Почему Lua отстой

• Объявление переменной является глобальным по умолчанию и выглядит точно так же, как назначение.

```
do
   local realVar = "foo"
   real_var = "bar" -- Oops
end
print(realVar, real_var) -- nil, "bar"
```

• Разыменование на несуществующем ключе возвращает ноль вместо ошибки. Это вместе с вышеупомянутым пунктом делает орфографические ошибки в Lua опасными и, в основном, скрытыми.

• Если vararg (аргумент переменной длины) находится в середине списка аргументов, то только первый аргумент будет учтён.

```
local function fn() return "bar", "baz" end
print("foo", fn()) -- foo bar baz
print("foo", fn(), "qux") -- foo bar qux
```

• Одновременно можно держать только один vararg (аргумент переменной длины) (в `...`).

• Невозможно сохранить varargs (аргументы переменной длины) для дальнейшего.

• Невозможно выполнять перебор varargs (аргументов переменной длины).

• Невозможно видоизменять varargs (аргументы переменной длины) непосредственно.

• Можно собрать varargs в таблицах, чтобы сделать все эти действия, но тогда необходимо будет позаботиться об исключении нулевых значений, которые имеют силу в varargs, но являются сигналом конца таблиц, как, например, \0 в C-строках.

• Табличные индексы начинаются с единицы в литералах массива и в стандартной библиотеке. Можно использовать индексацию, основанную на 0, но тогда невозможно будет использовать ни одно из указанных действий.

• Выражения `break`, `do while` (`while (something) do`, `repeat something until something`) и `goto`существуют, но нет `continue`. Странно.

• Операторы отличаются от выражений, а выражения не могут существовать вне операторов:

```
>2+2
  stdin:1: unexpected symbol near '2'
>return 2+2
  4
```

• Строковая библиотека Lua по умолчанию обеспечивает только подмножество регулярных выражений, которое само несовместимо с обычными регулярными выражениями PCRE.

• Нет способа по умолчанию для копирования таблицы. Можно написать функцию для этого, которая будет работать, пока вы не пожелаете скопировать таблицу, используя __index-метаметод.

• Нет способа наложить ограничения на аргументы функции. «Безопасные» функции Lua представляют собой мешанину из кода проверки типа.

• Отсутствует модель объекта. Само по себе это не имеет большого значения, но приводит к некоторым противоречиям — строковый тип может быть обработан как объект, включая метатаблицу и строковые значения, вызванные этим методом. Это не действует для любого другого типа.

```
>("string"):upper()
  STRING (СТРОКА)
>({1,2,3}):concat()
  stdin:1: attempt to call method 'concat' (a nil value)
>(3.14):floor()
  stdin:1: attempt to index a number value
```