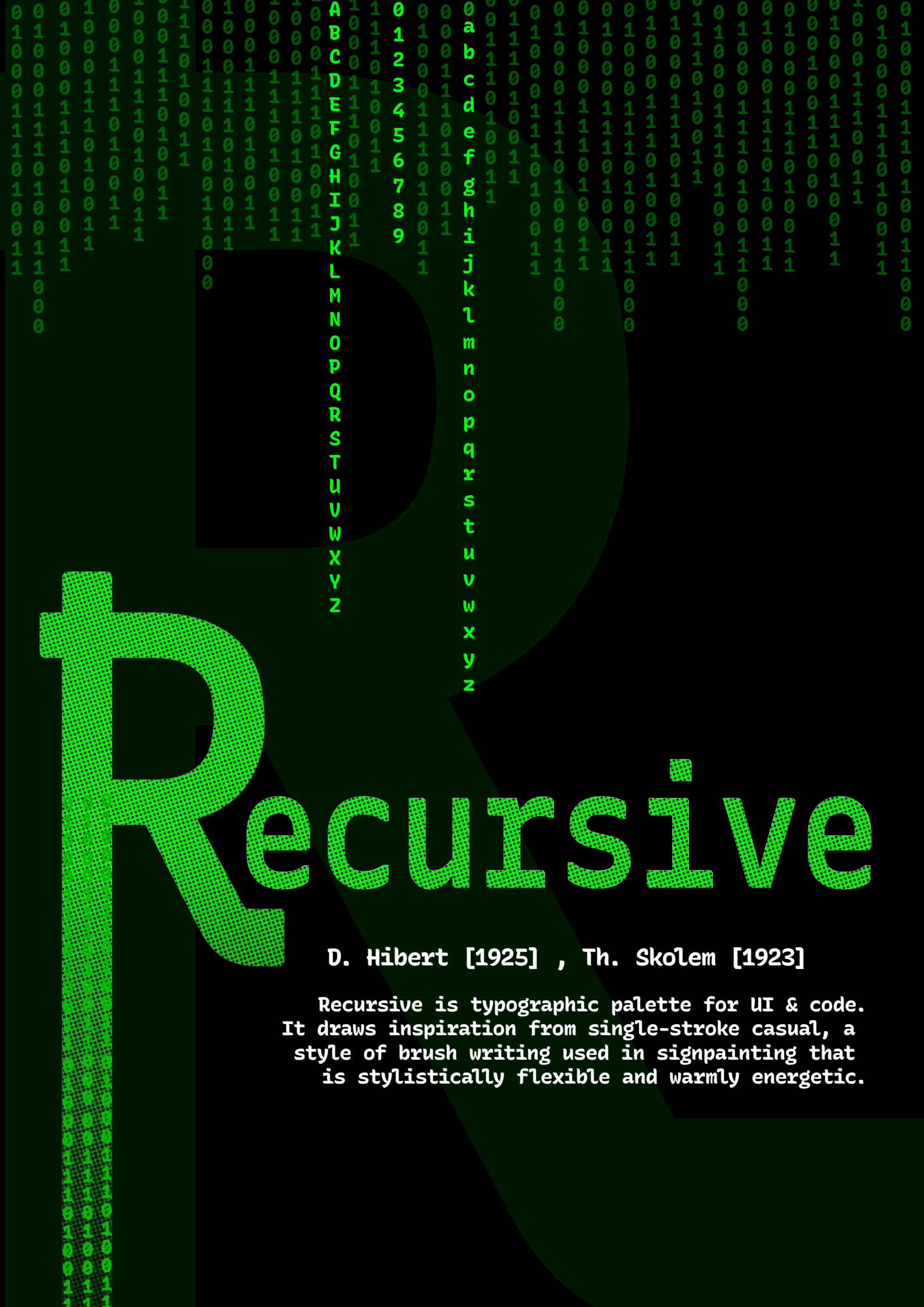


D. Hibert [1925] , Th. Skolem [1923]

Recursive is typographic palette for UI & code. It draws inspiration from single-stroke casual, a style of brush writing used in signpainting that is stylistically flexible and warmly energetic.

```
Recursive{} {
    Recursive(
        <Recursive>
            <Recursive
                Recursive
                    Recursive
                        Recursive
                            Recursive
                                "Recursive"
                            Recursive
                        Recursive
                    Recursive
                Recursive
            </Recursive>
        </Recursive>
    )Recursive
Recursive ]

Recursive{} {
    Recursive(
        <Recursive>
            <Recursive
                Recursive
                    Recursive
                        Recursive
                            Recursive
                                "Recursive"
                            Recursive
                        Recursive
                    Recursive
                Recursive
```

TABLE OF CONTENTS

<!DOCTYPE html>

RECURSIVE_

Recursive Mono & Sans is a variable type family inspired by casual script signpainting and designed for better code & UI. In programming, “recursion” is when a function calls itself, using its own output as an input. Recursive Mono was used as a tool to help build itself: it was used to write Python scripts to automate work and generate specimen images, and it was used in the HTML, CSS, and JS to create web-based proofs & prototypes. Through this active usage, Recursive Mono was refined to be not just warm, but also deeply useful for all-day work. Recursive Sans borrows characters from its parent mono but adjusts many key glyphs for comfortable readability in text. Its metrics are “superplexed” – glyphs take up the exact same horizontal space, across all styles.

As a 3-axis variable font, this allows for fluid transitions between weight, slant, and “expression” (casual to strict letterforms), all without text or layout reflow. In turn, this enables new interactive possibilities in UI – and makes for a uniquely fun typesetting experience.

A B C D E F G
H I J K L M N
O P Q R S T U
V W X Y Z a b
c d e f g h i
j k l m n o p
q r s t u v w
x y z

1 2 3 4 5

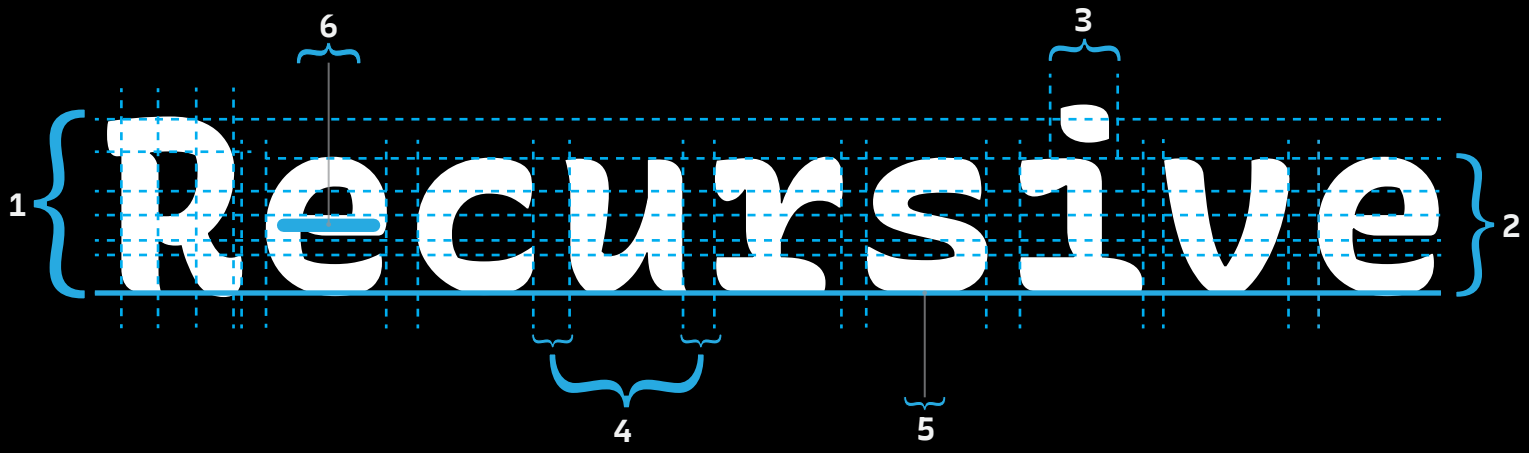
6 7 8 9 0

! @ # ^ & *

$ % + - = ? \

“ ” ; : , . ‘ _

( ) < > { } [ ]



1 Cap height
2 X height
3 Ascender
4 Kerning
5 Baseline
6 Crossbar

¶ 1 The height of uppercase letters from the baseline to the top of the letter.

¶ 2 The height of lowercase letters from the baseline to the top of the letters, excluding ascenders and descenders.

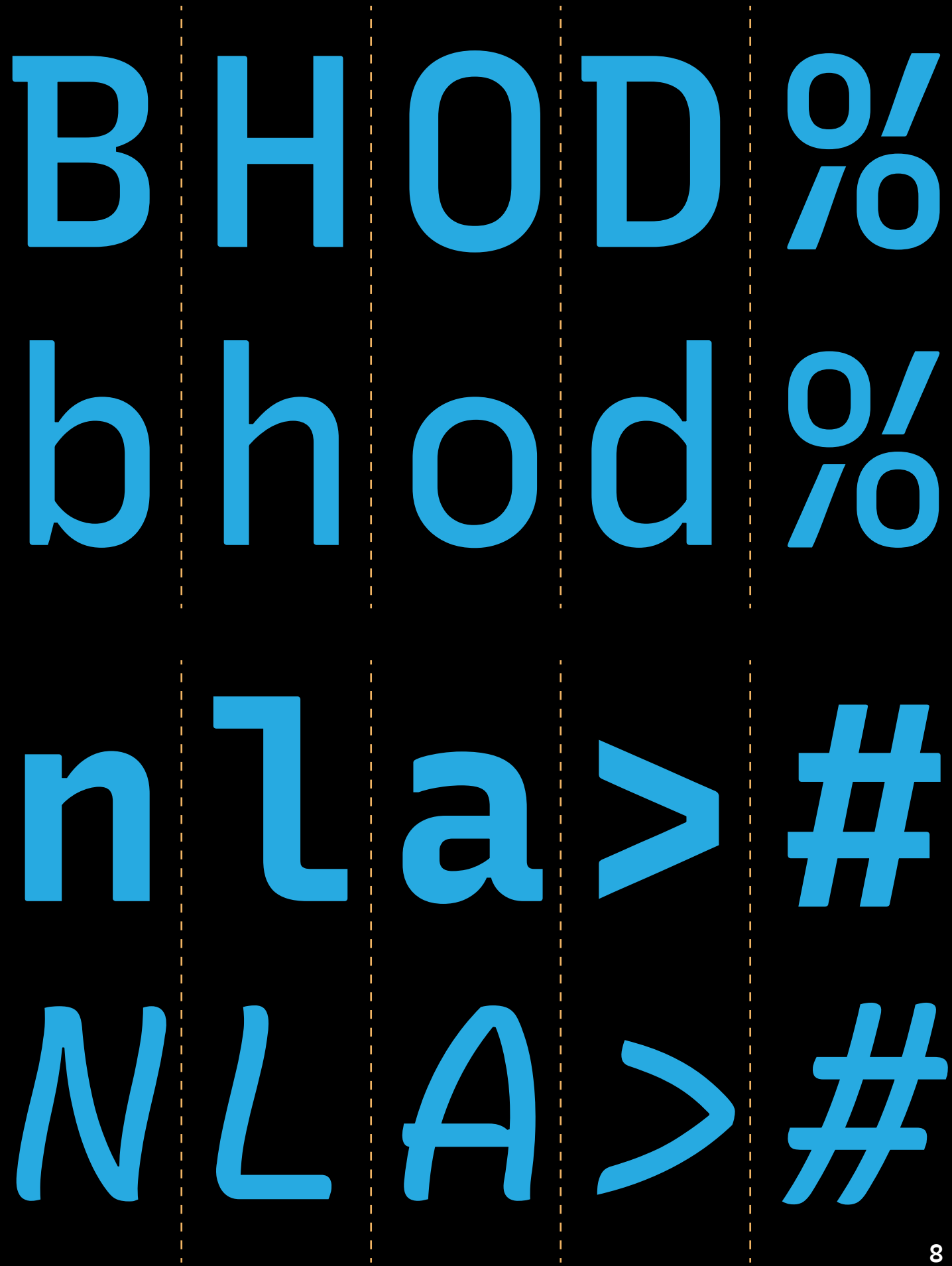
¶ 3 The part of a lowercase letter that extends above the x-height

¶ 4 The horizontal space between individual characters.

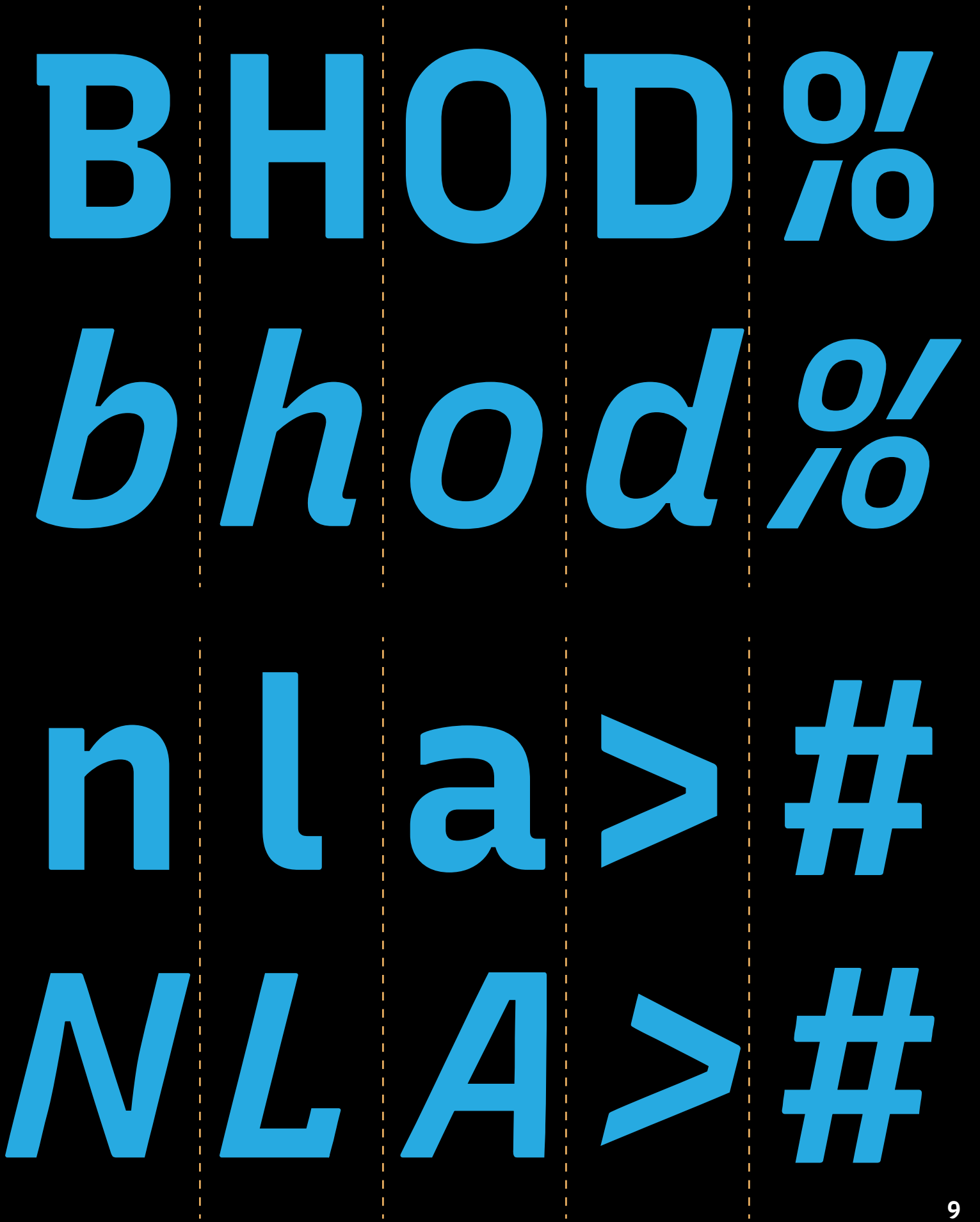
¶ 5 The invisible line on which all characters sit.

¶ 6 The horizontal stroke that connects two stems or other parts of a letter.

Subfamily: **Mono** Every character gets the same width as every other, arcoss styles.



Subfamily: **Sans** *Glyphs are superplexed - each keeps its width across all styles.*



MONO

“Programming isn’t about what you know; its about what you can figure it out.” Linear Light 10 pt

“Programming isn’t about what you know; its about what you can figure it out.” Casual Light 11 pt

“Programming isn’t about what you know; its about what you can figure it out.” Linear Medium 12 pt

“Programming isn’t about what you know; its about what you can figure it out.” Linear SemiBold 13 pt

“Programming isn’t about what you know; its about what you can figure it out.” Casual SemiBold 14 pt

“Programming isn’t about what you know; its about what you can figure it out.” Linear Bold 15 pt

“Programming isn’t about what you know; its about what you can figure it out.” Casual Bold 16 pt

“Programming isn’t about what you know; its about what you can figure it out.” Linear Black 17 pt

“Programming isn’t about what you know; its about what you can figure it out.” Casual Black 18 pt

“Programming isn’t about what you know; its about what you can figure it out.” Linear ExtraBlack 19 pt

“Programming isn’t about what you know; its about what you can figure it out.” Causal ExtraBlack 20 pt

SANS

Linear Light
10 pt

“Programming isn’t about what you know;
its about what you can figure it out.”

Casual Light
11 pt

“Programming isn’t about what you know;
its about what you can figure it out.”

Linear Medium
12 pt

“Programming isn’t about what you know;
its about what you can figure it out.”

Linear SemiBold
13 pt

“Programming isn’t about what you know;
its about what you can figure it out.”

Casual SemiBold
14 pt

“Programming isn’t about what you know;
its about what you can figure it out.”

Linear Bold
15 pt

“Programming isn’t about what you know;
its about what you can figure it out.”

Casual Bold
16 pt

“Programming isn’t about what you know;
its about what you can figure it out.”

Linear Black
17 pt

“Programming isn’t about what you know;
its about what you can figure it out.”

Casual Black
18 pt

“Programming isn’t about what you know;
its about what you can figure it out.”

Linear ExtraBlack
19 pt

“Programming isn’t about what you know;
its about what you can figure it out.”

Causal ExtraBlack
20 pt

“Programming isn’t about what you know;
its about what you can figure it out.”

ITALIC

*Recursive uses its *Casual* axis (`CASL`) to offer a range of personality, allowing you to get just the right tone for any context.*

Mono Linear light Italic 12 pt

Segment

Sans Casual Black Italic 99 pt

Light to ExtraBlack And everything in between `wght`

Sans Casual Medium Italic 16 pt

Extraordinarily versatile.

Mono Linear Medium Italic 29 pt

Casual (`CASL 1`) echoes the soft & curvy brush strokes of casual signpainting, but simplifies these forms for a striking and inviting tone. This makes it ideal for web headlines, code snippets, and command line interfaces.

Sans Casual Italic 12 pt

Along this axis, letterforms adjust in stroke curvature, contrast, and terminals to go from a sturdy, rational *Linear* to a friendly, energetic *Casual*.

Mono Casual Italic 10 pt

200 languages.

Sans Linear Black Italic 61 pt

Linear (`CASL 0`) styles have subtly-flattened edges and simple, open forms. This optimizes readability and enables enhanced focus in dense information, such as long-form text and complex code.

Mono Linear SemiBold Italic 11 pt

italics can emphasize key point in text

Sans Linear ExtraBold Italic 22 pt

Powerful

Sans Casual ExtraBlack Italic 56 pt

UI

Mono Casual Bold Italic 133 pt

SPECIAL CHARACTERS_

special

À É Ơ Ȳ Ŝ Ñ Ō

ḥ ĵ ɲ δ V Ể ȷ ţ

³] ₅ ŕ ǋ İ Ź

▶ ẳ ž ḷ ď × ʼ \ ╤

‹ ⅞ x ỳ Ỹ Ş ḻ Ş Ï Ổ

Ș Ȓ Ắ ṯ ş ṟ ħ œ θ ¿ ฿

Ệ =<< >>— <+> [x]

Ǆ ◆ Ǽ Đ © ∂ ≠ £ ∏ ฿ ḷ Ż Ő » ▷ ﬂ :: ˀ

\r 6 (» õ ề þ φ % ∞ ¡ Σ ǜ ñ ě Ģ √ ?? ẁ ã º

ǿ ŋ ā ệ ŷ ½ ý

ồ ₄ İ *** 9 dz ↰

♡ &&& † ǵ Ẅ ï π φ §

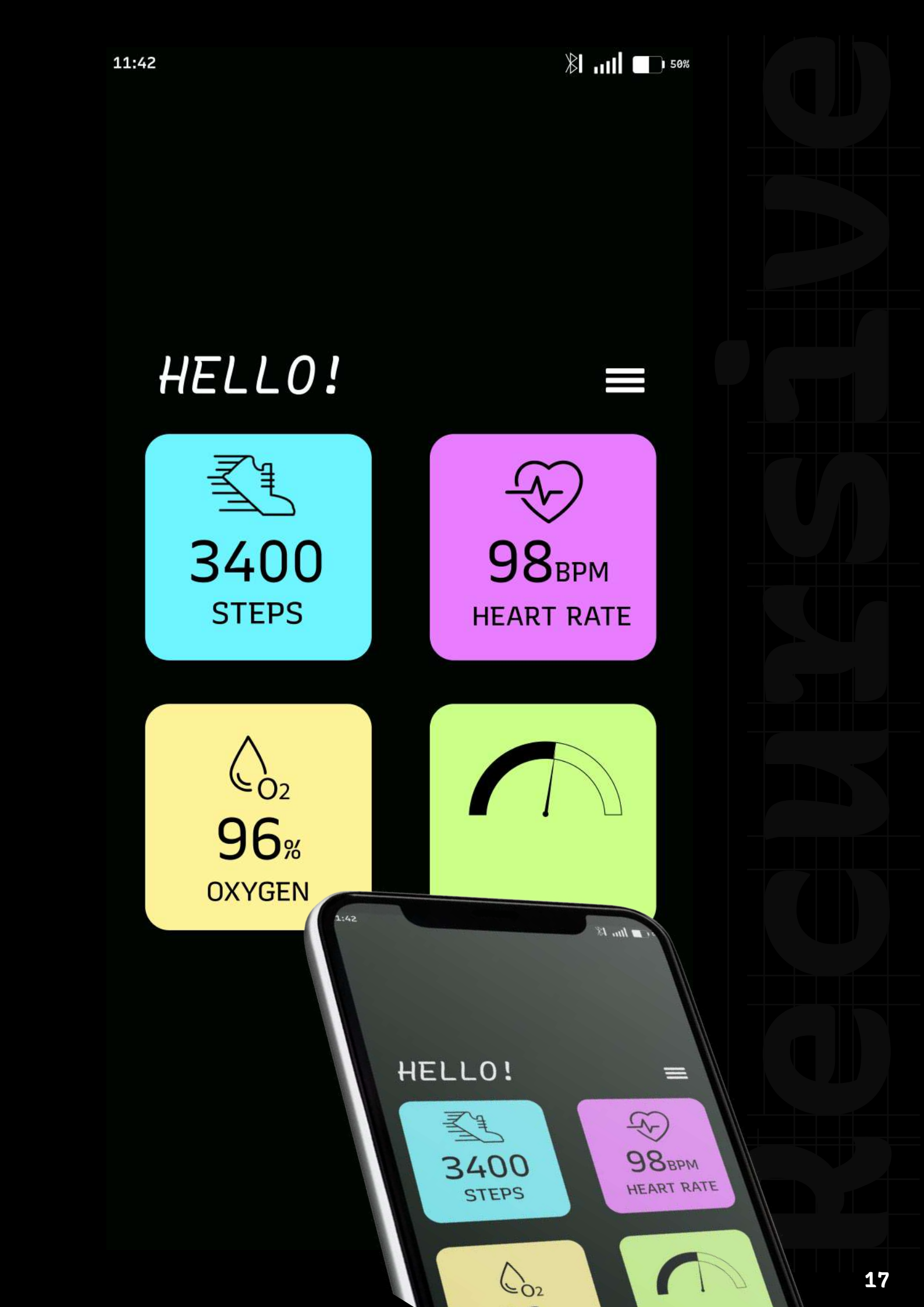
¿ ₴ ₭ ₸ € ₿ ₦ -=

å ȓ Ż ü ḻ ä \t ↑

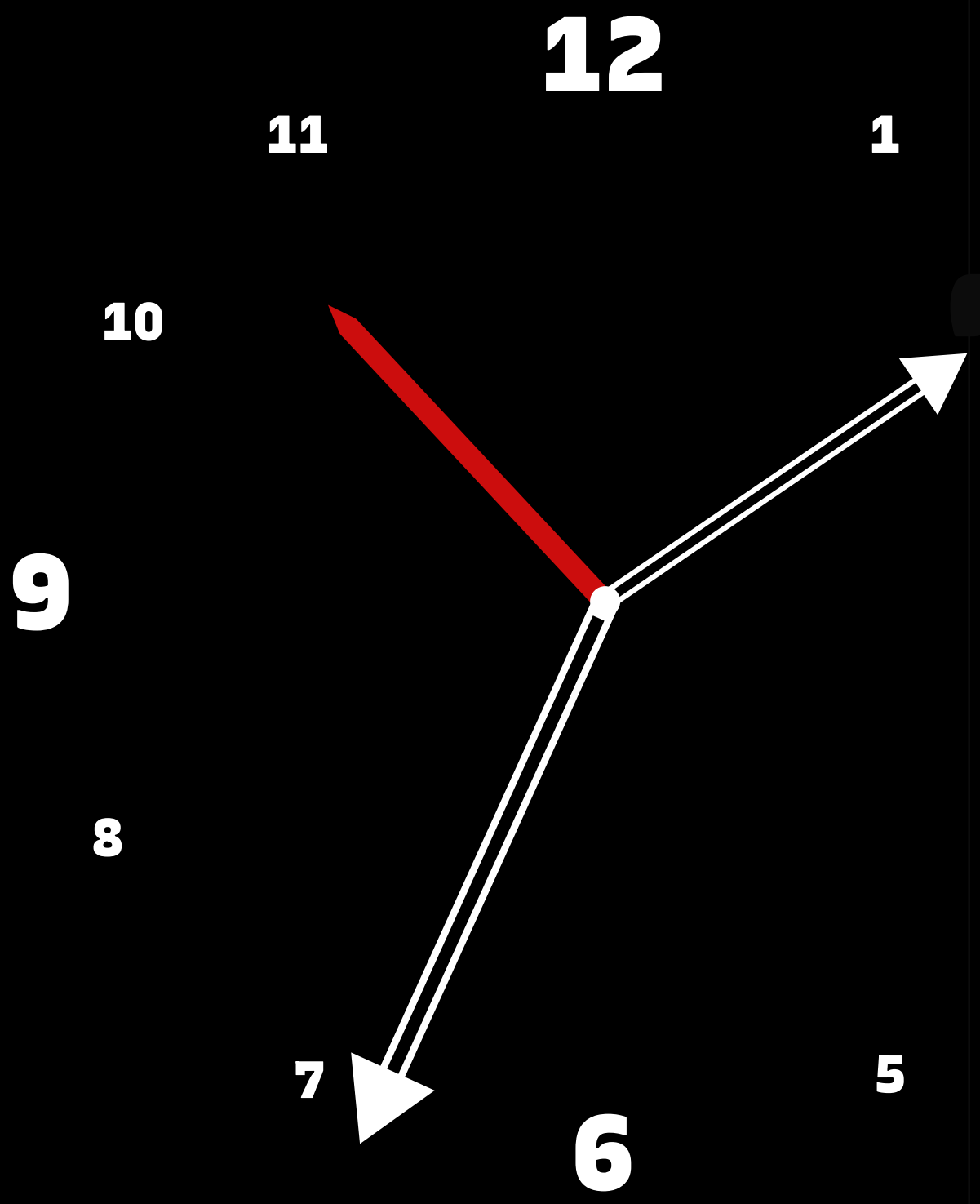
ķ Š ẩ Ǳ Λ ḉ ň ų æ Œ

Ħ į ő ↘ ฿ ¢ ffi ₺ ▽ ⁷

Recursive


11:42
50%
HELLO!
3400
STEPS
98BPM
HEART RATE
O2
96%
OXYGEN
HELLO!
3400
STEPS
98BPM
HEART RATE
O2

4:35
MONDAY, MAY 12


12
1
2
3
4
5
6
7
8
9
10
11

```
Stephens-MacBook-Pro:typemedia18 stephennixon$ gd
success open and validate gatsby-configs — 0.006 s
success load plugins — 0.427 s
success onPreInit — 0.007 s
success initialize cache — 0.006 s
success copy gatsby files — 0.053 s
success onPreBootstrap — 0.005 s
success source and transform nodes — 0.640 s
success building schema — 0.219 s
success createPages — 0.037 s
success createPagesStatefully — 0.030 s
success onPreExtractQueries — 0.002 s
success update schema — 0.029 s
success extract queries from components — 0.162 s
success run static queries — 1.443 s — 7/7 4.85 queries/second
success run page queries — 0.007 s — 5/5 819.77 queries/second
success write out page data — 0.004 s
success write out redirect data — 0.001 s
success Build manifest and related icons — 0.079 s
success onPostBootstrap — 0.080 s

info bootstrap finished - 4.908 s

 DONE  Compiled successfully in 2285ms                        11:16:23 AM


You can now view typemedia-2018 in the browser.

  http://localhost:8000/

View GraphiQL, an in-browser IDE, to explore your site's data and schema

  http://localhost:8000/___graphql

Note that the development build is not optimized.
To create a production build, use npm run build
```

add-sloped-grid-of-guides.py

Run Comment Uncomment Indent Dedent Save Reload New Open Edit With...

```python
# if the glyph already has a bunch of guides clear them and set default guides to be unlocked; oth
if len(g.guidelines) > 10:
    print("🤖 There are more than 10 guidelines. Now clearing guides.")
    clearGuides()
    # set to False to unlock guidelines
    setDefault("glyphViewLockGuides", False)

else:
    print("🤖 Applying a grid of guidelines with a size of " + str(gridSize))
    print("🤖 You have a leftover of " + str(UPM % divisions) + " units.")
    clearGuides()
    setDefault("glyphViewLockGuides", True)

    italicOffset = f.lib["com.typemytype.robofont.italicSlantOffset"]
    # add vertical guides
    for x in range(0, g.width+gridSize, gridSize):

        if italicOffset:
            xPos = italicOffset
        else:
            xPos = 0
        xPos += x
        g.appendGuideline((xPos, 0), 90+italicAngle)

    # add horizontal guides
    for y in range(0, int(UPM), gridSize):
        startAt = f.info.descender
        startAt += y
        g.appendGuideline((italicOffset, startAt), 0)

    # make guides magnetic, if you want to (It may disrupt drawing, though ¯\_(ツ)_/¯)
    for guide in g.guidelines:
        # control level of "magnetism" for vertical and horizontal
        if guide.angle == 90:
            guide.magnetic = 0
        else:
            guide.magnetic = 0

    # make guides magnetic
    for guide in g.guidelines:
        guide.locked = True
```

Server Side Rendering v2

styled-components supports concurrent server side rendering, with stylesheet rehydration. The basic idea is that everytime you render your app on the server, you can create a ServerStyleSheet and add a provider to your React tree, that accepts styles via a context API.

This doesn't interfere with global styles, such as keyframes or injectGlobal and allows you to use styled-components with React DOM's SSR, or even Rapscallion.

The basic API goes as follows:

```
import { renderToString } from 'react-dom/server'
import { ServerStyleSheet } from 'styled-components'

const sheet = new ServerStyleSheet()
const html = renderToString(sheet.collectStyles(<YourApp />))
const styleTags = sheet.getStyleTags() // or sheet.getStyleElement()
```

The collectStyles method wraps your element in a provider. Optionally you can use the StyleSheetManager provider directly, instead of this method. Just make sure not to use it on the client-side.

```
import { renderToString } from 'react-dom/server'
import { ServerStyleSheet, StyleSheetManager } from 'styled-components'

const sheet = new ServerStyleSheet()
const html = renderToString(
  <StyleSheetManager sheet={sheet.instance}>
    <YourApp />
  </StyleSheetManager>
)

const styleTags = sheet.getStyleTags() // or sheet.getStyleElement()
```

The sheet.getStyleTags() returns a string of multiple <style> tags. You need to take this into account when adding the CSS string to your HTML output.

Alternatively the ServerStyleSheet instance also has a getStyleElement() method that returns an array of React elements.

NOTE

sheet.getStyleTags() and sheet.getStyleElement() can only be called after your element is rendered. As a result, components from sheet.getStyleElement() cannot be combined with <YourApp /> into a

```
Recursive{} {
        Recursive(
            <Recursive>
                <Recursive
                    Recursive
                        Recursive
                            Recursive
                                Recursive
                                    “Recursive”
                                Recursive
                            Recursive
                        Recursive
                    Recursive
                </Recursive>
            </Recursive>
        )Recursive
Recursive ]

Recursive{} {
        Recursive(
            <Recursive>
                <Recursive
                    Recursive
                        Recursive
                            Recursive
                                Recursive
                                    “Recursive”
                                Recursive
                            Recursive
                        Recursive
                    Recursive
```


Recursive