

# LẬP TRÌNH DI ĐỘNG

GV: TRẦN THANH TUẤN

[tttuan@caothang.edu.vn](mailto:tttuan@caothang.edu.vn)



# Ngôn ngữ lập trình Dart

<https://dart.dev>

<https://dartpad.dev>



# Dart

Dart is a client-optimized language for fast apps on any platform

[Watch video](#)

Supported by Google

Dart is free and open source 



**Optimized  
for UI**

Develop with a programming language specialized around the needs of user interface creation



**Productive  
development**

Make changes iteratively: use hot reload to see the result instantly in your running app



**Fast on all  
platforms**

Compile to ARM & x64 machine code for mobile, desktop, and backend. Or compile to JavaScript for the web



# Ngôn ngữ lập trình Dart

- **Dart** là ngôn ngữ lập trình *mã nguồn mở* do Google phát triển vào năm 2011
- **Dart** là ngôn ngữ lập trình hướng đối tượng



# Ngôn ngữ lập trình Dart

## Các keyword

abstract	else	import	show
as	enum	in	static
assert	export	interface	super
async	extends	is	switch
await	extension	late	sync
break	external	library	this
case	factory	mixin	throw
catch	false	new	true
class	final	null	try
const	finally	on	typedef
continue	for	operator	var
covariant	Function	part	void
default	get	required	while
deferred	hide	rethrow	with
do	if	return	yield
dynamic	implements	set	



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Sử dụng từ khoá **var** để khai báo biến
- Sử dụng từ khoá **final**, **const** để khai báo hằng số

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

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

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

```
final String nickname = 'Bobby';
```

```
const bar = 1000000; // Unit of pressure (dynes/cm2)
```

```
const double atm = 1.01325 * bar; // Standard atmosphere
```

```
int? lineCount; // Uninitialized variables that have a nullable type have  
an initial value of null
```



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Các kiểu dữ liệu:
  - [Numbers](#) (int, double)
  - [Strings](#) (String)
  - [Booleans](#) (bool)
  - [Lists](#) (List, also known as *arrays*)
  - [Sets](#) (Set)
  - [Maps](#) (Map)
  - [Runes](#) (Runes; often replaced by the characters API)
  - [Symbols](#) (Symbol)
  - The value null (Null)



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu [Numbers](#) (int, double):
  - int
    - Integer values no larger than 64 bits, depending on the platform. **On native** platforms, values can be from  $-2^{63}$  to  $2^{63} - 1$ . **On the web**, integer values are represented as JavaScript numbers (64-bit floating-point values with no fractional part) and can be from  $-2^{53}$  to  $2^{53} - 1$ .
  - double
    - 64-bit (double-precision) floating-point numbers, as specified by the IEEE 754 standard.



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu Numbers (int, double):

```
var x = 1; var hex = 0xDEADBEEF;
```

```
var exponent = 8e5;
```

```
var y = 1.1; var exponents = 1.42e5;
```

```
num x = 1; // x can have both int and double values
```

```
x += 2.5;
```

```
double z = 1; // Equivalent to double z = 1.0.
```





# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu [Numbers](#) (int, double):

```
// String -> int
```

```
var one = int.parse('1');
```

```
// String -> double
```

```
var onePointOne = double.parse('1.1');
```

```
// int -> String
```

```
String oneAsString = 1.toString();
```

```
// double -> String
```

```
String piAsString = 3.14159.toStringAsFixed(2);
```



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu Strings (String):
  - A Dart string (String object) holds a sequence of UTF-16 code units.
  - Can use either single or double quotes to create a string.

```
var s1 = 'Single quotes work well for string literals.';
```

```
var s2 = "Double quotes work just as well.";
```

```
var s3 = `It\'s easy to escape the string delimiter.`;
```

```
var s4 = "It's even easier to use the other delimiter.";
```



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu Strings (String):

// Check whether a string contains another string.

```
'Never odd or even'.contains('odd');
```

// Does a string start with another string?

```
'Never odd or even'.startsWith('Never');
```

// Does a string end with another string?

```
'Never odd or even'.endsWith('even');
```

// Find the location of a string inside a string.

```
'Never odd or even'.indexOf('odd') == 6;
```



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu Strings (String):

```
// Convert to uppercase.
```

```
structured web apps'.toUpperCase();
```

```
// Convert to lowercase.
```

```
'STRUCTURED WEB APPS'.toLowerCase();
```



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu [Strings](#) (String):

```
// Grab a substring.
```

```
'Never odd or even'.substring(6, 9) // 'odd'
```

```
// Split a string using a string pattern.
```

```
var parts = 'structured web apps'.split(' ');
```

```
parts.length //3
```

```
parts[0] //'structured'
```



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu [Strings](#) (String):

```
// Trim a string.
```

```
'hello'.trim() //'hello'
```

```
// Check whether a string is empty.
```

```
''.isEmpty;
```

```
// Strings with only white space are not empty.
```

```
' '.isEmpty;
```



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu Strings (String):

```
var greetingTemplate = 'Hello, NAME!';
```

```
var greeting = greetingTemplate.replaceAll(RegExp('NAME'), 'Bob');
```




# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu Strings (String):

```
var sb = StringBuffer();  
sb.write('Use a StringBuffer for ');  
sb.writeAll(['efficient', 'string', 'creation'], ' ');  
sb.write('.');  
var fullString = sb.toString();
```



```
sb  
..write('Use a StringBuffer for ')  
..writeAll(['efficient', 'string', 'creation'], ' ')  
..write('.');
```





# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu [Lists](#) (List, also known as *arrays*):

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

```
var list = [ 'Car', 'Boat', 'Plane' ];
```

```
var constantList = const [1, 2, 3];  
// constantList[1] = 1; // This line will cause an error.
```



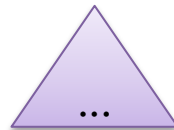
# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu [Lists](#) (List, also known as *arrays*):

```
var list = [1, 2, 3];  
var list2 = [0, ...list];  
// list2.length → 4
```

```
var list;  
var list2 = [0, ...?list];  
// list2.length → 1
```



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu [Lists](#) (List, also known as *arrays*):

```
var nav = [  
  'Home',  
  'Furniture',  
  'Plants',  
  if (promoActive) 'Outlet'  
];
```

```
var listOfInts = [1, 2, 3];  
var listOfStrings = [  
  '#0',  
  for (var i in listOfInts) '#$i'  
];
```

if

for



# Ngôn ngữ lập trình Dart

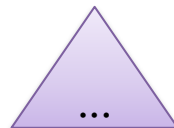
## Biến và Kiểu dữ liệu

- Kiểu dữ liệu [Sets](#) (Set):
  - A set in Dart is an unordered collection of unique items

```
var halogens = {'fluorine', 'chlorine', 'bromine', 'iodine', 'astatine'};
```

```
var names = <String>{};  
// Set<String> names = {}; // This works, too.  
// var names = {}; // Creates a map, not a set.
```

```
var elements = <String>{};  
elements.add('fluorine');  
elements.addAll(halogens);  
elements.length //5
```



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu [Sets](#) (Set):

```
final constantSet = const { 'fluorine', 'chlorine', 'bromine', 'iodine', 'astatine', };  
// constantSet.add('helium'); // This line will cause an error.
```



# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu [Maps](#) (Map):
  - A map is an object that associates keys and values
  - Each key occurs only once, but can use the same value multiple times

```
var gifts = {  
  // Key: Value  
  'first': 'partridge',  
  'second': 'turtledoves',  
  'fifth': 'golden rings'  
};
```

```
var nobleGases = {  
  2: 'helium',  
  10: 'neon',  
  18: 'argon'  
};
```

```
var gifts = Map<String, String>();  
gifts['first'] = 'partridge';  
gifts['second'] = 'turtledoves';  
gifts['fifth'] = 'golden rings';
```

```
var nobleGases = Map<int, String>();  
nobleGases[2] = 'helium';  
nobleGases[10] = 'neon';  
nobleGases[18] = 'argon';
```



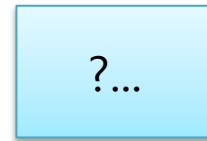
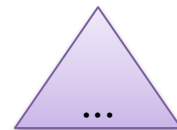
# Ngôn ngữ lập trình Dart

## Biến và Kiểu dữ liệu

- Kiểu dữ liệu Maps (Map):

```
var gifts = {'first': 'partridge';  
gifts['fourth'] = 'calling birds';  
gifts.length //2
```

```
final constantMap = const { 2: 'helium', 10: 'neon', 18: 'argon', };  
// constantMap[2] = 'Helium'; // This line will cause an error.
```



# Ngôn ngữ lập trình Dart

## Các toán tử

- Toán tử số học

Toán tử	Mô tả	Ví dụ
+	Cộng	$2 + 3 \rightarrow 5$
-	Trừ	$2 - 3 \rightarrow -1$
*	Nhân	$2 * 3 \rightarrow 6$
/	Chia	$5 / 2 \rightarrow 2.5$
~/	Chia lấy phần nguyên	$5 ~/ 2 \rightarrow 2$
%	Chia lấy phần dư	$5 \% 2 \rightarrow 1$





# Ngôn ngữ lập trình Dart

## Các toán tử

- Phép so sánh

Toán tử	Mô tả
>	Lớn hơn
>=	Lớn hơn hoặc bằng
<	Bé hơn
<=	Bé hơn hoặc bằng
==	Bằng
!=	Khác



# Ngôn ngữ lập trình Dart

## Các toán tử

- Toán tử logic

Toán tử	Mô tả
!	Phủ định
&&	Và
	Hoặc



# Ngôn ngữ lập trình Dart

## Các toán tử

- Kiểm tra kiểu dữ liệu

Toán tử	Mô tả	Ví dụ
as	Ép kiểu	<code>(employee as Person).firstName = 'Bob';</code>
is	Kiểm tra kiểu dữ liệu	<pre>if (employee is Person) {   // Type check   employee.firstName = 'Bob'; }</pre>
is!	Ngược lại với is	



# Ngôn ngữ lập trình Dart

## Các toán tử

- Phép gán

Toán tử	Mô tả	Ví dụ
=		
+=		
-=		
*=		
/=		
~/=		
%=		
??=	Assign value to b if b is null; otherwise, b stays the same	b ??= value;



# Ngôn ngữ lập trình Dart

## Các toán tử

- Toán tử điều kiện:
  - `condition ? expr1 : expr2`
    - *If condition is true, evaluates expr1 (and returns its value); otherwise, evaluates and returns the value of expr2.*
  - `expr1 ?? expr2`
    - *If expr1 is non-null, returns its value; otherwise, evaluates and returns the value of expr2.*

```
var visibility = isPublic ? 'public' : 'private';
```

```
String playerName(String? name) => name ?? 'Guest';
```



# Ngôn ngữ lập trình Dart

## Cấu trúc điều khiển

- If và else

```
if (isRaining()) {  
    you.bringRainCoat();  
} else if (isSnowing()) {  
    you.wearJacket();  
} else {  
    car.putTopDown();  
}
```



# Ngôn ngữ lập trình Dart

## Cấu trúc điều khiển

- Cấu trúc lặp **for**

```
var message = StringBuffer('Dart is fun');  
for (var i = 0; i < 5; i++) {  
    message.write('!');  
}
```

```
var callbacks = 0;  
for (var i = 0; i < 2; i++) {  
    callbacks.add(() => print(i));  
}  
callbacks.forEach((c) => c());
```

```
for (var candidate in candidates) {  
    candidate.interview();  
}
```



# Ngôn ngữ lập trình Dart

## Cấu trúc điều khiển

- Cấu trúc lặp **while**, **do ... while**

```
while (!isDone()) {  
    doSomething();  
}
```

```
do {  
    printLine();  
} while (!atEndOfPage());
```





# Ngôn ngữ lập trình Dart

## Cấu trúc điều khiển

- Break và continue

```
while (true) {  
    if (shutdownRequested())  
        break;  
    processIncomingRequests();  
}
```

```
for (int i = 0; i < candidates.length; i++) {  
    var candidate = candidates(i);  
    if (candidate.yearsExperience < 5) {  
        continue;  
    }  
    candidate.interview(); }  
}
```



# Ngôn ngữ lập trình Dart

## Cấu trúc điều khiển

- Switch case

```
var command = 'OPEN';  
switch (command) {  
  case 'CLOSED':  
    executeClosed();  
    break;  
  case 'OPEN':  
    executeOpen();  
    break;  
  default:  
    executeUnknown();  
}
```



# Ngôn ngữ lập trình Dart

## Cấu trúc điều khiển

- Switch case

```
var command = 'OPEN';  
switch (command) {  
  case 'CLOSED':  
    executeClosed();  
    ERROR: Missing break  
  case 'OPEN':  
    executeOpen();  
    break;  
  default:  
    executeUnknown();  
}
```



# Ngôn ngữ lập trình Dart

## Cấu trúc điều khiển

- Switch case

```
var command = 'CLOSED';  
switch (command) {  
  case 'CLOSED': // Empty case falls through.  
  case 'NOW_CLOSED':  
    executeNowClosed();  
    break;  
}
```

Runs for both CLOSED and NOW\_CLOSED.



# Ngôn ngữ lập trình Dart

## Cấu trúc điều khiển

- Switch case

```
var command = 'CLOSED';  
switch (command) {  
  case 'CLOSED':  
    executeClosed();  
    continue nowClosed;  
  nowClosed:  
  case 'NOW_CLOSED':  
    executeNowClosed();  
    break;  
}
```

Continues executing at the nowClosed label.

Runs for both CLOSED and NOW\_CLOSED.



# Ngôn ngữ lập trình Dart

## Hàm

```
// Define a function.  
void printInteger(int aNumber) {  
    print('The number is $aNumber.');// Print to console.  
}  
  
// This is where the app starts executing.  
void main() {  
    var number = 42; // Declare and initialize a variable.  
    printInteger(number); // Call a function.  
}
```



# Ngôn ngữ lập trình Dart

## Hàm

```
bool isNoble(int atomicNumber) {  
    return _nobleGases(atomicNumber) != null;  
}
```

```
isNoble(atomicNumber) {  
    return _nobleGases(atomicNumber) != null;  
}
```

```
bool isNoble(int atomicNumber) => _nobleGases(atomicNumber) != null;
```



# Ngôn ngữ lập trình Dart

## Hàm

```
/// Sets the (bold) and (hidden) flags ...
```

```
void enableFlags({bool? bold, bool? hidden}) {...}
```

```
/// Sets the (bold) and (hidden) flags ...
```

```
void enableFlags({bool bold = false, bool hidden = false}) {...}
```

```
// bold will be true; hidden will be false.
```

```
enableFlags(bold: true);
```

```
String say(String from, String msg, [String? device]) {
```

```
    var result = '$from says $msg';
```

```
    if (device != null) {
```

```
        result = '$result with a $device';
```

```
    }
```

```
    return result;
```

```
}
```





# Ngôn ngữ lập trình Dart

## Ghi chú

```
//This is single line comment
```

```
/*  
This  
is  
multi line  
comment  
*/
```

```
/// This  
/// is  
/// documentation  
/// comment
```



# BÀI TẬP



1. Lấy danh sách các số lẻ trong một danh sách các số nguyên.
2. Lấy danh sách các số chẵn trong một danh sách các số nguyên.
3. Đọc tên của tháng N trong năm.
4. Đọc số N có 3 chữ số thành Tiếng việt không dấu.
5. Kiểm tra số nguyên N có phải là số nguyên tố không?
6. Kiểm tra số nguyên N có phải là số hoàn thiện không?
7. Tìm chữ số lớn nhất của số nguyên dương N.
8. Tìm chữ số nhỏ nhất của số nguyên dương N.
9. Tìm số đảo ngược của số nguyên dương N.
10. Hãy đếm số lượng chữ số lớn nhất của số nguyên dương N.
11. Hãy đếm số lượng chữ số nhỏ nhất của số nguyên dương N.
12. Hãy kiểm tra số nguyên dương N có toàn chữ số lẻ hay không?
13. Hãy kiểm tra số nguyên dương N có toàn chữ số chẵn hay không?
14. Hãy kiểm tra số nguyên dương N có phải là số đối xứng hay không?
15. Hãy kiểm tra các chữ số của số nguyên dương N có tăng dần từ trái sang phải hay không?

# BÀI TẬP



16. Hãy kiểm tra các chữ số của số nguyên dương  $N$  có giảm dần từ trái sang phải hay không?
17. Cho 2 số nguyên dương  $a$  và  $b$ . Hãy tìm ước chung lớn nhất của 2 số này.
18. Cho 2 số nguyên dương  $a$  và  $b$ . Hãy tìm bội chung nhỏ nhất của 2 số này.
19. Giải phương trình bậc nhất:  $ax + b = 0$
20. Giải phương trình bậc hai:  $ax^2 + bx + c = 0$
21. Tính tổng của dãy số sau:  $S(N) = 1 + 2 + 3 + \dots + N$
22. Tìm số nguyên dương  $N$  nhỏ nhất sao cho  $1 + 2 + \dots + N > 10000$
23. Kiểm tra năm  $N$  có phải là năm nhuận hay không?
24. Hãy cho biết tháng  $N$  có bao nhiêu ngày?
25. Tìm ngày kế của ngày  $D$  tháng  $M$  năm  $Y$ .
26. Tìm ngày trước của ngày  $D$  tháng  $M$  năm  $Y$ .
27. Tính xem ngày  $D$  tháng  $M$  năm  $Y$  là ngày thứ bao nhiêu trong năm?
28. Kiểm tra các phần tử trong danh sách số nguyên có tăng dần hay không?
29. Kiểm tra các phần tử trong danh sách số nguyên có giảm dần hay không?
30. Hãy cho biết các phần tử trong danh sách số nguyên có bằng nhau không?

# DO – DON'T – PREFER – AVOID - CONSIDER

Style

## Identifiers

DO name types using UpperCamelCase.

DO name extensions using UpperCamelCase.

DO name libraries, packages, directories, and source files using lowercase with underscores.

DO name import prefixes using lowercase with underscores.

DO name other identifiers using lowerCamelCase.

PREFER using lowerCamelCase for constant names.

DO capitalize acronyms and abbreviations longer than two letters like words.

PREFER using `_`, etc. for unused callback parameters.

DON'T use a leading underscore for identifiers that aren't private.

DON'T use prefix letters.

## Ordering

DO place `"dart:"` imports before other imports.

DO place `"package:"` imports before relative imports.

DO specify exports in a separate section after all imports.

DO sort sections alphabetically.

## Formatting

DO format your code using dart format.

CONSIDER changing your code to make it more formatter-friendly.

AVOID lines longer than 80 characters.

DO use curly braces for all flow control statements.



# DO – DON'T – PREFER – AVOID - CONSIDER

Usage

## **Libraries**

DO use strings in part of directives.

DON'T import libraries that are inside the src directory of another package.

DON'T allow an import path to reach into or out of lib.

PREFER relative import paths.

## **Null**

DON'T explicitly initialize variables to null.

DON'T use an explicit default value of null.

PREFER using ?? to convert null to a boolean value.

AVOID late variables if you need to check whether they are initialized.

CONSIDER copying a nullable field to a local variable to enable type promotion.

## **Strings**

DO use adjacent strings to concatenate string literals.

PREFER using interpolation to compose strings and values.

AVOID using curly braces in interpolation when not needed.



# DO – DON'T – PREFER – AVOID - CONSIDER

Usage

## **Collections**

DO use collection literals when possible.

DON'T use .length to see if a collection is empty.

AVOID using Iterable.forEach() with a function literal.

DON'T use List.from() unless you intend to change the type of the result.

DO use whereType() to filter a collection by type.

DON'T use cast() when a nearby operation will do.

AVOID using cast().

## **Functions**

DO use a function declaration to bind a function to a name.

DON'T create a lambda when a tear-off will do.

DO use = to separate a named parameter from its default value.

## **Variables**

DO follow a consistent rule for var and final on local variables.

AVOID storing what you can calculate.

## **Members**

DON'T wrap a field in a getter and setter unnecessarily.

PREFER using a final field to make a read-only property.

CONSIDER using => for simple members.

DON'T use this. except to redirect to a named constructor or to avoid shadowing.

DO initialize fields at their declaration when possible.



# DO – DON'T – PREFER – AVOID - CONSIDER

Usage

## **Constructors**

DO use initializing formals when possible.

DON'T use late when a constructor initializer list will do.

DO use ; instead of {} for empty constructor bodies.

DON'T use new.

DON'T use const redundantly.

## **Error handling**

AVOID catches without on clauses.

DON'T discard errors from catches without on clauses.

DO throw objects that implement Error only for programmatic errors.

DON'T explicitly catch Error or types that implement it.

DO use rethrow to rethrow a caught exception.

## **Asynchrony**

PREFER async/await over using raw futures.

DON'T use async when it has no useful effect.

CONSIDER using higher-order methods to transform a stream.

AVOID using Completer directly.

DO test for Future<T> when disambiguating a FutureOr<T> whose type argument could be Object.



# DO – DON'T – PREFER – AVOID - CONSIDER

Design

## **Names**

DO use terms consistently.

AVOID abbreviations.

PREFER putting the most descriptive noun last.

CONSIDER making the code read like a sentence.

PREFER a noun phrase for a non-boolean property or variable.

PREFER a non-imperative verb phrase for a boolean property or variable.

CONSIDER omitting the verb for a named boolean *parameter*.

PREFER the “positive” name for a boolean property or variable.

PREFER an imperative verb phrase for a function or method whose main purpose is a side effect.

PREFER a noun phrase or non-imperative verb phrase for a function or method if returning a value is its primary purpose.

CONSIDER an imperative verb phrase for a function or method if you want to draw attention to the work it performs.

AVOID starting a method name with get.

PREFER naming a method to `__()` if it copies the object's state to a new object.

PREFER naming a method as `__()` if it returns a different representation backed by the original object.

AVOID describing the parameters in the function's or method's name.

DO follow existing mnemonic conventions when naming type parameters.





# DO – DON'T – PREFER – AVOID - CONSIDER

Design

## **Libraries**

PREFER making declarations private.

CONSIDER declaring multiple classes in the same library.

## **Classes and mixins**

AVOID defining a one-member abstract class when a simple function will do.

AVOID defining a class that contains only static members.

AVOID extending a class that isn't intended to be subclassed.

DO document if your class supports being extended.

AVOID implementing a class that isn't intended to be an interface.

DO document if your class supports being used as an interface.

DO use mixin to define a mixin type.

AVOID mixing in a type that isn't intended to be a mixin.

## **Constructors**

CONSIDER making your constructor const if the class supports it.



# DO – DON'T – PREFER – AVOID - CONSIDER

Design

## Members

PREFER making fields and top-level variables final.

DO use getters for operations that conceptually access properties.

DO use setters for operations that conceptually change properties.

DON'T define a setter without a corresponding getter.

AVOID using runtime type tests to fake overloading.

AVOID public late final fields without initializers.

AVOID returning nullable Future, Stream, and collection types.

AVOID returning this from methods just to enable a fluent interface.

## Parameters

AVOID positional boolean parameters.

AVOID optional positional parameters if the user may want to omit earlier parameters.

AVOID mandatory parameters that accept a special "no argument" value.

DO use inclusive start and exclusive end parameters to accept a range.

## Equality

DO override hashCode if you override ==.

DO make your == operator obey the mathematical rules of equality.

AVOID defining custom equality for mutable classes.

DON'T make the parameter to == nullable.



# DO – DON'T – PREFER – AVOID - CONSIDER

Design

## **Types**

DO type annotate variables without initializers.

DO type annotate fields and top-level variables if the type isn't obvious.

DON'T redundantly type annotate initialized local variables.

DO annotate return types on function declarations.

DO annotate parameter types on function declarations.

DON'T annotate inferred parameter types on function expressions.

DON'T type annotate initializing formals.

DO write type arguments on generic invocations that aren't inferred.

DON'T write type arguments on generic invocations that are inferred.

AVOID writing incomplete generic types.

DO annotate with dynamic instead of letting inference fail.

PREFER signatures in function type annotations.

DON'T specify a return type for a setter.

DON'T use the legacy typedef syntax.

PREFER inline function types over typedefs.

PREFER using function type syntax for parameters.

AVOID using dynamic unless you want to disable static checking.

DO use Future<void> as the return type of asynchronous members that do not produce values.

AVOID using FutureOr<T> as a return type.



