Functional Programming - OCaml

By: Sanchit Gupta

More *strict, accurate, efficient* and *secure* programming!

Gujarat Forensic Sciences University
Digital Forensics and Information Security

Things we will be looking at in this presentation.

1. Brief Introduction
2. Programming using OCaml

Brief Introduction

1

1.1 **What is Functional Programming**

1.2 **Why Functional Programming**

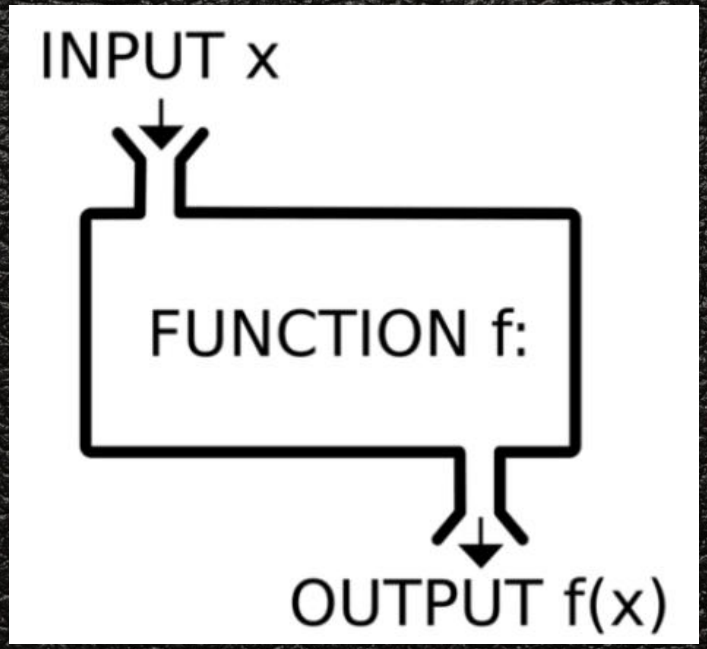
1.3 **Is Functional Programming easy?**

1.4 **Scope**

1.1 What is Functional Programming?

" *Functional* "

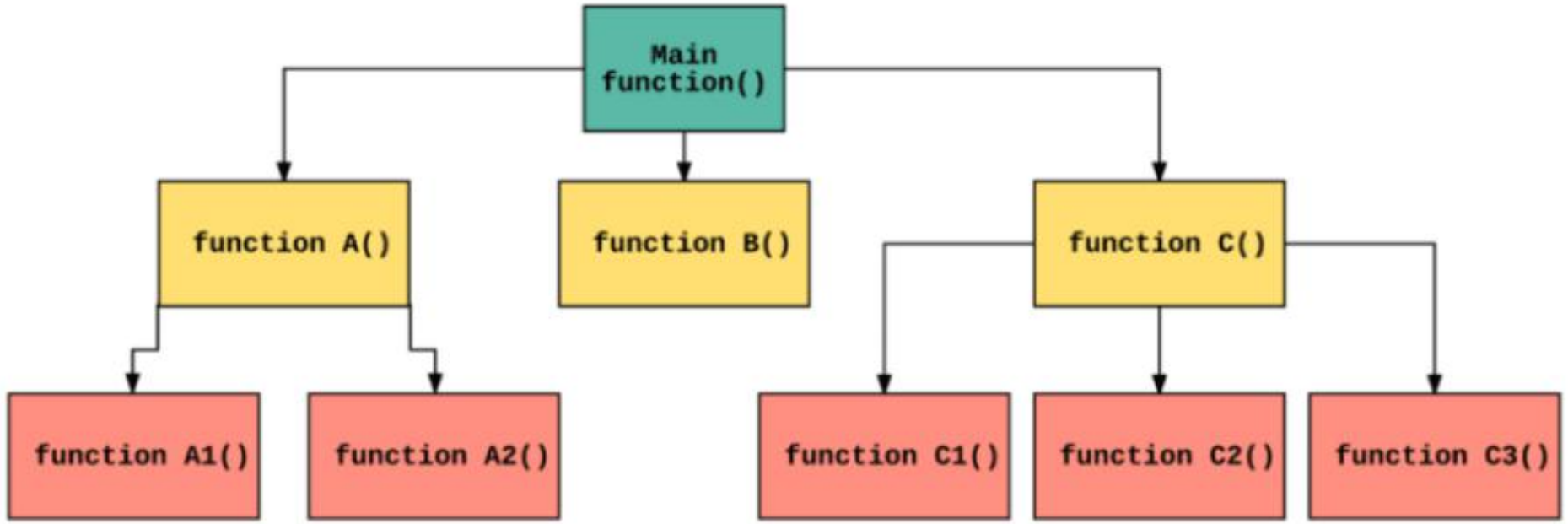
This kind of programming is called *"Functional Programming"* because the programs consist of entirely of functions. Even the main program is the function.



" *Functions in Function*

The key concept of Functional Programming is Modularity. The program functions are always defined in terms of other functions which are further defined in terms of smaller functions.

1.1 What is Functional Programming?



Functional Flowchart

1.2 **Why Functional Programming**

- Small modules can be coded quickly and easily.
- Can be used to code real exact mathmetical functions in otder to match definition.
- Function signatures are more meaningfull.
- General purpose modules can be reusable, which leads to faster development of next programs.
- Debugging is easy
- Strict Syntax.
- Interdependent function: Need to understand whole heirarchy of functions wirtten above a perticular function

1.3 **Is Functional Programming easy?**

Yes (Kinda)! Provided you have ***not*** done programming at all!

No! For the programmer who ***have been involved in OOPs***, it become intesively hard for them to think in functional perspective.

We will see the programming stuff soon!! Again my effort would be to make it easy for you guys to understand and get an essence of it.

1.4 **Scope**

1

Well... It is being used by Facebook, Google, Youtube, WhatsApp and other more big organisations...so you can imagine.

2

Like Python, It also has built-in libraries for many things ... for example, clinet-server communication, networking, routing ... etc.

3

It doesn't stop with organisations...many banks such as Barclays, HSBC, HDFC ...they all use FP.

4

It provides secure environment and secure commmunication eastablishment.

Programming using OCaml

2

2.1 **Simple Functions**

2.2 **Pattern Matching - if**

2.3 **Getting deep with Lists**

2.4 **Recursion and more**

We will not be looking at Data Structure & Algorithms (Graphs, Linked Lists, BSTs) and we will also not be discussing I/O.

2.1 Simple Functions

```
val sum : int -> int -> int = <fun>
let sum x y = x + y

val double : int -> int = <fun>
let double x = sum x x

val triple : int -> int = <fun>
let triple x = ...??.... (You find out)

val myFunct : ('a -> 'b) -> 'a -> b' =
<fun>
let myFunct = fun f -> fun x -> f x
# myFunct double 5

let addFloats x y = x +. y
```

2.2 Pattern Matching -using if

```
type weekday = Mon | Tue |  Wed |
Thur | Fri | Sat | Sun;;

val how_i_feel : weekday -> string =
<fun>
let how_i_feel day =
  match day with
| Mon -> "sleepy"
| Tue -> "grumpy"
| Wed -> "sneezy"
| Thur -> "dopsy"
| Fri -> "happy"
| Sat -> "bashful"
| Sun -> "doc"

# how_i_feel Tue;;
- : string = "grumpy"
```

Q) Make program for *Boolean AND Operation*?

Hints:

1) declare a *type like:* `type mbool = .... | ....`

2) `let mand b1 b2 = ......`

3) `match (b1, b2) with ....`

4) You can user _ as a "wildcard"

2.1Add your title

```
type mbool = True | False
```

```
let mand b1 b2 =
match (b1, b2) with
| True, True -> True
| _, _ -> False

val mand : mbool -> mbool
-> mbool = <fun>
```

2.3 & 2.4 Getting Deep with Lists and Recursion

```
Tuple: (1, 3, 7, 8, 2, 3, 2, 9);;

List: [1; 3; 7; 8; 2; 3; 2; 9];;

List within List: [[1; 3; 5]; [2; 4; 6]];;

Tuple within List: [(1, 'a'); (2, 'b')];;

Array: [|4; 7; 8; 9|];;
a = [|4; 7; 8; 9|];;
a.(0);; would give 4
a.(3);; would give ..?..

let somefunction ls = match ls with
| [] -> true
| hd :: tl -> false
```

Add every second element:

```
let rec printsec ls = match ls
with
| [] -> 0
| [hd, tl] -> hd
| hd :: tl :: tll -> hd +
(printsec tll)
```

Find the last element of the list:

```
let rec findLast ls = match ls
with
| [] -> None
| [x] -> (Some x)
| x :: xs -> findLast xs;;
```

Q) Find 'k'the element in list?

THANKS!