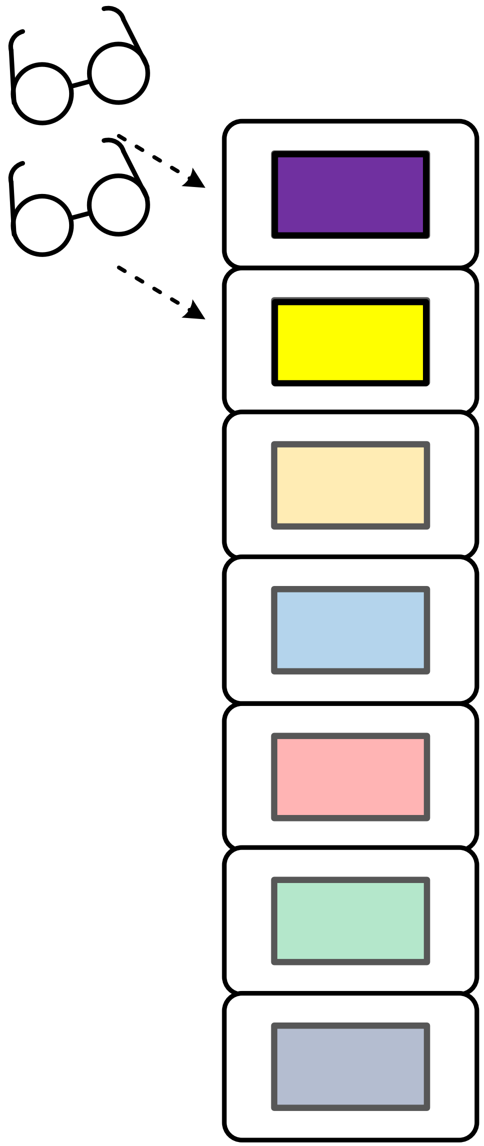
Sequences

Kit Eason
www.kiteason.com
@kitlovesfsharp

What is a Sequence?

- A list of potential values
- Computed on demand
- All elements same type
- Can’t assign to elements
- IEnumerable



Creating a Sequence

- **From a range expression**
  - `let integers = {1..1000}`
- **From a sequence expression**
  - `let integers = seq {for i in 1..1000 do yield i}`
  - `let integers = seq {for i in 1..1000 -> i}`
- **Using a function in the Seq module**
  - `let integers = Seq.init 1000 (fun i -> i+1)`
  - `let integers = Seq.initInfinite (fun i -> i+1)`
- **From an IEnumerable**

```
let Extensions (dir : string) =
    Directory.EnumerateFiles(dir)
     |> Seq.map (fun name -> Path.GetExtension(name))
     |> Seq.distinct
```

Seq.init

- Takes a length
- …and a generator function
- …which takes an int

```
val init : count:int -> initializer:(int -> 'T) -> Seq<'T>
```

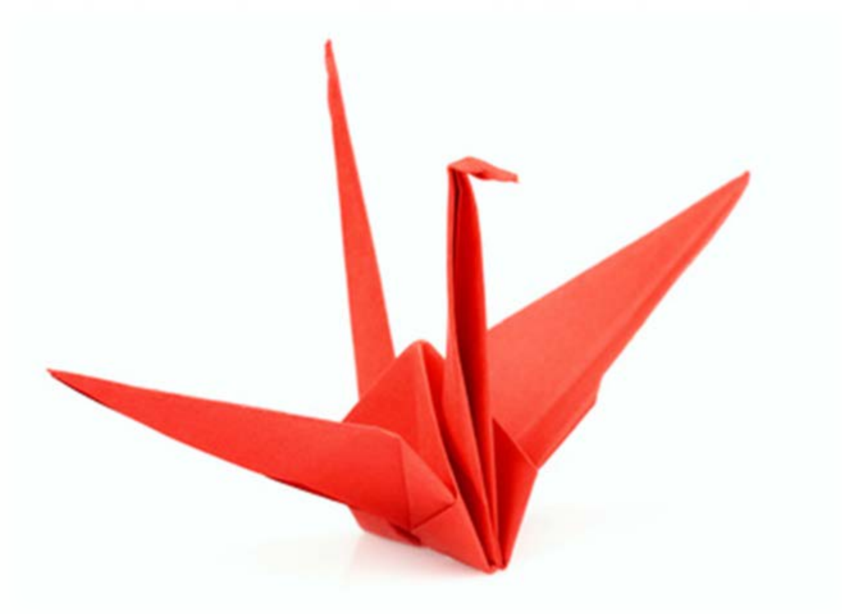
Seq.initInfinite

- Doesn’t take a length
- Still takes a generator function
- Length limited by Int32.MaxValue

```
val initInfinite : initializer:(int -> 'T) -> Seq<'T>
```

Seq.unfold

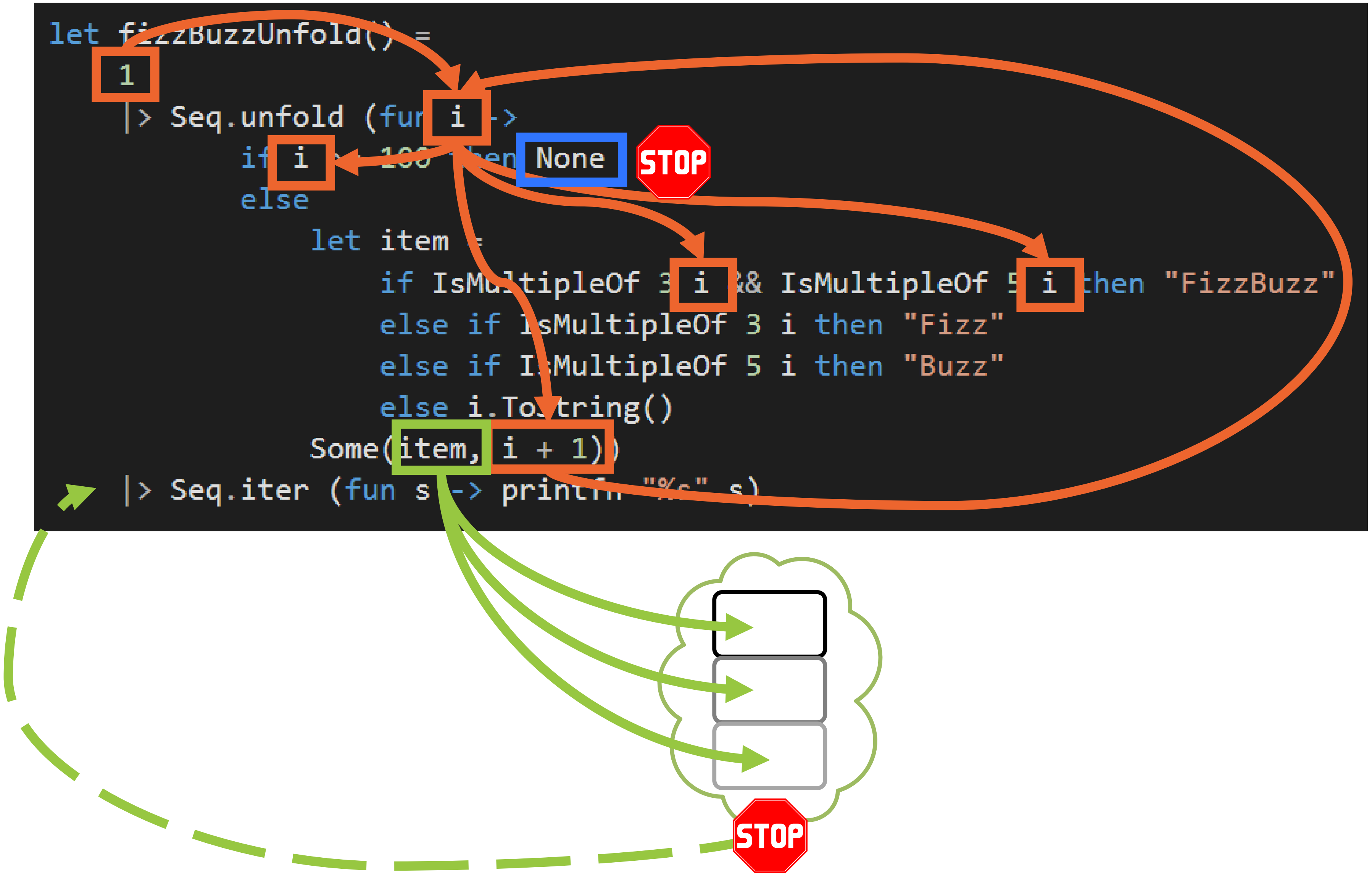
- Use when element *n* needs to depend on element *n-1*
- Not easy
- …but don't worry!



FizzBuzz

- Series of integers from 1
- When divisible by 3, say ‘Fizz’ instead
- When divisible by 5, say ‘Buzz’ instead
- When divisible by 3 and 5, say ‘FizzBuzz’ instead

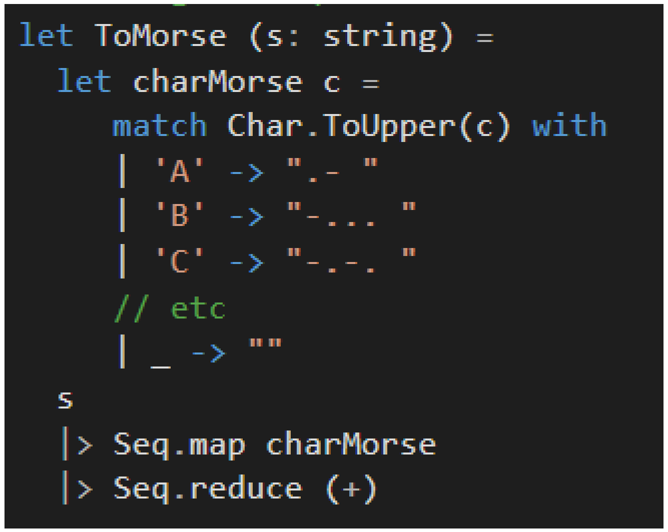
FizzBuzz with Unfold



```
let fizzBuzzUnfold() =
    1
    |> Seq.unfold (fun i ->
          if i > 100 then None
          else
              let item =
                  if IsMultipleOf 3 i && IsMultipleOf 5 i then "FizzBuzz"
                  else if IsMultipleOf 3 i then "Fizz"
                  else if IsMultipleOf 5 i then "Buzz"
                  else i.ToString()
              Some(item, i + 1))
    |> Seq.iter (fun s -> printfn "%s" s)
```

Collections as Sequences

- **Anything which implements IEnumerable is a sequence!**
  - Strings

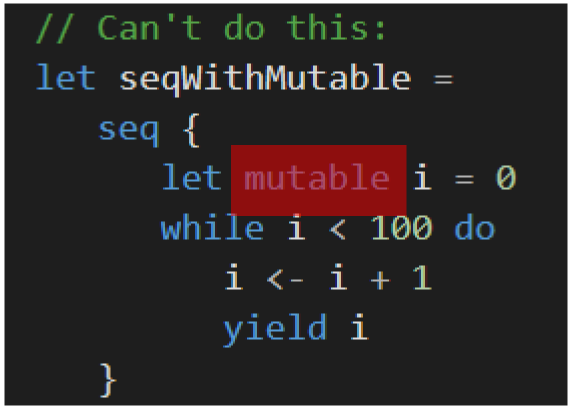


```
let ToMorse (s: string) =
  let charMorse c =
     match Char.ToUpper(c) with
     | 'A' -> ".- "
     | 'B' -> "-... "
     | 'C' -> "-.-. "
     // etc
     | _ -> ""
  s
  |> Seq.map charMorse
  |> Seq.reduce (+)
```

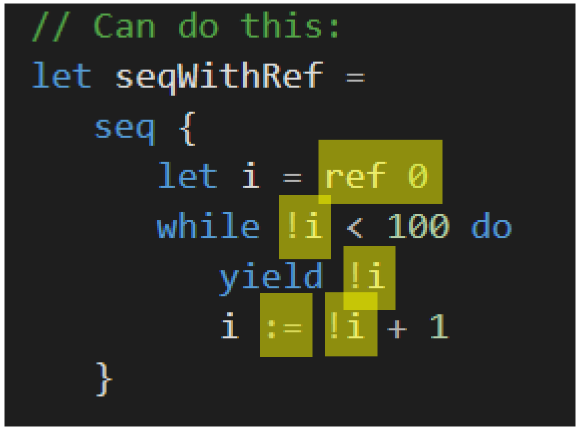
  - Arrays
  - Many, many .NET functions

More on Sequence Expressions

- seq { … }
- Must yield something
- One-legged if statements are fine
  - `if i%2 = 0 then yield i`
- No mutables!
  - Use reference cells instead



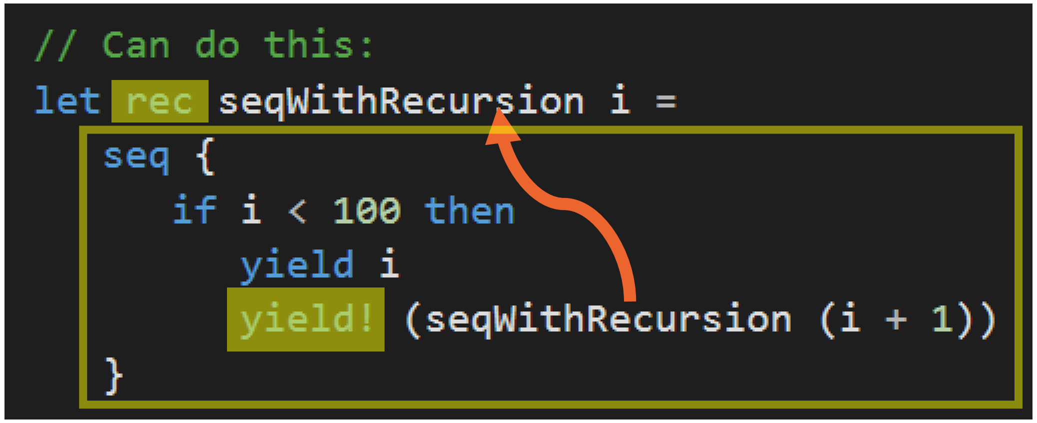
```
// Can't do this:
let seqWithMutable =
    seq {
        let mutable i = 0
        while i < 100 do
            i <- i + 1
            yield i
    }
```



```
// Can do this:
let seqWithRef =
    seq {
        let i = ref 0
        while !i < 100 do
            yield !i
            i := !i + 1
    }
```

Recursive Sequence Expressions

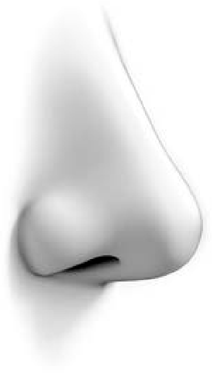
- Declare a recursive function
- Entire body is seq {…}
- Call the function recursively within the {…}
- Use yield! to yield the elements returned by the recursion

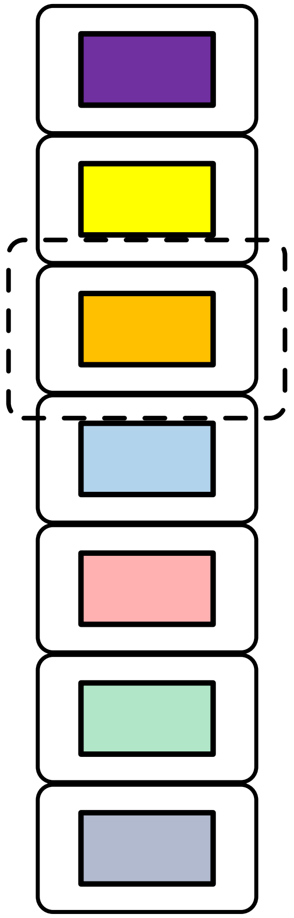


```
// Can do this:
let rec seqWithRecursion i =
    seq {
        if i < 100 then
            yield i
            yield! (seqWithRecursion (i + 1))
    }
```

Seq.nth

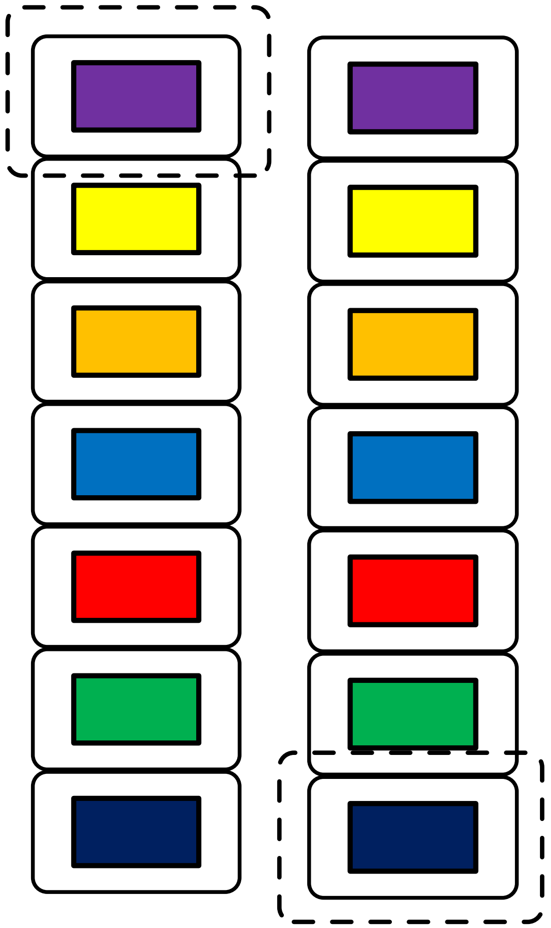
- **Takes an index value**
- **Array-like access**
- **Zero-based**
- **Care when sequence defined with Unfold or recursive seq{}**
  - Elements up to that index will be calculated
  - Consider using Seq.cache
- **It's a smell**

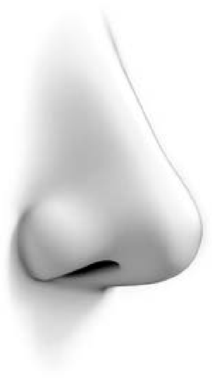




Seq.head and Seq.last

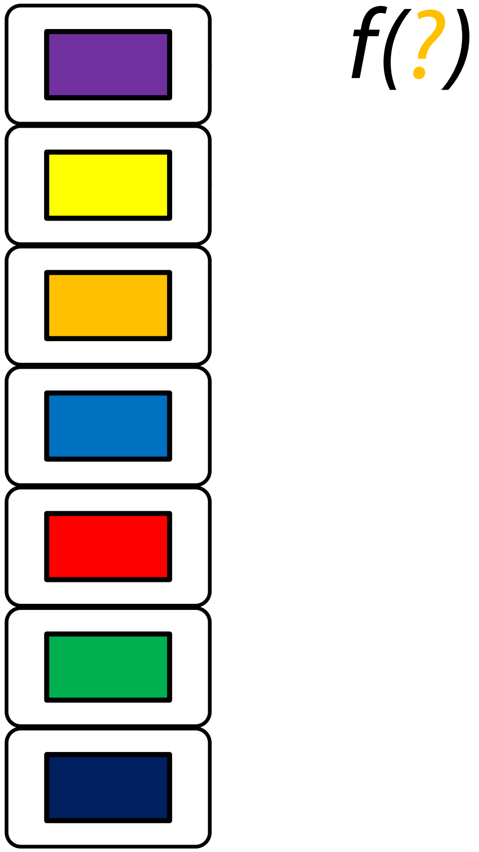
- Seq.head – the first element
- Seq.last – the last element
- Smell?





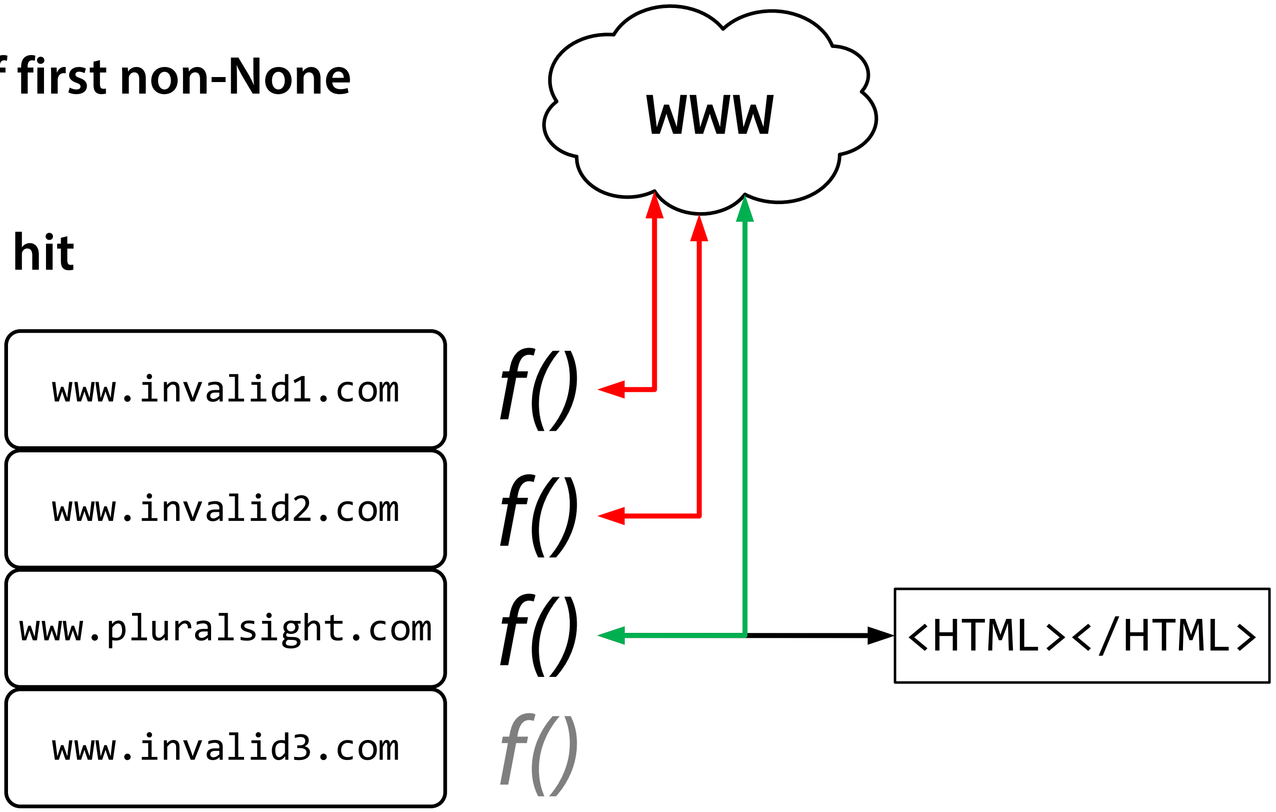
Seq.find and Seq.tryFind

- Seq.find takes a sequence and a Boolean function
- Applies function to each element
- Returns first element where function returned true
- Seq.tryFind returns an option value
- …and None if no matches



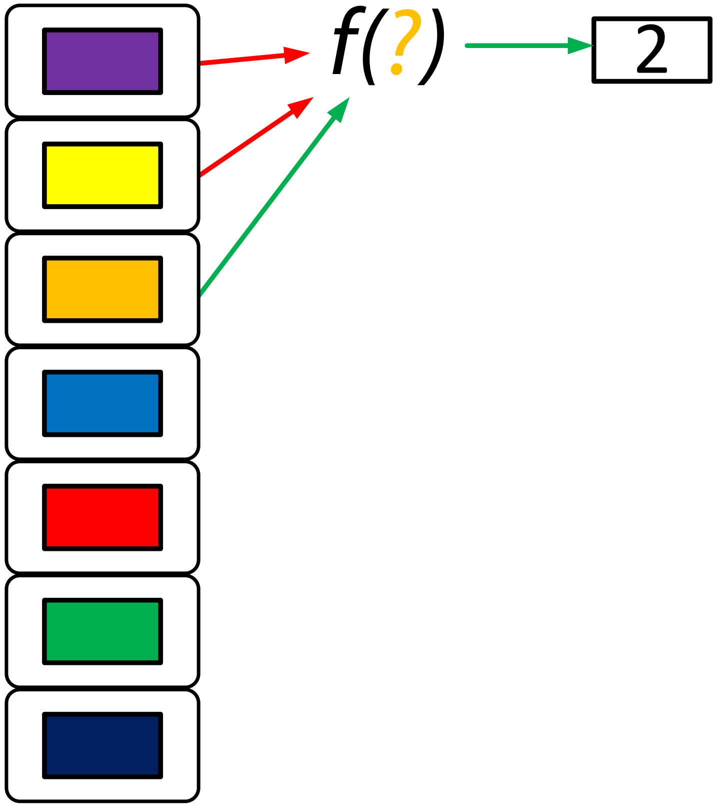
Seq.pick

- Takes a function and a sequence
- Function must return an option
- Returns value of first non-None result
- There must be a hit
- Seq.tryPick



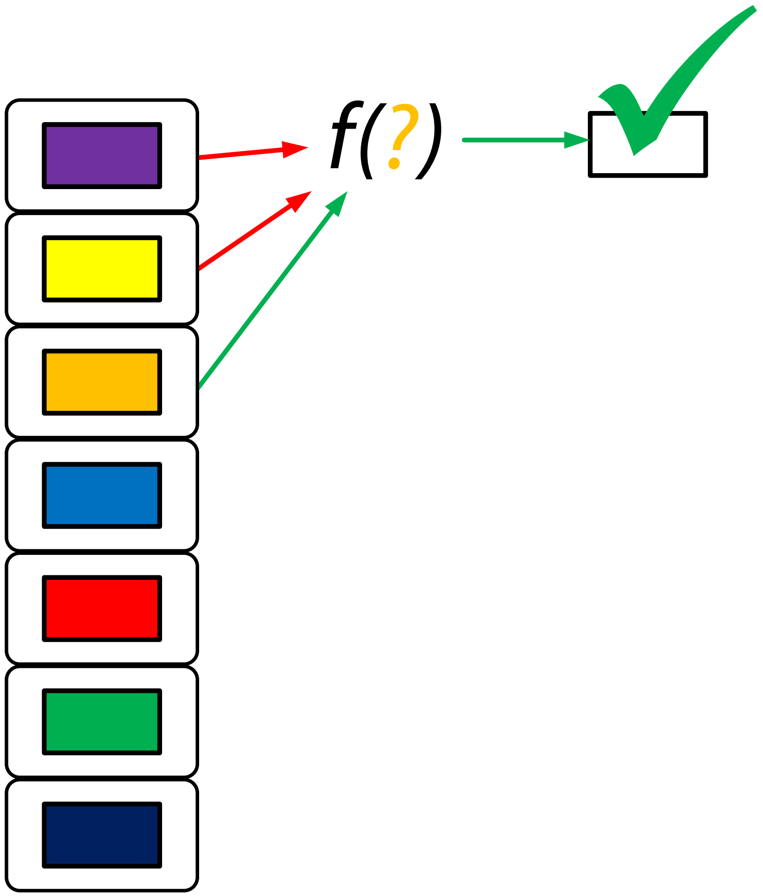
Seq.findIndex and Seq.tryFindIndex

- Work like Seq.find and Seq.tryFind
- Return the index of the matching element
- findIndex must find a value
- tryFindIndex returns None for no match



Seq.exists

- Returns true if the supplied function returns true for any element

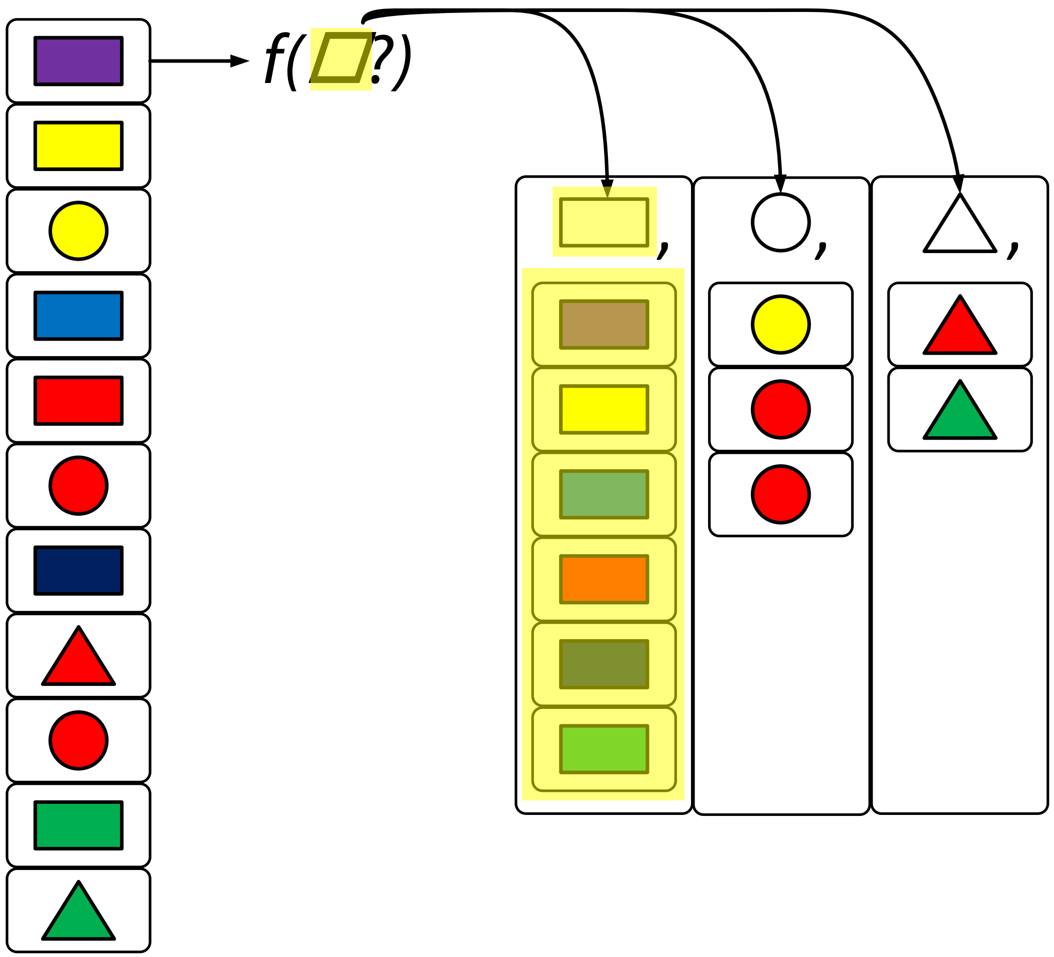


Seq.filter and Seq.choose

- Seq.filter returns elements where supplied function returns true
- Seq.choose returns those where function returns Some(value)

Seq.groupBy

- Groups a sequence by results of function
- Function might get an element property…
- …or do some calculation
- Returns sequence of key/value pairs
- Keys are distinct results
- Values are sequences of matching elements

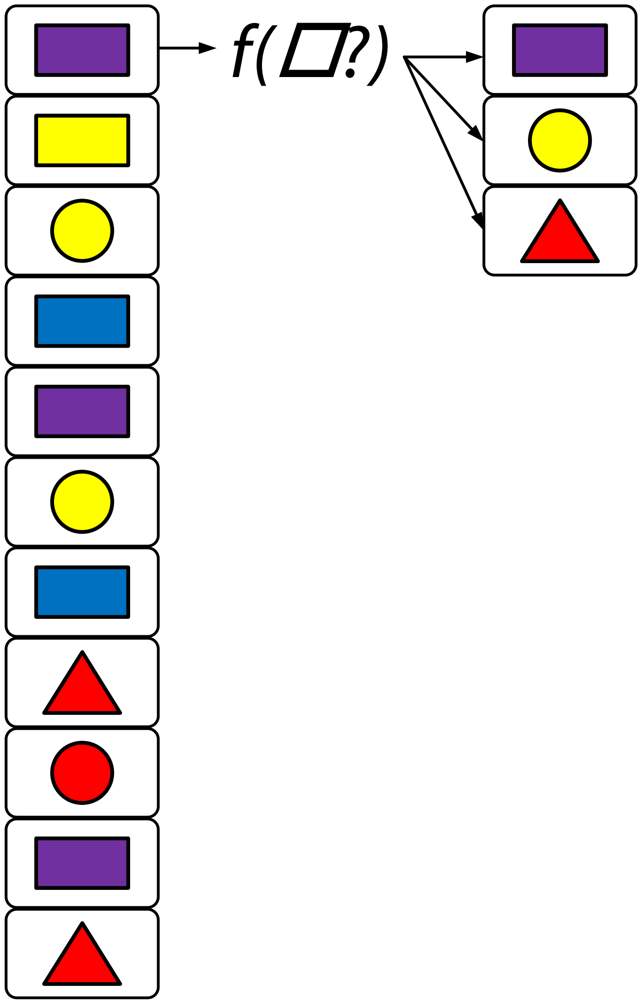


Seq.distinct

- **Takes a sequence**
- **Returns unique elements**
- **Uses default equality of element type**

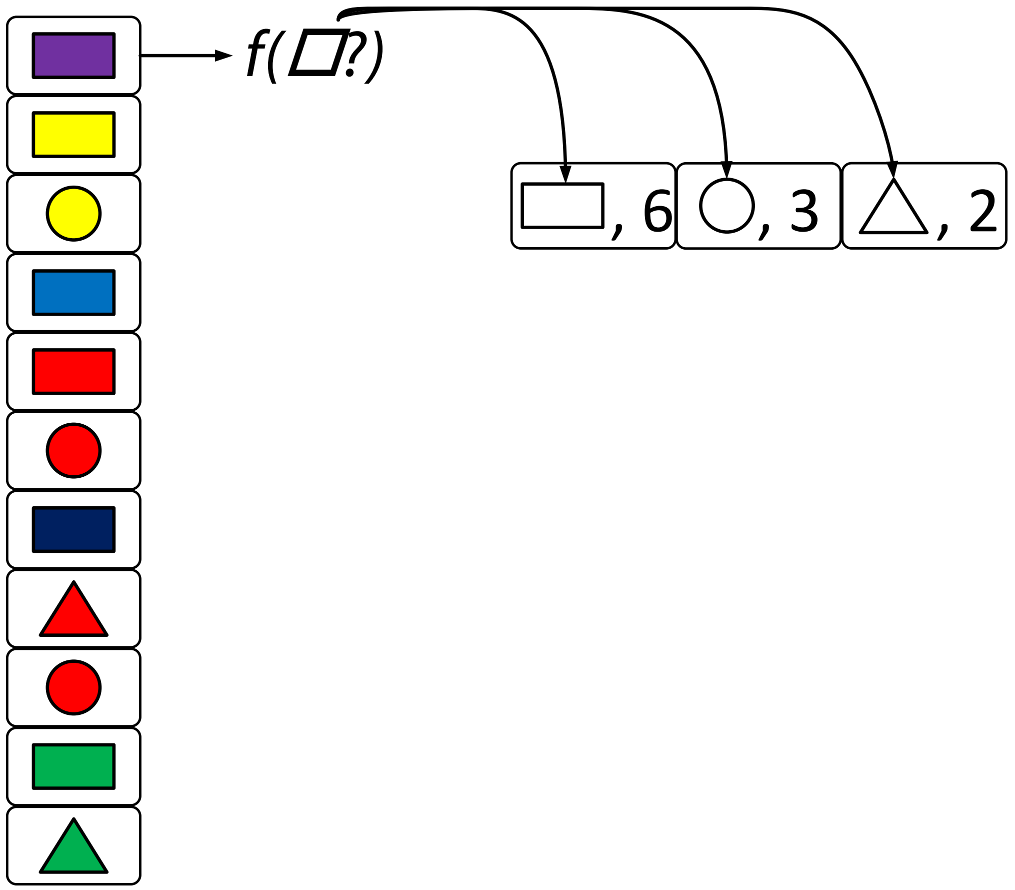
Seq.distinctBy

- Takes a function argument
- Function return type must support Equality



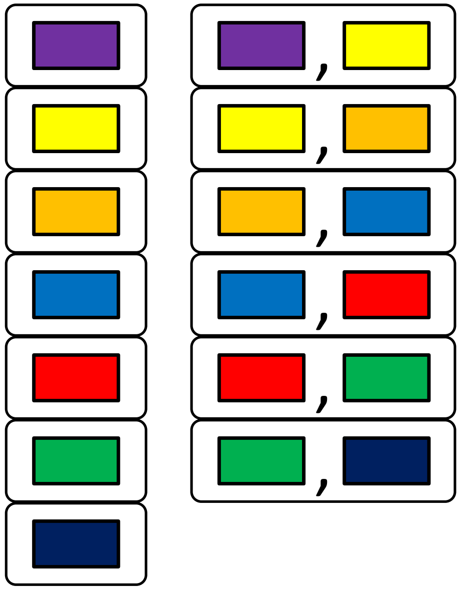
Seq.countBy

- Takes a function argument
- Function return type must support Equality
- Produces count for each distinct returned key

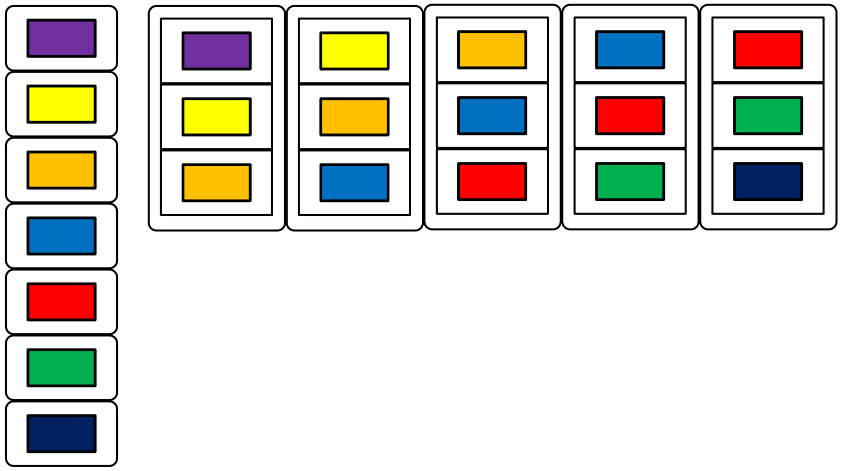


Seq.pairwise and Seq.windowed

Seq.pairwise



Seq.windowed



Seq.pairwise

- Takes a sequence
- Returns a sequence of 2-tuples
- First tuple == 1st and 2nd elements
- Second tuple == 2nd and 3rd
- …etc

Seq.windowed

- Takes a sequence
- …and a length
- Returns a sequence of arrays of that length
- First array: elements 0..length-1
- Second array: elements 1..length
- …etc

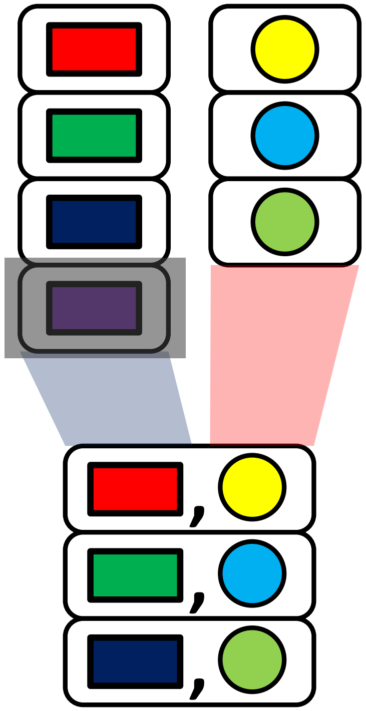
Seq.windowed

Seq.collect

- Takes an input sequence
- …and a function which produces a sequence of elements
- Applies function to each input element
- Concatenates results

There’s Much More in Seq!

- Doing something imperative with a sequence?
- Look in the Seq module first!
- Most things in Array are in Seq
  - Seq.iter, Seq.map, Seq.sum…
- Seq.zip allows different lengths



Sequences: the Trade-Offs

- Performance

- Maintainability

Maintainability

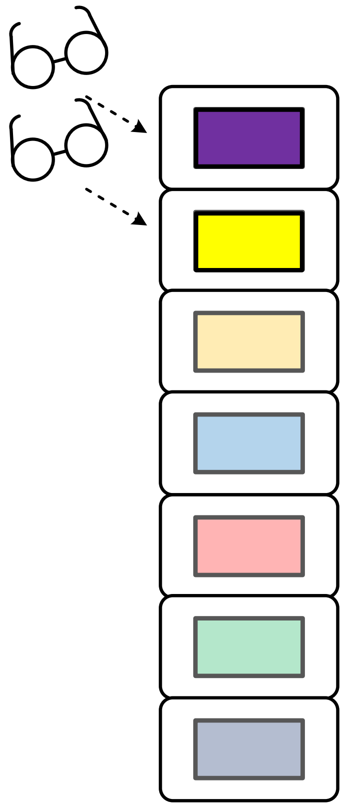
- **When is the code being executed?**
- **Debugging (stepping) can get confusing**
- **Use Array.ofSeq to get a known point**
  - (Don’t forget to take it out again!)

Performance – Repeated Evaluation

- Sequence elements evaluate on demand – Good!
- Sequences elements re-evaluate on re-demand – Bad!
- Is the underlying resource still available?
- Avoid accessing elements more than once
- Use a different structure – an Array?
- Use Seq.cache?

Summary

- A list of compute-on-demand values
- A .NET IEnumerable



Summary

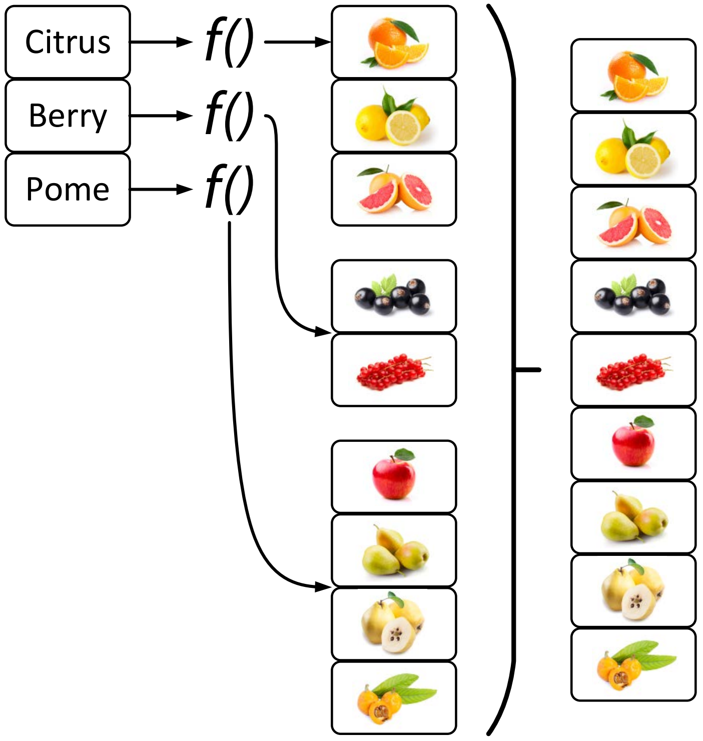
- **Create with range**
  - `let integers = {1..1000}`
- **…sequence expression**
  - `let integers = seq {for i in 1..1000 do yield i}`
- **…or function from Seq module**
  - `let integers = Seq.init 1000 (fun i -> i+1)`
- **Or treat any IEnumerable as a sequence**

Summary

- Create sequence whose values depend on previous values with…
- …Seq.unfold
- …or a recursive sequence expression – easier!

Summary

- **Seq module: many useful functions**
  - Seq.find / Seq.tryFind
  - Seq.pick
  - Seq.head / Seq.last
  - Seq.groupBy
  - Seq.pairwise / Seq.windowed



Sequence Gotchas

- When are your values being evaluated?
- Are they being evaluated more than once?
- Seq.cache