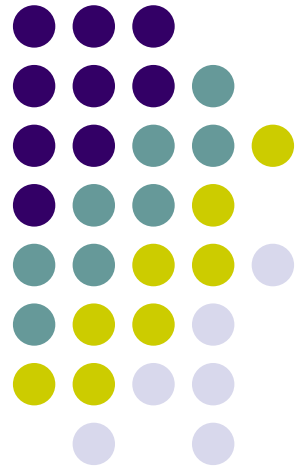
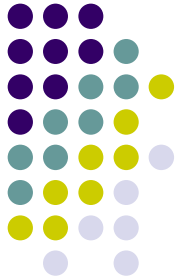
# 情報処理概論

工学部 物質科学工学科
応用化学コース
機能物質化学クラス



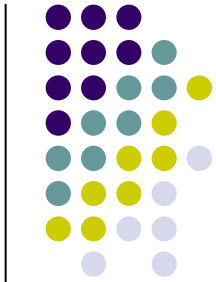
## 第4回

2005年　5月12日



○　計算

- 数学関数の利用
- 表示形式の調整



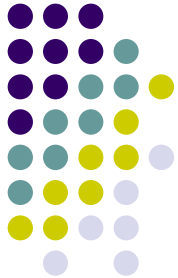
# 計算結果の表示

- 文字列と同様write文を利用する

- 複数の計算結果や文字列を表示する場合
  カンマ( , ) で区切る

```
program sample1
  write(*, *) -20.5 * 3.7 + 4.0
  write(*, *) 'total = ', 95 + 33 + 25 + 84
stop
end program
```

# Fortran での計算



- 四則演算： 基本的に算数，数学と同じ
  - ただし，記号の違いに注意

    × → ＊

    ÷ → ／

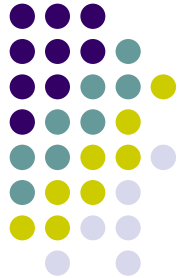
    べき乗 → ＊＊

- 計算順序も算数，数学の数式と同じ
  - ( ) → べき乗 → ＊，／ → ＋，－
  - 順番が不安なときは （ ） を使う

- 括弧は （ ） だけを利用する
  - {} や [] は使わない.
  - ただし，何重に使っても良い

    例） (((x + y) * (z – (x – y)**2)) + (z – y) * 2)/2

# 整数と実数

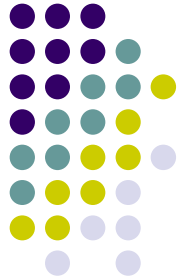


- 同じ値でも整数と実数はコンピュータの中では表現が全く違う

- Fortranでは
  小数点が付いた数値データは実数
  小数点が付いてない数値データは整数

  例）
  ３　　整数
  ３．０　実数

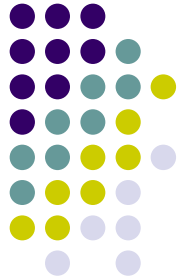
# 実数と整数の割り算



- 実数の割り算
  2.0/3.0=0.666666…

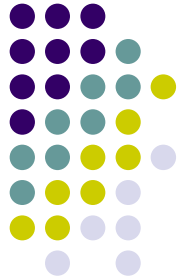
- 整数の割り算は切り捨てて整数型に
  4/3=1
  27/5=5
  10/11=0

# Fortran での計算における注意点



- 整数と実数が混在した式は避ける.
  - 小数点以下が切り捨てられて予想外の結果が得られる場合がある.
  - 混在が避けられない場合はすべて実数で記述する.
    (1/2)*(10.5 + 5.9)
    → (1.0 / 2.0) * (10.5 + 5.9)

- 乗算記号の省略は不可
  (1 + 2)(3 + 4)　→　(1 + 2) * (3 + 4)

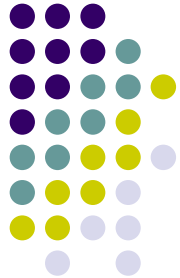


- 計算

○ 数学関数の利用

- 表示形式の調整

# 数学関数の利用

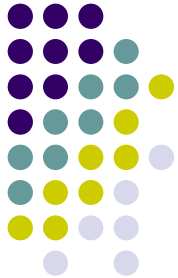


- Fortran には予め主要な数学関数が組込まれている

→　組込み関数

  - 平方根
    sqrt( )
  - 三角関数
    sin( ) , cos( ) , tan( ) ,
    asin( ) , acos( ) , atan( )
  - 対数
    log( ) , exp( )

# 組み込み関数の利用



- 組込み関数を利用する場合はプログラムの最初に intrinsic文で予め名前を宣言する

- 利用法:

  関数名(値)

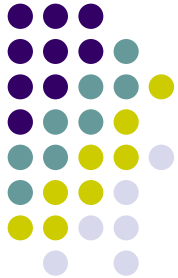
  実行すると関数値に置き換わる.

  例) sqrt(4.0) → 2.0

- 関数の中で計算をしたりさらに関数を呼んでも良い

```
program sample2
  intrinsic sqrt, exp
  write(*, *) sqrt(exp(2.0) * 1.0 – 0.5)
stop
end program
```

# 三角関数の角度指定

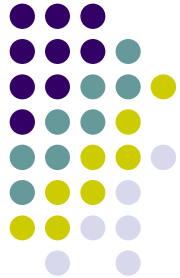


- 三角関数の角度はラジアンで指定する
  - 1.0 ラジアン　＝　180.0 / π 度
  - 1.0 度　＝　π / 180.0 ラジアン

例)　60度の　cos

```
cos(60.0 * 3.14159265 / 180.0)
```

# 1つの文を
# 複数行にまたがって記述



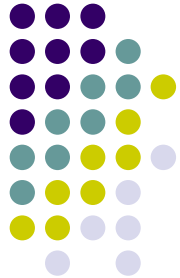
- 計算式などが長くなると，プログラムが読みにくい．

```
write(*, *) 'total = ', 80 + 90 + 24 + 45 + 78 + 52 +
25 + 66 + 74
```

- 行末に＆を書くと次の行に継続する

```
write(*, *) 'total = ', 80 + 90 + 24 &
                      + 45 + 78 + 52 &
                      + 25 + 66 + 74
```

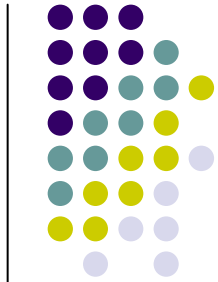
- 注意：　文字列の途中では継続できない



- 計算
- 数学関数の利用

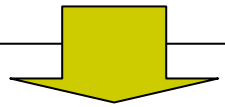
○　表示形式の調整

# 表示の桁揃え



- 例) 表示内容を列挙するだけだと
  桁はバラバラ

```
program sample3
  write(*, *) 'John', (51.0 + 63.0 + 72.0) / 3.0
  write(*, *) 'Paul', (6.0 + 15.0 + 8.0) / 3.0
  write(*, *) 'George', (83.0 + 76.0 + 91.0) / 3.0
  write(*, *) 'Richard', (67.0 + 82.0 + 86.0) / 3.0
stop
end program
```

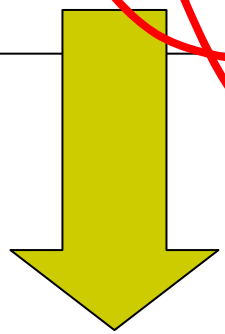


```
John 62.0000000
Paul 9.66666698
George 83.3333359
Richard 78.3333359
```

# 書式の指定

- 書式を指定すると桁を揃えることができる

```
program sample4
  write(*, '(A10,1X,F5.2)') 'John', (51.0 + 63.0 + 72.0) / 3.0
  write(*, '(A10,1X,F5.2)') 'Paul', (6.0 + 15.0 + 8.0) / 3.0
  write(*, '(A10,1X,F5.2)') 'George', (83.0 + 76.0 + 91.0) / 3.0
  write(*, '(A10,1X,F5.2)') 'Richard' &
                          , (67.0 + 82.0 + 86.0) / 3.0
stop
end program
```

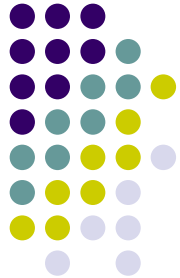


文字列を10桁の幅に表示
1桁分の空白表示
実数を5桁の幅に表示
（ただし小数点以下は2桁まで表示）

```
      John 62.00
      Paul  9.67
    George 83.33
   Richard 78.33
```

# 書式の指定方法

- write文の括弧内で指定
  `write(*, '(書式)')` 表示内容



- 書式で用いる記号

| | |
|---|---|
| I*w* | 表示スペース *w*桁に整数を表示 |
| F*w*.*m* | 表示スペース *w*桁に実数を表示（小数点以下 *m*桁） |
| E*w*.*m* | 表示スペース *w*桁に指数形式で実数を表示<br>（小数点以下 *m*桁） |
| A*w* | 表示スペース *w*桁に文字列を表示 |
| *n*X | *n*桁の空白を表示 |

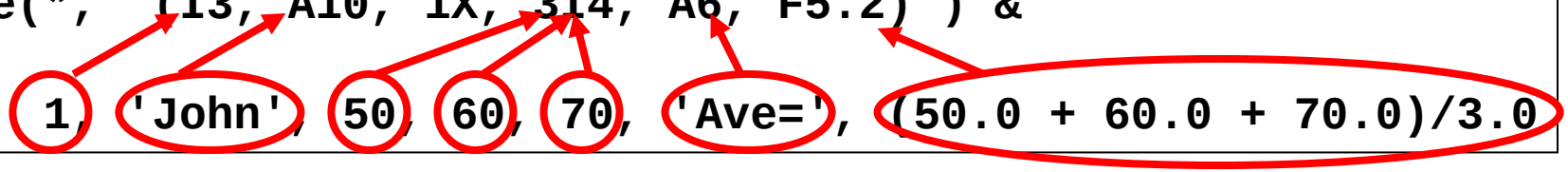
- 複数のデータを表示する場合，カンマ（,）で区切る
- 同じ指定が続く場合，以下のように回数を指定できる

`F5.2, F5.2, F5.2` → `3F5.2`

# 書式の利用上の注意

- 書式で指定された表示形式と表示データは数，種類とも一致していなければならない

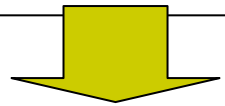
```
write(*, '(I3, A10, 1X, 3I4, A6, F5.2)') &
     1, 'John', 50, 60, 70, 'Ave=', (50.0 + 60.0 + 70.0)/3.0
```



  - 一致していない場合，異常表示やエラーが発生

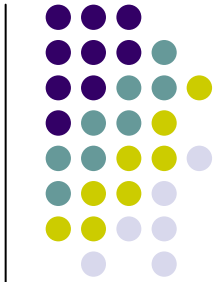
# 書式の利用例

```
write(*, '(I3, A10, 1X, A6, I4, A6, I4, A6, I4, A6, F5.2)') &
       1, 'John', 'Math=', 50, 'Eng=', 60, 'Chem=', 70 &
       , 'Ave=', (50.0 + 60.0 + 70.0)/3.0
write(*, '(I3, A10, 1X, A6, I4, A6, I4, A6, I4, A6, F5.2)') &
       2, 'Paul', 'Math=', 6, 'Eng=', 15, 'Chem=', 8 &
       , 'Ave=', (6.0 + 15.0 + 8.0)/3.0
write(*, '(I3, A10, 1X, A6, I4, A6, I4, A6, I4, A6, F5.2)') &
       3, 'George', 'Math=', 83, 'Eng=', 76, 'Chem=', 91 &
       , 'Ave=', (83.0 + 76.0 + 91.0)/3.0
write(*, '(I3, A10, 1X, A6, I4, A6, I4, A6, I4, A6, F5.2)') &
       4, 'Richard', 'Math=', 67, 'Eng=', 82, 'Chem=', 86 &
       , 'Ave=', (67.0 + 82.0 + 86.0)/3.0
```



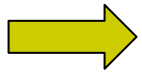
```
  1      John  Math=  50  Eng=  60 Chem=  70  Ave=60.00
  2      Paul  Math=   6  Eng=  15 Chem=   8  Ave= 9.67
  3    George  Math=  83  Eng=  76 Chem=  91  Ave=83.33
  4   Richard  Math=  67  Eng=  82 Chem=  86  Ave=78.33
```

# 書式の便利な使い方



- 同じパターンが続く場合 （ ） でまとめる

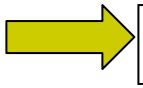
```
A6, I4, A6, I4, A6, I4
```



```
3(A6, I4)
```

- 文字列データを書式中に記述する

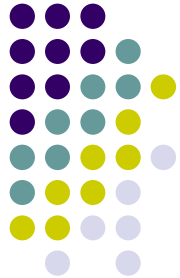
```
write(*, '(A6, F5.2)') 'Ave =', (50.0 + 60.0 + 70.0) / 3.0
```



```
write(*, '(" Ave = ", F5.2)') (50.0 + 60.0 + 70.0) / 3.0
```

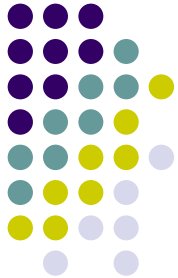
- 何度も使う書式を1回にまとめる
  - format文

# format文の利用例



```
program sample5
  write(*, 100) &
        1, 'John', 'Math=', 50, 'Eng=', 60, 'Chem=', 70 &
        , (50.0 + 60.0 + 70.0)/3.0
  write(*, 100) &
        2, 'Paul', 'Math=', 6, 'Eng=', 15, 'Chem=', 8 &
        , (6.0 + 15.0 + 8.0)/3.0
  write(*, 100) &
        3, 'George', 'Math=', 83, 'Eng=', 76, 'Chem=', 91 &
        , (83.0 + 76.0 + 91.0)/3.0
  write(*, 100) &
        4, 'Richard', 'Math=', 67, 'Eng=', 82, 'Chem=', 86 &
        , (67.0 + 82.0 + 86.0)/3.0
  100 format(I3, A10, 1X, 3(A6, I4), '  Ave=', F5.2)
stop
end program
```

# format文の利用法

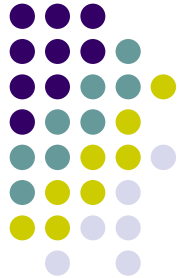


- 先頭に行番号を指定

```
行番号  format(書式)
```

  - write文等から参照
  - 別の format文に同じ行番号を対応させないよう注意

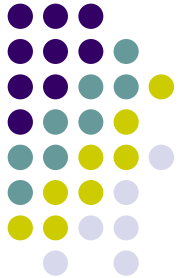
# 演習



- 半径 4.5cmの円に内接する正三角形の面積を計算し，表示するプログラムの作成

- とりあえず書式は用いなくてもよい
  - 完成後、時間に余裕があったら試してみる

# 既存のプログラムを元に作ると簡単



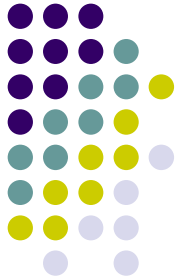
- まず作業ディレクトリに移動し、
（5月になったので may というディレクトリを作っても良い）
emacs で以下のプログラムを入力.

```
program sample2
  intrinsic sqrt, exp
  write(*, *) sqrt(exp(2.0) * 1.0 – 0.5)
stop
end program
```

- コンパイル,実行し,正しく動作することを確認.

- emacs で再度このプログラムを開き,
計算式を演習の内容にあわせて変更する.

# 今までに多かったミス



- ファイル名の末尾が .f90 になっていなかった

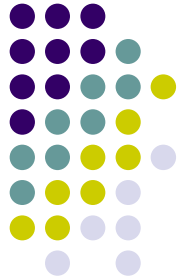
- プログラム名に使えない記号を使っていた
  program test.f90

- 乗算記号＊を忘れていた
  2.0 ( 3.0 + exp(4.0))

- , と . を打ち間違えた
  30,0 * 3,14159265

- プログラム中の文字列以外に全角文字が入っていた
  - 半角／全角 キーの押し忘れ（特に空白）

# 補足:
# 計算機の中での整数の扱われ方

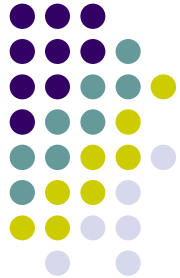


- どんなデータも必ず2進数で格納
  - 先週の講義で説明した通り
- 整数には，通常 2進数 32桁を使用
  - 扱える整数の範囲:
    $0 \sim 2^{32} - 1$
    (= 0 ～ 4,294,967,295)
    または
    $-2^{31} \sim 2^{31} - 1$
    (= -2,147,483,648 ～ 2,147,483,647)

0000 ... 000
0000 ... 001
0000 ... 010
...
1111 ... 111

32桁

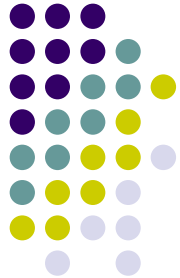
# 補足：
# ビットとバイト



- 1ビット(bit)　＝　2進数の1桁
  - コンピュータにおける情報の最小単位

- 1バイト(byte)　＝　8ビット
  - コンピュータの記憶場所（番地）の単位
    - なぜ8ビット?
      - 半角英数字全部に番号付け可能（$2^8$=256）
      - 2のべき乗だと2進数で扱いやすい
    - 32ビット整数は4バイト分の記憶場所を使用

# 補足:
# 2進数の負数



- x + (-x) = 0 なので
  足して0になる数(補数)を算出すればよい

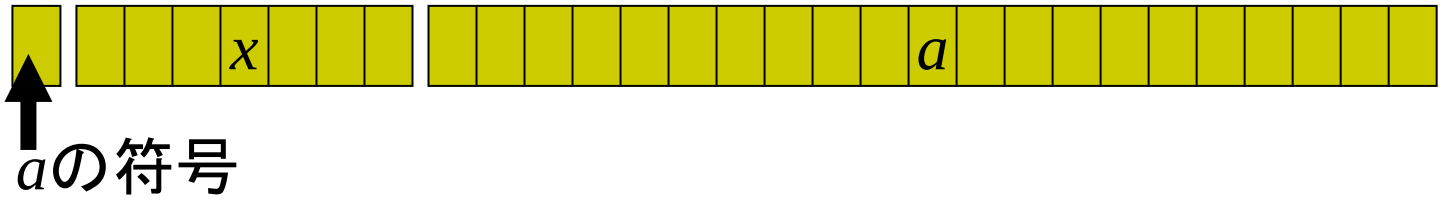
- 2進数の補数の算出は簡単:
  0と1を反転させて1を足す
  - 例えば2進数4桁で整数を表す場合:
    (10進表記を ( )$_{10}$, 2進表記を ( )$_2$で表すとする.)
    $(5)_{10}$ ＝ $(0101)_2$
    $(-5)_{10}$ ＝ $(1011)_2$
    $(5)_{10} + (-5)_{10} = (0101)_2 + (1011)_2 = (10000)_2$
    - 4桁を超える部分は無視される.
- コンピュータの2進数では符号付の値の場合,
  最初の1桁が符号になる.
  0 → 正または0
  1 → 負

# 補足:
# 計算機の中の実数

- コンピュータ内で実数は

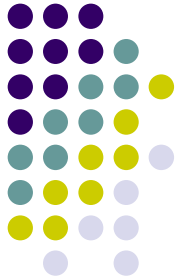
  $a \times 2^x$

  として扱われ



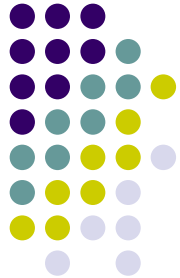

  の形で表現する
  - ただし $0.0 \leqq a < 1.0$
  - $a$: 仮数, $x$: 指数
- 2進数の桁数に応じて表現できる絶対値の範囲や精度が決まる.
  - $a$の桁数 → 精度
  - $x$の桁数 → 最大絶対値

# 補足：
# 実数の精度



- 単精度
  - ４バイト（＝３２ビット）
  - メモリが少なかった頃は良く用いられた
  - 表現可能な桁数が少ないので誤差が出易い
- 倍精度
  - ８バイト（＝６４ビット）
  - 近年，メモリが潤沢に確保できるようになり
    誤差の不安からこちらが主流に

# 補足：
# 複素数



- （実部，虚部）で表現
  - 例） (0.1, -2.3)
  - 単精度： ８バイト，　倍精度：１６バイト
  - こちらも現在は倍精度が主流

注意：　関数の引数としての複素数
　後述する関数の引数として複素数を渡す場合，
　以下の通り　dcmplx を用いて
　複素数であることを明記する.

```
exp(dcmplx(0.0,30.0*3.14/180.0))
```