

## テーマ2の目標 プログラミング体験

プログラミングのために開発された技法を学ぶ

- 配列, 文字列 教科書 3.1
- 関数(サブルーチン) 教科書 3.2

## レポート課題

課題2 循環小数

課題3 暗号解読に挑戦

## 本日の講義内容

1. 配列とその使い方 宿題
2. 文字列の処理方法 宿題

# 1. 配列

## 複数のデータを格納できる変数

### 復習: アニメーションプログラムの例

プログラム smile.rb

大きな数を各変数に格納(絵のデザイン)

```
# 出力: スマイルマーク
d1 = 10000000000000000000000000000000
d2 = 10000000000110000110000000000000
d3 = 10000000000110000110000000000000
d4 = 10000000000000000000000000000000
d5 = 10000011000000000000000110000000
d6 = 1000000110000000000001100000000
d7 = 1000000011000000000110000000000
d8 = 10000000001111111100000000000000
d9 = 10000000000000000000000000000000
d10 = 10000000000000000000000000000000
```



めんどうだけど  
これはしょうがない

画面への出力

```
t = 0
while t < 30
  puts(d1)
  puts(d2)
  puts(d3)
  puts(d4)
  puts(d5)
  puts(d6)
  puts(d7)
  puts(d8)
  puts(d9)
  puts(d10)
  puts()
  sleep(0.1)
  t = t + 1
end
```



めんどう

## 1. 配列

# 複数のデータを格納できる変数

復習: アニメーションプログラムの例

プログラム smile.rb

大きな数を配列に格納

```
# 出力: スマイルマーク  
d[0] = 10000000000000000000000000000000  
d[1] = 1000000000011000011000000000000  
d[2] = 1000000000011000011000000000000  
d[3] = 10000000000000000000000000000000  
d[4] = 1000001100000000000000011000000  
d[5] = 1000000110000000000001100000000  
d[6] = 1000000011000000000110000000000  
d[7] = 1000000000111111110000000000000  
d[8] = 10000000000000000000000000000000  
d[9] = 10000000000000000000000000000000
```

配列 d

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

# 1. 配列

## 複数のデータを格納できる変数

### 復習: アニメーションプログラムの例

プログラム smile.rb

画面への出力

```
t = 0
while t < 30
  puts( d[ 0 ] )
  puts( d[ 1 ] )
  puts( d[ 2 ] )
  puts( d[ 3 ] )
  puts( d[ 4 ] )
  puts( d[ 5 ] )
  puts( d[ 6 ] )
  puts( d[ 7 ] )
  puts( d[ 8 ] )
  puts( d[ 9 ] )
  puts()
  sleep(0.1)
  t = t + 1
end
```

```
t = 0
while t < 30
  k = 0
  while k < 10
    puts( d[ k ] )
    k = k + 1
  end
  puts()
  sleep(0.1)
  t = t + 1
end
```

```
for k in 0..9
  puts( d[ k ] )
end
```

先週習いたかった !!

## 1. 配列

# 複数のデータを格納できる変数

同種のデータを多数扱うときに便利

例: 6個の総和を求める

配列 a

0	1	2	3	4	5
2	4	6	8	10	12

プログラム sum6.rb

```
a = [2, 4, 6, 8, 10, 12]
```

↑ 配列の初期値の決め方

```
s = 0  
k = 0  
while k < 6  
  s = s + a[k]  
  k = k + 1  
end
```

```
puts(s)
```

```
s = 0  
for k in 0..5  
  s = s + a[k]  
end
```

添え字

Ruby では配列の添え字は0 から

数学でも  
似た表現があるよ

$$\text{総和} = \sum_{k=0}^5 a_k$$

↑  
添え字

# 1. 配列

## 複数のデータを格納できる変数

例: 不定個の総和を求める

配列 a

0	1	2	3	4	5	...
						...

$$\sum_{k=0}^n a_k$$

プログラム sum.rb

```
a = gets().split.map(&:to_i)
n = a.length
# 以下が総和の計算部分
s = 0
for k in 0..(n-1)
  s = s + a[k]
end
puts(s)
```

↑  
整数を配列に入力する方法。(これは決まり文句)

「配列名.length」で配列の要素数が得られる。

↑ ピリオド

```
$ ruby sum.rb
-3 8 19 -4
20
$
```

実行例

個々に空白で区切る  
改行がデータの終わり

# 1. 配列 (Ruby での使い方)

例: 最大値を求める

配列 a

	0	1	2	3	4	5	...
							...

$\max(a_0, a_1, \dots, a_n)$

max.rb

```
a = gets().split.map(&:to_i)
n = a.length

# 以下が計算部分
max = -10000 # マイナス無限大と言える数
maxj = -1
for j in 0..(n-1)
```

宿題 (1)

```
end
puts(max, maxj)
```

```
$ ruby max.rb
-3 8 19 -4
19
2
$
```

実行例

## 2. 文字列

文字が並んだもの(1文字の場合もある)  
Ruby では文字が格納された配列で表わす

(例)

str = "Coffee"

引用符

	0	1	2	3	4	5
str	C	o	f	f	e	e

str[ 2 ]

n = str.length ← str に格納されている文字列の長さ

stringPrint.rb

```
puts("文字列を入力しよう")
str = gets().chomp
n = str.length

for i in 0..(n-1)
  puts( str[i] )
end
```

文字列を入力する方法

```
$ ruby stringPrint.rb
文字列を入力しよう
Ice%%cream
I
c
e
%
...
```

Terminal の画面



## 補足：文字と文字コード ASCII

- ・ 文字もコンピュータ内では数字（正確には 2 進列）で表わされる
  - ・ 文字を 2 進列で表すこと（または表したものを）を **文字コード** という
  - ・ 文字コードはいろいろあるが、英数字を表すもので 世界的で最も普及しているのが **ASCII** である
- ※ 仮名，漢字は SJIS, EUC, Unicode 等で表わされている

（例）

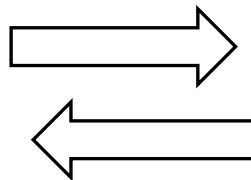
ASCII を求める

```
ss = "Cabcz"  
aa = ss.unpack("C*")
```

文字列に直す

```
aa = [ 67, 97, 98, 99, 122 ]  
ss = aa.pack("C*")
```

	0	1	2	3	4
ss	C	a	b	c	z



	0	1	2	3	4
aa	67	97	98	99	122

## 2. 文字列(Ruby での使い方)

例: 英小文字のみ画面に出す

英小文字のみ画面に出力するプログラムを作ろう!

abcPrint.rb

```
puts("文字列を入力しよう")
ss = gets().chomp
leng = ss.length
aa = ss.unpack("C*")

for i in 0..(leng-1)
  宿題 (2)
end
```

ヒント: a の ASCII = 97

～

z の ASCII = 122

```
$ ruby abcPrint.rb
文字列を入力しよう
Ice%%cream
c
e
c
r
e
a
m
$
```

Terminal の画面

# CS第1 レポート課題2(予告)

## 課題 循環小数の循環を止めよ！

配列は同じようなデータを統一的に処理するには便利な道具だが、それ以外にも賢い使い方がいくつかある。その例を考えてみよう。

```
junkan.rb
puts("分母 d を下さい")
d = gets().to_i
print("1 / ", d, " を求めます¥n")
stop = 0; leng = 0; x = 1
while stop != 1
  x = x * 10
  q = x / d
  leng = leng + 1
  print(leng, ":", q, "¥n")
  sleep(0.5) # ゆっくり表示するため
  x = x % d
  if x == 0
    stop = 1
  end
end
end
```

**これだと無限に  
小数を出し続ける  
場合がある！！**

画面に出力する別方法。  
改行をしないので見栄え  
良く出力できる。最後の  
"¥n" は改行のコード。

# まとめ: Ruby (1)

## 【演算子】

演算	使用例	意味
+	$x + y$	x と y の足し算
-	$x - y$	x から y の引き算
*	$x * y$	x と y の掛け算
/	$x / y$	x を y で割った商
%	$x \% y$	x を y で割った余り
**	$x ** y$	x の y 乗

## 【関係演算子】

関係	使用例	意味
>=	$x \geq y$	x は y より大きいかまたは等しい
>	$x > y$	x は y より大きい
==	$x == y$	x は y と等しい
!=	$x != y$	x は y は等しくない
<	$x < y$	x は y より小さい
<=	$x \leq y$	x は y より小さいかまたは等しい

## まとめ: Ruby (2)

### 【論理演算子】

論理記号	使用例	意味
&&	x && y	x と y の論理積 (両方が真のとき真)
	x    y	x と y の論理和 (少なくとも一方が真のとき真)
!	!x	x の否定 (x が真のとき偽, x が偽のとき真)

## まとめ: Ruby (3)

**【配列】**

初期設定     `aa = [ 0, 0, -5, 4 ]`  
                 `aa = Array.new(4)` ← 要素数 4 の配列生成し aa とする  
                 `aa = Array.new(4, 0)` ← 各要素の初期値が 0

指定方法     `aa[ i ]` = aa の *i* 番目 (添え字 *i* は 0 から)

コマンド     `aa.length` = 配列 aa の長さ (= 要素数)

                 ※「添え字」は「インデックス」(index) ともいう.

**【文字列】**

初期設定     `s = "Coffee+milk"`

指定方法     `s[ i ]` = *s* の *i* 文字目 (添え字 *i* は 0 から)

コマンド     `s.length` = 文字列 *s* の長さ

`a = s.unpack("C*")` ← *s* の各文字を ASCII に直して  
                 配列 *a* に格納する

`s = a.pack("C*")` ← 配列 *a* の各数字を文字に直して  
                 文字列用変数 *s* に格納する

## まとめ: Ruby (4)

### 【繰り返し文】

```
while 条件式  
  ...  
end
```

← 条件式の成立している間  
... を繰り返す

```
for m in a..b  
  ...  
end
```

← 変数 m の値を a から b まで  
1 ずつ増加させながら ... を繰り返す

### 【条件分岐文】

```
if 条件式  
  ..(A) .. ← 条件式の成立したときは ..(A) ..を実行  
else  
  ..(B) .. ← そうでないときは ..(B) ..を実行  
end
```

省略可

## まとめ: Ruby (5)

### 【入出力】

`puts( a, b, "hello", c)` ← 変数 `a`, `b` の値, 文字列 `hello`, 変数 `c` の値を改行しながら画面に表示する

`print( a, b, "hello", c, "¥n")` ← 上と同様. ただし改行はしない. 空白も空けない. 従って, 最後には改行記号を画面に出すことで改行させる.

※ ¥ は「バックスラッシュ」という記号を表わしている. 改行を意味する記号として用いられることが多い.

Mac では Option キーと ¥ キーを同時に押すとタイプできる.