

第3章 プログラミングの基 本

3-1. プログラムの処理順

書いたプログラムの処理順について、基本的なルールを紹介します。

それでは今回から
プログラムの書き方を
教えていくぞ



そういえば
そういう話
だったわね

手順ばかり
だったから
忘れていたわ



僕もです



えー すまんな
Javaはプログラムを
書くだけでは
動かないからな



というわけで
前回までの話で
プログラムを
書く場所は
分かったよな？



srcフォルダ内に
クラスという
物を作って
書くんですよね？



作成したクラス

A screenshot of the Eclipse IDE interface. On the left, the Package Explorer shows a project named 'Sample' with a 'src' folder containing a 'sample' package and a 'Hello.java' file. The 'Hello.java' file is highlighted with a red box. On the right, the code editor shows the following Java code:

```
package sample;

public class Hello {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO 自動生成されたメソッド・スタブ
    }
}
```

The code editor window has a red border around the code area. Labels at the bottom indicate the selected file ('Hello.java') and the inputted program ('入力したプログラム').

そうだ
今回はこのプログラムが
どんな順番で処理されるか
説明する



順番？

ああ

プログラムは
左から右に処理される

右端に来ると次の行へと
処理が進んでいく



→処理1; →処理2; →処理3;

→処理4;

→処理5;

→処理6; →処理7;

→処理8;

実際のプログラムで
処理順を見てみよう



→ int a;
→ a = 1; a = 2; a = 3;
→ a = 4;
→ a = 5;
→ a = 6; a = 7;
→ a = 8;

ふーん
横書きの文章と
同じなのね

そうだ





● プログラムの処理の流れ

プログラムは、横書きの文章と同じように、**左から右**に処理が進みます。そして右端に来ると次の行に（**上から下**に）処理が進みます。これは、普通の横書きの文章の読み方と同じです。

このプログラムの処理は、「**；（セミコロン）**」で区切られます。

説明 プログラムの処理の流れ

```
→処理1; →処理2; →処理3;  
→処理4;  
→処理5;  
→処理6; →処理7;  
→処理8;
```

● 複数行

1つの処理は複数の行にわたって書いても構いません。また、1つの行に複数の処理を書いても構いません。

ソースコード 1行ずつ処理を書く

```
no1 = 1 + 2 + 3 + 4 + 5;  
no2 = 6 + 7 + 8 + 9 + 10;
```

ソースコード 1つの処理を複数行にわたって書く

```
no1 = 1 + 2  
      + 3 + 4 + 5;  
no2 = 6 + 7 +  
      8 + 9 + 10;
```

ソースコード 複数の処理を1つの行に書く

```
no1 = 1 + 2; no2 = 3 - 4; no3 = 5 + 6;
```

3-2. コメント

プログラム中に埋め込む、プログラムとして利用されない文字領域について紹介します。

さて 今回は
プログラムとして
処理されない内容を
プログラム内に書く
方法を説明する



…

?



はあっ？

プログラムなのに
プログラムじゃないの？

疑問に思うのは
当然だろうな



実はプログラムには
動作とは関係ない
メモや解説も
多く書き込むんだ

メモ

解説



このプログラムとして
扱われない書き込みを
コメントと呼ぶ

プログラム

コメント



そしてコメント扱いに
した部分をコメント領域と言い

プログラム

コメント

コメント領域



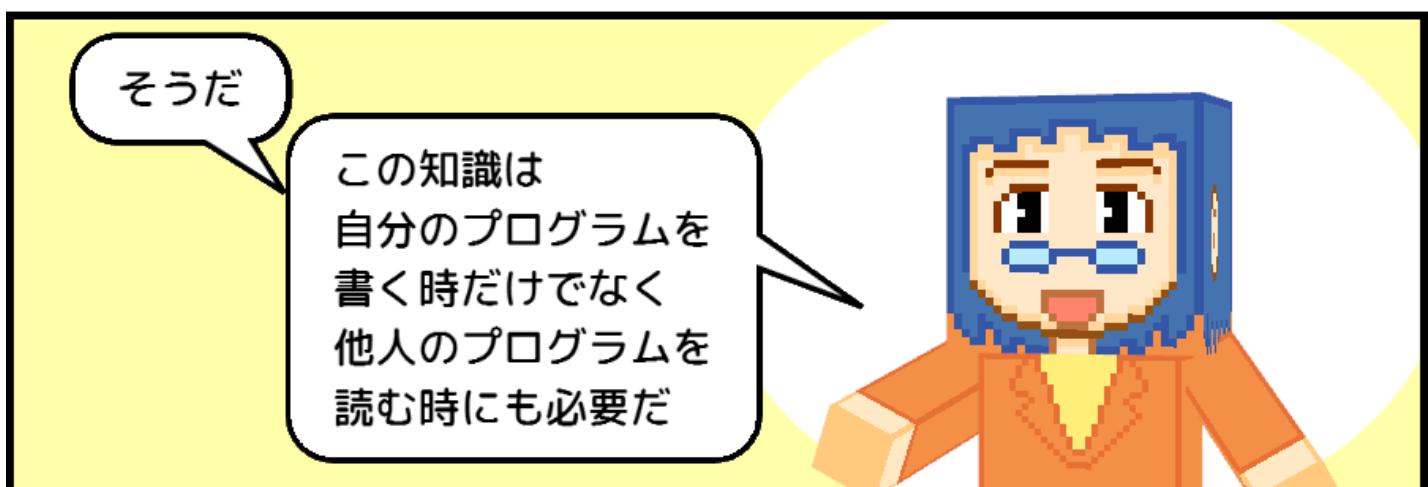
プログラムの一部を
コメント化することを
コメントアウトと呼ぶんだ

プログラム

コメント

コメントアウト





```
int a;  
a = 1;    a = 2;  
a = 3;    a = 4;  
a = 5;    a = 6;
```

この一部に「//」と
書くと その右側が
コメント化される

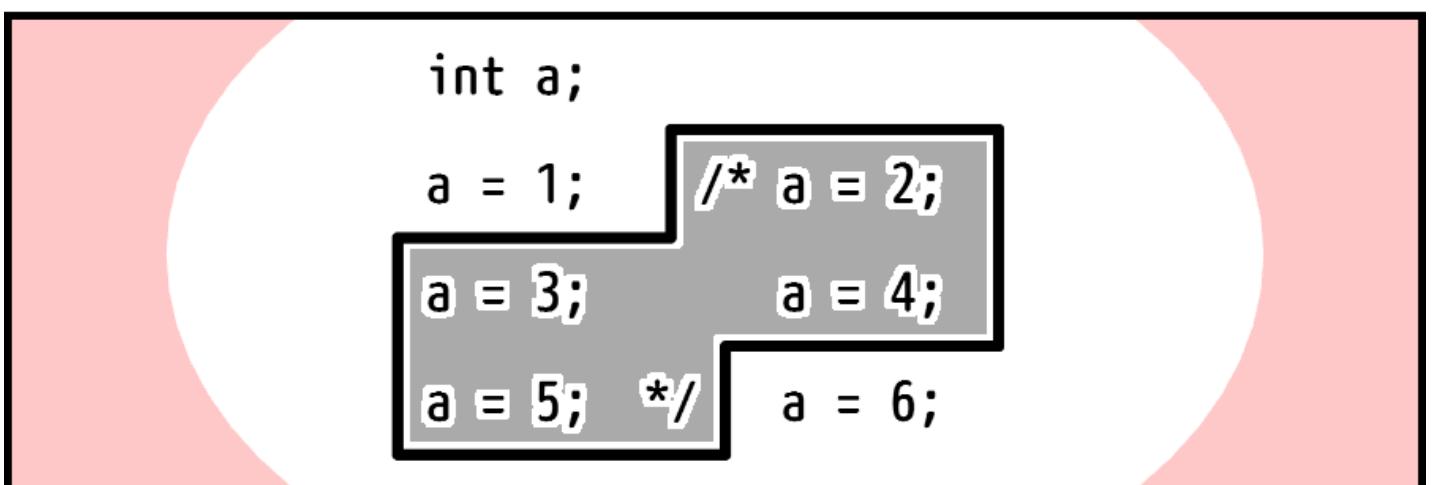


```
int a;  
a = 1;        // a = 2;  
a = 3;        a = 4;  
// a = 5;     a = 6;
```

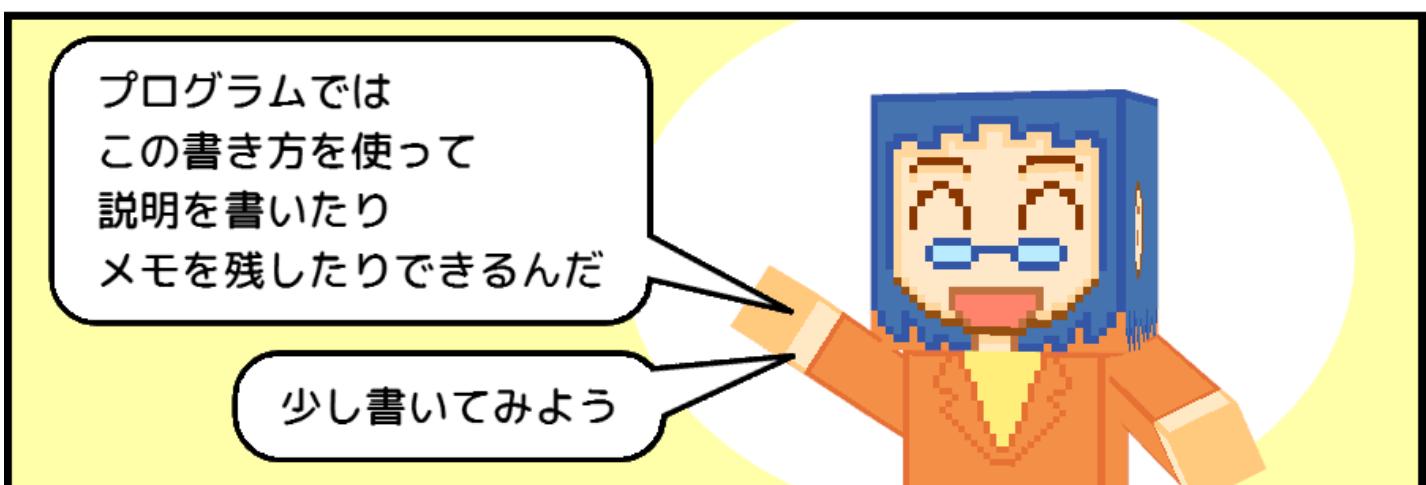
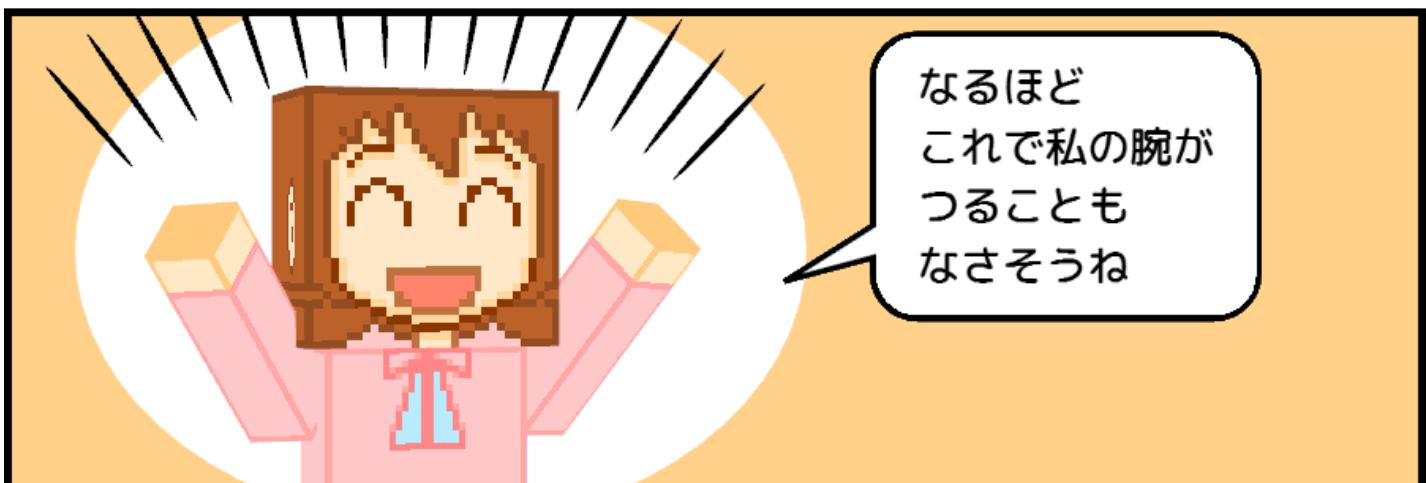


ふーん でも沢山の行に
//って書くのは面倒ね

100行とかあったら
腕がつって死んで
しまうわ



```
/*
a = 1;    a = 2;
a = 3;    a = 4;
a = 5;    a = 6;
*/
```



```
/*  
 * 1~3の数字を設定  
 */
```

```
int a;  
a = 1; // aは1  
a = 2; // aは2  
a = 3; // aは3
```



随分
分かりやすくなっただわね

そうだろう



プログラムだけだと
ソースコードは理解しづらい

またしばらく経って見直すと
何を書いたのか分からなくなる



だからコメントは
しっかりと
書き込んでくれ



分かったわ
前向きに
検討するわ



● コメントとは

プログラムはテキストファイルに書きます。そしてJavaコンパイラは、このテキストファイルを読み取り、Java用の中間ファイルを作ります。しかし、Javaコンパイラは、全ての文字をプログラムと見なすわけではありません。プログラム中には、コンパイラが無視する文字も書き込みます。

この、プログラム中で使われない文字を「**コメント**」と呼びます。

また、コメントが書かれた部分を「**コメント領域**」などと呼びます。そして、プログラムの一部をコメント化することを「**コメントアウト**」と呼びます。

● コメントの活用

ソースコードは、そのままでは読み難いです。そのためコメントを付けて分かりやすくします。また、後で見直した時に必要な情報を書いておきます。

● コメントの書き方1 「//」

コメントを書くには、2つの方法があります。1つ目の方法は「**//**」と書く方法です。この場合、その行の「**//**」の右側が全てコメントとみなされます

ソースコード

1行目のno2以降は処理されない、2行目のno6以降も処理されない

```
no1 = 1 + 2; // no2 = 3 - 4; no3 = 5 + 6;  
no4 = 1 + 2; no5 = 3 - 4; // no6 = 5 + 6;
```

● コメントの書き方2 「/*～*/」

コメントを書く2つ目の方法は「**/***」「***/**」と書く方法です。この場合、記号の間が全てコメントとみなされます。この領域は、複数行にわたっても構いません。

ソースコード

「no2 = 3 - 4;」の部分は無視される

```
no1 = 1 + 2; /*no2 = 3 - 4;*/ no3 = 5 + 6;
```

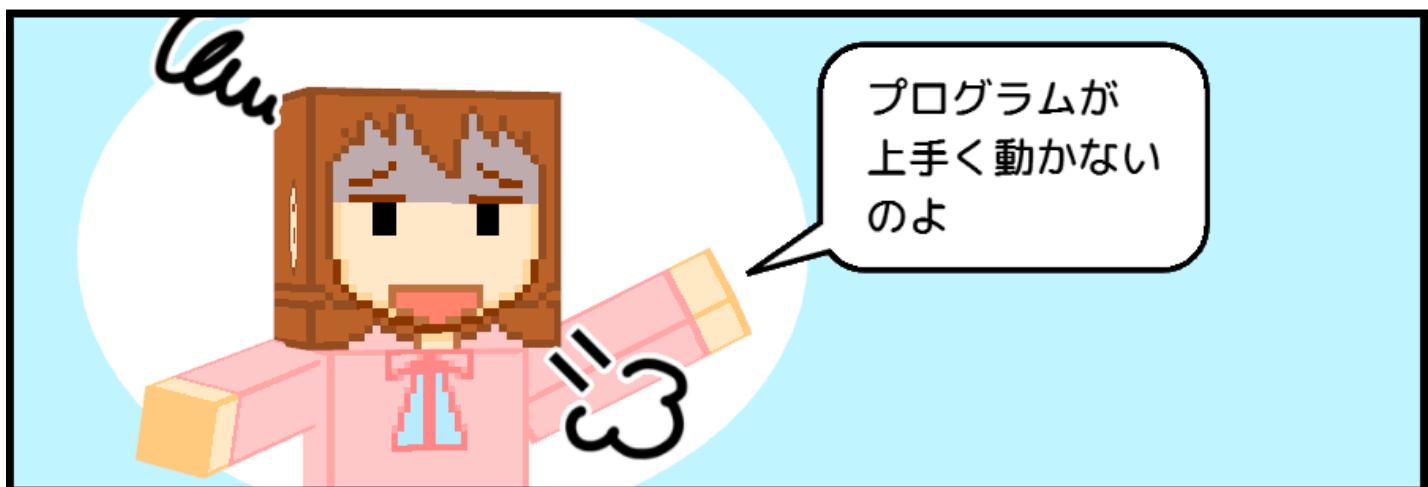
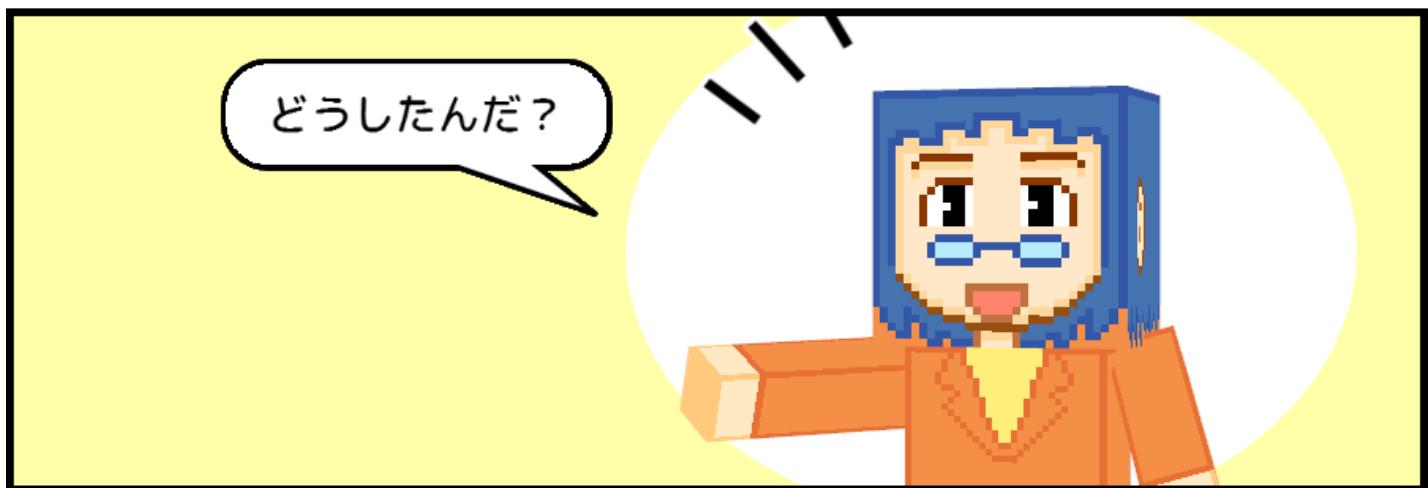
ソースコード

no2から、no6の手前までが無視される

```
no1 = 1 + 2; /* no2 = 3 - 4; no3 = 5 + 6;  
no4 = 1 + 2; no5 = 3 - 4; */ no6 = 5 + 6;
```

3-3. エラー

プログラムが動かない「エラー」について、実行前と実行中の両方について紹介します。



?



どっちのエラー?
何それ

Javaでは プログラムを
実行する前に出るエラーと

プログラムを実行した後に
出るエラーがある



書く前と後?
どういうこと?

そうだな
それじゃあ今回は
Javaのエラーについて
説明しよう



まず実行前に
出るエラーだ

これはプログラムの
書き間違えなどで生じる



書き間違えなんて
気付かないわよ

自分では
分からぬから
間違うんだもの



大丈夫だ

Eclipseは
そういう間違いを
画面に表示して
指摘してくれる



ほうっ
それは気に
なるわね



Eclipseでは各行の左側に
×や！といった
マークが表示される



```
3 public class Sample {  
4     public static void main(String[] args) {  
5         test();  
6         int;  
7     }  
8     static void test() {  
9         int i = 1 / 0;  
10    }  
11}  
12}
```

×と！

×はエラーで
！は修正した方が
よい警告になる



このマークに
マウスを載せると
詳しい内容が
表示される



```
3 public class Sample {  
4     public static void main(String[] args) {  
5         test();  
6     }  
7     テキスト "int" に構文エラーがあります。このテキストの後には VariableDeclarator を指定する必要があります  
8 }  
9     static void test() {  
10    ローカル変数 i の値は使用されていません  
11 }  
12 }  
13 }
```

マウスを載せると
ポップアップ



この場合は
コンソールに
エラーが表示される



A screenshot of a Java IDE's console window. The window title is "Sample (16) [Java アプリケーション] C:\Program Files\Java\jre7\bin\javaw.exe (2013/05/18 15:59:15)". The console output shows a stack trace:

```
<終了> Sample (16) [Java アプリケーション] C:\Program Files\Java\jre7\bin\javaw.exe (2013/05/18 15:59:15)
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at sample.Sample.test(Sample.java:9)
    at sample.Sample.main(Sample.java:5)
```

この表示は
最初の行がエラーの原因で
それ以降の行は発生箇所だ



上の表示だと
Sample.javaの
9行目でエラーが
起きている



sample.Sample.testは
sample/Sampleというパスの
test命令でエラーが
起きているという意味だ



その下にも
同じような行が
あるんだけど何？



これは
sample.Sample.testを
呼び出した場所だ



Sample.javaの5行目
main命令の中で
呼び出されたことを
意味している



こんな感じで
複数行書いてあれば
エラーの呼び出し元を
順にたどれるんだ



また青字を
クリックすれば
対応したコードの場所に
ジャンプできるぞ



この実行前と後の
表示を活用することで
エラーを取り除け
るんですね？



そうだ
上手く活用してくれ



● コンパイルエラーと警告

「Eclipse」でプログラムを書いていると、ソースコードの左側に「x」や「!」といった記号が表示されることがあります。これらは、エラーや警告を表す記号です。この2つの記号について説明します。

説明

実行前のエラー表示と警告表示

x エラー表示

! 警告表示

「x」は、エラーを表す記号です。この記号が表示されている場合は、プログラムに問題があります。問題の多くは、プログラムの書き方が間違っています。

「！」は、警告を表す記号です。この記号が表示される場合は、プログラムを非推奨の書き方で書いています。そのままではエラーとは見なされませんが、可能ならば修正した方がよいです。

どちらの記号の場合でも、この記号にマウスを載せると、エラーや警告の詳細が表示されます。このメッセージはF2を押せば選択できるようになります。

詳細は、英語で表示される場合もあります（日本語化に対応していない内容の場合）。その際、意味が分からなければ、丸ごとコピペして検索エンジンで調べるとよいです。解説しているサイトを見つけることができます。

ソースコードにエラーが表示されている場合は指示に従い、まずはエラーがなくなるようにしなければなりません。そうしなければ、実行したりJavaアプリケーションを出力したりすることはできません。

● 実行時エラー

作成したアプリケーションを実行すると、コンソールにエラーが表示されることがあります。

説明

実行後のエラー表示

```
Exception in thread "main" java.lang.ArithmetiсException: / by zero
at sample.Sample.test(Sample.java:9)
at sample.Sample.main(Sample.java:5)
```

このエラー表示は、最初の行がエラーの原因で、それ以降の行は発生箇所です。

以下、サンプルを元にして解説をします。

ソースコード

エラーの起きるコード [src/sample/Sample.java](#)

```
package sample;

public class Sample {
    public static void main(String[] args) {
        test();
    }

    static void test() {
        int i = 1 / 0;
    }
}
```

出力

実行時エラー

```
Exception in thread "main" java.lang.ArithmeticsException: / by zero
at sample.Sample.test(Sample.java:9)
at sample.Sample.main(Sample.java:5)
```

上記の例では、最初の行の「Exception in thread "main" java.lang.ArithmeticsException: / by zero」がエラーの原因になります。

2行目の「at sample.error.test([error.java:9](#))」は、エラーが起きた場所です。

3行目の「at sample.error.main([error.java:5](#))」は、「sample.error.test」を呼び出した場所になります。

長いプログラムでは、もっと多くの行が表示されます。その場合はいずれも、1つ前の行を呼び出している場所になります。

また、このエラー行「[error.java:9](#)」「[error.java:5](#)」をクリックすれば、Webページのリンクのように、プログラム内のその場所にジャンプできます。このリンク先を確かめることで、エラーの発生理由を調査できます。

3-4. 2進数

プログラムを学ぶ上で、避けては通れない「2進数」について紹介します。

今回から数回は
プログラミング
から離れて
コンピュータの
知識の話をする



じゃあ
楽ができる
のね？

いや
難しい話を
するつもりだ

Cuu



げつ

面倒なことに
なったわね

さてみんなは
2進数という言葉は
知っているか？



・
・
・



あの
知らないん
ですけど



教えて
ください



進数というのは
数の数え方だ

1 2 3 4



?



数え方…
なのですか？

そうだ
普段使っている
数え方は10で
桁が上がる

... 7 8 9

10 11 ...

0	0	[0]
[1]	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
↓	↓	↓
1	8	0

10進数は各桁0～9

0
1
:
9
10
11
:
99
100

10進数は10ごとに桁が上がる

こういった数え方を
10進数と言うんだ



ふーん
じゃあ2進数は
2で桁が上がるの？



十六 むつ

そんなに
簡単ではない
でしょう



いや遊が正解だ
2進数は2で桁が
上がる数え方なんだ



2進数は各桁0～1

0 [1]	0 [1]	[0] 1	0 [1]
↓	↓	↓	↓
1	1	0	1

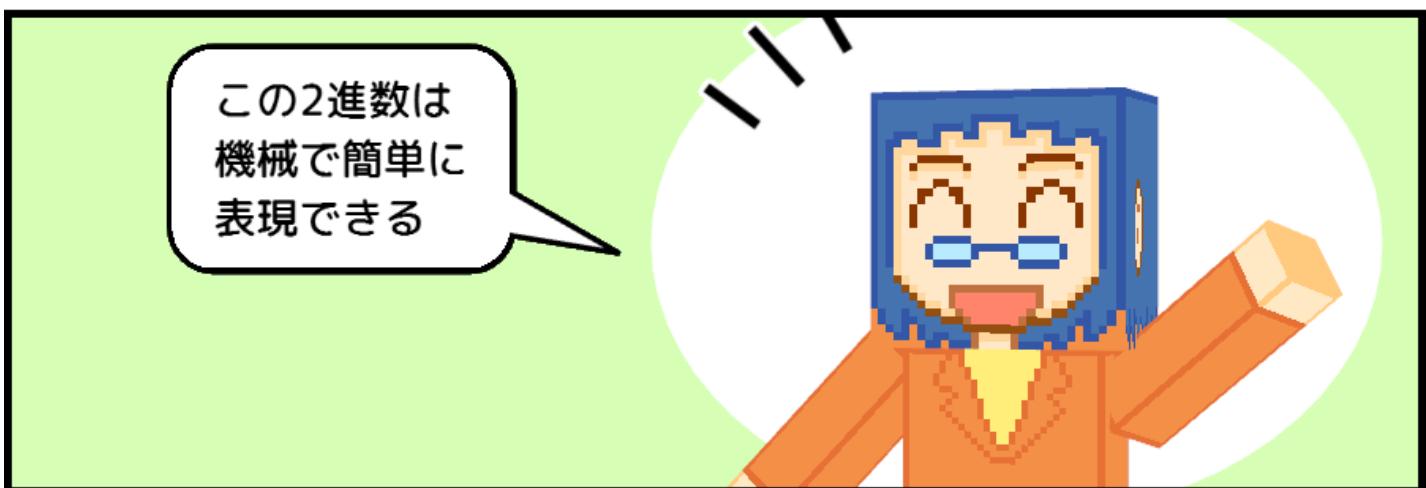
2進数は2ごとに桁が上がる

0
1
10
11
100
101
110
111
1000



よっしゃ！
私の方が
正しかったわ

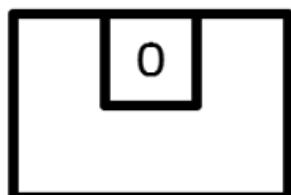




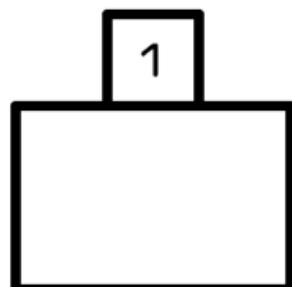
たとえばスイッチの
オフを0 オンを1
にして表すことが
可能だ



スイッチオフ



スイッチオン



2進数のスイッチ表現



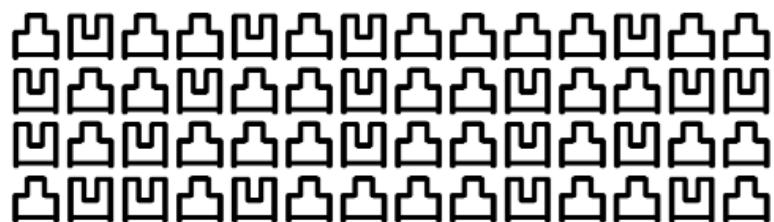
※ このスイッチ1つ分をbitと呼ぶ。上の図は4bit。

コンピュータの情報を
記録するメモリは
こういったスイッチが
大量に並んでいるんだ



メモリの概念図

ぎっしり



Javaの
プログラミングでは
このメモリの様子を
想像した方が分かり
やすい場合が多い



というわけで
あらかじめ2進数と
メモリについて
話をしたんだ



Class



ふーん
難しい仕組みに
なっているのね

● 10進数と2進数

「**進数**」は数字の数え方です。普段使っている10で桁が上がる数え方は「**10進数**」と呼びます。これは人間の手の指が10本なので、利用されている数え方です。

説明 **10進数**

0
1
2
:
9
10
11
12
:
99
100

対してコンピュータでは「**2進数**」という数え方を利用します。これは、2で桁が1つ上がる数え方です。2進数は機械で表現しやすい数え方です。スイッチのオフを0、オンを1として表現できます。

説明 **2進数**

0
1
10
11
100
101
110
111
1000

また、2進数の0、1で表す単位（1桁分）を「**bit (binary digit)**」と呼びます。

説明 **ビットの例**

1.....1bit
11.....2bit
111.....3bit
1111.....4bit

● コンピュータの中身

Windowsなどのパソコンは「**ハードディスク**」と言う装置にアプリケーションの実行ファイルやデータを保存しています。そして、アプリケーションが実行されると「**メモリ**」という装置に、そのデータが読み込まれて、「**CPU (Central Processing Unit : 中央演算処理装置)**」という場所で計算を行います。

メモリは、ハードディスクと同じように情報を格納する場所ですが、ハードディスクよりも圧倒的に高速です。プログラムの基本的な動作は、このメモリ上の情報をCPUで計算して、再びメモリに書き込むというものです。

● メモリ

メモリは、コンピュータが計算などを行う際に、一時的に情報を記録する場所です。

このメモリは、オンオフを切り替えられるスイッチが大量に並んでいるような状態になっています。このスイッチがオフの時は0を、オンの時には1を表します。

この0、1の並びを利用して、コンピュータでは2進数で数値を扱っています。

3-5. 16進数

プログラムを学ぶ上で、避けては通れない「16進数」について紹介します。

さあ今回は
16進数の話を
するぞ



デジャブかしら
前回も似たような
話を聞いた気が
するんだけど

Cuu

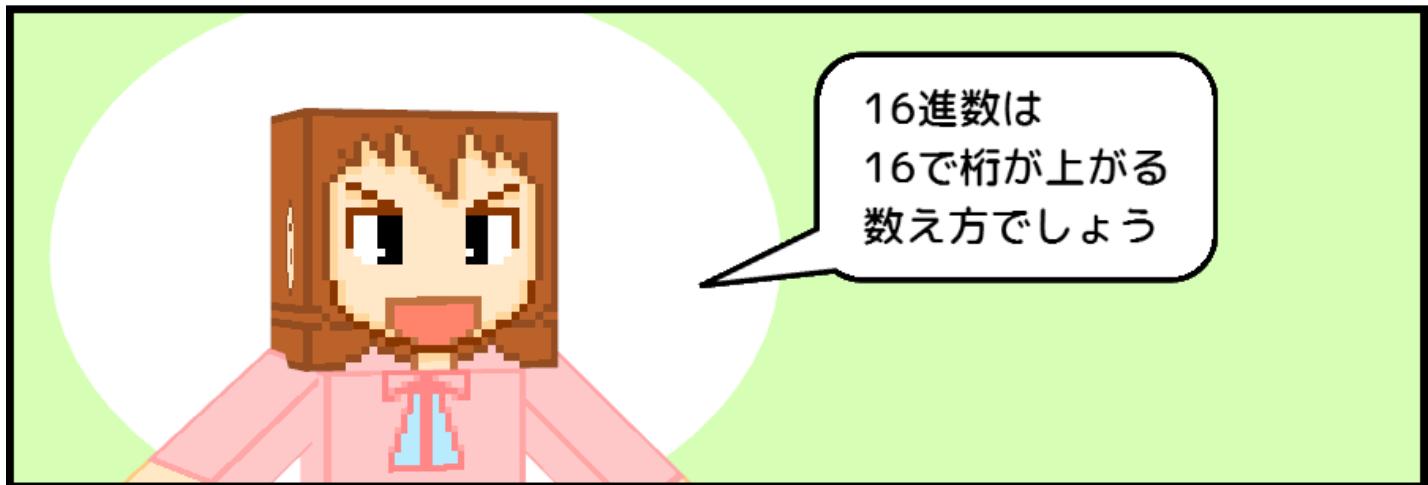
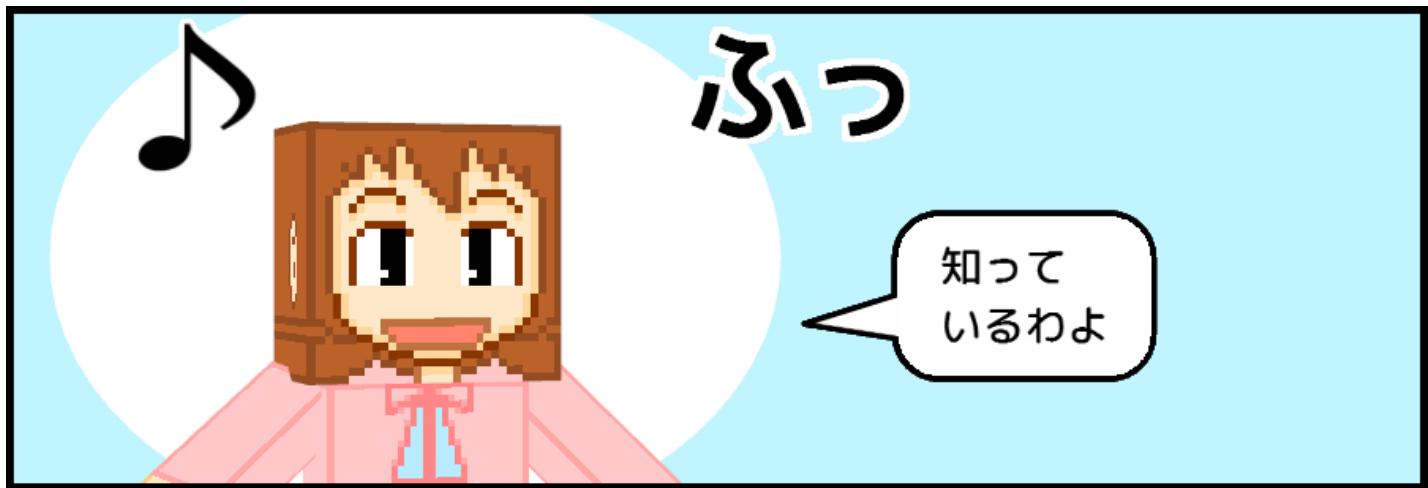


遊ちゃん
前回は2進数
だよ



そうだ 今回は
2進数ではなく
16進数だ







そして人間は
普段10進数を
利用している



違う数え方を
利用していると
色々と面倒そうね

Cuu



ああだから
互いに歩み寄った数え方を
プログラムではよく使うんだ



それが
16進数になる

ビンバツ



うーん
歩み寄ったと
言われても
ピンと来ないわね

それじゃあ
詳しく解説して
いこう

まずは16進数の
1ケタ分の数字を
示すぞ

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
										10	11	12	13	14	15



16進数は各桁0～F

0	0	[0]
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
A	[A]	A
B	B	B
C	C	C
D	D	D
E	E	E
F	F	F
↓	↓	↓
1	A	0

0
1
:
F
10
11
:
1F
20

16進数は16ごとに桁が上がる



それで
このどこが
プログラミングと
関係があるの？

2進数では
頻繁に桁が上がるから
人間には使いづらい



2ずつ桁が上がる

0
1
10
11
100

それに
10進数の数字を
2進数で書くと
非常に長くなる



10進数 「9」は 2進数だと「1001」

10進数 「99」は 2進数だと「1100011」

10進数 「999」は 2進数だと「1111100111」

え?



これじゃあ
何を書いているのか
分からぬわね

そうだろう
でも16進数なら
頻繁に桁が上がら
ないから
使いやすい



10進数 「9」は 16進数だと「9」
10進数 「99」は 16進数だと「63」
10進数 「999」は 16進数だと「3E7」

それに
16進数の1桁分は
2進数と相性が
いいんだ



16進数 「F」は 2進数だと「1111」
16進数 「FF」は 2進数だと「1111 1111」
16進数 「FFF」は 2進数だと「1111 1111 1111」

2進数の1桁分は
オンオフのスイッチ
1つで表現できた



1



実は
16進数の1桁分は
オンオフのスイッチ
4つで表現できるんだ



「 F F F
1 1 1 1
凸 凸 凸 凸」

ちなみに
16進数2桁分は
スイッチ8つで
表現できる



$\Gamma \text{ F } \Gamma$ 1 1 1 1 	$\Gamma \text{ F } \Gamma$ 1 1 1 1
---	---

※ この16進数の2行分(8bit分)を1byteと呼ぶ
1byteは、 $16 \times 16 = 256$ で、0～255の値を表現できる

このスイッチの
数の4や8も
2進数で表現
しやすい数だ



使用メモリ量を
管理している場所

各桁を2進数で管理
 $4 = 0 \sim 11$:
 $8 = 0 \sim 111$:

管理

16進数：1桁と2桁

4bit
1 1 1 1

8bit
1 1 1 1 1 1 1 1

8進数だと各桁が
3bit 6bitになり
コンピュータでは
管理しづらい数になる



使用メモリ量を
管理している場所

各桁を2進数で管理

3 = 0~10 : 
6 = 0~101 : 

管理

8進数：1桁と2桁

3bit

111
凸凸凸

6bit

111 111
凸凸凸 凸凸凸

!



3bit 6bitでも
4bit 6bitと同じ
スイッチ数が必要なのね

それに桁が上がると
端数の計算が
煩雑になる



というわけで
16進数はコンピュータで
扱いやすい数え方なんだ





ふーん
だから16進数を
使うのね

● 16進数

「**16進数**」は、16ごとに桁が上がる数え方です。「0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F」と桁が1つ上がります。

16進数では「A」は10、「B」は11、「C」は12、「D」は13、「E」は14、「F」は15を表します。

表 10進数と16進数の対応

10進数	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
16進数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

● 2進数、10進数、16進数

人間は10進数を、コンピュータは2進数を使います。プログラミングでは、このそれぞれの立場で都合のよい16進数を使います。

16進数の1桁は、2進数の4桁（4bit）で表せます。

表 16進数の1、2桁を2進数で表現

16進数	2進数
F	1111
FF	1111 1111

ここで、10に近い8ではなく16を使うのはなぜかと疑問を持つかもしれません。

8進数ではなく16進数が採用されているのは、8（3bitで桁が上がる）よりも16（4bitで桁が上がる）方がコンピュータには扱いやすいからです。

コンピュータでは、数値はメモリ上に記録されます。そして、数値に使っているメモリーのサイズを、別の場所に記録します。この際、記録用に使われる数字も、2進数で管理されます。

2進数（2を基準とした数え方）では、2で割り切れない3よりも、2で割り切れる4の方が都合がよいです。そのため、8進数よりも16進数の方が、コンピュータにとって都合がよい数字になります。

● byte

コンピュータの世界では「**byte**」という単位がよく出てきます。

プログラムを書く場合、byteは8bit（2進数で8桁の値、 $2^8 = 256$ ）になります。

1byteで表現可能な数値を10進数で表すと「0～255」になります。

1byteで表現可能な数値を2進数で表すと「0～1111111」になります。

1byteで表現可能な数値を16進数で表すと「0～FF」になります。

通常、1byteは2桁の16進数で表現されます（例：3F、FF）。

3-6. 変数という概念

プログラムの動作では最重要の概念である「変数」について紹介します。

今回は変数という
概念について
教えるぞ



いっこうに
Javaの話に
ならないわね



もう少しだけ
我慢してくれ

Class



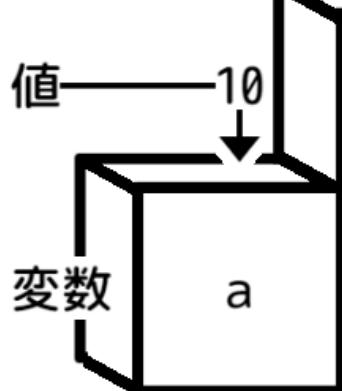
変数は
プログラミング
言語に関係なく
重要な概念だ



仕方ないわね
聞いてあげるわ

変数は
値を格納する
箱のようなものだ

※箱は以降
右の図で
表す



変数の名前には
英数字と一部の
記号が使える

そして小文字の
アルファベットから
書き始めるんだ



price

no1

no2

変数の例

変数

price

変数

no1

変数

no2

この変数には
「=(イコール)」
を使って
値を代入する



値を代入

```
price = 100;  
no1 = 10;  
no2 = 11;
```

値—— 100
↓

変数 price

値—— 10
↓

変数 no1

値—— 11
↓

変数 no2



イコールは
「同じ」という
数学の意味とは
違うのですね

ああだから
注意が必要だ

プログラミングでは
イコールは代入を意味する



?



ふーん
でもなんで
変数なんてものを
使うの？

プログラミングでは
この変数を利用して
様々な計算や処理を行
うんだ



たとえば
割引率の計算を
変数を使わないので
書いてみる



変数を使わないので
割引率を計算

$$100 * 0.7$$

$$1980 * 0.6$$

$$3000 * 0.8$$

こういった計算式を
毎回書くのは面倒だ



変数を使うと
計算式を再利用できる
ようになる



変数を使って
割引率を計算

```
price = 100;  
rate = 0.7;  
result = price * rate;
```

このpriceとrateの
値を変更するだけで
計算結果を変えられる



こういった変数の
利用については
今後詳しく
紹介していく



というわけで
次回からJavaの
話に戻るぞ



● 変数

「**変数**」は、プログラムで使う値を格納するための箱のようなものです。

図 変数と値

↓
値

変数

変数に値を入れるには、「=（イコール）」を使います。左側に変数を、右側に値を書くことで、左辺の変数に値を格納することができます。

図 変数に値を入れる

no = 9;

↓
値9

変数no

↓ ↓ ↓ ↓ ↓

↓
値9

変数no

こうやって変数に値を格納することで、プログラムでは、値を変えるだけで同じ計算を再利用できるようになります。

以下、変数を使わない計算と、変数を使った計算の例を示します。

説明

変数を使わないので割引率を計算

100*0.7

1980*0.6

3000*0.8

ソースコード

変数を使って割引率を計算

```
price = 100;  
rate = 0.7;  
result = price * rate;
```

変数を使えば、このpriceとrateの値を変更するだけで計算を変えられます。そして、プログラムでは、この変数の値を変更する多くのルールが存在します。

Javaでは一般的に、変数は小文字で始めます。