



Mobile Programming with Flutter

เริ่มพัฒนาแอปพลิเคชันบนมือถือด้วย Flutter



ทำความเข้าใจกับพื้นฐานการเขียน Flutter

“จริง ๆ แล้ว Flutter ไม่ใช่ภาษาโปรแกรม
แต่เป็น **Framework** ที่ใช้ในการ
สร้าง **User Interface**”

“จริง ๆ แล้ว Flutter ไม่ใช่ภาษาโปรแกรม
แต่เป็น **Framework** ที่ใช้ในการ
สร้าง **User Interface**”

“ภายในจะประกอบไปด้วย **Package** และ
Function ที่จำเป็นต่าง ๆ ในการเขียน code”

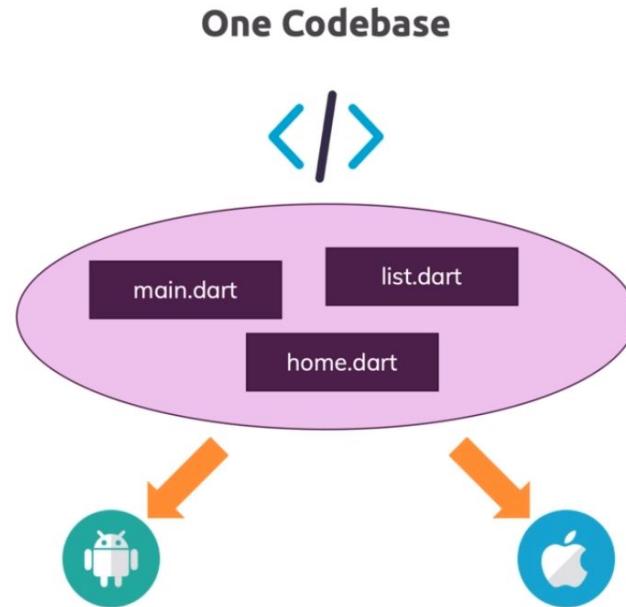
และ ถ้าตามว่า แล้วถ้าอยากเขียน App Flutter ต้องใช้ภาษาอะไรเขียน ?



Dart



Dart





มาเริ่มเขียน Dart ไปพร้อมกัน

A screenshot of a web browser window titled "DartPad". The address bar shows the URL <https://dartpad.dev>. The main content area displays the DartPad interface with the following elements:

- DartPad logo** and **DartPad title**.
- Create** button.
- Create with Gemini** button.
- Samples** button.
- Code editor** containing the following Dart code:

```
1 void main() {  
2     print('Hello, World!');  
3 }
```
- Tool icons** (Question mark, Refresh, Copy, Paste) and a **Run** button.
- Output panel** displaying the output of the code: **Hello, World!**

main() = จุดเริ่มรันโปรแกรม

print() = สั่งแสดงข้อความใน **console**

Variables

การประกาศตัวแปร

```
1 var name = 'Voyager I';
2 var year = 1977;
3 var flybyObjects = ['Jupiter', 'Saturn'];
4
```

Data Types & Built-in Functions

ใช้ var ให้ Dart เดา type ให้

Dart เป็น type-safe แต่ไม่ต้องประกาศ type ก็ได้

Print ตัวแปรพื้นฐาน (Beginner)

DartPad

Create Create with Gemini Samples

```
1 var name = 'Voyager I';
2 var year = 1977;
3
4 void main() {
5   print(name);
6 }
```

Voyager I

Quiz : แสดงค่าในตัวแปรที่เหลือ



DartPad

+ Create

★ Create with Gemini

≡+ Samples

```
1 var name = 'Voyager I';
2 var year = 1977;
3
4 void main() {
5   print(name);
6   print(year);
7 }
```



Run

Voyager I
1977

Print และ String Interpolation

Quiz : แสดงข้อความที่มีตัวแปรซ่อนอยู่ใน String

ชื่อยาน: Voyager I

เปิดตัวเมื่อปี: 1977

ข้อมูลเดิม: Voyager I เปิดตัวเมื่อปี 1977

DartPad

Create Create with Gemini Samples

```
1 var name = 'Voyager I';
2 var year = 1977;
3
4 void main() {
5   print('ชื่อยาน: $name');
6   print('เปิดตัวเมื่อปี: $year');
7   print('ข้อมูลเต็ม: $name เปิดตัวเมื่อปี $year');
8 }
9 |
```

?

Run

ชื่อยาน: Voyager I
เปิดตัวเมื่อปี: 1977
ข้อมูลเต็ม: Voyager I เปิดตัวเมื่อปี 1977

Quiz : Print แบบจัดรูปข้อความ (Formatting)

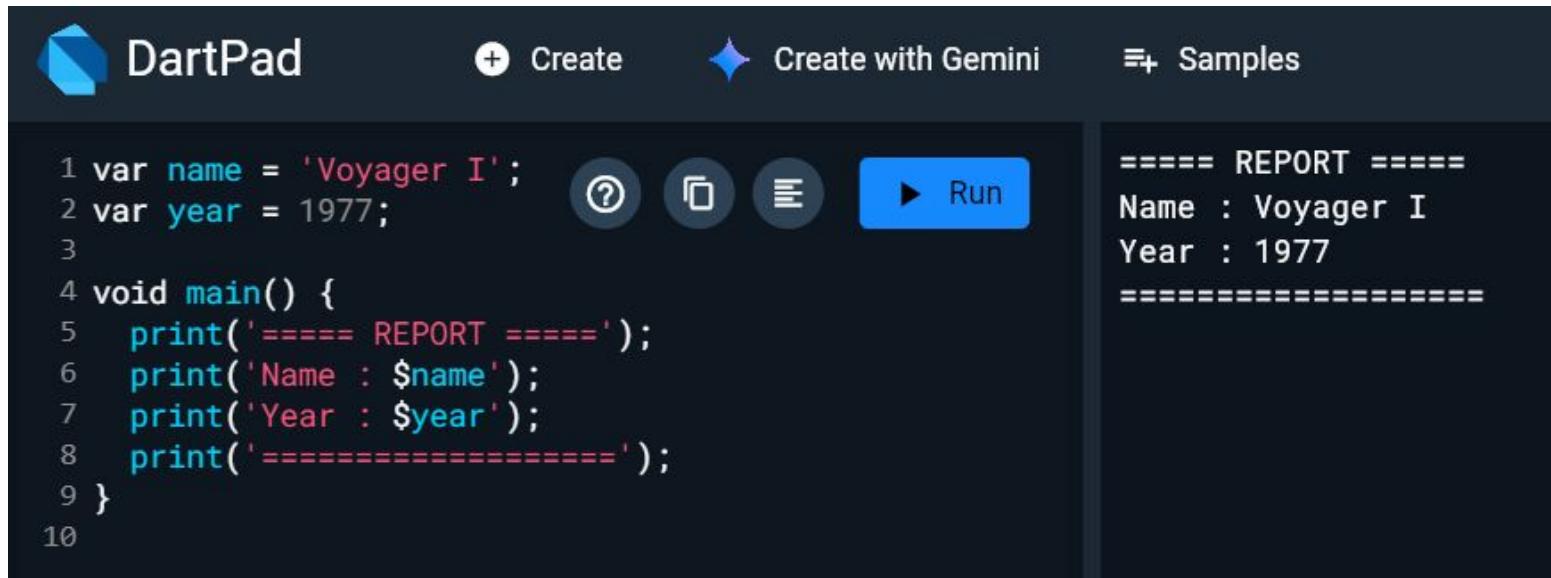
===== REPORT =====

Name : Voyager I

Year : 1977

=====

ลองทำให้มันแสดงออกมาแบบนี้หน่อย



DartPad

Create Create with Gemini Samples

```
1 var name = 'Voyager I';
2 var year = 1977;
3
4 void main() {
5   print('===== REPORT =====');
6   print('Name : $name');
7   print('Year : $year');
8   print('===== ');
9 }
10
```

Run

===== REPORT =====
Name : Voyager I
Year : 1977
=====

Quiz : ให้สร้างข้อความ debug

```
[INFO] Program started
[DEBUG] name = Voyager I
[DEBUG] year = 1977
[SUCCESS] All variables loaded
```

DartPad

Create Create with Gemini Samples

```
1 var name = 'Voyager I';
2 var year = 1977;
3
4 void main() {
5   print('[INFO] Program started');
6   print('[DEBUG] name = $name');
7   print('[DEBUG] year = $year');
8   print('[SUCCESS] All variables loaded');
9 }
10
```

Run

```
[INFO] Program started
[DEBUG] name = Voyager I
[DEBUG] year = 1977
[SUCCESS] All variables loaded
```

Print และ Expression

DartPad

Create Create with Gemini Samples

```
1 var name = 'Voyager I';
2 var year = 1977;
3
4 void main() {
5   print('อายุของยาน: ${2025 - year} ปี');
6 }
7
```

Run

อายุของยาน: 48 ปี

print การคำนวณกับที่

การประกาศตัวแปรใน Dart

Here's an example of creating a variable and initializing it:

```
var name = 'Bob';
```

Variables store references. The variable called `name` contains a reference to a `String` object with a value of "Bob".

The type of the `name` variable is inferred to be `String`, but you can change that type by specifying it. If an object isn't restricted to a single type, specify the `Object` type (or `dynamic` if necessary).

```
Object name = 'Bob';
```

Another option is to explicitly declare the type that would be inferred:

```
String name = 'Bob';
```

Null safety

Null safety prevents an error that results from unintentional access of variables set to `null`. The error is called a null dereference error. A null dereference error occurs when you access a property or call a method on an expression that evaluates to `null`. An exception to this rule is when `null` supports the property or method, like `toString()` or `hashCode`. With null safety, the Dart compiler detects these potential errors at compile time.

For example, say you want to find the absolute value of an `int` variable `i`. If `i` is `null`, calling `i.abs()` causes a null dereference error. In other languages, trying this could lead to a runtime error, but Dart's compiler prohibits these actions. Therefore, Dart apps can't cause runtime errors.

Null safety introduces three key changes:

1. When you specify a type for a variable, parameter, or another relevant component, you can control whether the type allows `null`. To enable nullability, you add a `?` to the end of the type declaration.

```
String? name // Nullable type. Can be `null` or string.
```

```
String name // Non-nullable type. Cannot be `null` but can be string.
```

Null safety

-
2. You must initialize variables before using them. Nullable variables default to `null`, so they are initialized by default. Dart doesn't set initial values to non-nullable types. It forces you to set an initial value. Dart doesn't allow you to observe an uninitialized variable. This prevents you from accessing properties or calling methods where the receiver's type can be `null` but `null` doesn't support the method or property used.
 3. You can't access properties or call methods on an expression with a nullable type. The same exception applies where it's a property or method that `null` supports like `hashCode` or `toString()`.

Sound null safety changes potential **runtime errors** into **edit-time** analysis errors. Null safety flags a non-null variable when it has been either:

- Not initialized with a non-null value.
- Assigned a `null` value.

This check allows you to fix these errors *before* deploying your app.

Default value

Uninitialized variables that have a nullable type have an initial value of `null`. Even variables with numeric types are initially null, because numbers—like everything else in Dart—are objects.

```
int? lineCount;  
assert(lineCount == null);
```

ℹ Note: Production code ignores the `assert()` call. During development, on the other hand, `assert(condition)` throws an exception if *condition* is false. For details, check out [Assert](#).

Default value

With null safety, you must initialize the values of non-nullable variables before you use them:

```
int lineCount = 0;
```

You don't have to initialize a local variable where it's declared, but you do need to assign it a value before it's used. For example, the following code is valid because Dart can detect that `lineCount` is non-null by the time it's passed to `print()`:

```
int lineCount;

if (weLikeToCount) {
    lineCount = countLines();
} else {
    lineCount = 0;
}

print(lineCount);
```

Top-level and class variables are lazily initialized; the initialization code runs the first time the variable is used.

Default value

```
int lineCount;

if (weLikeToCount) {
    lineCount = countLines();
} else {
    lineCount = 0;
}

print(lineCount);
```

Late variable

Late variables

The `late` modifier has two use cases:

- Declaring a non-nullable variable that's initialized after its declaration.
- Lazily initializing a variable.

Often Dart's control flow analysis can detect when a non-nullable variable is set to a non-null value before it's used, but sometimes analysis fails. Two common cases are top-level variables and instance variables: Dart often can't determine whether they're set, so it doesn't try.

If you're sure that a variable is set before it's used, but Dart disagrees, you can fix the error by marking the variable as `late`:

```
late String description;

void main() {
  description = 'Feijoada!';
  print(description);
}
```

Late variable

⚠ If you fail to initialize a `late` variable, a runtime error occurs when the variable is used.

When you mark a variable as `late` but initialize it at its declaration, then the initializer runs the first time the variable is used. This lazy initialization is handy in a couple of cases:

- The variable might not be needed, and initializing it is costly.
- You're initializing an instance variable, and its initializer needs access to `this`.

In the following example, if the `temperature` variable is never used, then the expensive `readThermometer()` function is never called:

```
// This is the program's only call to readThermometer().  
late String temperature = readThermometer(); // Lazily initialized.
```

 main.dart

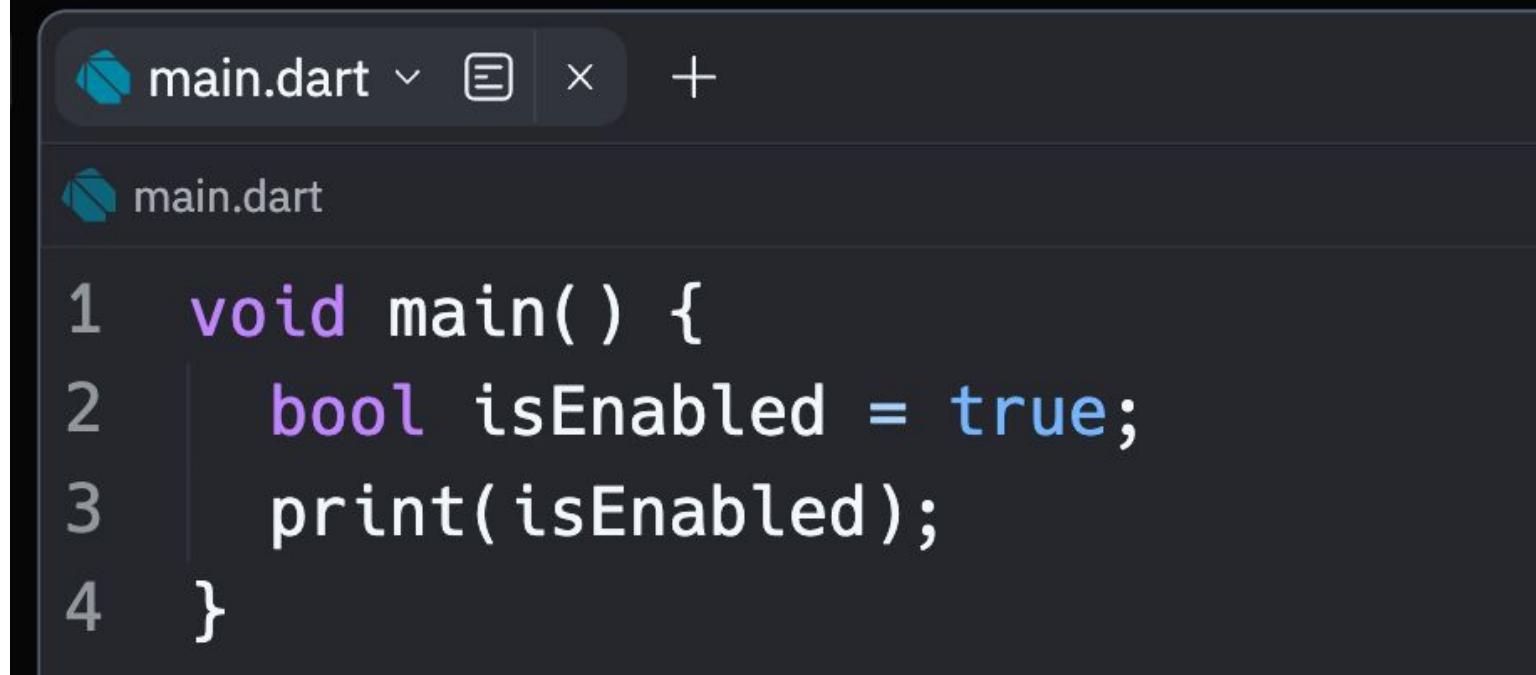
```
1 void main() {  
2     //Integer value  
3     int num1 = 1;  
4     //Declare a double value  
5     double num2 = 1.5;  
6     print(num1);  
7     print(num2);  
8 }
```

การใช้งานข้อมูลประเภทตัวเลข

The screenshot shows the DartPad interface. At the top, there are navigation links: 'DartPad' (with a logo), 'Create', 'Create with Gemini', and 'Samples'. Below the code area, there are icons for help, refresh, and copy, followed by a blue 'Run' button. To the right of the run button is the text 'borntoDev'. The code itself is a simple main function:

```
1 void main() {  
2     String band = 'borntoDev';  
3     print(band);  
4 }
```

การใช้งานข้อมูลประเภทข้อความ



The screenshot shows a dark-themed code editor window titled "main.dart". The code inside the editor is:

```
1 void main() {  
2     bool isEnabled = true;  
3     print(isEnabled);  
4 }
```

การใช้งานข้อมูลประเภทตรรกะ

DartPad

Create Create with Gemini Samples

```
1 void main() {  
2   bool isEnabled = True;  
3   print(isEnabled);  
4 }
```

Run

```
compileNewDDC  
main.dart:2:19: Error: Undefined name 'True'.  
    bool isEnabled = True;  
                      ^^^^
```

dart:core library

CLASSES

BigInt

bool

Comparable

DateTime

Deprecated

double

Duration

Enum

Expando

Finalizer

Function

Future

int

Invocation

Iterable

Iterator

List

Map

MapEntry

Match

Null

num

Object

Pattern

pragma

Record

RegExp

RegExpMatch

Runelitator

Runes

Set

Sink

StackTrace

Snapshot

List<E> class (abstract)

An indexable collection of objects with a length.

Subclasses of this class implement different kinds of lists. The most common kinds of lists are:

• Fixed-length list

An error occurs when attempting to use operations that can change the length of the list.

• Growable list

Full implementation of the API defined in this class.

The default growable list, as created by `[]`, keeps an internal buffer, and grows that buffer when necessary. This guarantees that a sequence of `add` operations will each execute in amortized constant time. Setting the length directly may take time proportional to the new length, and may change the internal capacity so that a following add operation will need to immediately increase the buffer capacity. Other list implementations may have different performance behavior.

Example of fixed-length list:

```
final fixedLengthList = List<int>.filled(5, 0); // Creates fixed-length list.
print(fixedLengthList); // [0, 0, 0, 0, 0]
fixedLengthList[0] = 87;
fixedLengthList.setAll(1, [1, 2, 3]);
print(fixedLengthList); // [87, 1, 2, 3, 0]
// Fixed length list length can't be changed or increased
fixedLengthList.length = 0; // Throws
fixedLengthList.add(499); // Throws
```

Example of growable list:

```
final growableList = <String>['A', 'B']; // Creates growable list.
```

To add data to the growable list, use `operator[]=`, `add` or `addAll`.

```
growableList[0] = 'G';
print(growableList); // [G, B]
growableList.add('X');
growableList.addAll(['C', 'B']);
print(growableList); // [G, B, X, C, B]
```

To check whether, and where, the element is in the list, use `indexOf` or `lastIndexOf`.

```
final indexA = growableList.indexOf('A'); // -1 (not in the list)
final firstIndexB = growableList.indexOf('B'); // 1
final lastIndexB = growableList.lastIndexOf('B'); // 4
```

To remove an element from the growable list, use `remove`, `removeAt`, `removeLast`, `removeRange`.

Example of fixed-length list:

```
final fixedLengthList = List<int>.filled(5, 0); // Creates fixed-length list.  
print(fixedLengthList); // [0, 0, 0, 0, 0]  
fixedLengthList[0] = 87;  
fixedLengthList.setAll(1, [1, 2, 3]);  
print(fixedLengthList); // [87, 1, 2, 3, 0]  
// Fixed length list length can't be changed or increased  
fixedLengthList.length = 0; // Throws  
fixedLengthList.add(499); // Throws
```

Example of growable list:

```
final growableList = <String>['A', 'B']; // Creates growable list.
```

To add data to the growable list, use `operator[]=`, `add` or `addAll`.

```
growableList[0] = 'G';  
print(growableList); // [G, B]  
growableList.add('X');  
growableList.addAll({'C', 'B'});  
print(growableList); // [G, B, X, C, B]
```

การใช้งานข้อมูล List

```
void main() {  
    var scores = [1, 3, 4, 2];  
    scores.remove(1);  
    print(scores);  
}
```

การใช้งานข้อมูล List

```
void main() {  
    var scores = [1, 3, 4, 2, 5];  
    print('Length: ${scores.length}');  
}
```

การใช้งานข้อมูล List

```
void main() {  
    var scores = [1, 3, 4, 2, 5];  
    //  
    print('First: ${scores.first}');  
    print('Last: ${scores.last}');  
}
```

การใช้งานข้อมูล List

```
void main() {  
    var scores = [];  
    print(scores.isEmpty); // true  
    print(scores.isNotEmpty); // false  
}
```

การใช้งานข้อมูล List

const = ตัวแปรที่กำหนดค่าได้ครั้งเดียว บังคับกำหนดค่าเริ่มต้น
แก้ไขไม่ได้

Static = ใช้แพรกายใน Class เดียวกัน

Final = ใช้กำหนดค่าได้ครั้งเดียว ต้องมีค่าเริ่มต้นเสมอ และ มีการ
กำหนดค่าให้ไม่สามารถแก้ไขได้ แต่ถ้าเป็น Ref/Obj Data type
จะสามารถแก้ไขค่าได้ แต่ไม่สามารถกำหนดค่าตัวแปรใหม่ได้

Final and const

If you never intend to change a variable, use `final` or `const`, either instead of `var` or in addition to a type. A final variable can be set only once; a const variable is a compile-time constant. (Const variables are implicitly final.)

i Note: Instance variables can be `final` but not `const`.

Here's an example of creating and setting a `final` variable:

```
final name = 'Bob'; // Without a type annotation
final String nickname = 'Bobby';
```

You can't change the value of a `final` variable:

x static analysis: error/warning

```
name = 'Alice'; // Error: a final variable can only be set once.
```

Final and const

Use `const` for variables that you want to be **compile-time constants**. If the `const` variable is at the class level, mark it `static const`. Where you declare the variable, set the value to a compile-time constant such as a number or string literal, a `const` variable, or the result of an arithmetic operation on constant numbers:

```
const bar = 1000000; // Unit of pressure (dynes/cm2)
const double atm = 1.01325 * bar; // Standard atmosphere
```

The `const` keyword isn't just for declaring constant variables. You can also use it to create constant *values*, as well as to declare constructors that *create* constant values. Any variable can have a constant value.

```
var foo = const [];
final bar = const [];
const baz = []; // Equivalent to `const []`
```

Final and const

You can omit `const` from the initializing expression of a `const` declaration, like for `baz` above. For details, see [DON'T use `const` redundantly](#).

You can change the value of a non-final, non-`const` variable, even if it used to have a `const` value:

```
foo = [1, 2, 3]; // Was const []
```

You can't change the value of a `const` variable:

```
x static analysis: error/warning
baz = [42]; // Error: Constant variables can't be assigned a value.
```

You can define constants that use [type checks and casts](#) (`is` and `as`), [collection if](#), and [spread operators](#) (`...` and `...?`):

```
const Object i = 3; // Where i is a const Object with an int value...
const list = [i as int]; // Use a typecast.
const map = {if (i is int) i: 'int'}; // Use is and collection if.
const set = {if (list is List<int>) ...list}; // ...and a spread.
```

Note: Although a `final` object cannot be modified, its fields can be changed. In comparison, a `const` object and its fields cannot be changed: they're *immutable*.

For more information on using `const` to create constant values, see [Lists](#), [Maps](#), and [Classes](#).

To check whether, and where, the element is in the list, use `indexOf` or `lastIndexOf`.

```
final indexA = growableList.indexOf('A'); // -1 (not in the list)
final firstIndexB = growableList.indexOf('B'); // 1
final lastIndexB = growableList.lastIndexOf('B'); // 4
```

To remove an element from the growable list, use `remove`, `removeAt`, `removeLast`, `removeRange` or `removeWhere`.

```
growableList.remove('C');
growableList.removeLast();
print(growableList); // [G, B, X]
```

To insert an element at position in the list, use `insert` or `insertAll`.

```
growableList.insert(1, 'New');
print(growableList); // [G, New, B, X]
```

การใช้งานข้อมูล List

To replace a range of elements in the list, use [fillRange](#), [replaceRange](#) or [setRange](#).

```
growableList.replaceRange(0, 2, ['AB', 'A']);  
print(growableList); // [AB, A, B, X]  
growableList.fillRange(2, 4, 'F');  
print(growableList); // [AB, A, F, F]
```

To sort the elements of the list, use [sort](#).

```
growableList.sort((a, b) => a.compareTo(b));  
print(growableList); // [A, AB, F, F]
```

To shuffle the elements of this list randomly, use [shuffle](#).

```
growableList.shuffle();  
print(growableList); // e.g. [AB, F, A, F]
```

To find the first element satisfying some predicate, or give a default value if none do, use [firstWhere](#).

```
bool isVowel(String char) => char.length == 1 && "AEIOU".contains(char);  
final firstVowel = growableList.firstWhere(isVowel, orElse: () => ''); // ''
```

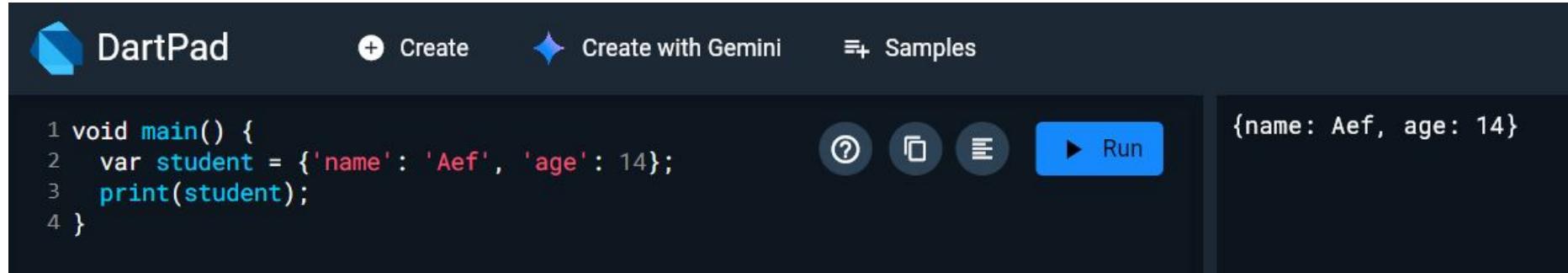
There are similar [lastWhere](#) and [singleWhere](#) methods.

A list is an [Iterable](#) and supports all its methods, including [where](#), [map](#), [whereType](#) and [toList](#).

Lists are [Iterable](#). Iteration occurs over values in index order. Changing the values does not affect iteration, but changing the valid indices—that is, changing the list's length—between iteration steps causes a [ConcurrentModificationError](#). This means that only growable lists can throw [ConcurrentModificationError](#). If the length changes temporarily and is restored before continuing the iteration, the iterator might not detect it.

It is generally not allowed to modify the list's length (adding or removing elements) while an operation on the list is being performed, for example during a call to [forEach](#) or [sort](#). Changing the list's length while it is being iterated, either by iterating it directly or through iterating an [Iterable](#) that is backed by the list, will break the iteration.

การใช้งานข้อมูล List



DartPad

Create Create with Gemini Samples

```
1 void main() {  
2   var student = {'name': 'Aef', 'age': 14};  
3   print(student);  
4 }
```

Run

{name: Aef, age: 14}

การใช้งานข้อมูล Map

ฟังก์ชันที่มักพบ และ ใช้งานได้บ่อย สำหรับ Map



```
1 void main() {  
2   var student = {'name': 'Aef', 'age': 14};  
3   print(student);  
4  
5   // เพิ่มข้อมูลใหม่ลงใน Map  
6   student.addAll({  
7     'school': 'ABC School',  
8     'grade': 8,  
9     'age': 15,           // ถ้าดีบี้ช้า จะเขียนทับค่าเดิม  
10    });  
11  
12   print(student);  
13 }  
14 |
```



Run

addAll()

```
{name: Aef, age: 14}  
{name: Aef, age: 15, school: ABC School, grade: 8}
```

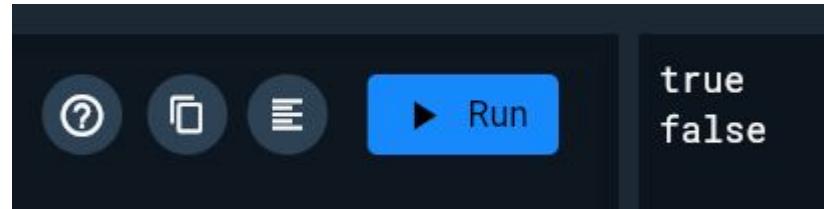
DartPad interface showing Dart code and its execution results.

The code in the editor:

```
1 void main() {  
2   var student = {'name': 'Aef', 'age': 14};  
3  
4   // ตรวจสอบว่ามี key ชื่อนี้หรือไม่  
5   print(student.containsKey('name')); // true  
6   print(student.containsKey('grade')); // false  
7 }  
8
```

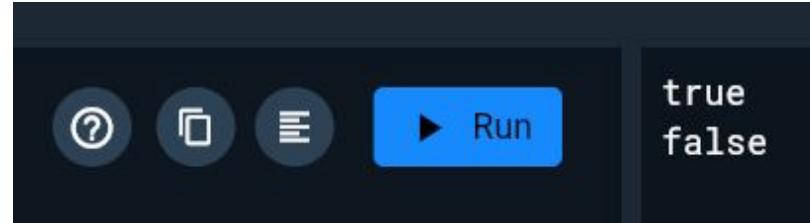
The run results panel shows the output of the `print` statements:

```
true  
false
```



containsKey()

```
1 void main() {  
2     var student = {'name': 'Aef', 'age': 14};  
3  
4     // ตรวจสอบว่ามี value นี้หรือไม่  
5     print(student.containsValue('Aef'));    // true  
6     print(student.containsValue(20));        // false  
7 }  
8
```



containsValue()

DartPad

Create Create with Gemini Samples

```
1 void main() {  
2   var student = {'name': 'Aef', 'age': 14};  
3   // ใช้ forEach() เพื่อวนลูป key และ value  
4   student.forEach((key, value) {  
5     print('$key : $value');  
6   });  
7 }  
8  
9 }
```

Run

name : Aef
age : 14

forEach()

ຕົວຢ່າງ forEach() ແລະ arrow function

DartPad

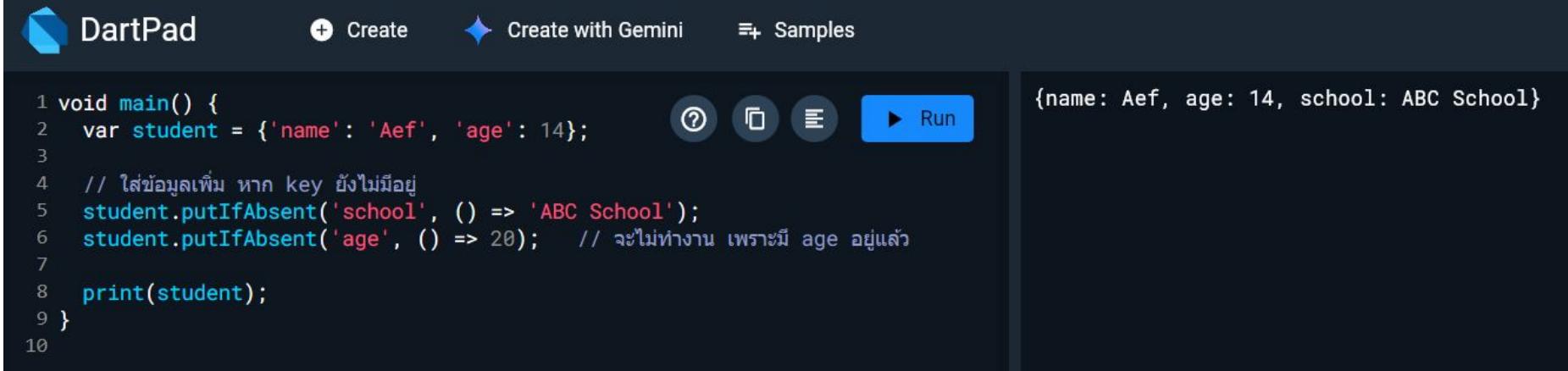
Create Create with Gemini Samples

```
1 void main() {  
2   var student = {'name': 'Aef', 'age': 14};  
3  
4   var label = {  
5     'name': 'ชื่อ',  
6     'age': 'อายุ',  
7   };  
8  
9   student.forEach(  
10     (key, value) => print('${label[key]} : $value'),  
11   );  
12 }  
13
```

Run

ชื่อ : Aef
อายุ : 14

forEach()



DartPad

Create Create with Gemini Samples

```
1 void main() {  
2   var student = {'name': 'Aef', 'age': 14};  
3  
4   // ใส่ข้อมูลเพิ่ม หาก key ยังไม่มีอยู่  
5   student.putIfAbsent('school', () => 'ABC School');  
6   student.putIfAbsent('age', () => 20);    // จะไม่ทำงาน เพราะมี age อยู่แล้ว  
7  
8   print(student);  
9 }  
10
```

Run

{name: Aef, age: 14, school: ABC School}

putIfAbsent()



```
1 void main() {  
2   var student = {'name': 'Aef', 'age': 14, 'school': 'ABC School'};  
3  
4   print(student); // ก่อนลบ  
5  
6   // ลบข้อมูลตามชื่อ key  
7   student.remove('school'); // ลบคีย์ school ออก  
8   student.remove('grade'); // ไม่มี key นี้ จึงไม่มีผลใด ๆ  
9  
10  print(student); // หลังลบ  
11 }  
12
```

```
{name: Aef, age: 14, school: ABC School}  
{name: Aef, age: 14}
```

remove()



DartPad

+ Create

◆ Create with Gemini

≡+ Samples

```
1 void main() {  
2     var student = {'name': 'Aef', 'age': 14, 'grade': 8, 'school': 'ABC School'};  
3  
4     print(student);    // ก่อนลบ  
5  
6     // ลบเฉพาะข้อมูลที่ value เป็นตัวเลข (int)  
7     student.removeWhere((key, value) => value is int);  
8  
9     print(student);    // หลังลบ  
10 }  
11
```

removeWhere()



```
1 void main() {  
2   var student = {'name': 'Aef', 'age': 14, 'grade': 8, 'school': 'ABC School'};  
3  
4   print({  
5     name: Aef, age: 14, grade: 8, school: ABC School}  
6     {name: Aef, school: ABC School})  
7   // ผลลัพธ์  
8   student.removeWhere((key, value) => value is int);  
9  
10  print(student); // หลังลบ  
11 }
```

removeWhere()



```
1 void main() {  
2     var student = {'name': 'Aef', 'age': 14, 'school': 'ABC School'};  
3  
4     print(student);    // ก่อนลบห้องหนด  
5  
6     // ลบข้อมูลห้องหนดใน Map  
7     student.clear();  
8  
9     print(student);    // หลังลบห้องหนด  
10 }  
11
```

clear()



```
1 void main() {  
2   var student = {'name': 'Aef', 'age': 14, 'school': 'ABC School'};  
3  
4   print(st {name: Aef, age: 14, school: ABC School}  
5   // ลบข้อมูล  
6   {}  
7   student.clear();  
8  
9   print(student); // หลังลบทั้งหมด  
10 }  
11
```

clear()



DartPad

Create

Create with Gemini

≡

```
1 void main() {  
2   var student = {'name': 'Aef', 'age': 14};  
3  
4   print(student); // ก่อนอัปเดต  
5  
6   // อัปเดตอายุ  
7   student.update('age', (value) => (value as int) + 1);  
8  
9   // อัปเดตชื่อใหม่  
10  student.update('name', (value) => 'Aef Sirasit');  
11  
12  print(student); // หลังอัปเดต  
13 }  
14
```

update()



DartPad

Create Create with Gemini

```
1 void main() {  
2   var student = {'name': 'Aef', 'age': 14};  
3  
4   print(student); // {name: Aef, age: 14}  
5   // อัปเดต name  
6   student.update('name', (value) => 'Aef Sirasit');  
7   print(student); // {name: Aef Sirasit, age: 14}  
8  
9   // อัปเดต age  
10  student.update('age', (value) => value + 1);  
11  
12  print(student); // {name: Aef Sirasit, age: 15}  
13}  
14
```

update()



DartPad

Create

Create with Gemini

Samples

```
1 void main() {  
2   var student = {'name': 'Aef', 'age': 14};  
3  
4   // สร้าง Map ใหม่จาก Mapเดิม  
5   var newStudent = Map.from(student);  
6  
7   print(student);  
8   print(newStudent);  
9 }  
10 |
```



Run

{name: Aef, age: 14}
{name: Aef, age: 14}

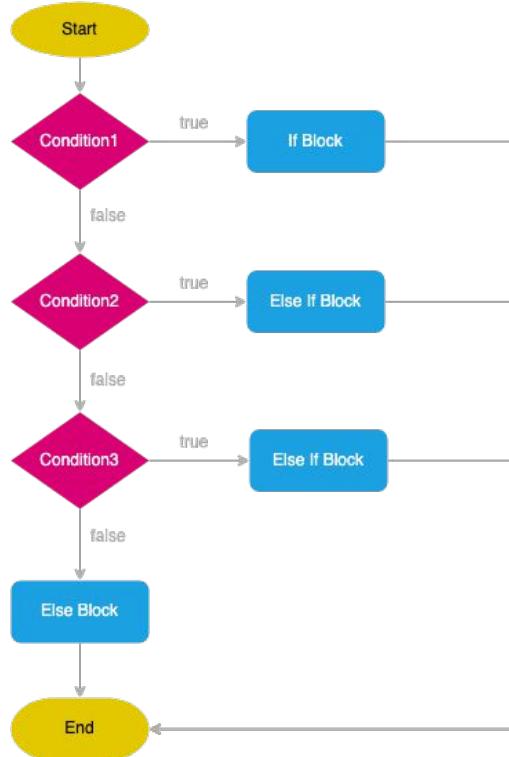
Map.from()



การใช้งาน conditional logic

```
if (condition1)
{
    // statement 1
}
else if (condition2)
{
    // statement 2
}
else if (condition3)
{
    // statement 3
}
else
{
    // else statement
}
```

การใช้เงื่อนไขใน Dart จะเป็น if, else if และ else



การใช้เงื่อนไขใน Dart จะเป็น if, else if และ else

```
1 void main() {  
2     var score = 49;  
3  
4     if (score > 0) {  
5         print('มีคะแนน');  
6     }  
7 }
```

Quiz 1 : พลัพร์ที่แสดงผลคือ ?

```
1 void main() {  
2     var score = 49;  
3  
4     if (score >= 50) {  
5         print('เกือบถูก');  
6     } else {  
7         print('เกือบผ่าน');  
8     }  
9 }
```

Quiz 2 : พลัพร์ที่แสดงผลคือ ?

```
1 void main() {  
2     var score = 49;  
3  
4     if (score >= 90) {  
5         print('เอา A ไปเลย');  
6     } else if (score >= 50) {  
7         print('เกี๊ยบผ่าน');  
8     } else {  
9         print('เกี๊ยบตก');  
10    }  
11 }
```

Quiz 3 : พลัพร์ที่แสดงผลคือ ?



การใช้งาน Loop ใน Dart

```
1 void main() {  
2     for (var i = 1; i <= 5; i++) {  
3         print('รอบที่ $i');  
4     }  
5 }
```

การใช้งาน Loop for สำหรับ Dart

```
1 void main() {  
2     var numbers = [10, 20, 30, 40, 50];  
3  
4     for (var i = 0; i < numbers.length; i++) {  
5         print('รอบที่ ${i + 1} = ${numbers[i]}');  
6     }  
7 }
```

การใช้งานข้อมูล List

```
1 void main() {  
2     var numbers = [10, 20, 30, 40, 50];  
3  
4     for (var i = 0; i < 3; i++) {  
5         print('รอบที่ ${i + 1} = ${numbers[i]}');  
6     }  
7 }  
8
```

```
รอบที่ 1 = 10  
รอบที่ 2 = 20  
รอบที่ 3 = 30
```

การใช้งาน Loop for สำหรับ Dart

```
1 void main() {  
2     var count = 1;  
3  
4     while (count <= 5) {  
5         print('รอบที่ $count');  
6         count++;  
7     }  
8 }
```

การใช้งาน Loop while สำหรับ Dart

```
1 void main() {  
2     var count = 1;  
3  
4     do {  
5         print('รอบที่ $count');  
6         count++;  
7     } while (count <= 5);  
8 }  
9 |
```

รอบที่ 1
รอบที่ 2
รอบที่ 3
รอบที่ 4
รอบที่ 5

การใช้งาน Loop do while สำหรับ Dart

```
1 void main() {  
2     List<String> fruits = ['apple', 'banana', 'cherry'];  
3     for (var fruit in fruits) {  
4         print(fruit);  
5     }  
6 }  
7 |
```

การใช้งาน Loop for-in สำหรับ Dart

```
void main() {  
    List<int> numbers = [1, 2, 3, 4, 5, 13, 7, 8, 9, 10];  
    for (var number in numbers) {  
        if (number % 13 == 0) {  
            print(number);  
            break;  
        }  
    }  
}
```

การใช้งาน break สำหรับ Dart



OOP with Dart

```
1 class Car {  
2   String brand;  
3  
4   Car(this.brand);  
5 }
```

การใช้งาน class ใน Dart

```
1 class Car {  
2     String brand;  
3  
4     Car(this.brand);  
5 }
```

**Quiz : จากความรู้ที่มี เราจะสร้าง Obj car
ใน main ได้อย่างไร ?**

```
class Car {  
    String brand;  
  
    Car(this.brand);  
}  
  
void main() {  
    Car mycar = new Car("BMW");  
    print(mycar.brand);  
}
```

**Quiz : จากความรู้ที่มี เราจะสร้าง Obj car
ใน main ได้อย่างไร ?**

```
class Car {  
    String brand;  
  
    Car(this.brand);  
}  
  
void main() {  
    var myCar = new Car('Toyota');  
    print(myCar.brand);  
}
```

เราสามารถใช้ var แทนได้เช่นกัน

```
class Car {  
    String brand;  
    Car(this.brand);  
}  
  
class ElectricCar extends Car {  
    ElectricCar(String brand) : super(brand);  
}
```

Quiz : จากความรู้ที่มี เราจะสร้าง Obj ElectricCar ใน main ได้อย่างไร ?

```
void main() {  
    var myNewCar = new ElectricCar("Tesla");  
    print(myNewCar.brand);  
}
```

Quiz : จากความรู้ที่มี เราจะสร้าง Obj ElectricCar ใน main ได้อย่างไร ?

```
class Tesla implements ElectricCar {  
    // Implement the interface here  
}
```

การใช้งาน interface ใน Dart

```
// Interface
abstract class Shape {
    double getArea();
}

// Implementing the Shape interface
class Square implements Shape {
    Square(this.side);
    final double side;

    @override
    double getArea() => side * side;
}

class Circle implements Shape {
    Circle(this.radius);
    final double radius;

    @override
    double getArea() => 3.14 * radius * radius;
}
```

ตัวอย่าง 1 : การใช้งาน interface ใน Dart

```
// Interface
abstract class Shape {
    double getArea();
}

// Implementing the Shape interface
class Square implements Shape {
    Square(this.side);
    final double side;

    @override
    double getArea() => side * side;
}

class Circle implements Shape {
    Circle(this.radius);
    final double radius;

    @override
    double getArea() => 3.14 * radius * radius;
}
```

Quiz ຕົວອຍ່າງ 1 : ເຮັດໃຊ້ຢັງໄຟກະ ?

```
// Interface
abstract class Shape {
    double getArea();
}

// Implementing the Shape interface
class Square implements Shape {
    Square(this.side);
    final double side;

    @override
    double getArea() => side * side;
}

class Circle implements Shape {
    Circle(this.radius);
    final double radius;

    @override
    double getArea() => 3.14 * radius * radius;
}
```

```
void main(){
    var myCircle = new Circle(10);
    print(myCircle.getArea());
}
```

314

Quiz ตัวอย่าง 1 : เรียกใช้ยังไงนะ ?

```
1 // Interface
2 abstract class Vehicle {
3     void move();
4     void stop();
5 }
6
7 // Implementing the Vehicle interface
8 class Car implements Vehicle {
9     @override
10    void move() {
11        print('Car is moving');
12    }
13
14    @override
15    void stop() {
16        print('Car has stopped');
17    }
18 }
19
```

```
19
20 class Bike implements Vehicle {
21     @override
22     void move() {
23         print('Bike is moving');
24     }
25
26     @override
27     void stop() {
28         print('Bike has stopped');
29     }
30 }
```

ตัวอย่าง 2 : การใช้งาน interface ใน Dart

```
1 // Interface
2 abstract class Vehicle {
3     void move();
4     void stop();
5 }
6
7 // Implementing the Vehicle interface
8 class Car implements Vehicle {
9     @override
10    void move() {
11        print('Car is moving');
12    }
13
14    @override
15    void stop() {
16        print('Car has stopped');
17    }
18 }
19
```

```
20 class Bike implements Vehicle {
21     @override
22     void move() {
23         print('Bike is moving');
24     }
25
26     @override
27     void stop() {
28         print('Bike has stopped');
29     }
30 }
```

Quiz ตัวอย่าง 2 : เรียกให้รุณนต์วิ่งยังไงละ ?

```
1 // Interface
2 abstract class Vehicle {
3     void move();
4     void stop();
5 }
6
7 // Implementing the Vehicle interface
8 class Car implements Vehicle {
9     @override
10    void move() {
11        print('Car is moving');
12    }
13
14    @override
15    void stop() {
16        print('Car has stopped');
17    }
18 }
19
```

```
20 class Bike implements Vehicle {
21     @override
22     void move() {
23         print('Bike is moving');
24     }
25
26     @override
27     void stop() {
28         print('Bike has stopped');
29     }
30 }
```

```
void main(){
    var myCar = new Car();
    myCar.move();
}
```

Quiz ตัวอย่าง 2 : เรียกให้รุณนัตวิ่งยังไงละ ?

```
main.dart +  
  
main.dart  
  
1 // Interface  
2 abstract class Vehicle {  
3     void move();  
4     void stop();  
5 }  
6  
7 // Implementing the Vehicle interface  
8 class Car implements Vehicle {  
9     @override  
10    void move() {  
11        print('Car is moving');  
12    }  
13  
14    @override  
15    void stop() {  
16        print('Car has stopped');  
17    }  
18 }  
19
```

```
19  
20 class Bike implements Vehicle {  
21     @override  
22     void move() {  
23         print('Bike is moving');  
24     }  
25  
26     @override  
27     void stop() {  
28         print('Bike has stopped');  
29     }  
30 }
```

Car is moving

```
void main(){  
    var myCar = new Car();  
    myCar.move();  
}
```

Quiz ตัวอย่าง 2 : เรียกให้รถยนต์วิ่งยังไงนะ ?

```
1 // Interface
2 abstract class Printer {
3     void printDocument();
4     void scanDocument();
5 }
6
7 // Implementing the Printer interface
8 class LaserPrinter implements Printer {
9     @override
10    void printDocument() {
11        print('Printing document with Laser Printer');
12    }
13
14    @override
15    void scanDocument() {
16        print('Scanning document with Laser Printer');
17    }
18 }
```

```
20 class InkjetPrinter implements Printer {
21     @override
22     void printDocument() {
23         print('Printing document with Inkjet Printer');
24     }
25
26     @override
27     void scanDocument() {
28         print('Scanning document with Inkjet Printer');
29     }
30 }
```

ตัวอย่าง 3 : การใช้งาน interface ใน Dart

main.dart

```
1 // Interface
2 abstract class Printer {
3     void printDocument();
4     void scanDocument();
5 }
6
7 // Implementing the Printer interface
8 class LaserPrinter implements Printer {
9     @override
10    void printDocument() {
11        print('Printing document with Laser Printer');
12    }
13
14    @override
15    void scanDocument() {
16        print('Scanning document with Laser Printer');
17    }
18 }
```

```
20 class InkjetPrinter implements Printer {
21     @override
22     void printDocument() {
23         print('Printing document with Inkjet Printer');
24     }
25
26     @override
27     void scanDocument() {
28         print('Scanning document with Inkjet Printer');
29     }
30 }
```

Quiz ຕົວອຍ່າງ 3 : ໃຊ້ InkjetPrinter ຍັງໄວ ?

main.dart

```
1 // Interface
2 abstract class Printer {
3     void printDocument();
4     void scanDocument();
5 }
6
7 // Implementing the Printer interface
8 class LaserPrinter implements Printer {
9     @override
10    void printDocument() {
11        print('Printing document with Laser Printer');
12    }
13
14    @override
15    void scanDocument() {
16        print('Scanning document with Laser Printer');
17    }
18 }
```

```
20 class InkjetPrinter implements Printer {
21     @override
22     void printDocument() {
23         print('Printing document with Inkjet Printer');
24     }
25
26     @override
27     void scanDocument() {
28         print('Scanning document with Inkjet Printer');
29     }
30 }
```

```
33 void main( ){
34     var myPrinter = new InkjetPrinter();
35     myPrinter.printDocument();
36 }
```

Quiz ຕົວອຍ່າງ 3 : ໃຊ້ InkjetPrinter ຍັງໄວ ?

```
main.dart x +  
main.dart  
  
1 // Interface  
2 abstract class Printer {  
3   void printDocument();  
4   void scanDocument();  
5 }  
6  
7 // Implementing the Printer interface  
8 class LaserPrinter implements Printer {  
9   @override  
10  void printDocument() {  
11    print('Printing document with Laser Printer');  
12  }  
13  
14  @override  
15  void scanDocument() {  
16    print('Scanning document with Laser Printer');  
17  }  
18 }
```

```
20 class InkjetPrinter implements Printer {  
21   @override  
22   void printDocument() {  
23     print('Printing document with Inkjet Printer');  
24   }  
25  
26   @override  
27   void scanDocument() {  
28     print('Scanning document with Inkjet Printer');  
29   }  
30 }
```

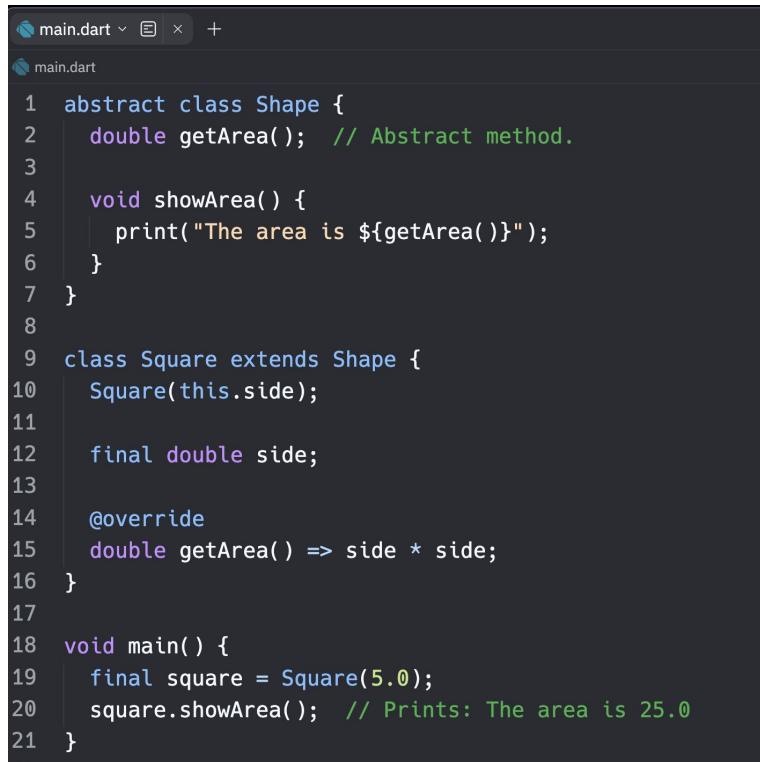
```
_ Console x Shell +  
dart main.dart  
Printing document with Inkjet Printer  
[]
```

```
33 void main() {  
34   var myPrinter = new InkjetPrinter();  
35   myPrinter.printDocument();  
36 }
```

Quiz ຕົວອຍ່າງ 3 : ໃຊ້ InkjetPrinter ຍັງໄວ ?

```
abstract class AbstractCar {  
    void honk(); // Abstract method  
}
```

การใช้งาน Abstract ใน Dart



```
main.dart
```

```
1 abstract class Shape {
2     double getArea(); // Abstract method.
3
4     void showArea() {
5         print("The area is ${getArea()}");
6     }
7 }
8
9 class Square extends Shape {
10    Square(this.side);
11
12    final double side;
13
14    @override
15    double getArea() => side * side;
16 }
17
18 void main() {
19    final square = Square(5.0);
20    square.showArea(); // Prints: The area is 25.0
21 }
```

ຕົວຢ່າງ 1 :ການໃຊ້ງານ Abstract ໃນ Dart

```
main.dart ✓ +  
main.dart  
1 abstract class Shape {  
2     double getArea(); // Abstract method.  
3  
4     void showArea() {  
5         print("The area is ${getArea()}");  
6     }  
7 }  
8  
9 class Square extends Shape {  
10    Square(this.side);  
11  
12    final double side;  
13  
14    @override  
15    double getArea() => side * side;  
16 }  
17  
18 void main() {  
19    final square = Square(5.0);  
20    square.showArea(); // Prints: The area is 25.0  
21 }
```

```
>_ Console ✕  Shell ✕ +  
dart main.dart  
The area is 25.0  
> █
```

ตัวอย่าง 1 : การใช้งาน Abstract ใน Dart

```
main.dart
1 abstract class Animal {
2     Animal(this.name);
3
4     final String name;
5
6     void speak(); // Abstract method.
7 }
8
9 class Dog extends Animal {
10    Dog(String name) : super(name);
11
12    @override
13    void speak() {
14        print('$name says woof!');
15    }
16 }
17
18 void main() {
19    final dog = Dog('Rex');
20    dog.speak(); // Prints: Rex says woof!
21 }
```

ตัวอย่าง 2 : การใช้งาน Abstract ใน Dart

```
main.dart
abstract class Animal {
    Animal(this.name);
    final String name;
    void speak(); // Abstract method.
}
class Dog extends Animal {
    Dog(String name) : super(name);
    @override
    void speak() {
        print('$name says woof!');
    }
}
void main() {
    final dog = Dog('Rex');
    dog.speak(); // Prints: Rex says woof!
}
```

```
>_ Console < x < Shell < +  
▶ dart main.dart  
Rex says woof!  
▶ █
```

ตัวอย่าง 2 : การใช้งาน Abstract ใน Dart

```
main.dart
abstract class Vehicle {
  String get brand; // Abstract getter.
  int get year;    // Abstract getter.

  void honk() {
    print('$brand says honk!');
  }
}

class Car extends Vehicle {
  Car(this.brand, this.year);

  @override
  final String brand;

  @override
  final int year;
}

void main() {
  final car = Car('Toyota', 2020);
  car.honk(); // Prints: Toyota says honk!
}
```

ຕົວຢ່າງ 3 :ການໃຊ້ງານ Abstract ໃນ Dart

```
main.dart
abstract class Vehicle {
  String get brand; // Abstract getter.
  int get year;    // Abstract getter.

  void honk() {
    print('$brand says honk!');
  }
}

class Car extends Vehicle {
  Car(this.brand, this.year);

  @override
  final String brand;

  @override
  final int year;
}

void main() {
  final car = Car('Toyota', 2020);
  car.honk(); // Prints: Toyota says honk!
}
```

```
_ Console × Shell × +
```

```
dart main.dart
Toyota says honk!
```

ตัวอย่าง 3 : การใช้งาน Abstract ใน Dart



klubmaot Flutter



-  Mobile
-  Web
-  Desktop
-  Embedded

**“บน macOS เราสามารถสร้างแอปที่เดียว
ให้ใช้ได้ทั้ง iOS, Android ได้เลย”**

**“แต่สำหรับ Windows และ Linux
จะทำได้เฉพาะผู้ใช้ Android เท่านั้น”**



สิ่งที่เราต้องเตรียมสำหรับ Flutter

- **Flutter SDK**

- Flutter SDK สำหรับจัดการ Project Flutter
- Git สำหรับใช้งาน Flutter SDK

- **Platform Tools**

- Android Studio สำหรับพัฒนาบน Android
- XCode สำหรับใช้พัฒนาบน iOS

- **Virtual Device**

- Android สำหรับ Preview App
- iOS สำหรับ Preview App



เริ่มติดตั้ง Flutter



uu Windows

[Set up an editor >](#)

Windows install

[Get started](#) > [Install](#) > Windows

System requirements

To install and run Flutter, your development environment must meet these minimum requirements:

- **Operating Systems:** Windows 10 or later (64-bit), x86-64 based.
- **Disk Space:** 1.64 GB (does not include disk space for IDE/tools).
- **Tools:** Flutter depends on these tools being available in your environment.
 - [Windows PowerShell 5.0](#) or newer (this is pre-installed with Windows 10)
 - [Git for Windows 2.x](#), with the **Use Git from the Windows Command Prompt** option.

If Git for Windows is already installed, make sure you can run `git` commands from the command prompt or PowerShell.

Get the Flutter SDK

1. Download the following installation bundle to get the latest stable release of the Flutter SDK:

[flutter_windows_3.10.3-stable.zip](#)

For other release channels, and older builds, check out the [SDK archive](#).



uu macOS

https://docs.flutter.dev/get-started/install/macOS

Flutter

Multi-Platform Development Ecosystem Showcase Docs Get started

Racing Forward at I/O 2023 with Flutter and Dart
Read the announcement!

Get started

1. Install

2. Set up an editor

3. Test drive

4. Write your first app

5. Learn more

From another platform?

Dart language overview

Stay up to date

Samples & tutorials

User interface

Navigation & routing

Data & backend

Accessibility & localization

Platform integration

Packages & plugins

Testing & debugging

macOS install

Get started > Install > macOS

System requirements

To install and run Flutter, your development environment must meet these minimum requirements:

- Operating Systems: macOS, version 10.14 (Mojave) or later.
- Disk Space: 2.8 GB (does not include disk space for IDE/tools).
- Tools: Flutter uses git for installation and upgrade. We recommend installing Xcode, which includes git, but you can also [install git separately](#).

Important: If you're installing on an Apple Silicon Mac, you must have the Rosetta translation environment available for some ancillary tools. You can install this manually by running:

```
$ sudo softwareupdate --install-rosetta --agree-to-license
```

Set up an editor

Contents

System requirements

Get the Flutter SDK

Run flutter doctor

Downloading straight from GitHub instead of using an archive

Update your path

Platform setup

iOS setup

Install Xcode

Set up the iOS simulator

Create and run a simple Flutter app

Deploy to iOS devices

Android setup

Install Android Studio

Set up your Android device

Set up the Android emulator

Agree to Android Licenses

macOS setup

Additional macOS requirements

Next step