ما هو Flutter؟

Flutter هو إطار عمل (Framework) برمجي تم تطويره بواسطة شركة Google.

وصف مبسط:

- أداة لبناء تطبيقات الموبايل (أندرويد وآيفون)
- يدعم تطبيقات الويب وسطح المكتب
- يمكنك كتابة كود واحد يعمل على جميع الأجهزة

مميزات Flutter

1. شكل جذاب وسريع

- تصميم تطبيقات عصرية وجذابة
- أداء سريع ومشابه للتطبيقات الأصلية

2. سهولة الاستخدام

- مناسب للمبتدئين في البرمجة

- سهل التعلم والعمل عليه

3. مدعوم من Google
- تحديثات مستمرة
- دعم كبير من مجتمع المطورين العالمي

اللغة البرمجية: Dart

- لغة برمجة سهلة وبسيطة للتعلم
- مطورة من قبل Google

أنواع البيانات الأساسية في Dart

1. int : للأعداد الصحيحة
- مثل: 1، 6757، 90+30

2. double: للأعداد العشرية
- مثل: 9.9

3. String: للنصوص
- باستخدام ' ' أو " "

4. bool: للقيم المنطقية
- true أو false

هيكل الجملة البرمجية

مثال: `int number = 10;`

المكونات الأساسية:

1. نوع البيانات (int)
- يحدد نوع المتغير (عدد صحيح)
- لا يقبل الأرقام العشرية

2. اسم المتغير (number)
- المكان لتخزين البيانات
- يمكن تسميته بأي اسم مناسب

3. التهيئة (= 10)

- تعيين القيمة الأولية للمتغير

ملاحظة مهمة: الفاصلة المنقوطة (؛)

- تُنهي الجملة البرمجية

- تخبر مترجم Dart بانتهاء التعليمة

امثله علي data types في dart

```dart
void main() {
  // int نوع من متغير
 // ( 25عدد صحيح)
  int age = 25;
  print('العمر: $age');

  // double نوع من متغير
 // (1.75عدد عشري)
  double height = 1.75;
  print('الطول: $height');

  // bool نوع من متغير
 // (قيمة منطقية true)
  bool isStudent = true;
  print('هل هو طالب؟ $isStudent');
}
```

```
النتيجة
العمر: 25
الطول: 1.75
هل هو طالب؟ true
```

مثال اخر

```
void main() {
  // المتغيرات تعريف
  int age = 30;              // نوع من متغير int
  double weight = 85.6;      // نوع من متغير double
  String city = "القاهرة"; // نوع من متغير String
  bool isEmployed = false; // نوع من متغير bool

  // واحد سطر في القيم طباعة
  print("العمر: $age, الوزن: $weight, المدينة: $city, يعمل:
$isEmployed");
}
```

```
                                                    //النتجه المتوقعه
          : falseيعمل, القاهرة: المدينة: 85.6, الوزن: 30, //العمر
                                                    //
```

مثال اخر

```
void main() {
  // المتغيرات تعريف
  int numberOfBooks = 12;
  double temperature = 23.5;
  String favoriteColor = "أزرق";
  bool hasPet = true;


  print("الكتب عدد: $numberOfBooks, الحرارة درجة: $temperature,
```

```
المفضل اللون : $favoriteColor, أليف حيوان لديه : $hasPet");
}
```

القيم المتوقعه:

```
//عدد الكتب 12, درجة الحرارة 23.5 : اللون المفضل : أزرق ,لديه
حيوان أليفtrue :
```

السؤال الاول:

بناءً على البيانات الأساسية، تحدث عن فريقك المفضل، عدد بطولاته، عدد اللاعبين فيه، وهل يشارك في كأس العالم ام لا؟

الجزء الثاني:

ما هو var؟

`var` هي **keyword**(كلمة مفتاحية) في لغة **Dart** تُستخدم للإعلان عن متغير دون الحاجة إلى تحديد نوعه بشكل صريح. يقوم الـ **compiler** (المترجم) بتحليل القيمة المُسندة للمتغير أثناء وقت الترجمة وتحديد نوع البيانات تلقائيًا.

كيف تعمل `var` مع compiler؟

1. عندما تكتب الكود باستخدام **`var`**، يقوم الـ **compiler** بالنظر إلى القيمة التي تُسندها للمتغير.
2. بناءً على هذه القيمة، يحدد الـ **compiler** نوع البيانات (مثل `int`, `String`, `double`, أو `bool`).
3. بعد تحديد النوع، يثبّت الـ **compiler** نوع المتغير، ولا يمكنك تغييره لاحقًا.

مثال:

```
void main() {
  var age = 30; //     النوع تم تحدده int
  var name = "أحمد"; //   String النوع تم تحدده ك
  var isStudent = true; // bool النوع تم تحديده
  print("العمر: $age, الاسم: $name, طالب: $isStudent");
}
```

كيف تعمل `var` مع compiler؟

1. عندما تكتب الكود باستخدام **`var`**، يقوم الـ **compiler** بالنظر إلى القيمة التي تُسندها للمتغير.
2. بناءً على هذه القيمة، يحدد الـ **compiler** نوع البيانات (مثل `int`, `String`, `double`, أو `bool`).
3. بعد تحديد النوع، يثبّت الـ **compiler** نوع المتغير، ولا يمكنك تغييره لاحقًا.

سؤال 2

من خلال الكود اعطي مثال علي استخدام var في لغه dart ؟

ما هي dynamic في لغة Dart؟

dynamic هي **keyword**(كلمة مفتاحية) في Dart تُستخدم للإعلان عن متغير يكون **نوعه غير ثابت** ويمكنه أن يحتفظ بأي نوع من البيانات (مثل int, String, double, bool, أو حتى كائنات) وتغيير النوع في وقت التشغيل(**Runtime**)

مثال:

```
void main() {
  dynamic variable = 10; // int يبدأ كعدد صحيح
  print("القيمة: $variable, النوع: ${variable.runtimeType}");
// النوع: int

  variable = "مرحبًا"; //  String يتغير النوع الي نصوص
  print("القيمة: $variable, النوع: ${variable.runtimeType}");
 // النوع: String

  variable = 3.14; // يتغير النوع مره الي رقم عشري يخعلامث
  print("القيمة: $variable, النوع: ${variable.runtimeType}");
 // النوع: double
}
```

مثال 2: استخدام dynamic مع عمليات متعددة:

```
void main() {
  dynamic data = "Dart Programming"; // نص (String)
  print("النص: $data, الطول: ${data.length}"); // خاصية استخدام
النصوص

  data = 42; // صحيح عدد إلى النوع تغيير (int)
  print("القيمة: $data, مضروبة ×2: ${data * 2}");

  data = true; // منطقي إلى النوع تغيير (bool)
  print("القيمة: $data, النوع: ${data.runtimeType}");
}
```

سؤال 3

من خلال الكود اعطي امثله علي استخدام dynamic في لغه dart؟

سؤال 4

بشرح نظري وعملي من خلال الكود اذكر الفرق بين var , dynamic؟

final

- **التعريف:**
  - تُستخدم الكلمة المفتاحية **`final`** لتعريف متغير يُسند له قيمة مرة واحدة فقط، ولا يمكن تغييرها بعد ذلك.
  - يتم تحديد القيمة أثناء **وقت التشغيل** .(Runtime)
- **الخصائص:**
  - يمكن أن تكون القيمة ثابتة أو تُحسب في وقت التشغيل.
  - تُستخدم عند الحاجة إلى متغير ثابت لكن قيمته تُعرف أثناء تشغيل البرنامج.
- **مثال:**

```
void main() {
  final int age = 30;
//  القيمه هنا ثابته ومعروفه ولا يمكن تغيرها
```

```
    final currentTime = DateTime.now(); ثابته ولكن
  // تحدد القيمه أثناء وقت التشغيل
    print("الحالي الوقت: $currentTime");
  }
```

:Const

- **التعريف:**
  - تُستخدم الكلمة المفتاحية **`const`** لتعريف متغير ثابت تكون قيمته معروفة ومُحددة أثناء **وقت الترجمة** .(Compile Time)
  - تُستخدم مع القيم الثابتة فقط، مثل الأرقام أو النصوص أو الأشياء التي لا تتغير مطلقًا.
- **الخصائص:**
  - القيمة يجب أن تكون معروفة أثناء **وقت الترجمة**.
  - تُستخدم عادةً للثوابت الرياضية أو النصوص التي لا تتغير أبدًا.
- **مثال:**

```
void main() {
  const double pi = 3.14;
//قيمه ثابته اثناء وقت الترجمه
  const String appName = "MyApp";

  // const currentTime = DateTime.now();
//   سيظهر خطأ لان القيمه يتم تحددها وقت التشغيل
  print("التطبيق اسم: $appName, قيمة باي: $pi");
}
```

مثال:

حدد ما إذا كان الكود التالي صحيحًا أم لا. إذا كان هناك خطأ، اذكر السبب وقم بإصلاحه.

```
void main() {
  const double discount = 0.1;
  const double price = 100;

  if (price > 50) {
    const double finalPrice = price - (price * discount);
    print("السعر النهائي: $finalPrice");
  } else {
    print("لا يوجد خصم");
  }
}
```

الاجابه:

الكود هنا يعمل لأنه يعتمد فقط على **قيم ثابتة** و**عمليات حسابية ثابتة** أثناء وقت الترجمة.(Compile Time)

السؤال رقم 5

في البرنامج التالي، هل الكود يعمل بشكل صحيح؟ إذا كان هناك خطأ، اذكر السبب وأصلح الكود؟

```
void main() {
  final age = 25;
  age = 30; // قيمة تغيير age
  print("العمر: $age");
}
```

السؤال رقم 6

ما هو الفرق بين final , cost وضح اجأبتك من خلال الشرح النظري والكود؟

الجزء الثالث:

البيانات المعقدة (Complex Data) في لغة Dart

في لغة **Dart**، البيانات المعقدة هي أنواع البيانات التي يمكنها تخزين وإدارة مجموعات من القيم أو بيانات ذات بنية أكثر تعقيدًا مقارنة بالبيانات الأساسية (مثل `int`, `String`, `bool`). تشمل البيانات المعقدة **القوائم(Lists)** ، **الخرائط(Maps)** ، **المجموعات(Sets)** ، **الكائنات (Objects)**، وغيرها.

أنواع القوائم في dart

قائمة ديناميكية الطول:(Growable List)

- يمكن تغيير طول القائمة بإضافة أو حذف العناصر.
- هذا النوع الأكثر شيوعًا.

مثال:

```
void main() {
  List<String> growableList = ["أحمد", "محمد"];
  growableList.add("سارة"); // اضافه عنصر جديد
  growableList.remove("محمد"); // حذف عنصر
  print(growableList); // ["أحمد", "سارة"]
}
```

2.خصائص القوائم:

2.1.القوائم المرتبة:(Ordered)

- العناصر داخل القائمة مُرتبة، ويمكنك الوصول إليها باستخدام الفهرس(Index).
- مثال:

```
void main() {
  List<int> numbers = [10, 20, 30];
  print(numbers[0]); // 10 (العنصر الأول )
}
```

2.4.أنواع بيانات مختلفة:(Mixed Data Types)

- باستخدام قائمة ديناميكية(**List<dynamic>**) ، يمكن أن تحتوي القائمة على أنواع بيانات مختلفة.

- Int, String, bool

```dart
void main() {
  List<dynamic> mixedList = [1, "Dart", true];
  print(mixedList); // [1, "Dart", true]
}
```

قوائم فارغة (Empty Lists):

- يمكن إنشاء قائمة فارغة ثم إضافة عناصر لاحقًا.

```dart
void main() {
  List<String> emptyList = [];
  emptyList.add("عنصر جديد");
  print(emptyList); // ["عنصر جديد "]
}
```

الوصول إلى العناصر باستخدام الفهرس (Index):

- يمكنك الوصول إلى أي عنصر باستخدام الفهرس.
- Dartتستخدم فهرسًا يبدأ من **0**.

```dart
void main() {
  List<int> numbers = [10, 20, 30];
  print(numbers[1]); // 20 ( العنصر الثاني )
}
```

طول القائمة: (List Length)

- يمكنك معرفة عدد العناصر في القائمة باستخدام خاصية **length**

```
void main() {
  List<int> numbers = [10, 20, 30];
  print(numbers.length); // 3
}
```

إضافة عناصر إلى القائمة:

- باستخدام دوال مثل **add** و**addAll**.

```
void main() {
  List<int> numbers = [10, 20];
  numbers.add(30); // 30 رقم  اضافه
  numbers.addAll([40, 50]); / /  اضافه عده عناصر
 print(numbers); // [10, 20, 30, 40, 50]
}
```

حذف العناصر من القائمة:

- باستخدام دوال مثل **remove** و**removeAt**

```
oid main() {
  List<int> numbers = [10, 20, 30];
  numbers.remove(20); // 20حذف العنصر
  numbers.removeAt(0); // 10 حذف العنصر
  print(numbers); // [30]
}
```

الفرق الأساسي بين **remove** و**removeAt** في لغة **Dart** هو في طريقة تحديد العنصر المراد حذفه من القائمة:

remove:

- تُستخدم لحذف العنصر من القائمة بناءً على قيمته.
- إذا كانت القيمة موجودة، يتم حذف أول ظهور لهذه القيمة في القائمة.
- إذا لم تكن القيمة موجودة، فلا يحدث شيء.
- تُعيد `true` إذا تم حذف العنصر، و `false` إذا لم يتم العثور على العنصر

```
void main() {
  List<int> numbers = [10, 20, 30, 40];
  bool result = numbers.remove(20);
 //  20 يحذف العنصر الذي قيمته
  print(numbers); // [10, 30, 40]
  print(result);  // true

  result = numbers.remove(50); // العنصر رقم 50 غير موجود
  print(numbers); // [10, 30, 40]
  print(result);  // false
}
```

removeAt

- تُستخدم لحذف العنصر من القائمة بناءً على **الفهرس (Index).**
- يجب تحديد رقم الفهرس (index) للعنصر المراد حذفه.
- تُعيد القيمة التي تم حذفها.
- إذا كان الفهرس غير صحيح (خارج حدود القائمة)، سيتم طرح استثناء (Error).

بمثال:

```
void main() {
```

```
  List<int> numbers = [10, 20, 30, 40];
  int removedElement = numbers.removeAt(2);

 10 هنا هو رقم 2 رقمindexال
  print(numbers); // [10, 20, 40]
  print(removedElement); // 30

  // numbers.removeAt(5);
  // خطأ! الفهرس 5 خارج حدود القائمة
}
```

التحقق من وجود عنصر معين:

- باستخدام دالة **contains**.

```
•
   void main() {
     List<int> numbers = [10, 20, 30];
     print(numbers.contains(20)); // true
     print(numbers.contains(40)); // false

   }
```

لإخراج العناصر التي تحتوي على الحرف a من القائمةnames ، يمكنك استخدام دالة **where** مع **lambda function** للتحقق من وجود الحرف داخل كل عنصر.

```
void main() {
  List<String> names = ["ahmed", "maner", "noor"]; // قائمة
بالأسماء


  // where استخدام
  لتصفية الأسماء التي تحتوي على الحرف 'a' //
  List<String> filteredNames = names.where((name) =>
name.contains('a')).toList();

  print(filteredNames); // ["ahmed", "maner"]
}
```

إذا كنت تريد البحث داخل القائمه بغض النظر عن حالة الأحرف (كبيرة أو صغيرة)، يمكنك استخدام **toLowerCase**

```
List<String> filteredNames = names.where((name) =>
name.toLowerCase().contains('a')).toList();
```

ترتيب العناصر:

- باستخدام دالة **sort** لترتيب العناصر.

```
void main() {
  List<int> numbers = [30, 10, 20];
  numbers.sort(); // ترتيب تصاعدي
  print(numbers); // [10, 20, 30]
}
```

-

3.قوائم متداخلة:(Nested Lists)

- يمكن أن تحتوي القائمة على قوائم أخرى.

```
void main() {
  // منفصلة قوائم تعريف
  List<String> names = ["أحمد", "محمد", "سارة"]; // قائمة الأسماء
  List<int> numbers = [1, 2, 3];                 // قائمة الأرقام
  List<String> letters = ["أ", "ب", "ت"];        // قائمة الحروف

  // أنشاء قائمه متداخله
  List<dynamic> nestedList = [names, numbers, letters];

  // الوصول الي القائمه المتداخله داخل الحروف
  List<String> extractedLetters = nestedList[2];
  // الوصول الي الحروف
  print("الحروف: $extractedLetters");

  // الوصول الي الحرف الثاني في قائمه الحروف
  String secondLetter = extractedLetters[1];
```

```
    print("الحرف الثاني: $secondLetter");
  }
```

شرح الكود:

1. **تعريف القوائم المنفصلة:**
   - `names`: تحتوي على الأسماء.
   - `numbers`: تحتوي على الأرقام.
   - `letters`: تحتوي على الحروف.
2. **إنشاء القائمة المتداخلة:**
   - `nestedList` تجمع القوائم الثلاثة في قائمة واحدة.
3. **الوصول إلى قائمة الحروف:**
   - باستخدام الفهرس `[2]` نصل إلى قائمة الحروف داخل `nestedList`.
4. **الوصول إلى حرف معين:**
   - باستخدام الفهرس داخل قائمة الحروف `[1]` نصل إلى الحرف الثاني وهو "ب".

النتيجة عند تشغيل الكود:

```
الحروف: [أ, ب, ت]
الحرف الثاني: ب
```

قائمة ثابتة الطول (Fixed-length List)

خصائص القائمة ثابتة الطول

المميزات الأساسية:

- لا يمكن تغيير طول القائمة بعد إنشائها
- يمكن تعديل القيم الموجودة
- لا يمكن إضافة عناصر جديدة أو حذف عناصر

مثال توضيحي بالكود

```
void main() {
  // إنشاء قائمة ثابتة الطول بـ 5 عناصر
  List<int> fixedList = List.filled(5، 1)؛

  // طباعة القائمة الأصلية
  print(fixedList)؛
  // الناتج: [1، 1، 1، 1، 1]

  // تعديل العنصر الأول (الindex صفر)
```

```
fixedList[0] = 10؛
print(fixedList)؛
// الناتج: [10، 1، 1، 1، 1]

// تعديل العنصر الثاني (الindex واحد)
fixedList[1] = 50؛
print(fixedList)؛
// الناتج: [10، 50، 1، 1، 1]
{
```

ملاحظات مهمة

محاولة الإضافة

```
// محاولة الإضافة ستؤدي إلى خطأ
// fixedList.add(20)؛ // هذا سيسبب خطأ!
```

شرح الكود

- `List.filled(5، 1)` يعني:
  - إنشاء قائمة طولها 5 عناصر

- تعبئة كل العناصر بالقيمة 1

القيود

- لا يمكن:
  - إضافة عناصر جديدة
  - حذف عناصر موجودة
- يمكن فقط:
  - تعديل قيم العناصر الموجودة
  - الوصول إلى العناصر عبر الفهرسة

الخلاصة:

القوائم في Dart مرنة وقوية، وتوفر العديد من الخصائص مثل:

1. ثابتة الطول أو ديناميكية.
2. مرتبة ومكررة.
3. تتيح الوصول إلى العناصر باستخدام الفهرس.
4. تدعم العمليات مثل الإضافة والحذف والترتيب.
5. يمكن أن تحتوي على عناصر متداخلة وأنواع بيانات مختلفة.

السؤال السابع

"قم بإنشاء قائمة ديناميكية تحتوي على مزيج من أنواع البيانات المختلفة (مثل النصوص والأرقام والقوائم الفرعية). قم بتطبيق العمليات التالية:

1. **إضافة عنصر جديد إلى القائمة.**
2. **حذف عنصر بناءً على القيمة.**

3. **الوصول إلى عنصر معين باستخدام الفهرس وتعديله.**
4. **إضافة قائمة فرعية داخل القائمة الرئيسية.**
5. **ترتيب قائمة فرعية إذا كانت تحتوي على أرقام.**

الـ Mapهي واحدة من الهياكل الشائعة التي تُستخدم لتخزين بيانات على شكل أزواج مفتاح/قيمة..(Key/Value)

ما هو الـMap ؟

- **Map**هو هيكل بيانات يخزن البيانات في شكل زوج:
  - **المفتاح :(Key)** يتم استخدامه للوصول إلى القيمة.
  - **القيمة :(Value)** هي البيانات المرتبطة بالمفتاح.
- المفتاح يجب أن يكون **فريدًا**، بينما يمكن أن تتكرر القيم.
- يمكن أن تكون المفاتيح والقيم أي نوع من البيانات، مثل `int` أو `String`أو حتى كائنات.

إنشاءMap

يمكن إنشاء الـ Map بطرق متعددة:

1. **باستخدامLiterals**
2. عندما تُنشئ Map باستخدام **Literal**، فأنت تستخدم الأقواس { } لتحديد الأزواج مباشرة، كما في الأمثلة التالية:

```dart
void main() {
  var myMap = {
    'name': 'Ali',
    'age': 25,
    'country': 'Egypt',
  };

  print(myMap); // {name: Ali, age: 25, country: Egypt}
}
```

2باستخدامConstructor

```dart
void main() {
  var myMap = Map();
  myMap['name'] = 'Ali';
  myMap['age'] = 25;
```

```
                                print(myMap); // {name: Ali, age: 25}
                                                                    }
```

العمليات الأساسية على الـMap

إضافة عناصر

```
void main() {
  var myMap = {};
  myMap['city'] = 'Cairo';
  print(myMap); // {city: Cairo}
}
```

.الوصول إلى القيم

```
void main() {
  var myMap = {'name': 'Ali', 'age': 25};
  print(myMap['name']); // Ali
}
```

3.التحقق من وجود مفتاح أو قيمة

- **`containsKey()`**: للتحقق مما إذا كان المفتاح موجودًا.
- **`containsValue()`**: للتحقق مما إذا كانت القيمة موجودة.

```
• void main() {
    var myMap = {'name': 'Ali', 'age': 25};
    print(myMap.containsKey('name')); // true
    print(myMap.containsValue(30));   // false
  }
```

4.حذف عنصر

```
5. void main() {
     var myMap = {'name': 'Ali', 'age': 25};
     myMap.remove('age');
```

```
    print(myMap); // {name: Ali}
  }
```

6. تحديث القيم

```
void main() {
  var myMap = {'name': 'Ali', 'age': 25};
  myMap['age'] = 26;
  print(myMap); // {name: Ali, age: 26}
}
```

خصائصMap

keys

إرجاع قائمة تحتوي على جميع المفاتيح.

```
void main() {
  var myMap = {'name': 'Ali', 'age': 25};
  print(myMap.keys); // (name, age)
}
```

. values:

إرجاع قائمة تحتوي على جميع القيم.

```
void main() {
  var myMap = {'name': 'Ali', 'age': 25};
  print(myMap.values); // (Ali, 25)
}
```

3. length:

إرجاع عدد الأزواج (المفتاح/القيمة) الموجودة.

```dart
void main() {
  var myMap = {'name': 'Ali', 'age': 25};
  print(myMap.length); // 2
}
```

isNotEmpty: وisEmpty

للتحقق إذا كان الـ Map فارغًا أو لا.

```dart
void main() {
  var myMap = {};
  print(myMap.isEmpty); // true
  print(myMap.isNotEmpty); // false
}
```

لتكرار على الـMap

يمكنك التكرار على الأزواج باستخدام **forEach** أو الحلقات.

باستخدام: forEach

```dart
void main() {
  var myMap = {'name': 'Ali', 'age': 25};
  myMap.forEach((key, value) {
    print('$key: $value');
  });
  // Output:
  // name: Ali
  // age: 25
}
```

باستخدام for-in:

```dart
void main() {
  var myMap = {'name': 'Ali', 'age': 25};
  for (var key in myMap.keys) {
    print('$key: ${myMap[key]}');
  }
}
```

نواع الـMap

فيDart ، يمكن تحديد نوع المفتاح والقيمة باستخدام:Generics

عام. Map

```dart
void main() {
  Map<String, int> ageMap = {
    'Ali': 25,
    'Ahmed': 30,
  };
  print(ageMap); // {Ali: 25, Ahmed: 30}
}
```

دوال متقدمة فيMap

1. addAll()

لإضافة عناصر من Map آخر.

```dart
void main() {
  var myMap = {'name': 'Ali'};
  myMap.addAll({'age': 25, 'city': 'Cairo'});
```

```
  print(myMap); // {name: Ali, age: 25, city: Cairo}
}
```

clear()

لحذف جميع العناصر.

```
void main() {
  var myMap = {'name': 'Ali', 'age': 25};
  myMap.clear();
  print(myMap); // {}
}
```

3. map()

لتطبيق دالة على كل زوج وإرجاع Map جديد.

```
void main() {
  var myMap = {'a': 1, 'b': 2};
  var newMap = myMap.map((key, value) => MapEntry(key, value *
2));
  print(newMap); // {a: 2, b: 4}
}
```

ما هي MapEntry؟

- **MapEntry** هو كائن يمثل زوجًا من المفتاح والقيمة (Key-Value Pair) داخل الخريطة (Map).
- يُستخدم عند الحاجة إلى إنشاء إدخالات جديدة أثناء تحويل أو تعديل محتويات خريطة موجودة باستخدام دالة مثل **map**.

مثال 2: البحث فيMap

```dart
void main() {
  var myMap = {'Ali': 25, 'Ahmed': 30, 'Sara': 22};

  String name = 'Ahmed';
  if (myMap.containsKey(name)) {
    print('$name is ${myMap[name]} years old.');
  } else {
    print('$name not found.');
  }
}
```

اكتب برنامجًا باستخدام لغة Dart يقوم بما يلي:

1. إنشاء Map يحتوي على أسماء موظفين كـ Keys ورواتبهم كـ Values.على سبيل المثال:

```
{
  'Ali': 3000,
 'Sara': 4000,
'Ahmed': 3500,
 'Laila': 4500
}
```

2. أضف موظفًا جديدًا) على سبيل المثال 'Khaled' :براتب 5000 (إلى الـMap.
3. تحقق إذا كان هناك موظف باسم 'Ali' موجود في الـ Map. إذا كان موجودًا، اطبع راتبه.

4. قم بحذف موظف باسم 'Ahmed' من الـMap.
5. اطبع قائمة بجميع أسماء الموظفين (Keys) وقائمة بجميع الرواتب (Values)بشكل منفصل.
6. قم بإنشاء Map جديد يحتوي على نفس الموظفين ولكن مع زيادة 10% على رواتبهم.

Set هي نوع بيانات يُستخدم لتخزين مجموعة من العناصر الفريدة. بمعنى أن **Set** لا يسمح بوجود قيم مكررة. يتم استخدامه عندما تريد التأكد من أن كل عنصر في المجموعة يظهر مرة واحدة فقط.

إنشاء Set

يمكن إنشاء **Set** في Dart باستخدام القيم مباشرة (Literal) أو باستخدام **Constructor**.

```
void main() {
  // إنشاء Set باستخدام Literal
  var mySet = {1, 2, 3, 4};

  // عرض العناصر
  print(mySet); // {1, 2, 3, 4}

  // إضافة عنصر جديد
  mySet.add(5);
  print(mySet); // {1, 2, 3, 4, 5}

  // إضافة محاولة عنصر موجود بالفعل
  mySet.add(3);
  print(mySet); // {1, 2, 3, 4, 5} (لأن إضافة تتم لم العنصر
القيمة 3 موجودة بالفعل)

  // إزالة عنصر
  mySet.remove(2);
  print(mySet); // {1, 3, 4, 5}
}
```

إنشاء Set باستخدام Constructor

إذا كنت تريد إنشاء **Set** فارغًا ثم إضافة العناصر لاحقًا:

```
void main() {
  // إنشاء Set فارغ باستخدام Constructor
  var emptySet = <int>{}; // Set من النوع int
  emptySet.add(10);
  emptySet.add(20);
  print(emptySet); // {10, 20}
}
```

العمليات على Set

يمكنك إجراء العديد من العمليات على المجموعات مثل الاتحاد، التقاطع، والفرق.

مثال على العمليات:

```
void main() {
  var setA = {1, 2, 3, 4};
  var setB = {3, 4, 5, 6};

  // الاتحاد (Union): يجمع كل العناصر من المجموعتين
  var unionSet = setA.union(setB);
  print(unionSet); // {1, 2, 3, 4, 5, 6}

  // التقاطع (Intersection): العناصر المشتركة بين المجموعتين
  var intersectionSet = setA.intersection(setB);
  print(intersectionSet); // {3, 4}

  // الفرق (Difference): العناصر الموجودة في setA وليست في setB
```

```
  var differenceSet = setA.difference(setB);
  print(differenceSet); // {1, 2}
}
```

أهم الدوال المستخدمة مع Set

1. **add(element):** لإضافة عنصر.
2. **remove(element):** لإزالة عنصر.
3. **contains(element):** يتحقق ما إذا كان العنصر موجودًا.

```
print(setA.contains(2)); // true
```

4. **length:** يعيد عدد العناصر.

```
print(setA.length); // 4
```

5. **isEmpty / isNotEmpty:** للتحقق إذا كانت المجموعة فارغة أم لا.

مقارنة بين Set و List

الميزة	Set	List
تكرار العناصر	لا يسمح بالعناصر المكررة	يسمح بالعناصر المكررة
الترتيب	غير مرتب	مرتب

الميزة	Set	List
الأداء في البحث	سريع	أبطأ

Setمفيدة جدًا عندما تحتاج إلى مجموعة عناصر فريدة، مثل قائمة بأسماء الطلاب بدون تكرار، أو مجموعة خيارات فريدة في تطبيق.

قارن بين set, Map من خلال كود dart؟

في لغة **Dart**، الكلمة المفتاحية **switch** تُستخدم لتنفيذ أحد الفروع المختلفة بناءً على قيمة معينة. تُعتبر بديلًا عن استخدام **if-else** عندما تكون هناك العديد من الحالات المرتبطة بقيمة معينة.

```
void main() {
  var day = 'Monday';

  switch (day) {
    case 'Monday':
      print('It\'s the first day of the week!');
      break;

    case 'Friday':
      print('It\'s the weekend!');
      break;

    default:
      print('It\'s a normal day.');
      break;
  }
}
```

هذا الكود يستخدم جملة **switch** لفحص قيمة المتغير day وطباعة رسالة بناءً على اليوم الذي يمثله.

Case هنا تعني في حاله تواقق الشرط

قم بالامر التالي

break ضروري: ومعناها اذا تحقق الشرط توقف

إذا لم تضف break في نهاية الحالة، فسيستمر تنفيذ الكود في باقي الحالات (ما يُعرف بـ **fall-through behavior**). ولكن Dart لا يسمح بذلك إلا في حالات خاصة.

اكتب برنامجًا بـ **Dart** يُطالب المستخدم بإدخال رقم من 1 إلى 7. الرقم يُمثل أيام الأسبوع (1 = الأحد، 2 = الاثنين، ... 7 = السبت). استخدم جملة **`switch`** لطباعة اسم اليوم المقابل للرقم. إذا أدخل المستخدم رقمًا غير صالح (ليس بين 1 و 7)، اطبع رسالة "رقم غير صالح.؟

في لغة **Dart**، جملة **for loop** تُستخدم لتكرار تنفيذ كتلة من التعليمات لعدد محدد من المرات.

```
for (initialization; condition; increment/decrement) {
// الكود الذي يتم تكراره
}
```

شرح التركيب:

1. **initialization التهيئة**
   - تُستخدم لتعريف وتهيئة متغير التحكم في الحلقة.
   - يتم تنفيذها مرة واحدة فقط عند بدء الحلقة.
2. **condition الشرط:**
   - شرط يستمر التكرار طالما كان صحيحًا.(`true`)
   - إذا أصبح الشرط خاطئًا(`false`) ، تنتهي الحلقة.
3. **increment/decrement الزيادة/النقصان:**
   - يتم تنفيذ هذه العبارة في نهاية كل تكرار.
   - تُستخدم عادةً لتحديث متغير التحكم في الحلقة.

4.
```
void main() {
  for (int i = 1; i <= 5; i++) {
    print('Iteration $i');
  }
}
```

شرح الكود:

`int i = 1;`

- يتم تهيئة المتغير `i` بقيمة 1 عند بدء الحلقة.

2. **`i <= 5;`**
   - تستمر الحلقة طالما أن قيمة `i` أقل من أو تساوي 5.

3. **`i++;`**
   - يتم زيادة قيمة `i` بمقدار 1 بعد كل تكرار.

لتكرار على العناصر:(For-each Loop)

عند العمل مع قوائم أو مجموعات، يمكنك استخدام **`for`** لتكرار كل عنصر.

```
void main() {
  var numbers = [10, 20, 30, 40];
  for (var num in numbers) {
    print(num);
  }
}
```

شرح الكود:

- **`numbers`**: قائمة تحتوي على القيم.
- **`for (var num in numbers)`**: لكل عنصر في القائمة، قم بتعيينه إلى `num` وقم بتنفيذ الكود.

- اكتب برنامجًا بـ **Dart** يقوم بطباعة جدول الضرب لعدد 128 يُدخله المستخدم. استخدم جملة `for` لتنفيذ ذلك

في لغة **Dart**، الكائن (Object) هو أحد أهم المفاهيم الأساسية Dart .هي لغة كائنية التوجه(Object-Oriented Language) ، مما يعني أن كل شيء في Dart عبارة عن كائن، سواء كان رقمًا، نصًا، أو حتى كلاسًا مخصصًا.

ما هو الكائن(Object) ؟

الكائن هو نسخة من الكلاس (Class) يتم إنشاؤها في الذاكرة، ويمثل **مجموعة من البيانات (الخصائص)** و**السلوكيات (الدوال)**.
يمكن للكائنات أن تتفاعل مع بعضها البعض من خلال استدعاء الوظائف (الدوال) أو تعديل الخصائص.

ما هو الكلاس(Class) ؟

الكلاس هو بمثابة القالب (Template) الذي يُستخدم لإنشاء الكائنات. يحتوي الكلاس على:

1. **الخصائص (Properties):** وهي بيانات الكائن.
2. **الدوال (Methods):** وهي السلوكيات أو الأفعال التي يمكن للكائن تنفيذها.

أمثله

1. تعريف الكلاس:

```
class Car {
  String brand = '';
  int speed = 0;

  void drive() {
    print('$brand is driving at $speed km/h');
  }
}
```

```dart
void main() {
  // إنشاء كائن من النوع Car
  var myCar = Car();

  // تعديل الخصائص
  myCar.brand = 'Toyota';
  myCar.speed = 120;

  // استدعاء الدوال
  myCar.drive(); // Toyota is driving at 120 km/h
}
```

ملاحظات حول الكائنات في Dart:

1. كل شيء كائن:

- Dart تعتبر كل شيء كائنًا بما في ذلك الأنواع البدائية مثل الأرقام والنصوص.

الدوال المُنشئة (Constructors):

2. تستخدم لإنشاء الكائنات وتهيئة قيم الخصائص عند إنشائها.

```dart
class Car {
  String brand;
  int speed;

  // دالة منشئة
  Car(this.brand, this.speed);

  void drive() {
    print('$brand is driving at $speed km/h');
  }
}

void main() {
  var car = Car('Tesla', 180);
```

```
  car.drive(); // Tesla is driving at 180 km/h
}
```

-

أهمية الـ Constructor في لغةDart

الدالة **Constructor**هي دالة خاصة في الكلاس يتم استدعاؤها تلقائيًا عند إنشاء كائن (Object) جديد من الكلاس. الغرض الأساسي منها هو **تهيئة الخصائص (Properties)** للكائن أو **تنفيذ أي عملية إعداد (Initialization)** ضرورية عند إنشاء الكائن.

ما هي فوائد الـConstructor ؟

1. **تهيئة الخصائص عند إنشاء الكائن:**
   - يوفر الـ Constructor طريقة مباشرة لتحديد قيم الخصائص للكائن الجديد.
   - بدلاً من تعديل الخصائص يدويًا بعد إنشاء الكائن، يمكنك تمرير القيم أثناء الإنشاء
   -

```
class Car {
  String brand;
  int speed;


  Car(this.brand, this.speed);

  void drive() {
    print('$brand is driving at $speed km/h');
  }
}

void main() {
  var myCar = Car('Honda', 140); // مباشرة القيم تمرير
  myCar.drive(); // Honda is driving at 140 km/h
}
```

أمثلة على Parameterized Constructorفي Dart

Parameterized Constructor هو دالة منشئة (Constructor) تقبل معلمات (Parameters) عند إنشاء الكائن، مما يسمح لك بتمرير القيم مباشرةً عند إنشاء الكائن لتهيئة خصائصه.

كلاس سيارة(Car)

```
class Car {
  String brand;
  int speed;
// Constructorارسال هذه القيم داخل الي

  // Parameterized Constructor
  Car(this.brand, this.speed);

  void drive() {
    print('$brand is driving at $speed km/h');
  }
}

void main() {
  var myCar = Car('Toyota', 120); //
// myCar نستطيع من خلاله الوصول الي car من objectهو عياره عن
البيانات الموجوده في الكلاس وتمرير القيم لها
  myCar.drive(); // Toyota is driving at 120 km/h

  var anotherCar = Car('Honda', 140);
  anotherCar.drive(); // Honda is driving at 140 km/h
}
```

كلاس شخص(Person)

```
class Person {
  String name;
  int age;
   // Constructor ارسال البيانات داخل
  // Parameterized Constructor
```

```
  Person(this.name, this.age);

  void introduce() {
    print('Hi, my name is $name and I am $age years old.');
  }
}

void main() {
  var person1 = Person('Ahmed', 25);
// person1  عن عباره هوobject  الكلاس منPerson  لنستطيع الوصول
الي البيانات الموجوده داخله
  person1.introduce(); // Hi, my name is Ahmed and I am 25 years
old.

  var person2 = Person('Sara', 30);
  person2.introduce(); // Hi, my name is Sara and I am 30 years
old.
}
```

كلاس (Point)

```
class Point {
  int x;
  int y;

  // Parameterized Constructor
  Point(this.x, this.y);

  void display() {
    print('Point: ($x, $y)');
  }
}

void main() {
  var point1 = Point(3, 4);
  point1.display(); // Point: (3, 4)

  var point2 = Point(10, 15);
  point2.display(); // Point: (10, 15)
}
```

مثال بسيط: تمرير دالة إلىConstructor

```
class ActionHandler {
  // دالة لتخزين قيمة
  Function action;

  // Constructor يستقبل الدالة
  ActionHandler(this.action);

  //  تستدعي الدالة
  void executeAction() {
    print('Executing the action...');
    action(); // استدعاء الدالة الممررة
    print('Action executed successfully!');
  }
}
```

```
void main() {
  var handler = ActionHandler(() {
    print('Hello, this is the passed function!');
  });

  // استدعاء دالة التنفيذ
  handler.executeAction();

  print('------');

  // تمرير دالة تقوم بحساب قيمة
  var calculateHandler = ActionHandler(() {
    int result = 5 + 3;
    print('The result of the calculation is: $result');
  });

// استدعاء دالة التنفيذ
  calculateHandler.executeAction();
}
```

إنشاء class يحتوي على متغير بقيمة ثابتة و من خلال constructor نمرر

Function

ليها ثلاث وظائف

1. زيادة الرقم.

2. تقليل الرقم.

3. إعادة الرقم إلى الصفر.