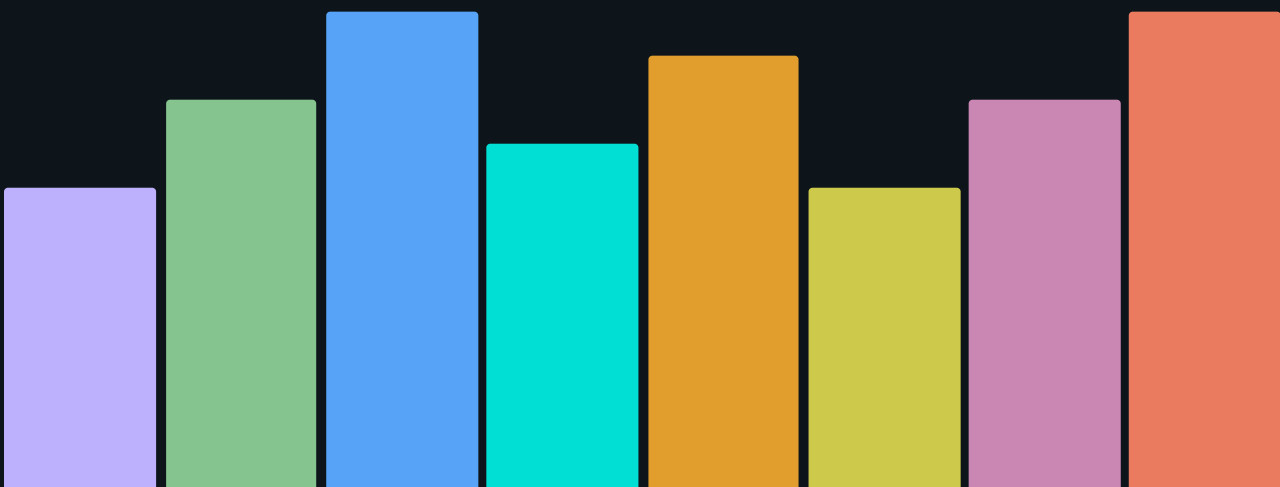


Color that lets your eyes rest.

Color theory for screens and code, and the science
behind Kvile



Color that lets your eyes rest

Color theory for screens and code, and the science behind Kvile

Copyright 2026 Christoph Pilka.

All numbers in this book come from Kvile's generator and its check report (`dist/tokens.json`, `dist/report.md`). Every figure is computed from the same data, not drawn by hand.

This book explains why the palette looks the way it does. It is not medical advice. Where research is mixed, the text says so.

Contents

PART I: COLOR FOR SCREENS

LIGHT, NOT PAINT

HOW YOU SEE COLOR

DESCRIBING COLOR

CONTRAST

COLOR VISION DIFFERENCES

COLOR IN CODE

PART II: KVILE

THE IDEA BEHIND KVILE

BUILDING THE PALETTE

THE FOUR VARIANTS

VERIFICATION

DECISIONS, TRADE-OFFS AND LIMITS

USING THIS IN YOUR OWN WORK

APPENDICES

A The palettes

B Check results

C Glossary and sources

Before you start

You stare at a screen for most of your working day, and most of what you look at is text with color on it. Color decides how fast you can find a function name, whether an error stands out, and how tired your eyes feel at six in the evening.

This book has two parts. **Part I** is color theory for people who build things on screens: how a display makes color, how eyes and brains read it, how to describe color so you can reason about it, what contrast really measures, and how color vision differs between people. **Part II** explains Kvile, a set of four editor and terminal themes: the idea, the method, the numbers, and the trade-offs, including what is not proven.

You do not need a design background. If you can read a hex code and a short JavaScript function, you can follow all of it. Two kinds of boxes mark the practical parts:

IN PRACTICE

What to do with an idea when you build your own tools or themes.

KVILE DECISION

A choice made in Kvile, and the reason for it.

A note on honesty: Kvile was checked by calculation, using standard formulas and a published simulation of color vision deficiency. It has not yet been tested with a panel of users. Part II is careful to separate what the math guarantees from what it only suggests.

PART I

Color for screens

How displays make color, how eyes read it, and the vocabulary and measurements you need to design with it.

01

Light, not paint

A screen does not reflect color. It makes it. That one fact explains most of what surprises people about color on displays.



Additive color

Paint and ink are *subtractive*: they absorb parts of the light that falls on them, and you see what is left. A screen is *additive*. Each pixel is made of three tiny light sources, red, green and blue, and the color you see is the sum of what they emit. All three at full power give white; all three off give black (on an OLED screen, truly off).

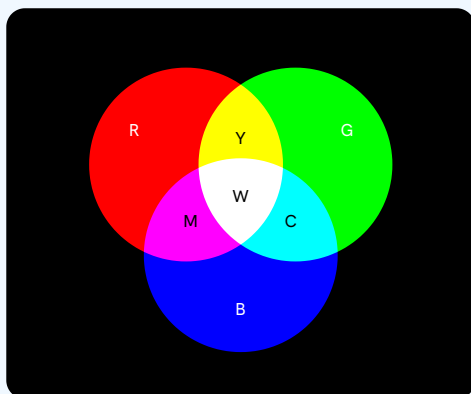


Figure 1. Red, green and blue light adds up. Overlapping pairs give yellow, magenta and cyan, all three give white. This is why mixing colors on a screen does not behave like mixing paint.

A color on a screen is therefore three numbers, one per primary. In CSS and most editors they are written as eight-bit values from 0 to 255, usually as a hex code: `#57a3f7` means red 87, green 163, blue 247. That gives $256 \times 256 \times 256$, about 16.7 million combinations.

The numbers are not light

The number in a hex code is *not* proportional to the amount of light. It is gamma encoded: the standard color space of the web, sRGB, stores values on a curve that is close to a power of 2.2. The reason is historical but still useful, because your eyes are more sensitive to small differences in dark tones than in bright ones, so the encoding spends more of its 256 steps on the dark end.

The consequence is easy to get wrong. The value 128, halfway on the scale, produces only about 22% of the light of 255. If you average two colors by averaging their hex values, or blend them in a naive way, the result is darker and muddier than light would be. Careful software first converts to *linear light*, does the math, and converts back. Every calculation in this book, and every check in Kville, does exactly that.

IN PRACTICE

Treat hex as a storage format, not as a design space. To make a color lighter, rotate it, or blend it, convert it to a perceptual color space first (Chapter 3).

Gamut: not every color fits

The three primaries of sRGB are specific colors, and the triangle they span, the *gamut*, is only part of the colors a human can see. Many current displays, including recent Apple screens, cover a wider gamut called Display P3. Colors outside sRGB cannot be written as ordinary hex codes at all.

When a color you asked for falls outside the gamut you are working in, something has to give. There are three variables to change, lightness, chroma (how colorful) and hue, and the least damaging one to give up is chroma. The color becomes a little less vivid but keeps its lightness, so contrast is unaffected, and keeps its hue, so it still means the same thing. Kville does this everywhere, and stays inside sRGB so that it looks the same on every display.

Your screen is not my screen

Two people looking at the same hex code see different light. Brightness settings, color profiles, night-shift modes, panel technology and the room around you all change what arrives at the eye. A theme cannot control any of that. What it can do is leave *margins*: contrast comfortably above the minimum, and colors that remain distinguishable when the screen is dim, washed out, or tinted warm in the evening. Later chapters show how Kville builds these margins in.

02

How you see color

Color is not a property of an object. It is a conclusion your visual system draws from three kinds of signals.



Rods, cones and three signals

The retina has two families of light sensors. *Rods* work in dim light and see no color. *Cones* work in normal light and come in three types, usually called L, M and S for long, medium and short wavelength. Their sensitivities peak roughly around 560, 530 and 420 nanometers, in the yellow-green, green and violet-blue parts of the spectrum, and they overlap a lot. Color is not a reading of one sensor. The brain compares the three signals, and the comparison gives you what you call a hue.

Very soon after the cones, the signals are recombined into three *opponent channels*: one for light versus dark, one for red versus green, one for blue versus yellow. This is why there is no reddish green and no bluish yellow in ordinary vision, and why many color systems describe a color with one lightness axis and two color axes. Chapter 3 returns to this.

Lightness does the reading

The light-versus-dark channel is the sharp one. Fine detail, edges and the shapes of letters are carried mostly by differences in *luminance*. The color channels have much lower spatial resolution. You can see this in a picture where the color is blurred but the lightness kept sharp: it still looks sharp. Blur the lightness and keep the color sharp, and it falls apart.

For text, that has a direct consequence. A colored word is read first because it is lighter or darker than its background, and only second because it has a hue. Hue helps you sort and group, but it does not make a word legible. This is why Kvile solves each color's *lightness* against the background first, and treats hue as a separate step.

Why pure blue is a bad text color

Two features of the eye make saturated blue particularly poor for small text on a dark background. First, S cones are sparse, and they are essentially absent from the very center of the fovea, the small area used for reading. Fine detail in the blue channel is therefore

resolved poorly. Second, the lens focuses different wavelengths at slightly different distances (longitudinal chromatic aberration), so thin, saturated red or blue strokes can look blurry or seem to sit at a different depth than white text next to them.

The old terminal default of pure #0000ff on black is the textbook example. It is very dark in terms of luminance and very saturated in terms of color, which is the worst combination for reading. Kvile's blue for functions on the Natt background is #57a3f7: lighter, less saturated, and with 7.0:1 contrast.

Context changes color

A color does not look the same on every background. This is *simultaneous contrast*: the same gray square looks lighter on a dark surround and darker on a light one, and a color pushes its neighbors toward its opposite. The eye also *adapts*. After a while in a dark room the eye is tuned to dark, so a bright screen feels glaring; after a while on a white page, a gray looks darker than it is.

IN PRACTICE

Never judge a color on a white page or a swatch chart alone. Judge it on the background and next to the colors it will actually sit with. Kvile checks every accent against the editor background *and* against the selection color, because syntax colors are regularly read on both.

03

Describing color

To reason about color you need a model where the numbers mean something. Hex codes do not. OKLCH does.



RGB and HSL: convenient, but misleading

RGB describes how much of each primary to emit. It is the language of the hardware and not of perception: you cannot tell from the numbers how light a color looks, or how far apart two colors seem.

HSL (hue, saturation, lightness) rearranges the RGB cube into something that feels more intuitive, and it is what many CSS tutorials teach. But its "lightness" is not perceptual lightness. Pure yellow and pure blue both have an HSL lightness of 50%, and one of them is more than ten times brighter than the other:

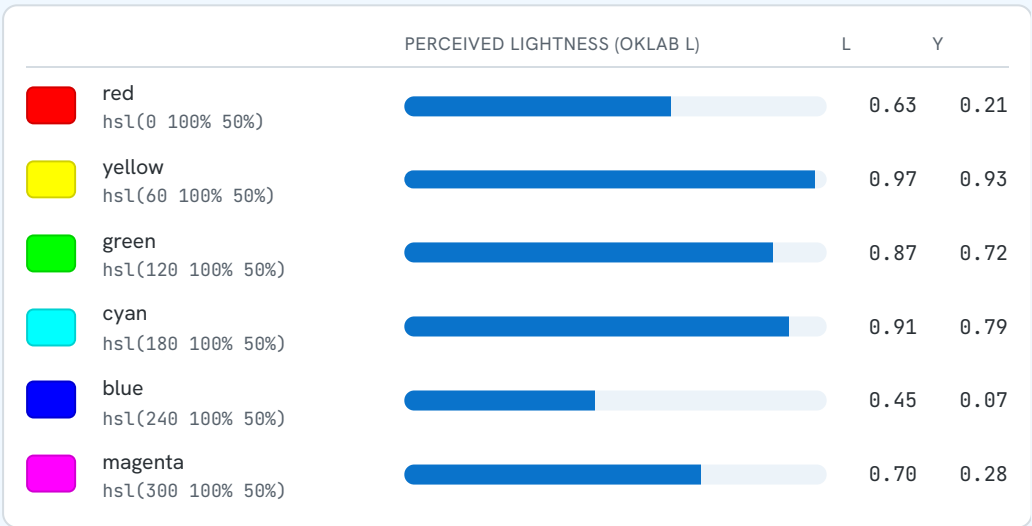


Figure 2. HSL lightness 50% is not equal brightness. Y is the relative luminance (0 to 1) used by contrast formulas, L is OKLab lightness. Yellow is almost 13 times as luminous as blue at the same HSL "lightness".

That is why a palette built by rotating the hue in HSL looks uneven: some colors jump out, others sink into the background.

Perceptual color spaces

A *perceptual* color space is built so that equal numerical steps look like roughly equal steps to a viewer. The classic one is CIELAB (1976), which uses a lightness based on a cube-root curve. It is better than HSL but has known weaknesses, such as blues drifting toward purple when you change lightness.

In 2020 the Swedish engineer Björn Ottosson published **Oklab**, designed to fit current data on perceived lightness, chroma and hue with simple math (two matrix steps and a cube root). It has since been adopted by CSS (the `oklab()` and `oklch()` functions) and by many design tools. The polar form, **OKLCH**, describes a color as:

- **L**, lightness, from 0 (black) to 1 (white);
- **C**, chroma, the colorfulness, from 0 (gray) up to roughly 0.3 at most;
- **h**, hue, an angle in degrees (about 30 is red, 110 yellow, 145 green, 195 cyan, 265 blue, 330 magenta).

HSL, same saturation and lightness (70%, 60%) for every hue



Perceived lightness runs from 0.53 to 0.88. Contrast on the Natt background runs from 3.2:1 to 13.1:1.

OKLCH, same lightness and chroma (0.76, 0.115) for every hue



Perceived lightness runs from 0.76 to 0.76. Contrast on the Natt background runs from 8.1:1 to 9.0:1.

Figure 3. Two strips of 24 hues. Top: HSL values held constant. Bottom: OKLCH lightness and chroma held constant. The bottom strip looks even because its lightness really is nearly even. Where a color does not fit in sRGB, chroma is reduced.

Oklab also gives a usable measure of *difference*: the straight-line distance between two colors, often written ΔE . Scaled by 100, a difference of about 2 is just noticeable for two flat patches side by side. Kvile uses this distance to make sure that syntax colors can be told apart (Chapter 8).



Which space for what

MODEL	WHAT "LIGHTNESS" MEANS	GOOD FOR
RGB / hex	Nothing. Not a lightness.	Storing and sending colors, what the hardware wants.
HSL	An arithmetic average of channels. Not perceptual.	Quick tweaks. Not for building palettes.
CIELAB	Perceptual, cube root of luminance.	Print workflows, color difference in older tools.
Oklab / OKLCH	Perceptual, with better hue behavior.	Designing palettes, ramps and accessible color systems.

IN PRACTICE

Design in OKLCH, store as hex. Pick a lightness first (it decides contrast), then a chroma (it decides how loud), then a hue (it decides what it means).

04

Contrast

The most important number in a theme, and the most misunderstood one.



What the number measures

The contrast ratio in the Web Content Accessibility Guidelines (WCAG 2.x) is built from *relative luminance* Y , the amount of light a color emits compared with white, weighted by how sensitive the eye is to each primary: $Y = 0.2126 R + 0.7152 G + 0.0722 B$, computed on linear values. Green carries most of the weight, blue almost none, which is the fact behind the poor legibility of blue text.

The ratio between two colors is then

$$(Y_{\text{lighter}} + 0.05) / (Y_{\text{darker}} + 0.05)$$

It runs from 1:1 (identical) to 21:1 (black on white). The constant 0.05 stands for a small amount of stray light (screen reflections, flare in the eye). It stops the ratio from exploding for very dark colors.

RATIO	WCAG ROLE
3:1	Minimum for large text and for user interface components and graphics (success criterion 1.4.11).
4.5:1	Minimum for normal body text, level AA (1.4.3).
7:1	Enhanced contrast for normal text, level AAA (1.4.6).

<code>const rest = true;</code>	1.5:1	
<code>const rest = true;</code>	3.0:1	large text, UI
<code>const rest = true;</code>	4.5:1	body text, AA
<code>const rest = true;</code>	7.0:1	AAA
<code>const rest = true;</code>	10.0:1	
<code>const rest = true;</code>	12.8:1	Kvile Natt text
<code>const rest = true;</code>	16.0:1	

Figure 4. The same line of code in gray on the Natt background, at increasing contrast. Kvile Natt body text sits at 12.8:1; the top step is close to the maximum possible on this background.

A floor, and also a ceiling

WCAG treats contrast as a floor: more is always acceptable. For a screen you stare at all day, that is too simple. Maximum contrast, pure white on pure black, is bright and hard-edged. Many people with astigmatism report that bright text on a black background seems to bleed or glow, an effect often called halation, and large bright areas in a dark room are tiring because the eye has adapted to the dark. Research on reading shows that, in general, more luminance helps performance, so there is no hard physiological limit to cite. A ceiling is a *comfort design choice*, not a measured threshold.

KVILE DECISION: TEXT HAS A TARGET WINDOW

Kvile sets a contrast *range* for body text, not just a minimum: 7-13:1 for Natt (actual 12.8:1), 5.5-10:1 for Skumring (9.1:1), 7-14:1 for Dag (12.6:1) and 12-21:1 for Kontrast (17.7:1). The generator solves the text color so that it lands inside the window, and the build fails if it does not.

Limits of the WCAG 2 formula

The WCAG 2 ratio is symmetric (black on white and white on black give the same number), and it ignores font size, weight and the fact that light-on-dark and dark-on-light text are perceived differently. Newer models, such as the Accessible Perceptual Contrast Algorithm (APCA), try to account for polarity and spatial frequency. APCA is under discussion for a future WCAG 3, which is not final.

KVILE DECISION: WCAG 2.X AS THE YARDSTICK

Kvile uses the WCAG 2 ratio because it is the standard that every audit tool understands, anyone can reproduce the numbers, and a build can check it with one function (Chapter 12 lists it in full). The ratio is used as a floor for every color and as a ceiling for body text, and the choices around it are made explicit rather than hidden in a more elaborate model.

Comments are not decoration

A common fashion is to fade comments to a dim gray, often below 3:1. But comments carry design rationale, warnings and TODOs, and are read constantly. Kvile keeps comments at a measured floor in each variant: $\geq 4.5:1$ in Natt, $\geq 3.5:1$ in Skumring, $\geq 4.5:1$ in Dag and $\geq 7:1$ in Kontrast. They look quieter than code through a cooler hue and lower chroma, and through italics in editors that support it, but not through low legibility.

05

Color vision differences

Roughly one in twelve men and one in two hundred women have a red-green color vision deficiency. A theme that ignores that is wrong for a large group of its readers.



What differs

Color vision deficiency (CVD) is caused by cones that are missing or whose sensitivity is shifted. Deficiencies are named after the cone type affected: **protan** (L cones), **deutan** (M cones) and the rarer **tritan** (S cones). In the strong form, *dichromacy*, one cone type is missing (protanopia, deuteranopia, tritanopia). In the more common weak form, *anomalous trichromacy*, the cone type is present but shifted (protanomaly, deuteranomaly, tritanomaly). Prevalence figures vary by population, but the often quoted numbers for people of Northern European descent are about 8% of men and 0.5% of women with red-green deficiency; worldwide, estimates come to around 200 million people.

What matters for design is what *collapses*. For a deutan or protan viewer, many reds, oranges, browns and greens, and some purples and blues, land on the same axis and become hard to tell apart. Lightness differences, and to a degree differences in how blue or yellow a color is, remain. The channel that carries them is the light-dark one, which is unaffected.

Simulating it

Gustavo Machado, Manuel Oliveira and Leandro Fernandes published a physiologically based model of color vision deficiency in 2009 (IEEE Transactions on Visualization and Computer Graphics). It simulates a deficiency with a single 3×3 matrix applied to a color in linear RGB, with a severity parameter from 0 (typical vision) to 1 (dichromacy), and it handles anomalous trichromacy as well as dichromacy. Kville uses the matrices at **severity 1.0**, the strongest form. Designing for the worst case also covers the milder ones.

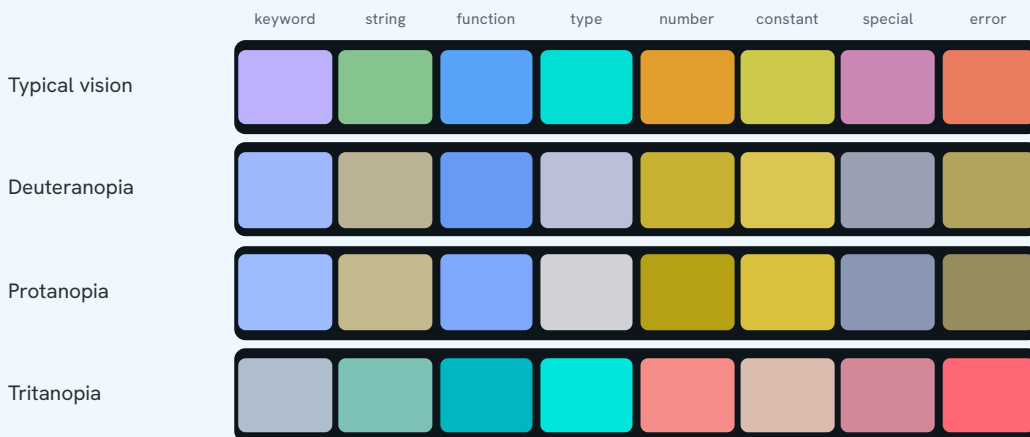


Figure 5. The eight Kvile Natt accent colors as seen with typical vision and, by simulation, with deuteranopia, protanopia and tritanopia. Each color keeps a distinct lightness or yellowness, even where hue collapses.

The design rule

If two syntax roles differ *only* by hue, a CVD reader may see one color for both. They must also differ in lightness or chroma, enough that the difference survives each simulated deficiency. A palette where every accent is rendered at exactly the same lightness looks harmonious and is a trap: under protanopia or deuteranopia, a lot of it merges.

IN PRACTICE

Do not encode meaning in hue alone. Test your palette with a simulator, or with the matrices above, and measure the distance between each pair, not just whether it "looks fine" to you. Use a second channel for anything critical: an icon, a label, bold or italic, position.

06

Color in code

What syntax highlighting can and cannot do, and what the research says about dark mode, light mode and tired eyes.



Does syntax highlighting help?

Intuition says yes, and developers overwhelmingly use it. Research is more modest. Advait Sarkar (2015, Psychology of Programming Interest Group) ran a controlled within-subjects experiment with 10 participants and eye tracking, and found that syntax coloring significantly shortened the time to complete comprehension tasks, with a smaller benefit for more experienced programmers. In a much larger experiment, Hannebauer, Hesenius and Gruhn (2018, *Empirical Software Engineering*) tested 390 undergraduate Java students and found no evidence that syntax highlighting improved novices' comprehension of source code.

The results are not contradictory, since the tasks, participants and sample sizes differ, but they tell us not to overstate the claim. Color is a helpful sorting layer, not a magic one. Which brings the design target into focus: a theme should be *cheap to live with*. If its benefit is modest, its cost over eight hours (glare, noise, confusion between categories) must be close to zero.

Fewer categories, used consistently

The highlighting layer works only as long as you can remember what each color means. Kville uses **eight roles**: keywords, strings, functions, types, numbers, constants, special (regex, escapes, decorators) and errors. The same eight hues carry the same meaning in the editor, the terminal's sixteen ANSI colors, git diffs, prompts and status messages, so that red means "wrong" and green means "fine" in all of them.

Light or dark?

Studies on reading generally find a *positive polarity advantage*: dark text on a light background gives better performance in proofreading and acuity tasks than light text on dark. Piepenbrock, Mayr and Buchner (2013, 2014) found it in younger and older adults, and found it was larger for small characters. A frequently proposed explanation is pupil size: a bright display narrows the pupil, which reduces optical blur. Earlier work by Buchner, Mayr

and Brandt (2009) found that performance rises with display luminance for either polarity, which suggests that background brightness matters more than which side the text is on. The Nielsen Norman Group's review of this literature adds that fatigue measures did not differ significantly between polarities, and that the advantage can vanish in bright ambient light.

The honest summary is that there is no clear winner. Light themes tend to help small text and bright rooms. Dark themes are easier on large bright areas in dim rooms, and some people find them calmer. Some people with cataract or other cloudy ocular media may read better on dark. A mismatch between the screen and the room, for example a white window in a dark room, is the part you can actually fix. That is the reason Kville ships **four** variants in a range, and not a single "best" theme.

What about blue light?

Bright light in the evening, especially light rich in short wavelengths, can delay the body clock and the onset of sleep; that is well established for strong sources. Whether the small amount from a code editor matters is much less clear, and it is easy to oversell. Kville makes no claim to filter blue light or protect your retina. What it does is moderate: lower overall luminance in the dark variants, a ceiling on text contrast, and no large pure white areas.

A theme cannot replace a break

Screen work reduces how often you blink, and sustained near focus tires the eye muscles. The most reliable remedies are not about color: look away from the screen regularly, keep the screen about an arm's length away, and match the screen's brightness to the room. A calm theme removes one avoidable source of strain. It does not remove the others.

PART II

Kvile

Kvile is Norwegian for "to rest". This part explains the idea, the method, the numbers, and the choices behind four themes for editors and terminals.

07

The idea behind Kvile

Many themes are designed to look good in a screenshot. Kvile is designed to feel good at hour eight.



Five principles

1. **Calm.** Moderate chroma, a ceiling on text contrast, and no pure black or pure white. A color has to earn its loudness. Only errors are allowed to be loud, and they are loud through hue, not brightness.
2. **Legible.** Every color, not only the text, has a measured contrast floor against the background and against the selection color.
3. **Inclusive.** Every pair of syntax colors stays distinguishable under simulated protanopia, deuteranopia and tritanopia.
4. **Coherent.** One palette, eight meanings, the same in the editor, the terminal, the shell prompt, the diff and the browser. Every port is generated from one file, so they cannot drift apart.
5. **Verifiable.** The palette is not drawn, it is computed. If a check fails, the build fails.

Four variants for four situations

The names are Norwegian: **Natt** (night), **Skumring** (dusk), **Dag** (day) and **Kontrast** (contrast). They share hues and meanings and differ in how light the page is and how much contrast it asks of your eyes.

VARIANT	MODE	BACKGROUND	TEXT CONTRAST	ACCENT FLOOR	COMMENT FLOOR
Natt night	dark	 #0d151b	12.8:1	4.5:1	≥ 4.5:1
Skumring dusk	dark	 #1a232b	9.1:1	4:1	≥ 3.5:1
Dag day	light	 #f0f8ff	12.6:1	4.5:1	≥ 4.5:1
Kontrast high contrast	dark	 #050a0f	17.7:1	7:1	≥ 7:1

- **Natt** is the reference: a dark blue-gray, balanced contrast, the one the others are measured against.

- **Skumring** is a lighter dark, with deliberately softer contrast, for dim rooms or for people who find Natt too sharp. It is the one variant that trades some legibility margin for softness, and its lower floors are written down.
- **Dag** is a light theme in a cool off-white, for bright rooms and small text.
- **Kontrast** is the accessibility variant: the darkest background, the highest text contrast, and every accent at 7:1 or more.

KVILE DECISION: A COOL, TINTED NEUTRAL

Instead of a neutral gray, every background and surface carries a small amount of blue (hue 245, chroma 0.012 to 0.02). A tinted neutral sits more naturally beside colored accents, and the shift is far below what you consciously notice. The cool hue is also the one nearest to the Nordic feel of the name.

08

Building the palette

A step-by-step account of how Kvile is computed: color space, neutrals, text, accents, tuning and the terminal colors.



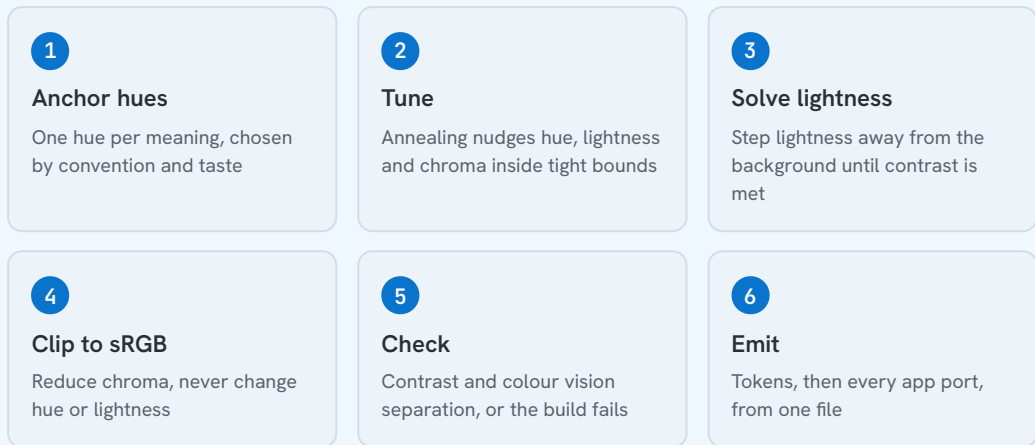


Figure 6. From anchor hues to every app, in one script. All steps run in OKLCH and are deterministic: the same inputs always give the same palette.

Step 1: work in OKLCH

Everything is designed in OKLCH and converted to sRGB hex only at the end, using the conversion in Chapter 12. When a color is outside the sRGB gamut, chroma is reduced in steps of 0.002 until it fits; lightness and hue are never touched, so contrast and meaning survive.

Step 2: neutrals

The background is a single OKLCH color: hue 245, a small chroma, and a lightness per variant (Natt 0.19, Skumring 0.25, Dag 0.975, Kontrast 0.14). The other neutrals come from a 12-step ramp between the background lightness and the text lightness, eased so that the first steps stay close to the background:

$$L(t) = L_{bg} + (L_{fg} - L_{bg}) \cdot t^{1.6}, \quad t = 0 \dots 1 \text{ in 12 steps, chroma reduced by 45\% toward the end}$$

The exponent 1.6 bunches the early steps near the background. Step 1 becomes *surface* (panels, sidebars), step 2 *raised* (cards, inputs, inactive tabs) and step 3 *border*. Subtle layers are needed far more often than strong ones, and an even ramp would give you too few of them.

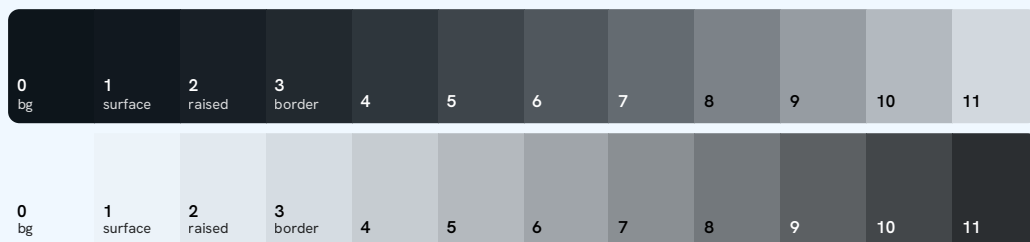


Figure 7. The 12-step neutral ramp for Natt (top) and Dag (bottom). Steps 1, 2 and 3 are the surface, raised and border colors.

The *selection* color is computed separately: lightness moved 0.1 toward the text from the background, chroma 0.045, hue 250. It is more colorful than the rest of the neutrals, because selected text sits on it and the highlight needs to be seen at a glance. The build then requires that text on selection keeps at least 4.5:1, and that even the weakest accent keeps at least 3:1.

Step 3: text and comments

Text and comments are found by search, not by hand. Starting from the target lightness, the generator steps the lightness away from the background in increments of 0.004 until the contrast target is met, then, for text, steps back if the ceiling is exceeded. The comment color uses hue 232 with chroma 0.03, slightly more blue-green than text so that it reads as a different kind of text rather than as a dimmer one. Muted text (labels, line numbers) is the average between the text and comment targets.

Step 4: eight accent roles

The eight accents each have an anchor hue chosen by convention and taste: purple for keywords, green for strings, blue for functions, cyan for types, orange for numbers, yellow for constants, pink for special syntax and red-orange for errors. They share a starting

lightness and chroma per variant, then each role gets a small lightness nudge (named **dL** in the generator), because colors of equal lightness give a CVD reader no way to separate them.

ROLE	ANCHOR HUE	USED FOR	TOLERANCE	TUNED HUE, ACROSS THE FOUR VARIANTS
 keyword	300°	keywords, control flow	±25°	279° to 316°
 string	152°	strings, success	±20°	141° to 172°
 function	245°	functions, links, focus	±20°	243° to 252°
 type	200°	types, classes	±25°	175° to 220°
 number	60°	numbers, warnings in prompts	±15°	48° to 75°
 constant	98°	constants, booleans, warnings	±15°	83° to 109°
 special	345°	regex, escapes, decorators	±20°	329° to 344°
 error	25°	errors, deletions	±10°	15° to 35°

The final lightness of each accent is then *solved*: starting from the variant's accent lightness plus the role's nudge, the generator steps lightness away from the background until the contrast floor for that variant is met (if it is not already) (4.5:1 in Natt and Dag, 4:1 in Skumring, 7:1 in Kontrast).

Step 5: tuning by simulated annealing

With anchor hues alone, some pairs are too close for comfort. The figure below shows how close: with no tuning, the closest pair of colors is only about 1.6 to 2.0 apart in OKLab units under deuteranopia, in all four variants. That is at the edge of "just noticeable", while Kvile requires 6.

The problem is small but has many constraints: 8 roles give 28 pairs, and each pair must be separated under 4 conditions (typical, deuteranopia, protanopia, tritanopia), so there are 112 distance constraints, and each role has three free parameters (hue, lightness nudge, chroma nudge). Kville solves it with **simulated annealing**:

1. Start from the anchor values. Compute the *cost*: the sum over all pairs and conditions of the squared shortfall below the required distance plus a margin of 0.6 (so that rounding to hex cannot push a pair below the limit), plus a weak penalty for drifting away from the anchor hue and lightness.
2. For 6000 iterations, pick one role at random and perturb its hue (within the tolerance above), lightness nudge (± 0.055) and chroma nudge.
3. Accept the move if it lowers the cost. If it raises it, accept it anyway with a probability that shrinks as the "temperature" falls, which lets the search escape local minima.
4. Keep the best solution found.

The random number generator is seeded, so the result is the same every time; the palette is a build artifact, not a lucky draw. The required distances are **10** for typical vision and **6** for each deficiency.

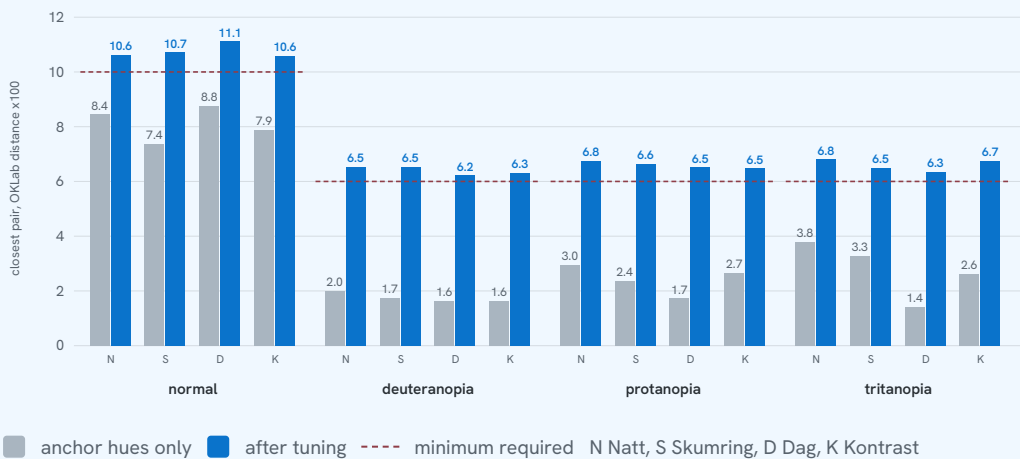
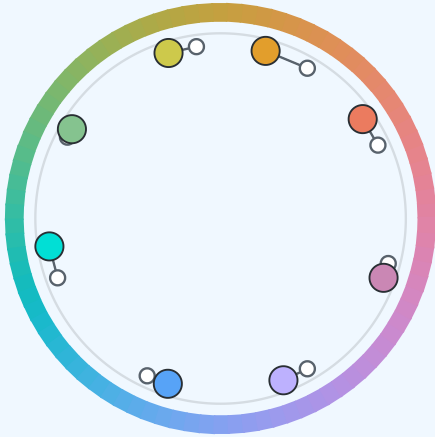
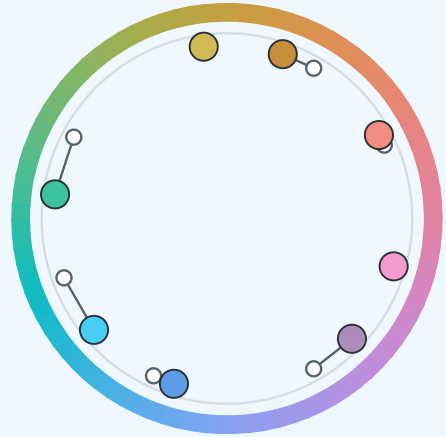


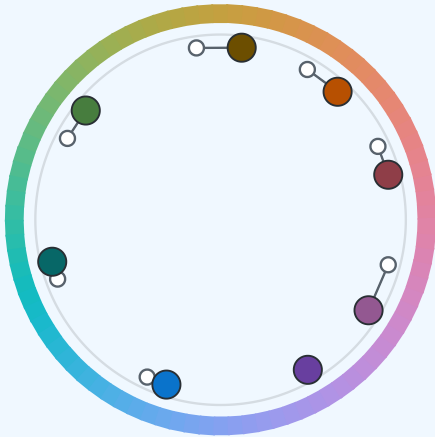
Figure 8. Closest pair of accents, before and after tuning, for each variant and each kind of vision. Dashed lines mark the required minimum. Without tuning, every check fails, typical vision included; with it, all pass.



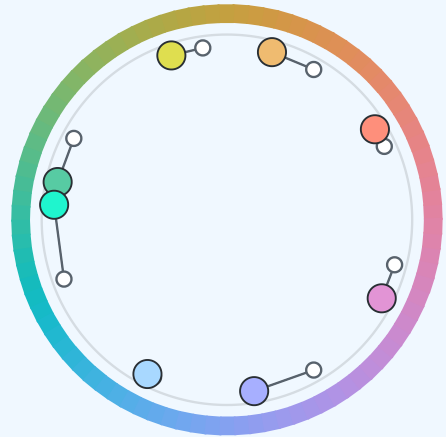
Kvile Natt



Kvile Skumring



Kvile Dag



Kvile Kontrast

Figure 9. Anchor hues (open rings) and tuned hues (filled dots) on the hue circle, for the four variants. Tuning moves a hue by a few degrees to at most the tolerance, so purple stays purple and red-orange stays red-orange. The colored ring is hue at constant lightness and chroma.

Step 6: the terminal and the rest

The sixteen ANSI terminal colors are derived from the roles. Red is the error color, green the string color, yellow the constant color, blue the function color, magenta the keyword color, cyan the type color. Bright variants are the same hue and chroma with lightness shifted by 0.06 away from the background. The "black" slot is the raised surface, "white" is the muted

text, bright black is the comment color and bright white the text color. Status colors reuse the roles: error, warning (constant), success (string) and info (function). Because everything in the terminal comes from the same eight roles, tools like git, bat, delta, fzf or a shell prompt automatically agree with your editor.

09

The four variants

The same syntax in each, from one set of rules.





Figure 10. One TypeScript sample in all four variants. The highlighted line shows the selection color; note that the text remains readable on it.

Natt



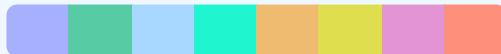
Skumring



Dag



Kontrast



How they differ

Natt has a background lightness of 0.19 and accents at about 0.76. It is the anchor of the family: text at 12.8:1, accents at 6.6:1 or more. **Skumring** raises the background to 0.25 and lowers accent lightness to about 0.74 with less chroma (0.095 against 0.115 in Natt), which brings accents closer to the background and the whole picture closer to a single tone. **Dag** inverts the problem: on a background of 0.975, accents must be pulled down to about 0.5 to

reach 4.5:1, which makes them darker and calmer than a light theme's usual pastels. **Kontrast** has the darkest background (0.14), almost white text (0.96) and accents lifted to 0.82 so that every one has at least 7:1.

KVILE DECISION: FOUR VARIANTS, ONE MEANING

A color keeps the same *role* in every variant, but it does not have the same *value*. Natt blue and Dag blue are different numbers, because the same blue cannot satisfy contrast on a dark and a light page. This is the reason a theme cannot simply be inverted to make its light twin.

10

Verification

Every number that was promised, and the results of checking it.



What is checked

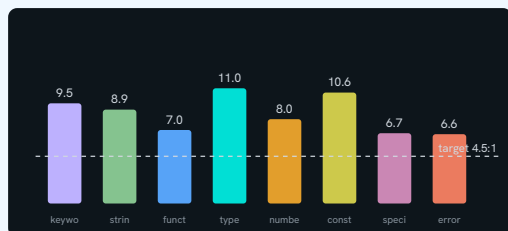
Every build of the palette runs these checks, and exits with an error if any one fails:

- **Text contrast** must land inside the variant's window (Chapter 4).
- **Comment contrast** must be above its variant's floor.
- **Each of the eight accents** must meet the variant's accent floor against the background.
- **Text on selection** must be at least 4.5:1, and the **weakest accent on selection** at least 3:1.
- **Syntax color separation:** the closest pair of accents must be at least 10 apart for typical vision and 6 apart for each of the three simulated deficiencies (OKLab distance × 100).

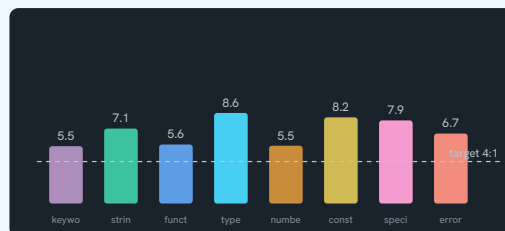
All four variants pass all checks. The closest pairs, which show how much room is left:

VISION	NATT	SKUMRING	DAG	KONTRAST	REQUIRED
normal	10.6 number / constant	10.7 number / error	11.1 keyword / special	10.6 number / constant	10
deuteranopia	6.5 number / constant	6.5 type / special	6.2 string / number	6.3 number / error	6
protanopia	6.8 keyword / function	6.6 keyword / function	6.5 number / constant	6.5 keyword / special	6
tritanopia	6.8 number / special	6.5 string / function	6.3 string / function	6.7 special / error	6

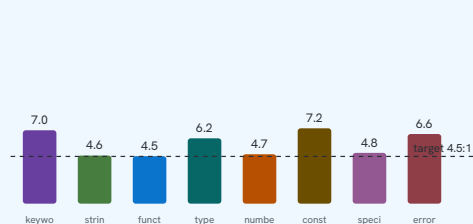
Kvile Natt



Kvile Skumring



Kvile Dag



Kvile Kontrast



Figure 11. Contrast of each accent against the background, per variant. The dashed line is the floor for that variant. The floor is a minimum, not a goal. In Natt, Skumring and Kontrast the starting lightness already clears it with room to spare; in Dag, several accents land just above 4.5:1 because a light background leaves little room.

Reading a distance matrix

The two matrices show the distance between every pair of Natt accents. Under typical vision, no pair is below 10. Under deuteranopia, the entire matrix shrinks, and a few pairs approach the threshold of 6. The tuning step of Chapter 8 exists to keep that closest pair at or above 6.

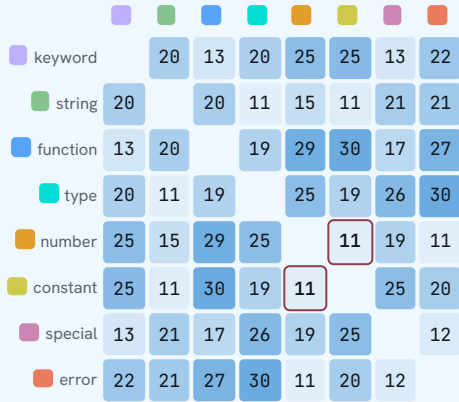


Figure 12. Natt, typical vision. Cell value: OKLab distance $\times$ 100 (threshold 10). The darkest outlined cell is the closest pair.

The full check tables are in Appendix B.

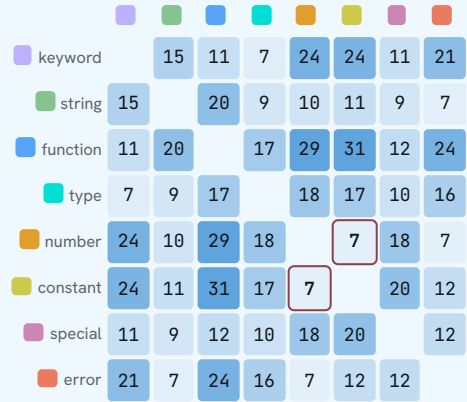


Figure 13. Natt, simulated deuteranopia (threshold 6). Colors that were far apart move closer together, as expected.

11

Decisions, trade-offs and limits

No palette is free. Here is what Kvile costs, why it was chosen anyway, and where the evidence ends.

Decisions

WHY OKLCH?

HSL is convenient but its lightness is not perceptual (Chapter 3), so a rotated palette is uneven. CIELAB has better lightness but drifts in blue. Oklab has simple math, good hue behavior and a built-in distance, and it is now part of CSS.

WHY SEPARATE THRESHOLDS FOR CVD?

Typical vision needs 10; each deficiency needs 6. The simulation at severity 1.0 removes a color dimension, so distances inevitably shrink, and requiring the same 10 would force unnatural colors. A distance of 6 is still about three times a just-noticeable difference. These thresholds are Kvile design choices, not standards.

WHY DOES TEXT HAVE A CEILING?

A comfort decision, not a measured limit (Chapter 4). Natt text is 12.8:1 and Kontrast, the exception that exists for people who want more, is 17.7:1.

WHY DOES SKUMRING ACCEPT LOWER FLOORS?

Skumring is deliberately soft. It accepts a 4:1 accent floor and 3.5:1 for comments. The weakest accent on selection is 3.88:1, close to its 3:1 floor. People who need maximum legibility have Natt and Kontrast.

WHY DO THE TUNED HUES DRIFT?

To keep colors separable under deficiency, hues move a few degrees from their anchors (Natt keyword: 300° to 291°, error: 25° to 35°). The tolerance bounds the drift so that no role changes its identity.

WHY IS DAG SO DARK IN ITS ACCENTS?

On a near-white background, an accent reaches 4.5:1 only at about half lightness. A pastel is simply not an option if text must be legible. The reward is that Dag accents look closer to ink than to highlighter.

What is not proven

- **No user study yet.** Everything here is checked by calculation. The formulas are standard, but real eyes on real screens in real rooms are a different test, and a study with typical and color-deficient participants is still to be done.
- **CVD simulation is a model.** It is a published and widely used one, but individuals differ, and severity 1.0 is a limiting case.
- **WCAG 2 contrast ignores polarity and font weight.** The ceilings and floors in Kvile are therefore a mix of standards and judgment.
- **Distance thresholds are choices.** A distance of 2 is a just-noticeable difference for flat patches; it is not a statement that 6 or 10 suffices for every reader at every glyph size.
- **Apps differ.** Each port maps the eight roles onto what that app exposes. Some apps use fewer scopes, or tint things themselves, so the same palette can look different between two programs.
- **sRGB only.** Kvile stays inside sRGB for predictability. A Display P3 edition could use more chroma at the same contrast.

What would improve it

A study with a mixed panel, including readers with CVD and astigmatism; an APCA-based check next to the WCAG 2 one; a Display P3 edition for supporting displays; and per-app tuning once each port has been used for a while in the real application.

12

Using this in your own work

You do not need Kvile to use its method. Here is the recipe, with the code.



A checklist for any palette

1. Choose colors in OKLCH. Set lightness first, then chroma, then hue.
2. Decide the background, then the text, and set a contrast *window* for the text, not only a minimum.
3. Solve every other color's lightness against the background, not by eye.
4. Check each color on every background it will meet, including selection, hover and zebra rows.
5. Give each meaning its own lightness, not only its own hue.
6. Run all colors through a CVD simulation and measure the distance between pairs.
7. Clip to the gamut by reducing chroma, never by changing lightness or hue.
8. Generate every app's theme from one source of truth, and make the build fail when a check fails.
9. Keep the neutrals tinted by a small amount, and the accents below pastel and above neon.
10. Look at it for a day. Metrics do not replace living with a theme.

The two functions you need

These two functions are the whole mathematical core of Kvile's checks. The first computes the WCAG contrast ratio, the second converts OKLCH to a hex code and reduces chroma when a color falls outside sRGB. Both reproduce Kvile's generator, and both run as they are in Node or a browser.

Kvile Natt

```
1 // WCAG 2.x contrast ratio between two sRGB colors, each given as "#rrggbb"
2 const toLinear = (v) => (v <= 0.04045 ? v / 12.92 : ((v + 0.055) / 1.055) ** 2.4);
3
4 function luminance(hex) {
5   const n = parseInt(hex.slice(1), 16);
6   const [r, g, b] = [n >> 16, (n >> 8) & 255, n & 255].map((v) => toLinear(v / 255));
7   return 0.2126 * r + 0.7152 * g + 0.0722 * b;
8 }
9
10 export function contrast(a, b) {
11   const [hi, lo] = [luminance(a), luminance(b)].sort((x, y) => y - x);
12   return (hi + 0.05) / (lo + 0.05);
13 }
14
15 // OKLCH to hex. Outside sRGB, lower chroma and keep lightness and hue.
16 const toGamma = (v) => (v <= 0.0031308 ? 12.92 * v : 1.055 * v ** (1 / 2.4) - 0.055);
17
18 export function oklchToHex(L, C, h) {
19   for (let c = C; c >= 0; c -= 0.002) {
20     const a = c * Math.cos((h * Math.PI) / 180);
21     const b = c * Math.sin((h * Math.PI) / 180);
22     const l = (L + 0.3963377774 * a + 0.2158037573 * b) ** 3;
23     const m = (L - 0.1055613458 * a - 0.0638541728 * b) ** 3;
24     const s = (L - 0.0894841775 * a - 1.291485548 * b) ** 3;
25     const rgb = [
26       4.0767416621 * l - 3.3077115913 * m + 0.2309699292 * s,
27       -1.2684380046 * l + 2.6097574011 * m - 0.3413193965 * s,
28       -0.0041960863 * l - 0.7034186147 * m + 1.707614701 * s,
29     ];
30     if (rgb.every((v) => v >= -1e-4 && v <= 1 + 1e-4)) {
31       const bytes = rgb.map((v) => toGamma(Math.min(1, Math.max(0, v))) * 255);
32       return '#' + bytes.map((v) => Math.round(v).toString(16).padStart(2,
33         '0')).join('');
34     }
35   }
36   return '#808080';
37 }
```

Using Kvile's tokens

Kvile's generator writes `dist/tokens.json` and `dist/kvile.css`. In CSS, a variant is a data attribute, and every color is a variable:

Kvile Natt

```
1  /* kvile.css, generated */
2  [data-kvile="natt"] {
3    --kvile-bg: #0d151b;
4    --kvile-fg: #d1d8df;
5    --kvile-function: #57a3f7;
6  }
7
8  .editor {
9    background: var(--kvile-bg);
10   color: var(--kvile-fg);
11 }
12 .editor a { color: var(--kvile-function); }
```

Because the variants share names, switching theme is a single attribute change, and any component you build with the variables inherits the checked contrast.

A

Appendix A: the palettes

All values, per variant. L, C and h are OKLCH. Contrast is against each variant's background.

Kvile Natt night

	NEUTRAL	HEX	CONTRAST ON BG			
	bg	#0d151b	1.0:1			
	surface	#11181f	1.0:1			
	raised	#181f26	1.1:1			
	border	#22292f	1.3:1			
	selection	#192d41	1.3:1			
	fg	#d1d8df	12.8:1			
	muted	#87929c	5.8:1			
	comment	#6e838f	4.7:1			

	ACCENT	HEX	L	C	H	CONTRAST ON BG
	keyword	#bdb1fe	0.798	0.125	291°	9.5:1
	string	#85c38f	0.761	0.098	149°	8.9:1
	function	#57a3f7	0.705	0.145	252°	7.0:1
	type	#01dfd5	0.815	0.145	189°	11.0:1
	number	#e29e2c	0.749	0.145	75°	8.0:1
	constant	#cdc94b	0.815	0.145	107°	10.6:1
	special	#ca87b4	0.705	0.100	340°	6.7:1
	error	#eb7b5f	0.705	0.145	35°	6.6:1

Kvile Skumring dusk

	NEUTRAL	HEX	CONTRAST ON BG
	bg	#1a232b	1.0:1
	surface	#1d262d	1.0:1
	raised	#232c33	1.1:1
	border	#2c343c	1.3:1
	selection	#283c51	1.4:1
	fg	#bec5cb	9.1:1
	muted	#85919b	4.9:1
	comment	#687d89	3.7:1

	ACCENT	HEX	L	C	H	CONTRAST ON BG
	keyword	#ac8cb9	0.685	0.075	316°	5.5:1
	string	#3dc2a0	0.735	0.125	172°	7.1:1
	function	#5d9de5	0.685	0.125	252°	5.6:1
	type	#46cef3	0.793	0.125	220°	8.6:1
	number	#c88c3b	0.685	0.120	71°	5.5:1
	constant	#d0ba53	0.788	0.125	98°	8.2:1
	special	#f49bd0	0.795	0.124	344°	7.9:1
	error	#f18c7e	0.745	0.125	29°	6.7:1

Kvile Dag day

	NEUTRAL	HEX	CONTRAST ON BG			
	bg	#f0f8ff	1.0:1			
	surface	#ecf3f9	1.0:1			
	raised	#e2e9ef	1.1:1			
	border	#d5dce2	1.3:1			
	selection	#c0d9f3	1.4:1			
	fg	#292f34	12.6:1			
	muted	#58626c	5.8:1			
	comment	#5d727d	4.7:1			

	ACCENT	HEX	L	C	H	CONTRAST ON BG
	keyword	#683f9f	0.465	0.150	300°	7.0:1
	string	#477d3e	0.535	0.110	141°	4.6:1
	function	#0a73cb	0.550	0.160	252°	4.5:1
	type	#076767	0.467	0.124	194°	6.2:1
	number	#b75001	0.555	0.152	48°	4.7:1
	constant	#6c4e00	0.445	0.126	83°	7.2:1
	special	#935891	0.547	0.110	329°	4.8:1
	error	#8f3e48	0.475	0.110	15°	6.6:1

Kvile Kontrast high contrast

	NEUTRAL	HEX	CONTRAST ON BG			
	bg	#050a0f	1.0:1			
	surface	#070e13	1.0:1			
	raised	#0f151b	1.1:1			
	border	#1a2126	1.2:1			
	selection	#0d2034	1.2:1			
	fg	#ebf3fa	17.7:1			
	muted	#a9b5c0	9.5:1			
	comment	#8aa0ac	7.3:1			

	ACCENT	HEX	L	C	H	CONTRAST ON BG
	keyword	#a7afff	0.778	0.160	279°	9.7:1
	string	#57cba3	0.765	0.122	167°	9.9:1
	function	#a8d8ff	0.863	0.132	243°	13.2:1
	type	#1ff5ce	0.868	0.160	175°	14.2:1
	number	#efbb70	0.822	0.110	75°	11.4:1
	constant	#dfde4f	0.875	0.160	109°	13.9:1
	special	#e294d5	0.765	0.126	333°	8.9:1
	error	#fe8f7b	0.765	0.142	31°	8.9:1

B

Appendix B: check results

The complete output of the build's checks.

Kvile Natt

CHECK	VALUE	RULE	
text on background	12.8:1	7-13:1	pass
comment on background	4.65:1	$\geq$ 4.5:1	pass
text on selection	9.78:1	$\geq$ 4.5:1	pass
keyword on background	9.55:1	$\geq$ 4.5:1	pass
string on background	8.95:1	$\geq$ 4.5:1	pass
function on background	7.01:1	$\geq$ 4.5:1	pass
type on background	10.98:1	$\geq$ 4.5:1	pass
number on background	8.03:1	$\geq$ 4.5:1	pass
constant on background	10.57:1	$\geq$ 4.5:1	pass
special on background	6.7:1	$\geq$ 4.5:1	pass
error on background	6.6:1	$\geq$ 4.5:1	pass
weakest accent on selection	5.04:1	$\geq$ 3:1	pass

VISION	CLOSEST PAIR	DISTANCE	MIN	
normal	number / constant	10.6	10	pass
deuteranopia	number / constant	6.5	6	pass
protanopia	keyword / function	6.8	6	pass
tritanopia	number / special	6.8	6	pass

Kvile Skumring

CHECK	VALUE	RULE	
text on background	9.13:1	5.5-10:1	pass
comment on background	3.7:1	≥ 3.5:1	pass
text on selection	6.49:1	≥ 4.5:1	pass
keyword on background	5.46:1	≥ 4:1	pass
string on background	7.14:1	≥ 4:1	pass
function on background	5.62:1	≥ 4:1	pass
type on background	8.63:1	≥ 4:1	pass
number on background	5.51:1	≥ 4:1	pass
constant on background	8.21:1	≥ 4:1	pass
special on background	7.92:1	≥ 4:1	pass
error on background	6.67:1	≥ 4:1	pass
weakest accent on selection	3.88:1	≥ 3:1	pass

VISION	CLOSEST PAIR	DISTANCE	MIN	
normal	number / error	10.7	10	pass
deuteranopia	type / special	6.5	6	pass
protanopia	keyword / function	6.6	6	pass
tritanopia	string / function	6.5	6	pass

Kvile Dag

CHECK	VALUE	RULE	
text on background	12.63:1	7-14:1	pass
comment on background	4.7:1	≥ 4.5:1	pass
text on selection	9.33:1	≥ 4.5:1	pass
keyword on background	6.99:1	≥ 4.5:1	pass
string on background	4.58:1	≥ 4.5:1	pass
function on background	4.52:1	≥ 4.5:1	pass
type on background	6.22:1	≥ 4.5:1	pass
number on background	4.7:1	≥ 4.5:1	pass
constant on background	7.18:1	≥ 4.5:1	pass
special on background	4.84:1	≥ 4.5:1	pass
error on background	6.63:1	≥ 4.5:1	pass
weakest accent on selection	3.34:1	≥ 3:1	pass

VISION	CLOSEST PAIR	DISTANCE	MIN	
normal	keyword / special	11.1	10	pass
deuteranopia	string / number	6.2	6	pass
protanopia	number / constant	6.5	6	pass
tritanopia	string / function	6.3	6	pass

Kvile Kontrast

CHECK	VALUE	RULE	
text on background	17.73:1	12-21:1	pass
comment on background	7.29:1	≥ 7:1	pass
text on selection	14.71:1	≥ 4.5:1	pass
keyword on background	9.67:1	≥ 7:1	pass
string on background	9.91:1	≥ 7:1	pass
function on background	13.18:1	≥ 7:1	pass
type on background	14.2:1	≥ 7:1	pass
number on background	11.37:1	≥ 7:1	pass
constant on background	13.91:1	≥ 7:1	pass
special on background	8.92:1	≥ 7:1	pass
error on background	8.92:1	≥ 7:1	pass
weakest accent on selection	7.41:1	≥ 3:1	pass

VISION	CLOSEST PAIR	DISTANCE	MIN	
normal	number / constant	10.6	10	pass
deuteranopia	number / error	6.3	6	pass
protanopia	keyword / special	6.5	6	pass
tritanopia	special / error	6.7	6	pass

C

Appendix C: glossary and sources

Glossary

Additive color	Mixing by adding light, as on a screen.
Anchor hue	The starting hue of a role, before tuning.
Chroma (C)	How colorful a color is. Zero is gray.
CVD	Color vision deficiency.
Contrast ratio	WCAG measure from 1:1 to 21:1, using relative luminance.
Gamma	The curve that maps stored numbers to emitted light.
Gamut	The set of colors a display or color space can show.
Halation	The glow or bleed some people see around bright text on dark.
Hue (h)	The color angle, in degrees, in OKLCH.
JND	Just-noticeable difference.
Lightness (L)	In OKLCH, perceptual lightness from 0 to 1.
Linear light	Values proportional to the physical light emitted.
Oklab / OKLCH	A perceptual color space and its polar form.
Polarity	Dark text on light (positive) or light text on dark (negative).
Relative luminance (Y)	Light emitted, compared with white, 0 to 1.
Role	A syntax meaning such as keyword or string that has its own color.
sRGB	The standard color space of the web.
Simulated annealing	A search method that accepts some worse moves early to avoid dead ends.
Token	A named color value, such as keyword or bg.

Sources and further reading

- Machado, G. M., Oliveira, M. M., Fernandes, L. A. F. (2009). A physiologically-based model for simulation of color vision deficiency. *IEEE Transactions on Visualization and Computer Graphics*, 15(6). Project page: inf.ufrgs.br/~oliveira/pubs_files/CVD_Simulation/CVD_Simulation.html
- Ottosson, B. (2020). A perceptual color space for image processing (Oklab). bottosson.github.io/posts/oklab/
- W3C. Web Content Accessibility Guidelines (WCAG) 2.2: relative luminance, success criteria 1.4.3, 1.4.6 and 1.4.11. w3.org/TR/WCAG22/
- Sarkar, A. (2015). The impact of syntax colouring on program comprehension. 26th Annual Workshop of the Psychology of Programming Interest Group (PPIG). ppig.org/papers/2015-ppig-26th-sarkar1/
- Hannebauer, C., Hesenius, M., Gruhn, V. (2018). Does syntax highlighting help programming novices? *Empirical Software Engineering*. doi.org/10.1007/s10664-017-9579-0
- Piepenbrock, C., Mayr, S., Buchner, A. (2013, 2014). Studies on positive and negative display polarity and proofreading performance, including positive display polarity being particularly advantageous for small character sizes. Summarized at cogsci.nl/blog/is-bright-text-on-a-dark-background-a-good-idea
- Nielsen Norman Group. Dark mode vs. light mode: which is better? nngroup.com/articles/dark-mode/

Kvile's own numbers: `dist/tokens.json` and `dist/report.md` in the repository, generated by `generate.mjs`.