NOTES ON LUA-5.2

VAIBHAV KARVE

These notes were last updated July 16, 2018. They are notes taken from my reading of Lua 5.2 reference manual.

1. Introduction and Basic Concepts

(1) Lua is an *extensional* language.
(2) Lua is *dynamically typed* i.e. variables do not have types, only values do.
(3) Inline commenting in Lua is achieved by typing two hyphens: `-- This is a comment`.
(4) Lua has 8 basic types:
   (a) *nil* type is the default. It is the type of the value `nil`. Similar to *Nonetype* in python.
   (b) *boolean*
   (c) *numbers* which stores double precision floating-point numbers.
   (d) *string*
   (e) *function*
   (f) *userdata* for storing arbitrary C data types.
   (g) *thread* for independent threads of execution, used to implement coroutines.
   (h) *table* for associative arrays which can be indexed with any Lua value except `nil`and `NaN`.
      - Tables can be heterogeneous i.e. they can contain values of all types (except `nil`).
      - A table with index set $\{1, \ldots, n\}$ for some integer n is a *sequence*.
      - Any key with value `nil`is not considered a part of the table.
      - Any key that is not part of a table has value `nil`.
      - The values of a table fields can be of any type. In particular, table field values can be functions.
      - Indexing of tables follows the definition of raw equality in the language:

        `a[i]==a[j]` $\iff$ `i` and `j` are raw equal.

(5) For a table `a`, Lua treats `a.name` as syntactic sugar for `a["name"]`.
(6) Tables, functions, threads and (full) userdata values are *objects*. Objects do not contain values, they contain references to values.
(7) An error message can be generated by calling the Lua function `error` and error message can be passed as a string argument to this function.

(8) Every value in Lua can have a *metatable*, which is an ordinary Lua table that defines the behavior of the original value under certain special operations.
   - Behavior can be changed by setting specific fields in the metatable.
   - Keys of the metatable are *event* names, corresponding field values are *metamethods*.
   - Metatable of any value can be queried by using the `getmetatable` function.
   - The metatable of a table can be replaced by using the `setmetatable` function. Metatables of other values cannot be changed in Lua because values of type other than table and full userdata all share a single metatable per type.
   - Each operation is identified by a string in the metatable. The key for each operation is two underscores + the name of the operation. Example: `"__add"` for addition.
   - The metamethod of an object for an event can be retrieved as such: `metatable(obj)[event]`. Access to a metamethod results in raw output and does not invoke other metamethods. Also, access to objects with no metamethods results in `nil`.

(9) Operations controlled by metamethods:
   - `"__add"` encodes the `+` operation.
   - `"__sub"` encodes the `-` operation.
   - `"__mul"` encodes the `*` operation.
   - `"__div"` encodes the `/` operation.
   - `"__mod"` encodes the `%` operation.
   - `"__pow"` encodes the `^` (exponentiation) operation.
   - `"__unm"` encodes the unary `-` operation (for creating negative numbers).
   - `"__concat"` encodes the `..` (string concatenation) operation.
   - `"__len"` encodes the `#` (string length) operation.
   - `"__eq"` encodes the `==` operation.
   - `"__lt"` encodes the `<` (less than) operation.
   - `"__le"` encodes the `<=` (less than or equal) operation. If this metamethod is absent then Lua assumes `a<=b` $\iff$ `not (b<a)`.
   - `"__index"` encodes the indexing in tables (the get-value function). This is what allows one to access a value by calling `a[key]`, where `a` is some table.
   - `"__newindex"` encodes the addition of new key-value pairs to the table. It is sort of a set-value function, allowing us to write expressions like `a[key] = blah`.
   - `"__call"` encodes the operation of calling the value stored in a variable.

(10) The expression $a \neq b$ is encoded as `a` $\sim=$ `b`.

(11) Garbabge collection and memory management is automatic in Lua.

2. The Language