# Programmazione Funzionale

Esercitazione 1

**Esercizio 1.** Dopo aver imesso le seguenti dichiarazioni, qual'è il valore de l'espressione `f 3`?

```
let a = 0;;
let f = function x -> x + a ;;
let a = "zero";;
```

Dopo aver imesso le seguenti dichiarazioni, qual'è il valore de l'espressione `f 5`?

```
let a = 0;;
let f = function x -> x + a ;;
let a = 4;;
```

Dopo aver imesso le seguenti dichiarazioni, qual'è il valore de l'espressione `f 3`?

```
let a = 0;;
let f = function x -> x + a ;;
let x = 3;;
```

Dopo aver imesso le seguenti dichiarazioni, qual'è il valore de l'espressione `f 3`?

```
let g = function x -> 3*x;;
let f = function x -> g(x+2) ;;
let g = function x -> 0;;
```

**Esercizio 2.** Indicare qual'è delle seguenti espressioni e un espressione riducibile o un valore (i.e espressione non riducibile):

- `3`
- `3+2`
- `(3,3,"frase")`
- `(2,4+2,0)`
- `function x -> 3*x`
- `(function x -> 3*x) 2`
- `(0,true, (function x -> 0) 2)`
- `if (true) then (false) else (true)`

**Esercizio 3.** Determinare il tipo delle seguenti espressioni:

- `422`
- `"this"`
- `true`
- `(function true -> true | false -> false)`
- `(43,"siu",3.)`
- `(function x -> 0) 3`
- `(function x -> x+1)`
- `(function (x,y) -> x)`
- `function f - > (function x -> f(2*x))`
- `function f - > (function (x,y) -> (y,f(2*x)))`

**Esercizio 4.** Scrivere in Ocaml le seguenti funzioni e indicare il loro tipo:

- $f : \mathbb{N}^2 \times \mathbb{N}^2 \to \mathbb{N}^2, (x, y), (x', y') \mapsto (x + x', y + y')$.
- la funzione che prende una tupla $(x, y, z, t)$ e ritorna $x$.
- La funzione che prende una tupla $(x, y, z, t)$ e ritorna la coppia $(t, y)$.
- La funzione che prende una funzione $f$ e la applica a 0 cioè ritorna $f(0)$.
- La funzione che prende in input una funzione $f$ e ritorna una funzione $g : x \mapsto f(x + 1) + 1$.

**Esercizio 5.** Dare la forma curryficata delle seguenti funzioni e indicare il suo tipo:

- `function (f,n) -> (function x -> f(x)+n)`
- `function (f,n) -> true`

- `function (f,g) -> (function x -> f(x) + g(x))`
- `function (f,g) -> (function x -> f(g x ))`

**Esercizio 6.** Dare la forma decurryficata delle seguenti funzioni e indicare il suo tipo:
- `function f -> (function g -> (function x -> f g x))`
- `function f -> (function g -> (function x -> (f x,g x)))`

**Esercizio 7.** Dare una funzione di tipo `int` $\times$ `int` $\to$ `float` che prende due interi e ritorna la loro divisione in $\mathbb{R}$.

**Esercizio 8.** Dare una funzione di tipo `int` $\times$ `string` $\to$ `bool` che ritorna true per una coppia $(n, s)$ se e solo se $s$ rappresenta un intero $s_r$ e $n = s_r$. La funzione non deve mai sollevare eccezioni.

**Esercizio 9.** Sia due tipi $\alpha$ e $\beta$. Definire la funzione swap : $\alpha \times \beta \to \beta \times \alpha$ che prende una coppia $(a, b)$ e ritorna $(b, a)$

Dato due tipi $A$ e $B$. Il *prodotto cartesiano* di due funzioni $f_1 : A \to A$ e $f_2 : B \to B$ e la funzione $(f_1, f_2) : A \times B \to A \times B, (a, b) \mapsto (f_1(a), f_2(b))$.

Mostrare che ogni prodotto cartesiano di due funzioni $f_1 : A \to A$ e $f_2 : B \to B$ puo essere scritto come la composizione della funzione `swap` e di funzioni che corrispondono a l'identita sulla seconda componente i.e. funzioni della forma seguente (dove `E` e un espressione qualsiasi):

```
let f = function
    (a,b) -> ((E) a , b);;
```

**Esercizio 10.** Definire le due funzione polimorfe $proj_1 : \alpha \times \beta \to \alpha, (a, b) \mapsto a$ e $proj_2 : \alpha \times \beta \to \beta, (a, b) \mapsto b$ in OCAML.

Fissando due tipi $A$ e $B$. Mostrare che, ogni funzione $f : A \times B \to A \times B$ puo essere scritto il prodotto cartesiano $(proj_1 f, proj_2 f)$.

**Esercizio 11.** Definire la somma `somma` dei tipi `int` e `bool`, dare una funzione di tipo `somma` $\to$ `bool` che ritorna `true` se l'input e un intero e `false` se l'input e un booleano.

**Esercizio 12.** Sia il tipo `intsum` la somma del tipo `int` con `int`, con construttori respettivi $I$ e $J$. Cioe consideriamo di aver dichiarato:

```
type intsum = I of int | J of Int;;
```

Ricordandosi che una funzione `intsum` $\to'$ a sara della forma seguente (dove `E` e `E'` sono espressioni qualsiasi ):

```
let Right_E = function
    | I u -> (E) u
    | J u -> (E') u;;
```

Argomentare perche qualsiasi funzione di tipo `intsum` $\to'$ a puo essere definita come una composizione di funzioni della forma seguente (dove `E` e un espressione qualsiasi):

```
let Right_E = function
    | I u -> I (E) u
    | J u -> J u;;


let Left_E = function
    | I u -> I u
    | J u -> J (E) u;;


let Project = function
    | I u -> u
    | J u -> u;;
```