



# Inhaltsverzeichnis

---

## Modul:Color

This module is used primarily by [Vorlage:TI](#), eliminating the need for external color converters and preventing mismatch between color coordinates.

### Usage

To use this module, you may use one of the above listed templates or invoke the module directly. All functions that accept **hexadecimal triplets** also handle the shorthand three-digit format.

To convert a hexadecimal triplet to an RGB triplet as comma-separated values:

```
{{#invoke:Color|hexToRgbTriplet|color}}
```

To convert a hexadecimal triplet to the **CMYK color model** without a color profile:

```
{{#invoke:Color|hexToCmyk|color|precision=?|pctsign=?}}
```

To convert a hexadecimal triplet to **HSL or HSV**:

```
{{#invoke:Color|hexToHsl|color|precision=?}}
```

```
{{#invoke:Color|hexToHsv|color|precision=?}}
```

To convert a hexadecimal triplet to the perceptual **CIE LCh<sub>uv</sub>** color space:

```
{{#invoke:Color|hexToCielch|color|precision=?}}
```

To mix two colors in the more physically correct linear RGB space:

```
{{#invoke:Color|hexMix|color1|color2|proportion|min=?|max=?}}
```

The following parameters are optional:

- **precision**: defaults to 0 (zero)
- **pctsign**: set to 0 (zero) to suppress percent signs in the generated output
- **proportion**: proportion of `color2`, defaults to 50
- **min**: minimum value of proportion range, defaults to 0
- **max**: maximum value of proportion range, defaults to 100

```
-- Introduction: https://colorspace.r-forge.r-project.org/articles/color_spaces.f
local p = {}

local function isempty(v)
    return v == nil or v == ''
end

local function hexToRgb(color)
    local cleanColor = color:gsub("#35;", "#"):match('^[%s#]*(.)[%s;]*$')
    if (#cleanColor == 6) then
        return {
            r = tonumber(string.sub(cleanColor, 1, 2), 16),
            g = tonumber(string.sub(cleanColor, 3, 4), 16),

```

```
        b = tonumber(string.sub(cleanColor, 5, 6), 16)
    }
elseif (#cleanColor == 3) then
    return {
        r = 17 * tonumber(string.sub(cleanColor, 1, 1), 16),
        g = 17 * tonumber(string.sub(cleanColor, 2, 2), 16),
        b = 17 * tonumber(string.sub(cleanColor, 3, 3), 16)
    }
end
error("Invalid hexadecimal color " .. cleanColor, 1)
end

local function round(v)
    if (v < 0) then
        return math.ceil(v - 0.5)
    else
        return math.floor(v + 0.5)
    end
end

local function rgbToHex(r, g, b)
    return string.format("%02X%02X%02X", round(r), round(g), round(b))
end

local function rgbToCmyk(r, g, b)
    if (r > 255 or g > 255 or b > 255 or r < 0 or g < 0 or b < 0) then
        error("Color level out of bounds")
    end
    local c = 1 - r / 255
    local m = 1 - g / 255
    local y = 1 - b / 255
    local k = math.min(c, m, y)
    if (k == 1) then
        c = 0
        m = 0
        y = 0
    else
        local d = 1 - k
        c = (c - k) / d
        m = (m - k) / d
        y = (y - k) / d
    end
    return { c = c * 100, m = m * 100, y = y * 100, k = k * 100 }
end

local function rgbToHsl(r, g, b)
    if (r > 255 or g > 255 or b > 255 or r < 0 or g < 0 or b < 0) then
        error("Color level out of bounds")
    end
    local channelMax = math.max(r, g, b)
    local channelMin = math.min(r, g, b)
    local range = channelMax - channelMin
    local h, s
    if (range == 0) then
        h = 0
    elseif (channelMax == r) then
        h = 60 * ((g - b) / range)
        if (h < 0) then
            h = 360 + h
        end
    elseif (channelMax == g) then
        h = 60 * (2 + (b - r) / range)
    else
        h = 60 * (4 + (r - g) / range)
    end
end
```

```
        end
        local L = channelMax + channelMin
        if (L == 0 or L == 510) then
            s = 0
        else
            s = 100 * range / math.min(L, 510 - L)
        end
        return { h = h, s = s, l = L * 50 / 255 }
    end

local function rgbToHsv(r, g, b)
    if (r > 255 or g > 255 or b > 255 or r < 0 or g < 0 or b < 0) then
        error("Color level out of bounds")
    end
    local channelMax = math.max(r, g, b)
    local channelMin = math.min(r, g, b)
    local range = channelMax - channelMin
    local h, s
    if (range == 0) then
        h = 0
    elseif (channelMax == r) then
        h = 60 * ((g - b) / range)
        if (h < 0) then
            h = 360 + h
        end
    elseif (channelMax == g) then
        h = 60 * (2 + (b - r) / range)
    else
        h = 60 * (4 + (r - g) / range)
    end
    if (channelMax == 0) then
        s = 0
    else
        s = 100 * range / channelMax
    end
    return { h = h, s = s, v = channelMax * 100 / 255 }
end

-- c in [0, 255], condition tweaked for no discontinuity
-- http://entropymine.com/imageworsener/srgbformula/
local function toLinear(c)
    if (c > 10.314300250662591) then
        return math.pow((c + 14.025) / 269.025, 2.4)
    else
        return c / 3294.6
    end
end

local function toNonLinear(c)
    if (c > 0.00313066844250063) then
        return 269.025 * math.pow(c, 1.0/2.4) - 14.025
    else
        return 3294.6 * c
    end
end

local function srgbToCielchuvD65o2deg(r, g, b)
    if (r > 255 or g > 255 or b > 255 or r < 0 or g < 0 or b < 0) then
        error("Color level out of bounds")
    end
    local R = toLinear(r)
    local G = toLinear(g)
    local B = toLinear(b)
    -- https://github.com/w3c/csswg-drafts/issues/5922
```



```
local X = 0.1804807884018343 * B + 0.357584339383878 * G + 0.412390799265
local Y = 0.07219231536073371 * B + 0.21263900587151027 * R + 0.715168678
local Z = 0.01933081871559182 * R + 0.11919477979462598 * G + 0.950532152
local L, C, h
if (Y > 0.00885645167903563082) then
    L = 116 * math.pow(Y, 1/3) - 16
else
    L = Y * 903.2962962962962962963
end
if ((r == g and g == b) or L == 0) then
    C = 0
    h = 0
else
    d = X + 3 * Z + 15 * Y
    if (d == 0) then
        C = 0
        h = 0
    else
        -- 0.19783... and 0.4631... computed with extra precision
        -- in which case (u,v) ≈ (0,0)
        local us = 4 * X / d - 0.19783000664283678994
        local vs = 9 * Y / d - 0.46831999493879099801
        h = math.atan2(vs, us) * 57.2957795130823208768
        if (h < 0) then
            h = h + 360
        elseif (h == 0) then
            h = 0 -- ensure zero is positive
        end
        C = math.sqrt(us * us + vs * vs) * 13 * L
        if (C == 0) then
            C = 0
            h = 0
        end
    end
end
return { L = L, C = C, h = h }
end

local function srgbMix(t, r0, g0, b0, r1, g1, b1)
    if (t > 1 or t < 0) then
        error("Interpolation parameter out of bounds")
    end
    if (r0 > 255 or g0 > 255 or b0 > 255 or r1 > 255 or g1 > 255 or b1 > 255)
        error("Color level out of bounds")
    end
    local tc = 1 - t
    return {
        r = toNonLinear(tc * toLinear(r0) + t * toLinear(r1)),
        g = toNonLinear(tc * toLinear(g0) + t * toLinear(g1)),
        b = toNonLinear(tc * toLinear(b0) + t * toLinear(b1))
    }
end

local function formatToPrecision(value, p)
    return string.format("%. " .. p .. "f", value)
end

local function getFractionalZeros(p)
    if (p > 0) then
        return "." .. string.rep("0", p)
    else
        return ""
    end
end
```



```
function p.hexToRgbTriplet(frame)
    local args = frame.args or frame:getParent().args
    local hex = args[1]
    if (hex) then
        local rgb = hexToRgb(hex)
        return rgb.r .. ', ' .. rgb.g .. ', ' .. rgb.b
    else
        return ""
    end
end

function p.hexToCmyk(frame)
    local args = frame.args or frame:getParent().args
    local hex = args[1]
    if (hex) then
        local p = tonumber(args.precision) or 0
        local s = args.pctsign or "1"
        local rgb = hexToRgb(hex)
        local cmyk = rgbToCmyk(rgb.r, rgb.g, rgb.b)
        local fk = formatToPrecision(cmyk.k, p)
        local fc, fm, fy
        local fracZeros = getFractionalZeros(p)
        if (fk == 100 .. fracZeros) then
            local fZero = 0 .. fracZeros
            fc = fZero
            fm = fZero
            fy = fZero
        else
            fc = formatToPrecision(cmyk.c, p)
            fm = formatToPrecision(cmyk.m, p)
            fy = formatToPrecision(cmyk.y, p)
        end
        if (s ~= "0") then
            return fc .. "%, " .. fm .. "%, " .. fy .. "%, " .. fk .. "
        else
            return fc .. ", " .. fm .. ", " .. fy .. ", " .. fk
        end
    else
        return ""
    end
end

function p.hexToHsl(frame)
    local args = frame.args or frame:getParent().args
    local hex = args[1]
    if (hex) then
        local p = tonumber(args.precision) or 0
        local rgb = hexToRgb(hex)
        local hsl = rgbToHsl(rgb.r, rgb.g, rgb.b)
        local fl = formatToPrecision(hsl.l, p)
        local fs, fh
        local fracZeros = getFractionalZeros(p)
        local fZero = 0 .. fracZeros
        if (fl == fZero or fl == 100 .. fracZeros) then
            fs = fZero
            fh = fZero
        else
            fs = formatToPrecision(hsl.s, p)
            if (fs == fZero) then
                fh = fZero
            else
                fh = formatToPrecision(hsl.h, p)
                if (fh == 360 .. fracZeros) then

```

```

                fh = fZero -- handle rounding to 360
            end
        end
    end
    return fh .. "°", " .. fs .. "%", " .. fl .. %"
else
    return ""
end
end

function p.hexToHsv(frame)
    local args = frame.args or frame:getParent().args
    local hex = args[1]
    if (hex) then
        local p = tonumber(args.precision) or 0
        local rgb = hexToRgb(hex)
        local hsv = rgbToHsv(rgb.r, rgb.g, rgb.b)
        local fv = formatToPrecision(hsv.v, p)
        local fs, fh
        local fracZeros = getFractionalZeros(p)
        local fZero = 0 .. fracZeros
        if (fv == fZero) then
            fh = fZero
            fs = fZero
        else
            fs = formatToPrecision(hsv.s, p)
            if (fs == fZero) then
                fh = fZero
            else
                fh = formatToPrecision(hsv.h, p)
                if (fh == 360 .. fracZeros) then
                    fh = fZero -- handle rounding to 360
                end
            end
        end
    end
    return fh .. "°", " .. fs .. "%", " .. fv .. %"
else
    return ""
end
end

function p.hexToCielch(frame)
    local args = frame.args or frame:getParent().args
    local hex = args[1]
    if (hex) then
        local p = tonumber(args.precision) or 0
        local rgb = hexToRgb(hex)
        local LCh = srgbToCielchuvD65o2deg(rgb.r, rgb.g, rgb.b)
        local fL = formatToPrecision(LCh.L, p)
        local fC, fh
        local fracZeros = getFractionalZeros(p)
        local fZero = 0 .. fracZeros
        if (fL == fZero or fL == 100 .. fracZeros) then
            fC = fZero
            fh = fZero
        else
            fC = formatToPrecision(LCh.C, p)
            if (fC == fZero) then
                fh = fZero
            else
                fh = formatToPrecision(LCh.h, p)
                if (fh == 360 .. fracZeros) then
                    fh = fZero -- handle rounding to 360
                end
            end
        end
    end
end
```



```
                end
                end
                return fL .. ", " .. fC .. ", " .. fh .. "°"
            else
                return ""
            end
        end
    end

function p.hexMix(frame)
    local args = frame.args or frame:getParent().args
    local hex0 = args[1]
    local hex1 = args[2]
    if (isempty(hex0) or isempty(hex1)) then
        return ""
    end
    local t = args[3]
    if (isempty(t)) then
        t = 0.5
    else
        t = tonumber(t)
        local min = tonumber(args.min) or 0
        local max = tonumber(args.max) or 100
        if (min >= max) then
            error("Minimum proportion greater than or equal to maximum")
        elseif (t < min) then
            t = 0
        elseif (t > max) then
            t = 1
        else
            t = (t - min) / (max - min)
        end
    end
    local rgb0 = hexToRgb(hex0)
    local rgb1 = hexToRgb(hex1)
    local rgb = srgbMix(t, rgb0.r, rgb0.g, rgb0.b, rgb1.r, rgb1.g, rgb1.b)
    return rgbToHex(rgb.r, rgb.g, rgb.b)
end

return p
```