高等学校情報科「情報Ⅰ」教員研修用教材

Swift 版（第 3 章のみ）

※本書の Swift は iOS バージョン 12.1.4 および Swift Playgrounds アプリバージョン 2.2 にて動作確認済み

文部科学省
MINISTRY OF EDUCATION,
CULTURE, SPORTS,
SCIENCE AND TECHNOLOGY-JAPAN

第３章　コンピュータとプログラミング

◆本単元の学習内容

【学習内容の全体像】

(3)コンピュータとプログラミング

(ア) コンピュータの仕組み	(1) コンピュータの仕組み コンピュータの構成，演算の仕組み，AND・OR・NOT，真理値表 (2) 計算誤差 計算誤差，プログラミングを使った計算誤差の確認
(イ) アルゴリズムとプログラミング	(1) 外部装置との接続 計測・制御，センサ，アクチュエータ，計測・制御プログラム (2) 基本的プログラム アルゴリズム，プログラム，フローチャート，順次・分岐・反復，変数 (3) 応用的プログラム 配列，乱数，関数，WebAPI (4) アルゴリズムの比較 探索アルゴリズムの比較，ソートアルゴリズムの比較
(ウ) モデル化とシミュレーション	(1) モデル化とシミュレーション モデル，モデルの分類，プログラミングを使ったシミュレーション (2) 確定モデルと確率モデル 確定モデルのシミュレーション，確率モデルのシミュレーション (3) 自然現象のモデル化とシミュレーション 自然現象のモデル化とシミュレーション，モデルの妥当性の検討

(全　体)

自然現象や社会現象の問題点を発見し，コンピュータやプログラミングを活用し解決策を考えられるようにする。

【学習目標】

○問題解決にコンピュータや外部装置を活用する活動を通して情報の科学的な見方・考え方を働かせて，コンピュータの仕組みとコンピュータでの情報の内部表現，計算に関する限界などについて理解させる方法を身に付ける。
○アルゴリズムを表現しプログラミングによってコンピュータや情報通信ネットワークの機能を使う方法や技能，生活の中で使われているプログラムを見いだして改善しようとすることなどを通じて情報社会に主体的に参画しようとする態度を育成する方法を身に付ける。
○モデル化やシミュレーションの考え方を様々な場面で活用できるようにするために，問題発見や解決に役立て，問題の適切な解決方法を考える力を育成する方法を身に付ける。

【本単元の取扱い】

○「情報 I」の (4) 情報通信ネットワークとデータの活用との関連も意識しながら，授業を組み立てる。
○（ウ）について，自然現象や社会現象などの問題を主体的に発見し，解決策を考える活動を取り入れる。

【中学校までの学習内容との関連】

○コンピュータとプログラミングについては，中学校技術・家庭科技術分野の内容「D 情報の技術」の学習内容である計測・制御のプログラミングなどの項目を踏まえて，情報と情報技術を活用した具体的な問題解決の中で扱う。

学習11 コンピュータの仕組み

■研修内容

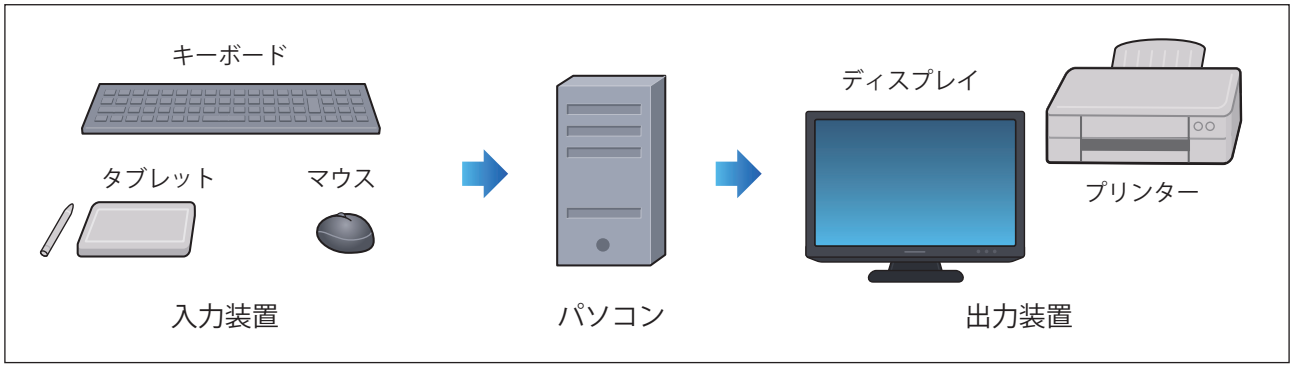
【研修の目的】

- □ コンピュータの構成，各構成要素の働きを理解する。
- □ AND，OR，NOT について理解する。
- □ コンピュータは論理回路などから構成されており，演算の基本は論理演算であることについて理解する。
- □ コンピュータでの計算の仕組みについて，グループ活動での役割を演じる活動を通じて，生徒に考えさせる授業ができるようになる。
- □ コンピュータが数値計算で誤差が生じるのは，取り扱うデータがビット数の限られた有限のものであることについて理解する。
- □ プログラミング言語を活用し，計算誤差の原因と問題点を生徒に考えさせる授業ができる。

コンピュータの仕組み

（1）構成要素

ブラウザのアイコンをクリックして検索をする時について考えてみよう。この時，マウスやキーボードで入力してコンピュータが動き，画面に結果が表示される。



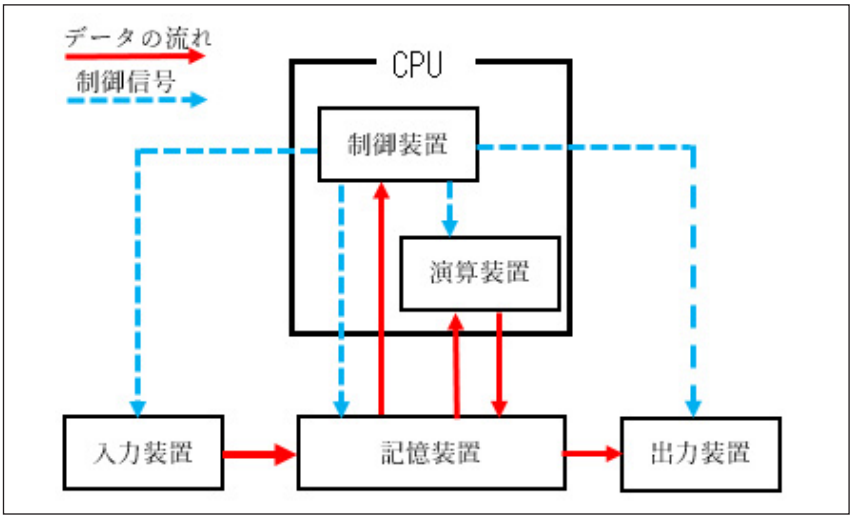
図表 1　入力と出力

さらにコンピュータの中の様子について考えてみよう。コンピュータは，制御装置，演算装置，記憶装置，入力装置，出力装置の 5 つで構成される。制御装置と演算装置を合わせて CPU（Central Processing Unit）という。CPU は記憶装置から命令やデータを取り出して処理を行う。

入力装置や出力装置を含め，すべてを制御するのが制御装置，様々な演算を行うのが演算装置である。CPU の動作速度が速いほど，1 動作あたりに実行する命令や処理するデータが多いほど性能の良い CPU となる。また，用途や目的に合わせてメモリや HDD（ハードディスク）などの大きさを決める必要がある。

CPU にとっての作業場所がメモリである。十分な広さが必要であり，特に複数の仕事を同時に進行する場合は広くする必要がある。

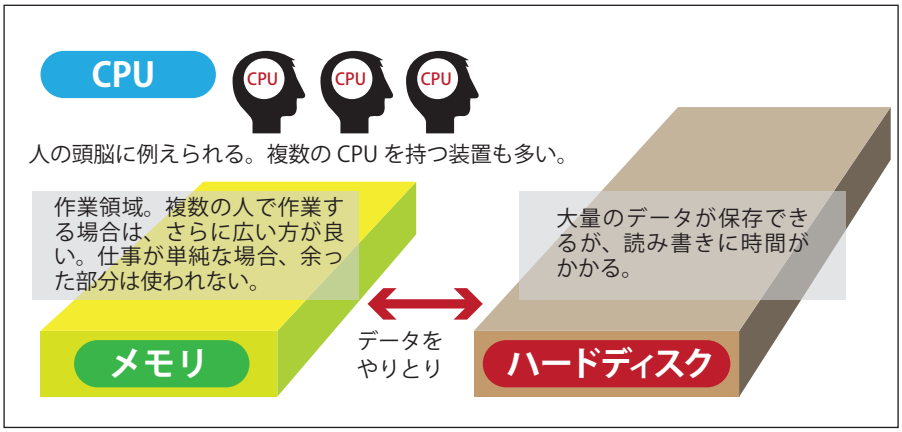
ここがせまいと，コンピュータ全体の処理速度が落ちる。ハードディ



図表 2　コンピュータの中の様子

スクは，情報を保存しておく引き出しと考えてもらいたい。大量の情報を保存できるが読み書きに時間がかかる。

メモリが少ない場合は，あふれたデータをハードディスクに保存するため，ハードディスクとのデータのやり取りにほとんどの時間がとられ，いくら CPU が高速でも実行速度は上がらない。必要十分なメモリがあって，ハードディスクのかわりにデータの読み書きの速度が速い SSD（Solid State Drive）などを使うと，コンピュータの実質的な動作速度を上げることができる。

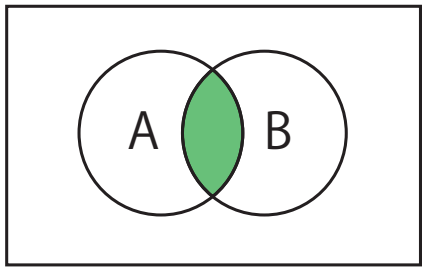


図表 3　CPU とメモリとハードディスク

（2）論理演算

現在のコンピュータの演算装置は，0 か 1 のみで表現される 2 進数を扱うように設計されている。2 進数を大きな桁で非常に高速で演算することによりすべての演算を行う。その基本的な演算は四則演算とは異なり AND，OR，NOT の 3 つである。0 を「なし」，1 を「あり」と考え，ベン図と表で表現すると以下のようになる。

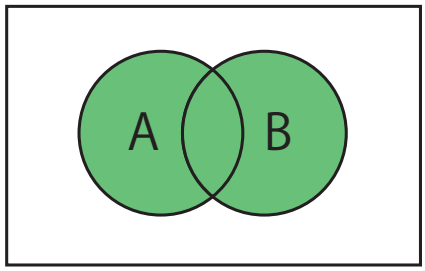
A and B　A と B の両方を含む



A and B (A,B はそれぞれ 0 または 1)
0 and 0 = 0（なしとなしの重なりはなし）
0 and 1 = 0（なしとありの重なりはなし）
1 and 0 = 0（ありとなしの重なりはなし）
1 and 1 = 1（ありとありの重なりはあり）
※and はかけ算に似ている。and を論理積という。

A	B	
0	0	0
0	1	0
1	0	0
1	1	1

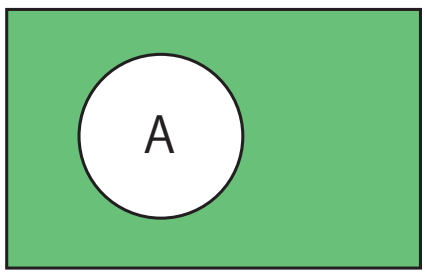
A or B　A と B のどちらかを含む



A or B (A,B はそれぞれ 0 または 1)
0 or 0 = 0（なしとなしのどちらかはなし）
0 or 1 = 1（なしとありのどちらかはあり）
1 or 0 = 1（ありとなしのどちらかはあり）
1 or 1 = 1（ありとありのどちらかはあり）
※or は足し算に似ている。or を論理和という。

A	B	
0	0	0
0	1	1
1	0	1
1	1	1

not A　A ではない



not A (A は 0 または 1)
not 0 = 1（なしではないものはあり）
not 1 = 0（ありではないものはなし）
※not で 0 は 1,1 は 0 になる。not を否定という。

A	
0	1
1	0

図表 4　論理演算

＜演習 1＞

以下の論理演算のベン図を書き，表を完成させましょう。

（1）A and B の領域を反転したもの

（2）A or B の領域を反転したもの

（3）A と B のどちらかが 1 の時は 1，A と B のどちらとも 1 の時は 0，A と B のどちらとも 0 の時は 0 となるもの

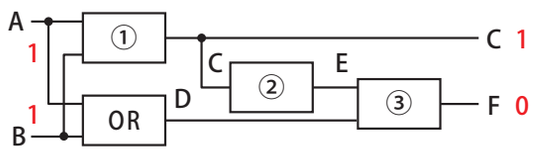
（3）論理演算を用いた二進数の足し算

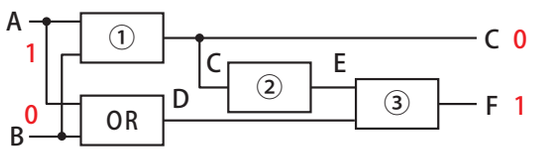
それでは，AND, OR, NOT を用いて 2 進数の足し算について考えてみる。論理演算の基本は，AND,OR,NOT なので，この 3 つで基本的にどんな計算も表すことができる。

0＋0＝ 0
0＋1＝ 1　　or で対応できる
1＋0＝ 1
1＋1＝10　　or で対応できない
A＋B＝CF　C は桁上がり，F は 1 桁目の値　※ A も B も 1 の時だけ C=1，F=0 になる。

＜演習 2＞

次の論理回路は入力の A と B がともに 1 の場合，出力の C が 1，F が 0 となります。また入力の A が 1，B が 0 の場合，出力の C が 0，F が 1 となるものです。この論理回路の①，②，③に適切な論理演算を入れ，上記の二進数の足し算を成立させましょう。





真理値表					
入力		途中経過		出力	
A	B	D	E	C	F
0	0	0	1	0	0
0	1	1	1	0	1
1	0	1	1	0	1
1	1	1	0	1	0

真理値表					
入力		途中経過		出力	
A	B	D	E	C	F
0	0	0	1	0	0
0	1	1	1	0	1
1	0	1	1	0	1
1	1	1	0	1	0

図表 5　論理回路と真理値

（4）四則演算の考え方

（3）では論理演算を用いた 2 進数の足し算を紹介した。他の計算は以下のように行う。

（引き算）

5－3＝2 の計算は，5 に 7 を足して 1 桁目の数字を用いる。

※ここで 7 は 3 に足して 10 になる数字（1 桁あがる数字，補数という）。

引き算は補数を使った足し算

（掛け算）及び（割り算）

2 進数の掛け算は左シフトと足し算，割り算は右シフトと引き算を組み合わせて筆算と同じように行う。

※シフト演算とは，2 進数で書かれたビット列を左または右にずらす操作。

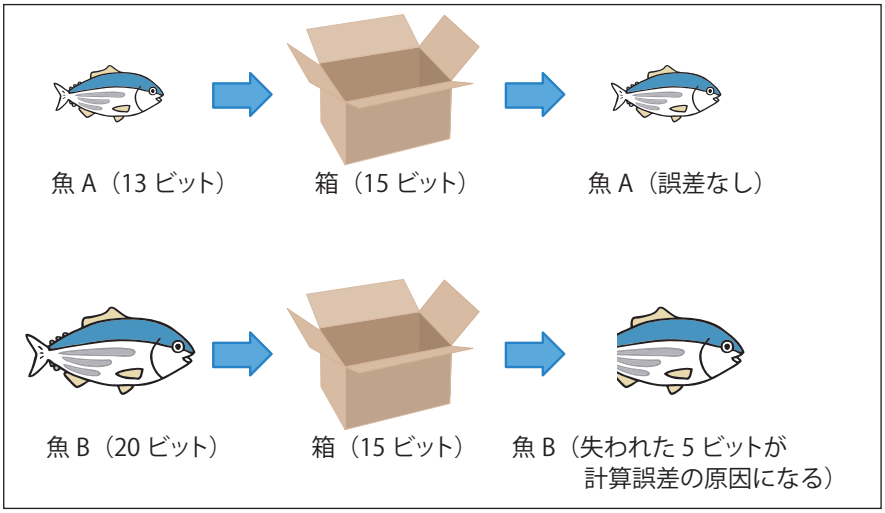
計算誤差について

（1）誤差の考え方

コンピュータで変数は指定されたビット数の箱に入っていると考える。この箱（定められたビット数）に収まるものであれば誤差は生じないが，収まらないものは誤差が生じる。

例えば，15 ビットの箱を準備したとき，13 ビットの魚 A は箱に収まるが，20 ビットの魚 B は箱に入らない。このとき失われた 5 ビットが計算誤差の原因となる。

コンピュータに計算させる時には，必要な精度に見合った箱を用意するとともに，計算の途中でこの箱からはみ出さないように注意する必要がある。



図表 6　誤差の考え方

（2）プログラミングで誤差を体験する

学習指導要領解説でプログラミング言語については，「アルゴリズムを基に平易にプログラムを記述できる」と記載されている。「平易」とは，書きやすく読みやすいことであり，誰が書いても同様のコードになることが望ましい。また，人工知能や統計の使用を含め，目的に沿ったライブラリなどが豊富にあれば多様な問題解決に対応することができる。

本 PDF 資料では Swift を使ってプログラムのコードを書く。iPad の Safari ブラウザから，以下の Web サイトを開くと，対応した Swift Playground Book を導入でき，プログラムコードを実行できる。

Swift Playgound Book の配布ページ　(Swift Playgrounds アプリを導入した iPad で開く)
https://iedtech.jp/SwiftPG/
【学習 11】,【学習 13】～【学習 17】に対応する Book　「情報 I プログラミング資料」
【学習 12】に対応する Book　「BBC micro:bit 情報 I 」

コンピュータでは，数値は小数点を含まない「整数」，または小数点を含む「浮動小数点数」のどちらかで扱われる。これらの数値は，コンピュータの内部では固定の桁数で扱われることが多い。

Double 型で指定した変数に桁数の多い数字や，その変数型で扱うことのできる最大値より大きい数字を代入して表示するプログラムと実行例を示す。

プログラム

```
1  var x = Double.greatestFiniteMagnitude    //Double 型の最大値を代入
2  vprintln(x)                               // 画面に x の値を表示
3  x = 1.797693134862315799999e+308          //Double 型の最大値に 5 桁の 9 を追加して代入
4  vprintln(x)                               // 画面に x の値を表示
5  x = 1.8e+308                              //float 型の最大値より大きな値を代入
6  vprintln(x)                               // 画面に x の値を表示
```

プログラムの実行結果

```
1.7976931348623157e+308
1.7976931348623157e+308
Inf
```

図表 7　オーバーフローの発生

1 行目の実行結果は Double 型で表示できる最大値を表示し，2 行目では小数点以下に数値を追加しても，扱える桁数を超えているため値が切り捨てられて表示され，3 行目は扱える値以上の数値は表示できないため，オーバーフローが起こっている。また，整数同士の減算と小数同士の減算のプログラムを実行してみると以下のようになる。

「整数」のみで減算した結果と「小数」のみで減算した結果を表示するプログラム

```
1  let x = 28 - 27                  //x に 28-27 の計算結果を入れる
2  vprintln(x)                      // 画面に x の値を表示
3  let y = 0.28-0.27                //y に 0.28-0.27 の計算結果を入れる
4  vprintln(y)                      // 画面に y の値を表示
```

プログラムの実行結果

```
1
0.010000000000000009
```

図表 8　コンピュータの計算誤差

小数の 0.28 から小数 0.27 を引くと，本来であれば計算結果は 0.01 になるはずだが，プログラムを実行した結果を見ると 0.01 にはならず，コンピュータの計算誤差が発生していることが分かる。

＜演習 3＞

「0.28-0.27」のような計算の誤差は一見すると微々たる誤差でしかないが，実際に惑星探査機が地球から木星まで行って帰る軌道計算に使用した場合，このような計算誤差が大きな問題になるケースがある。それはなぜか考えてみましょう。また，「0.28-0.27」の計算の誤差を起こさない方法を考えてみましょう。

■学習活動と展開

【学習活動の目的】

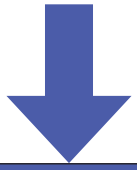
- コンピュータは論理回路などから構成されており，演算の基本は論理演算であることについて理解する。
- コンピュータが数値計算で誤差が生じるのは，取り扱うデータがビット数の限られた有限のものであることについて理解する。
- プログラミング言語を用いて，計算誤差の問題点を理解し，解決する方法を身に付ける。

◯学習活動とそれを促す問い

	問　い	学習活動
展開 1	論理演算の仕組みを図と表で表現してみよう。	AND，OR，NOT の仕組みを図と表で表現し，それらを使った演算を考える。
展開 2	プログラミング言語を用いて，2つの数の計算誤差の問題を解決しよう。	２つの数の差を計算することにより，計算誤差が生じている原因を考え，その解決方法を考える。

展開 1	
問　い	論理演算の仕組みを図と表で表現してみよう。
学習活動	・AND，OR，NOT の仕組みを図と表で表現し，それらを使った演算を考える。
指導上の留意点	・AND，OR，NOT の仕組みについては，生徒たちの身近な例に例えて説明を行う。

展開 2	
問　い	プログラミング言語を用いて，２つの数の差を計算してみよう。
学習活動	・２つの数の差を計算することにより，計算誤差が生じている原因を考え，その解決方法を考える。
指導上の留意点	・生徒の状況に応じて計算を行うための方法（表計算ソフトウェアで行うか，プログラムを作成するか）を選択する ・プログラムの作成を行う場合は，生徒の状況に応じて適切なプログラミング言語の選択を行う ・誤差がない計算をする方法は変数の型を変える方法や，整数にして計算し，計算した結果を小数にするなど，解決方法が複数あることを示す。



まとめ	
まとめ	・コンピュータの論理回路と論理回路によるコンピュータの計算の仕組みを理解し，プログラミング言語による数値計算の誤差の原因とその解決策を考えることで，コンピュータでの適切な計算方法を理解する。

学習12 外部装置との接続

■研修内容

【研修の目的】

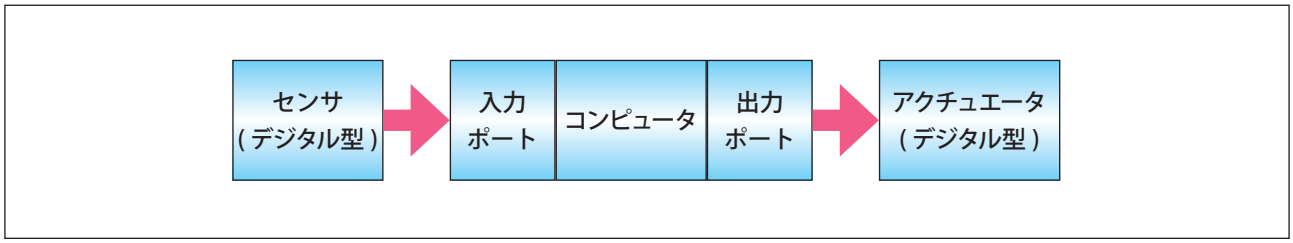
- □ コンピュータによる計測・制御システムを構成するセンサやアクチュエータ，AD コンバータや DA コンバータなどの各装置の役割とそれらの構成の仕組みを理解する
- □ センサからデータを入力したりアクチュエータへデータを出力したりする基本的なプログラムについて，これらのプログラムを作成する活動を通して，生徒に作成させる授業ができるようになる。
- □ 生活に役立つ装置などを構想させる学習活動を通して，生活における諸問題の解決を生徒に考えさせる授業ができるようになる。

（1）コンピュータによる計測・制御

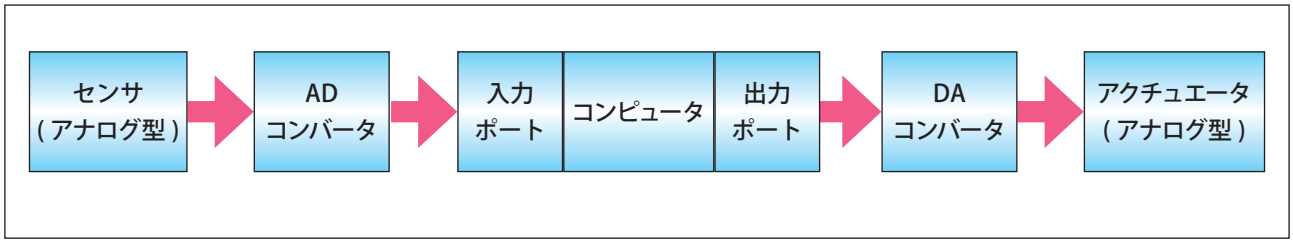
センサやアクチュエータなどの計測・制御システムを構成する要素の機能や仕組みを理解し，計測・制御システムの中では一連の情報がプログラムによって処理されており，順次，分岐，反復という処理手順の組み合わせによって実現されていることを理解する。

センサなどの装置を使って自然現象などの量を測ることを計測といい，ある目的の状態にするためにアクチュエータなどの装置を操作することを制御という。

コンピュータを使った計測は，センサなどの外部装置から得た計測データを入力ポート（入力端子）を経由してコンピュータに入力することになる。また，コンピュータを使った制御は，コンピュータから出力ポート（出力端子）を経由してアクチュエータなどの外部装置を操作するデータを出力することになる。図表 1 は二値的に動作するデジタル型のセンサやアクチュエータの場合の構成であり，連続的に動作するアナログ型のセンサやアクチュエータの場合は，アナログをデジタルに変換する AD コンバータやデジタルをアナログに変換する DA コンバータを経由させる図表 2 の構成になる。



図表 1　デジタル型のセンサやアクチュエータによるコンピュータ計測制御システム



図表 2　アナログ型のセンサやアクチュエータによるコンピュータ計測制御システム

（2）センサとアクチュエータ

センサは，自然現象の情報を人間や装置などが扱いやすい信号に置き換える装置であり，温度，照度，加速度，距離などを測定するセンサがいろいろな装置に内蔵されて使われている。

身近なセンサとして以下の様な例がある。

使用例	センサの種類	センサの役割
冷蔵庫	温度センサ	冷蔵庫内の温度を検知する。
エアコン	温度センサ・湿度センサ	室内の気温・湿度を検知する。
煙感知器	煙センサ	煙を検知する。
リモコン	赤外線センサ	赤外線による信号を検知する。
自動車	超音波，レーザー，レーダー等のセンサ	前方や後方の障害物を検知する。
スマートフォン	加速度センサ	傾きや動きを検知する。

図表 3　センサの例

また，アクチュエータはコンピュータが制御した情報を物理的に伝える装置であり，各種のモータなどが使われている。アクチュエータの動きは直進運動と回転運動に分けることができ，これらの動力源として，電気･油圧･空気圧がある。DC モータは車輪のように連続して回転させ，サーボモータはロボットの腕の関節のように，角度を指定して回転させる目的で使用する。身近なアクチュエータとして以下の様な例がある。

使用例	アクチュエータの種類	アクチュエータの役割
ドローン	ブラシレスＤＣモーター	直流によって回転運動し，制御性に優れる。プロペラの回転数を変えることで姿勢を制御する。
建設機械	油圧シリンダ	油圧によりシリンダ内のピストンを動かす。大出力が可能であり，クレーンやパワーショベルなどに使われる。
ド　ア	空気圧シリンダ	構造がシンプルで安価である。空気を圧縮膨張させシリンダ内のピストンを動かし，ドアの開閉などを行う。

図表 4　アクチュエータの例

（3）センサの値の取得とアクチュエータの制御の方法

一般的に，入力ポートに接続されたセンサの値を取得するには，ポートから入力した生（なま）のデータをセンサの特性に合わせてプログラム上で計算式によって温度などの物理量に変換する必要がある。同様に，出力ポートに接続されたアクチュエータを制御する場合も，計算式によってモーターの回転数などの物理量をポートに出力するデータに変換する必要がある。しかし，教育用プログラミング言語では，温度などの物理量を直接取得できる命令や，モータの回転数などの物理量を直接制御できる命令が用意されている場合があり，その場合は計算式による変換が不要になる。

（4）中学校技術・家庭科技術分野との接続

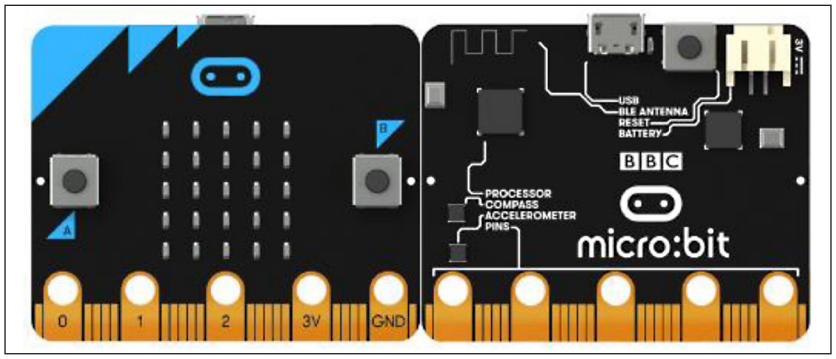
平成 29 年に公示された中学校学習指導要領では，技術･家庭科技術分野「D 情報の技術」(3) において，計測･制御のプログラミングによって解決する活動を通して，計測・制御システムの仕組みを理解し，安全・適切なプログラムの制作，動作の確認及びデバッグができること，などが示されている。図表 5 には中学校の計測・制御例で，ワンボード型のコンピュータとブロック，センサ，アクチュエータとビジュアルプログラミング言語を使ったロボットが示されている。このプログラムは，障害物までの距離を光センサで読み取り，設定値以下の場合にモータの回転を停止させるようになっている。高等学校「情報 I」では，このような中学校での学習を踏まえて指導することになる。

＜演習 1＞

(1) 身近なセンサやアクチュエータは，他にはどんなものがあるでしょうか。調べてみましょう。
(2)「身近な問題で困っていること」をテーマとして，問題を解決する装置をセンサやアクチュエータを使って実現する方法を考えてみましょう。

（5）センサの値をもとに LED を制御するプログラム

ここでは，イギリスで教育用に作られ，実践例が豊富にあり，各種のセンサと簡易な表示装置が内蔵され，多くのプログラミング言語で制御可能な外部機器として micro:bit を用い，センサ（加速度センサ）で得たデータに応じて LED を制御するプログラミングを行う。



図表 5　micro:bit

出典：「はじめよう micro:bit」（https://microbit.org/ja/guide/）

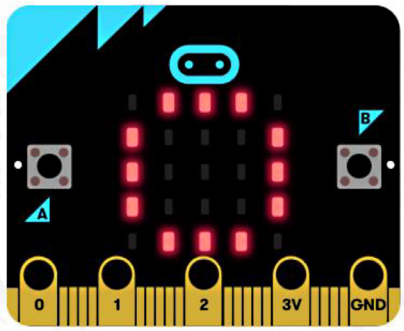
① 順次

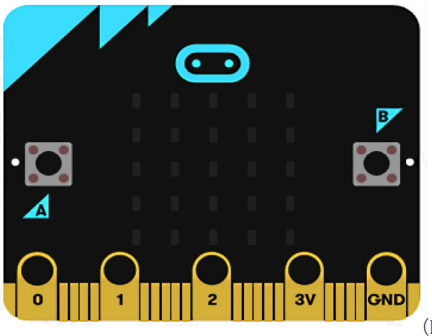
順次の例として，ここでは LED で○を 1 秒間点灯したあと消灯するプログラムを示す。

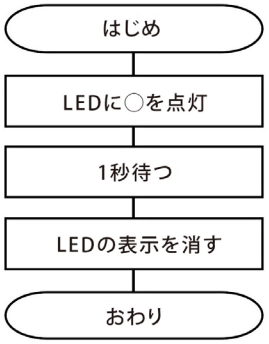
順次のプログラム

```
let frameOne = MicrobitImage("""
. X X X .
X . . . X
X . . . X
X . . . X
. X X X .
""")
frameOne.showImage()                    // イメージを表示
sleep(3)                                // 3秒停止
clearScreen()                           // 表示をクリア
```

プログラムの実行結果







「はじめよう micro:bit」（https://microbit.org/ja/guide/）を加工して作成

図表 6　順次

② 分岐

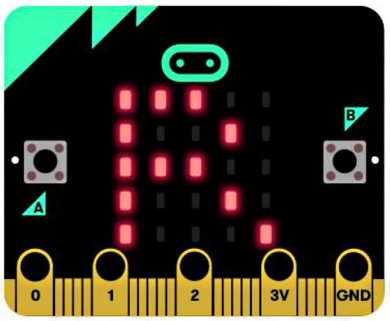
分岐の例として，micro:bit の加速度センサが右に傾いているデータを検出すれば，ＬＥＤに「R」が点灯し，そうでなければ LED に「-」が点灯するプログラムを示す。

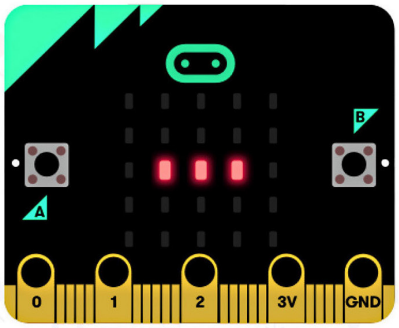
分岐のプログラム

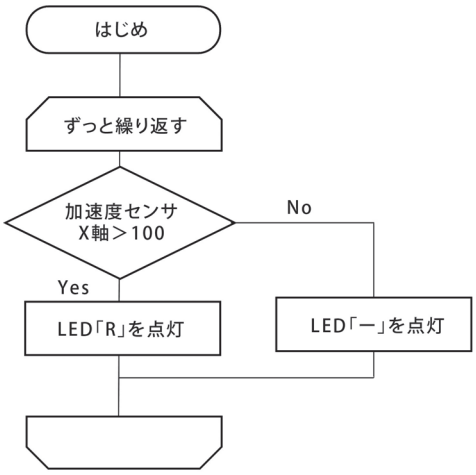
```
setAccelerometerPeriod(.ms160)
let frame1 = MicrobitImage("""
X X X X .
X . . . X
X X X X .
X . . X .
X . . . X
""")
let frame2 = MicrobitImage("""
. . . . .
. . . . .
X X X X X
. . . . .
. . . . .
""")

onAcceleration({(accelerationValues) in
    let x = accelerationValues.x
    if x > 100 {
        frame1.showImage()
    } else {
        frame2.showImage()
    }
})
```

プログラムの実行結果







図表７　分岐

「はじめよう micro:bit」（https://microbit.org/ja/guide/）を加工して作成

＜演習２＞

加速度センサでｘ軸の右の傾きにはＬＥＤに「R」と表示させることができている。同様に左に傾いたら「L」と表示させるフローチャートとプログラムを作成しましょう。

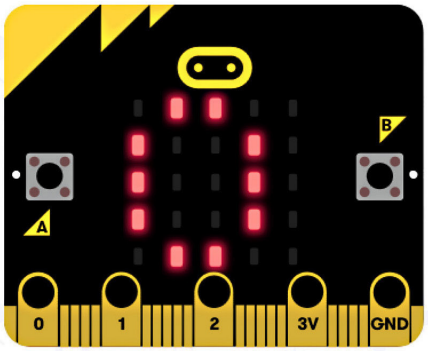
③ 反復

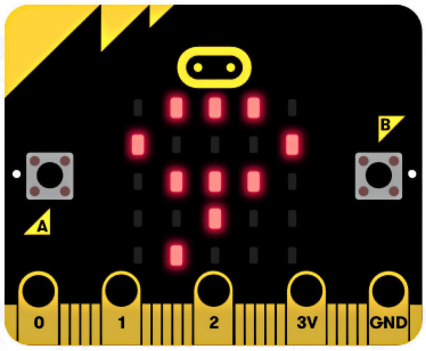
反復の例として，0 から 9 までの数字を順に表示するプログラムを示す。

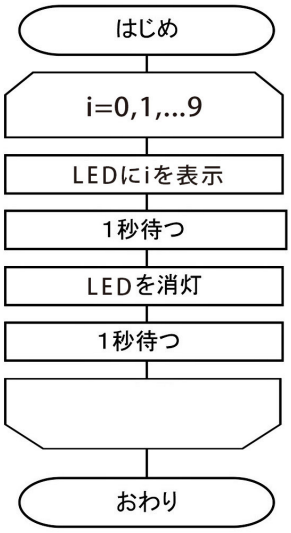
反復のプログラム

```
1  for i in 0..<10 {
2      var textToDisplay = String(i)
3      showString(textToDisplay)
4      sleep(1)
5  }
```

プログラムの実行結果







「はじめよう micro:bit」
（https://microbit.org/ja/guide/）を加工して作成

図表 8　反復

＜演習 3＞

加速度センサ以外のセンサからデータを取得し，センサのデータに応じた動きを LED やアクチュエータにさせてみましょう。

＜参考文献・参考サイト＞

・「はじめよう micro:bit」https://microbit.org/ja/guide/

■学習活動と展開

【学習活動の目的】

- コンピュータによる計測・制御システムを構成するセンサやアクチュエータ，AD コンバータや DA コンバータなどの各装置の役割を理解する。
- センサからデータを入力したりアクチュエータへデータを出力したりする基本的なプログラムを作成できるようになる。
- 生活に役立つ装置などを構想させる学習活動を通して，生活における諸問題の解決を生徒が解決できるようになる。

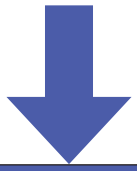
◯学習活動とそれを促す問い

	問　い	学習活動
展開 1	センサとアクチュエータを使って身近な問題点を解決する方法を考えてみよう。	身近な問題点を発見し，センサやアクチュエータでその問題点が解決できないか考える。
展開 2	プログラムに問題が発生した場合に，原因を見つけ解決してみよう。	プログラムに問題が発生した場合に，原因を見つけ出し，解決策を考える。
展開 3	センサから入力された値に応じて，動きが変わるプログラムを作成しよう。	センサから入力された値に応じて，LED やアクチュエータの動きが変わるプログラムを作成する。

展 開 1	
問　い	センサとアクチュエータを使って身近な問題点を解決する方法を考えてみよう。
学習活動	・身近な問題点を発見し，センサやアクチュエータでその問題点が解決できないか考える。
指導上の留意点	・他のセンサの種類についても，その仕組みを説明し，様々なセンサやアクチュエータを組み合わせたプログラミングを行わせる

展開 2	
問　い	プログラムに問題が発生した場合に，原因を見つけ解決してみよう。
学習活動	・プログラムに問題が発生した場合に，原因を見つけ出し，解決策を考える。
指導上の留意点	・プログラミングで解決する方法は 1 つではないことを伝え，様々な意見を出すよう生徒に促す。

展開 3	
問　い	センサから入力された値に応じて，動きが変わるプログラムを作成しよう。
学習活動	・センサから入力された値に応じて，LED やアクチュエータの動きが変わるプログラムを作成する。
指導上の留意点	・加速度センサ以外のセンサや，アクチュエータに出力させるなど，さまざまなセサとアクチュエータを活用したプログラムを作成させる。



まとめ	
まとめ	・センサとアクチュエータをコンピュータで制御していることを理解し，身近な問題を解決する装置のアイデアを練り，そのアイデアをプログラミングで形にすることができるようになる。

学習13 基本的プログラム

■研修内容

【研修の目的】

- □ すべてのアルゴリズムが「順次」,「分岐」,「反復」の要素の組合せで構成されていることについて理解する。
- □ アルゴリズムを流れ図などを用いて図式化することができるようになる。
- □ 意図する処理がどのようにすればコンピュータに伝えることができるかについて，プログラムを制作する活動を通じて，生徒に考えさせる授業ができるようになる。
- □ 「順次」,「分岐」,「反復」の構造を持つ基本的プログラムについて，プログラムを作成する活動を通じて，生徒に考えさせる授業ができるようになる。

（1）アルゴリズムとプログラミング

問題を解決するための方法や手順をアルゴリズムといい，アルゴリズムをコンピュータが実行できる形式であらわしたものをプログラムという。アルゴリズムは流れ図（フローチャート）などを用いて図式化するとわかりやすい。アルゴリズムはプログラミング言語を用いてプログラムにすることで，コンピュータに実行させることができる。

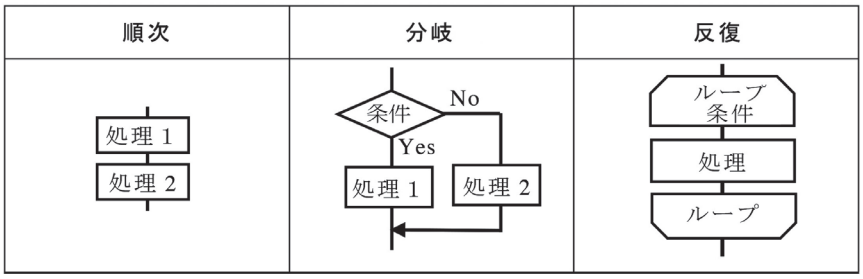
（2）制御構造

どのようなアルゴリズムでも，処理の流れは，順次，分岐，反復の３つの構造の組み合わせで構成されている。このような処理の流れを制御構造という。

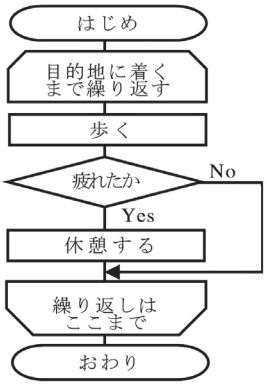
- ・順次　１つ１つの処理を順番に行う
- ・分岐　ある条件に応じて異なる処理を実行する
- ・反復　ある条件が満たされている間はその処理を繰り返し実行する

（3）流れ図（フローチャート）

流れ図を使って表現すると，処理の流れを視覚的に表現することが可能となる。制御構造を流れ図で表現するにあたり，各構造は図表１のようになり，３つの構造を組み合わせて流れ図を書くと，図表２のように表現することができる。



図表１　制御構造



図表２　歩行

- 順次
  上から下へ記述された順に処理を実行する。処理 1 を実行した後に処理 2 を実行する。
- 分岐
  条件により処理を選択する。条件が真のときは処理 1 を実行し，偽のときは処理 2 を実行する。
- 反復
  処理を繰り返し実行する。条件が真の間，処理を繰り返し実行し，偽になるとループ終端の下にある処理を実行する。

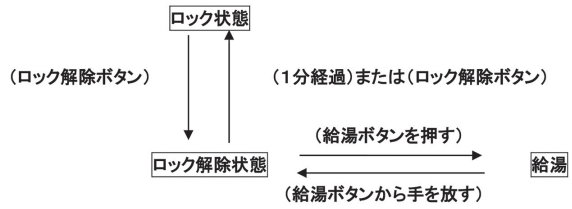
＜演習 1＞

「ペットボトルのふたを開ける」という動作を　①順次　②分岐　③反復　の３つの構造を使ってアルゴリズムとして表現しましょう。また，表現したアルゴリズムを流れ図を使って図式化しましょう。

アルゴリズムを表現する方法は多様である。ここでは，フローチャートを紹介したが，目的や内容に応じて適切なものを使い分けるようにしたい。

給湯ポット（ボタンの種類：ロック解除ボタン、給湯ボタン）

【状態遷移図】□状態　（　）イベント



【状態遷移表】

イベント ＼ 状態	ロック状態	ロック解除状態	給湯
（ロック解除ボタン）	ロック解除	ロック	■
（1 分経過）	■	ロック	■
（給湯ボタンを押す）	■	給湯	（給湯継続）
（給湯ボタン手からを放す）	■	■	（給湯停止）

※ロック解除状態でロック解除ボタンを押しても状態がロックに遷移しないメーカーもある。

図表 3　状態遷移図と状態遷移表の例

※ UML（Unified Modeling Language）に含まれるアクティビティ図やシーケンス図のように，複数の処理が互いに通信しながら動作を進める様子を表現できる表記法も利用されている。

（4）制御構造のプログラム例

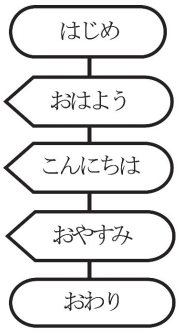
本学習では分岐構造についてプログラムのコードを書く。本書で使用する Swift の関数等の説明は以下の図表 4 に示す。なお，関数はプログラムコード内の定義で，文法ではない。本書で用いている関数は独自に用意したもので，Swift の他の環境では動作しない

関数等	機　能
vprint(" 文字列 ")	改行しないで文字列を表示する
vprintln(“ 文字列 ”)	改行を付けて文字列を表示する。

図表 4　本資料で独自に用いている関数の説明

■順次の例

今回の例では「おはよう」「こんにちは」「おやすみ」の順に画面にメッセージが表示されるプログラムを作成するものとする。流れ図は以下の図表 5 の通りになり，プログラミングの処理を日本語で記述したリストは図表 6 の通りになる。



図表 5　流れ図

1：“ おはよう ” と画面に表示
2：“ こんにちは ” と画面に表示
3：“ おやすみ ” と画面に表示

図表 6　日本語記述

図表 5 の流れ図，図表 6 の日本語記述をコードに置き換えたのが図表 7 である。処理は上の行から下の行の順に処理が進められる。

```
1  vprintln(" おはよう ")      // おはようと画面に表示
2  vprintln(" こんにちは ")    // こんにちはと画面に表示
3  vprintln(" おやすみ ")      // おやすみと画面に表示
```

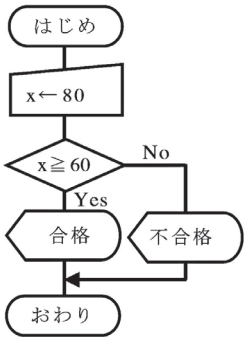
図表 7　コード

＜演習 2＞

図表 7 のコードにおいて，「こんにちは」と「おやすみ」の間に「こんばんは」と表示されるようにプログラムを追加しましょう。

■分岐の例

点数 x が 60 点以上ならば，「合格」を表示し，そうでなければ「不合格」を表示するプログラムを作成するものとする。流れ図は以下の図表 8 の通りになり，プログラミングの処理を日本語で記述したリストは図表 9 の通りになる。ただし x の値は 80 とする。



図表 8　流れ図

1：x =80 とする。
2：もし x >= 60 ならば「合格」とする。
3：それ以外の場合は「不合格」と表示する。

図表 9　日本語記述

図表 8 の流れ図，図表 9 の日本語記述をコードに置き換えたのが図表 10 である。今回の例では x=80 であるので，x>=60 の条件を満たし，「合格」と表示がされる。このように分岐の場合は「『合格』と表示する処理」と「『不合格』と表示する処理」は同時に実行されることがないことに留意が必要である。

```
1  let x = 80                  //x = 80 とする
2  if (x >= 60) {              //x が 60 以上のときの処理
3      vprintln(" 合格 ")
4  } else {                    //x が 60 以上でないときの処理
5      vprintln(" 不合格 ")
6  }
```

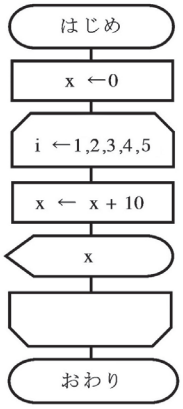
図表 10　コード

＜演習 3＞

（1）図表 10 のコードにおいて，x の値だけを変更して「不合格」と表示されるようにプログラムを変更しましょう。
（2）図表 10 のコードを参考に「30 点未満ならば『再試験』そうでなければ『合格』と表示される」プログラムを作成しましょう。

■反復の例

変数 x に対して 10 を 5 回加算をしながらその都度 x の値を表示するプログラムを作成するものとする。流れ図は以下の図表 11 の通りになり，プログラムの処理を日本語で記述したリストは図表 12 の通りになる。ただし x の初期値は 0 とする。



図表 11　流れ図

```
1 ：x = 0 とする。
2 ：i = 1,2,3,4,5 とカウントを進めながら
                    以下の処理を繰り返し行う。
    2 － 1 ：x に 10 を加える。
    2 － 2 ：x の値を表示する。
```

図表 12　日本語記述

図表 11 の流れ図，図表 12 の日本語記述をコードに置き換えたのが図表 13 である。

今回の例では 10 を 5 回加えるので，最終的な x の値は 50 となる。また，カウントのための変数 i をプログラムの中で使用することも可能であり，「繰り返している回数をその都度表示する」といった処理も行うことができる。

```
1  var x = 0
2  vprintln("x = " + String(x))
3  for i in 1..<6 {        //for ループで 1 から 6 未満の間 1 ずつカウントしながら繰り返す
4      x += 10
5      vprintln("x = " + String(x))
6  }
```

図表 13　コード

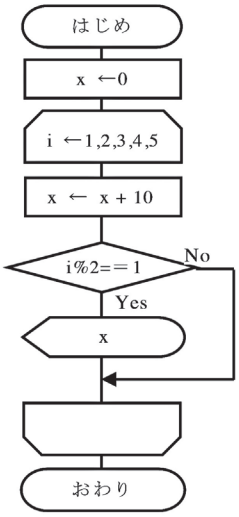
＜演習 4＞

（1）図表 13 のコードを参考に「x の値を表示すると同時に繰り返している回数をその都度表示する」プログラムを作成しましょう。
（2）図表 13 のコードを参考に繰り返す回数だけを変更して「x の値が 100 になる」プログラムを作成しましょう。
（3）図表 13 のコードを参考に「x = x + 10」の部分だけを変更し，「1+2+3+4+5」の値を表示するプログラムを作成しましょう。

■分岐と反復を組み合わせた例

分岐と反復を組み合わせると，「繰り返していく中である条件を満たす場合だけ特定の処理を行う」ことが可能となり，アルゴリズムの表現の幅を広げることができる。

ここでは変数 x に対して 10 を 5 回加算をするが，「繰り返している回数が奇数回の場合だけ」x の値を表示するプログラムを作成するものとする。流れ図は以下の図表 14 の通りになり，プログラムの処理を日本語で記述したリストは図表 15 の通りになる。ただし x の初期値は 0 とし，奇数であることを判定する式は 2 で割った余りが 1 であることを示す「i%2==1」を使用するものとする。



図表 14　流れ図

1 ：x = 0 とする。
2 ：i = 1,2,3,4,5 とカウントを進めながら
　　以下の処理を繰り返し行う。
　2 － 1 ：x に 10 を加える。
　2 － 2 ：もし i%2==1 ならば x の値を表示する。

図表 15　日本語記述

図表 14 の流れ図，図表 15 の日本語記述をコードに置き換えたのが図表 16 である。今回の例では 10 を 5 回加えるが，i が奇数の時だけ x の値を表示しているので，x = 10,30,50 の場合だけ値が画面に表示される。

```
1  var x = 0
2  vprintln("x = " + String(x))
3  for i in 1..<6 { //for ループ
4      x += 10
5      if( i%2==1 ) {                          //2 で割った余りが 1 の時の処理
6          vprintln("x = " + String(x))
7      }
8  }
```

図表 16　コード

＜演習 5 ＞

図表 16 のコードを参考に「i が偶数の時だけ x に 10 を加え，その都度 x の値を表示する」プログラムを作成しましょう。

■学習活動と展開

【学習活動の目的】

- 分岐とそのプログラムを理解し，作成する方法を身に付ける。
- 反復とそのプログラムを理解し，作成する方法を身に付ける。
- 分岐と反復を組み合わせたプログラムを理解し，作成する方法を身に付ける。
- 分岐と反復を組み合わせると様々な手順を表現することができることを理解する。

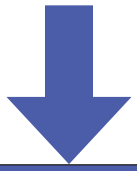
○学習活動とそれを促す問い

	問　い	学習活動
展開 1	分岐のプログラムを作ってみよう。	分岐について考え，そのプログラムを作成する。
展開 2	反復のプログラムを作ってみよう。	反復について考え，そのプログラムを作成する。
展開 3	分岐と反復を組み合わせたプログラムを作ってみよう。	分岐と反復を組み合わせたプログラムを作成し，反復処理の最中に条件分岐していることを確認する。

展開 1	
問　い	分岐のプログラムを作ってみよう。
学習活動	・分岐について考える。 ・分岐のプログラムを作成する。
指導上の留意点	・分岐は２択の処理であり，条件によって処理する内容が変化することを解説する。 ・分岐のプログラムを作成する際には，値を変化することによって処理が変化していることを確認させる。

展開 2	
問　い	反復のプログラムを作ってみよう。
学習活動	・反復について考える。 ・反復のプログラムを作成する。
指導上の留意点	・反復の説明においては反復する回数と反復する回数に応じて変化する変数がどのように変化しているかを意識させる。 ・反復のプログラムを作成する際には，反復する回数と反復する回数に応じて変化する変数がどのように変化しているかを画面表示等で確認するとよい。

展開 3	
問　い	分岐と反復を組み合わせたプログラムを作ってみよう。
学習活動	・分岐と反復を組み合わせたプログラムを作成する。
指導上の留意点	・プログラムを作成する際には，反復する回数と反復する回数に応じて変化する変数がどのように変化しているかを把握させ，分岐の条件による処理の変化が自分の想定しているとおりか確認させる。



まとめ	
まとめ	・分岐と反復を組み合わせるとアルゴリズムの表現の幅が飛躍的に広がることを具体的なアルゴリズムの例を通して理解させる。

学習14 応用的プログラム

■研修内容

【研修の目的】

- □ プログラム上で配列や乱数などの関数を取り扱うための具体的な方法について理解する。
- □ 関数を用いてプログラムをいくつかのまとまりに分割してそれぞれの関係を明確にして構造化できるようになる。
- □ API の概念を理解し，外部のビッグデータや処理機能を活用できるようになる。

（1）配列

プログラムは，変数にデータを代入したり，参照したりすることを繰り返しながら動作しているが，複雑なプログラムになってくると，変数名の異なる変数が多くなり，取り扱いが困難になるため，配列が使われることが多い。配列は複数の値を一つの名前（変数名）と番号（添字）によって管理する仕組みである。これを使うと同じデータ型の変数を複数宣言する必要がなくなるため，簡潔にプログラムを記述することができるようになる。その配列の中にあるひとつひとつの変数を要素といい，要素に付けられた番号を添字という。

配列の値の参照や配列への値の代入は，配列名と添字を使って，「配列名 [添字]」として扱うことが多い。添字を変数で指定することで，その値を変えることで要素を自由に扱うことができる。

配列名 a

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
56	3	62	17	87	22	36	83	21	12

図表 1　配列の例

■配列を用いたプログラムの例

10 個の数値からなる配列があったときに，配列内の数値の一部を取り出し，足し算するプログラムを以下の図表 2 に示す。

Swift では配列は 0 番目から始まることに留意する必要がある。このプログラム例では配列 a の 3 番目と 7 番目の数値を足し算している。a[3]=17，a[7]=83 であるので，足し算の結果は 100 が表示される。

```
1  var a = [ 56,3,62,17,87,22,36,83,21,12]        // 配列 a の定義
2  var sum = 0
3  sum = a[3]+a[7]
4  vprintln(sum)
```

図表 2　配列を用いたプログラムの例 1

次は 10 個の数値からなる配列があったときに，配列内の数値の合計を求めるプログラムを以下の図表 3 に示す。

このプログラムの実行結果は sum=56+3+62+17+87+22+36+83+21+12=399 となる。なお a[i] で i の値が変化するたびに配列内を巡回していることに留意すること。

```
1  var a = [ 56,3,62,17,87,22,36,83,21,12]
2  var sum = 0
3  for i in 0..<10 {
4      sum = sum + a[i]      //a[0] ～ a[9] まで全て加えていく
5  }
6  vprintln(sum)
```

図表 3　配列を用いたプログラムの例 2

＜演習 1＞

図表 3 のコードを参考にして「配列 a の中の最小値を表示する」プログラムを作成しましょう。

（2）乱数

ある一定の範囲内において，すべての数が同じ確率で現れるような数のことを乱数という。コンピュータはその特質上計算で値を生成するので計算による擬似的な乱数しか生成できない。これを擬似乱数（以後「乱数」と表現する）という。この乱数をプログラム内で用いることにより，実行するたびに処理内容が変わるような変則的なプログラムを作成することが可能となり，プログラムを現実世界のシミュレーションなどに活用することが可能となる。

■乱数を用いたプログラムの例

ある値 a があるときに，0 ～ 9 までの数をランダムに発生させる乱数 r と比較して，大きい場合に「a の方が大きい」，小さい場合に「a の方が小さい」，等しい場合に「当たり」と表示するプログラムを図表 4 に示す。

```
1  var a = 5
2  var r = Int.random(in: 0..<10)     //0 ～ 9 までの値をランダムに発生
3  if( a == r ) {
4      vprintln(" 当たり ")
5  } else if ( a>r ){
6      vprintln("a の方が大きい ")
7  } else {
8      vprintln("a の方が小さい ")
9  }
```

図表 4　乱数を用いたプログラムの例 1

また，乱数の生成の範囲を「0 から」ではなく「1 から」に変更したい場合も少なからずある。このような場合は生成した乱数に対して 1 を加えることで，生成した乱数に対して一律に計算がなされるので，0 ～ 9 の範囲を 1 ～ 10 の範囲に変更することが可能となる。

以下の図表 5 は「1 ～ 10」までの乱数を表示するプログラムのコードである。「Int.random(in: 0..< 10)+1」とすることにより「0 ～ 9」から「1 ～ 10」に変更することが可能となる。

```
1  var a = 5
2  var r = Int.random(in: 0..<10)+1    //1 ～ 10 までの値をランダムに発生
3  vprintln(r)
```

図表 5　乱数を用いたプログラムの例 2

＜演習 2＞

上の図表 5 のコードにおいて「a と r の値を表示」するようにプログラムに追加を行い，「当たり」と表示されているときには確かに a=r であることを確認しましょう。

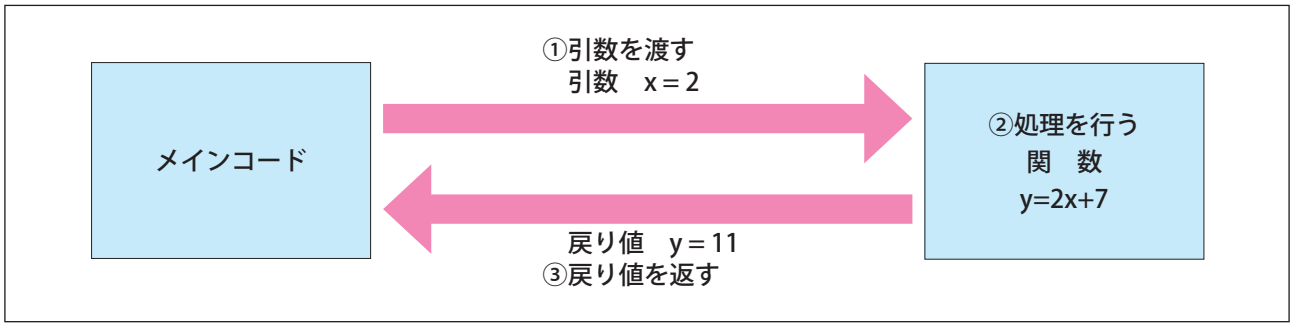
（3）関数

一般にシステム開発などで行われるプログラムは大規模なので，コードの長さも膨大となる。このような場合，文章の章，節，段落のような適切な構造を作らないと，コードの可読性は著しく低くなってしまう。

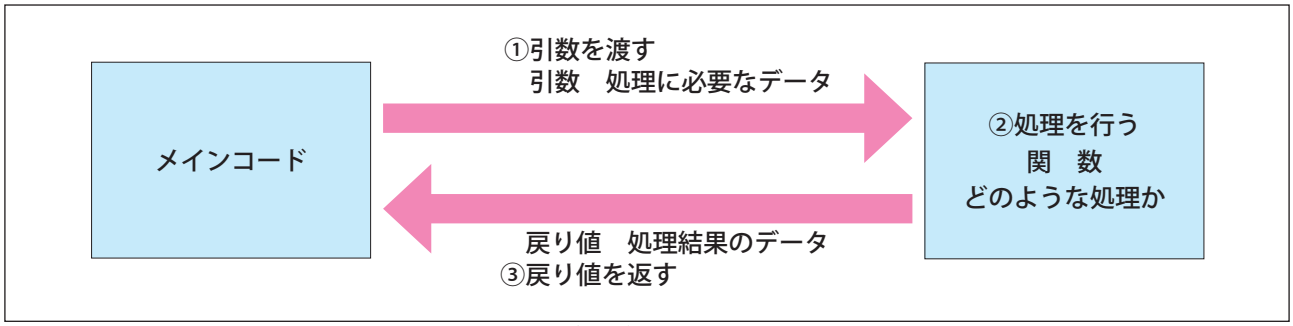
そこでプログラムの機能ごとにメインコードから切り離し，関数という単位で分割することで，プログラムを機能ごとに整理，構造化することが可能となる。また，作成した関数はメインコード内で何回でも呼び出すことが可能となるため，作成した機能を再利用することが可能となる。

数学での関数は y=f(x) のように表現されるが，コンピュータプログラムの関数では x を引数（ひきすう），y を戻り値と呼んでいる。（図表 6 参照）

実際に関数で分割するときには「どのような処理を行っているか」と同時に，引数（処理に必要なデータは何であるか），戻り値（処理結果のデータは何であるか）を把握する必要がある。（図表 7 参照）



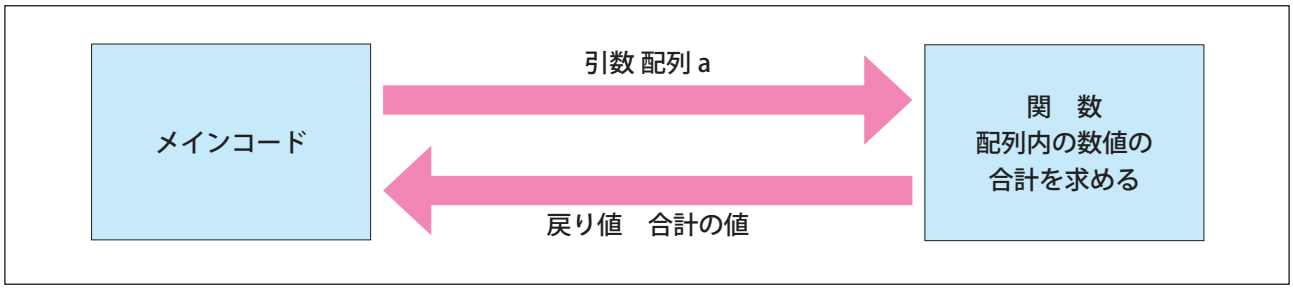
図表６　数学における関数



図表７　プログラムにおける関数

■関数で分割したプログラムの例

図表３のコードを関数に分割することを考える。関数の処理と引数と戻り値は次の通りになる。（図表８参照）



図表８　関数の処理と引数と戻り値

関数に分割した後のコードは以下の図表９である。

関数定義を扱う場合は func 文を使う。この例では dimsum() という関数で引数を a として定義しており，戻り値は return で指定する。この関数を使用するには sum=dimsum(a) としておくと，a を引数として処理結果が sum に代入される。また，[Int] は引数が整数型配列を，-> Int は戻り値が整数型であることを指定している。引数の前にあるアンダーバーは，関数呼び出し時に名前指定を省略できることを許可している。

```
1  func dimsum(_ a: [Int]) -> Int {  // 合計を求める関数 dimsum()
2      var sum = 0
3      for i in a {
4          sum = sum + i
5      }
6      return sum
7  }
8  let a = [56,3,62,17,87,22,36,83,21,12]
9  var sum = dimsum(a)     // 作った関数 dimsum() を呼び出し
10 vprintln(sum)
```

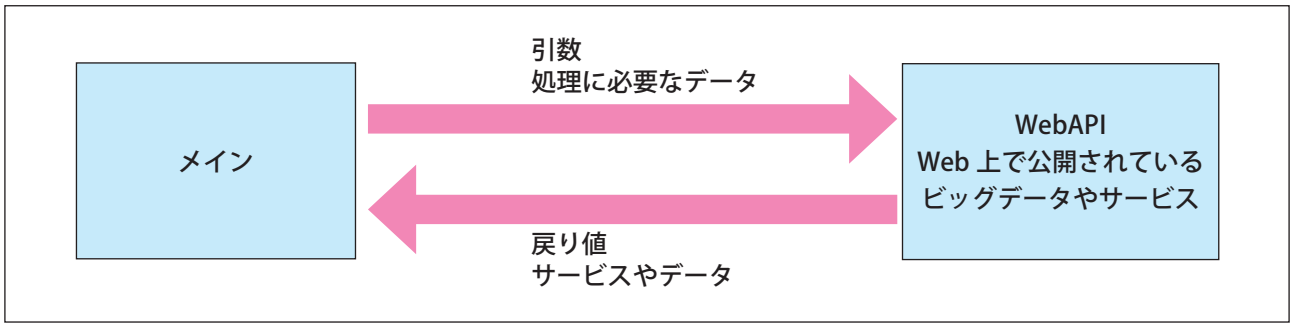
図表９　関数で分割したプログラムの例

＜演習３＞

上の図表９のコードを参考にして「指定した回数 p 回 2 を加算するプログラム」を作成し，関数に分割してください。なお引数は「繰り返す回数 p」，戻り値は「加算した結果」としてください。

（4）WebAPI

一般にシステム開発では作成するプログラムは大規模なので，コードの長さも膨大となり，開発期間もその分大きくかかる。だが，多くのシステムでは「ボタンを押したらウィンドウが開く」や「ボタンを押したらファイルが保存される」処理について共通していることが多い。共通している処理について API(Application Program Interface) として関数に近い形で提供されている。この API を使って開発することで，開発期間の短縮につながる。インターネット上のサービスやビッグデータへの Web の通信を利用したアクセス機能を提供している API を WebAPI といい，現在のシステム開発において利用されている。（図表 10 参照）



図表 10　WebAPI の概念

■ WebAPI を利用したプログラムの例

WebAPI を利用し，郵便番号からそれに対応する住所を表示するプログラムを図表 11 に示す。
なお必要なサービスやデータと引数と戻り値は次の通りである。

必要なサービスやデータ：郵便番号に係わるデータ
引数：郵便番号
戻り値：郵便番号に対応する住所

また，この例で使用する WebAPI は「http://zipcloud.ibsnet.co.jp/」が提供するものである。この例では指定する郵便番号を URL に続く「api/search?zipcode=100-0013」で指定している。

```
1  // 使用する WebAPI の URL
2  let client =
3  HttpGetClient("http://zipcloud.ibsnet.co.jp/api/search?zipcode=100-0013")
4  // 通信を実行
5  let status = client.task()
6  // 通信済のデータからキーを指定して値を取得
7  vprintln("address1 = " + client.get(key: "address1"))
8  vprintln("address2 = " + client.get(key: "address2"))
```

図表 11　WebAPI を用いたプログラムの例

＜演習 4＞

（1）図表 11 のコードにおいて指定する郵便番号を自分の住んでいる住所の郵便番号にして住所が正しく表示されることを確認してください。
（2）WebAPI には他にどのようなものがあるか調べ，その WebAPI を使用するとどのようなプログラムを作成することが可能になるか考えましょう。

■学習活動と展開

【学習活動の目的】

- **配列や乱数を用いたプログラムを作成できる力を身に付ける。**
- **プログラムの機能を整理するために関数に分割し，構造化する力を身に付ける。**
- **関数の理解から API の必要性を理解する。**

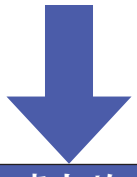
○学習活動とそれを促す問い

	問　い	学習活動
展開 1	配列を用いたプログラムを作ってみよう。	配列の必要性を理解させ，配列を用いたプログラムを作成する。
展開 2	乱数を用いたプログラムを作ってみよう。	乱数を用いて表現できることを理解させ，乱数を用いたプログラムを作成する。
展開 3	プログラムを関数で分割しよう。	関数で分割する必要性を理解させ，プログラムを関数で分割する。

展 開 1	
問　い	配列を用いたプログラムを作成しよう。
学習活動	・配列の必要性を考える ・配列を用いたプログラムを作成する。
指導上の留意点	・配列の形でデータを一括して扱う利点を解説する。 ・配列の添字の変化と参照する要素の動きを追えるようにする。 ・総和を求めるプログラム作成などで配列内のすべての要素を追うことができるようにする。

展開 2	
問　い	乱数を用いたプログラムを作成しよう。
学習活動	・乱数の必要性を考える。 ・乱数を用いたプログラムを作成する。
指導上の留意点	・乱数を用いるとどのようなプログラムが作れるか考えさせる。 ・実行するたびに乱数の値が変化し，処理内容も変化することを確認させる。 ・自分自身で指定する範囲内の乱数を生成できるようにする。

展開 3	
問　い	プログラムを関数で分割しよう。
学習活動	・プログラムを関数で分割する必要性を考える。 ・関数を用いてプログラムを分割する。 ・分割した関数を他のメインコードで利用する。
指導上の留意点	・システム開発ではコードが膨大になることから機能ごとに分割する必要性を説明する。 ・分割する際には「関数の機能」「引数」「戻り値」を意識させてからプログラミングを行う。 ・「関数の機能」「引数」「戻り値」が適切であれば，他のメインコードにおいても関数が使用可能であることをプログラムを作成することで理解させる。



まとめ	
まとめ	・数多くのシステムで共通して使用されている機能は API として提供されていることを説明する。 ・特に WebAPI を活用すると，外部のサービスやビッグデータを使用することが可能になり，様々なシステム開発の場面で活用されていることを説明する。

学習15 アルゴリズムの比較

■研修内容

【研修の目的】

- □ 代表的な探索アルゴリズムのうち，線形探索と二分探索のアルゴリズムを理解する。
- □ 代表的なソートアルゴリズムのうち，選択ソートとクイックソートのアルゴリズムを理解する。
- □ アルゴリズムやデータ数，探索する値が異なれば，処理時間が異なることを理解する。
- □ 効率的なアルゴリズムについて，アルゴリズムを比較する活動を通じて，生徒に考えさせる授業ができるようになる。

（1）探索アルゴリズム　線形探索と二分探索

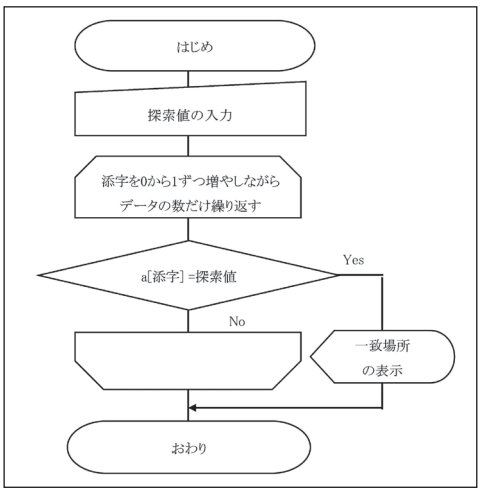
配列などの中から必要なデータを探し出すことを探索といい，線形探索や二分探索などがある。

■線形探索

線形探索は配列を先頭から順に比較しながら探索値に一致するデータを探し出す探索方法である。探索値と配列内のデータを先頭から順番に比較していき，一致したデータがあれば，その場所（配列の添字）を表示する。図表 1 のような 7 個のデータ a[0] ～ a[6] があり，探索値が「82」であるとする。右のフローチャートでは下記の①～③のように，図表 1 の左側から順に探索値の「82」と比較していくことになる。

配列の要素	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
データ	61	15	82	77	21	32	53

図表 1　探索対象の配列



図表 2　線形探索

① a[0] の「61」と探索値「82」を比較し，異なるので次に進む。
② a[1] の「15」と探索値「82」を比較し，異なるので次に進む。
③ a[2] の「82」と探索値「82」を比較し，一致するので a[2] にあることがわかる。

■プログラム例

コード例を以下の図表 3 に示す。a は探索対象の配列であり，p は探索値である。

```
1  func linesearch (_ data:[Int], _ value:Int)  {      // 線形探索関数
2      for i in 0..<data.count {
3          if (data[i] == value) {
4              vprintln(" 見つかりました ")
5              break
6          }
7      }
8  }
9  let a = [61,15,82,77,21,32,53]    // 配列に値を代入
10 let p = 82 // 探す値
11 linesearch(a, p)
```

図表 3　コード

■二分探索

配列の中から探索範囲を半分ずつ狭めながら目的のデータを探し出す探索方法である。
ここでは 7 個のデータ a[0] ～ a[6] があり昇順にソートされており，探索値が「43」であるとする。

配列の要素	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
データ	25	33	43	51	66	71	88

図表 4　探索対象の配列

①中央値＞探索値より，上限添字の入れ替え

まず，探索範囲の中央を求める。配列の下限は a[0]，上限は a[6] なので，中央は「(下限添字＋上限添字) ÷ 2 =(0+6) ÷ 2=3」のように求められる。従って，探索範囲の中央にある a[3] のデータ「51」と探索値「43」を比較する。

「43」は a[3] の「51」より小さいので，探索範囲は a[0] ～ a[2] に絞られる。

②中央値＜探索値より，下限添字の入れ替え

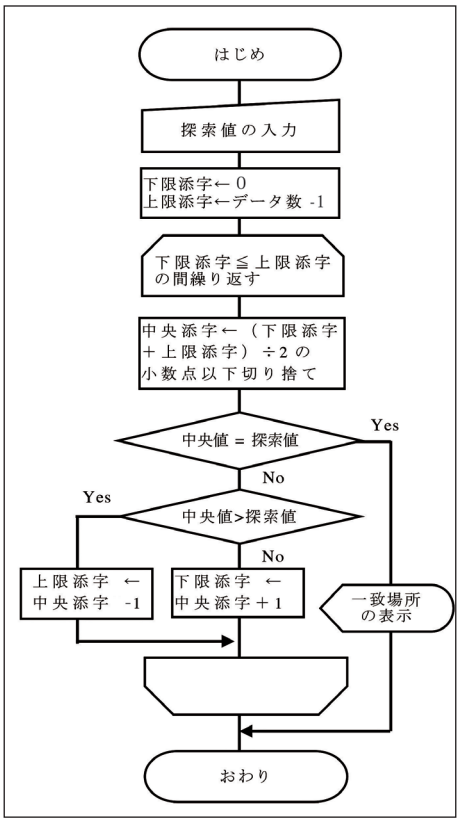
新たな探索範囲の下限は a[0]，上限は a[2] なので中央は「(0+2) ÷ 2=1」のように求められる。従って，a[1] のデータ「33」と探索値「43」を比較する。

「43」は a[1] の「33」より大きいので，探索範囲は a[2] に絞られるため，下限添字を中央添字の「1」を 1 つ増やした値の「2」に変更する。

③中央値＝探索値より，検索終了

新たな探索範囲の下限は a[2]，上限は a[2] なので中央は「(2+2) ÷ 2=2」である。a[2] のデータ「43」と「43」を比較する。「43」は a[2] の「43」と一致するので，「43」が a[2] にあることが分かる。

※人間であれば，下限が a[2] で上限が a[2] であれば，探索値が a[2] であることは当たり前に感じる。しかし，コンピュータは，定められたアルゴリズムによって動くため，下限と上限が一致するということは中央値もそれと一致するということにしかならない。アルゴリズムに従って，次のステップで中央値と探索値が一致して，探索が完了する。



図表 5　二分探索

■プログラム例

コード例を以下の図表 6 に示す。a は探索対象の配列であり，p は探索値である。

```
1  func binsearch (_ data:[Int], _ value:Int, _ min:Int, _ max:Int)  {
2      if (min > max) {
3      } else {
4          let mid = min + Int((max - min) / 2)
5          if (data[mid] > value) {
6              binsearch(data, value, min, mid-1)  // 半分より下の配列で２分探索
7          } else if (data[mid] == value) {
8              vprintln("見つかりました")
9          } else {
10             binsearch(data, value, mid+1, max)  // 半分より上の配列で２分探索
11         }
12     }
13 }
14 let a = [25,33,43,51,66,71,88]
15 let p = 43 // 探す値
16 binsearch(a, p, 0, a.count)                    //0 ～ a 配列の末尾の範囲で２分探索
```

図表 6　コード

＜演習 1＞

図表3および図表6のコードにおいて，「探索した回数を表示」できるようにプログラムを変更しましょう。

■線形探索と二分探索の比較

左側から探索を行う線形探索においては探索値が左端にあるか，右端にあるかによって探索回数が大きく変わり，1 回の探索で見つかる場合もあれば，全てのデータを探索する場合も出てくるので，一般的にデータ数が多い場合は線形探索では時間がかかることが多い。

二分探索の場合は 1 回の探索で見つからなかった場合でも探索範囲をデータ数の半分に絞ることができるので，線形探索と比較すると，データ数が大きくなっても探索にかかる時間はそれほど増加しない場合が多い。

最大探索回数だけを比較すると，回数の少ない二分探索がよいアルゴリズムと考えがちだが，二分探索には事前にデータを並べ替えておく必要があり，一概によいアルゴリズムとは言い切れない。例えば「事前にデータが並び替えられている保証がない場合」や「データの数がそれほど多くなく，シンプルなアルゴリズムの方が望ましい場合」などは線形探索の方が有用な場合もある。

＜演習 2＞

線形探索と二分探索の最大探索回数を求め，図表 7 に記入しましょう。

データ数	2	10	100	1000	10000
線形探索での最大探索回数	2				
二分探索での最大探索回数	2				

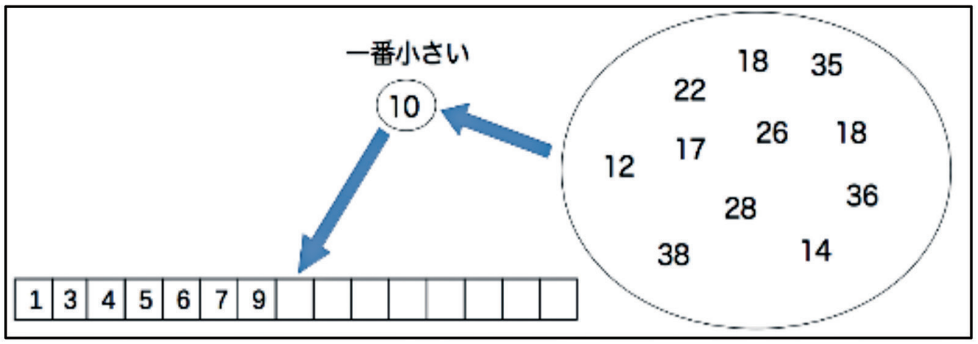
図表 7　線形探索と二分探索での最大探索回数の比較

（2）ソートアルゴリズム　選択ソートとクイックソート

配列などの中を昇順，降順に並べ替えることをソートといい，選択ソートやクイックソートなどがある。

■選択ソート

選択ソートは配列内のデータから最小値を探索し，最小値から順に取り出すことで並べ替えを実現するアルゴリズムである。（図表 8 参照）



図表 8　選択ソートの概念

■プログラム例

コード例を以下の図表 9 に示す。a はソート対象の配列である。なお a[i] と a[j] の値を入れ替える際には変数 temp を使って以下のように行っている。

1　a[i]の値を一時的に temp に入れておく(temp=a[i])
2　a[i]の値を a[j]の値に置き換える(a[i]=a[j])
3　a[j]の値を temp の値に置き換える(a[j]=temp)

```
1  var a = [7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
2  vprintln("ソート前")
3  for value in a { vprint(String(value) + " ")}
4  vprintln("")
5  for i in 0..<a.count-1 {
6      for j in i+1..<a.count {
7          if(a[i] > a[j]) {
8              let temp = a[i]
9              a[i] = a[j]
10             a[j] = temp
11         }
12     }
13 }
14 vprintln("ソート後")
15 for value in a { vprint(String(value) + " ") }
16 vprintln("")
```

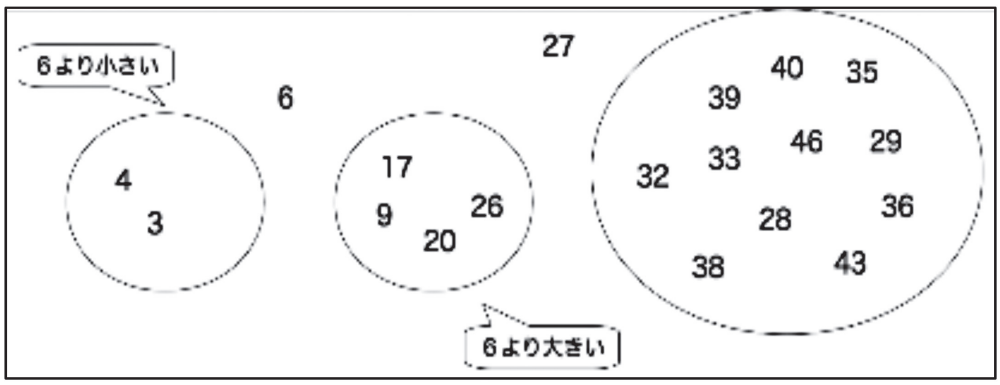
図表 9　コード

＜演習 3＞

図表 9 のコードにおいて，「データを入れ替えた回数を表示」できるように変数 c を追加してプログラムを変更しましょう。

■クイックソート

クイックソートは配列内の一つのデータを軸として，大小 2 つに分割した後，分割したデータに対して同じ処理を再度行うことにより並べ替えを実現するアルゴリズムである。（図表 10 参照）図表 10 では 27 と比較して分割した後，小さい方をさらに 6 と比較して分割している。



図表 10　クイックソートの概念

■プログラム例

コード例を以下の図表 11 に示す。a はソート対象の配列である。

```
1  func qsort(_ data:inout[Int], _ start:Int, _ end:Int) {
2      let pivot = data[ (start + end)/2]
3      var i = start
4      var j = end
5      while true {
6          while data[i] < pivot {
7              i += 1
8          }
9          while data[j] > pivot {
10             j -= 1
11         }
12         if ( i >= j) {
13             break
14         }
15         var temp = data[i]
16         data[i] = data[j]
17         data[j] = temp
18         i += 1
19         j -= 1
20     }
21     if( start < i - 1 )
22     {
23         qsort(&data, start, i - 1 )
24     }
25     if( end > j + 1 )
26     {
27         qsort(&data, j + 1, end)
28     }
29 }
30
31 var a = [7,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
32 vprintln(" ソート前 ")
33 for value in a {
34     vprint(String(value) + " ")
35 }
36 vprintln("")
37 vprintln(" クイックソート配列の値 ")
38 qsort( &a, 0, a.count-1)
39 vprintln(" ソート後 ")
40 for value in a {
41     vprint(String(value) + " ")
42 }
43 vprintln("")
```

図表 11　コード

＜演習 4＞

（1）図表 11 のコードにおいて,「軸の数値を基準にデータの大小の 2 分割が終了したときに配列 a の内容を表示」できるように「vprintln(" 分割完了 "+a+a[M])」を追加してプログラムを変更しましょう。

（2）（1）の結果を見て，どの値が軸になっているのか確認しましょう。

（3）選択ソートとクイックソートの比較

データ数がN個の場合，選択ソートにおいては「複数の要素から最小値を探す」処理をN-1回行うことになる。したがってデータ数が大きくなればなるほど「最小値を探す」処理の回数が膨大になり，処理に必要な時間が多くなりやすい。

それに対してクイックソートでは基準値を元に２分割しながら並べ替えの処理を行うので，二分探索と同じようにデータ数が大きくなっても処理の回数が膨大になりにくい。

しかしながら，クイックソートでは逆順に並べ替えられている場合など，ある条件下では比較を行う回数が膨大になり，選択ソートよりも処理に必要な時間が多くなることもある。

ソートアルゴリズムの場合においても，必ずしもクイックソートの方が優れたアルゴリズムであるとは言い切れないことがわかる。

■学習活動と展開

【学習活動の目的】

- **線形探索のアルゴリズムとそのプログラムを理解し，作成する方法を身に付ける。**
- **二分探索のアルゴリズムとそのプログラムを理解し，作成する方法を身に付ける。**
- **線形探索と二分探索における探索回数の比較を通して効率的なアルゴリズムについて考える力を身に付ける。**
- **同じ問題に対しても複数の解決方法があることを理解する。**

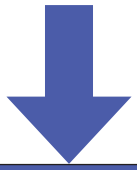
○学習活動とそれを促す問い

	問　い	学習活動
展開 1	線形探索のプログラムを作ってみよう。	線形探索のアルゴリズムについて考え，そのプログラムを作成する。
展開 2	二分探索のプログラムを作ってみよう。	二分探索のアルゴリズムについて考え，そのプログラムを作成する。
展開 3	最大探索回数を比較してみよう。	線形探索と二分探索の最大探索回数や平均探索回数を比較する。

展 開 1	
問　い	線形探索のプログラムを作ってみよう。
学習活動	・線形探索のアルゴリズムについて考える。 ・線形探索のアルゴリズムをもとにしてそのプログラムを作成する。
指導上の留意点	・プログラムを作成する前に，実際の数値例を使って，値がどのように比較，移動しているかを目視で確認しておく。 ・この学習項目は配列についての理解が前提となるため，前もって配列の指導を行っておく。

展開 2	
問　い	二分探索のプログラムを作ってみよう。
学習活動	・二分探索のアルゴリズムについて考える。 ・二分探索のアルゴリズムをもとにしてそのプログラムを作成する。
指導上の留意点	・プログラムを作成する前に，実際の数値例を使って，値がどのように比較，移動しているかを目視で確認する。その際，中央値の添字，上限の添字，下限の添字をその都度確認させながら行うとよい。 ・二分探索は，事前に昇順または降順にデータを整列させる必要があることについて注意する。

展開 3	
問　い	最大探索回数を比較してみよう。
学習活動	・具体的な配列と探索値を示して，線形探索と二分探索における最大探索回数や平均探索回数を比較する。
指導上の留意点	・生徒の状況に応じて，データ数が n の場合の最大探索回数や平均探索回数についての一般的な関係式を示し，具体的なデータ数を与えてそれらの値を計算させるとよい。



まとめ	
まとめ	・問題の解決には様々な方法があり，それらを比較検討し，最善の方法を見つけることの重要性について理解させる。

学習16 確定モデルと確率モデル

■研修内容

【研修の目的】

- □ モデルの構造を決定し，モデルを数式などで表現し，コンピュータを活用してモデルを動かしてシミュレーションを行うというモデル化とシミュレーションの方法を理解する。
- □ 数式モデルを作成する活動を通じて，生徒に数式モデルを考えさせる授業ができるようにする。
- □ プログラミング言語を活用して，モデルを動かしてシミュレーションする方法について理解する。
- □ モデルの妥当性について，検討する活動を通して，生徒に考えさせる授業ができるようにする。

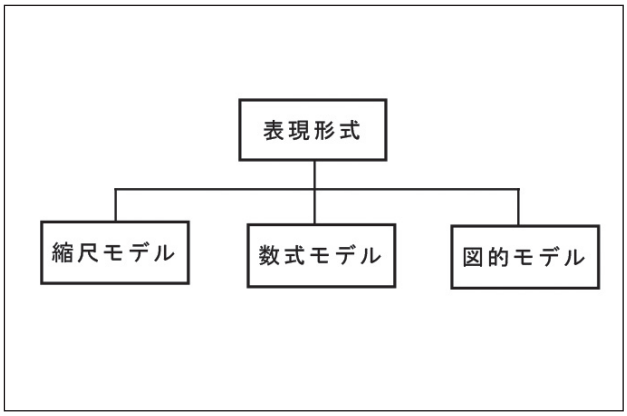
（1）モデル化とシミュレーション

モデルとは，事物や現象の本質的な形状や法則性を抽象化して，より単純化したものを指す言葉であり，事物や現象のモデルを作ることをモデル化と呼ぶ。

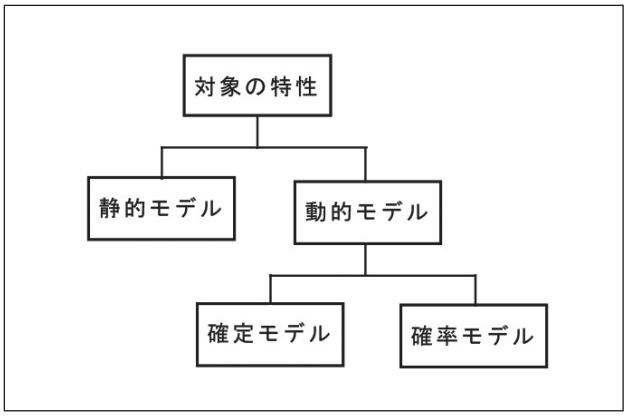
妥当性の高いモデル化ができれば，これを使って実際には行う事が困難な実験を計算だけで行なったり，複雑な現象を再現したりするための手段としても活用できる。

モデルを使って実際にどのような現象が起こるのかを予測する事を一般にシミュレーションと呼んでいる。シミュレーションを行う際に活用されるのがプログラミング言語や表計算ソフトウェア，シミュレーションソフトウェアなどである。

モデルは，表現形式や対象の特徴によって分類することができる。静的モデルは，プラスチックモデルや建築図面などのように時間的要素を含まないもので，動的モデルはレジの待ち行列や，気象予測，生物の成長など，時間的要素を含んだ事象のモデルである。また，確定モデルは不規則な現象を含まず，方程式などで表せるモデルであり，確率モデルはサイコロやクジ引きのような不規則な現象を含んだモデルである。



図表１　モデルの表現形式による分類の例



図表２　対象の特性による分類の例

（2）確定モデルのシミュレーション

ここでは，確定モデルの例として，複利法による預金金額の時間的変化を考える。

複利法は元金に利息を加算し，それを次の期間の元金として利息を計算していく方法である。

法則が一定のものは簡単に数式に表すことができる。数式に表す事もモデル化である。

●複利法の数式モデル

次期の金額＝現在の金額＋利息

利息＝現在の金額×利率×時間間隔

この数式モデルをプログラムに置き換えて 10 年間の預金額の変化を計算すると，次のようになる。

```
var yokin = 100000      // 預金額の初期値
let riritsu = 0.05      // 利率
var risoku = 0          // 最初の利息は 0
for i in 0..<10 {
    risoku  = Int(Double(yokin)*riritsu)
    yokin = yokin + risoku
    vprintln(String(i + 1) + "年目:" + String(yokin))
}
```

図表 3　預金の複利計算のプログラム

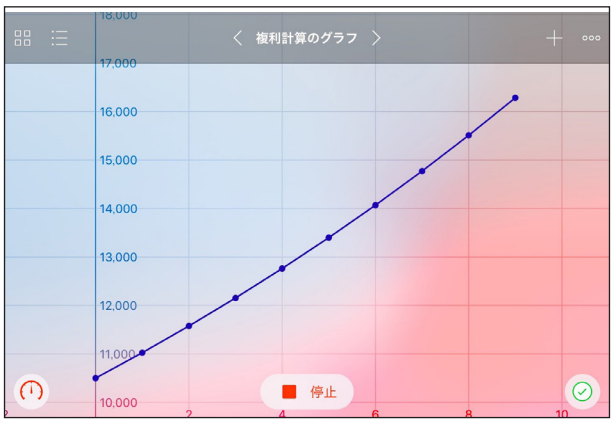
```
1 年目: 105000
2 年目: 110250
3 年目: 115762
4 年目: 121550
5 年目: 127627
6 年目: 134008
7 年目: 140708
8 年目: 147743
9 年目: 155130
10 年目: 162886
```

図表 4　プログラムの実行結果

この実行結果から，５％の利率の場合には９年ほどで１．５倍を超えることがわかる。
さらにこの数式モデルの計算結果をわかりやすく評価するため，グラフで表示する。
グラフ化する際には，グラフの表題と各軸の単位などのラベル表示をしておく。

```
let riritsu = 0.05                  // 利率
let data = XYData()                 // グラフ表示用の配列
var yokin = [10000]                 // 預金額格納配列の初期値
var risoku = 0                      // 最初の利息は 0
for i in 0..<10 {
    risoku  = Int(Double(yokin[i])*riritsu)
    yokin.append(yokin[i]+risoku)
    data.append(y: Double(yokin[i]+risoku))
}
let lineplot = LinePlot(xyData: data)   // グラフを描画
lineplot.label = "複利計算"
```

図表 5　グラフ化のプログラム



図表 6　グラフ

＜演習 1＞

複利計算のプログラムを動かし，グラフを表示してみましょう。また，このシミュレーション結果から，利息計算がどのような特徴をもっているか，どのようにして判断すれば良いか，考えてみましょう。
※計算する値（パラメータ）を変更しながら考えてみてください。

（3）確率モデルのシミュレーション

ここでは，サイコロの目の出方について，その傾向をとらえるためのモデル化とシミュレーションについて考えてみる。

6 面体のサイコロは各面積が等しく，振り方に偏りが無ければ，目の出方は各面が 6 分の 1 の確率で出現するはずである。まず実際のサイコロを使って各目の出現確率を算出してみる。

＜演習 2＞

4 人でグループを作り，各自が 50 回ずつサイコロを振って，出た目の回数の記録をとってください。次に全てのグループの出た目の回数を集計し，全員の繰り返し試行の結果としてください。結果から，サイコロの各面が一様に同じ確率で出現する公平な道具なのかを判断してみてください。

コンピュータ言語には，このような一様乱数を発生させるための関数が標準的な機能として提供されている。一般的には指定した区間の中から一様にランダムな値（一様乱数）を得られる関数となっている。次のプログラムは一様乱数を発生させて表示する。

```
1  let ransuu = Double.random(in: 0..<1)   // 乱数の発生
2  vprintln("乱数 :" + String(ransuu))
```

図表 7　乱数発生のプログラム

プログラムを実行すると，0 ～ 1 の範囲で実行する度に異なった結果が得られることが確認できる。このような乱数を発生させる関数を使ってサイコロのモデルをプログラミングしてみる。

整数で発生させる乱数は Int.random() 関数で提供されている。

```
1  let number = 100                                  // 回数
2  var deme:[Int] = [0, 0, 0, 0, 0, 0]               // 各出目のカウント用配列
3  for i in 0..<number {
4      let d = Int.random(in: 0..<6) + 1
5      deme[d-1] += 1
6      vprint(String(d) + " ")
7  }
8  vprintln("")
9  vprint("出現数 :")
10 for value in deme {
11     vprint(String(value) + " ")
12 }
13 vprintln("")
```

図表 8　サイコロのプログラム

4 行目では，1 以上，6+1 未満の乱数を発生し，小数点以下を切り捨てて 1 から 6 の整数を得ている。実行結果は次のようになる（数えた数値は毎回異なる）。

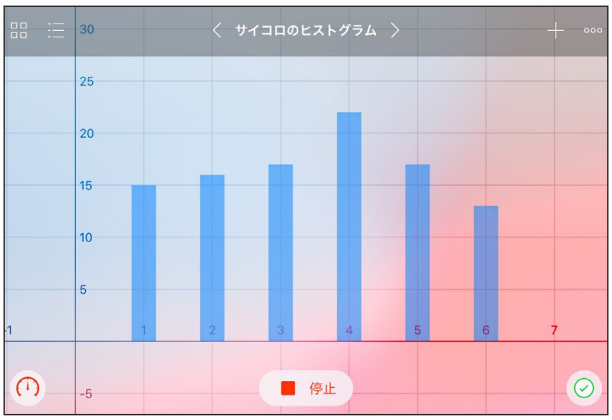
```
[5 4 3 6 1 4 3 1 4 3 4 6 6 3 4 1 2 1 5 6 4 2 2 2 4 1 2 4 1 3 5 4 3 1 6 5 5
 1 6 2 3 3 2 3 5 2 5 5 3 5 4 4 1 2 2 3 3 2 1 2 1 1 3 2 2 5 3 4 5 6 3 5 6 5
 2 4 6 6 1 5 4 5 4 6 3 5 2 4 5 1 2 4 5 3 3 5 4 6 1 2]
出現数: [15, 18, 18, 18, 19, 12]
```

図表 9　実行結果

1から6の目の出現数を度数分布（棒グラフ）に表現するためのプログラムは次のようになる。

```
let number = 100                                    // 回数
let data = XYData()                                 // グラフ描画用の配列
var counter:[Double] = [0, 0, 0, 0, 0, 0]           // 各出目のカウント用配列
for i in 1 ... number {
    counter[Int.random(in: 0..<6)]+=1
}
let bp = BarPlot(data: counter)                     // グラフを描画する
```

図表 10　度数分布（棒グラフ）表現のプログラム



図表 11　実行結果

＜演習 3＞

プログラムを実行してみて，試行回数が結果に与える影響について考察してみてください。コンピュータの乱数の発生がどの程度の試行回数で妥当と判断できるでしょうか。

確率モデルを使ったシミュレーション手法に，モンテカルロ法がある。モンテカルロ法の特徴は対象のモデルに乱数を大量に生成して入力し，近似解を得ようとする手法である。

この方法を使って，円周率を求めてみる。

平面の第 1 象限部分の X 方向に 1，Y 方向に 1 の長さをもつ正方形内に，ランダムに点を打つことを考える。ランダムに打った点のうち，半径 1，中心座標（0，0）の単位円の方程式「r*r = x*x + y*y」の第 1 象限部分の範囲に含まれた点の数と，全ての点の数との割合は，点の数が多ければ多いほど，円の第 1 象限部分の面積（π／4）に近づいてくるはずである。（確率的に近づいてくる。）

このような事に気が付けば，円周率を求めるプログラムを次のように書くことができる。

```
let totalcount = 1000                       // 計算回数
var incount = 0                             // 円の内側に入った数
var  x  = 0.0                               // X 座標保存用
var y = 0.0                                 // Y 座標保存用
var indata = XYData()                       // 円に入ったデータのグラフ描画用配列
var outdata = XYData()                      // 円から外れたデータのグラフ描画用配列
for i in 1 ... totalcount {
    x = Double.random(in: 0..<1.0)
    y = Double.random(in: 0..<1.0)
    if x*x+y*y<1.0 {
        incount+=1
    }
}
vprintln("円周率:"+String (Double(incount )/Double(totalcount)*4.0))
```

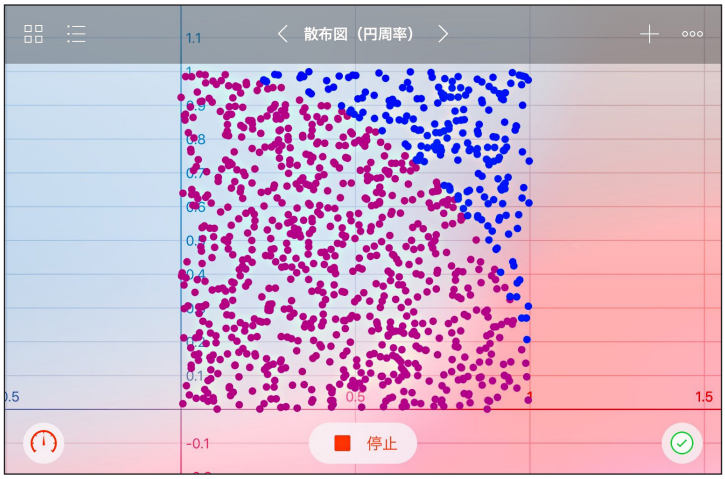
図表 12　円周率を求めるプログラム

このプログラムを何度か実行してみると，毎回結果が異なることがわかる。また，点の総数を増やすほど，結果が円周率に近づくことも分かる。

点の様子がわかるよう，円の中に入った点と円の外だった点の色を変えて散布図を作成するプログラムを以下に示す。

```
let totalcount = 1000                                    // 計算回数
var incount = 0                                          // 円の内側に入った数
var  x  = 0.0                                            // X 座標保存用
var y = 0.0                                              // Y 座標保存用
var indata = XYData()                                    // 円に入ったデータのグラフ描画用配列
var outdata = XYData()                                   // 円から外れたデータのグラフ描画用配列
for i in 1 ... totalcount {
    x = Double.random(in: 0..<1.0)
    y = Double.random(in: 0..<1.0)
    if x*x+y*y<1.0 {
        indata.append(x: x, y: y)
        incount+=1
    } else {
        outdata.append(x: x, y: y)
    }
}
let scatterIn = ScatterPlot(xyData: indata)              // 入ったデータでグラフ描画
scatterIn.color = Color.pink
let scatterOut = ScatterPlot(xyData: outdata)            // 外れたデータでグラフ描画
scatterOut.color = Color.blue
scatterIn.label = "円周率:"+String (Double(incount )/Double(totalcount)*4.0)
```

図表 13　散布図を作成するプログラム



図表 14　実行結果

＜演習 4＞

プログラムを実行してみて，試行回数が結果に与える影響について考察してみてください。確率的なモデルから得られた結果を予測するシミュレーションは実用的にも使われています。

インターネット等でモンテカルロ法がどのような事に使われているのか調べてみましょう。

■学習活動と展開

【学習活動の目的】

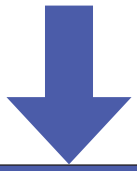
- モデル化とシミュレーションについて，数式モデルをプログラム化する方法を考え身に付ける。
- プログラムを使わなくてもシミュレーションは可能であるが，その方法では全く実用的にならない事を理解する。
- シミュレーション結果を考察し，モデルの特性についての理解が結果から可能であることを理解し身に付けている。

○学習活動とそれを促す問い

	問　い	学習活動
展開 1	一様乱数とは何か，何に使えそうか考えてみよう。	グループに分かれ，サイコロを使って出現する目を数え，よりよい結果を得るために全ての試行結果を合算する。
展開 2	一様乱数を使ったプログラミングで，シミュレーションし，結果を考察しよう。	iPad を使ってサイコロのシミュレーションを行う。得られた結果を考察し，試行回数やプログラムの構造について理解を深める。

展開 1	
問　い	一様乱数とは何か，何に使えそうか考えてみよう。
学習活動	・グループを編成し，サイコロを何に使えそうか考える。 ・アイデアがまとまったら，グループごとに発表を行う。 ・サイコロの形状が均一なものであれば，等しくランダムな目が出現する。確率的に一様であることを確認するため，繰り返し試行を行う。 ・各グループから得られた結果を合算する。
指導上の留意点	・単なる作業にならないように，どうすればより良い結果が得られるのかを考えさせながら進め，回数などは特に定めずに時間で区切って繰り返し試行を行う。 ・サイコロ以外の方法で乱数を発生させる方法についても考えさせて進めるとよい（ビュフォンの針やコイン投げなど） ・生徒全員で得られた結果はグループ単位ではなく必ず全員が持つよう促し，次のプログラミングの理解に繋がる点を留意させながら進める。

展開 2	
問　い	一様乱数を使ったプログラミングで，シミュレーションし，結果を考察しよう。
学習活動	・コンピュータを使ってプログラミングを行う。 ・エラーが発生した場合の訂正や，分からない部分を調べる活動を行って完成させる。 ・シミュレーション結果から，乱数の発生が確率的で，確定的でない事を理解し，プログラムによって大量の試行が可能である点を理解する。 ・プログラムに用いるパラメータの変更によって結果が異なることから，十分な試行のための計算コストについても理解する。
指導上の留意点	・エラーが発生した場合に，答えを調べるのではなく，エラーの意味を理解させるように場合によってはエラーをクラス全体で共有しながらプログラミングを進める。 ・どうしてもプログラミングが完成しそうもない生徒には，丁寧にプログラムコードの意味を教えていく。最終的に完成できなくても結果を知ることができれば良い。 ・手作業による繰り返し試行との違いについて考えさせながら結果を検討させる。



まとめ	
まとめ	・プログラミングによって，膨大な量の繰り返し試行が可能であり，シミュレーション結果を評価して改善する事の重要性について認識させる。 ・乱数による計算が確率的に一様になる事への理解を深めさせ，今後の授業への興味・関心を高めさせる。

学習17 自然現象のモデル化とシミュレーション

■研修内容

【研修の目的】

- □ 自然現象のモデルの構造を決定し，コンピュータを活用してモデルを動かしてシミュレーションを行う方法を理解する。
- □ 数式モデルを作成する活動を通じて，生徒に数式モデルを考えさせる授業ができるようになる。
- □ プログラミング言語などを活用してモデルを動かしてシミュレーションする方法について理解する。
- □ モデルの妥当性について，検討する活動を通して，生徒に考えさせる授業ができるようになる。

（1）物体の放物運動のモデル化

物体の放物運動をモデル化し，初速度と投げ上げる角度によって，物体がどのような軌跡を描いて飛んでいくのか，遠くに飛ばすためにはどのような角度で投げるとよいのかという問いの答えを導く。ここでは，単純化のために，空気抵抗を省いてモデル化する。

物体の移動する軌跡は物体の水平（X）方向と鉛直（Y）方向の移動に分けて考え表現することとし，垂直方向には重力加速度を受ける。

ここでは，三角関数は速度を縦と横に分けて考えるために用い，運動を表す式では，水平方向は一定の速度，鉛直方向は，一定の割合で下向きに速度が増えていくと考えてモデル化している。自然現象を表すためには，数学や物理の知識が必要なことも生徒に伝えるようにする。

●水平方向

水平方向は摩擦を受けない条件から等速運動であり，任意の時点での加速度は投げ上げた時の初速度と等しい。

微小時間後の位置　＝　現在の位置　＋　現在の速度　×　時間間隔
現在の速度　＝　初速度
また，物体を投げ上げる角度と初速度から，水平方向の初速度は，
水平方向の初速度　＝　初速度　×　ｃｏｓ（投げ上げる角度）
である。

●鉛直方向

鉛直方向は重力加速度によって速度が変化していく。任意の時点の速度に対し，微小時間後の速度は重力加速度の影響を受けて減少しているため，この間の速度の平均（平均速度）を用いて算出することとする。

微小時間後の位置　＝　現在の位置　＋　平均速度　×　時間間隔
平均速度　＝　（現在の速度　＋　微小時間後の速度）／２
微小時間後の速度　＝　現在の速度　－　重力加速度　×　時間間隔
また，
鉛直方向の初速度　＝　初速度　×　ｓｉｎ（投げ上げる角度）
である。

（2）物体の放物運動のプログラムによるシミュレーション

作成した数式モデルをプログラムで記述して実行してみる。さらに，条件を変えながらシミュレーションを繰り返し，問題解決に役立てる事を理解する。

プログラム中に用いる変数と諸条件の対応をわかりやすくするためにも，変数表を作成しておくとよい。

変数名	意　味	変数名	意　味
dt	時間間隔	v0	初速度
x	水平位置	vx	水平速度
y	鉛直位置	vy	鉛直速度
g	重力加速度	angle	投げ上げ角度

図表 1　変数表

さらにこの数式モデルのシミュレーション結果をわかりやすく評価するため，グラフで表示する。

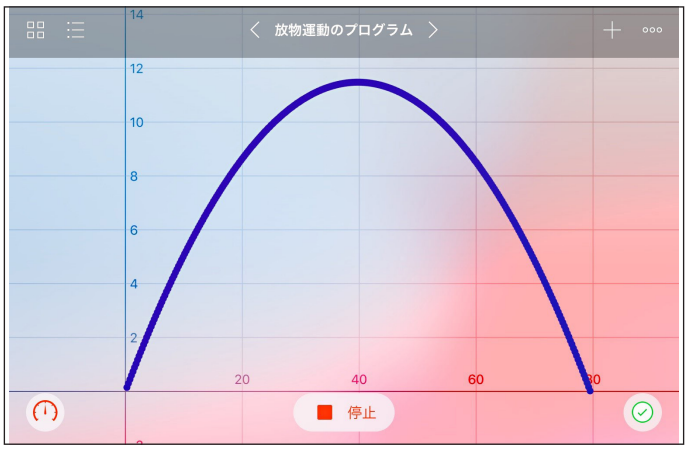
●弧度法（ラジアン）

コンピュータにおける算術演算において，三角関数の引数となる角度は弧度法（ラジアン）である。ラジアンとは，円（扇型）の半径の長さと等しい弧の長さとなる中心角を 1 とする単位で，0 度＝ 0 ラジアン，90 度＝ π ／ 2 ラジアン，180 度は π ラジアン，360 度＝ 2 π ラジアンとなる。

角度からラジアンへの変換は角度に π ／ 180 を掛ければよい。

```
1  import UIKit                                  //π の値を使うためにインポート
2  let dt=0.01                                   // 微小時間
3  let v0=30.0                                   // 初速度
4  let g=9.8                                     // 重力加速度
5  let data = XYData()                           // グラフ表示用配列
6  var x = 0.0                                   // 原点
7  var y = 0.0                                   // 原点
8  var vx = [0.0]                                // 速度の配列
9  var vy = [0.0]                                // 速度の配列
10 let angle = 30.0*Double.pi/180.0              // 角度をラジアンで設定
11 vx=[v0*cos(angle)]                            //X 方向の初速度
12 vy=[v0*sin(angle)]                            //Y 方向の初速度
13 for i in 0..<1000 {
14     vx.append(vx[i])
15     vy.append(vy[i]-g*dt)
16     x = x+vx[i]*dt
17     y = y+(vy[i]+vy[i+1])/2.0*dt
18     if y < 0.0 {break}                        //Y 座標が 0 を下回ったらループを抜ける
19     data.append(x: x , y: y)
20 }
21 let lineplot = LinePlot(xyData: data)        // グラフを描画
22 lineplot.label = " 放物運動 "
```

図表 2　放物運動のプログラム



図表 3　実行結果

＜演習 1＞

放物運動のプログラムを実行してみて，結果を確認してください。正しいと判断できる結果が出たら，投げ上げる角度を変更して再び実行し，どの角度が一番遠くに飛ぶのか，どの角度が一番高く飛ぶのかシミュレーションによって調べてください。

（3）生命体の増加シミュレーション

自然界における生物の個体数は，ある一定の環境内で一気に繁殖することがよく知られている。

おおむね，その環境は限られた範囲内であり，その範囲内に生存しうる個体数の上限をここでは環境収容力と呼ぶ。単位時間あたりに個体数が増加する数は，個体数×増加率で求めることができ，個体数が多くなればなるほど増加数が増える。

また，単位時間当たりに個体が減少する数は，個体数×減少率で示すことができ，増加数同様に個体数の数に比例して減少する。

一方，環境収容力と個体数の増加は個体の減少に強く働くこととなるため，ここでは，

減少率　＝　（現在の個体数／環境収容力）×増加率

と定義する。この定義によって，個体数は環境収容力以上の数に増えると増加より減少数の方が増え，一方環境収容力より個体数が少なければ増加数が増える。

以上の諸量について次のように設定し，プログラミングしてみる。

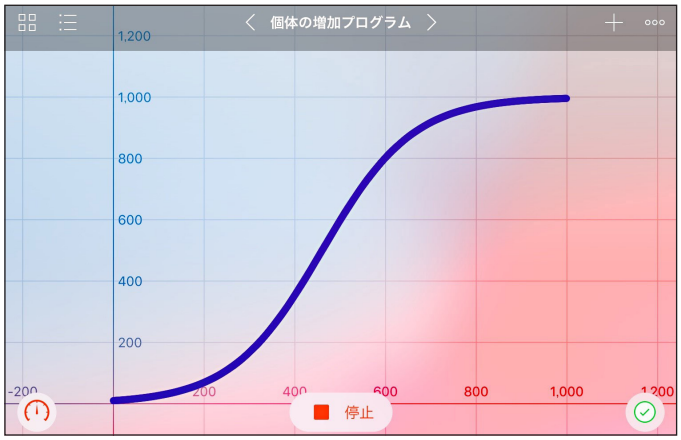
変数名	意　味	変数名	意　味
n	個体数	zoukasuu	増加数
zouka	増加率	gensyousuu	減少数
capacity	環境収容力		

図表 4　諸量の設定

```
1  let zouka = 0.01                                    // 増加率
2  let capacity = 1000.0                               // 環境収容力
3  var n = [10.0]                                      // 個体数の配列の初期値
4  var zoukasuu = 0.0                                  // 増加数保存用
5  var gensyousuu = 0.0                                // 減少数保存用
6  let data = XYData()                                 // グラフ描画用配列
7  for i in 0..<1000 {
8      zoukasuu = n[i]*zouka
9      gensyousuu = n[i]*n[i]/capacity*zouka
10     n.append(n[i]+zoukasuu-gensyousuu)
11     data.append(y: n[i+1])
12 }
13 let lineplot = LinePlot(xyData: data)              // グラフ描画
14 lineplot.label = "固体数の増加"
```

図表 5　個体の増加プログラム

このプログラムを実行すると，次のような結果が得られる。



図表 6　実行結果

プログラムでは単位時間あたりの増加率を 1 %としているが，増加率を引き上げれば生物の繁殖速度を実感できる。

＜演習 2＞

プログラムを実行し，結果を確認したら，増加率を変更してグラフがどのように変化するか確認してください。

個体数が制約条件の中で成長するこのような曲線をロジスティック曲線と言い，環境収容力に収束する S 字曲線として有名である。

（4）ランダムウォークのシミュレーション

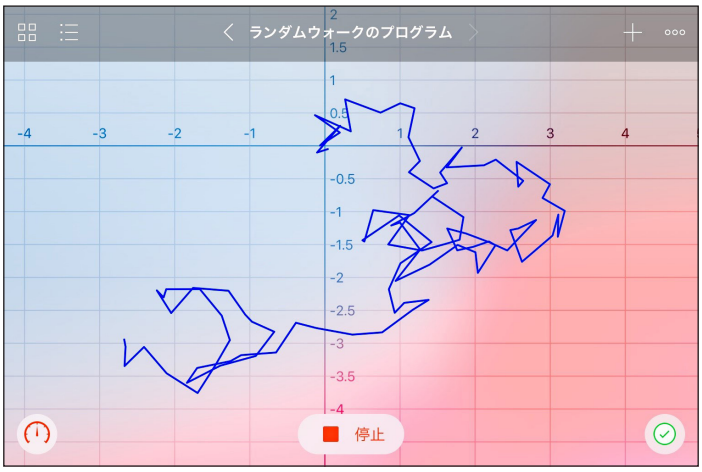
ブラウン運動や株価の変動など，不確実な現象のシミュレーションとして用いられるモデルに，ランダムウォークがある。

ここでは，乱数を使ったシミュレーションとしてランダムウォークを取り上げ，複雑な物の動きについて，視覚的に確認できるようなシミュレーションを行ってみる。

```
1  let totalcount = 100                          // 計算回数
2  var  x  = 0.0                                 // X 座標の初期値
3  var y = 0.0                                   // Y 座標の初期値
4  var data = XYData()                           // グラフ描画用配列
5  for i in 1 ... totalcount {
6      x = x + Double.random(in: 0..<1.0)-0.5
7      y = y + Double.random(in: 0..<1.0)-0.5
8      data.append(x: x, y: y)
9  }
10 let lineplot = LinePlot(xyData: data)         // グラフ描画
11 lineplot.symbol = nil
12 lineplot.color = Color.blue
```

図表 7　プログラム

このプログラムを実行すると，実行する度に異なる次のようなグラフ表示がされる。



図表 8　実行結果

＜演習 3＞

ランダムウォークの乱数式 random() から 0.5 を減算するのは何故か考えてみてください。また，ランダムウォークになんらかの特性を持たせるとしたら，どこを変更すれば良いかも考えてください。

ランダムウォークは，でたらめに行動する生物や，人間の行動などを分析するため，様々な分野のシミュレーションの基礎として利用されており，近年では機械学習によく用いられている。

■学習活動と展開

【学習活動の目的】

- 物体の放物運動について，数式モデルを作成するなどの活動を通して，モデル化の方法を身に付ける。
- 物体の放物運動について，作成した数式モデルからプログラムの作成を通して，シミュレーションを行う方法を身に付ける。
- 作成したプログラムを使って，物体を最も遠くまで飛ばす条件を見付ける活動を通して，モデル化とシミュレーションによる問題解決の方法を身に付ける。

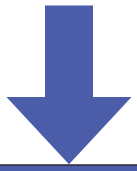
○学習活動とそれを促す問い

	問　い	学習活動
展開 1	物体の放物運動について，仮説を立てて数式モデルを作成してみよう。	放物運動における物体の速度や位置を求める数式モデルを作成する。
展開 2	プログラムを作ってシミュレーションを行ってみよう。	プログラミング言語や，表計算ソフトを使ってプログラムを作成し，シミュレーションを行う。
展開 3	物体を遠くまで飛ばす方法について考えてみよう。	どのような角度で投げれば最も遠くまで飛ぶかについて，プログラムの値を変えながら考える。

展開 1	
問　い	物体の放物運動について，仮説を立てて数式モデルを作成してみよう。
学習活動	・微小時間内は等速運動であるという仮説を立て，放物運動における速度や位置を求める数式モデルを作成する。
指導上の留意点	・速度が変化する運動であっても，微小時間内では一定の速度で運動しているというヒントを与え，生徒自らが数式を立てることができるようにする。

展開 2	
問　い	プログラムなどを作ってシミュレーションを行ってみよう。
学習活動	・数式モデルをもとにして，シミュレーションのための表計算ソフトウェアのワークシートやプログラムを作成し，シミュレーションを行う。
指導上の留意点	・生徒の状況に応じてシミュレーションを行うための方法（表計算ソフトウェアで行うか，プログラムを作成するか）を選択する。 ・プログラムの作成を行う場合は，生徒の状況に応じて適切なプログラミング言語の選択を行う。

展開 3	
問　い	物体を遠くまで飛ばす方法について数式に入れる値を変えて見つけよう。
学習活動	・作成したプログラムや表計算ソフトウェアのワークシートを用いて，速さを一定にしたまま，角度の値を変えながら x-y グラフから最大到達距離を調べる。
指導上の留意点	・45 度の方向に投げると最も遠くに到達することについて，グラフから読み取って気付かせるようにする。



まとめ	
まとめ	・問題の分析，モデルの作成，シミュレーションのためのプログラムや表計算ソフトウェアのワークシートの作成，モデルの妥当性の検討，最大到達距離の条件を見付けるなどの一連の活動を通して問題解決の方法を理解する。

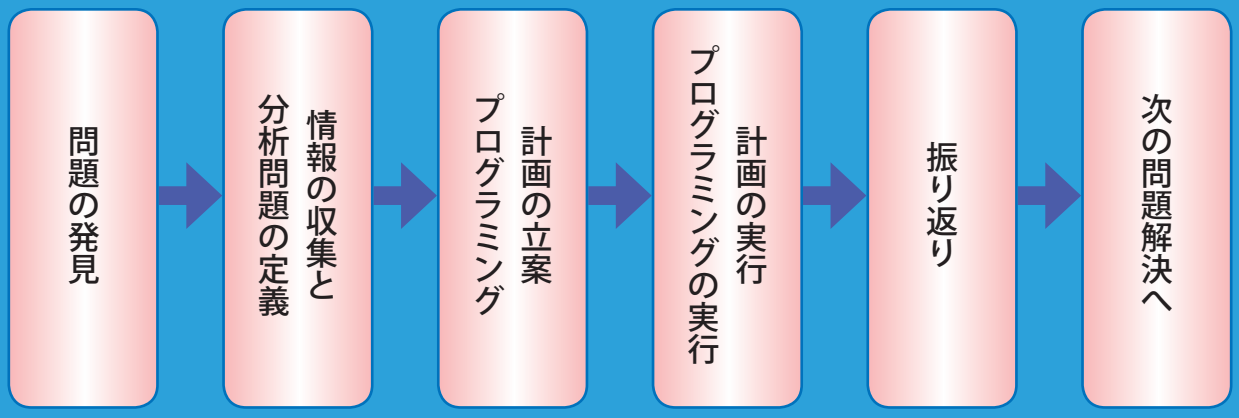
◆全体を通じた学習活動の進め方

【全体を通じた学習活動の目的】

・自然現象や社会現象の問題点を発見し，コンピュータやプログラミングを活用した解決策を提案する。

【全体を通じた学習活動の流れ】

自然現象や社会現象の問題の発見から分析，コンピュータやプログラミングを活用した解決方法の提案，評価，改善など，一連の学習活動を行う。



コンピュータやプログラミングを活用した授業を計画するにあたり，留意しておくべきこととして

① 効果的なプログラミングの活用
② プログラミングの習熟度に合わせた対応
③ モデルベースで自然現象や社会現象の問題点を発見させる
④ 体験的な活動を効果的に活用する
⑤ シミュレーションにおける具体的なデータの活用

などが挙げられる。この単元では中学校までの知識を前提としているが，上記①から⑤は密接に関連し影響しあっているため，学校の状況や生徒の実態等に応じて設計すると良いだろう。

①効果的なプログラミングの活用	・プログラミングを行う時は，何をねらいとしてプログラミングを行うか明確にする。 ・プログラミングによる解決手段は１つでなく，様々な解決手段があることを伝える。 ・プログラミングで問題を解決できたとしても，より最適な処理手順がないか，問題を細分化し，処理を最適化した手順を模索させる。
②プログラミングの習熟度に合わせた対応	・プログラミングに慣れていない生徒については，細かく管理することにより活動はしやすくなると考えられるが，自ら解決していく主体性も下がるため，生徒同士で進行状況を相互チェックさせフォローアップさせる活動を取り入れる。 ・プログラミングに精通した生徒に対しては，プログラミングに慣れていない生徒へのフォローやデバック作業，あるいは先生役としてライブプログラミングしながら説明させるなどの教える活動に取り組ませ，理解を深めさせる。また，完成したプログラムをより発展させる方法がないか，Web で調べ学習させる活動を取り入れる。

③モデルベースで現象の問題点を考える	・自然現象や社会現象の問題点を，生徒自らモデルベースで発見できるようにする。 ○現象の説明について，モデルを構築して説明することが効果的であることを伝える。 ○最初からモデルを作るのは難しいため，モデルを提示し，モデルを比較することなどを通して，仮説となるモデルを構築させる。 ○目に見えない現象の仕組みについて理解させるのは難しいため，説明モデルの活用と吟味を十分に行う。
④体験的な活動を効果的に活用する	・理解しにくい抽象的な概念を，身体を介して得られた感覚情報を効果的に取り入れることで理解させる。 ○センサとアクチュエータの制御をプログラミングし，実行結果を身体的に体験できる活動を行う。 ○加算器などの科学的な仕組みについて，身体を介して理解させる活動を状況に応じて取り入れる。
⑤シミュレーションにおける具体的なデータの活用	・生徒にとって身近で具体的なシミュレーションの題材を取り扱う。 ○シミュレーションで扱うデータは，可能であれば生徒たちが自らの手で集められるデータを取り扱う。 ○センサの位置情報のデータなどを活用し，集めたデータをもとにシミュレーションを行う活動を取り入れる。

【全体を通じた学習活動を行ううえでの注意点】

（3）コンピュータとプログラミングでは使用するプログラミング言語の選択をしなければならない。その際に，①選択したプログラミング言語の開発環境を整えることが可能か，教育現場の環境に応じたプログラミング言語を選択し，動作確認を行う。②選択したプログラミング言語の開発環境を整えるのが難しい場合は，インストール不要なプログラミング言語を選択，あるいはオフライン環境で動くエディタなどを活用する。③教える側が得意なプログラミング言語を選択するのではなく，授業のねらいを生徒が達成するのに最適なプログラミング言語を選択する，の以上の３点を留意する。

また，プログラミングによる本格的な解決活動までには技術的にも時間的にも難しいケースが想定される。そのような場合は

・問題点を明確化し，何をねらいとしてプログラミングを行うかしっかりと確認する
・シミュレーションで，データを用いた客観的な分析や考え方等を獲得する必要性を認識させる
・解決方法の探索，結果の予測段階で，プログラミングやシミュレーション等の考え方を用いることが出来ないかを考えさせる
・相手に的確に分かりやすく伝える情報デザインの考え方を意識してプログラミングさせる

など，（1）情報社会の問題解決（2）コミュニケーションと情報デザイン（4）情報通信ネットワークとデータの活用との連携を意識しながら，解決策の提案に留める方法も考えられる。