HUMAN PROGRAMMING

BEYOND THE VERBAL THINGS

Digital Clinic Vision

(下) ヒューマンプログラミング　編

2025.4.11

4 質の高い教育をみんなに

CommerceKitchen

※デザインにおいて画像生成AIを用いています。

はじめに:人と機械に共通のプログラミングを考える

コンピューターのプログラミングは、基本的には1と0を正しく並べるだけの簡単な仕事です!!!
奥は深いですが。。。

その、並び順、つまり指示が正しければ、コンピュータは必ずその指示に従うので人をプログラムするより、ずっと簡単。才能にあふれつつも、プライドがないので非常に有用な存在です。

人＝人と、人＝コンピュータのコミュニケーションが近づいてきた

※AIのことを言っているわけではありません。

ところで、プログラミングを先に知ってから、ビジネス英語やテクニカルコミュニケーションと言うのを学ぶと案外共通点が多いことに驚かされます。例えば

- 結論から先に言う文化
- Specific terms (専門用語など)を正しく使う
- 箇条書き
- 英語で、can, should, must を正しく使い分ける
- 英単語における難易度インデックス (どの単語はどのくらいの教育水準の人が理解できるかを数値化)
- 論理のピラミッド
- foot in door, face in door といった説得のロジック

これらは、メモリ管理、関数定義、クラス=インスタンスモデル、IDの付与、場合分け、ウォーターフォール、アジャイル、といったコンピュータサイエンスの成果と強いアナロジー(類似性)を感じさせます。

YouTubeを見ていると、英語圏の小学生が難なくプログラミング言語の内容を読み取ってしまうのに驚かされますが、実は英語で書いているからだけではなく、文化的なロジックも自然なのでしょう。

人と向き合っている人こそプログラミングに向く

時々、人と付き合うのがあまり得意ではないが、コンピューターのプログラミングだったら楽しいから、得意だからやっていると言う人がいます。それがダメだとは思いませんが、実務の世界で自分が作ったプログラムを、他人に使わせると結局人と人との付き合いが始まっていろいろな問題が起こる。

「ユーザーのことを考えてない」
「私たちの仕事を理解していない」
「仕事の流れを変えてしまう」(これについては、あながちNGとは言えない)

アプリケーションを通して人づきあい---それが直接face-to-faceであったり、地球の裏側にいる見知らぬ人とのコミュニケーションかもしれませんが、これらのプログラムとユーザーコミュニケーションのベクトルを合わせることができないか、というのがヒューマンプログラミングの主旨です。

ここまで高性能なコンピュータが安く手に入るようになると、やっぱりガソリンエンジンならガソリンエンジンに詳しい人、テキスタイルならテキスタイルに詳しい人が、プログラミングを覚えた方がより良い結果につながるでしょう。

ここで言いたいのは、

・技術の発展により、機械を説得する方法と人を説得する方法には共通点があることを指摘し、

・獣医師のような、手に職をつけている人こそプログラミングを覚えるのに向いている

ということです。(後者は以前から指摘されていたことです。)

“Human programming beyond verbal things.”
ヒトをプログラミングすることは可能なのか?

本書は動物病院の開業から廃業までの流れに沿ってデジタル化の指南書を目指しています。本書自体は大量の文書になっていますが、実際の診察の場面でのパートでは一貫して

ペットの病院は接客業であり、飼い主との円滑な信頼のために話術が重要

これも注目すべきヒューマンプログラミングなのですが、もしかしてこの価値観がいつまでもバックヤードでも電話とあいまいな言語から脱却できずに、業界全体が次のステージへ行こうという流れを妨げているのでは、という観点から

まず上巻では２つのBeyond verbal things つまり、

言語に頼らない

ことの重要性を扱っています。具体的には

■　患者とのコミュニケーションにおけるノンバーバル(=音声に頼らない)コミュニケーション

・OsiriX劇場
・５Sなどの可視化された危機管理、経営管理手法
・業務/処理の手順、スキルを目に見える形で共有する方法

■　カルテの作成において、モデリングとビジュアルレコーディング

・梅棹本による情報正規化テクニックによる問題分析
・画像や動画を残すスキル

について紹介しました。これらはスタッフレベルを対象としています。

後半は、AIがついに人語を解するようになったのを受けて、

人、旧来のコンピュータ、AIの三者に共通のプログラミングテクニックがあるのでは?

IT化によって社会が二相に別れることは避けられないという前提に立ち、上層部、すなわち経営・管理サイドが備えるべきもう2つのスキルを中心にスキルの整理をしてみようという大胆な試みになります。

■　テクニカルコミュニケーション

言語に頼る部分は、国語力とか英会話とかいった何語が話せる話せないではなく、専門用語とピラミダルロジックであることを説き、テクニカルコミュニケーションを使いこなすための訓練方法をまとめています。できないビジネスマンがいう英語力はおそらくこのことです。

■　0と1のコミュニケーション世界

デジタルの世の中で、備えるべきセンス、センスを磨くための初歩的なプログラミング技術と、世界を動かしているアルゴリズムについて解説しています。最高意思決定レベつのアルゴリズムこそ基本は単純なので、この説明は成り立つと思います。

★注意★　本書は一貫して長文をだらだらと書き殴るタイプのカルテ記録方式、とは違う可能性を探り続けるもので、言語をまったく使わない、ではなく、専門用語など短めの言語を多用する、言語化能力の向上を前提とする、など使い方の見直しを提案するものです。

問題解決能力

こういう人は向いてない

器用貧乏な人より、面倒くさがり

　どうすれば自分ひいてはみんなが楽できるかを考える怠け者は成果が期待できる

反骨精神

■ショートカッター

　直線ルートが最短距離だと思い込んでいる

■非線形性

　ノンリニアリティが理解できない

■ピラミダルシンキングができない

ITのデメリットを知っておく
プロジェクトにおける「手戻り」と「後発の強み」

■「手戻り」とは

IT導入時の最大のデメリット・リスクの一つで、プロジェクトがどうやっても進まなくなること。行き当たりばったりの最適化を繰り返すと本当にどうしようもない事態に陥る。建築で言えば地盤や基礎で失敗したのに気づかずに途中まで建ててしまったようなもの

・コンピューターは融通が利かない
・プロジェクト全体のロジックがおかしいと開発の途中で行き詰まる
・実際にはほとんど全部作り直すことになり、これを手戻りと呼ぶ

■手戻りの事例

言うまでもなくOS

Windows1.0 / 2.11 / 3.x / 95 – 98 – Me / NT – 2000 – XP / 7- 8- 10
Apple System 1.0 – 7 / MacOSX / macOS

手戻りに合わないしくみを見つけた事例

・簡単に組み替えられるようにしたもの
UNIX (パイプライン)
Advanced Visual Systems AVS (データフロープログラミング)

・間に挟まってデータを処理する動作
ジャストシステム ATOK (フロントエンドプロセッサ)　以後世界が変わる

対策としては①うまく回っているものをパクるか、②買収するのが吉。
自分でやらなければならないときは、
・複数のサブプロジェクトを常に走らせておく
・アジャイル　※複数のサブプロジェクトは大前提。アジャイルだからと言って独り歩きはさせない
・手詰まりになってもいいように、再利用可能なモジュールごとに開発しておく

■「後発の強み」とは

もう一つのリスクとして、

・投資規模が大きく、一度導入するとしばらく買い直せない
・スタッフに柔軟なアタマが無いと、最初に覚えた仕事のやり方を変えようとしない
・一時期は、パソコンなどの機器の性能が向上しながら価格は下がっていたので出遅れた方が投資が低くて済んでいた
・何より、先達の失敗をみてから始められる

そのため後から始めた方が有利な面もある、というのが後発の強み

はじめに「しばらくはこのやり方で」始め、小集団活動を通してカイゼンを試みるのがよい
上からあーでもない、こーでもないをやると組織がおかしくなる。

あるいは外国のように一斉解雇を繰り返すか。

スキルレベル曲線　一定であることはあり得ない

忘却曲線があるため、スキルには減衰力が働くので、必ず上がるか下がるかどちらかに動く

のに

どこかで一度覚えたものは忘れない、と決め付けている部分がある。

●スキルの分類

・Focused skill　　- 本人の存在意義を支えるスキル
・Unfocused skill　- 自分の存在意義とは関係ないと考えるスキル

●向上心

FスキルとUスキルのそれぞれについて向上心があるかないか、定期的に振り返る必要がある。

※CD-R現象　　一度覚えたスキルから先に進まなくなる現象のこと。若い頃に覚えたテクニックのまま死ぬまで続けてしまう。

※風化　　しばらくやっていないのが原因でスキルのレベルが落ちること

●スキルのレベル

・教わって、できなかった　→　できる
・できる　→　よく手際良く / 手早く / 低コスト / 高い成功率　でできる
・人に教えることができる

人によっては曲芸の皿回しにたとえていうこともある。

■本人が意識しなければならないこと

階層分析をする

・自分ができていたことを人に任せ続けていくうちに何もできない状況になっていないか。
・特に自分がFスキルを支えているUスキルがないか点検する

例

- TPLO
  - 縫合
  - オペ
    - 処置
      - プレートの正確な固定
      - 正確な切断
      - トラブルシューティング
  - プランニング
    - CAD
    - プレートの選択
  - 麻酔管理
    - オペの準備
      - 必要な器具類の手配
      - 滅菌
        - ブリーフィング
        - オーナーから同意をもらうスキル

■アシスタント(あるいはサポートスタッフ・業者)が注意しなければならないこと

最初のスキルレベルでサポートプラン(出勤頻度、保守料金)などを決めても必ず割りに合わなくなる点に注意する。

VekkAi式リスキリング: 固有名詞に強くなる訓練

■ポイント

テクノロジーが幅を利かせる時代になり、固有名詞をきちんと覚えて使えるようにする技術が必要。

・自然とできる人は自動的に有能な人材に見られる。

・新しいことを理解できていても固有名詞でアドレスする(=何の話をしているのかを一発で相手に伝えること)ができないと、無能とみなされる。

固有名詞に強くなるには、抽象化の訓練が有効。以下のように行う。

■トレーニング方法

○○○○　は　●●●●　の一種である　と説明する訓練をする

・カブトムシ　は　昆虫の一種　　※虫、生き物はアウト。
・バナナ　　は　果物の一種
・半導体　　は　物質の一種

これを日頃から行う。

・GPTは

・生体モニタとは

・採血とは

・減価償却とは

■レベル2: 数珠繋ぎ

最低 4 段の項目でつないでみる。

・カブトムシは

　昆虫の一種。昆虫は

　　生物の一種。生物は　　※生物はちょっと難しいです

　　物体のうち、代謝と複製をすることができるもの

・生体モニタとは

　医療機器の一種。医療機器は

　　機器のうち医療行為に使うことを目的とした道具

　　道具とは・・・・

辞書を引く癖をつけろ、と学校の先生が言っていたらその先生は正しかったと言える。

理解する技術 **recause:** ケーススタディ ハブとルーターの違い

例として、外見がそっくりで違いがよくわからない、ルーターとハブの違いを理解してみよう

学校の先生は資料を調べて歴史的な裏付けから物を説明しようとするのが普通だろう。

しかし、recauseはテキストを見ずにやるものなので、これによって何かを理解する過程は、結果的に歴史的な事実に沿う場合と、まったく自分が自由に想像した架空の歴史で同じ結果を得る場合がある。

ある物事についてrecause法で理解ができたと思ったら、一応歴史も調べておくこと。自分が作り出したあらすじと歴史になっているあらすじが似ていたら理解能力が高いということになる。

①元々は複数台のコンピューターを対等に接続して使っていた。

とりあえず接続するための装置としてハブが誕生。

②ところでハブだけでネットワークを世界中に広げてしまうと台数が膨大で通信の切り替えも大変だし、テロリストのパソコンが大統領のパソコンに簡単にアクセスできてしまう。

そこでまず組織ごとにネットワークに名前なり番号なりをつけて、互いのネットワーク間に仕切りを設け、上下関係をつけることで流れをコントロールする装置が思いつく。

これがルーター。

余談だが、人工衛星のいくつかは通信システムがTCP/IP。つまり理論上は目の前のパソコンと同じシステムになっていて、場合によってはpingが返ってくるらしい。

何かと話がすごく複雑に見えるのは「上塗り」が入っているからかも(リコーズという学びの技術)

例として、よく出てくる「IPアドレス」を理解してみよう

①元々は機器に番号を振って管理しよう、というアイデアです。簡単な話でしょう?

1, 2, 3, … と振っていくのは何も難しくない

②世界中のコンピューターを繋いでお互いに通信できるようにしたいよね、という上塗りが入る

当初の発想は、

国番号、(その国の中の)地域番号、(地域内の)組織番号、①の機器番号

こうして、 142.250.192.132 というような番号の付け方が考案された。

※10-20-30-1のように書かないのはたぶん、電話番号などと混同しないように、ではないかと思われる。

※1970年代の発想だといまのように数億台のコンピューターが繋がる世界は想像できいなかったのです。

で、

メモリも少なかったので、それぞれ254種類ずつの割り当てでいいだろうということになったのですが、

・国は増え、
・組織も大学や政府機関どころか民間企業まで入ってきて
・パソコンも数億台となり、

足りなくなった。

そこで、

③全部のコンピューターを繋いじゃうとセキュリティも危ないし、

・組織ごとに1〜10台程度の代表機によるアクセスとし、これには②のアドレスを使う
・他の数台〜数千台は代表機が伝言ゲームのような通信をすればいいよね。
・組織の内部に振る番号は①のようなやり方が楽だけどやっぱり部署とかあるから②のアイデアを内部で使ったらいいんじゃね?

というわけで、組織の内部は②との混同をさけるため、②で空いているいくつかのアドレスを使用禁止にして、

10.X.Y.Z
172.16.X.Y
192.168.0.X

のどれかを使うように、さらに「上塗り」されました。※絶対のルールではない

④さらに組織の中でもなんとなく重ならないようにしたいので、いきなり100番から始めたり、240番台を使いたいだのという不思議な会話が発生します。

ちなみに②をグローバルIP、③をローカルIPといいます。④には特に名称はありません。

このように、中核部分では単純な規則ですがいろいろなアイデアの「上塗り」が何層にも施されることにより、Wikipediaで調べても「全くわからない」「意味不明」「何でこんな面倒臭いことになってんの?」という嫌悪を抱くものになってしまうことがよくあります。

　似たような複雑な概念もほどいてみれば実は誰でも考え付きそうなアイデアからはじまっているのでそこから考えてみることをリコーズ(recause)する、と呼んで、おすすめしています。

コミュニケーションスキル

もう一度はじめから

テクニカルコミュニケーション編:
実は日本語が正しくない

■ちゃんと日本語使えているか?

何もかも「英語力」で片付けていないか?

日本語のコミュニケーションスキルはできていると思い込んでいないか?

スキル

・予防線をはる

・3択問題へ持ち込む

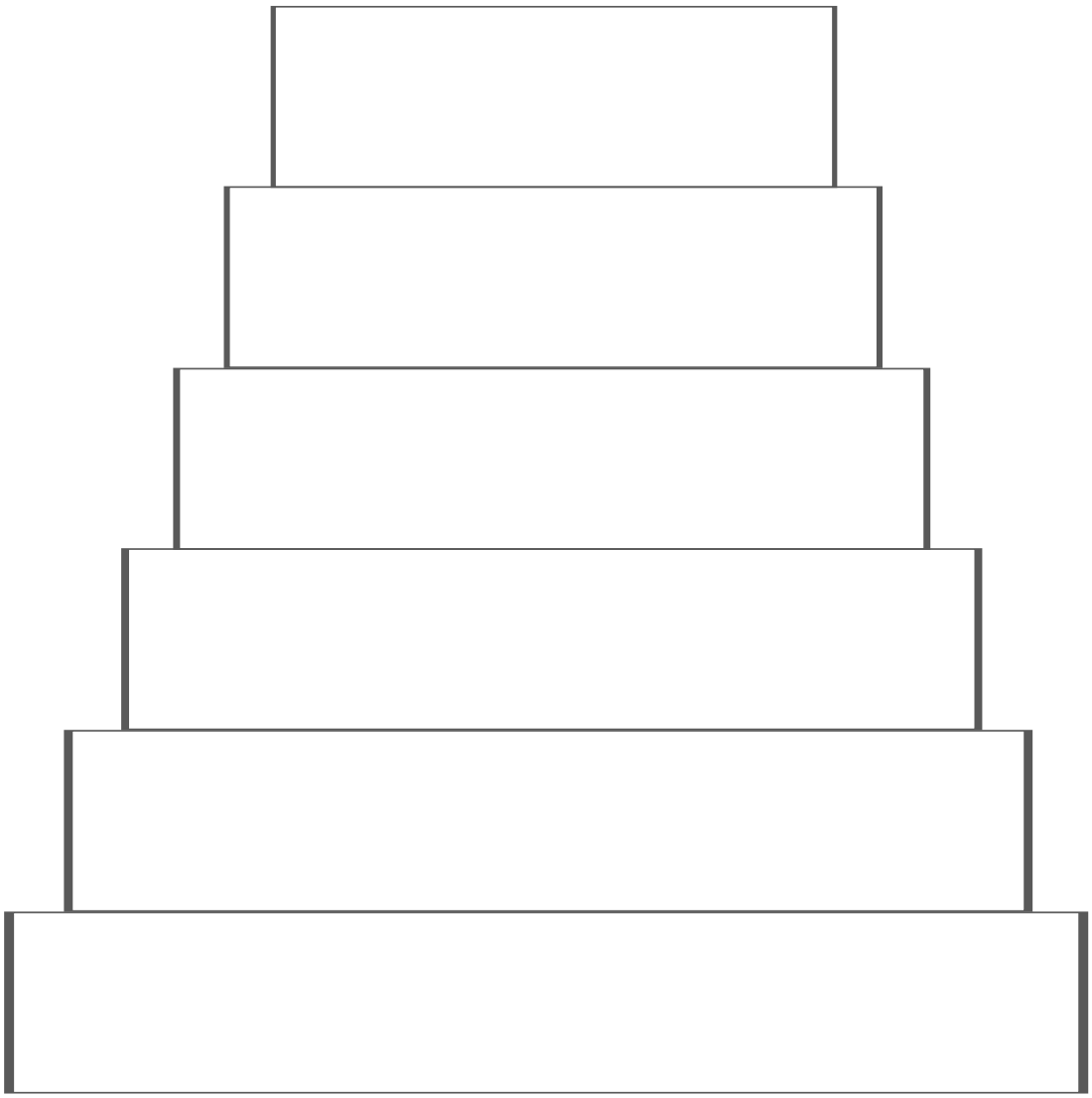
・専門用語を使う

・Specific Term をきちんと定義しながら話す

・論理のピラミッド

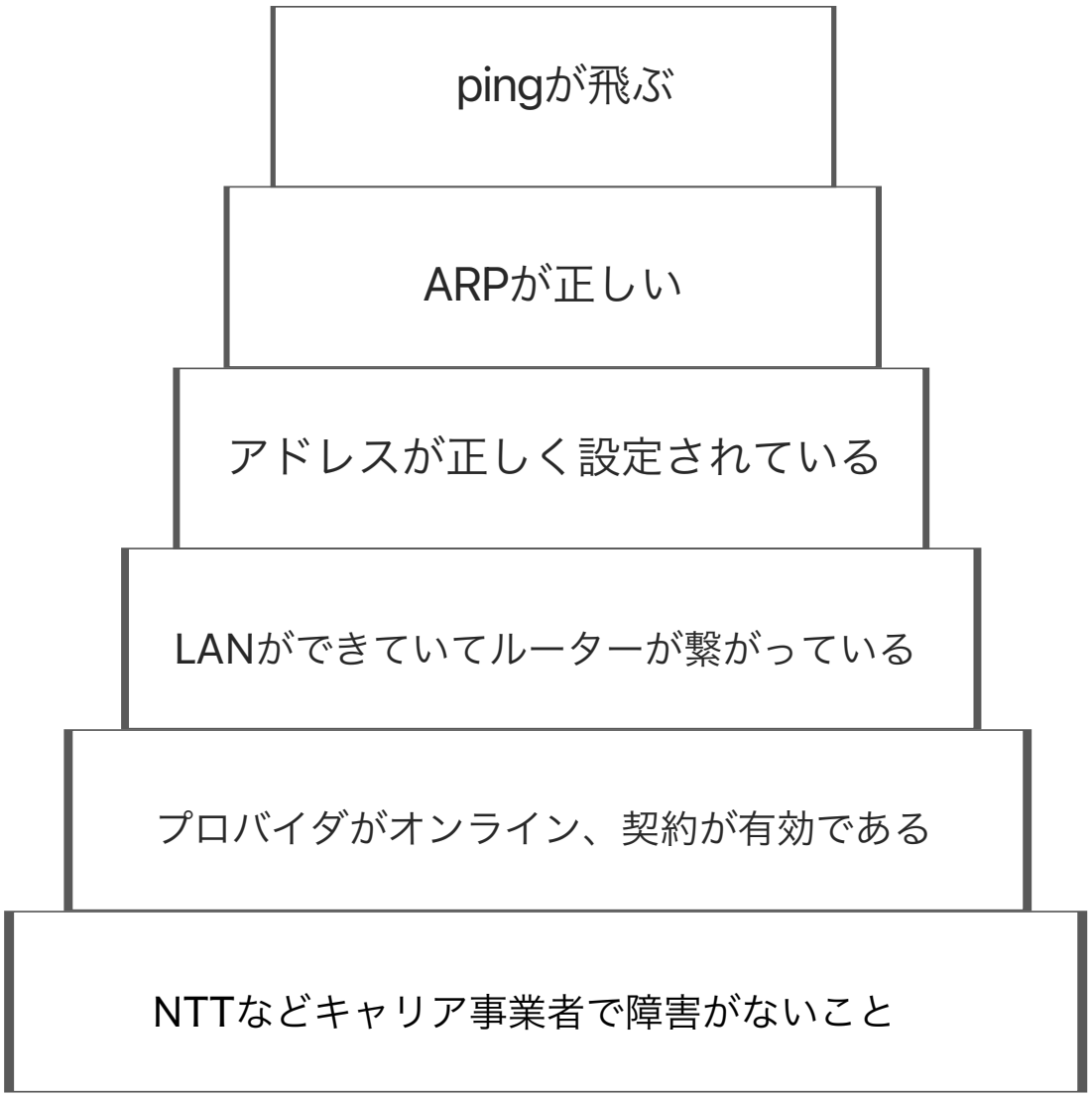
プログラミングは「相手の気持ちがわかるか」がカギ

シンキングシート: Pyramidal Thinking



シンキングシート: Pyramidal Thinking

例　インターネット接続の状態確認



Bitworks

石器
→鉄器
→プラスチック
→レアアース

→ビット

World of bit: 1 bit

表現できる情報は2種類。

ON / OFF

true / false

ある / ない

Yes / No

踏切: 開 / 閉

.....

[0
1

World of bit: 2 bits

表現できる情報は4種類。

・信号の色

　無灯、青、黄、赤

00

01

10

11

World of bit: 4 bits

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

表現できる情報は16種類。

・0から9までの数

・さらに
+, -, x, ÷, =, クリアなど
→電卓のキーコード

World of bit: 8 bits

00000000
00000001
00000010
00000011
00000100
.
.
.
.
.
.
00001111
00010000
00010001
00010010
.
.
.
.
.
.
10010010
10010011
.
.
.
.
.
.
11111101
11111110
11111111

表現できる情報は256種類。

・0から9までの数
a-z までのアルファベット
A-Zまでのアルファベット
一般的な記号
= ASCIIコード

・さらにカタカナと記号

・モノクロ画素の明るさ
0(黒) 〜 255(白)

・機械語

実際は複数回に分けて命令を送っていたりする。

World of bit: 16 bits

0000000000000000
0000000000000001
0000000000000010
0000000000000011
0000000000000100
.
.
.
.
.
.
0111111111111100
0111111111111101
0111111111111110
0111111111111111
1000000000000000
.
.
.
.
.
.
1001001000001100
1001001000001101
.
.
.
.
.
.
1111111111111101
1111111111111110
1111111111111111

表現できる情報は65536種類。

・0から65535までの整数

・-32768 〜 32767までの
符号付き整数

・地球上にあるほとんどの文字
Unicode

・音の大きさ(65536段階)

時間方向に続けると音楽を
表現できる。

CD(Compact Disc)
16bits x 44100Hz

・カラー画素
R(ed) G(reen) B(lkue)各色に
5ビットずつ割り当てる

空間方向に広げると写真の
ような画像が表現できる

World of bit: 24 bits

000000000000000000000000
000000000000000000000001
000000000000000000000010
000000000000000000000011
000000000000000000000100
.
.
.
.
.
.
.
.
.
.
011111111111111111111100
011111111111111111111101
011111111111111111111110
011111111111111111111111
100000000000000000000000
.
.
.
.
.
.
.
.
.
.
100100100111111000110011
100100100111111000110010
.
.
.
.
.
.
.
.
.
.
111111111111111111111101
111111111111111111111110
111111111111111111111111

表現できる情報は16777216種類

・カラー画素(Full color)
R(ed) G(reen) B(lkue)各色に
8ビットずつ割り当てる

5ビットより色数が増える

World of bit: 32 bits

00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
.
.
.
.
.
.
.
.
.
.
.
.
01111111111111111111111111111110
01111111111111111111111111111111
10000000000000000000000000000000
.
.
.
.
.
.
.
.
.
.
.
10010010011111100011001000000000
10010010011111100011001000000001
.
.
.
.
.
.
.
.
.
.
.
11111111111111111111111111111101
11111111111111111111111111111110
11111111111111111111111111111111

表現できる情報は4294967296種類

・浮動小数点数

・10ビットカラー画素(Full color)
R(ed) G(reen) B(lkue)各色に
10ビットずつ割り当てる

・IPアドレス

アルゴリズムの発想: デジタイジリティ(digitized depth)

一人の人間をデジタルで表現するとする

例えばのレベルわけ

★レベル1

名前と生年月日、性別、出身国だけ

★レベル2

前のレベルに加えて、

顔写真

★レベル3

前のレベルに加えて、

動画

★レベル4

前のレベルに加えて、

3Dスキャン結果

データを読解する

中身が見えていればプログラミングは難しくない

プログラミングが難しい原因は

・内部状態を想像しながらやらなければならない

・動作テスト、デバッグが面倒

・開発環境が逆に面倒事を起こす

ダンプを読もう: テキストファイル

```
00000000  22 53 6f 6d 65 20 70 65  6f 70 6c 65 20 63 61 6e  |"Some people can|
00000010  20 72 65 61 64 20 57 61  72 20 61 6e 64 20 50 65  | read War and Pe|
00000020  61 63 65 20 61 6e 64 20  63 6f 6d 65 20 61 77 61  |ace and come awa|
00000030  79 20 74 68 69 6e 6b 69  6e 67 20 69 74 27 73 20  |y thinking it's |
00000040  61 20 73 69 6d 70 6c 65  20 61 64 76 65 6e 74 75  |a simple adventu|
00000050  72 65 20 73 74 6f 72 79  2e 20 4f 74 68 65 72 73  |re story. Others|
00000060  20 63 61 6e 20 72 65 61  64 20 74 68 65 20 69 6e  | can read the in|
00000070  67 72 65 64 69 65 6e 74  73 20 6f 6e 20 61 20 63  |gredients on a c|
00000080  68 65 77 69 6e 67 20 67  75 6d 20 77 72 61 70 70  |hewing gum wrapp|
00000090  65 72 20 61 6e 64 20 75  6e 6c 6f 63 6b 20 74 68  |er and unlock th|
000000a0  65 20 73 65 63 72 65 74  73 20 6f 66 20 74 68 65  |e secrets of the|
000000b0  20 75 6e 69 76 65 72 73  65 22 0a 0a 2d 20 4c 65  | universe"..- Le|
000000c0  78 20 4c 75 74 68 6f 72  0a                       |x Luthor.|
```

ダンプを読もう: Raw画像

```
00000000  00 00 00 c5 00 00 00 c5  00 00 00 c5 00 00 00 c5  |................|
*
000003c0  00 c0 7a c4 00 00 7a c4  00 00 7a c4 00 40 7a c4  |..z...z...z..@z.|
000003d0  00 00 7b c4 00 80 7a c4  00 40 78 c4 00 c0 77 c4  |..{...z..@x...w.|
000003e0  00 40 78 c4 00 80 79 c4  00 40 7a c4 00 80 7a c4  |.@x...y..@z...z.|
000003f0  00 c0 7a c4 00 00 7a c4  00 c0 79 c4 00 40 78 c4  |..z...z...y..@x.|
00000400  00 00 7a c4 00 40 79 c4  00 80 7a c4 00 80 79 c4  |..z..@y...z...y.|
00000410  00 80 7a c4 00 40 78 c4  00 40 79 c4 00 80 76 c4  |..z..@x..@y...v.|
00000420  00 40 77 c4 00 c0 77 c4  00 40 79 c4 00 00 77 c4  |.@w...w..@y...w.|
00000430  00 00 78 c4 00 40 77 c4  00 40 77 c4 00 00 76 c4  |..x..@w..@w...v.|
00000440  00 00 00 c5 00 00 00 c5  00 00 00 c5 00 00 00 c5  |................|
*
00000b90  00 40 7a c4 00 c0 7a c4  00 c0 7a c4 00 00 7b c4  |.@z...z...z...{.|
00000ba0  00 40 7a c4 00 80 78 c4  00 80 78 c4 00 80 78 c4  |.@z...x...x...x.|
00000bb0  00 40 7a c4 00 80 7a c4  00 40 7b c4 00 40 7b c4  |.@z...z..@{..@{.|
00000bc0  00 00 7b c4 00 80 79 c4  00 00 79 c4 00 00 79 c4  |..{...y...y...y.|
00000bd0  00 00 7a c4 00 c0 79 c4  00 40 79 c4 00 40 79 c4  |..z...y..@y..@y.|
00000be0  00 00 79 c4 00 80 79 c4  00 00 79 c4 00 c0 79 c4  |..y...y...y...y.|
00000bf0  00 c0 7a c4 00 40 7a c4  00 80 79 c4 00 00 79 c4  |..z..@z...y...y.|
00000c00  00 c0 7b c4 00 c0 79 c4  00 40 7a c4 00 c0 78 c4  |..{...y..@z...x.|
00000c10  00 c0 79 c4 00 00 78 c4  00 c0 78 c4 00 80 76 c4  |..y...x...x...v.|
00000c20  00 c0 78 c4 00 00 77 c4  00 80 75 c4 00 40 74 c4  |..x...w...u..@t.|
00000c30  00 40 74 c4 00 40 74 c4  00 c0 74 c4 00 80 74 c4  |.@t..@t...t...t.|
00000c40  00 00 76 c4 00 00 76 c4  00 c0 78 c4 00 80 78 c4  |..v...v...x...x.|
00000c50  00 00 79 c4 00 00 79 c4  00 80 79 c4 00 00 78 c4  |..y...y...y...x.|
00000c60  00 40 7a c4 00 c0 78 c4  00 80 7a c4 00 00 7b c4  |.@z...x...z...{.|
00000c70  00 00 00 c5 00 00 00 c5  00 00 00 c5 00 00 00 c5  |................|
```

ソースコード

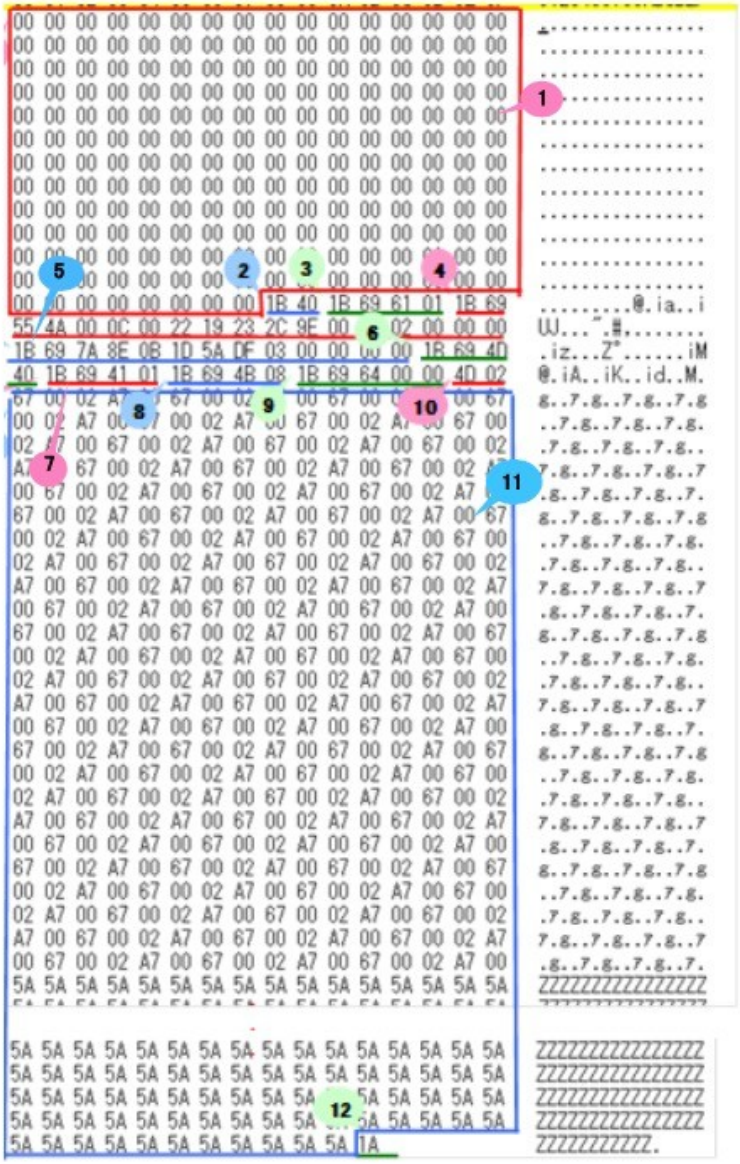
```
00000000  23 69 6e 63 6c 75 64 65  20 3c 73 74 64 69 6f 2e  |#include <stdio.|
00000010  68 3e 0a 0a 69 6e 74 20  6d 61 69 6e 28 29 20 7b  |h>..int main() {|
00000020  0a 20 20 20 20 70 72 69  6e 74 66 28 22 42 65 79  |.    printf("Bey|
00000030  6f 6e 64 20 74 68 65 20  4c 61 6e 67 75 61 67 65  |ond the Language|
00000040  73 5c 6e 22 29 3b 0a 20  20 20 20 72 65 74 75 72  |s\n");.    retur|
00000050  6e 20 30 3b 0a 7d 0a                              |n 0;.}.|
```

コンパイル結果(実行形式バイナリ) *抜粋

```
00000000  cf fa ed fe 0c 00 00 01  00 00 00 00 02 00 00 00  |................|
00000010  11 00 00 00 20 04 00 00  85 00 20 00 00 00 00 00  |.... ..... .....|
00000020  19 00 00 00 48 00 00 00  5f 5f 50 41 47 45 5a 45  |....H...__PAGEZE|
00000030  52 4f 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |RO..............|
00000040  00 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  19 00 00 00 88 01 00 00  |................|
00000070  5f 5f 54 45 58 54 00 00  00 00 00 00 00 00 00 00  |__TEXT..........|
00000080  00 00 00 00 01 00 00 00  00 40 00 00 00 00 00 00  |.........@......|
*
00008180  a1 0c 36 ae 34 dc d3 c1  7b ae 56 77 25 f5 7d 2d  |..6.4...{.Vw%.}-|
00008190  57 b7 ad 7f ac b2 58 6f  c6 e9 66 c0 04 d7 d1 d1  |W.....Xo..f.....|
000081a0  6b 02 4f 58 05 ff 7c b4  7c 7a 85 da bd 8b 48 89  |k.OX..|.|z....H.|
000081b0  2c a7 ad 7f ac b2 58 6f  c6 e9 66 c0 04 d7 d1 d1  |,.....Xo..f.....|
000081c0  6b 02 4f 58 05 ff 7c b4  7c 7a 85 da bd 8b 48 89  |k.OX..|.|z....H.|
000081d0  2c a7 75 c5 6f a4 24 b4  48 dd 3a b1 a1 91 53 52  |,.u.o.$.H.:...SR|
000081e0  89 a7 25 a3 95 45 ab 8b  19 26 2a ad d1 84 d1 cf  |..%..E...&*.....|
000081f0  50 bc df d5 eb 86 f1 95  66 03 a1 2c 06 36 f4 63  |P.......f..,.6.c|
00008200  1f 66 f1 0d 0b b8 d4 ee  a3 d2 5b 82 e0 e5 92 02  |.f........[.....|
00008210  b4 ed ad 7f ac b2 58 6f  c6 e9 66 c0 04 d7 d1 d1  |......Xo..f.....|
00008220  6b 02 4f 58 05 ff 7c b4  7c 7a 85 da bd 8b 48 89  |k.OX..|.|z....H.|
00008230  2c a7 ad 7f ac b2 58 6f  c6 e9 66 c0 04 d7 d1 d1  |,.....Xo..f.....|
00008240  6b 02 4f 58 05 ff 7c b4  7c 7a 85 da bd 8b 48 89  |k.OX..|.|z....H.|
00008250  2c a7 ad 7f ac b2 58 6f  c6 e9 66 c0 04 d7 d1 d1  |,.....Xo..f.....|
00008260  6b 02 4f 58 05 ff 7c b4  7c 7a 85 da bd 8b 48 89  |k.OX..|.|z....H.|
00008270  2c a7 89 1d cb 29 e1 b6  23 05 70 75 04 ac 05 2c  |,....)..#.pu...,|
00008280  54 dd 28 9e 37 e0 43 05  ae 6f cc ea 49 7b 55 6f  |T.(.7.C..o..I{Uo|
00008290  4f 6d                                            |Om|
```

ダンプを読もう: プリンタに送られているデータ

以下の図は、「2.2.1 事前の準備」で作成した印刷データをバイナリエディターで表示したものです。



番号	コマンド名称	説明
1	無効指令	200 バイト分の無効指令を送っていることが分かります。
2	初期化	初期化コマンドを送っています。
3	動的コマンドモード切替	プリンターをラスターモードに切替えます。 プリンターにラスターデータを送信する前に、このコマンドを送ってください。
4	ジョブ ID 指定コマンド	(QL-720NW のみ) 内部仕様コマンドです。 製品版ドライバーでは出力するコマンドですが、お客様の方で特に送る必要はありません。
5	印字情報指令	印刷データの用紙サイズ情報を送ります。 ここでは、「1.1" × 3.5" (29 mm × 90 mm)」のダイカットラベルであることを意味しています。
6	各種モード設定 (1Bh + 69h + 4Dh + 00h)	カットオプションなどを指定できるコマンドです。 ここでは、オートカットを指定してます。
7	オートカット枚数指定	オートカットの枚数を指定します。
8	拡張モード設定	拡張的な機能を指定できるコマンドです。 ここではカットアットエンドを指定しています。
9	余白設定	ダイカットでは余白量を指定できないため、余白量は 0 としてコマンドを送っています。
10	圧縮モード選択	(QL-720NW のみ) TIFF モードを指定しています。
11	ラスターデータ	ラスターデータが続きます。
12	排出を伴う印字指令	1 ページの印刷であるため、1 ページ目の最後に送ります。
13	動的コマンドモード切替	(QL-600 のみ) 3.で切り替えたコマンドモードを本体デフォルトのモードにリセットします。 「排出を伴う印字指令」の後に、このコマンドを送ってください。

Brotherコマンドリファレンスより

コミュニケーション スキル

プログラミングは コンピューターとの コミュニケーション

ちゃんとしたプログラミングのために(1)

■プログラミングとは、究極的には自前のプログラミング言語を定義する作業である。

●スクールで記述するレベル(50行未満)

●鼻紙コード、使い捨てレベル

==== ここに巨大な壁 ====

●インタープリターレベル

独自あるいは工業標準で規定されているデータフォーマットを読み込み、

事実上、自前のプログラミング言語を作成しているのと同じことになる。

つまり、プログラミングというのは、出発点のプログラミング言語を拡張するなりなんなりして、さらに高いレベルのプログラミング言語を作る作業と見ることができる。

■上記の事例

Microsoft Excel – 実体はXLS(OLE Object)を読み込み、実行して、操作・修正が加えられたものをXLSとして書き出すソフトウェア。

各種CAD

ゲーム -

設定ファイルとか、プレイデータと呼ばれるものを言語と見立ててればゲームも結局はインタープリターの構造になる。

■用語

パース(parse)、デコード(decode)

　　設定ファイルや各種のフォーマット(以後、ソースと呼ぶ) を読み込み、プログラミング言語のメモリモデルに変換すること。通常ソースはディスク上にあり、メモリモデルはメモリ上にある。

　　逆にメモリモデルをソースとして書き出すのは、保存あるいはエンコードと呼ばれる。

API (Application Programming Interface)
　　メモリモデルに対して定義した操作のこと。これがここでいう「独自の言語」にあたる。

コンバーター
　　ソースとは異なる仕様のデータを出力するプログラム。変換が目的なので、コンバーターと呼ぶ。

　　例として、JPEG→TIFF 変換など。

　　ソースと同じ仕様のデータを出力するアプリケーションでも別の仕様のデータを出力することはよくあって、この時は「エクスポート」という言い方をして区別していることが多い。

　　例えばAdobe PhotoshopはPSD形式のインタープリターだが、JPEGなどにエクスポートできる。

なので、どんなプログラミング言語を習得するべきかという議論は自分が作成したいと思う「言語」のスタイルにもっとも近いものを用いるのが理想的で、①闇雲にいま流行っている言語に飛びつかない、②でも拡張性が低い言語はさける方がよい。

これでおかしくなったプロジェクトはいっぱい見てきた。だから有料で自分で手が出せない言語とか選ぶのは愚の骨頂であり、また最近流行っているノーコードもほとんどは使い捨てレベルしかできないのは明らかである。(もちろんいいところはある)

なぜプログラミング言語が存在するのか?

それぞれのプロセッサ(CPU)には、それぞれの機械語(インストラクションセット)というものがある。

機械語というのは、

・〜という命令を受けたら、こういうことが起きますよ

というルールの集まり。この機械語で指示を与えることをプログラミングと言っていた。

しかし、機械語には大きな問題があって、

・メーカーが違うとインストラクションセットががらっと変わること。下手すると同じメーカーでも時期によって少しずつ変わっていくこと

・そもそも単純すぎる作業しかできないので、簡単なゲームを一つ作るにも効率が悪すぎて膨大な時間とコストがかかってしまう。

そこで、

・人間がより楽にコンピューターに指示を出せるような動かし方をしたい

・メーカー間の違いを吸収するため、新しいインストラクションを定義してどのメーカーのコンピューターでも実行できるようにしたい

という自然な発想からプログラミング言語が生まれた。

ちなみに

■機械語とは何か

機械語というのは実体は0と1の集まり。慣例的に16進数表記にしていることが多い。

最初はこれを手で入力していた。やがてパンチカードとか紙テープを使うようになる。

■アセンブラの登場

コンピューターの能力あるいはやらせたい仕事の規模が大きくなると、一般人の想像をはるかに超える勢いでプログラミングの負担は急激に大きくなる。

さすがに16進数表記も辛いということで、簡単な英語表現したもの(本当はこれをインストラクションという)をメーカーが定義しており、この英語表現を書くと0と1の集まりに変換するプログラムを作って効率を上げた。このプログラムはアセンブラと呼ばれる。

アセンブラは作業のオートメーションに過ぎない。命令は1対1の対応、つまり100行書いたら100個のインストラクションが出るに過ぎないがこれでもだいぶ楽だった。

■アセンブラが進化

アセンブラは進化して定型処理にユーザー自身が名前を与えて再利用できる機能を持つ。サブルーチンと呼ばれるこの機能はオートメーションをさらに加速させた。

文字列処理が次第に可能になってくる

オートメーションの先に機械語では想定しえない処理手順が考案される。

ループ、条件付きループ、他条件分岐、リスト、キュー

これらはもはやアセンブラではなく、プログラミング言語というべき水準に到達している。

BCPL

この頃、FORTRAN (FORmula TRANslator = つまり数式変換器という意味)、COBOL

なぜプログラミング言語が存在するのか?

■マイコンの発明

インテル

電卓用の集積回路が多目的のマイクロプロセッサとして進化したことを受け、こちらも大型計算機のようなプログラミング言語を作りたいという横着者が現れる。

※横着者はポジティブな意味

FORTRANを走らせるには性能が不足していたのでマイコンレベルで新しい言語が提案される。

特に人気を博したのがBASICという言語で、あえて誤解を恐れずに言ってしまうとJavaScriptやPythonの祖先であると言える。

だから古い言語や流行らない言語をバカにしてはいけない。FORTRANやCOBOLもやろうと思えば最新の言語並みの機能を実装することは可能なのだ。ただし、それをやると国際規格に縛られている言語は規格の修正が必要だし、それでFORTRANで書かれたプログラムが間違いなく動作するかどうかを担保できない、という社会の仕組み的な面倒なことがあり、それなら違うプログラミング言語で書いた方が話が早いのである。

早期にBASIC言語の実装に成功した人の中にビル・ゲイツという人物がいた。彼はプログラミング言語の効果が絶大であることを理解し、マイクロプロセッサ用のソフトウェアを提供していく会社を始める。

プログラミング言語の世界地図(1) 一般業務向け

一般業向けの言語選択の私見です。この場面では短期的に十分なアウトプットを出せる言語を選ぶべき。WORA(=Write Once, Run Anywhere)、WOWP(=Write Once,Works Permanently)つまり、一度書いたプログラムはどこでも動作して、いつまでもきちんと動き続けることを最重視しています。

近年、Lua, Rust, Go, Kotlin, VisualBasic, Swift など話題性がありますが、基本的に歴史がない言語や企業が主導する言語はあまりお勧めできません。歴史がない言語はできないことがいっぱいある。企業が主導する言語は仕様に思想的な一貫性がなく、また文法がころころ変わる。

1. P言語群　(オープン、PMあり、Linux/Mac/Win全部ある)

Perl, Python, PHP を指す。設計者はそれぞれLarry Wall, Guido van Rossum, Rasmus Lerdorf 。

- ちょっとしたデータ処理などに強い。すべてオープン&フリー。
- PM(=パッケージマネージャ)という、高度なことがやりたい時に膨大な量の拡張機能がインターネットからダウンロードできるのも支持される理由。

この中では、AIブームに対応したPythonが現在は大人気。弱点は開発環境が地味なこと。それに伴い、マウスやメニューなどを配置してのアプリ開発には向いてない。個人、あるいは内輪、クラウドサーバーなどでの応用がメインとなる。

PythonにはJupyter(ジュパイター)という開発支援アプリがあるので少しはマシだが、Perl/Rubyは、昔ながらの対話、つまりコマンドの繰り返しになる。

2. ECMAScript (オープン、NodeJSに限ればPMあり。Linux/Mac/Win全部ある)

- 元々JavaScriptと呼ばれていた、ブラウザ上で実行する前提の言語。Netscape Communicationにいた Brendan Eichが設計したとされる。
- P言語と同様にOS上で実行するNodeJSという処理系が開発され、P言語よりも便利な場面も多い。
- 後述のJavaと名前が被るため、ECMAScriptとか、TypeScript(方言)として考えた方が吉。

これもマウスやメニューなどを配置してのアプリ開発には向いてない。工数もP言語よりは大きいので、クラウドサービス用に向く。NodeJSはクライアントとサーバーのそれぞれを同じ言語で書ける世界を目指したもの。

JavaScript単独では開発生産性は悪い。ReactやVue, jQuery, bootstrapなどのオプションコードを組み合わせるのが大前提。

プログラミング言語の世界地図(2) アプリ開発向け

販売を前提とするアプリ開発向けの言語選択の私見です。この場面では実行速度と開発工期の短縮、チーム開発に対応できる機能が重視されます。

1. C　(フリー/商用、ほとんどのプラットフォームにある)

ハードウェア制御に関係するようなソフト開発の絶対王者。その代わりデバッグの時間やコストが莫大で、納期は読めない。

- ここに紹介されるほとんどの言語はCで書かれているので、C言語をマスターすればプログラミングの頂点に立てることになるが、時間とコストに絶対的な覚悟が必要です。かじる程度が関の山。

Windows/LinuxではC++、MacではObjective-Cというオブジェクト指向機能を拡張した派生種を使うことが用意される。これは、GUIアプリの開発の工数を減らすために導入されたもの。Windows 3.1やmacOS 7の頃はCで書くしかなかったのだが、これがなんとも後から読み返すのが辛すぎるコードになりがちだった(可読性が低い、といいます)

設計はAT&T Bell研究所にいたDennis Ritchieら。

2. Java (準オープン。PMは無い。Linux/Mac/Win全部ある)

早い時期にWORAを目指し、かつ当時流行っていたオブジェクト指向言語として有名。C言語でなくてもGUIアプリが書ける。販売できるアプリを書くならいまだに中心的。

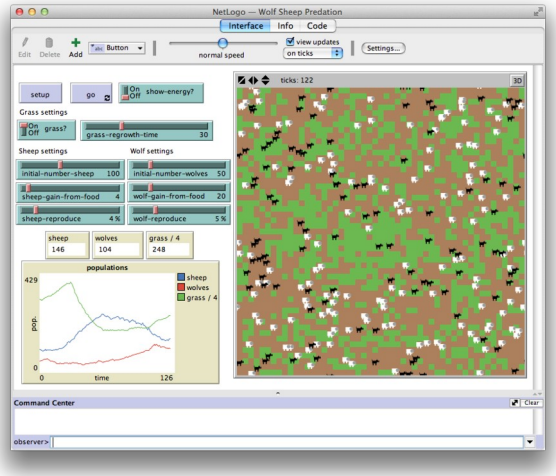
SunMicrosystems にいた Bill Joyという人が作ったことになってるし、実績もあるが、Sunが、この世にオープンソースの存在を絶対に認めないOracleに買収されると有料化。

・一つ覚えで一生戦いたがる日本人のハートを射止めた。
・JavaScriptとJavaは別物
・マイクロソフトのC#はJavaのデッドコピー
・GoogleのAndroidも最初はJavaでアプリを作る前提

プログラミング言語の世界地図(3) フィールド特化型

特定の応用分野に強いもので、プログラミング言語としては恥ずかしくないだけの機能的があるもの。これはそれぞれのジャンルにいくつか存在していると思われるが、ここでは、私が分かる範囲だけ列挙します。

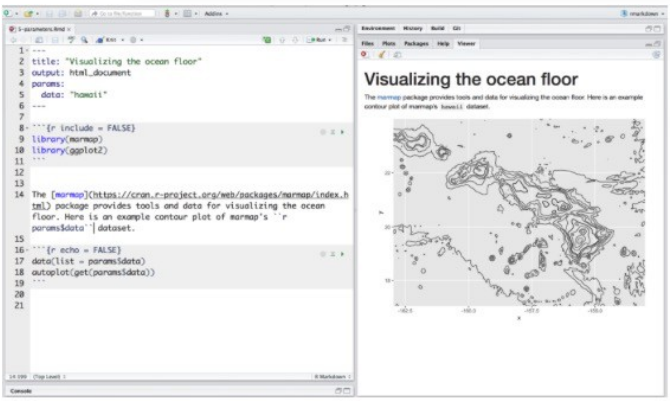
1. **NetLogo** (シミュレーション)



シミュレーション分野の中でも特にエージェント型と称する実際の行動パターンなどの解析を得意とする。言語仕様としては教育用言語の草分けであるLogo(Seymour Papert MIT教授による)を参考に作られている。

グラフィカルな開発支援アプリを持つ。基本的には単語ベースのプログラムコードが先にあり、その内容を表示したり、ユーザからの入力を受け取るために、ボタンやグラフを配置できる。

2. R (統計やデータ解析分野)



This example is explored in further detail as a lesson on the R Markdown Website.

オープンソース。本当はS言語(商用)を先に紹介すべきところである。Rは、Sをオープンで作り直したところからスタートしている。

R Studioという開発支援アプリがあり、特にアプリ中の、Rmarkdownという、Wikiライクな記法のレポートを書きながら所々にR言語のコードや処理結果を埋め込めると言う表記方法が極めて特徴的で、極論すれば、このままレポートや論文ができていると言うシステムが成立し、本家のSを凌駕した。

3. Mathematica (数学)

理論物理学者のStephen Wolframが開発。高度な数学処理を前提とするプログラミング言語。高価だが、ラズベリーパイには無料版が付いている。

4. MATLAB (画像処理、制御)

数学者Cleve Molerが開発。数学的な処理を得意とする言語。SimLinkというビジュアルプログラミング環境をオプションで設定している。

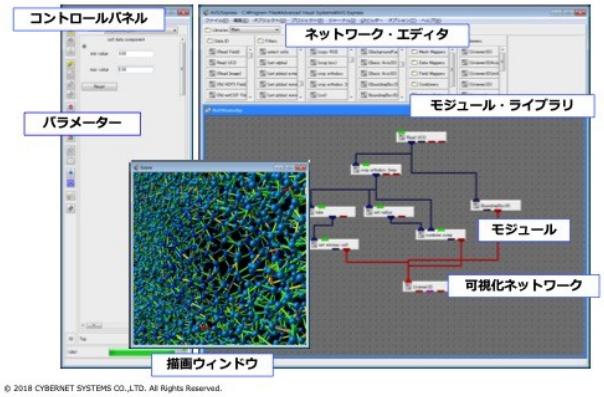
5. MUMPS (医療情報)

MGH(マサチューセッツ総合病院)で、医療情報処理を目的に開発された言語。固有のデータベースエンジンを言語仕様として持っている。設計は古く、途中でデータ量の増大に耐えられなかったのか普及しなかったが、いまなら実用的に操作するのでは無いかと思われる。

プログラミング言語の世界地図(6) ビジュアルプログラミング

グラフィカルなユーザインターフェースを用いてブロックを並べるようなやり方で、処理を記述するタイプのプログラミング言語。。。言語といえるかどうか。ブロックが先にあり、コードは後から書きます。

1. AVS (商用)

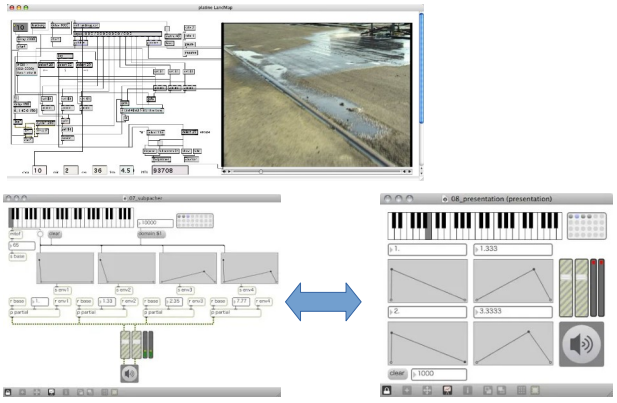


モジュールと呼ばれるブロックを並べてつなぐことで、所定の結果を得ることができると言うツール。初期のビジュアルプログラミングツールで科学技術計算の結果をグラフィックにするのに効果を発揮した。

ビジュアルプログラミングツールの特徴は、ブロックに大まかな分類があって
(1)データソース
(2)データの加工、変形プロセス
(3)結果の表示方法
のそれぞれに整理される。プログラミング能力は脆弱なのだが、C言語等を用いてブロックを開発し、機能拡張することが可能であるのが普通。

最近流行っているローコード、ノーコードは、この拡張機能がないものがあるので要注意。

2. Max/MSP (商用)



出典: https://yoppa.org/ssaw10/912.html

電子楽器の制御や音楽制作を目的とするビジュアルプログラミング環境。これも、老舗

プログラミング環境として考えた場合、最大の特徴がプレゼンテーションモードを持っていること。

プログラミングをしているときには、複雑な配線が並んでいるがユーザにこのまま使わせるのはちょっと大変なので必要なパーツだけを、異なるレイアウトに再配置されると言うモードを持っている。

3. LabView (商用)

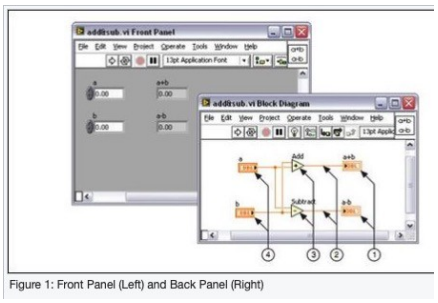
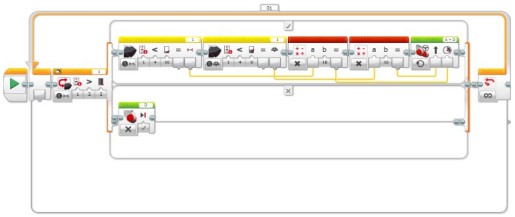


Figure 1: Front Panel (Left) and Back Panel (Right)

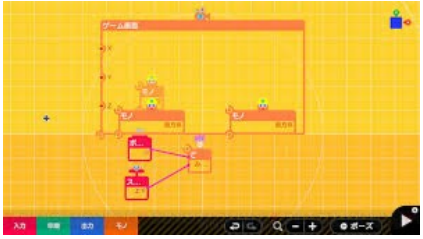
仮想インスツルメンテーションといい、制御プラントや実験設備の構築を容易にする。

プラント(設備)全体のデータの流れをビジュアルに表現するものだが、ブロックの一部は実際のセンサやロボット、バルブなどに1対1で対応していて、本当にプラントが動作したり、ロボットを動かすことができる。データの流れをそのまま見ることができるバックパネルも充分有用だが、フロントパネルというユーザインタフェースを別途作ることもできる。(右図上)



大学や産業用な感じだが、子供達向けのLEGOロボット制御に特化したバージョンもある。(右図)

4. つくってわかるはじめてゲームプログラミング (商用)



ブロックのサイズが変更でき、そのレイアウトのままゲームステージが作られる。

プログラミング言語の世界地図(4) 準プログラミング言語

1.マクロ

プログラミング言語単体として設計されたものではなく、アプリケーションソフトウェアの自動化を目指して作り込まれているものをマクロ言語といいます。先にデータ資源が配置されて、それに対する処理を記述するスタイル

- 図面ソフトのAutoCADが内蔵するAutoLISP
- Microsoft Excel / PowerPoint / Word などのVBA
- Apple HyperCard に対する HyperTalk
- ECMAScriptもWebブラウザHTML用のマクロと見なすことができる。

そのアプリケーションの自動化が目的なので、道を外れた事は一切できない。

2. RADツール

Rapid Application Developmentツール。一般のプログラミング言語の敷居を下げる目的で、ビジュアルな記述方法を前面に出すもの。実際には裏方さんとしてプログラミング言語が走っていることがよくある。

- Scratch (初期バージョンは裏方にはSmallTalkという、その筋では有名な言語あり)

近年提案されているローコード、ノーコードツールも、結局はRADツールとして設計されているものがまとも。結局作成してもプログラムの細かい動作は裏方言語に頼らざるを得ないが、上手に使えば短時間で所定のアウトプットを得ることが可能であると思われる。

3. ゲームエンジン

3Dグラフィックを求めるアプリの場合、ゲームエンジンで作成する流れが大きくなってきました。ただし、星の数あってどれがいいかなんとも言えません。おそらくまともなシステム構成は、(1)レベルエディタという舞台を作成するソフト、(2)実行を支えるプログラミング言語、(3)その言語用のRADツール、を合体させたような感じ。冒頭のWORAを意識していて、エンジンで作成した後はWindows / Mac / スマホはもとより、Nintendo SwitchやPlayStationでも動作するようものもある。

・Unreal Engine　　企業主導
・OGRE
・Godot　　オープン
・O3DE
・Defold
・STRIDE
・PlayCanvas

* Unityという人気のエンジンがありますが、いきなり課金ルールの変更がなされる事件があったため、お勧めしません。

プログラミング言語の世界地図(5) 言語のための言語

これらはちょっと特殊な用途の言語です。プログラミング言語と呼ぶには、機能は足らないことがほとんどですが開発業務の工数削減、効率的な分業化、標準化による競争の促進やスペシャリスト人材の育成に大きく貢献している部分です。

1. G-code　機械制御言語

加工機械を制御することを前提として開発された言語。しかしながら、ロボットや3Dプリンターにも応用されていて、最近では住宅用3Dプリンターでも使われていたりする。

2. TeX系言語などの文書整形言語

テフ、と読む。活版、印刷技術をデジタル的に置き換えることを意図した言語で、論文作成や楽譜等の作成に使われている。標準では非常に扱いにくいとされ、拡張仕様のLaTeX 、AMS-TeX、など～TeXをそれぞれの分野に合わせて使っていることがある。

旧くはroffなどが知られ、最近ではWikiとその発展型のマークダウンが広まっている。

3. PostScript　　プリンタ制御言語

非常に複雑な印刷結果を得るために、プリンタ側にパソコンよりも高性能なプログラミング能力を持たせた。現在ではパソコン側で充分処理可能になっていて存在意義は薄い。後継の言語(といえるか?)としてはPDFが知られている。逆ポーランド記法を採用している点もユニーク。

4. SQL, GraphQLなどのデータベース制御言語(クエリ言語)

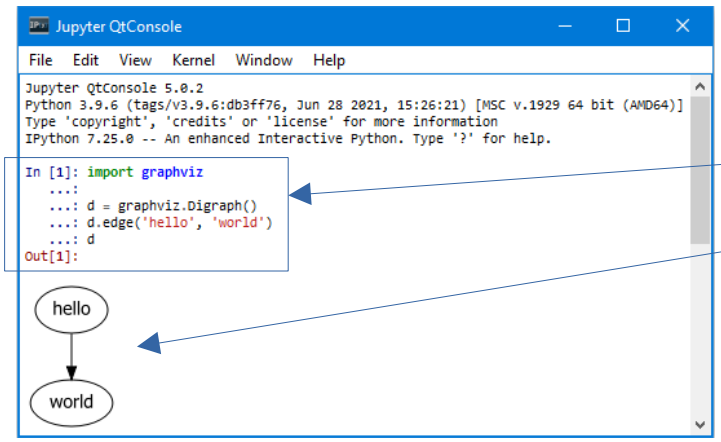
リレーショナルデータベースエンジンをコントロールするときに使われる標準的な言語。実際にはメーカーごとに方言がある。

5. Cisco IOS, YAMAHAルーターなどのルータープログラミング言語

ルーターの細かい動作を設定するための言語。本来は設定ファイルで良さそうなものだが、あまりに細かい動作を記述するために、結果的にプログラミング言語のような文法を持つことになった。ヤマハに至っては最新版ではLuaというスクリプト言語を統合している。

6. GraphViz,PlantUML,Mermaidなどの、Code to Diagrams系

文字で記述して、ダイアグラムを作成するタイプの言語。当然、一般的な一般的なプログラミング言語でやるよりも工数が少ないことを売りにしている。



これを書くだけで、

図が作成される。

プログラミング言語の世界地図(6) キワモノ

1. FORTRAN / Fortran (John Backus)

歴史的な言語。科学技術計算用に開発された、極めて初期の言語で、発表された時は、プログラミング時間を1/10以下に抑えたにも関わらず実行速度があまり低下しなかったのでしばらくプログラミング言語の主軸となった。コンパイラ型言語のお手本でもある。

なぜキワモノなのかというと、大成功を納めたせいで言語仕様の拡充ができなくなりつつもいまだに生きている点を指摘したい。なお他の言語での開発効率はさらに1/10, 1/10と進んでいっているので、もはやFORTRANが効率的だというわけではない。

2. COBOL　(Grace B.Hopper 女史)

これも歴史的な言語。事務処理用に開発された初期の言語。記述効率が悪いが、人間の言葉に近づけた設計で支持され、いまだに大量のコードが現役という、化け物級の言語。実行しているコンピュータに蛾が混入して不具合を起こしたことから、プログラムの不具合をバグと呼ぶことになった。

3. APL (現J言語)

ギリシャ文字を使用するプログラミング言語。かなり古くからあり、考え方は現在のMATLABや、Pythonの数値計算ライブラリ numpy に通じている。

FORTRAN/COBOL/APLなどは、プログラミング言語の本来の役割を忘れてはいない。

以下を理解するために、リストという言葉を知っておく必要がある。リストとはざっくりいうと単語を数珠繋ぎにしたものである。表と混同しやすい。

4. LISP (LISt Processor)

プログラミング言語のキワモノといえばこれでしょう。計算機科学史に燦然と輝く言語。コンピュータはあらゆる情報をビット列として表現する。ならばその一段上でアトム(単語のようなもの)列であらゆる情報を表現すればもっと高度な処理を容易に記述できるのではないか？というアイデア。結果として強力な言語処理機能を誇り、数式処理システムほかFORTRANやCOBOLでは到底できないことを実現。問題は大量のかっこが入れ子になっていて読みにくいこと。

6. Prolog

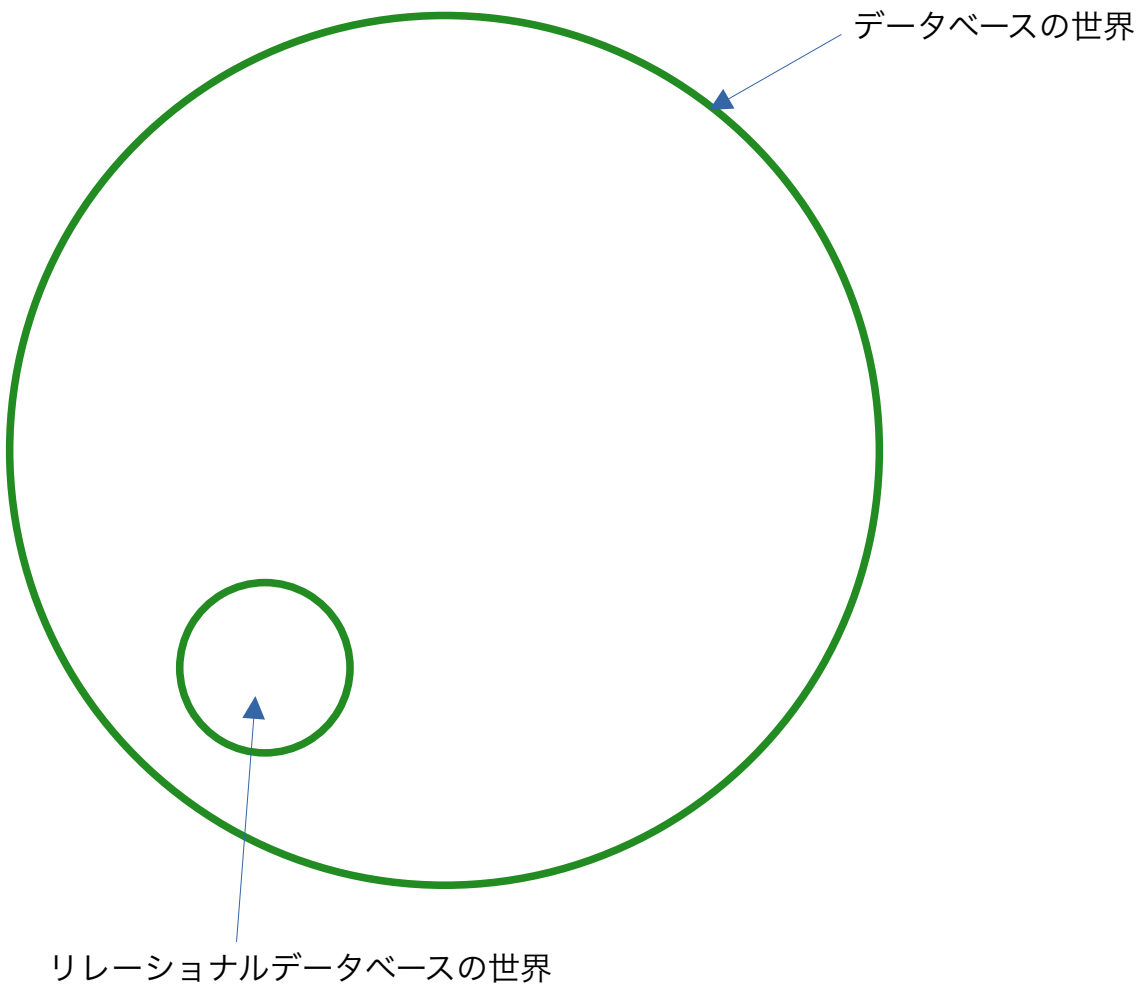
AI (人工知能) 構築を目的とする言語。とは言ってもいまのような畳み込みニューラルネットワーク、深層学習を意図したものではなく、述語論理式という宣言文を積み重ねることでルールベースとよぶデータベースを作成するもの。ルールベースは宣言文を上手に接続することで、プログラミングされていない処理手順を生み出すことができる、とする。

5. Smalltalk

プログラミング言語のキワモノといえばこれ。

アルゴリズム

“データベース”への正しい理解が必要



これさえ知っていれば食いっぱぐれは無いと言われるリレーショナルデータベース、特にSQL対応のもの、がまるで世界のすべてであるかのような話になっていることが多い

もちろんSQLもかなり奥深い世界で、TRIGGER多用型のロジックとか
SELECTネストのテクニックなどさまざまなノウハウがある。

が、しかし、

データベースという分野は実ははるかに広い世界で、RDBはもともと当時のコンピューターの技術水準で何とか実用的な検索をするために考案されたものでしかない。特に、コード化がきちんとできるものでないと検索処理そのものの実用速度が出ない。医療分野でのIT化で面倒なのは情報の形態がリレーショナルデータベースととても相性が悪いことにある。

たとえば以下のようなシステムはRDBではないか、補助的にRDBを使っているにすぎない。

・X(Twitter)
・MUMPS (後継はIntersystems Cache) 初期の医療情報システム
・いわゆるファイルシステム (NTFS, XFS, HPFS など)
・PACS (DICOM画像アーカイブ)

用語を整理するためのツール
・マインドマップ
・アウトライナー

記号の読み方とASCIIコードを覚える

変数を使うことの基礎を理解する　　ここまでは言語関係ない

テキストファイルを操る

バイナリファイルを操る

信号を操る

人のツールを使う
・API
・Learn by example

人に使わせるツール
・バリデーション
・コンベンション
・コメント

再利用できるツール
・関数の定義

ビジュアリゼーション

文芸的プログラミング

アルゴリズムに目が行きがちだが、私はアルゴリズムについてはリコーズ、つまり自分で一度は考えてみるようにしないといけないと思う。有名なソート(並べ替え)も自分で一つは考えてみて、それから他の人のアイデアに触れるのがベストだろう。

高度なアルゴリズムとは

初歩:
ループ、条件分岐

リカーシブコール

文字列の評価
・処理されるべきデータである文字列をプログラムとみなして実行する機能

巡回セールスマン問題

ポリモーフィズム
・特定のデータ型に対してしか定義されていない処理を別の型にも使えるようにする

バックトラッキング

アウトオブオーダー
並列実行

アルゴリズム主義から、フォーマット主義へ

プログラミング=アルゴリズムだと思っていたら、大間違い

巷では、多くのプログラミング教室が開催されています。

いくつか体験したことがありますがデータフォーマットについて説明しているところは見たことがありません。

私自身も、最初はロジックがとても大事だと思って

- Numerical Recipesのような有名な本をバイブルだと思ったり、
- Algorithms + Data Structures = Programs をすごいと思ってしまったり

しましたが、21世紀になってからは、以下の理由からデータフォーマットを軸に見た方が正解だと思うようになりました。案外楽しいものになっていますよ。

言語やアプリは短命だが、データフォーマットは長生きする

プログラミング言語には、流行廃りがあり、アプリに至ってはさらに短命です。

実は、時間の流れに強いのは、データフォーマット（正式には交換データ形式、Interchangeable Data Formatという）であることを知るべきです。

この、データフォーマット、初期のものはただの状態保存を目的としたので、あくまでもアプリケーションが存在してこそのデータでしたが、アプリケーションが使えなくなると、データがまとめて死んでしまうと言う問題に直面します。

- 80年代に重宝されたワープロの保存データはいまのパソコンではほとんど読み取れません。
- 70年代に作成されたアプリケーションは、いまのパソコンでは実行できません。

文書データやアプリケーションにおいて、新しければ新しいほど良いと言うのは正しくありません。

時折、新しいものほど優れているんですよと言い出して、システムの更新を継続させる輩がいます。そういうビジネスもいいでしょう。私は、そんなの当人が利益を死守しているだけで人類が先に進まないので好きではありません。

問題は、更新に伴って商売のツボ、あるいは処理の本質を押さえていたロジックが捨てられていくと、実は実態はどんどん悪くなるということです。

- 飛行機の予約システム更新に伴い、以前はみんながばらけるように席を提案していたのを、効率的に乗り降りできるようにするために、奥の方から順番にすし詰めにすると言うロジックを入れて客を他社に奪われて嫌われてしまったり
- 人材派遣会社のシステムで、派遣社員の顧客からの評価に根拠のない係数をかけて、結局、重要な顧客に使えない人を送ってしまったり
- 銃に、本人の認証システムをつけてしまったり

経験あるプログラマは、データの中にもロジックが存在することを知っている

後々のことまで考えるとデータフォーマットを正しく設計し、内容を説明した文書を共有するようになります。やがて、データフォーマットをデコード、つまり再現するためにアルゴリズムが発動するメカニズムが最も便利だ、となるのです。

API (えーぴーあい)

なぜITエリートの給料はすごいのか?

なぜ英語力が求められるのか?

そして、

その病院はなぜ、
いつまでも昭和のままなのか?

基礎知識　IFTTT(イフト) というサービスを知ってますか?

IFTTTというクラウドサービスがあります。自動化プラットフォームと呼ばれるもので、

Google, Facebook, TikTok, EverNote などの既存のクラウドサービスを組み合わせて、自分が欲しいクラウドを作ることができる、というサービス

例えば、

★スマートフォンの位置情報サービス（ロケーションサービス） + メール

= 乗換駅に着いたら、自動的に家族にメールを送る

★工場の温度管理サービス + メールサーバー + SNS

= 障害が発生したら、関係者に自動でメールを送信し、さらにSMSやSlackで知らせる

★監視カメラに特定の人が映ったら、特定のハッシュタグを付けたTwitterのつぶやきをXに自動投稿

★子供が学校の最寄りの駅の改札を通った時に母親のスマホにメールが飛んでくる

★Instagramの写真をDropboxにバックアップする

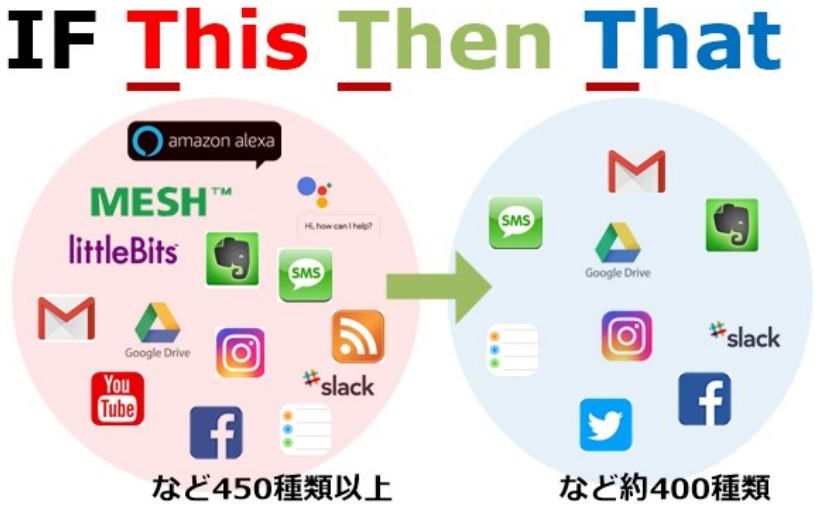
こんなサービスを自分で作れる。このサービスのポイントは

1. 既知のサービスを組み合わせるので、開発がほとんどない、が全く不要である。

2. いったん作ったサービスは人手を介さずに動作する、ということ。

それぞれのサービスに一切変更を加えずに、

なんでこんなことができるのか?



さまざまなサービスが連携できる「IFTTT（イフト）」の使い方

https://atmarkit.itmedia.co.jp/ait/articles/1711/22/news031.html

IFTTTは、(まともな)クラウドサービスが持つAPIを利用している

■　(前ページから続く)

IFTTTが動作する秘密は API (アプリケーションプログラミングインターフェイス)があるからです。

アメリカのIT企業で開発されているウェブサービスの多くは、最初から自動化を考えているものがほとんどで、APIの設計からスタートしています。画面のデザインは後回し。これが自動化を簡単に実現しています。

APIというのは、かなり噛み砕いて説明すると、

サーバーをリモートコントロールするための、(コントロールする側の)機械のためのホームページの集まり。

【重要】　人間のためのページではない。だからデザインどころか色すらない

日本ではここに大きな誤解があり、インターネット上で動作しているサービスだったら、全てクラウドであると言うふうに言い換えられてしまっています。これらは2000年の頃に登場した ASP (アプリケーションサービスプロバイダ) と呼ばれるもので、(アメリカでいう)クラウドを名乗る資格はない。

作る順序が逆なのよ。

これが決定的な差となるのです。

実は、APIが作られるのはIFTTTのためではありません。特に大規模化、つまり数十から数万台で世界中のユーザーにサービスを提供しながらも機能をどんどん追加するので、サーバーをオートテストする必要があり、手作業で動作確認なんかしていられないのです。この設計構造の違い、というか文化の違いが、もうどうにもならなくなってしまっています。

■　ITエンジニアの報酬水準にあるとんでもない錯覚

アメリカの場合は、Webサービスの開発はほとんどインハウスで行われています。
対して日本の場合はほとんどが委託業務です。委託業務の場合は定期的な進捗報告の時に、仕事がさも進んだように見せる必要があります。そうなると当然APIなんかどうでもよくて、とにかく動作するやつを作れば、誰も文句は言わないわけです。(実際にはアメリカでも最初はAPIは重視されてなかった)

これ、90年代にWebサービスの時代が来るぞ、てなったときに最初これで日本も変わると思ったんですけど、結局何事もカタチから入る国なものだから、何にも知らないWebデザイナーの営業力が強すぎて、結局APIを書く人間たちが奴隷扱いとなっているんです。

アメリカでAPIを書く仕事は案外大変で、ここに多額の給料が支払われています。日本は砂上の楼閣、しかもそれを作っている人間に価値を認めずに安い給料を設定し、美しいデザインを提示しておじいちゃんたちを喜ばせる方々に、なぜかとんでもない報酬が支払われているパラダイスとなっているのです。

そして、もっと恐ろしいことが起きた、、、(次頁へ続く)

日本発の**HA(**ホームオートメーション**)**は否定され、**IoT**が登場。日本家電は窮地に・・・・でも気づかない**!?**

■　(前ページから続く)

①まず、ハード面

・半導体技術の目覚ましい発展で、わずか数ミリ角の部品にサーバーを作り上げることが可能になる。

・スマホは10年前のサーバーよりも高性能!

②次にソフト面

・オープンなOSであるLinuxがルーター機能を搭載

→インターネットの普及で一役買い、数百万円していたルーターが数千円に

※余談になりますが、みんなルーター持ってるでしょう？だからみんなLinux使ってるんです
※なお、AndroidもベースはLinuxです。

このLinuxがWindowsなどとは異なり、元々サーバーOSだった
↓
デジカメや、冷蔵庫、洗濯機、便座、監視カメラにまでサーバーが入って、リモートコントロールできることに。

これらをアプライアンスとか、IoTと呼びます。

ここで重要な事は、Linuxを搭載するアプライアンスであるからには、APIは絶対に持っていると言うことです。(ソニー以外の日本ブランドは知らんけど)

もしメーカーがエンジニアにAPI書かせていないとすれば、経営管理が全然ダメということ。そんな会社の製品を買ってはいけないし、勤めるにしても長居は無用と判る。商社である場合ははめられていると思って良い。

Linuxに背を向けてWindowsに尻尾を振った結果がこれです。そして、いまでもその姿勢は変わらない。政治力とか、英語力とか関係なく、ソフトでこの国はまた負けるのです。

エコー装置。ユーザーは画質しか気にしてないが、システム設計はGEがはるかに先進的。日本勢はひたすらGEを追いかけているだけ。

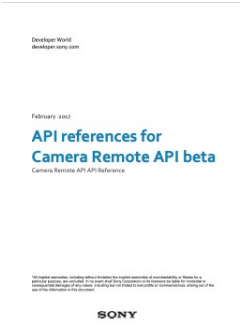
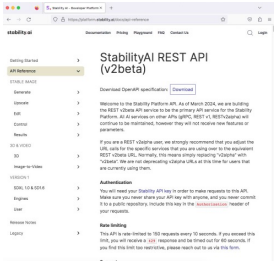
日本製のカーナビ(左)。もう何十年も進化していない。最初はアメリカのカーナビが道路しか表示しないので、かなり単純だと馬鹿にしていた・・・Apple がCarPlayを出してきてアプリが車と連動するまでは。

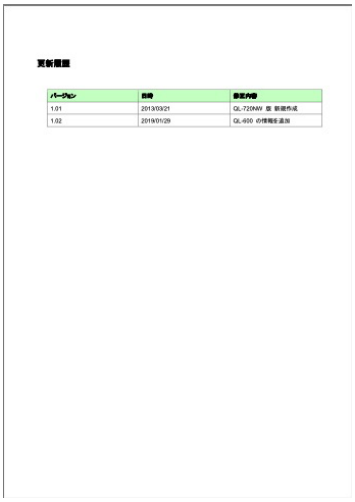
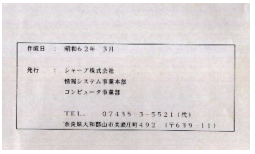
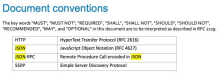
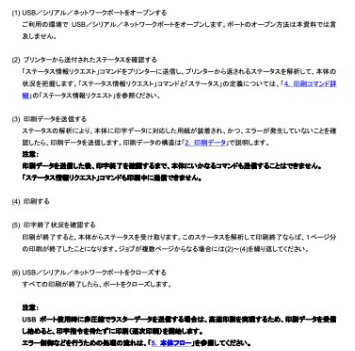
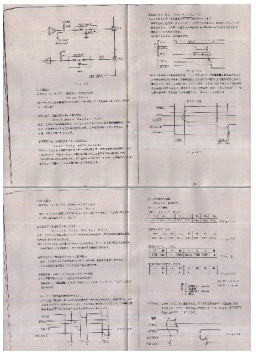
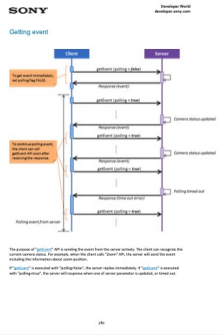
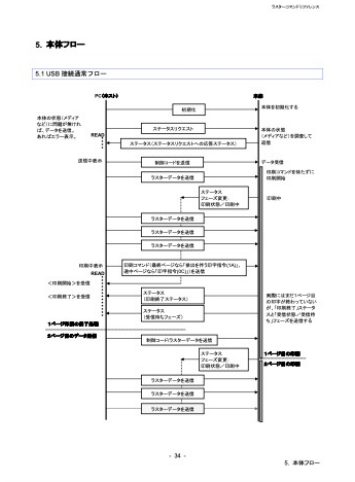
実際のAPIはどんなものなのか?

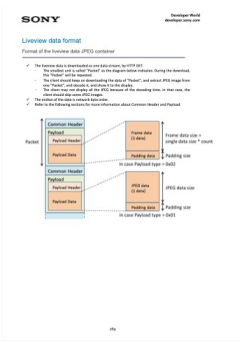
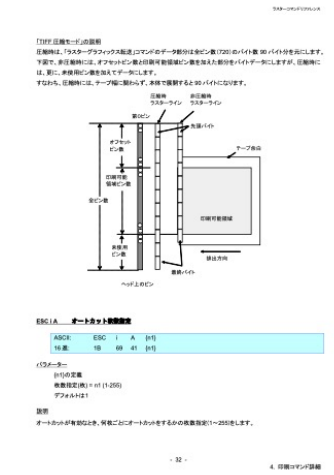
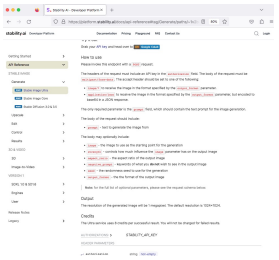
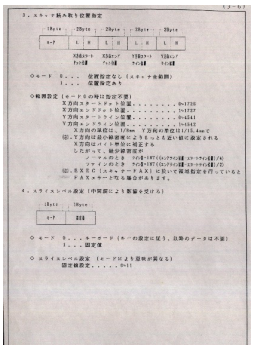
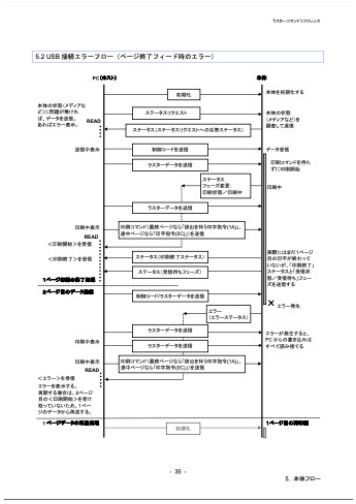
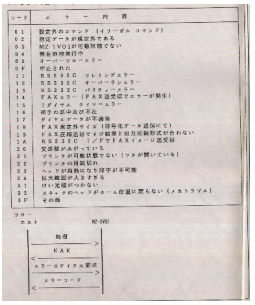
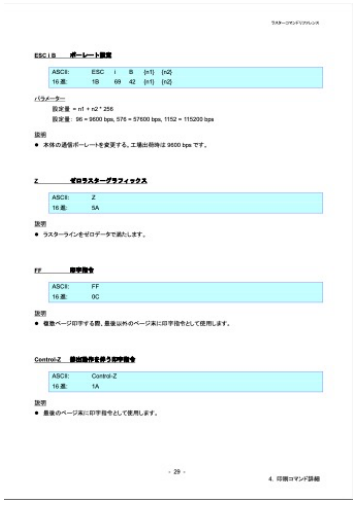
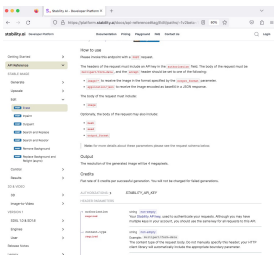
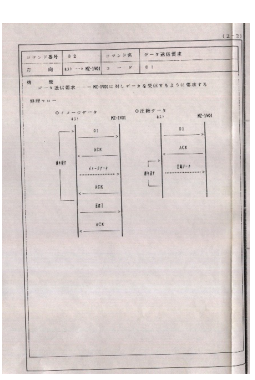
ここではAPIはどういうものなのか実物を見てみます。

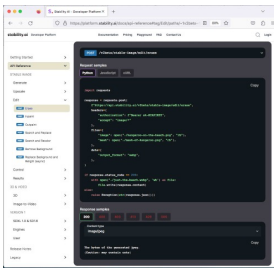
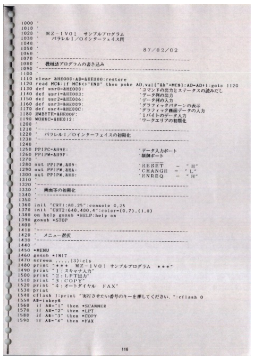
基本は API Specification とか API Reference というマニュアルで出てきます。書かれているのは、以下のようなものになります。

- リビジョン(変更履歴、修正履歴)　細かい変更点等について詳しく書きます。大変重要です。
- プロトコル
- フローシーケンス図　アクセスする側と、される側のやりとりする内容と、その流れ。
- コマンド-レスポンスの一覧
    - 認証方法
    - ペイロード（＝やりとりするデータ)の内容
    - ペイロードの転送方法
    - エラーコードの一覧

セクション	ソニーデジカメ	ブラザープリンタ	Stable Diffusion API	シャープFAX
表紙				
	ソニーのデジタルカメラαシリーズのAPI仕様書です。これをもとにして当社ではαから直接DICOMに取り込むカメラアプリを作成。たぶん世界で唯一だと思います。	ブラザー工業のラベルプリンタのAPI仕様書。 多くのモデルはUSB接続ですが、これはWi-Fiで接続するタイプのものです。これをもとにバーコードラベルの出力機能を作っています。	Webサービスの一例として、いま話題の生成AIの中からstble diffusionを選んでみました。こちらはWebサイトとして公開されています。	参考のために昭和の頃に作成された、シャープの技術文書も掲載します。一般には入手できませんがご厚意でコピーさせていただいたものです。これをもとにスキャナとして使えるプログラムが書けました。

セクション	ソニーデジカメ	ブラザープリンタ	Stable Diffusion API	シャープFAX
変更履歴	 全部で280頁ほどあります。		見つけられませんでした。	 変更履歴はありませんでしたが、発行日はちゃんと書いてあります。このころにはもう当たり前なわけです。
プロトコル	 コンベンション(用語の定義、共通ルールなど)を兼ねてSSDP、及びHTTP上でJSON-RPCを用いることが説明されています。	 機種によってUSBバルク通信、シリアル通信、TCP/IPのいずれかを使えることが説明されます。USBは非常に複雑でバルク通信以外にシリアル over USB、TCP/IP over USBなどいろいろあり、仮にUSBのバルクといってもエンドポイントという装置番号を探す手間があったりと大変です。	最初タイトルに REST API と書いてあるので、HTTPとわかります。	 シリアルとパラレルが選択できること、ケーブルの結線パターンから通信パラメータまで書かれてます。
フローシーケンス	 全8ページ	 通常操作のフローで5ページ	複数ステップにおよぶフローがないからか、シーケンスは特に書かれていないようです。	フローはコマンド毎に丁寧に書かれています。

セクション	ソニーデジカメ	ブラザープリンタ	Stable Diffusion API	シャープFAX
ペイロード情報	 6ページ。通信時のペイロードと保存時のペイロードは内容が異なることなどが説明されています。			
エラー対処	 全2ページ	 本書ではエラー処理もシーケンス図になっている。 先のフロートは別に4ページ	Responses › 200 Generation was successful. › 400 Invalid parameter(s), see the errors field for details. › 403 Your request was flagged by our content moderation system. › 413 Your request was larger than 10MiB. › 422 Your request was well-formed, but rejected. See the errors field for details. › 429 You have made more than 150 requests in 10 seconds. › 500 An internal error occurred. If the problem persists contact support. 個々のコマンド毎に書かれています。	
コマンド	 これがAPIのいわば本体です。このようなフォーマットで、1機能に1ページ以上を割り当てて、243ページに及んで続いています。	 14ページしかないが詰め込んである。		

セクション	ソニーデジカメ	ブラザープリンタ	Stable Diffusion API	シャープFAX
サンプルプログラム	プログラムソースが添付されていました。	プログラムソースが添付されていました。	 コマンドの説明毎にサンプルが示されています。	

そして気づいて欲しいこと

一般にページ数はユーザーマニュアルよりも膨大で、かつちょっとしたミスも許されない内容の集まりになります。

ソニーの例ではAPI文書は、英語版しか公開されていません。そうなんです、英語ができなければほぼ開発はできないことになります。(翻訳しているくらいならわかる人に代わってもらうのが経営的には良い) 一般民生向けのデジカメがこれですから、業務系のシステム開発で英語ができるかどうかは決定的です。

洗練されたアプリケーションはこのようにAPIブロックと、ユーザーインターフェースブロック(=みなさんが目にするメニューやボタンのこと)が分離されていて、別の職業といっても良いくらい求められるスキルと、給料に差があるのです。ユーザーインターフェースが分離していれば デスクトップパソコン版とモバイル版を複数出すのも容易になりますよね。逆にできていないとOSがアップデートしただけで動かない、 開発担当者が辞めてしまったのでもう対応できません、と言う事態に陥るわけです。どこの耳鏡でしたっけ？

デジカメのAPI活用例。数十台のカメラを同時に制御して3Dモデリングしている。人間では無理。

http://thegnomonworkshop.com/