
Atelier

Le langage de programmation Dart



Dart est un langage de programmation open source développé par Google et qui est apparu en 2011. Il est utilisé pour le développement d'applications mobiles, Web, de bureau et d'autres applications intégrant la technologie Internet des objets (IoT).

Dart est un langage de programmation orienté objet (OOP) avec une syntaxe similaire à C++, Java et Javascript.

Grâce à Dart, Flutter est devenu le Framework multiplate-forme le plus rapide qui offre des performances de type natif pour Android et iOS. En fait, un langage natif, communique directement avec les composants d'un appareil, sans interprètes intermédiaires, ce qui lui confère une très grande vitesse.

Dart est également un langage de programmation dynamique. La machine virtuelle Dart offre la possibilité d'exécuter du code directement sans avoir besoin de le compiler au préalable.

Editeur en ligne Dart :

<https://dartpad.dev/>

I. main

```
main()  
  
{  
  
}
```

main() – C'est le début de notre programme à partir duquel il sera exécuté. Il s'agit d'une fonction spéciale, obligatoire, de niveau supérieur où l'exécution du programme commence. Ajoutons quelque chose à l'intérieur de main().

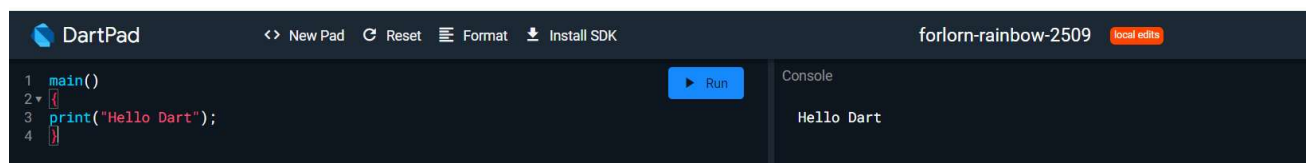
```
main()

{

print("Hello Dart");

}
```

print() est une fonction utilisée pour afficher le contenu entre guillemets doubles à l'écran.



II. Les variables

L'écriture d'une variables Dart se fait en spécifiant d'abord le type des données, suivi du nom de la variable, du signe égal = et de la valeur initiale de la variable. L'écriture des noms de variables Dart se fait en utilisant la convention de dénomination camelCase, par exemple monPrénom.

Un type de données est une classification de données qui indique au compilateur ou à l'interpréteur comment le programmeur a l'intention d'utiliser les données. La plupart des langages de programmation prennent en charge une variété de types de données, notamment les nombres, les chaînes de caractères et les booléens.

Les types de données de base dans Dart sont divisés en trois types :

Type de données Nombre : int (entier), double (à virgule)

Type de données texte : String

Et le type de données booléen : bool (true, false)

De plus, Dart possède d'autres types de données. Tels que les listes, les set (ensembles), les map, les runes et les symboles.

Si nous créons une variable Dart en utilisant le mot-clé **var**, la variable sera affectée à n'importe quel type de données.

Cependant, si nous créons une variable avec un type de donnée, la variable ne peut être assignée qu'à une valeur qui correspond au type de donnée spécifié.

```
main()
```

```

{

// créer une variable nombre de type int

var nombre = 124;

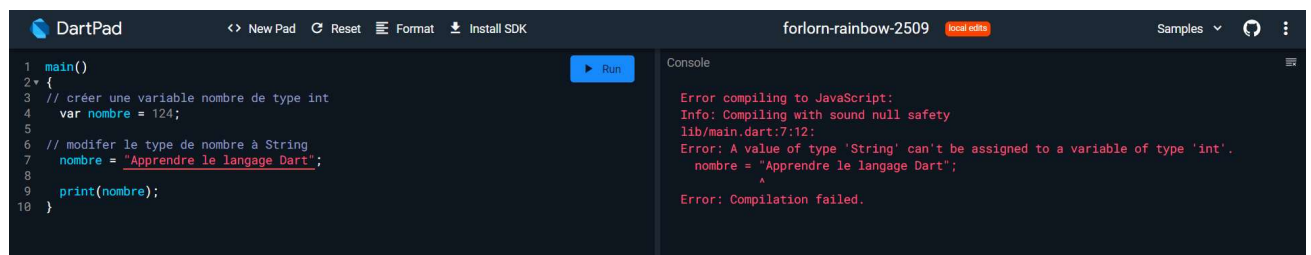

// modifier le type de nombre à String

nombre = "Apprendre le langage Dart";


print(nombre);

}

```



Si vous affectez une variable lorsque vous la définissez, Dart fixe le type de la variable. Dans ce cas, la variable nombre sera de type int, et si vous voudrez lui affecter une valeur de type String, vous aurez une erreur.

Cependant, si vous n'affectez pas la variable lorsque vous la définissez, la variable sera de type **Dynamic** et vous pouvez lui attribuer n'importe quel type.

```

main()

{

// créer une variable non définie

var apprendre;

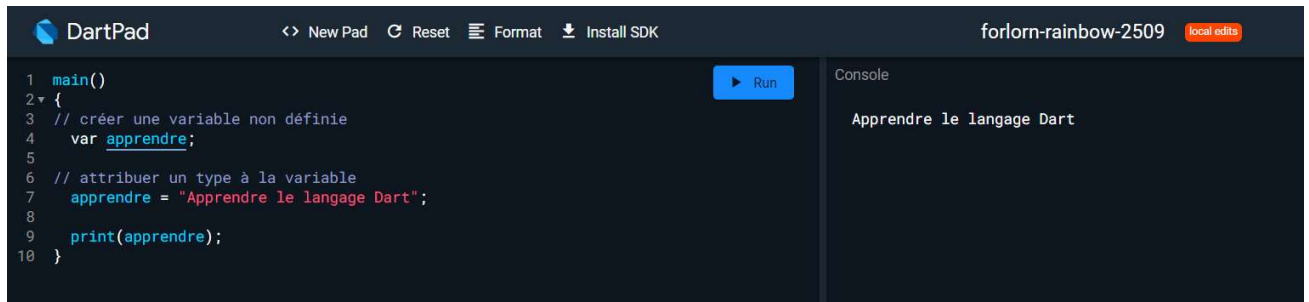

// attribuer un type à la variable

apprendre = "Apprendre le langage Dart";

```

```
print(apprendre);
```

```
}
```



The screenshot shows the DartPad web interface. The top bar includes the DartPad logo, navigation links (New Pad, Reset, Format, Install SDK), and a user profile section (forlorn-rainbow-2509, local edits). The main editor area contains the following Dart code:

```
1 main()
2 {
3   // créer une variable non définie
4   var apprendre;
5
6   // attribuer un type à la variable
7   apprendre = "Apprendre le langage Dart";
8
9   print(apprendre);
10 }
```

A blue "Run" button is visible next to the code. To the right, the "Console" panel displays the output: "Apprendre le langage Dart".

```
main()
```

```
{
```

```
// créer une variable non définie
```

```
var apprendre;
```

```
// attribuer un type à la variable
```

```
apprendre = "Apprendre le langage Dart";
```

```
print(apprendre);
```

```
// attribuer un autre type à la variable
```

```
apprendre = 10;
```

```
print(apprendre);
```

```
}
```



```
1 main()
2 {
3   // créer une variable non définie
4   var apprendre;
5
6   // attribuer un type à la variable
7   apprendre = "Apprendre le langage Dart";
8
9   print(apprendre);
10
11  // attribuer un autre type à la variable
12  apprendre = 10;
13
14  print(apprendre);
15 }
```

Console

```
Apprendre le langage Dart
10
```

Lorsque nous utilisons l'inférence de type, le type de données de la variable est déterminé par la valeur initiale que nous lui transmettons. Cependant, si nous modifions la valeur de cette variable avec un type de données différent, nous aurons une erreur.

Lorsque nous utilisons l'inférence de type, la variable est toujours typée statiquement. La solution dans ce cas est le type de données dynamique. Ce type de données a été créé spécifiquement par Dart pour les besoins de données de différents types pendant l'exécution du programme. Pour déclarer une variable de type dynamique on utilise simplement le mot clé **dynamic**.

```
main()

{

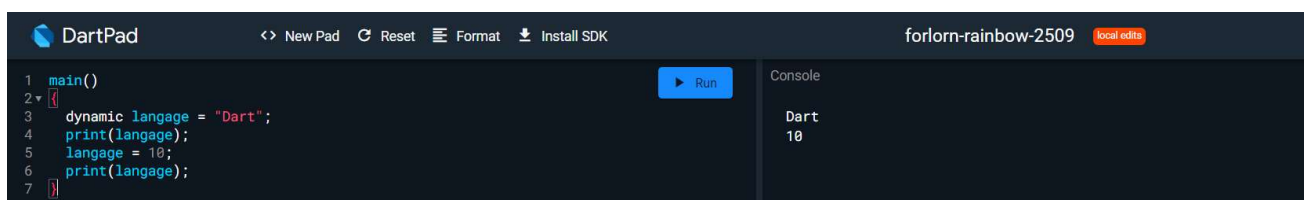
  dynamic langage = "Dart";

  print(langage);

  langage = 10;

  print(langage);

}
```



```
1 main()
2 {
3   dynamic langage = "Dart";
4   print(langage);
5   langage = 10;
6   print(langage);
7 }
```

Console

```
Dart
10
```

Les mots clés **final** et **const** sont utilisés pour déclarer des constantes. Le langage de programmation Dart permet de créer des variables que vous ne pouvez pas modifier à l'aide du mot-clé **final** ou **const**.

```
main() {
```

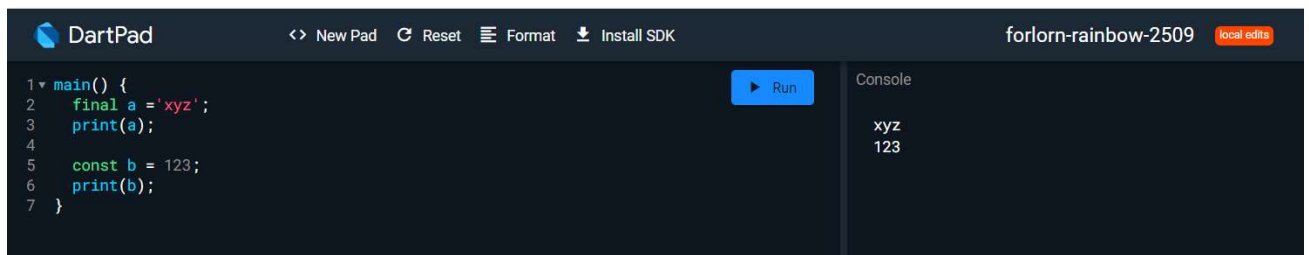
```
final a='xyz';

print(a);


const b = 123;

print(b);

}
```



The screenshot shows the DartPad web interface. The top bar includes the DartPad logo, navigation links (New Pad, Reset, Format, Install SDK), and a user profile section with the name 'forlorn-rainbow-2509' and a 'local edits' indicator. The main editor area displays the Dart code from the previous block, with line numbers 1 through 7 on the left. A blue 'Run' button is positioned to the right of the code. The console output on the right shows the results of the execution: 'xyz' and '123'.

```
main() {

  final a='xyz';

  print(a);


  const b = 123;

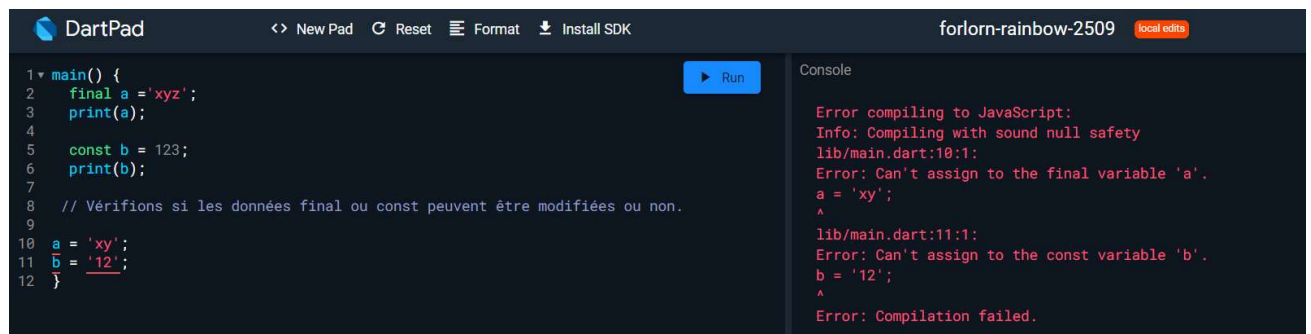
  print(b);


  // Vérifions si les données final ou const peuvent être modifiées ou non.


  a = 'xy';

  b = '12';

}
```



```
1 main() {  
2   final a = 'xyz';  
3   print(a);  
4  
5   const b = 123;  
6   print(b);  
7  
8   // Vérifions si les données final ou const peuvent être modifiées ou non.  
9  
10  a = 'xy';  
11  b = '12';  
12 }
```

Console

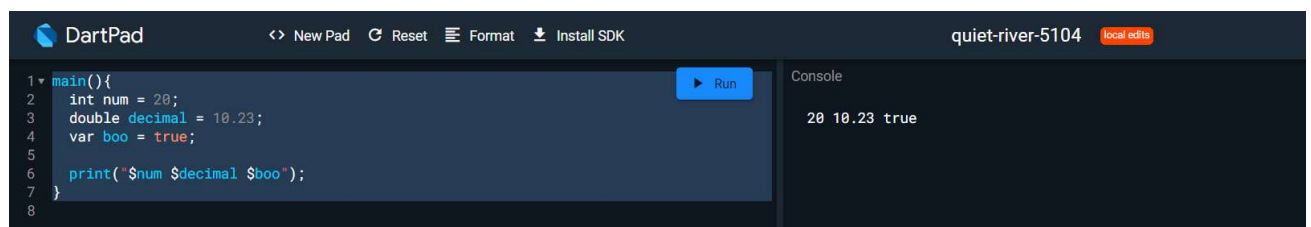
```
Error compiling to JavaScript:  
Info: Compiling with sound null safety  
lib/main.dart:10:1:  
Error: Can't assign to the final variable 'a'.  
a = 'xy';  
^  
lib/main.dart:11:1:  
Error: Can't assign to the const variable 'b'.  
b = '12';  
^  
Error: Compilation failed.
```

Le code vous donnera une erreur de compilation car une fois qu'un type de données est déclaré comme final ou const nous ne pouvons plus le modifier pendant l'exécution.

III. Mettre une variable dans une chaîne de caractères

Nous pouvons inclure n'importe quelle variable ou n'importe quelle expression à l'intérieur d'une chaîne de caractères en utilisant le symbole de préfixe \$ dans le langage Dart.

```
main(){  
  
  int num = 20;  
  
  double decimal = 10.23;  
  
  var boo = true;  
  
  
  print("$num $decimal $boo");  
  
}
```



```
1 main(){  
2   int num = 20;  
3   double decimal = 10.23;  
4   var boo = true;  
5  
6   print("$num $decimal $boo");  
7  
8 }
```

Console

```
20 10.23 true
```

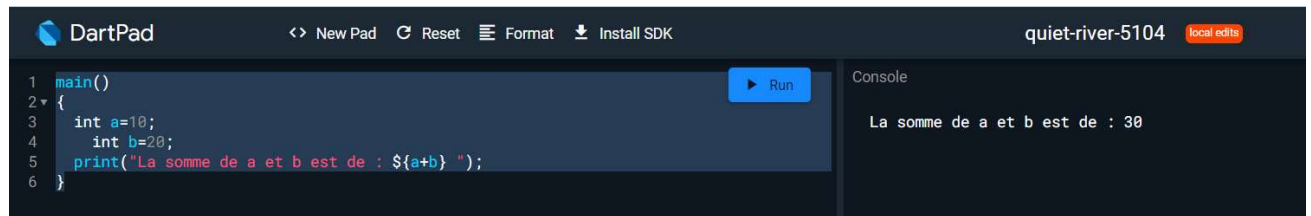
De plus, nous pouvons utiliser l'identifiant '\${}', pour inclure une expression dans une chaîne.

```
main()  
  
{  
  
  int a=10;
```

```
int b=20;

print("La somme de a et b est de : ${a+b} ");

}
```



IV. Quelques propriétés et méthodes de chaînes de caractères

isEmpty : une propriété en lecture seule. Elle renvoie true si la chaîne est vide, false dans le cas contraire.

isNotEmpty : renvoie true si la chaîne n'est pas vide.

length : Elle renvoie le nombre d'unités de code UTF-16.

hashCode : renvoie un code de hachage dérivé des unités de code de la chaîne.

toLowerCase() : convertit tous les caractères en minuscules.

toUpperCase() : convertit tous les caractères en majuscules.

trim() : supprime les espaces du début et de la fin.

replaceAll(Pattern from, String replace) : remplace toutes les sous-chaînes qui correspondent à from par replace et renvoie la chaîne finale.

replaceRange(int start, int end, String replacement) : remplace la sous-chaîne de start à end par la sous-chaîne replacement.

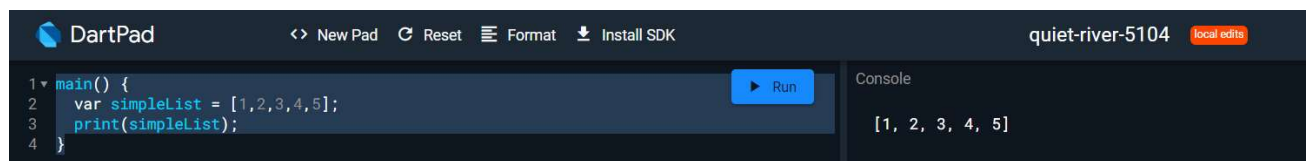
toString() : renvoie un objet sous forme de chaîne.

V. Les listes

Les listes sont des structures de données couramment utilisées dans le langage de programmation Dart. C'est des conteneurs de données où vous pouvez stocker plusieurs valeurs différentes. Puis, les affecter toutes au même nom de variable.

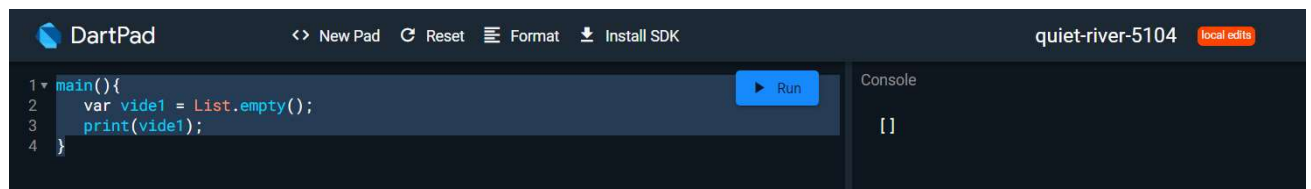
Les tableaux sont l'une des structures de données les plus courantes fournies par les langages de programmation. Cependant, Dart ne fournit pas de tableaux, mais à la place, nous avons des listes qui sont plus ou moins identiques aux tableaux.

```
main() {  
  
  var simpleList = [1,2,3,4,5];  
  
  print(simpleList);  
  
}
```



Créer une liste vide :

```
main(){  
  
  var vide1 = List.empty();  
  
  print(vide1);  
  
}
```



Nous allons créer une liste en spécifiant le type de données. Sachez qu'une liste peut contenir des données de différents types.

```
main(){  
  
  var lst1 = <String>[];  
  
  print(lst1);  
  
  List<String> lst2 = ['A', 'B', 'C'];  
  
  print(lst2);  
  
  var lst3 = [74, 'Paris', 'mardi'];  
  
}
```

```
print(lst3);

}
```

```

1 main(){
2   var lst1 = <String>[];
3   print(lst1);
4   List<String> lst2 = ['A', 'B', 'C'];
5   print(lst2);
6   var lst3 = [74, 'Paris', 'mardi'];
7   print(lst3);
8 }

```

Console

```

[]
[A, B, C]
[74, Paris, mardi]

```

Chaque élément possède son numéro d'index qui indique sa position dans la liste.

```
main(){

    var lst3 = [74, 'Paris', 'mardi'];

    print("index 1 : ${lst3[1]}");

}
```

```

1 main(){
2   var lst3 = [74, 'Paris', 'mardi'];
3   print("index 1 : ${lst3[1]}");
4 }

```

Console

```
index 1 : Paris
```

Un itérable permet de parcourir une collection d'éléments de manière séquentielle.

```
main(){

    var lst3 = [74, 'Paris', 'mardi'];

    lst3.forEach ((n) => print('valeur : ${n} '));

}
```

```

1 main(){
2   var lst3 = [74, 'Paris', 'mardi'];
3   lst3.forEach ((n) => print('valeur : ${n} '));
4 }

```

Console

```

valeur : 74
valeur : Paris
valeur : mardi

```

Les méthodes Dart sont très utiles pour récupérer des données.

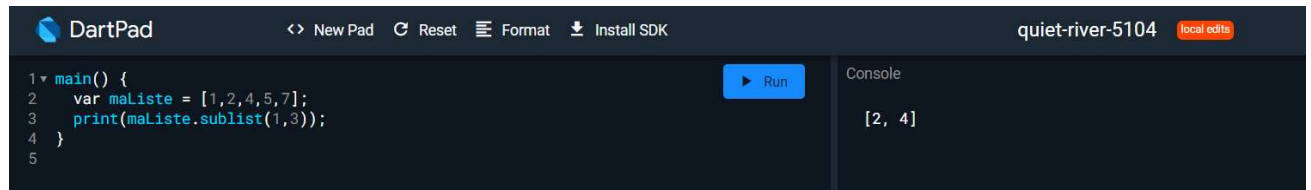
sublist() : renvoie une nouvelle liste contenant des éléments de la liste.

```
main() {

  var maListe = [1,2,4,5,7];

  print(maListe.sublist(1,3));

}
```



shuffle() : réorganise l'ordre des éléments dans la liste donnée de manière aléatoire.

reversed : un getter qui inverse l'itération de la liste en fonction de l'ordre de liste donné.

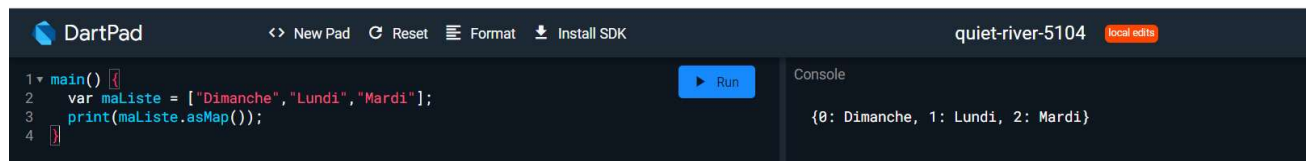
asMap() : renvoie une représentation clé/valeur de la liste.

```
main() {

  var maListe = ["Dimanche","Lundi","Mardi"];

  print(maListe.asMap());

}
```



add() : ajoute un nouvel élément dans la liste donnée.

addAll(), **insert()**, **insertAll()** sont également utilisées pour insérer des éléments.

length : renvoie le nombre total d'éléments dans une liste donnée.

isEmpty : renvoie un booléen si la liste donnée est vide ou non.

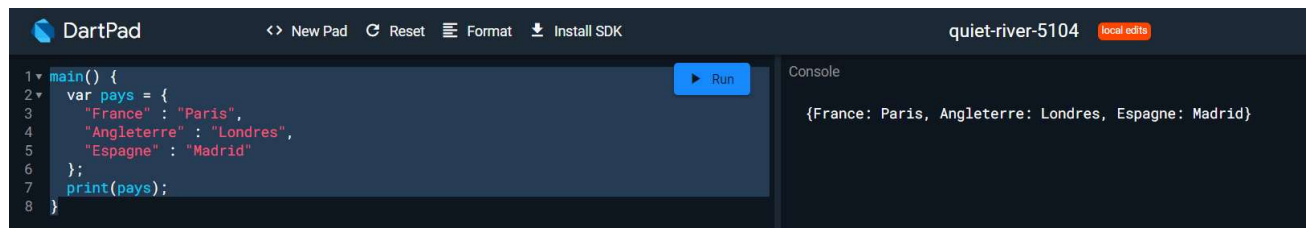
VI. Les données map

Les données map dans Dart sont une collection de paires clé-valeur qui ne sont pas séquentielles. Chaque valeur est associée à une clé, ce qui signifie que chaque clé doit être unique, mais la même valeur peut être utilisée plusieurs fois.

Deux éléments possédant la même valeur resteront uniques en ayant des clés distinctes.

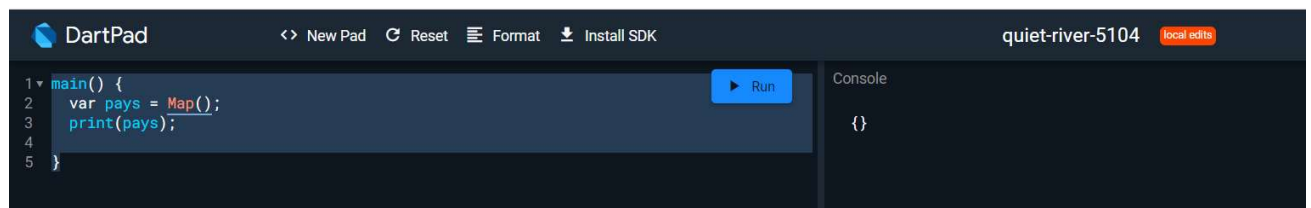
Voici un exemple de map où les clés sont les noms des pays et les valeurs sont leurs capitales.

```
main() {  
  
  var pays = {  
  
    "France" : "Paris",  
  
    "Angleterre" : "Londres",  
  
    "Espagne" : "Madrid"  
  
  };  
  
  print(pays);  
  
}
```



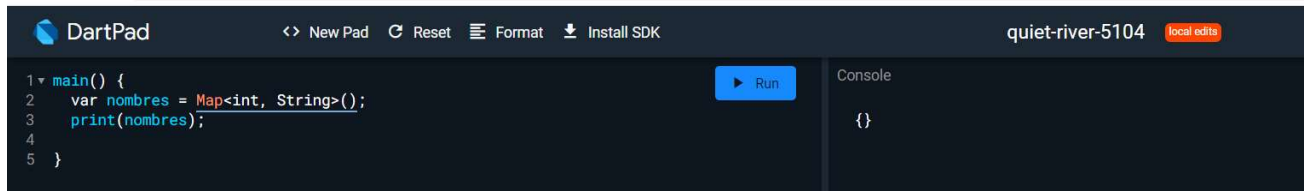
De plus, nous pouvons utiliser le constructeur Map().

```
main() {  
  
  var pays = Map();  
  
  print(pays);  
  
}
```



Sachez également que nous pouvons utiliser le mot clé new avec map.

Afin de déclarer le type de données map nous utiliserons la syntaxe suivante :



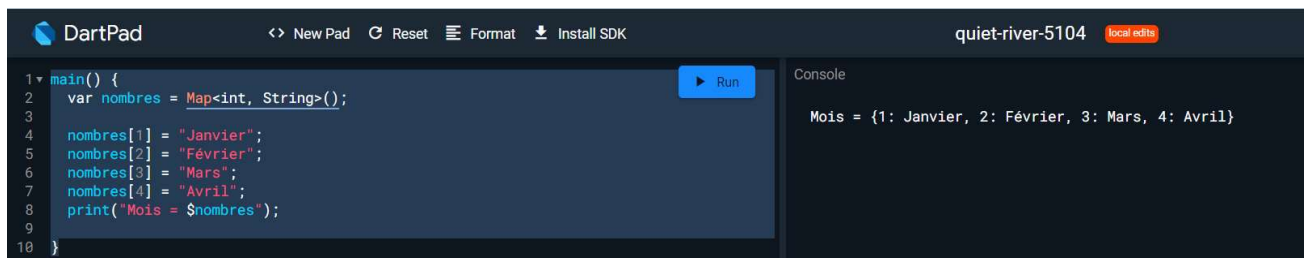
```
1 main() {  
2   var nombres = Map<int, String>();  
3   print(nombres);  
4  
5 }
```

Console

```
{}
```

Nous allons ajouter des éléments à notre map nombres en déterminant les clés et les valeurs.

```
main() {  
  
  var nombres = Map<int, String>();  
  
  nombres[1] = "Janvier";  
  
  nombres[2] = "Février";  
  
  nombres[3] = "Mars";  
  
  nombres[4] = "Avril";  
  
  print("Mois = $nombres");  
  
}
```



```
1 main() {  
2   var nombres = Map<int, String>();  
3  
4   nombres[1] = "Janvier";  
5   nombres[2] = "Février";  
6   nombres[3] = "Mars";  
7   nombres[4] = "Avril";  
8   print("Mois = $nombres");  
9  
10 }
```

Console

```
Mois = {1: Janvier, 2: Février, 3: Mars, 4: Avril}
```

Dans notre exemple, nous avons déclaré le type de nos éléments, les clés sont de type int et les valeurs sont de type String.

L'objet map possède une propriété qui permet d'accéder à toutes les clés et les valeurs.

```
main() {  
  
  var nombres = Map<int, String>();  
  
  nombres[1] = "Janvier";  
  
  nombres[2] = "Février";  
  
  nombres[3] = "Mars";  
  
  nombres[4] = "Avril";  
  
}
```

```

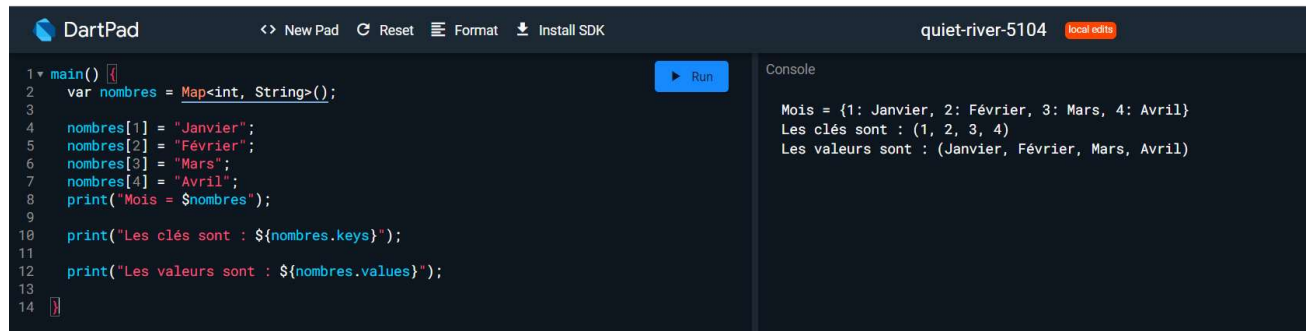
print("Mois = $nombres");

print("Les clés sont : ${nombres.keys}");

print("Les valeurs sont : ${nombres.values}");

}

```



The screenshot shows the DartPad interface. The code editor on the left contains the following Dart code:

```

1 main() {
2   var nombres = Map<int, String>();
3
4   nombres[1] = "Janvier";
5   nombres[2] = "Février";
6   nombres[3] = "Mars";
7   nombres[4] = "Avril";
8   print("Mois = $nombres");
9
10  print("Les clés sont : ${nombres.keys}");
11
12  print("Les valeurs sont : ${nombres.values}");
13
14 }

```

The console on the right shows the output:

```

Mois = {1: Janvier, 2: Février, 3: Mars, 4: Avril}
Les clés sont : (1, 2, 3, 4)
Les valeurs sont : (Janvier, Février, Mars, Avril)

```

La méthode `remove()`, permet de supprimer l'élément clé-valeur spécifié.

```

main() {

  var pays = {

    "France" : "Paris",

    "Angleterre" : "Londres",

    "Espagne" : "Madrid"

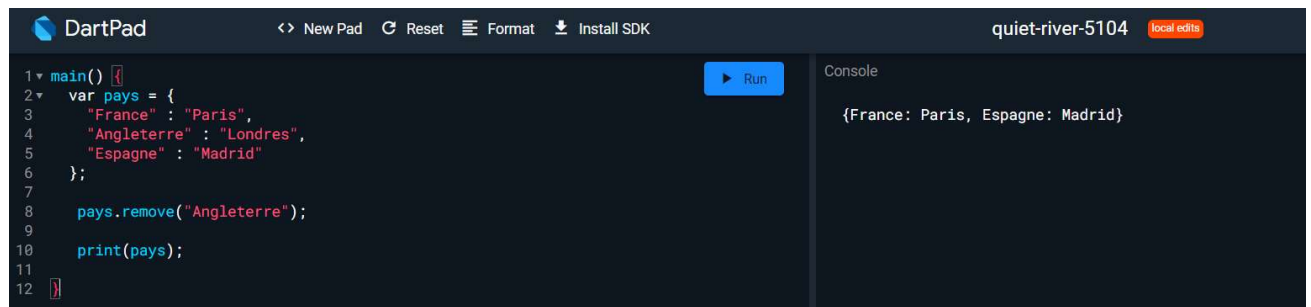
  };

  pays.remove("Angleterre");

  print(pays);

}

```



The screenshot shows the DartPad interface. The code editor on the left contains the following Dart code:

```

1 main() {
2   var pays = {
3     "France" : "Paris",
4     "Angleterre" : "Londres",
5     "Espagne" : "Madrid"
6   };
7
8   pays.remove("Angleterre");
9
10  print(pays);
11
12 }

```

The console on the right shows the output:

```

{France: Paris, Espagne: Madrid}

```

Comme avec les listes, nous pouvons utiliser d'autres propriétés et méthodes comme :

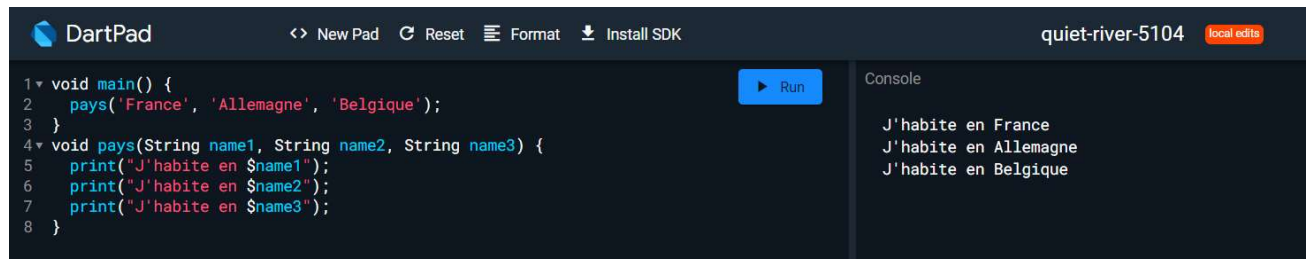
`length`, `isEmpty`, `isNotEmpty`, `addAll`, `clear`, `forEach`.

VII. Les types de paramètres dans les fonctions Dart

Les paramètres sont classés en deux types dans Dart : obligatoires et optionnels. Puis, les paramètres optionnels sont divisés en nommés ou positionnels.

Voici un exemple de **paramètres obligatoires** :

```
void main() {  
  
  pays('France', 'Allemagne', 'Belgique');  
  
}  
  
void pays(String name1, String name2, String name3) {  
  
  print("J'habite en $name1");  
  
  print("J'habite en $name2");  
  
  print("J'habite en $name3");  
  
}
```



Le mot clé void veut dire que la fonction ne retourne aucune valeur.

Cependant, nous pouvons ne pas spécifier les paramètres optionnels lors de l'appel d'une fonction. Les paramètres facultatifs doivent être déclarés après les paramètres obligatoires. De plus, les paramètres optionnels peuvent avoir des valeurs par défaut, qui seront utilisées, si vous ne les spécifiez pas lors de l'appel de la fonction.

Si un paramètre optionnel ne reçoit pas de valeur, il est défini sur NULL.

On utilise des crochets [] pour spécifier des **paramètres optionnels de position** dans Dart. En fait, il faut savoir quelle position correspond à quel paramètre.

```
void main() {  
  
  pays('France', 'Allemagne');
```

```

}

void pays(String name1, String name2, [String? name3]) {

    print("J'habite en $name1");

    print("J'habite en $name2");

    print("J'habite en $name3");

}

```

The screenshot shows the DartPad interface with the following code in the editor:

```

1 void main() {
2   pays('France', 'Allemagne');
3 }
4 void pays(String name1, String name2, [String? name3]) {
5   print("J'habite en $name1");
6   print("J'habite en $name2");
7   print("J'habite en $name3");
8 }

```

The console output on the right shows:

```

J'habite en France
J'habite en Allemagne
J'habite en null

```

Même si nous n’avons pas passé un troisième argument nous n’avons pas d’erreur. Il y a une valeur null dans le résultat car aucune valeur n’est affectée au paramètre name3.

Les accolades `{}` sont utilisées pour les **paramètres optionnels nommés** dans Dart. Comme les paramètres nommés sont référencés par leur nom, ils peuvent donc être utilisés lors de l’appel dans un ordre différent de celui de la déclaration. Nous pouvons les appeler en utilisant `param : value`.

```

void message(String param1, {String? param2}){

    print('param1 : $param1');

    print('param2 : $param2');

}

void main() {

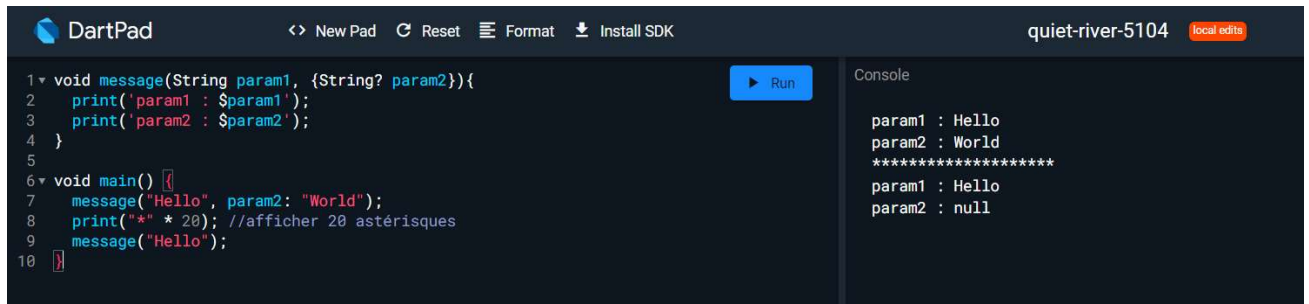
    message("Hello", param2: "World");

    print("*" * 20); //afficher 20 astérisques

    message("Hello");

}

```

The screenshot shows the DartPad interface. The code editor contains the following Dart code:

```
1 void message(String param1, {String? param2}){  
2   print('param1 : $param1');  
3   print('param2 : $param2');  
4 }  
5  
6 void main() {  
7   message("Hello", param2: "World");  
8   print("*" * 20); //afficher 20 astérisques  
9   message("Hello");  
10 }
```

The console output on the right shows the results of the execution:

```
param1 : Hello  
param2 : World  
*****  
param1 : Hello  
param2 : null
```

Lorsque nous n'avons pas spécifié param2, il a été retourné avec une valeur null et la fonction a été exécutée. Si nous utilisons des paramètres obligatoires nous aurons une erreur.

Notez l'utilisation du point d'interrogation après String. En fait, depuis Dart 2.12, le langage prend en charge la sécurité nulle. Ici, **?** indique explicitement qu'une variable ou un paramètre **peut être nul**.

Il est possible de déclarer des **valeurs par défaut avec les paramètres optionnels**. Toutefois, elles seront utilisées si nous ne passons aucune valeur à ces paramètres. Le signe **=** est utilisé pour affecter une valeur par défaut à un paramètre.

```
void message(String param1, {String? param2 = "Dart"}){  
  
  print('param1 : $param1');  
  
  print('param2 : $param2');  
  
}  
  
void main() {  
  
  message("Hello", param2: "World");  
  
  print("*" * 20);  
  
  message("Hello");  
  
}
```



The screenshot shows the DartPad interface with the same code as the previous screenshot, but with a default value for param2:

```
1 void message(String param1, {String? param2 = "Dart"}){  
2   print('param1 : $param1');  
3   print('param2 : $param2');  
4 }  
5  
6 void main() {  
7   message("Hello", param2: "World");  
8   print("*" * 20);  
9   message("Hello");  
10 }
```

The console output on the right shows the results of the execution:

```
param1 : Hello  
param2 : World  
*****  
param1 : Hello  
param2 : Dart
```

VIII. Les classes Dart

Dart est un langage orienté objet, et tous ce que vous manipulez dans un programme Dart est un objet. Un objet représente une instance d'une classe et une classe est un modèle ou une instance d'un autre objet.

Pour faire simple, une classe est un modèle qui stocke les propriétés de l'objet ainsi que les actions qu'ils peut effectuer.

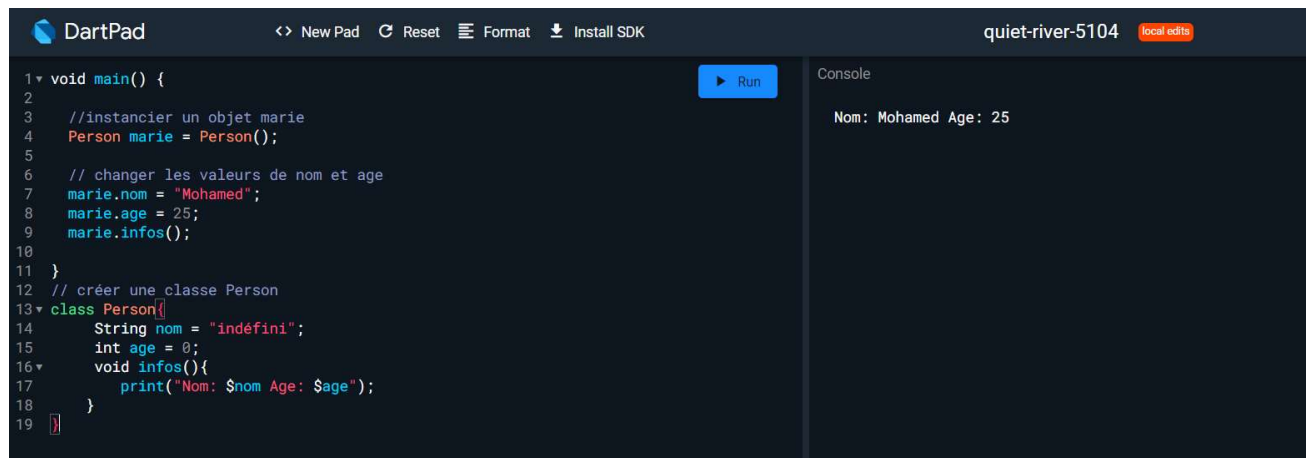
Nous pourrions utiliser le type nullable en utilisant le signe ?, ce qui rend les valeurs initiales optionnelles.

Dans une classe Dart nous pouvons définir des méthodes spéciales, appelées constructeurs. Dart appelle le constructeur lors de la création d'un nouvel objet d'une classe. Les constructeurs effectuent l'initialisation de l'objet.

Si Dart ne trouve aucun constructeur dans la classe, il créera automatiquement un constructeur sans paramètres.

```
void main() {  
  
    //instancier un objet  
    Person marie = Person();  
  
    // changer les valeurs de nom et age  
    marie.nom = "Mohamed";  
    marie.age = 25;  
    marie.infos();  
  
}  
  
// créer une classe Person  
class Person{  
    String nom = "indéfini";  
    int age = 0;  
    void infos(){  
        print("Nom: $nom Age: $age");  
    }  
}
```

```
}  
  
}
```



The screenshot shows the DartPad web interface. The editor contains the following Dart code:

```
1 void main() {  
2  
3   //instancier un objet marie  
4   Person marie = Person();  
5  
6   // changer les valeurs de nom et age  
7   marie.nom = "Mohamed";  
8   marie.age = 25;  
9   marie.infos();  
10  
11 }  
12 // créer une classe Person  
13 class Person{  
14   String nom = "indéfini";  
15   int age = 0;  
16   void infos(){  
17     print("Nom: $nom Age: $age");  
18   }  
19 }
```

The console on the right displays the output: "Nom: Mohamed Age: 25".

Le constructeur prend le même nom de la classe et utilise un mot clé this.

```
class Person{  
  
    String nom = "";  
  
    int age = 0;  
  
    // le constructeur  
    Person(String nom, int age){  
  
        this.nom = nom;  
  
        this.age = age;  
  
    }  
  
    void infos(){  
  
        print("Nom: $nom Age: $age");  
  
    }  
  
}
```

```

void main() {

    //instancier un objet

    Person pers1 = Person('Mohamed', 25);


    pers1.infos();

}

```

The screenshot shows the DartPad web editor interface. The top bar includes the DartPad logo, navigation links (New Pad, Reset, Format, Install SDK), and a user identifier 'quiet-river-5104' with a 'local edit' button. The main editor area displays a Dart class 'Person' with attributes 'nom' and 'age', a constructor, and an 'infos()' method. The 'main()' function instantiates a 'Person' object and calls 'infos()'. A 'Run' button is visible next to the code. On the right, the 'Console' panel shows the output: 'Nom: Mohamed Age: 25'.

```

1 class Person{
2     String nom = "";
3     int age = 0;
4
5     // le constructeur
6     Person(String nom, int age){
7
8         this.nom = nom;
9         this.age = age;
10    }
11
12    void infos(){
13        print("Nom: $nom Age: $age");
14    }
15 }
16
17 void main() {
18     //instancier un objet
19     Person pers1 = Person('Mohamed', 25);
20
21     pers1.infos();
22 }

```

Console

Nom: Mohamed Age: 25

En plus simple :

```

class Person{

    String nom = "";

    int age = 0;


    // le constructeur

    Person(this.nom, this.age);


    void infos(){

        print("Nom: $nom Age: $age");
    }
}

```

```

    }

}

void main() {

    //instancier un objet

    Person pers1 = Person('Mohamed', 25);

    pers1.infos();

}

```

The screenshot shows the DartPad web editor interface. The top bar includes the DartPad logo, navigation links (New Pad, Reset, Format, Install SDK), and a session identifier 'quiet-river-5104' with a 'local edits' indicator. The main editor area displays the Dart code from the previous block, with line numbers 1 through 20. A blue 'Run' button is visible next to the code. To the right of the code editor is a 'Console' panel showing the output: 'Nom: Mohamed Age: 25'.

Les types de données Dart intégrés n'autorise pas une valeur nulle. Cependant, si nous voulons stocker une valeur nulle dans un objet de la classe, nous pouvons utiliser la classe nullable, en ajoutant l'opérateur ? à la définition de type.

```

class Person{

    String nom = "";

    int age = 0;

    // le constructeur

    Person(this.nom, this.age);

    void infos(){

```

```

        print("Nom: $nom Age: $age");
    }
}

void main() {

    //instancier un objet

    Person pers1 = Person('Mohamed', 25);

    pers1.infos();

    //instancier un 2ème objet

    Person? pers2 ;

    pers2?.infos();

}

```

The screenshot shows the DartPad web interface. The editor on the left contains the following Dart code:

```

1 class Person{
2     String nom = "";
3     int age = 0;
4
5     // le constructeur
6     Person(this.nom, this.age);
7
8     void infos(){
9         print("Nom: $nom Age: $age");
10    }
11 }
12
13 void main() {
14     //instancier un objet
15     Person pers1 = Person('Mohamed', 25);
16
17     pers1.infos();
18
19     //instancier un 2ème objet
20     Person? pers2 ;
21
22     pers2?.infos();
23 }
24

```

The console on the right shows the output of the program:

```

Nom: Mohamed Age: 25

```

L'opérateur `?` vérifie la valeur de la variable, si elle n'est pas nulle, alors l'accès à ses champs et méthodes se produit. Si elle est nulle, la référence à la variable sera ignorée. Rien ne s'affiche, mais vous n'aurez pas une erreur.

Nous pouvons utiliser des constructeurs nommés pour pouvoir initialiser une classe de différentes manières.

En fait, pour définir plusieurs constructeurs nous écrivons `nom_du_constructeur.Nom`.

Il peut être utilisé comme une alternative à la surcharge de constructeur trouvée dans d'autres langages.

```
class Person{

    String nom = "";

    int age = 0;

    // constructeur simple

    Person(this.nom ,this.age) ;

    // constructeur nommé

    Person.fromArray(List array) {

        nom = array[0];

        age = array[1];

    }

}

void main() {

    Person p1 = Person("Mohamed",25);

    print ("${p1.nom} ${p1.age}");

    Person p2 = Person.fromArray(["Mariem",20]);

    print ("${p2.nom} ${p2.age}");

}
```

```
1 class Person{
2   String nom = "";
3   int age = 0;
4
5   // constructeur simple
6   Person(this.nom ,this.age) ;
7
8   // constructeur nommé
9
10  Person.fromArray(List array) {
11    nom = array[0];
12    age = array[1];
13  }
14 }
15
16 void main() {
17
18   Person p1 = Person("Mohamed",25);
19
20   print ("${p1.nom}  ${p1.age}");
21
22   Person p2 = Person.fromArray(["Mariem",20]);
23
24   print ("${p2.nom}  ${p2.age}");
25
26 }
```

▶ Run

Console

```
Mohamed 25
Mariem 20
```