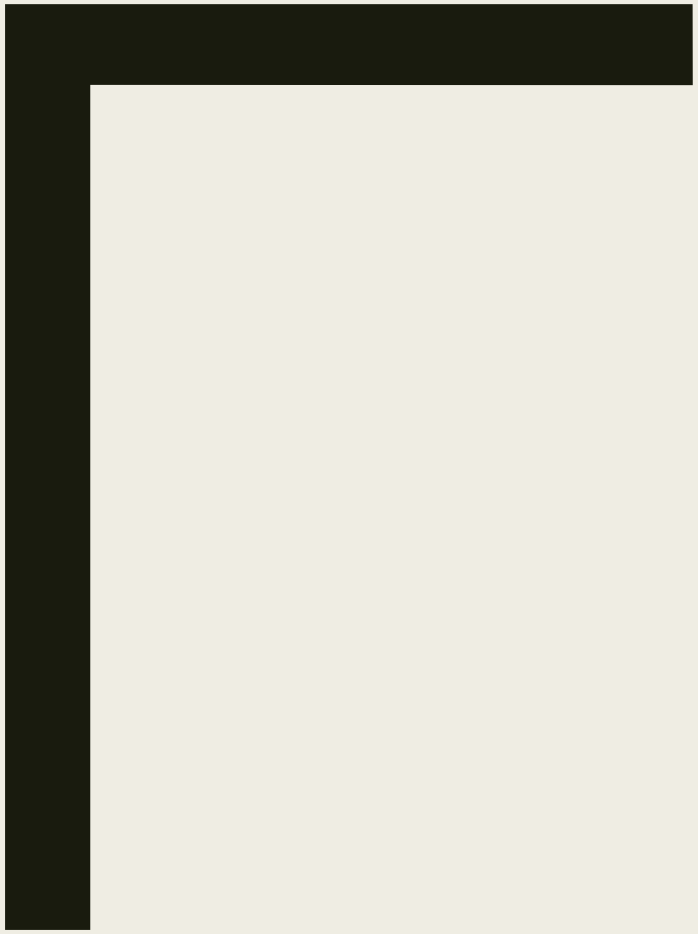
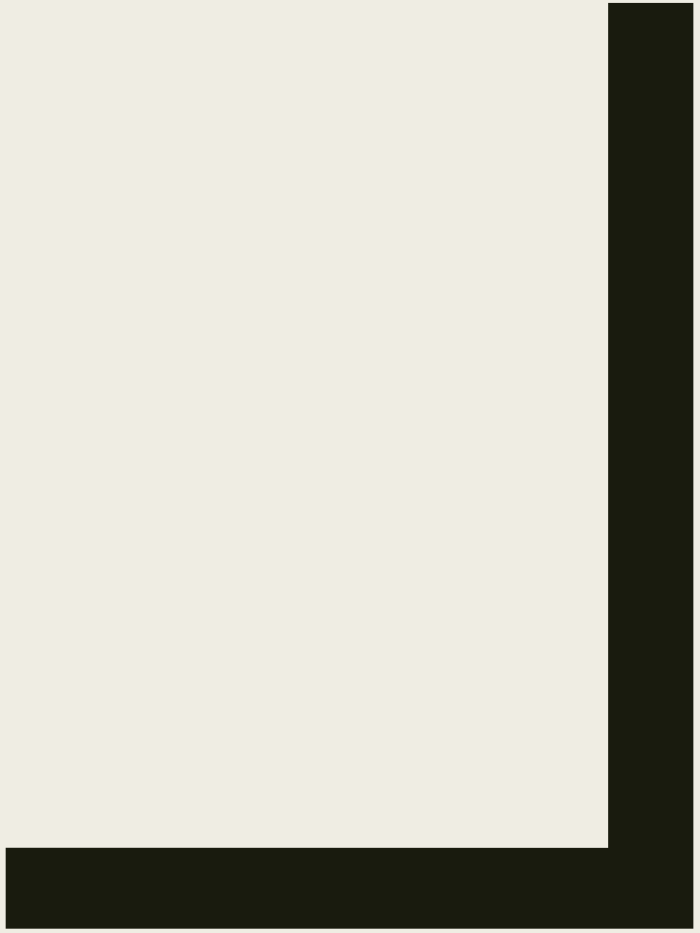




JAVA

Functional programming

FUNCTIONAL PROGRAMMING

How is this different than what I already do?

Can I use functional programming in line of business applications?

Why should I learn functional programming?

COMMON QUESTIONS

Just do one thing

Don’t depend on anything else but their arguments

And always give us the same result

FUNCTIONS

Functional programming Languages?

Functional Programming

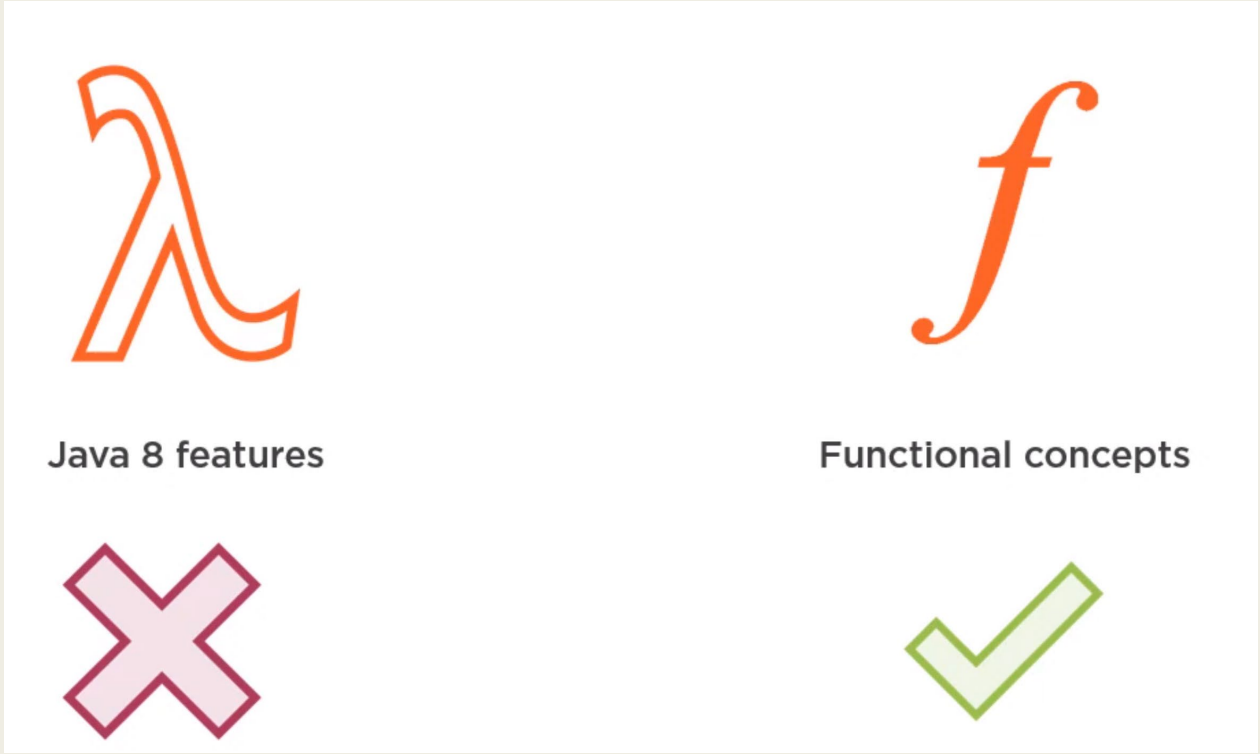
A programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.

It's not the language that makes programming functional.

It's the way you write the code.

Practicing is the key

This lecture is about concepts, not features.



Imperative

```
List<Order> shipped = new ArrayList<>();
for (Order order : orders)
   if (order.isShipped())
      shipped.add(order);
```

Functional

```
List<Order> shipped =
      orders.stream()
            .filter(Order::isShipped)
            .collect(Collectors.toList());
```

The Single Responsibility Principle

```
class Order {
    private Customer customer;
    private OrderStatus orderStatus;

    public void issueRewards() {
        this.orderStatus = OrderStatus.REWARDS_ISSUED;
        if (this.getOrderRewards() != null) {
            this.customer.addToRewardBalance(this.getOrderRewards());
        }
    }
}
```

Split the method in two

```
class Order {
    private Customer customer;
    private OrderStatus orderStatus;

    public void issueRewards() {
        this.orderStatus = OrderStatus.REWARDS_ISSUED;
    }

    public void updateBalanceReward() {
        if (this.getOrderRewards() != null) {
            this.customer.addToRewardBalance(this.getOrderRewards())
        }
    }
}
```

```
order.issueRewards();
```

```
order.issueRewards();
order.updateBalanceReward();
```

```java
class Order {

    private OrderStatus orderStatus;

    public void issueRewards() {
        this.orderStatus = OrderStatus.REWARDS_ISSUED;
    }

    public void updateBalanceReward(Customer customer) {
        if (this.getOrderRewards() != null) {
            customer.addToRewardBalance(this.getOrderRewards());
        }
    }
}
```

```java
order.issueRewards();
order.updateBalanceReward();
```

```java
Customer customer = //...
order.issueRewards();
order.updateBalanceReward(customer);
```

"I fear not the man who has practiced 10,000 kicks once, but I fear the man who has practiced one kick 10,000 times."

"It is better to have 100 functions operate on one data structure than 10 functions on 10 data structures."

Alan Perlis

Functional programming Data Structures

OOP vs. FP

Object-oriented	Functional Programming
More data structures	Fewer data structures
Fewer functions per data structure	More functions per data structure
10 functions on 10 data structures	100 functions on 1 data structure

Unix Philosophy

Make each program do one thing well

Expect the output of every program to become the input of another

Design software to be tried early

Use tools over unskilled labor

Functional Programming Philosophy

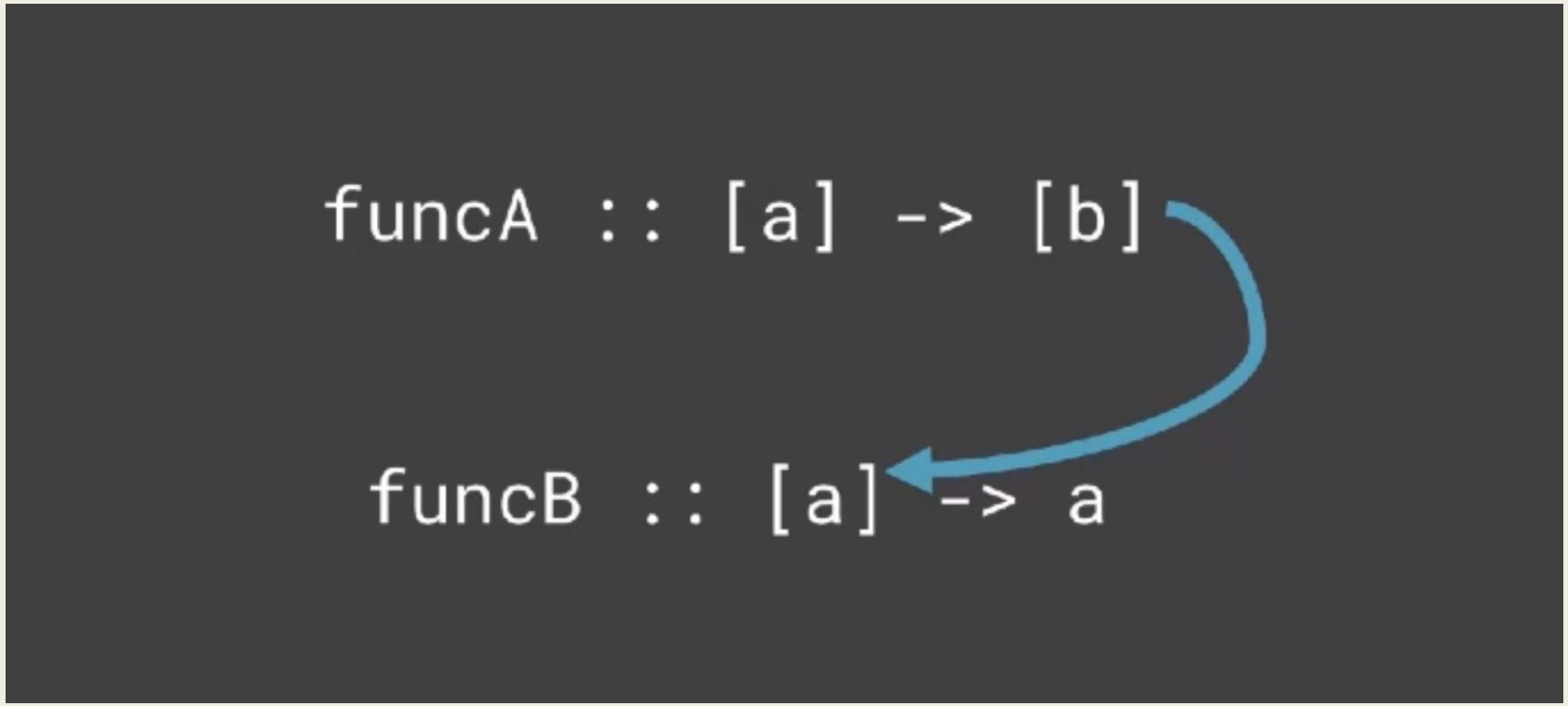
Make each function do one thing well

Expect the output of every function to become the input of another

Design functions to be tested early

Type Signatures

```
findAuthor :: (a, [a]) -> [a]
def findAuthor author, authors
    filter(x => x.contains(author), authors)
end
```


funcA :: [a] -> [b]
funcB :: [a] -> a

Immutable Data

An object whose state cannot be modified after it is created.

How can my program do anything if the data can't change?

Immutability pseudo example

```
def addToCart (cart, item) do
    %{
        items = cart.items.concat(item),
        total = cart.total + item.price
    }
end
```

Eliminate Side Effects

```
customer.getRewardBalance(); // 10

order.issueRewards();

customer.getRewardBalance(); // 15
```

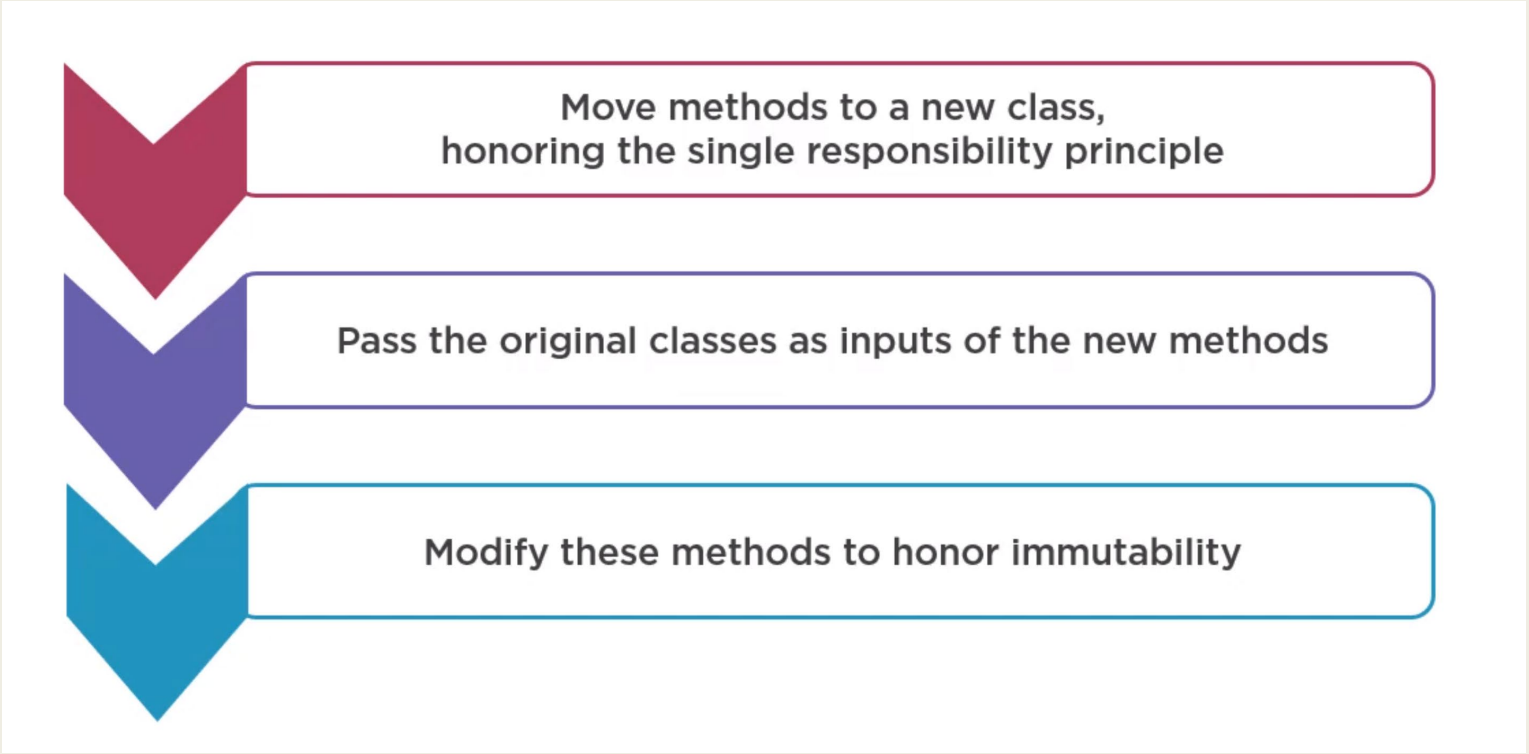
Side effect

- Mutation of variables
- Printing to the console
- Writing to files, databases, or anything in the outside world

Functions Are Black Boxes

Pure Functions Benefits

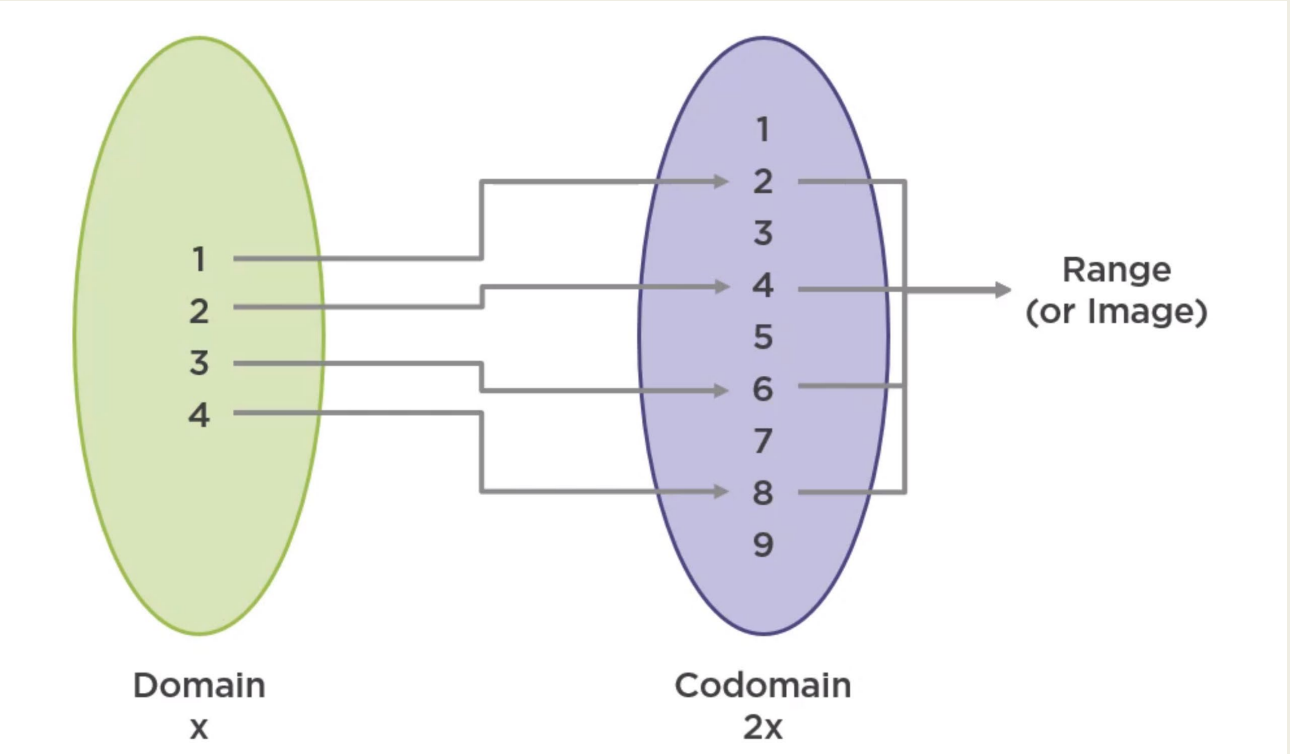
From Imperative to Functional



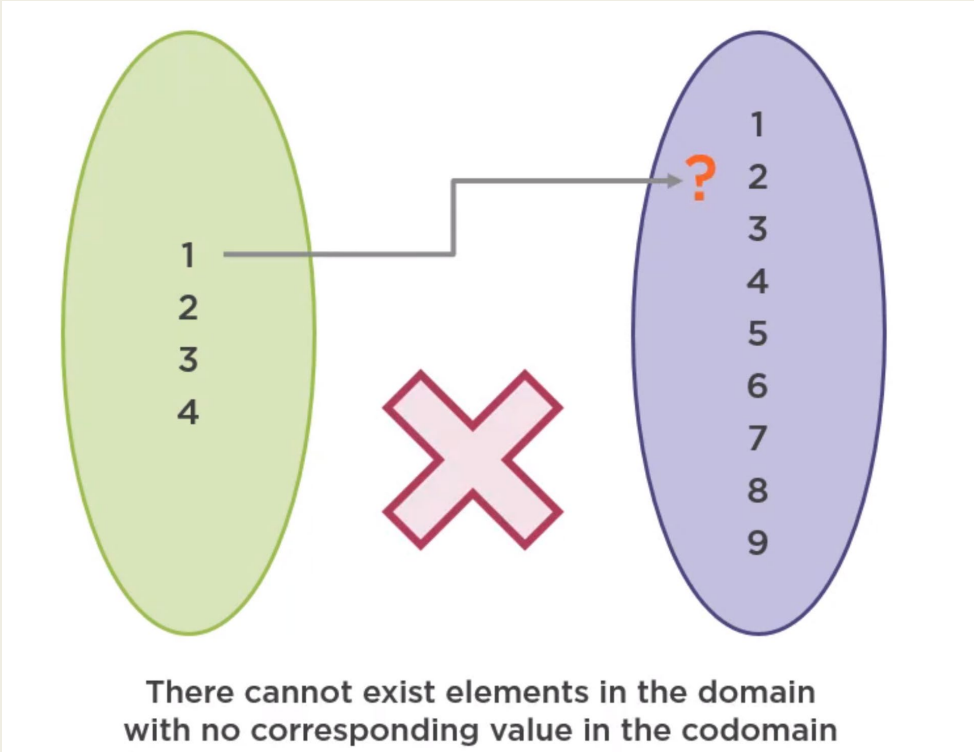
UNDERSTANDING FUNCTIONS

Mathematical Functions domains

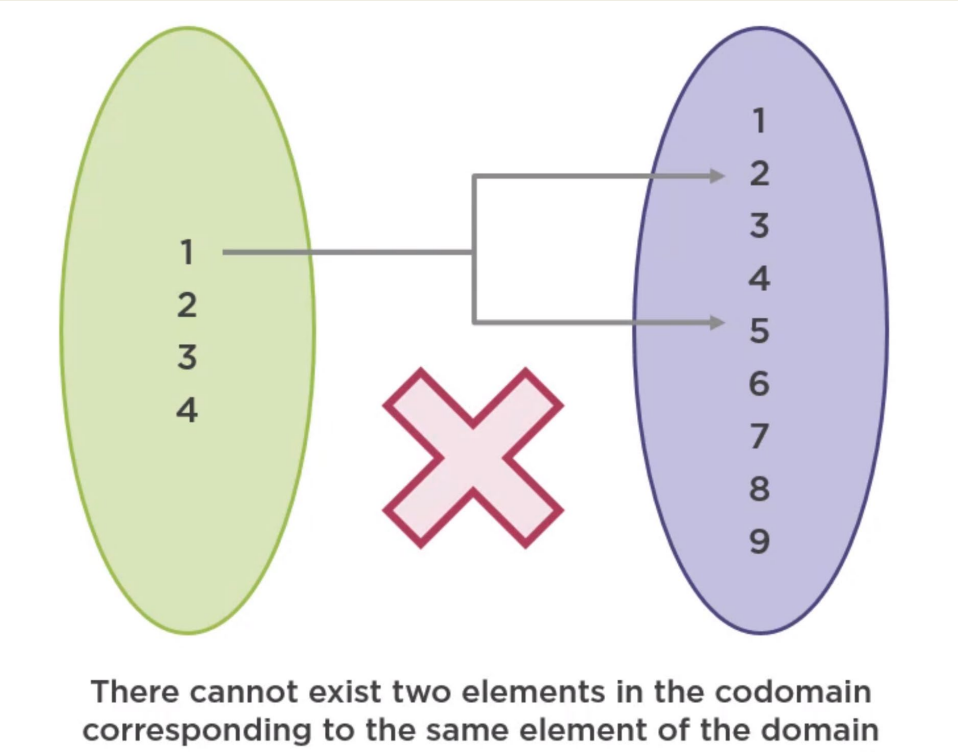
$$f : A \rightarrow B$$



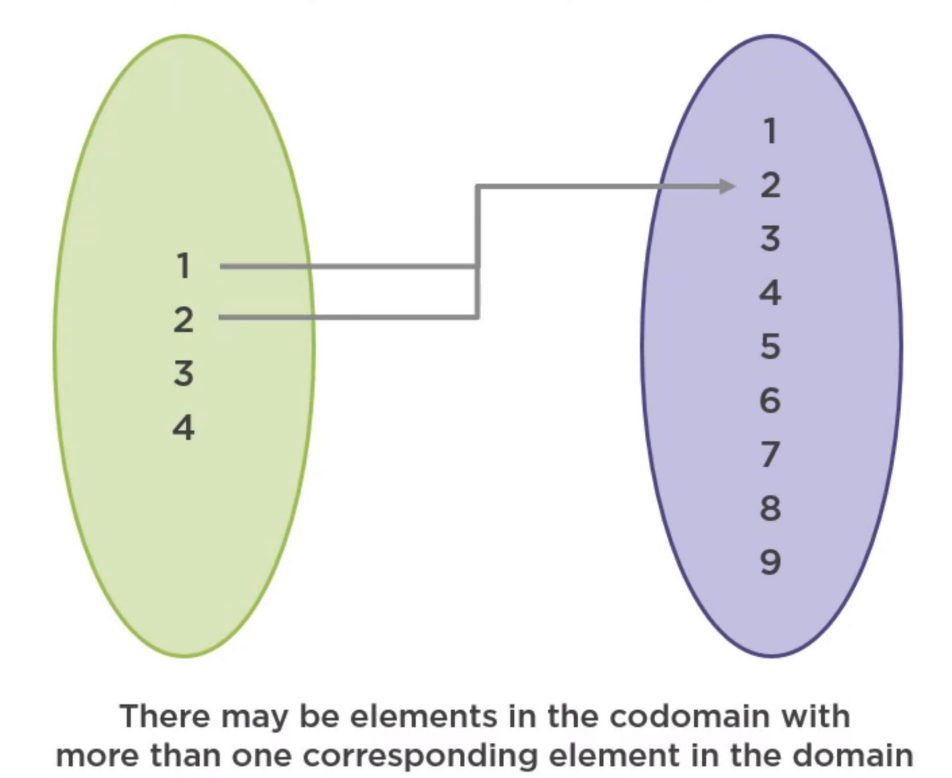
Mathematical Functions domains

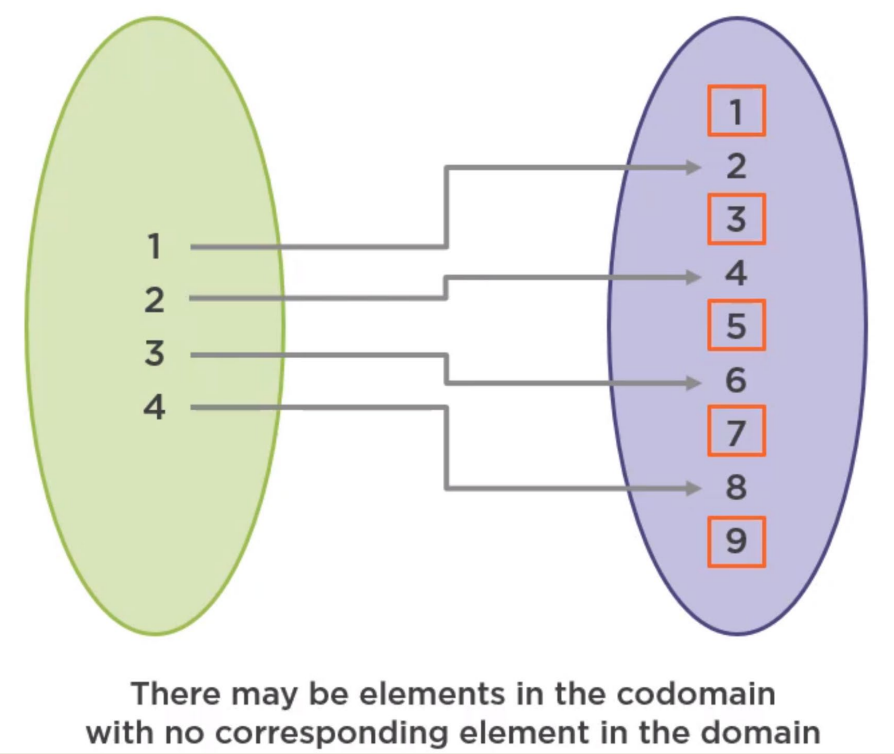


Mathematical Functions domains

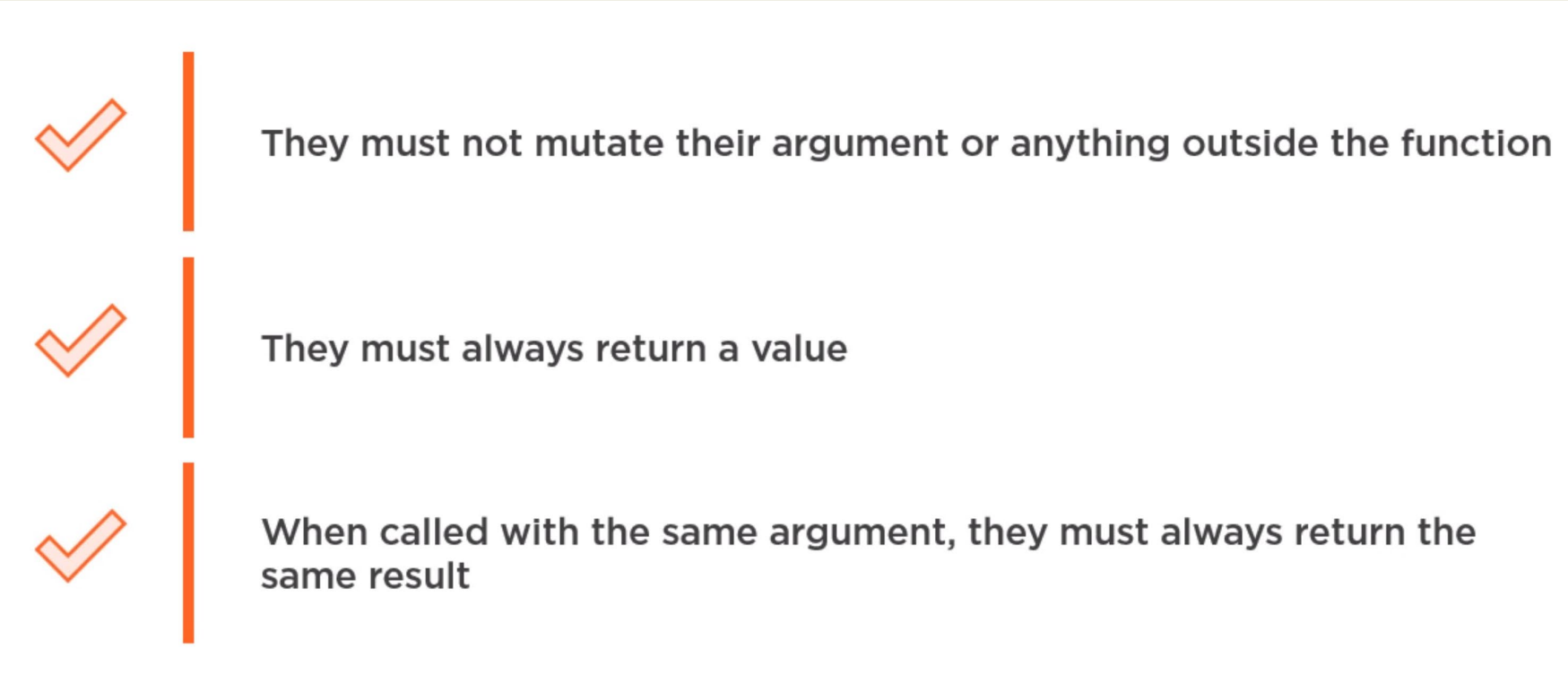


Mathematical Functions domains





Reconcile Programming and Math Functions



Lambda Expression

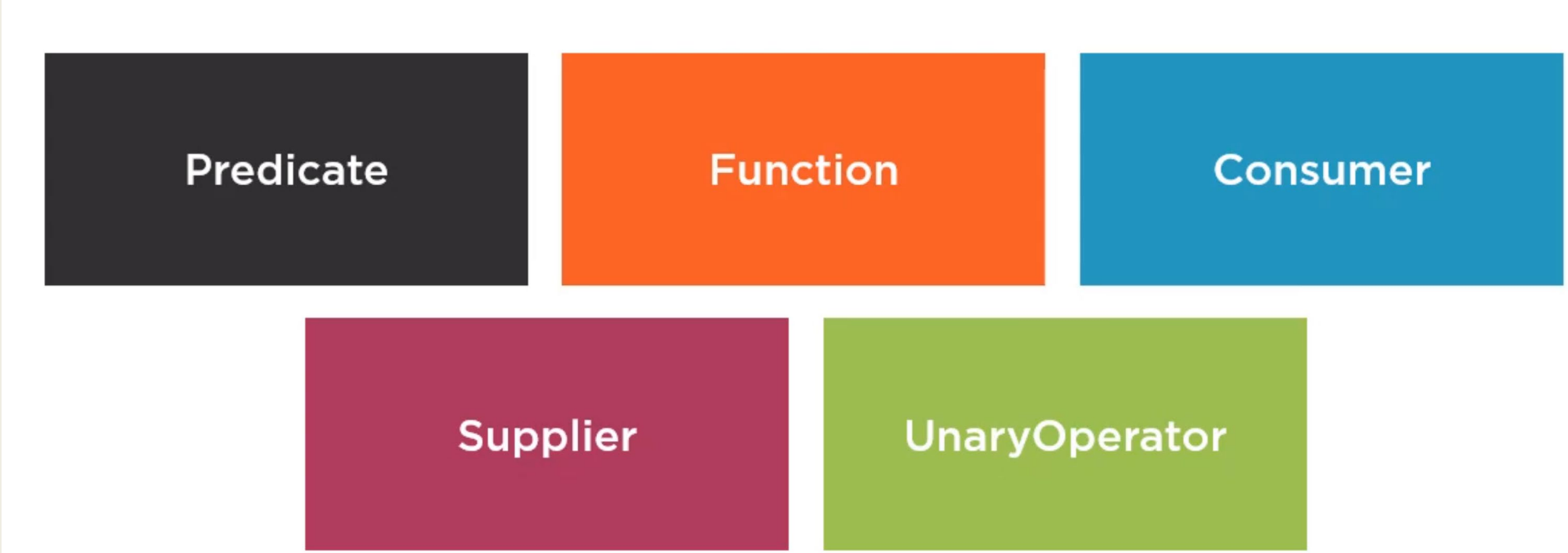
as a lambda expression

```
Function<Integer, Integer> addOne = arg -> arg + 1;
```

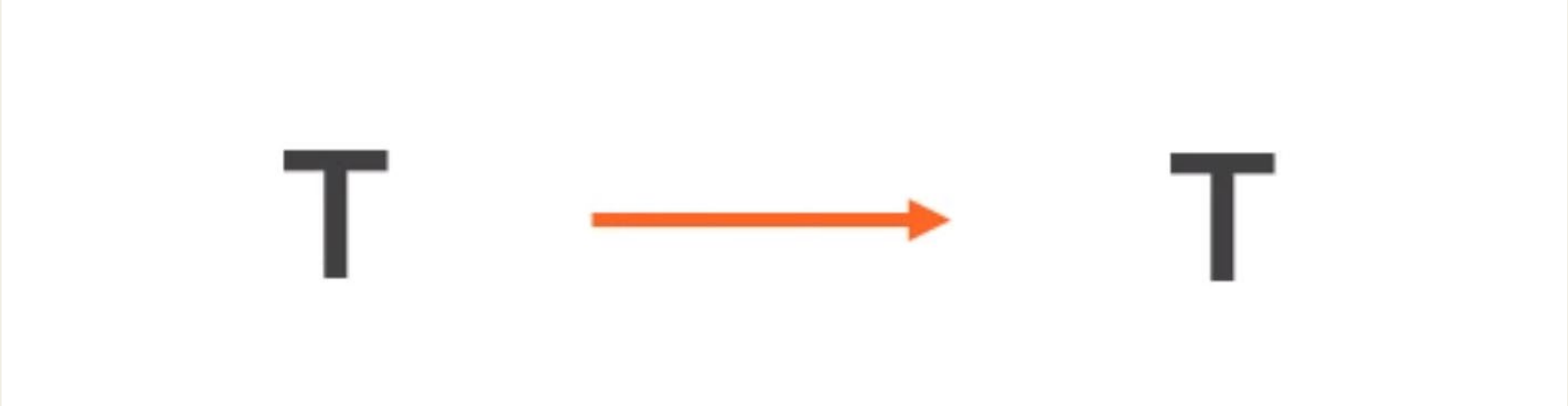
```
System.out.println( addOne.apply(2) );
```

We need to
apply the function

Main Functional Interfaces



Unary operator




```java
UnaryOperator<Integer> doubleIt = x -> x * 2;
UnaryOperator<String> shout = s -> s + "!!!";

System.out.println(doubleIt.apply(6));    // 12
System.out.println(shout.apply("Hey"));   // Hey!!!
```

Predicate


```java
Predicate<Integer> isPositive = x -> x > 0;

System.out.println(isPositive.test(10));  // true
System.out.println(isPositive.test(-5));  // false
```

Consumer & Supplier

```java
Consumer<String> greet = name -> System.out.println("Hello " + name);

greet.accept("Alice");  // Prints: Hello Alice
```

```java
Supplier<Double> lazyPi = () -> 3.14159;

System.out.println(lazyPi.get());  // 3.14159
System.out.println(lazyPi.get());  // 3.14159 again (runs every time)
```

```
hFunc( l -> l + 1 )
```

Takes a function as its input

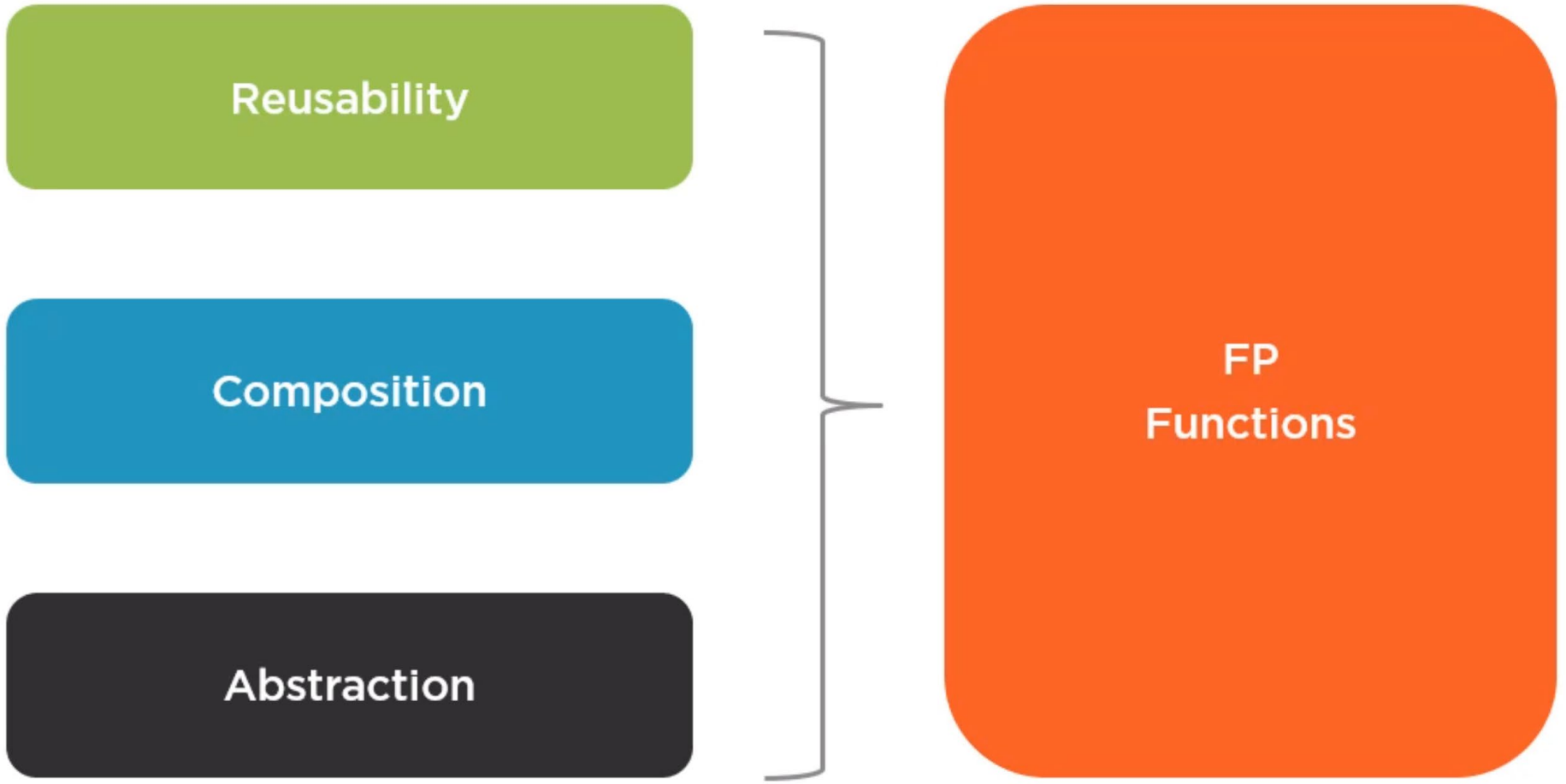
```
Function<Long, Long>
f = hFunc()
```

Returns a function as its output

```
Function<Long, Long>
f = hFunc( l -> l + 1 )
```

Or both

HIGH-ORDER FUNCTION


Reusability
Composition
Abstraction
FP
Functions

```java
List<Integer> filteredList = new ArrayList<Integer>();
for (int n : listOfNumbers) {
    if (n % 3 == 0) {
        filteredList.add(n);
    }
}
```

```java
List<Integer> filteredList = listOfNumbers.stream()
                                          .filter(n -> n % 3 == 0)
                                          .collect(Collectors.toList());
```

```java
Predicate<Integer> divisableBy3 = n -> n % 3 == 0;
List<Integer> filteredList = listOfNumbers.stream()
                                          .filter(divisableBy3)
                                          .collect(Collectors.toList());
```

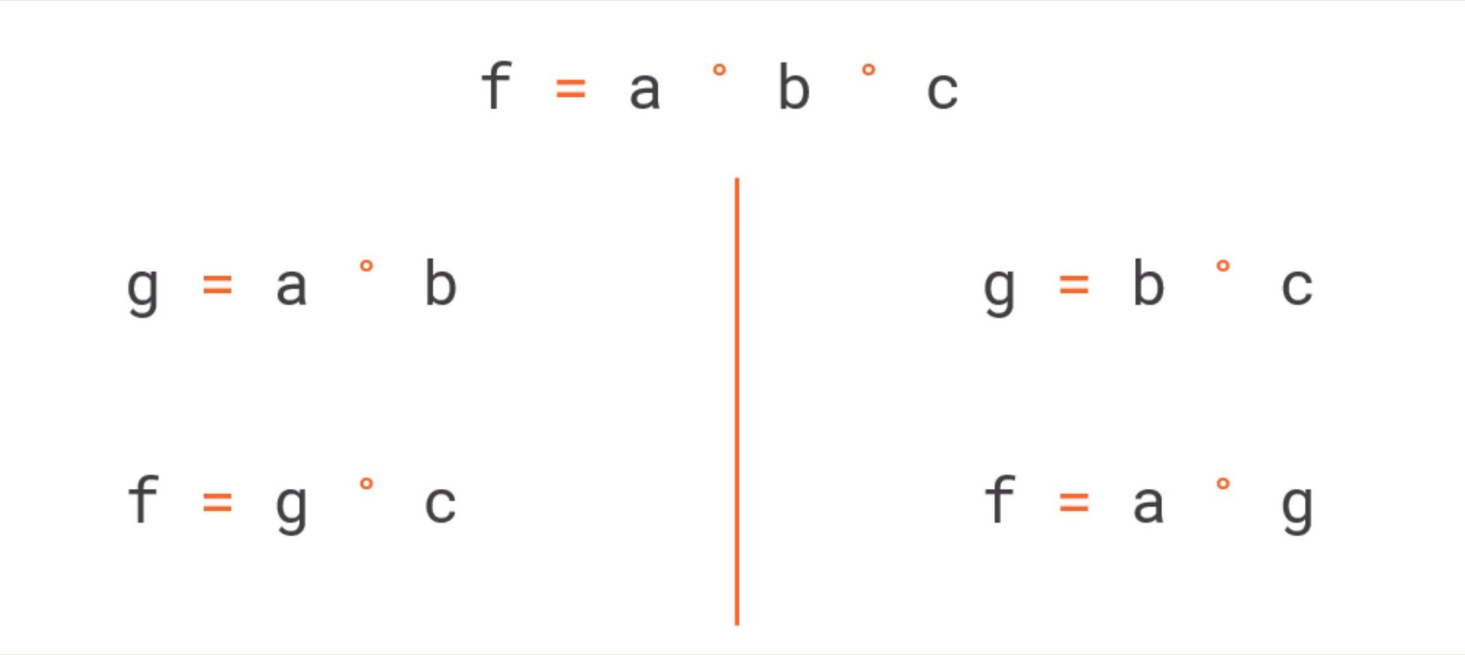
Composition

- Nesting functions, passing the result of one function as the input of the next

```
f( x ) = x + 10

g( x ) = x * 10

f ∘ g ( x ) = f( g( x ) )

                f( x * 10 )

                 ( x * 10 ) + 10
```

Associative property

- Mathematical principle that proves that the grouping of values does not affect the results



Why Functional Programming Matters

Caching

Laziness

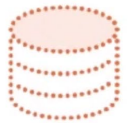
Parallelism and Concurrency

CACHING EXAMPLE

Sales Dashboard Application

Performance dashboard

Slow changing data



Caching results

Memoization

To speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again.

Cached Function

```
def calculateTotalPrice(items) do
    if(cache.contains(items)) do
        cache[items]
    else
        tot = reduce((item, acc) => acc = acc + item.price, 0, items)
        cache[items] = tot
        tot
    end
end
```

Lazy Evaluation

delays the evaluation of an expression until its value is needed

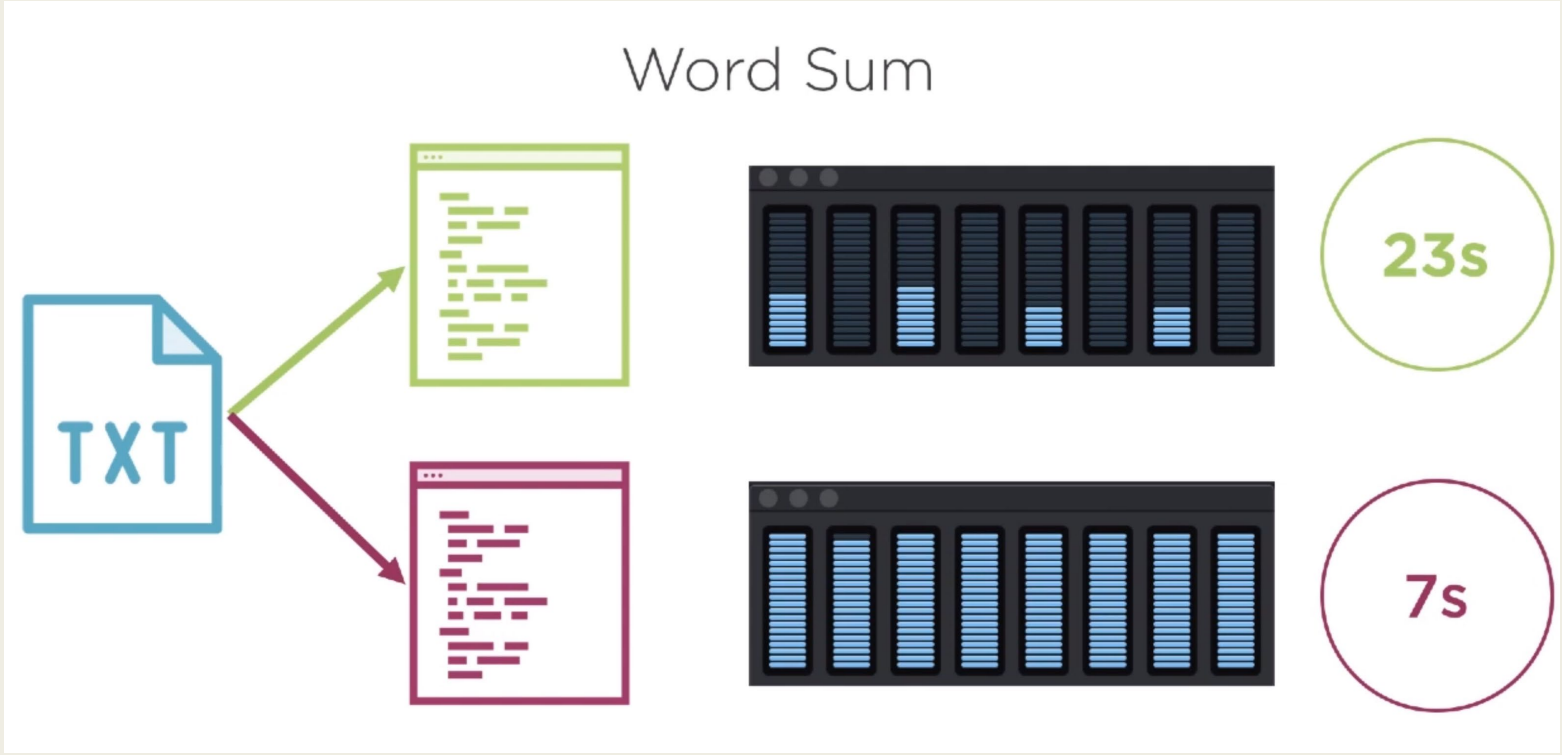
LAZINESS

Lazy Evaluation defers execution

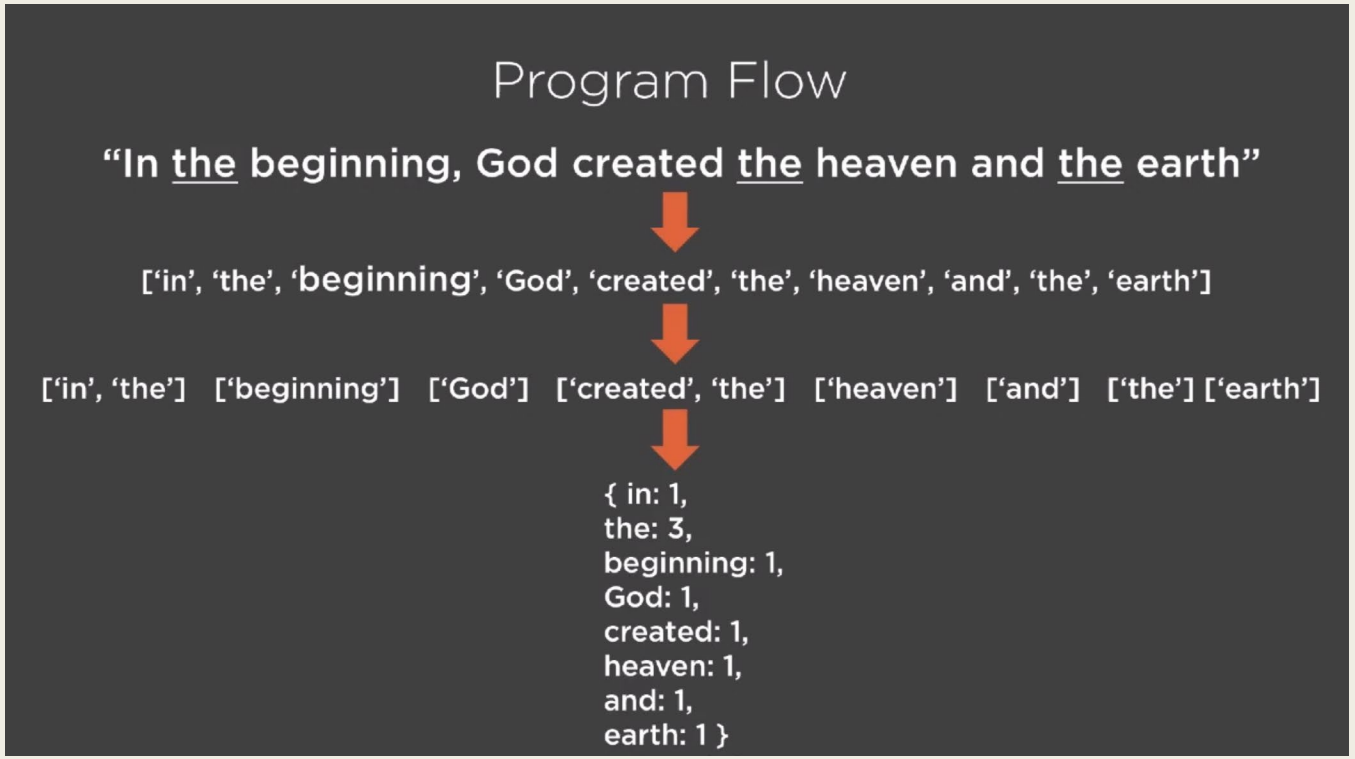
Lazy Evaluation: Database

```
nebraskaTaylors = users
  |> where(lastName = 'Taylor')
  |> where(state = 'NE')
  |> orderBy('firstName')
```

Parallelism and Concurrency



Parallelization due to immutability



FUNCTIONAL PROGRAMMING IS EQUIPPED FOR THE FUTURE!

QUESTIONS?