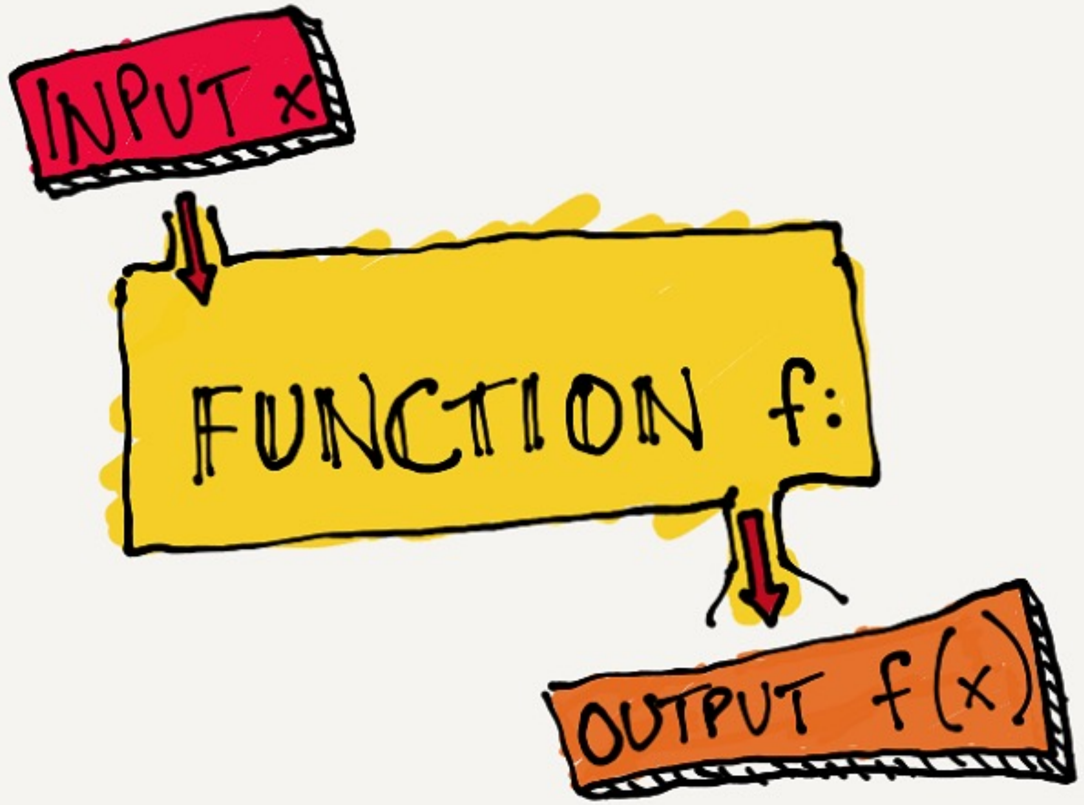
ALGORITHM AND COMPUTATIONAL THINKING 2

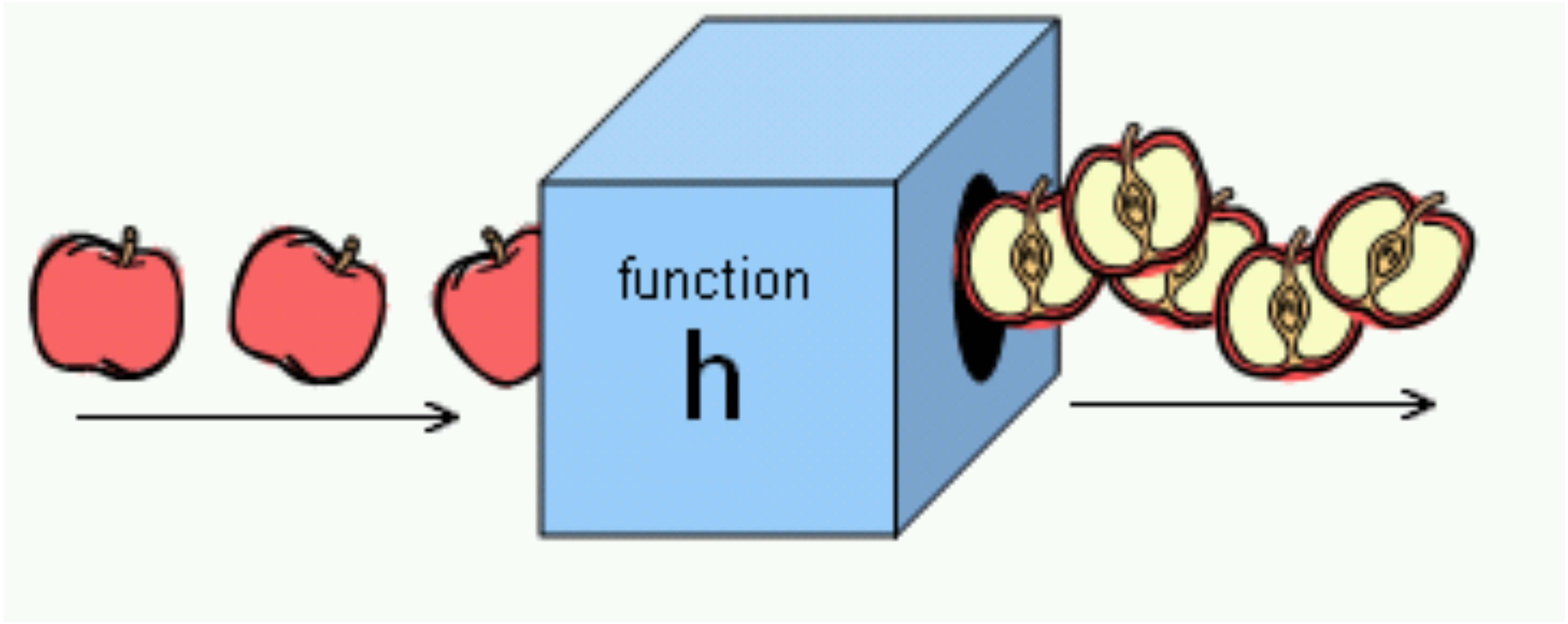
WEEK 2 – Functions (part 1)



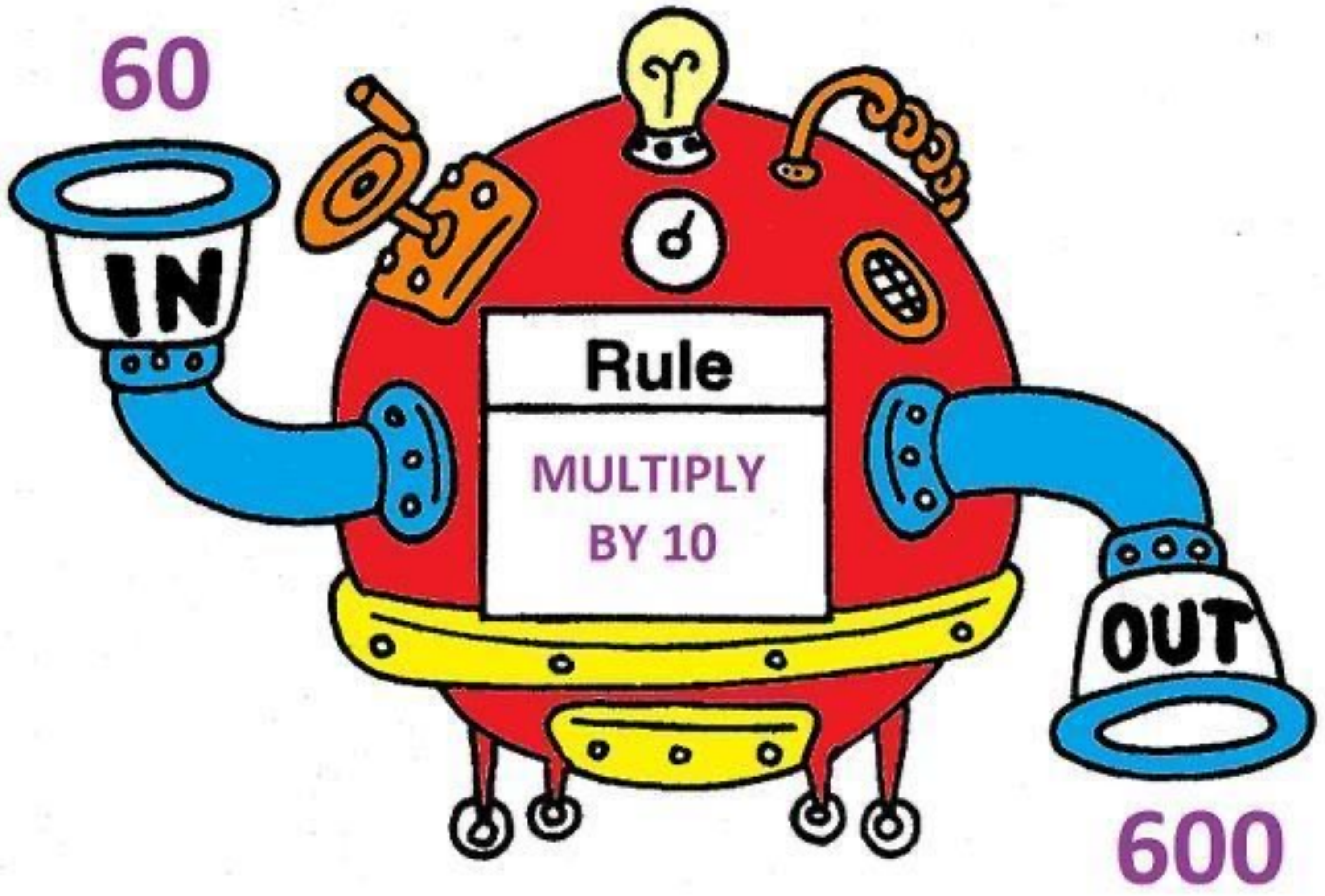
🥇 Session objectives 🥇

- ✓ **Purpose and structure** of functions.
- ✓ Deconstruct the **mechanism of functions**: *inputs, execution, outputs.*
- ✓ Interpret functions using **pseudocode**.
- ✓ Understand functions syntax in C code: **prototypes, definitions, and calls**.
- ✓ Understand the **Top-Down Design**.

Let’s get started with this image!

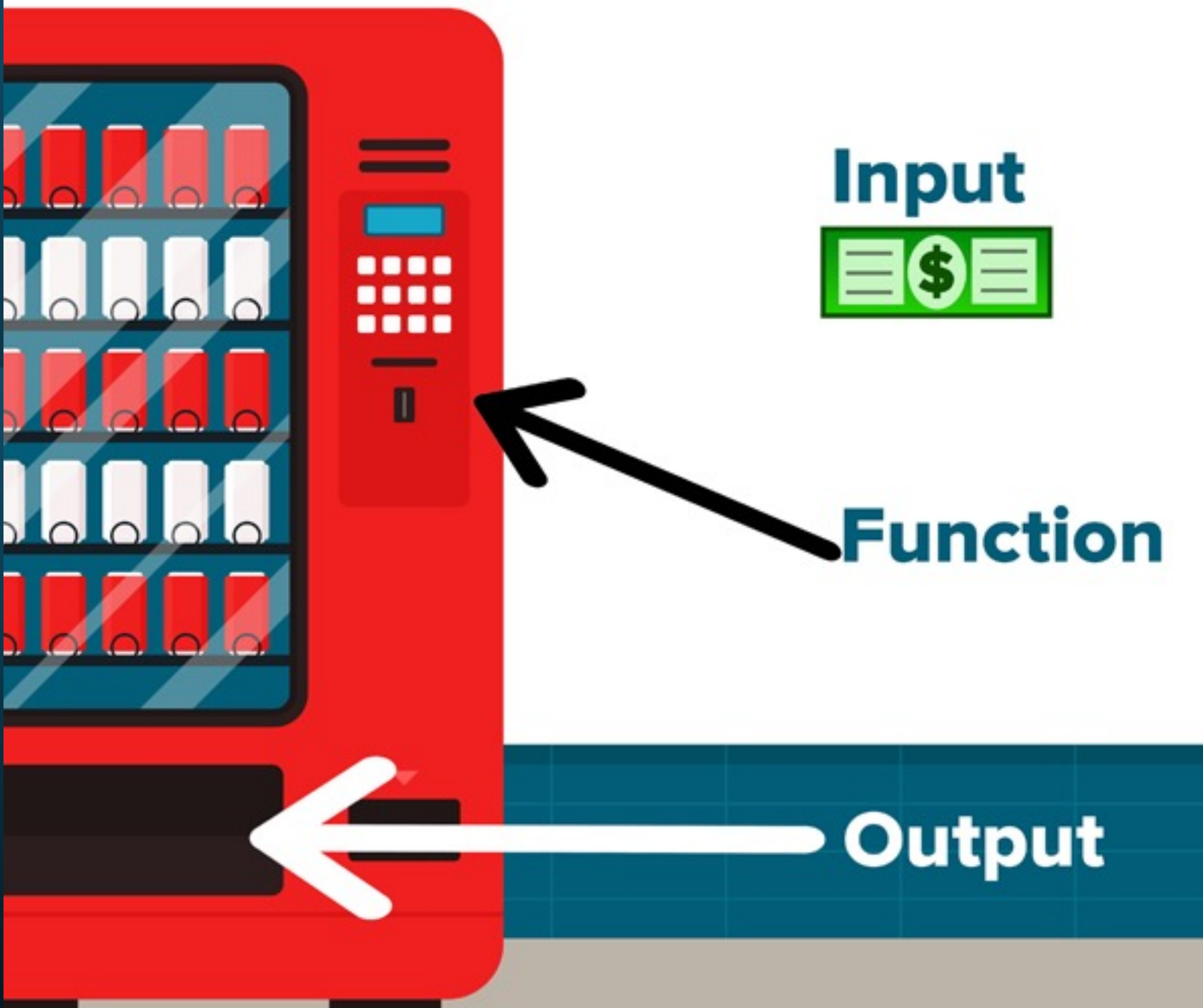


How about this image?



Last image!

What do you think?



EXPLAIN

How to Make **a Banana Smoothie?**

We want to make a single banana smoothie. We would write the bellow pseudo code…

```
Pill a banana
Add 1 cup of milk
Add a spoon of honey
Blend everything
Pour into glass
```

REFLECTION

How to Make **3 Banana Smoothies?**

We want to make 3 banana smoothies, at different places of our code….

```
Pill a banana
Add 1 cup of milk
Add a spoon of honey
Blend everything
Pour into glass
```

```
Pill a banana
Add 1 cup of milk
Add a spoon of honey
Blend everything
Pour into glass
```

```
Pill a banana
Add 1 cup of milk
Add a spoon of honey
Blend everything
Pour into glass
```

Repetitive,
unorganized…
hard to change later !

How can we make that cleaner?

EXPLAIN

Let’s use **Functions**

A function is a block of code which only runs when it is called.

```
makeSmoothie()
makeSmoothie()
makeSmoothie()
```

Function calls

Function definition

makeSmoothie

```
Pill a banana
Add 1 cup of milk
Add a spoon of honey
Blend everything
Pour into glass
```

*We **reduced the repetition** of code by calling a function many times.*

What about strawberry and coconut smoothies?

You can pass data, known as ***parameters****, into a function*

```
makeSmoothie(banana)
makeSmoothie(coconut)
makeSmoothie(strawberry)
```

Function
calls with a **parameter**

Function
parameter

```
makeSmoothie   fruit
Pill the fruit
Add 1 cup of milk
Add a spoon of honey
Blend everything
Pour into glass
```

Parameter
Use in function

We have created a ***modular*** *code by adding parameter to the function.*

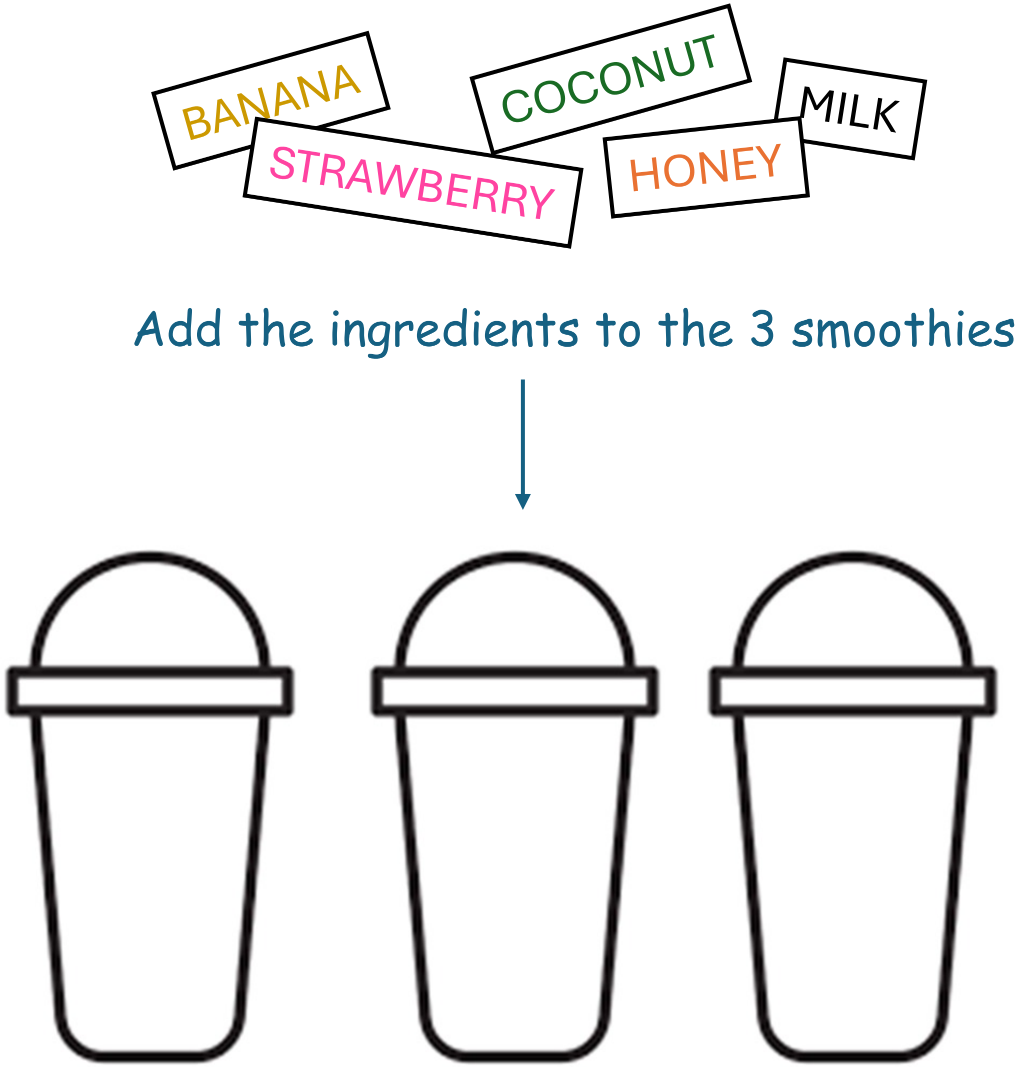
GAME

What will this code produce?

```
makeSmoothie(banana, false)
makeSmoothie(banana, true)
makeSmoothie(strawberry, true)
```

makeSmoothie | fruit | isKhmerStyle

```
Pill the fruit

If (isKhmerStyle)
       Add 1 cup of milk
Else
       Add a spoon of honey

Blend everything
Pour into glass
```



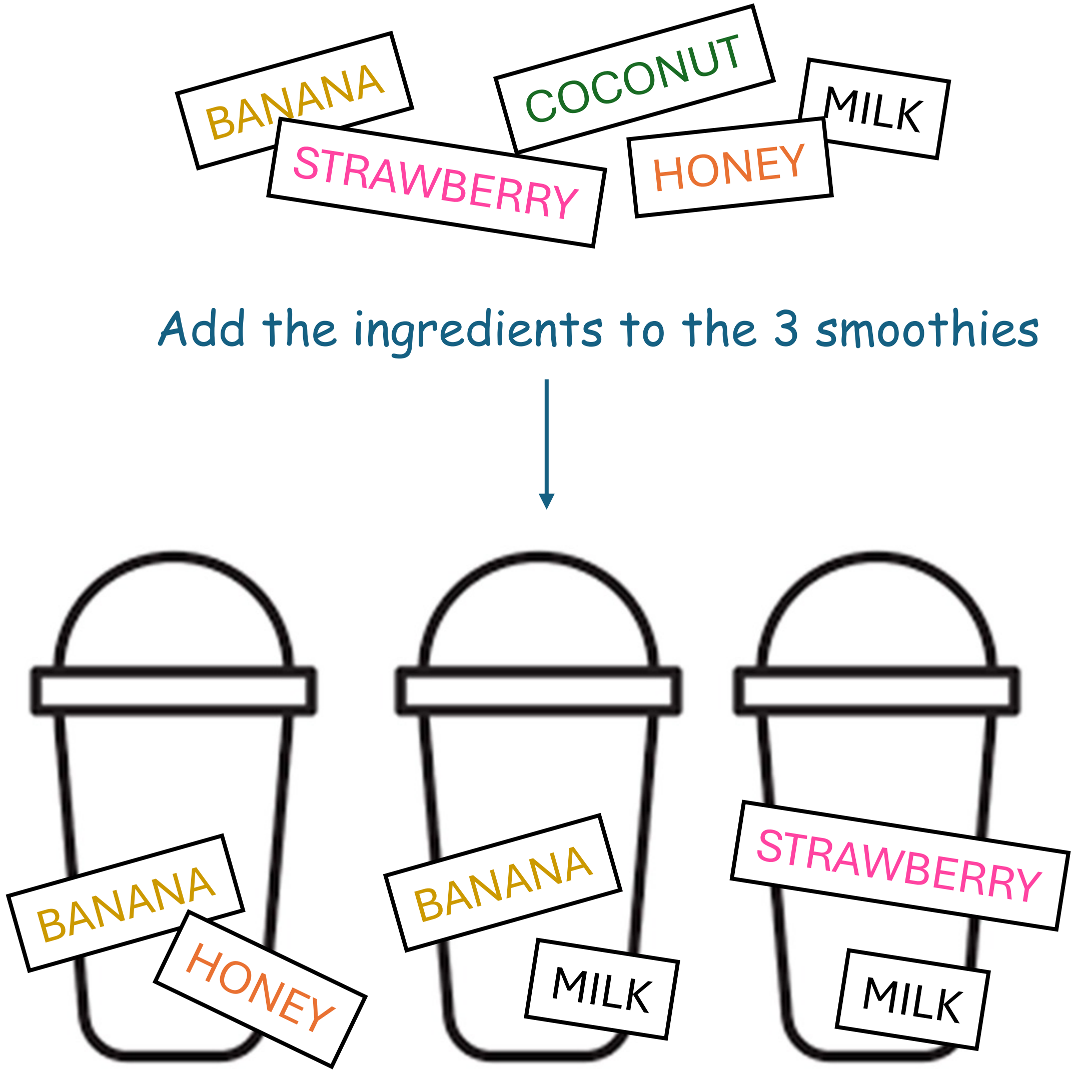
ANSWER

What will this code produce?

```
makeSmoothie(banana, false)
makeSmoothie(banana, true)
makeSmoothie(strawberry, true)
```

makeSmoothie	fruit	isKhmerStyle

```
Pill the fruit

If (isKhmerStyle)
       Add 1 cup of milk
Else
       Add a spoon of honey

Blend everything
Pour into glass
```



EXPLAIN

Let’s **drink** our smoothies!

*A function can **return a value** (the return)*

```
smoothie1  = makeSmoothie(banana)
smoothie2  = makeSmoothie(coconut)

drink(smoothie1)
```

We can **get** the **return** value

We can **use** the **return** value

makeSmoothie | fruit → smoothie

```
Pill the fruit
Add 1 cup of milk
Add a spoon of honey
Blend everything
Pour into glass
return smoothie
```

A Function can **return a value !**

*A function can return **something useful**.*
*That return value can be **saved**, **passed**, or **used** in other functions.*

EXPLAIN

Let’s **deconstruct** a Function

function name, parameters, return, body

A function is defined by a **name** using **camelCase** as naming convention.

A function can have **parameters** (or not)

A function can a **return** (or not)

PSEUDO CODE

```
function add(int a, int b) returns int
    print('we add a and b')
    int c = a + b
    return c
```

A function has list of statements (the function **body**)

EXPLAIN

The function call **flow**

Define the code once, and use it many times.

A program start with a main() function

```
function main()
        int result1 = add(2,8)
        int result2 = add(4,4)
```

```
function add(int a, int b) returns int
        return a + b
```

EXPLAIN

The function call **flow**

Define the code once, and use it many times.

```
function main()
        int result1 = add(2,8)
        int result2 = add(4,4)
```

We call the function add with the arguments 2 and 8.

```
function add(int a, int b) returns int
        return a + b
```

EXPLAIN

The function call **flow**

Define the code once, and use it many times.

```
function main()
      int result1 = add(2,8)
      int result2 = add(4,4)
```

```
                     2            8
function add(int a, int b) returns int
      return a + b
```

The function main is waiting for the end of the function add() execution.

The program is executing the function add() with the parameters 2 and 8.

EXPLAIN

The function call **flow**

Define the code once, and use it many times.

```
function main()
         int result1 = add(2,8)
         int result2 = add(4,4)
```

The function main is waiting for the end of the function add() execution.

```
                          2          8
function add(int a, int b) returns int
         return a + b
                     10
```

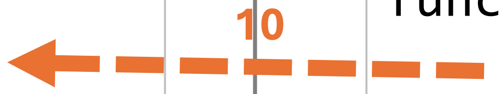
The function add() is **ending** and will **return a result**.

EXPLAIN

The function call **flow**

Define the code once, and use it many times.

```
function main()
        int result1 = add(2,8)
        int result2 = add(4,4)
```



```
function add(int a, int b) returns int
        return a + b
```

1. Add(2, 8) returns 10
2. The number 10 is assigned to the variable result1
3. We go to the next statement

EXPLAIN

The function call **flow**

Define the code once, and use it many times.

```
function main()
        int result1 = add(2,8)
        int result2 = add(4,4)
```

We call the function add with the arguments 4 and 4.

```
function add(int a, int b) returns int
        return a + b
```

EXPLAIN

The function call **flow**

Define the code once, and use it many times.

```
function main()
        int result1 = add(2,8)
        int result2 = add(4,4)
```

The function main is waiting for the end of the function add() execution.

4 4

```
function add(int a, int b) returns int
        return a + b
```

The program is executing the function add() with the parameters 4 and 4.

EXPLAIN

The function call **flow**

Define the code once, and use it many times.

```
function main()
        int result1 = add(2,8)
        int result2 = add(4,4)
```

The function main is waiting for the end of the function add() execution.

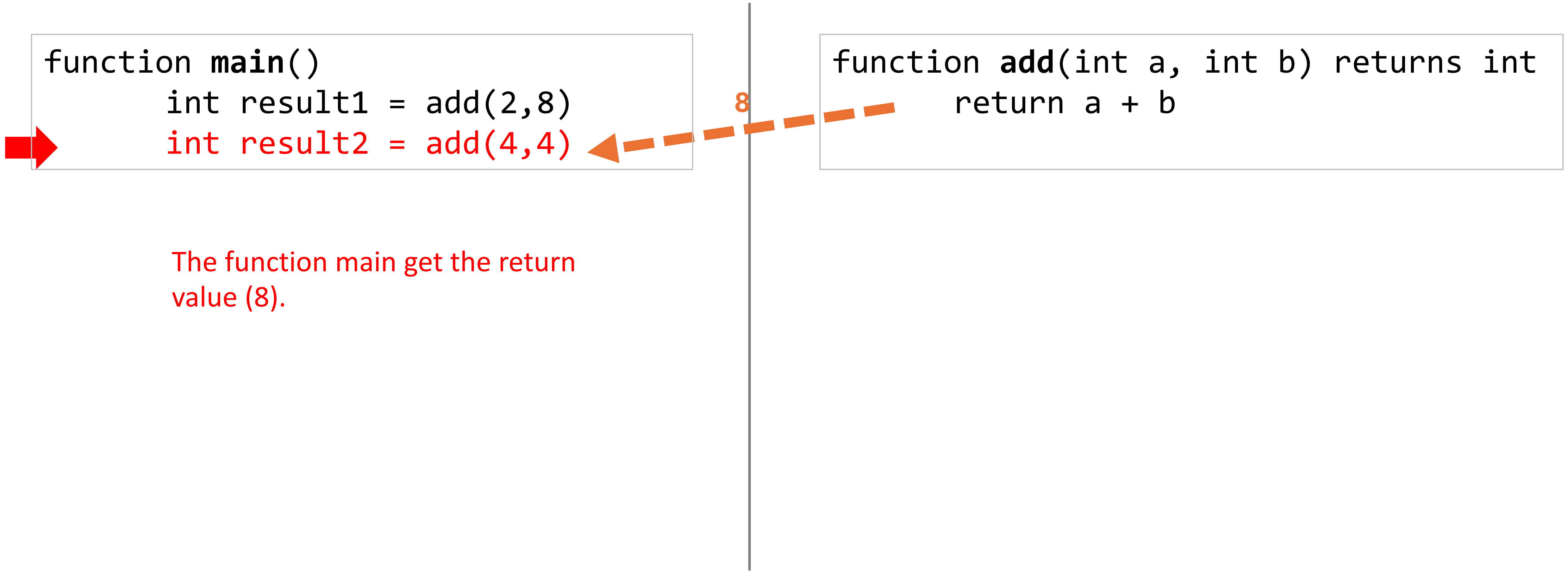
```
                          4           4
function add(int a, int b) returns int
        return a + b
                   8
```

The function add() is **ending** and will **return a result**.

EXPLAIN

The function call **flow**

Define the code once, and use it many times.



```
function main()
        int result1 = add(2,8)
        int result2 = add(4,4)
```

```
function add(int a, int b) returns int
        return a + b
```

The function main get the return value (8).

QUIZ

Q1

What’s **wrong** with this function?

PSEUDO CODE

```
function add(a, b) returns int
    print(a + b)
```

A) The function is missing parameters.

B) The print statement should come after the return.

C) The function says it returns a value, but it doesn’t actually return anything.

D) The parameters are not integers.

Q1 ANSWER

What’s **wrong** with this function?

PSEUDO CODE

```
function add(a, b) returns int
    print(a + b)
```

A) The function is missing parameters.

B) The print statement should come after the return.

[✓ C)] The function says it returns a value, but it doesn’t actually return anything.

D) The parameters are not integers.

QUIZ

Q2 What is the return type of this function?

PSEUDO CODE

```
function isEven(n) returns ??
    return n % 2 == 0
```

A) Int

B) Boolean

C) No return

D) char

Q2 ANSWER

What is the return type of this function?

PSEUDO CODE

```
function isEven(n) returns bool
    return n % 2 == 0
```

A) Int

[✓ B)] Boolean

C) No return

D) char

QUIZ

Q3

Fill in the Blank

PSEUDO CODE

```
function multiply(a, b) returns int
    ____________


result = multiply(3, 4)
print(result)
```

a) return a * b

b) a * b

c) print(a * b)

d) int c = a*b

return c

Q3 ANSWER

Fill in the Blank

PSEUDO CODE

```
function multiply(a, b) returns int
    __________


result = multiply(3, 4)
print(result)
```

[✓ a)] return a * b

b) a * b

c) print(a * b)

[✓ d)] int c = a*b

return c

QUIZ

Q4

What will this code print?

PSEUDO CODE

```
function cube(x) returns int
    return x * x * x



print(cube(2) + cube(3) )
```

A) 18

B) 27

C) 35

D) 125

Q4 ANSWER

What will this code print?

PSEUDO CODE

```
function cube(x) returns int
    return x * x * x



print(cube(2) + cube(3) )
```

A) 18

B) 27

[✓ C) 35]

D) 125

EXPLAIN

Functions in C Language

How to write and call a function in C ?

A function is **defined** by a **name (*)**

A function can have **parameters** (or not)

A function can a **return** (or not)

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

int main() {
    int result = add(4,2);

    return 0;
}
```

A function is **called** by its name

(*) We use CAMEL case to write C language function names !

EXPLAIN

Functions in C language

A function can return nothing (void)

This function returns nothing

```
void printNumber(int number) {
    printf("Your number is %d \n", number);
}


int main() {
    printNumber(5);
    printNumber(10);

    return 0;
}
```

```
Your number is 5
Your number is 10
```

QUIZ

Predict the **Output**

C CODE

```c
int doubleIt(int x) {
    return x * 2;
}

int main() {
    int result = doubleIt( doubleIt(3) );
    printf("%d\n", result);

    return 0;
}
```

a) 6

b) 9

c) 12

d) 18

ANSWER

Predict the **Output**

```
int doubleIt(int x) {
    return x * 2;
}                                   12
                          ┌─────────────────────────┐
                                              6
                                      ┌─────────────┐
int main() {
    int result = doubleIt( doubleIt(3) );
    printf("%d\n", result);

    return 0;
}
```

a) 6

b) 9

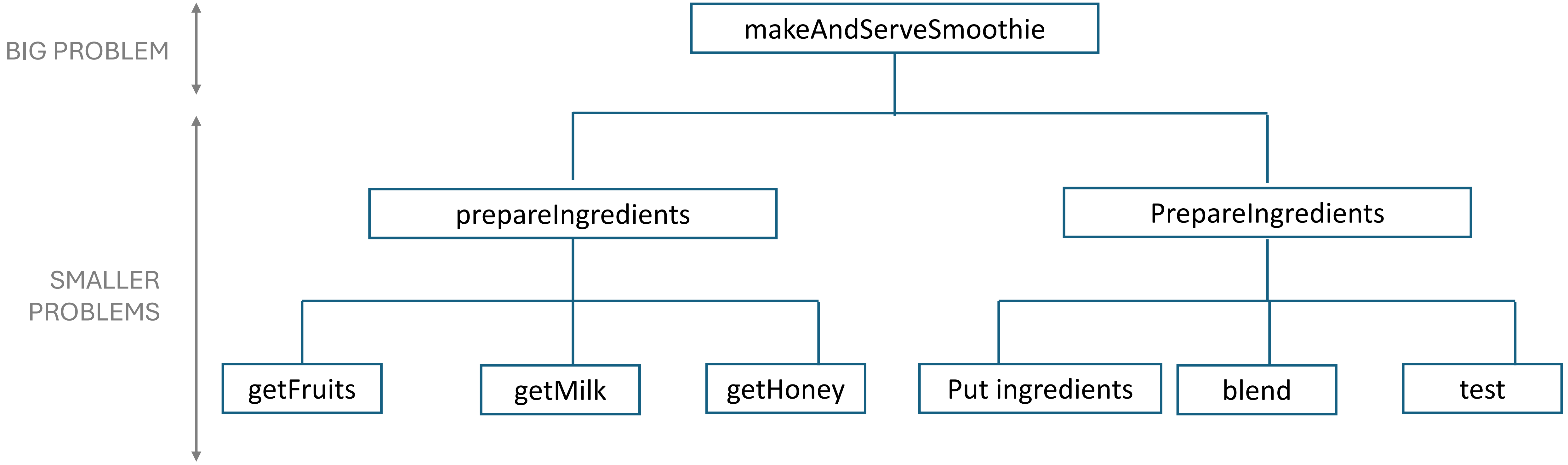
[✓ c) 12]

d) 18

EXPLAIN

Top-Down Design

Break big problems into smaller problems — and solve each small part.



EXPLAIN

Top-Down Design & **Functions**

Break a big problem....

```
function playGame()
    secret = generateSecretNumber()
    guess = getPlayerGuess()
    checkGuess(secret, guess)

function generateSecretNumber() returns int
        (some code)

function getPlayerGuess() returns int
        (some code)

function checkGuess(int secret, int guess)
        (some code)
```

... into smaller problems

CHALLENGE

Your turn !

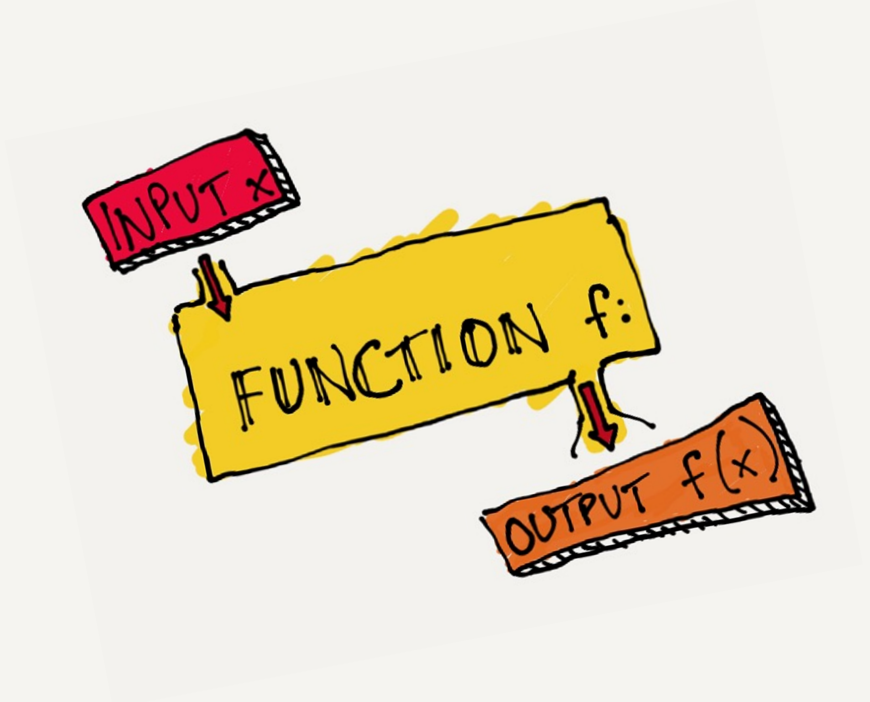
Think of another big problem to solve and break it down using **top-down design** with **function**

```
function xxx()
          (some code)


function xxx() returns xxx
          (some code)


function xxx() returns xxx
          (some code)


function xxx(int xxx, int xxx)
          (some code)
```

Then we will share works in group or to the whole class

What you **know now**

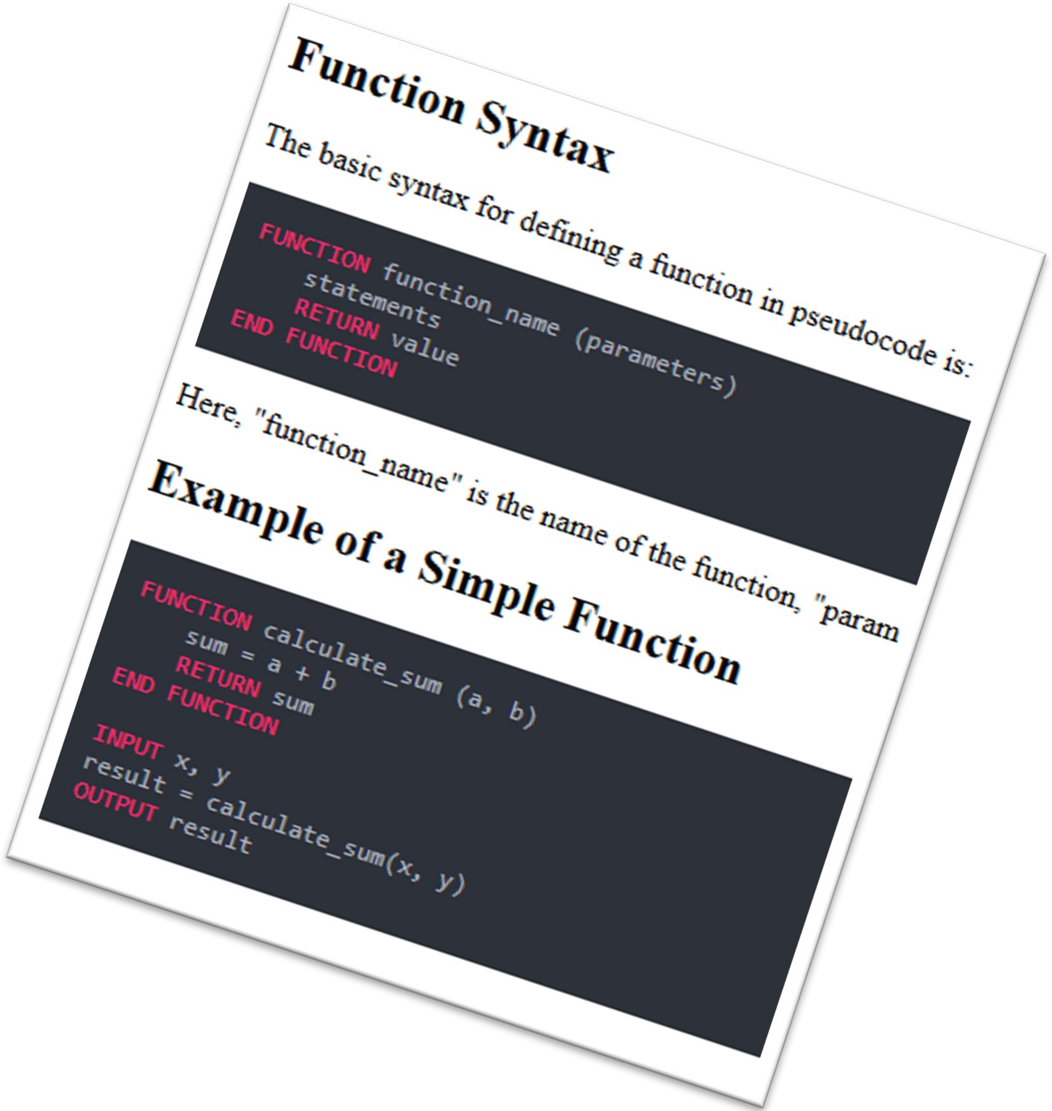
- ✓ **Purpose and structure** of functions.
- ✓ Deconstruct the **mechanism of functions**: *inputs, execution, outputs.*
- ✓ Interpret functions using **pseudocode**.
- ✓ Understand functions syntax in C code: **prototypes, definitions, and calls**.
- ✓ Understand the **Top-Down Design**.



FOR NEXT TIME

READ THE THEORY ABOUT FUNCTIONS IN PSEUDO CODE

https://pseudocode.deepjain.com/guides/functions/



Function Syntax

The basic syntax for defining a function in pseudocode is:

```
FUNCTION function_name (parameters)
    statements
    RETURN value
END FUNCTION
```

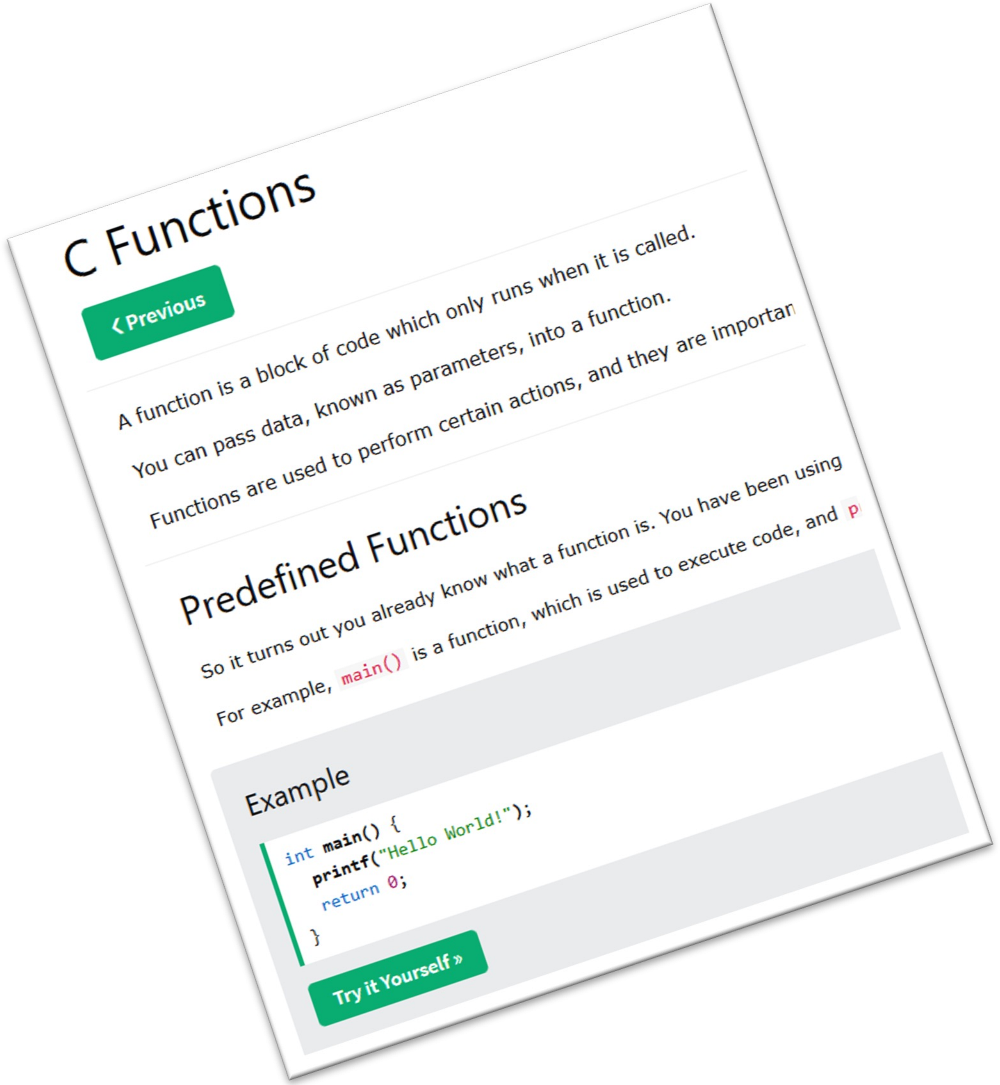
Here, "function_name" is the name of the function, "param

Example of a Simple Function

```
FUNCTION calculate_sum (a, b)
    sum = a + b
    RETURN sum
END FUNCTION
INPUT x, y
result = calculate_sum(x, y)
OUTPUT result
```

READ THE THEORY ABOUT FUNCTIONS IN C

https://www.w3schools.com/c/c_functions.php



C Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

Functions are used to perform certain actions, and they are importan

Predefined Functions

So it turns out you already know what a function is. You have been using

For example, `main()` is a function, which is used to execute code, and p

Example

```
int main() {
  printf("Hello World!");
  return 0;
}
```