Flutter

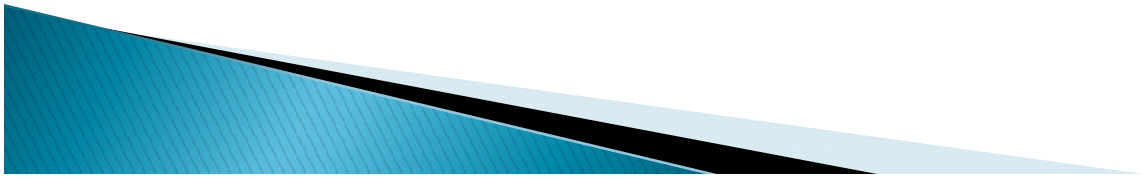
พัฒนา Mobile App ด้วย Flutter (Android และ iOS)

โค้ชเอก

Codingthailand.com

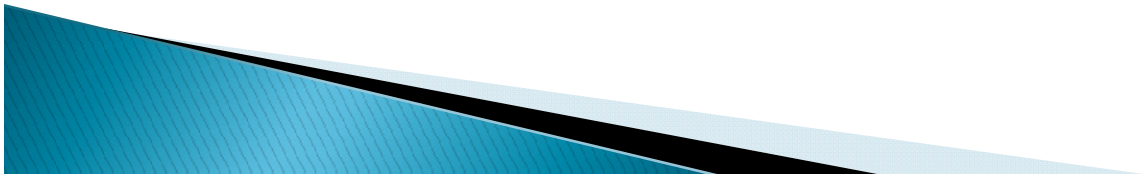
เรียนรู้ภาษา Dart

https://dart.dev/



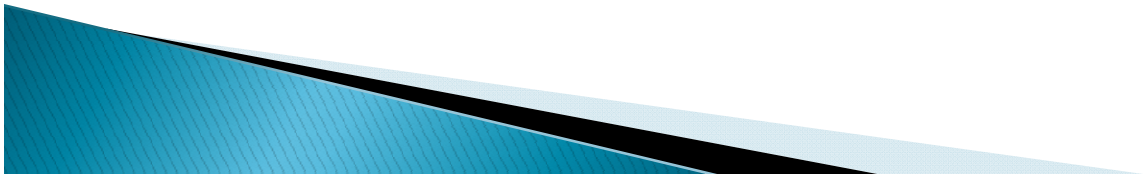
Dart

- Dart is a client-optimized language for fast apps on **any platform.**
- Develop with a programming language specialized around the needs of user interface creation.
- Make changes iteratively: use hot reload to see the result instantly in your running app
- Compile to ARM & x64 machine code for mobile, desktop, and backend. Or compile to JavaScript for the web



Important concepts

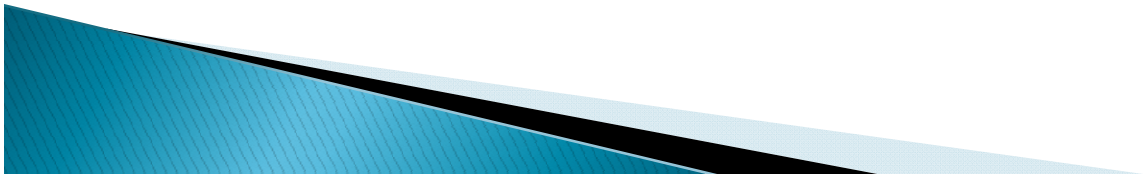
- **Everything you can place in a variable is an *object***, and every object is an instance of a *class*. All objects inherit from the Object class.
- Although Dart is **strongly typed**
- **Dart supports generic types,** like List<int> (a list of integers) or List<dynamic> (a list of objects of any type).
- Uninitialized variables have an **initial value of null.**



Hello World

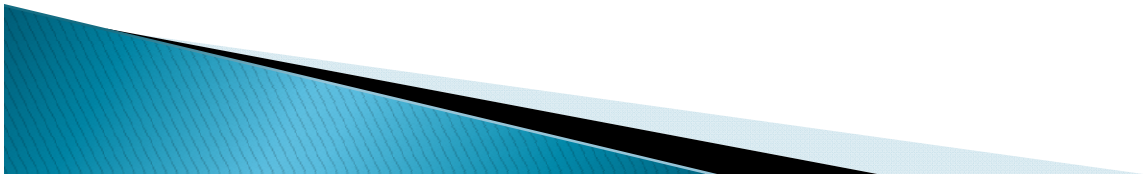
- ทดลองเขียนโค้ดได้ที่ https://dartpad.dev/

```
void main() {
  print('Hello, World!');
}
```

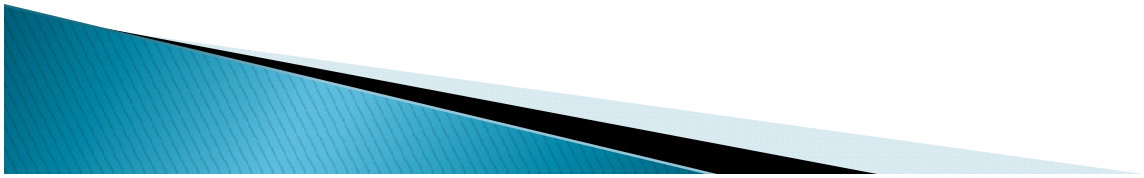


Comments

```
// This is a normal, one-line comment.

/// This is a documentation comment, used to document libraries,
/// classes, and their members. Tools like IDEs and dartdoc treat
/// doc comments specially.

/* Comments like these are also supported. */
```

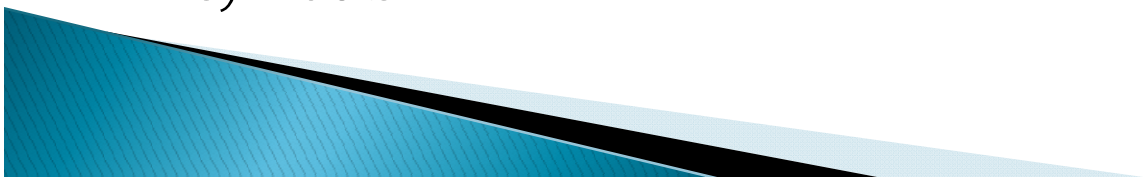


Variables



Built-in types

- numbers
- strings
- booleans
- lists (also known as *arrays*)
- sets
- maps
- runes (for expressing Unicode characters in a string)
- symbols



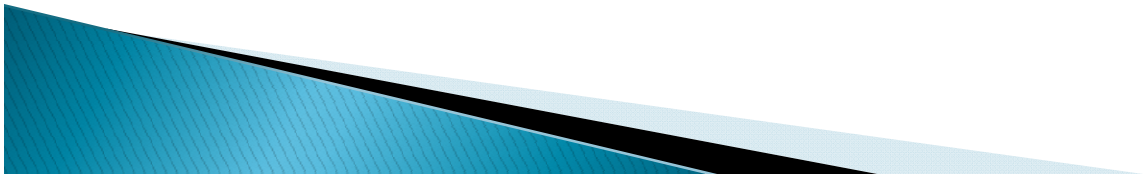
Numbers

- Dart numbers come in two flavors:
- int Integer values no larger than 64 bits, depending on the platform. On the Dart VM, values can be from -2^{63} to $2^{63} - 1$. Dart that's compiled to JavaScript uses JavaScript numbers, allowing values from -2^{53} to $2^{53} - 1$.

```
int salary = 10000;
```

- Double 64-bit (double-precision) floating-point numbers, as specified by the IEEE 754 standard.

```
double salary = 10000.25;
```

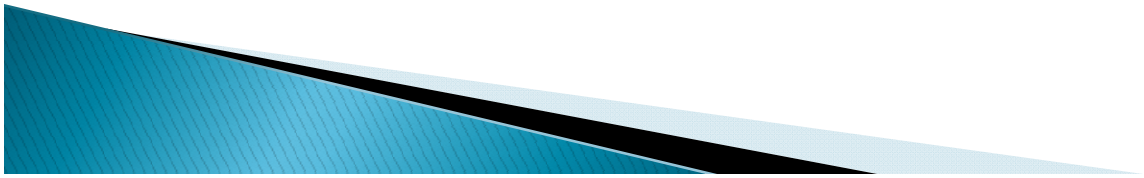


How you turn a string into a number?

```
// String -> int
var one = int.parse('1');
assert(one == 1);

// String -> double
var onePointOne = double.parse('1.1');
assert(onePointOne == 1.1);

// int -> String
String oneAsString = 1.toString();
assert(oneAsString == '1');

// double -> String
String piAsString = 3.14159.toStringAsFixed(2);
assert(piAsString == '3.14');
```

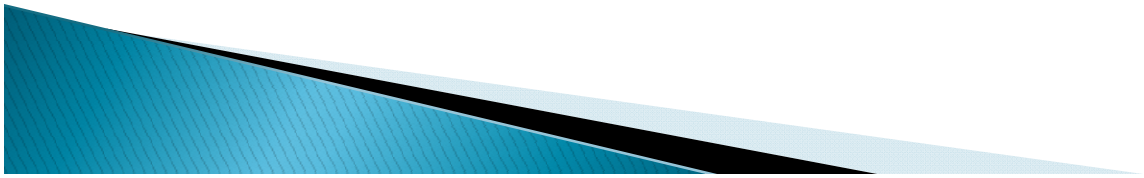
Strings

```
void main() {

 var salary = 1000;
 var name = 'codingthailand';

 var s1 = 'Single $salary work well for \n' + ' a ' + 'string literals.';
 var s4 = "It's even ${name.toUpperCase()} to use the other delimiter.";

 print(s1);
 print(s4);

}
```



Booleans

```
void main() {

var isShow = true;
bool isActive = false;

var message = isShow ? 'Hello true': 'Hello false';
var message2 = isActive == true ? 'Hello true': 'Hello false';

print(message2);

}
```



Lists (คล้าย Array ในภาษาอื่นๆ)

- In Dart, arrays are List objects, so most people just call them *lists*.
- https://api.dart.dev/stable/2.7.0/dart-core/List-class.html

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

print(list[0]);
print(list.length);

var list2 = [0, ...list]; // spread operator (...)
print(list2[1]);

var list3 = [1, 2, 3];
list3.add(499);
print(list3);
list3.removeLast();
list3.removeAt(0);
print(list3);

for (var l in list3) {
  print(l);
}
```



```dart
var list = ['apples', 'bananas', 'oranges'];
list.forEach((item) {
  print('${list.indexOf(item)}: $item');
});
```

Maps

- A map is an object that associates **keys and values.**

```
void main() {

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

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

print(gifts['second']);
print(nobleGases[10]);

}
```

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

print(gifts['first']);
gifts['fourth'] = 'calling birds'; // Add a key-value pair
print(gifts);
print(gifts['fourth']);
```

Variables

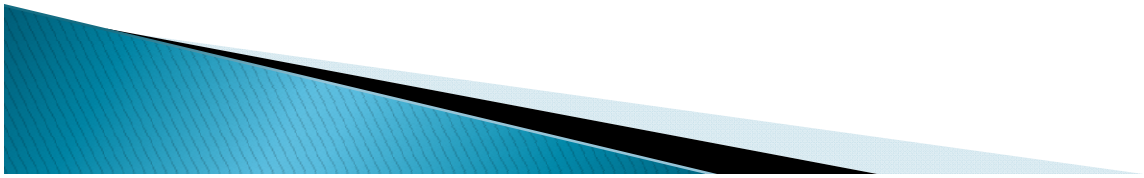
- Even in type-safe Dart code, **most variables don't need explicit types.**

```
var name = 'Voyager I';
var year = 1977;
var antennaDiameter = 3.7;
var flybyObjects = ['Jupiter', 'Saturn', 'Uranus', 'Neptune'];
var image = {
  'tags': ['saturn'],
  'url': '//path/to/saturn.jpg'
};
```

```
print(image['url']);
print(flybyObjects[0]);
print(antennaDiameter);
```

final and const

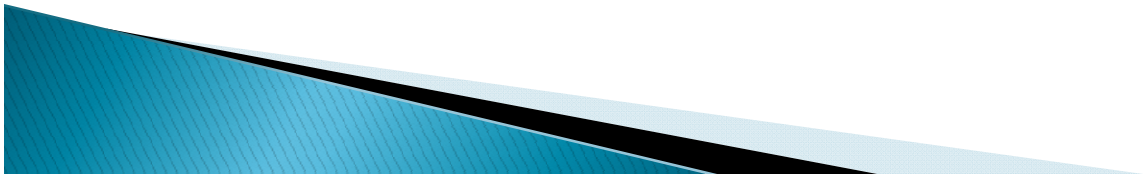
- **If you never intend to change a variable,** use const or final
- Use **const** for variables that you want to be **compile-time constants**.
- A **final** variable can **be set only once.**
- A final top-level or **class variable is initialized the first time it's used.**
- Instance variables **can be final** but not const.
- **Final instance variables must be initialized** before the constructor body starts



final and const

```dart
void main() {
  final name = 'Bob'; // Without a type annotation
  final String nickname = 'Bobby';

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

}
```

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

 bar = 10;
 print(atm);

}
```



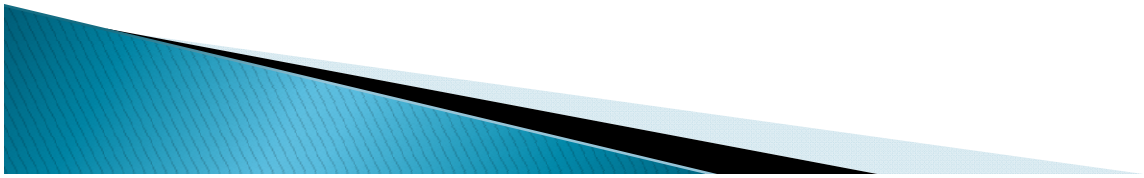
Control flow statements

```
if (year >= 2001) {
  print('21st century');
} else if (year >= 1901) {
  print('20th century');
}

for (var object in flybyObjects) {
  print(object);
}

for (int month = 1; month <= 12; month++) {
  print(month);
}

while (year < 2016) {
  year += 1;
}
```

Functions

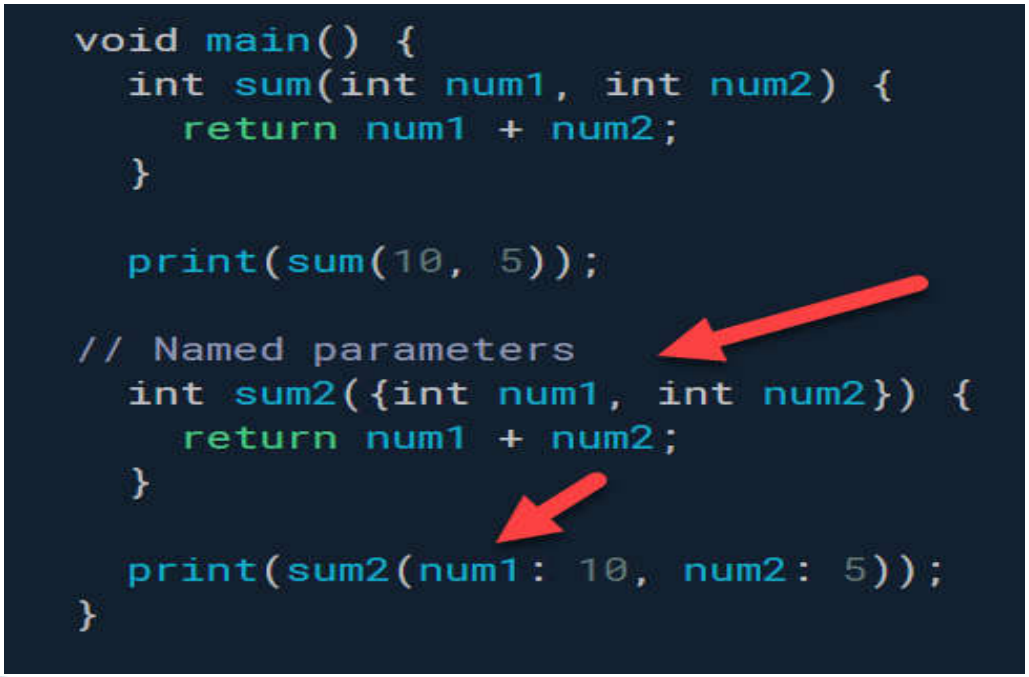
- We recommend specifying the types of each function's arguments and return value:

```
int fibonacci(int n) {
  if (n == 0 || n == 1) return n;
  return fibonacci(n - 1) + fibonacci(n - 2);
}

var result = fibonacci(20);
```



Functions

- Named parameters



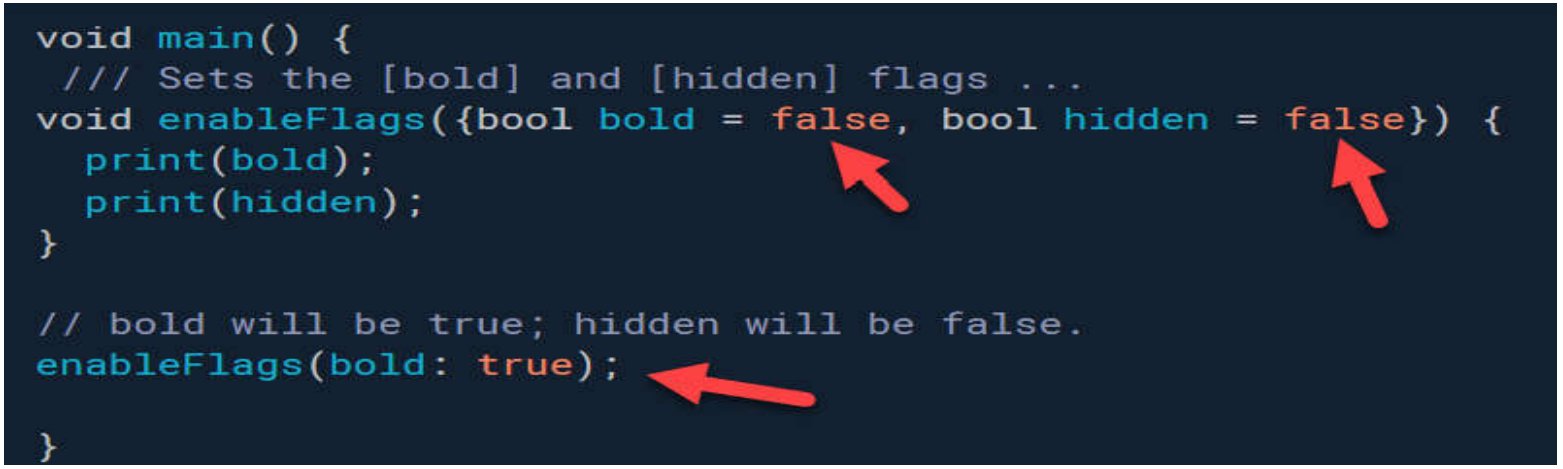
Functions

- Positional parameters: Wrapping a set of function parameters in **[] marks them as optional positional parameters**:

```
void main() {
  String say(String from, String msg, [String msg2]) {
  var result = '$from says $msg';
  if (msg2 != null) {
    result = '$result and says $msg2';
  }
  return result;
}

  print(say('john', 'hello', 'hi'));
  print(say('john', 'hello'));
}
```

Functions

- Default parameter values



```
void main() {
 /// Sets the [bold] and [hidden] flags ...
void enableFlags({bool bold = false, bool hidden = false}) {
  print(bold);
  print(hidden);
}

// bold will be true; hidden will be false.
enableFlags(bold: true);

}
```

Operators

Description	Operator
Multiplicative	* / % ~/
additive	+ -
logical AND และ logical OR	&& และ ||
if null	??
conditional	*expr1* ? *expr2* : *expr3*
assignment	= *= /= += -= &= ^= *etc.*
equality	== !=

ดูเพิ่มเติมได้ที่ https://dart.dev/guides/language/language-tour#operators

Classes

```
class Employee {
  String name;
  double salary;

  // Constructor
  Employee(this.name, this.salary) {
    // Initialization code goes here.
  }

  // read-only non-final property
  double get salaryWithVat => salary + (salary * 0.07);

  // Method.
  void showData() {
    print('Name is: $name');
    if (salary >= 50000) {
      print('Position: Manager');
    } else {
      print('Position: Staff');
    }
  }
}
```

```
void main() {
 var emp = Employee('John', 60000);
     emp.showData();
     var empVat = emp.salaryWithVat;
     print(empVat);
}
```

Named constructors

- Use a named constructor to implement **multiple constructors for a class** or to provide extra clarity:

```
class Point {
  num x, y;

  Point(this.x, this.y);

  // Named constructor
  Point.origin() {
    x = 0;
    y = 0;
  }
}
```

Getters and setters

```
class Rectangle {
  num left, top, width, height;

  Rectangle(this.left, this.top, this.width, this.height);

  // Define two calculated properties: right and bottom.
  num get right => left + width;
  set right(num value) => left = value - width;
  num get bottom => top + height;
  set bottom(num value) => top = value - height;
}

void main() {
  var rect = Rectangle(3, 4, 20, 15);
  assert(rect.left == 3);
  rect.right = 12;
  assert(rect.left == -8);
}
```

Inheritance

```
class Television {
  void turnOn() {
    _illuminateDisplay();
    _activateIrSensor();
  }
  // ...
}

class SmartTelevision extends Television {
  void turnOn() {
    super.turnOn();
    _bootNetworkInterface();
    _initializeMemory();
    _upgradeApps();
  }
  // ...
}
```

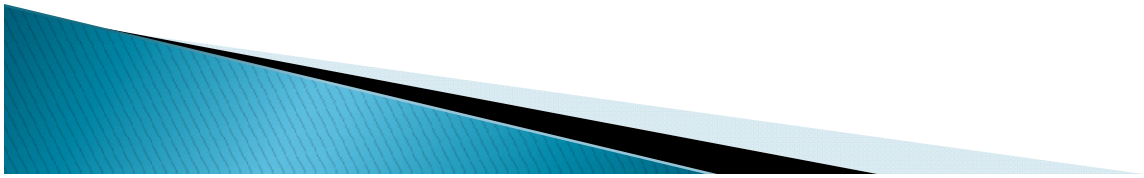
Overriding members

- **Subclasses can override instance methods, getters, and setters**. You can use the @override annotation to indicate that you are intentionally overriding a member:

```
class SmartTelevision extends Television {
  @override
  void turnOn() {...}
  // ...
}
```

Class variables and methods

```
class Queue {
  static var initialCapacity = 16;
  static int sum(int a, int b) {
    return a + b;
  }
}

void main() {
  print(Queue.initialCapacity);
  print(Queue.sum(10, 5));
}
```



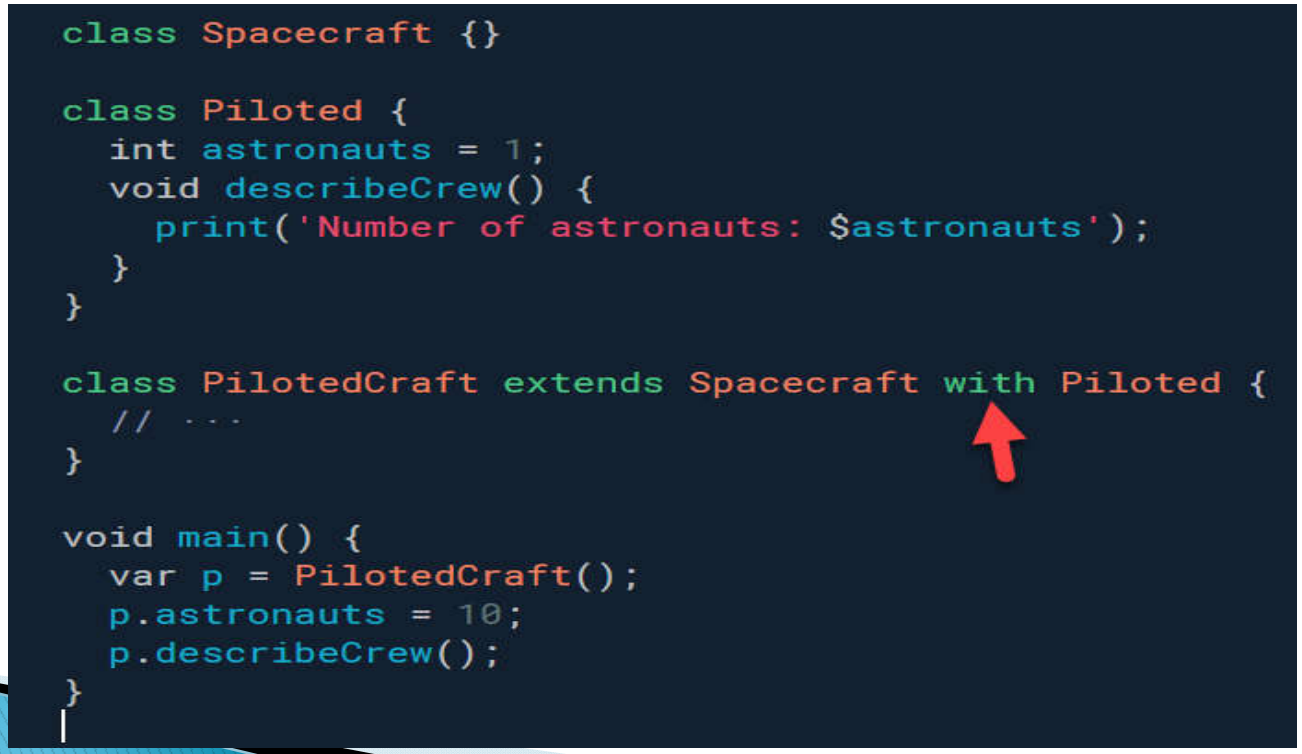
Generics

```dart
var names = List<String>();
names.addAll(['Seth', 'Kathy', 'Lars']);
names.add(42); // Error
```

```dart
T first<T>(List<T> ts) {
  // Do some initial work or error checking, then...
  T tmp = ts[0];
  // Do some additional checking or processing...
  return tmp;
}
```

Mixins

- Mixins are a way of reusing code in multiple class hierarchies.



```
class Spacecraft {}

class Piloted {
  int astronauts = 1;
  void describeCrew() {
    print('Number of astronauts: $astronauts');
  }
}

class PilotedCraft extends Spacecraft with Piloted {
  // ...
}

void main() {
  var p = PilotedCraft();
  p.astronauts = 10;
  p.describeCrew();
}
```

Cascade notation (..)

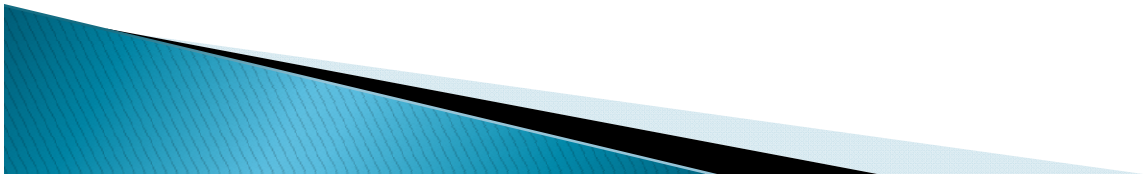
- Cascades (..) allow you to make **a sequence of operations on the same object.**

```
Demo d1 = new Demo();
Demo d2 = new Demo();


// Without Cascade Notation
d1.setA(20);
d1.setB(25);
d1.showVal();
// With Cascade Notation
d2..setA(10)
  ..setB(15)
  ..showVal();
```

Libraries

- Use import to specify how a namespace from one library is used in the scope of another library.

```
// Importing core libraries
import 'dart:math';

// Importing libraries from external packages
import 'package:test/test.dart';

// Importing files
import 'path/to/my_other_file.dart';
```



Libraries

- Specifying a library prefix

```dart
import 'package:lib1/lib1.dart';
import 'package:lib2/lib2.dart' as lib2;

// Uses Element from lib1.
Element element1 = Element();

// Uses Element from lib2.
lib2.Element element2 = lib2.Element();
```

- Importing only part of a library

```dart
// Import only foo.
import 'package:lib1/lib1.dart' show foo;

// Import all names EXCEPT foo.
import 'package:lib2/lib2.dart' hide foo;
```

Async

- Avoid callback hell and make your code much more readable by using **async and await.**

```
Future<void> printWithDelay(String message) {
  return Future.delayed(oneSecond).then((_) {
    print(message);
  });
}
```

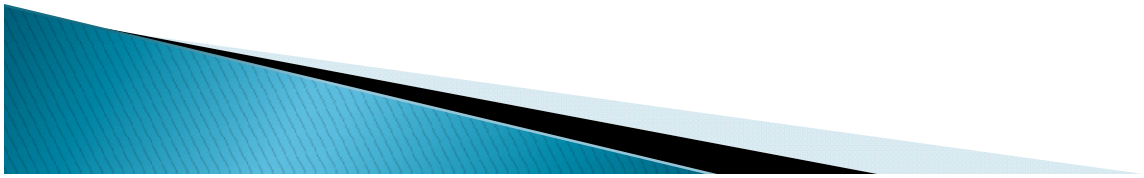
```
const oneSecond = Duration(seconds: 1);
// ···
Future<void> printWithDelay(String message) async {
  await Future.delayed(oneSecond);
  print(message);
}
```

Async

- Future<T> class

- https://api.dart.dev/stable/2.7.0/dart-async/Future-class.html

```
Future<int> future = getFuture();
future.then((value) => handleValue(value))
      .catchError((error) => handleError(error));
```



Exceptions

- To raise an exception, use throw:

```
if (astronauts == 0) {
  throw StateError('No astronauts.');
}
```

To catch an exception, use a try statement with on or catch (or both):

```
try {
  for (var object in flybyObjects) {
    var description = await File('$object.txt').readAsString();
    print(description);
  }
} on IOException catch (e) {
  print('Could not describe object: $e');
} finally {
  flybyObjects.clear();
}
```

The end