The primary objective of this document is to facilitate a conceptual comprehension of “What constitutes system development” for individuals lacking IT skills and industry experience. This endeavor is not aimed at presenting efficient methodologies or challenging widely accepted norms; rather, it seeks to methodically consolidate information for those desiring a fundamental grasp of essential points.

Contents

I Console 3
I-1 コンソール 3
I-2 ターミナルとシェル 3
I-3 コマンド 3
I-4 パイプとリダイレクト 4
I-5 正規表現 4
I-6 正規表現の例 4
II An Introduction to Coding in Rust 4
II-1 コーディング 4
II-2 変数 5
II-3 型 5
II-4 関数 5
II-5 コメント 5
II-6 制御フロー 5
II-7 所有権 5
II-8 非同期処理 5
II-9 構造体 5
II-10 Trait 5
II-11 例題 5
III Algorithm 6
III-1 空間計算量 6
III-2 バブルソート 6
III-3 選択ソート 6
III-4 挿入ソート 6
III-5 シェルソート 6
III-6 バケットソート 6
III-7 分布数え上げソート 6
III-8 基数ソート 6
III-9 マージソート 6
III-10 ヒープソート 6
III-11 クイックソート 6
III-12 二分探索 6
III-13 フィボナッチ数列 6
III-14 線形探索 7
III-15 三文探索 7
III-16 N-Queen 問題 7
III-17 数独 7
III-18 最大和問題 7
III-19 ナップサック問題 7
III-20 部分和問題 7
III-21 最小個数部分和問題 7
III-22 最長共通部分列問題 (Longest Common Subsequence) 7
III-23 レーベンシュタイン距離 7
IV Data Analytics 7
IV-1 単回帰分析 7
IV-2 重回帰分析 7
IV-3 最小二乗法 7
IV-4 最尤法 7
IV-5 スプライン回帰 7
IV-6 ロジスティック回帰モデル 7
IV-7 線形判別 7

IV-8 フィッシャー判別 7
IV-9 k 近傍法 7
IV-10 主成分分析 7
IV-11 階層的クラスタリング 7
IV-12 混合正規分布 7
IV-13 スペクトラルクラスタリング 7
V Category Theory 7
V-1 圏論とは 7
VI An Introduction to Coding in Haskell 7
VI-1 純粋な関数プログラミング 7
VII Architecture 7
VII-1 システムの設計 7
VII-2 クリーンアーキテクチャへの招待 8
VII-3 単体テスト 8
VII-4 結合テスト 8
VII-5 総合テスト 8
VII-6 テスト駆動開発 8
VIII Git 8
VIII-1 分散型バージョン管理システム 8
VIII-2 Git の仕組み 8
VIII-3 ブランチの運用 9
IX Web 9
IX-1 沿革 9
IX-2 JavaScript 9
IX-3 JavaScript エンジン 10
IX-4 JavaScript ランタイム 10
IX-5 JavaScript を動かしてみる 10
IX-6 DOM 11
IX-7 Web アセンブリ 11
IX-8 Deno 11
IX-9 JSON Web Token 11
IX-10 Remix 11
X Database 11
X-1 データベースとは 11
X-2 SQL 11
X-3 NoSQL 11
XI Network Infrastructure 11
XI-1 インターネットとネットワークの定義 11
XI-2 インターネットの仕組み 11
XI-3 基礎通信理論 12
XII Hardware 12
XII-1 PC の内容 12
XIII Technical Writing 13
XIV Typst 13
Bibliography 13
APPENDIX A: Foo 13
APPENDIX B: Bar 13

I Console

I -1 コンソール

コンソールとは，OS を搭載したコンピュータに接続されたディスプレイおよびキーボードである． コンピュータとの対話を可能にするための物理ハードウェア，入出力システムともいえる．

「文字だけの黒い画面」というキーワードで一般に想起される画面は，仮想コンソールである． これは入出力を仮想的な空間に提供するものであり，この環境は CUI（Character-based User Interface）ともいう． CUI は，我々が日々利用する GUI（Graphical User Interface）と対をなす存在であり，文字に基づいたユーザーインターフェースを提供する．

インターフェース

“Interface"という語はハードウェアでもソフトウェアでも登場する．主に情報産業では，異なる二つのモノ(人間や機器)を接触させるための境界面として用いられる語である．

I -2 ターミナルとシェル

仮想コンソール（ CUI ）を提供するアプリケーションは一般に*ターミナル*と呼ばれる． Windows におけるコマンドプロンプトや，MacOS における iTerm2 や Warp などがターミナルにあたる． これらは GUI 上で CUI を操作するための窓口になる．

本題となるコンピュータへの命令は*コマンド*という． コマンドを解釈し，実行する仕組みをシェルという． シェルの役割はコンピュータとの対話を実際に担うアプリケーションであり，Bash や zsh，fish など種類が存在する．

シェル

エンジニアでない一般ユーザが OS と対話するためのシェルに，Windows のエクスプローラーなどがある．CUI のシェルと GUI のシェル，操作しやすいユーザ層は当然異なる．

要するに，エンジニアがコンピュータへ命令するためには基本的に「ターミナルを起動し(シェルを介して)コマンドを入力」する． また，コマンドを入力する行のことをコマンドラインという． 単語が重要なのではなく，この CUI 構成を理解することは今後役に立つ．

I -3 コマンド

ここでは，基本的なコマンドを紹介する． 細かいオプションまでは解説をしない． なお，ここで使用するコマンドは Windows コマンドプロンプトでは使用できない．

I -3 -1 ファイルの一覧化： ls

まずはフォルダの内容を確認したい． そのようなときに打つコマンドである． 何回打っても良い．

```
1 ls
```
Command

I -3 -2 ディレクトリの移動： cd

今いるフォルダから移動したいときに打つ．

```
1 cd {path} // {移動先のパス}へ移動
2 cd ..     //  1つ上のディレクトリへ移動
3 cd ~      //  ホームへ移動
```
Command

コマンドライン

本書では，コマンドラインへの入力表記において{}を代入記号，//をコメントとする．これらは実際に入力せず，可読性を補助する．

I -3 -3 現在地 ： pwd

プロンプトに書いてあるディレクトリ名をわざわざ絶対パスで表示する． あまり使わないが ,重大な作業をしているときに使った記憶がある．

```
1 pwd
```
Command

I -3 -4 ディレクトリの作成 ： mkdir

思ったよりお世話になる． メイクディレクトリと読んでしまうが正解は知らない． 知っておいて損はない．

```
1 mkdir {ディレクトリの名前}
```
Command

I -3 -5 ファイルの閲覧 ： cat

本来は複数のファイルを結合(concat)して表示するコマンドだが，単一のファイルをのぞき見することで使うことが多い．

```
1 cat {ファイル名}
```
Command

I -3 -6 ページャ ： less

本来のファイル閲覧用コマンドだが，大体は cat コマンドで足りてしまう． less を使用した後は，Q キーを押下すると終了する．

```
1 less {ファイル名}
```
Command

I -3 -7 容量の確認 : du

「このフォルダは何 MB あるのか」といった疑問を解決する． ちなみに，-hs オプションを付与すると内訳を非表示にできるため，ファイル数が異常に多いディレクトリで有効である．

```
1 du {フォルダ名}                    Command
```

I -3 -8 検索 : grep

ファイルを検索することは多々ある． よく使うオプションつきで紹介する．

```
1 grep -rn {path} -e ${text}         Command
```

- r : 再帰的にサブディレクトリを含める
- n : マッチした行番号を表示
- e : 検索する文字列を指定

I -3 -9 改名もしくは移動 : mv

便利だが，個人的にリスクがあると思っているコマンドの１つである． ２つめの引数が存在するパスならそこへ１つ目の引数に指定されたファイルを移動させる． 存在しないパスなら，１つ目の引数に指定されたファイルを２つ目の文字列に改名する．

```
1 mv {path} {path} // 改名           Command
2 mv {path} {rename} // 新しい名前に変更
```

移動先を誤字脱字すると，その文字列が新しい名前となるので注意が必要である． ただし，-iv オプションを付与すると警告してくれるので必須である．

I -3 -10 削除 : rm

mv より危険なコマンドである． 削除をコマンドで実施する場合，取り返しがつかない (= ゴミ箱に移動しない)ため，-iv オプションを付与して警告を有りにする習慣をつけよう．

```
1 rm -iv {対象ファイル}              Command
```

I -4 パイプとリダイレクト

I -5 正規表現

正規表現は，文字列のパターンを表現するための記法である． これにより，特定の条件を満たす文字列を効果的に検索したり，置換したりすることができる． 基本的な正規表現を以下に示すが，正規表現のあとにコロン:を付与するため注意してほしい．

I -5 -1 Character Class : 文字

- [abc]: いずれか一つ
  - => a, b, c
- [^abc]: 含まない
  - => d など

I -5 -2 Metacharacters : メタ文字

- .: 任意の 1 文字
- \n: 改行コード
- ^: 行の先頭
- $: 行の末尾
- \d: 半角数字
- \s: 空白
- \S: 空白以外すべて
- \w:半角英数字とアンダースコア
- \l:半角英小文字
- \u:半角英大文字

I -5 -3 Quantifiers : 量指定

- *: 直前の要素が 0 回以上繰り返し(ワイルドカードともいう)
- +: 直前の要素が 1 回以上繰り返し
- ?: 直前の要素が 0 回または 1 回
- {n}: 直前のパターンを n 回繰り返し

I -5 -4 Grouping : グルーピング

- (): 括弧内のパターンを 1 つの要素として扱う

I -6 正規表現の例

電話番号は以下の正規表現が適用できる．

```
^(\d{3}-\d{4}-\d{4}|\d{10})$
```

ハイフンを含む形式 (例: 123-4567-8901) またはハイフンなしの形式(例: 1234567890)にマッチする．

II An Introduction to Coding in Rust

II -1 コーディング

プログラムを開発する作業全般をプログラミングといい ,特にソースコードを書く工程をコーディングという． コーディングにおいて，シンタックスとデザインパターンは中核を成す．

シンタックスとは，プログラミング言語の仕様として定められた構文規則を指す． デザインパターンについては Wiki[1] を引用する．

> ソフトウェア開発におけるデザインパターンまたは設計パターン (英: design pattern) とは，過去のソフトウェア設計者が発見し編み出した設計ノウハウを蓄積し，名前をつけ，再利用しやすいように特定の規約に従ってカタログ化したものである．
>
> · Wikipedia

これより， シンタックスに関連する事項を説明する． 通常，伝統的なアプローチでは「ハローワールド関数」から始めることが一般的だが，本書では変数に関する説明から始める．

[1] https://w.wiki/rvm

II -2 変数

以下のコードはどのような処理か考えてみよう．

```rust
let x = 4;
```

x は変数，4 は整数リテラル[2]である． それと同時に， x は 4 を束縛 (bind) する所有者である． let は変数宣言のキーワードとなっており，以下の構文を想定している．

```rust
let PATTERN = EXPRESSION;
```

整数リテラル 4 単体であっても， 右辺は式である． 式が返却する値は 4 であり，左辺のパターン x に当てはめて束縛される．

パターンであるという認識はとても重要で，以下はエラーにならない．

```rust
let (a, b) = (4, 5);
```

ただし， パターンには 2 つの形式が存在する． 上記の _Irrefutable(論駁不可能)なパターンと， *let 式で扱うことができない _Refutable*(論駁可能)なパターンである． この _Refutability(*論駁可能性*)の概念については， *エラーメッセージで見かけた際に対応できるように，慣れておく必要がある*． Refutable_ なパターンはまた別の章で触れる．

変数宣言は，値 4 に別のラベル x を張り替えるような作業ではなく，所有者 x に 4 が束縛されるという認識をもってほしい．

II -3 型

Rust 標準のプリミティブ型を表にした． 型の名前に含まれる数字は， ポインタのサイズ (ビット)を指す．

基本データ型	型
符号付き整数	isize, i8, i16, i32, i64
符号無し整数	usize, u8, u16, u32, u64
浮動小数点数	f32, f64
文字 (Unicode のスカラー値)	char

特に数値を表す型の許容値は以下となる．

型	MIN	MAX
i8	-128	127
i16	-32768	32767
i32	-2^{32}	$2^{32}-1$
i64	-2^{64}	$-2^{64}-1$
u8	0	255
u16	0	65535
u32	0	2^{32}
u64	0	2^{64}
f32	凡そ -21 億	凡そ +21 億
f64	凡そ -9 京	凡そ +9 京

isize および usize は実行環境 PC の CPU のビット数によって適切なサイズに変化する．

II -3 -1 型推論

初期化の際に評価値の型をチェックするだけでなく， その後にどのような使われ方をしているかを見て推論する．

II -4 関数

ハローワールドプログラムを観察する．

```rust
fn main() {
    println!("Hello, world!");
}
```

II -5 コメント

II -6 制御フロー

II -6 -1 if

II -6 -2 loop

II -6 -3 while

II -6 -4 for

II -7 所有権

II -8 非同期処理

II -9 構造体

II -10 Trait

II -11 例題

II -11 -1 じゃんけん

Bob と Carol は， 以下のルールに従って N 回じゃんけんを行った．

[2] リテラルとはソースコードに生で書かれたデータであり，値そのものである．例えば，4(リテラル)に 5(値) を束縛することはできない．

- グー> チョキ，パー> グー，チョキ> パー
- 勝者に1点
- 勝者が連続したとき，+1点多く付与

$$\text{input} \leftarrow \begin{pmatrix} N & \\ B_i & C_i \\ \vdots & \vdots \\ B_N & C_N \end{pmatrix}$$

Bob と Carol の得点をそれぞれ計算せよ． ただし，以下の条件を制約とする．

- $1 \leq N \leq 300$
- $B_i, C_i \in \{1, 2, 3\}$

II -11 -2 ナンバー 2

N個の整数列が与えられるとき， その中で 2 番目に大きな数と小さな数をそれぞれ出力せよ． ただし，この整数列は重複する数を含む場合がある．

$$\text{input} \leftarrow \begin{pmatrix} N & \\ x_0 & \cdots \ x_N \end{pmatrix}$$

II -11 -3 門番

N行M列のマス目がある． N行1列目のマス目がスタート地点，1行M列目がゴール地点とする． スタート地点とゴール地点以外のマス目にはそれぞれ一人ずつ門番が入っている． i行j列目からi行$j+1$列目のマスに移動するには，門番へ$p_{\mathrm{i\{j+1\}}}$円支払う必要がある．ゴール地点までへ到達するまでにかかるコストの最小値を求めよ． ただし，以下の条件を制約とする．

$$\text{input} \leftarrow \begin{pmatrix} N & M & & \\ p_{00} & p_{01} & \cdots & p_{0\mathrm{M}} \\ p_{10} & p_{11} & \cdots & p_{1\mathrm{M}} \\ \vdots & \vdots & \ddots & \vdots \\ p_{\mathrm{N}0} & p_{\mathrm{N}0} & \cdots & p_{\mathrm{NM}} \end{pmatrix}$$

$$p_{\mathrm{N}0}, p_{0\mathrm{M}} = 0$$

- 斜め移動不可

II -11 -4 文字列

III Algorithm

III -1 空間計算量

プログラムで問題を処理するとき，条件分岐ifや繰り返しforを組み合わせることが多々ある． 問題が解けるまでに要する手順（ステップ）の数を時間計算量といい，かたや問題が解けるまでに必要なメモリサイズを空間計算量という． 空間計算量も時間計算量も少ないアルゴリズムが優れたアルゴリズムであるが，一般的にはメモリを多く費やせば短時間で解け， 長時間かければ少ないメモリ容量で済むという「時間と空間のトレードオフ」の関係がある．

同じ問題に対して，複数のアルゴリズムが存在している中， 解決したい問題のデータ量が増大しても計算量が膨大に膨れ上がらないアルゴリズムは優秀といえる．

III -1 -1 ランダウ記法

ランダウ記法$\mathcal{O}()$は， アルゴリズムの時間計算量や空間計算量を大まかに評価するために使われる． アルゴリズムが入力サイズに対してどの程度の計算時間やメモリを必要とするかを表す．

III -2 バブルソート

バブルソートの最悪の計算量は$\mathcal{O}(n^2)$である．全ての要素を比較し， 必要に応じてスワップ操作を行うことにより， 入力配列が逆順にソートされている場合は最大限の比較とスワップが必要になる．

```rust
fn bubble_sort<T: Ord>(arr: &mut [T]) {
    if arr.is_empty() {
        return;
    }
    let mut sorted = false;
    let mut n = arr.len();
    while !sorted {
        sorted = true;
        for i in 0..n - 1 {
            if arr[i] > arr[i + 1] {
                arr.swap(i, i + 1);
                sorted = false;
            }
        }
        n -= 1;
    }
}
```

III -3 選択ソート

III -4 挿入ソート

III -5 シェルソート

III -6 バケットソート

III -7 分布数え上げソート

III -8 基数ソート

III -9 マージソート

III -10 ヒープソート

III -11 クイックソート

III -12 二分探索

III -13 フィボナッチ数列

III -14 線形探索

III -15 三文探索

III -16 N-Queen 問題

III -17 数独

III -18 最大和問題

III -19 ナップサック問題

III -20 部分和問題

III -21 最小個数部分和問題

III -22 最長共通部分列問題 (Longest Common Subsequence)

III -23 レーベンシュタイン距離

IV Data Analytics

IV -1 単回帰分析

IV -2 重回帰分析

IV -3 最小二乗法

IV -4 最尤法

IV -5 スプライン回帰

IV -6 ロジスティック回帰モデル

IV -7 線形判別

IV -8 フィッシャー判別

IV -9 k 近傍法

IV -10 主成分分析

IV -11 階層的クラスタリング

IV -12 混合正規分布

IV -13 スペクトラルクラスタリング

V Category Theory

V -1 圏論とは

VI An Introduction to Coding in Haskell

VI -1 純粋な関数プログラミング

VII Architecture

VII -1 システムの設計

実際，main 関数のみ定義し，そこにすべてを定義した main.rs ファイルのみで本番稼働させているシステムは限りなく 0 に近い．main 関数は数行書き，最小限の責務を担う関数を複数定義してそれらを組み合わせて成り立たせるのが一般的だろう．変数名xやyではなく，わかりやすい英語を使うだろう．もちろん，main 関数にすべて書いた巨大な main 関数でも，変数名a, b, cだらけでも同じ挙動はするのだが，様々な問題がある．

- 作者(コーダー)が離職･入院･異動したとき
- 障害が発生したとき
- リファクタリングを実施するとき

上記のような"保守開発"フェーズでは，なにより保守性がキーとなる．誰が読んでも，同じような改修の仕方になるような，そしてすぐ理解できるようなものがいいのである．保守性が高いシステムは何か起きてもエラーの種類によってはすぐ改修することができるだろう．これは運用サービスの品質に直結する．

品質のために保守性を担保するのは当然のことなのだが，それ以前に SE としてのマナーでもあってほしい．誰かが保守に困るようなものを世にリリースするべきではない．また，業種にもよるが，感覚的に殆どの SE という職種はプロジェクト単位で参画したり退却したりする．転職が珍しくない環境下で，同じ担当者が何年も同じ会社にいると思わない方が良い．

さて，設計論は 1 つのベストプラクティスがあるわけではなく，複数存在する．ここではその中の 1 つのみ案内する．

VII -2 クリーンアーキテクチャへの招待

ドメインモデル，ユースケース，プレゼンテーションに分かれる．これらは輪となり，変更が少ない情報ほど内に配置される．

ドメインモデルは設計の核であり，そのシステムを開発することになった発端でもある．この核はシステム開発となんら関係ない普遍的なものになり，一般にビジネスロジックという．例えば，「じゃんけんではグーがチョキに勝つ」等の情報レベルで，ここに変更を入れることはできない．システム仕様ではなく，原則や規則にあたる．

ユースケースは，ドメインモデルを取扱い，ユーザ(クライアント)からの指令を果たす．例えば，チョキとグーという二つの入力を受け取り，どちらが勝者なのか判定を返却する．ただし，ユースケース自身は「チョキよりグーの方が強い」ことに関心は無く，あくまでドメインモデルに問合せているに過ぎない．また，問合せ結果を返却する責務に留まり，クライアントに対して直接「勝ちました /負けました」を表示するのは，プレゼンテーションが担当する．

プレゼンテーションは常にクライアントと対峙している．フォントサイズ，色，位置，テキスト入力など，システムとユーザが接する窓口 (インターフェース) である．飲食店の場合，ホールがプレゼンテーション，料理をマニュアル通りに調理するキッチンがユースケース，料理そのものがドメインモデルに相当する．

実際の開発プロセスでは，より複雑な概念や抽象的な事象を捉えていく必要があるため，ここではざっくりとした説明に留めた．エンティティと値オブジェクト，ドメイン貧血，インフラストラクチャー層や依存性逆転の法則にまだ触れていないため，じゃんけんや飲食店の例は当たらずといえども遠からずである．それぞれの責務ごとに明確にレイヤーを隔離することにより，テスタビリティや保守性を担保する．

VII -3 単体テスト

VII -4 結合テスト

VII -5 総合テスト

VII -6 テスト駆動開発

VIII Git

VIII -1 分散型バージョン管理システム

VIII -1 -1 Git と GitHub

VIII -2 Git の仕組み

VIII -2 -1 ワーキングディレクトリ

ワーキングディレクトリは，Git リポジトリ内のファイルやディレクトリが実際に存在する場所である．開発者は，このディレクトリ内でファイルを作成，編集，削除する．ワーキングディレクトリ内の変更は，Git がトラッキングし，コミットする前にステージングに追加する必要がある．

VIII -2 -2 ステージング

ステージングは，ワーキングディレクトリから Git リポジトリに変更を追加するプロセスである．開発者は，変更したファイルをステージングに追加し，次にコミットするための準備を整える．これにより，コミットする変更を選択的に管理し，変更の内容やコミットの粒度を制御することができる．

VIII -2 -3 リポジトリ

リポジトリは，Git がバージョン管理されたファイルや履歴を保存するデータベースである．プロジェクトの全ての変更履歴やメタ

データが保存され，複数のブランチやコミットからなる完全な履歴が管理される． ローカルリポジトリとリモートリポジトリの 2 つのタイプがあり，開発者はローカルで作業を行い，変更をリモートリポジトリにプッシュして共有する．

VIII -2 -4 コミット

コミットは，Git リポジトリに変更を永続化する操作である． 開発者は，ワーキングディレクトリ内で変更を行った後，変更をステージングしてコミットする． 変更内容の説明や作成者の情報などのメタデータが含まれ，プロジェクトの変更履歴に新しいスナップショットを追加し，過去の状態に戻るための参照ポイントを提供する．

VIII -2 -5 プッシュ

プッシュは，ローカルリポジトリの変更をリモートリポジトリに送信する操作である． 開発者は，ローカルで行ったコミットをリモートに反映させるためにプッシュする． これにより，チームメンバー間でのコードの共有や同期が可能になります．

VIII -2 -6 プル

プルは，リモートリポジトリからローカルリポジトリに変更を取得する操作である． 開発者は，リモートで行われた変更をローカルに反映させるためにプルを実行する． これにより，他のチームメンバーの変更を取得し，自分の作業と統合することができる．

VIII -2 -7 ブランチ

ブランチは，プロジェクトの異なるバージョンや機能ごとに分岐するための機能である．開発者は，新しいブランチを作成し，独立した作業を行うことができる． これにより，複数の作業を同時に進行し，コードの変更を安全に管理することが可能になる．

VIII -3 ブランチの運用

IX Web

IX -1 沿革

本格的にワールドワイドウェブ（ www ）が普及したのは 1995 年頃だった． 当時は静的な HTML（ HyperText Markup Language ）ページが主流だった． HTML は情報を構造化し，文書の意味や見出し，段落，リストなどを表現していた． ページのデザインやスタイルは限定的だったが，これが Web の基盤となり，今日の進化したウェブページの基礎となった．

ユーザはインターネットブラウザを介して Web サーバと通信し， HTML を取得している．さらに， CSS や JavaScript などがページデザインを制御することにより， インタラクティブな体験を得られている．

IX -1 -1 MPA

ブラウザは Web サーバへ HTTP リクエストを送信する． Web サーバ内で HTML ページを構築し，JavaScript や CSS とともにに HTTP レスポンスを返却する． この処理をリクエストのたびに繰り返すため， ページ遷移の度に読み込みが発生する． この仕組みを Multiple Page Application(MPA) という． MPA はページ遷移の速度に問題を抱えていた．

IX -1 -2 SPA

MPA のデメリットを解消する仕組みが Single Page Application(SPA) である． MPA は，SPA のように毎回フルの HTML ページを返却するのではなく，差分のみを更新する． Ajax[3]を使用し，差分情報のリクエストを送信し，サーバサイドは差分箇所を示す Json を返却する． 差分のみの更新は MPA と比べ高速であり， ページ遷移が快適になる． しかし，初回アクセス時に全ページの描写に必要な JavaScript を取得するため，ローディングコストが発生し，読み込みが重くなる． 代表的なライブラリに React がある．

IX -1 -3 SSG

Static Site Generation(SSG) は，ビルド時に静的な HTML ファイルを生成する． アプリをビルドするたびに， 全てのページが作成される．ユーザーが Web サイトにアクセスするとこれらのファイルをロードされ， サーバーは余計な作業をする必要がなくなる． ホームページやブログなどの更新頻度が少ない静的コンテンツに適切なアプローチである． 代表的なフレームワークに Next.js がある．

IX -1 -4 SSR

SSR(Server Side Rendering) とは， 初期表示速度の問題を解決するために， サーバーサイドでページの内容をレンダリングするアプローチである． ユーザーは最初のページロードで完全なページコンテンツを受け取る． 一度ページを読み込むと， その後のデータの取得や更新は最小限のリソースで行えるため，ネットワークトラフィックやサーバーロードを軽減できる． SSR ページはリクエストごとにサーバー上で生成されるため， 常に最新のデータを表示する． 通販サイトや SNS プラットフォームのような， パーソナライズされた（動的）コンテンツを持つアプリケーションに適切である． 代表的なフレームワークに Remix がある．

IX -2 JavaScript

Javascript は 1995 年 に Brendan Eich 氏 によって開発された． クライアントサイド[4]で動

[3] Web ページを表示した状態のまま，新しいページの読込などを伴わずに Web サーバ側と通信し， 非同期的に表示内容を変更する技術．

[4] サーバではなくユーザー側のモバイルや PC 上を指す．

作するスクリプト言語として誕生し，HTML に動きや対話性をもたらした．

HTML で定義された要素に対し，イベントや条件に応じて操作や変更を行うことも機能の一つである． 例えば，ボタンをクリックしたときにアラートを表示したり，入力フォームの値を検証したり，画像を切り替えたり，アニメーションやゲームを作ることも可能である． JavaScript ライブラリ[5]を利用して高度な Web アプリケーションも開発されている．

著名な SNS もすべて，ブラウザ上では JavaScript が動いている．

IX -3 JavaScript エンジン

JavaScript コードを解析するコアプログラムを JavaScript エンジンという． Web ブラウザはエンジンを搭載されているため，JavaScript を実行できる．

Browser	Engine
Google Chrome	V8
Microsoft Edge	V8
FireFox	SpiderMonkey
Safari	JavaScriptCore

JavaScript エンジンを独自に再開発するプロジェクト[6]も存在する．

IX -4 JavaScript ランタイム

JavaScript エンジンを含む，JavaScript 実行環境をランタイムという．

- Node.js(V8 を採用)
- Bun.sh(JavaScriptCore を採用)
- Deno(V8 を採用)

IX -5 JavaScript を動かしてみる

任意のディレクトリで新規の HTML ファイルを作成する．

```sh
toucn index.html
```

以下はいわゆるトースト通知を実装したスクリプトである． このままコピー＆ペーストし，index.html を Web ブラウザで開くと JavaScript の挙動を確認できる． JavaScript が埋め込まれた HTML ファイルは，特別なコマンドや操作は必要とせず，Web ブラウザに自動的に実行される．

```HTML
<!DOCTYPE html>
<html lang="jp">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Exaple</title>
  </head>
  <body>
    <div id="toast">
      <p>Hello, JavaScript!</p>
    </div>
    <button class="button">Enter</button>

    <script>
      document.addEventListener("DOMConte
function () {
        const toast = document.querySelector("#toast");
        const button = document.querySelector(".button");

        button.addEventListener("click", () => {
          toast.style.visibility = "visible";
          setTimeout(function () {
            toast.style.visibility = "hidden";
          }, 3000);
        });
      });
    </script>
  </body>
</html>
```

<script>タグで囲まれている箇所が JavaScript にあたる． 本書では，JavaScript のシンタックスについては触れないが，Coding セクションが理解できていれば問題ない． 既に Rust がプログラミング言語の中で母国語のような存在に昇格できていれば，おそらく JavaScript もある一定レベルまで読むことができるだろう．

[5] 流行り廃りは https://stateofjs.com から確認できる．

[6] https://github.com/boa-dev/boa

IX -6 DOM

IX -7 Web アセンブリ

IX -8 Deno

IX -9 JSON Web Token

IX -10 Remix

IX -10 -1 CDN

CDN(Contents Delivery Network) とは，Web コンテンツを迅速に効率よくユーザーに配信するためのネットワークである． コンテンツの大容量化やネット人口の増加が進む中， 昨今ではホームページの表示やコンテンツのダウンロードに時間がかかることも少なくない．CDN はコンテンツの表示・配信を高速化し，ユーザーのストレスを軽減する．

CDN では，キャッシュサーバと呼ばれる代理サーバが，オリジンサーバに代わってコンテンツを配信する． キャッシュサーバは，オリジンサーバからキャッシュしたコンテンツを保存している．

CDN ベンダ[7]はキャッシュサーバーを世界各地に分散して所有している． ユーザーからアクセス要求があると，オリジンサーバに代わり，物理的距離が最も近いキャッシュサーバが応答する．

オリジンサーバもコンテンツ配信を行うが，その負荷は CDN を利用しない場合より遥かに小さくなる． アクセス集中が抑えられ，表示・配信の遅延やサーバダウンのリスクが低くなる．

IX -10 -2 Remix

Remix は CDN エッジサービスである Cloudflare Workers で SSR を使える React ベースの Web フレームワークである．

X Database

X -1 データベースとは

X -2 SQL

X -3 NoSQL

XI Network Infrastructure

XI -1 インターネットとネットワークの定義

ネットワークとはコンピュータとコンピュータが繋がることで情報を通信できる仕組みであり，構造を指す． 外部ネットワーク同士が繋がりあい， 結果的に形成された地球上を駆け巡る巨大なネットワークの総称がインターネットである．

XI -2 インターネットの仕組み

総務省を引用する．[1]

> ネットワーク上で，情報やサービスを他のコンピュータに提供するコンピュータをサーバ，サーバから提供された情報やサービスを利用するコンピュータをクライアントと呼びます．私たちが普段使うパソコンや携帯電話，スマートフォンなどは，クライアントにあたります．
>
> · 総務省

XI -2 -1 IP アドレス

IP アドレスは， コンピューターやスマートフォンなどがインターネット上で通信するときに使われる， そのデバイスを特定するための「住所」のようなものである． 家の住所があるように， コンピューターもそれぞれが異なる IP アドレスを持っている． これにより，データがどのデバイスに届くかが決まる．

XI -2 -2 DNS

DNS は，Domain Name System（ドメインネームシステム）の略であり，人間が覚えやすいウェブサイトの名前（例: www.google.com ）を，コンピューターが理解しやすい IP アドレスに変換するシステムである． 言い換えると，DNS はインターネット上での住所録のようなものです． あるウェブサイトにアクセスするとき，DNS がそのウェブサイトの名前を IP アドレスに変換してくれます．

XI -2 -3 サブネットマスク

サブネットマスクは， IP アドレスをグループ分けする． これにより，同じネットワーク内のデバイス同士は通信しやすくなる． IP アドレスは通常， ネットワーク部とホスト部に分かれており，サブネットマスクは，どの部分がネットワークで， どの部分がホストであるかを示す． ネットワークを効果的に管理し，通信を効率的に行う目的がある．

XI -2 -4 ゲートウェイ

ゲートウェイは， コンピューターネットワークにおいて異なるプロトコルやネットワーク間で通信を中継・変換するためのデバイスやソフトウェアのことを指す． 簡単に言えば，異なる種類のネットワークやプロトコル間で通信を円滑に行うための出入り口のようなものである． 内部ネットワークと外部ネットワークの中継地である．

XI -2 -5 LAN

LAN（Local Area Network ）は， 狭い範囲内でコンピューターやデバイスがつながっているネットワークのことである． 学校内のコン

[7] Cloudflare, Amazon CloudFlont, Azure CDN, Google Cloud CDN 等

ピューター，家庭内のデバイス，あるいはオフィス内のコンピューター同士が，ケーブルや無線などで直接つながっている状態を指す．

XI -2 -6 ファイアウォール

ファイアウォールは，内部ネットワークを守るための防御壁のようなものである． 例えば，会社の警備員が不審な人が入ってこないように見張っているように，ネットワーク上においても不正アクセスや悪意あるプログラムから防衛する必要がある．

ファイアウォールは，ネットワークに入るデータや通信をチェックし，ウイルスや不正アクセスを遮断することで，コンピューターや情報を安全を確保する．

XI -2 -7 DMZ

DMZ は，DeMilitarized Zone(非武装地帯)の略であり，外部と内部のネットワークの間においてファイアウォールで隔離された中立的なセグメント(区域)を指す． 外部のユーザーやデバイスがネットワークにアクセスする際の対話エリアであり，ここに置かれたサーバーやシステムが外部からのアクセスを受け付け，内部の重要なデータやシステムに直接アクセスされないようにする．

外部ネットワークにはあらゆる脅威が存在する． DMZ は，内部への侵略を目論む攻撃者から内部情報資産を防衛するための仕組みとして機能する．

XI -3 基礎通信理論

XI -3 -1 プロトコル

プロトコルは，通信やデータ交換を行う際の取り決めや手順の規約である． これは，情報の送受信方法や通信の制御方法など，通信に関する様々なルールや規則を定義する． 例えば，インターネットで広く使われている TCP/IP プロトコルスイートは，データ転送の手順やエラーの処理方法などを規定する．

XI -3 -2 パケット

パケットは，データを転送する際の基本的な単位である． 大きなデータを複数のパケットに分割して送信し，それらのパケットがネットワークを通じて目的地に到着した後，再び組み立てて元のデータを再構築する． 宛先の情報，送信元の情報，データの一部，およびエラーチェックのための情報(ヘッダー)が含まれる．

XII Hardware

XII -1 PC の内容

XII -1 -1 マザーボード

マザーボードは，コンピューターの中核となる基板であり，他のコンポーネントが相互に通信し，協調して動作するための接続ポイントを提供する． CPU，RAM，ストレージ，グラフィックボードなどのコンポーネントがマザーボードに接続され，それらの間でデータや電力がやり取りされる． マザーボード上には，CPU ソケット，RAM スロット，拡張カードスロット(PCI Express や PCI)，ストレージ接続ポート(SATA や M.2)，USB ポート，ネットワークポートなどが備わっている．

XII -1 -2 CPU

CPU (Central Processing Unit)は，コンピューターの中央処理装置であり，プログラムの実行やデータの処理を担当する． 命令を解釈して実行し，算術演算や論理演算を行います． コンピューターの性能や処理速度は，CPU の種類やクロック周波数，コア数などによって決まる． 一般的な CPU には，インテルや AMD などのメーカーから製造されたものがある．

XII -1 -3 グラフィックボード

グラフィックボードは，コンピューターにおいてグラフィック処理を担当する拡張カードである． 高解像度のビデオや 3D グラフィックスを処理するために使用されます． グラフィックボードには，GPU (Graphics Processing Unit)が搭載されており，画像の描画や処理を高速かつ効率的に行う． ゲーミング PC やデザインワークステーションなどで高性能なグラフィックボードが利用される．

XII -1 -4 GPU

GPU (Graphics Processing Unit)は，グラフィック処理を担当するプロセッサであり，主に 3D グラフィックスの描画や処理に使用される． 大量の計算を並列処理することが得意であり，グラフィックスだけでなく，科学計算や機械学習などの分野でも利用されます． NVIDIA や AMD などのメーカーから，様々な用途に合わせた GPU が提供されている．

XII -1 -5 ストレージ

ストレージは，コンピューター内にデータを保持するための装置である． 一般的なストレージには，HDD (Hard Disk Drive)や SSD (Solid State Drive)がある． HDD は磁気ディスクを使用してデータを保存し，大容量のデータを格納することができる． 一方，SSD はフラッシュメモリを使用してデータを保存し，高速でアクセスが可能ある． OS やアプリケーション，ユーザーデータなどを永続的に保存する．

XII -1 -6 RAM

RAM (Random Access Memory)は，コンピューターが実行中のプログラムやデータを一時的に保持するためのメモリである． CPU がアクセスしやすい高速なメモリであり，データの読み書きが速く行える． コンピューターの処

理速度や多重タスク処理能力に直接影響を与えるため，十分な容量と高速なアクセス速度が要求される．

XIII Technical Writing

XIV Typst

Bibliography

[1] 総務省，“インターネットの仕組み”．[Online]. Available at: https://www.soumu.go.jp/main_sosiki/cybersecurity/kokumin/basic/basic_service_02.html

APPENDIX A: Foo

APPENDIX B: Bar