Lec 5 : ILFP

Agenda :
- Lists
- Sorting & Searching in functional style

Recap

- Evaluation strategies [CBN, CBV]
- Lexical scoping, binding rules
  - let clause
  - where clause

- Discussion of quiz in the lab session.

- Eg: lexical scoping

Today : We will start with tuples and lists
- Some simple data structures which are already supported in Haskell

Tuples: Heterogeneous containers

- pairs of int, (3, 5)
  Eg: Coord., pos-velocity etc.
- mixed pairs, (name, ph-no)

Predefined fncs

- fst (5, "Hello") = 5
- snd (5, "Hello") = "Hello"

- fst ((1, 'a'), "Hello")
  = (1, 'a')

Note: fst, snd won't work any longer tuple other than pair.

Lists

- Tuples, by defⁿ, can hold fixed number of elements
  - pairs hold two, triples hold three, ...
- Lists can hold arbitrary number of Elements
- But lists are homogeneous data structures
  i.e. [5, "Hello"] is disallowed!

Lists are assembled using [ ], instead of ( ) in tuples!

Eg: let a = [1, 2, 3] , [] → Empty list

- Colon operator

  → Cons operator [Constructor]

  Eg: Element : lst

  → Element appends to the head of the list.

  Eg: a = 100 : [] → [100]

  a = 1 : [2, 3, 4] → [1, 2, 3, 4]

- Lists are implemented as linked lists in Haskell.
  - ⟶ Very different performance characterstics than arrays!

- Predefined functions

```
length :: Foldable t ⇒ t a → INT
          [a] ⟶ INT
head   :: [a] ⟶ a
```

Recursively

```
Let a = h :: tl in

length a = if a = []
           then 0
           else 1 + length tl
```

Pattern Matching

- we attempt to match values against patterns
- and possibly bind variables to successful matches

From imperative languages pt. of view this may look similar to switch statements

- pattern-matching can be used not only for primitive type values but also for composed types (such as tuples etc)