# Why Functional Programming Matters

**Nathan Taylor**
SOFTWARE ENGINEER

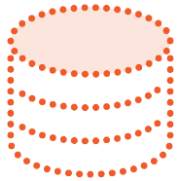
@taylonr taylonr.com

# Caching

# Sales Dashboard Application

**Performance dashboard**

**Slow changing data**



**Caching results**

Developers traumatized

# Memoization

To speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again.

Pure functions are easier to memoize

# Slow Running Function

```
def calculateTotalPrice(items) do
    array.reduce((item, acc) => acc = acc + item.price, 0, items)
end
```

# Cached Function

```
def calculateTotalPrice(items) do
    if(cache.contains(items)) do
        cache[items]
    else
        tot = reduce((item, acc) => acc = acc + item.price, 0, items)
        cache[items] = tot
        tot
    end
end
```

# Redux

Store

- API data
- Application data

Memoized


Selector

- Calculations
- Getter

# Pure functions simplify memoization

# Laziness

# Lazy Evaluation

delays the evaluation of an expression until its value is needed

```
print length([2+1, 3*2, 1/0, 5-4])
```

# Evaluation Example

```
print length([2+1, 3*2, 1/0, 5-4])
```

# Eager Evaluation

**Result: DivideByZeroException**

```
print length([2+1, 3*2, 1/0, 5-4])
```

# Lazy Evaluation

**Result: 4**

Infinite list
or
database

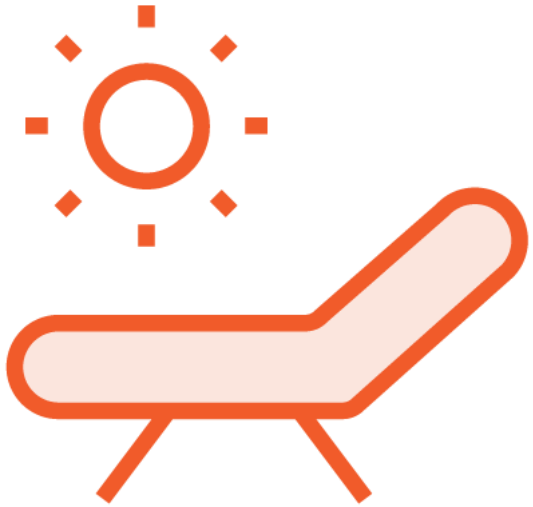
# Lazy Evaluation: Database

```
nebraskaTaylors = users
  |> where(lastName = 'Taylor’)
  |> where(state = 'NE’)
  |> orderBy('firstName')

nebraskaTaylors.map(user => print user.firstName)
```

# Lazy evaluation defers execution

# Laziness

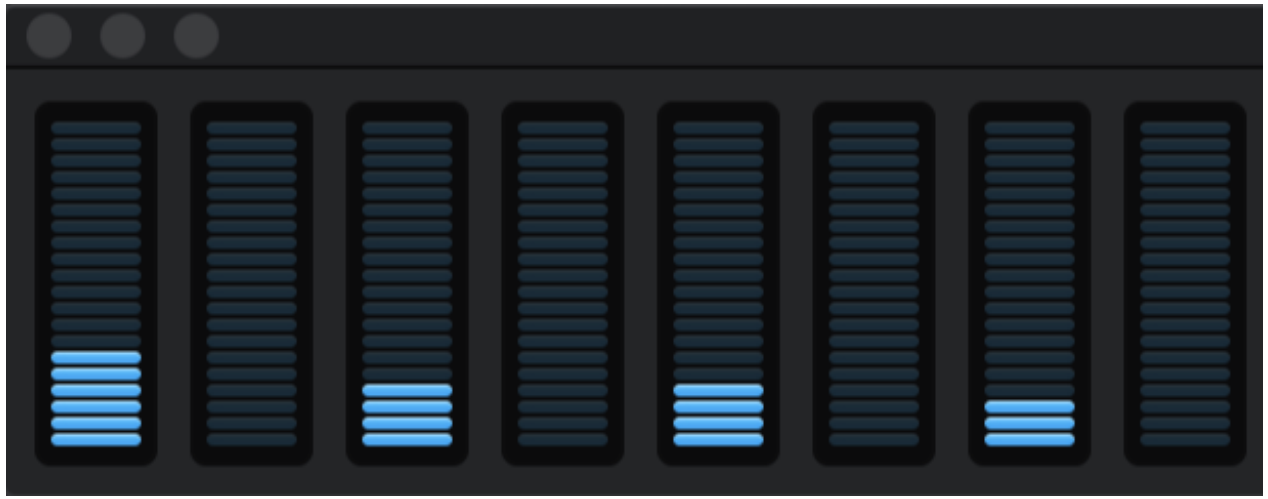


**Powered by pure functions**

Functional languages are not required to be lazy

# Parallelism and Concurrency

# Normal CPU usage



# Word Sum

## Immutable data

# Program Flow

**“In the beginning, God created the heaven and the earth”**

[‘in’, ‘the’, ‘beginning’, ‘God’, ‘created’, ‘the’, ‘heaven’, ‘and’, ‘the’, ‘earth’]

[‘in’, ‘the’] [‘beginning’] [‘God’] [‘created’, ‘the’] [‘heaven’] [‘and’] [‘the’] [‘earth’]

{ in: 1,
the: 3,
beginning: 1,
God: 1,
created: 1,
heaven: 1,
and: 1,
earth: 1 }

# Parallelization due to immutability and laziness

# Performance gains with parallelization

# Functional programming is equipped for the future

# In the Next Module

**Next steps**