

devfest

Reusable & Customizable Functions using Composition

 Google Developer Groups
Bandung



Property of
Erico Darmawan Handoyo





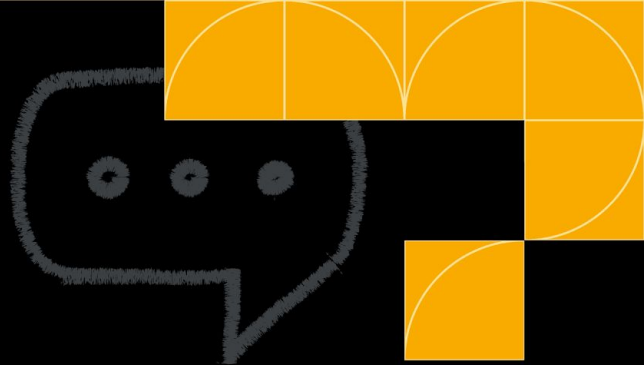
Erico Darmawan Handoyo

- Parahyangan Catholic University
- Bandung Institute of Technology
- Lecturer @ Maranatha Christian University
- The Most Awesome Spectacular Legendary Programming Book Author
- Flutter tutorial creator @ YouTube
- Flutter Trainer
- Traveling ✈️✈️✈️✈️✈️
- Eating 🍗🍗🍗🍗🍗
- Watching movies 🎬🎬🎬
- Anything costs money 💵💵💵💵💵
- Coding 💻➡️🌐💎



```
Text('Simple Statement or URL',  
style: TextStyle(  
  color: Colors.yellow[200],  
),  
),  
),  
s.star,  
r: Colors.yellow[500],  
Text('23'),
```

devfest



**The more you code, the
greater the chance of
making mistakes.**



Property of
Erico Darmawan Handoyo



The more you code, the greater the chance of making mistakes.

What to do:

1. Your codes must be **reusable**.
2. Your codes must be **customizable**.



Reusable & customizable functions
using **Composition**.

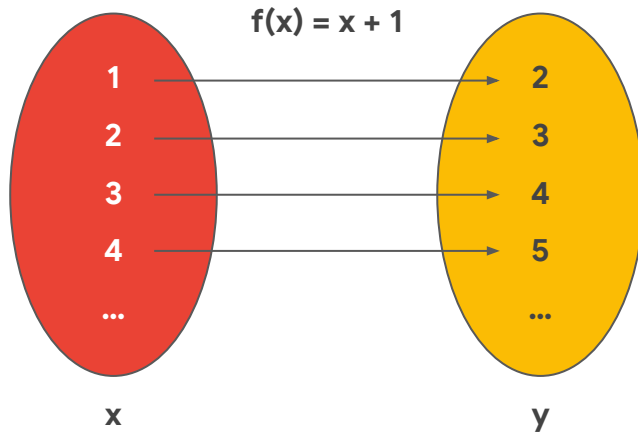
The benefits:

1. **Shorter** development **time**.
2. **Reducing** the possibility of making **mistakes**.
3. **Efficient** use of **resources**.



Functions in Mathematics

Function: an **expression**, **rule**, or **law** that defines a **relationship** between **domain** and **codomain**.



```
int f(int x) {  
    return x + 1;  
}
```

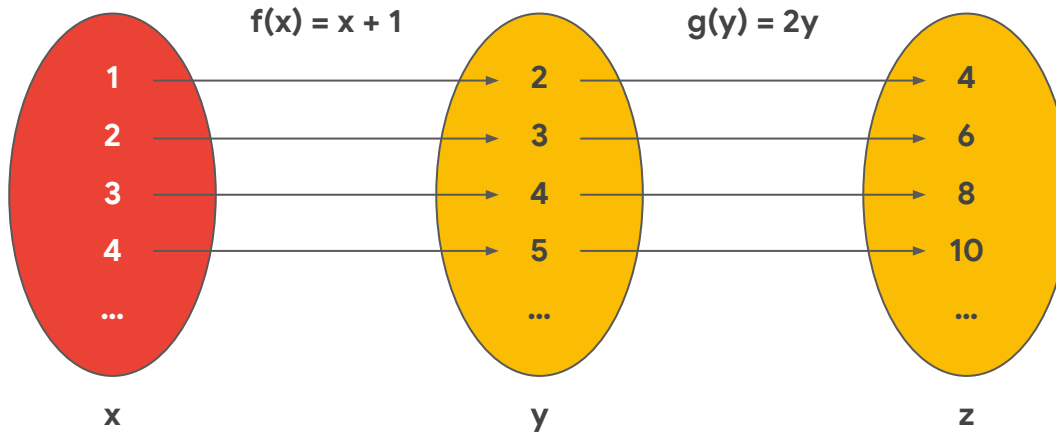


```
int f(int x) => x + 1;
```



Function Composition in Mathematics

The combination of two or more function to form a new function



$$y = f(x) = x + 1$$

$$z = g(y) = 2y$$

$$z = (g \circ f)(x) = g(f(x))$$



How to Make Function Composition

1. Make **reusable simple** functions which has **one thing** to do.
2. **Combine** those simple functions to make a **more complex** function as required, using following rules:
 - a. Put the **first function to execute** at the **end** of the list and always add **the next function** at the **start** of the list.
 - b. The **return type** of a function must be the **same** as the **parameter type** of the function to be **executed next**.
 - c. Create a function that **runs** the list of **functions backwards**.



Function Composition Example



Example 01: Calculating Total Price

```
void main(List<String> args) {  
    double price = 125000;  
  
    double totalPrice = calculatePrice(price, [applyTax(11), applyDiscount()]);  
    print(totalPrice);  
}  
  
double Function(double price) applyDiscount([double discount = 20]) =>  
    (price) => price - price * discount / 100;  
  
double Function(double price) applyTax([double tax = 10]) =>  
    (price) => price + price * tax / 100;  
  
double calculatePrice(double price, List<Function> modifiers) {  
    for (var modifier in modifiers.reversed) {  
        price = modifier(price);  
    }  
  
    return price;  
}
```

See also:

- [Positional & Named Parameter](#)
- [Arrow Syntax, Function sebagai first-class Object, Anonymous Function](#)
- [Var & Dynamic Data Type](#)
- [List & List Mapping](#)



Example 02: Naming Convention (snake_case → PascalCase)

```
List<String> splitWordsWithUnderscore(String string) {  
    return string.split('_');  
}  
  
List<String> capitalizeWords(List<String> words) {  
    return words  
        .map((word) => word[0].toUpperCase() + word.substring(1))  
        .toList();  
}  
  
String joinWords(List<String> words) {  
    return words.join();  
}  
  
String Function(String) stringModifier(List<Function> modifiers) => (string) {  
    dynamic result = string;  
    for (var modifier in modifiers.reversed) {  
        result = modifier(result);  
    }  
  
    return result;  
};
```

See also:

- [Strings](#)



Example 02: Naming Convention (snake_case → PascalCase)

```
void main(List<String> args) {  
    String snakeCase = 'this_is_snake_case';  
  
    String Function(String) snakeToPascal = stringModifier([  
        joinWords,  
        capitalizeWords,  
        splitWordsWithUnderscore,  
    ]);  
  
    String pascalCase = snakeToPascal(snakeCase);  
  
    print(pascalCase);  
}
```

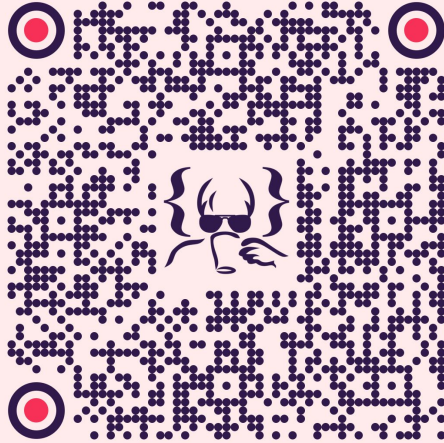


Compose vs Pipe

Compose and pipe serve the **same purpose**: chaining together several functions and executing them sequentially. The difference is, **compose** executes the functions from **right to left** / **bottom to top**, while **pipe** executes functions from **left to right** / **top to bottom**.



Thank You



youtube.com/@ericodarmawan



s.id/komunitas-flutter



s.id/flixid



ericodarmawan.com



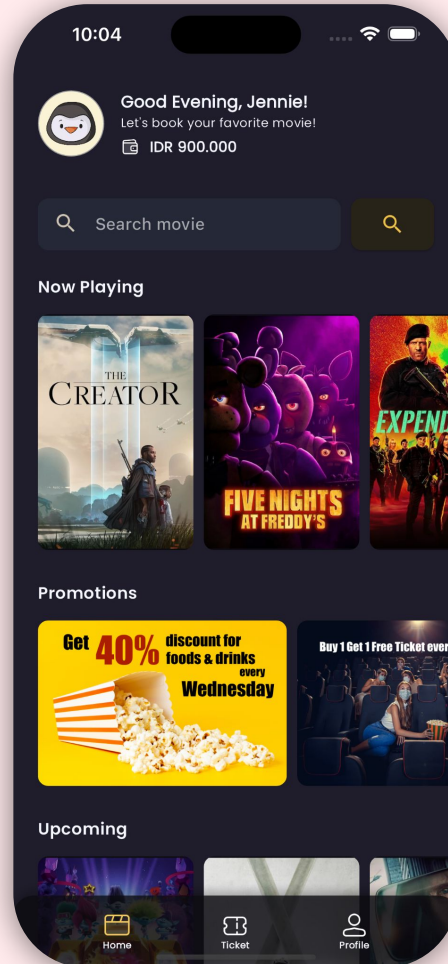
s.id/fb-ericodh



s.id/ig-ericodh



s.id/in-ericodh



- Separation of Concerns
- Clean Architecture
- Riverpod State Management
- Firebase Auth, Storage, & Firestore

~~IDR 399.000~~
IDR 199.000
02 - 06 Desember 2023



s.id/flixid

割 KODE PROMO 割

DEVFEST-BANDUNG2023

Property of
Erico Darmawan Handoyo

