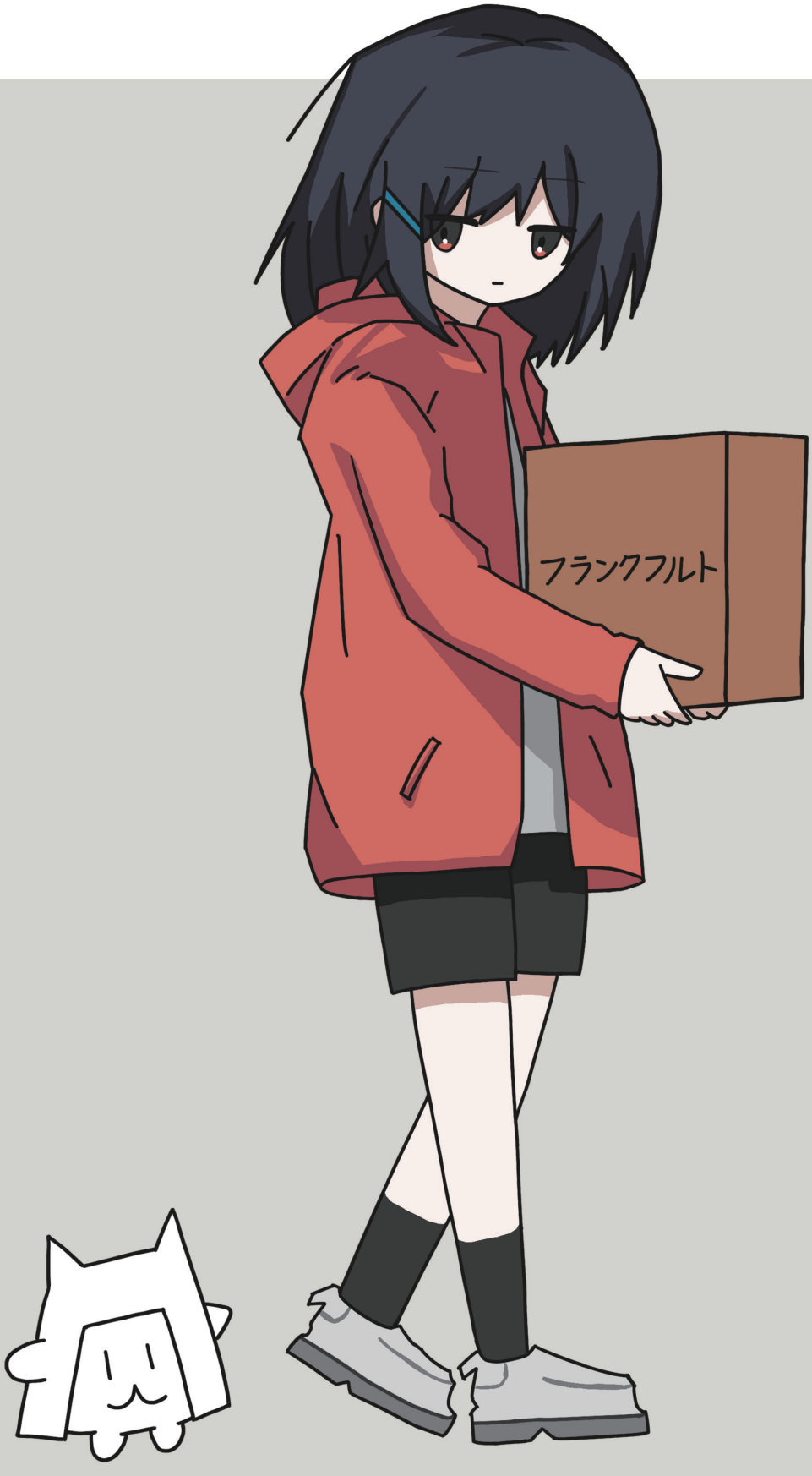
フランクフルト

2025 Vol.1

驚額の殿堂 技術同人誌

KYOGAKU-DENDO TECH BOOK



読んで学士（情報科学）目指すんぽ！

目次

自作 JVM ことはじめ

ちゅるり（@chururi_）

はじめに

> 我々は利益を無視した食品の販売には飽き足らず，同人誌の執筆にも手を出してしまいました．本当に愚かなことだと思います．意味ないですよー．

はじめまして，ちゅるりです．現在は筑波大学 理工情報生命学術院システム情報工学研究群 情報理工学位プログラムでM1をしておりますが，過去はscs2（総合学域群第2類）の1期生として筑波大学に入学し，その後mast（情報メディア創成学類）に移行してクネクネしていました．

さて，Java仮想マシン（Java Virtual Machine，JVM）とは，プログラミング言語Javaを実行するためのプログラムです．とはいえ，近年ではKotlinやScalaなどJVM上で動作するプログラミング言語が普及しているため，必ずしもJava専用とは言えない形になってきています．

なぜ JVM を自作するのか

Java の登場から 30 年以上経った今でも，数多のサービスの Web バックエンドから SIM カードに至るまで広く利用されています．他方，このようなJVMの爆発的な普及は，その優れた設計の裏付けでもあります．JVM の自作によってプログラミング言語ランタイムのエッセンスを学ぶことができます．

そして何よりも，プログラミング言語処理系の実装は楽しいのです．「自分で書いたプログラムに怒られる」という，極めて気持ちの良い体験ができます．何を言っているのかわからないでしょう．コンパイラやランタイムのような処理系の開発では，異常な状態になった際や想定外の入力がされた際などのエラーを「検知」し，「出力」する処理の実装がつきものです．自作 JVM にフォーカスして例を述べると，`java.lang.NullPointerException` という状態を検知し，`Exception in thread "main" java.lang.NullPointerException...` というスタックトレースを組み立て，表示する処理に該当します．自分で作ったこの仕組みから怒られるのは楽しそうではありませんか？

魅力：OpenJDK のバグを踏み抜く

ここで少々コラムです．私が昨年 JVM を実装していた頃，自作 JVM 上で動作している標準ライブラリが通常ではあり得ない挙動を示しました．自分のプログラムとよくテストされている標準ライブラリ，どちらが信用ならないのかは明白で，圧倒的に前者です．ゆえに自作 JVM の隅々までチェックしたのですが，どこにもバグっぽい箇所はありません．そして，もしや…と思って OpenJDK の実装を読みに行ったところ，なんとバグがあったのです．Contribution Chance!! と思って issues を見るとすでに起票されており，絶望しましたがかなり面白かったです．

（詳細：https://slides.itsu.dev/slide/3cc372be-52fa-4df3-80e8-cde80afd6847）

本稿について

本稿では，自作 JVM を行う際の指針について述べます．この指針とは，個別の要素の具体的な実装ではなく，もう少し JVM 全体をマクロに見たときにどのように自作を進めていけばよいのか，ということです．すなわち，ある特定の言語でクラスローダを実装する方法，のようなことは述べません．

本稿を通して，自作 JVM の沼に堕ちていただけることを強く願っております．

自作 JVM の動向

自作 JVM の前例

すでに多くの方々が自作 JVM をされてきています．ここでは国内の方に絞って，いくつかの実装を紹介します（敬称略）．機能の豊富さに違いはあれど，国内でも様々な言語による JVM の実装があります．

- php-java / めもりー
  - https://github.com/php-java/php-java

システムプログラミング入門としての自作スレッドスケジューラー

lapla（@lapla_cogito）

はじめに

ソフトウェアの重要な存在意義として，物事の自動化と抽象化を挙げることができます．自動化により我々の作業量は減少し，抽象化により取り組む課題の本質的な事項に集中しやすくなるという恩恵があります．

しかし自然なことではありますが，同時にこれらの性質は物事の根幹的な機序を隠蔽しうります．特に扱われる技術の幅が多様化の一途を辿る近年では，コンピューターの基盤的技術を扱う，いわゆるシステムプログラミングに触れる機会は多くの人にとって減少しているように思います．

他方，触れようと思っても中々敷居が高く映ってしまうのがシステムプログラミングの世界ではないでしょうか．特に近年は自作OS等に代表される，いわゆる「手を動かしてみる」系の資料が増えつつありますが，中々難しくて途中で挫折してしまったり，書籍を買ったは良いが読めずに積みっぱなしになってしまっているといったことも多くあると思います．

そこで本記事では，システムプログラミングに入門するにあたっての足掛かりとして，スレッドの実行順序を決定するソフトウェアであるスレッドスケジューラーが適しているという仮説を立て，グリーンスレッドスケジューラーをRustで作成します．これがシステムプログラミングに入門するにあたっての足掛かりとして適していると考えている理由は次の通りです：

- 高級プログラミング言語（Rust）からアセンブリによる低レイヤー操作まで広く触れられる
- システムコール，シグナル，割り込み，スタック，メモリオーダリング等の周辺知識にも多く触れることができる
- スケジューラーの機能面での拡張の余地が大きい

本記事の内容がシステムプログラミングに興味を持つきっかけとなり，より様々な技術に触れる一助となれば幸いです．

本記事の読み方

本記事で説明すること・しないこと

本記事の主たる目的は前述した通り，システムプログラミングに入門する足掛かり的な役割として読者の方々にスレッドスケジューラーを自作していただくことにあります．しかしながら紙面の都合上，実装にあたって必要な前提知識について全てを説明しきることは到底できません．

よって本記事では基本的に応用情報技術者試験レベルの知識を前提として，スレッドスケジューラーの実装過程での醍醐味的な周辺知識の説明に集中することとします．また，実装に関してもRustそのものの文法等については説明を割愛しますので予めご了承ください．

以上の点により，もし不明な概念が登場した場合は適宜手元での検索を頼ってください．最近だとChatGPTをはじめとしたLLMも理解の強力な一助となってくれることでしょう．

サンプル実装

本記事で実装するスレッドスケジューラーのサンプル実装はhttps://github.com/lapla-cogito/lachesisリポジトリで公開しており，次の環境で行われました：

```
$ rustc -V
rustc 1.90.0 (1159e78c4 2025-09-14)
```

この実装は次の環境での`cargo test`及び`cargo run`による動作確認を行っています：

- WSL2 on Windows 11 Home on Intel Core i7-13700F
- Arch Linux on ThinkPad X280
- macOS Sequoia on M4 MacBook Pro

HTML + CSS だけで JAN バーコード生成

Ryoga（@Ryoga_exe）

はじめに

CSS 黒魔術へようこそ。Ryoga[1] と申します。一時期 HTML + CSS だけで何かを作ることにハマっていたことがあり、CSS を書くたびにその表現力の高さに驚かされることがしばしばあります。

「CSS（+ HTML）がチューリング完全である」という話は有名[2] ですが、実際に CSS だけで何か実用的なものを作る試みは、あまり多くないように思えます。

そこで本記事では、CSS だけでバーコードを生成するコードを書き、その可能性を探っていきます。

要件

「barcode css」などでググってみるといくつか CSS でバーコードを描画するようなプロジェクトがヒットしますが、私が調べた限り、

- ユーザが値を入力できる
- ユーザの入力に応じてバーコードを生成する
- 入力のハンドリングも含め、JavaScript を一切使わない
- 特殊フォントに依存しない

を実現しているものはありませんでした。

本記事では、上記を満たす HTML + CSS のみのバーコード生成器を実装することとします。

JAN バーコードの仕様

バーコード生成器を実装するには、当然その仕様を理解する必要があります。

しかし、バーコードと言ってもこの世には様々なバーコードの種類が存在します。今回は、POS レジなどで広く使われているバーコードである JAN コード（EAN コード）を扱います。

さらに JAN コードには標準コード（EAN-13 / 13 桁）と短縮コード（EAN-8 / 8 桁）があり、両者で符号化の手順が一部異なりますが、ここでは 13 桁の標準コード（EAN-13）を対象とします。

次節から、詳細な仕様について確認していきます。

チェックデジット

EAN-13 では、先頭 12 桁がデータ、末尾 1 桁がチェックデジットです。チェックデジットは次の手順で求められます（右端＝最下位桁から数える）。

1. 右端から見て奇数位置の桁の合計を S_o、偶数位置の桁の合計を S_e とする。
2. $T = 3 \times S_o + S_e$ を計算する。
3. チェックデジット C は $C = (10 - (T \bmod 10)) \bmod 10$ で求める。

最後にもう一度 mod 10 を取っているのは、「10 から引いた結果が 10 になる」パターンを 0〜9 の一桁に正規化するためです。

例として `490210210764`（先頭 12 桁）を例にすると、以下のように計算されます。

1. • $S_o = 9 + 2 + 0 + 1 + 7 + 4 = 23$
   • $S_e = 4 + 0 + 1 + 2 + 0 + 6 = 13$
2. $T = 23 \times 3 + 13 = 82$
3. $C = 10 - 2 = 8$

よって、`490210210764` のチェックデジットは `8` です。

EAN-13 の構成

バーコードはモジュールとキャラクタという単位で構成されます。モジュールは、バーコードにおける一番細い線または空白です。キャラクタは数字一文字のことです。

EAN-13 は左から

- レフトマージン（Quiet Zone）
- レフトガードバー（Start/Left Guard）
- レフトデータバー（Left Data）
- センターバー（Center Guard）

[1] https://x.com/Ryoga_exe

[2] https://hoo89.hatenablog.com/entry/2014/09/12/164712

君も最強のパッケージマネージャ、Cargo にコントリビュートしよう！

motorailgun（@motorailgun_s）

はじめに

こんにちは、motorailgun です。情報科学類に 3 年次編入でやってきてから早いもので 3 年目となりました。

筑波大学というところは素晴らしい場所で、コンピューターのソフトウェアからハードウェアまで、広範に興味を持った同士たちが自然に集まっています。その中には OSS に対するコントリビュート（貢献）を趣味にする人も多くいます。

この記事では、そんな周囲に影響されてコントリビュートを始めた私の学びや気付きを粛々と共有させていただきます。

なお、コントリビュートには様々な手段がありますが、今回の記事では主に機能改善や修正のためにコードを変更し、変更を投稿すること（パッチを投げること）を扱います。

Cargo について

私がパッチを投げているソフトウェアは cargo[10] といい、Rust という新興プログラミング言語向けのパッケージマネージャです。言語の公式から提供されており、パッケージ管理のしやすさや機能性、高速性などで定評があります[11]。現代の言語の例に漏れず Rust も充実したエコシステムを備えており、その中のパッケージ管理やビルドシステムを担うものとなります。

Rust 言語に触れたことのない方々向けに補足すると、Rust を利用するにあたってはほとんど cargo を利用することになります。もちろん、そのような使い方はコンパイラである rustc へのラッパーに過ぎないのですが、cargo が便利であるからこそそうした方が便利であるわけです。むしろ、rustc を直に利用する方法を知らないユーザーの方が多いのではないでしょうか。

cargo にはいくつかのサブコマンドが存在します。`cargo build` でビルド、`cargo fmt` でフォーマット、`cargo add` でリポジトリから依存ライブラリを追加、などです。`cargo --help` とすると、オプションやサブコマンドの一覧が見られます。

以下は、プロジェクトを作成して実行するまでの流れを示したものです。簡単でしょう？

```
# プロジェクトの作成
$ cargo new sandbox
    Creating binary (application)
`sandbox` package
note: see more `Cargo.toml` keys
and their definitions at https://
doc.rust-lang.org/cargo/reference/
manifest.html

$ cd sandbox
$ cat src/main.rs
fn main() {
    println!("Hello, world!");
}

$ cargo run
   Compiling sandbox v0.1.0 (/
private/tmp/sandbox)
    Finished `dev` profile
[unoptimized + debuginfo] target(s)
in 0.50s
     Running `target/debug/sandbox`
Hello, world!

$ sed -ie "s/world/Rust/g" src/
main.rs
```

[10] https://github.com/rust-lang/cargo

[11] C や C++のようなシステムプログラミング向け言語に、広く利用されるような存在がなかったことも一因ではあります。

HCI カーソル探訪

いなにわうどん（@kyoto_inaniwa）

はじめに

秋風の気持ち良い季節になりましたね. こんにちは, いなにわうどんです. 驚額の殿堂も 3 年目の出店となり, 遂に同人誌を出すことになりました！

さて, 最近私はヒューマンコンピュータインタラクション (HCI) という分野に関心を持っています. HCI とは人間と計算機のインタラクション(関わり合い)を探求する学問領域であり, コンピュータサイエンスのほかに人間工学等の要素も含んだ横断的, 学際的な分野です. PC, スマートフォン, ヘッドマウントディスプレイ, スマートウォッチ, ゲーム機等の幅広い端末に対して, 様々なインタフェースの研究が行われています.

そんな HCI の研究対象となってきた関心事の一つに**カーソル**[21]があります[22]. 本稿では, カーソルにまつわる法則や, これまでに提案されてきた多様なカーソルを紹介したうえで, Chrome 拡張機能としてオリジナルのカーソルを作ってみることを目的とします[23].

カーソルとポインティング

マウスやトラックパッド等のデバイスを用いて, ターゲットを指し示すことを**ポインティング**と呼びます. そして, ポインティング手法の性能を定式化したモデルとして, フィッツの法則 [1] という代表的な法則が存在します. これはターゲットまでの距離を D, ターゲットの幅を Wとするとき, 移動時間（ターゲットを選択するまでの時間）MT は以下の式で表されるというものです（a, b はそれぞれマウス等の装置に依存する定数).

$$MT = a + b \log_2 \left(\frac{D}{W} \right)$$

ポインティングではなるべく速く(かつ正確に)ターゲットを選択することが重視されます. 上記の式に基づくと, ターゲット幅を大きくするか, ターゲットまでの距離を小さくすることにより, 移動時間を短縮できることがわかります. しかしながら, ターゲットは所与のものとして与えられるため, 移動や変形が難しい状況も多いです. そこで, カーソル側を変化させることにより, 移動時間を短縮する試みが行われてきました. 次ページでは, これまでに行われてきた著名なカーソル研究をいくつか紹介します.

コラム①：画面端のオブジェクトは早く到達できる？

ページ数稼ぎのために 1 ページ目ながら早速コラムです. メニューバーやタスクバーといった頻繁にアクセスされるメニューは, 素早くアクセスするために画面端（エッジ）に配置されることが多いですが, この理屈もフィッツの法則を用いて説明することができます[24].

カーソルは画面端に到達するとその場で停止するため, エッジに存在するオブジェクトは, 実際のオブジェクトよりも大きな幅または高さを持っていると仮定することができます. ゆえに, フィッツの法則における W が増大し, MT が減少するというわけです. Appert ら [2] は, 2 回の実験を通じてエッジに配置されたターゲットのポインティング性能を調査しています. 実験結果より, エッジに配置されたターゲットは通常のポインティングよりも高速であり, またパフォーマンスにはカーソルの進入角度や視認性が影響することが示されました.

これを踏まえて, 私もページ端にカーソルを持っていくことで Overleaf のサイドメニューを開閉する

[21]最近は LLM と統合された IDE であるところの Cursor が検索クエリを奪っており, 解せない

[22]同様の意味を持つ用語として「ポインタ」がありますが, 本稿ではカーソルに統一します. ポインタ探訪と書くと C？と聞かれそう

[23]実装や技術メインの記事が多い中で読み物的な要素が強いですが, どうぞ箸休め的な感じでお読みください

[24]最近の共通テストにも出題されたらしい

[25]https://github.com/inaniwaudon/overleaf-13inch

[26]https://いなにわうどん.みんな

OT から実装する ドキュメント共同編集アプリ

あすと（@asuto）

はじめに

はじめまして，coins21 のあすとです．

皆さんは Google Docs (https://docs.google.com) や HackMD (https://hackmd.io) のようなドキュメント共同編集のためのアプリケーションを使ったことはありますか？

このようなアプリでは，複数のユーザーが並行して同じ箇所を編集しても，なぜかうまく同期されながら動いています．特に回線が不安定な環境で編集したときなどは，反映されるのが遅くなって不具合が起きないか不安になる人もいるかもしれません．ここでは，操作変換 (Operational Transformation，以下 OT) という技術が使われており，これにより操作の因果関係を保ちながら，並行して複数人の操作を受けても全てのクライアントが最終的に同じ状態に収束します．

車輪の再発明

ところで，ソフトウェアエンジニアの間では，既存のアプリやライブラリなどを時にはフルスクラッチ[41]で自分で再実装することで，根本的な仕組みを理解しながらソフトウェアの自作を楽しむ「車輪の再発明」[42]という文化が存在しています．他の記事の執筆者の中にもよくやっている人がいますね．

セルフホスト

また，Google や Microsoft などの営利企業の中央集権的なサービスは，自分の手が届かない場所にデータが保存される以上情報漏洩のリスクがあったり，サービスの仕様変更で不便になったり，障害などの要因でデータが失われてしまったりする可能性があります．そのため，自分にとって重要な情報はこのようなサービスにできるだけ依存せずに，自分の所有するコンピュータやサーバあるいは VPS といった，自己責任ではあるが自分自身で制御できる環境で自由にカスタマイズしてサービスを運用する「セルフホスト」[43]という文化も存在しています．

本記事の目的

なぜ私が今回のアプリを作ろうと考えたかと言うと，既存のサービスには書きづらいような，取り扱いに配慮が必要な文書を特定の人に見せて共同で編集したい場面が以前あり，かつ OT そのものに興味があったので，これを機に OT を一から実装してみようと思い，作ったという経緯です．

そこで本記事では，このドキュメント共同編集のための Web アプリの動作や実装について解説します．

アプリは以下の URL で公開しています．

https://docs.asuto.dev

> Warning
> **オンプレミスでの運用のため，一時的にアクセスできない場合があります．また，予告なく公開を停止する可能性があります．**

本アプリの実装は，以下のリポジトリにあります．

https://github.com/asuto15/coedit

また，誤字脱字や内容の訂正があれば，以下の URL に掲載します．

https://storage.asuto.dev/kyogaku-dendo/error-corrections.md

アプリの機能紹介

今回実装したアプリの主な機能は次のとおりです．なお誌面の都合上，以降は OT とサーバ・クライアント間の通信に関する機能を中心に解説していきます．

[41]既存のライブラリなどに依存せず全ての機能を自分で実装するという意味 https://e-words.jp/w/フルスクラッチ.html
[42]本記事ではこの言葉を学習目的の再実装として肯定的な意味で用いています https://ja.wikipedia.org/wiki/車輪の再発明
[43]読者の中にもこういうページを見るのが好きな人がいると思います https://www.reddit.com/r/selfhosted/

驚額の殿堂 3（スリー）の Web サイトを作った話

浅香ひなた（@akira__okumura）

はじめに

皆様、はじめまして！浅香ひなたです。私は普段、デジタルアーカイブの研究を行っている元mast21、現情報学学位プログラム[44]M1 です。

さて、そんな私はこの度、驚額の殿堂 3（スリー）の Web サイト[45]の開発担当をいたしました！ 普段 Web 開発などはあまり行っていないのですが、今年は驚額の殿堂の技術班の皆様がフルパワーでいろんなシステムを構築しており[46]、巡り巡って Web 開発が私の元に降ってきました。

今年は同人誌も出すということで、せっかくなので Web 開発の裏話や小ネタを紹介しようかなと思いこの記事を書いています。技術屋さんじゃなくても楽しめると思います！ 楽しんで読んでいただけたらと思います！

コンセプト、設計思想

毎年奇抜な Web サイトを作っている驚額の殿堂ですが、今年のサイトのコンセプトは**「manaba の UI っぽくする」**でした[47]。例年は、90 年代や 00 年代を思わせるような秀逸なデザインでしたが、ここにきて最も大学生に親しみのある Web サイト[48]のパロディをすることになりました。学園祭の企画なので多めに見てください……

開発にあたっては、さまざまなライブラリやフレームワークがあり、比較的簡単に Web 開発ができるこのご時世にまさかの **HTML/CSS/バニラ JS だけ**でサイトを構築しております。例年の名残です。技術屋さんでなくてもこの 3 つの技術を理解すれば誰でも manaba 風サイトが作れます！ それぞれ役割が違うので、ざっと書いておきます。

HTML とは

まずは Web サイトの「骨組み」になるのが HTML（エイチティーエムエル）です。これがないとページは存在しません。HTML は「ここは見出し」「ここは本文」「ここはボタン」みたいな、文書の構造や役割をコンピューターが理解できるようにするための言語です。Web ブラウザや検索エンジン、そして支援技術（音声読み上げなど）も、HTML の“意味”を読んで動いてくれます。

CSS とは

次に出てくるのが CSS（シーエスエス）。これは HTML にデザインを着せる役割を持っています。文字の色や大きさ、余白、配置など「見た目の部分」はぜんぶ CSS が担当です。banana では、「本家 manaba の雰囲気」を出すために、たくさんのパーツを CSS だけで再現しています。

JavaScript とは

JavaScript（ジャバスクリプト）。これはページを動かす仕組みを作れます。ボタンを押したらメニューが開く、日付が自動で表示される、画面を切り替えられる――そういう「動き」はぜんぶ JavaScript が担っています。今回はあんまり攻めた使い方はしていないですが、結構なんでもできます。

---

[44]正式名称は『筑波大学人間総合科学学術院 人間総合科学研究群 情報学学位プログラム』。長い。

[45]https://kyogaku.yokohama.dev

[46]本当に色々なシステムを 1 から作っていてすごすぎる。

[47]攻めすぎ

[48]浅香調べ