


# Dart 2: tipos de datos

Tipos de datos incluidos

 Marcos Alejandro · Aug 15, 2018 · 3 min read

👤 📄 📌 📧 ⋮

- numbers
- strings
- booleans
- lists (también conocidas como arrays)
- maps
- runes (para expresar caracteres Unicode en un string)
- symbols

## NUMBERS

int y double son subtipos de num.

```
// Enteros sin punto decimal, no mayores de 64bits.  
int x = 1;  
int hex = 0xDEADBEEF;  
  
// Números decimales, de 64bits.  
double y = 1.1;  
double exponents = 1.42e5;
```

## Convertir una cadena a un número, y viceversa.

```
var one = int.parse('1'); // de string a int  
  
var onePointOne = double.parse('1.1'); // de string a double  
  
String oneAsString = 1.toString(); // de int a string  
  
String piAsString = 3.14159.toStringAsFixed(2); // double a string
```

## STRINGS

```
var s1 = 'Las comillas simples funcionan bien para los literales';  
var s2 = "Las comillas dobles funcionan igual de bien";  
var s3 = 'El fácil escapar del delimitador de cadena con \n';
```

Puede poner el valor de una expresión dentro de una cadena usando ***{expresión}***. Si la expresión es un identificador se pueden omitir los **{}**.

Puede concatenar cadenas utilizando literales de cadena adyacentes o el operador **+**.

Otra forma de crear una cadena multilíneas es utilizando comillas tripe con comillas simples o dobles:

```
var s1 = '''  
You can create  
multi-line strings like this one.  
''';  
  
var s2 = """This is also a  
multi-line string.""";
```

**Raw String:** para crear una cadena “cruda” usamos el prefijo **r**.

```
var s = r'In a raw string, not even \n gets special treatment.';
```

## BOOLEANS

Solo dos objetos tiene el tipo **bool**: los literales **true** y **false**. Por este motivo para chequear si existe un valor usamos:

```
// Chequear una cadena vacía.  
var fullName = '';  
assert(fullName.isEmpty);  
  
// Chequear cero.  
var hitPoints = 0;  
assert(hitPoints <= 0);  
  
// Chequear null.  
var unicorn;  
assert(unicorn == null);  
  
// Chequear NaN.  
var iMeantToDoThis = 0 / 0;  
assert(iMeantToDoThis.isNaN);
```

## LISTAS

En Dart las *colecciones* o *arrays* son objetos **List**. El índice del primer elemento es cero. Puede obtener la longitud de una lista y referirse a los elementos tal como lo haría en Javascript.

```
var list = [1, 2, 3]; // lista simple que infiere el tipo List<int>  
  
assert(list.length == 3);  
assert(list[1] == 2);  
  
list[1] = 1;  
assert(list[1] == 1);  
  
// Crea una lista constante en tiempo de compilación  
var constantList = const [1, 2, 3];
```

## MAPAS

Un map es un objeto que asocia claves y valores. Tanto las claves como los valores pueden ser cualquier tipo de objeto. Cada key es única.

```
// Se infiere el tipo Map<String,String>  
var gifts = {  
  'first': 'partridge',  
  'second': 'turtledoves',  
  'fifth': 'golden rings'  
};  
  
// Se infiere el tipo Map<int,String>  
var nobleGases = {  
  2: 'helium',  
  10: 'neon',  
  18: 'argon',  
};
```

Puede crear los mismos objetos utilizando un constructor de maps.

```
var gifts = Map();  
gifts['first'] = 'partridge';  
gifts['second'] = 'turtledoves';  
gifts['fifth'] = 'golden rings';  
  
var nobleGases = Map();  
nobleGases[2] = 'helium';  
nobleGases[10] = 'neon';  
nobleGases[18] = 'argon';
```

Se puede añadir un nuevo par key-value, y recupere un valor tal y como lo haríamos en Javascript.

```
gifts['fourth'] = 'calling birds'; // Agrega un par key-value  
gifts['first'] // recupera el valor para la clave "first"
```

Si intentamos recuperar una clave que no existe en el **map**, devuelve **null**.

```
// Crear un map que sea constante en tiempo de compilación.  
final constantMap = const {  
  2: 'helium',  
  10: 'neon',  
  18: 'argon',  
};
```

## RUNES

Unicode define un valor numérico único para cada letra, dígito y símbolo utilizado en todos los sistemas de escritura del mundo. Debido a que un String en Dart es una secuencia de unidades de código de UTF16, la expresión de valores de 32bits dentro de una cadena requiere una sintaxis especial.

La forma habitual de expresar un punto de código Unicode es **\uXXXX**, donde XXXX es un valor hexadecimal de 4 dígitos. Por ejemplo, podemos usar **\u2665** para representar un corazón (♥). Para especificar más o menos de 4 dígitos hexadecimales, coloque el valor entre corchetes. Por ejemplo, el emoji (👹) is **\u{1f600}**.

## SYMBOLS

Un objeto symbol representa un operador o identificador declarado en un programa Dart. Puede que nunca necesite utilizar símbolos, pero son de un valor incalculable para las API que se refieren a identificadores por nombre, ya que la minificación cambia los nombres de identificador pero no los símbolos del identificador.

Para obtener el símbolo de un identificador, utilice un símbolo literal, que es solo **#** seguido por el identificador:

```
#radix  
#bar
```

Dart Flutter Mobile App Development

👤 62 🗨

👤 📄 📌 📧 ⋮

## More from Marcos Alejandro

Follow



Aug 15, 2018

## Dart 2: variables

Las variables almacenan referencias. La variable llamada **name** contiene una referencia a un objeto *String* con un valor de **"Bob"**.

```
var name = 'Bob'; // Se infiere que la variable es de tipo String  
  
String name = 'Bob'; // Se especifica el tipo de la variable  
  
dynamic name = 'Bob'...
```

Read more · 2 min read

👤 15 🗨 1

📄 📌 📧 ⋮

Aug 15, 2018

## Dart 2: Introducción

**¿Por qué Dart?**  
*Los desarrolladores de Google utilizan Dart para crear aplicaciones de alta calidad y de misión crítica para iOS, Android y Web. Con características dirigidas al desarrollo del lado del cliente, Dart es ideal para aplicaciones móviles y web.*

### Características

- **Productivo:** La sintaxis de Dart es clara y concisa...

Read more · 2 min read

👤 73 🗨

📄 📌 📧 ⋮

Aug 14, 2018

## Flutter: tests de widgets — parte 3

Muchos de los widgets que construimos no sólo muestran información, sino que también responden a la interacción del usuario. Esto incluye botones en los que los usuarios pueden puntear, arrastrar elementos por la pantalla o introducir texto en un TextField.

Para probar estas interacciones, necesitamos una forma de simularlas en...

Read more · 3 min read

👤 3 🗨

📄 📌 📧 ⋮

Aug 14, 2018

## Flutter: tests de widgets — parte 2

Para localizar widgets en un entorno de prueba, necesitamos usar clases **Finder**. Aunque es posible escribir nuestras propias clases de **Finder**, generalmente es más conveniente localizar widgets usando las herramientas proporcionadas por el paquete **flutter\_test**.

En esta guía, veremos la constante **find** proporcionada por el paquete **flutter\_test** y demostraremos cómo...

Read more · 2 min read

👤 3 🗨

📄 📌 📧 ⋮

Aug 14, 2018

## Flutter: tests de widgets — parte 1

Para probar las clases de Widgets, necesitaremos algunas herramientas proporcionadas por el paquete **flutter\_test**, que se entrega con el SDK de Flutter.

El paquete **flutter\_test** proporciona las siguientes herramientas para probar Widgets:

- El **WidgetTester**, que nos permite construir e interactuar con Widgets en un entorno de prueba.
- La función **testWidgets**...

Read more · 4 min read

👤 1 🗨

📄 📌 📧 ⋮

