Blocks, procs && lambdas

#16 ruby workshop, 2013-03-24
Vidmantas Kabošis

Closure (CS)

- function or reference to a function together with a **referencing environment**
- closure <..> allows a function to access those non-local variables even when invoked outside of its immediate lexical scope [1]

1 - http://en.wikipedia.org/wiki/Closure_%28computer_science%29

Closures in Ruby

1. Block
2. Proc
3. proc
4. lambda

Blocks

- NOT an object (mind=blown)
- Implicit pass/call
- Explicit pass/call*
- Speed! Explicit invoke is considerably slower (~50%)*
- Implicit block within implicit block?
- Implicit block behaves like Proc*

Proc

- Pretty usual method object
- Call
- Pass any number of argument(s)
- respond_to(:arity)
- Flow control keywords (**return**, break, redo, retry, ...)

proc

- Ruby 1.8: proc == lambda
- Ruby 1.9 and up: proc == Proc
  - lambda?

lambda

- Count of arguments
- return == break (proc)
- ->

end

- Implicit blocks are not objects, but can become an object when referenced (explicitly)
- Proc.new == proc, don't care about args too much & returns
- lambda counts args & diminutive returns

end!