```
-- Week 2, Activity02
-- We are going to start the implementation of our language.
-- We'll start with arithmetic expressions to get the general idea of what are
-- going to do and the general architecture of our implementation.

-- The general architecture of our implementation:
--
-- SOURCE CODE -> FRONTEND -> AST -> BACKEND -> OPCODES -> VIRTUAL MACHINE
--
-- In more details:
--    - Source code: obvious
--    - Frontend: in our case it will be a PARSER, but it can be type systems,
--                and all kind of analysis, to build an intermediate
--                representation of the program (in our case an AST). The
--                frontend is the part of the compiler that will be related to
--                the source code, it's all about the input of the compiler.
--    - AST: the intermediate representation of the program
--    - Backend: in our case it's a CODE GENERATOR, that get's the intermediate
--               representation (AST) and generates the final output of the
--               compiler, that can be MACHINE CODE or, in our case, the OPCODES
--               for a VIRTUAL MACHINE. The backend is all about the output of
--               the compiler.

-- Our frontend will be a PARSER that builds an AST from an input, for example:
--    Source:              (4+3) * (5 - 1)
--
--    AST:                        *
--                              /   \
--                            +       -
--                           / \     / \
--                          4   3   5   1
--
-- The AST will be represented in Lua tables: each node is a table that contains
-- subtables for the corresponding subtrees.

-- Our backend will be a CODE GENERATOR that will use a standard technique
-- called a STACK MACHINE. Each expression generates code that leaves its value
-- on the top of the stack. A stack machine is an abstraction that we assume
-- that our virtual machine have a stack and each must generate code that leaves
-- the value of the expression on the top of the stack.
-- The "AST -> CODE GENERATOR -> Stack Machine" is basically a postorder
-- traversal of the AST. For the AST above, the stack dinamyc will be:
--                           push 4
--                           push 3
--                           add
--                           push 5
--                           push 1
--                           sub
--                           mult
-- In a stack machine, all operations pop the operands and push the results, so,
-- for example, in the our AST example, will get this on our stack machine:
--      4   ->   3   ->   7   ->   5   ->   1  ->  4  ->  28
--               4                 7        5      7
--                                          7
-- In the end, the only value in the stack is the result of the expression.
-- The stack machine allow us to have very complex arithmetic expressions.
```