```lua
-- Week 2, Activity 4:
-- Our first interpreter (only numbers)

local lpeg = require "lpeg"
local pt = require "pt"
local loc = lpeg.locale()

-------------------------------------------------------------------------------
-- FRONTEND: PARSER
-- Our frontend is a parser that gets a source code as input and produces an
-- intermediate representation of the program in an AST
-------------------------------------------------------------------------------
-- Initial patterns:
local spc = loc.space^0
local vazio = -lpeg.P(1)
local sinal = lpeg.S("+-")^-1

-- Function that get's a number and return a node of AST
function node(numero)
   return {
      tag = "numero",
      val = tonumber(numero)
   }
end

-- What is a number? Note that an AST node is returned
local numero = ((sinal * loc.digit^1) / node) * spc

-- The parser per si:
local function parse(input)
   return numero:match(input)
end


-------------------------------------------------------------------------------
-- BACKEND: CODE GENERATOR
-- Our backend is a code generator that get's an AST and generate the final
-- output of the compiler
-------------------------------------------------------------------------------
-- The compiler per si:
local function compile(ast)
   if ast.tag == "numero" then
      return {"push", ast.val}
   end
end


-------------------------------------------------------------------------------
-- INTERPRETER
-- Receives the intermediate code produced by the compiler and empty stack and,
-- when finished, leaves the result of the expression on the top of the stack.
-------------------------------------------------------------------------------
-- The interpreter:
local function run(code, stack)
   local pc = 1                    -- program counter
   local top = 0                   -- top of stack
   while pc <= #code do
      if code[pc] == "push" then
         pc = pc + 1
         top = top + 1
```

```
            stack[top] = code[pc]
        else
            error("unknown instruction")
        end
        pc = pc + 1
    end
end


-- Tests:
-- Get's the source code (only a number for now):
local input = io.read("a")

-- The frontend (parser) generates as AST:
local ast = parse(input)
print(pt.pt(ast))

-- The backend (code generator) compiles AST to intermediate code:
local code = compile(ast)
print(pt.pt(code))

-- We run the interpreter passing as arguments the intermediate code and AST:
local stack = {}
run(code, stack)
print(stack[1])
```