ChatGPT

Encoding RGB Colors into Pronounceable, Reversible Names

Introduction

Encoding 24-bit RGB color values (e.g. hex `#RRGGBB`) into human-friendly names is challenging. Traditional named colors (like CSS color keywords or X11 colors) cover only a small fraction of the 16,777,216 possible RGB values, and their names are often arbitrary. We seek a **lossless encoding** that assigns every color a unique **pronounceable name**. Key requirements include:

- **Full reversibility:** The mapping from color to name (and back) must be one-to-one with no information loss.
- **Human readability & auditory clarity:** Names should be easy to read, spell, and pronounce. They should be distinct when spoken aloud or read by text-to-speech, minimizing confusion.
- **No ambiguous case/symbol usage:** Avoid encodings that rely on differences invisible in speech (e.g. case-sensitivity or punctuation). For example, treating `“abc”` vs `“Abc”` as different codes would be problematic since they sound identical.
- **Phonetic consistency across languages:** Where possible, use phonemes and patterns that are broadly pronounceable. The scheme should ideally be adaptable or localizable so speakers of different languages can still use distinct, valid names without tongue-twisters.

Below, we first examine existing color-naming systems and why they fall short of these goals. Then we explore algorithms and encoding schemes designed to meet the above criteria. Finally, we compare promising approaches in a summary table and recommend directions for implementation.

Existing Color Naming Systems and Their Limitations

- **CSS/X11 Named Colors:** Web standards define about *140 named colors* (e.g. `"red"`, `"LightSeaGreen"`). These names are intuitive for basic colors but cover only a tiny subset of RGB space [1]. They are not *reversible* (many colors have no name, and some names like `"aqua"`/`"cyan"` refer to the same hex). In practice, their **usability** is high for common colors but **scope** is very limited.

- **XKCD Color Survey Names:** A crowdsourced list of *954 color names* resulted from the XKCD color survey [2]. Examples include creative names like `"dusty lavender"` or `"vomit green"`. This set is larger and reflects how people describe colors, improving descriptive **clarity**. However, it still covers only ~1,000 points in color space [3]. Many RGB values remain unnamed, so it's not exhaustive or reversible. Usability is good for communication (since names are real phrases), but consistency suffers – some names are quirky or overlapping (e.g. multiple shades all called “blueish”).

- **Resene (Paint) Colors:** Paint manufacturers like Resene catalog thousands of colors each with unique names (Resene's online library includes *over 6,000 named swatches* [4]). This vastly expands the naming **scope**, far beyond CSS or XKCD lists. Names like *“Alabaster”*, *“Bali Hai”*, or *“Butter Yellow”* are often chosen to be evocative. Despite the breadth, these are **not systematic**:

they're curated labels, not an algorithmic encoding. They don't cover *every* possible RGB value, and new colors would need new coined names. Also, some names can be long or culturally specific. Thus, while human-friendly, they don't fulfill the **full reversibility** requirement. (Other curated lists – e.g. Crayola crayon colors, Pantone names – have similar limitations in scope and consistency.)

- **Color Taxonomies (ISCC–NBS, Munsell, etc.):** Scientific naming systems classify colors by hue/ lightness/saturation categories (e.g. "vivid yellowish green" in the ISCC–NBS nomenclature). These provide **descriptive clarity** and are grounded in perception [5]. However, they bucket the continuum into broad categories rather than uniquely naming each RGB; many distinct colors share the same descriptor. They are great for communication (since terms like "light blue" are easily understood) but inherently **lossy** – not a one-to-one encoding of exact values.

In summary, existing name sets either prioritize human meaning (but lack completeness) or are codes like hex values (complete but not human-friendly). None fully satisfy our goal of a **comprehensive, pronounceable, reversible** naming scheme. This motivates creating an algorithmic encoding that yields a unique "name" for any RGB color while remaining easy to speak and remember.

Designing a Pronounceable, Reversible Encoding

To achieve a **lossless mapping to pronounceable names**, we need to encode the 24-bit color value into a sequence of syllables or words. The design considerations include: how to partition the bits, what "alphabet" of sounds to use, and how to ensure the result is pleasant and unambiguous. Several approaches can be considered:

Syllable-Based Phonetic Encoding (Proquints and Beyond)

One proven approach is to encode binary data as a sequence of **pronounceable syllables**. A notable example is the *Proquint* scheme (PRO-nounceable QUINTuplets) by Daniel Wilkerson, which maps each 16-bit chunk into a 5-letter "word" of alternating consonant and vowel letters [6]. In Proquint, **4 bits** choose a consonant and **2 bits** choose a vowel, repeated in a C–V–C–V–C pattern to cover 16 bits [7]. For example, an IP address `127.0.0.1` becomes `"lusab-babad"` under Proquint encoding [8]. The generated "words" are nonsense, but they **follow English-like phonotactics**, making them relatively easy to read and spell. The consonant and vowel sets are carefully chosen to avoid letters that sound confusable (for instance, Proquint omits ambiguous letters like *c, x, q* and uses a fixed set of 16 consonants and 4 vowels) [9]. Separators (like hyphens) can be used between syllable blocks for readability, though they don't carry data [10].

Building on this idea, we can encode a 24-bit color in a similar fashion. **One practical design** is to split the 24 bits into two 12-bit chunks, and encode each chunk as a **CVC syllable** (Consonant–Vowel–Consonant). Using a larger phonetic alphabet (e.g. 32 possible consonants = 5 bits, and 4 vowels = 2 bits) yields exactly 12 bits per CVC syllable [11]. This gives us two pronounceable syllables covering all 24 bits. For example, in a prototype called the "Over-Color" scheme, the color `#4E9AF8` is encoded as **"lum-kiv…"** (full example below) – a unique, pronounceable name [12]. Each syllable is drawn from a controlled lexicon so that, even though the words are artificial, they avoid problematic combinations and can be sounded out easily.

Notably, this approach is **fully reversible and collision-free**: given the mapping tables for consonants and vowels, the decoding algorithm can reconstruct the exact RGB value from the two-syllable string. The names are roughly word-length (6–7 letters plus a hyphen), which is short enough to be manageable. They look like gibberish "words," but are designed to **"look like language"** – e.g. their frequency distribution of syllable patterns can mimic natural language Zipfian trends [13], meaning they don't appear as random jumbles. An additional benefit is that this method is purely algorithmic (bitwise operations and table lookups), so implementation is straightforward and fast in any programming language [14].

Example – Over-Color Scheme: Every RGB value is first packed into a 24-bit integer and split into two 12-bit halves [14]. Each 12-bit half is encoded as one consonant-vowel-consonant syllable. To improve usability, the Over-Color design adds a few **extra characters**: after the two core syllables, it appends: (a) a letter indicating a broad **lightness** bucket, (b) a letter for a **chroma/saturation** bucket, and (c) a final **check vowel** for error-checking [5]. These suffix characters are inspired by how human color naming often adds modifiers like "light" or "deep" (ISCC–NBS system) [15], and by parity-check digits in data transmission. The check vowel is computed from the data bits (e.g. an XOR of the other components) to catch single-letter errors [16] – if a name is mistyped or misheard, an invalid parity vowel hints at the mistake. In total, the encoded name might have ~8–9 letters plus an optional hyphen. For `#4E9AF8` (a bright blue), the scheme yields **"lum-kiveut"** [12]. Here `"lum"` and `"kiv"` encode the 24-bit value, `"e"` might encode a lightness category (e.g. medium-bright), `"u"` a chroma category (e.g. vivid), and `"t"` is the parity check letter. The result is a **unique, pronounceable identifier** for that exact color. A person could read "lum-kiveut" over the phone, and the listener (or a computer) could decode it unambiguously back to `4E9AF8`.

This syllabic encoding approach excels in **algorithmic simplicity** and meets all the criteria: - *Reversible:* By design – the encoding is just a base-N representation in a phonetic alphabet [11]. - *Audibly clear:* Only a restricted set of phonemes is used (no two syllables or letters sound too alike), and the optional parity vowel adds robustness against slips. - *No case or hard-to-say symbols:* Outputs are all lower-case letters forming pseudo-words. The only punctuation (hyphen) is for visual grouping and can be pronounced as "dash" or omitted. - *Multi-language potential:* The scheme can be **localized by swapping alphabets** [17]. For instance, one could create a Cyrillic consonant/vowel table and encode the same bits into pronounceable Russian-like syllables. Similarly, one could adapt to languages with different phonemic sets or even map the syllables to a **fictional script** for stylistic purposes [18]. The underlying encoding remains the same; only the symbols representing the phonemes change. This flexibility means the concept can accommodate cultural/linguistic preferences (for example, avoiding sounds that speakers of a certain language find difficult).

Other examples of syllabic or phonetic encodings include **Bubble Babble**, a scheme used to represent binary data (like SSH fingerprints) as sequences of syllables. Bubble Babble produces strings like `"xesef-disof-gytuf-katof-movif-baxux"`, which look nonsensical but are divided into vowel-consonant combinations that can be pronounced [19]. It's a similar idea, though Proquints/Over-Color use a more constrained pattern to improve readability. In practice, *any* fixed mapping of bit patterns to pronounceable syllables can serve – the differences lie in which phonemes are allowed and how the bits are distributed among them.

Word-Based Encoding (Real Dictionary Words)

Another approach is to encode colors as a combination of actual words from a dictionary. The concept is akin to the **PGP Word List** or **Diceware** passphrases, where binary data is converted into a sequence of common words for easier memorization. For a 24-bit color, one could use either:

- **Two words from a large list**, or
- **Three words from a smaller list**.

For example, if we had a list of 4096 short words, we could map the first 12 bits of a color to one word and the last 12 bits to a second word, since $4096 = 2^{12}$. This would yield combinations like `"Mango Fuzz"` or `"Tiger Lamp"` (hypothetically). Alternatively, the PGP word list approach uses 256 words and encodes each byte as one word. Using that, a 3-byte RGB value would become three words (e.g. `"tree arrow bucket"` – just an illustration). The **advantage** of word-based schemes is that real words can be easier to spell and recall than made-up syllables. They also have built-in redundancy: mishearing one word might be caught because it doesn't make sense in context or isn't in the list at all. In fact, the PGP word list was designed such that the words for byte values are phonetically distinct (e.g. no two words in the list sound too similar) [7] [9], and even/odd byte positions use different lists to detect transpositions [10].

However, there are notable **trade-offs**: - **Name length:** Two or three words will typically have more characters (and syllables) than a two-syllable proquint. e.g. `"lavender notebook"` is longer to say and write than a single invented word like `"lumkiv"`. This can affect usability when communicating or typing the names. - **Ambiguity in spelling:** Real English words can have tricky spelling or homophones. (Though a curated list can avoid homophones, one still might confuse `"grey"` vs `"gray"` or have to clarify if a word is uncommon.) Made-up syllables, by contrast, can be consistently phonetic. - **Semantic distraction:** Words carry meaning. Using random word pairs might produce combinations that either inadvertently describe a different color or object ("Blue Banana") or just sound nonsensical ("Electric Crayon"). This can be amusing but might also confuse users or not be universally culturally neutral. There's also a risk of offensive combinations if words aren't filtered carefully. - **Vocabulary limits:** To cover 16 million values, the word list approach stretches beyond the size of typical controlled vocabularies. A list of 4096 safe, short, distinct words is feasible (for reference, Diceware uses 7776 common words, each encoding ~12.9 bits [20]). But maintaining and localizing such a list in multiple languages is a substantial task. Each language would need its own carefully vetted list to preserve the pronounciation advantage.

In summary, word-based encoding is **highly human-readable** (the names sound like phrases or sentences), but the multi-word names may be longer than an optimized syllabic code. They are reversible if the word-to-value mapping is fixed, but implementation requires storing the dictionaries. This approach may be worthwhile if maximizing memorability is more important than minimizing length or if one wants the codes to double as passphrases. For color identification purposes, though, users might find arbitrary word pairs only slightly more meaningful than a made-up word – both ultimately label a color in an arbitrary way.

Phonotactic Name Generators

Between strict syllable encoding and fixed dictionaries lies an approach of **algorithmically generated "names"** following linguistic patterns. Instead of a rigid CVC-CVC format, one could design a generator that composes a variable-length pronounceable name from the 24-bit input. For example, certain bits might choose a syllable structure or stress pattern, and others pick letters or phonemes, constructing a name that looks more like a natural name (perhaps 2–3 syllables long, not all the same pattern). This is akin to how some videogames or fantasy novel authors generate plausible character or place names via phonotactic rules.

The benefit would be **greater variety and naturalness** in the names – not every color name would look like the same kind of gibberish. Some might accidentally resemble real words or be more lyrical (which could aid memory). You could also design the generator to avoid specific undesired sequences (like anything that spells a real profanity, for instance).

However, ensuring **full reversibility** becomes more complex here. The encoding must be deterministic and *injective* (one-to-one). This likely means you'd still define a fixed mapping from bitfields to particular syllable choices, just with a more complex grammar. Every possible output "name" must map uniquely to one input color. It's doable (essentially creating a custom base-N number system where "digits" are syllables of various shapes), but the complexity of encoding/decoding increases. Moreover, if names have variable length or form, users might be unsure if a given name is complete or truncated. A fixed-length or clearly delimited format has the advantage in an encoding context that the boundaries are known (like the hyphen-separated blocks in proquints).

In practice, the **syllabic approach with fixed pattern** already achieves a language-like feel and is easier to implement. So, phonotactic generators are an interesting area for future exploration (perhaps to improve aesthetics of the names), but they would need careful design to maintain the strict one-to-one property.

Symbol or Color-Code Based Mnemonics

For completeness, one could consider non-verbal or hybrid schemes – for instance, representing a color by a sequence of colored emoji or icons. For example, the red, green, and blue components could each be mapped to an icon or a color name in another language, resulting in a visual "code". While this might be fun in a UI (e.g. showing as a code for a color), it **fails the pronounceability test**. Ultimately, someone will try to read it ("blue dolphin guitar?"), which is neither standardized nor straightforward. Thus, purely symbolic encodings tend not to meet our goals – they might serve as memory aids or additional annotations (like a pictogram alongside the spoken name), but we still need a textual name for unambiguous communication.

Having considered these methods, the **syllable-based encoding** (inspired by Proquints) emerges as a particularly promising solution. It balances brevity, clarity, and completeness. We can fine-tune it with ideas from human linguistics (like the lightness/chroma modifiers) and data integrity (parity check) to further enhance usability. Below is a comparison of the discussed approaches, including existing naming systems and the new encoding schemes:

Comparison of Alternatives

Scheme / System	Reversible?	Pronounceable & Distinct	Scope (Coverage)	Notes (Pros/Cons)
CSS/X11 Color Names	No – one name maps to a specific RGB (many colors lack names) [1].	Yes (simple common words)	~140 names (very limited) [1].	Easy for basic colors; not unique for arbitrary colors (e.g. no name for `#123456`). Duplicates exist (e.g. "aqua" = "cyan"). Useful in web design, but not an encoding system.

Scheme / System	Reversible?	Pronounceable & Distinct	Scope (Coverage)	Notes (Pros/Cons)
XKCD Survey Names	No – only covers popular colors, not every value [21].	Mostly yes (natural phrases)	954 names [2].	Covers most "normal" colors with fun, crowd-sourced labels. Names are understandable (e.g. "pale yellow"), but many RGB values still have no unique name. Not systematic or extensible without another survey.
Resene Paint Colors	No – finite set of paint colors, others unnamed.	Yes (real words, often two-word names)	~1,300–6,000 names (depending on catalog) [4].	Huge curated palette used in design/paint industry. Names are creative and human-friendly (e.g. "Salsa", "Butterfly Bush"). Still nowhere near 16 million; not designed to encode arbitrary colors. Good inspiration for descriptors but not a coding scheme.
ISCC–NBS / Munsell Notation	No – describes color in categories, not unique per RGB.	Partially (uses standardized terms, e.g. "light", "dark", basic hues)	Continuous spectrum (all colors fit into some category) but **not unique**.	Useful for descriptive naming and understanding color relationships. However, many different RGB values share the same name (e.g. dozens of varieties might all be "moderate red"). Not an exact identifier.
Hex Code (baseline)	Yes – `#RRGGBB` is fully specific.	No – alphanumeric code, not pronounceable as a word.	All 16,777,216 colors.	The default we want to improve upon. Hex is compact and precise, but reading out "4E 9A F8" is error-prone (letters sound like others, one digit wrong breaks it). Great for machines, poor for humans.

Scheme / System	Reversible?	Pronounceable & Distinct	Scope (Coverage)	Notes (Pros/Cons)
Proquint/ Phonetic Syllables	Yes – one-to-one mapping from RGB to syllables [11].	Yes – outputs pseudo-words following phonetic rules. Designed to avoid homophones.	All 16,777,216 colors (by construction).	*Example:* `#4E9AF8` → "lumkiv…". Highly systematic and compact (~2–3 syllables per color). Easy to parse and transmit verbally. Needs carefully chosen letter sets to avoid confusion [9]. Hyphen separators can be used for readability without affecting encoding [10]. This is the core of the **Over-Color** scheme [11].
Over-Color (CVC-CVC + extras)	Yes – extension of proquint, with added checksum.	Yes – pronounceable nonsense words; suffix gives hints (light/dark, etc.).	All colors (extensible to larger gamuts too) [17].	*Example:* `#4E9AF8` → "**lum-kiveut**" [12]. Two CVC syllables encode the RGB exactly; plus `e`/`u` indicating lightness/chroma bins and `t` as a parity check [5]. Very user-friendly: a transcription error is likely caught by parity. Lightness/chroma tags make names a bit more meaningful (you can tell roughly if a color is light/dark or vivid by those letters, analogous to saying "light" or "deep"). Slightly longer (8–9 letters) but still reasonable. Open to localization (could swap in other consonant/vowel sets or even non-Latin scripts) [18].

Scheme / System	**Reversible?**	**Pronounceable & Distinct**	**Scope (Coverage)**	**Notes (Pros/Cons)**
Bubble Babble Encoding	Yes – algorithmic (checksum included by design).	Mostly – outputs a series of syllables like "bab", "ded", etc.	All colors (in theory).	*Example:* might produce names like "xesef-disof-…". Similar to proquint in using alternating vowels, but the specific phoneme set differs. Includes a checksum and always outputs a fixed pattern with hyphens. Readable but slightly longer output than optimized proquint (due to fixed padding). Not tailored for color specifically, but applicable.
Dictionary Words (2–3 words)	Yes – if using fixed word lists for each chunk.	Yes – real English words, likely easy to say.	All colors (if list is big enough).	*Example:* `#4E9AF8` → "**guitar lava**" (hypothetical). Two-word scheme requires 4K+ unique words; a three-word scheme could use shorter list (256 or 1024 words) for each. Very clear when spoken, but lengthier to convey. Needs large curated lexicon (with no collisions or confusable words). May produce odd combinations and requires multilingual word lists for localization.
"Hex Speak" or Base-N Codes	Yes – encodes value in an alternate base (e.g. base-36, etc.).	No – often includes digits or unpronounceable sequences.	All colors (by definition).	e.g. Base-36 might encode `0x4E9AF8` as "3W6O8" – shorter than hex, but still not word-like. Some hex-speak might form a word ("FACE" for 0xFACE), but most are not readable. Lacks consistent pronunciation and could accidentally form rude or confusing strings. Not suitable for verbal use.

Scheme / System	Reversible?	Pronounceable & Distinct	Scope (Coverage)	Notes (Pros/Cons)
Phonetic Algorithm (variable)	Yes – if carefully designed, but decoding is complex.	Intended to be, but quality depends on design.	All colors.	This is a category for hypothetical advanced name generators. Could yield more naturalistic names (e.g. “Loranda”, “Sappiny”). Increases complexity and risk of error in decoding. Not yet a standard; would require significant testing to ensure no collisions and ease of use.

(Table legend: “Reversible” means a unique mapping both ways. “Pronounceable & Distinct” reflects whether names are likely to be spoken correctly and not easily mistaken for one another. “Scope” indicates how many distinct colors the scheme can name; all new schemes aim for full 24-bit coverage. The Notes highlight strengths and weaknesses for usability, clarity, and implementation.)

Recommendations and Future Development

Most promising approach: The **syllable-based encoding (CVC patterns inspired by Proquints)**, especially the enhanced **Over-Color** variant, appears to best satisfy the project goals. It provides a **practical balance** between brevity and clarity. Every RGB value gets a compact name that can be vocalized and remembered. Features like the lightness/chroma suffix and parity check in Over-Color specifically address human factors (making the gibberish a bit more informative and robust against mistakes) [5] . We recommend using this method as a starting point for implementation.

Implementation strategy: Start by defining the consonant and vowel sets for the encoding. Based on prior art, a **set of consonants** around 20–32 in number (excluding ones that sound too similar) and **4–5 vowels** works well [9] . For example, the Proquint set (b, d, f, g, h, j, k, l, m, n, p, r, s, t, v, z for consonants and a, i, o, u for vowels) could be expanded slightly if needed for 5-bit consonant values [22] . Test the chosen phoneme set by generating many random color names and checking for any unintended words or hard-to-pronounce combos. Minor adjustments (like avoiding starting a syllable with “v” if it tends to sound like “b” on a bad phone connection) can improve auditory distinguishability.

The encoding algorithm itself is straightforward bit math (split integer, lookup letters) [23] . This can be implemented in a small library in various languages. In fact, libraries for **Proquint** already exist [24] , and those can be adapted to 24-bit and tweaked letter sets. Similarly, **Hashids/Sqids** libraries (which encode integers to URL-safe strings) could be repurposed, though those focus on shortness over pronounceability [25] [26] .

Testing and iteration: Once implemented, it’s important to test the scheme in real use. For example, generate a random set of color names and have users read them to each other to see if the decoding matches. This can reveal if certain letters are consistently misheard (e.g. “m” vs “n”) or if certain syllables are confusing. If the parity check is implemented, verify that it indeed flags errors in practice. Real-world use (maybe a plugin for a color-picker tool that labels colors with these names) could provide feedback on whether the names are as *“human-friendly”* as intended. Adjust the phonetic alphabet or add clarifying rules as needed (for instance, one might decide to avoid producing double vowels or any output that looks like a real common word, to prevent misunderstanding).

Comparisons with alternatives: Although the proquint-style encoding is recommended, it could be worthwhile to also implement a **two-word dictionary prototype** for comparison. For instance, take a known word list (like the Diceware list [20] or another source of 4096 short words) and encode some colors with it, then ask users which they find easier to remember or communicate – the made-up syllable or the two English words. It may turn out that for quick reference (e.g. telling a colleague "use the color *rose tiger* for the background"), real words have merit. If so, one might consider offering both forms in a software context (a bit like how URLs sometimes have both a short hashid and a human word code). However, maintaining consistency is key – **one canonical format** should be the standard for reversibility, to avoid confusion. The simpler syllable format likely wins out for being unambiguous and concise.

Future work: The framework of this encoding can extend to other color systems and depths. For example, for 32-bit colors (with alpha channel or HDR), you could just add another syllable or another suffix letter – the method scales by chaining more syllables for more bits [17]. For multi-dimensional color spaces (CMYK, etc.), you can design a mapping (the Over-Color notes suggest one syllable for K and three for CMY in CMYK [17]). This could allow consistent naming across different color models. Another avenue is **localization**: creating language-specific consonant/vowel tables so that, say, a Japanese version yields pronounceable katakana sequences for colors, or an Italian version uses that language's phonetics. This might involve consulting linguists to ensure the letter combinations chosen are valid and distinct in the target language.

Finally, while the names generated are not meant to convey meaning, it's interesting to consider **mnemonic enhancements**. For instance, the lightness/chroma indicators already give a hint of the color's appearance. One could imagine encoding the hue in a more explicit way (perhaps a set of syllables that roughly correspond to red, green, blue, etc., so that part of the name hints at the hue). This ventures into *semi-reversible* territory (since any such hint would either reduce uniqueness or be wrong for edge cases), but it might be done at a coarse level without sacrificing uniqueness – similar to how the lightness tag works. Such ideas tread a fine line between a **truly arbitrary code** and a **descriptive name**. The current recommendation is to keep the encoding mostly abstract (so users learn it as a code), rather than overload it with too much semantic meaning that might mislead (e.g. one shouldn't think a color name ending in "-red" actually is a red shade unless that's guaranteed). Nonetheless, exploring this could be a future improvement to make the names even more intuitive.

Conclusion: By using a pronounceable encoding scheme, we can assign every RGB color a unique "name" that humans can handle in spoken or written communication. Among the options, a **proquint-style syllabic code** augmented with human-inspired tweaks offers an excellent solution. It combines the precision of a code with the familiarity of language. Going forward, implementing this scheme and comparing it with other human-friendly codes will ensure we create a naming system that is not only technically reversible but truly optimized for people.

---

[1] CSS Colors
https://www.w3schools.com/cssref/css_colors.php

[2] [3] [21] Color Survey Results – xkcd
https://blog.xkcd.com/2010/05/03/color-survey-results/

[4] Resene Paints - Over 6000 Colour Swatches To View & Download
https://www.resene.co.nz/swatches/?srsltid=AfmBOortqVW6P_cZq5ORQxY7-fIMBfuFfWAas8VHjwBLU7uTa0kxDcqM

[5] [11] [12] [13] [14] [15] [16] [17] [18] [20] [23] [24] [25] [26] Untitled document
https://docs.google.com/document/d/1X99oQLhn6Vvgvlh4c5Yv9m9qyxyKodIy5ofG3QG-7nY

[6] [7] [8] [9] [10] [19] [22] readable - Does a pronounceable encoding exist? - Stack Overflow
https://stackoverflow.com/questions/1648206/does-a-pronounceable-encoding-exist