4. Basic のプログラミング

§4.1　For 文

コンピュータは単純な『繰り返し作業』を得意とする。まず、繰り返し回数が決まっている場合のプログラムを作る。

Check Point	1. For 文の文法を理解する。 2. ループの概念をつかむ。

4.1.1　文法

① For ループのブロック

繰り返しを記述するには『繰り返しループ』の『始め(For)』と『終わり(Next)』を指定する。繰り返しループを一つの構造ブロックとして捉える（図 4.1.1）。

```
For　ループ変数 = 初期値 To 最終値 Step きざみ値
    繰り返される文
Next ループ変数
```

図 4.1.1

② 繰り返し回数

繰り返し回数を設定するために『**ループ変数**』を指定する。また、ループ変数の『**初期値**』と『**最終値**』で繰り返し回数が決まる。

> 注意:
> Step きざみ値 を省略すると“ きざみ値 ”が 1 となる

図 4.1.2 の場合、ループを回るたびに 1 ずつ増加させて、ループ変数 『 i 』の値が 10 になるまで For と Next で囲まれた文を繰り返す。

ループする回数が増えるにつれて『 Cells(i ,1) 』が、『 Cells(1, 1) 』、『 Cells(2, 1) 』、・・・ 、『 Cells(10, 1)』と変化する。

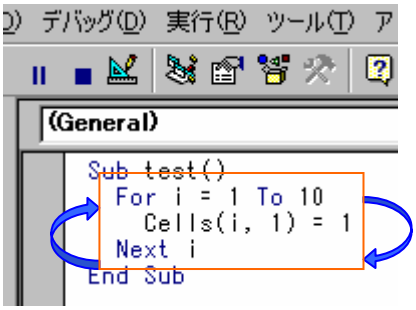
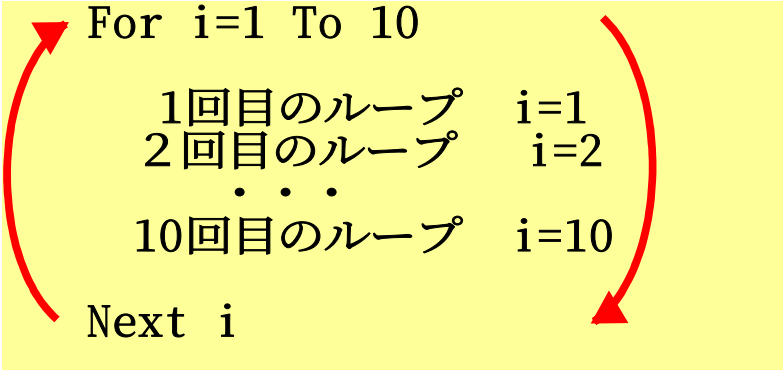


図 4.1.2



課題 1.　　ループ変数の値を表示しなさい！

課題 2.　　1 から 10 まで横に書き出すにはどうしたら良いか？

課題 3.　『 0 ～ 360 』まで 10 おきに、整数 θ を発生させる。
さらに、y＝sin(θ) の計算をする。

4.1.2　Forの応用

①　X = X + 1 の意味

『＝（イコール）』は代入しなさいという意味で、数学の等式の意味ではない！つまり、『 X = X + 1 』は、X という変数に『 1 』を加え、それを新しい値として X（自分自身）に代入しなさいということである（図 4.1.3）。

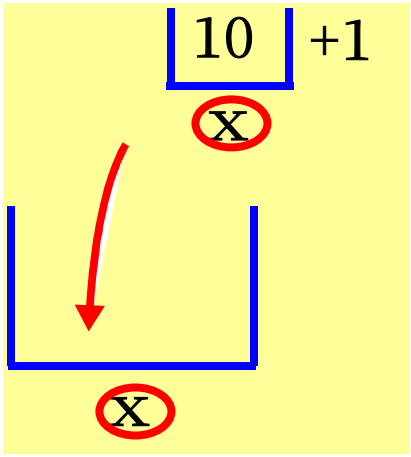


図 4.1.3

②　奇数表示

『 n = 1 』は『変数』という入れ物に数値『 1 』を代入することです（図 4.1.4）。

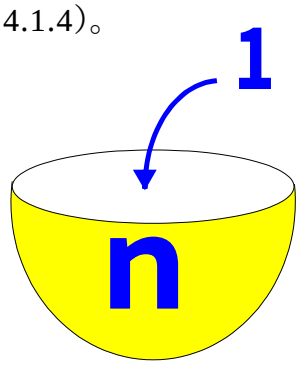


図 4.1.4

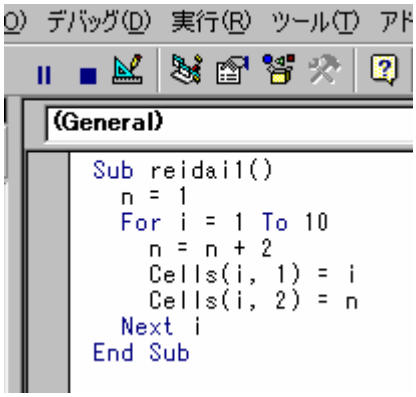


図 4.1.5

図 4.1.5 は奇数を表示するプログラムである。

課題 1.　偶数を表示するプログラムを作りなさい！

課題 2.　借金の複利計算をしなさい！元金 G 円、年利 r %で Y 年借金すると、
借金の合計は、$G(1 + r / 100)^Y$ 円となる。

③　点数の合計の計算

図 4.1.6 のプログラムは、合計点の変数『 **Sum** 』の初期値として『 **0** 』 を代入する。

『 For ループ 』中で乱数を使って、点数を発生させる。『 **Rnd** 』 は、『0 以上 1 未満』の乱数を発生する関数である。

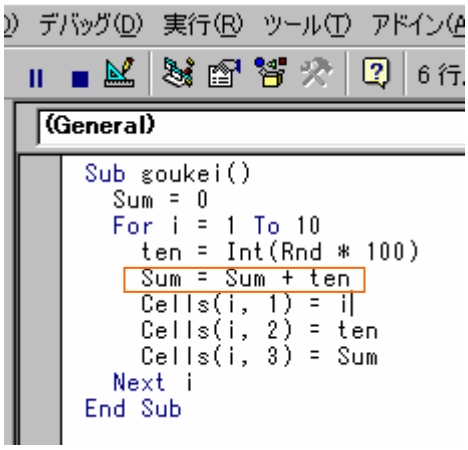


図 4.1.6

注意：

『 **=RAND()** 』は、Excel のコマンドであるが、VBA のコマンドは『 **Rnd** 』となる。

『 **Int(x)** 』は、x を越えない最大の整数である。『 **Int(Rnd*100)** 』でランダムな 2 桁の整数が発生する。

また、『 **Sum = Sum + ten** 』は、合計点の変数『 **Sum** 』に 2 桁の整数 『 **ten** 』 を加算する。

例えば、

Int(-2.3)　→　-3

Int(2.5)　→　2

④　$\sum_{k=1}^{N} k = 1+2+3+4+\cdots+k+\cdots+N$　の計算

『 **Sum = Sum + i** 』と変更するだけで数学の公式の証明ができる(図 4.1.7)。

変数『 **Sum** 』の数値は、図 4.1.8 のように変化する。

```
Sub wa()
  n = 10                ' n=10 までの和を計算する
  Sum = 0               ' 変数 Sum に初期値0を代入
  For i = 1 To n
    Sum = Sum + i       ' (新しいSum)←(古いSum) + i
  Next i
  Cells(1, 1) = n
  Cells(2, 1) = Sum
  Cells(3, 1) = n * (n + 1) / 2
End Sub
```

図 4.1.7

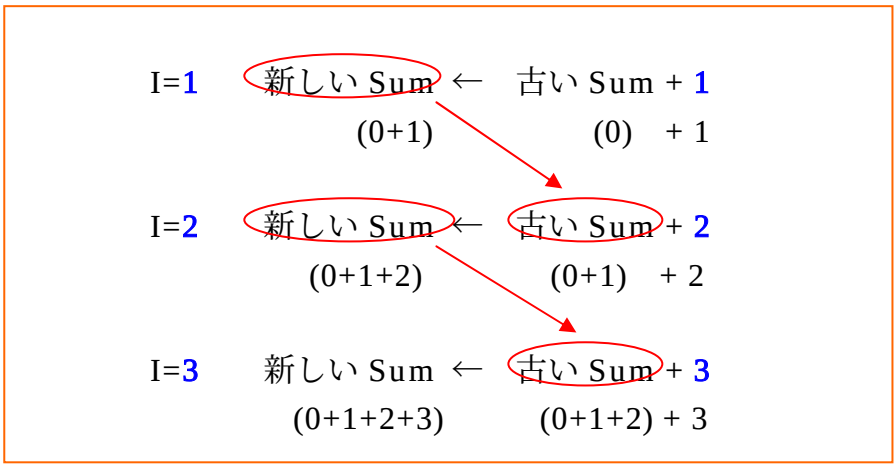


図 4.1.8

⑤　A = A * 2 の意味

変数『 A 』を 2 倍して、自分自身『 A 』に代入することである。これを繰り返すと『 2^n 』の計算ができる（図 4.1.9)。

変数『 A 』の数値は、図 4.1.10 のように変化する。また、初期値を『 A=1 』としないと、A の値は『 0 』のままになる。

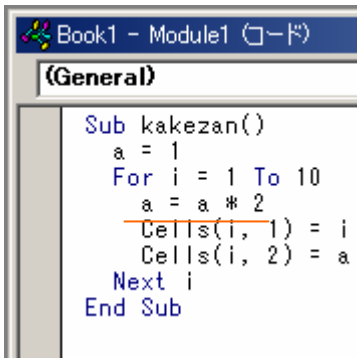


図 4.1.9

注意：

変数の初期値は自動的に『 0 』になります。特に、割り算のときに注意する必要がある。

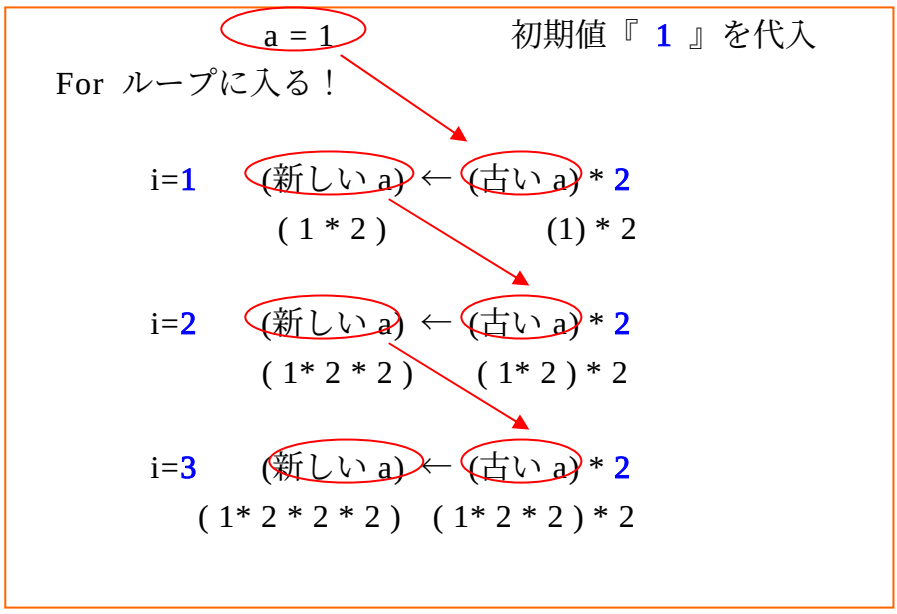


図 4.1.10

課題 1　平均点を計算しなさい！

課題 2　分散を計算しなさい！

課題 3　$\sum_{k=1}^{N} k^2 = 1^2 + 2^2 + 3^2 + \cdots + N^2 = \frac{N(N+1)(2N+1)}{6}$　を証明しなさい！

課題 4　n！（階乗）の計算をしなさい！

分散は、$\frac{\sum_{k=1}^{N}(x_k - \bar{x})^2}{N}$　で計算される！

4.1.3　多重ループ

For ループの中に For ループを入れることができる。複雑なプログラムをブロック単位で考え単純化して、多重ループをマスターする。

① 2 重ループ

多重ループを作るときに、『**ループ変数**』が同じにならないように気をつける。まず、『 **For ループ（ループ変数 i ）** 』と『 **For ループ（ループ変数 j ）** 』を用意する（図 4.1.11）。

```
For i =1 to 10
        …
        …
Next i
```

```
For j =1 to 10
        …
        …
Next j
```

図 4.1.11

『 **i ループ** 』ブロックの中に『 **j ループ** 』を入れると図 4.1.12 のようになる。

図 4.1.12 のように For ループをブロック構造とみなして、ブロックごとに段落をつけるとプログラムが見やすくなる。

プログラムの流れが視覚的に分かりやすくなることによって、プログラムのミスが少なくなる！

```
Sub taju()
  k = 0
  For i = 1 To 5
    For j = 1 To 5
      k = k + 1
      Cells(k, 1) = i
      Cells(k, 2) = j
    Next j
  Next i
End Sub
```

図 4.1.12

② 3 重ループ

『 **i ループ** 』、『 **j ループ** 』、『 **k ループ** 』の 3 重ループは図 4.1.13 のようになる。

```
Sub taju3()
  n = 0
  For i = 1 To 5
    For j = 1 To 5
      For k = 1 To 5
        n = n + 1
        Cells(n, 1) = i
        Cells(n, 2) = j
        Cells(n, 3) = k
      Next k
    Next j
  Next i
End Sub
```

図 4.1.13

課題 1.　乱数を使って、3 行× 3 列の行列を表示しなさい！

課題 2.　九九表を作りなさい！

課題 3.　乱数を使って 100 マス計算の問題と解答を作りなさい！

§4.2　条件判断

条件によって場合わけして、別々の処理を行う。条件判断文をマスターして、複雑な条件処理プログラムも分かりやすくつくる。

Check Point	1. 複数の文（ブロック）を理解して、ブロック If の概念をマスターする。 2. 複数条件判断 Select Case を学ぶ。

4.2.1　If文

①　文法

If（もし）、 Then を「そうなら（条件を満たせば)」、 Else を「そうでないなら（条件を満たさなければ)」と解釈する。例えは、条件 A を満たすときは A ブロックを実行して B ブロックは処理しない。一方、条件 B を満たすときは A ブロックを飛ばして B ブロックを処理する。

```
If 条件文 A Then
    A
ElseIf 条件文 B Then
    B
End If
```

```
Sub gusu()
  yu = 0
  For i = 1 To 20
    ten = Int(100 * Rnd)
    Cells(i, 1) = ten
    Cells(i, 2) = ""
    If ten > 80 Then
      yu = yu + 1
      Cells(i, 2) = "優"
    End If
  Next i
  Cells(21, 2) = yu
End Sub
```

図 4.2.1

乱数を使って点数を発生させて点数が８０点より大きいとき、『優』と表示するプログラムは図 4.2.1 のようになる。

漢字を表示するときは、“ ”で漢字を囲むこと。“ ”で囲むことによって文字型変数になる。文字型変数 A に『優』を代入するときは、

A＄＝“ 優 ”

と書き表せる。

②　条件判断を行う演算子（関係演算子）

大きい、小さいなどの大小関係を判定する演算子を『**関係演算子**』という。

演算子	意味
＝	等しい
<>	等しくない
<	小さい
<=	小さいか等しい（以下）
>	大きい
=>	大きいか等しい（以上）

③　条件式の組み合わせ（論理演算子）

数学での『 0 ≦ x ≦ 1 』という条件は、プログラム上で、『 0 <= x And x <= 1 』となります。このように 2 つ以上の条件式を組み合わせるものを『**論理演算子**』と呼びます。

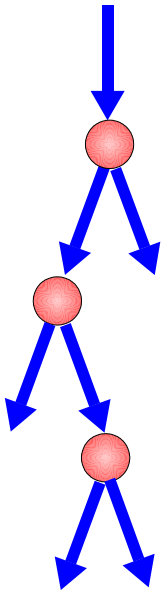
カッコをうまく使って、条件式を組み合わせることもできます。

演算子	意味	優先順位
Not	否定	10
And	かつ	11
Or	または	12

```
If (0 < x And x < 1) Or y > 0 Then
```

If 文の中に If ブロックを入れることもできる（図 4.2.2）が、『 ElseIf 』を用いた多分岐（図 4.2.3）のほうが計算処理速度を上げることができる。

図 4.2.4 のプログラムは『良』の判定を ElseIf 』を用いて行っている。



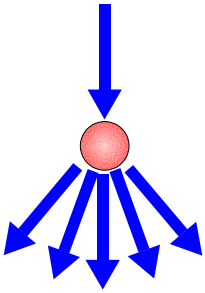


図 4.2.3
ElseIf の概念

図 4.2.2
If 文を別々につなげる場合

```
Sub gusu()
  yu = 0
  ryo = 0
  For i = 1 To 20
    ten = Int(100 * Rnd)
    Cells(i, 1) = ten
    Cells(i, 2) = ""
    Cells(i, 3) = ""
    If ten >= 80 Then
      yu = yu + 1
      Cells(i, 2) = "優"
    ElseIf 70 <= ten And ten < 80 Then
      ryo = ryo + 1
      Cells(i, 3) = "良"
    End If
  Next i
  Cells(21, 2) = yu
  Cells(21, 3) = ryo
  If yu >= 5 Then
    Beep
    MsgBox "優の人数が5人以上です！" & yu
  End If
End Sub
```

図 4.2.4

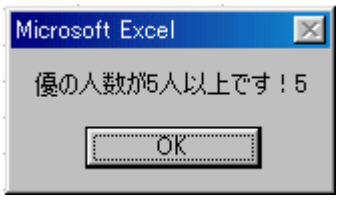


図 4.2.5

『 MsgBox 』メッセージボックスの表示
『 & 』は、文字連結演算子で、2 つの文字や式を連結するときに用います。
『 Beep 』は、ビープ音です。

課題 1.　『優』『良』『可』『不可』の判定をしなさい！

④　偶数判断

２で割った余りが『０』になったとき、その数が偶数であることが分かる。図 4.2.6 は『偶数』の判断をして表示するプログラムである。

```
Sub kisu()
  gu = 0
  ki = 0
  For i = 1 To 20
    ten = Int(100 * Rnd)
    Cells(i, 1) = ten
    Cells(i, 2) = ""
    Cells(i, 3) = ""
    If ten Mod 2 = 0 Then
      gu = gu + 1
      Cells(i, 2) = "偶数"
    Else
      ki = ki + 1
      Cells(i, 3) = "奇数"
    End If
  Next i
  Cells(21, 2) = gu
  Cells(21, 3) = ki
End Sub
```

図 4.2.6

『 Mod 』整数演算の余り
(割り算をしたときの“余り”を計算するコマンド)
『 Mod 』演算子の両脇は必ず１つ以上の空白をとる。

5 Mod 3　　→ 2

⑤　最大値

初期値として、できるだけ小さい値を変数『 ten 』に入れておく（図 4.2.7)。変数『 ten 』が今までの最大値の変数『 Max 』よりも大きい場合、最大値の更新を行う。そのときの最大値のデータ番号を『 nMax 』に代入する。

最後に、最大値の表示をする。

```
Sub saidai()
  Max = -999
  For i = 1 To 20
    ten = Int(100 * Rnd)
    Cells(i, 1) = ten
    Cells(i, 2) = ""
    If Max < ten Then
      Max = ten
      nMax = i
    End If
  Next i
  Cells(nMax, 2) = "最大"
End Sub
```

図 4.2.7

課題 1.　最小値を求めなさい！

課題 2.　2 桁の整数で、3 の倍数を表示しなさい。

変数名について！

大文字・小文字の区別はない。つまり、『 Sum 』『 SUM 』『 sum 』は同じ変数として扱われる。

変数はアルファベットと数字だけ（英数半角）を用いる。変数の先頭の文字は、必ずアルファベットで始める。(漢字や全角のアルファベットを使ってエラーになる場合が多く見られる！)

変数は、『予約語（コマンド、ステートメント、関数)』 (青色で先頭の文字が自動的に大文字になる)以外のものを用いる。また、プロシージャ名と同じ変数を用いてはならない！

4.2.2　Select Case文

前節で説明した成績判定プログラム（If 文）よりも、もっとスマートな複数の条件判断することができる。

① 文法

『 Select Case 』でブロックの始まりと『式（文字列、数値）』を設定する。『 End Select 』でブロックの終わりを指定する。

『 Case 』節の引数（ひきすう）が一致したとき、そのブロック処理を行う。

『 Is 』は、値の範囲を指定する『比較演算子』と共に使われる。

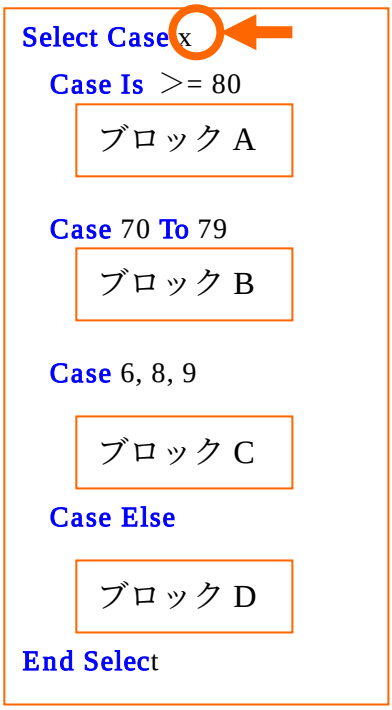


図 4.2.8

```
Sub seiseki()
  yu = 0
  ryo = 0
  ka = 0
  fuka = 0
  Num = 30
  For i = 1 To Num
    ten = Int(100 * Rnd)
    Cells(i, 1) = ten
    Cells(i, 2) = ""
    Cells(i, 3) = ""
    Cells(i, 4) = ""
    Cells(i, 5) = ""
    Select Case ten
      Case Is >= 80
        yu = yu + 1
        Cells(i, 2) = "優"
      Case 70 To 79
        ryo = ryo + 1
        Cells(i, 3) = "良"
      Case 60 To 69
        ka = ka + 1
        Cells(i, 4) = "可"
      Case Else
        fuka = fuka + 1
        Cells(i, 5) = "不可"
    End Select
  Next i
  Cells(Num + 1, 2) = yu
  Cells(Num + 1, 3) = ryo
  Cells(Num + 1, 4) = ka
  Cells(Num + 1, 5) = fuka
End Sub
```

図 4.2.9

複数の式や範囲を示すときは、『 To 』、『 Is 』を組み合わせることができる。

Case 1 To 3, Is ＞ 10

4.2.3　構造化プログラミング

明確な『アルゴリズム』とデータ構造に基づいて問題をトップ・ダウンで機能分解する。『連接(Sequence)』、『判断(if then else)』、『反復(while)』といった明確な制御構造だけを用いて GOTO レスプログラミングをする。

その結果、生産性が高く、信頼性の高いプログラムを作成することができる。『後判定反復（ do loop)』、『複数条件判断（ select case)』、『所定回反復（ for)』なども必要になる。

アルゴリズム:

問題を解くための論理や手順。一般に、ある問題を解くためのアルゴリズムは複数存在する。効率の良いアルゴリズムを考えるのがプログラミングで難しい所である。

§4.3　繰り返し

For 文で回数が決まっている繰り返し処理を行ったが、ここでは回数が決まっていない繰り返しについて説明する。また、ループからの強制脱出を上手く使えるようにする。

Check Point

1. While 文を学ぶ。
2. Do Loop 文の前判定反復、後判定反復をマスターする。

4.3.1　While文

①　文法

繰り返しループの始めに条件を判定するので『**前判定反復**』と呼ばれる。『 **While** 』と『 **Wend** 』で囲まれた文を繰り返す(図 4.3.1)。

```
While 条件文
  繰り返される文
Wend
```

図 4.3.1

②　借金返済計画

図 4.3.2 のプログラムでは、利子と月々の返済金額によって、返済期間が変わる。このように繰り返し回数の決まっていない計算は数多くある。

```
Sub keikaku()
  m = 0                  ' 月
  Y = 1                  ' 年
  gp = 10000             ' 月賦
  r = 12.5               ' 利子（%）
  S = 200000             ' 借金
  goukei = S
  While goukei > hensai
    m = m + 1
    goukei = S * (1 + r / 100) ^ Y
    hensai = gp * m
    Cells(m, 1) = Y
    Cells(m, 2) = goukei
    Cells(m, 3) = goukei - hensai
    If m Mod 12 = 0 Then
      Y = Y + 1
    End If
  Wend
End Sub
```

図 4.3.2

コメント文:

『 ‘（単一引用符） 』の後にコメント（注釈）を書くことができる(**緑色**になる)。コメント文は実行されないのでプログラムの説明に使われる。

プログラムのバグ取りにコメント文が有効に利用できる。

無限ループに注意！

ループの処理で記述を間違えると終了しないループになることがある。例えば、『 **n ＝n+1** 』を書き忘れるとループ条件は『 **n = 1** 』のままで『 **Cells(1,1)<>””** 』となり、永久に終わらない。

このように終わらないループを『 **無限ループ** 』と呼ぶ。無限ループは実行するまで発覚しない。実行前に、無限ループかどうかを確認すること！

なお、無限ループでプログラムが終了しなかった場合、『 **Esc** 』か『 **Ctrl** 』+『 **Break** 』キーを押すと強制終了できる。

4.3.2　Do Loop文

① 文法

『 Do 』ステートメントは、『 Do 』と『 Loop 』の間を繰り返して処理をする（図 4.3.3）。指定した条件が満たされている間『 While 』、または、指定した条件に達するまで『 Until 』の一定の処理を繰り返す。

図 4.3.3

また、『 前判断 』（図 4.3.4）は条件を判断した後に処理を実行するので、条件によっては 1 回も処理を行わない場合がある。一方、『 後判断 』（図 4.3.5）は条件を満たしていなくても最低一回処理を行う。

Do Loop 文は While 文に比べて汎用な繰り返し文である。

図 4.3.4

② 後判定反復 (Until)

i が 10 以上になるまで繰り返す（図 4.3.6）。

図 4.3.5

```
i=0
Do
   i = i + 1
   Cells(i, 1) = i
Loop Until i >= 10
```

図 4.3.6

```
i = 0
Do
   i = i + 1
   Cells(i, 2) = i
Loop While i < 10
```

図 4.3.7

③ 後判定反復 (While)

i が 10 以上になるまで繰り返す（図 4.3.7）。

④ 前判定反復 (Until)

i が 10 以上になるまで繰り返す（図 4.3.8）。

```
i = 0
Do Until i >= 10
   i = i + 1
   Cells(i, 3) = i
Loop
```

図 4.3.8

```
i = 0
Do While i < 10
   i = i + 1
   Cells(i, 4) = i
Loop
```

図 4.3.9

⑤ 前判定反復 (While)

i が 10 以上になるまで繰り返す（図 4.3.9）。

4.3.3　ループからの脱出

①　脱出の使い方

ループからの強制脱出したいとき、『 **Exit** 』を使う。任意の場所におけるメリットがある。図 4.3.10 の場合、乱数を発生させて 0.2 より小さい数が発生したら『 **Do Loop** 』を脱出して終了する。

Exit Do	⇒	Do ループから脱出
Exit For	⇒	For ループから脱出
Exit Sub	⇒	「サブルーチン」で説明
Exit Function	⇒	「サブルーチン」で説明

```
Sub dasyutsu()
  Columns("A:B").Select
  Selection.ClearContents
  i = 0
  Do
    i = i + 1
    x = Rnd
    Cells(i, 1) = i
    Cells(i, 2) = x
    If x < 0.2 Then
      Cells(i, 1) = "脱出"
      Cells(i, 2) = x
      Exit Do
    End If
  Loop
End Sub
```

図 4.3.10

②　脱出の応用

数値計算のときに収束条件によって計算を終了するときに『 **脱出** 』を上手く使う。スターリングの公式の証明するプログラムは図 4.3.11 のようになる。

スターリングの公式

$\log N!$　　$N \log N - N$　$(N \gg 1)$

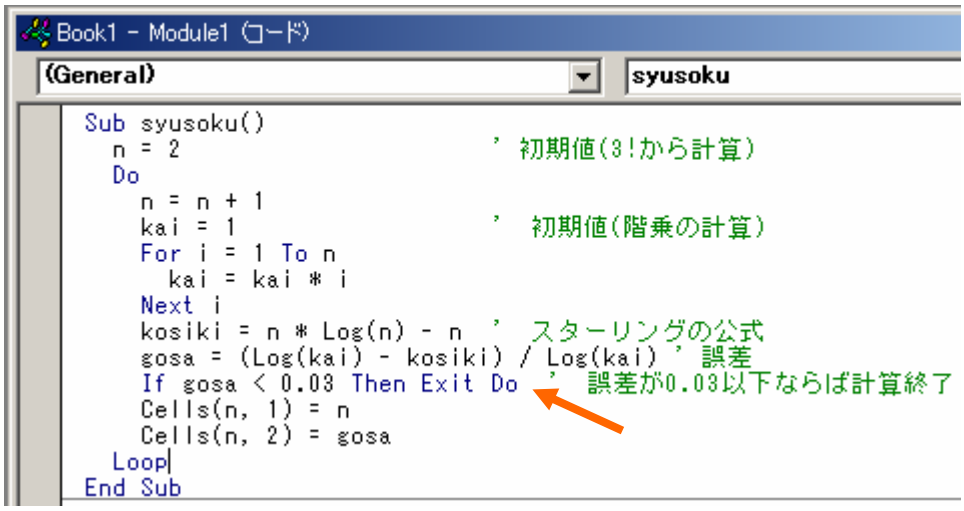


図 4.3.12

課題.1　$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^{n} = e$　の証明をしなさい。

§4.4　配列

配列はデータをまとめて管理するときに威力を発揮する。特に次節に学ぶプロシージャ間のデータの受け渡しに非常に役立つ。

Check Point	1. 配列の長所を理解する。 2. 2 次元、3 次元の配列を利用してプログラムを分かりやすくする。

4.4.1　配列の基本

①　配列の概念

例えば、TOEIC テストの点数などを学科単位でデータをまとめて管理したいときがある。その時、次のようにいちいち単純変数に入れていては大変である。

```
Zairyo01=555
Zairyo02=369
Zairyo03=412
・・・
```

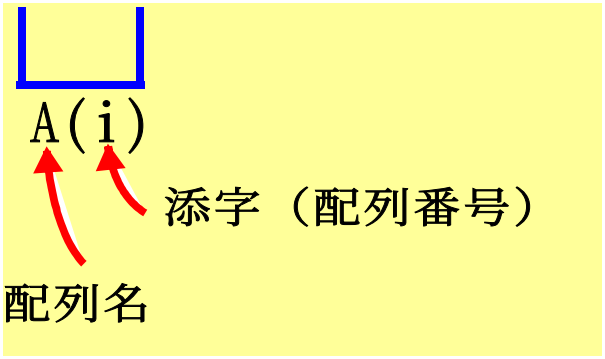


図 4.4.1

そこで配列という概念を導入する。配列とは、『 配列名 』と『 配列番号（添え字）』からなり、一連のデータに背番号をつけるということである（図 4.4.1）。

Zairyo(1)は、1 番目の箱、つまり出席番号 1 の人の点数が入る変数。Zairyo(19) は、出席番号 19 番の人の点数が入ることになる（図 4.4.2）。

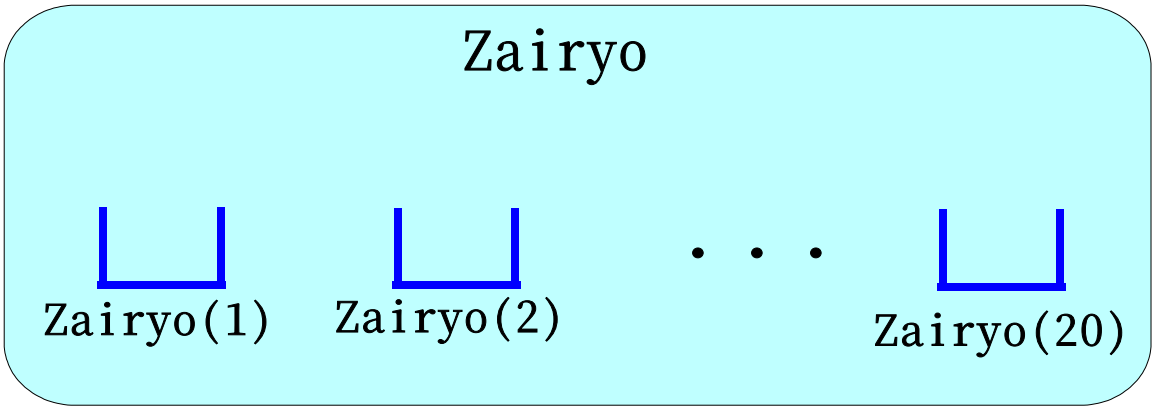


図 4.4.2

② 1 次元の配列

実際のプログラムでは、最初に『 **Dim** 』で箱の大きさ（箱の数）を指定する。図 4.4.3 のプログラムでは、乱数で与えられた点数をまとめてデータの初期化(0 点)をしている。

```
Sub hairetu()
  Dim Zairyo(100)
  n = 32
  For i = 1 To n
    Zairyo(i) = Int(990 * Rnd)
    Cells(i, 1) = Zairyo(i)
  Next i
  For i = 1 To n
    Zairyo(i) = 0
  Next i
End Sub
```

図 4.4.3

③ 2 次元の配列

ここまでは、１次元の配列だったが、次に 2 次元の配列を考える。以下のような 2 次元のデータは、2 次元配列になる。

	1. 国語	2. 社会	3. 数学	4. 理科	5. 英語
1. 中田	56	60	71	63	60
2. 高原	80	64	77	73	75
3. 川口	72	59	90	75	80
4. 柳沢	71	74	82	69	59
5. 平野	60	55	68	84	85
6. 明神	79	53	83	79	66
7. 中沢	88	61	77	53	91

表 1

プログラムでは最初に『 **Dim** ten(20,5) 』のように宣言します。

例えば、 ten(3,2)=59
ten(5,3)=68
ten(7,4)=53
となる。

課題 1. 表 1 の個人の平均と各科目の平均点を計算しなさい。

4.4.2　配列の応用

2 次元の配列は行列と対応する。行列 $A=\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$ の i 行、j 列の成分 a_{ij} は、配列『 a(i, j) 』と等価である。

① 行列の足し算

i 行、j 列の成分であらわすと、$c_{ij} = a_{ij} + b_{ij}$ となるので、行列の足し算をプログラムで書くと、図 4.4.4 のようになる。

『 c(i,j) = a(i,j) + b(i,j) 』

このように、まとめてデータを計算することが可能になり、複雑な計算が非常に簡単になる。

図 4.4.4

Book1 - Sheet1 (コード)

(General)

```
Sub matrix_add()
  Dim a(3, 3), b(3, 3), c(3, 3)
  a(1, 1) = 3:  a(1, 2) = -2: a(1, 3) = 1
  a(2, 1) = -1: a(2, 2) = 5:  a(2, 3) = 0
  a(3, 1) = -2: a(3, 2) = 0:  a(3, 3) = 7
  b(1, 1) = 0:  b(1, 2) = 0:  b(1, 3) = 9
  b(2, 1) = 1:  b(2, 2) = -1: b(2, 3) = 1
  b(3, 1) = 3:  b(3, 2) = -2: b(3, 3) = -5

  For i = 1 To 3
    For j = 1 To 3
      c(i, j) = a(i, j) + b(i, j)
      Cells(i, j) = a(i, j)
      Cells(i + 4, j) = b(i, j)
      Cells(i + 8, j) = c(i, j)
    Next j
  Next i
End Sub
```

② 行列の掛け算

C = AB の掛け算を計算すると、例えば、c_{21} 成分は、

$$\begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

から、$c_{21} = a_{21} b_{11} + a_{22} b_{21} + a_{23} b_{31}$ となる。

i 行，1 列の場合は、$c_{i1} = a_{i1} b_{11} + a_{i2} b_{21} + a_{i3} b_{31}$

i 行，j列の場合は、$c_{ij} = a_{i1} b_{1j} + a_{i2} b_{2j} + a_{i3} b_{3j}$

結局、

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} = \sum_{k=1}^{3} a_{ik}b_{kj}$$

となる。

課題 1.　行列の掛け算のプログラムを作れ！

課題 2.　A^n の計算をしなさい！また、次式の証明をしなさい！

$$A^n = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{pmatrix}^n = \begin{pmatrix} \alpha^n & 0 & 0 \\ 0 & \beta^n & 0 \\ 0 & 0 & \gamma^n \end{pmatrix}$$

§4.5　プロシージャ

プロシージャとは、プログラムの単位機能化（仕事の分担）によって、プログラムをコンパクトにすることである。その結果、プログラムが読みやすくなり、エラーが少なくなったり、バグ取りも簡単になったりする。効率的なプログラミングには必要不可欠である。

Check Point	1. Sub プロシージャと Function プロシージャの違いを理解する。 2. プロシージャ間のデータのやり取りを把握する。

4.5.1　Subプロシージャの定義と呼び出し

プログラム言語の文法に従って記述される命令文をステートメントと言う。また、『ステートメント』の集まりを『コード』と呼び、1 つ以上の機能を持ったもののまとまりが『プロシージャ』となる。

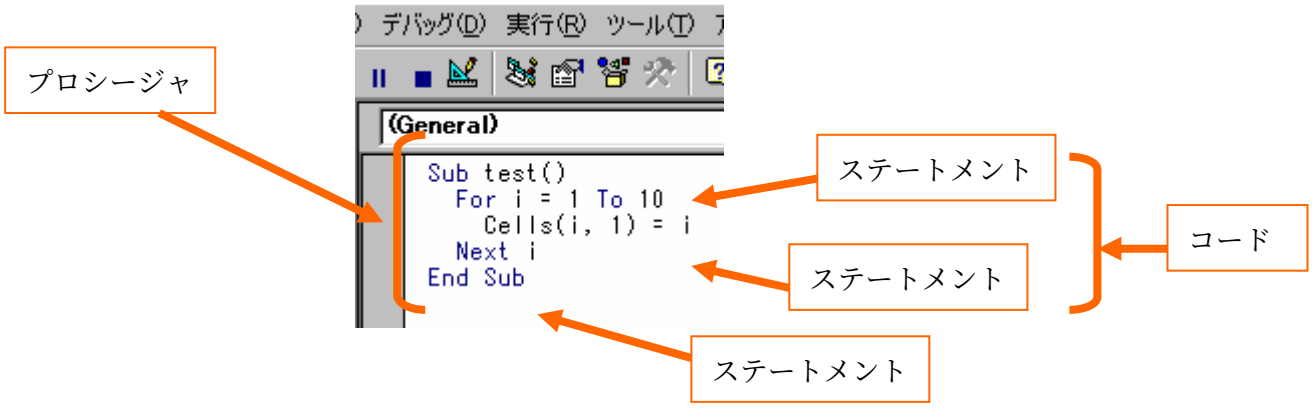


図 4.5.1　Sub プロシージャ

図 4.5.1 のように Sub プロシージャは『 **Sub** 』ステートメントと『 **End　Sub** 』ステートメントで定義される。

Sub プロシージャの呼び出しは図 4.5.2 のように

『**Call　プロシージャ名**』

で行われます。

```
Sub test()
  n = 10
  Call display(n)
End Sub
Sub display(n)
  For i = 1 To n
    Cells(i, 1) = i
  Next i
End Sub
```

図 4.5.2

人間が同時に扱う「複雑さ」には限界があるので、自分の能力を超えたとき、バグの発生率は飛躍的に上昇する。効率良く高品質のプログラムを書くために、プログラムを機能ごとに分類してプログラム内容をブラックボックスにします。

複雑な電子回路では、機能ごとにモジュール化してモジュールの組み合わせで製品化する。多様な電子回路での資産化・共有化が進み、また、故障した場合、モジュールごと交換して、復旧が早くなるメリットがある。

つまり、プロシージャを利用することは「複雑さを管理する」ことである。

昔の BASIC では、プロシージャ間の変数の独自性が保てなかったため、知らないうちに変数が置き換わっていることが多々あった。例えば、図 4.5.3 の計算で、『test プロシージャ』で使っている変数 I が『test1 プロシージャ』で同じ変数を使っているため、変数『 I 』の書き換えが起きる。

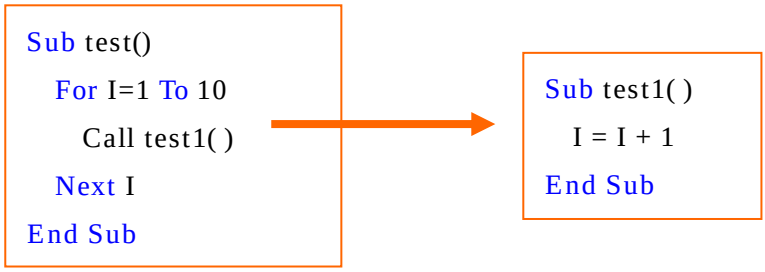


図 4.5.3

そのため、『プロシージャの独立』と、同じ変数でも混同しない『**引数**（ひきすう）』を導入する。

① 変数の独立

プロシージャ内の変数を『**ローカル変数**』として、他のプロシージャと同じ変数を使っても影響を受けない。

つまり、厚い壁の中（プロシージャ）にある変数は、図 4.5.4 のように外に出ることができない！（他のプロシージャの変数を書き換えない）

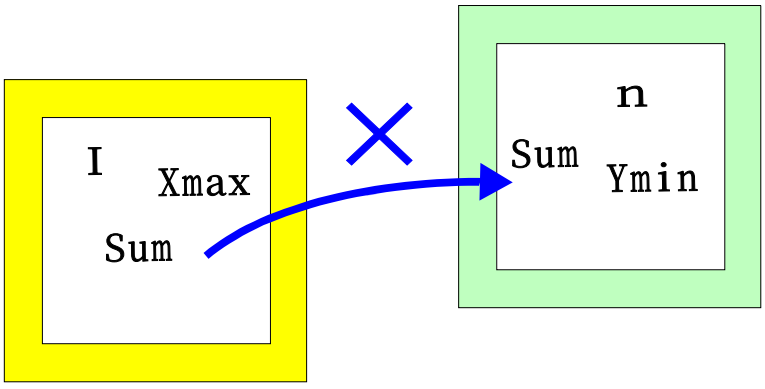


図 4.5.4

② データのやり取り

プロシージャ間で独立性を保ったまま、『引数（ひきすう）』を介してデータを授受する。

厚い壁に穴をあけて、引数どうしでデータのやり取りをする（図 4.5.5）。限定したデータのモニター・監視することによって不正処理を防ぐ。

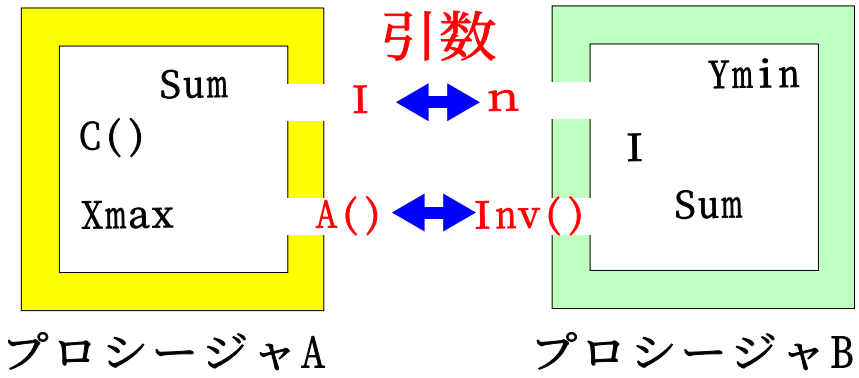


図 4.5.5

例えば、『プロシージャ A』の『 引数 I 』のデータは、『プロシージャ B』の『 引数 n 』に引き渡されるので、『プロシージャ B』の『 変数 I 』と干渉しない。

注意

- 引数の型（整数型、文字型など）が合っていれば変数名が異なってもよい。
- 配列の引数は配列どうしでなければならない。
- 配列の次元も同じになる必要がある。

③ 例題

指定した文字をｎ個のセルに書き、場所も指定できる（図 4.5.6）。

```
Sub test2()
  a$ = "abe"
  Call disp(a$, 10, 1)
  Call disp("great !", 20, 2)
End Sub
Sub disp(moji$, n, posi)
  For i = 1 To n
    Cells(i, posi) = moji$
  Next i
End Sub
```

図 4.5.6

回転を表す一次変換の計算をする（図 4.5.7）。

$$R(\theta)=\begin{pmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{pmatrix}$$

円周率は、頻繁に使う定数なので、『ローカル変数』よりも、引数を介さなくてもよい『グローバル変数』としたほうが便利である。

```
Dim pi As Double
Sub kaiten()
  Dim p(2), q(2)
  pi = 4# * Atn(1#)          'π  円周率
  the = 30                   'θ  [°]
  p(1) = 1: p(2) = 0         ' q = R p
  Call rotation(the, p(), q())
  Cells(1, 1) = p(1)
  Cells(1, 2) = p(2)
  Cells(3, 1) = q(1)
  Cells(3, 2) = q(2)
End Sub
Sub rotation(x, a(), b())
  rad = x * pi / 180#        ' [°] -> [rad]
  b(1) = a(1) * Cos(rad) - a(2) * Sin(rad)
  b(2) = a(1) * Sin(rad) + a(2) * Cos(rad)
End Sub
```

図 4.5.7

④　グローバル変数の設定

Sub プロシージャの前に、

```
Dim pi As Double
```

と指定する（図 4.5.7）。Double は『**倍精度実数型**』を意味します。グローバル変数になるので、引数を介さなくても、すべてのプロシージャで『 pi = 3.14159・・・ 』となる。

Excel には『 pi() 』の関数があったが、VBA にはない。そこで、

$\tan(\pi/4) = 1$　　$\text{atan}(1) = \pi/4$（ラジアン）

の関係を利用して、

$\pi = 4 \times \text{atan}(1)$　(ラジアン)

と計算できる。

実際の計算では、the の値が Sub プロシージャの x に引き渡され、

配列 p(1), p(2) のそれぞれの値は、配列全体として p( ) となり、Sub プロシージャの a( ) に引き渡される（図 4.5.8）。

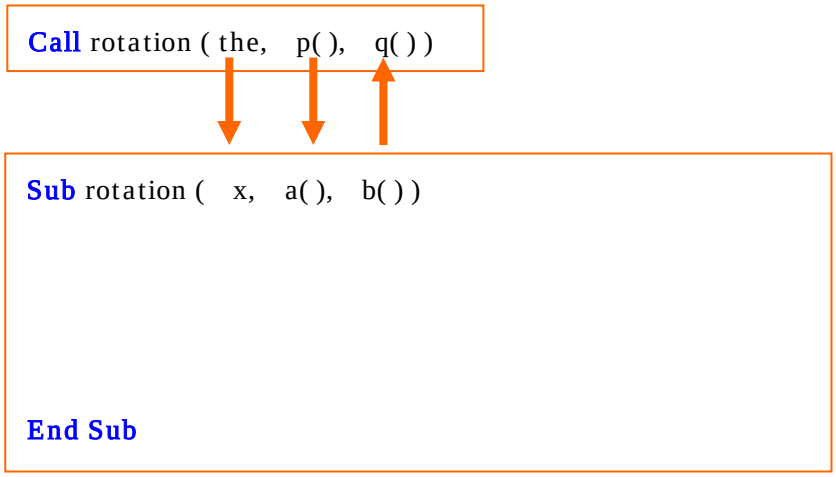


図 4.5.8

プロシージャの計算結果は、b(1), b(2) に代入されて、引数 q( ) の配列全体に戻される。

課題 1.　y = sin θ　(0≦θ≦π) のグラフを 30 ° 回転させて、グラフに書きなさい！

課題 2.　プロシージャをうまく使って $A^n = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{pmatrix}^n = \begin{pmatrix} \alpha^n & 0 & 0 \\ 0 & \beta^n & 0 \\ 0 & 0 & \gamma^n \end{pmatrix}$ の証明するプログラムを作り直しなさい！

4.5.2　Functionプロシージャの定義と呼び出し

Sub プロシージャと異なり『 **Function プロシージャ** 』は**戻り値**を持つ。

① 戻り値

例えば、**f(x) = sin(x)** の関数は、『 **x ＝ π / 2** 』の値に対して、**sin(π/2)** に『 **1** 』の値を返す（図 4.5.9）。

このことを

関数 **sin(x)** は**戻り値**を持つ

という。

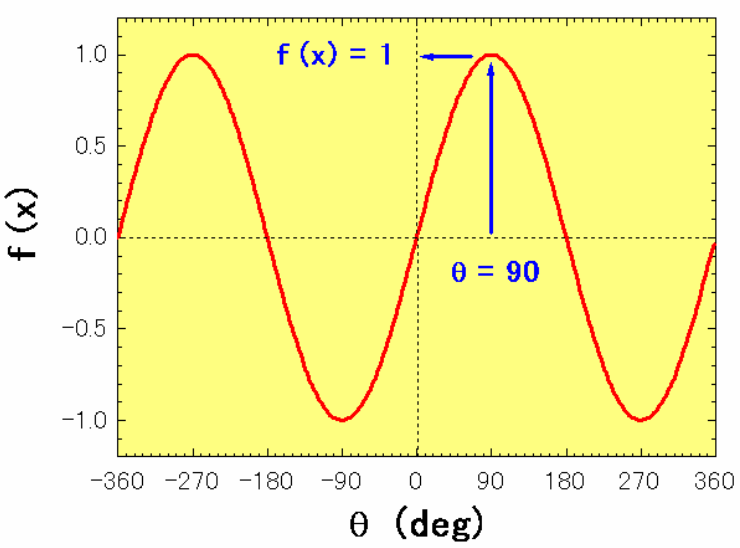


図 4.5.9

② 定義

Excel の表計算ではアークサインの関数はあるが、Basic にはない。そこで、アークタンジェントを使って、アークサインを計算しないといけない。

この場合、『 **Function プロシージャ** 』を用いると簡単になる。

図 4.5.10、**asin(z)** という新しい関数をユーザーが定義するプログラムである。『**戻り値**』の設定と戻り値の流れは図 4.5.11 のようになる。

```
Dim pi As Double
Sub test()
  pi = 4# * Atn(1#)  ' 円周率 π
  z = Sqr(3#) / 2#
  the = asin(z)
  Cells(1, 1) = z
  Cells(1, 2) = the
End Sub
Function asin(w)
  If 0 <= w And w < 1 Then
    z = w * w / (1 - w * w)
    asin = Atn(Sqr(z)) * 180# / pi#
  ElseIf -1 < w And w < 0 Then
    z = w * w / (1 - w * w)
    asin = -Atn(Sqr(z)) * 180# / pi#
  ElseIf w = 1 Then
    asin = 90
  ElseIf w = -1 Then
    asin = -90
  End If
End Function
```

図 4.5.10

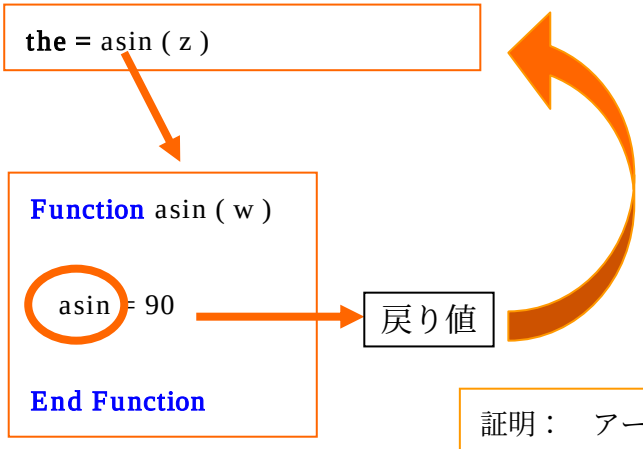


図 4.5.11

証明：　アークタンジェントを用いたアークサイン

$$\theta = \arctan\left\{\sqrt{\frac{y^2}{1-y^2}}\right\}$$

$$y = \sin\theta \quad \Rightarrow \quad \theta = \sin^{-1}(y) = \arcsin(y)$$

$$\tan\theta = \frac{\sin\theta}{\cos\theta} = \frac{\sin\theta}{\sqrt{1-\sin^2\theta}} = \frac{y}{\sqrt{1-y^2}} = \sqrt{\frac{y^2}{1-y^2}}$$

③　例題 1

乱数を使って８０点以上が出るまで、表示しなさい。

```
Sub test()
  Columns("A:A").Select
  Selection.ClearContents
  n = 0
  While seiseki() < 80
    n = n + 1
    Cells(n, 1) = seiseki()
  Wend
End Sub
Function seiseki()
  seiseki = Int(Rnd * 100)
End Function
```

Function サブルーチンは、戻り値を持つので
条件文に条件文に直接使うことができる。

④　例題 2

円の面積を数値積分で求めなさい。

長方形で近似する場合は、

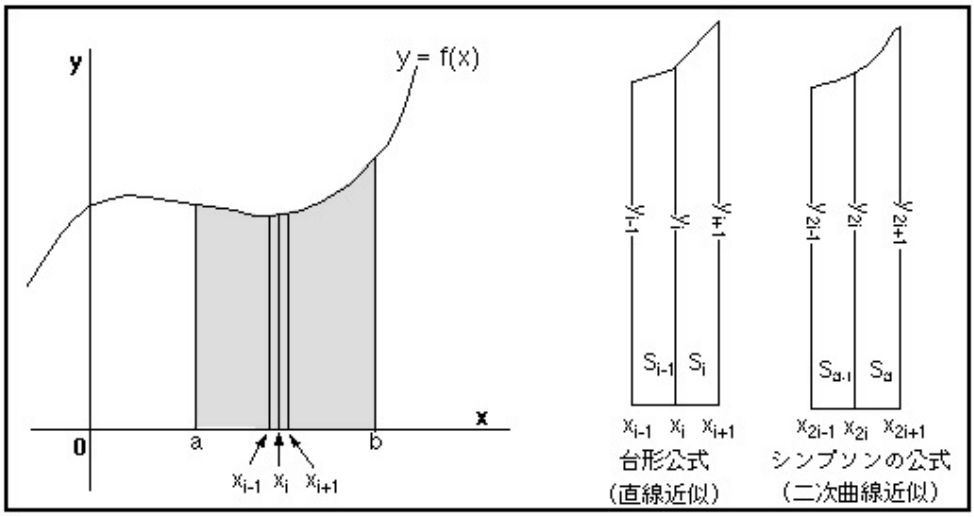
$$dS = dx \times \sqrt{r^2 - x^2}$$

$$S = \int_0^r \sqrt{r^2 - x^2}\,dx$$

```
Sub menseki()
  n = 1            ' きざみ数
  r = 1            ' 半径
  For k = 1 To 15
    n = n * 2
    dx = r / n
    S = 0          ' 面積
    For i = 1 To n
      x = dx * i
      y = en(x, r)
      S = S + y * dx
    Next i
    Cells(k, 1) = n
    Cells(k, 2) = S * 4
  Next k
End Sub
Function en(x, r)
  en = Sqr(r ^ 2 - x ^ 2)
End Function
```

課題 1.　台形公式を用いて、円の面積を求めなさい！

課題 2.　シンプソンの公式を用いて円の面積を求めなさい！



§4.6　ファイル処理

計算結果を他のアプリケーションソフトで見たり、計算結果をメールの添付ファイルで送ったりしてデータの共有を行いたい場合がある。ファイル処理を学んでデータの読みこみ・書きこみをする。

Check Point	1. 複数のファイルの読み込み・書き込みを学ぶ。 2. データ共有化によるメリットを考える。

4.6.1　データの書き込み

① 文法

『 Open 』でファイルを開き、『 Close 』でファイルを閉じる。ファイルの書き込みでは『モード』が『 Output 』となる。

一度に数多くのファイルが開けるので、『ファイル名』と『ファイル番号』を対応付ける必要がある。また、『ファイル名』は『文字型変数』となる。ファイル名の中にデータを入れるステートメントは

```
Open ファイル名 For モード As ファイル番号
  Print ファイル番号, “test!”
Close ファイル番号
```

図 4.6.1

『 Print #(ファイル番号), 変数 1, 変数 2 , 変数 3 』

となる。変数の数に応じてコンマで区切って横に並べる。

② プログラム例

実際にプログラムを作る（図 4.6.2）。保存するドライブが省略されているので『 test.dat 』は

『マイ ドキュメント』

に保存されている。

```
Sub File1()
  Open "test.dat" For Output As #1
    For i = 1 To 20
      Print #1, i, Rnd
    Next i
  Close #1
End Sub
```

図 4.6.2

『 Z:¥ 』はZドライブという意味である。Cドライブの『 Program Files 』フォルダの中の『 Dell 』フォルダに保存したい場合は、

"C:¥Program Files¥Dell¥test.dat"

となる。『 ¥ 』でフォルダ名を区切る。

4.6.2　データの読み込み

①　文法

　書き込みとの違いは、ファイルの読み込みでは『モード』が『 Input 』となる。また、ファイル名の中のデータを読み込むステートメントは

『 Input #(ファイル番号), 変数 1, 変数 2 , 変数 3 』

となる（図 4.6.3）。

```
Sub File2()
  Open "test.dat" For Input As #2
    For i = 1 To 20
      Input #2, x, y
      Cells(i, 1) = x
      Cells(i, 2) = y
    Next i
  Close #2
End Sub
```

図 4.6.3

②　プログラム例

　前節で保存されたファイルを読み込み、そのデータをセルに書き出す（図 4.6.3）。次にデータの個数がわからないファイルの場合は、繰り返しの回数が決まっていない場合に使った『 While 』 ～ 『 Wend 』を利用する。

　『 EOF (End of File) 』は、ファイルの終わりで『 真 (True) 』を返す。『 EOF (n) 』は、『 ファイル番号 n 』の終わりを探すということである。

```
Sub File3()
  Open "test.dat" For Input As #3
    i = 0
    While Not EOF(3)
      i = i + 1
      Line Input #3, dummy$
      Cells(i, 1) = dummy$
    Wend
  Close #3
  num = i
  Cells(num + 1, 1) = num
End Sub
```

図 4.6.4

　一般に変数がいくつ並んでいるか分からないので、データを一行ずつ読み込む。そのためには、一行を『文字型変数』として『 Line Input #(ファイル番号) 』を使う。

　『 Input #(ファイル番号), 変数 1, 変数 2 , 変数 3 』

の部分が、

　『 Line Input #(ファイル番号), dummy$ 』

となる（図 4.6.4）。

③　複数のファイル

　ファイルからデータを読み込み、計算して、解析データを違うファイルに保存したい場合がある（図 4.6.5）。

　同時に複数のファイルを取り扱う場合、必ず、『ファイル番号』を変えないといけない。

```
Sub File5()
  Open "test.dat" For Input As #1
    Open "test2.dat" For Output As #2
      For i = 1 To 20
        Input #1, x, y
        Cells(i, 1) = x
        Cells(i, 2) = y
        z = 2 * y
        Print #2, x, y, z
      Next i
    Close #2
  Close #1
End Sub
```

図 4.6.5

ファイル名を変えて保存すること。
同じファイル名の場合は上書きするので注意すること！

ファイル形式には『 シーケンシャル 』、『 ランダム 』、『 バイナリ 』がある。ここでは、テキストファイルに読み書きする『 シーケンシャル 』について述べた。

ランダムアクセスは、ファイルを開いたままデータを読み書きができ、データアクセスが迅速に、しかも簡単に行える。

バイナリアクセスは、ファイル内の任意のバイト位置に読み書きができるので画像ファイルなどに使われる。