```lua
-- Week 1, Activity 16: Summations
-- Some more complex forms; space treatment for
-- free format languages

local lpeg = require "lpeg"

espaco = lpeg.S(" \n\t")^0
numero = lpeg.C(lpeg.R("09")^1) * espaco
operador = lpeg.P("+") * espaco

-- Note that spaces are handled in above patterns by
-- putting it after each token, so it is "automatically"
-- handled after each token, and the only space that we
-- will have to wory about is at the beginning at the
-- full pattern, as below:
soma = espaco * numero * operador * numero
print(soma:match("12 + 20"))

-- The above pattern is almost correct, but it is
-- capturing incorrectly expressions, like this:
print(soma:match("12 + 20asf"))

-- To correct this, we explicitly have to finish
-- the capture at the end of expressions:
vazio = -lpeg.P(1)
soma = espaco * numero * operador * numero * vazio
print(soma:match("12 + 20"))
print(soma:match("12 + 20abc"))

-- How we can process the addition? It can be processed
-- in Lua but, as our expressions become more complex,
-- it will be impossible: in this case Lua will have to
-- do what LPeg is alredy doing (syntactic analysis to
-- handle priorities, etc.). It is best to use LPeg
-- itself to do the process. We call this feature
-- "aggregated captures", captures that get other
-- captures, process them, and do whatever needs to
-- be done.
-- The "trick" here is to define a new funcion and,
-- in our pattern, capture the expression passing it
-- to our function and use a "function capture" in our
-- pattern.
--
-- Function to add numbers:
function somar(a, b)
   return a + b
end
-- Pattern with a function capture: we use the "/ function"
-- to indicate to which function the expression captured is
-- send to
soma = espaco * (numero * operador * numero) / somar * vazio
print(soma:match("12 + 20"))
--
-- Note that on above pattern we are capturing strings, and
-- Lua are coercing to numbers. It is more polite to explicitly
-- cast the string to number conversion, directly on the number
-- pattern, as in:
numero = (lpeg.R("09")^1 / tonumber) * espaco
-- Note that we don't need lpeg.C anymore, because we are using a
-- tonumer function capture, the whole capture goes to the funcion capture.
-- To test the above:
soma = espaco * (numero * operador * numero) / somar * vazio
print(soma:match("12 + 20"))

-- How we can sum several numbers? First, we must change our pattern
-- to permit the capture of one number followed by 0 or more operator
-- and other numbers:
soma = espaco * (numero * (operador * numero)^0) * vazio
print(soma:match("12"))
print(soma:match("12 + 32"))
print(soma:match("12 + 32 + 14 + 90"))

-- So, how we handle this variable number of captures?
-- One option is to use a variadic function to handle the
-- variable number of captures, but in LPeg a better solution
-- is to use a Table Capture:
soma = espaco * lpeg.Ct(numero * (operador * numero)^0) * vazio
print(soma:match("12 + 32 + 14 + 90"))              --> prints table address

local pt = require "pt"                  --> library to print tables
                                         --> cp lesson-0/pt.lua /opt/lua/lib/lua/5.4
print(pt.pt(soma:match("12 + 32 + 14 + 90")))

-- Now we have a list (in a table) that can be processed easyly. We
-- create a function, usualy called "fold", that will receive the
-- list and run the summations:
function fold(lst)
   acc = lst[1]
   for i = 2, #lst do
      acc = acc + lst[i]
   end
   return acc
end
-- And now we send the table captured to the fold function:
soma = espaco * lpeg.Ct(numero * (operador * numero)^0) / fold * vazio
print(soma:match("12 + 32 + 14 + 90"))

-- We have to extend this reasoning to others operators like
-- subtraction, multiplication and division.
```