Use Functional Programming to Make a Bulls and Cows Solver

用 Functional Programming 來解 1A2B 吧

smailzhu

July 31, 2022

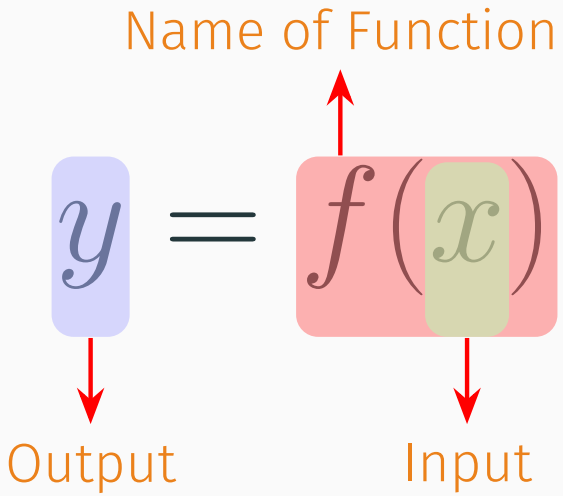
Any question? sli.do/coscup22-fp

Table of contents

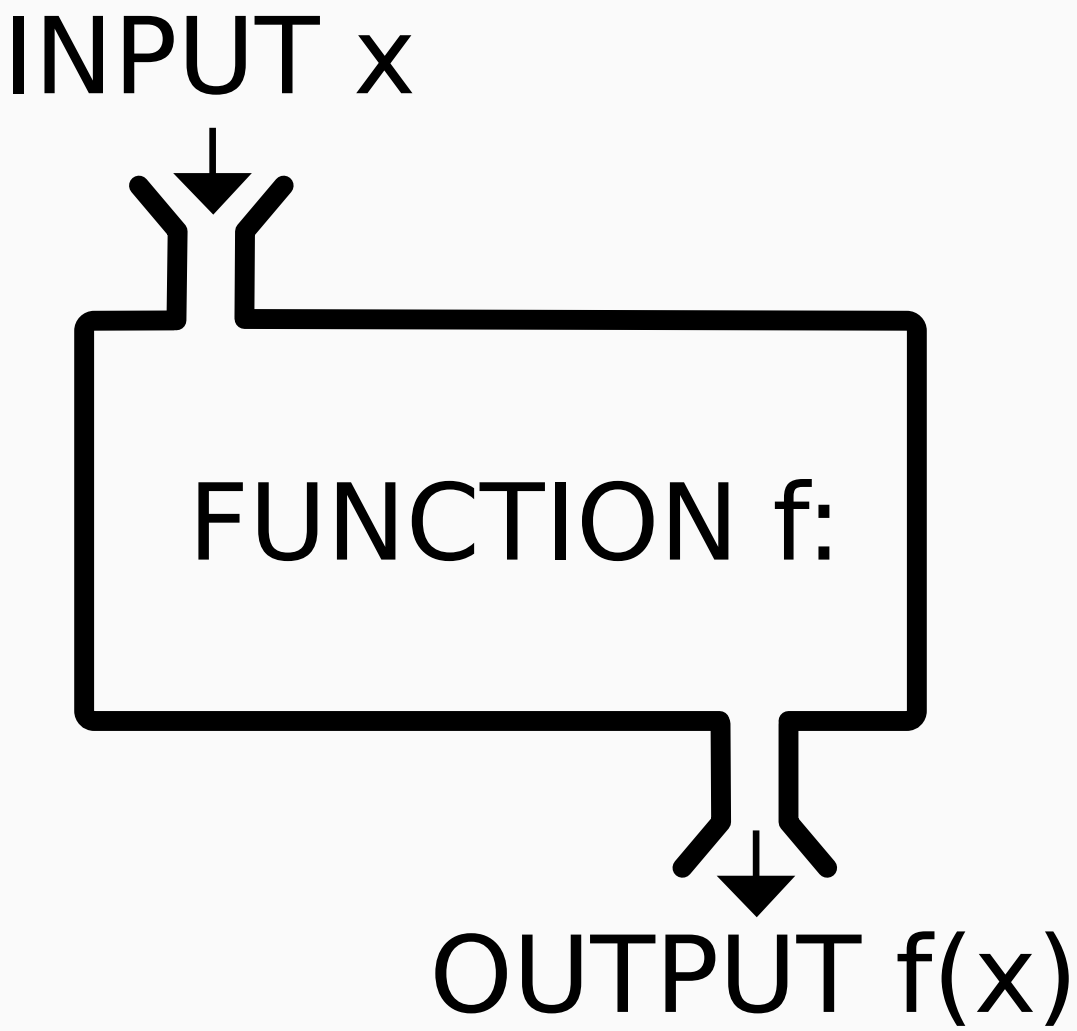
sli.do/coscup22-fp

What is Functional Programming

What is a Function?



What is a Function?



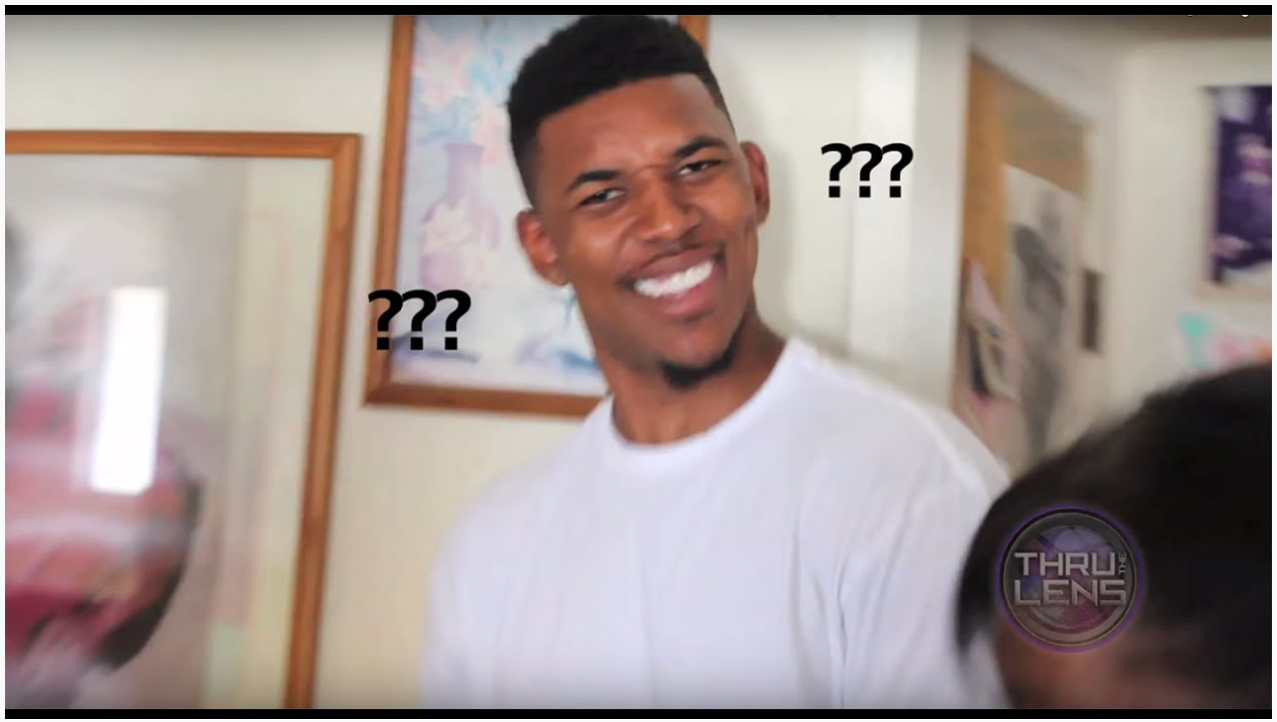
What is Functional Programming?

Functional programming represents a programming paradigm in which the computations are evaluated by mathematical functions. The paradigm avoids changing state and using mutable data.[1]

Functional programming **代表一種以數學函式來計算的程式設計法。這種設計法避免狀態改變及使用可變資料。**

[1] Stefania Loredana Nita and Marius Mihailescu. *Haskell Quick Syntax Reference*. Apress, 2019.

Huh???



What is Functional Programming?

In a nutshell, NO SIDE EFFECTS!

What is Functional Programming?

In a nutshell, NO SIDE EFFECTS!

“副作用就像是謊言。”

— **無瑕的程式碼**, Robert C. Martin

What is Side Effects?

```
1  var x = 2;
2
3  console.log(x) // 2
4
5  function add2(){
6      x = x + 2;
7  }
```

What is Side Effects?

```
1  var x = 2;
2
3  console.log(x) // 2
4
5  function add2(){
6      x = x + 2;
7  }

8  add2();
9
10 console.log(x); // 4
```

What is Side Effects?

```
1  var x = 2;
2
3  console.log(x) // 2
4
5  function add2(){
6      x = x + 2;
7  }

8  add2();
9
10 console.log(x); // 4
```

Not functional, but how to fix it?

Let it be Functional

```
1  var x = 2;
2
3  console.log(x) // 2
4
5  function add2(y: number){
6      return y + 2;
7  }
```

Let it be Functional

```
1  var x = 2;
2
3  console.log(x) // 2
4
5  function add2(y: number){
6      return y + 2;
7  }

8  add2(x);
9
10 console.log(x); // 2
```

Let it be Functional

```
1  var x = 2;
2
3  console.log(x) // 2
4
5  function add2(y: number){
6      return y + 2;
7  }

8  add2(x);
9
10 console.log(x); // 2
```

No side effects ヽ(o ^ ▽ ^ o)ノ

Everything is function

Everything is function

Functions are first class

Everything is function

Functions are first class

can be passed, returned, assigned, etc..

Everything is function

Functions are first class

can be passed, returned, assigned, etc..

Data is immutable

Everything is function

Functions are first class

can be passed, returned, assigned, etc..

Data is immutable

State cannot change after creation

Quite cute, right?

Questions?

sli.do/coscup22-fp

Let’s Make a Bulls and Cows Solver

1A2B

- Secret number: 4271
- Opponent's try: 1234
- Answer: 1A2B (1 bull and 2 cows)
  If the matching digits are in their right positions, they are "bulls", if in different positions, they are "cows".

Bulls and Cows Example

Secret number: `9527`

Source: `https://github.com/smailzhu/COSCUP2022`

Bulls and Cows Example

Secret number: `9527`

- I guess "`34`$\underline{\texttt{72}}$" $\rightarrow$ 0A2B

Source: `https://github.com/smailzhu/COSCUP2022`

Bulls and Cows Example

Secret number: `9527`

- I guess "`34`$\underline{\texttt{72}}$" $\rightarrow$ 0A2B
- I guess "$\underline{\texttt{2}}$`064`" $\rightarrow$ 0A1B

Source: `https://github.com/smailzhu/COSCUP2022`

Bulls and Cows Example

Secret number: `9527`

- I guess "`34`$\underline{\texttt{72}}$" → 0A2B
- I guess "$\underline{\texttt{2}}$`064`" → 0A1B
- I guess "9$\underline{\texttt{7}}$`48`" → 1A1B

Source: `https://github.com/smailzhu/COSCUP2022`

Bulls and Cows Example

Secret number: `9527`

- I guess "`34`$\underline{\texttt{72}}$" → 0A2B
- I guess "$\underline{\texttt{2}}$`064`" → 0A1B
- I guess "`9`$\underline{\texttt{7}}$`48`" → 1A1B
- I guess "`1547`" → 2A0B

Source: `https://github.com/smailzhu/COSCUP2022`

Bulls and Cows Example

Secret number: `9527`

- I guess "`3472`" → 0A2B
- I guess "`2064`" → 0A1B
- I guess "`9748`" → 1A1B
- I guess "`1547`" → 2A0B
- I guess "`1349`" → 0A1B

Source: `https://github.com/smailzhu/COSCUP2022`

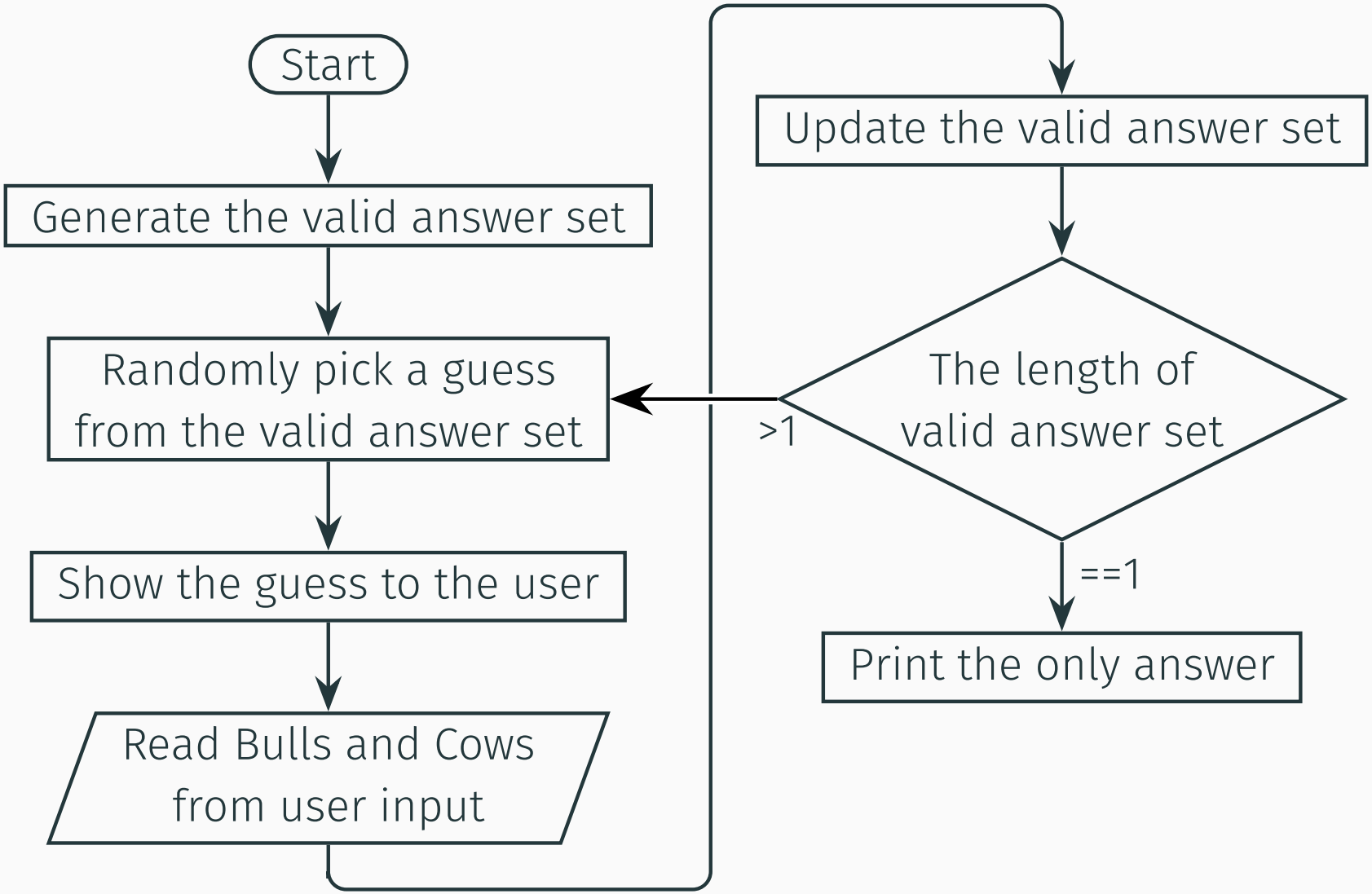
Bulls and Cows Example

Secret number: `9527`

- I guess "`3472`" → 0A2B
- I guess "`2064`" → 0A1B
- I guess "`9748`" → 1A1B
- I guess "`1547`" → 2A0B
- I guess "`1349`" → 0A1B
- "9527"

Source: `https://github.com/smailzhu/COSCUP2022`

Flow Chart


Start
Generate the valid answer set
Randomly pick a guess from the valid answer set
Show the guess to the user
Read Bulls and Cows from user input
Update the valid answer set
The length of valid answer set
>1
==1
Print the only answer

Generate all valid answer

What is valid answer?

- without duplicate digit. e.g. 1234, 9527, ~~1231~~, ~~6666~~

Generate all valid answer

What is valid answer?

- without duplicate digit. e.g. 1234, 9527, ~~1231~~, ~~6666~~

so,

1. Generate all permutation
2. Remove invalid permutation

Check list has any duplicate elements

- **nub** can remove duplicate elements in a list

```
nub [1,2,3,4,3,2,1,2,4,3,5] -- [1,2,3,4,5]
```

Check list has any duplicate elements

- `nub` can remove duplicate elements in a list

```
nub [1,2,3,4,3,2,1,2,4,3,5] -- [1,2,3,4,5]
```

- make magic happen

```
1 {- check if list has duplicate element
2  -
3  - hasDuplicates [1,2,3,4] == False
4  - hasDuplicates [1,2,3,1] == True -}
5
6 hasDuplicates :: Eq a => [a] -> Bool
7 hasDuplicates xs = length (nub xs) /= length xs
```

Generate all valid answer

1. `replicateM`: replicateM n act performs the action act n times, and then returns the list of results:

```
replicateM 4 [1,2,3,4,5]
--[[1,1,1,1],[1,1,1,2],...,[5,5,5,4],[5,5,5,5]]
```

Generate all valid answer

1. `replicateM`: replicateM n act performs the action act n times, and then returns the list of results:

```
replicateM 4 [1,2,3,4,5]
--[[1,1,1,1],[1,1,1,2],...,[5,5,5,4],[5,5,5,5]]
```

2. Remove invalid permutation

```
filter (>5) [1,2,3,4,5,6,7,8] -- [6,7,8]
filter (\x -> length x > 2) ["aaaa","bbb","cc"]
    -- ["aaaa", "bbb"]
```

Generate all valid answer

1. `replicateM`: replicateM n act performs the action act n times, and then returns the list of results:

```
replicateM 4 [1,2,3,4,5]
--[[1,1,1,1],[1,1,1,2],...,[5,5,5,4],[5,5,5,5]]
```

2. Remove invalid permutation

```
filter (>5) [1,2,3,4,5,6,7,8] -- [6,7,8]
filter (\x -> length x > 2) ["aaaa","bbb","cc"]
    -- ["aaaa", "bbb"]
```

3. Combine them

```
allAnswer n x = filter (\x -> not $
   hasDuplicates x) $ replicateM n x
allAnswer 4 [0..9] --
   [[1,2,3,4],[1,2,3,5],...,[6,7,8,9]]
```

Before We Count Bulls

- What is `(x:xs)`?

```
sum :: [Int] -> Int
sum [] = 0
sum (x:xs) = x + sum xs
```

Before We Count Bulls

- What is `(x:xs)` ?

```
sum :: [Int] -> Int
sum [] = 0
sum (x:xs) = x + sum xs
```

- For example: `sum [1,2,3]`

Before We Count Bulls

- What is `(x:xs)` ?

```
sum :: [Int] -> Int
sum [] = 0
sum (x:xs) = x + sum xs
```

- For example: `sum [1,2,3]`

```
sum [1,2,3] = 1 + sum [2,3]
```

Before We Count Bulls

- What is `(x:xs)` ?

```
sum :: [Int] -> Int
sum [] = 0
sum (x:xs) = x + sum xs
```

- For example: `sum [1,2,3]`

```
sum [1,2,3] = 1 + sum [2,3]
sum [2,3] = 2 + sum [3]
```

Before We Count Bulls

- What is `(x:xs)` ?

```
sum :: [Int] -> Int
sum [] = 0
sum (x:xs) = x + sum xs
```

- For example: `sum [1,2,3]`

```
sum [1,2,3] = 1 + sum [2,3]
sum [2,3] = 2 + sum [3]
sum [3] = 3 + sum []
```

Before We Count Bulls

- What is `(x:xs)` ?

```
sum :: [Int] -> Int
sum [] = 0
sum (x:xs) = x + sum xs
```

- For example: `sum [1,2,3]`

```
sum [1,2,3] = 1 + sum [2,3]
sum [2,3] = 2 + sum [3]
sum [3] = 3 + sum []
sum [] = 0
```

Count Bulls

- What is `(x:xs)` ?

```
sum :: [Int] -> Int
sum [] = 0
sum (x:xs) = x + sum xs
```

- How to count Bulls?

```
checkA :: Eq a => [a] -> [a] -> Int
checkA [] [] = 0
checkA (x:xs) (y:ys)
           | x == y    = 1 + (checkA xs ys)
           | otherwise = checkA xs ys
```

Count Bulls

- How to count Bulls?

```haskell
checkA :: Eq a => [a] -> [a] -> Int
checkA [] [] = 0
checkA (x:xs) (y:ys)
            | x == y    = 1 + (checkA xs ys)
            | otherwise = checkA xs ys
```

- Example

```haskell
checkA [4,2,7,1] [1,2,3,4]
```

Count Bulls

- How to count Bulls?

```haskell
checkA :: Eq a => [a] -> [a] -> Int
checkA [] [] = 0
checkA (x:xs) (y:ys)
            | x == y    = 1 + (checkA xs ys)
            | otherwise = checkA xs ys
```

- Example

```
checkA [4,2,7,1] [1,2,3,4]
4\=1 -> checkA [2,7,1] [2,3,4]
```

Count Bulls

- How to count Bulls?

```haskell
checkA :: Eq a => [a] -> [a] -> Int
checkA [] [] = 0
checkA (x:xs) (y:ys)
            | x == y    = 1 + (checkA xs ys)
            | otherwise = checkA xs ys
```

- Example

```
checkA [4,2,7,1] [1,2,3,4]
4\=1 -> checkA [2,7,1] [2,3,4]
2==2 -> 1 + checkA [7,1] [3,4]
```

Count Bulls

- How to count Bulls?

```haskell
checkA :: Eq a => [a] -> [a] -> Int
checkA [] [] = 0
checkA (x:xs) (y:ys)
            | x == y    = 1 + (checkA xs ys)
            | otherwise = checkA xs ys
```

- Example

```
checkA [4,2,7,1] [1,2,3,4]
4\=1 -> checkA [2,7,1] [2,3,4]
2==2 -> 1 + checkA [7,1] [3,4]
7\=3 -> checkA [1] [4]
```

Count Bulls

- How to count Bulls?

```haskell
checkA :: Eq a => [a] -> [a] -> Int
checkA [] [] = 0
checkA (x:xs) (y:ys)
            | x == y    = 1 + (checkA xs ys)
            | otherwise = checkA xs ys
```

- Example

```
checkA [4,2,7,1] [1,2,3,4]
4\=1 -> checkA [2,7,1] [2,3,4]
2==2 -> 1 + checkA [7,1] [3,4]
7\=3 -> checkA [1] [4]
1\=4 -> checkA [] []
```

Before We Count Cows

1. Subtract

```
subtract 3 5 -- 2
subtract 6 3 -- -3
```

Before We Count Cows

1. Subtract

```
subtract 3 5 -- 2
subtract 6 3 -- -3
```

2. Check if element in a list

```
elem 1 [1,2,3,4,5] -- True
elem 14 [1..10] -- False
```

Before We Count Cows

1. Subtract

```
subtract 3 5 -- 2
subtract 6 3 -- -3
```

2. Check if element in a list

```
elem 1 [1,2,3,4,5] -- True
elem 14 [1..10] -- False
```

3. Count elements in a list

```
filter (>5) [1,2,3,4,5,6,7,8] -- [6,7,8]
length [6,7,8] -- 3
length $ filter (>5) [1..8] -- 3
```

Before We Count Cows

1. Subtract

```
subtract 3 5 -- 2
subtract 6 3 -- -3
```

2. Check if element in a list

```
elem 1 [1,2,3,4,5] -- True
elem 14 [1..10] -- False
```

3. Count elements in a list

```
filter (>5) [1,2,3,4,5,6,7,8] -- [6,7,8]
length [6,7,8] -- 3
length $ filter (>5) [1..8] -- 3
```

4. Map (higher-order function)

```
map square [1, 2, 3, 4, 5] -- [1, 4, 9, 16, 25]
```

Magic to Count Cows

1. Before we count Cows

```
elem 1 [1,2,3,4,5] -- True
map square [1..5] -- [1,4,9,16,25]
```

2. Count the number of same elements in two lists

```
f1 xs ys = map (\y -> elem y xs) ys
f1 [2,3,4] [5,2,4] -- [False,True,True]
-- map (\y -> elem y [2,3,4]) [5,2,4]
```

Magic to Count Cows

1. Before we count Cows

```
elem 1 [1,2,3,4,5] -- True
map square [1..5] -- [1,4,9,16,25]

length $ filter (>5) [1..8] -- 3
```

2. Count the number of same elements in two lists

```
f1 xs ys = map (\y -> elem y xs) ys
f1 [2,3,4] [5,2,4] -- [False,True,True]
-- map (\y -> elem y [2,3,4]) [5,2,4]

f2 xs ys = length $ filter (True==) $ f1 xs ys
f2 [2,3,4] [5,2,4] -- 2
-- length $ filter (True==) [False,True,True]
```

Let's Count Cows

1. Count the number of same elements in two lists

```
f1 xs ys = map (\y -> elem y xs) ys
f1 [2,3,4] [5,2,4] -- [False,True,True]
-- map (\y -> elem y [2,3,4]) [5,2,4]

f2 xs ys = length $ filter (True==) $ f1 xs ys
f2 [2,3,4] [5,2,4] -- 2
-- length $ filter (True==) [False,True,True]
```

2. How to count cows?

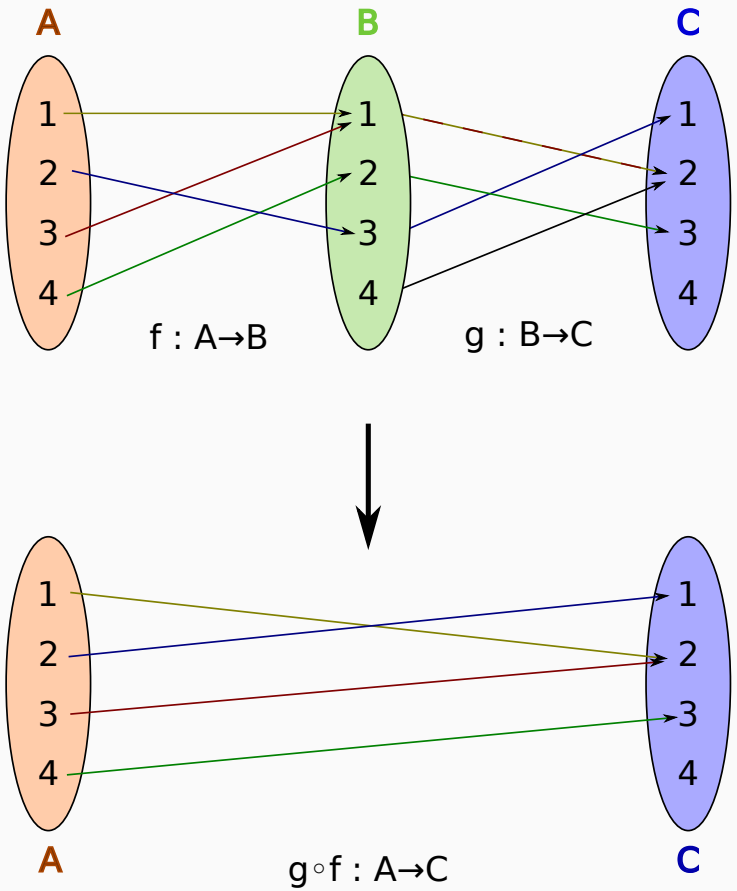
```
checkAB :: Eq a => [a] -> [a] -> (Int, Int)
checkAB xs ys = (a_num, b_num)
  where a_num = checkA xs ys
        b_num = subtract a_num $ length $
           filter (True==) $ map (\y -> elem y
           xs) ys
```

Questions?

sli.do/coscup22-fp

Function Composition

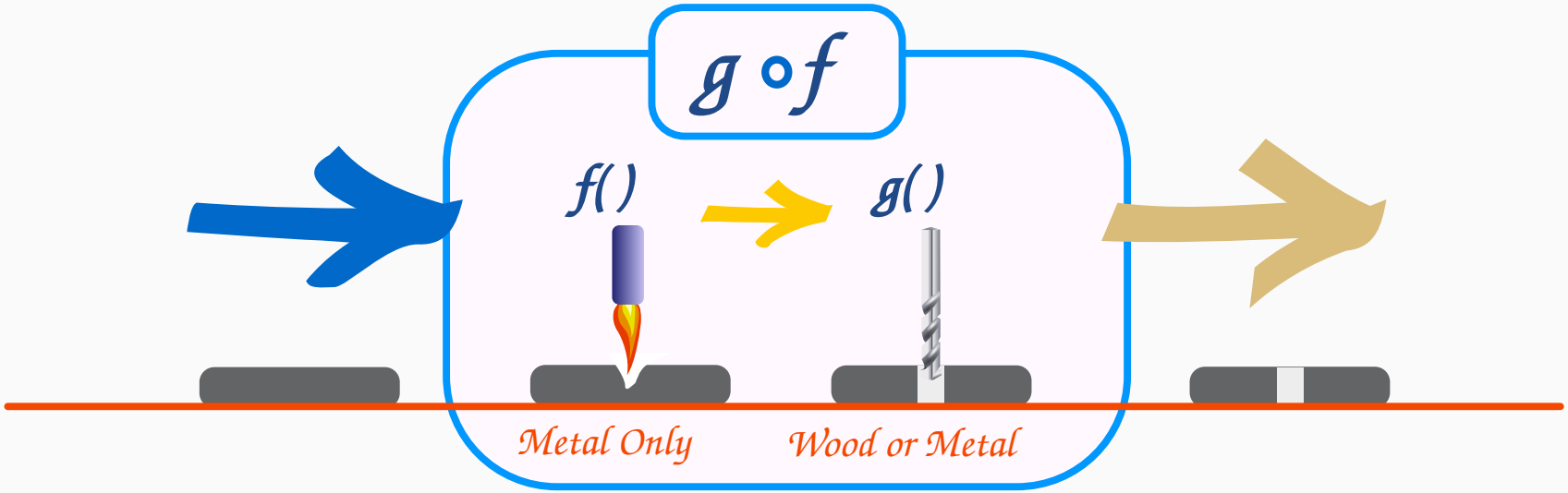
$$g \circ f(x) = g(f(x))$$



Example for a composition of two functions [2]

[2] From wikimedia commons by Stephan Kulla

Function Composition



Composition of Functions [3]

[3] https://www.mathsisfun.com/sets/functions-composition.html

Update Valid Answer

```
{- Update answer pool
 - choose the solutions which match the rules base
    on the test
 -}
updateAnswers :: Eq a => [[a]] -> [a] -> (Int, Int)
    -> [[a]]
updateAnswers ans test rules
        = filter ((rules==).(checkAB test)) ans
```

Questions?

sli.do/coscup22-fp