




FP TOUR

The text "FP TOUR" is centered in a black, sans-serif font. It is flanked by two large, thick black L-shaped brackets. The bracket on the left is positioned in the upper-left area, with its vertical leg extending downwards and its horizontal leg extending to the right. The bracket on the right is positioned in the lower-right area, with its vertical leg extending upwards and its horizontal leg extending to the left. The background is a solid light beige color.



CONCETTI CHIAVE

- “Assenza” di side effects
 - Funzioni
 - *Totalità*
 - *Higher order functions*
 - Immutabilità
 - Referential transparency
- 

Side effect

- Cos'è?
 - *Qualsiasi interazione con il “mondo esterno”, non solo Db, Api, ma anche STDin STDOUT*
- A che serve un programma che non ha side effects?
 - *A nulla*
- Nella programmazione funzionale i side effects vengono eseguiti alla “fine del mondo”
- Si può pensare ad un programma funzionale come ad una funzione enorme che descrive una computazione senza side effects e che verrà eseguita alla “fine del mondo”.

Funzioni

■ Totalità

- *Una funzione è totale se ritorna un valore per ogni possibile input, anche in caso di errore*
- *In OOP siamo abituati a gestire gli errori con le eccezioni, in FP i tipi ci aiutano a wrappare le eccezioni (e.g. Either, Try)*

■ HOF

- *Funzioni che prendono come argomenti o ritornano altre funzioni*

■ Scrivere funzioni in questo modo porta alla composizione:

- *Passare il risultato di una funzione come input della successiva*

Immutabilità

- Un dato immutabile non può essere modificato dopo la sua creazione
- Se vogliamo “modificare” un oggetto(case class) possiamo crearne un'altra istanza modificata.
 - *È più oneroso in termini di memoria (ma la “vecchia” istanza prima o poi verrà pulita dal GC)*
 - *È estremamente threadsafe, si può condividere un oggetto tra diversi thread senza rischi*

Referential transparency

- Una funzione è *referential transparent* quando può essere sostituita con il suo valore di ritorno e viceversa senza modificare il risultato del programma
- Esempi:
 - *la funzione multiply è referential trasparente?*
 - *e la funzione giveMoneyTo?*
- Benefici: scrivendo funzioni RT sei libero di sostituire funzioni con i loro valori! Nel refactoring è molto importante perchè siamo in grado di sostituire funzioni con valori e viceversa senza doverci aspettare comportamenti indesiderati!



TYPES



- Functor
- Monad

Functor

- Problema: in programmazione funzionale abbiamo a che fare con valori Immutabili. Occorrono modi per accedere a quei valori e creare altre strutture dati con i valori modificati.
 - *Esempio List*
- Soluzione -> Functor
 - *Il funtore è il tipo che rappresenta le strutture dati “mappabili”. In Scala lo scriviamo come una TypeClass perchè vogliamo estendere il concetto di “mappabilità” ad alcune strutture dati ma non a tutte!*