

Switch y for:

List y <>

Introducción a Dart y Control
de flujo e iteración



Yosafat Coronel
GDSC ESCOM IPN
GitHub: YosafatM



Developer Student Clubs
ESCOM - IPN

Itinerario



- Aclaración sobre == y dynamic
- Switch: case y break
- For y continue
- List
 - [] NO es null
 - Suscriptibles
 - Operador de diamante
 - Iteración de suscriptibles

Aclaración sobre ==

dynamic en expresiones

A la mitad de la clase menciono 'No como otro lenguaje', sin embargo sí es como otro lenguaje y hasta di ejemplos (JavaScript es al que me refiero), pues sí se pueden comparar entre distintos tipos:

replit.com/@YosafatCoronel/4-dynamic

Switch, case, break

El castillo de ifs

Cuando usamos varios if y preguntamos por la misma variable y los valores que puede tener, entonces el **switch viene bien**:

```
● ● ●  
  
if (line == 'z') {  
    print('Hasta la vista, baby');  
} else if (line == 'a') {  
    print('Perro dice: Guau guau');  
} else if (line == 'b') {  
    print('Gato dice: Miau');  
} else {  
    print('Eso no es gracioso');  
}
```

```
● ● ●  
  
switch (line) {  
    case 'z':  
        print('Hasta la vista, baby');  
        break;  
  
    case 'a':  
        print('Perro dice: Guau guau');  
        break;  
  
    case 'b':  
        print('Gato dice: Miau');  
        break;  
  
    default:  
        print('Eso no es gracioso');  
        // El break aquí es innecesario  
}
```

Break: (en switch)

Aquí break también hace que nos salgamos de la estructura, si varios casos hacen lo mismo, entonces se pondrían así:

```
● ● ●

switch (line) {
    case 'z':
    case 'a':
    case 'b':
        print('Aquí llega casi todo');
        break;

    default:
        print('Eso no es gracioso');
        // El break aquí es innecesario
}
```

Break: en estructuras

Break nos saca de la estructura de control más cercana en la que esté contenida, si usamos un switch en el ejercicio que usamos, no funcionaría:

replit.com/@YosafatCoronel/4-while-y-switch

Breaking good

Si queremos que el break salga de una estructura en específico, entonces a la estructura le podemos poner un identificador:

```
[<id>:] <ciclo/switch> {  
    break [<id>];  
}
```

Los : son necesarios para ponerle identificador

for y continue

Ciclo for

Cuando queremos una cantidad precisa de veces a ejecutar un código, entonces, el for es muy útil.



```
for (int i = 0; i < 5; i++) {  
    // Código a ejecutar  
}
```

Ciclo for

*Se ejecuta la
primera vez que se
llega al for*

*Se ejecuta cada vez
que termina el
código dentro del for*

for (<stm1> <expr> <stm2>)

*Se evalúa cada vez
que termina el for,
y la primera vez
que llega*

Continue: ciclos y switch

Si en alguna de las veces que entra al ciclo no queremos que se ejecute, pero que siga con el código, para eso nos sirve continue:

```
[<id>:] <ciclo/switch> {  
    continue [<id>];  
}
```

Al igual que break, si no usa identificador, toma el ciclo más cercano

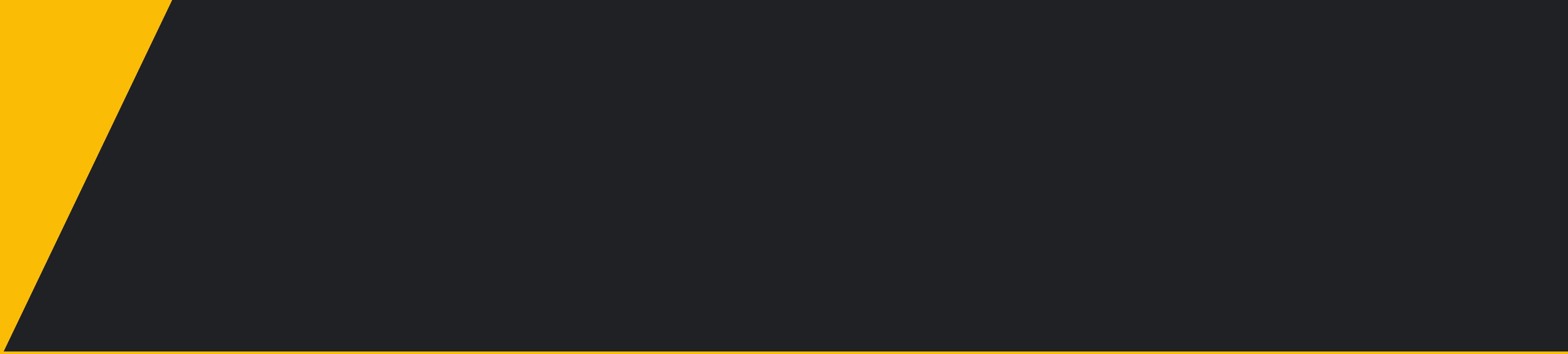
Continue: ¿switch?

Un identificador que usa por dentro switch, es el case, sin embargo no podemos usarlo en el break, pero sí en el continue:

```
case <val1>:
```

```
continue case <val2>;
```

Con esto, al final el caso val1, entonces ejecutaría el caso val2



A new challenger...

...approaches: List

Las listas en Dart son un tipo de dato que pueden contener valores de un mismo tipo (eso incluye dynamic), para decirle qué tipo de dato va a contener se usa el operador de diamante, se pueden inicializar con corchetes y valores por defecto:

*Tipo de
dato*

Genérico

*Recomendable si
no se puso el tipo*

*Los valores
son opcionales*

List<tipo> p = <tipo>[vals];

List: [] no es null

Hay una gran diferencia entre [] (una lista vacía) y null, en el primero ya está inicializado, y es necesario si se usa null-safety:

```
final array = <int>[1, 2, 3];
```

```
final lista = <int>[];
```

```
print(lista == null); //false
```

Tipos subscriptibles (1)

Un tipo es subscriptible/suscriptable cuando se trata de una serie de valores del mismo tipo donde cada uno tiene una posición. Para acceder a cada valor se usan los corchetes con un valor índice:

subscriptible[val];

*val no necesariamente
es un entero*

'Abc'[0];

lista[0];

Tipos subscriptibles (2)

Como se trata de una serie de valores, entonces debe de existir una cuenta de cuántos elementos contiene el tipo, para eso se usa la propiedad (<- esto lo veremos en el tema de clases) `length`:

`subscriptible.length;`

*Siempre regresa
un entero*

Operador de Diamante

El operador de diamante `<>`, nos sirve para indicar un genérico (<- tema que veremos en Genéricos), es muy útil para no escribir código duplicado, List es Iterable, y según la documentación es:

List<E>

E en este caso representa Element, es decir, se compone de elementos

Iterable<T>

T en este caso representa Type, es decir, itera con un mismo tipo

Iteración: subscriptable

Los for (aunque claro que también se puede usar el while y do while) resultan útiles al querer ejecutar un mismo código un número definido de veces (length):

```
• • •  
List<String> names = [  
    'Yosafitness',  
    'Yosafat',  
    'Yosabusiness'  
];  
  
for (int i = 0; i < names.length; i++) {  
    print('Mucho gusto, ${names[i]}');  
}
```

Reto

Momento de codificación

1: Usando un **for/while/do-while** itera sobre el texto de abajo y cuenta las vocales con **if/switch**, impríme las cuentas al final

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Extra: usa **if**, **continue** y el método **toUpperCase**, para que no cuente las vocales mayúsculas

```
print('a'.toUpperCase()); // Imprime A
```

Dudas y preguntas

¡Muchas gracias!

