Funciones

La funciones son unas estructuras del lenguaje que permiten agrupar una serie de instrucciones que realizan una tarea específica. Esta estructura puede recibir nombres diversos, como por ejemplo: *métodos, subrutinas* o *procedimientos*. Al igual que en muchos otros lenguajes de programación, las funciones en Lua, permiten la abstracción y reutilización de fragmentos de código. En un programa en Lua se pueden definir tantas funciones como sean necesarias y sin restricción alguna sobre cuál es su tarea específica.

Las funciones pueden ser definidas por parte del usuario, pero también pueden estar definidas en el propio lenguaje de programación. Un ejemplo típico sería la función `print()`, que forma parte del propio lenguaje Lua y que permite escribir un valor en la consola.

Definición de una función

> La notación [...] significa que los elementos entre corchetes son opcionales. En caso de usarse, se deben escribir sin los corchetes.

La sintáxis de una función en Lua es:

```
[local] function nombre_funcion ([argument1, argument2, ..., argumentn])
  -- instrucciones
  [return variable1, variable2, ..., variablen]
end
```

Las funciones en Lua pueden aceptar valores de entrada, conocidos como argumentos y opcionalmente devolver un valor. Tanto los argumentos son opcionales como lo es también la devolución de un valor. El uso de los argumentos y la devolución de un valor estarán condicionados a la tarea específica que tenga la función. Veamos con más detalle los componentes:

- **local**: este elemento opcional define el ámbito de la función a local, que limita el acceso a la misma solamente desde el mismo módulo. Si no se especifica este elemento la función tendrá entonces un ámbito global (podrá ser accedida desde cualquier parte del programa).
- **function** y **end**: son las palabras clave que definen la función.
- **nombre_funcion**: es el nombre que se le da a la función. El nombre de la función junto con los argumentos constituye la signatura o firma de la función y esta debe ser única en el ámbito de la función (global o local).

- **Argumentos**: los argumentos son valores que una función acepta y que son usados para realizar una tarea. Los argumentos son opcionales, por lo que se puede crear una función sin definir ninguno. Por otra parte, el número de argumentos que puede aceptar una función es ilimitado.
- **Instrucciones**: conjunto de instrucciones que serán ejecutadas cada vez que la función sea invocada. Aquí se puede usar cualquier estructura del lenguaje, incluidas las funciones.
- **return**: palabra clave que define los valores que retorna la función. Es opcional por lo que se puede omitir. Si se quiere devolver uno o varios valores se define entonces una lista separada por comas de las variables y/o expresiones que se quieren devolver.

Ejemplos de funciones en Lua

A continuación se muestran dos ejemplos simples de funciones que realizan la suma de dos números. En el primero la función usa variables globales tanto para obtener los datos como para registrar el resultado. En el segundo ejemplo, la función acepta dos argumentos, los operandos `a` y `b` que serán usados para realizar la operación, que sera devuelta usando la palabra clave `return` .

```
a = 5
b = 3
c = 0

function suma_numeros()
  c = a + b
end

suma_numeros()

print("El resultado de la suma de a + b es: " .. c)
```

A las variables `a` y `b` se les asignan los valores 5 y 3. La función se define sin argumentos, por lo que, se usan directamente los valores de las variables. Dentro de la función se realiza la suma cuyo resultado se asigna a la variable `c`. Como puedes observar, en la función no se define tampoco ningún valor de retorno. Para ejecutar la función debemos **invocarla**, que corresponde con la instrucción `suma_numeros()`. El resultado del código anterior es:

```
$lua funcion_basica_sumar_1.lua
El resultado de la suma de a + b es: 8
```

En el resultado se observa que la suma de `a` y `b` se ha asignado a la variable `c`, cuyo valor después de invocar a la función es 8. Vamos a ver ahora otro ejemplo pasando argumentos y usando el retorno de valores:

```
va = 5
vb = 3

function suma_numeros(a, b)
  return "   El valor de la suma a + b es: " .. a + b
end

print("Suma pasando dos variables como argumentos:")
print(suma_numeros(va, vb))
print("Suma pasando dos literales a la funcion:" )
print(suma_numeros(6, 5))
```

En el ejemplo anterior, se invoca la función `suma_numeros` de dos formas distintas: una pasando las variables `va` y `vb` como argumentos; la otra, pasando los argumentos como valores literales. El resultado devuelto por la función, puede ser impreso directamente en la consola usando la instrucción `print()`. El resultado es:

```
$ lua funcion_basica_sumar_2.lua
Suma pasando dos variables como argumentos:
   El valor de la suma a + b es: 8
Suma pasando dos literales a la funcion:
   El valor de la suma a + b es: 11
```

Argumentos de una función

Los argumentos en las funciones son opcionales. Si se definen, no existe un límite en cuanto a su número. El conjunto de argumentos que se definan se conocen con el nombre de parámetros formales.

Cada argumento definido, debe recibir un nombre único. Cada uno de estos argumentos constituye una variable cuyo ámbito es local a la función, esto es, solo se pueden acceder desde las instrucciones en el cuerpo de la función. El ciclo de vida de estas variables empieza con la asignación de valores durante la invocación y termina cuando se sale de la función, ya sea por llegar a una instrucción `return`, `break` o bien porque se han ejecutado todas las instrucciones. Al terminar la ejecución de la función los argumento son eliminados y ya no serán accesibles.

Llamada a una función

Las funciones son definidas primero y ejecutadas después en aquellas partes del código donde sean necesarias. Para hacer uso de las funciones que han sido definidas, hay que **invocarlas**.

Cuando se invoca a una función, se pasarán los valores de los argumentos, si se hubieran definido. Los valores de los argumentos pueden ser definidos de diversas formas, como por ejemplo: variables, literales, valores retornados por funciones, etc.

Si la función retorna un valor, este puede ser asignado a una variable, una tabla, asignado como parámetro de otra función o incluso ignorarlo si no lo necesitamos.

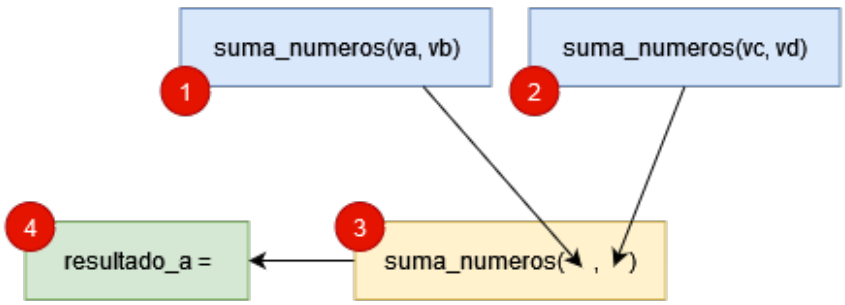
En el siguiente ejemplo se muestra una combinación de paso de argumentos por variable y por resultado de función:

```
va = 5
vb = 3
vc = 6
vd = 4

function suma_numeros(a, b)
  return a + b
end

print("Suma pasando el resultado de dos funciones:")
resultado_a = suma_numeros(suma_numeros(va, vb), suma_numeros(vc, vd))
print("   El resultado de la suma de va + vb + vc + vd es: " .. resultado_a)
```

En el ejemplo anterior podemos ver que se han anidado las llamadas a la función s`uma_numeros`, usando el valor devuelto por las llamadas anidadas para los argumentos de la llamada madre. Esto es posible, porque el intérprete de Lua, irá ejecutando las funciones desde la más profunda a la más externa, asegurando que todos los valores estarán disponibles. En el siguiente diagrama se muestra el orden en el que serán ejecutados los componentes:



El resultado que obtenemos al ejecutar el código de ejemplo anterior es:

```
$ lua funcion_basica_sumar_3.lua
Suma pasando el resultado de dos funciones:
   El resultado de la suma de va + vb + vc + vd es: 18
```

Asignación y paso de funciones

En el apartado anterior hemos presentado el paso de una función como argumento para otra. Este es uno de los métodos de asignación de funciones. A parte de este una función se puede asignar a una variable y también a una tabla. Observa el ejemplo a continuación:

```
-- Función para imprimir el resultado por consola
function mostrar_resultado(resultado)
    print("El resultado es: ", resultado)
end

-- Función anónima que es asignada a la variable sumar
sumar = function (a, b, funcion_mostrar)
    c = a + b
    funcion_mostrar(c)
end

mostrar_resultado(6)
sumar(5, 3, mostrar_resultado)
```

La primera función es una función común, que acepta un solo argumento y que no retorna ningún valor. La segunda función es asignada a una variable, sin darle ningún nombre, conocida como función anónima. Este tipo de definición permite que una variable porte la referencia a la función. Cuando se define una función de este modo, solo se podrá llamar a esta función usando la variable, como se muestra en la línea 13. También en la llamada de la línea 13, podemos observar que hemos pasado el nombre de la función `mostrar_resultado` como argumento, para que, posteriormente se pueda llamar desde dentro de la función a ésta función pasada como argumento (línea 9).

Función con argumentos variables

En ocasiones puede presentarse la necesidad de definir una función que admita un número de argumentos variable que no puede ser determinado cuando se define la función. Para este tipo de

casos Lua ofrece las funciones con argumentos variables, en las que la definición de los argumentos se realiza usando tres puntos consecutivos: `...`. Veamos un ejemplo para comprender mejor de qué se trata.

```
function suma(...)
    local suma = 0
    local argumentos = {...}

    for i,v in ipairs(argumentos) do
        suma = suma + v
    end

    return suma
end



print("La suma de los numeros 1,3,8,11,16 es: " .. suma(1,3,8,11,16))
```

Cuando se ejecuta el código anterior se obtiene el siguiente resultado:

```
$ lua funcion_basica_sumar_5.lua
La suma de los numeros 1,3,8,11,16 es: 39
```

Revisión #3
Creado 23 mayo 2023 09:03:38 por Guillermo
Actualizado 31 julio 2023 12:02:47 por Guillermo