L'asynchrone

- Dart propose de nombreuses fonctionnalités permettant de gérer l'asynchrone facilement.
- Les fonctions asynchrones retourne un résultat après une période de temps nécessaire à une opération asynchrone : par exemple une requête HTTP.

12.1 Utilisation de async / await

L'utilisation est la même qu'en JavaScript / TypeScript, à savoir qu'il n'est possible d'utiliser await que dans une fonction async.

```
Future attendreActionAsynchrone() async {
  valeur = await actionAsynchrone();
  // faire quelque chose avec valeur
}
```

12.2 Gestion d'erreur

De même qu'en JavaScript, la gestion d'erreur des fonctions asynchrones se fait avec des blocs try / catch, et éventuellement avec finally

```
try {
  valeur = await actionAsynchrone();
} catch (e) {
  print(e);
}
```

13. Les commentaires

```
int i = 0; // Commented out text
```

```
void main() {
for (int i = 0; i < 5; i++) {
/* This is commented, and
will not affect code */
print('hello ${i + 1}');
}
}
```

14. Les opérateurs

Les opérateurs permettent de manipuler de façon mathématique des valeurs souvent par le biais de la variable à laquelle elles sont assignées.

En Dart, il existe plusieurs types d'opérateurs.

- Opérateurs arithmétiques
- Égalité et opérateurs relationnels
- Expressions conditionnelles
- Opérateurs logiques
- Opérateurs de test de type
- Opérateurs d'assignation
- Opérateurs de bits et de décalage

14.1 Les opérateurs arithmétiques

Les opérateurs arithmétiques sont utilisés pour faire des opérations arithmétiques sur des valeurs.

Operateurs	Explications
+	Addition
-	Soustraction
*	Multiplication
/	Division
~/	Division qui retourne comme résultat un entier
%	modulo (retourne le reste de la division)
++variable	variable = variable + 1 (pré incrémentation)
variable++	variable = variable + 1 (post incrémentation)
--variable	variable = variable - 1 (pré décrémentation)
variable--	variable = variable - 1 (post décrémentation)

```
var x = 12;
var y = 25;
int sum  = x + y;
int sous = x - y;
double pourcentage = (x / y) * 100;
```

14.2 Égalité et opérateurs relationnels

Les opérateurs relationnels et d'égalité comparent leur premier opérande à leur second opérande pour tester la validité de la relation spécifiée. Le résultat d'une expression relationnelle.

Opérateur	Relation testée
<	Premier opérande inférieur au second opérande
>	Premier opérande supérieur au second opérande
<=	Premier opérande inférieur ou égal au second opérande
>=	Premier opérande supérieur ou égal au second opérande
==	Premier opérande égal au second opérande
!=	Premier opérande différent du second opérande

```
int ageRequis = 24 ;
int monAge = 19 ;
print(monAge >= ageRequis); //false
```

14.3 Expressions conditionnelles

Dart a deux opérateurs qui vous permettent d'évaluer de manière concise des expressions qui pourraient sinon nécessiter des instructions:

- Condition ? expression1 : expression2

(opérateur ternaire) Si la condition est vraie, évalue expression1 (et renvoie sa valeur); sinon, évalue et renvoie la valeur de expression2.

Exemple :

```
bool reussi = false;
int nombreDeParticipant = 20 ;
int nombre = ( reussi == true ) ? ( ++nombreDeParticipant ) : ( --nombreDeParticipant );
print(nombre); //19
```

- expression1 ?? expression2

  Si expression1 est non nul, retourne sa valeur; sinon, évalue et renvoie la valeur de expression2

Exemple :

```
var nom = null;
var username = nom ?? 'Utilisateur X';
print(username); // Utilisateur X
```

14.4 Opérateurs logiques

Les opérateurs logiques effectuent des opérations de ET logique && , de OU logique || et d'inversion ! généralement sur des expressions booléennes.

x	y	x && y	x || y	!x	!y
true	true	true	true	false	false
true	false	false	true	false	true
false	true	false	true	true	false
false	false	false	false	true	true

```
int x = 100;
int y = 103;
var resultat = !(( x < 123 ) && (y > 123));
print(resultat); // true
```

x < 123 est vrai donc le résultat est true

y > 123 est faux donc le résultat est false

(x < 123) && (y > 123) soit true et false ce qui donne false

! ((x < 123) && (y > 123)) soit inversion de false

ce qui donne comme résultat true

14.5 Opérateurs de test de type

En Dart, les opérateurs as, is et is! sont utilisés pour tester le type d'une variable. Si la condition est exacte, elle retourne true sinon false est retourné.

Opérateurs	Explications
as	utilisé comme opérateur de type.
is	première expression est du même type que la deuxième expression(objet)
is!	première expression n'est du même type que la deuxième expression(objet)

```
var x = 12;
var y = 'OK';
print(x is int); //true
print(y is String); //true
print(x is double); //false
```

14.6 Opérateurs d'assignation

Les operateurs d'assignation permettent ,comme leur nom l'indique, d'assigner une valeur à une variable.

Les plus communs sont :

Operateurs	Significations
=	assigne une valeur
a –= 1	a = a - 1
a += 1	a = a + 1
a *= 2	a = a * 2
a /= 2	a = a / 2
a %= 2	a = a % 2

```
double x = 12;
print(x –= 1); //11.0
print(x += 1); //12.0
print(x *= 2); //24.0
print(x /= 2); //12.0
print(x %= 2); //0.0
```