

佐賀大学全学教育機構

「情報科学の世界II」 講義資料

担当者: 山下義行

R4 年度後期

目次（一部変更の可能性あり）

第 1 章	はじめに	1
1.1	講義の概要	1
1.2	授業の進め方	1
1.3	成績	1
1.4	講義資料など	2
第 2 章	様々なコンピュータ・システム	3
2.1	システムの規模による分類	3
2.2	システムの用途による分類	6
2.3	休憩: 数量を表す修飾子	9
第 3 章	情報科学/工学の歴史	10
3.1	ENIAC 以前	10
	紀元前 3 世紀	10
	12 世紀	10
	17 世紀	10
	19 世紀	10
	1910 年代	11
	1920 年代	11
	1930 年代	12
3.2	ENIAC 以後	12
	1940 年代	12
	1950 年代	12
	1960 年代	13
	1970 年代	14
	1980 年代	15
	1990 年代	16
	2000 年代以降	17

第 4 章 コンピュータの基本構成	19
4.1 基本構成要素	19
4.2 入力装置	19
4.3 記憶装置	20
4.4 演算装置	21
4.5 出力装置	21
4.6 記憶装置のないコンピュータ？	22
4.7 並列コンピュータ	23
第 5 章 p5.js プログラミング（基本編）	26
5.1 プログラミング言語 p5.js	26
注意：	26
参考：	27
5.2 動かしてみる	27
注意：	28
5.3 簡単な p5.js プログラムの実行（文字列や数値の出力）	32
注意（p5.js、JavaScript のセミコロン問題）：	34
5.4 命令の逐次実行（30 秒後の時刻の計算）	36
時刻（秒）の取得	36
変数	37
代入文	37
5.5 条件判定と実行の分岐（時間制の変更）	41
if 文	41
条件式	41
コメント	42
if-else 文（あるいは単に、if 文）	43
else if 文	45
考察	47
考察	47
考察	48
5.6 段下げ（インデントーション）	49
5.7 フローチャート（プログラムの図式化）	50
5.8 繰り返し実行とループ（階乗の計算）	53
while 文	54

考察	56
考察	56
考察	57
5.9 定型的な繰り返し実行	59
for 文	59
注意：	60
5.10 データの列	60
配列	61
第 6 章 p5.js プログラミング (グラフィックス編)	62
6.1 様々な線画	62
6.1.1 三角形	62
6.1.2 正多角形	63
考察	63
6.1.3 星型正多角形	65
6.1.4 らせんと渦巻き	65
考察	65
6.1.5 RGB カラー	67
6.1.6 線の色、幅の変更	68
考察	69
6.1.7 その他の形状	69
6.2 様々な面画	70
6.2.1 三角形	70
考察	71
6.2.2 様々な形状	71
円弧	71
考察	72
楕円	72
長方形	72
考察	72
6.3 CG アニメーション	74
6.3.1 様々な CG アニメーション	74
アニメーションプログラムの基本形	74
左から右へ黒い領域が広がる	74

左から右へ線分が移動する	75
星が回転する	76
星の回転速度を変える	77
6.3.2 イベント処理、その他	78
マウスの位置	78
考察	78
マウスのクリック	78
考察	79
考察	79
文字列の描画	79
6.3.3 モグラたたき風ゲームの開発	80
考察	82
第 7 章 p5.js プログラミング (計算編)	83
7.1 平方根の計算	83
(1) 二分法	83
考察	85
(2) モンテカルロ法	85
考察	87
(3) ニュートン・ラフソン法	87
考察	89
7.2 円周率の計算	89
(1) n 角形と $2n$ 角形の漸化式	90
(2) モンテカルロ法	92
(3) 逆三角関数の公式	93
(4) ガウス＝ルジャンドルの反復計算	95
第 8 章 アルゴリズムとプログラム	97
8.1 プログラム開発手順	97
8.2 アルゴリズムとは	98
第 9 章 小数点数の不思議	101
9.1 コンピュータ内での数値の種類	101
9.2 浮動小数点数	102
9.3 浮動小数点演算の誤差	103

(1) 丸め誤差	103
(2) 情報落ち	104
(3) 桁落ち	105
考察	105
第 10 章 さまざまなプログラミング言語	106
10.1 プログラミング言語の種類	106
10.2 低水準プログラミング言語	106
10.2.1 機械語	106
10.2.2 アセンブリ言語	106
10.3 高水準プログラミング言語	107
10.3.1 FORTRAN	108
10.3.2 COBOL	109
10.3.3 ALGOL	109
10.3.4 Pascal	111
10.3.5 C	111
10.3.6 C++	111
10.3.7 Basic	112
10.3.8 Ada	112
10.3.9 Java	113
10.3.10 JavaScript	113
10.3.11 Python	115
10.4 プログラミング言語の比較	115
10.5 補足	116
10.5.1 高水準プログラミング言語の状況	116
第 11 章 p5.js プログラミング (インターネット編)	118
11.1 コンピュータネットワークの歴史	118
11.1.1 TCP/IP	118
寄り道	119
11.1.2 World Wide Web	120
11.1.3 URL	120
11.2 p5.js プログラムによる Web サーバへのアクセス (気象予報)	122
11.2.1 佐賀の気象予報データをテキスト形式で取得する	123
11.2.2 佐賀の気象予報データを読みやすく表示する	129

11.2.3 佐賀の気象予報データをキャンバスにビジュアルに表示する	134
11.2.4 佐賀県のデータを福岡県のデータに変更する	134
11.3 p5.js プログラムによる Web サーバへのアクセス（地震情報）	137
11.4 p5.js プログラムによる Web サーバへのアクセス（その他）	137
第 12 章 まとめ	140

第1章 はじめに

ここでは授業の内容、授業の進め方、成績評価の方法について述べます。ただし、コロナ禍など様々な状況の変化で変更されることもあります。

1.1 講義の概要

この科目は、コンピュータについて全く知識のない学生を対象にしています。

前半では、初心者が抱くであろう素朴な疑問: コンピュータの種類、用途、コンピュータ開発の歴史についてゆっくりと丁寧に解説します。

後半では、様々な種類のコンピュータプログラミングを体験します。コンピュータを知る上で最も手っ取り早いのはコンピュータのプログラムを作ってみることです。主にプログラミング言語 p5.js を用いたプログラミングを解説し、コンピュータの様々な利用方法を紹介します。

1.2 授業の進め方

当初、対面授業を予定していましたが、履修者数が 100 名を超えるため、教室が確保できず、オンデマンド授業へ変更することとします。

講義は原則、オンデマンドで進めます。講義資料 (pdf ファイル、解説動画) は Microsoft Teams にアップロードします。

1.3 成績

小テスト点、期末試験点で決定します。

毎週の小テスト 以下の二つからなります。

確認テスト その週の授業内容について簡単なオンライン試験を実施します。成績点を 2 点 (満点の場合) とします。

疑問、質問、感想の入力 授業内容について疑問、質問、感想を記入してください。成績点を 1 点 (ただし 20 字以上入力していること) とします。

- 1 授業は全部で15回ありますから、小テストの成績点の合計は45点です。
- 2 **期末試験点** オンライン試験で実施します。55点満点です。
- 3 小テスト点、期末試験点の合計が60点以上で、この科目の合格です。

4 **1.4 講義資料など**

- 5 授業の講義資料、様々な解説動画は Microsoft Teams で配信します。
- 6 小テストは佐賀大学オンライン試験システム：
<https://opes.oge.saga-u.ac.jp/>
- 7
- 8 で行います。そのためのPCの設定については解説動画で説明します。

第2章 様々なコンピュータ・システム

講義の始まりとして、この章ではコンピュータの種類と用途について概説し、コンピュータとは何かという疑問に実例に通して紹介します。

コンピュータのことを初めて学ぶ人には内容や言葉が難しいと感じられるかもしれませんが、気にする必要はありません。概要を理解するだけで十分です。

2.1 システムの規模による分類

コンピュータを規模（占有面積、体積、価格）で分類することは最も簡単な分類法です。小さい方から見ていきましょう。

組み込み型コンピュータ：いまやほとんどの電気製品（洗濯機、電子レンジなど）の中には極小規模のコンピュータが組み込まれています。ここで本質的な疑問として、これらをコンピュータと呼んでいいのかという問題があります。これについては4章で議論します。

今はまだ限定的ですが、今後これらの組み込み型コンピュータが当たり前のようにインターネットと接続されるようになれば、生活に劇的な変化をもたらすと考えられています。IOT（Internet of Things、もののインターネット）がそれです。

携帯電話、スマートフォン：単なる電話ではなく、コンピュータです。むしろ携帯電話、スマートフォンを電話として使用している頻度は、今や極めて低いのではないのでしょうか。コンピュータに電話機能が付属していると言った方がよいでしょう。

タブレットコンピュータ：機能的にはスマートフォンと同等ですが、サイズが大きい分だけハードウェアの構成に余裕があり、結果、総合的な性能はスマートフォンよりも高く、操作も容易です。最近の小中高生では、タブレットを用いた学習が一般的になりつつあります。

家庭用ゲーム：ゲーム機の中身もコンピュータです。しかもひとつの規格で膨大な数が売れるため、薄利多売が効き、下記のパーソナルコンピュータなどと比較するとかなり安価です。最近ではゲームのシェアはスマートフォンやタブレットコンピュータに奪われつつあります。

パーソナルコンピュータ：個人用コンピュータの総称です。なおパソコンという呼び方は日本独自です。世界共通ではもっぱらPC（Personal Computer）あるいは単にコンピュータと呼びます。

ノート PC: ノートほどには薄くも軽くもないものの、軽量化も進み、今や高価な文房具のひとつとして仕事になくてはならないものです。低価格化、高性能化、加えて可搬性¹という特徴から下記のデスクトップ型はノート型にほとんど取って代わられました。

デスクトップ PC: 文字通り、机の上において使用するタイプの PC です。ディスプレイと本体が分離されているものが一般的ですが、最近ではディスプレイの背後に本体を組み込んだタイプ (iMac など) もあります。

一般にノート PC よりも高性能ですが、ノート PC の性能向上 — 文房具としてはもはや十分な性能に達しています — および処理のクラウド化によってデスクトップ PC の必要性は大きく低下しています。

デスクトップ型ハイエンド PC: 研究室での少し負荷の高い計算や高速演算を要求されるゲームには高価格帯のグラフィックスプロセッシングユニット (GPU) を搭載した PC が利用されています。特に後者用途のものはゲーミング PC などと呼ばれています。

ワークステーション: ハイエンド PC をさらに高速演算、高耐久、高信頼に性能向上させたコンピュータを指します。価格はハイエンド PC の数倍程度〜10 枚程度です。

歴史的には PC にやや先駆けて誕生し、これまで主に高性能 PC という位置付けで利用されてきました。しかし、2000 年以降、PC の高機能化が急速に進み、ハイエンド PC とワークステーションの区別はあいまいになっています。

サーバ (サーバ用ホストマシン): PC、ワークステーションが個人利用を想定しているのに対して、高速ネットワークを介して多数のクライアントの仕事を処理するコンピュータです。ここでいう仕事とは、メールサービス、ウェブ処理、データベース処理、ファイル処理、プリンタの管理、トランザクション処理 (銀行業務のコンピュータ処理など) などを指します。サーバ用ホストマシンは複数台のコンピュータから構成されることもあり、高速処理、連続稼働を高い信頼性と共に実現する必要があります。たとえば東京証券取引所のコンピュータは、現在、約 400 台のコンピュータで構成されたサーバです。1 台当たりの価格は数百万円程度です。

大型コンピュータ: 汎用大型コンピュータ、メインフレームとも呼ばれます。コンピュータが高価であった時代には今の PC 以下の能力しか持たないコンピュータがビル 1 棟を占有するほどの大きさでした。図 2.1 は、歴史的な名機として名高い IBM System/360 の写真です。価格は 1 台当たり (現在の物価で) 数十億円から数百億円です。

歴史的な役割はほとんど終わっており、最近では上記のサーバに取って代わられています。上に例示した東京証券取引所のコンピュータは、2010 年に大型コンピュータからサーバへ変

¹蛇足ですが、ノート PC は衝撃に弱いので、自転車のかごに直接入れて運搬などの扱いをすると確実に壊れます。



図 2.1: 大型コンピュータの例：IBM System/360（Wikipedia“System/360” から転載）

1 更されています。しかし、高い信頼性を有するため、高価（コストパフォーマンスが悪い）
2 ですが銀行業務など特に高い信頼性が要求される業務には今でも利用されています。

3 **スーパーコンピュータ：**高い演算能力を持つコンピュータの総称です。最近では1,000PFLOPS²を
4 超える能力のものも開発されています。実行方式は歴史と共に変遷しており、現在は超並列
5 コンピュータ³が主流です。TOP500⁴と呼ばれるスーパーコンピュータのランキングが有
6 名です。日本最速のスーパーコンピュータ：富嶽が2022年8月現在、世界2位にランクさ
7 れています。

8 図 2.2 は一世代前の日本のスーパーコンピュータ：京の筐体群を側面から見た写真です。
9 800 個を超える筐体が計算機室のフロアに敷き詰められています。

10 **分散コンピュータ：**ネットワークを介して複数のコンピュータを結合し、並列計算を実行する仮
11 想的なコンピュータシステムです。その規模は様々であって、定義上はインターネットを介
12 して世界中のコンピュータをつなぐ場合から、ひとつの研究室内の数台のコンピュータをつ
13 なぐ場合まで含まれますが、一般には世界規模の分散コンピュータを指します。

14 たとえば、“SETI@home”では、電波望遠鏡の信号をコンピュータ処理し、地球外知的生
15 命体の証拠を発見するプロジェクトが、世界中の余剰コンピュータ（夜間に使われない PC
16 やゲーム機）を利用して行われました（このプロジェクトは2020年から休止しています）。

²FLOPS は FLoating point operations Per Second の略であり、加減乗算を1秒間に何回実行できるかを表す単位です。P（ペタ）は10の15乗を表します（2.3節参照）。1 PFLOPS は加減乗算を1秒間に10の15乗回（=1000兆回）実行できることになります。なお、除算は加減乗算に比べ複雑な処理のため、FLOPSの対象からは外されています。

³複数台のコンピュータを同時に実行させる型のコンピュータを並列コンピュータと呼びます。超並列コンピュータでは数千台、数万台のコンピュータを同時に実行させます。

⁴<https://www.top500.org>



図 2.2: スーパーコンピュータの例: 京 (日本、理化学研究所) (Wikipedia“京 (スーパーコンピュータ)” から転載)

グリッドコンピュータ: 分散コンピュータの一つですが、特に高速通信と高速計算を目指したものを指します。グリッドの名前の由来は電力網 (GRID) とされ、ちょうど送電線を通して送電される電力をコンセントから手軽に使用できるように、通信回線で繋がれたコンピュータの膨大な計算パワーを手軽に使用できることを指しています。関連した概念としてクラウドコンピュータがありますが、これはコンピュータの規模、形態というよりもその用途を表す術語と考えた方がわかりやすいため、次節で述べます。

2.2 システムの用途による分類

今やコンピュータの利用は社会の隅々にまで浸透しています。その利用の内容を列挙します。

大規模数値計算: コンピュータはまさに高速計算機械としての要求から開発されました。コンピュータは単に高速計算を行う道具ではありませんが、計算が主要な用途であることには変わりはありません。コンピュータによる演算 ($1 + 2 = 3$, $0.3 \times 0.4 = 0.12$ などの計算) を特に数値計算 (numerical calculation) と呼びます。数値計算は今やほとんど全ての科学/工学に必須の技法です。素粒子質量の計算、銀河系の生成シミュレーション、建物の耐震解析、気象予測、核兵器の設計、自動車の車体の設計、タンパク質の性質の解析、人口知能の機械学習など無数の応用が考えられます。また計算物理、計算化学などの数値計算に特化した科学分野も確立されています。

事務処理: 会社における給与計算などが典型です。かつては大型計算機を用いていましたが、今や同等の仕事は PC で十分にこなせます。計算処理は PC で十分ですが、会社などでの事務処理の場合、依然として紙面書類の作成などが必要であるため、それ相応の周辺機器が必要です。最近のデジタルトランスフォーメーション (DX) の流れは、紙面の削減にとどまらず、

旧態依然としたビジネス状況を変革していくでしょう。

金融オンラインシステム: コンピュータの最も大きな用途は、数値計算よりも金融のオンラインシステムかもしれません。第一に顧客データ（預金高、氏名、住所など）を正しく管理する仕組みが必要です。また個人顧客レベルでは銀行端末からの預け入れ、引き出し、振り替えを正確に遂行する仕組みも必要です。さらには各金融会社間の資金の移動も円滑に正確に行う必要があります。金融オンラインシステムではこれらを1回のミスもなく何年も稼働し続ける完成度の高いコンピュータプログラムを作ることが要求されます。商用コンピュータの販売と同時に始まった金融オンラインシステムの開発は既に40年以上に渡って継続されており、システム全体はもはや個人の管理能力をはるかに越えるほど巨大化していると言われていいます。最近のみずほ銀行でのシステム障害はそのシステムの複雑さを物語っています。

システム開発側の視点から見れば、ソフトウェア会社の主要な顧客は銀行を始めとする金融関係です。もしあなたがソフトウェア会社にプログラマーとして入社したならば、かなりの頻度で金融業務に関連するプログラム開発を担当することになるでしょう。

情報検索、データベース: コンピュータは単に計算をするための道具ではありません。コンピュータを用いて、膨大なデータを管理することも可能です。そのような目的のためのシステムはデータベース（database、データの蓄積基地）と呼ばれています。データベースでは単に大量のデータを蓄積するだけではなく、その中から必要なデータを適切かつ高速に検索できる仕組みが必要です。

たとえば図書館の蔵書データベースは今や図書館利用には欠かせないものです。かつてそれらは紙製の蔵書カードで管理されていました。しかし今はコンピュータ端末から瞬時に蔵書を検索可能です。最近では日本全国の大学図書館の蔵書を同時に検索することも可能です。

多くの会社は顧客データベースを持っています。最近ではビッグデータと呼ばれる様々な統計情報がビジネスに利用されています。それらを有効に活用することは今や非常に重要です。

また忘れてならないことに、インターネット上での情報検索があります。利用者が必要とする情報を簡単なキーワードからの的確に検索するために、優秀な検索手法や検索ソフトウェアの重要性が増しています。逆にユーザの立場からは、必要な情報を的確に検索する技術を身につける必要があります。

ビデオゲーム: 家庭用TVを映像の出力先とするビデオゲーム（TVゲーム）は、今やコンピュータ利用の重要な分野のひとつになっています。最近ではインターネットを利用したネットワークゲームも流行しています。

デジタル家電: 組み込み型コンピュータの話は2.1節で取り上げました。それらを組み込んだ家電品を特に「デジタル家電」と呼ぶようです。ひと昔前の制御用マイコンと異なり、最近の組

み込み型コンピュータは通信機能を持っていることが特徴です。

クラウドコンピューティング： これは、ネットワークを介したコンピュータの新しい利用方法です。コンピュータの形態や規模を表す言葉ではなく、むしろ利用形態のひとつです。種類、数、規模が雲（クラウド）のように不確定で膨大なコンピュータの集合体にネットワークを介して可能な様々なサービスを総称してクラウドコンピューティングと呼んでいます（amazonでできることを思い起こしましょう）。コンピュータ・ネットワーク、インターネットが進歩したことによるコンピュータの新しい利用方法です。現在も急速な拡大を続けています。技術的にはグリッドコンピュータを用いる場合がありますが、コンピュータの形態自体は雲の向こうに隠れて見えない、見えなくてもよいという位置付けです。

クラウドによるサービスには主に以下の3種類が考えられます。以下は、総務省の解説記事：

クラウドサービスとは？

https://www.soumu.go.jp/main_sosiki/cybersecurity/kokumin/basic/basic_service_13.html

からの引用です。

SaaS（サース、サーズ：Software as a Service）... インターネット経由での、電子メール、グループウェア、顧客管理、財務会計などのソフトウェア機能の提供を行うサービス。以前は、ASP（Application Service Provider）などと呼ばれていました。

PaaS（パース：Platform as a Service）... インターネット経由での、仮想化されたアプリケーションサーバやデータベースなどアプリケーション実行用のプラットフォーム機能の提供を行うサービス。

IaaS（アイアース、イアース：Infrastructure as a Service）... インターネット経由で、デスクトップ仮想化や共有ディスクなど、ハードウェアやインフラ機能の提供を行うサービス。HaaS（Hardware as a Service）と呼ばれることもあります。

ネットワーク管理： たとえば日本からアメリカへコンピュータネットワークを介してデータを送信した場合、そのデータは日本とアメリカをつなぐネットワーク上の複数のコンピュータをパケット・リレーしてアメリカへ届くことになります。この際のパケットリレーを実行することもコンピュータの大きな役割になります。また日本からアメリカへの経路は複数存在しますから、どの経路を辿るか、それを決定することもコンピュータの仕事です。ネットワーク上をウイルスなどの悪意を持ったソフトウェアが流れないように防衛する仕組みも重要です。コンピュータ・ネットワークはそれ自身で（数値計算のような）特定の仕事をを行う訳ではありませんが、ちょうど高速道路が我々の社会生活に必要であるのと同じように、社会基盤とし

て重要です。今やコンピュータ・ネットワークが止まれば、社会活動のほとんど全てが機能不全に陥ります。今やネットワーク管理は高速道路の管理よりも重要かもしれません。

2.3 休憩: 数量を表す修飾子

コンピュータでは様々な数量単位 — bit (ビット), byte (バイト), Hz (ヘルツ), bps, FLOPS (フロップス) など — が使用されますが、大きな数を用いるときには特に K (キロ)、M (メガ)、G (ギガ) などの修飾子を付けて数字の数を減らし、見やすくすることがあります。これらはあまりに当たり前過ぎて誰も教えてくれませんが、知らないと恥ずかしい事項ですから、ここで確認しておきます。なお、余裕があるならば、これらの語の語源を調べてみてください。おもしろい発見があるかもしれません。

K (キロ) : $10^3 = 1000$ 。ただしメモリ容量を表すときには、1000 ではなく $2^{10} = 1024$ を K とします。

M (メガ) : $10^6 = 1000000$ 。ただし K と同様にメモリ容量などで使用されるときには、 10^6 ではなく、 $2^{20} = 1048576$ を M とします。

G (ギガ) : 10^9 。メモリ容量については同様です。

T (テラ) : 10^{12} 。メモリ容量については同様です。

P (ペタ) : 10^{15} 。メモリ容量については同様です。

E (エクサ) : 10^{18} 。メモリ容量については同様です。現時点ではまだこの修飾子を使用されることはほとんどありませんが、10 年後、20 年後にはかなり頻繁に用いられるようになるかもしれません。

コンピュータの性能を知る上で最も基本的な数値はコンピュータの動作周波数 (Hz)、演算能力 (FLOPS)、メモリ容量 (byte) です。現在、最高水準のコンピュータでは、動作周波数は G Hz、演算能力は P FLOPS、メモリ容量は P byte のオーダーに達しています。

1 第3章 情報科学/工学の歴史

2 ここではコンピュータの歴史について学びます。初めて見聞きする言葉の羅列にとまどうかもし
3 れませんが、暗記する必要はありません。言葉の意味を完全に理解する必要もありません。コン
4 ピュータの歴史の大きな流れを把握できれば、それで十分です。

5 3.1 ENIAC 以前

6 コンピュータが、現在の我々の知っている形で登場するのは第二次世界大戦前後です。3.2 節で
7 紹介する ENIAC はそのような最初のコンピュータですが、しかし、それ以前に幾つもの萌芽的な
8 事項が歴史に登場します。ここではまずそれらを概観します。

9 紀元前 3 世紀

10 1. アルキメデスが円周率を 3 桁まで計算。以後、円周率の計算は各時代のコンピュータ（や人
11 間）の演算能力を表すシンボルになります。

12 12 世紀

13 2. アルゴリズムの語源となる「Liger Algorithmi（アルフワリズム¹の書）」がペルシャ語からラ
14 テン語へ翻訳される。アルゴリズム（algorithm）は情報科学/工学にとってきわめて重要な
15 概念です（8.2 節で学びます）。

16 17 世紀

17 3. 円周率が 32 桁まで計算される。

18 19 世紀

19 4. イギリス人バベッジ（Charles Babbage）が、エイダ（Augusta Ada Byron）夫人²の支援の
20 もと、解析機械（analytical engine）を構想、試作する。イギリスはちょうど産業革命の頃

¹アルフワリズムは 9 世紀初めのアッバス朝の数学者とされています。

²詩人バイロンの娘でもあります。



図 3.1: タイガー計算機 (Wikipedia“機械的計算機”から転載)

1 (日本は江戸時代)であり、蒸気機関を動力源にする機械式コンピュータが構想されました。
 2 この解析機械は歴史的に忘れ去れていましたが、最近になって歴史上最初のコンピュータと
 3 しての評価が固まってきています。エイダ夫人は、歴史上最初のプログラマ³と呼ばれてい
 4 ます。

5 5. 数学者ド・モルガン (de Morgan) が著書「Formal Logic (形式論理)」を発表 (1847)。論
 6 理学は、数学のみならず、コンピュータの理論としても重要です。

7 1910 年代

8 6. さまざまな機械式計算機⁴が発明される。日本でも大本寅二郎が手回し式計算機「タイガー計
 9 算機」(図 3.1)を国産開発、発売 (1924) し、好評を博します。

10 1920 年代

11 7. タイプライタなどの事務機器メーカだった CTR 社が IBM (International Business Machine)
 12 社と改名⁵。後に世界最大のコンピュータ製造会社となり、PC が現れる 1980 年代まで世界
 13 の頂点に君臨します。

³このことに敬意を表して、アメリカ国防省によって設計主導されたプログラミング言語は Ada と名付けられました。

⁴正確には、計算機 (コンピュータ) ではなく、四則演算を高速に行う卓上計算機に近いものです。「高速」とは言え、手回し式なので 1 回の演算に数秒掛るという速度です。当時の理系大学生は、教授の研究補助で、朝から晩までタイガー計算機の取っ手を回していたそうです。

⁵映画「2001 年宇宙の旅」に出てくるコンピュータの名前 HAL は IBM の 3 文字のアルファベットを 1 文字ずらして (I→H, B→A, M→L) 作った名前として有名です。

1 1930 年代

- 2 8. ドイツ系ユダヤ人ゲーデル (Kurt Gödel) が不完全性定理を証明 (1931)。ある種の数学的
3 体系では真とも偽とも証明できない命題が存在することを示しました。数学のみならず思想/
4 哲学にも衝撃を与えました。コンピュータ処理の限界を示したことにもなっています。
- 5 9. イギリス人チューリング (Alan Turing) が チューリング機械を発表 (1937)。これは最も有
6 名なコンピュータの数学理論です。
- 7 10. アルゴリズムの数学的基礎が固まる。語源は先に述べた通りですが、現代的な意味でのアル
8 ゴリズム (とプログラム) の定義がこの時代に確立されます。

9 3.2 ENIAC 以後

10 電子計算機 (現代的意味でのコンピュータ) の発明は情報科学/工学の発展を加速させていきま
11 す。ENIAC 以降、本格的なコンピュータ時代がやってきます。

12 1940 年代

- 13 11. 真空管式コンピュータ ENIAC の開発がアメリカで始まる (1943)。1946 年に完成。17,468
14 本の真空管で動作し、消費電力 150kW、幅 24m× 奥行き 0.9m× 高さ 2.5m、重量 30 トンと
15 いう規模でした。同様の開発プロジェクトとして、同時期に EDVAC、EDSAC、Mark I な
16 どのコンピュータが相次いで開発されます⁶。図 3.2 に当時の ENIAC の写真を載せます。
- 17 12. ドイツ系ユダヤ人フォン・ノイマン (von Neumann) が プログラム内蔵方式 (stored program
18 method) を提案した⁷ (1945)。この方式を採用した世界初のコンピュータはイギリスの
19 EDSAC (1949) とされています。以後、現在までほとんど全てのコンピュータはこの方式
20 に基づいています。これをノイマン型コンピュータとも呼びます。
- 21 13. オートマトン (自動機械) の理論の研究が始まる。これは情報科学の基礎理論のひとつです。

22 1950 年代

- 23 14. プログラミング言語 FORTRAN の開発が始まる (1954)。歴史上最初の 高水準プログラミ
24 ング言語 (high-level programming language) です。

⁶歴史上最初の電子計算機を ENIAC とする見方は広く受け入れられていますが、特定用途に限れば、1939 年に試作された ABC (Atanasoff-Berry Computer) とする考えもあります。

⁷実際の発案者はノイマンではなく、その辺、少しゴタゴタした事情があったらしい。興味のある人はインターネットで検索してみてください。いろいろ出てきます。

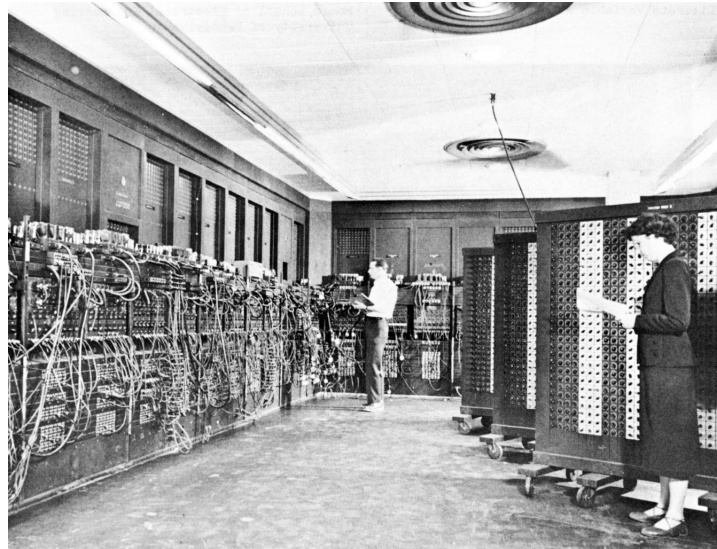


図 3.2: ENIAC コンピュータ (“U.S. Army Photo”, Wikipedia“ENIAC ” から転載)

- 1 15. 人工知能 (AI) という言葉が発明される (1955)。この当時の AI は現在のディープラーニン
2 グを用いる AI とは異なり、記号を用いた AI です。
- 3 16. 日本電信電話公社がパラメトロン計算機 武蔵野 I を開発する (1957)。これは世界史的な意
4 義は薄いのですが、当時の日本で独創的なコンピュータが開発された事例として有名です。
- 5 17. MIT のマッカーシー (MacCarthy) が関数型プログラミング言語 Lisp を開発 (1958)。こ
6 の言語は、人工知能などの記述言語として現在でも使用されています。
- 7 18. チェス・ゲームのプログラムが作られ始める。

8 1960 年代

- 9 19. 時分割処理システムが MIT で開発される (1961)。これによって、ひとつのコンピュータで
10 同時に複数の仕事を行うことが可能になります。
- 11 20. アメリカで ASCII コード (ASCII code) が制定される (1963)。英文字、数字、基本記号の
12 コード体系として今でも利用されています。
- 13 21. IBM 社システム/360 シリーズを発表 (1964)。汎用大型コンピュータの歴史的な名機として有
14 名です。その後、UNIX が広まるまでのほとんど全ての汎用大型コンピュータのひな形とな
15 りました⁸。

⁸日本のコンピュータメーカーは競って IBM 互換機を開発しましたが、それが IBM 産業スパイ事件 (1982) につながっています。



図 3.3: PDP-11 (Wikipedia“PDP-11 ” から転載)

22. チューリング賞が創設される (1966)。残念ながら、未だ日本人の受賞者はいません (数名の候補者は挙がりましたが)。

23. アメリカ国防総省の推進で大学間ネットワーク ARPANET が構築される (1967)。これがインターネットの第一歩になりました。

1970 年代

24. DEC 社がミニコンピュータ PDP-11 を発表 (1970)。小型コンピュータ (といっても一部屋分くらいの大きさ) の名機として有名です (図 3.3 参照)。

25. Intel 社が 4bit マイクロプロセッサチップ 4004 を開発 (1971)。後の PC 時代はここからスタートすると言っても過言ではないでしょう。この開発に日本人⁹が関わっていたことは有名です。

26. プログラミング言語 C が開発される (1972)。その後、UNIX の記述言語として世の中に浸透していきます。

27. 通信規約 (プロトコル) TCP/IP が設計される (1974)。現在のインターネットで使用されているプロトコルです。

28. アメリカ人ビル・ゲイツ (Bill Gates) が PC 用言語 BASIC を開発し、マイクロソフト社を設立 (1975)。ゲイツはこのマイクロソフト社の経営で世界一の富豪になりますが、その秘訣は経営手法 (製品を販売会社などに売る際に、開発した製品の全ての権利を売却するのではなく、使用権のみを売却するという手法) がそれまでのソフトウェア開発会社と異なっていたためです。

⁹嶋正利氏 (ビジコン社、当時)。

- 1 29. ローカルエリアネットワーク Ethernet が設計される (1975)。現在のインターネット技術
2 (特にハードウェア面で) の出発点と考えてよいでしょう。
- 3 30. クレイ (Cray) 社がスーパーコンピュータ Cray1 を発表 (1975)。ベクトル・コンピュータ
4 と呼ばれる第一世代のスーパーコンピュータです。
- 5 31. 未解決問題だった地図の四色塗り分け問題がコンピュータを用いて肯定的に証明される (1976)。
6 コンピュータを数学の重要な証明に用いた最初の例として有名です。
- 7 32. バークレー版 UNIX が開発される (1976)。UNIX OS が世界に広がるきっかけとなりました。
- 8 33. 東芝が日本語ワードプロセッサを開発 (1978)。コンピュータの入力装置はいわゆるタイプラ
9 イタ (現在のキーボード) を基盤とするのですが、その意味で日本語の扱いには障壁となっ
10 ていました。それを突破する技術として日本のコンピュータ史上特筆すべき発明です。
- 11 34. インテル社が 16bit プロセッサ 8086 を発表 (1978)。現在ほとんど全ての PC に組み込まれ
12 ているプロセッサのひな形です。
- 13 35. X 線 CT 装置の開発に対してノーベル賞が贈られる (1979)。これは、コンピュータを用い
14 た機器開発に対する最初のノーベル賞です。
- 15 36. 構造化プログラミング (structured programming) が提唱される。これは現在最も普及して
16 いるプログラミング技術の基本です。
- 17 37. ビデオゲームが開発、販売され始める。コンピュータゲームは以前から存在しましたが、い
18 ずれも文字端末を用いたものでした。しかし 70 年代の後半から少しずつ画像出力装置が安
19 価になり、その結果、ビデオゲームが商業ベースに乗ってきます。
- 20 38. 公開鍵暗号法の研究が進む。通信に暗号はつきものですが、その理論の中でも最も有名かつ
21 最もよく利用されている方法です。

22 1980 年代

- 23 39. マイクロソフト社が MS-DOS を開発 (1981)。
- 24 40. Sun 社 (現・サン・マイクロシステムズ) がワークステーションを発表 (1981)。このコン
25 ピュータ以降「ワークステーション (WS)」という言葉が高性能デスクトップ型コンピュ
26 タの名称として定着しますが、最近では WS と PC の境界があいまいになり、死語になりつ
27 つあります。

- 1 41. IBM 社が IBM PC を発表 (1981)。現在のパーソナルコンピュータ (PC) の原型です。そ
2 れまで IBM 社は汎用大型コンピュータの開発、販売で世界の頂点に君臨していましたが、自
3 社の開発したこの PC によって皮肉にもハードウェア・メーカーとしての頂点の地位を手放
4 すことになります。
- 5 42. 日本電気 (NEC) が PC9801 を発売 (1982)。日本のパーソナルコンピュータでは IBM PC
6 ではなく、PC98 シリーズがその後十年以上に渡って最も大きなシェアを持ち、特に 1987 年
7 には店頭販売の 90% のシェアでした (Wikipedia より)。
- 8 43. プログラミング言語 C++ を設計 (1983)。オブジェクト指向言語でありながら、伝統的なプ
9 ログラミングスタイルを踏襲しており、広く受け入れられています。
- 10 44. 任天堂ファミリーコンピュータを発売 (1983)。任天堂がトランプなどの卓上娯楽品販売の会
11 社から大転換、飛躍するきっかけとなりました。これまでに約 6000 万台が出荷されました。
- 12 45. アップル (Apple) 社が Macintosh シリーズを発表 (1984)。
- 13 46. 円周率の計算で 10 億桁を達成 (1989)。10 億桁の数値そのものは重要ではなく¹⁰、コンピュ
14 タの計算能力の象徴として重要です。
- 15 47. 論理型プログラミング言語 Prolog が普及。述語論理に基づくプログラミング言語として特
16 に日本で盛んに研究されます。AI 研究の第二の波の象徴とされています。

17 1990 年代

- 18 47. UNIX ライクなオペレーティングシステムのカーネル (中核プログラム) Linux¹¹ の発表
19 (1991)。無償 (フリー) かつクリーン (ソースプログラムが公開されていてプログラムの内
20 容が第三者にも把握できる) な基幹ソフトウェアとして現在、広く使用されています。
- 21 48. World Wide Web (WWW) プロジェクトが発表される (1991)。当時普及しつつあったイン
22 ターネット上の応用手法の主流となり、インターネットを一般家庭まで普及させる原動力と
23 になりました。インターネットは当初は研究者間のデータのやり取りなどに使用されていまし
24 ましたが、1990 年半ばの商用化によって個人の趣味的な homepage などが作られるようになり、
25 現在の状況を迎えます。インターネットの普及・発展は、人類史上の最重要事項のひとつに
26 なると思われます。
- 27 49. ソニー・コンピュータエンタテインメントがプレイステーションを発売 (1994)。3 次元コン
28 ピュータグラフィックスをハードウェアでサポートした最初のゲーム機です。

¹⁰SF 小説の中には、円周率の膨大な数字列の中に宇宙の真理が秘匿されている、というモチーフを持つものがありますが、はたして...

¹¹正式な発音は規定されていませんが、多くの人がリナックスと発音するようです。

- 1 50. マイクロソフトが Windows 95 を開発、発売（1993）。Windows 自体は'80 年代から開発、
2 販売されていましたが、CPU のパワー不足からユーザを満足させるものではありませんでし
3 た¹²。しかし'90 年代に入ってやっと CPU のパワーが追いつき、急速に広まっています。
4 特に機種非依存¹³を実現した Windows95 からはほとんど全ての PC マシンに実装されるよ
5 うなり、パーソナルコンピュータの標準 OS の地位を得ています。
- 6 51. Amazon 社の設立（1993）。
- 7 52. サン・マイクロシステムズがプログラミング言語 Java をリリース（1996）。
- 8 53. IBM のチェス専用スーパーコンピュータ、ディープ・ブルー（Deep Blue）がチェス世界チャ
9 ンピオンガルリ・カスパロフに勝利（1997）。その後、将棋、囲碁においてもコンピュータ
10 の目覚ましい性能向上が見られています。
- 11 54. Google 社の設立（1998）。当初、インターネット上の検索サービスを主軸として展開し、そ
12 の後、様々なオンラインサービスを行う巨大企業となっている。

13 2000 年代以降

- 14 53. 超並列コンピュータの開発。Cray1 のような一台で高速演算を行う方式は 90 年代初めに性
15 能向上が鈍化し、代わって安価なコンピュータ数千台を同時に実行させて高速演算を行う並
16 列方式へ移行しています。特に最近ではインターネットを使った並列方式の研究が盛んです
17 （2 章 GRID コンピュータを参照）。
- 18 54. ノート PC の普及。1980 年代末に商用化されたノート PC は携帯性の良さから急速に普及し
19 ました。部品の小型化、低電力化、バッテリー性能の向上、液晶技術の進歩などで家庭用、個人
20 用にはノート PC の性能で事足りる状況です。ちなみに 2015 年 1 年間の世界全体での PC の
21 出荷台数は 3 億 1,300 万台であり、その内にノート PC の占める割合は約 68%に当たります。
- 22 55. オブジェクト指向（object-oriented）の実用化。1980 年から始まったオブジェクト指向プロ
23 グラミングは 2000 年代に入り、実用期に入ります。
- 24 56. 家庭用ゲーム機の普及。表 3.1 に過去の代表的なゲーム機の発売開始年と累計販売数をまと
25 めました。ゲーム機は同一製品が大量に売れるため、開発に膨大なコストを掛けて安価に販
26 売してもビジネスとして成り立つビジネスモデルになっています。

¹²マウスをクリックしてからウインドウが開いていく様子がスローモーションのように見えた、というくらいに CPU のパワー不足は深刻でした。

¹³かつては同じ PC という規格のコンピュータであっても、A 社製では動くが、B 社製では動かないという状況が普通でした。

表 3.1: 家庭用ゲーム機の累計販売台数（または出荷台数）

発売開始年	ゲーム機名	販売台数（万）
1983	ファミリーコンピュータ	6,200
1989	ゲームボーイ	12,000
1990	スーパーファミコン	5,000
1994	PlayStation	10,200
	セガサターン	920
1996	Nintendo64	3,300
1998	ドリームキャスト	910
1999	PlayStation 2	15,500
2002	Xbox	2,400
2004	ニンテンドー DS	15,400
2005	Xbox360	8,000
	PlayStation 3	8,000
2006	Wii	10,100
2011	ニンテンドー 3DS	8,000
2013	PlayStation 4	11,700
2017	ニンテンドー Switch	11,100
2020	PlayStation 5	1,930

情報は主に Wikipedia による

57. コンピュータウイルスの発生。ネットワークや記録メディアの進歩と共に、コンピュータシステムに障害を起こさせるソフトウェア（ウイルス）が大量発生し、ときとして社会全体に深刻なダメージを与えるようになってきました。
58. Apple 社が iPhone を発売する（2007）。スマートフォンの歴史は実質的にここから始まります。スマートフォンは携帯電話の発展形と見なされますが、実際にはスマートフォンはコンピュータとしての機能が主であり、膨大な種類のアプリケーション（アプリ）が提供されています。それによって私たちの生活様式は大きく変化しています。ちなみに iPhone シリーズは 2018 年末までに 22 億台以上（Wikipedia：iPhone による）を出荷しています。Android 端末などを含めた全ての種類のスマートフォンの総出荷台数は 120 億台以上です。
59. クラウド処理の普及。一例として、Amazon の AWS（Amazon Web Services）が 2006 年に開始され、様々なウェブサイトやアプリケーションに対してオンラインでサービスを提供しています。AWS のシェアは全世界の 1/3 に相当する巨大なものになっています。非公式ながら Apple 社の iCloud も一部、AWS を利用していると言われています。

第4章 コンピュータの基本構成

前章まではコンピュータの分類、歴史を述べてきました。コンピュータを外から眺めることはここまでとして、この章からはコンピュータの動く仕組みを内に入って詳細に見ていきます。まずこの章ではコンピュータの基本構造を概説します。

4.1 基本構成要素

どんなコンピュータも、演算装置（CPU、プロセッサとも呼びます）、記憶装置、入力装置、出力装置の四つの基本構成要素から成っています（図 4.1 参照）。

通常、入力装置で取得したデータは演算装置へ転送され、それが記憶装置へ書き込まれます。入力装置から直接、記憶装置へデータが転送されることは通常は行われません¹。記憶装置と演算装置の間では何度もデータをやりとりし、データの加工を繰り返します。そして加工されたデータは出力装置へ転送されます。コンピュータの利用者は、出力装置への出力結果を見て、次の入力を行います。その入力から次の出力が作り出され、... というサイクルを通して、利用者はコンピュータを操作していきます。

上の四つの装置のどのひとつが欠けて、それをコンピュータとは呼びません。以下の節ではそれぞれの構成要素について述べます。

4.2 入力装置

コンピュータに限らず、何らかの仕事をする機械は、外界の情報を内部へ取り込む入力部を持っていなければなりません。コンピュータはその入力に応じて自身の内部状態を更新して行きます。コンピュータ利用者の立場から見れば、利用者は入力装置を通してコンピュータを操作します。

キーボード、マウスは現在の標準的な入力装置です。最近ではタッチパネルが普及しつつあります。それ以外にもトラックボール、イメージスキャナなども入力装置です。ゲーム機のコントローラ・パッドも立派な入力装置です。ネットワーク接続用ソケットや USB ソケットは一般には入力装置とは呼びませんが、広い意味では入力装置になり得ます。キーボードやマウスを持たず、単に

¹演算装置を介さないデータ転送処理を DMA 転送（Direct Memory Access）などと呼び、高速処理が要求される箇所で使用されます。

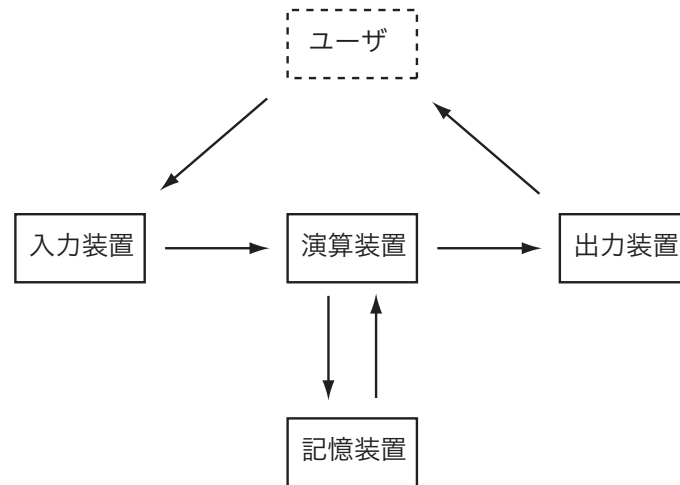


図 4.1: コンピュータの基本構成要素

- 1 ネットワークに接続されただけのコンピュータもありえます。そのような場合、ネットワーク経由
- 2 で別のコンピュータのキーボードやマウス情報を取り込むことも可能です。
- 3 今や全く利用されませんが、20 年以上前には紙テープや厚紙カードが入力に利用されていまし
- 4 た²。さらに古い ENIAC の時代には入力はパネル上の数百個のスイッチや、ケーブル・ジャックの
- 5 接続によって行っていました。

6 4.3 記憶装置

- 7 入力された情報や演算装置による計算結果を一時的/長期的に蓄える装置は全て記憶装置です。
- 8 単に記憶装置あるいはメモリ（memory）と言えば、一般には演算装置から直接読み書き可能なメ
- 9 イン・メモリ（主記憶装置）を指しますが、広い意味ではハードディスクなども記憶装置です。
- 10 最近のコンピュータでは、記憶装置は演算装置からの距離によって階層化されています。演算装
- 11 置に最も近い層にはキャッシュ・メモリ³があり、次にメイン・メモリがあります。ここまでを通
- 12 常、内部記憶装置と呼びます。内部記憶装置の外には外部記憶装置があり、具体的にはハードディ
- 13 スクなどの周辺装置、さらにネットワークを介してアクセス可能なネットワーク共有型ハードディ
- 14 スク、さらに遠くにはインターネットを介してアクセス可能な記憶装置等が考えられます。
- 15 記憶装置には二つの役割があります。ひとつは高速にデータを読み書きすることです。もうひと
- 16 つは大量のデータを記憶することです。通常、演算装置に近いほど高速、小容量、高価であり、演
- 17 算装置から離れるに従い、低速、大容量、安価です。演算装置の近くでは少量の同じデータが何度
- 18 も演算装置によってアクセスされる傾向にありますから、高速性が要求されます。これに対して、

²昭和 30 年代、40 年代の SF 映画などで見るができます。

³メインメモリの低速性を補うために、演算装置とメインメモリの間に存在し、最も重要なデータのみを格納する高速かつ小容量のメモリのことをキャッシュメモリと呼びます。

- 1 演算装置から遠いデータは頻繁にアクセスされることは稀（まれ）です⁴から、高速アクセスより
2 も大容量であることが要求されます。

3 4.4 演算装置

4 CPU（Central Processing Unit）あるいは単にプロセッサとも言います。

5 コンピュータにおける代表的な演算は数値計算です。しかしコンピュータは単に計算を行う機械
6 ではなく、様々な論理演算も可能です。またそのような演算をある規則に従って順次自動的に実行
7 していくことも可能です。

8 現在、世の中で最も広く利用されている商用プロセッサは、Intel 社の x86 系プロセッサです。
9 x86 系とは、Intel 社が 1978 年に開発した 16bit プロセッサ 8086 の後継プロセッサの総称です。も
10 しろん、8086 は現在は発売/使用されておらず、Xeon や Core Duo、Core i-3/5/7 などの x86 系
11 を拡張したものが使用されています。

12 その他にもかつてはサン・マイクロシステムズ社の UltraSPARC プロセッサ、IBM 社の POWER
13 プロセッサ、PlayStation 3 に搭載された Cell プロセッサ⁵などが商用マシンに搭載されました。最
14 近では、Apple 社の M1、M2 プロセッサが Mac、iPad に搭載されています。

15 また、最近では 1 台のコンピュータに複数のプロセッサを搭載するマルチコア・プロセッサが一
16 般的です⁶。これについては後節で少し丁寧に解説します。

17 4.5 出力装置

18 出力装置は、コンピュータの実行結果を外部へ通知するために必須です。

19 最も代表的な出力装置は、液晶ディスプレイ装置（liquid crystal display、LCD）です。以前は
20 CRT（Cathode-Ray Tube）ディスプレイ装置 — いわゆるブラウン管ディスプレイ装置 — が主
21 流でした。30 年以上前の CRT ディスプレイ装置では横 80 文字、縦 24 文字の英数字と簡単な記
22 号しか表示できませんでした。しかしメモリの価格が下がるに従い、（本来は計算結果の記憶など
23 に用いる）メモリの一部分を画面情報の記憶⁷に充てる余裕が生じ、その結果ディスプレイは単な
24 る文字を表示するだけではなく、画像を表示できるようになりました。1980 年代後半からは白黒
25 の高精細 CRT ディスプレイ装置が商用化され、さらに 1990 年代半ばからは現在のようなフルカ
26 ラー高精細ディスプレイ装置が一般的に利用されるようになりました。しかし、最近では CRT は
27 姿を消し、代わりにカラー液晶パネルが用いられています。

⁴通常コンピュータでは、頻繁にアクセスされるデータは演算装置の近いところへ（半）自動的に移動されるような設計がなされます。

⁵ソニー・コンピュータエンタテインメント、ソニー、IBM、東芝の共同開発です。

⁶プロセッサ内の特に演算部、実行制御部のことをプロセッサ・コア（processor core）と呼びます。マルチコアとは、プロセッサコアが複数個（マルチ）搭載されていることを意味します。

⁷ディスプレイ装置の情報の記憶に特化したメモリを特にビデオ・メモリと呼びます。

1 プリンタも重要な出力装置です。かつて1970年代にはCRTディスプレイ装置すら存在せず、英
2 数字のみを印字可能なプリンタがほとんど唯一の出力装置でした。1985年にApple社がPC用レー
3 ザープリンタを発売します。また1990年代に入るとレーザープリンタよりも廉価なインクジェッ
4 ト方式のプリンタが実用化されます。

5 その他の出力装置として、ネットワーク接続用ソケットやUSBソケットも出力部と考えてよい
6 でしょう。

7 やや横道にそれますが、最近では生体とコンピュータをつなぐ研究が盛んです。人工視力や人工
8 聴力の研究は20年以上前から行われています。また表面筋電位を使って人体の動作をサポートす
9 る装置も実用化に近づいています。人間の脳波をコンピュータに取り込み、キーボードを介せず、
10 直接、頭で考えたことをコンピュータに入力する研究も始まっています。これらの装置は、身体に
11 障害者を持つ人をコンピュータでサポートする上で非常に重要です。また、生体とコンピュータを
12 直接つなぐ装置⁸として、今後の社会の在り方に大きな影響を与えるかもしれません。

13 4.6 記憶装置のないコンピュータ？

14 入力、出力、記憶、演算装置が全て備わっているものがコンピュータです。その中のひとつが欠
15 けたならば、それは何なのでしょう。

16 まず入力装置、出力装置のないものは、オペレータ（人間）との間で情報が一方通行になってし
17 まいますから、これはコンピュータとは言えません。演算装置の欠けたものは、単なるメモ帳で
18 す。情報を記録することはできても、演算装置を持たないため、その情報を加工することができま
19 せん。

20 記憶装置の欠けたものは世の中で広く利用されています⁹。たとえばデジタル音声信号をアナロ
21 グ音声信号に変換する装置はオーディオ装置（CDプレイヤーなど）に普通に組み込まれています。
22 この信号処理装置は、音声信号の入出力装置を備え、かつ信号変換のための演算装置を備えていま
23 すが、通常コンピュータのような記憶装置は付いていません。同様にラジオは電波を入力し、使用
24 者はラジオのスイッチ類を操作して、チャンネルや音量の変更を入力できます。そして電波信号は
25 ラジオ内部の演算装置によって実際の音声に変換され、出力されます。

26 このように、多くの電気製品は入出力装置および演算装置を備えています。しかし記憶装置を持
27 たないため、それらをコンピュータとは呼びません。記憶装置を欠いた装置（あるいは機械、シス
28 テム）の大きな特徴は、定められた単一の処理しかできないことです。信号変換装置はそれ以外の
29 処理には利用できません。ラジオはラジオ以外の何物でもありません。

⁸映画「マトリックス」で主人公らが後頭部にコンピュータの端子を差し込んでマトリックスの仮想世界に接続していま
した。あのような接続装置は絵空事ではありません。

⁹ただし記憶装置を全く持たない訳ではなく、ごく少量の記憶は許すこととします。

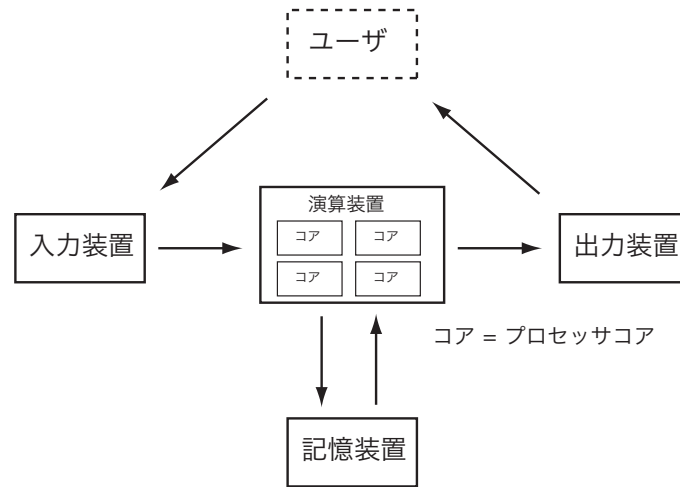


図 4.2: マルチコアプロセッサを搭載するコンピュータの構成

1 逆の視点から言えば、たとえばビデオゲーム機はコンピュータです。何故ならば、ゲーム機では
 2 特定のゲームしかできない訳ではなく、ゲームのソフトを変更することで、様々なゲームが可能
 3 だからです。コンピュータは記憶装置に設定されたプログラムによって変幻自在に処理内容を変
 4 えることができます。このことがコンピュータをコンピュータたらしめており、この性質を万能性
 5 (universality) と呼びます。コンピュータはプログラム（ソフトウェア）によって様々な処理を実
 6 行できる万能機械¹⁰なのです。逆に言えば、現在、記憶装置を持たない電気製品であっても、記憶
 7 装置を付加すればある種の万能性を持つかもしれません。たとえば最近の HD 付き DVD レコー
 8 ダーの中身はコンピュータそのものです。

9 4.7 並列コンピュータ

10 コンピュータの基本構成は図 4.1 の通りですが、最近のコンピュータはそれほど単純ではありま
 11 せん。コンピュータの使命のひとつである高速演算を行うために、半導体メーカー、コンピュータ
 12 メーカーは 20 世紀中は主にひとつの CPU の性能を上げることを目指してきました。しかし半導
 13 体製造技術の理論的限界に近づき、今世紀に入って多くの半導体メーカーは複数の演算装置で高
 14 速化を目指す方向へ舵を切りました。結果、現在のコンピュータは様々な形で並列コンピュータに
 15 なっています。

16 第一に、「演算装置は CPU である」と 4.4 節で述べましたが、最近の CPU は — それが比較的
 17 安価なノート PC の CPU であっても — 複数の演算回路で構成されています。それを図示したも
 18 のが図 4.2 です。ここに「演算回路」は演算処理の中核部分を意味し、そのことからこれを一般に

¹⁰もちろんここでいう「万能」は「神様は万能 (almighty) である」と言うときのそれとは異なり、コンピュータが何でもできる訳ではなく、理論的、物理的な限界があります。

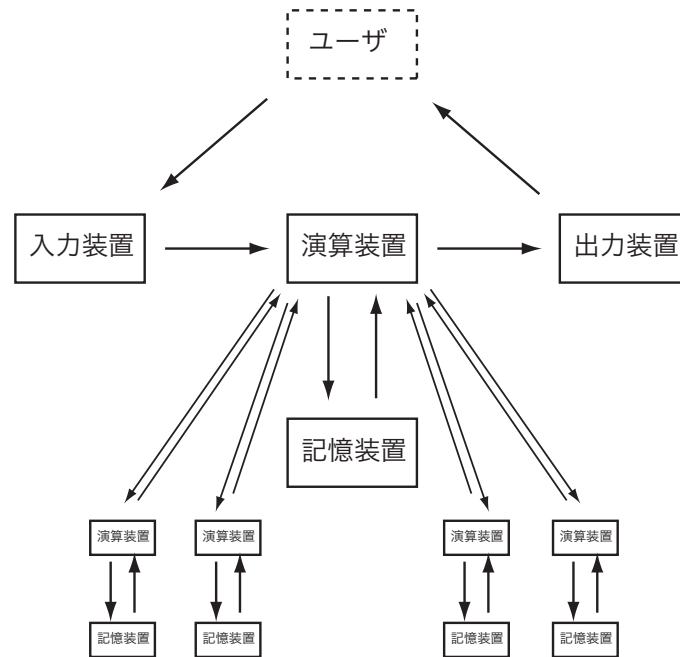


図 4.3: 分散メモリ型並列コンピュータの構成

1 プロセッサコア（または単にコア）と呼んでいます。ひとつの CPU には複数のプロセッサコアが
 2 搭載され、それらプロセッサコアは演算装置が入力装置、出力装置、記憶装置とのやりとりを行
 3 う周辺回路などを共有しています。複数のコアを持つことからこのような CPU をマルチコアプロ
 4 セッサと呼びます。プロセッサコアは同時並列的に動作できるため、もし 4 個のコアがあるなら
 5 ば、最大 4 倍の処理性能を持つことになります。最近の PC 用 CPU では 2 コア～16 コアのものが
 6 一般に販売されています。

7 マルチコアプロセッサではひとつの LSI 集積回路に複数の演算回路（プロセッサコア）を搭載
 8 しますが、さらなる高速化のためにマルチコアプロセッサを並列動作させるコンピュータもありま
 9 す。たとえば 2 節で紹介したスーパーコンピュータ「富嶽」では 52 コアを搭載したマルチコアプ
 10 ロセッサ 158,976 個を同時並列で動作させて高速計算を行います。コアの総数は 8 百万を超えま
 11 す。それを図示したものが図 4.3 です。演算装置（マルチコアプロセッサ）と記憶装置のペアが多
 12 数あり、演算装置間が高速のコンピュータネットワークで結合された形です。

13 図 4.3 のコンピュータでは、各演算装置がそれぞれ記憶装置を持っています。その構成の問題点
 14 は、演算装置 P_A は演算装置 P_B に付随する記憶装置の内容 M_B にアクセスできない点です。 P_A
 15 が M_B にアクセスするには P_B 経由で M_B の内容を P_A に転送する必要があります。が、そのよ
 16 うな転送が増えるとシステム全体として計算効率が落ちてしまいます。そこで考えられるのが、図
 17 4.4 のような構成の並列コンピュータです。この場合、記憶装置は共有されているため、いちいち
 18 データを転送する必要がありません。

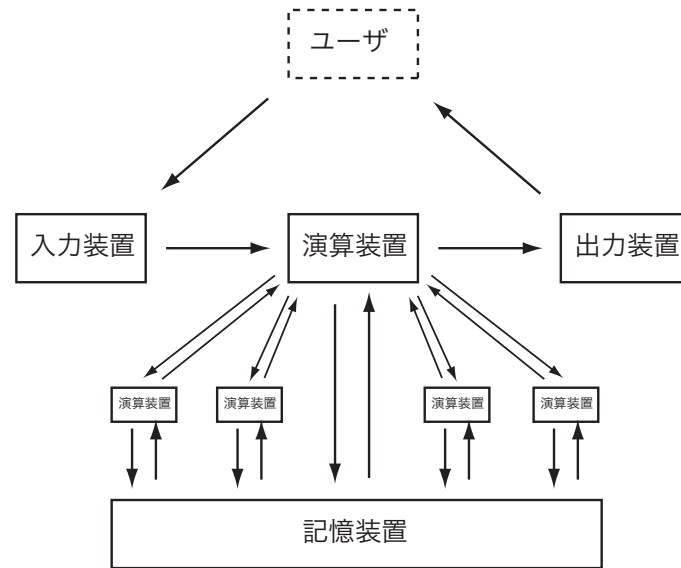


図 4.4: 共有メモリ型並列コンピュータの構成

- 1 以上、この節では図 4.1 の基本構成とは異なる 3 種類の並列コンピュータの構成を紹介しました。
- 2 これらについて詳細を知る必要はありません。が、現実のコンピュータシステムは基本構成とはかなり異なったものになっていることを知っておく必要があります。
- 3

1 第5章 p5.js プログラミング（基本編）

2 コンピュータを手っ取り早く知るには、プログラミングを学ぶのが一番です。プログラミングを
3 知ることなくコンピュータについてアレコレ論じても、結局、コンピュータの本質には辿り着けま
4 せん。幸いなことに、今では簡単なプログラミングで様々なことができるようなソフトウェアツ
5 ルが整備されています。ここでは、グラフィックス用プログラミング言語として人気がある p5.js
6 を用います。

7 まずこの章では p5.js について基礎的な内容を紹介します。そして次の章では p5.js の得意分野
8 であるグラフィックスのプログラミングを紹介します。その次の章では、平方根、円周率を題材に、
9 計算的な処理の方法を紹介します。

10 5.1 プログラミング言語 p5.js

11 p5.js はプログラミング言語 Processing をプログラミング言語 JavaScript の上に移植したもの
12 です。

13 その Processing は、MIT (Massachusetts Institute of Technology、マサチューセッツ工科大
14 学) メディアラボがプログラミング言語 Java の上に開発したものです。コンピュータグラフィッ
15 クス (CG) を含む電子的ビジュアルアートを簡単にプログラミングできることも目的に開発され
16 ており、実際、他のプログラミング言語、環境を用いた場合よりも簡単に CG プログラムを記述で
17 きると評価されています。

18 JavaScript は、Web ブラウザの上で動作するプログラミング言語です。Web の利用の拡大と共
19 に発展し続けており、様々な機能を有するプログラミング言語として成長しています。

20 **注意：** JavaScript と Java は名前が似ていますが、その中身はほとんど別物です。これについて
21 はネット上で多くの混乱があることが報告されています。あるサイト（日本語さえわかればわか
22 る！Java と JavaScript の違い：

23 <https://eng-entrance.com/java-javascript>

24 ）に「インドとインドネシアくらい違う」と書かれており、まさにそれが両者の関係です。



図 5.1: p5.js ホームページのトップページ

1 参考： プログラミングに不慣れな人には以下の資料も役に立つかもしれません。

2 文系大学生のための p5.js 入門

3 <https://zenn.dev/ojk/books/intro-to-p5js>

4 5.2 動かしてみる

5 p5.js のホームページは、「p5.js」でネット検索すれば上位に現れます。以下が、大元の英語版の
6 URL です。

7 <https://p5js.org>

8 日本語版も用意されており、その URL は以下です。

9 <https://p5js.jp>

10 このページの様子は図 5.1 の通りです。

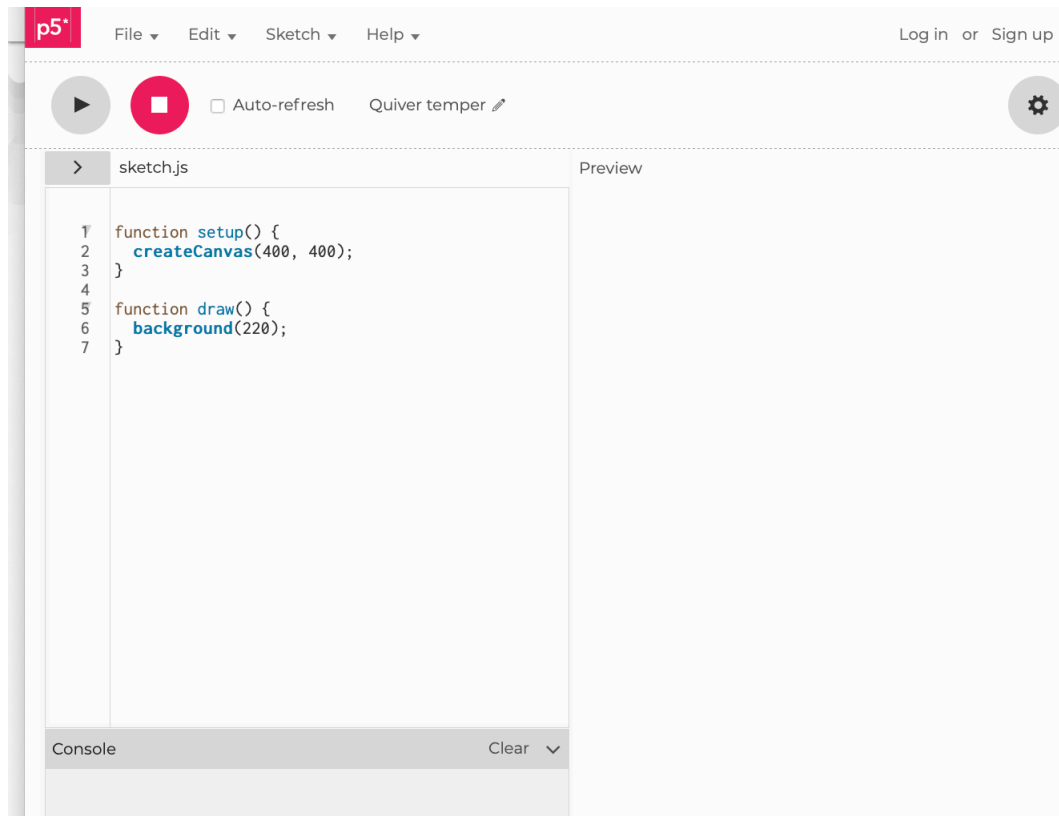


図 5.2: Web エディタ

1 p5.js のプログラムを試しに動かしてみるために、**みなさんの PC に p5.js の言語処理系をイン**
2 **ストールする必要はありません。**図 5.1 の下の方、赤枠の部分「新しい p5.js Web エディタで今す

3 ぐ試してみてください」をクリックしましょう。そうすると、新たに図 5.2 のようなウィンドウが

4 開きます。

5 **注意：** Web ブラウザの中には p5.js のエディタがうまく動作しない場合があります。その場合の

6 最も簡単な解決法は「別のブラウザを使ってみる」という安直な方法です。講義担当者の経験では、

7 MacOS/Safari でエディタが不調になることがしばしばあり、その場合、Firefox を使っています。

8 図 5.2 の左側半分がエディタ、右側半分が実行画面です。2022 年 9 月時点では、エディタにはす

9 でに以下のプログラム：

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);
}
```

1 が載せられています。

2 このプログラムは、関数 `setup()` でキャンバス（CG 画像を描くオブジェクト）を初期化し、関
3 数 `draw()` で CG 画像を描画します。なお、ここでいう「関数」とは処理のまとまりを表します。
4 一般に関数と言えば関数値を求めるためのものですが、プログラミング言語の場合、値を求める
5 だけではなく、様々な処理をひとまとめにする単位として関数を用います。`setup()` は主にグラ
6 フィックス描画の準備を行うための関数の名前です。プログラム実行では最初に 1 回だけ実行され
7 るものと約束されています。`draw()` は CG アニメーションの各フレームの描画を行うための関数
8 の名前です。`setup()` が実行された後、毎秒数十回¹ 定期的に実行されるものと約束されていま
9 す。`draw()` で描画する画像を少しずつ変えていくと、CG アニメーションを作成可能です。それ
10 については後章で解説します。

11 さて、上のプログラムでは、横 400 画素 × 縦 400 画素の大きさのキャンバスを作り（`createCanvas(400,`
12 `400)`）、そのキャンバスの全画素を明るさ 220 の灰色で塗りつぶします（`background(220)`）。同
13 じ色で塗りつぶすだけのため、画面は全く変化しません。

14 さっそく動かしてみます。ページ左上の  のボタンを押します。そうすると、ページが図 5.3
15 のように変化するはずですが、ページ右側に 400 × 400 の画素の大きさの灰色の正方形が描画されて
16 います。画面は静止したままですが、上の述べたように、画面が変化しないアニメーションを実行
17 中です。実行を止めるには、ページ左上の  のボタンを押します。

18 せっかくの機会ですから、CG アニメーションの実行を実感できるプログラムを実行してみま
19 しょう。先ほどのプログラム中の関数 `draw()` を以下に変更します。

```
function draw() {
  background(frameCount%256);
}
```

20 ここに `frameCount` はアニメーションでの描画フレーム数（コマ数）を表すものです。通常、1 秒
21 間に数十フレームずつ描画が進んでいきます。プログラムでは、`frameCount` の 256 の剰余を計算
22 し（よって `frameCount%256` の値は、0, 1, 2, ..., 254, 255, 0, 1, ... と変化します）、それを正方
23 形の明るさとするプログラムです。上のプログラムを実行すると、正方形の明るさが黒 → 灰色 →
24

¹ 正確な数値は後章で解説します。

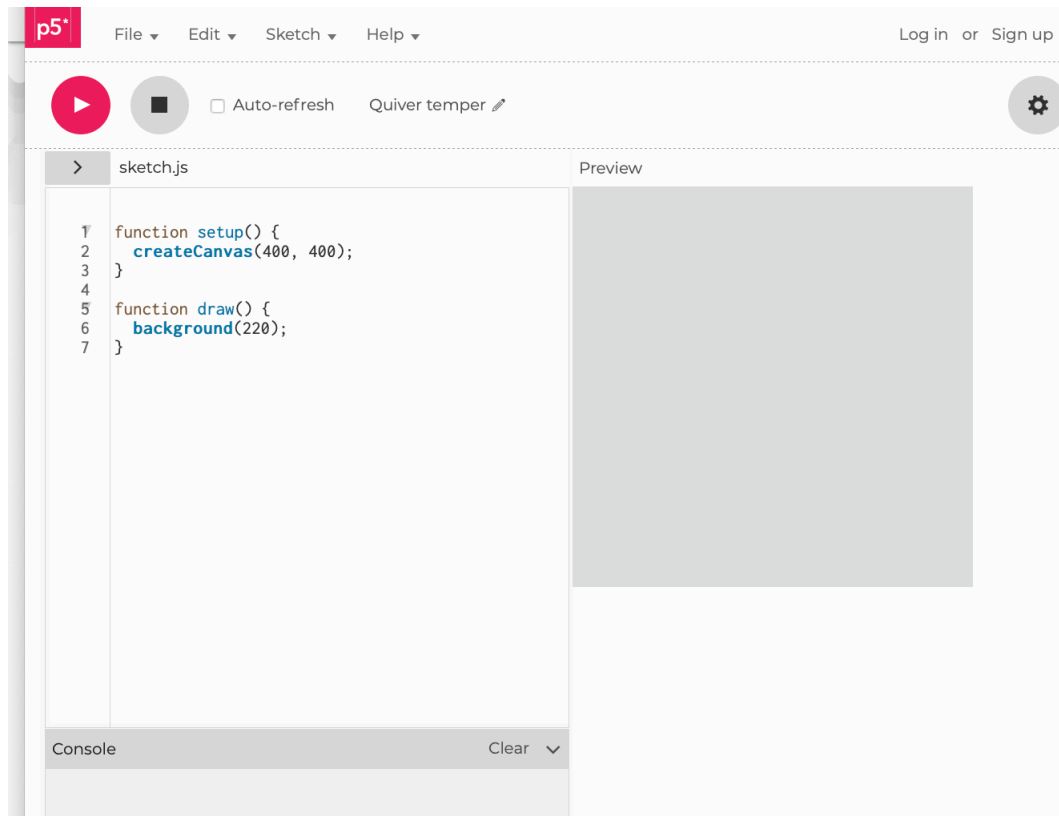


図 5.3: 実行中画面

- 1 白 → 黒 → ... と繰り返し変化していく様子（図 5.4 参照）が見えるはずです。
- 2 Web エディタのその他の機能を以下に概説します。まず、メニューについてです。なお、メニュー
- 3 のコマンドリストは状況によって変化することがあります。以下は、初期状態での説明です。
- 4 **File ...** プログラムファイルに関するコマンドメニューです。
- 5 **New ...** エディタで編集中のプログラムを初期化します。
- 6 **Save ...** プログラムを保存します。保存先は PC ではなく、p5.js のサイト・サーバです。よっ
- 7 て、事前に ID を作り、サインインしておく必要があります。
- 8 **Examples ...** 様々なサンプルプログラムをエディタにロードできます。
- 9 **Edit ...** 編集に関するコマンドメニューです。以下のコマンドはショートカットキーで起動すると
- 10 便利です。
- 11 **Tidy Code ...** プログラムを整形します。
- 12 **Find ...** 文字列検索を行います。

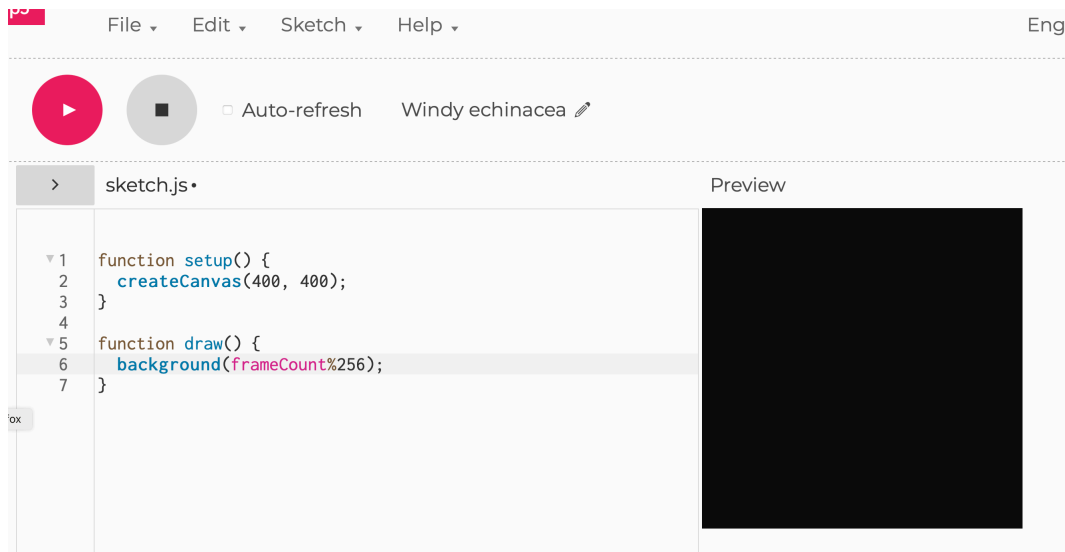
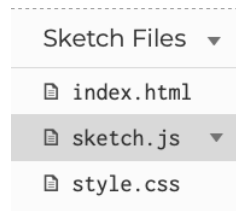


図 5.4: 実行中画面：ちょうど正方形が黒色になったときの画面

- 1 **Find Next ...** 次の文字列を検索します。
- 2 **Find Previous ...** 直前の文字列を検索します。
- 3 **Sketch** p5.js では、プログラムのことを**スケッチ**（sketch）と呼びます。ここで図 5.2 のエディタ
- 4 ウィンドウの  の直下の  をクリックしましょう。そうすると、そこに



- 6 というファイル名の並びが表示されます。実は p5.js のプログラム（スケッチ）はこの三
- 7 つのファイルから成っています。sketch.js がプログラムの本体です。サフィックス「.js」が
- 8 示すように、これは実は JavaScript のプログラムです。index.html はプログラムが動く環境
- 9 を設定している html ファイルです。Web ブラウザはこのファイルを読み込んで、このファ
- 10 イルに記載されている“sketch.js”を実行します。style.css はページの見た目を指定するスタ
- 11 イルシートです。

- 12 Sketch メニューの以下のコマンドは、プログラムの構築や実行に関するものです。

- 13 **Add File** プログラム実行環境（三つのファイル群）に新たにファイルを追加します。

- 14 **Add Folder** 同上、フォルダーを追加します。

- 1 **Run** プログラムを起動します。  と同じ機能です。
- 2 **Stop** プログラムの実行を終了します。  と同じ機能です。
- 3 **Help** 各種のサポートコマンドです。
- 4 **Keyboard Shortcuts** このエディタで利用できるキーボードショートカットの表示と編集
- 5 が可能です。
- 6 **Reference** p5.js のホームページを新しいウインドウで表示します。
- 7 **About** p5.js の関連ページのリストを表示します。
- 8 これらのコマンドの使い方、その他のコマンドについてはテキスト中で適宜、取り扱うこととし
- 9 ます。

10 5.3 簡単な p5.js プログラムの実行（文字列や数値の出力）

- 11 新たにプログラミング言語を学ぶとき、最初に学ぶべき事柄は値（文字列や数値）の出力方法で
- 12 す。プログラムの実行結果の出力が分からなければ、プログラムが正しく実行できたか否か判断で
- 13 きないからです。p5.js は CG 用の言語ですが、計算はもちろん必要です。そこで、これ以降、当
- 14 分の間はグラフィックス処理は封印し、プログラミングの基本的な事項を学んでいきます。
- 15 まず、p5.js で文字列 "Hello, World" をコンソールエリアに出力するには以下の囲みのように
- 16 記述します。

```
function setup() {           // プログラムの初期設定関数
  print("Hello, World");     // 文字列の出力
}
```

- 17
- 18 このプログラムの実行結果は図 5.5 です。p5.js のエディタの下に Console（＝コンソール）エリア
- 19 があります。print() の出力はここに表示されると約束されています。
- 20 ここに

```
21 function setup() {           // プログラムの初期設定関数
```

- 22 はプログラムの初期設定を行うための手続きの開始部分です。この手続きのことをプログラミング
- 23 言語の分野では一般に**関数**（function）と呼びます。関数と言えば、値を計算する数式を思い浮か
- 24 べるかもしれません。もちろん値を計算する関数も作ることができますが、値を計算するのではな
- 25 く、プログラミング言語ではコンピュータの操作を行い、値を計算しない手続きも広い意味で関数
- 26 と呼びます。

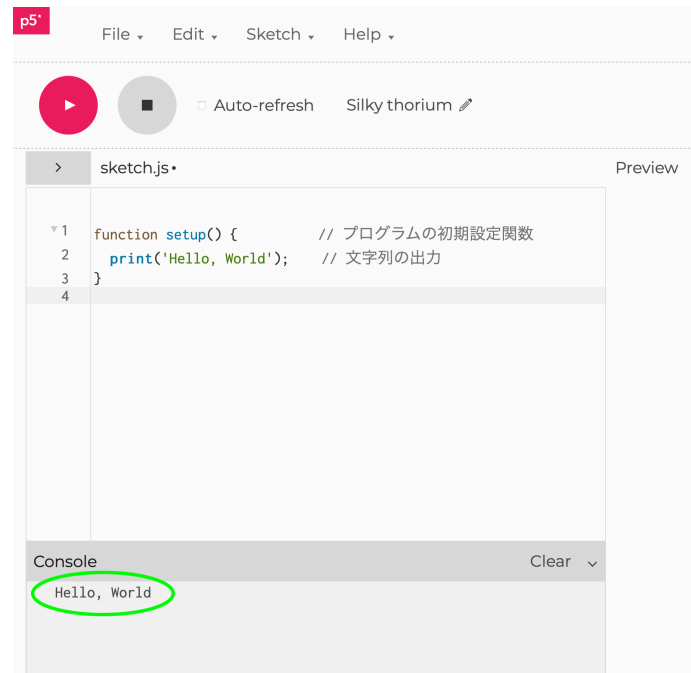


図 5.5: ハローワールドの表示：コンソールエリアの緑の枠の中に Hello, World が表示されている。

1 上の行の先頭部分の `function` は、これから関数を定義することを宣言するキーワードです。既
 2 に述べたように、`setup` は初期設定を行う関数のための名前であり、p5.js の中でその役割のため
 3 に用意されている名前です。このような名前を**予約語**（reserved word）といいます。全てのプロ
 4 グラミング言語にはそのような予約語がいくつかあり、その組み合わせで複雑なプログラムを組み
 5 上げていきます。

6 名名の後の括弧 `()` はそれが関数であることを示すための記号です。さらにその後の `{` とさらに
 7 後ろの `}` の間に手続き群を記述します。この `{ ... }` の部分を**ブロック**（block）と呼びます。ブ
 8 ロックはひとまとまりの処理の部分を表すものです。

9 ブロック内の

10 `print("Hello, World");` // 文字列の出力

11 は文字列 `Hello, World` の出力処理です。ここに、`print` は、p5.js に事前に用意されている、文字
 12 列や数値を出力するための関数です。この関数も値を求めるものではなく、表示処理を行います。
 13 `("Hello, World")` の部分は、関数 `print` のパラメータです。関数のパラメータのことをプログ
 14 ラミング言語の分野では**引数**（ひきすう、英語では argument）と呼びます。「いんすう」とは呼ば
 15 ないことに注意してください。これ以降、慣例に従い、「パラメータ」ではなく、「引数」という呼
 16 び方を用いることとします。

17 p5.js では、二重引用符 `" "` または一重引用符 `' '` で囲まれた部分は文字列定数を表します。

1 単に Hello, World と書いただけでは、文字列の始まりと終わりが分からないため、必ず引用符で
 2 囲みます。("Hello, World") は Hello, World という文字列が引数として関数 print に渡され、
 3 それがあるまま表示されます。関数を実行することを「関数を呼び出す (call)」、「関数呼び出し
 4 (function call) を行う」などと言います。関数呼び出し print("Hello, World"); の末尾にセミ
 5 コロン ';' があることに注意してください。セミコロンは処理の区切りを表します。

6 **注意 (p5.js、JavaScript のセミコロン問題)：** 実は、p5.js (とそのベース言語の JavaScript)
 7 ではプログラム中の処理を自動的に区別する機能があります。そのため、セミコロンを付けなくて
 8 もエラーは起きず、ほとんどの場合にプログラムはプログラマが意図した通りに正常に動作しま
 9 す。たとえば以下のプログラムは全く問題なく動作します。

```
function setup() {  
  print("Hello, World")  // 文末のセミコロンが無い  
}
```

10

11 ただし常に問題がない訳ではなく、詳細は述べませんが、プログラマが意図しない動作が起きるこ
 12 とがあります。それがかなり特殊な場合であるため、世の中には JavaScript のプログラムに「セ
 13 ミコロンを付けない」派と「付けておいた方が安心」派の両派に分かれています。この講義テキス
 14 トでは後者に組することとします。

15 次に、計算式 123+1 の値を出力したい場合は以下のプログラムを用います。

```
function setup() {  
  print(123+1);          // 加算の計算と結果の出力  
}
```

16

17 このプログラムを実行すると、加算が実行され、以下の計算結果がコンソールエリアに表示され
 18 ます。

19 124

20 連続する出力を行うにはたとえば次のような2行のプログラムを作ります。

```
function setup() {  
  print("Hello, World");  
  print(123+1);  
}
```

21

22 このプログラムを実行すると、以下の出力を得ます。

```
1   Hello, World
2   124
```

3 なお、関数 `print()` では、**その表示操作の終了時に自動的に改行する**と約束されています。よっ
4 て上のプログラムの出力結果は2行になります。

5 関数 `print()` には複数の引数をカンマ `,` で区切って与えることもできます。たとえば

```
6   function setup() {
      print("Hello, World", 123+1);
    }
```

7 このプログラムを実行すると、以下のように、1行に二つの出力を得ます。

```
8   Hello, World  124
```

9 以上、出力の仕方が分かったところで、様々なプログラムの作成へ進みます。

```
function setup() {
  a = second();
  b = (a+30)%60;
  print(b);
}
```

図 5.6: 30 秒後の時刻（秒）を表示するプログラム

1 5.4 命令の逐次実行（30 秒後の時刻の計算）

2 全てのプログラムは、ひとつ以上の文（statement）の並びです。個々の文は、それぞれ特定の
3 操作を表します。文の末尾には必ずセミコロン「;」を置きます。

4 この章で述べるプログラム実行に関する最初の原理は以下の通りです。

5 **第一原理** プログラム中の文は上から順番にひとつずつ実行されていく。

6 たとえば、前節最後で示した二つの連続する出力のあるプログラム：

```
function setup() {
  print("Hello, World");
  print(123+1);
}
```

7
8 には二つの出力文が含まれ、その二つが上から順に実行されました。よって最初に Hello, World
9 と出力され、次に 124 と出力されました。もしこれを逆順に並べ、

```
function setup() {
  print(123+1);
  print("Hello, World");
}
```

10
11 とすれば、最初に 124 と出力され、次に Hello, World が出力されます。

12 もう少し複雑なプログラム例に取り組みましょう。図 5.6 は、30 秒後の時刻（秒）を計算して出
13 力する p5.js プログラムです。まず、このプログラムに含まれる新規事項について説明します。

14 **時刻（秒）の取得** 1 行目の等号記号 = の右辺の関数 `second()` は、コンピュータの時計の分を数
15 値として求める関数です。p5.js では他に関数 `hour()`、`second()`、`day()`、`month()`、`year()` が
16 それぞれ文字通りの意味で用意されています。たとえばプログラムが実行された瞬間の時刻が 11
17 時 34 分 27 秒ならば、数値 27 が関数の値として求められます。

1 **変数** コンピュータにメイン・メモリと呼ばれる記憶装置があることは前章で述べた通りです。p5.js
 2 などの高水準プログラミング言語では、そのメモリ上の適当な領域（コンピュータが自動的に割り
 3 当てる領域）に名前を付けることができ、その領域のことを変数（variable）と呼びます。変数に
 4 は自由な名前²を付けることができ、その名前を変数名（variable name）と呼びます。たとえば図
 5 5.6 のプログラムには a、b という変数名が使用されています。変数にはデータを格納することがで
 6 き、逆に変数からデータを取り出すことができます。使用できる変数の個数に制限がありません³。

7 **代入文** 「 $x = e$ 」という形の文を「代入文（assignment statement）」と呼びます。右辺 e は一般
 8 的な算術式です。左辺 x は変数名です。代入文を実行すると、コンピュータは右辺の算術演算式
 9 e の値 v を計算し、その値 v を左辺の変数 x の表す記憶領域に格納します。以下では、このこと
 10 を簡単に「式 e の値を変数 x に代入する」と言うことにします。式の中に変数が含まれる場合に
 11 は、その変数を対応する記憶領域に格納されている値が式の計算に参照されます。式中で用いるこ
 12 とのできる主要な演算子は以下の通りです。

13 + ... 加算演算子。

14 - ... 減算演算子。

15 * ... 乗算演算子。通常の算数、数学ではバツ印 $\times$ や中点 $\bullet$ で表されたり、あるいは演算子は省略さ
 16 れることもあります。p5.js に限らずほとんど全てのプログラミング言語では乗算演算子は
 17 専ら * で表します。演算子を省略することはできません。記号 * をアスタリスク（asterisk）
 18 またはスターと呼びます。日本語訳では星印と呼びます。記号 * を乗算記号に用いるのは、
 19 初期のコンピュータではキーボード上に $\times$ や $\bullet$ に対応する記号がなかったためです。

20 / ... 除算記号。通常の算数、数学では $\div$ や分数形式 $\frac{a}{b}$ で表されますが、p5.js に限らずほとんど
 21 全てのプログラミング言語では / を用います。記号 / をスラッシュ（slash）と呼びます。

22 % ... 剰余演算子。剰余計算はプログラミングではしばしば必要となります。最近の多くのプログ
 23 ラミング言語ではパーセント記号を演算子として利用しています⁴。式 $m\%n$ では m を n で
 24 割った余りが式の値になります。

25 必要ならば、括弧（ ）を用いて式を束ねることができます。ただし、中括弧 { }、角括弧 [] を式
 26 に用いることはできません⁵。

27 さて、図 5.6 のプログラムを再度読んでみましょう。

²自由とはいえ、名付けには一定の制限があり、一般には英文字の後に英文字または数記号が 0 個以上続くような文字列と考えてください。

³実際にはコンピュータのメモリは有限なので、無限に多くの変数を使用できる訳ではありません。しかし、人が手書きするプログラムに現れる変数の数はせいぜい数百個であり、使用する変数が多過ぎてプログラムが動かないというような事態は起こりません。

⁴これは 1980 年代以降に流行った C 言語その後継の C++、Java などの影響と思われます。

⁵角括弧 [] は式中で使用できるのですが、それは p5.js では別の意味を持ちます。

```
function setup() {
  c = 30;
  a = second();
  b = (a+c)%60;
  print(b);
}
```

図 5.7: 30 秒後の時刻 (秒) を表示するプログラム (その 2)

```
function setup() {
  a = second();
  a = (a+30)%60;
  print(a);
}
```

図 5.8: 30 秒後の時刻 (秒) を表示するプログラム (その 3)

- 1 まず、最初の文：
- 2 `a = second();`
- 3 は、右辺の値が時刻の秒 (よって 0 から 59 の値) であるような代入文です。
- 4 最初の文の実行が終了した後に 2 番目の文：
- 5 `b = (a+30)%60;`
- 6 が実行されます。この文も代入文です。a の値に 30 を加え、60 の剰余を計算し、計算結果を b に
- 7 代入します。もし a に 34 が入っているならば、b には 4 が代入されます。
- 8 2 番目の文の実行が終了した後に 3 番目の文：
- 9 `print(b);`
- 10 が実行されます。この文では、b の値を出力します。もし b に 4 が入っているならば、コンソー
- 11 ルエリアに
- 12 4
- 13 と出力されます。3 番目の文の実行が終了した時点でプログラム全体の実行が終了です。
- 14 ところで、図 5.7 のプログラムも図 5.6 のプログラムと同等の働きをします。図 5.8、図 5.9 もそ
- 15 うです。
- 16 このように、ひとつの処理をプログラムで表す場合、様々な表現が可能です。

```
function setup() {
  print((second()+30)%60);
}
```

図 5.9: 30 秒後の時刻（秒）を表示するプログラム（その 4）

また、プログラムでは空白文字「」を随時、追加・削除しても構いません。たとえば、上で考察した代入文 $b = (a+30)\%60$; は

$b=(a+30)\%60$, $b = (a + 30) \% 60$, $b = (a + 30) \% 60$

など書いても構いませんが、読み易い書き方が良いのは言うまでもありません。一般には「2 項演算子の前後にひとつ空白を入れる」方法が読みやすいと考えられており、上では最右の書き方がそれに当たります。関数呼び出し `second()` は

`second ()`, `second ( )`

と書いても問題ありませんが、

`sec ond ( )`, `secon d()`

と書くことはできません。関数名 `second` は、いわば単語であり、単語を空白を挟んで分解すると意味が変わってしまうからです。

プログラムは、コンピュータへの指示を書き下した文章です。文章という意味では日本語や英語などの自然言語の文章と類似した性質を持ち、ある意味を表す文章の書き方は唯一ではなく、表現上の幅があり、同一の文章でなくとも同じ意味を表すことが可能です。同じ意味を表す場合でも、どの表現が分かりやすいのか、どの表現が誤解を与えやすいか、という違いがあります。

振り返って、図 5.7 のプログラムは変数 `c` の使い方が冗長で、無駄です。わざわざ定数 2 を `c` に代入して使用する必要はありません。一般に、冗長性はプログラム理解の妨げになります。これに対して、図 5.9 のプログラムは 4 つのプログラムの中で最も簡素な表現です。しかし 1 行に全ての仕事（入力操作、乗算、出力操作）が凝縮されており、初心者には少し理解が難しいプログラムになっています。どのようなプログラムが読みやすいか（プログラムの意味するところをプログラマが容易に読解できるか）、その基準はそう単純ではなく、一筋縄では行かない難しい問題です。

ところで、図 5.8 のプログラムの 2 行目：

$a = (a+30)\%60$;

では、代入文の左辺にも右辺にも変数 `a` が使用されており、注意が必要です。この行が実行される直前に `a` に 34 が格納されていたと仮定します。代入文の実行では、まず右辺の式が計算されま

1 す。よって 34 が取り出され、式の値 4 が求められます。そして、その 4 が左辺の a に代入され
2 ます。結果として、この行の実行後には a には 4が入っています。元の値 34 はどこにも残ってい
3 ません。上書き (overwrite) されたのです。上書きされた値は決して復元できません。もしその値
4 がプログラムの他の場所でも必要ならば、a を上書きすべきではなく、図 5.6 のプログラムのよう
5 に別の変数 b を用いるべきです。また、上書きはプログラムの読み易さを低下させます⁶から、で
6 きれば避けた方がよいでしょう。しかし、後に示すように、本質的に上書きを避けられないプログ
7 ラムも存在します。

⁶上書きを行うと、変数に格納されている値が途中で変化することになります。そうすると、プログラムのその部分を理解する上で、前後の文脈、順序関係を強く意識する必要が生じ、その分だけ理解すべき内容が複雑になってしまうからです。

```
function setup() {
  a = hour();           // 現在の時間を取得
  if (a >= 12) {         // もし 12 以上ならば
    a = a-12;           // 12 を引く
  }
  print(a);             // 時間を表示
}
```

図 5.10: 24 時間制の時間を 12 時間制へ変更するプログラム

5.5 条件判定と実行の分岐（時間制の変更）

5.4 節では、文が順に実行されていくことを説明しました。これはプログラムの実行の大原則ですが、これだけでは複雑な処理はできません。次の原理が必要です。

第二原理 プログラム中では、条件判定を行い、その判定結果の真偽に従い、次に実行する文を取捨選択できる。

簡単な例として、24 時間制の時間（0～23）を 12 時間制（0～11）へ変更するプログラムを図 5.10 のように考えます。まず、このプログラムに含まれる新規事項について説明します。

if 文 プログラム中の以下の形の部分を if 文（if statement）と呼びます。

```
if ( 条件式 ) {
  then 部
}
```

if 文の実行では、まず初めに条件式の値を求めます。その値が真（true）ならば then 部の文を実行し、この if 文の実行を終了します。その値が偽（false）ならば then 部を実行せずに、この if 文の実行を終了します。なお、then 部にはひとつ以上の文を書くことができます。

条件式 条件式とは、最終的な式の値が真または偽になる式のことです。典型的な条件式は数値の比較を行う式です。比較に使用できる 2 項演算子は以下の通りです。

< ... 左辺の数値が右辺の数値よりも小さいならば真。さもなければ偽。

<= ... 左辺の数値が右辺の数値よりも小さい、または等しいならば真。さもなければ偽。数学の $\leq$ ではないことに注意してください⁷。

> ... 左辺の数値が右辺の数値よりも大きいならば真。さもなければ偽。

⁷そもそも $\leq$ は通常のキーボードでは直接使用できない（エディタで打鍵したり、画面に表示したりできない）ので、代わりに $<=$ を用います。

```
function setup() {
  a = hour();           // 現在の時間を取得
  if (a < 12) {          // もし 12 よりも小さいならば
    print(a);           // そのまま表示する
  } else {
    print(a-12);        // さもなくば、12 を引いた値を表示する
  }
}
```

図 5.11: 24 時間制の時間を 12 時間制へ変更するプログラム (その 2)

- 1 >= ... 左辺の数値が右辺の数値よりも大きい、または等しいならば真。さもなくば偽。
- 2 == ... 左辺の数値が右辺の数値と等しいならば真。さもなくば偽。等号記号 = ではなく、等号記号を二つ接続したもの == であることに注意してください。= は代入文の演算子です。
- 3
- 4 != ... 左辺の数値が右辺の数値と異なるならば真。さもなくば偽。数学の $\neq$ とは異なることに注意してください⁸。
- 5

- 6 また、条件式には真偽値そのものも使用可能で、true は真値を表し、false は偽値を表します。

- 7 **コメント** プログラムが複雑になってくると、プログラム理解がだんだん難しくなってきます。その
- 8 ような場合にプログラムの中に様々な注釈を記しておけばプログラム理解に役立ちます。そのよ
- 9 うな注釈を**コメント** (comment) と呼びます。p5.js では各行の// (ダブルスラッシュ) から行の終
- 10 わりまでをコメントと見なします。コメントはプログラムの実行に何の影響も及ぼしませんから、
- 11 何を書いても構いません。

- 12 さて、図 5.10 のプログラムの実行の様子を解説します。

- 13 まず現在の時間を変数 a に格納します。次の 2 行目から 4 行目が if 文です。まず、if 文の条件式
- 14 の真偽を判定します。もし a の値が 12 以上でない (12 よりも小さい) ならば、条件式は成り立
- 15 ちません。よって 3 行目の文は実行せず、制御は 5 行目へ進みます。もし、a の値が 12 以上なら
- 16 ば、条件式は成り立ちます。よって 3 行目を実行し、式 a-12 の値を a に代入します。この if 文
- 17 の then 部はこの 1 行だけですから、if 文の実行はこれで終了し、制御は 5 行目へ移動します。5
- 18 行目では、a の値を出力します。

- 19 図 5.11 のプログラムも同じ表示を行うプログラムです。このプログラムに含まれる新規事項に
- 20 ついて説明します。

⁸ これも <= の場合と同様に、 $\neq$ は通常のキーボードでは直接使用できないので、代わりに != を用います。

```
function setup() {
  a = hour();           // 現在の時間を取得
  if (a >= 12) {         // もし 12 以上ならば
    a = a-12;           // 12 を引く
  } else {
  }
  print(a);
}
```

図 5.12: 24 時間制の時間を 12 時間制へ変更するプログラム (その 3)

- 1 **if-else 文 (あるいは単に、if 文)** プログラム中の以下の形の部分を if-else 文 (if-else statement)、
- 2 または簡単に if 文と呼びます。

```
if ( 条件式 ) {
  then 部
} else {
  else 部
}
```

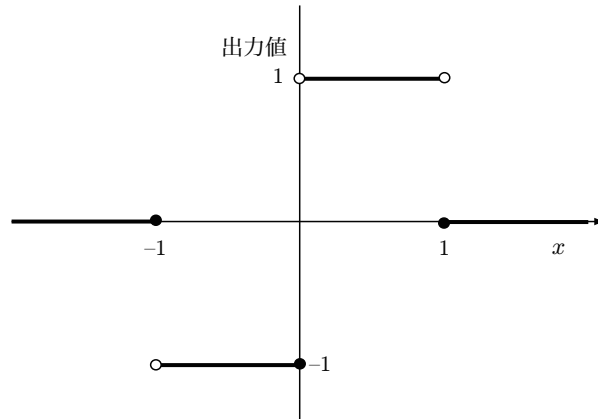
- 4 ここに、then 部/else 部にはそれぞれ 0 個以上の文を書くことができます。if-else 文の実行では、
- 5 条件式の値が真ならば then 部の文を実行し、偽ならば else 部を実行します。

- 6 図 5.11 のプログラムでは、もし a の値が 12 よりも小さいならば、単に a の値を出力します。さ
- 7 おなくば式 a-12 の値を出力します。

- 8 ところで、図 5.10 のプログラムで用いた if 文には else 部がありませんでした。else 部を持たな
- 9 い if 文は、if-else 文の、else 部に文がない特殊な場合と考えることができます。よって、図 5.10 の
- 10 プログラムを if-else 文を用いて書くならば、図 5.12 のように書き換えることができます。この場
- 11 合、else 部が空です。また、図 5.13 のように、条件式の真偽判定を逆転した書き方も可能です。こ
- 12 の場合は then 部が空です。しかし、わざわざ空な部分を書いても無意味です。通常、図 5.12、図
- 13 5.13 のようなプログラムを書くことはありません。

```
function setup() {  
  a = hour();          // 現在の時間を取得  
  if (a < 12) {         // もし 12 よりも小さいならば  
  } else {  
    a = a-12;          // さもなくば 12 を引く  
  }  
  print(a);  
}
```

図 5.13: 24 時間制の時間を 12 時間制へ変更するプログラム (その 4)

図 5.14: 入力値 x と出力値のグラフ

1 if 文、if-else 文を入れ子にして複雑な条件判定を行わねばならない場合もあります。
 2 たとえば、正負の値 x について以下のような出力を行うプログラムを検討しましょう（図 5.14
 3 参照）。

- 4 1. x が 0 以下の数であり、
 - 5 (a) x が -1 以下ならば、0 を出力。
 - 6 (b) さもなくば (x が -1 よりも大きいならば)、 -1 を出力。
- 7 2. さもなくば (x が 0 よりも大きいであり)、
 - 8 (a) x が 1 よりも小さいならば、1 を出力。
 - 9 (b) さもなくば (x が 1 以上ならば)、0 を出力。

10 このような入出力の関係は電気回路にありそうな関係です。このプログラムは図 5.15 のように書
 11 くことができます。ただし、正負の整数値 x として 現在の時刻の秒から 30 を引いて、10 で割っ
 12 た値（そうすると x の値は -3.0 から 3.0 の少数点数値の範囲です）を用います。また、ここでは
 13 プログラムの実行結果として x と出力値のペアを表示します。

14 p5.js では上のプログラムは図 5.16 のように書くこともできます。このプログラムは次の新規事項
 15 を含みます。

16 **else if 文** 以下の形も if 文の拡張された形です。

```
function setup() {
  x = (second()-30)/10;           // 現在の秒から 30 を引く
  if (x <= 0) {
    if (x <= -1) {
      print(x,0);
    } else {
      print(x,-1);
    }
  } else {
    if (x < 1) {
      print(x,1);
    } else {
      print(x,0);
    }
  }
}
```

図 5.15: if-else 文が入れ子になる例

```
function setup() {
  x = (second()-30)/10;           // 現在の秒から 30 を引く
  if (x <= -1) {
    print(x,0);
  } else if (x <= 0) {
    print(x,-1);
  } else if (x < 1) {
    print(x,1);
  } else {
    print(x,0);
  }
}
```

図 5.16: if-else 文が入れ子になる例 (その2)

```

if ( 条件式 1 ) {
  then 部
} else if ( 条件式 2 ) {
  else-if-then 部 2
1  ...
  } else if ( 条件式 n ) {
    else-if-then 部 n
  } else {
    else 部
  }

```

2 then 部/else-if-then 部 2/.../else-if-then 部 n/else 部にはそれぞれ 0 個以上の文を書くことができ
 3 ます。上の if 文の実行では、上から順に条件式を判定します。そして最初に真になった条件式に対
 4 応する then 部/else-if-then 部を実行します。もし全ての条件式が成り立たないならば、else 部を実
 5 行します。通常の if 文において **else** と else 部は省略可能です。

6 **考察** 以下のプログラムが、図 5.16 のプログラム と同じ意味を持つとします。このとき、[空白
 7 1]～[空白 6] に適切な式、代入文を埋めなさい。

```

function setup() {
  x = (second()-30)/10;           // 現在の秒から 30 を引く
  if (x >= 1) {
    [空白 1]
  } else if ([空白 2]) {
    [空白 3]
  } else if ([空白 4]) {
    [空白 5]
  } else {
    [空白 6]
  }
}
8

```

9 **考察** 以下のプログラムは、三つの数を変数 a、b、c へ入力し、その中の最大値を変数 max に求
 10 め、表示するプログラムです。空白部に適切な式、代入文を埋めなさい。

```
function setup() {
  a = hour();
  b = minute();
  c = second();
  if (a > b) {
    if ([空白 1]) {
      [空白 2]
    } else {
      [空白 3]
    }
  } else {
    if ([空白 4]) {
      [空白 5]
    } else {
      [空白 6]
    }
  }
  print(max)
}
```

1

- 2 **考察** 以下のプログラムも、三つの数を変数 a、b、c へ入力し、その中の最大値を変数 max に求
- 3 め、表示するプログラムです。空白部に適切な式、代入文を埋めなさい。

```
function setup() {
  a = hour();
  b = minute();
  c = second();
  max = a
  if ([空白 1]) {
    [空白 2]
  }
  if ([空白 3]) {
    [空白 4]
  }
  print(max)
}
```

4

```
function setup() {
  a = hour();           // 現在の時間を取得
  if (a < 12) {          // もし 12 よりも小さいならば
    print(a);           // そのまま表示する
  } else {
    print(a-12);        // さもなくば、12 を引いた値を表示する
  }
}
```

図 5.17: 段下げの無いプログラム

```
function setup() {
  a = hour();           // 現在の時間を取得
  if (a < 12) {          // もし 12 よりも小さいならば
    print(a);           // そのまま表示する
  } else {
    print(a-12);        // さもなくば、12 を引いた値を表示する
  }
}
```

図 5.18: 段下げが破茶滅茶なプログラム

1 5.6 段下げ（インデントーション）

前節のプログラム（図 5.10 から図 5.13）を見直しましょう。たとえば図 5.11 のプログラムでは、if 文の中の 3 行目と 5 行目の文の行の開始位置が 2 文字分だけ右にずれています。これはプログラムを見易くするための作法（さくほう⁹）で、一般に「段下げ（インデントーション、indentation）」、あるいは簡単にインデントなどと呼ばれています。

たとえば図 5.11 のプログラムを図 5.17 のように書いたらどうでしょうか。プログラムの構造（if-else 文がひとつあって、then 部と else 部にそれぞれ文がひとつずつ含まれること）が読み取りにくくなります。さらに、わざと図 5.18 のように書いたらどうでしょうか。ますます読みにくくなっています。ただし、読みにくいプログラムであっても、規則に従って正しく書かれたプログラムは正常に実行できます。図 5.17 のプログラムも図 5.18 のプログラムも正しく動作します。

プログラムが一過性のものであるならば、読みにくいプログラムでも問題ないでしょう。しかし、プログラムは常に改訂・改良されて長く使用され続ける資産です。資産としてもプログラムを保全するためには、読みやすいプログラムの作成が欠かせません。

幸いなことに p5.js のエディタにはプログラムを自動的にきれいに整形してくれるツールが用意

⁹作法が、儀礼的なしきたりを意味する場合には「さほう」と読み、ものの作り方を意味する場合には「さくほう」と読むのが一般的です。

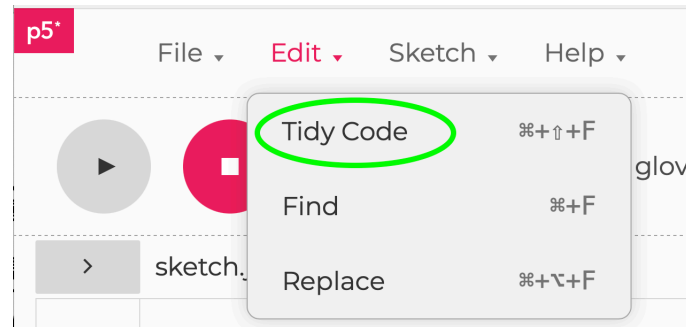


図 5.19: プログラムの自動整形コマンド

- 1 されています。図 5.19 の “Tidy Code”（＝コードを整頓する）がそれです。このコマンドを実行
- 2 すると p5.js が想定する読みやすいプログラムに自動整形されます。プログラム開発の途中で適時
- 3 実行してください。

4 5.7 フローチャート（プログラムの図式化）

5 プログラムを理解することは容易な作業ではありません。自分の書いたプログラムを読むこと
 6 は、基本的な実行の流れがもともと自分の頭の中に入っているため、さほど難しいことはありません。
 7 しかし他人の書いたプログラムを読むときには、そのプログラムを書いた人がどのようなア
 8 イディアでそのプログラムを構築したかを、プログラムの文面からひとつひとつ読み解いていく必
 9 要があります。これには大変な時間がかかり、容易なことではありません。

10 他人の書いたプログラムを読むことは他人の書いた文章を読むことに似ています。しかし、プロ
 11 グラムは日本語や英語などの文章と異なり、厳密な意味を持ちます。そのため読解にも厳密な理解
 12 が要求され、そのストレスは日本語、英語の文章の比ではありません¹⁰。また自分の書いたプログ
 13 ラムであっても何年も前に書いたプログラムの内容は忘れてしまっていることがほとんどです。か
 14 ら、他人の書いたものと同じ位に読みにくいものです。

15 プログラムの読解が難しいという問題は ENIAC などの初期の電子式コンピュータが開発された
 16 直後から認識されていました。そして、これまで様々な解決法が提案されてきました。その解決
 17 法のひとつが、段下げの励行です。そしてもうひとつの方法として、プログラムを図式化すること
 18 も古くから行われています。図式化にも様々な手法がありますが、初級者向けにはフローチャート
 19 （流れ図、flow chart）を用いる方法が一般的です。

20 たとえば図 5.20 は図 5.10 のプログラムをフローチャート化したものです¹¹。

¹⁰ プログラムを読み書きするには、高い集中力が要求されます。そのため、プログラマとして働けるのは 30 代が限界であるという「プログラマ 30 代定年説」が語られます。一面において真理ですが、プログラミングの開発環境や方法論は年々進歩しており、限界年齢も伸びているように思われます。

¹¹ JIS に準拠した方法で作図しました。

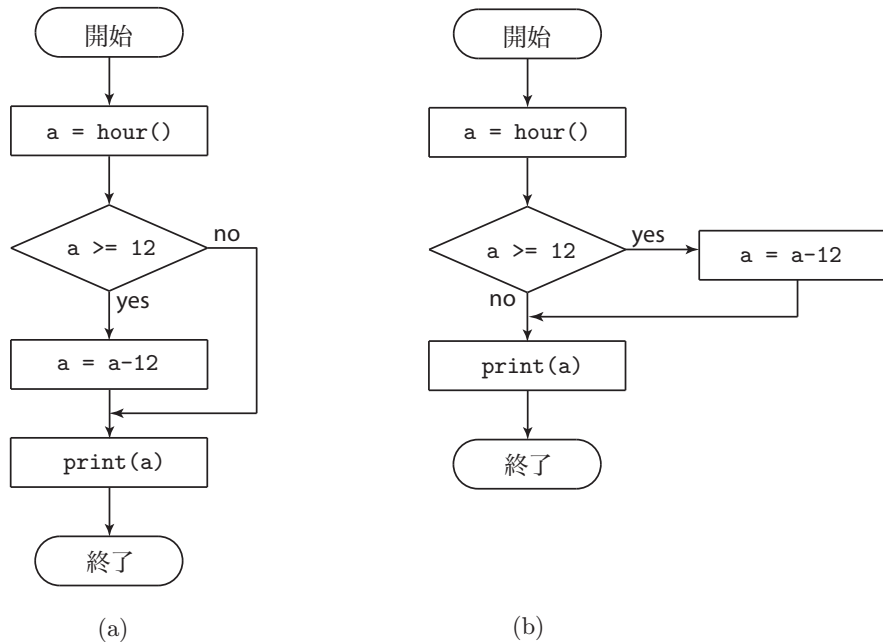


図 5.20: 図 5.10 のプログラムのフローチャート

1 まず初めに図 5.20 (a) の読み方/書き方を解説します。フローチャートは通常、上から下へ向
2 かって処理が進むように描きます。プログラムの開始位置、終了位置は丸角ブロックの中に「開
3 始」、「終了」を記載して表します。角ブロックには、代入文などの基本的な文を記入します。図で
4 は p5.js の文をそのまま記入していますが、「a = hour()」の代わりに「a に時刻 (時間) を入力
5 する」と書いても構いません。ブロック間には矢印 (または線) を引いて、処理の流れる方向を明
6 らかにします。if 文の条件式は、菱形 (◇) 内に記載し、条件式が真の場合と偽の場合の処理の流
7 れをそれぞれ矢印で表します。一般に条件式が真になる方向の矢印に yes を、偽になる方向の矢
8 印に no をそれぞれ付加します。if 文が終了し、処理が合流する箇所では、矢印を合流させます。
9 フローチャートの書き方では、処理が上から下へ、左から右へ流れるように作図するのが原則です
10 が、それに従ってもフローチャートの書き方は一意ではありません。図 5.20 (b) も図 5.10 のプロ
11 グラムのフローチャートです。

12 さて、図 5.20 の (ふたつの) フローチャートと図 5.10 のプログラムを見比べてください。処理
13 の流れを把握するにはフローチャートの方が理解しやすいことに気付くはずですが (ただ、この例題
14 は簡単過ぎるため、そう思わない人もいるかもしれません)。

15 この章ではプログラムを示した後にフローチャートを紹介しました。しかし、通常はプログラム
16 を書く前にまずフローチャートを作り、その後にプログラムを書く方が、大きな考え違いなどを未
17 然に防ぐことができ、プログラム開発は効率的であるとも言われています¹²。

¹²このことはプログラミングの初心者には当てはまりますが、中級以上のプログラマでは必ずしも成り立たず、フローチャートは高度で複雑なプログラムを作る上では弊害が多いとも言われています。そこで、フローチャートに代わる様々な

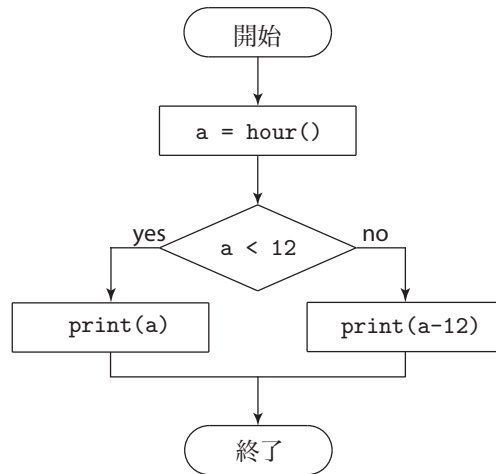


図 5.21: 図 5.11 のプログラムのフローチャート

- 1 ところで図 5.11 のプログラムも図 5.10 のプログラムと同等でした。その図 5.11 のプログラムの
- 2 フローチャートは図 5.21 の通りです。図 5.20 と図 5.21 を見比べると、二つのプログラムの構造的
- 3 な違いが一目瞭然に理解できます。

図式化手法が検討されています。最近の統合化設計手法 UML では 4、5 種類の異なる図式化を同時に行います。

```
function setup() {
  a = 5*4*3*2*1;
  print(a);
}
```

図 5.22: 5 の階乗 5! を計算し、出力するプログラム

```
function setup() {
  n = 10;           // (1), n の初期設定
  a = 1;           // (2), a の初期設定
  i = 1;           // (3), i の初期設定
  while (i <= n) {  // (4), i が n 以下である限り、繰り返す
    a = a * i;      // (5), a を i 倍して a を更新
    i = i + 1;      // (6), i を 1 増やす
  }                // (7), ループ本体のブロックの末尾
  print(a);
}
```

図 5.23: 10 の階乗 10! を繰り返しを用いて計算し、出力するプログラム

5.8 繰り返し実行とループ（階乗の計算）

コンピュータの実行に関する第三の原理は以下の通りです。

第三原理 プログラム中に繰り返し実行部を作ること、多数回の類似の処理を簡素に表現できる。

たとえば、図 5.22 の p5.js プログラムは 5 の階乗 5! ($5 \times 4 \times 3 \times 2 \times 1$) を計算し、出力するプログラムです。それでは次に、100! ($= 100 \times 99 \times \dots \times 2 \times 1$) を計算するプログラムはどのように書けばよいのでしょうか。原理的には代入文を以下のように書くこともできます。

```
a = 100*99*98*97*96*95*94* ... 途中省略 ... *3*2*1
```

ただし、プログラム中の文字数は 300 文字近くになります。さらに、10000! を計算するときにはどうでしょうか。そのプログラムの大きさは 48000 文字を超えるはずで、48000 文字のプログラムを用意することは原理的には可能であっても、この方法は実際上、破綻しています。

これを解決するためには、繰り返しを用いた図 5.23 のようなプログラムを作ります。繰り返しのことをプログラミング言語ではしばしばループ (loop) と呼びますから、これ以降、繰り返しとループを同じ意味の言葉として用いていきます。以下では、このプログラムに含まれる新規事項について説明します。

表 5.1: 図 5.23 のプログラムの実行の様子

	行	プログラム	n	a	i	備考
1	(1)	n = 10;	10			10 を入力
2	(2)	a = 1;	10	1		
3	(3)	i = 1;	10	1	1	
4	(4)	while (i <= n) {	10	1	1	条件式の値は真
5	(5)	a = a*i;	10	1	1	
6	(6)	i = i+1;	10	1	2	
7	(7)	}	10	1	2	
8	(4)	while (i <= n) {	10	1	2	条件式の値は真
9	(5)	a = a*i;	10	2	2	
10	(6)	i = i+1;	10	2	3	
11	(7)	}	10	2	3	
12	(4)	while (i <= n) {	10	2	3	条件式の値は真
13	(5)	a = a*i;	10	6	3	
14	(6)	i = i+1;	10	6	4	
15	(7)	}	10	6	4	
...	...	...	...	...	...	...
...	...	...	...	...	...	...
39	(4)	}	10	362880	10	
40	(4)	while (i <= n) {	10	362880	10	条件式の値は真
41	(5)	a = a*i;	10	3628800	10	
42	(6)	i = i+1;	10	3628800	11	
43	(7)	}	10	3628800	11	
44	(4)	while (i <= n) {	10	3628800	11	条件式の値は偽
45	(8)	print(a)	10	3628800	11	3628800 を出力

1 **while 文** プログラム中の以下の形の部分：

```

2   while ( 条件式 ) {
      ループ本体
3   }
```

3 を **while 文**（while statement）と呼びます。ここに、ループ本体には 0 個以上の文を書くことが
4 できます。while 文の実行では、条件式の値が偽ならば、ループ本体を実行せず、while 文の実行を
5 終了します。もし条件式の値が真ならばループ本体を実行し、制御は再び while 文の先頭に戻り、
6 条件式の判定を行います。そして、もし条件式の値が... と、条件式の値が偽になるまでループ本体
7 の実行を繰り返します。

8 図 5.23 のプログラムの様子を変数の値を追跡しながら調べてみましょう。表 5.1 はプログラム

1 の実行の様子を1行毎に追いかけたものです。表の最左カラムは実行ステップ番号です。1から21
 2 まで順に付番してあり、つまりプログラムの実行に21ステップ掛かっています。左から2番目の
 3 カラムは、実行する行（(1)から(8)）を示しています。（4）から（7）が繰り返し実行されてい
 4 ることに注意してください。その右のカラムには、その行に対応する文を載せました。n、a、iの
 5 カラムはそれぞれの変数の値を示しています。プログラムの実行と共に変数の値は変化していきま
 6 す。表の最右カラムには、実行に関する補足実行を載せました。

7 さて、図5.23のプログラムの1行目（コメント（1）の行）を実行する前には如何なる変数も値
 8 が決まっていません。よって、表5.1の1行目の三つの変数の値は空白のままにしています¹³。1
 9 行目を実行し、キーボードから4を入力したと仮定します。そうすると、変数nに4が代入され
 10 ます（表の2行目）。次に、プログラムの2行目、3行目を実行すると、変数a、iの値が順に定
 11 まります。結果、プログラム中の以下の3行目：

```
12  n = 10;           // (1), n の初期設定
13  a = 1;           // (2), a の初期設定
14  i = 1;           // (3), i の初期設定
```

15 によって各変数の初期設定ができました。

16 次にプログラムの4行目から7行目はwhile文です。

```
17  while (i <= n) {   // (4), i が n 以下である限り、繰り返す
18      a = a * i;      // (5), a を i 倍して a を更新
19      i = i + 1;      // (6), i を 1 増やす
20  }                 // (7), ループ本体のブロックの末尾
```

21 このwhile文では条件式 $i \leq n$ が成り立つ限り、while文の実行を繰り返します。

22 nが10、iは1の状況で条件式を計算すると、この条件式は真です（iの値はnの値以下です）。
 23 よって制御はループ本体へ移動します。そしてプログラムの5行目、6行目を実行します。5行目
 24 ではaにiの値を掛け込みます。6行目ではiの値をひとつ増やします。そして、7行目のend
 25 でループ本体の処理が終了しますから、制御は4行目へ戻り、再び条件式を評価し、その値が真な
 26 らば...とwhile文の実行を繰り返していきます。

27 このwhile文では、nの値は不変です。iの値はひとつずつ増えていきます。よって、ループを
 28 繰り返して実行すれば必ず条件式 $i \leq n$ の値はある時点で真から偽に転じ、while文の実行は終
 29 了します。

30 while文を抜けた¹⁴ならば、最後に8行目のputs aを実行し、aの値を出力して、プログラム
 31 全体の実行は終了です。このaの値が階乗数 $n!$ であることはプログラムから明らかです。

¹³ 正確には変数の値が未定義なのではなく、プログラム開始時点では変数そのものが存在しません。

¹⁴ ループ実行を終了することをしばしば「抜ける」と言います。

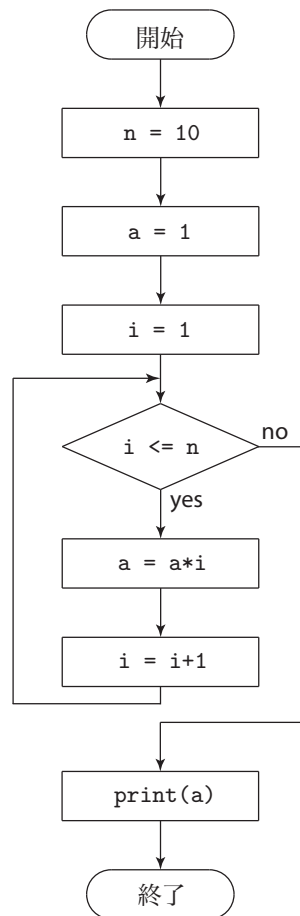


図 5.24: 図 5.23 のプログラムのフローチャート

このプログラムのフローチャートは図 5.24 の通りです。このフローチャートの中に循環路（閉路）があることに気付きます。この循環路は、図 5.23 のプログラムの中の while 文に対応しており、必ず n 回だけ繰り返して実行されます。このような、プログラム中の繰り返し部分のことをループ（loop）と呼びます。while 文の繰り返し部分をループ本体と呼んだ理由は、while 文全体がループを成しており、while 文の繰り返し部分がループの中核部分を成しているからです。

ひとつの処理を実現するプログラムが一種類に限らないことは while 文を用いる場合も同じです。図 5.25 は、図 5.23 のプログラムとは異なる形ですが、やはり入力値の階乗を計算するプログラムです。

考察 図 5.25 のプログラムの実行の様子を表 5.1 と同様に示しなさい。

考察 図 5.25 のプログラムのフローチャートを図 5.24 と同様に示しなさい。

コンピュータを用いた処理のほとんどは類似の作業の繰り返しです。たとえば銀行のオンライン

```
function setup() {
  n = 10;           // (1), n の初期設定
  a = 1;           // (2), a の初期設定
  while (n >= 1) {  // (3), n が 1 以上である限り、繰り返す
    a = a * n;      // (4), a を n 倍して a を更新
    n = n - 1;      // (5), n を 1 減らす
  }                // (6), ループ本体のブロックの末尾
  print(a);
}
```

図 5.25: 10 の階乗 10! を繰り返しを用いて計算し、出力するプログラム (その 2)

```
function setup() {
  n = 10;
  j = 1;
  while (j <= n) {
    a = 1;
    i = 1;
    while (i <= j) {
      a = a * i;
      i = i + 1;
    }
    print(a);
    j = j + 1;
  }
}
```

図 5.26: 1 の階乗、2 の階乗、...、10 の階乗を二重ループを用いて計算し、出力するプログラム

1 端末は、預け入れ/支払い/振り替え等の処理を利用者毎に繰り返しています。ビデオ・ゲームで
 2 は、コントロール・パッドからの入力を検知し、それをゲーム内の世界に反映する作業をえんえん
 3 と繰り返しています。ワードプロセッサを用いてこの文章を打ち込んでいるときには、キーボード
 4 からの入力を検知し、キーの組み合わせからカナ漢字変換を行い、漢字データをコンピュータの
 5 ディスプレイ上に表示する作業を繰り返しています。このように、コンピュータの処理の大半は繰
 6 り返し処理 (ループ処理) であり、ループ処理を理解し使いこなすことは非常に重要です。

7 ループの中にループを作ることも可能です。そのようなループを多重ループ (multi-nested loop)
 8 と呼びます。たとえば図 5.26 のプログラムでは、入力された数を n とするとき、 $1!$ 、 $2!$ 、...、 $n!$ を
 9 順に計算し、出力します。このプログラムのフローチャートは図 5.27 です。二重ループを持って
 10 いることが分かります。

11 **考察** 図 5.26 のプログラムの実行の様子を表 5.1 と同様に示しなさい。

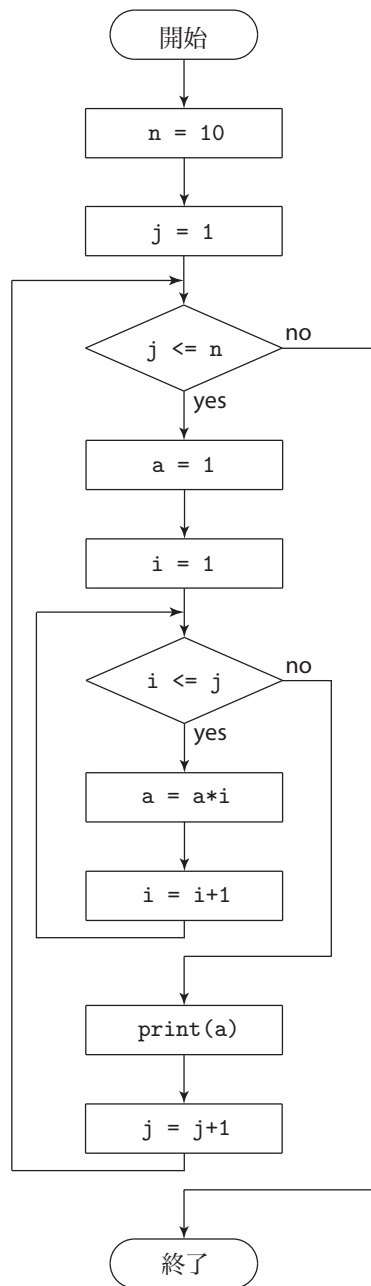


図 5.27: 図 5.26 のプログラムのフローチャート

```
function setup() {
  n = 10;
  for (j = 1; j <= n; j = j+1) {
    a = 1;
    for (i = 1; i <= j; i = i+1) {
      a = a * i;
    }
    print(a);
  }
}
```

図 5.28: 1 の階乗、2 の階乗、...、10 の階乗を for 文を用いて計算し、出力するプログラム

5.9 定型的な繰り返し実行

ここまで while 文を使った繰り返し処理を説明してきましたが、while 文の使い方にひとつの定型的なパターンがあることに気付いた人も多いでしょう。たとえば、図 5.26 のプログラムの場合、外側の while ループにおいて j の値が 1 から n までひとつずつ増えています。また内側の while ループにおいては i の値が 1 から j までひとつずつ増えています。このように、変数の値をひとつずつ増やしながら処理を繰り返すパターンはしばしばプログラム中に現れるため、p5.js ではそれを簡単に記述できる機能が用意されています。

図 5.28 のプログラムは図 5.26 のプログラムと同じ働きをしますが、while 文の代わりに for 文が使用されています。以下にその for 文を解説します。

for 文 プログラム中の以下の形の部分：

```
for ( 変数名 = 初期値 ; 変数名 <= 最終値 ; 変数名 = 変数名 +1 ) {
  ループ本体
}
```

を **for 文** (for statement) と呼びます。これは以下の while 文を等価です。

```
変数名 = 初期値 ;
while ( 変数名 <= 最終値 ) {
  ループ本体
  変数名 = 変数名 +1;
}
```

つまり、for 文は while 文の特殊形です。繰り返し処理の多く¹⁵は for 文を用いた方が while 文を用いた場合よりも簡潔に記述できます。

¹⁵全ての繰り返しを for 文で書ける訳ではありません。それについて述べることは少々詳しくなり過ぎるのでこの講義では省略します。プログラミング概論で詳細に学ぶことになります。

```
function setup() {
  score = [10,4,5,2,8];

  sum = 0;
  for (i = 0; i < score.length; i = i+1) {
    sum = sum+score[i];
  }
  print(sum/score.length);
}
```

図 5.29: 平均値を計算するプログラム

1 では for 文さえあれば while 文は不要かと問われれば、for 文の定型的な処理では記述できない
2 複雑な問題も存在しますから、その場合には while 文を用います。

3 **注意：** 変数値を 1 だけ増やす代入文として、ここでは

4 変数名 = 変数名 +1、

5 図 5.28 のプログラムで言えば、 $i = i+1$ 、 $j = j+1$ を用いています。しかし、p5.js、JavaScript、
6 あるいはその他のプログラミング言語ではその代わりとしてしばしば

7 変数名 ++、

8 $i++$ 、 $j++$ を用います。この ++ は値を 1 増やす単項演算子として C 言語で初めて導入され、その
9 簡潔性から広く利用されています。反面、初学者には分かりにくい上、熟練者にはプログラムの誤
10 りを誘導しやすいとも言われています。この講義テキストでは功罪両面を考慮し、この講義が初学
11 者向けであることから ++ は用いないことにします。同様の理由で 2 項演算子 += も用いないこと
12 とします。

13 5.10 データの列

14 プログラム実行に関する最後の原理は大量のデータの処理に関するものです。

15 **第四原理** 配列を用いてデータの並びを一括処理できる。

16 図 5.29 は、5 人の点数（score）が 10, 4, 5, 2, 8 だったときの、平均点を計算するプログラムで
17 す。ここに以下の新規事項があります。

1 **配列** プログラム中の以下の形の部分：

2 `配列名` = [`値 0`, `値 1`, ..., `値  $n-1$`] ;

3 は、 n 個のデータの並びである配列を作っている部分です。配列に含まれる各要素の読み書きには

4 `配列名` [`0 以上の 0 整数値`]

5 という形でアクセスできます。配列の大きさは配列名の後に `.length` を付けると知ることができます。
6

7 図 5.29 のプログラムでは変数 `sum` に点数の合計値を計算しています。そして最後に合計値を
8 `score.length` で割って平均値を求めます。

9 配列は大規模な計算を行う上では必須の機能です。たとえば大学入試センター試験では 50 万人
10 近い受験生の成績を集計しますが、各受験生の成績は一旦、配列に格納され、処理されています。

11 配列を用いたプログラムは非常に重要ですが、この講義テキストの守備範囲を超えるため、以上
12 の簡単な紹介のみで終わることとします。興味を持った人は自学してください。たとえば 5.1 節で
13 紹介した「文系大学生のための p5.js 入門」には上の紹介よりもさらに踏み込んだ内容が載ってい
14 ます。

1 第6章 p5.js プログラミング（グラフィックス編）

2 この章では p5.js で 2 次元コンピュータグラフィックスに挑戦します。3 次元コンピュータグラフィックスはこの講義テキストの守備範囲を超えるため、割愛します。

4 6.1 様々な線画

5 この節では様々な線画、特に多角形の描画とそのバリエーションを紹介します。

6 6.1.1 三角形

7 まず、簡単な例題として三角形を描くプログラムを紹介します。図 6.1 がそのプログラム、図 6.2 がその実行結果です。以下は、このプログラムで使用されている関数 `createCanvas()`、`line()` の解説です。なお、このテキストでは実行結果を表示するウィンドウ内の描画領域をキャンバスと呼ぶこととします。

11 以下は既に紹介した内容ですが、再確認します。

12 `setup()` ... 初期化関数です。プログラムの実行において初めに 1 回だけ実行されます。`draw()` が定義されていない場合には、この関数での描画結果がプログラムの実行結果としてキャンバスに保持されます。

```
function setup() {  
  createCanvas(640, 480);  
  
  line(200, 100, 100, 300);  
  line(100, 300, 500, 300);  
  line(500, 300, 200, 100);  
}
```

図 6.1: 三角形の描画プログラム

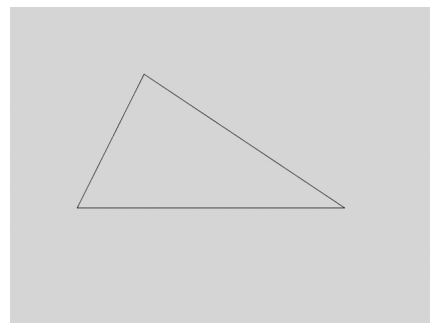


図 6.2: 左のプログラムの実行結果

1 `createCanvas(width, height)` ... キャンバスの幅、高さを画素を単位とする定数値で指定しま
2 す。この関数を明示的に呼び出さない場合、`width=height=100` がデフォルト値です。何か
3 を描画した後にこの関数を呼び出した場合、それまでの描画がクリアされますから、注意し
4 てください。

5 `line(x1, y1, x2, y2)` ... 2次元 X-Y 座標上の点 (x1,y1) から点 (x2,y2) へ直線を描きます。

6 ところで、p5.js の 2 次元座標系は **キャンバスの左上が原点** です。Y
7 座標値は下に向かって増えていきます。p5.js が左上を原点とする強い理由はハードウェア（メモ
8 リ、ディスプレイ装置）が左上を原点とする方式に適しており、それがそのまま採用されているた
9 めです。同じ理由で多くのグラフィックスシステムでは「左上が原点」を採用しています。

10 図 6.1 のプログラムではキャンバスを生成後、3 本の線分を引いています。

11 6.1.2 正多角形

12 次に正 6 角形に挑戦します。

たとえば座標点 (300, 250) に中心を持つ半径 200 の円を想定し、その円に内接する正 6 角形を
描画します。そのような正 6 角形の 6 個の頂点座標は以下の通りに表わすことができます。ここに
 $i = 0, 1, \dots, 5$ です。

$$(200 \cos(60i^\circ), 200 \sin(60i^\circ)) + (300, 250) \quad (6.1)$$

13 上の数式では角度を度数で表わしていますが、実際の数値計算ではラジアンで計算せねばなりませ
14 ん。一般に度数 d をラジアン r に変更するには式 $r = d \cdot \pi / 180$ で計算すればよいのですが、p5.js
15 では以下の変換関数 `radians()` が用意されています。

16 `radians(degrees)` ... 度数で表された角度 `degrees` に対応するラジアン値を返します。逆の方
17 向の計算（ラジアンを度数へ変換）は関数 `degrees(radians)` で可能です。

18 以上の考察をプログラム化したものが図 6.3、その実行結果が図 6.4 です。

さて、正 6 角形の議論は簡単に正 n 角形へ拡張できます。つまり式 (6.1) を

$$\left(200 \cos \left(\left(\frac{360}{n} \right) i^\circ \right), 200 \sin \left(\left(\frac{360}{n} \right) i^\circ \right) \right) + (300, 250)$$

19 に変えればいいだけです。そのプログラムが図 6.5、実行結果は図 6.6 の通りです。

20 **考察** 図 6.5 のプログラムにおいて n を十分大きくすれば近似的に円を描くことができるはず
21 す。人の目で見て円と区別できない n を自分の目で調べなさい。

```
function setup() {
  createCanvas(600, 500);

  xoffset = 300;
  yoffset = 250;
  r = 200;

  for (i = 0; i < 6; i = i+1) {
    t1 = radians(60 * i);
    x1 = r * cos(t1) + xoffset;
    y1 = r * sin(t1) + yoffset;

    t2 = radians(60 * (i + 1));
    x2 = r * cos(t2) + xoffset;
    y2 = r * sin(t2) + yoffset;

    line(x1, y1, x2, y2);
  }
}
```

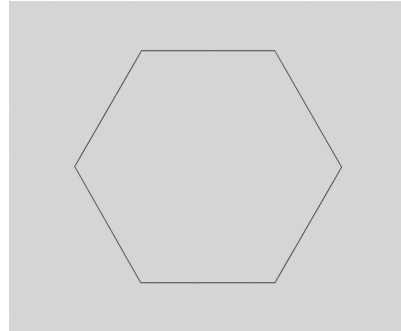


図 6.4: 図 6.3 のプログラムの実行結果

図 6.3: 6 角形の描画プログラム

```
function setup() {
  createCanvas(600, 500);

  xoffset = 300;
  yoffset = 250;
  r = 200;

  n = 11;           // 角数の設定
  incAngle = 360/n; // 角度の増分値の計算

  for (i = 0; i < n; i = i+1) {
    t1 = radians(incAngle * i);
    x1 = r * cos(t1) + xoffset;
    y1 = r * sin(t1) + yoffset;

    t2 = radians(incAngle * (i + 1));
    x2 = r * cos(t2) + xoffset;
    y2 = r * sin(t2) + yoffset;

    line(x1, y1, x2, y2);
  }
}
```

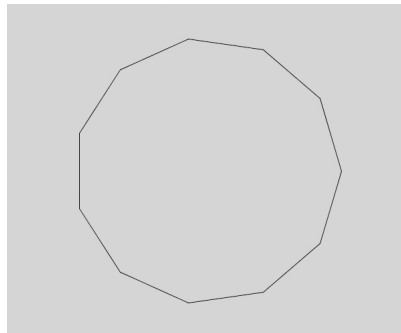


図 6.6: 図 6.5 のプログラムの実行結果

図 6.5: 正 n 角形の描画プログラム: $n = 11$ の場合

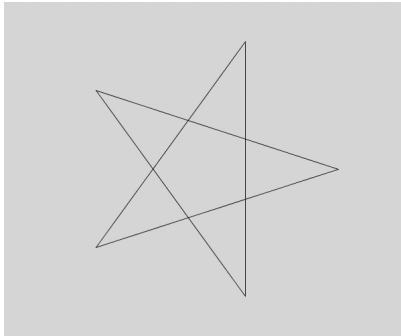


図 6.7: 五芒星 (= 正 5/2 角形)

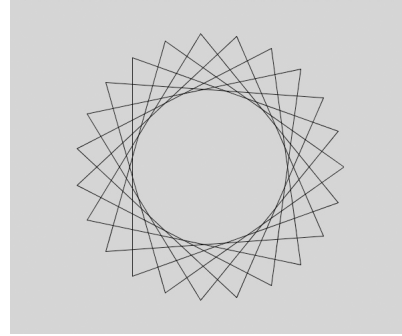


図 6.8: 図 6.9 のプログラムで作成した
正 23/7 角形

1 6.1.3 星型正多角形

2 さらにプログラムを改造します。多角形の n 個の頂点について、隣接する頂点間を描画するので
3 はなく、頂点を飛ばして描画すると星型の図形を描画できます。五芒星 (図 6.7 参照) が著名な例
4 です。

5 一般に正 n 角形の頂点を、飛び数 m で結んで (頂点を $m - 1$ 個飛びで結んで) 描かれる図形を
6 **正 n/m 角形**、総じて**星形正多角形** (regular star polygon) と呼びます。ただし、ここでは n と
7 m が互いに素とします¹。五芒星は正 5/2 角形に相当します。図 6.8 は正 23/7 角形です。これを描
8 画するプログラムは図 6.9 の通りです。

9 6.1.4 らせんと渦巻き

10 ここまで作ってきた多角形のプログラムでは内接円の中心は (300, 250) に固定していました。こ
11 の中心を角度と共に平行移動することで**らせん** (螺旋、helix) を作ることができます。たとえば
12 図 6.10 のようなものです。

13 ここまで作ってきた多角形のプログラムでは半径 $r = 200$ の円の内接点を用いました。この半
14 径 r を角度と共に単調増加させることで**渦巻き** (spiral) を描画できます。たとえば図 6.11 のよう
15 なものです。

16 **考察** 図 6.10 や図 6.11 のような画像を作成するプログラムを作成しなさい。

¹ n と m が互いに素でない場合には議論が少し複雑になり、複合型という形状を考えねばなりません。たとえば六芒星がこの場合に当たります。

```
function setup() {
  createCanvas(600, 500);

  xoffset = 300, yoffset = 250;
  r = 200;
  n = 23;
  m = 7; // mの宣言を追加
  incAngle = 360 / n;

  for (i = 0; i < m * n; i = i+m) // iをmずつ増やす
  {
    t1 = radians(incAngle * i);
    x1 = r * cos(t1) + xoffset;
    y1 = r * sin(t1) + yoffset;

    t2 = radians(incAngle * (i + m)); // 終点をm個先へ変える
    x2 = r * cos(t2) + xoffset;
    y2 = r * sin(t2) + yoffset;

    line(x1, y1, x2, y2);
  }
}
```

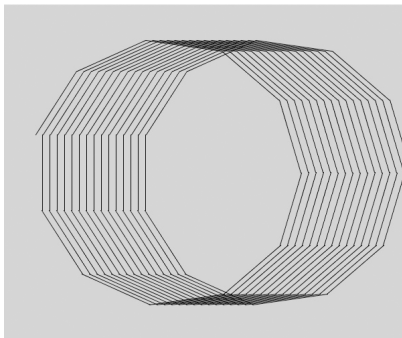
図 6.9: 正 n/m 角形の描画プログラム: $n = 23$ 、 $m = 7$ の場合

図 6.10: らせんの例

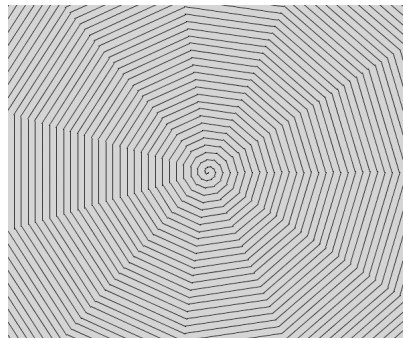


図 6.11: 渦巻きの例



図 6.12: 3 原色による加色混合

1 6.1.5 RGB カラー

2 CG では、赤色 (Red)、緑色 (Green)、青色 (Blue) の **3 原色** (three primary colors) を適当
 3 な輝度で加色混合²して様々な色を作っています。加色混合はちょうど図 6.12 のような、赤、緑、青
 4 の光による色の混ぜ合わせに相当し、このようにして作られる色は 3 原色の頭文字を取って **RGB**
 5 **カラー** (RGB color) と呼ばれています。また以下では赤、緑、青の各色をしばしば簡単に **R**、**G**、
 6 **B** と略記します。

7 一般的な RGB カラーでは 3 原色の各色は 256 段階の輝度を表わすことができ、計 1670 万
 8 ($= 256^3$) 通りの色を表現可能です。1670 万色は人間が区別できる色数を越えているとされます。
 9 データ表現では R、G、B の順に 3 原色の輝度値を並べるのが一般的です。RGB 値の典型的な例
 10 は以下の通りです。

11 赤 ... (255, 0, 0)

12 緑 ... (0, 255, 0)

13 青 ... (0, 0, 255)

14 黄 ... (255, 255, 0)

15 空色 ... (0, 255, 255)

16 紫 ... (255, 0, 255)

17 上の例では各色の輝度値を 0 または 255 としましたが、その間の数値を用いることで微妙な色合
 18 いを表現することも可能です。

²これに対比する言葉は減色混合です。絵の具の色の混ぜ合わせがそれに相当します。

```
function setup() {
  createCanvas(500, 500);

  background(128);

  for (i = 100; i <= 400; i = i+100) {
    stroke(i / 2);
    strokeWeight(i / 10);
    line(100, i, 400, i);
  }
}
```



図 6.13: 背景色、線の色をグレースケールで指定し、さらに線幅を変えるプログラム

図 6.14: 左のプログラムの実行結果

```
function setup() {
  createCanvas(500, 500);

  background(0, 255, 0);

  for (i = 100; i <= 400; i = i+100) {
    stroke(i / 2, 0, 0);
    strokeWeight(i / 10);
    line(100, i, 400, i);
  }
}
```

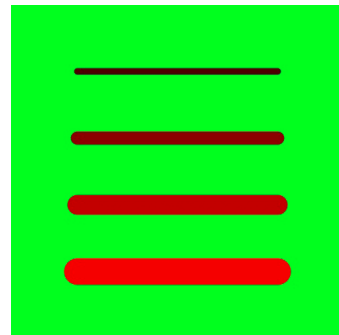


図 6.15: 背景色、線の色を RGB カラーで指定し、さらに線幅を変えるプログラム

図 6.16: 左のプログラムの実行結果

1 6.1.6 線の色、幅の変更

2 ここまでの描画では線の色、幅はデフォルト値（色は黒、幅は1画素）を用いてきましたが、こ
3 れを変更することができます。

4 図 6.13 は、背景色、線の色、線の幅を様々な明るさの灰色に変更するプログラムの例です。図
5 6.14 がその実行結果です。図 6.15 は、背景色、線の色をフルカラー RGB 値で指定するプログラ
6 ムです。その実行結果が図 6.16 です。

7 これらのプログラムでは以下の関数を用いてます。

8 `background(gray)` ... キャンバスの背景色をグレイスケール値で指定します。0 が黒色、255 が
9 白色、1~254 は暗い灰色から明るい灰色です。プログラム中でこの関数を呼び出さない場合

- 1 は明るい灰色がデフォルト値³です。
- 2 `background(red, green, blue)` ... キャンバスの背景色を RGB 値 (`red, green, blue`) で指定し
3 ます。
- 4 `stroke(gray)` ... 線の色をグレイスケールで指定します。プログラム中でこの関数を呼び出さな
5 い場合は黒色がデフォルト値です。
- 6 `stroke(red, green, blue)` ... 線の色を RGB 値 (`red, green, blue`) で指定します。
- 7 `strokeWeight(weight)` ... 線の幅を画素単位で指定します。デフォルト値は 1 です。
- 8 **考察** p5.js の関数 `strokeCap()` の処理内容を調べなさい。

9 6.1.7 その他の形状

- 10 p5.js の線の形状として、直線だけではなく、円弧、楕円、長方形などを関数で容易に描画でき
11 ます。しかしこれらは、次節で扱う面積を持つ形状の境界線と考えることができますから、この節
12 では説明を省略します。

³明るい灰色の具体的な数値は不明です。

```
function setup() {
  createCanvas(640, 480);

  stroke(0, 255, 0);
  strokeWeight(10);

  fill(255, 0, 0);

  triangle(200, 100, 100, 300, 500, 300);
}
```

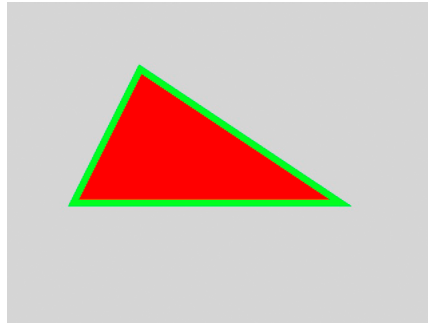


図 6.17: 内部を赤色で塗り、境界を緑色で塗る三角形の描画プログラム

6.2 様々な面画

複数の線分で囲まれた領域を特定の色で塗りつぶすことで面画を作ることができます。

6.2.1 三角形

まず、簡単な例題として三角形の内側を赤色で塗るプログラムを図 6.17 に示します。実行結果は図 6.18 です。

このプログラムでは、図 6.2 と同じ形状の三角形を、境界線を緑色で太さが 10 画素の線分で描き、内部を赤色で塗っています。これには以下の関数 `fill()`、`triangle()` を用います。

`fill(gray)` ... 描画形状の内部の色をグレースケールで指定します。

`fill(red, green, blue)` ... 描画形状の内部の色を RGB 値 (`red, green, blue`) で指定します。

`triangle(x1, y1, x2, y2, x3, y3)` ... 2次元 X - Y 座標上の 3 点 (x_1, y_1)、(x_2, y_2)、(x_3, y_3) でひとつの三角形を指定します。

プログラム中で `fill()` が呼ばれないときの内部色のデフォルト値は白です。

図 6.18 の緑の境界線を描画したくないときには図 6.19 のようにプログラムを改造します。実行結果は図 6.20 です。図 6.17 のプログラムの `stroke(0, 255, 0)` を線分の描画を行わないことを設定する関数 `noStroke()` へ変更するだけです。

逆に図 6.18 の内部を塗りたくないときには図 6.21 のようにプログラムを改造します。実行結果は図 6.22 です。図 6.17 のプログラムの `fill(255, 0, 0)` を図形の内部を塗らないことを設定する関数 `noFill()` へ変更するだけです。

```
function setup() {
  createCanvas(640, 480);

  noStroke(); // この行を変更
  strokeWeight(10); // 削除してもよい

  fill(255, 0, 0);

  triangle(200, 100, 100, 300, 500, 300);
}
```

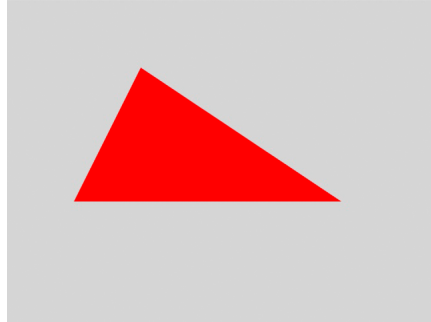


図 6.19: 境界線を描かない三角形の描画プログラム

図 6.20: 左のプログラムの実行結果

```
function setup() {
  createCanvas(640, 480);

  stroke(0, 255, 0);
  strokeWeight(10);

  noFill(); // この行を変更

  triangle(200, 100, 100, 300, 500, 300);
}
```

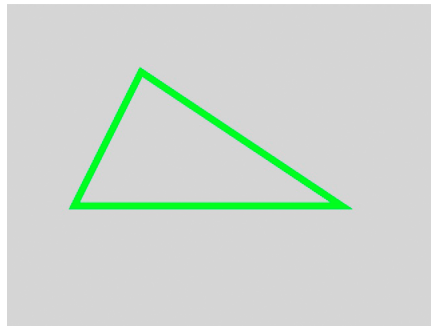


図 6.21: 内部を塗らない三角形の描画プログラム

図 6.22: 左のプログラムの実行結果

- 1 関数 `noStroke()` および `noFill()` の両方を同時に呼び出した場合にはキャンバスには何も描
- 2 画されません。

- 3 **考察** p5.js の関数 `strokeJoin()` の処理内容を調べなさい。

4 6.2.2 様々な形状

5 円弧

- 6 円弧型の図形を描画するには以下の関数を用います。

- 7 `arc(x, y, w, h, startAngle, endAngle)` ... (x,y) に円の中心座標を指定し、w に x 方向（水
- 8 平方方向）の幅、h に y 方向（垂直方向）の幅を指定します。引数 `startAngle` には円弧の開
- 9 始角度、`endAngle` に円弧の終了角度を指定します。

- 10 使用例が図 6.23 のプログラム、実行結果は図 6.24 です。この例題では、やや縦長の円弧で、角度
- 11 が 0 度から 90 度（ラジアンで言えば $\pi/2 = \text{HALF_PI}$ ）までの全体の 1/4 の大きさの円弧を描画し

```
function setup() {
  createCanvas(100, 150);

  arc(50, 75, 80, 120, 0, HALF_PI);
}
```



図 6.23: 縦長の 1/4 の円弧の描画プログラム

図 6.24: 左のプログラムの実行結果

```
function setup() {
  createCanvas(200, 100);

  ellipse(100, 50, 150, 60);
}
```

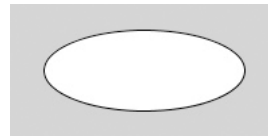


図 6.25: 横長の楕円の描画プログラム

図 6.26: 左のプログラムの実行結果

- 1 ます。
- 2 **考察** 上記関数にさらに引数 `mode` を加えた関数 `arc(x, y, w, h, startAngle, endAngle,`
- 3 `mode)` について調べなさい。

4 楕円

- 5 楕円を描画するには以下の関数を用います。
- 6 `ellipse(x, y, w, h) ...` (x,y) に円の中心座標を指定し、`w` に x 方向（水平方向）の幅、`h` に y
- 7 方向（垂直方向）の幅を指定します。幅は楕円の長径、短径に相当し、長半径（長径の 1/2）、
- 8 短半径（短径の 1/2）ではないことに注意してください。
- 9 使用例が図 6.25 のプログラム、実行結果は図 6.26 です。

10 長方形

- 11 長方形を描画するには以下の関数を用います。
- 12 `rect(x, y, w, h) ...` (x,y) に長方形の左上の点の座標（中心の座標ではありません）を指定し、
- 13 `w` に x 方向（水平方向）の幅、`h` に y 方向（垂直方向）の幅を指定します。
- 14 使用例が図 6.27 のプログラム、実行結果は図 6.28 です。

- 15 **考察** 関数 `quad()` を用いれば、任意の四辺形を描画できます。詳細を調べてください。

```
function setup() {  
  createCanvas(200, 100);  
  
  rect(10, 10, 150, 60);  
}
```

図 6.27: 横長の長方形の描画プログラム



図 6.28: 左のプログラムの実行結果

6.3 CG アニメーション

描画プログラムにパラメータを導入し、そのパラメータを時間と共に連続的に更新させながら描画を繰り返せば、CG アニメーションが可能です。p5.js では CG アニメーションを非常に簡単に実装できます。

この節ではまず p5.js における CG アニメーションの基本技術を解説します。そしてマウス、キーボードを用いたイベント処理について紹介します。

6.3.1 様々な CG アニメーション

アニメーションプログラムの基本形

実は p5.js のプログラムには二つの形態があります。

ひとつは、ここまで紹介してきた CG アニメーションを行わない p5.js のプログラムです。この形のプログラムを p5.js では **static sketch** (静的なスケッチ) と呼んでいます。

もうひとつは、CG アニメーションを行うための p5.js のプログラムで、以下の二つの関数を定義せねばなりません。この形のプログラムを **active sketch** (動的なスケッチ) と呼んでいます。

setup() ... 6.1.1 節でも紹介しましたが、この関数に初期化処理を記述します。プログラムの開始直後にシステムから 1 回だけ呼び出されます。

draw() ... ここに毎回の描画処理を記述します。**setup()** の終了後、デフォルトの設定では **draw()** は 1 秒間に数十回ずつ繰り返し呼び出されます。その回数 (頻度) は個々の PC やプログラムの実行状況に依存します。プログラムを終了するには終了ボタン  を押してください。

左から右へ黒い領域が広がる

図 6.29 は非常に簡単な CG アニメーションのプログラム例です。図 6.30 はその実行途中の様子です。このプログラムでは、以下の処理を行います。

1. 関数 **setup()** では、キャンバスのサイズを 300×200 に設定します。
2. 関数 **draw()** では、**(frameCount, 0)** から **(frameCount, 100)** まで縦に 100 画素の長さの線分を描画します。
3. **frameCount** は自動的に 1 ずつ値が増えていきます。増える時間間隔は PC の種類やプログラムの実行状況によって様々です。これについては後に再度、解説します。

```
function setup() {
  createCanvas(300, 200);

  background(200);
}

function draw() {
  line(frameCount, 0, frameCount, 100);
}
```



図 6.30: 左のプログラムの実行途中の様子

図 6.29: 黒い領域が右側へ広がっていく CG アニメーションのプログラム

```
function setup() {
  createCanvas(300, 200);

  background(200);
  // この1行は削除してもよい
}

function draw() {
  background(200);
  line(frameCount, 0, frameCount, 100);
}
```



図 6.32: 左のプログラムの実行途中の様子

図 6.31: 線分が右へ移動していく CG アニメーションのプログラム

- 1 結果、キャンバス内に (0,0) から (0,100) の線分、(1,0) から (1,100) の線分、(2,0) から (2,100)
- 2 の線分... が順に引かれるため、図 6.30 のように、左から右へ黒い領域がだんだんと広がっていき
- 3 ます。x の値が 300 を超えても描画処理は続きますが、もはや線分はキャンバス外のため、キャン
- 4 バス上の変化は無くなります。

5 左から右へ線分が移動する

- 6 図 6.29 のプログラムでは線分の描画に伴って黒い領域が広がっていきました。というのも、キャン
- 7 バスに過去に描画した内容が消されないまま残っているからです。

- 8 もし過去の (直前の) 描画をクリアし、常に新しい描画像のみを表示したいならば、draw() の
- 9 冒頭で関数 background() を呼び出します。

- 10 図 6.31 は図 6.29 に関数 background() を追加したプログラムです。実行途中の様子が図 6.32

```

function setup() {
  createCanvas(600, 500);
}

function draw() {
  background(200);

  xoffset = 300;
  yoffset = 250;
  r = 200;
  n = 23;
  m = 7; // m の宣言を追加
  incAngle = 360 / n;

  for (i = 0; i < m * n; i = i+m)
  {
    t1 = radians(incAngle * i + frameCount);      // 回転角を加算
    x1 = r * cos(t1) + xoffset;
    y1 = r * sin(t1) + yoffset;

    t2 = radians(incAngle * (i + m) + frameCount); // 回転角を加算
    x2 = r * cos(t2) + xoffset;
    y2 = r * sin(t2) + yoffset;

    line(x1, y1, x2, y2);
  }
}

```

図 6.33: 星が回転する CG アニメーションのプログラム

- 1 1 です。実行では縦の線分が左から右へ移動していく様子が見えるはずです。
- 2 2 さて、上に述べた二つの事項：(1) `setup()` と `draw()` を記述すること、(2) 必要に応じて
- 3 3 `background()` を呼び出すことが p5.js における CG アニメーションの基本です。

4 星が回転する

- 5 CG アニメーションの少し複雑な例として、星が回転するプログラムを図 6.33 に示します。これ
- 6 は図 6.9 を改造したものです。変数 `frameCount` を星の回転角として `draw()` の中の描画に用いま
- 7 す。よって 360 回呼ばれると星は 1 回転します。講義担当者の PC ではデフォルトでは関数 `draw()`
- 8 は 1 秒間に 30 回呼ばれます。このような画像の更新頻度 — これをフレームレート (frame rate)
- 9 とも呼びます — には一般に **FPS** (Frames Per Second、フレーム/秒、fps とも書く) という単位
- 10 を用います。30FPS の描画速度/フレームレートならば、星は 12 秒で 1 回転することになります。

```
function setup() {
  createCanvas(600, 500);
  frameRate(10);           // この1行を追加
}
```

以下、同様

図 6.34: 星がゆっくり回転する CG アニメーションのプログラム

1 星の回転速度を変える

2 p5.js では描画速度の設定を以下の関数で変更できます。

3 `frameRate(fps)` ... FPS 値を引数 `fps` で指定します。複雑な描画を行う場合などには、仮に大き
 4 な FPS 値を指定してもその描画速度を達成できないことがあります。この関数は `setup()`
 5 内から呼ぶことが想定されており、`draw()` から呼んだ場合の効果（実行途中で描画速度を変
 6 えること）は保証されません。

7 図 6.34 は、描画速度を 10FPS で実行するプログラムです。星がゆっくりと回転することを確認
 8 してください。

9 関数 `frameRate()` を呼ばない場合、PC の種類に応じてシステムが自動判定したフレームレ
 10 トで動作します。またキーボード、マウスからの入力を行うと、システムが自動的に（勝手に）フ
 11 レームレートを上げる場合があります。一定のフレームレートが必要な場合には必ず `frameRate()`
 12 を呼んでください。

13 一般にはフレームレートが高いとコンピュータの計算負荷が高まりますが、アニメーションは滑
 14 らかです。逆の場合には計算負荷は減りますが、アニメーションがぎこちない動きになってしまい
 15 ます。どの程度のフレームレートでアニメーションが滑らかに見えるかには個人差があります。

```
function setup(){
  createCanvas(500,500);
}

function draw(){
  print(mouseX,mouseY);
}
```

図 6.35: マウスの位置をコンソールに表示するプログラム

```
function setup(){
  createCanvas(500,500);
}

function draw(){
  rect(mouseX,mouseY,10,10);
}
```

図 6.36: マウスの位置に 10 × 10 の正方形を描画するプログラム

6.3.2 イベント処理、その他

p5.js ではマウスやキーボードからの入力を受け付けることができます。これを利用すれば、対話的 (interactive) なアプリケーション、たとえばビデオゲームを簡単に開発できます。以下ではマウスイベントの取得方法などを紹介し、簡単なビデオゲームを作ってみます。

マウスの位置

キャンバス上のマウスの x 座標値、 y 座標値はシステム定義のグローバル変数 `mouseX`、`mouseY` の値として常にプログラム中のどの位置からでも読み取り可能です。ただし、マウスが実行ウィンドウから出たときには変数の値は更新されません。

図 6.35 は `draw()` が呼び出されたときの位置をコンソールに表示するプログラムです。コンソールはエディタのプログラム・エリアの下 (ウィンドウの左下) にあります。コンソール出力関数 `print()` は引数の値をコンソールに出力する関数です。`draw()` は繰り返し実行されますから、図 6.35 のプログラム実行中はコンソールにマウスの位置が出力され続けます。

これを応用し、`print()` を `rect()` に変更した図 6.36 のプログラムではマウスの位置に 10 × 10 の大きさの正方形を描画します。実際にプログラムを動かしてみましょう。

考察 `mouseX`、`mouseY` に類似したシステム定義の変数として `pmouseX`、`pmouseY` も用意されています。`mouseX`、`mouseY` との違いについて調べなさい。

マウスのクリック

p5.js ではマウスイベントに関する以下の関数で**イベント駆動処理** (event driven processing)⁴が可能です。マウスが実行ウィンドウ内にあるときのみに有効です。

⁴ イベント駆動処理は、アプリケーションフレームワークを用いる最近のプログラム開発の典型的な処理方法です。プログラムは通常、イベント待ち状態にあって、イベントが発生したときに特定の関数が自動的に呼び出されます。

```
function setup() {
  createCanvas(500, 500);
}

function draw() {}

function mousePressed() {
  print(mouseX, mouseY);
}
```

```
function setup() {
  createCanvas(500, 500);
}

function draw() {}

function mousePressed() {
  rect(mouseX, mouseY, 10, 10);
}
```

図 6.37: マウスが押されたときの位置を
コンソールに表示するプログラム

図 6.38: マウスが押されたときの位置に
10 × 10 の正方形を描画するプログラム

1 `mousePressed()` ... マウスボタンが押された時に自動的に呼び出される関数です。押されたボタ
2 ンの種類はシステム定義のグローバル変数 `mouseButton` で知ることができます。また、シ
3 ステム定義の `boolean` 型グローバル変数 `mousePressed` が `true` にセットされます。詳細は
4 各自で調べてください。

5 **考察** `mousePressed` と類似の関数として、`mouseReleased()`、`mouseClicked()`、`mouseMoved()`、
6 `mouseDragged()`、`mouseWheel(MouseEvent event)` も利用可能です。詳細を自ら調べなさい。

7 図 6.37 は、マウスが押された時の位置を `print()` でコンソールに出力するプログラムです。図
8 6.38 は、マウスが押された時の位置に正方形を描画するプログラムです。この二つのプログラム
9 には本体が空の `draw()` が定義されていることに注意してください。この関数定義を削除すること
10 はできません。削除すると、このプログラムは `active sketch` ではなく、`static sketch` と見なされ、
11 アニメーションが発動しないからです。

12 **考察** マウスイベントと同様に、p5.js ではキーボードイベントも使用できます。システム定義のグ
13 ローバル変数 `key`、`keyCode`、`keyPressed`、関数 `keyPressed()`、`keyReleased()`、`keyTyped()`
14 が用意されています。それらの内容を調べなさい。

15 文字列の描画

16 p5.js には、キャンバス（コンソールではありません）に文字列を描画する機能が用意されてい
17 ます。主な関数を以下にまとめます。

18 `text(String str, x, y)` ... 文字列 `str` を座標位置 `(x, y)` に描画します。文字の色は別途 `fill()`
19 で指定します。

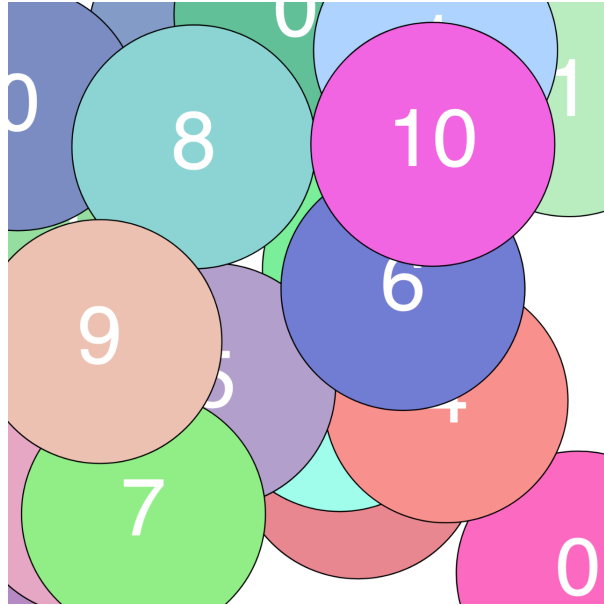


図 6.39: モグラたたき風ゲームの実行の様子

- 1 `textSize(size)` ... 描画する文字の大きさを画素の単位で指定します。
- 2 `textAlign(alignX)` ... 引数に、文字列の左右の位置付けを左詰 (`LEFT`)、中央並び (`CENTER`)、
- 3 右詰 (`RIGHT`) で指定します。デフォルトは左詰です。
- 4 `textAlign(alignX, alignY)` ... 第1引数は上と同じです。第2引数に、文字列の縦の位置付け
- 5 を `TOP`、`BOTTOM`、`CENTER`、`BASELINE` で指定します。これらの違いは自ら調べてください。
- 6 デフォルトは `BASELINE` です。
- 7 `textLeading(leading)` ... 文字列が改行コード '`\n`' を含み、描画が複数行に渡る場合の行間を
- 8 画素の単位で指定します。デフォルトでは `textSize()` で指定した値 (すなわち行間は1行)
- 9 です。
- 10 より凝った描画も可能です。詳細は自ら調べてみてください。描画例は次のゲームの例で紹介し
- 11 ます。

12 6.3.3 モグラたたき風ゲームの開発

- 13 ここまで述べた内容を基にして講義担当者が図 6.40 のような簡単なゲームを作ってみました。
- 14 このゲームでは一定時間毎にキャンバス上のランダムな位置に円が現れます。その円の中にマウ
- 15 スを移動させてクリックすると1点ゲットできます。
- 16 このプログラムに、まだ解説していない以下の機能を用いています。

```

function setup() {
  createCanvas(500, 500);

  background(255);           // ゲーム開始直後にキャンバスをクリア
  textAlign(CENTER, CENTER); // 文字列を縦横、指定位置の中央に描画する。
  frameRate(45);             // 1秒間に45コマ描画
}

diameter = 200;              // 円の直径
interval = 20;               // 円の表示間隔をフレーム数で指定
x = 0;
y = 0;                       // 円の中心位置
score = 0;                   // 得点

function draw() {
  if (frameCount % interval == 0) { // 次の円を描画するタイミングならば

    fill(random(100,255),      // R、G、Bの輝度値をそれぞれ乱数で求めるが
          random(100,255),      // パステル調の明るい色になるように調整
          random(100,255));

    x = random(500);           // x座標値を0~500の乱数で求める。
    y = random(500);           // y座標値を0~500の乱数で求める。
    ellipse(x, y, diameter, diameter); // 円を描画する。

    textSize(diameter/3);      // 文字サイズをdiameter/3に
    fill(255);                 // 文字を白色で
    text(score, x, y);         // 経過得点を表示
  }
}

function mousePressed() {
  d = dist(mouseX, mouseY, x, y); // マウスの位置と円の中心の距離
  if (d < diameter / 2) score = score+1;
}

```

図 6.40: 図 6.39 のゲームのプログラム

- 1 `random(high)` ... 0 以上、`high` よりも小さい範囲の一樣乱数値を求めます。
- 2 `random(low, high)` ... `low` 以上、`high` よりも小さい範囲の一樣乱数値を求めます。
- 3 `dist(x1, y1, x2, y2)` ... 二つの座標値 (`x1,y1`) と (`x2,y2`) の距離を計算します。同等の計算は
4
$$\text{sqrt}((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2))$$
- 5 でも可能ですが、一般に組み込み関数を用いた方が高速な計算が期待できます。
- 6 他にも、三角関数、対数／指数関数、天井関数、床関数などが利用可能です。
- 7 残念ながら、プログラムを簡単にするため、このゲームは永遠に終了しません。
- 8 **考察** 円が 10 個表示されたならばゲームが終了するように、プログラムを改造しなさい。

1 第7章 p5.js プログラミング（計算編）

2 この章では計算に特化したプログラムを紹介します。コンピュータによる計算の奥深さが感じら
3 れるでしょうか。

4 7.1 平方根の計算

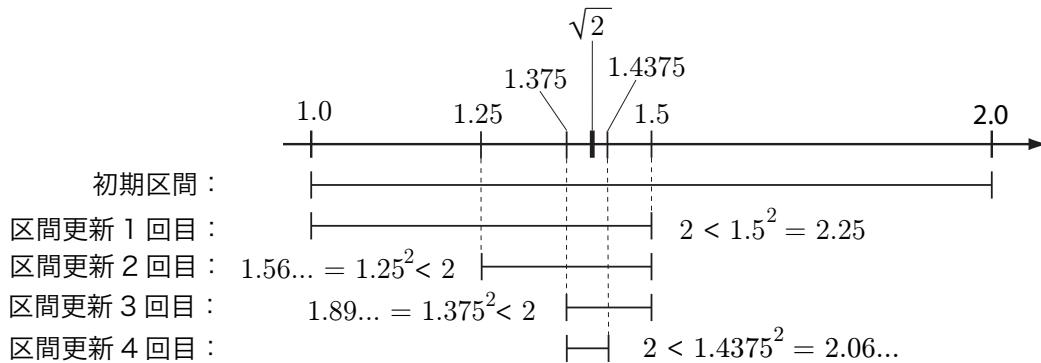
5 平方根の計算には様々な方法が知られています。筆算では開平法と呼ばれる計算法が知られてい
6 ますが、コンピュータでは通常、開平法は用いません。コンピュータは人手に比べ計算が圧倒的に
7 高速なため、その特徴を活かした力技で計算します。開平法のような、ある意味、チマチマした手
8 法は人手の計算では有効ですが、コンピュータには（可能ですが）適さないのです。ここでは三つ
9 の方法を紹介します。

10 (1) 二分法

11 今、2 の平方根 $\sqrt{2}$ を求めることを考えましょう（図 7.1 参照）。 $\sqrt{2}$ が区間 $[1, 2]$ にあることは
12 明らかです。そこで、 $\sqrt{2}$ がこの区間中のどの辺りにあるかを粗く知るために、この区間の中間点
13 1.5 についてその二乗を計算します。 $1.5^2 = 2.25 > 2$ ですから、平方根の単調性から、 $1.5 > \sqrt{2}$
14 が成り立ちます。よって区間 $[1, 2]$ を前半部 $[1, 1.5]$ と後半部 $[1.5, 2]$ の二つの区間に分けるなら
15 ば、求める平方根は区間 $[1, 1.5]$ にあるはずです。次に区間 $[1, 1.5]$ の中間点 1.25 を考え、その二
16 乗を計算します。 $1.25^2 = 1.5625 < 2$ ですから $1.25 < \sqrt{2}$ が成り立ち、 $\sqrt{2}$ は区間 $[1.25, 1.5]$ にあ
17 ることが分かります。そこで、次に区間 $[1.25, 1.5]$ の中間点 1.375 を考え、その二乗を計算し... と
18 続けて行けば、 $\sqrt{2}$ が存在する区間の幅は 1 回の計算で半分になり、求める値 $\sqrt{2} = 1.41421356...$
19 を絞り込んでいきます。

20 この方法を手順として述べると以下の通りです。

- 21 1. 初期の区間を定め、
- 22 2. 区間の中間値を求め、
- 23 3. 中間値の二乗を求め、
- 24 4. 区間を更新します。

図 7.1: 二分法によって $\sqrt{2}$ を含む区間が狭められていく過程

- 1 5. 上の 1. から 4. を一定回数だけ繰り返します。
- 2 p5.js で計算するために、上の手順をさらに詳細に設計してみましょう。
- 3 1. まず二つの変数 `minv` と `maxv` を用いて区間 `[minv, maxv]` を表すと約束します。初期値は
- 4 `minv=1, maxv=2` とします。
- 5 2. 区間の中間値と中間値の二乗を変数 `midv` と `midvsq` に格納します。
- 6 3. もし `midvsq` の値が 2 よりも小さいならば、中間値 `midv` を新たな `minv` とします。さもな
- 7 くば `midv` を新たな `maxv` とします¹。
- 8 4. 上の処理を 30 回繰り返すこととします。

9 プログラムは図 7.2 の通りです。なお、`min`、`max` は p5.js の最小値、最大値を求める関数の名前

10 のため、そのままでは使用できません。そこで `min`、`max`、`mid` の代わりに `minv`、`maxv`、`midv` を

11 用いることとします。このプログラムを実際に動かしてみると、以下のような出力になります。

```

12 1 1.5
13 2 1.25
14 3 1.375
15 4 1.4375
16 5 1.40625
17 6 1.421875
18 7 1.4140625
19 8 1.41796875
20 9 1.416015625
21 10 1.4150390625
22 11 1.41455078125
23 12 1.414306640625
24 13 1.4141845703125
25 14 1.41424560546875

```

¹平方根 $\sqrt{2}$ は無限小数なので、`midvsq` が 2 に等しくなることはありません。よって「`midvsq` が 2 に等しいときには...

```

function setup() {
  minv = 1;
  maxv = 2;
  for (i = 1; i <= 30; i = i+1) {
    midv = (minv+maxv)/2;
    print(i,midv);           // 計算途中の値を出力
    midvsq = midv*midv;
    if (midvsq < 2) {
      minv = midv;
    } else {
      maxv = midv;
    }
  }
  print((minv+maxv)/2);      // 30 回繰り返した後の計算結果を出力
}

```

図 7.2: $\sqrt{2}$ を二分法で計算するプログラム

```

1  15 1.414215087890625
2  16 1.4141998291015625
3  17 1.4142074584960938
4  18 1.4142112731933594
5  19 1.4142131805419922
6  20 1.4142141342163086
7  21 1.4142136573791504
8  22 1.4142134189605713
9  23 1.4142135381698608
10 24 1.4142135977745056
11 25 1.4142135679721832
12 26 1.414213553071022
13 27 1.4142135605216026
14 28 1.414213564246893
15 29 1.4142135623842478
16 30 1.4142135614529252
17 1.4142135623696959

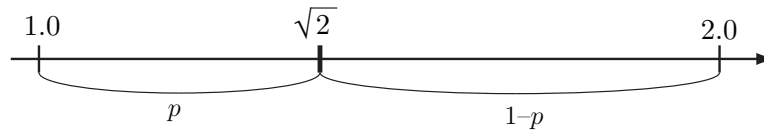
```

18 中間値が真値 $\sqrt{2} = 1.414213562373095048801688\dots$ に近付いている様子が分かります。二分法で
 19 は 1 回毎に区間が $1/2$ になっていきます。よって n 回繰り返すことで中間値と真値の誤差は $(1/2)^n$
 20 以下になります。

21 **考察** 図 7.2 のプログラムを改造して、2 の立方根 $\sqrt[3]{2}$ を求めるプログラムを作りなさい。

22 (2) モンテカルロ法

23 モンテカルロはカジノで有名な街ですが、カジノでの必勝法の研究からモンテカルロ法と呼ばれ
 24 ている、おもしろい計算手法が考え出されました。

図 7.3: 区間 $[1, 2]$ の中のランダムな数 x が $x^2 < 2$ となる確率 p

```
function setup() {
  m = 10000;
  n = 0;
  for (i = 0; i < m; i = i+1) {
    r = 1+random();
    if (r*r < 2) {
      n = n+1;
    }
  }
  print(n/m+1);
}
```

図 7.4: $\sqrt{2}$ をモンテカルロ法で計算するプログラム

1 今、区間 $[1, 2]$ の中の適当な数 x をひとつランダム (random) に選んで (たとえば 1.2498、1.342、
 2 1.98345、.... などの任意の数を選んで)、その数の二乗 x^2 を計算します。このとき、 x^2 の値が 2
 3 よりも小さい確率は $p = \sqrt{2} - 1$ です。何故ならば、区間 $[1, \sqrt{2}]$ の中の数は二乗しても 2 以下で
 4 す。区間 $[\sqrt{2}, 2]$ の中の数は二乗すると 2 以上です。ランダムに選ぶ数が区間 $[1, 2]$ に一様な確率
 5 で選ばれるならば、数が $[1, \sqrt{2}]$ に含まれる確率と $[\sqrt{2}, 2]$ に含まれる確率は区間の幅に比例しま
 6 す (図 7.3 参照)。よって $p = \sqrt{2} - 1$ となるはずですが、これを变形すると $p + 1 = \sqrt{2}$ です。つま
 7 り、 x^2 の値が 2 よりも小さい確率を求めれば、 $\sqrt{2}$ を知ることができます。

8 このことに注目し、実際に区間 $[1, 2]$ の中から M 個の数をランダムに求め、その二乗が 2 より
 9 も小さい回数 N を求めます。そうすると確率 p は N/M です。そして $p + 1 = N/M + 1 = \sqrt{2}$
 10 が成り立ちます。

11 このように、ランダムな数を用いて確率的な計算を行う方法を一般にモンテカルロ法と呼んでい
 12 ます。この種の方法は正確な数値を高速に求めることには不向きですが、考え方が非常に単純かつ
 13 素朴なため、効果的な計算方法がない複雑な数値の計算や様々な事象のシミュレーションに有効で
 14 す²。

15 プログラムは図 7.4 の通りです。関数 random() は、区間 $[0, 1]$ の一様乱数を求める p5.js の関数
 16 です。ここでは 1 万個の乱数を生成しています。

²たとえば素粒子の質量を理論的に計算するためにモンテカルロ法が利用されています。人工知能の学習にも利用されています。

このプログラムを実際に動かしてみました。結果は以下の通りです。乱数を用いた実行のため、得られる数値の下から数桁分は実行の度に異なった値になることを注意しておきます。

1.4077

一万回繰り返したにも関わらず、真値 $\sqrt{2} = 1.41421356\dots$ から大きな誤差があります。参考のために、プログラムの `m` の値を 100000、1000000、10000000 に変えて実行した結果は以下の通りです。

1.41521 ... 100000 回の場合

1.414217 ... 1000000 回の場合

1.4141155 ... 10000000 回の場合

試行回数と共に少しずつ真値へ近づいていることが分かります。

モンテカルロ法では試行回数 n に対して誤差が $1/\sqrt{n}$ になることが知られており、繰り返し回数を 100 倍しても誤差は $1/\sqrt{100} = 1/10$ にしかなりません。二分法 (1 回で誤差が $1/2$ になる方法) と比較すると明らかに非効率です。このため、モンテカルロ法の応用範囲は限られています。しかし、効果的な計算方法が見つからない問題を解くための最後の砦です。

考察 図 7.4 のプログラムを改造して、2 の立方根 $\sqrt[3]{2}$ を求めるプログラムを作りなさい。

(3) ニュートン・ラフソン法

二分法はモンテカルロ法よりも少ない繰り返し回数で高い精度が得られる計算方法でした。しかし、ここで紹介するニュートン・ラフソン法 (あるいは単にニュートン法) は、二分法よりもさらに高速に平方根を求めることのできる方法です。この方法は平方根を計算する最も実用的な方法として知られ、実際に広く使用されています。

ニュートン・ラフソン法で $\sqrt{2}$ を計算する場合、二次方程式 $y = x^2 - 2$ の、 $y = 0$ のときの解 (つまり $x = \sqrt{2}$) を求めることを考えます。

図を用いて説明しましょう。図 7.5 は $y = x^2 - 2$ のグラフです。グラフと x 軸の交点座標は $(\sqrt{2}, 0)$ ですから、この交点を計算で求めることができれば $\sqrt{2}$ の値を知ることができます。

今、 x 軸上の点 $(x_n, 0)$ を任意にひとつ与えます。ただし、 $x_n > \sqrt{2}$ となる点、たとえば $x_n = 2$ とします。この点から垂直に直線を伸ばし、グラフ $y = x^2 - 2$ と交わる点 (x_n, y_n) を求めます。ここに、 $y_n = x_n^2 - 2$ を満たすことに注意します。次に、 (x_n, y_n) を通り、グラフ $y = x^2 - 2$ と接するような直線を $y = ax + b$ と表わすとき、 $a = 2x_n$ 、 $b = -x_n^2 - 2$ が成り立ちます³。この直線 $y = ax + b$ が x 軸と交わる点を $(x_{n+1}, 0)$ とするとき、 $x_{n+1} = -b/a$ が成り立ち、すなわち

$$x_{n+1} = \frac{x_n}{2} + \frac{1}{x_n}$$

³ $y = ax + b$ は点 (x_n, y_n) を通過し、かつ $y = ax + b$ の傾き a は $x = x_n$ において $dy/dx = d(x^2 - 2)/dx = 2x$ に等しく、かつ $y_n = x_n^2 - 2$ であるという条件から a と b を具体的に求めることができます。

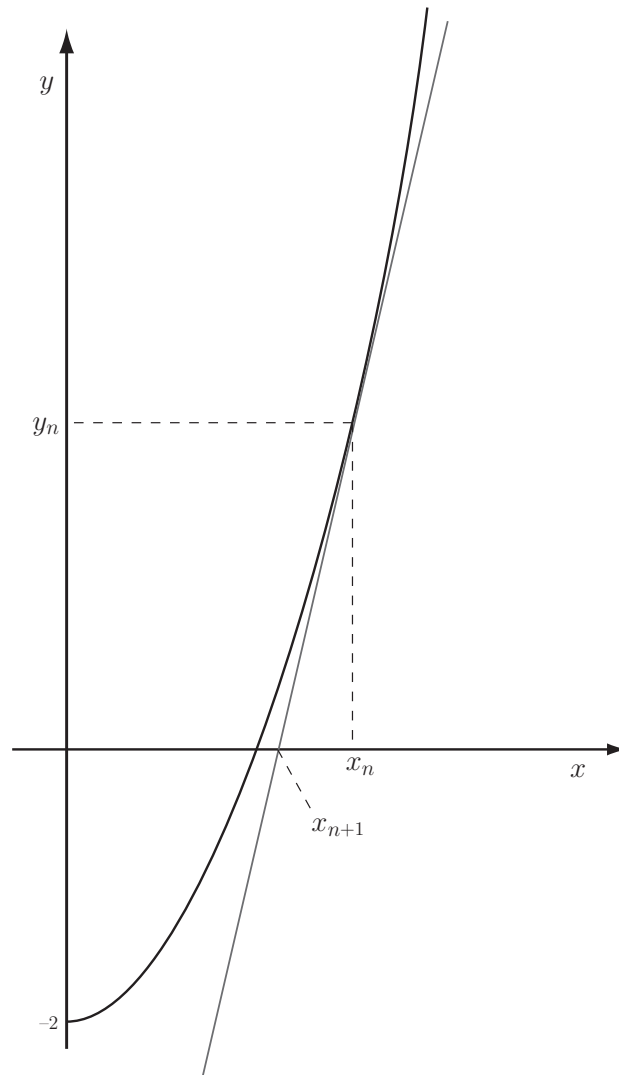


図 7.5: ニュートン・ラフソン法の1ステップ

が成り立ちます。グラフの性質（図 7.5 参照）から、明らかに $x_n > x_{n+1} > \sqrt{2}$ が成り立ちます。よって、この計算を繰り返すことで、

$$x_n > x_{n+1} > x_{n+2} > x_{n+3} > \dots > \sqrt{2}$$

- 1 と、 x 軸との交点の x 座標値は限りなく $\sqrt{2}$ に近づいていくことになるでしょう。

上の考察をまとめます。漸化式：

$$\begin{aligned} x_0 &= 2, \\ x_{n+1} &= \frac{x_n}{2} + \frac{1}{x_n} \end{aligned}$$

- 2 を用いて、数列の値を $x_0, x_1, x_2, x_3, \dots$ と計算していくと、値は急速に（二分法よりもさらに急

```

function setup() {
  x = 2;
  for (i = 1; i <= 10; i = i+1) {
    print(i,x);          // 計算途中の値を出力
    x = x/2+1/x;
  }
  print(x);              // 10 回繰り返した後の計算結果を出力
}

```

図 7.6: $\sqrt{2}$ をニュートン・ラフソン法で計算するプログラム

速に) $\sqrt{2}$ に近づいていきます。なお、 x_0 の値は $\sqrt{2}$ よりも大きい任意の数値でよいのですが、明らかに $\sqrt{2}$ よりも大きい 2 を与えるのが単純かつ安全です。

図 7.6 はこの計算を行う p5.js のプログラムです。以下はその出力です。

```

1 2
2 1.5
3 1.4166666666666665
4 1.4142156862745097
5 1.4142135623746899
6 1.414213562373095
7 1.414213562373095
8 1.414213562373095
9 1.414213562373095
10 1.414213562373095

```

5 回目で 10 桁以上の高い計算精度を得ています。

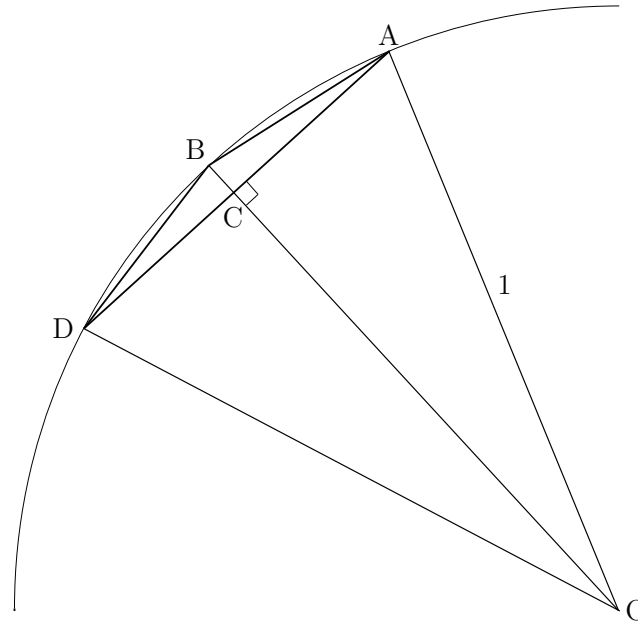
考察 2 の立方根 $\sqrt[3]{2}$ を求めるニュートン・ラフソン法のプログラムを作りなさい。

さて、この節では平方根の 3 種類の計算方法を紹介しました。ひとつの問題が与えられたとき、それを解く方法はひとつとは限りません。プログラマは様々な方法の特徴を理解し、それらを状況に応じて使い分けねばなりません。

7.2 円周率の計算

二つ目の例題として円周率の計算方法を紹介します。

円周率は任意の円の円周の長さをその円の直径の長さで割った数と定義されますが、平方根同様に様々な計算方法が知られています。

図 7.7: 正 n 角形と正 $2n$ 角形の関係

1 (1) n 角形と $2n$ 角形の漸化式

2 まず、古典的な計算方法として円に内接する正 n 角形を考えましょう。正 n 角形の外周の長さ
3 を円の直径で割った数は、 n が大きくなるに従い、円周率に近づいて行きます。

4 正 n 角形の外周の長さ L_n が分かっているならば、正 $2n$ 角形の外周の長さ L_{2n} は L_n から比
5 較的に容易に計算できることが知られています。この関係式を導いてみましょう。

今、半径 1 の円に内接する正 n 角形のひとつの辺が図 7.7 の線分 AD であると仮定します。そう
すると、AD の中間点 C を通る直線は辺を正確に二分しますから、線分 AB、BD は正 $2n$ 角形の
二つの辺です。ここで $\angle ACO$ は直角ですから、三平方の定理から以下の式が成り立ちます。

$$|AC|^2 + |CO|^2 = 1$$

$$|AC|^2 + |BC|^2 = |AB|^2$$

この 2 式を以下のように変形します。

$$|CO| = \sqrt{1 - |AC|^2} \quad (7.1)$$

$$|BC| = \sqrt{|AB|^2 - |AC|^2} \quad (7.2)$$

以下は明らかに成り立ちます。

$$|BC| + |CO| = 1$$

```
function setup() {
  L = 6;
  for (n = 6; n < 10000000; n = n*2) {
    print(n,L/2);
    L = 2*n*sqrt(2-sqrt(4-(L/n)*(L/n)))
  }
}
```

図 7.8: π を多角形近似で計算するプログラム

この $|BC|$ 、 $|CO|$ に式 (7.1)、式 (7.2) を代入すると以下の通りです。

$$\sqrt{|AB|^2 - |AC|^2} + \sqrt{1 - |AC|^2} = 1$$

この式を $|AB|$ を左辺に置くように変形すると以下の式を得ます。

$$|AB| = \sqrt{2 - 2\sqrt{1 - |AC|^2}}$$

この式に、 $|AC| = |AD|/2$ を代入すると以下の式を得ます。

$$|AB| = \sqrt{2 - \sqrt{4 - |AD|^2}}$$

さて、正 n 角形の外周の長さ L_n は $n|AD|$ 、正 $2n$ 角形の外周の長さ L_{2n} は $2n|AB|$ です。これを上の式へ代入すると

$$L_{2n} = 2n\sqrt{2 - \sqrt{4 - (L_n/n)^2}} \quad (7.3)$$

- 1 ところで、半径 1 の円に内接する正 6 角形について $L_6 = 6$ が成り立ちます。そこで上の式 (7.3)
- 2 を用いて $L_6 = 6$ から出発し、 L_{12} 、 L_{24} 、... を求めて行けば、次第に 2π に近づいて行くはずで
- 3 す。そして最後にその値を 2 で割れば、円周率が求められます。
- 4 図 7.8 は、この方法をプログラム化したものです。以下は、その実行結果をまとめたものです。

5 正 12 角形	3.10582854123025
6 正 24 角形	3.13262861328124
7 正 48 角形	3.13935020304687
8 正 96 角形	3.14103195089053
9 正 192 角形	3.14145247228534
10 正 384 角形	3.14155760791162
11 正 768 角形	3.14158389214894
12 正 1536 角形	3.14159046323676
13 正 3072 角形	3.14159210604305
14 正 6144 角形	3.14159251658815
15 正 12288 角形	3.14159261864079
16 正 24576 角形	3.14159264532122

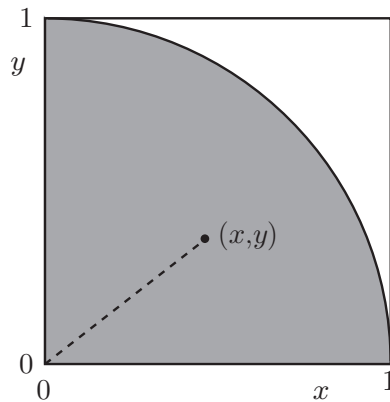


図 7.9: 正方形の面積と 4 分円の面積

```

1  正 49152 角形      3.14159264532122
2  正 98304 角形      3.14159264532122
3  正 196608 角形     3.14159264532122
4  正 393216 角形     3.14159366984943
5  正 786432 角形     3.14159230381174
6  正 1572864 角形    3.1416086962248
7  正 3145728 角形    3.14158683965504
8  正 6291456 角形    3.14167426502176
9  ...

```

円周率の真値 3.141592653589793238462... と比較してみてください。正 24576 形から正 196608 角形のときに 3.1415926 まで正しい計算ができています。しかし、辺の数がそれよりも増えると、正しい値からはだんだん離れていることが分かります。これは計算誤差に起因します。計算誤差が生じる理由については後章で解説します。

(2) モンテカルロ法

平方根の計算と同様に、円周率の計算にもモンテカルロ法が適用できます。基本的なアイデアとして、図 7.9 の正方形の面積と 4 分円（4 分割した円）の面積の比を利用することです。

今、図 7.9 の正方形の 1 辺の長さを 1 とします。よって、その正方形の面積は 1 です。次に図 7.9 の 4 分円の面積は半径 1 の円の面積の 1/4 です。よって $\pi/4$ です。ここで図 7.4 のプログラムで用いた `random()` を使って、 $0 \leq x \leq 1$ 、 $0 \leq y \leq 1$ の範囲の矩形領域の中の点 (x, y) をランダムに求めたとします。そうすると、その点が 4 分円の中に含まれる確率 ($x^2 + y^2 \leq 1$ となる確率) は

$$\frac{\text{4 分円の面積}}{\text{正方形の面積}} = \frac{\pi/4}{1} = \frac{\pi}{4}$$

となります。よって、この確率を 4 倍すれば、円周率が求められるはずです。

これを実行するプログラムは図 7.10 の通りです。以下はその結果です。

```
function setup() {
  m = 1000;
  n = 0;
  for (i = 0; i < m; i = i+1) {
    x = random();
    y = random();
    if (x*x+y*y < 1) {
      n = n+1;
    }
  }
  print(4*n/m)
}
```

図 7.10: π をモンテカルロ法で計算するプログラム

```
1  3.072      ... 1000 回の場合
2  3.1372     ... 10000 回の場合
3  3.13748    ... 100000 回の場合
4  3.142348   ... 1000000 回の場合
5  3.1417712  ... 10000000 回の場合
6  3.14157832 ... 100000000 回の場合
```

7 モンテカルロ法が実用に堪えないことは既に平方根の計算で見た通りですが、素朴に円周率を計
8 算できる点で興味深い手法です。

9 (3) 逆三角関数の公式

10 円周率を多数桁計算する方法として三角関数の性質を用いる方法が知られています。

まず、以下の式が成り立つことを思い出しましょう。

$$\tan\left(\frac{\pi}{4}\right) = 1$$

上式を変形すると

$$\arctan(1) = \frac{\pi}{4}$$

11 そこで、 $\arctan(1)$ を何らかの方法で計算し、求められた値を 4 倍すればそれが円周率です。

ここでテイラー展開を用いると、逆正接関数 $\arctan(x)$ は以下のように多項式近似できます。

$$\arctan(x) = \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} x^{2i+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \dots$$

特に $x = 1$ のときには、

$$\arctan(1) = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$$

```
function setup() {
  sum = 0
  for (i = 0; i < 10; i = i+1) {
    sum = sum + (-1)**i/(2*i+1);
    print(i, 4*sum);
  }
}
```

図 7.11: π をグレゴリー級数で計算するプログラム

- 1 となります。この式をグレゴリー級数と呼びます⁴。この式を愚直に計算すれば円周率に辿り着け
 2 るはずです。上の式は $i = 0, \dots, \infty$ の無限級数和ですが、実際には有限の計算しかできません。そ
 3 こで、 $i = 0, \dots, N$ までの和 $\sum_{i=0}^N (-1)^i/(2i+1)$ を順次求めてみます。図 7.11 は、 $N = 14$ とし
 4 て級数和を求める p5.js プログラムです。
 5 ここに、 $a**i$ は累乗数 a^i を表します。
 6 プログラムの実行結果は以下の通りです。

```
7      0 4
8      1 2.666666666666667
9      2 3.466666666666667
10     3 2.8952380952380956
11     4 3.3396825396825403
12     5 2.9760461760461765
13     6 3.2837384837384844
14     7 3.017071817071818
15     8 3.2523659347188767
16     9 3.0418396189294032
```

- 17 値が増えたり、減ったりを繰り返しながら、円周率に近づいていきますが、しかし、近く速度は決
 18 して速いとは言えず、じれったく感じるかもしれません。というのも、級数和 $\sum_{i=0}^N (-1)^i/(2i+1)$
 19 の値は N の増加と共に加算と減算を交互に繰り返し、数値が真値 ($= \pi/4$) を挟んで上下に激しく
 20 振れて、なかなか収束しないためです。

上の式の問題点を改良すべく、現在では単純なテイラー展開式ではなく、より計算に適した公式を用いています。たとえば以下は 18 世紀に発見された公式 (マチンの公式) です。

$$4 \arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right) = \frac{\pi}{4}$$

この公式の左辺各項をテイラー展開で置き換えると以下の式を得ます。

$$4 \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} \left(\frac{1}{5}\right)^{2i+1} - \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} \left(\frac{1}{239}\right)^{2i+1} = \frac{\pi}{4}$$

⁴驚くべきことに、この級数は既に 15 世紀には知られていたようです。

```
function setup() {
  a = 1;
  b = 1/sqrt(2);
  t = 1/4;
  p = 1;
  for (i = 1; i < 10; i = i+1) {
    a2 = (a+b)/2;
    b2 = sqrt(a*b);
    t2 = t-p*(a-a2)*(a-a2);
    p2 = 2*p;

    a = a2;
    b = b2;
    t = t2;
    p = p2;

    print(i, (a+b)*(a+b)/(4*t));
  }
}
```

図 7.12: π をガウス＝ルジャンドルの反復法で計算するプログラム

さらにこれを有限級数和に置き換えた (∞ を N に置き換えた) 式は以下の通りです。

$$4 \sum_{i=0}^N \frac{(-1)^i}{2i+1} \left(\frac{1}{5}\right)^{2i+1} - \sum_{i=0}^N \frac{(-1)^i}{2i+1} \left(\frac{1}{239}\right)^{2i+1} \approx \frac{\pi}{4}$$

- 1 詳しくは述べませんが、この式を用いた計算方法では N と共に急速に円周率に近づくことが知ら
- 2 れています。

マチンの公式を用いた場合よりもさらに高速に円周率を計算するには、以下の公式を用いることができます。

$$12 \arctan\left(\frac{1}{49}\right) + 32 \arctan\left(\frac{1}{57}\right) - 5 \arctan\left(\frac{1}{239}\right) + 12 \arctan\left(\frac{1}{110443}\right) = \frac{\pi}{4}$$

- 3 これは日本人 (高野喜久雄) によって発見された公式です。この公式は日本で円周率が 10 億桁計
- 4 算された (1990 年当時の世界記録) ときに用いられました。

5 (4) ガウス＝ルジャンドルの反復計算

- 6 4 番目に述べる方法は、最近、しばしば用いられるものです。計算方法は、まず四つの数列 a_n 、
- 7 b_n 、 t_n 、 p_n について以下の初期値：

$$\begin{aligned}
 a_1 &= 1, \\
 b_1 &= \frac{1}{\sqrt{2}}, \\
 t_1 &= \frac{1}{4}, \\
 p_1 &= 1
 \end{aligned}$$

と漸化式 ($n \geq 1$):

$$\begin{aligned}
 a_{n+1} &= \frac{a_n + b_n}{2}, \\
 b_{n+1} &= \sqrt{a_n \cdot b_n}, \\
 t_{n+1} &= t_n - p_n \cdot (a_n - a_{n+1})^2, \\
 p_{n+1} &= 2 \cdot p_n
 \end{aligned}$$

- 1 で反復計算します。そして円周率の値を以下の式で求めます。

$$\pi \approx \frac{(a_n + b_n)^2}{4 \cdot t_n}$$

- 2 この計算方法の理論的背景は複雑であるため、ここでは述べません。興味のある人は、たとえば

3 <https://ja.wikipedia.org/wiki/ガウス＝ルジャンドルのアルゴリズム>

- 4 などを参考にしてください。この方式は、計算手順が単純であり、また計算効率が良い（真値への
5 収束が速い）ことが特長です。

- 6 図 7.12 は、この方法に基づく円周率を計算する p5.js プログラムです。以下はその実行例です。

```

7
8 1 3.1405792505221686
9 2 3.141592646213543
10 3 3.141592653589794
11 4 3.141592653589794
12 5 3.141592653589794
13 6 3.141592653589794
14 7 3.141592653589794
15 8 3.141592653589794
16 9 3.141592653589794

```

- 17 ここまで示してきた実行結果の中で最も急速に真値へ近づくことが分かります。

第8章 アルゴリズムとプログラム

プログラムというものがどのようなものか、ここまで p5.js のプログラミングを通して簡単に紹介しました。これまでコンピュータやプログラミングのことを全く知らなかった人もここまでの講義内容を通して、それらがぼんやりと見えてきたのではないのでしょうか。それらの知識を踏まえ、この章では、個別のプログラミングではなく、プログラム開発全般について紹介します。

8.1 プログラム開発手順

一般に、プログラムを作る作業は次のように進みます。

(A) プログラム化すべき問題が与えられる。

(B) 問題を分析し、処理内容の大きな流れ、基本構造を決める。

(C) プログラミングと直接関係しない、比較的抽象度の高いレベルで処理内容を詰める。

(D) 特定のプログラミング言語、プロセッサ向きにプログラムを作るための前準備/設計を行う。

(E) 実際にプログラムを書き下す。

(F) プログラムの間違いを修正する。

たとえば平方根を求めるという問題が与えられたとしましょう。これが上の (A) の段階です。

通常、問題が与えられた直後にいきなりプログラムを書くことはありません。まず、どのような方法で平方根を求めるか、基本となる数式やアイディアは何か、考えをまとめていきます。前節では平方根の解法として 3 種類を紹介しました。そのいずれを利用するか、あるいは別の方法を用いるか、各解法の特徴を考慮して決定します。これが (B) の段階です。

そして、そのアイディアに致命的な欠陥はないか等に注意しながら、さらに条件判定式やプログラムの分岐構造、ループ構造を固めていきます。これが (C) の段階です。この段階ではまだ特定のコンピュータを想定している訳ではなく 5 章で述べたプログラムの四大原理の下に一般的な処理手順を完成させます。この処理手順のことを一般に**アルゴリズム** (algorithm) と呼びます。問題が簡単な場合には、プログラマは自分の頭の中に無意識の内にアルゴリズムを作り上げるかもしれませんが、少し複雑な場合には、簡単なメモを取るかもしれません。しかし、より複雑な場合にはフローチャートを作り、記憶領域の使用表を作り、手順を明確な形にまとめていくのが一般的です。

1 アルゴリズムが定まったならば、それを特定のプログラミング言語やプロセッサ上に実装（実
2 現）するために、その言語やプロセッサに固有の設計を行います。たとえば7.1節(1)では、変数
3 名、変数の使い方を決めました。これが(D)に相当します。

4 そして最後にプログラミングを行います。これが(E)です。プログラムを書き下しても、それ
5 で終わりではありません。ほとんどの場合、最初に書いたプログラムには間違い — これを**バグ**
6 (bug)¹と呼びます — があります。そこで、プログラムを実際に何度も実行し、バグをひとつひとつ
7 取り除いていかねばなりません。これが(F)です。

8 上の(B)から(F)は明瞭に区別されている訳ではありません。通常、(A)側をプログラム開発の
9 上流、(F)側を下流と呼びます。上流(A)から下流(F)へ向かう途中で不都合を発見したら、逆戻
10 りして設計し直すことも必要です。そのような作業を繰り返して、プログラムは形になっていき
11 ます。

12 8.2 アルゴリズムとは

13 既に述べたように、アルゴリズムとは処理手順のことです。「計算法」「算法」と言ってもよいで
14 しょう。必ずしも厳密な定義がある訳ではなく、気軽に使われていることも事実です。

15 アルゴリズム（＝処理手順）の条件を取って厳密に定義すると、以下のようになります。

- 16 1. 手順通りに実行すれば、正しい答えを求めることができる。
- 17 2. 手順にあいまいさが無い。また個々人の経験や感性に応じて手順の解釈が異なるということ
18 がない。
- 19 3. 手順の実行は、必ず有限回の操作で終了/停止する。無限に実行し続けることはない。
- 20 4. 手順は、特定のコンピュータ、プロセッサ、プログラミング言語を想定して作られていない。

21 1. は当然成り立つべき事項です。

22 2. では、たとえば「どちらの手順を実行してもよい」というあいまいさは許されません。また
23 「スマートと思う手順を実行しなさい」「おもしろい方の手順を実行しなさい」というのも許されま
24 せん。何がスマートか、何がおもしろいかは個人の趣味・趣向に依るからです。次に実行すべき事
25 項は必ず客観的に明確に定まっていなければなりません。

26 3. の条件は重要です²。計算が永遠に終わらないかもしれないということは非常に不都合です。
27 アルゴリズムを作るときには、必ず有限停止することを保証すべきです。

¹英単語 bug とは虫のことです。特にゴキブリやシラミなどのありがたたくない虫を指します。プログラムのバグとは、プログラムに潜む、虫酸の走るような間違いのことなのです。

²1.～4.の中で3.のみ成り立たない手順を「セミ・アルゴリズム」と呼ぶことがあります。たとえば、論理学のある種の命題をコンピュータで自動的に証明する場合に、正しい命題は有限回の手順で自動的に証明できるが、間違った命題は永遠に証明を終えることができない場合があります。世の中には3.が成り立たない手順が多数存在することが分かっています。

```

int main() {
    double minv = 1.0;
    double maxv = 2.0;
    for (int i = 0; i < 30; i = i+1) {
        double midv = (minv+maxv)/2.0;
        cout << i << " " << midv << endl;
        if (midvsq < 2.0) {
            minv = midv;
        } else {
            maxv = midv;
        }
    }
    cout << (minv+maxv)/2.0 << endl;
}

```

図 8.1: $\sqrt{2}$ を二分法で計算するプログラム (C++版)

4. の条件は、アルゴリズムとプログラムを区別する重要な条件です。
- プログラムはコンピュータ上で実際に実行可能な手順であり、その意味で重要です。これに対してアルゴリズムは上記 (B)、(C) の段階で作られる中間生産物に過ぎません。しかし実は、中間生産物であるアルゴリズムがプログラムの本質を表しており、多くの場合、アルゴリズムはプログラムよりも重要です。
- 7.1 節を振り返ってみましょう。「平方根を計算せよ」という問題が与えられた場合、どの手法（二分法、モンテカルト法、ニュートン・ラフソン法のいずれ）を用いるかが、プログラムの性質、性能を決める要因となります。もしモンテカルト法を選んだならば、どんなにプログラムを工夫したとしても実行時間の割に計算精度はほとんど向上しません。しかしニュートン・ラフソン法は、特別にプログラムの作り方を工夫しなくとも短時間で高精度の計算が可能です。この計算手法がアルゴリズムに他なりません。そしてプログラムは、そのアルゴリズムを特定の記述法を用いて実体化し、コンピュータで実行可能にしたものになります。
- 別の説明をします。現代の代表的なプログラミング言語である C++ 言語では、図 7.2 のプログラムと同様のプログラムを図 8.1 のように記述します。二つのプログラムは同じアイデアを異なる記述法で表現したものです。そして、異なる表現であるにも関わらず、両者には共通するアイデアが存在します。それがアルゴリズムです。
- プログラム化すべき問題が与えられたならば、それについてアルゴリズムを考えねばなりません。簡単な場合にはアルゴリズムは頭の中だけにあるかもしれませんが、あるいは分厚い文書として記述されるかもしれません。そしてアルゴリズムに基づいて実際のプログラムを作成することになります。一般に、ひとつの問題には複数のアルゴリズムが存在します。またひとつのアルゴリズムには、それをどのようなコンピュータ、プログラミング言語に実装するかによって複数のプログラ

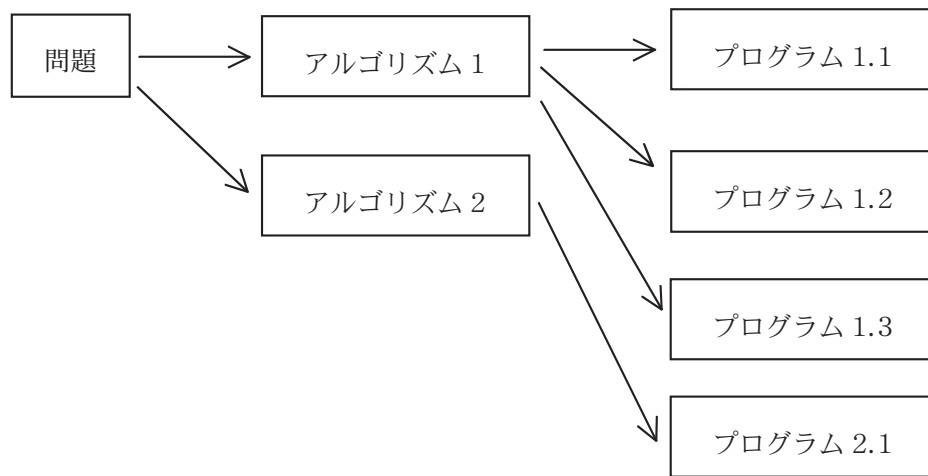


図 8.2: 問題、アルゴリズム、プログラムの関係

1. ラムが存在します (図 8.2 参照)。

第9章 小数点数の不思議

一般にコンピュータは「正確な計算を高速に行う」と盲信されているところがあります。しかし、コンピュータでの計算はしばしば奇妙な結果を示します¹。この講義テキストでも7章の多角形を用いた円周率の計算では、計算の途中からかえって数値誤差が増えるような事態が起きました。これらはコンピュータ内での数値の取り扱い方に起因します。

9.1 コンピュータ内での数値の種類

コンピュータでの数値の表現は二種類に大別されます（下図参照）。ひとつは整数、もうひとつは小数点数です。小数点数には固定小数点数と浮動小数点数の二種類があります。

整数は、..., -3, -2, -1, 0, 1, 2, 3, ... などの、負の自然数、0、正の自然数のことです。プログラミングでは通常、 -2^{31} ($=-2,147,483,648$) から $2^{31}-1$ ($=2,147,483,647$) の範囲の整数を取り扱うことができますが、最近のコンピュータでは -2^{63} ($=-9,223,372,036,854,775,808$) から $2^{63}-1$ ($=9,223,372,036,854,775,807$) の範囲まで扱うことも可能です²。

しかし整数だけでは0.5、3.1415のような、1よりも小さな精度を持つ数値を扱うことができません。そのような小数点を使って表される数値（＝少数点数）の表現方法には固定小数点数（fixed-point

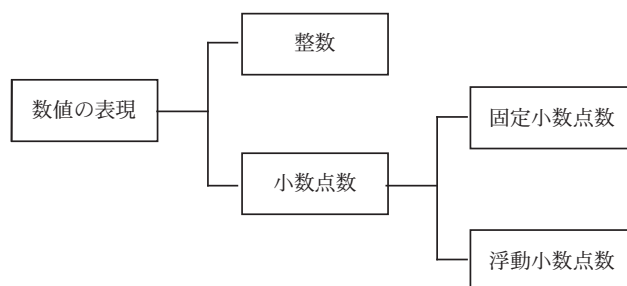


図 9.1: 数値の種類

¹かつてコンピュータには全くの素人の、数学や物理の大先生が「コンピュータは便利だ」と聞いて、試しにコンピュータを使ってみたところ、理論値とかけ離れた計算値が出て「コンピュータは使い物にならん!」と怒ったという話を関係者から聞いたことがあります。

²プログラミング言語によっては任意桁の大きさの整数を扱うことができるものもありますが、計算速度はかなり遅くなります。

number) と浮動小数点数 (floating-point number) の二種類の形式があります。

固定小数点数は、たとえば小数点以下3桁の精度を持つような数値 0.001、0.123、4.665 などです。この場合、小数点の位置が最下桁から3桁目と4桁目の間に固定されており、それが固定小数点数の名前の由来です。これらの数は1000倍すれば1、123、4665などの整数に置き換えることができます。よって固定小数点数は本質的には整数と同等の扱いが可能です。固定小数点の欠点は、精度が固定されているため、状況に応じて精度を変更するなどの融通の効く処理ができない点です。結局、整数と同等であり、整数で代替できるため、現在のコンピュータでは固定小数点数はサポートされていません。

これに対して、浮動小数点数は非常に1以下の小さな数値から非常に大きな数値までをひとつの形式の中で扱うことができるように工夫された数値の表現です。現在のコンピュータでは、整数と浮動小数点数を扱うハードウェアを実装しています。計算の奇妙な振る舞いは、浮動小数点数に起因します。以下、その話です。

9.2 浮動小数点数

浮動小数点数とは、読んで字の如く、小数点が数値の上を浮動³する形式の数値です。これだけの説明ではチンプンカンプンですから、たとえば123.45という数値を考えましょう。これは別の表現では 1.2345×10^2 と表すことができます。また77777は 7.7777×10^4 、0.034567は 3.4567×10^{-2} 、-3.1415は -3.1415×10^0 と表すことができます。いま示した四つの数値:

$$1.2345 \times 10^2, 7.7755 \times 10^4, 3.4567 \times 10^{-2}, -3.1415 \times 10^0$$

の共通点は、いずれも0でない数字が1の位(くらい)から始まるように小数点の位置が調整されている点です。その調整のために 10^n という倍数を用いています。この調整によって、如何なる数も必ず

$$a_1 . a_2 a_3 \cdots a_k \times 10^n$$

という形式で表すことができます。この形式を浮動小数点数の**正規形** (normal form) と呼びます。あるいはこの表現を単に浮動小数点数と呼びます。また $a_1 . a_2 a_3 \cdots a_k$ の部分を 仮数部 (significand)、 n の部分を 指数部 (exponent) と呼びます。仮数部の長さ k は有限かつ固定です。ここでは $k = 5$ とします⁴。

さて、ここで浮動小数点数の(正規形の)四則演算例をいくつか示します。まず以下の加算を考

³「浮動 (floating)」という語は、フラフラと動き、落ち着きのない様子を想起させますが、浮動小数点数について、その連想はあながち間違いではありません。

⁴実際のコンピュータの k の長さは、10、20、40などの大きな数です。後に述べるように、浮動小数点数には誤差が付きますといえますから、 $k = 5$ くらいの長さでは実用に耐えません。ここでは説明の便宜上、小さな k にしました。

えましょう。

$$\begin{aligned} & 4.0000 \times 10^2 + 7.0000 \times 10^2 \\ \rightarrow & 11.0000 \times 10^2 \\ \rightarrow & 1.1000 \times 10^3 \end{aligned}$$

上の計算では、和 11.0000×10^2 を正規形 1.1000×10^3 へ変換しています。次の加算:

$$\begin{aligned} & 4.0000 \times 10^1 + 9.7000 \times 10^2 \\ \rightarrow & 0.4000 \times 10^2 + 9.7000 \times 10^2 \\ \rightarrow & 10.1000 \times 10^2 \\ \rightarrow & 1.0100 \times 10^3 \end{aligned}$$

では、加算する二つの数の指数部が異なります。そこで、まず、大きい方の数に指数部を揃えます。そして仮数部どうしを加算し、最後に正規形へ変換しています。乗算:

$$\begin{aligned} & 2.0000 \times 10^2 * 8.0000 \times 10^4 \\ \rightarrow & (2.0000 * 8.0000) \times 10^{(2+4)} \\ \rightarrow & 16.0000 \times 10^6 \\ \rightarrow & 1.6000 \times 10^7 \end{aligned}$$

- 1 では、仮数部どうし、指数部どうしで乗算し、その後に正規形へ変換します。
- 2 このように、浮動小数点数の計算では、正規形の浮動小数点数どうしで四則演算を行い、その結
- 3 果を正規形の浮動小数点数に変換します。

4 9.3 浮動小数点演算の誤差

- 5 既に述べたように、浮動小数点数の仮数部の長さは有限であり、固定長です。このため、数値の
- 6 表現には誤差が生じます。そして誤差はしばしば奇妙な現象を引き起こします。ここでは代表的な
- 7 3種類の現象を紹介します。これらは、コンピュータの計算が必ずしも常に正しい結果を導く訳で
- 8 はないことを示唆しています。

9 (1) 丸め誤差

仮数部の大きさが5桁であることに注意し⁵、以下の式を計算してみます。

$$(1.0000 \times 10^0 / 3.0000 \times 10^0) * 3.0000 \times 10^0$$

⁵指数部は特に奇妙な現象と関係ないので、たとえば高々2桁としておきましょう。当面、指数部の大きさを気にする必要はありません。

計算は以下のように進みます⁶。

$$\begin{aligned} &\rightarrow (0.3333 \times 10^0) * 3.0000 \times 10^0 \\ &\rightarrow (3.3330 \times 10^{-1}) * 3.0000 \times 10^0 \\ &\rightarrow 9.9990 \times 10^{-1} \end{aligned}$$

- 1 本来は1を3で割って3を掛けるのですから、答えは1になるべきです。しかし、仮数部が有限
- 2 の大きさを持つために、1よりも小さな数値が求められました。コンピュータで浮動小数点計算を
- 3 行くと、仮数部の最下桁にこのような小さな誤差が発生します。これを丸め誤差と呼んでいます。
- 4 丸め誤差は浮動小数点計算では避けて通れない誤差です。演算を繰り返し行くと丸め誤差が堆積し
- 5 ていきます。

6 (2) 情報落ち

以下の式には加算と減算が含まれます。

$$3.0000 \times 10^{-3} + 2.0000 \times 10^2 - 2.0000 \times 10^2$$

この式の加算と減算の優先順位を、左側優先として

$$(3.0000 \times 10^{-3} + 2.0000 \times 10^2) - 2.0000 \times 10^2$$

と解釈すると、演算は以下のように進みます。

$$\begin{aligned} &\rightarrow (0.0000 \times 10^2 + 2.0000 \times 10^2) - 2.0000 \times 10^2 \\ &\rightarrow (2.0000 \times 10^2) - 2.0000 \times 10^2 \\ &\rightarrow 2.0000 \times 10^2 - 2.0000 \times 10^2 \\ &\rightarrow 0.0000 \times 10^2 \\ &\rightarrow 0.0000 \times 10^0 \end{aligned}$$

- 7 すなわち答えは0です。元の式は普通書くと $0.003 + 200 - 200$ を意味しますから、普通に計算すれ
- 8 ば答えは0.003のはずです。しかし上の浮動小数点演算では、演算の途中で0.003が消えてしまい、
- 9 答えは0になりました。0.003が消えた理由は、 $200 (= 2.0000 \times 10^2)$ に比べ $0.003 (= 3.0000 \times 10^{-3})$
- 10 が5桁以上小さ過ぎたためです。このような現象を情報落ちと呼んでいます。

⁶これは浮動小数点演算の一例に過ぎません。演算の方法はプロセッサによって種々異なります。誤差を小さくする様々な方法が現在も研究されています。

1 (3) 桁落ち

今、測定精度が3桁しかない測定器からコンピュータへ二つの浮動小数点数 2.0005×10^2 と 2.0002×10^2 を入力し、その差を求めるとします。

$$\begin{aligned} & 2.0005 \times 10^2 - 2.0002 \times 10^2 \\ \rightarrow & 0.0003 \times 10^2 \\ \rightarrow & 3.0000 \times 10^{-2} \end{aligned}$$

2 このとき、求められた数値 3.0000×10^{-2} はどのような意味を持つでしょうか。元の数値の精度が
3 3桁ですから、実は 3.0000×10^{-2} という数値の全体が測定誤差の範囲内であって、如何なる有用
4 な情報も持ち得ません。このように、大きさが近い二つの数値の間の減算によって、信頼できない
5 桁の数値がもっともらしく浮動小数点数の最前面に現れて出てくる現象を 桁落ちと呼んでいます。
6 ここでは測定器から入力された数値の差を求めましたが、この例に限らず、桁落ちは数値計算で
7 は頻繁に起こりえます。(1) に示した丸め誤差は演算の度に発生し、浮動小数点数の中に少しずつ
8 堆積していきます。そのようにして作られた浮動小数点数どうしの減算で求められた数は、やはり
9 桁落ちしているかもしれないのです。

10 上に示したように、浮動小数点数の演算には常に誤差が付きます。特に情報落ちや桁落ち
11 は浮動小数点数の特異な現象であり、この特徴を認識しないままにプログラムを作ると、全く想
12 像しなかったような変な計算結果に出会うことがあります。誤差を押さえる最も一般的な方法は、
13 仮数部の桁数を十分大きくすることです。仮数部を約20桁くらいの大きさに設定すると、たいて
14 いの科学計算において十分とされています。それでも心配な場合（たとえば原子炉のコンピュー
15 タ・シミュレーションのような、決して間違いの許されない場合）には仮数部を40桁以上に設定
16 して計算することもあります。

考察 和:

$$\sum_{n=1}^{1000000} \frac{1}{n^2} = \frac{1}{1} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{999999^2} + \frac{1}{1000000^2}$$

17 を計算する際に、できるだけ誤差（情報落ち）を小さくする計算手順を示しなさい。また、そのプ
18 ログラムを作りなさい。

1 第10章 さまざまなプログラミング言語

2 講義テキストにはここまで様々なプログラミング言語が名前だけ登場しています。この章ではそ
3 れらプログラミング言語を簡単に紹介します。

4 コンピュータにおける技術革新はすぐにプログラミング言語に取り入れられます。よって歴史的
5 に著名なプログラミング言語を知ることはコンピュータの歴史を知る一助ともなります。

6 10.1 プログラミング言語の種類

7 プログラミング言語は**高水準プログラミング言語** (high-level programming language) と**低水
8 準プログラミング言語** (low-level programming language) に大別されます。高水準言語はハード
9 ウェア・アーキテクチャを意識させない言語構造を持ち、人手によるプログラミングのしやすさ、
10 プログラムの読みやすさを指向したプログラミング言語です。一般に「プログラミング言語」と言
11 えば、これらを指しますし、p5.js は高水準プログラミング言語の範疇に入ります。これに対して
12 低水準言語はハードウェア・アーキテクチャを反映した言語構造を持ち、特定のハードウェアで高
13 速に動作する利点を持ちます。

14 以下ではまず低水準言語に触れ、その後に高水準言語の代表的なものをいくつか紹介します。

15 10.2 低水準プログラミング言語

16 かつてはコンピュータをプログラミングするには低水準プログラミング言語を用いざるを得ませ
17 んでした。

18 10.2.1 機械語

19 機械語 (machine language) は、プロセッサが直接解読可能な2進数データ列 (ビット列) で
20 す。現在、機械語を読むことも機械語でプログラムを作ることはほとんどあり得ません。

21 10.2.2 アセンブリ言語

22 アセンブリ言語 (assembly language) は、機械語に1対1対応するプログラミング言語です。2
23 進数データ列である機械語はそのままでは人間にとってあまりに読みづらいため、キーワードなど

```

sum10:                // 処理 sum10 の開始
    set r3, $0         // r3 に 0 をセット
    set r4, $0         // r4 に 0 をセット
label0: brgt r4, $10, label1 // もし r3 が 10 よりも大きいならば label1 へ行く
    add r3, r3, r4      // r3 に r4 を足し込む
    add r4, r4, $1      // r4 を 1 増やす
    br label0          // label0 へ行く
label1: set r0, r3      // 出力用の r0 に r3 をセット
    call print         // print 処理を実行
    end                // プログラム終了

```

図 10.1: アセンブリ言語のプログラム例

1 を用いて少し読み易く表したものです。本質的に機械語と同じものです。

2 図 10.1 は、あるアセンブリ言語を用いて「1 から 10 までの数の和を求める」プログラムを記述
 3 した例です。ここに r0、r3、r4 はプロセッサ (CPU) 内の小容量メモリの個々の記憶領域を表
 4 します。これらは**レジスタ** () と呼ばれます。JavaScript の変数に相当するものですが、極めて高
 5 速にアクセスできる点が JavaScript の変数とは異なります。ただしレジスタは数十個しかありま
 6 せんから、うまく使い回さないとすぐに足りなくなってしまうます。

7 図 10.1 のプログラム中の `set r3 $0` は代入文 `r3 = 0;` に相当します。`brgt r3, $10, label1`
 8 は、「もし r3 が 10 よりも大きいならば、制御が label1 へ移動する」ことを表します。`add r3, r3,`
 9 `r4` は代入文 `r3 = r3+r3` に、`add r3, r3, $1` は代入文 `r3 = r3+1` に相当します。`br label0`
 10 は、「条件に依らず、制御が label0 へ移動する」ことを表します。`call print` は、「r0 の値を出
 11 力する」ことを表します。結果として、1 から 10 までの数の和が出力されることが分かるでしょう。

12 機械語は図 10.1 の各行が 2 進数で表されたものになりますから、よほど慣れたプログラマでな
 13 い限り、機械語を直接読むことはできません。

14 このように人間に非常に読みにくいプログラムですが、高速に実行できる点が大きな利点です。
 15 しかしあまりに読みにくいため、仮に実行速度が少々遅いとしても、現在のプログラミングでは高
 16 水準プログラミング言語が用いられています。

17 10.3 高水準プログラミング言語

18 高水準言語は**手続き型プログラミング言語** (procedural programming language、以下、手続き
 19 型言語) と**宣言型プログラミング言語** (declarative programming language、以下、宣言型言語)
 20 に大別されます (表 10.1 参照)。

21 手続き型言語とは手順の時系列によって処理内容を記述する種類の言語です。処理内容を順番に
 22 記述していくため、考え方が単純で平易なプログラミングが可能です。しかし、処理の順序を少し

表 10.1: 様々な高水準プログラミング言語の分類

	分類 1	分類 2	言語名	誕生 時期	利用状況 (私見による)
高水準	手続き型 (命令型)	手続き型 (命令型)	FORTRAN FORTRAN77 Fortran90 ...	1950 年代	科学計算分野に利用されている
			COBOL	1950 年代	商業分野に利用されている
			ALGOL	1960 年代	歴史的使命を終えた
			Pascal	1970	ほとんど利用されていない
			C	1972	利用されている
			C++	1980 年代	使用されている
			Basic	1964 から	使用されている
			Ada	1980 年代	ほとんど利用されていない
			Java	1990 年代	使用されている
			JavaScript	1990 年代	利用が拡大している
			Python	1990 年代	利用が拡大している
	宣言型	省略	-	-	-

- 1 間違えただけで予想もしない結果になることが多く、複雑なプログラムの開発では注意力が要求さ
- 2 れます。宣言型言語とは、処理内容をその数学的性質に着目して、あたかも数式を書くように記述
- 3 する種類の言語です。これについては専門的になりすぎますから、この講義では省略します。
- 4 代表的なプログラミング言語を表 10.1 にまとめました。これらについて以下にプログラム例を
- 5 付けて簡単に解説します。

10.3.1 FORTRAN

- 7 FORTRAN（フォートランと読む）は、歴史上最初にコンパイラが作られた高水準プログラミ
- 8 ング言語です¹。FORTRAN の名称は formula translator（数式変換器）に由来します。1954 年に
- 9 考案され、1957 年に IBM704 上で実際に稼働する最初のコンパイラがリリースされました。誕生
- 10 から半世紀経った今もなお使用され続けています。寿命の長い理由のひとつは、当初から実行
- 11 性能を重視する設計思想を貫き、科学技術計算に広く使用された点にあります。もうひとつの理
- 12 由は、言語仕様を時代に合わせて少しずつ変更/拡張していき、それがユーザーに受け入れられた
- 13 点にあります。代表的な仕様を列挙すると、FORTRAN I/II/III/IV, FORTRAN 66, FORTRAN
- 14 77, Fortran 90, Fortran 2003 等となります²。最近では並列コンピュータでの利用を目的に HPF

¹FORTRAN 開発の中心メンバーであった John Backus は、FORTRAN の開発その他の業績によって 1977 年にチューリング賞を受賞しました。

²1977 年以前のものは FORTRAN と記し、それ以降のものは Fortran と記します。

<pre> SUM = 0 DO 1000 I = 1, 10 SUM = SUM+I 1000 CONTINUE WRITE(6,*) SUM END </pre> (a) FORTRAN 66	<pre> SUM = 0 DO I = 1, 10 SUM = SUM+I ENDDO WRITE(6,*) SUM END </pre> (b) Fortran 90
---	--

図 10.2: FORTRAN のプログラム例

- 1 (High Performance Fortran) など提案/開発されています。
- 2 図 10.2 に 1 から 10 までの数の和を求めるプログラムを FORTRAN 66 と Fortran 90 で記述
- 3 した例を示します。両者の違いに言語仕様の変遷を見ることができます。

4 10.3.2 COBOL

5 FORTRAN が第一世代の科学技術計算用プログラミング言語であるのに対して、COBOL（コボ

6 ル）は第一世代の事務処理用プログラミング言語です。COBOL の名称は common business oriented

7 language に由来します。1960 年に最初の言語仕様が策定され、同じ年に最初のコンパイラが開発

8 されました。FORTRAN 同様に、時代と共に言語仕様を少しずつ改変しながら、現在まで使用さ

9 れ続けている長寿の言語です。

10 図 10.3 はプログラム例です³。まず、宣言部が長いという特徴を持っています。これはプログラ

11 ムの保守性を高めるためです。次に、私たちがよく知る形式の代入文がなく、英文のような代入文

12 "MOVE 0 TO WK-SUM." や "ADD I TO WK-SUM." が使用されています。これはプログラムの可読

13 性を高めるためです。WK-SUM は WK から SUM を減ずる演算式ではなく、ひとつの変数を表します。

14 マイナス記号 - は COBOL では減算記号ではありません⁴。また、初期の COBOL の言語仕様には

15 浮動小数点データ型がありませんでした。逆に COBOL では任意桁数の整数型演算が可能です。

16 これらの特徴は COBOL が事務処理用言語として設計されたためです。

17 10.3.3 ALGOL

18 ALGOL（アルゴル）は、FORTRAN、COBOL の開発に刺激を受けて、それまでのプログラ

19 ミングの諸概念を整理/明確化して作られた言語です。名前は algorithmic language に由来します。

20 国際的な協力の下に言語仕様が検討され、1958 年に最初の仕様が作られました。しかしそのとき

21 には既に FORTRAN と COBOL が広く普及しており、ALGOL が実際に使われることがありませ

³このプログラムは 1985 年に策定された COBOL 第 3 次規格に基づきます。

⁴ただし、- の前後に空白などの区切り記号がある場合には減算記号とみなされます。

```

IDENTIFICATION  DIVISION.
PROGRAM-ID.      TEST.
ENVIRONMENT     DIVISION.
DATA            DIVISION.
WORKING-STORAGE SECTION.
01  WK-SUM      PIC 9(3)  COMP.
01  I           PIC 9     COMP.
PROCEDURE DIVISION.
    MOVE 0 TO WK-SUM.
    PERFORM VARYING I FROM 1 BY 1 UNTIL I > 10
        ADD I TO WK-SUM
    END-PERFORM.
    DISPLAY WK-SUM.

```

図 10.3: 数の和を求める COBOL のプログラム例

```

begin
  integer i;
  real sum;
  sum := 0;
  for i := 1 step 1 until 10 do
    sum := sum + i;
  Print Real(sum)
end

```

(a) ALGOL

```

program test(input, output)
var i : integer,
    sum : real;
begin
  sum := 0;
  for i := 1 to 10 do
    begin
      sum := sum+i
    end;
    writeln(sum)
  end.

```

(b) Pascal

図 10.4: ALGOL/Pascal のプログラム例

- 1 んでした。しかし、言語解析技術など、その後のプログラミング言語の研究に与えた影響は小さ
- 2 くありません。アルゴリズムを説明するときには今でもよく「ALGOL-like (ALGOL ライクな)」
- 3 という言い回しが使われます。
- 4 図 10.4(a) に、1 から 10 までの数の和を求めるプログラムを ALGOL で記述した例を示します⁵。
- 5 文末のセミコロンや `for` に、現在のプログラミング言語の源流を見ることができます。

⁵このプログラムが正しいプログラムである保証はありません。言語仕様書やインターネット上のサンプル・プログラムを真似て作りました。

<pre>int main(){ int i; float sum = 0; for (i = 1; i <= 10; i++) sum += i; printf("%f\n",sum); }</pre> (a) C	<pre>int main(){ float sum = 0; for (int i = 1; i<= 10; i++) sum += i; cout << sum << endl; }</pre> (b) C++
--	---

図 10.5: C/C++のプログラム例

10.3.4 Pascal

Pascal (パスカル) は、ALGOL の仕様策定にも関わったスイス工科大のヴィルト (Wirth) 教授が、効率的なプログラム開発とプログラム実行ができる言語を目指して開発した言語です。構造的なプログラム開発が可能でありながら、機能を欲張らず、プログラムの重要概念がコンパクトにまとめられている点が特徴です。情報学科での教育用言語にも適しており、1970 年代初めから 80 年代終わりまで幅広く使用されました。Pascal という名称は、数学者パスカルに由来します。

図 10.4(b) に、1 から 10 までの数の和を求めるプログラムを Pascal で記述した例を示します。

10.3.5 C

AT&T のベル研究所 (Bell laboratories) で開発された C (シー) は、UNIX を記述した言語として有名です。Pascal などと比較してエレガントな言語とはいいたいのですが、OS をほとんどこの言語だけで記述できるなど応用範囲が広く、急速に普及しました。現在、ほとんどのプログラムの開発において C はアセンブリ言語に取って替わったと言ってよいでしょう。1972 年に最初の処理系が開発されました。C の仕様策定と標準化は、1983 年に ANSI⁶に受け継がれ、これが今日 ANSI C と呼ばれているものです。

図 10.5(a) に、1 から 10 までの数の和を求めるプログラムを C で記述した例を示します。

10.3.6 C++

C++ (シープラスプラス) は、1980 年前半にベル研究所で開発された拡張 C 言語です。C 言語の構文規則を洗練し、機能を整理した言語でもあります。最も大きな特徴は C 言語にオブジェクト指向を導入した点にあります。C++ の仕様は ANSI C にも影響を与えました。なお、「オブジェクト指向」については専門的になりすぎますから、ここでは解説を省略します。

⁶the American National Standards Institute

```

10 sum = 0
20 for i = 1 to 10
30     sum = sum+i
40 next
50 print sum
60 end

```

図 10.6: Basic のプログラム例

- 1 図 10.5(b) に、1 から 10 までの数の和を求めるプログラムを C++ で記述した例を示します。た
 2 だしこの例題では C 言語と C++ の違いが際立ちません。

3 10.3.7 Basic

- 4 Basic (ベーシック)⁷ は、コンピュータがきわめて高価であった時代に数 K バイトから数 10K
 5 バイトのメモリ容量に収まるように作られた言語です。最初の処理系は 1964 年に開発されていま
 6 すが、その後、1980 年代末までパーソナルコンピュータ用言語として普及しました。機能は限定
 7 され、実行速度は遅いのですが、覚えやすいという特長から主に初心者用として普及してしまし
 8 た。しかし最近では Visual Basic のように高機能な言語も作られています。

- 9 図 10.6 に、1 から 10 までの数の和を求めるプログラムを Basic で記述した例を示します。なお、
 10 Basic の言語仕様には多くの改良版、方言があり、図 10.6 が典型的な Basic のプログラムという訳
 11 ではありません。

12 10.3.8 Ada

- 13 Ada (エイダ) は、米国国防総省が組み込みシステム向けの信頼性・保守性（つまり兵器の信
 14 頼性・保守性）に優れたプログラミング言語の開発を目指して言語仕様を国際入札によって選定し
 15 た言語です。1977 年から検討が始まり、1983 年に言語仕様が策定されました。Ada にはプログラ
 16 ム抽象化などのその当時の先進的な概念が導入されました。しかし言語仕様が複雑で大き過ぎたた
 17 めコンパイラの開発に手間取り、大企業や軍事などの特別な環境以外では普及が進まず、そうこう
 18 している間に多くのプログラマに忘れ去られつつあります。言語の名称は、歴史上最初のプログラ
 19 マと言われるエイダ夫人（3 章を思い出しましょう）に由来します。

- 20 図 10.7 に、1 から 10 までの数の和を求めるプログラムを Ada で記述した例を示します。プログ
 21 ラムの雰囲気が ALGOL や Pascal に似ています。C プログラムなどと比べると、読み易そうな印
 22 象を受けます。

⁷歴史的には全て大文字の BASIC という名称がオリジナルの言語名称ですが、現在はほとんど全ての派生言語が Basic という名称を使っています。

```

with Ada.Text_IO; use Ada.Text_IO;
procedure main is
    sum: Float;
begin -- of main
    sum := 0;
    for i in 1..10 loop
        sum := sum+1;
    end loop;
    put(sum);
end main;

```

図 10.7: Ada のプログラム例

```

class Test {
    public static void main(String[] args){
        float sum = 0;
        for (int i = 1; i <= 10; i++)
            sum += i;
        System.out.println(sum);
    }
}

```

図 10.8: Java のプログラム例

1 10.3.9 Java

2 Java（ジャバ）は、1990 年初めに家電組み込み用言語として開発が進められましたが、折しも
 3 インターネットの拡大期にあったため、その目的をネットワーク対応にシフトし、Web ブラウザ
 4 で動作できる点で大きな注目を集めるようになりました。1995 年に最初の処理系が開発され、そ
 5 の後も改良が続けられています。Java の構文は C++ に類似しており、C/C++ プログラマには馴
 6 染みやすかった点も普及の一因と思われます。現在、最も普及している言語のひとつです。

7 図 10.8 に、1 から 10 までの数の和を求めるプログラムを Java で記述した例を示します。プロ
 8 グラムが C や C++ に似ていることに気づくでしょう（図 10.5 参照）。

9 10.3.10 JavaScript

10 JavaScript は、この授業のプログラミングで用いた p5.js のベース言語です。Web ブラウザ上で
 11 直接実行できる点が FORTRAN を始めとする過去のプログラミング言語と異なります。上に紹介
 12 した Java もブラウザ上でプログラムを実行する機能を有しますが、直接実行できる訳ではなく、
 13 少し手間が掛かります。それに対して、JavaScript の場合、プログラムを直接、HTML 文書（ブ

<pre> <html> <head> <script> onload = function(){ sum = 0; for (i = 1; i <= 10; i++) sum += i; alert(sum); } </script> </head> <body></body> </html> </pre> (a) JavaScript	<pre> function setup() { sum = 0; for (i = 1; i <= 10; i++) sum += i; alert(sum); } </pre> (b) p5.js
--	--

図 10.9: JavaScript/p5.js のプログラム例

1 ラウザ上に表示されるファイル) に埋め込むことができます。近年、コンピュータを用いた様々な
 2 処理が Web ブラウザ上で行われるようになっており、その利用が拡大しています。JavaScript は
 3 その中核的な技術のひとつと見なされています。

4 JavaScript は 1995 年に開発されました。Java と名前が似ているのは、両者を開発するチームが
 5 業務提携の関係にあったため、それ以前に LiveScript と呼ばれていた言語が改名したためです。当
 6 初は、実行速度が遅い、ブラウザによって動作が異なる（互換性が低い）などの問題がありました
 7 が、徐々に解消され、現在ではよく利用される言語のひとつになっています。

8 図 10.9 (a) は、JavaScript によるプログラム例です。ここまで紹介してきた様々なプログラミ
 9 ング言語のプログラムと類似していることに気づきます。1990 年代以降のほとんどのプログラミ
 10 ング言語の構文は、C 言語のそれをベースとしているためです（そうでないと 1980 年代に C 言語
 11 の ICT 教育を受けたプログラマを取り込むことができないためです）。なお、`alert()` は結果を
 12 ブラウザではなく、ポップアップウインドウに表示する関数です。これを HTML ファイルとして、
 13 たとえば `test.html` という名前のファイルとして PC に保存し、適当なブラウザで `test.html` を
 14 ファイル読み込みすると直ちに計算が実行され、結果がポップアップウインドウに表示されます。

15 繰り返しになりますが、p5.js は JavaScript をベース言語としており、JavaScript に一皮被せた
 16 ような作りになっています。よって JavaScript のプログラムは p5.js のエディタ上でも動作します。
 17 図 10.9 (b) は、図 10.9 (a) の JavaScript プログラム部分を p5.js の関数 `setup()` にそのまま移植
 18 したものです。このプログラムは正常動作し、結果がポップアップウインドウに表示されます。と
 19 いうのも、p5.js のプログラムは HTML ファイルにコピーされて JavaScript のプログラムとして
 20 実行されているためです。

```

sum = float(0)
for i in range(1,11):
    sum += i
print(sum)

```

図 10.10: Python のプログラム例

1 10.3.11 Python

2 Python (パイソン) は、Java とほぼ同時期にオランダで開発され、その後、改良が継続されて
 3 きた言語です。多数の有用なライブラリ（たとえば機械学習のライブラリ）が開発されています。
 4 プログラムサイズが小さくなるように言語が設計されており、結果、プログラムの可読性が高く、
 5 開発効率が良いことから急速に利用が広がっています。

6 図 10.10 に、1 から 10 までの数の和を求めるプログラムを Python で記述した例を示します。

7 まず気づくことは文末にセミコロン ; がありません。セミコロンは C 言語以降のほとんどのプ
 8 ログラミング言語に採用されている記法ですが、それがないとプログラミング言語として成り立た
 9 ない訳ではありません。

10 2 行目の for 文の `in range()` を用いる書き方は最近のプログラミング言語の流行りの書き方
 11 です。

12 Python のブロック構造は各行のインデントの深さで決まるルールになっています。よって、図
 13 10.10 のプログラムを

```

14 sum = float(0)
15 for i in range(1,11):
16     sum += i
17     print(sum)

```

18 と書くと、`print(sum)` はループの中で 10 回繰り返されることになります。このルールはプログ
 19 ラマにプログラムの書き方を強制することになるため、「誰が書いても同じ形のプログラムになり
 20 やすい」＝「プログラムが読みやすい」という利点があると考えられています。

21 10.4 プログラミング言語の比較

22 ここまで代表的なプログラミング言語を簡単に眺めてきました。

23 全ての言語について和計算を行うプログラム例を示しました。結果、どのプログラムも似た形に
 24 なっていることに気づくはずです。実際、多くの手続き型言語は共通する概念を持っています。思
 25 いつくまに挙げてみると...

26 **条件分岐** ... 処理内容を選択する。

- 1 繰り返し ... 同一の処理を一定回数繰り返す。
- 2 ブロック構造 ... 処理単位を階層化する。
- 3 手続き呼び出し/関数呼び出し ... 処理内容を抽象化/共通化する。
- 4 配列 ... 大量のデータの並び。
- 5 構造型/レコード型 ... 複数データをまとめて階層化する。（この授業では解説していません）
- 6 入出力 ... データの入出力を行う。
- 7 これらの実現方法は言語によって異なる場合があります。しかし、ほとんどの手続き型言語ではこ
- 8 れら概念（機能）をサポートしており、類似したプログラミングが可能です。よって、ひとつの手
- 9 続き型言語を一定レベルまで掘り下げてマスターすると、その他の手続き型言語は簡単にマスター
- 10 できるものです。

11 10.5 補足

12 10.5.1 高水準プログラミング言語の状況

13 オランダの TIOBE というソフトウェア会社が毎月、その時点で話題になっているプログラミン
14 グ言語のランキングを

15 <https://www.tiobe.com/tiobe-index/>

16 に発表しています。ここでいう「話題」（英語の原文では popularity）とは、主にインターネット
17 の検索に基づいたランキングです。詳細は上のページを参照してもらうこととし、2022 年 9 月の
18 ランキングは表 10.2 の通りです。過去、java、C、C++ は常に上位 3 位以内にいましたが、評価
19 値は徐々に下がってきており、今年は C++ が 3 位から陥落し、代わりに Python が一位に入りました。

21 このランキングは話題性によるランキングですから、その言語が利用されている頻度や言語の優
22 秀性とは直接関係しないことに注意してください。また、このランキングは上下動は激しく、信頼
23 性に欠けるとの指摘もあります。ひとつの参考資料として理解してください。

表 10.2: 今、話題のプログラミング言語のランキング (TIOBE Programming Community index による)

#	言語名	#	言語名	#	言語名
1	Python	18	R	35	D
2	C	19	Perl	36	PowerShell
3	java	20	Ruby	37	ABAP
4	C++	21	Julia	38	Awk
5	C#	22	Scratch	39	LabVIEW
6	Visual Basic	23	SAS	40	TypeScript
7	JavaScript	24	(Visual)FoxPro	41	Groovy
8	Assembly language	25	COBOL	42	Erlang
9	SQL	26	Rust	43	Haskell
10	PHP	27	Prolog	44	cg
11	Objective C	28	Ada	45	Transact-SQL
12	Go	29	Lua	46	Bash
13	Delphi/Object Pascal	30	Lisp	47	Clojure
14	MATLAB	31	PL/SQL	48	Apex
15	Fortran	32	Dart	49	ActionScript
16	Swift	33	Scala	50	Elixir
17	Classic Visual Basic	34	Kotlin	...	...

1 第11章 p5.js プログラミング（インターネット編）

2 講義テキスト最後の話題はインターネットに接続するプログラムの作成です。そのために、まず、
3 インターネット全般について少し詳しく解説し、次に p5.js のプログラムを用いてインターネット
4 からデータを取得してみます。

5 11.1 コンピュータネットワークの歴史

6 3章を思い出しましょう。以下のような項目がありました。

- 7 1. アメリカ国防総省の推進で大学間ネットワーク ARPANET が構築される（1967）。これがイ
8 ンターネットの第一歩になりました。
- 9 2. 通信規約（プロトコル）TCP/IP が設計される（1974）。現在のインターネットで使用され
10 ているプロトコルです。
- 11 3. ローカルエリアネットワーク Ethernet が設計される（1975）。現在のインターネット技術
12 （特にハードウェア面で）の出発点と考えてよいでしょう。
- 13 4. World Wide Web（WWW）プロジェクトが発表される（1991）。以下、略。

14 11.1.1 TCP/IP

15 ここではまず 2. に注目します。2. は、コンピュータとコンピュータが通信する際の通信規約 –
16 通信の方法に関する約束事 – が標準化されたことを意味します。通信規約が定まれば、それに従
17 う限り、どんなコンピュータでも互いに通信できることとなり、この時点からコンピュータ間通信
18 が飛躍的に発展することとなりました。

19 2. には「TCP/IP」という用語が現れますが、これは IP（Internet Protocol）と TCP（Trans-
20 mission Control Protocol）の二つの通信規約からなる言葉です。

21 IP は、データを送信側コンピュータから受信側コンピュータへ送るための通信規約です。「IP ア
22 ドレス」という用語を見聞きしたことはないでしょうか。その名前は通信規約 IP に由来します。
23 IP アドレスは、インターネットに直接つながっているコンピュータ毎に割り当てられるコンピュー

タの住所のことです。たとえば Google の Web サーバの IP アドレスは、0～255（2 進数 8bit）の
四つの数の並びで表すと 172.217.175.35 です。IP は、データ（パケットとも言います）をそのデー
タに付随する IP アドレスに基づいて受信側コンピュータへ届けることを目的とした通信規約です。
TCP は、規約のレベルでは IP の上に付加された通信規約であり、IP によって送受信される
データが送信側から受信側へ**確実に**伝送されることを保証します。TCP の元の用語 Transmission
Control Protocol（伝送制御プロトコル）はまさにそれを表すものです。たとえば送信側が

「送ったデータは届きましたか？」

と受信側へ問い合わせ、受信側が

「だいぶ時間が経つようですが、まだそのデータは届いていません。」

と返事をし、送信側が

「それでは再送します。」

と受信側へ答え、受信側が

「届きました！」

と返事をするような、そのような確認を行うための通信規約です。ネットワークは停電などによっ
て突然、切断されることもまれではありません。もちろんデータの到着の確認しない場合に比べれ
ば相当の手間（コスト）が掛かるのですが、重要なデータ（たとえば銀行間の送金など）のやりと
りには確実性は必須です。

インターネットを流れるほとんどのデータは IP+TCP のルールの下に転送されています。例外
は、動画などの垂れ流しのデータです。データが転送の途中で少しだけ失われ、映像の質が落ちた
としても、人の生死に関わるような重大な問題は生じませんから、動画データの転送には TCP は
用いず、別の通信規約の UDP（User Datagram Protocol）などを用います。

寄り道 IP アドレスは 2 進数です。これまでは 32 bit の IP アドレス（IPv4 = IP バージョン
4）が使用されてきました。よって IP アドレスを 16 進数（16 進数 1 桁で 2 進数 4 桁に相当）で表
すならば、00000000₁₆ から FFFFFFFF₁₆ までの約 43 億個が使用可能です。この数は世界の人口
（現在、約 80 億）よりも少なく、IP アドレスは一人 1 個を割り当てることができない状況になっ
ています。コンピュータの所有が当たり前の現在においてこれは深刻な状況で、IP アドレス枯渇
問題などとも言います。

そこで、次世代の IP アドレス（IPv6）は 128 bit の大きさを持つように制定されました。128
bit 整数の最大値は 16 進数で表せば FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF₁₆ となり、
10 進数では約 10^{38} に相当します。つまり IPv6 では 10^{38} 個のコンピュータをインターネットに接

1 続可能です。これならば一人 1 個は言うに及ばず、一人当たり 10^{28} 個のコンピュータをインター
 2 ネットに接続可能なほどに十分な余裕があります。IPv6 は、エレベータ、ライト、空調システム、
 3 バスなど世の中のほとんど全てのものに IP アドレスを割り付け、リモート操作を可能する IOT
 4 (Internet Of Things、物のインターネット) には不可欠の技術と考えられています。

5 11.1.2 World Wide Web

6 次に項目 4. に注目します。データが TCP/IP の制御の下に世界中を転送され回り始めた 1990
 7 年代初頭に、より簡単なデータ管理、データ転送の仕組みが望まれるようになり、欧州原子核研究
 8 機構 (CERN) の技術者によって世界で最初の Web サーバが開発され、同時に Web ページが公開
 9 されます。これは CERN の実験で得られた膨大なデータを世界中の物理学者と共有することが目
 10 的でした。しかし、その便利さは単に物理学での利用にとどまらず、瞬く間に世界中で同様のシス
 11 テムが開発され、また Web ブラウザが開発され、現在のインターネットの雛形が形作られます。

12 WWW の基本技術は情報の提供を行う Web サーバの技術とそのサーバを利用し、主に情報を
 13 表示するクライアント側の Web ブラウザの技術です。Web サーバと Web ブラウザの間は、前節
 14 で見た TCP/IP で繋がれ、さらにその上に Web 用通信規約である HTTP (Hypertext Transfer
 15 Protocol) が用いられています。

16 HTTP の「Hypertext」が表すように、HTTP はハイパーテキスト表現を基本とする通信規約
 17 です。ハイパーテキストとは — 実はみなさんは Web ブラウザを利用することで既に無意識にこ
 18 れを理解しているのですが — ボタンやメニュー、コマンドを介して複数の文書が相互に関連付け
 19 られた状況やシステムのことです。このアイディアは古くから知られていました (Wikipedia を参
 20 照のこと) が、具体的な姿を見せ始めるのはマウスとビットマップディスプレイを装備したコン
 21 ピュータ上に GUI (グラフィカルユーザーインターフェース) が実装されるようになる 1980 年代
 22 からです。Apple 社の Macintosh に HyperCard (1987) は現在の Web ブラウザの前身とみるこ
 23 とができます。

24 11.1.3 URL

25 既に述べたように、インターネットにつながっている全てのコンピュータには IP アドレスが割り
 26 当てられており、コンピュータ間通信ではこれを用いてデータの送受信を行います。Google の Web
 27 サーバの IP アドレスは 172.217.175.35 ですから、Web ブラウザに “https://172.217.175.35”
 28 と入力してサイトにアクセスすると、実際、Google のサーバーにつながります。

29 しかし、Web ブラウザから Web サーバにアクセスするとき、IP アドレスを指定することは今
 30 やまれです。数字の並びは人にとっては認識しにくい情報です (この辺りの議論は前章の機械語
 31 の話と似ています) から、現在では IP アドレスを直接指定するのではなく、それを文字列に置き

- 1 換えた URL（Uniform Resource Locator）を用いるのが一般的です。ブラウザに与えた文字列の
- 2 URL は、専用のコンピュータ（この用途のコンピュータを DNS サーバと呼びます）によって数字
- 3 の並びである IP アドレスに変換され、目的の Web サーバと通信を行います。
- 4 URL でサーバを特定する仕組みを高度化すると、ひとつの URL に複数の IP アドレスを対応さ
- 5 せたり、逆に複数の URL をひとつの IP アドレスに対応させるなどの対応付けが可能となります。
- 6 その結果、インターネットの世界は以前よりも柔軟で、しかし複雑な構造になっています。

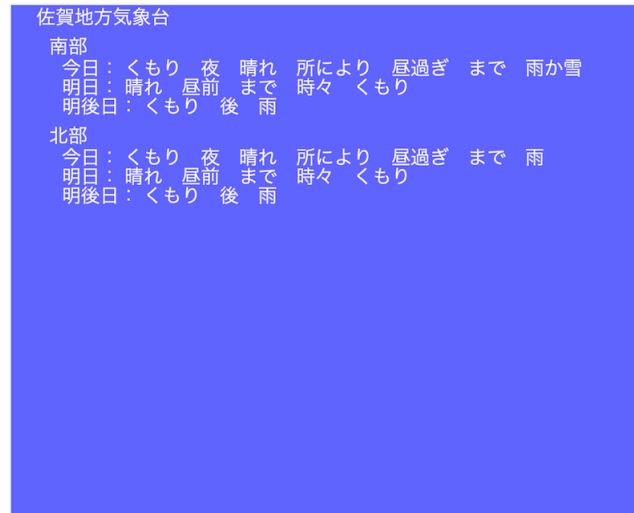


図 11.1: 佐賀県の 3 日間の気象予報

11.2 p5.js プログラムによる Web サーバへのアクセス (気象予報)

一般に、Web ブラウザから Web サーバへデータ送信要求があれば、サーバはデータをブラウザに返信します。実はサーバから見れば送信先がブラウザか否かは関係なく、正しいルール (通信規約=プロトコル) の下での送信要求にはルール通りに応えるだけです。

この節では、その送信要求を p5.js のプログラムから行ってみます。もしプログラムで Web サーバのデータを取得できれば、簡易な自前の Web ブラウザを作ることでもあります。ここではその真似事をやってみます。p5.js には HTTP の下でコンピュータ間通信を行い、データを取得する関数が用意されています `httpGet()`¹ から、Web サーバへのアクセスは容易です。

この節では例題として、p5.js で図 11.1 のような気象予報情報をキャンバスに表示するプログラムを作ります。このために気象庁の Web サーバにアクセスします。プログラムを少し変えるだけで、図 11.2 のように福岡県の気象予報表示も可能です。

なお、この節の内容は主に以下のサイトを参考にしました。

- (1) p5.js リファレンス (`httpGet()`) , <https://p5js.org/reference/#/p5/httpGet>
- (2) p5.js Web Editor 上に気象庁の天気予報情報を表示してみる (HTTP リクエストには `axios` を利用) , <https://qiita.com/youtoy/items/1a8ba048a3798c7875a6>
- (3) 気象庁公式の天気予報の情報 (JSON) を `curl`・`Node.js` で取得し `Node.js` での処理を試す, <https://qiita.com/youtoy/items/932bc48b03ced5a45c71#3> 日間の天気予報の取得

¹p5.js には HTTP 下の通信をサポートする関数として他に `httpPut()`、`httpDo()` もあります。

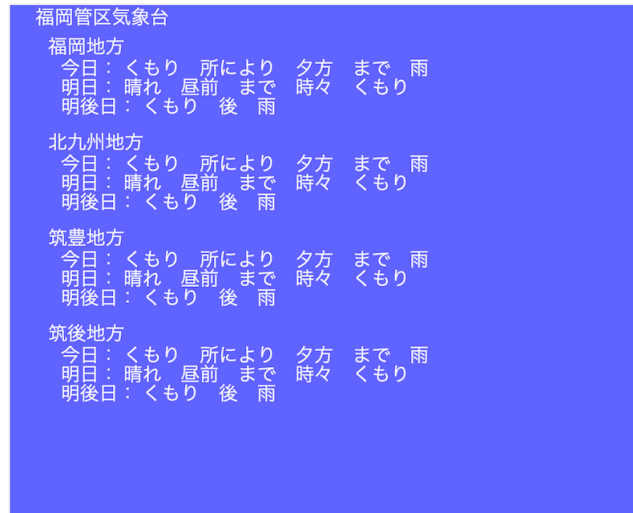


図 11.2: 福岡県の 3 日間の気象予報

11.2.1 佐賀の気象予報データをテキスト形式で取得する

図 11.3 は、気象庁のサーバから佐賀県の 3 日間分の気象予報データを取得する最も単純なプログラムです。ただし、このプログラムでは図 11.1 の表示は得られません。このプログラムを実行すると、2023 年 1 月 20 日現在ではコンソールに図 11.4 が出力されます。佐賀県の気象予測データらしきものが表示されていることが分かります。

まずはこのプログラムの説明です。ここでは以下の関数を用いています。

`preload()` ... これは p5.js の `setup()`、`draw()` と同様に、必要に応じてプログラマが定義すべき関数です。`setup()` は描画処理の初期化を行います、その描画処理よりもさらに以前に実行すべき処理はこの関数 `preload()` の中に記述します。一般に、画像をファイルから読み込むなどのデータの事前準備が「さらに以前」に行うべき処理です。`preload` は「先に読み込む」の意味です。インターネットにアクセスし、データを取得する処理もそれに該当しますから、図 11.3 では `preload()` の中で `httpGet()` を呼び出します。

`httpGet(url, type, callback)` ... Web サーバからデータを取得する関数です。

最初の引数 `url` にデータを取得したいページを URL を表す文字列で指定します。

2 番目の `type` にはデータの形式を文字列で指定します。これには、`"json"`、`"jsonp"`、`"binary"`、`"arrayBuffer"`、`"xml"`、`"text"` のいずれかが指定可能です。図 11.3 ではデータを単純な文字列で取得するため、`"text"` を指定します。

3 番目の引数の `callback` には、Web サーバからデータを取得した直後に実行したい関数を指定します。ただし 1 引数関数でなければなりません。その引数に自動的にデータが格

```

httpData = null;          // プログラム全体で使う変数 (大域変数)

function preload() {
  httpGet(
    "https://www.jma.go.jp/bosai/forecast/data/forecast/410000.json", // URL
    "text",                    // データ形式
    setHttpData              // 関数
  );
}

function setHttpData(data) {
  httpData = data;          // httpGet() が取得したデータを httpData へ代入
}

function setup() {
  background(255, 0, 0);    // キャンバスを赤で塗る
}

function draw() {
  if (httpData != null) { // もし data が null でないならば
    background(0, 0, 255); // キャンバスを青で塗る
    print(httpData);        // httpData をコンソールに表示
    noLoop();               // draw の呼び出しを停止
  }
}

```

図 11.3: 佐賀県の気象予報を気象庁から取得するプログラム (TEXT 版)

1 納されます。図 11.3 では関数 `setHttpData()` を自ら定義し、それを引数として指定してい
2 ます。

3 関数が引数になるというのは少し特殊です。というのも、この講義テキストでこれまで紹
4 介してきたプログラムで、関数それ自体を引数にするものはありませんでした。このような
5 仕組みをコールバックと呼びますが、この講義の範囲を超えますから、これ以上の詳細な解
6 説は省略します。

7 `setHttpData(data)` ... この関数の関数名はプログラマが自由に定めて構いません。仮に名前を
8 変更したならば、`httpGet()` の第 3 引数も同じく変更せねばなりません。図 11.3 の関数本
9 体には代入文がひとつあるだけで、Web サーバから取得されたデータが格納された引数の
10 `data` を変数 `httpData` に代入しています。

11 `noLoop()` ... これは p5.js の関数です。通常、p5.js では関数 `draw()` が 1 秒間に数十回自動的に
12 呼び出されますが、この関数を実行するとその繰り返しの呼び出しが止まります。

13 さて次に図 11.3 のプログラムの実行の様子を時系列で順に説明します。図 11.5 にそれを図示し

1 ましたので参考にしてください。

2 1. 図 11.3 の 1 行目：

3 `httpData = null; // プログラム全体で使う変数`

4 には関数の外に変数が定義されています。このような変数は大域変数 (global variable) と呼
5 ばれ、複数の関数の間でデータを共有するために用います。この場合、大域変数 `httpData`
6 に、値が無いことを表す特殊なデータ `null` が代入されます²。

7 2. 上に述べたように、`setup()` の実行の前に `preload()` が実行され、関数本体の関数 `httpGet()`
8 が呼び出されます。プログラムは `httpGet()` の第一引数の URL へアクセスし、データ取得
9 を開始します。関数 `httpGet()` の実行は、データがサーバから到着するのを待たないですぐ
10 に終了します。よって `preload()` の実行もすぐに終了します。ただし、コンピュータの内部
11 では `httpGet()` によるデータ取得が継続されます。

12 3. `setup()` が実行され、キャンバスが赤で塗りつぶされます。

13 4. 一般に Web サーバからのデータの取得には数秒程度の時間が掛かります。もしデータがサー
14 バからプログラムへ到着したならば、`httpGet()` の第 3 引数に設定した関数 `setHttpData()`
15 が `setup()`、`draw()` の実行とは別に実行されます。そうすると、`setHttpData()` の代入文：

16 `httpData = data; // httpGet() が取得したデータを data へ代入`

17 が実行されます。引数 `data` にはサーバからのデータが格納されていると約束されています
18 (そうなるように `httpGet()` は作られています) から、上の代入文によって、サーバからの
19 データは大域変数 `httpData` に格納されます。

20 5. `draw()` が繰り返し実行されます。以下、その処理です。

21 (a) `httpData` が `null` と異なるか否か判定します。

22 (b) もし等しい (まだデータがサーバから到着していない) ならば、すぐに今回の `draw()`
23 の実行を終了します。数十ミリ秒後には次の `draw()` が始まります。

24 (c) もし異なる (データがサーバから到着し、`httpData` には `null` でない値が格納されて
25 いる) ならば、`httpData` にはサーバからのデータが到着し、格納されています。そこ
26 で、キャンバスが青で塗りつぶされ、`httpData` の値がコンソールへ出力され、`draw()`
27 の繰り返しが停止されます。

²`null` 以外でもよいのですが、こういう場合、`null` を使うのが一般的ですから、それに従います。

第 11 章 p5.js プログラミング（インターネット編）

- 1 結局、サーバからのデータが到着するまで、`draw()` が繰り返し実行され、到着したならば、デー
- 2 タが出力されて、プログラム全体の実行が終了します。キャンバスの色の変化（赤から青）はデー
- 3 タの到着の有無を教えてください。
- 4 このようにして、佐賀県の気象予報のデータが取得できる訳ですが、単なるテキストの表示（図
- 5 11.4）では予報の内容は読みづらいものです。

```
[{"publishingOffice": "佐賀地方気象台", "reportDatetime": "2023-01-20T11:00:00+09:00",
"timeSeries": [{"timeDefines": ["2023-01-20T11:00:00+09:00",
"2023-01-21T00:00:00+09:00", "2023-01-22T00:00:00+09:00"], "areas": [{"area": {"name": "
南部", "code": "410010"}, "weatherCodes": ["211", "101", "214"], "weathers": ["くもり 夜
晴れ 所により 昼過ぎ まで 雨か雪", "晴れ 昼前 まで 時々 くもり", "くもり 後
雨"], "winds": ["西の風 やや強く", "北の風 日中 南の風", "北東の風"], "waves": ["1 メ
ートル 後 0.5メートル", "0.5メートル", "0.5メートル"]}, {"area": {"name": "
北部", "code": "410020"}, "weatherCodes": ["211", "101", "214"], "weathers": ["くもり 夜
晴れ 所により 昼過ぎ まで 雨", "晴れ 昼前 まで 時々 くもり", "くもり 後 雨"],
"winds": ["西の風 やや強く 後 北の風", "北の風 後 北西の風", "北の風 後 北東の風
"], "waves": ["2.5メートル", "1.5メートル 後 1メートル", "1メートル 後 1.
5メートル"]}], {"timeDefines": ["2023-01-20T12:00:00+09:00",
"2023-01-20T18:00:00+09:00", "2023-01-21T00:00:00+09:00", "2023-01-21T06:00:00+09:00",
"2023-01-21T12:00:00+09:00", "2023-01-21T18:00:00+09:00"], "areas": [{"area": {"name": "
南部", "code": "410010"}, "pops": ["10", "0", "0", "0", "0", "0"], {"area": {"name": "北
部", "code": "410020"}, "pops": ["10", "0", "0", "0", "0", "0"]}],
{"timeDefines": ["2023-01-20T09:00:00+09:00", "2023-01-20T00:00:00+09:00",
"2023-01-21T00:00:00+09:00", "2023-01-21T09:00:00+09:00"], "areas": [{"area": {"name": "
佐賀", "code": "85142"}, "temps": ["10", "10", "0", "10"]}, {"area": {"name": "伊万里",
"code": "85116"}, "temps": ["11", "11", "-1", "9"]}]}, {"publishingOffice": "佐賀地方気
象台", "reportDatetime": "2023-01-20T11:00:00+09:00",
"timeSeries": [{"timeDefines": ["2023-01-21T00:00:00+09:00",
"2023-01-22T00:00:00+09:00", "2023-01-23T00:00:00+09:00", "2023-01-24T00:00:00+09:00",
"2023-01-25T00:00:00+09:00", "2023-01-26T00:00:00+09:00", "2023-01-27T00:00:00+09:00"],
"areas": [{"area": {"name": "佐賀県", "code": "410000"}, "weatherCodes": ["101", "214",
"206", "205", "204", "200", "200"], "pops": ["", "70", "70", "70", "50", "40", "40"],
"reliabilities": ["", "", "C", "C", "C", "B", "B"]}],
{"timeDefines": ["2023-01-21T00:00:00+09:00", "2023-01-22T00:00:00+09:00",
"2023-01-23T00:00:00+09:00", "2023-01-24T00:00:00+09:00", "2023-01-25T00:00:00+09:00",
"2023-01-26T00:00:00+09:00", "2023-01-27T00:00:00+09:00"], "areas": [{"area": {"name": "
佐賀", "code": "85142"}, "tempsMin": ["", "1", "5", "-5", "-5", "-2", "2"],
"tempsMinUpper": ["", "3", "6", "-3", "-3", "0", "3"], "tempsMinLower": ["", "-1", "3", "-7",
"-8", "-5", "-2"], "tempsMax": ["", "9", "11", "6", "1", "5", "8"], "tempsMaxUpper": ["",
"10", "13", "9", "4", "9", "12"], "tempsMaxLower": ["", "6", "10", "5", "0", "3", "5"]}],
"tempAverage": {"areas": [{"area": {"name": "佐賀", "code": "85142"}, "min": "1.6",
"max": "9.8"}]}, "precipAverage": {"areas": [{"area": {"name": "佐賀", "code": "85142"},
"min": "3.9", "max": "16.3"}]}}
```

図 11.4: 図 11.3 のプログラムの出力例: 上の方に天気概況、下の方に気温が記載されています。この節では天気概況のみを用います。

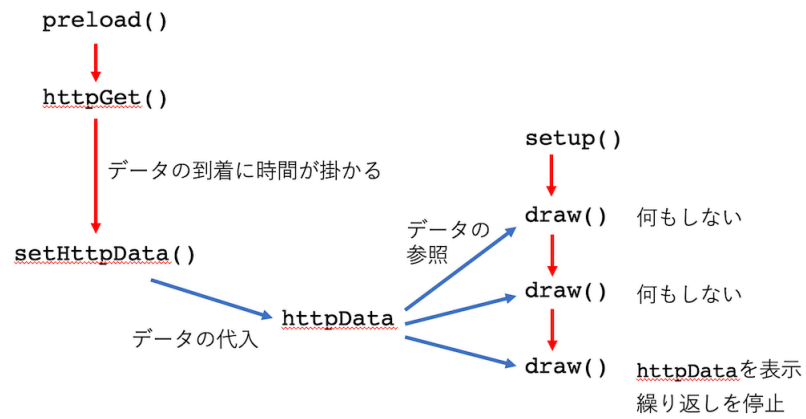


図 11.5: 大域変数と関数呼び出しのタイミング

```

httpData = null;

function preload() {
  httpGet(
    "https://www.jma.go.jp/bosai/forecast/data/forecast/410000.json", // URL
    "json",                    // ここのみ変更！
    setHttpData
  );
}
後略

```

図 11.6: 佐賀県の気象予報を気象庁から取得するプログラム (JSON 版、その 1)

11.2.2 佐賀の気象予報データを読みやすく表示する

図 11.4 の表示を見ると、括弧 ()、{ }、[] が整然と使われており、データは何らかの方法で構造化されているらしいことに気づきます。実はこのデータは JSON (JavaScript Object Notation) 形式と呼ばれるデータ形式になっています。JSON 形式は処理が簡単であるため、最近は Web データの処理を始め、利用が広がっています。

気象予報のデータが JSON 形式であることを利用して、単なる文字列としてではなく、JSON 形式に基づく構造を持ったデータを取得するには図 11.3 のプログラムの `httpGet()` の 2 番目の引数:

```
"text",          // データ形式
```

を

```
"json",          // データ形式
```

に変更し (図 11.6 参照)、プログラムを実行します。そうすると、p5.js エディタのコンソールには図 11.7 に示すように、

```
▶ (2) [Object, Object]
```

が表示されます。これはデータが二つのオブジェクトからなる配列であることを表しています。配列については 5.10 節で簡単に学んだことを思い出してください。

ここで ▶ をクリックしてみます。そうすると、▶ が ▼ に変わり、表示が図 11.8 のように変化し、データの内容が少し詳しく表示されます。

さらに「▶ (0) Object」の ▶ をクリックすると表示が図 11.9 のように変化し、(0) Object の中に `publishingOffice` と `reportDatetime` の二つの内容があることが分かります。このようにひとつのデータの中にさらにデータが含まれる階層構造になっている場合、そのデータは **オブジェクト**などと呼ばれます。

さらに図 11.9 の ▶ を何度もクリックしていくと図 11.10 のような表示を得ます。この図に現れる

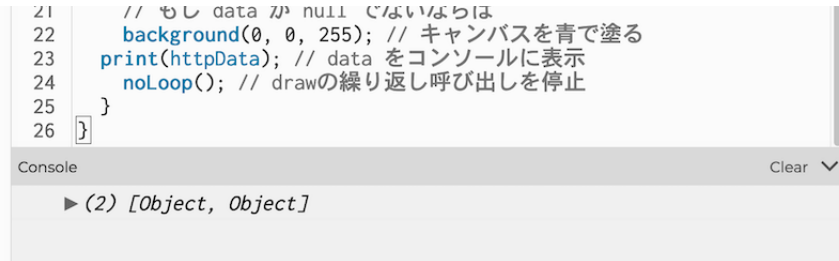


図 11.7: JSON 形式のデータの表示 (その 1)

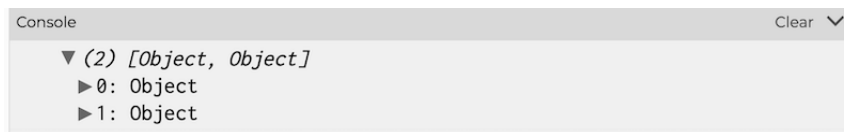


図 11.8: JSON 形式のデータの表示 (その 2)



図 11.9: JSON 形式のデータの表示 (その 3)



図 11.10: JSON 形式のデータの表示（その 4）

- 1 「佐賀地方気象台」、
- 2 「南部」、
- 3 「北部」、
- 4 「くもり 夜 晴れ 所により 昼過ぎ まで 雨か雪」、
- 5 「晴れ 昼前 まで 時々 くもり」
- 6 などをデータからうまく抜き出して表示すれば、佐賀県の気象予報ができそうです。それは実際可
- 7 能で、たとえば「佐賀地方気象台」は、
- 8 配列 `httpData` の 0 番目の要素の中の `publishingOffice` という名前の要素
- 9 ですから、プログラム中で
- 10 `httpData[0].publishingOffice`
- 11 として参照できます。ここに、「○△□という名前の要素」という部分は、ピリオド “.” の後に○
- 12 △□を付けて、「. ○△□」と書くルールです。他の文字列は以下の対応になります。
- 13 **南部** ... `httpData` の 0 番目の要素の中の `timeSeries` という名前の要素（＝配列）の 0 番目の
- 14 要素の中の `areas` という名前の要素（＝配列）の 0 番目の要素の中の `area` という名前の要

```

1     素の中の name という要素
2     = httpData[0].timeSeries[0].areas[0].area.name

3     くもり 夜 晴れ 所により 昼過ぎ まで 雨か雪 ... これは南部地方の今日の天気ですが、
4     httpData の 0 番目の要素の中の timeSeries という名前の要素 (=配列) の 0 番目の要素
5     の中の areas という名前の要素 (=配列) の 0 番目の要素の中の weathers という名前の要
6     素 (=配列) の 0 番目の要素 = httpData[0].timeSeries[0].areas[0].weathers[0]

7     晴れ 昼前 まで 時々 くもり ... これは南部地方の明日の天気ですが、
8     httpData[0].timeSeries[0].areas[0].weathers[1]
9     (最後の weathers[1] がひとつ上の最後の weathers[0] と異なります)

10    くもり 後 雨 ... これは南部地方の明後日の天気ですが、
11    httpData[0].timeSeries[0].areas[0].weathers[2]
12    (最後の weathers[2] がひとつ上の最後の weathers[1] と異なります)

13    北部 ... httpData[0].timeSeries[0].areas[1].area.name (areas[1] が「南部」の
14    areas[0] と異なります。)

15    くもり 夜 晴れ 所により 昼過ぎ まで 雨 ... これは北部地方の今日の天気ですが、
16    httpData[0].timeSeries[0].areas[1].weathers[0] (同上)

17    晴れ 昼前 まで 時々 くもり ... これは北部地方の明日の天気ですが、
18    httpData[0].timeSeries[0].areas[1].weathers[1] (同上)

19    くもり 後 雨 ... これは北部地方の明後日の天気ですが、
20    httpData[0].timeSeries[0].areas[1].weathers[2] (同上)

21    以上のことから、佐賀県の気象予報を見やすくコンソールに表示するプログラムが図 11.11 の通
22    りとなります。結果、以下のような表示を得ます。
23    佐賀地方気象台
24    南部
25    今日：くもり 夜 晴れ 所により 昼過ぎ まで 雨か雪
26    明日：晴れ 昼前 まで 時々 くもり
27    明後日：くもり 後 雨
28    北部
29    今日：くもり 夜 晴れ 所により 昼過ぎ まで 雨
30    明日：晴れ 昼前 まで 時々 くもり
31    明後日：くもり 後 雨

```

前略 (直前のプログラムと同じ)

```
function draw() {  
  if (httpData != null) {  
    background(100, 100, 255);  
  
    print(httpData[0].publishingOffice); // 佐賀地方気象台  
    print(httpData[0].timeSeries[0].areas[0].area.name); // 南部  
    print("今日： " + httpData[0].timeSeries[0].areas[0].weathers[0]);  
    print("明日： " + httpData[0].timeSeries[0].areas[0].weathers[1]);  
    print("明後日： " + httpData[0].timeSeries[0].areas[0].weathers[2]);  
    print(httpData[0].timeSeries[0].areas[1].area.name); //北部  
    print("今日： " + httpData[0].timeSeries[0].areas[1].weathers[0]);  
    print("明日： " + httpData[0].timeSeries[0].areas[1].weathers[1]);  
    print("明後日： " + httpData[0].timeSeries[0].areas[1].weathers[2]);  
  
    noLoop();  
  }  
}
```

図 11.11: 佐賀県の気象予報を気象庁から取得するプログラム (JSON 版、その 2)

1 11.2.3 佐賀の気象予報データをキャンバスにビジュアルに表示する

2 さて、いよいよ図 11.1 の描画です。プログラムの改造は簡単です。図 11.11 において `print()`
3 を用いて出力していた処理を 6.3.2 項 (79 ページ) で学んだキャンバスへの文字列描画関数 `text()`
4 を用いる処理へ変更します。それに伴ってキャンバスのサイズ、キャンバスの背景色にも少し修正
5 を加えます。図 11.12 がそのプログラムです。

6 11.2.4 佐賀県のデータを福岡県のデータに変更する

7 福岡県の気象予報データの URL は、佐賀県の URL の 410000 を 400000 に変えた以下の通り
8 です。

9 `"https://www.jma.go.jp/bosai/forecast/data/forecast/400000.json"`

10 図 11.11 のプログラムにこの変更を加えて表示すると、福岡県の二つの地方の表示を得ることがで
11 きます。しかし福岡県には気象区分で四つの地方があります。それを考慮するには、図 11.11、図
12 11.12 共に不備です。

13 地方の数は県によって様々なため、この際、`for` 文を使って地方の数だけ繰り返すプログラムを
14 作りましょう。それが図 11.13 です。なお、佐賀県、福岡県以外の URL も単に 410000、400000
15 を特定の別の数字列に変えるだけです。たとえば以下にその情報が載っています。図 11.13 のプロ
16 グラムで試してみてください。

17 気象庁の API と予報区のコード, <https://anko.education/webapi/jma>

前略 (この部分は直前のプログラムと同じ)

```
function setup() {
  createCanvas(500, 400);
  background(255, 0, 0);
}

function draw() {
  if (httpData != null) {
    background(100, 100, 255); // 背景を明るい青に変更

    size = 15;          // 文字サイズ 15 画素
    textSize(size);
    fill(255);          // 文字は白
    text(httpData[0].publishingOffice, // 佐賀地方気象台
          20, size); // 第 2、3 引数に描画位置を指定
    text(httpData[0].timeSeries[0].areas[0].area.name, // 南部
          30, 8 + 2 * size);
    text("今日： " + httpData[0].timeSeries[0].areas[0].weathers[0],
          40, 10 + 3 * size );
    text("明日： " + httpData[0].timeSeries[0].areas[0].weathers[1],
          40, 10 + 4 * size );
    text("明後日： " + httpData[0].timeSeries[0].areas[0].weathers[2],
          40, 10 + 5 * size );
    text(httpData[0].timeSeries[0].areas[1].area.name, // 北部
          30, 18 + 6 * size);
    text("今日： " + httpData[0].timeSeries[0].areas[1].weathers[0],
          40, 20 + 7 * size );
    text("明日： " + httpData[0].timeSeries[0].areas[1].weathers[1],
          40, 20 + 8 * size );
    text("明後日： " + httpData[0].timeSeries[0].areas[1].weathers[2],
          40, 20 + 9 * size );

    noLoop();
  }
}
```

図 11.12: 佐賀県の気象予報を気象庁から取得するプログラム (JSON 版、その 3)

前略 (この部分は直前のプログラムと同じ)

```
function draw() {
  if (httpData != null) {
    background(100, 100, 255); // 背景を明るい青に変更

    size = 15;          // 文字サイズ 15 画素
    textSize(size);
    fill(255);          // 文字は白
    text(httpData[0].publishingOffice, // 佐賀地方気象台
          20, size);
    for (i = 0; i < httpData[0].timeSeries[0].areas.length; i = i+1) {
      text(httpData[0].timeSeries[0].areas[i].area.name, // 地方
            30, 8 + (2+5*i) * size );
      text("今日： " + httpData[0].timeSeries[0].areas[i].weathers[0],
            40, 10 + (3+5*i) * size );
      text("明日： " + httpData[0].timeSeries[0].areas[i].weathers[1],
            40, 10 + (4+5*i) * size );
      text("明後日： " + httpData[0].timeSeries[0].areas[i].weathers[2],
            40, 10 + (5+5*i) * size );

    }
    noLoop();
  }
}
```

図 11.13: 佐賀県の気象予報を気象庁から取得するプログラム (JSON 版、その 4)

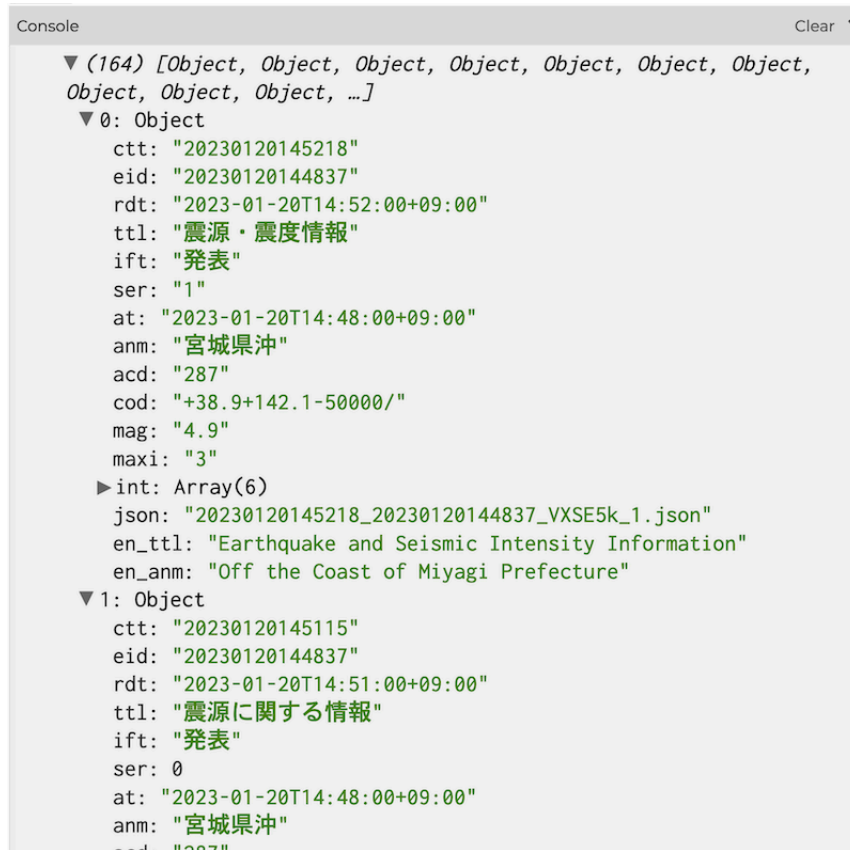


図 11.14: 気象庁地震データの内容

1 11.3 p5.js プログラムによる Web サーバへのアクセス（地震情報）

2 前節の方法を応用すれば、インターネットに公開されている様々な JSON データを取得し、p5.js
3 プログラムで表示可能です。

4 たとえば気象庁の地震情報は以下から取得できます。

5 <https://www.jma.go.jp/bosai/quake/data/list.json>

6 この JSON データの内容は、前節の方法で解析すると図 11.14 の通りです。これを元に作ったプ
7 ログラムが図 11.15 です。直近の五つの地震を表示することができます。描画例は図 11.16 です。

8 11.4 p5.js プログラムによる Web サーバへのアクセス（その他）

9 ここまで述べてきたように、JSON データの取得と表示はうまく行きます。

10 そこで次に、たとえばヤフーのホームページ：

11 <https://www.yahoo.co.jp>

```

httpData = null;

function preload() {
  httpGet(
    "https://www.jma.go.jp/bosai/quake/data/list.json",
    "json",
    setHttpData
  );
}

function setHttpData(data) {
  httpData = data;
}

function setup() {
  createCanvas(500, 400);
  background(255, 0, 0);
}

function draw() {
  if (httpData != null) {
    background(100, 100, 255);

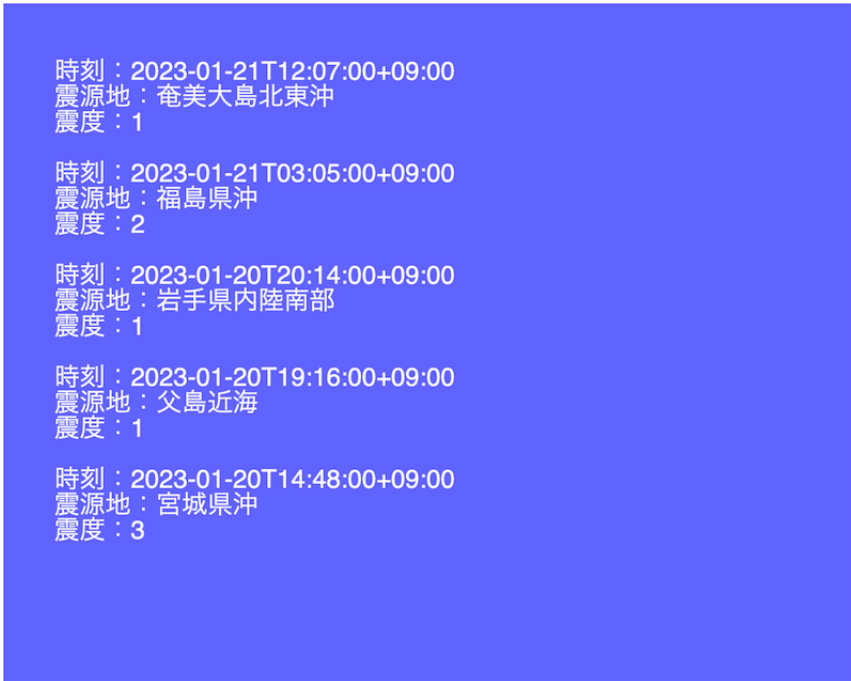
    size = 15;
    textSize(size);
    fill(255);

    for (i = 0; i < 5; i = i+1) {
      text("時刻："+httpData[i].at, 30, (3+4*i)*size);
      text("震源地："+httpData[i].anm, 30, (4+4*i)*size);
      text("震度："+httpData[i].maxi, 30, (5+4*i)*size);
    }

    noLoop();
  }
}

```

図 11.15: 気象庁の地震データをキャンバスに描画するプログラム



時刻：2023-01-21T12:07:00+09:00
震源地：奄美大島北東沖
震度：1

時刻：2023-01-21T03:05:00+09:00
震源地：福島県沖
震度：2

時刻：2023-01-20T20:14:00+09:00
震源地：岩手県内陸南部
震度：1

時刻：2023-01-20T19:16:00+09:00
震源地：父島近海
震度：1

時刻：2023-01-20T14:48:00+09:00
震源地：宮城県沖
震度：3

図 11.16: 図 11.15 のプログラムの描画例

- 1 同じ方法でアクセスしてみるのはいかがでしょうか。この場合、残念ながら Web サーバが接続を
- 2 拒否し、うまく行きません。というのは、最近の Web サーバは悪意のあるプログラムからのアク
- 3 セスを防止するため、様々な設定を行わない限り、直接のアクセスは許さない方針だからです。こ
- 4 れについてはこの講義の範囲を超えるため、解説を省略します。

1 第12章 まとめ

2 この講義では、次の事柄を学びました。

- 3 ● コンピュータの種類、用途、歴史、基本構成
- 4 ● p5.js を用いたプログラミング（基本、グラフィクス、計算、インターネット）
- 5 ● その他（アルゴリズムの概念、小数点数、様々なプログラミング言語）

6 プログラミングの理論に興味のある人は講義担当者の教養科目「情報科学の世界 I」において、
7 より数学的な内容をプログラミング言語 Scratch を使いながら学ぶことができます。

8 残念ながら、講義担当者の専門から少し離れた以下の分野の内容は手薄でした。興味のある人は
9 自学をお薦めします。あるいは学内の関連科目を履修することもひとつの方法と思います。

- 10 ● ネットワーク
- 11 ● 暗号
- 12 ● データベース
- 13 ● 人工知能

14 (以上)