Linguagem Lua

Origem

- Linha do Tempo (Versões):

- Criadores:
  - Desenvolvida pelo Instituto Tecgraf da PUC-Rio
- Influências:
  - Inspirado por Lisp, Scheme, Python e Pascal

Classificação

- Linguagem de Script
- Interpretada e de Tipagem Dinâmica:
  - Lua oferece um ambiente de interpretação REPL e possui tipagem dinâmica

Metatabelas e Metamétodos

- Exemplo de realista uso: Vetores 3D e suas aplicações em computação gráfica.
- Tabelas e Orientação a Objetos
  - Metatabelas controlam tabelas utilizando de metamétodos (permite a emulação de orientação a objetos).

Exemplos em Lua (Operações com vetores):

```lua
local Vetor2_meta = {
    __add = function(a, b)
        return Vetor2 (a.x + b.x, a.y + b.y)
    end,

    __sub = function(a, b)
        return Vetor2 (a.x - b.x, a.y - b.y)
    end,

    __mul = function (a, b)
        return Vetor2 (a.x * b.x, a.y * b.y)
    end,
    __call = function (self, ...)
        print("{" .. self.x .. "," .. self.y .. "}")
    end,
    __tostring = function(self)
        return "{" .. self.x .. "," .. self.y .. "}"
    end, -- Caso não houvesse a sobrescrição do metametodo iria ser retornado o endereço da table.
}

function Vetor2(x, y)
    local v = {x = x or 0 , y = y or 0}
    setmetatable(v, Vetor2_meta)
    return v
end
```

```lua
function Vetor2(x, y)
    local v = {x = x or 0 , y = y or 0}
    setmetatable(v, Vetor2_meta)
    return v
end

local a = Vetor2(5, 7)
local b = Vetor2(10, 3)
local soma = Vetor2(0, 0)
local subtracao = Vetor2(0, 0)

soma = a + b -- soma de vetores
subtracao = a - b -- subtração de vetores

local produto = Vetor2(0,0)

produto = a * b -- Produto de vetores

print("Vetor soma dos vetores A e B: ", soma)
print("Vetor subtração dos vetores A e B: ", subtracao)
print("Vetor produto escalar dos vetores A e B: ", produto)
```

Exemplos em C# (Operações com Vetores):

```
public partial struct Vector3 : IEquatable<Vector3>, IFormattable
{
    // *Undocumented*
    public const float kEpsilon = 0.00001F;
    // *Undocumented*
    public const float kEpsilonNormalSqrt = 1e-15F;

    // X component of the vector.
    public float x;
    // Y component of the vector.
    public float y;
    // Z component of the vector.
    public float z;
```

```
[MethodImpl(MethodImplOptionsEx.AggressiveInlining)]
public static Vector3 operator+(Vector3 a, Vector3 b) { return new Vector3(a.x + b.x, a.y + b.y, a.z + b.z); }
// Subtracts one vector from another.
[MethodImpl(MethodImplOptionsEx.AggressiveInlining)]
public static Vector3 operator-(Vector3 a, Vector3 b) { return new Vector3(a.x - b.x, a.y - b.y, a.z - b.z); }
// Negates a vector.
[MethodImpl(MethodImplOptionsEx.AggressiveInlining)]
public static Vector3 operator-(Vector3 a) { return new Vector3(-a.x, -a.y, -a.z); }
// Multiplies a vector by a number.
[MethodImpl(MethodImplOptionsEx.AggressiveInlining)]
public static Vector3 operator*(Vector3 a, float d) { return new Vector3(a.x * d, a.y * d, a.z * d); }
// Multiplies a vector by a number.
[MethodImpl(MethodImplOptionsEx.AggressiveInlining)]
public static Vector3 operator*(float d, Vector3 a) { return new Vector3(a.x * d, a.y * d, a.z * d); }
// Divides a vector by a number.
[MethodImpl(MethodImplOptionsEx.AggressiveInlining)]
public static Vector3 operator/(Vector3 a, float d) { return new Vector3(a.x / d, a.y / d, a.z / d); }

// Returns true if the vectors are equal.
public static bool operator==(Vector3 lhs, Vector3 rhs)
{
    // Returns false in the presence of NaN values.
    float diff_x = lhs.x - rhs.x;
    float diff_y = lhs.y - rhs.y;
    float diff_z = lhs.z - rhs.z;
    float sqrmag = diff_x * diff_x + diff_y * diff_y + diff_z * diff_z;
    return sqrmag < kEpsilon * kEpsilon;
}
```