

Mapas y Clases

Control de Flujo e Iteración,
Funciones y métodos



Yosafat Coronel
GDSC ESCOM IPN
GitHub: YosafatM



Developer Student Clubs
ESCOM - IPN

Itinerario

- Mapas
- Funciones útiles Higher-Order
 - Lectura de las hints
 - asMap
 - any, every
 - map
 - where, indexWhere
- Clases

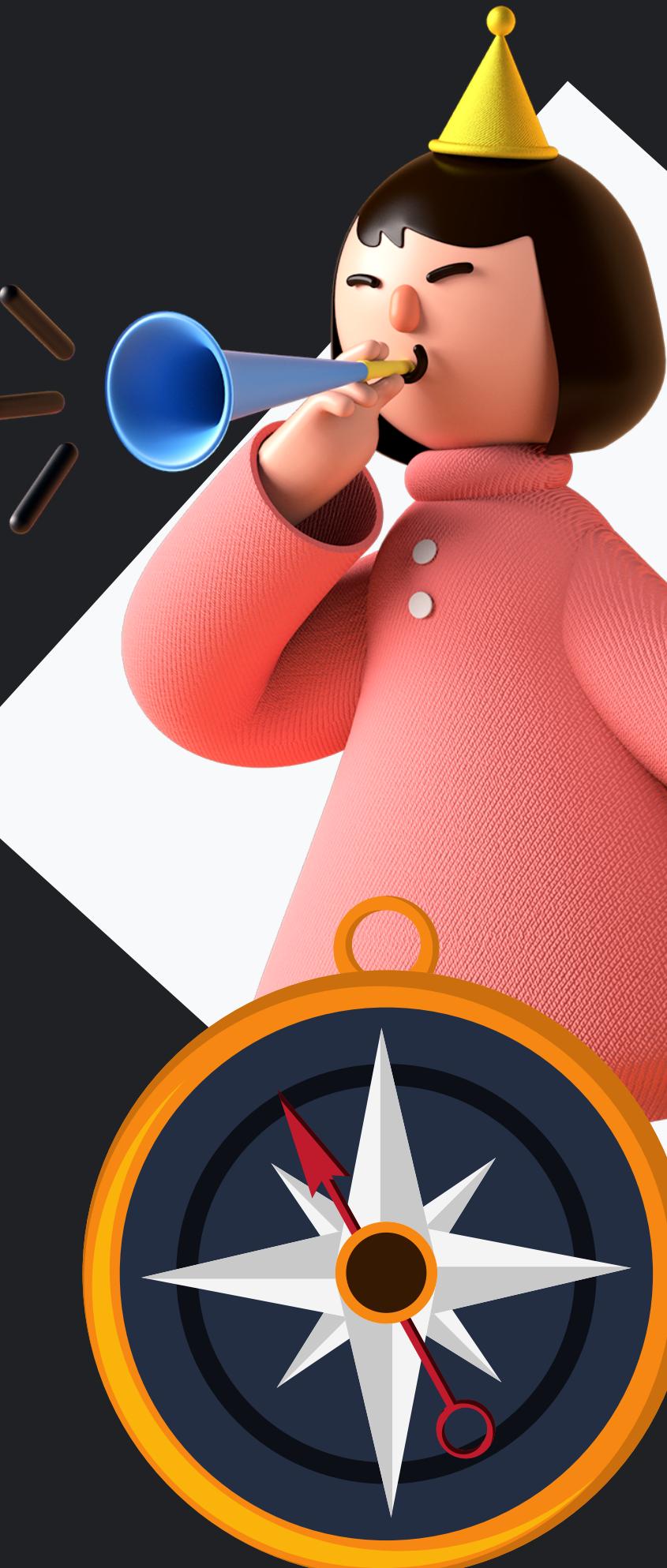


Higher-Order Calculator

<https://replit.com/@YosafatCoronel/8-HOC>

Mapas

```
<String, dynamic> {  
  'key': value,  
}
```



Mapas

La clase Map permite almacenar parejas de llave-valor. Ambos son genéricos (**K** y **V**). El ejemplo de arriba es una forma general, y el de abajo es

Map<K, V> nombre;

Map<int, String> people;

Mapas

Para inicializarlo, usamos llaves, podemos poner el operador de diamante si se está inferiendo el tipo del Mapa. Recordemos que el default es dynamic:

```
final mapa = <String, int>{  
    'Uno': 1,  
};
```

Las parejas se ponen como K: V separados por comas

El punto y coma es importante

Mapas

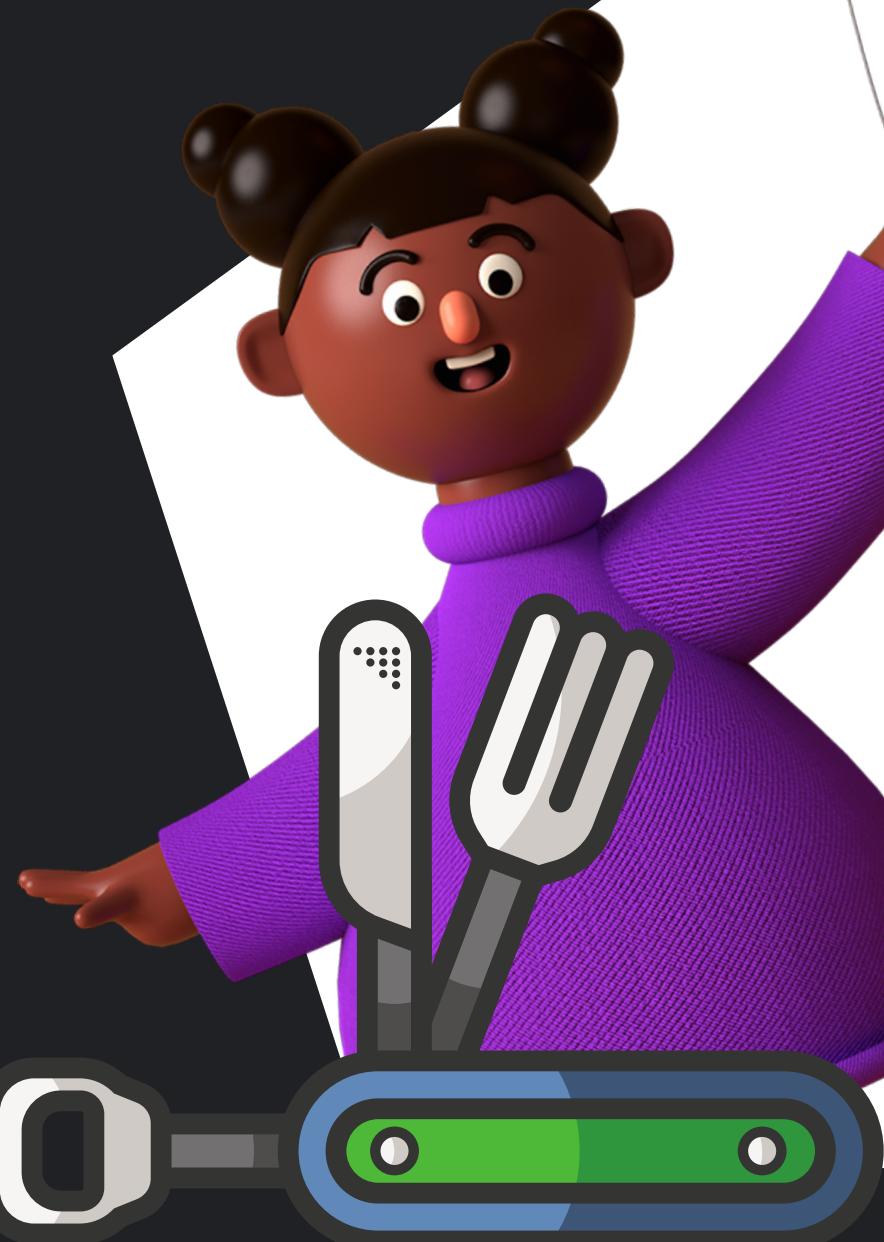
Los mapas son Sucriptibles e Iterables, por lo que podemos usar los corchetes para obtener un valor con el índice de las llaves:

```
<id>[<K>]; //Retorna <V>  
mapa['Uno'];
```

*Retorna 1 con el
mapa anterior*

Utilidades

asMap(), any(),
every(), map(),
indexWhere()



Hints

Al poner el autocompletado, las funciones y métodos muestran una documentación pequeña de lo que regresará y lo que pide como parámetros:

```
numeros.as|  
asMap() → Map<int, int>
```

<id>([params]) -> <return>

En este caso, regresa un Mapa<int, int>

asMap

Con una lista podemos obtener un Map<int, T> (donde T es el tipo de dato que contienen los elementos de la lista), con asMap:

El índice de los elementos es la llave del mapa, el elemento es el valor

```
var mapa = lista.asMap();
```

any

Itera los elementos del Map o List, si alguno de los elementos da true en la función parámetro, entonces da true.

```
[1, 2, 3].any  
any(bool Function(int) test) → bool
```

```
[1, 2, 3].any((e) => e <= 1);
```

Al evaluar da true

every

Itera los elementos del Map o List, si todos los elementos en el iterable dan true con la función parámetro, da true.

```
[1, 2, 3].every((e) => e < 5);  
every(bool Function(int) test) → bool
```

[1, 2, 3].every((e) => e<=1);

Al evaluar da false

map

Itera los elementos del Map o List, la función parámetro regresa un tipo genérico, al final se regresa un iterable (puede ser List o Map, u otros*).

```
[1, 2, 3].map((e) => e+1);  
map(T Function(int) toElement) → Iterable<T>
```

[1, 2, 3].map((e) => e+1);

Regresa [2, 3, 4]

index

Where

Devuelve el índice del elemento que de true en la función parámetro, se puede indicar de forma opcional el comienzo de la búsqueda.

```
[1, 2].indexWhere((e) => e==2);
```

```
indexWhere(bool Function(int) test, [int start = 0]) → int
```

```
[1, 2].indexWhere((e) => e==2);
```

Regresa 1

where

Devuelve un iterable con los elementos que den true con la función parámetro. El iterable será del tipo que iteramos (List o Map):

```
[-1, 2, -3].where((e) => e<0);
```

```
where(bool Function(int) test) → Iterable<int>
```

```
[-1, 2, -3].index((e) => e<0);
```

Regresa una lista con [-1, -3]

Clases

```
class <id> {  
    [contenido]  
}
```

Este tema es para varias clases



Clases

Hemos mencionado que hay definiciones llamadas **clases**. Aquí, definimos lo que puede (y a veces lo que no) hacer un tipo de objeto en específico.

```
class Dogtor{  
    void ladra() => print('Guau!');  
    void atiende() => print('Paracetamol');  
}
```

Aquí, definimos que puede usar `ladra` y `atiende`

Clases

De hecho, ya las hemos usado, por ejemplo, los suscriptibles **pueden** usar los corchetes para obtener un elemento. Los números ciertos operadores y los bool otros:

```
var doc = Dogtor();
doc.la|  
Ladra() → void
```

Dart va a autocompletarlo y tratarlo como definición

Clases

Una clase puede (o no) tener propiedades (variables dentro de una clase) y puede (o no) tener métodos (que son funciones dentro de una clase):

```
class Dogtor{  
    final laConfiable = 'Paracetamol';  
  
    void ladrar() => print('Guau!');  
    void recetar() => print(laConfiable);  
}
```

En este caso, laConfiable es una propiedad

Clases

A partir de ahora, la clase que definimos puede ser usado como tipo de dato, podemos hacer listas y mapas con esa definición:

```
void contratar(Dogtor doc) {  
    if (doc.laConfiable == 'Paracetamol') {  
        print('Contratado');  
    }  
}
```

Podemos recibir a Dogtor como parámetro

Clases

A partir de ahora, la clase que definimos puede ser usado como tipo de dato, podemos hacer listas y mapas con esa definición:

```
void contratar(Dogtor doc) {  
    if (doc.laConfiable == 'Paracetamol') {  
        print('Contratado');  
    }  
}
```

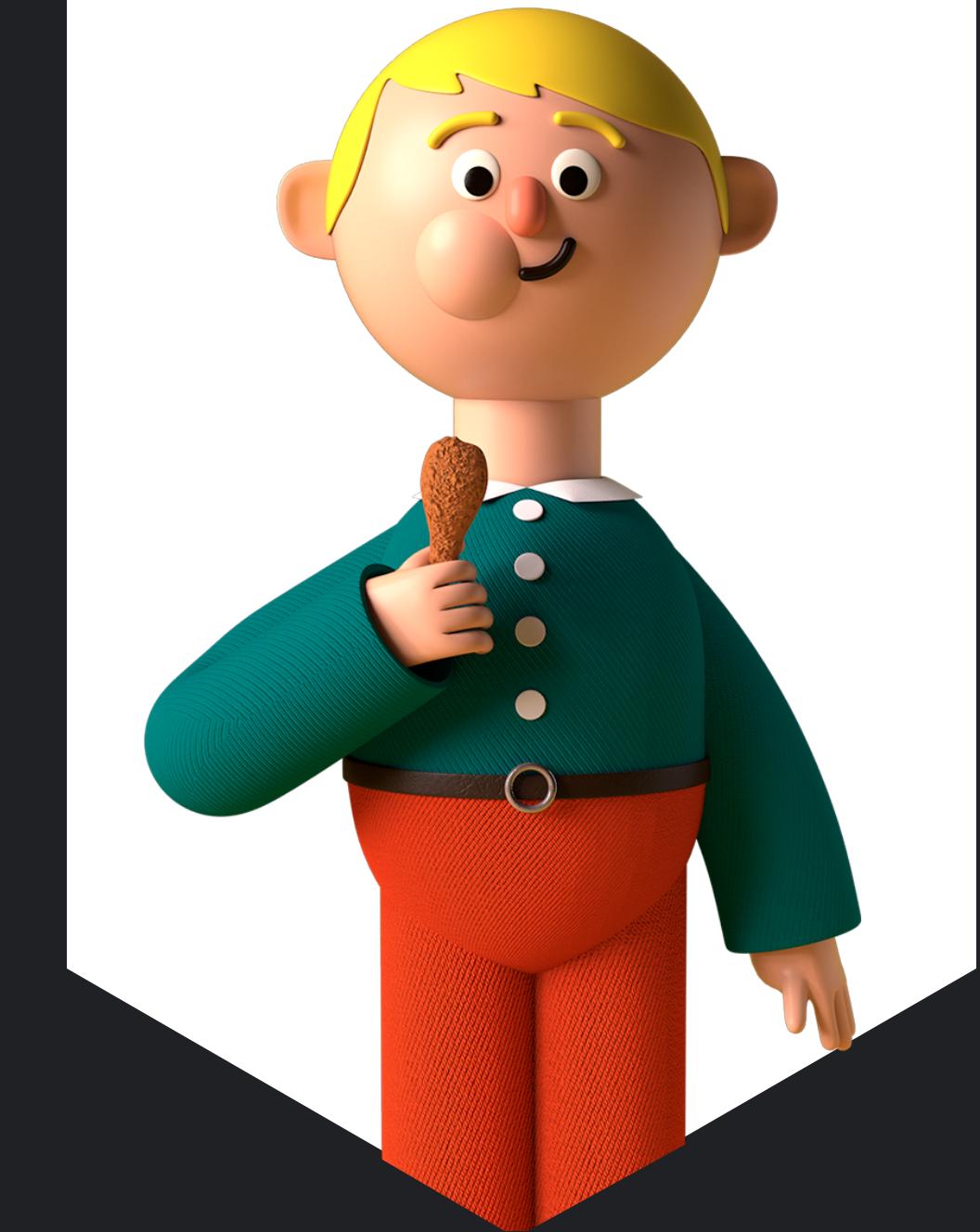
Podemos recibir a Dogtor como parámetro

Clases

¿De qué nos puede servir hacer nuestras propias clases?, recordemos que todo viene de una clase, lo que quiere decir:

Que en Flutter, todo también está en clases.
Por ejemplo, los botones.

¡Muchísimas gracias!



Developer Student Clubs
ESCOM - IPN