Lists, Pattern Matching and Recursion

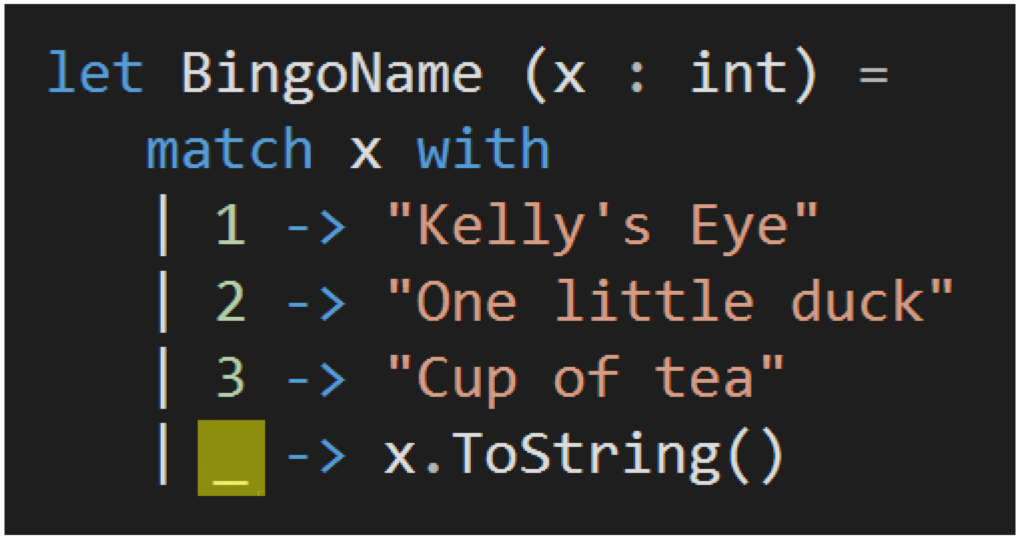
Kit Eason
www.kiteason.com
@kitlovesfsharp

pluralsight
hardcore dev and IT training

Pattern Matching

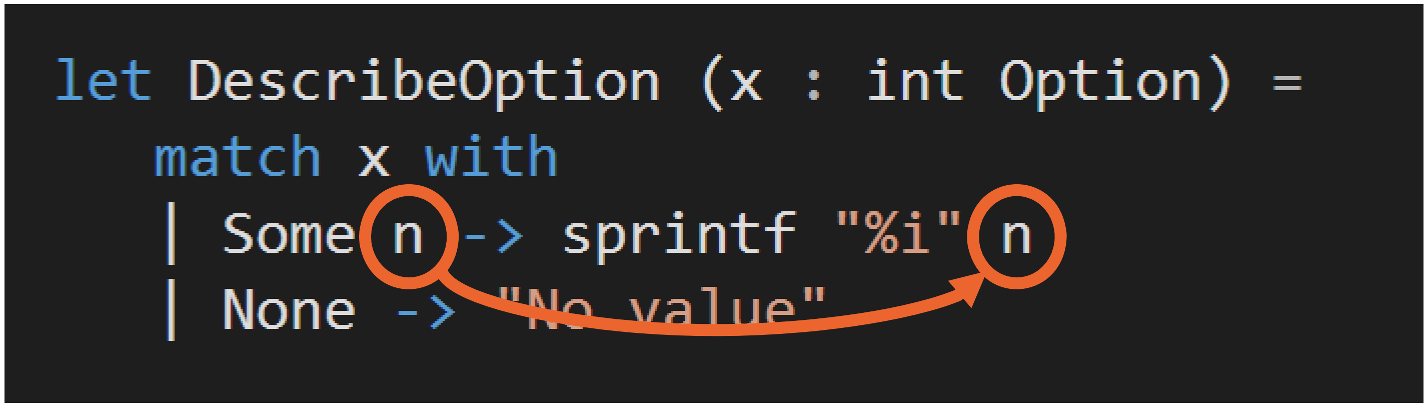
- Nothing to do with regex!
- Branch logic, assign values by ‘shape’ of data
- Backbone of F#’s ‘match’ construct

Match Statement as Switch Statement



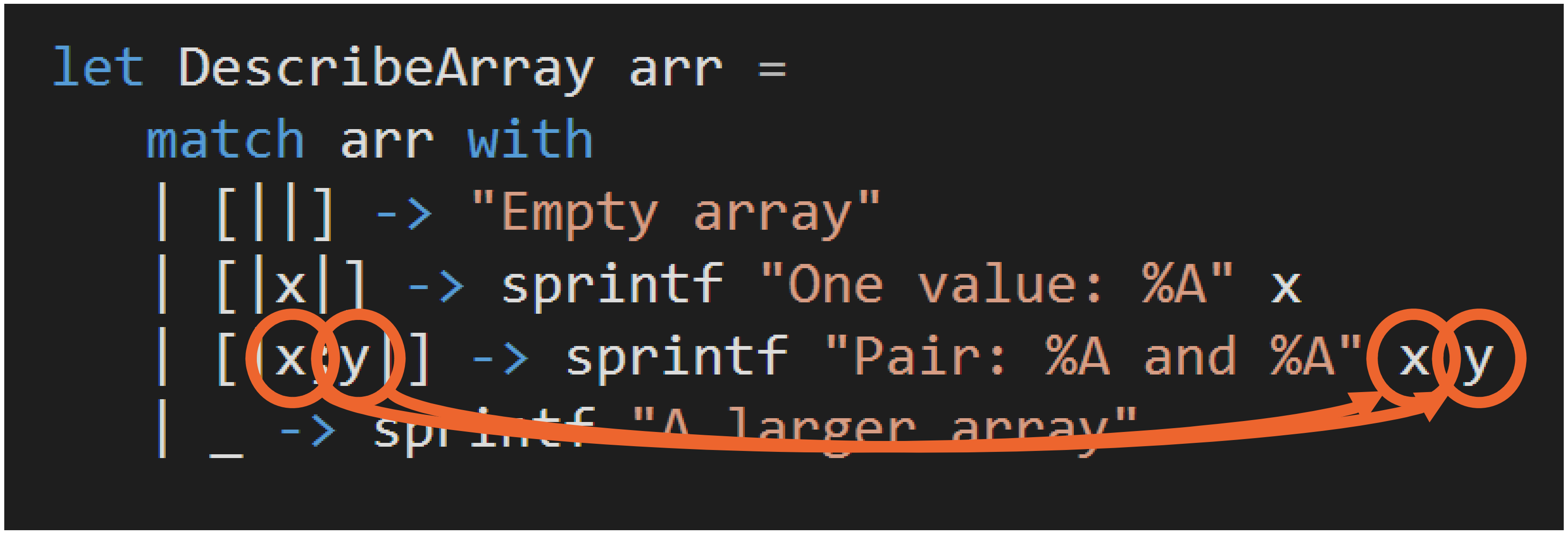
```
let BingoName (x : int) =
    match x with
    | 1 -> "Kelly's Eye"
    | 2 -> "One little duck"
    | 3 -> "Cup of tea"
    | _ -> x.ToString()
```

Match With ‘Identifier Pattern’



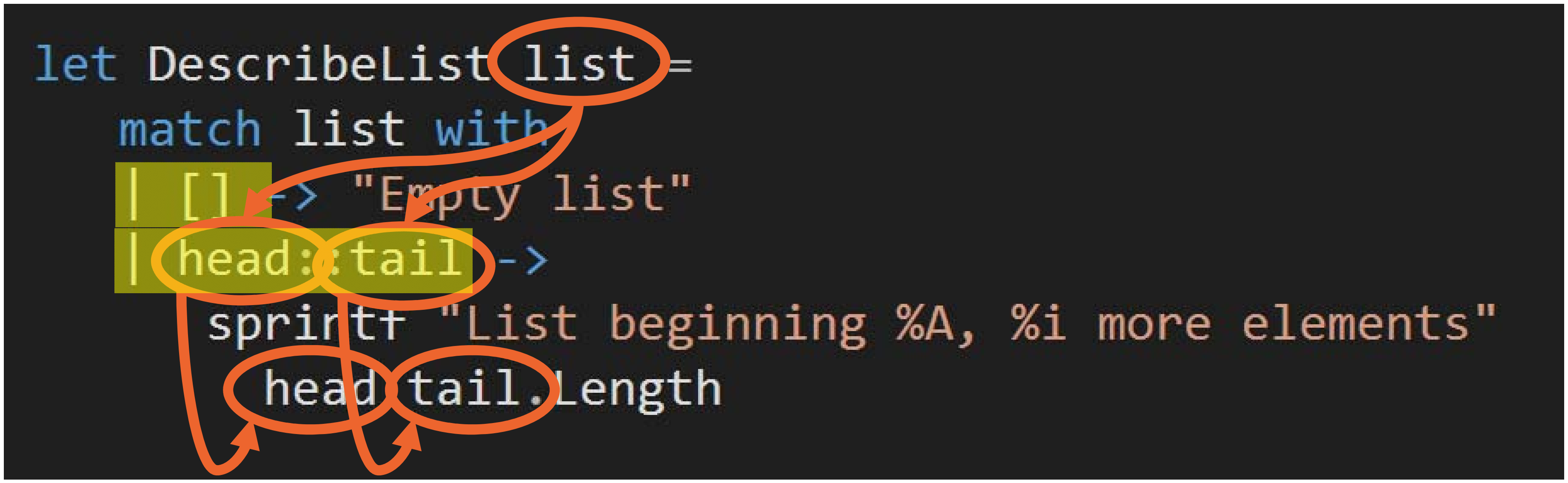
```
let DescribeOption (x : int Option) =
    match x with
    | Some n -> sprintf "%i" n
    | None -> "No value"
```

Match With ‘Array Pattern’



```
let DescribeArray arr =
    match arr with
    | [||] -> "Empty array"
    | [|x|] -> sprintf "One value: %A" x
    | [|x;y|] -> sprintf "Pair: %A and %A" x y
    | _ -> sprintf "A larger array"
```

Match With Cons Pattern

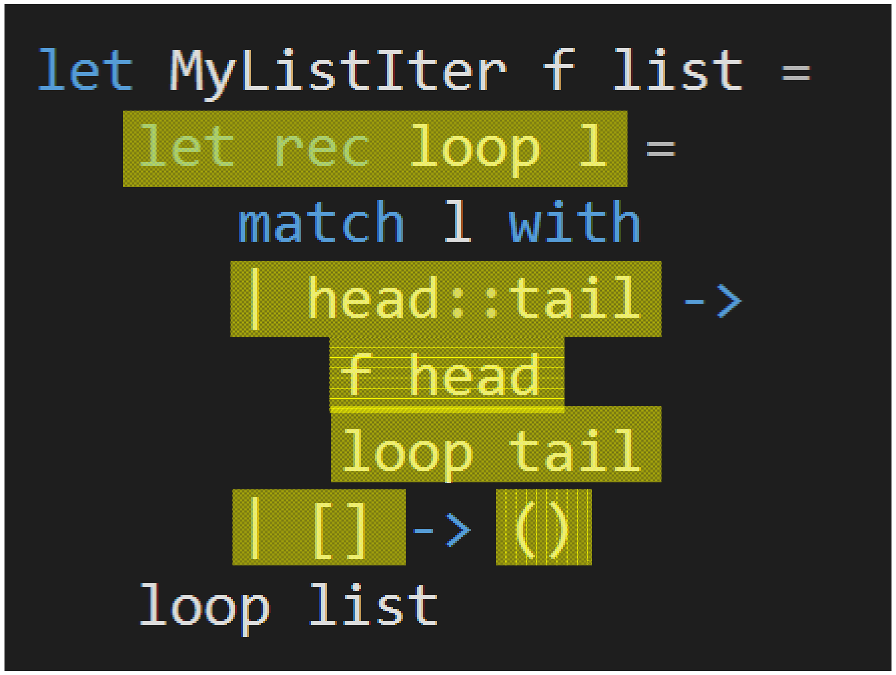


```
let DescribeList list =
    match list with
    | [] -> "Empty list"
    | head::tail ->
       sprintf "List beginning %A, %i more elements"
          head tail.Length
```

Recursive Traversal

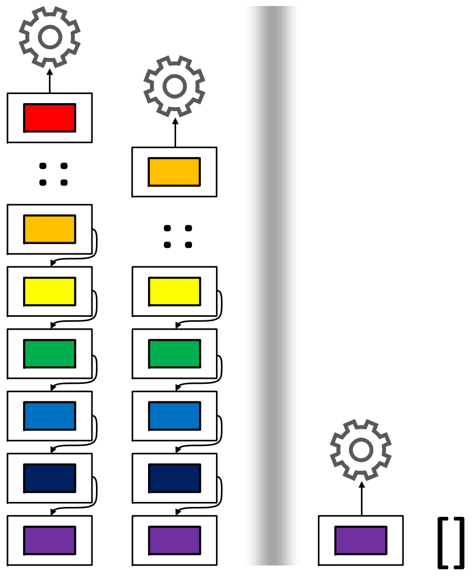
- To traverse a list use List.map, List.iter or
- …recursion!
- First iteration splits into head and tail
- Uses the head
- Passes the tail to the next iteration…
- …and so on
- Until the tail is empty

Re-Implementing List.iter



```
let MyListIter f list =
    let rec loop l =
        match l with
        | head::tail ->
            f head
            loop tail
        | [] -> ()
    loop list
```

Summary



- **Pattern matching: assignment and branching based on 'shape'**
- **Cons pattern – head::tail**
- **Recursive traversal**
  - head::tail -> recursive call
  - [] -> terminate recursion