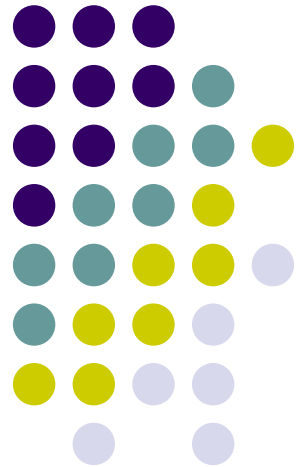
情報処理概論

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



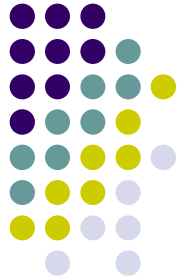
第5回

2005年　5月19日

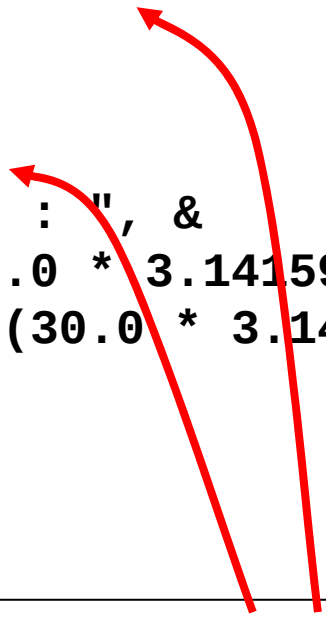
先週の演習

- 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

先週の演習の正解例

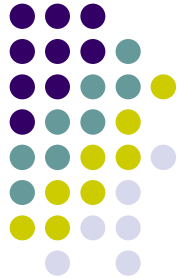


```
program triangle
  ! mathematical functions
  intrinsic sin, cos

  ! inscribed triangle
  write(*, *)  "Menseki : ", &
     2.0 * 4.5 * cos(30.0 * 3.14159265 / 180.0) &
     * (4.5 + 4.5 * sin(30.0 * 3.14159265 / 180.0)) &
     / 2.0

stop
end program
```



コメント（注釈）

プログラム中のコメント



- プログラムの説明
  - 長いプログラムになると，作った本人でも
    意味が分かりにくい場所が増えてくる
  - 他人が書いたプログラムはなおさら分かりにくい

- Fortran90 では！から行末までを
  「コメント（注釈）」として扱う
  - プログラムとしては何の意味も持たない部分
    - コンパイル時に無視される

  - プログラムの説明に利用

コメントの記述

- 基本的に後で自分や他人がプログラムを読む手助けとして記述する
  - プログラムやサブルーチンの説明
  - 変数の説明
  - 複雑な処理を行っている場所の説明

```
! SAMPLE2: Very simple test program
program sample2

  ! Program Body
  write(*, *) -20.5 * 3.7 + 4.0   ! Output result

stop
end program
```

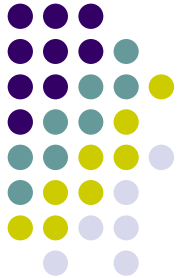
今日の目標

- 任意の半径の円について、内接する正三角形の面積を求める
  - 半径は”実行時”に入力

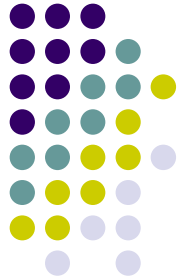
```
% f90 triangle.f90 -o triangle
% ./triangle
Hankei?
4.5
 Menseki :   26.30552163995232
% ./triangle
Hankei?
10.0
 Menseki :   129.9038105676658
```

キーボードから入力

利用する機能



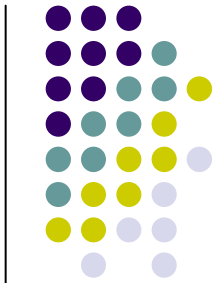
- 変数
  - 値を格納する場所
- read文
  - データの入力



○　変数

- read文

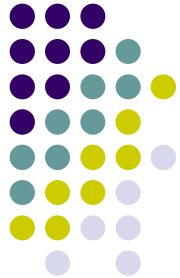
変数を利用したプログラム例（1）



- 半径 r

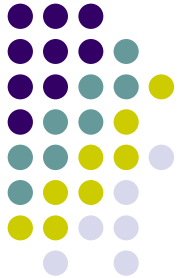
```
program triangle
  implicit none
  real(8) :: r

  ! mathematical functions
  intrinsic sin, cos

  r = 4.5    ! Hankei

  ! inscribed triangle
  write(*, *) "Menseki : ",  &
     2.0 * r * cos(30.0 * 3.14159265 / 180.0) &
     * (r + r * sin(30.0 * 3.14159265 / 180.0)) &
     / 2.0

stop
end program
```

変数を利用したプログラム例(2)



- 途中の計算結果(ラジアンの計算部分) rad

```
program triangle
  implicit none
  real(8) :: r, rad

  ! mathematical functions
  intrinsic sin, cos

  r = 4.5D0    ! Hankei
  rad = 30.0D0 * 3.14159265D0 / 180.0D0  ! Radian

  ! inscribed triangle
  write(*, *)  "Menseki : ", &
        2.0D0 * r * cos(rad) * (r + r * sin(rad)) / 2.0D0

stop
end program
```



変数

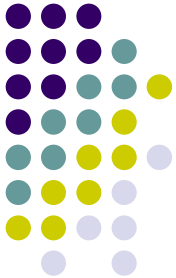
- 値を格納する場所

- 数学の”変数”とちょっと違う
  - 変数の名前と格納するデータの種類(=型) を必ず予め宣言

  - こんな使い方もできる
    - 変数 a の値を一つ増やす

```
a = a + 1
```

変数の宣言

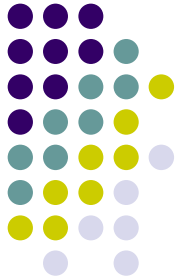


- プログラム中で使う変数の名前とデータ型の定義
  例)

```
real(8) :: r, rad
```

- 場所はプログラムの先頭
  - 宣言以外の文が始まる前
  - 関数の宣言 intrinsic は変数の宣言の先でも後でも可
- 1つの変数について宣言は1回だけ
- 同じデータ型の変数は , で区切って列挙できる

データの型



- 格納する値の種類
  - 整数、実数、複素数、文字
  - 主に使うのは以下の二つ

    real(8)　　倍精度実数型 (倍精度＝8バイト)

    integer　　整数型

  - 他に

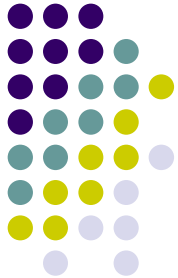
    real　　単精度実数型（単精度＝4バイト）

    character　文字型（1バイト）

    complex(8)　　倍精度複素数型

    complex　　単精度複素数型

implicit none
暗黙的な型宣言機能の無効化

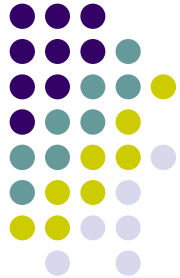


- 「暗黙的(implicit)な型宣言」機能
  - Fortranの標準機能
  - 予め宣言されずに使われている変数を名前に応じて自動的に型宣言
    - 最初の文字が ｉ～ｎで始まる変数は整数型
    - それ以外は単精度実数型

- 「暗黙的な型宣言」は便利だが危険
  - 間違った名前の変数を利用しても気づきにくい.
    - 例) 間違ってred を使っていてもエラーが出ない

```
program triangle
  real(8) :: r, rad
  r = 4.5D0
  red = 30.0D0 * 3.14159265 / 180.D0
  write(*,*)  r*cos(rad)*(r + r*sin(rad))
 ...
```

implicit none
暗黙的な型宣言機能の無効化

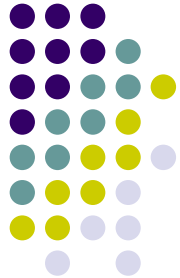


- 暗黙的な型宣言の機能を無効にすると
  - 明示的に宣言した変数以外はプログラム中で利用できない
  - 宣言されていない変数を使うとコンパイラが
    エラーメッセージで指摘してくれる

例) 以下のプログラムはエラーとなる

```
program triangle
  implicit none
  real(8) :: r, rad
  r = 4.5D0
  red = 30.0D0 * 3.14159265 / 180.D0
  write(*,*)  r*cos(rad)*(r + r*sin(rad))
 ...
```

変数へのデータの格納(代入)

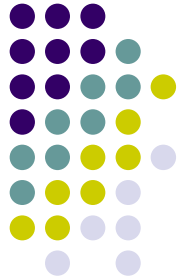


- ＝
  - 左側に「データを格納する変数」
    右側に「格納するデータ(もしくはその計算式)」
  - 例)

```
r = 4.5D0
```

```
rad = 30.0D0 * 3.14159265D0 / 180.0D0
```

倍精度実数の表記



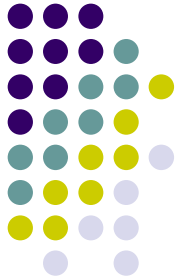
- 倍精度実数の計算を行うプログラムでは
  実数の最後に D0 (ディー・ゼロ)を付ける

```
rad = 30.0D0 * 3.14159265D0 / 180.0D0
```

  - D0 が付いていない実数は単精度として扱われる
    → 倍精度変数を使う意味が無い

  - 単精度実数＝4バイト使用、倍精度＝8バイト使用
    - 表現できる精度が違う
    - 演習を参照

変数に格納されたデータの利用



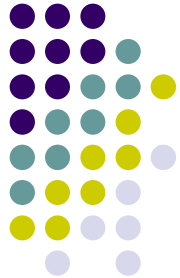
- write文でそのまま表示

```
write (*, *) 'Radian: ', rad
```

- 計算式や関数，サブルーチンの呼び出しに利用

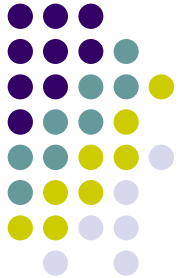
```
menseki = 2.0D0 * r * cos(rad) * (r + sin(rad))/2.0D0
```

変数を使う上での注意(1)
変数の名前



- 変数の名前に利用できるのは
  半角英数字と _
  - 先頭はアルファベット
  - 31文字以内

変数を使う上での注意(2)
データの格納

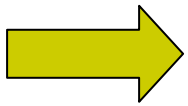


- Fortranの ＝ は ← に近いイメージ
  - ”等しい”ではなく”格納”
    - 以下の式も OK (t の値を一つ増やす)

```
t = t + 1
```

  - 変数 a と b に 0 を代入したい場合 :

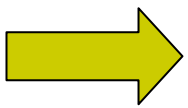
```
a = b = 0
```



```
a = 0
b = 0
```

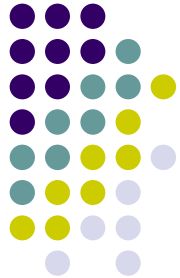
  - a + b の結果を変数 c に代入したい場合 :

```
a + b = c
```



```
c = a + b
```

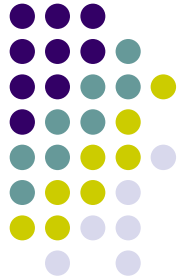
演習： 変数を使ったプログラム （ついでに倍精度と単精度の比較）



- 以下のプログラムを作成し翻訳実行する

```
program precision
  implicit none
  real(8) :: p1
  real :: p2
  intrinsic atan
  p1 = 4.0D0*atan(1.0D0)
  p2 = 4.0 * atan(1.0)
  write(*, *) p1, p2
stop
end program
```

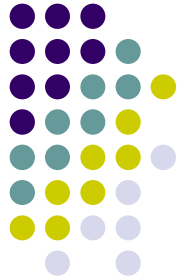
計算からπを導出する方法



- $\tan(\pi/4) = 1$
  なのでπは以下の式から計算できる
  $\pi = 4 * \tan^{-1}(1)$
  - Fortranでは $\tan^{-1}(1)$ → atan(1.0D0)
    πを使うプログラムでは，最初に計算しておくと便利

    例えば

```
program triangle
  implicit none
  real(8) :: pi, r, rad
  ! mathematical functions
  intrinsic sin, cos, atan

  pi = 4.0D0 * atan(1.0)
  r = 4.5D0     ! Hankei
  rad = 30.0D0 * pi / 180.0D0  ! Radian
  ...
```



- 変数

○ read文

プログラム中の値

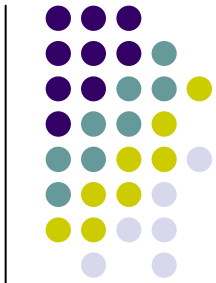
- 今まで: プログラムに全ての値を記述
  - 例) 円の半径

```
r = 4.5
```

→ 別の半径で計算したい場合,
・プログラム編集 (emacs test.f90)
・コンパイル (f90 test.f90 -o test)
・実行 (./test)
を再度行う必要がある.

- 実行時に半径を入力できれば
  - プログラム編集とコンパイルは一回だけ.
  - 実行のたびに半径を指定できる.

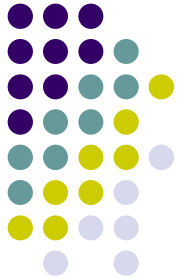
キーボードからのデータ入力
read 文



- 利用法
  read(*, *) 変数1, 変数2, ...
- 例: 半径を入力すると円の面積を
  表示するプログラム

```
program circle
  implicit none
  real(8) :: r,pi

  pi = 4.0D0 * atan(1.0D0)
  write(*, *) 'Hankei?'
  read(*, *) r
  write(*, *) 'Menseki : ', r * r * pi

stop
end program
```

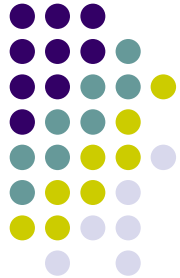
実行例



- 翻訳して実行

```
% f90 circ.f90 –o circ
% ./circ
Hankei?
5.0
 Menseki :  78.53981633974483
% ./circ
Hankei?
100.0
 Menseki :  31415.92653589793
%
```

read の書式指定



- read でも write と同様に書式を指定できる

```
read(*, '(F5.2)') r
```

  - 実際は入力データの桁合せが面倒なので
    あまり使わない