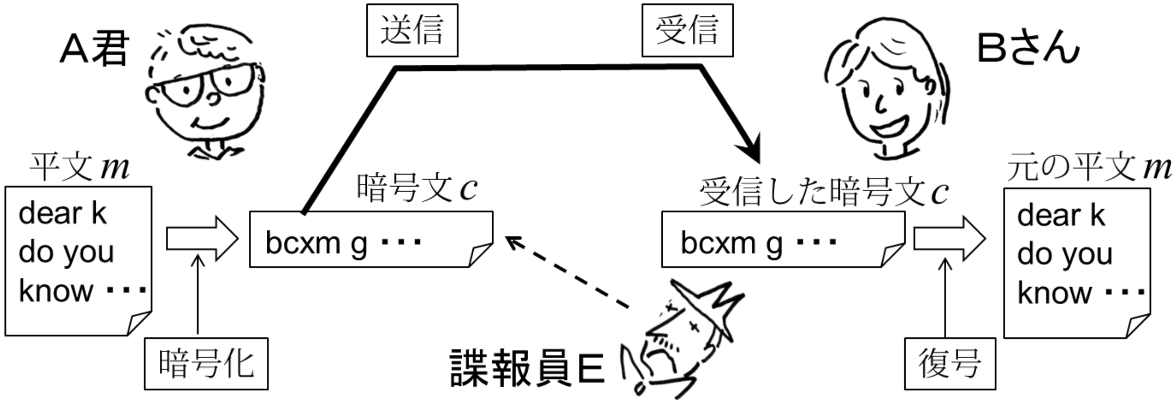
CS第1　テーマ2

テーマ2　プログラミング体験

レポート課題

2　循環小数
3　暗号解読に挑戦



A，B，Eの各々が使うソフトを作成する

講義の進め方

1. プログラミング技法
   - 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2

2. プログラミング技法についての実習
3. 課題2の説明
   - 暗号について　教科書 5.3

レポート課題3

4. 暗号化，復号，解読のプログラム作成の実習

CS第1　テーマ2

テーマ2　プログラミング体験

課題の進め方

1. プログラミング技法
   - 配列，文字列とその使い方
   - 関数，サブルーチン
2. プログラミング技法についての実習 ← 次週

本日の講義内容　プログラミングの基本的道具

1. 配列とその使い方 教科書 3.1 宿題
2. 文字列の処理方法 宿題
3. 次週予告：レポート課題2

1. 配列 ⟷ 複数のデータを格納することのできる変数

```
puts(d1); puts(d2); puts(d3); ...puts(d10)
```

同じような変数，同じような操作を個々に書くのは面倒

たとえば数学でも「**添え字**」を使う

$a_1, a_2, \ldots$ 　さらには　総和 $= \sum_{k=1}^{10} a_k$

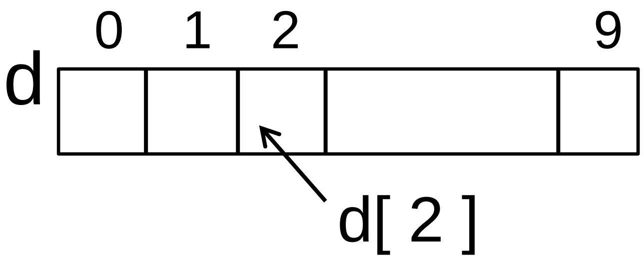
多くのプログラミング言語でもそのような書き方は可能．

Ruby では

（例）

```
d[0] = 10000100
d[1] = 10111010
d[2] = 10101100
d[3] = ...
```

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



先週習いたかった !!

・配列の**添え字**（インデックス）は 0 から．

1. 配列 ⟷ 複数のデータを格納することのできる変数

（例）総和を求める

$$総和 = \sum_{k=0}^{5} a_k$$

（個数は 6 個としよう）

sum6.rb

```
a = [2, 4, 6, 8, 10, 12 ]
k = 0;  s = 0
while k < 6
  s = s + a[ k ]
  k = k + 1
end
puts(s)
```

← 初期値の設定

a

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

（例）便利な書き方 for 文

sum6.rb

```
a = [2, 4, 6, 8, 10, 12 ]
s = 0
for  k  in 0..5
  s = s + a[ k ]
end
puts( s )
```

・配列の**添え字**（インデックス）は 0 から.

※ Ruby では，変数名はすべて小文字開始にすること.（教科書では大文字始まりにしている場合もあります.）

・変数 k の値を 0 から 5 まで変えながら繰り返す for 文

1. 配列 （例）総和を求める

$$総和 = \sum_{k=0}^{n} a_k$$

・加算するデータは入力データとして与える.
・個々に空白で区切る. 改行がデータの終わり.

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)
```

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

Terminal の画面

・整数を配列に入力する方法.（これは決まり文句として使って下さい. ）

・「 配列名 . length」で配列の要素数が得られる.
↑ピリオド

1. 配列 （例）最大値を求める

$$最大値 = \max(a_1, a_2, \ldots, a_n)$$

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
$
```

Terminal の画面

2. 文字列

↔ 文字の列．Ruby では配列のように扱うことができる．

（例）

str = "Coffee"

↑ 引用符 ↑

str

0	1	2	3	4	5
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

3. 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） として格納されている.
- 文字を 2 進列で表すこと（または表したもの）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.

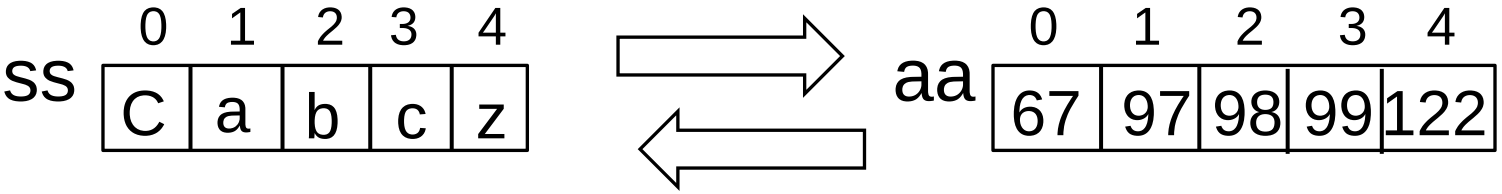
（例）

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*")
```



2. 文字列 （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

3. 次週のレポート予告

レポート課題2

テーマ　循環小数の循環を止める

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
```

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．

3. 次週のレポート予告

レポート課題2

テーマ　循環小数の循環を止める

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
```

これだと無限に小数を出し続ける場合がある!!

やるべきこと

配列をうまく使って循環小数になっても止まるようにプログラムを改良する．

まとめ（復習）　Ruby での書き方（その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 >= 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 <= y	x は y より小さいかまたは等しい

まとめ（復習） Rubyでの書き方（その2）

【論理演算子】

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

まとめ　Ruby での書き方（その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 での書き方（その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) ・・
else
   ・・(B)・・
end
```

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

（else の部分は）省略可

まとめ　Ruby での書き方（その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.