Numbers Tutorial

Internal representation

Some languages support one or more of the following number types by default:

- Integer
- Unlimited precision integer
- Single precision floating point
- Double precision floating point
- Complex numbers

In the interest of simplicity Lua supports only one type of number, floating point numbers. By default these are double precision floating point numbers, but Lua can easily be recompiled to support single precision floating point numbers should you so desire. If you are unfamiliar with floating point it may be advantageous for you to read about FloatingPoint numbers.

Using numbers

We can use the Lua interactive command line prompt as a calculator by prefixing an expression by `=`, e.g.,

```
Lua 5.0  Copyright (C) 1994-2003 Tecgraf, PUC-Rio
> = 1
1
> = 1 + 2
3
> = 3.1415927
3.1415927
> = 5 / 6
0.83333333333333
```

We can enter numbers and evaluate simple calculations. Lua can also understand exponent types for expressing numbers in the form *<value> * 10 ^ <exponent>*

```
> = 1.2345e6
```

```
1234500
> = 543.21E8
54321000000
> = 2.56e-4
0.000256
```

We can assign numbers to variables and do arithmetic:

```
> width = 7.5
> height = 12.7
> = width * height
95.25
> depth = 2.8
> area = width * height
> volume = area * depth
> print(area, volume)
95.25   266.7
```

The math library

Lua is equipped with a math library (see section 5.5 of the Reference Manual). The functions provided are as follows:

```
math.abs     math.acos    math.asin    math.atan    math.atan2
math.ceil    math.cos     math.deg     math.exp     math.floor
math.log     math.log10   math.max     math.min     math.mod
math.pow     math.rad     math.sin     math.sqrt    math.tan
math.frexp   math.ldexp   math.random  math.randomseed
```

We'll try a few of the functions as an example.

```
> = math.sqrt(101)
10.049875621121
> = math.pi
3.1415926535898
> = math.sin( math.pi/3 )
0.86602540378444
```

Read the MathLibraryTutorial for more details.

Conversion

You can convert strings to numbers using the function `tonumber()`. This takes a string argument and returns a number.

```
> = tonumber("123") + 25
148
> x = tonumber("123.456e5")
> print(x)
12345600
```

Coercion

Lua will automatically convert string and number types to the correct format in order to perform calculations. For example, if you try to apply an arithmetic operation to a string, Lua will try to convert that string to a number first, otherwise the operation will not work. If the string cannot be converted to a number an error is raised. This automatic conversion of types is called *coercion*.

```
> = 100 + "7"
107
> = "1000" + 234
1234
> = "hello" + 234
stdin:1: attempt to perform arithmetic on a string value
stack traceback:
        stdin:1: in main chunk
        [C]: ?
> = 234 + "1000"
1234
```

You can see where a string can be converted to number, the calculation succeeds. The string `"hello"` cannot be converted to a number and so an error occurs. In *statically typed* languages (e.g. C) this would cause an error as you cannot assign a value to a variable of an incompatible type. This works in Lua because it is *dynamically typed*. We can use the Lua function `type()` to get a description of the type of a particular object.

```
> x = "123"  -- a string
> print(x, type(x)) -- show the value of x and its type
123     string
> x = x + 7  -- add a number to the string which forces coercion
> print(x, type(x)) -- again show the value and type
130     number
```