

# Funciones y

# return

## Funciones y métodos

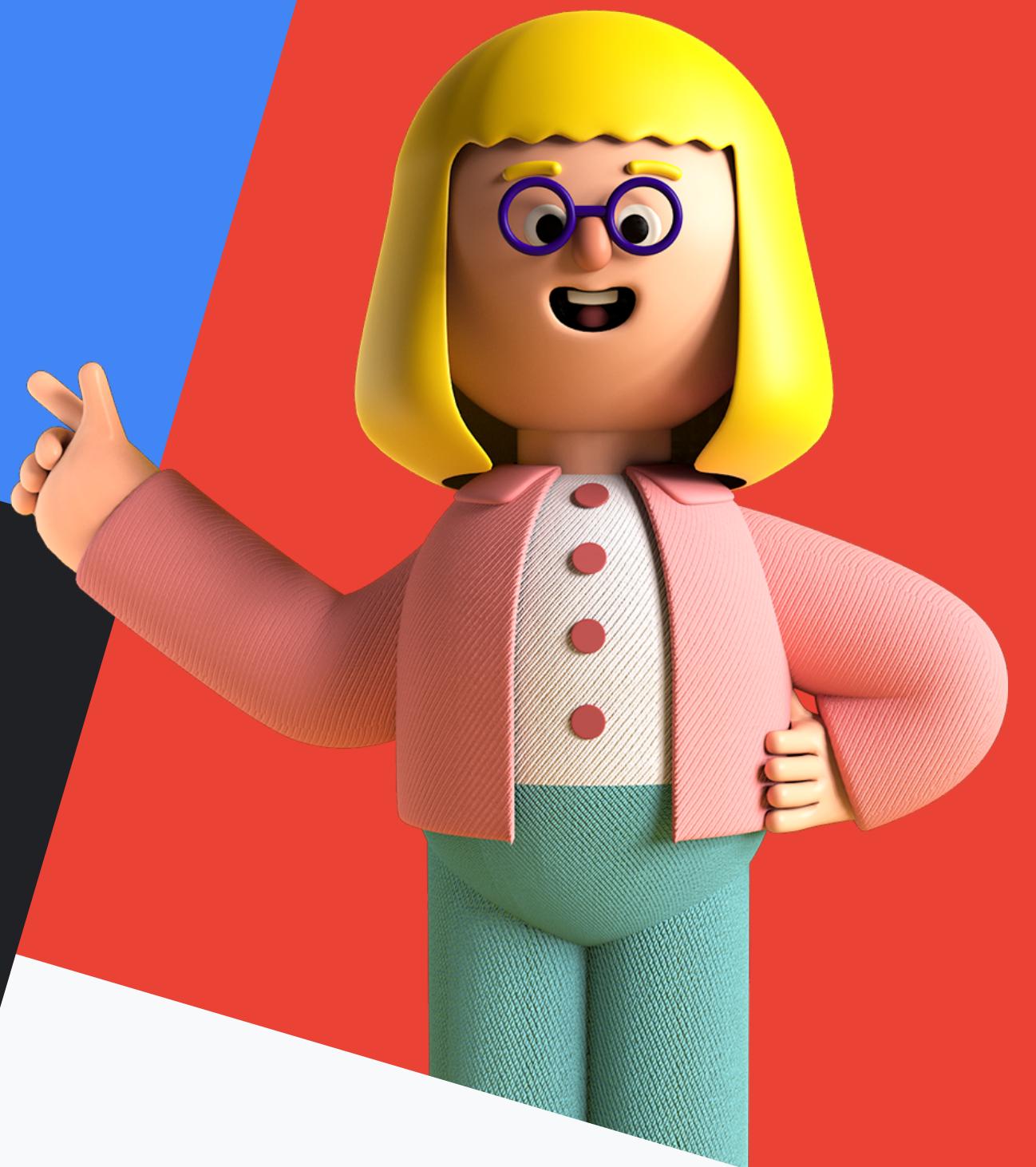


Yosafat Coronel  
GDSC ESCOM IPN  
GitHub: YosafatM



Developer Student Clubs  
ESCOM - IPN

- Estructura de las funciones
- Parámetros de función
  - Parámetros posicionales
  - Parámetros nombrados
  - Parámetros posicionales optionales
- Retorno de una función
- Reescritura de la calculadora



# Itinerario

# Funciones



# Funciones

Como vimos en la clase pasada, la forma de las funciones es:

```
<tipo> <id>([params]){
    [contenido]
}
```

# Funciones

Hasta ahora solo hemos visto con el tipo void, ¿pero qué pasa cuando queremos otro tipo y qué significa cada pedazo de lo que vimos:

**<tipo>**: Lo que se espera  
de la función

# Funciones

Hasta ahora solo hemos visto con el tipo void, ¿pero qué pasa cuando queremos otro tipo y qué significa cada pedazo de lo que vimos:

**<id/nombre>**: Con esto se identifica en el marco

# Funciones

Hasta ahora solo hemos visto con el tipo void, ¿pero qué pasa cuando queremos otro tipo y qué significa cada pedazo de lo que vimos:

**[params]: Lo que la función espera para servir**

# Funciones

Hasta ahora solo hemos visto con el tipo void, ¿pero qué pasa cuando queremos otro tipo y qué significa cada pedazo de lo que vimos:

**[contenido]: El cuerpo de la función (lleva return\*)**

# Parámetros de función



# Parámetros de función

Ayer utilizamos el marco de **alto nivel (top-level)** para realizar una calculadora, sin embargo, **es peligroso que cualquier función top level** pueda acceder a nuestras variables.

Para ello conviene tener una sugerencia: **usar el marco más pequeño posible** para lo que queremos hacer. Por ejemplo, las funciones que sumaban lo hace con variables en su marco.

# Parámetros de función

Los parámetros de una función constan de una lista de elementos separados por comas. Cada elemento es de la siguiente forma:

*Tipo de dato*

*esperado*

**<tipo> <var>**

*Nombre en  
el marco de  
la función*

# Parámetros posicionales

A este tipo de elementos se les llaman parámetros posicionales requeridos (da error si no se pasan todos), porque en el mismo orden que los definimos es como los esperamos:

*El mismo orden se espera (a, b)*

```
void suma(int a, int b) {  
    suma(3, 4); a tendrá el valor 3, b el valor 4
```

# ¿Argumentos?

En la definición de la función, la lista de variables esperadas son llamadas parámetros, pero en la llamada de una función, la lista de literales son llamados argumentos.

*Estos son parámetros*

```
void suma(int? a, int? b) {}
```

```
suma(3, 4);
```

*Estos son argumentos*

# Parámetros nombrados

Los parámetros nombrados son opcionales (aunque se pueden hacer obligatorios) y pueden tener valores por defecto, están entre llaves y separados por comas:

*Tipo de dato*

*esperado*

**<tipo> <var> [= <default>]**

*Nombre en*

*el marco*

*Opcional: Valor*

*por defecto*

# Parámetros nombrados

El orden de los parámetros nombrados no se requiere en los argumentos, pero, se requiere especificar qué valor es cuál. Pueden o no estar todos los parámetros:

```
void sum({a=1, int? b}){}  
sum(b: 5);
```

*El orden no importa, a tiene 1 por defecto  
a tendrá el valor 3, b el valor 5*

# Parámetros nombrados

Los parámetros nombrados suelen usarse mucho en Flutter porque queda clara qué argumento es qué parámetro. Sin embargo, son opcionales, para hacerlos obligatorios se usa required:

*El parámetro a ya no puede tener valor por defecto si es requerido*

**{required int a, int b = 1}**

*El parámetro b DEBE tener un valor pues no puede tener null*

Una función puede tener  
parámetros nombrado o  
posicionales, pero NO  
ambos

# Posicionales optionales

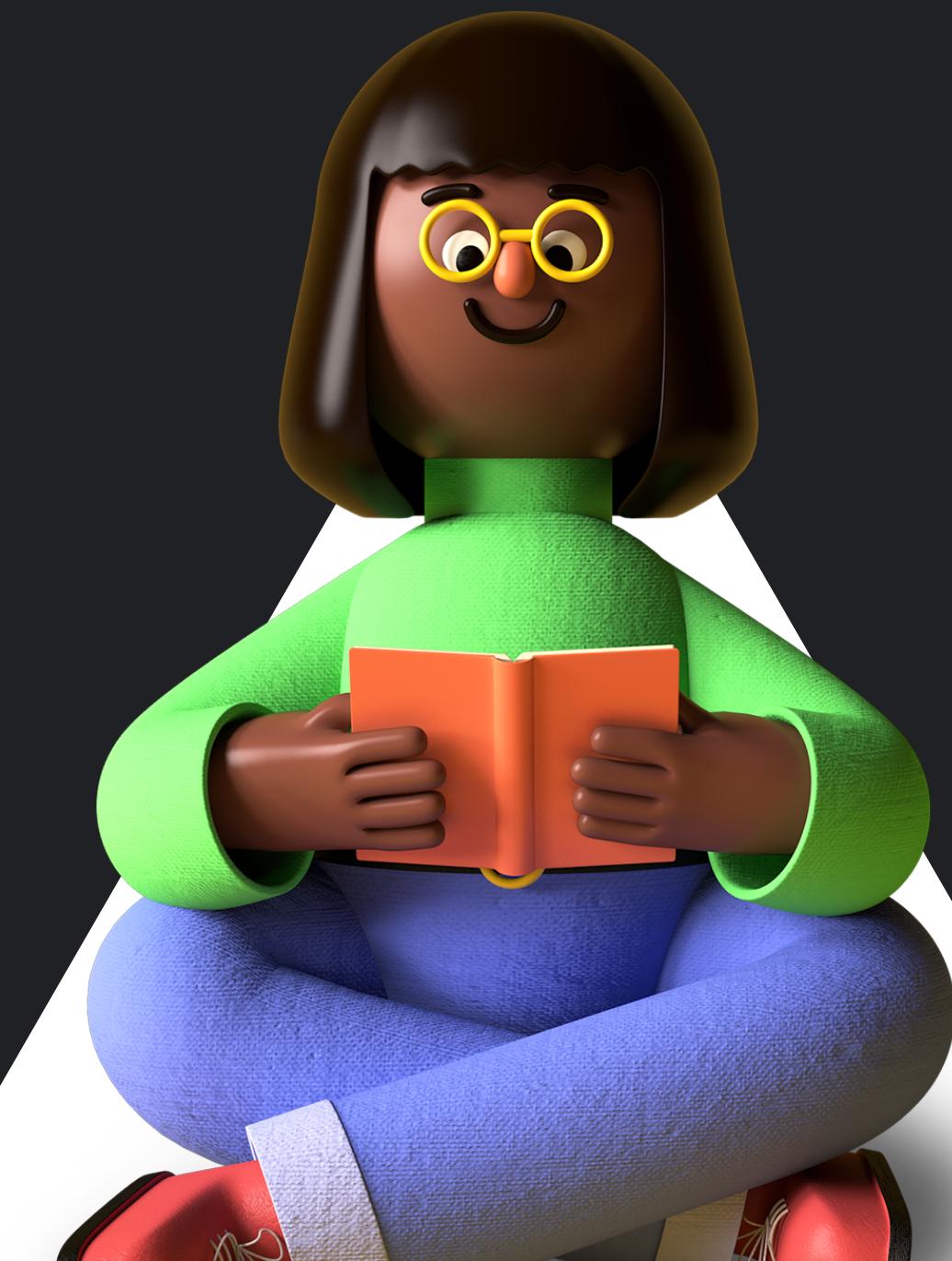
Si estamos usando parámetros posicionales, la forma de hacerlos optionales es la siguiente, solamente ponemos ese parámetro entre corchetes y ahora podemos ponerle un valor por defecto:

*Esto en amarillo es opcional*

[<tipo> <var> = <valor>]

*Estos corchetes DEBEN ir escritos en la definición para que sea opcional*

# Retorno de función



# Retorno de función

Las funciones SIEMPRE utilizan la palabra reservada return para regresar los esperado de una función:

```
return null;  
return;
```

*Cuando se espera void,  
nosotros no ponemos return,  
pero se hace de forma  
implícita. Estos dos return  
son equivalentes, se  
recomienda el segundo.*

# Keyword: return

Return indica que es un punto final de la función, cuando llega a un return se sale de la función y prosigue con el flujo, de hecho, ya lo hemos usado antes stdin.readLineSync() hace justamente eso:

*Entonces esto indica que tome la literal  
o el valor de una variable y lo regrese*

**return <expr del tipo>;**

# Keyword: return

```
int suma(int a, int b) {  
    return a + b;  
}
```

*La expresión del return DEBE ser del tipo que  
indicamos en la función (int en este caso)*

# Keyword: return

Podemos tener múltiples return en una sola función y con diferentes expresiones en ellos, esto es útil en varias situaciones. Pero al menos DEBE tener uno del tipo que definimos en la función.

```
if (a < 0) return -a;  
return a;
```

*Cuerpo de una función que regresa el valor absoluto de un número*

# Calculadora

Con todo esto ahora podemos reescribir la calculadora para seguir la recomendación de usar el marco más pequeño que necesitemos:

[replit.com/@YosafatCoronel/7-Calculadora](https://replit.com/@YosafatCoronel/7-Calculadora)

# Dudas y preguntas



# ¡Muchas gracias!

