# BUT2 RA Développement mobile :

Sommaire :


- Présentation de Flutter
- Lancer un programme dans un terminal
- Types et variables
- Chaines et listes
- Fonctions et paramètres
- Opérateurs et contrôles.
- Installation

# Présentation de Flutter

## Présentation Flutter

- Flutter est un Framework open-source créé par Google pour créer des applications multi-plateformes
- A partir d'un code unique écrit en Dart
- Flutter :
    - 1ère version preview mai 2017 (cibles : iOS, Android)
    - Flutter 1.0 décembre 2018
    - Flutter 1.5 05/2019 : apps web et desktop en alpha
    - Flutter 2.0 mars 2021 : web et desktop en stable, Dart 2.12
    - Flutter 2.10 février 2022 : windows stable
    - Flutter 3.0 mai 2022 : Dart 2.18

## Présentation Dart

Dart est un langage à objets, issu de Google :

- 2011 : pour remplacer JS

- rapidement le but devient d'écrire du multi-plateformes
- 2013 v1.0
- 2015 v1.9 : compilation vers JS
- 2018 v2.0 : amélioration de la sûreté du typage
- 11/2019 v2.6 : dart2native
- mars 2021 : v2.12 null safety
- 08/2022 : v2.18

Attention au code venant du net, s'il est pré-v2.12, vous allez devoir l'adapter !

## Flutter à l'IUT

- Installé en salles machines sous LINUX
- IDE utilisé : vs code
- installation sur machine personnelle assez longue...

# Lancer un programme dans un terminal

Pour ouvrir le terminal de VSCode :

- Ctrl + J (ou CMD + J sur Mac)

ou

- Dans la barre de menu en haut : View > Terminal

Cette commande sera utilisée dans ce cours pour demander à l'utilisateur d'écrire dans le terminal :

```
String? chaine = stdin.readLineSync();
```

Pour lancer le programme depuis VSCode :

- À gauche, cliquez sur la 4ème icone "Run and Debug"
- Puis cliquez sur le bouton "Run and Debug". Un fichier .vscode/launch.json est créé automatiquement
- Remplacer ce qui a été généré par les lignes suivantes :

```
/// .vscode/launch.json
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Terminal",
            "program": "my_file.dart", // <=== NOM DU FICHIER A EXECUTER
            "request": "launch",
            "type": "dart",
            "console": "terminal"
        },
    ]
}
```

# Types et variables

## Dart : types et variables (1)

- il est conseillé de profiter du typage au maximum, afin de voir les erreurs pendant que l'on code, et non quand le programme tourne.
- on favorisera donc `int, String, bool, etc...` plutot que `var` et `dynamic`.

```
void main() {

  // Typage explicite
  int x = 25;
  double y = 4.2;
  String s = "Hello!"; String s2='bjr';
  bool vf = true;

  // Typage implicite
  var xx = 25;
  xx = "toto"; // Error: A value of type 'String' can't be assigned to a variable
of type 'int'.
```

```
    // Typage dynamique  : type "dynamic"
    var dynamique; // Warning: Prefer typing uninitialized variables and fields
    dynamique="xxx";
    dynamique=38;

    // équivalent !
    dynamic dyn2 = 8;
    dyn2 = "toto";
  }
```

# Dart : types et variables (2)

- `main()` est le point d'entrée obligatoire
- toute variable est un objet, hérite de `Object`

```
void main() {

  int x = 25;

  //print("La valeur de x est " + x); NON !!!
  print("La valeur de x est " + x.toString());
  print("La valeur de x est $x");
}
```

- `print` : interdit de mélanger les types
- `print` avec conversion en chaîne obligatoire ou interpolation de variable `$maVariable`

# Dart : types et variables (3)

- typage dynamique en fonction du contenu
- `runtimeType` utilisable sur tout objet (`int`, `bool`, `String`, etc.)

```
void main() {

  var obj = {
    'nom': "Frölisch",
    'prenom': "Stefan",
    'age': 25
  };

  print("$obj ${obj.runtimeType}");
    // {nom: Frölisch, prenom: Stefan, dossard: 25} JsLinkedHashMap<String,
Object>

  Object obj2 = {
    'nom': "Frölisch",
    'prenom': "Stefan",
    'nationalité': "de"
  };
```

```
  print("$obj2 ${obj2.runtimeType}");
   //{nom: Frölisch, prenom: Stefan, nationalité: de} JsLinkedHashMap<String,
String>
}
```

# Dart : types et variables (4)

- Portée des variables au niveau du bloc
- les blocs englobés connaissent les variables des blocs englobants

```
int x = 72;

void main() {

  int y=25;
  print(x + y); // main connaît x car déclaré dans le bloc parent

  if (x+y > 100) {
      bool supp = true;
      print(supp);
  }
  print(supp); // supp : variable inconnue
}
```

# Dart : types et variables (5)

- Constantes `final`

```dart
void main() {

  // constante initialisée
  final pi = 3.14159;

  // constante initialisée en deux temps
  final double piSimple;

  piSimple = 3.14;

  // 2nde initialisation interdite dans les deux cas !!!
  pi = 5; // ERROR
  piSimple = 12; // ERROR
}
```

# Dart : types et variables (6)

- Constantes `const`

```
void main() {

  // constante initialisée dès la compilation
  const pi = 3.14;

  // constante initialisée en deux temps
  const int x; // ERROR
  x = 100; // ERROR

  const debut = DateTime.now(); // ERROR : valeur inconnue au moment de la
compilation
  final okdebut = DateTime.now();
```

- `const` et `final` déclarent des constantes, mais `const` est plus restrictif :
  - déclaration et initialisation à faire en une seule instruction
  - la valeur est connue dès la compilation
  - très utilisée dans les interfaces graphiques, pour accélérer le rendu

# Dart : types et variables (7)

- Sûreté du langage : *null safety*
- Par défaut, une variable typée ne peut avoir la valeur `null` ou `undefined`
- Permet de voir les erreurs pendant que l'on code. VSCode affichera une alerte si l'on oublie de traiter un cas possiblement null.

```
void main() {

  int x;  // <== N'autorise pas null
  print(x);
  // The non-nullable local variable 'x' must be assigned before it can be used.

  int? y;  // <== Autorise null avec "?"
  print(y);
  // null
}
```

# Chaines et listes

## Dart : les chaînes (1)

- Instances de `String`
- immuables :
    - pas de modification
    - toute méthode de `String` qui produit une `String` crée une nouvelle instance

```dart
void main() {

  String s1="Bonjour";
  print(s1.length); // 7
  String s2=s1.toUpperCase();
  String s3="BONJOUR";
  print(s1==s2); // false
  print(s2==s3); // true

}
```

# Dart : les chaînes (2)

```
print(s2.substring(2,4)); // NJ
print(s2[2]); // N
s2[2]= 'X'; // ERROR

print(s2.indexOf('O')); // 1
print(s2.lastIndexOf('O')); // 4

// la concaténation est déconseillée
String s4 = s2 + ' ' + s3; // BONJOUR BONJOUR

// l'interpolation est prérérable
String s5 = "$s2 $s3"; // BONJOUR BONJOUR

print(s2.split('')); // [B, O, N, J, O, U, R]
print(s2.split('').reversed.join('')); // RUOJNOB
```

# Dart : tableaux/listes

- Pas de distinction en dart

```dart
void main() {

/// Rappel : l'utilisation de `var` est déconseillé.
 var tab = ["le", "la", "les"]; // Type : List<String>
 for (int i=0; i < tab.length; i++) {
   print(tab[i]);
 }

 List<int> tab2 = [5, 6, 7];
 for (int nb in tab2) {
   print(nb);
 }
```

# Dart : tableaux/listes

- Possibilité d'utiliser la programmation fonctionnelle (`map`, `forEach`, `sort`, `reduce`, `any`, `every`, etc.)

```dart
void main() {
 List<String> tab = ["le", "la", "les"];

 tab.add("l'"); // ["le", "la", "les", "l"];
 //tab.add(25); // ERROR

 tab[0]="lu"; // Uniquement si tab[0] existe ; add sinon

 //print(tab[10]); // Error: RangeError (index): Index out of range: index should
be less than 4: 10

 tab.remove("lu"); // la 1ère occurence de 'lu' est supprimée
 tab.removeAt(0);
 tab.insert(1, "lo"); // [les, lo, l]
 print(tab.indexOf("l'")); // 2
 print(tab.contains("les")); // true
}
```

# Fonctions et paramètres

## Dart : fonctions (1)

- Le passage des paramètres se fait uniquement par valeurs
- Fonctions avec paramètres positionnés

```
void main() {
  int idx = recherche([5, 6, 7], 5);
  print(idx);
}

int recherche(List<int> tab, int elt) {
  return tab.indexOf(elt);
}
```

- tous les paramètres positionnés sont **obligatoires** et ne peuvent être `null`

# Dart : fonctions (2)

- Accepter les paramètres positionnés `null` (mais toujours obligatoires !)

```dart
void affiche(String? message) {
  if (message == null) {
    print('Le message est null');;
  }
  print(message);
}

void main() {
  affiche(null);
  affiche("Le ciel est bleu");
}
```

# Dart : fonctions (3)

- Paramètres positionnés optionnels avec valeur par défaut : crochets

```
void affiche([String message = "bleu"]) {
  print("Le ciel est $message");
}

void main() {
  affiche();
  affiche("au dessus de nous");
}
```

# Dart : fonctions (4)

- Paramètres positionnés optionnels sans valeur par défaut : crochets et ?

```dart
void affiche(String s1, [String? s2]) {
  print(s1);
  if (s2 != null) {
    print(s2);
  }
}

void main() {
  affiche("bjr");
  affiche("bjr", "hello");
}
```

- Les paramètres optionnels sont positionnés en fin. Ce qui suit est impossible :

```dart
void affiche(String s1, [String? s2], String s3) { ... } // Impossible
```

# Dart : fonctions (5)

- Paramètres nommés : entre accolades

```dart
int recherche({required List<int> tableau, required int element}) {
  return tableau.indexOf(element);
}

void main() {
  int idx = recherche(tableau:[5, 6, 7], element: 5);
  print(idx);
}
```

- Les paramètres nommés sont optionnels par défaut, `required` pour les rendre obligatoires

# Dart : fonctions (6)

- Paramètres nommés non obligatoires

```dart
void affiche({String? message}) {
  String debutPhrase = "le ciel est ";
  if(message == null){
    print("$debutPhrase indescriptible");
  } else {
    print("$debutPhrase $message");
  }
}

void main() {
  affiche();
  affiche(message:"parfois orange");
}
```

# Dart : fonctions (7)

- Paramètres nommés avec valeur par défaut
- Il faut priviliger le "=" car le ":" sera supprimé dans la prochaine version de Dart (dart 3)

```dart
void affiche({String? message = "bleu"}) {
  print("Le ciel est $message");
}

void main() {
  affiche(message: null);
  affiche();
  affiche(message: "couvert de nuages");
}
```

# Dart : fonctions (8)

- Mélange des paramètres positionnés et nommés

```dart
void affiche(String msg1, {String? msg2, String msg3 = "hola"}) {
  print(msg1);
  if (msg2 != null) {
    print(msg2);
  }
  print(msg3);
  print("-----");
}

void main() {
  affiche("bjr");
  affiche("bjr", msg2: "hello");
}
```

- D'abord les paramètres positionnés, puis les nommés

# Dart : fonctions (9)

- *Arrow functions* s'il n'y a qu'une seule instruction

```dart
// Sans retour
void affiche(String msg1) => print(msg1);

// Avec retour
String completeLaPhrase(String msg1) => "Le ciel est ${msg1.toUpperCase()}";

void main() {
  affiche("bjr");

  String phrase = completeLaPhrase('grand');
  print(phrase);
}
```

# Opérateurs et contrôles.

## Dart : opérateurs, structures de contrôle, etc. (1)

- cf C/C++/Java/(JS) `==` `!=` `++` `+=` `&&` `||` `!`
- Les ternaires sont très utilisées en Flutter

```
//
if (condition) {
  ...
} else if {autre} {
  ...
} else { ... }

//
switch (mode) {
  case 1:
    print("xxx");
    break;
  case 2:
```

```
        print("yyy");
        break;
      default:
        print("zzz");
    }

    // Ternaire
    int taille = (s=='F' ? 170 : 180);
```

# Dart : opérateurs, structures de contrôle, etc. (2)

```
while(i < 10 && !trouve) {
  i++;
}

do {
  ...
} while (!ok);
```

```
List<String> tab=["X","Y","Z"];

for(int i=0; i < tab.length; i++) {print(tab[i]);}

for(String c in tab) {print(c);}
```

# Dart : opérateur null safety

```dart
void affiche({String? couleur}) {

  int hauteur = couleur == null ? 0 : 42;
  print(hauteur);

  String couleurDuCiel = couleur ?? 'orange';
  print('Le ciel est $couleurDuCiel');

  couleur ??= 'vert';
  print('Maintenant, il est $couleur');

  print("----------");
}

void main() {
  affiche(couleur : 'bleu');
  affiche();
}
```

# Dart : break, return et continue

- Chacun va permettre d'aller plus ou moins loin dans l'execution de la fonction

```dart
void affiche(List<String> maListe) {
  for(String text in maListe) {
    print('text $text');
    if (text == 'trois'){
      print('BREAK'); // <============== ou 'RETURN' ou 'CONTINUE';
      break; // <======================= ou return; ou continue;
    }
    print('Après if');
  }
  print('Après for');
}

void main() {
  final mesChiffres = <String>['un', 'deux', 'trois', 'quatre', 'cinq'];
  affiche(mesChiffres); // Résultat page suivane
}
```

Résultats de la page précédante :

| BREAK | RETURN | CONTINUE |
|---|---|---|
| text un | text un | text un |
| Après if | Après if | Après if |
| text deux | text deux | text deux |
| Après if | Après if | Après if |
| text trois | text trois | text trois |

| BREAK | RETURN | CONTINUE |
|---|---|---|
| Après for | | text quatre |
| | | Après if |
| | | text cinq |
| | | Après if |
| | | Après for |

# Installation

## Installation (1)

- Suivre https://docs.flutter.dev/get-started/install
- Procédure un peu longue, en plusieurs étapes
- flutter : rapide (inclut dart)
- Android Studio : long
- SDK Android : looooong

## Installation (2)

En bref :

- installer Flutter
- modifier votre variable d'environnement PATH pour inclure le dossier d'installation de flutter
- cmd/terminal : `flutter doctor`

```
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 2.8.1, on Ubuntu 20.04.3 LTS 5.4.0-90-generic, locale fr_FR.UTF-8)
[✗] Android toolchain - develop for Android devices
    ✗ Unable to locate Android SDK.
      Install Android Studio from: https://developer.android.com/studio/index.html
      On first launch it will assist you in installing the Android SDK components.
      (or visit https://flutter.dev/docs/get-started/install/linux#android-setup for detailed instructions).
      If the Android SDK has been installed to a custom location, please use
      `flutter config --android-sdk` to update to that location.

[✓] Chrome - develop for the web
[!] Android Studio (not installed)
[✓] VS Code
[✓] Connected device (1 available)

! Doctor found issues in 2 categories.
```

---

## Installation (3)

- [Linux, MacOS] si chrome n'est pas trouvé : `export CHROME_EXECUTABLE=path/to/chrome`
- installer Android Studio https://developer.android.com/studio
- lancer Android studio : (wizard : next, next,... Finish) → un SDK Android est installé

---

## Installation (4)

- `flutter doctor`

```
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 2.8.1, on Ubuntu 20.04.3 LTS 5.4.0-90-generic, locale fr_FR.UTF-8)
[!] Android toolchain - develop for Android devices (Android SDK version 32.0.0)
    ✗ cmdline-tools component is missing
      Run `path/to/sdkmanager --install "cmdline-tools;latest"`
      See https://developer.android.com/studio/command-line for more details.
    ✗ Android license status unknown.
      Run `flutter doctor --android-licenses` to accept the SDK licenses.
      See https://flutter.dev/docs/get-started/install/linux#android-setup for more details.
[✓] Chrome - develop for the web
[✓] Android Studio (version 2020.3)
[✓] VS Code
[✓] Connected device (1 available)

! Doctor found issues in 1 category.
```

- sdkmanager est dans un sous-dossier du dossier d'install du sdk Android, par exemple : ~/android-sdks/tools/bin (on peut aussi l'installer en utilisant android studio)
- `flutter doctor --android-licenses` : tout accepter
- `flutter doctor` : tout doit être au vert

---

## Installation (5)

- lancer VSCode
- extensions :
    - Flutter (3.56.0), 5,4M installations

- Dart (3.60.1), 6,3M installations
- (conseillé) Error Lens
- (conseillé) mise en évidence des parenthèses

```
Command Palette -> Preference: Open Settings (JSON)
{
    "editor.bracketPairColorization.enabled": true,
    "editor.guides.bracketPairs":"active",
}
```