Lispz - Functional Functionality

http://lispz.net

https://github.com/CanFP/Talk-2016-06-15-Lispz-functional-functionality

https://docs.google.com/presentation/d/1iFG0HLO1ZI1sV3IYHQe_YtGU-NIIAfRsaaGeidd2MGc/edit?usp=sharing

Eager Expression Evaluation

```
(ref first ((new Date).getTime))
(delay 1532
  (ref second ((new Date).getTime))
  (console.log (- second first))
)
```

Lazy Expression Evaluation

```
(ref time! (once ((new Date).getTime)))

(ref first (time!))
(delay 1532
  (ref second (time!))
  (console.log (- second first))
)
```

Lazy Expression Evaluator - Usage

```
(ref fs (lambda [name branch]
  (ref repo  (once (repo> name)))
  (ref entries> (once (when (tree> (repo) fs.branch) [tree]
    (dict.from-list tree.tree "path")
  )))
  (ref read> (lambda [path] (github.read> (repo) fs.branch path)))
  (ref fs { name entries> read> branch: (or branch "master")})
))
##...
(ref repo (github.fs “paulmarrington/lispz” “master”)
##...
(ref lispz-js     (repo.read> "lispz.js"))
(ref groups       (when repo.entries> (group @)))
```

Lazy Expression Evaluator - Implementation

```
(global #once (lambda [lazy-expression]
  (ref first-time (=>
    (ref evaluated-value (lazy-expression))
    (action.update! { func: (=> evaluated-value) })
   evaluated-value
  ))
  (ref action (lispz.globals.stateful
    { func: first-time when: (new Date) }))
  (=> (action.func))
))
(macro once [*body] (#once (=> *body)))
```

Recursion without Tail-Call Optimisation

```
(ref sum (lambda [x y]
     (cond (> y 0)
       (sum (+ x 1) (- y 1))
     (else)
       x
     }
})

(console.log (sum 1 100000)) ## => ERROR
```

Recursion with Tail-Call Optimisation

```
(ref sum (recursion [x y]
     (cond (> y 0)
       (sum (+ x 1) (- y 1))
     (else)
       x
     }
})

(console.log (sum 1 100000)) ## => ERROR
```

Tail-Call Optimisation - Implementation

```
(global #recursion (lambda [context func]
    (lambda
        (ref args (*arguments))
        (cond context.queue
          (context.queue.push args)
        (else)  (do
          (context.update! { queue: [[args]]})
          (#join '' 'while(' (ref next-args (context.queue.shift)) '){'
            (context.update! { result: (func.apply null next-args)})
          '}'))
        )
        context.result
      )
  ))
  (macro recursion [?params *body]
    (#recursion (stateful) (lambda ?params *body))
  )
```

Currying - Implementation

```
(global curry (lambda [func]
  (ref curried (=>
    (ref args (*arguments 0))
    (cond (>= args.length func.length)
      (func.apply func args)  ## all done, run it
    (else)
      (=> ## otherwise return a partial function
        (curried.apply this (args.concat (*arguments 0)))
      )
    )
  ))
))
(macro curry [params *body] (lispz.globals.curry (lambda params *body)))
```

Caching - Implementation

```
(global stateful.cache (curry [store update key]
  (or (get store key) (do
    (store.update! key (update key)) (get store key)
  ))
))
## ...
  (ref dom-events (stateful))
  (ref post-dom-event> (lambda [address]
    (ref send-to-address (lambda [pkt] (message.send address pkt)))
    (stateful.cache dom-events (=> send-to-address) address)
  ))
## …
(element.addEventListener event-name (post-dom-event> address))
```

Caching

```
(ref exchange  (stateful))
(ref observers (stateful.cache exchange (=> (stateful []))))
## ...
(ref listen (lambda [address action>]
  ((observers address).push! (stateful { action> }))
))
## ...
(ref request> (lambda [address packet]
  (ref postman (lambda [obs] (promised (obs.action> packet obs))))
  (promise.all ((observers address).map postman))
))
```

Reactive Messaging

```
(using [message dom]
  (message.clear '/my-message-address/')
  (message.trace '/my-message.*/')

  (ref @click         (dom.click "my-message-address" document.body))
  (ref @mouse         (message.map @click "mouse" (lambda [event]
                        {x: event.clientX  y: event.clientY}
                      )))
  (ref @top-left      (message.filter @mouse "top-left" (lambda [pos]
                        (< pos.x pos.y)
                      )))
  (message.listen    @top-left (=> (console.log @.x @.y)))
)
```

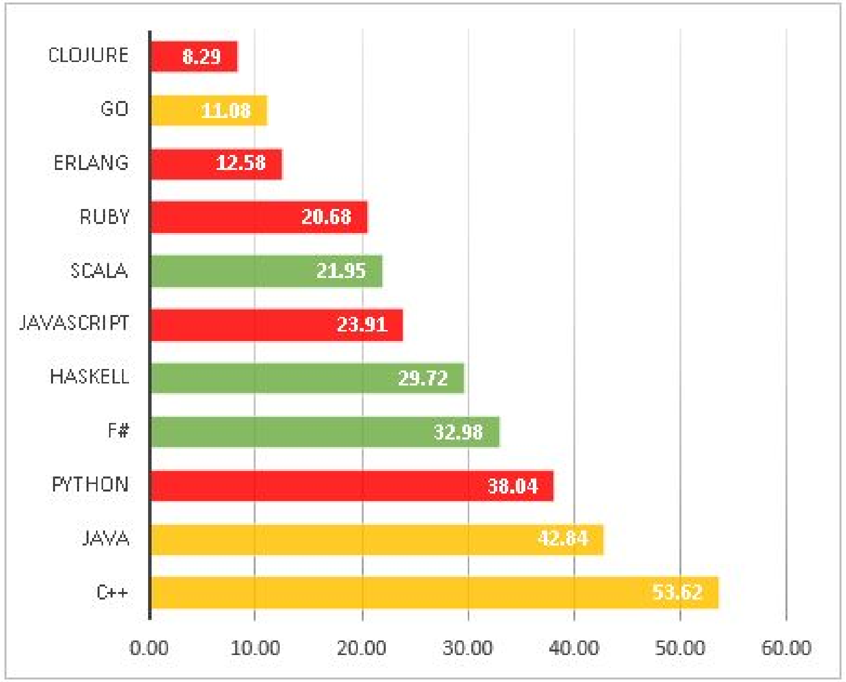
Reactive Messaging using Compose

```
(using [message dom]
   (message.clear '/my-message-address/')
   (message.trace '/my-message.*/')

   ## Same again using compose
   (cascade
      (=> (dom.click "my-message-address" document.body))
      (=> (message.map @ "mouse" (=> {x: @.clientX  y: @.clientY})))
      (=> (message.filter   @   "top-left"  (=> (< @.x @.y))))
      (=> (message.throttle @ 2000))
      (=> (message.listen   @ (=> (console.log @.x @.y))))
   )
)
```

A discussion on language and bug counts

http://labs.ig.com/static-typing-promise



My Contact Details

Email:	paul@marrington.net
Twitter:	paulmarrington
GitHub:	github.com/paulmarrington
Wordpress:	https://paulmarrington.wordpress.com/
Lispz:	http://lispz.net