

Visual Basic 基礎講座

CONTENTS

Visual Basic 概要	1
Visual Studio.....	1
コーディングの基本	3
コードの内容.....	4
表示の改行.....	5
コメント.....	6
変数	7
変数とは.....	7
変数宣言の構文.....	8
変数名.....	9
値の代入.....	10
変数の初期化.....	10
変数の出力.....	11
変数への入力.....	12
四則演算子	14
条件分岐	17
条件の真偽.....	17
If 文.....	19
Else 文.....	21
ElseIf 文.....	22
複合条件.....	25
Select 文（多方向分岐）.....	29
繰り返し	32
Do-Loop 文.....	32
複合代入演算子.....	35
For 文.....	36

配列	41
For Each 文	44
関数	46
関数の種類.....	46
関数の書式.....	47
自作メソッド（自作関数）	50
練習問題解答例	59
課題 1（コーディングの基本）	71
フォームアプリケーション.....	71
プロジェクトの作成.....	73
イベント.....	76
コントロールのプロパティを設定する	78
課題 2（変数）	80
Button.....	80
Label.....	80
TextBox.....	80
課題 3（四則演算）	83
課題 4（条件分岐）	87
Checkbox.....	87
RadioButton.....	90
課題 5（繰り返し）	93
ComboBox.....	93
課題 6（配列）	97
ListBox.....	97
課題 7（関数）	101
自作プロシージャ.....	101
課題解答例	111
最終課題	117
模範解答例.....	121
模範解答例（追加要件対応）	122
実行環境【Visual Studio】	125

プロジェクトを作成する.....	125
ソースファイルを追加する.....	128
プロジェクトの追加.....	130
デバッグ手法.....	133
デバッグの開始とデバッグの停止.....	133
ブレークポイント.....	134
ステップ実行.....	136
エディットコンティニュー.....	137
データヒント.....	138
ローカルウィンドウ.....	139
自動変数ウィンドウ.....	140
トレースポイント・条件付きブレークポイント.....	141
呼び出し履歴.....	144

Visual Basic 概要

Visual Basic は「ビジュアル・ベーシック」と読み、一般的には「VB(ブイビー)」と省略されて呼ばれます。

Visual Basic はもともと 1970 年代から 1980 年代の「Microsoft BASIC」、1990 年代の「Microsoft Visual Basic」の後継で、2000 年代からは、.NET Framework に対応した「Microsoft Visual Basic .NET」として広まりました。現在は.NET をつけずに Visual Basic と呼ばれていますが、.NET の機能は使えますし、VB.NET の後継ですので、別の言語ではありません。

Visual Basic は、さまざまな言語と比べると、比較的新しい言語です。C 言語や C++ 言語をベースに、Java 言語の良いところを取り入れたような言語となっています。

Visual Studio

VB を使って、Windows 上で動かすことができるソフトウェアを簡単に開発できます。Microsoft は VB や他の開発言語を含む開発環境のソフトウェアを Visual Studio という製品名で提供しています。Visual Studio には次の表のような種類があります。

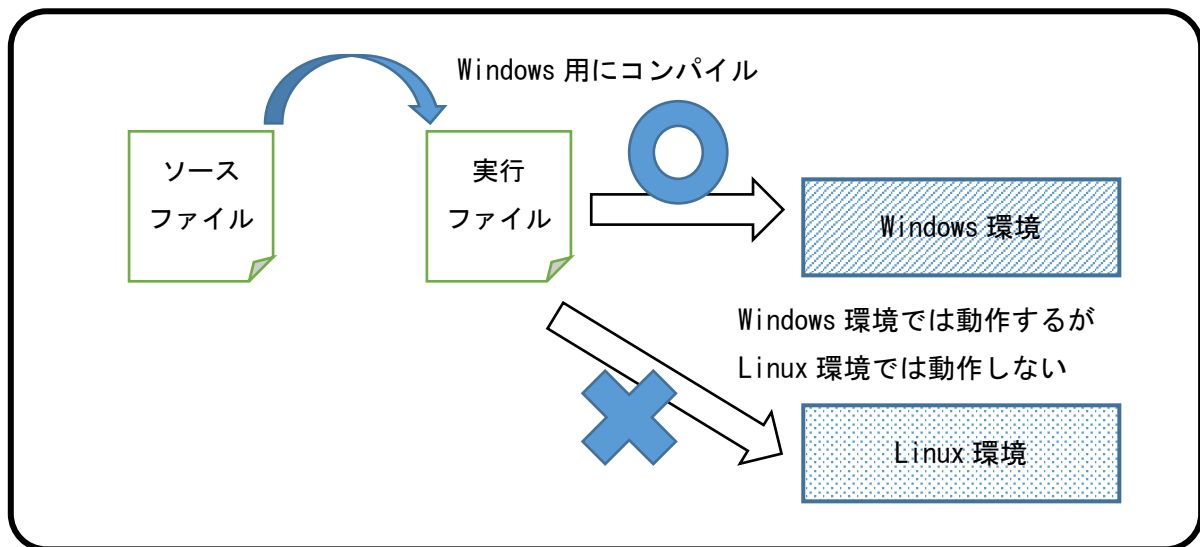
種類	詳細
Express	評価・学習用の無償版
Professional	個人開発向け
Premium	企業開発向け
Ultimate	全機能を搭載
Test Professional	テスター向け
Community	Professional の制限付き

実行スタイルの違い

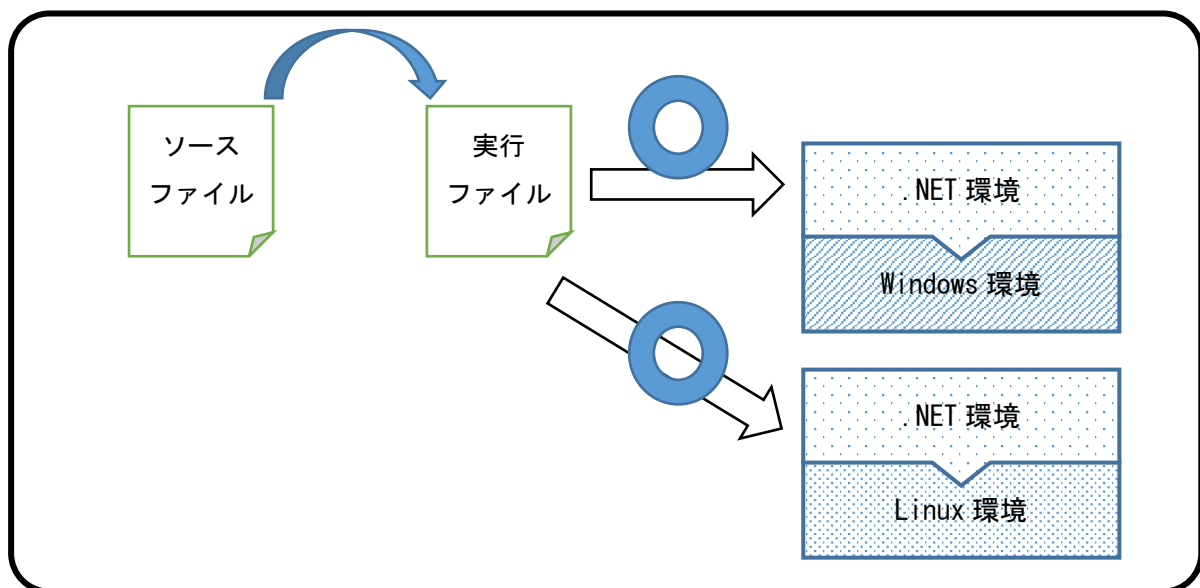
VB と他の言語との実行スタイルの違いを、C 言語を例に比較してみます。

VB も C 言語もプログラムを実行するにはコンパイルという作業が必要です。コンパイルとは、プログラムが記述されたファイル（ソースファイル）の中のプログラム（ソースコード）をコンピュータが直接実行可能な形式へ変換することです。コンパイルすることで、実行ファイルが生成されます。

C 言語は基本的に、実行環境に合わせてコンパイルをする必要があります。Windows 用にコンパイルしたものは Windows 上でしか動作せず、Linux 用にコンパイルしたものは Linux 上でしか動作しませんでした。



VB は .NET Framework が動く環境であれば、作成したアプリケーションを動作させることができます。



コーディングの基本

プログラムを記述していくことを「プログラミング」と言います。また、プログラムの一つ一つの命令をコードと呼ぶため、「コーディング」と言うこともあります。コードの集合がプログラムになりますので、意味合いは同じです。

コーディングの際の注意点は以下の通りです。

英数文字は全角ではなく半角を使います。(基本的に半角英数字を使って記述します)

(例) `main ← ○` `main ← x`

空白は「半角スペース」または「Tab」で区切ります。全角スペースは半角スペースに自動的に変換されるか、削除されます。

大括弧 []、中括弧 { }、小括弧 ()、ダブルクォーテーション " "、シングルクォーテーション ' ' は基本的に開始と終了で対になっています。また、セミicolon ;、コロロン : などの似たような記号があり、それぞれ用途が異なりますので、違いに注意してください。それぞれの用途は順次学習して行きます。

それでは最初に実行するソースコードを確認します。

```
1.  Module Module1
2.
3.      Sub Main()
4.
5.          Console.WriteLine("Hello World!")
6.
7.      End Sub
8.
9.  End Module
```

「1.」や「2.」は目安となる行数です。実際には入力しないでください。

ファイルの作成方法や実行の方法は巻末の参考資料を参照してください

【実行結果】

Hello World!

コードの内容

Visual Basic でコーディングをするとき、どのような機能を持ったプログラムを作成するにも、下記の 1 行目と 9 行目、3 行目と 7 行目はほぼ必須です。

【Sample01】

```
1. Module Module1
2.
3.     Sub Main()
4.
5.         Console.WriteLine("Hello World!")
6.
7.     End Sub
8.
9. End Module
```

3～7 行目は Main という名前のサブプロシージャと呼ばれるもので、Visual Basic でコンソールアプリケーションを実行したときに、最初に動作するものです。Sub～End Sub まだがサブプロシージャで、ひとつの命令の塊です。大きなプログラムになると、このプロシージャという命令の塊がいくつも必要になります。今回は、Main という名前の Sub プロシージャ（メインプログラムの内容が書かれている塊）が一つあります。

そのプロシージャで実行しているのが 5 行目の Console クラスの Write() メソッドです。「クラス」や「メソッド」については応用編で詳しく取り上げます。

Write() メソッドは () 括弧内に記述された内容をコマンドプロンプトに表示させる命令です。文字を表示させたい場合は " " ダブルクォーテーションで括ります。

最初の行と最後の行（Module の宣言）の記述は基本的に変わりませんので、今後のサンプルでは省略します。

表示の改行

先ほどのサンプルを少し変更してみましょう。

```
3.      Sub Main()  
4.  
5.          Console.WriteLine("Hello World!")  
6.          Console.WriteLine("Hello World!")  
7.  
8.      End Sub
```

【実行結果】

```
Hello World!Hello World!
```

プログラムを2行にしたのに、1行で表示されました。これは、`Console.WriteLine()`メソッドが、改行させる命令を含んでいないからです。1行ごとに改行しながら表示させるには、`Console.WriteLine()`メソッドを利用します。

先ほどのサンプルをさらに変更してみましょう。

```
3.      Sub Main()  
4.  
5.          Console.WriteLine("Hello World!")  
6.          Console.WriteLine("Hello World!")  
7.  
8.          Console.WriteLine("Hello World!")  
9.  
10.     End Sub
```

【実行結果】

```
Hello World!  
Hello World!  
Hello World!
```

ソースコードの7行目が改行されていますが、実行結果は改行されていません。ソースコード上のスペースや改行は実行結果に影響しませんので、人間が見やすいように適度に改行を含めます。

コメント

まずは下記のサンプルコードを確認してください。先ほどのサンプルに一部分を追記しました。

```
3.      Sub Main()  
4.          '挨拶を表示する  
5.          Console.WriteLine("おはよう")      'おはよう と表示する  
6.          Console.WriteLine("こんにちは")    'こんにちは と表示する  
7.          ' Console.WriteLine("こんばんは")  
8.  
9.      End Sub
```

この追記部分はコメントと呼ばれ、実際にプログラムが実行される際には無視される部分です。' シングルクォーテーションより右はプログラムに影響を与えない部分となります。プログラムの一部を無効化したり、補足や説明を自由に記述したりできます。

プログラムというものは自分が書いたものでさえ、日が経ってから見ると他人のプログラムと思えるほど、そのときの意図や流れがわからなくなるものです。ましてや他人が見た場合、解読はさらに困難になります。

そのために、何のための処理なのか、こういった流れなのかを要所要所にコメントを入れておくことが、わかりやすいコードを書くためには大切なことです。

変数

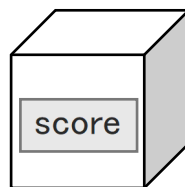
変数とは

コンピュータでは、様々な演算をする上で、必要なデータを一時的に保存しておく必要があります。どこに保存するかというと、パソコンのメモリです。メモリの管理は基本的にOSが行うので、メモリの中のどこに保存されるかは、その時のパソコンの状況によって変わる可能性があります。

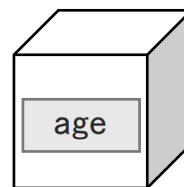
プログラムでは、実行の度にデータが保存される場所が変わっても大丈夫のように、目印となる名前を付けられます。その名前が付けられた、データを格納するためのメモリエリアを変数と言います。

変数を利用することで、実行の度に变化するデータの保存場所に左右されず、プログラムの命令でデータを格納したり取得したりできます。一度格納したデータを更新することもできます。

変数は下図のような「箱」に例えられることが多いです。箱にはラベルを付けておくことで、区別したり、中に入れたものを探したりしやすくなります。



変数



変数

変数宣言の構文

具体的に、プログラムで変数を作る方法について確認します。変数を用意する命令を変数宣言と言い、変数の名前と、格納するデータの型を記述します。変数の宣言には名前だけでなく、その変数の中にどのようなデータを格納するのかをあらかじめ決めておく必要があります。

```
Dim 変数名 As 格納するデータ型
```

実際の記述例は次のとおりです。

```
Dim score As Integer
```

上記を日本語にするならば、「Integer 型のデータを格納するための score という名前の変数を用意しなさい」という命令になります。OS の都合によってメモリ上のどこに変数を用意されるかは分かりませんが、この一文でその場所を score という名前で扱えるようにできました。そして、その変数 score で扱えるものは Integer 型だけに制限をかけています。Integer 型以外にも、データ型には次のようなものがあります。（下記以外にもデータ型に指定可能な型はありますが、基本的なものだけを取り上げています）

データ型名	データサイズ(バイト)	扱うデータ
Integer	4	整数 (-2147483648～2147483647)
Long	8	整数 (およそ 10^{-18} ～ 10^{18})
Double	8	小数 (およそ 10^{-308} ～ 10^{308})
String	可変長	文字列

変数を用意するにあたって、型を指定することで、扱えるデータの形式を制限してしまうことはデメリットのように感じられます。しかし、扱えるデータの形式をあらかじめ決めておくことで、間違ったデータの扱いを防ぐことができます。

たとえば、前述の「Dim score As Integer」の場合、何かの点数をプログラムで使うために、整数用の変数を用意しましたが、その変数で「試験科目名」などの整数ではないデータを扱おうとした場合、明らかに間違った使い方であると判断され、エラーになってくれます。エラーにならずにプログラムが進行した場合、思い通りの結果が得られないかもしれません。

変数名

変数名は基本的に任意で名付ける事ができますが、いくつかのルールがあります。

- 全角は使用できるが、基本的に使用しない
- 大文字・小文字は区別されない
- 先頭の文字は英文字あるいはアンダースコア
- 先頭以外に使える文字は英文字、アンダースコア、数字
- 予約語は使用できない

予約語とは、プログラム言語の中で、役割があらかじめ決められている単語です。下記のほとんどの単語は、今後テキストを読み進めていく中で、それぞれの役割について学習して行きます。使い方を学習すれば自然と変数名として使うことは避けるようになりますので、特に覚える必要はありません。

また、知らずに変数名にしてしまったとしてもエラーになりますので、エラーになったら修正すればよいという程度に思ってください。

予約語（一部）

And	As	Boolean	ByRef	Byte	ByVal
Call	Case	Catch	CBool	CByte	CChar
CDate	CDBl	CDec	Char	CInt	CLng
CObj	Const	Continue	CShort	CSng	CStr
Date	Decimal	Declare	Delegate	Dim	Do
Double	Each	Else	ElseIf	EndIf	Enum
FALSE	Finally	Function	Get	GoTo	Handles
If	Implements	In	Inherits	Integer	Interface
Is	Lib	Like	Long	Loop	Me
Mod	MustInherit	MustOverride	Namespace	Not	Nothing
Optional	Or	On	Overridable	Overrides	ParamArray
Private	Property	Protected	Public	ReDim	Resume
Return	Set	Short	Single	Static	Step
String	Sub	Then	Throw	TRUE	Try
Using	Variant	When	While	With	Xor

値の代入

変数にデータを格納することを代入と呼びます。

```
Dim score As Integer  
score = 90
```

上記の 1 行目は変数宣言で、2 行目が値の代入の命令です。2 行目を日本語にするならば、「変数 score の中に 90 を入れなさい」という命令になります。

Visual Basic での = の記号は、数学のような「右辺と左辺は等しい」という意味ではありません。「右辺の値を左辺に代入する」という意味になります。

よって、右辺には値（あるいは値を求める計算式）を、左辺はデータを受け取るための変数を記述します。

変数の初期化

変数の宣言と値の代入を 1 行にまとめることができます。最初に変数へ値を入れることを初期化と呼びます。

```
Dim score As Integer = 90
```

これを日本語にするならば、「Integer 型のデータを格納するための score という名前の変数を用意し、その中に 90 を入れなさい」という命令になります。

変数の出力

画面に文字を表示させる Console.WriteLine() メソッドの括弧の中に変数を記述すると、変数に格納されている値が表示されます。文字列と変数を連結させる場合は & 記号が必要です。

【Sample02】

新たなプロジェクトを追加しましょう

```

3.      Sub Main()
4.
5.          Dim month As Integer = 7
6.          Dim age As Integer = 28
7.          Dim height As Double = 173.4
8.          Dim blood As String = "AB"
9.
10.         Console.WriteLine("年齢は" & age & "歳です")
11.         Console.WriteLine(month & "月生まれです")
12.         Console.WriteLine("身長は" & height & "cm です")
13.         Console.WriteLine("血液型は" & blood & "型です")
14.
15.     End Sub

```

【実行結果】

```

年齢は 28 歳です
7 月生まれです
身長は 173.4cm です
血液型は AB 型です

```

文字列の結合

変数内のデータを含む文字列を生成するときは下記のように & 記号を利用することで結合できます。結合後の文字列は直接表示させることも可能です、変数への格納も可能です。

【Sample03】

新たなプロジェクトを追加しましょう

```

3.      Sub Main()
4.
5.          Dim month As Integer = 7
6.          Dim age As Integer = 28
7.          Dim str As String
8.
9.          str = month & "月で" & age & "歳になりました"
10.
11.         Console.WriteLine( str )
12.
13.     End Sub

```

【実行結果】

```

7 月で 28 歳になりました

```

変数への入力

プログラムの実行途中にデータの入力を促し、ユーザーが入力した値を変数に格納し、プログラム内で扱うことができます。

【Sample04】

新たなプロジェクトを追加しましょう

```

3.      Sub Main()
4.
5.          Dim name As String
6.
7.          Console.Write("お名前は？")
8.          name = Console.ReadLine()
9.
10.         Console.WriteLine(name & "さんですね")
11.
12.     End Sub

```

【実行結果】

お名前は？

実行すると最後まで処理されずに、カーソルが点滅して入力待ちの状態になりました。名前を入力して Enter キーを押してください。

お名前は？ 山田
山田さんですね

入力された値を表示して処理が終了しました。

```
Console.Write("お名前は？")
```

上記の部分は名前の入力を促す記述をしています。このように、ユーザーに入力を促す文章をプロンプトと呼びます。プロンプトは記述しなくても処理上は何の問題もありますが、ユーザーのために可能な限り記述すべきです。

```
name = Console.ReadLine()
```

上記の右辺の効果でコンソールが入力待ちの状態になります。また、コンソールに入力された内容を左辺へ代入します。

『Console.ReadLine()』の結果は String 型として発生するので、String 型の変数へ格納しています。もし 123 と入力しても、3 文字の文字列となります。

整数や小数のデータを入力させ、Integer 型や Double 型で取得するには下記のように記述します。

```
Dim abc As Integer
Dim xyz As Double
abc = Integer.Parse( Console.ReadLine() )
xyz = Double.Parse( Console.ReadLine() )
```

上記をまとめて次のように記述できます。

```
Dim abc As Integer = Integer.Parse( Console.ReadLine() )
Dim xyz As Double = Double.Parse( Console.ReadLine() )
```

練習問題 Practice01

新たなプロジェクトを追加しましょう

以下の実行結果になるようにコードを完成させてください。

【実行結果】

```
おはよう
こんにちは
こんばんは
```

練習問題 Practice02

新たなプロジェクトを追加しましょう

身長の入力を促し、入力された値を表示させて下さい。

ユーザーは不正な入力をしないものとします。

【実行結果例】

```
身長は？
```

入力待ち状態になるので、身長を入力し、Enter キーを押してください。

```
身長は？ 168.5
168.5cm ですね
```

四則演算子

Visual Basic での四則演算は数学と同様、足し算と引き算よりも掛け算と割り算が優先して計算されます。また、() で括った場合はその括弧内の演算が優先されます。優先度が同じ場合は左から順番に演算されます。

演算記号	計算	使用例	結果
+	加算	10 + 3	13
-	減算	10 - 3	7
*	乗算	10 * 3	30
/	除算	10 / 3	3.333...
¥	除算の商	10 ¥ 3	3
Mod	除算の余	10 Mod 3	1

【Sample05】

```

3. Sub Main()
4.
5.     Dim ans_i As Integer
6.     Dim ans_d As Double
7.     Dim num1 As Integer = 10
8.     Dim num2 As Integer = 3
9.
10.    ans_i = num1 + num2
11.    Console.WriteLine(ans_i)
12.
13.    ans_i = num1 - num2
14.    Console.WriteLine(ans_i)
15.
16.    ans_i = num1 * num2
17.    Console.WriteLine(ans_i)
18.
19.    ans_d = num1 / num2
20.    Console.WriteLine(ans_d)
21.
22.    ans_i = num1 ¥ num2
23.    Console.WriteLine(ans_i)
24.
25.    ans_i = num1 Mod num2
26.    Console.WriteLine(ans_i)
27.
28. End Sub

```

小数点以下まで答えを表示させたいので
Double 型で計算結果を取得しています

【実行結果】

```
13
7
30
3.33333333333333
3
1
```

「10 / 3」の計算結果は、「3」です。10÷3の結果である「3 あまり 1」の商、つまり 3 の方です。あまりの方を取得したい場合は「10 Mod 3」とします。「Mod」でひとつの演算子です。

今回は計算結果を一旦、変数へ格納していますが、下記のように計算結果を直接コンソールへ表示させることもできます。

```
Console.WriteLine( num1 + num2 )
```

表示形式の変更

数値の三桁区切りのカンマなど、値の表示形式の変更は `Format()` 関数を利用します。

使用例

```
Dim str1 As String = Format( 1000 , "#,###" )
```

上記の結果、変数 str1 には"1,000"が代入される。

```
Dim str2 As String = Format( 12.34 , "#.#" )
```

上記の結果、変数 str2 には"12.3"が代入される。

練習問題 Practice03

単価と数量の入力を促し、入力された値から合計金額を表示させて下さい。

ユーザーは不正な入力をしないものとします。

【実行結果例】

単価を入力してください。500

数量を入力してください。3

合計金額は 1,500 です。

練習問題 Practice04

3 科目のテストの結果を入力し、その合計点と平均点を求めてください。

ユーザーは不正な入力をしないものとします。

【実行結果例】

第 1 科目の結果を入力してください。79

第 2 科目の結果を入力してください。54

第 3 科目の結果を入力してください。96

合計点は 229 点です。

平均点は 76.3 点です。

条件分岐

ある条件を満たしているときにだけ行わせたい処理がある場合、条件分岐によってその機能を実現します。

条件の真偽

プログラムでは、ある条件を満たしていることを「true（トゥルー）」、満たしていないことを「false（フォルス）」と呼びます。これらは論理値と呼ばれ、日本語では true のことを「真（しん）」、false のことを「偽（ぎ）」と呼んでいます。簡単に表現するならば、true は○、false は×です。

比較演算子を使った論理値の算出

真偽を判別するために最も多く使われるのは比較演算子です。

演算子	使用例	式の意味
>	$a > b$	a が b より大きい場合に true
<	$a < b$	a が b より小さい場合に true
>=	$a \geq b$	a が b 以上の場合に true
<=	$a \leq b$	a が b 以下の場合に true
=	$a = b$	a と b が等しい場合に true
<>	$a \neq b$	a と b が等しくない場合に true

【Sample06】

```
3.      Sub Main()  
4.  
5.          Dim num As Integer = 10  
6.  
7.          Console.WriteLine( num >= 10 )  
8.          Console.WriteLine( num > 10 )  
9.          Console.WriteLine( num <= 10 )  
10.         Console.WriteLine( num < 10 )  
11.         Console.WriteLine( num = 10 )  
12.         Console.WriteLine( num <> 10 )  
13.  
14.     End Sub
```

【実行結果】

```
True  
False  
True  
False  
True  
False
```

実際はこのような真偽の結果のみを表示することはほとんどありません。真偽判定の具体的な利用方法は次の項目で紹介します。

If 文

まず、一つ目の条件分岐の文法です。

【Sample07】

```
3.      Sub Main()  
4.  
5.          Dim score As Integer = 90  
6.  
7.          If score >= 80 Then  
8.              Console.WriteLine("合格です")  
9.          End If  
10.  
11.          Console.WriteLine("処理を終了します")  
12.  
13.      End Sub
```

【実行結果】

```
合格です  
処理を終了します
```

score に代入する値を 70 に変えて実行すると、結果は次のようになります。

```
処理を終了します
```

if 文の構文は以下の通りです。

```
If 【条件】 Then  
    条件が「真」の場合に行わせたい処理  
    条件が「真」の場合に行わせたい処理  
    ...  
End If
```

if は英語で「もし〇〇なら」という意味です。〇〇の部分は判断材料となる条件が入ります。Visual Basic ではその条件を If~Then の間に記述し、その結果が true と判断された場合、Then~End If の間の処理を行います。false と判断された場合は Then~End If の間の処理は行いません。

先ほどのサンプルを少し変更します。

```
3.      Sub Main()  
4.  
5.          Dim score As Integer  
6.  
7.          Console.Write("点数を入力してください")  
8.          score = Integer.Parse( Console.ReadLine() )  
9.  
10.         If score >= 80 Then  
11.             Console.WriteLine("合格です")  
12.         End If  
13.  
14.         Console.WriteLine("処理を終了します")  
15.  
16.     End Sub
```

【実行結果】

```
点数を入力してください 90  
合格です  
処理を終了します
```

```
点数を入力してください 70  
処理を終了します
```

これで、ソースコード内を書き換えずに、異なる結果を得られます。条件分岐は、ある状態によって挙動を変化させられるプログラムの作り方なのです。

Else 文

条件が真のときだけでなく、偽のときにも処理を行わせたい場合は、if 文に続けて else 文を記述します。

```
3.      Sub Main()  
4.  
5.          Dim score As Integer  
6.  
7.          Console.Write("点数を入力してください")  
8.          score = Integer.Parse( Console.ReadLine() )  
9.  
10.         If score >= 80 Then  
11.             Console.WriteLine("合格です")  
12.         Else  
13.             Console.WriteLine("不合格です")  
14.         End If  
15.  
16.         Console.WriteLine("処理を終了します")  
17.  
18.     End Sub
```

【実行結果】

```
点数を入力してください 90  
合格です  
処理を終了します
```

```
点数を入力してください 70  
不合格です  
処理を終了します
```

Else は「そうでなければ（そうでないとき）」という意味です。言い換えると、「If 文の条件を満たさないとき」です。処理の対象は Else から End If の間になります。

ElseIf 文

If～Else～End If 文では一つの条件に対して、真か偽かで処理を分けられました。さらに細分化する方法があります。

```
3.      Sub Main()  
4.  
5.          Dim score As Integer  
6.  
7.          Console.Write("点数を入力してください")  
8.          score = Integer.Parse( Console.ReadLine() )  
9.  
10.         If score >= 80 Then  
11.             Console.WriteLine("合格です")  
12.         ElseIf score >= 60 Then  
13.             Console.WriteLine("補欠合格です")  
14.         ElseIf score >= 40 Then  
15.             Console.WriteLine("再試験です")  
16.         Else  
17.             Console.WriteLine("不合格です")  
18.         End If  
19.  
20.         Console.WriteLine("処理を終了します")  
21.  
22.     End Sub
```

【実行結果】

点数を入力してください 90
合格です
処理を終了します

点数を入力してください 70
補欠合格です
処理を終了します

点数を入力してください 50
再試験です
処理を終了します

点数を入力してください 30
不合格です
処理を終了します

条件分岐を行う際、If は必ず記述しますが、ElseIf や Else は必要に応じて記述します。ElseIf に関しては、サンプルコードのように複数個記述することができます。

```
If score >= 80 Then          '①
    Console.WriteLine("合格です")
ElseIf score >= 60 Then      '②
    Console.WriteLine("補欠合格です")
ElseIf score >= 40 Then      '③
    Console.WriteLine("再試験です")
Else                          '④
    Console.WriteLine("不合格です")
End If
```

たとえば score が 90 の場合、「60 以上」や「40 以上」という条件だけを見れば満たしますが、その前の「80 以上」という条件を既に満たしているので、「合格です」と表示されます。そして、②以降の条件は無視されます。

よって、②は「(80 未満で) 60 以上」ということになります。

練習問題 Practice05

以下の条件を満たすようにコーディングしてください。年齢に応じたメッセージを出すサンプルコードです。

ユーザーは不正な入力をしないものとします。

- ① 判断するための年齢（整数）を格納する変数 age を宣言する
- ② 年齢を入力させ、age に格納する
- ③ age の値が 100 以上なら「長寿です」と表示する
- ④ ③の条件を満たさず、age の値が 70 以上なら「悠々自適です」と表示する
- ⑤ ③と④の条件を満たさず、age の値が 20 以上なら「働き盛りです」と表示する
- ⑥ ③～⑤の条件を満たさず、age の値が 0 以上なら「学習中です」と表示する
- ⑦ ③～⑥の条件を満たしていない場合は「負の値が入力されました」と表示する

練習問題 Practice06

入力された 2 つの数値を比較して結果を表示します。
ユーザーは不正な入力をしないものとします。

【実行結果例 1】

入力された 2 つの数値を比較します。
1 つ目の数値を入力してください。10
2 つ目の数値を入力してください。20
大きい値は 20 です。

【実行結果例 2】

入力された 2 つの数値を比較します。
1 つ目の数値を入力してください。10
2 つ目の数値を入力してください。10
2 つの値は等しいです。

練習問題 Practice07

入力された数値が偶数か奇数かを調べて表示します。
ユーザーは不正な入力をしないものとします。

【実行結果例 1】

数値を入力してください。200
入力された値は偶数です。

【実行結果例 2】

数値を入力してください。55
入力された値は奇数です。

複合条件

条件分岐において、条件を複合的に指定できます。複数の条件を絡める場合には次のような論理演算子を利用します。

演算記号	True を返す条件
And	And の左側の条件と右側の条件が両方とも True の場合
Or	Or の左側の条件と右側の条件のどちらかが True の場合

複合条件を利用した If 文の構文は以下の通りです。

```
If 【条件 1】 And 【条件 2】 Then  
    条件が「真」の場合に行わせたい処理  
    ...  
End If
```

大まかな文法は前述の If 文と変わりません。条件部分のみが異なります。

条件を複数記述し、その間に And と記述すると、2 つの条件を総合的に判断します。両方とも True の場合に、条件全体の結果が True となります。

同様に、Or と記述すると、2 つの条件のどちらかが True である場合に、条件全体の結果が True となります。

【Sample08】

```
3.      Sub Main()  
4.  
5.          Dim math As Integer  
6.          Dim eng As Integer  
7.  
8.          Console.WriteLine("数学の点数を入力してください")  
9.          math = Integer.Parse( Console.ReadLine() )  
10.  
11.         Console.WriteLine("英語の点数を入力してください")  
12.         eng = Integer.Parse( Console.ReadLine() )  
13.  
14.         If math >= 80 And eng >= 80 Then  
15.             Console.WriteLine("合格です")  
16.         Else  
17.             Console.WriteLine("不合格です")  
18.         End If  
19.  
20.         Console.WriteLine("処理を終了します")  
21.  
22.     End Sub
```

【実行結果例 1】

```
数学の点数を入力してください 95  
英語の点数を入力してください 85  
合格です  
処理を終了します
```

【実行結果例 2】

```
数学の点数を入力してください 85  
英語の点数を入力してください 75  
不合格です  
処理を終了します
```

【実行結果例 3】

```
数学の点数を入力してください 75  
英語の点数を入力してください 65  
不合格です  
処理を終了します
```

```
14.          If math >= 80 And eng >= 80 Then
```

And の左側の条件である「数学の点数が 80 以上」と And の右側の条件である「英語の点数が 80 以上」のどちらも True の場合に、条件全体の結果が True となります。

```
3. Sub Main()  
4.  
5.     Dim math As Integer  
6.     Dim eng As Integer  
7.  
8.     Console.WriteLine("数学の点数を入力してください")  
9.     math = Integer.Parse( Console.ReadLine() )  
10.  
11.    Console.WriteLine("英語の点数を入力してください")  
12.    eng = Integer.Parse( Console.ReadLine() )  
13.  
14.    If math >= 80 Or eng >= 80 Then  
15.        Console.WriteLine("合格です")  
16.    Else  
17.        Console.WriteLine("不合格です")  
18.    End If  
19.  
20.    Console.WriteLine("処理を終了します")  
21.  
22. End Sub
```

【実行結果例 1】

```
数学の点数を入力してください 95  
英語の点数を入力してください 85  
合格です  
処理を終了します
```

【実行結果例 2】

```
数学の点数を入力してください 85  
英語の点数を入力してください 75  
合格です  
処理を終了します
```

【実行結果例 3】

```
数学の点数を入力してください 75  
英語の点数を入力してください 65  
不合格です  
処理を終了します
```

```
14.         If math >= 80 Or eng >= 80 Then
```

Or の左側の条件である「数学の点数が 80 以上」と Or の右側の条件である「英語の点数が 80 以上」のどちらか一方でも True の場合に、条件全体の結果が True となります。

練習問題 Practice08

誕生月の入力を促し、その結果を表示させてください。

ユーザーはアルファベットなどの不正な入力をしないものとします。

誕生月の有効な値の範囲は 1~12 とします。

【実行結果例 1】

誕生月を入力してください。12
12 月生まれですね

【実行結果例 2】

誕生月を入力してください。13
入力された値が不正です。

練習問題 Practice09

時刻（時）と時刻（分）の入力を促し、その結果を表示させてください。

ユーザーはアルファベットなどの不正な入力をしないものとします。

時刻（時）の有効な値の範囲は 0~23、時刻（分）の有効な値の範囲は 0~59 とします。

【実行結果例 1】

時刻（時）を入力してください。12
時刻（分）を入力してください。34
入力された時刻は 12:34 です。

【実行結果例 2】

時刻（時）を入力してください。24
時刻（分）を入力してください。10
入力された値が不正です。

【実行結果例 3】

時刻（時）を入力してください。24
時刻（分）を入力してください。60
入力された値が不正です。

Select 文（多方向分岐）

条件分岐のもう一つの文法を確認します。比較のために、まずは if 文を利用したサンプルを確認してください。

【Sample09】

```
3.      Sub Main()  
4.  
5.      Dim fire As Integer  
6.  
7.      Console.Write("火力を選んでください (1~3) ")  
8.      fire = Integer.Parse( Console.ReadLine() )  
9.  
10.     If fire = 1 Then  
11.         Console.WriteLine("弱火にしました。")  
12.     ElseIf fire = 2 Then  
13.         Console.WriteLine("中火にしました。")  
14.     ElseIf fire = 3 Then  
15.         Console.WriteLine("強火にしました。")  
16.     Else  
17.         Console.WriteLine("1~3 で選んでください。")  
18.     End If  
19.  
20.     End Sub
```

【実行結果例】

火力を選んでください (1~3) 1
弱火にしました。

火力を選んでください (1~3) 2
中火にしました。

火力を選んでください (1~3) 3
強火にしました。

火力を選んでください (1~3) 8
1~3 で選んでください。

次に、新たに学ぶ Select 文のサンプルです。動作結果は全く同じです。

【Sample10】

```
3.      Sub Main()  
4.  
5.          Dim fire As Integer  
6.  
7.          Console.Write("火力を選んでください (1~3) ")  
8.          fire = Integer.Parse( Console.ReadLine() )  
9.  
10.         Select Case ( fire )  
11.             Case 1  
12.                 Console.WriteLine("弱火にしました。")  
13.             Case 2  
14.                 Console.WriteLine("中火にしました。")  
15.             Case 3  
16.                 Console.WriteLine("強火にしました。")  
17.             Case Else  
18.                 Console.WriteLine("1~3 で選んでください。")  
19.         End Select  
20.  
21.     End Sub
```

Select 文の構文は以下の通りです。

```
Select Case 【条件判断の対象】
  Case 【値 1】
    行いたい処理
    行いたい処理
    ...
  Case 【値 2】
    行いたい処理
    行いたい処理
    ...
  Case Else
    行いたい処理
    行いたい処理
    ...
End Select
```

Select Case の後に記述するのは演算式でも構いませんが、変数を記述する場合はほとんどです。Select 文は If 文とは異なり、上記の【条件判断の対象】の中に記述された内容の値によって処理を分岐する方法です。

Select Case～End Select の中に判定したい値と、その値のときに行う処理を記述します。Case ○○は「○○の場合」という意味です。

Case Else は、いずれの Case にも当てはまらなかった場合に行われます。If 文に対する Else のような位置づけです。

また、Case Else は、ほかの Case ○○の後がないといけません。

繰り返し

同じような処理を何度も実行させたい場合、数回分であればコピー＆ペーストで作れるかもしれませんが、それが 1000 回だとしたら、大変です。そのようなとき、1000 回繰り返して処理を行わせるプログラムの書き方があります。

Do-Loop 文

Do-Loop 文は if 文に近い記述形式です。if 文と比較してみましょう。

【Sample11】

```
3.      Sub Main()  
4.  
5.          Dim cnt As Integer = 1  
6.  
7.          If cnt <= 3 Then  
8.              Console.WriteLine( cnt )  
9.              cnt = cnt + 1  
10.         End If  
11.  
12.     End Sub
```

【実行結果】

```
1
```

【Sample12】

```
3.      Sub Main()  
4.  
5.          Dim cnt As Integer = 1  
6.  
7.          Do While cnt <= 3  
8.              Console.WriteLine( cnt )  
9.              cnt = cnt + 1  
10.         Loop  
11.  
12.     End Sub
```

【実行結果】

```
1  
2  
3
```

Do-Loop 文の構文は以下の通りです。

```
Do While 【条件】
    繰り返して行わせたい処理
    繰り返して行わせたい処理
    ...
Loop
```

条件を満たしているとき、Do~Loop の中を実行する点では If 文も Do-Loop 文も同様です。If 文との違いは、最後の行（Loop の行）まで実行し終えた後、Do-Loop 文は条件判定の所へ戻り、条件を満たしている場合は再度処理部分を行う点です。

流れを順に見て行きましょう。

```
5.          Dim cnt As Integer = 1
6.
7.          Do While cnt <= 3
8.              Console.WriteLine( cnt )
9.              cnt = cnt + 1
10.         Loop
```

5 行目に宣言されている変数 cnt は 1 で初期化されています。7 行目では If 文と同様に、条件判定をしています。変数 cnt が 1 のとき、条件は true ですので、8, 9 行目の処理を行います。8 行目では「1」と表示させています。9 行目は、数学的には変な式ですが、プログラムではよくある記述です。この一行は、イコール = があるので、代入の命令です。代入とは、右辺の値を左辺へコピーする事です。9 行目もその考え方と同じく、まず右辺を計算します。現在、変数 cnt は 1 ですので、右辺は 1+1 で 2 です。よって、cnt = 2 となります。そして、10 行目に行きますが、if 文であればそのまま 11 行目以降へ進みますが、Do-Loop 文の場合は 10 行目から 7 行目へ戻ってきます。これが繰り返し文の動きです。

1 度目の 7 行目の処理と違っている点は、変数 cnt が 2 になっていることです。この状態で 7 行目を見ると、変数 cnt は 3 以下なので、まだ条件は true です。同様に、8 行目で「2」と表示され、9 行目で変数 cnt は 3 になります。10 行目の後、また 7 行目へ戻ってきます。このとき、変数 cnt は 3 ですので、まだ条件を満たします。さらに 8 行目で「3」と表示され、9 行目で変数 cnt は 4 になります。10 行目の後、また 7 行目へ戻ってきます。このとき、変数 cnt は 4 ですので、今回は条件を満たしません。条件を満たしていないので、8, 9 行目は実行されずに飛ばされて、11 行目以降に進みます。

今回のサンプルで、変数 cnt は最終的に 4 になっていることに注意してください。繰り返して処理を行った回数を数えるような変数をカウンタ変数と言います。

Do-Loop 文にはもう一つの構文があります。

Do

繰り返して行わせたい処理

繰り返して行わせたい処理

...

Loop While 【条件】

条件を満たしているとき、Do～Loop の中を実行する点では先ほどの Do-Loop 文も同様です。先ほどとの違いは、条件判定の位置です。先ほどのを前判定、上記を後判定ということがあります。Do から Loop の間の処理を繰り返すことはどちらも同じですが、もう一度行うか否かを判定するタイミングが異なります。

後判定の場合は、Do から Loop の間の処理を必ず一度は行いますが、前判定の場合は条件を満たさなければ Do から Loop の間の処理を行いません。

【Sample13】

```
3.      Sub Main()  
4.  
5.          Dim password As String = "ABCD"  
6.          Dim input As String  
7.  
8.          Do  
9.              Console.WriteLine("パスワードを入力してください ")  
10.             input = Console.ReadLine()  
11.  
12.             Loop While input <> password  
13.  
14.      End Sub
```

【実行結果例】

```
パスワードを入力してください AB  
パスワードを入力してください abcd  
パスワードを入力してください ABCD
```

上記のサンプルはパスワードの入力が一致するまで繰り返し入力させるプログラムです。このように、一度実行した結果をもとに、繰り返す必要があるかどうかを判定する場合は後判定が有効です。

複合代入演算子

先ほどのサンプルに次のような記述がありました。

```
cnt = cnt + 1
```

このように、ある変数を基にして、その変数を更新したいことは多々あります。そのようにときに演算の記述を省略する事ができます。

上記の記述方法では変数名を 2 回使用していますが、複合代入演算子を使用することで、変数名を 1 回しか使用しないで記述ができます。

演算記号	省略前	省略後	演算結果
+=	num = num + 2	num += 2	13
-=	num = num - 2	num -= 2	9
*=	num = num * 2	num *= 2	22
/=	num = num / 2	num /= 2	5
&=	str = str & "xyz"	str &= "xyz"	"abcxyz"

演算結果は変数 num に 11 が入っていた場合、str に"abc"が入っていた場合の物です。

最初は省略した記述方法に慣れないと思います。しかし、慣れている人は省略して記述していますので、読めないと困ります。慣れるまでは、変数を 2 つ使う書き方に、頭の中で変換してください。そのうち自然と記述できるようになります。

For 文

もう一つの繰り返しの構文は For 文です。先ほどの Do-Loop 文と比較してみましょう。

```
3.      Sub Main()  
4.  
5.          Dim cnt As Integer = 1  
6.  
7.          Do While cnt <= 3  
8.              Console.WriteLine(cnt)  
9.              cnt = cnt + 1  
10.         Loop  
11.  
12.     End Sub
```

【Sample14】

```
3.      Sub Main()  
4.  
5.          For cnt As Integer = 1 To 3 Step 1  
6.              Console.WriteLine(cnt)  
7.          Next  
8.  
9.     End Sub
```

【実行結果】

```
1  
2  
3
```

実行結果はまったく同じになります。

For 文の構文は以下の通りです。

```
For カウンタ変数の宣言 = 初期値 To 最終値 Step 増分
    繰り返して行わせたい処理
    繰り返して行わせたい処理
    ...
Next
```

```
5.          For cnt As Integer = 1 To 3 Step 1
6.              Console.WriteLine(cnt)
7.          Next
```

5 行目は繰り返しの回数をコントロールする情報が詰まっているので、処理の流れとともに確認して行きます。まずはカウンタ変数の宣言と初期化を行います。その次に現在の値が最終値に達していないかを判定します。変数 cnt は 1 ですので、まだ最終値の 5 ではありません。この場合、For～Next の間の部分の処理を行います。6 行目で「1」と表示し、7 行目まで行くと、次は 8 行目ではなく、5 行目に戻ってきます。5 行目に戻ってきたら、Step の次に記述されている増分の値をカウンタ変数に加算します。その結果、カウンタ変数 cnt は 2 になります。その後、最終値との判定を行います。変数 cnt は 2 なので、まだ続行します。よって、6 行目が実行されます。カウンタ変数の初期化は、4 行目から来たときだけ行われます。Do-Loop 文と同様に、7 行目の後に 5 行目に戻ってきますが、その場合はカウンタ変数の初期化は行われません。

あとは Step の分だけ変化を続けながらカウンタ変数の更新と最終値との判定を繰り返し、カウンタ変数が最終値を超えたら、For～Next の間を行わずに、8 行目以降へ進みます。

Do-Loop 文と For 文の使い分け

Do-Loop 文と For 文を比較するサンプルでは、カウントアップする処理でしたが、そのような場合、一般的には For 文が使われます。それは、前述のとおり、カウントアップに関する情報がまとまっているからです。

では Do-Loop 文はどのような場合に使われるのでしょうか。Do-Loop 文は、繰り返す回数が明確ではない場合に使われます。

比較のサンプルのように、相互に書き換えは可能なので、どちらでも記述は可能です。

練習問題 Practice10

次の Do-Loop 文を用いたサンプルを For 文に書き換えてください。

```
Sub Main()  
    Dim num As Integer = 1  
    Do While num <= 10  
        Console.WriteLine(num * num)  
        num = num+ 1  
    Loop  
End Sub
```

【実行結果】

```
1  
4  
9  
16  
25  
36  
49  
64  
81  
100
```

練習問題 Practice11

Do-Loop 文または For 文を使って、次の実行結果になるようにプログラムを作ってください。

【実行結果】

```
10  
9  
8  
7  
6  
5  
4  
3  
2  
1  
0  
Finish!
```

練習問題 Practice12

入力された数値の数だけ「▲」を横方向に表示します。「▲」は「さんかく」または「きごう」を変換すると、候補に出てきます。

ユーザーは不正な入力をしないものとします。

【実行結果例 1】

数値を入力してください。2
▲▲

【実行結果例 2】

数値を入力してください。5
▲▲▲▲▲

練習問題 Practice13

お店のレジのようなイメージのプログラムを作成してください。具体的には、入力した単価を加算していき、合計金額が 1000 円以上になった場合、「合計金額が 1000 以上になったので終了します。」と表示し、処理を終了させて下さい。合計金額の初期値は 0 です。

ユーザーは不正な入力をしないものとします。

【実行結果例】

金額を入力してください。300
合計金額は 300 円です。

金額を入力してください。220
合計金額は 520 円です。

金額を入力してください。111
合計金額は 631 円です。

金額を入力してください。500
合計金額は 1131 円です。

合計金額が 1000 以上になったので終了します。

練習問題 Practice14

ATM のようなイメージのプログラムを作成してください。具体的には、メニュー（預入／引出／照会／終了）から行う操作を選択し、それぞれの処理を実行します。実行結果例を参考に、作成してください。

ユーザーは不正な入力をしないものとします。

【実行結果例】

```
1:預入 2:引出 3:照会 0:終了 1
金額を入力してください 500

1:預入 2:引出 3:照会 0:終了 3
現在の金額は 500

1:預入 2:引出 3:照会 0:終了 2
金額を入力してください 200

1:預入 2:引出 3:照会 0:終了 3
現在の金額は 300

1:預入 2:引出 3:照会 0:終了 0
```

機能追加

上記のプログラムに、金額の上限と下限を設けてください。具体的には、1,000,000 を超える金額の預け入れと、0 未満となるような引き出しはできないようにしてください。

【実行結果例】

```
1:預入 2:引出 3:照会 0:終了 1
金額を入力してください 5000000
1000000 を超えてしまいます

1:預入 2:引出 3:照会 0:終了 3
現在の金額は 0

1:預入 2:引出 3:照会 0:終了 2
金額を入力してください 100
0 未満になってしまいます

1:預入 2:引出 3:照会 0:終了 0
```

配列

配列とは、複数のデータをまとめて管理する方法です。下記のサンプルでは、生徒 5 人分の試験の結果を扱うプログラムです。今までであれば、5 つの変数を宣言するところですが、今回は「配列」というデータの格納法を使用します。

【Sample15】

```

3.      Sub Main()
4.
5.          Dim score(4) As Integer
6.
7.          score(0) = 53
8.          score(1) = 39
9.          score(2) = 77
10.         score(3) = 95
11.         score(4) = 23
12.
13.         Console.WriteLine("生徒番号 1 : 得点は" & score(0) & "点です")
14.         Console.WriteLine("生徒番号 2 : 得点は" & score(1) & "点です")
15.         Console.WriteLine("生徒番号 3 : 得点は" & score(2) & "点です")
16.         Console.WriteLine("生徒番号 4 : 得点は" & score(3) & "点です")
17.         Console.WriteLine("生徒番号 5 : 得点は" & score(4) & "点です")
18.
19.     End Sub

```

【実行結果】

```

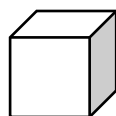
生徒番号 1 : 得点は 53 点です
生徒番号 2 : 得点は 39 点です
生徒番号 3 : 得点は 77 点です
生徒番号 4 : 得点は 95 点です
生徒番号 5 : 得点は 23 点です

```

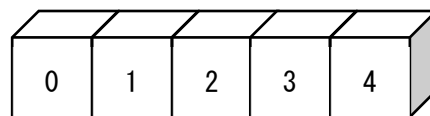
5 行目が「配列」の宣言方法です。この一行で 5 つの変数が一つにまとまった、「配列」を生成しています。配列生成時に記述する数字は「生成する要素数 - 1」です。

そして、この「配列」の中の一つの変数を今までのように扱うには、要素番号の指定が必要です。7~11 行目、13~17 行目のように、score(**要素番号**)と記述することで、変数の塊である配列の中から一つ、番号で選んだ変数を扱えるようになります。

このときの注意点は、要素番号は 0 から始まることです。最初のデータは 0 番目にあり、要素数が 5 つの配列に 5 番目はありません。



変数



配列

宣言と初期化

配列も変数と同様に、宣言と初期化を一度に行うことができます。

【Sample16】

```
3. Sub Main()  
4.  
5. Dim score() As Integer = { 53, 39, 77, 95, 23 }  
6.  
7. Console.WriteLine("生徒番号 1 : 得点は" & score(0) & "点です")  
8. Console.WriteLine("生徒番号 2 : 得点は" & score(1) & "点です")  
9. Console.WriteLine("生徒番号 3 : 得点は" & score(2) & "点です")  
10. Console.WriteLine("生徒番号 4 : 得点は" & score(3) & "点です")  
11. Console.WriteLine("生徒番号 5 : 得点は" & score(4) & "点です")  
12.  
13. End Sub
```

実行結果は同じです。

5 行目のように右辺に代入したい値を , で区切って入れ、中括弧 { } で括ります。変数宣言の部分で、要素数を記入してはいけません。右辺に記述された要素を読み取って、要素数が自動的に決定されます。

表示処理の部分に変更はありません。

繰り返し文と配列の組み合わせ

先のサンプルの表示処理の部分に注目すると、それぞれの行で異なるのは生徒番号と得点だけです。このように、処理内容の部分が一定のパターンで変化するときには繰り返し文を使って簡略化できることが多いです。

【Sample17】

```
3. Sub Main()  
4.  
5. Dim score() As Integer = { 53, 39, 77, 95, 23 }  
6.  
7. For cnt As Integer = 0 To 4  
8. Console.WriteLine("生徒番号" & (cnt + 1) &  
9. " : 得点は" & score(cnt) & "点です")  
10. Next  
11.  
12. End Sub
```

実行結果は同じです。

カウンタ変数 cnt は 0→1→2→3→4 と変化しながら For 文内の処理を行います。その場合、9 行目の score(cnt) は score(0)→score(1)→score(2)→score(3)→score(4) と変化して行きます。また、生徒番号は 1 から表示したいので、カウンタ変数 cnt に 1 を加算して調整しています。

練習問題 Practice15

3 科目のテストの結果を入力し、その合計点と平均点を求めてください。

3 つの点数を配列へ格納する繰り返しと、格納された点数を表示する繰り返しを順番に実行してください

ユーザーは不正な入力をしないものとします。

【実行結果例】

第 1 科目の結果を入力してください。 79
第 2 科目の結果を入力してください。 54
第 3 科目の結果を入力してください。 96

第 1 科目 : 79 点
第 2 科目 : 54 点
第 3 科目 : 96 点
合計点は 229 点です。
平均点は 76.3 点です。

For Each 文

For 文はカウンタ変数を変化させながら繰り返しを行います。For Each 文はカウンタ変数を使いません。For Each 文は指定したデータの集合から要素を一つずつ取り出して処理を行う繰り返し文で、要素数分だけ繰り返したら自動的に終了します。

【Sample18】

```
3. Sub Main()  
4.  
5. Dim score() As Integer = { 53, 39, 77, 95, 23 }  
6.  
7. For Each num As Integer In score  
8. Console.WriteLine("得点は" & num & "点です")  
9. Next  
10.  
11. End Sub
```

【実行結果】

```
得点は 53 点です  
得点は 39 点です  
得点は 77 点です  
得点は 95 点です  
得点は 23 点です
```

```
For Each 要素の変数の宣言 In データの集合  
    繰り返して行わせたい処理  
    ...  
Next
```

変数 score は配列で、5 つの要素を持っています。今回は For Each 文でこの 5 つの要素を順番に取り出し、処理を行っています。1 回目の繰り返しのときは変数 num に 53 が格納され、2 回目は 39 が、3 回目に 77 というように、配列の要素を順番に変数に格納しながら繰り返しが進んでいきます。

For 文と For Each 文の使い分け

前述のとおり、For Each 文はデータの集合に対して、一つずつアクセスして処理をする方法でした。For 文のサンプルでもデータの集合（配列）に対してカウンタ変数を変化させることで一つずつアクセスしていました。よって、本来であればFor Each 文で行うべき内容です。では、For 文はどのような場合に使うのでしょうか。

それは、変化する数値を、単純に扱いたいときです。変化する値を自在に変化させ、それに合わせて計算結果を得たり処理をしたりする場合には値を制御しやすいFor 文を利用します。

練習問題 Practice16

For Each 文を使って、次の実行結果になるようにプログラムを作ってください。ただし、表示させる内容は配列で用意してください。

【実行結果】

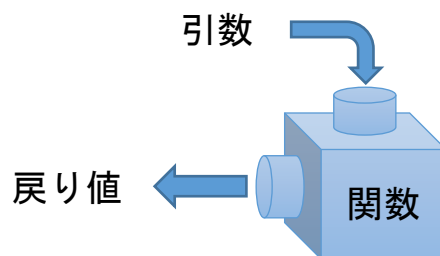
春	
夏	
秋	
冬	

関数

関数とは、一連の処理に名前がつけられたもので、その名前を指定するだけで一連の処理を行わせることができるものです。今までプロシージャやメソッドと呼んできたものと同じものだと思ってください。

関数は、下図のようなものを考えてみてください。

何か仕事をしてくれる箱に材料を入れると、加工後のモノが出てくるイメージです。

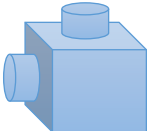





関数は存在するだけでは何も起こりません。呼び出して、初めて動作します。

関数で必要となる材料を「引数」と言い、利用する側が与えます。処理が終了すると得られる結果を「戻り値」と言います。

関数の種類

引数と戻り値が存在するかしないかの組み合わせで、関数は下記の4種類に分類できます。特に、戻り値があるものを Function プロシージャ、戻り値がないものを Sub プロシージャと言います。

プロシージャ	引数	あり	なし
	戻り値		
Function プロシージャ	あり		
Sub プロシージャ	なし		

関数の書式

Left() 関数を例にして、関数の書式を確認しましょう。インターネット等で調べると、下記のように、関数の使い方がまとめられています。

関数名	Left
引数	string 省略不可 , length 省略不可
戻り値	string
処理内容	引数 string で与えられた文字列の左端から引数 length で与えられた文字数の分だけ取り出します。

この情報をもとに、Left() 関数を利用してみましょう。

【Sample19】

```

3.      Sub Main()
4.
5.          Dim rtn As String
6.
7.          rtn = Left("ABCDEF", 4)
8.
9.          Console.WriteLine(rtn)
10.
11.      End Sub

```

【実行結果】

ABCD

関数を利用するときの基本的な書式は以下のとおりです。

rtn = Left ("ABCDEF" , 4))

③
①
②
①

① 関数名	引数を指定するための括弧も含めて覚えてください。引数の指定が必要なくても括弧の記述は必要です。
② 引数	関数の用途に合わせて必要な引数を指定します。複数指定する場合は、カンマで区切って記述します。 複数ある場合、第一引数、第二引数・・・と呼びます。
③ 戻り値	関数の結果を変数で受け取ります。関数が処理を行った結果が代入されるのであって、関数そのものが変数の中に入るわけではありません。

引数の指定

呼び出された関数が、渡された値を認識できれば良いので、渡す側（呼び出し側）はどのような方法でも「値」を渡せばよいのです。

【Sample20】

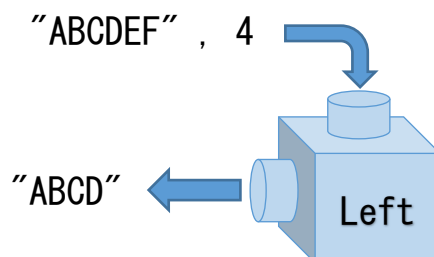
```
3. Sub Main()  
4.  
5.     Dim str As String = "ABCDEF"  
6.     Dim num As Integer = 4  
7.  
8.     Console.WriteLine( Left("ABCDEF", 4) )  
9.     Console.WriteLine( Left( str, num) )  
10.    Console.WriteLine( Left("ABC" & "DEF", 2 * 2) )  
11.    Console.WriteLine( Left(str, 2 * 2) )  
12.  
13. End Sub
```

8 行目のように直接値を指定すればもちろん処理をしてくれます。

9 行目のように変数で指定した場合は変数が渡されるわけではなく、変数の中身が渡されます。第二引数は変数 num が指定されていますが、変数 num が渡されるのではなく、変数 num に格納されている 4 という値が渡されます。

10 行目も同様に、第一引数は文字列の結合が行われた結果が、第二引数は掛け算が行われた結果が渡されます。

11 行目のように、第一引数と第二引数の指定の形式が一致している必要はありません。いずれも指定の形式は関係なく、値が渡されるということです。



練習問題 Practice17

Left の例を参考に、下記の関数を調べて利用し、結果を表示させてください。Replace の第三引数までとしてください。第四引数以降は省略可能です。

関数名	引数の型と用途	戻り値の型	関数の用途
Left	1:String もとになる文字列 2:Long 返す文字数	String	第一引数に与えられた文字列の左から、第二引数で指定された文字数分の文字列を返す
Right			
Mid			
Replace			

```
Sub Main()
    Dim str As String = "ABCDEF"
    Console.WriteLine( Left("ABCDEF", 3) )
End Sub
```

【実行結果】

```
ABC
```

自作メソッド（自作関数）

メソッドは関数と同じものと、今のところは思ってください。簡単に両者の違いを述べると、どこかに所属している関数のことをメソッドと言い、どこにも所属していないのが関数です。

今まで記述してきた Main は Module1 に所属しているメソッドです。コンソール画面に文字列を表示させるために利用してきた WriteLine は Console に所属しているメソッドです。前述の Left は関数です。メソッドについての詳しい解説は応用編で取り上げますので、今の段階では違いを意識する必要はありません。どちらも、「何か処理をしてくれる塊」という認識で問題ありません。

今からはそのメソッド・関数を作成するときの記述のルールを確認して行きます。

メソッドや関数は用意されているものを使うだけではありません。世の中の大勢の人が必要であろう処理は関数として用意されていることが多いですが、自分が必要な処理のすべてが用意されているわけではありません。必要なメソッド・関数は自作する必要があります。

まずは下記のサンプルを確認してください。メソッドを利用していない、今までの書き方です。

【Sample21】

```
3.  Module Module1
4.
5.      Sub Main()
6.
7.          Console.WriteLine("整数値を入力させ、変数 x に代入")
8.          Console.WriteLine("整数値を入力させ、変数 y に代入")
9.          Console.WriteLine("x と y を比較し、大きい方の数値を変数 z に代入")
10.         Console.WriteLine("変数 z を表示")
11.
12.     End Sub
13.
14. End Module
```

一連の処理を関数にまとめると、下記のようになります。

【Sample22】

```

1.  Module Module1
2.
3.      Sub MyComp()
4.
5.          Console.WriteLine("整数値を入力させ、変数 x に代入")
6.          Console.WriteLine("整数値を入力させ、変数 y に代入")
7.          Console.WriteLine("x と y を比較し、大きい方の数値を変数 z に代入")
8.          Console.WriteLine("変数 z を表示")
9.
10.     End Sub
11.
12.     Sub Main()
13.
14.         MyComp()
15.
16.     End Sub
17.
18. End Module

```

Main() メソッドの中の処理が「 MyComp() 」の一行だけになってしまいましたが、14 行目は 3～10 行目の処理の塊を指しています。14 行目にブレークポイントを設定し、「ステップイン実行」してみてください。14 行目の次は 3 行目が黄色くなります。そして 10 行目まで実行し終わった後は、また 14 行目に戻ってきます。(ブレークポイントやステップイン実行については巻末のデバッグ手法を参照してください。)

3 行目と 12 行目の記述を確認しましょう。今まで扱ってきた Main() メソッドとほぼ同じです。MyComp という名前は決まったものではなく、任意です。

Sub～End Sub まだがプロシージャの範囲で、この間に必要な処理を記述します。

メソッドの複数回呼び出し

作成したメソッドは何度も呼び出して処理を行わせることが可能です。

```

12.     Sub Main()
13.
14.         MyComp()
15.         MyComp()
16.
17.     End Sub

```

実行すると、5～8 行目の表示処理が 2 回行われます。「ステップイン実行」で確認すると、14 行目と、15 行目の 2 度、MyComp() メソッドの中を通っていることが確認できます。

先のサンプルの 5~8 行目で画面出力させている内容を具体的に実装すると、以下のようになります。

【Sample23】

```
1.  Module Module1
2.
3.      Sub MyComp()
4.
5.          Dim x As Integer
6.          Dim y As Integer
7.          Dim z As Integer
8.
9.          Console.Write("整数値を入力してください。")
10.         x = Integer.Parse( Console.ReadLine() )
11.         Console.Write("整数値を入力してください。")
12.         y = Integer.Parse( Console.ReadLine() )
13.
14.         If x > y Then
15.             z = x
16.         Else
17.             z = y
18.         End If
19.
20.         Console.WriteLine("大きい値は" & z & "です。")
21.         Console.WriteLine()
22.
23.     End Sub
24.
25.     Sub Main()
26.
27.         MyComp()
28.         MyComp()
29.
30.     End Sub
31.
32. End Module
```

【実行結果例】

```
整数値を入力してください。5
整数値を入力してください。10
大きい値は 10 です。

整数値を入力してください0
整数値を入力してください-10
大きい値は 0 です
```


練習問題 Practice18

下記のような定義を持つ関数を作成し、Main() 関数内で指定された回数分だけ呼び出してください。

関数名	Star
引数	なし
戻り値	なし
処理	★ と表示する

```
Module Module1
```

```
    ' Star() メソッドを作成してください
```

```
    Sub Main()
```

```
        Dim num As Integer
```

```
        Console.WriteLine("表示回数を入力してください。")  
        num = Integer.Parse( Console.ReadLine() )
```

```
        ' Star() メソッドを指定回数分、呼び出してください
```

```
    End Sub
```

```
End Module
```

【実行結果】

```
表示回数を入力してください。5
```

```
★★★★★
```

引数を持つメソッドの作成

先のサンプルを変更して、引数を渡せるようにします。現段階では比較対象となる値を、メソッド内の `Console.WriteLine()` メソッドで入力させるしかありませんが、例えば、ファイルから得るなど、異なる方法で発生した値を比較したいことがあるかもしれません。その場合は、メソッド内でデータの取得方法を決めてしまうのではなく、呼び出し側がメソッドへ比較対象となる値を与えるようにすればよいのです。

引数の渡し方はサンプルの中で確認していきましょう。

【Sample24】

```
1.  Module Module1
2.
3.      Sub MyComp(x As Integer, y As Integer)
4.
5.          Dim z As Integer
6.
7.          If x > y Then
8.              z = x
9.          Else
10.             z = y
11.          End If
12.
13.          Console.WriteLine("大きい値は" & z & "です。")
14.          Console.WriteLine()
15.
16.      End Sub
17.
18.      Sub Main()
19.
20.          Dim a As Integer
21.          Dim b As Integer
22.
23.          Console.Write("整数値を入力してください。")
24.          a = Integer.Parse(Console.ReadLine())
25.
26.          Console.Write("整数値を入力してください。")
27.          b = Integer.Parse(Console.ReadLine())
28.
29.          MyComp(a, b)
30.
31.      End Sub
32.
33. End Module
```

3行目の`()`括弧の中に変数宣言が、で区切られて2つあります。`MyComp()`メソッドの定義側から見ると、メソッドは2つの `Integer` 型のデータを受け取り、呼び出し側から見ると、2つの `Integer` 型のデータをメソッドへ渡す形になりました。また、引き渡された値はそれぞれ変数 `x`、変数 `y` へ格納されます。

受け取る側も、変数 `a` と変数 `b` が得られるのではなく、両変数に格納された値が得られることに注意してください。

練習問題 Practice19

下記のような定義を持つ関数を作成し、Main() 関数内で 1 回呼び出してください。

関数名	Star
引数	Integer 型
戻り値	なし
処理	引数で受け取った数値の分だけ★と表示する

Module Module1

```
' Star() メソッドを作成してください
```

```
Sub Main()
```

```
    Dim num As Integer
```

```
    Console.WriteLine("表示回数を入力してください。")
    num = Integer.Parse(Console.ReadLine())
```

```
' Star() メソッドを1回呼び出してください
```

```
End Sub
```

```
End Module
```

【実行結果】

```
表示回数を入力してください。5
★★★★★
```

練習問題 Practice20

下記のような定義を持つ関数を作成し、Main() メソッドから 1 回呼び出してください。

関数名	Multi
引数	Integer 型
戻り値	なし
処理	引数で受け取った数値を 2 乗して表示する

【実行結果例】

```
数値を入力してください。5
2 乗した値は 25 です。
```

戻り値を持つメソッドの作成

さらに MyComp () メソッドを変更して、比較結果を呼び出し元へ返すようにします。

【Sample25】

```

1.  Module Module1
2.
3.      Function MyComp(x As Integer, y As Integer) As Integer
4.
5.          Dim z As Integer
6.
7.          If x > y Then
8.              z = x
9.          Else
10.             z = y
11.          End If
12.
13.          Return z
14.
15.      End Function
16.
17.      Sub Main()
18.
19.          Dim a As Integer
20.          Dim b As Integer
21.          Dim c As Integer
22.          Console.WriteLine("整数値を入力してください。")
23.          a = Integer.Parse(Console.ReadLine())
24.
25.          Console.WriteLine("整数値を入力してください。")
26.          b = Integer.Parse(Console.ReadLine())
27.
28.          c = MyComp(a, b)
29.
30.          Console.WriteLine("大きい値は" & c & "です。")
31.          Console.WriteLine()
32.
33.      End Sub
34.
35. End Module

```

定義側の変更点を確認して行きます。まずは3行目、メソッド名の左が Sub から Function に変わっています。結果を返さないメソッドの場合は Sub~End Sub ですが、結果を返すメソッドの場合は Function~End Function に記述します。また、引数の記述の右に「戻り値の型」の記述が追加されました。13行目では具体的に何を返すのかを記述しています。今回は変数 z の内容を返すので、「Return」というキーワードを使って変数 z を指定しています。

呼び出し側での変更点は、関数から結果が得られるようになったことで、結果を受け取る必要が出てきました。今回は28行目で変数 c に受け取っています。

標準関数を利用したときの記述とまったく変わらないことを確認してください。

練習問題 Practice21

下記のような定義を持つ関数を作成し、Main() 関数から 1 回呼び出してください。また、呼び出した結果を受け取り、表示させてください。

関数名	Multi
引数	Integer 型
戻り値	Integer 型
処理	引数で受け取った数値を 2 乗して返す

【実行結果例】

```
数値を入力してください。5
2 乗した値は 25 です。
```

練習問題 Practice22

下記のような定義を持つ関数を作成し、Main() メソッドから 1 回呼び出してください。Main() メソッドでは、power() メソッドに渡すための 2 つの値を Console.ReadLine() メソッドにより取得してください。また、power() メソッドの結果を表示させてください。

関数名	Power
引数	Integer 型, Integer 型
戻り値	Integer 型
処理	第一引数で受け取った数値を基にして、第二引数で受け取った数値の分だけ繰り返して掛け算して返す

【実行結果例】

```
数値 1 を入力してください。2
数値 2 を入力してください。4
2 を 4 乗した値は 16 です。
```


練習問題解答例

練習問題 Practice01

```
Module Module1
    Sub Main()
        Console.WriteLine("おはよう")
        Console.WriteLine("こんにちは")
        Console.WriteLine("こんばんは")
    End Sub
End Module
```

練習問題 Practice02

```
Module Module1
    Sub Main()
        Dim height As Double
        Console.Write("身長は？")
        height = Double.Parse(Console.ReadLine())
        Console.WriteLine(height & "cm ですね")
    End Sub
End Module
```

練習問題 Practice03

```
Module Module1
    Sub Main()
        Dim tanka As Integer
        Dim suu As Integer
        Dim goukei As String
        Console.Write("単価を入力してください。")
        tanka = Integer.Parse(Console.ReadLine())
        Console.Write("数量を入力してください。")
        suu = Integer.Parse(Console.ReadLine())
        goukei = Format(tanka * suu, "#,###")
        Console.WriteLine("合計金額は" & goukei & "です。")
    End Sub
End Module
```

練習問題 Practice04

```
Module Module1

    Sub Main()

        Dim score1 As Integer
        Dim score2 As Integer
        Dim score3 As Integer
        Dim total As Integer
        Dim average As String

        Console.WriteLine("第1科目の結果を入力してください。")
        score1 = Integer.Parse(Console.ReadLine())
        Console.WriteLine("第2科目の結果を入力してください。")
        score2 = Integer.Parse(Console.ReadLine())
        Console.WriteLine("第3科目の結果を入力してください。")
        score3 = Integer.Parse(Console.ReadLine())

        total = score1 + score2 + score3
        average = Format(total / 3, "#.#")

        Console.WriteLine("合計点は" & total & "点です。")
        Console.WriteLine("平均点は" & average & "点です。")

    End Sub

End Module
```

練習問題 Practice05

```
Module Module1

    Sub Main()

        Dim age As Integer

        Console.WriteLine("年齢は？")
        age = Integer.Parse(Console.ReadLine())

        If age >= 100 Then
            Console.WriteLine("長寿です")
        ElseIf age >= 70 Then
            Console.WriteLine("悠々自適です")
        ElseIf age >= 20 Then
            Console.WriteLine("働き盛りです")
        ElseIf age >= 0 Then
            Console.WriteLine("学習中です")
        Else
            Console.WriteLine("負の値が入力されました")
        End If

    End Sub

End Module
```


練習問題 Practice06

```
Module Module1
    Sub Main()
        Dim num1 As Integer
        Dim num2 As Integer

        Console.WriteLine("入力された 2 つの数値を比較します。")

        Console.Write("1 つ目の数値を入力してください。")
        num1 = Integer.Parse(Console.ReadLine())
        Console.Write("2 つ目の数値を入力してください。")
        num2 = Integer.Parse(Console.ReadLine())

        If num1 > num2 Then
            Console.WriteLine("大きい値は" & num1 & "です。")
        ElseIf num1 < num2 Then
            Console.WriteLine("大きい値は" & num2 & "です。")
        Else
            Console.WriteLine("2 つの値は等しいです。")
        End If

    End Sub
End Module
```

練習問題 Practice07

```
Module Module1
    Sub Main()
        Dim num As Integer

        Console.Write("数値を入力してください。")
        num = Integer.Parse(Console.ReadLine())

        If (num Mod 2) = 0 Then
            Console.WriteLine("入力された値は偶数です。")
        Else
            Console.WriteLine("入力された値は奇数です。")
        End If
    End Sub
End Module
```

練習問題 Practice08

```
Module Module1

    Sub Main()

        Dim month As Integer

        Console.WriteLine("誕生月を入力してください。")
        month = Integer.Parse( Console.ReadLine() )

        If month >= 1 And month <= 12 Then
            Console.WriteLine(month & "月生まれですね")
        Else
            Console.WriteLine("入力された値が不正です")
        End If

    End Sub

End Module
```

練習問題 Practice09

```
Module Module1

    Sub Main()

        Dim hour As Integer
        Dim min As Integer

        Console.WriteLine("時刻（時）を入力してください。")
        hour = Integer.Parse( Console.ReadLine() )

        Console.WriteLine("時刻（分）を入力してください。")
        min = Integer.Parse( Console.ReadLine() )

        If hour >= 0 And hour <= 23 And
            min >= 0 And min <= 59 Then
            Console.WriteLine("入力された時刻は" &
                hour & ":" & min & "です")
        Else
            Console.WriteLine("入力された値が不正です")
        End If

    End Sub

End Module
```

練習問題 Practice10

```
Module Module1

    Sub Main()

        For num As Integer = 1 To 10
            Console.WriteLine(num * num)
        Next

    End Sub

End Module
```

練習問題 Practice11

```
Module Module1
    Sub Main()
        For num As Integer = 0 To 10
            Console.WriteLine(10 - num)
        Next

        Console.WriteLine("Finish!")
    End Sub
End Module
```

別解

```
Module Module1
    Sub Main()
        For num As Integer = 10 To 0 Step -1
            Console.WriteLine(num)
        Next

        Console.WriteLine("Finish!")
    End Sub
End Module
```

別解

```
Module Module1
    Sub Main()
        Dim num As Integer = 10
        Do While num >= 0
            Console.WriteLine(num)
            num -= 1
        Loop

        Console.WriteLine("Finish!")
    End Sub
End Module
```

練習問題 Practice12

```
Module Module1
    Sub Main()
        Dim num As Integer

        Console.WriteLine("数値を入力してください。")
        num = Integer.Parse(Console.ReadLine())

        For cnt As Integer = 1 To num
            Console.WriteLine("▲")
        Next

    End Sub
End Module
```

別解

```
Module Module1
    Sub Main()
        Dim num As Integer

        Console.WriteLine("数値を入力してください。")
        num = Integer.Parse(Console.ReadLine())

        Do While num > 0
            Console.WriteLine("▲")
            num -= 1
        Loop

    End Sub
End Module
```

練習問題 Practice13

```
Module Module1
    Sub Main()
        Dim total As Integer

        Do While total < 1000
            Console.WriteLine("金額を入力してください。")
            total += Integer.Parse(Console.ReadLine())

            Console.WriteLine("合計金額は" & total & "円です。")
            Console.WriteLine()
        Loop

        Console.WriteLine("合計金額が 1000 以上になったので終了します。")

    End Sub
End Module
```

練習問題 Practice14

```
Module Module1

    Sub Main()

        Dim money As Integer = 0
        Dim inpcmd As Integer

        Do
            Console.WriteLine("1:預入 2:引出 3:照会 0:終了 ")
            inpcmd = Integer.Parse(Console.ReadLine())

            Select Case inpcmd
                Case 1
                    Console.WriteLine("金額を入力してください ")
                    money += Integer.Parse(Console.ReadLine())

                Case 2
                    Console.WriteLine("金額を入力してください ")
                    money -= Integer.Parse(Console.ReadLine())

                Case 3
                    Console.WriteLine("現在の金額は" & money)

            End Select

            Console.WriteLine()

        Loop While inpcmd <> 0

    End Sub

End Module
```

追加機能を含む

```
Module Module1

    Sub Main()

        Dim money As Integer = 0
        Dim inpcmd As Integer
        Dim inpmny As Integer

        Do
            Console.WriteLine("1:預入 2:引出 3:照会 0:終了 ")
            inpcmd = Integer.Parse(Console.ReadLine())

            Select Case inpcmd
                Case 1
                    Console.WriteLine("金額を入力してください ")
                    inpmny = Integer.Parse(Console.ReadLine())

                    If (money + inpmny) <= 1000000 Then
                        money += inpmny
                    Else
                        Console.WriteLine("1000000 を超えてしまいます")
                    End If

                Case 2
                    Console.WriteLine("金額を入力してください ")
                    inpmny = Integer.Parse(Console.ReadLine())

                    If (money - inpmny) >= 0 Then
                        money -= inpmny
                    Else
                        Console.WriteLine("0 未満になってしまいます")
                    End If

                Case 3
                    Console.WriteLine("現在の金額は" & money)

            End Select

            Console.WriteLine()

            Loop While inpcmd <> 0

        End Sub

    End Module
```

練習問題 Practice15

```
Module Module1

    Sub Main()

        Dim score(2) As Integer
        Dim total As Integer
        Dim average As Double

        For cnt As Integer = 0 To 2
            Console.WriteLine("第" & cnt + 1 & "科目の結果を入力してください。")
            score(cnt) = Integer.Parse(Console.ReadLine())
        Next

        Console.WriteLine()

        For cnt As Integer = 0 To 2
            Console.WriteLine("第" & cnt + 1 & "科目：" & score(cnt) & "点")
            total += score(cnt)
        Next

        average = Format(total / 3, "#.##")

        Console.WriteLine("合計点は" & total & "点です。")
        Console.WriteLine("平均点は" & average & "点です。")

    End Sub

End Module
```

練習問題 Practice16

```
Module Module1

    Sub Main()

        Dim season() As String = {"春", "夏", "秋", "冬"}

        For Each str As String In season
            Console.WriteLine(str)
        Next

    End Sub

End Module
```

練習問題 Practice17

関数名	引数の型と用途	戻り値の型	関数の用途
Right	1:String もとになる文字列 2:Long 返す文字数	String	第一引数に与えられた文字列の右から、第二引数で指定された文字数分の文字列を返す。
Mid	1:String もとになる文字列 2:Long 開始位置 3:Long 返す文字数	String	第一引数に与えられた文字列から、第二引数で指定された位置から第三引数で指定された文字数分の文字列を返す。
Replace	1:String もとになる文字列 2:String 検索する文字列 3:String 置換する文字列	String	第一引数に与えられた文字列から、第二引数で指定された文字列を探し出し、第三引数で指定された文字列に置換する。その結果を返す。

Module Module1

Sub Main()

Dim str As String = "ABCDEF"

Console.WriteLine(Right("ABCDEF", 4))

Console.WriteLine(Mid("ABCDEF", 3, 2))

Console.WriteLine(Replace("ABCDEF", "CD", "cd"))

End Sub

End Module

練習問題 Practice18

```
Module Module1
    Sub Star()
        Console.Write("★")
    End Sub
    Sub Main()
        Dim num As Integer
        Console.Write("表示回数を入力してください。")
        num = Integer.Parse( Console.ReadLine() )
        For cnt As Integer = 1 To num
            Star()
        Next
    End Sub
End Module
```

練習問題 Practice19

```
Module Module1
    Sub Star(num As Integer)
        For cnt As Integer = 1 To num
            Console.Write("★")
        Next
    End Sub
    Sub Main()
        Dim num As Integer
        Console.Write("表示回数を入力してください。")
        num = Integer.Parse( Console.ReadLine() )
        Star(num)
    End Sub
End Module
```

練習問題 Practice20

```
Module Module1
    Sub Multi(val As Integer)
        Console.WriteLine("2 乗した値は" & val * val & "です。")
    End Sub
    Sub Main()
        Dim num As Integer
        Console.Write("数値を入力してください。")
        num = Integer.Parse( Console.ReadLine() )
        Multi(num)
    End Sub
End Module
```

練習問題 Practice21

```
Module Module1

    Function Multi(val As Integer) As Integer
        Dim ans As Integer = val * val
        Return ans
    End Function

    Sub Main()
        Dim num As Integer
        Dim ret As Integer

        Console.Write("数値を入力してください。")
        num = Integer.Parse( Console.ReadLine() )

        ret = Multi(num)

        Console.WriteLine("2 乗した値は" & ret & "です。")
    End Sub
End Module
```

練習問題 Practice22

```
Module Module1

    Function Power(num1 As Integer, num2 As Integer) As Integer
        Dim ans As Integer = 1
        For cnt As Integer = 1 To num2
            ans *= num1
        Next
        Return ans
    End Function

    Sub Main()
        Dim num1 As Integer
        Dim num2 As Integer
        Dim ret As Integer

        Console.Write("数値 1 を入力してください。")
        num1 = Integer.Parse( Console.ReadLine() )
        Console.Write("数値 2 を入力してください。")
        num2 = Integer.Parse( Console.ReadLine() )

        ret = Power(num1, num2)

        Console.WriteLine(num1 & "を" & num2 & "乗した値は" & ret & "です。")
    End Sub
End Module
```

課題 1（コーディングの基本）

フォームアプリケーション

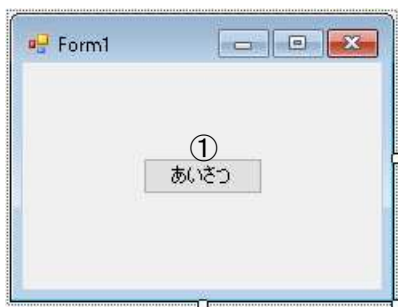
日常的に利用されているアプリケーションは、文法の学習で使用した文字ばかりのコンソール形式ではなく、ボタンやテキストボックスを利用したフォーム形式です。課題では、フォームアプリケーションを作りながら文法の確認をしていきましょう。

最初に作成するのは下記のようなアプリケーションです。まずは作成するアプリケーションの概要を確認しましょう。

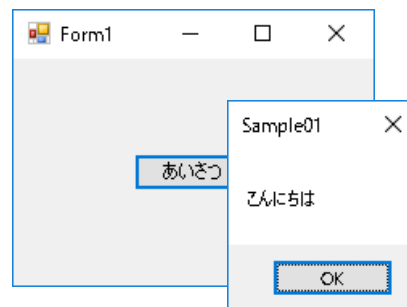
サンプル 1

フォームにボタンをひとつ配置し、押したときにメッセージボックスを表示させるアプリケーションです。

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	Button	Text	あいさつ

コントロール	①	イベント	Click
MsgBox () 関数を利用してメッセージボックスで「こんにちは」と表示させる。			

文言を表示させ、ユーザーに確認させたい場合は `MsgBox()` 関数を使います。

使用例

MsgBox(表示文字列)

処理：ユーザーにボタン押下操作を求めるフォームを表示する。

完成時のソースコードは次のとおりです。

```
Public Class Form1

    Private Sub Button1_Click(sender As Object,
                                e As EventArgs) Handles Button1.Click

        MsgBox("こんにちは")

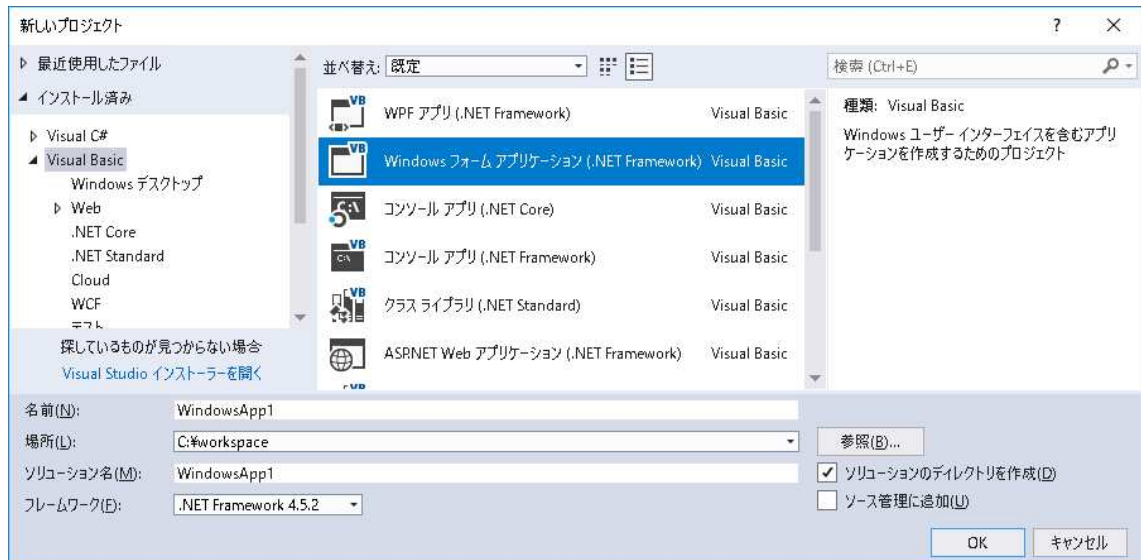
    End Sub

End Class
```

実際に作成する手順を確認しましょう。

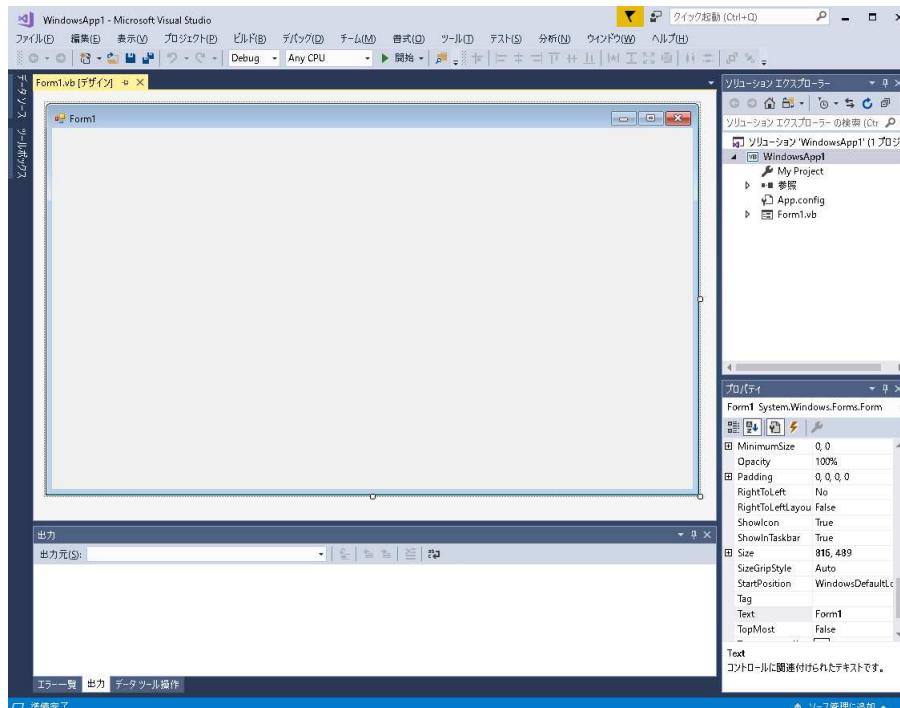
プロジェクトの作成

プロジェクトの作成の方法は基本的にコンソールアプリケーションと同様です。メインメニューからファイル>新規作成>プロジェクトと選択します。

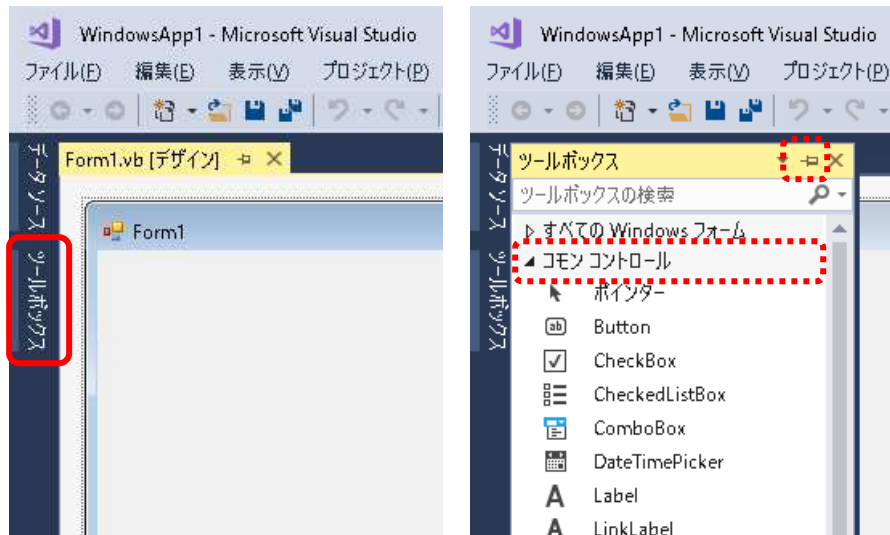


今回はコンソールアプリ (.NET Framework) ではなく、Windows フォームアプリケーション (.NET Framework) を選択します。

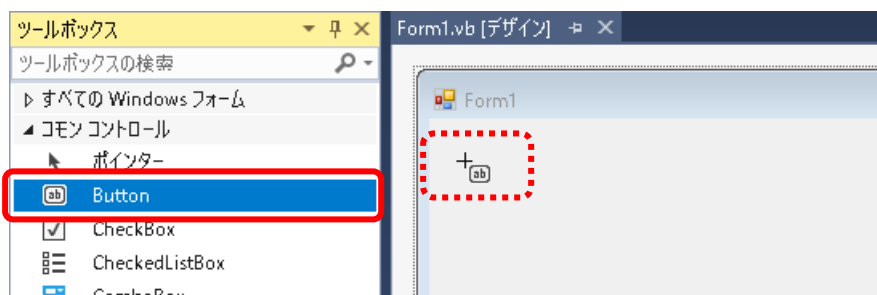
下図のように表示されます。



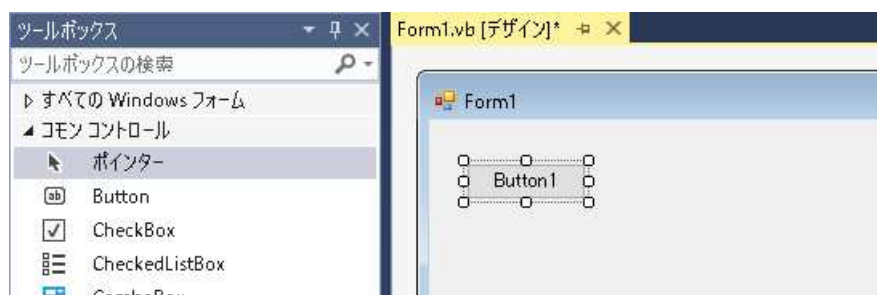
画面左端の「ツールボックス」をクリックすると、ウィンドウが表示されます。常に表示させたい場合はピンをクリックします。頻繁に使用するのはコンモンコントロールです。展開されていなければ▷をクリックして展開してください。



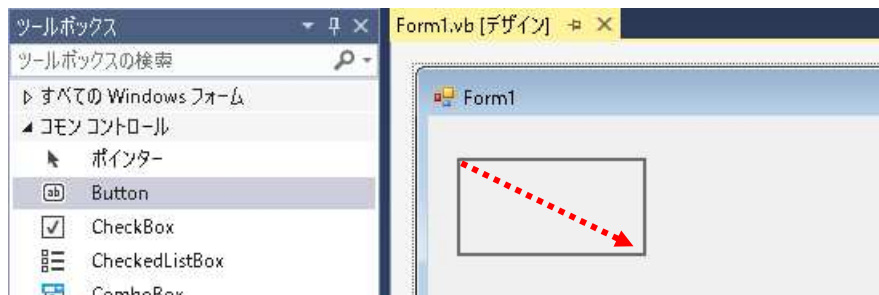
ツールボックス内のコントロール（部品：下図の場合は Button）をクリックし、フォームの上にカーソルを移動してください。



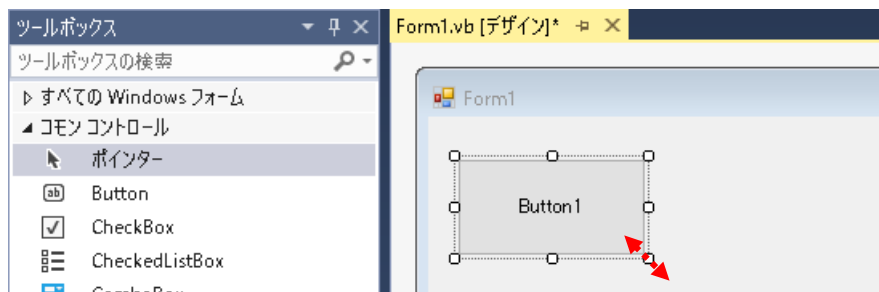
この状態でクリックすると、フォーム上に選択したコントロールがデフォルトの大きさに配置されます。



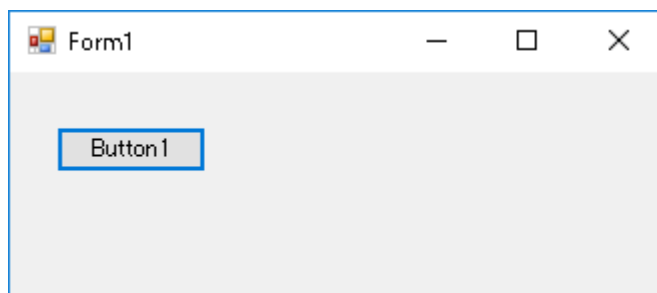
クリックではなく、ドラッグをしてからマウスを離すと、ボタンの大きさを決めながら配置できます。



位置や大きさは後から変更できますので、慎重に操作する必要はありません。



コンソールアプリケーションと同様に実行すると、フォームが起動します。



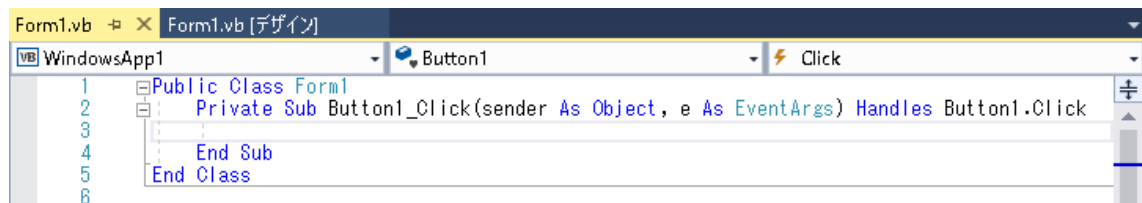
ボタンをクリックしても何も起きません。それは、ボタンをクリックしたときに何をするかを決めていないからです。

フォームアプリケーションのサンプルや問題におけるフォームのデザイン（配置の場所や大きさなど）は任意です。

イベント

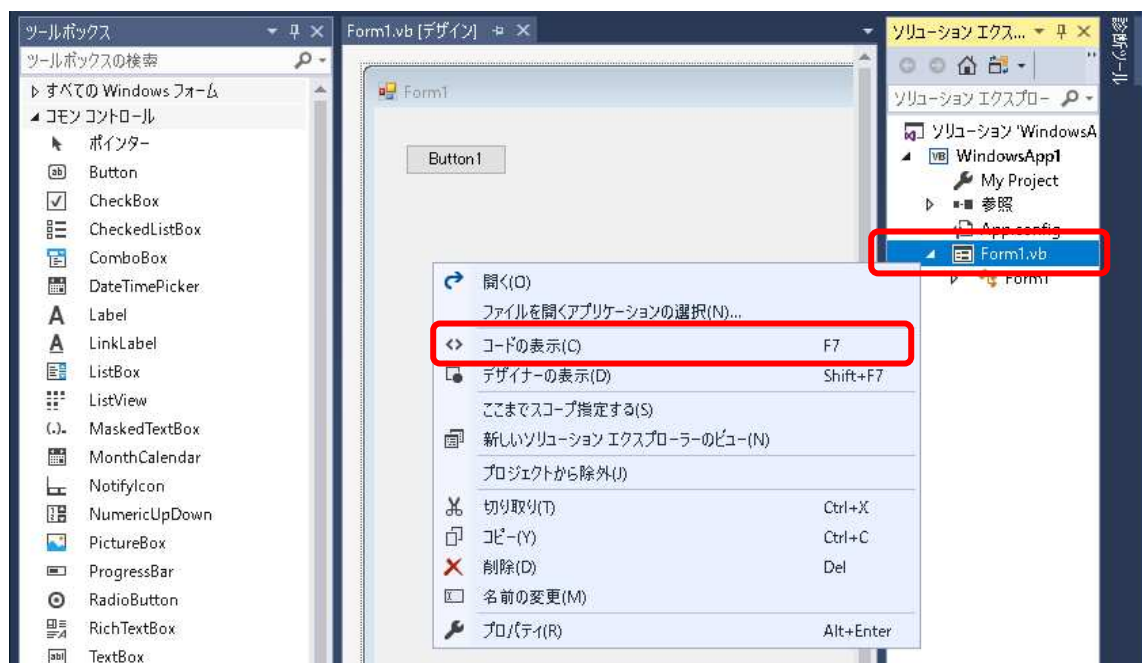
イベントとは、「ボタンをクリックしたとき」のように、特定のタイミングのことをイベントと言い、そのイベントが発生したときにプロシージャを実行させることができます。それをイベントプロシージャといいます。

イベントプロシージャを設定してみましょう。配置したボタンをダブルクリックしてください。



これだけでボタンクリックのイベントプロシージャが生成されました。この Sub~End Sub の間に、ボタンが押されたタイミングで実行したいプログラムを記述します。

ソリューションエクスプローラーから Form1.vb を右クリックし、「コードの表示」を選択してもソースコードエディタが表示されます。



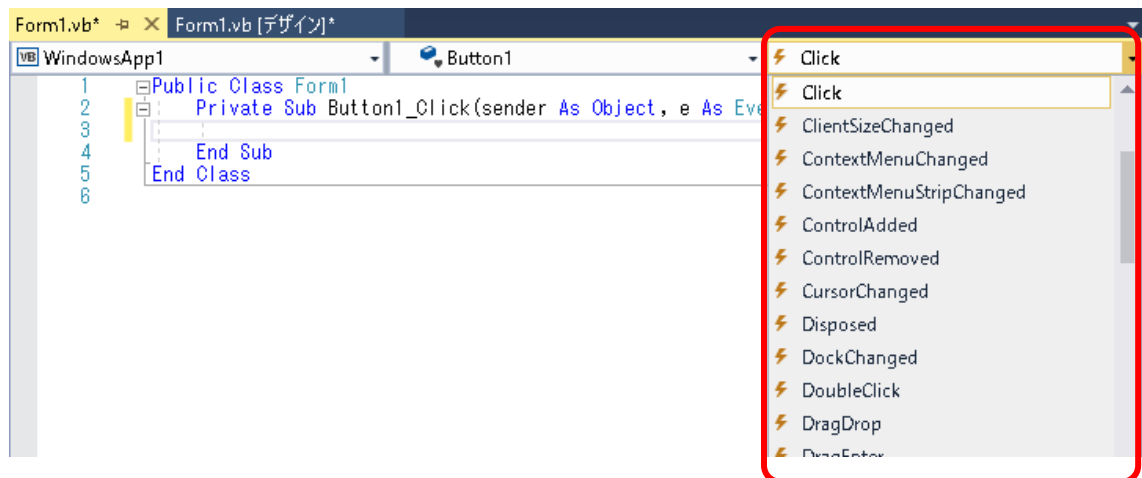
ボタンをクリックしたときの動作を設定しましょう。

生成されたイベントプロシージャを下記のように編集してください。プロシージャの宣言の行が紙面の都合で2行になっていますが、自動生成された1行のままで問題ありません。

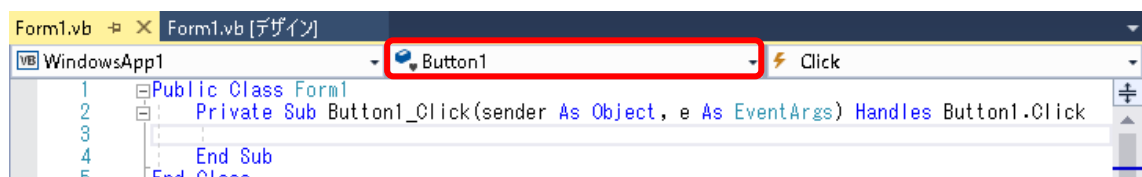
```
Public Class Form1
    Private Sub Button1_Click(sender As Object,
                                e As EventArgs) Handles Button1.Click
        MsgBox("こんにちは")
    End Sub
End Class
```

「デバッグの開始」で実行し、ボタンをクリックしてください。メッセージボックスが表示されることを確認してください。

コントロールをダブルクリックすると自動生成されるイベントは、そのコントロールで最も一般的なイベントですが、それ以外のイベントでプロシージャを作成したい場合はウィンドウ右上から選択できます。



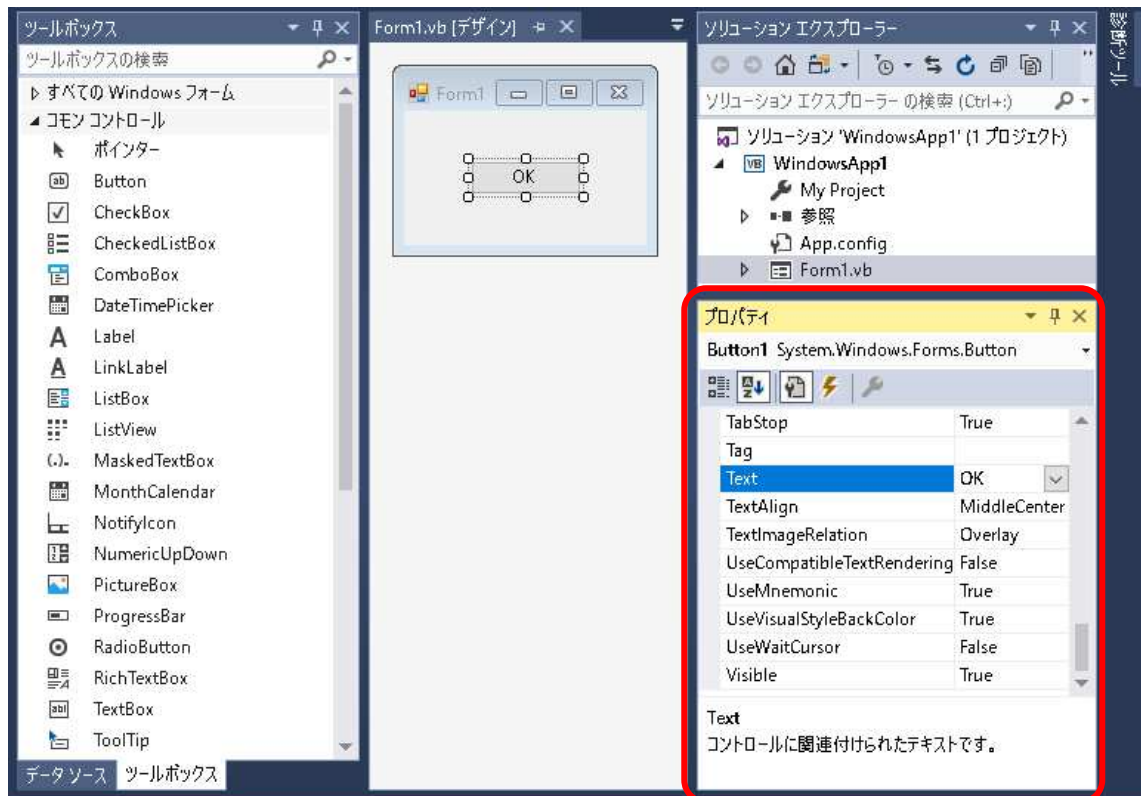
また、下図の部分ではフォーム上の別のコントロール（部品）を選択でき、対象を変更できます。



コントロールのプロパティを設定する

ツールボックスから配置したコントロールは、画面右下のプロパティウィンドウで様々な項目の設定が変更できます。

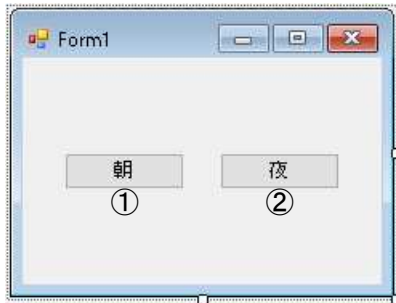
すべての項目は取り上げませんが、プロパティを選択している状態で F1 キーを押すとブラウザが起動し、公式のヘルプが表示されますので、調べてみてください。



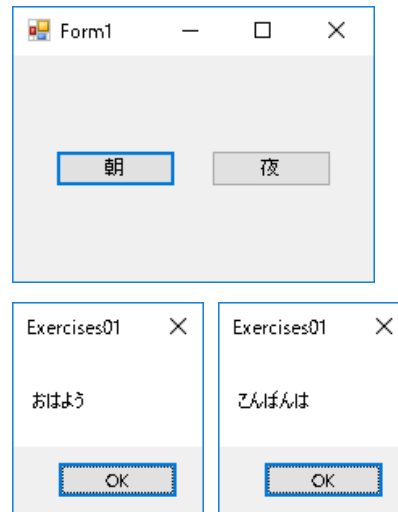
コントロール上に表示されている文字列は基本的に Text プロパティで設定できます。

問題 1

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	Button	Text	朝
②	Button	Text	夜

コントロール	①	イベント	Click
MsgBox () 関数を利用してメッセージボックスで「おはよう」と表示させる。			
コントロール	②	イベント	Click
MsgBox () 関数を利用してメッセージボックスで「こんばんは」と表示させる。			

課題 2（変数）

フォームから値を取得する方法とフォームへ値を出力する方法を確認します。

Button

先の課題に登場した Button を今後も利用します。Button はフォームアプリケーションで最も重要なコントロール（部品）です。フォームアプリケーション以外でも、日常的に様々なところで、同じような挙動をするものに触れていると思います。

Label

フォーム上に文字列を表示させる場合は Label を利用します。Label はユーザーが見るためのコントロールで、編集するためのコントロールではありません。プログラムで文字列を変更することはできますので、処理の結果を表示する場合に利用されることが多いです。また、入力のガイドのための文字列を配置する場合にも利用します。

TextBox

TextBox はユーザーに何かを入力させる場合に利用します。入力内容によって大きさを変更するなどして、ユーザーが入力しやすいように配慮するとよいでしょう。

基本的には改行を必要としない 1 行のデータを入力させるために利用されますが、文章の入力を求めたい場合は Multiline プロパティを True にすると複数行の入力が可能になります。デフォルトでは False に設定されています。

他にも下記のようなプロパティが備わっています。

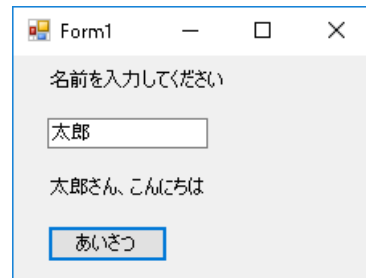
CharacterCasing	入力した文字を大文字にするか、小文字にするか、そのままにするかを選べます。製品コードなど、大文字や小文字のルールが決まっている項目を入力する場合に利用されます。
MaxLength	入力可能な最大文字数を制限できます。
PasswordChar	入力された文字列を指定した文字に置き換えて表示させます。一般的には「*」アスタリスクを設定することが多いです。表示は置き換わりますが、内部的には入力した値を保持しています。
ReadOnly	読み取り専用を設定できます。文字列の編集はできませんが、文字列を選択してコピーできます。切り取ったり張り付けたりして内容を変更することはできません。

サンプル 2

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	名前を入力してください
②	TextBox	—	—
③	Label	—	—
④	Button	Text	あいさつ

コントロール	④	イベント	Click
②に入力された文字と「さん、こんにちは」を文字列結合して③に表示させる。			

```
Public Class Form1

    Private Sub Button1_Click(sender As Object,
                               e As EventArgs) Handles Button1.Click

        Dim name As String

        name = TextBox1.Text
        Label2.Text = name & "さん、こんにちは"

    End Sub

End Class
```

基本的なコントロールの利用方法

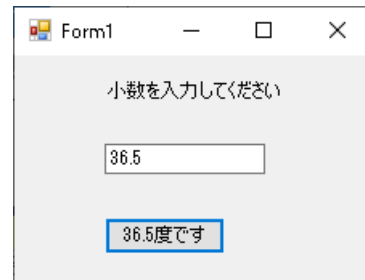
```
コントロール名. プロパティ名 = 設定する値
変数 = コントロール名. プロパティ名
```

問題 2

【デザイン例】



【実行結果例】



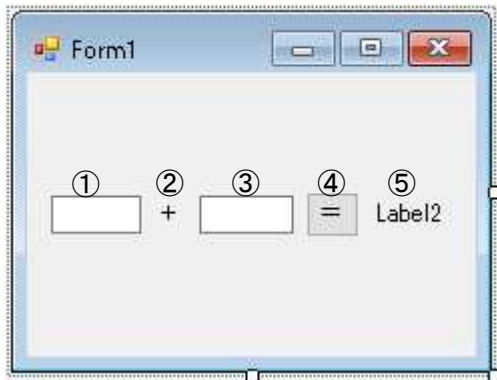
番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	小数を入力してください
②	TextBox	—	—
③	Button	Text	OK

コントロール	③	イベント	Click
②に入力された小数と「度です」を文字列結合して③に表示させる。			

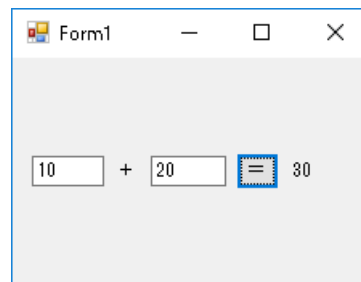
課題 3（四則演算）

サンプル 3

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	TextBox	(Name)	txtNum1
②	Label	Text	+
③	TextBox	(Name)	txtNum2
④	Button	(Name)	btnCalc
		Text	=
⑤	Label	(Name)	lblAns

コントロール	④	イベント	Click
①に入力された数と③に入力された数を一度数値変換し、足し算した結果を⑤へ表示させる。【不正な入力は考慮しない】			

```
Private Sub btnCalc_Click(sender As Object,  
                           e As EventArgs) Handles btnCalc.Click  
    Dim num1 As Integer  
    Dim num2 As Integer  
    num1 = Integer.Parse( txtNum1.Text )  
    num2 = Integer.Parse( txtNum2.Text )  
    lblAns.Text = num1 + num2  
End Sub
```

コントロールに名前を付けることで、どの部品に対して何を行っているのかが分かりやすくなりました。デザイン例の②のような、プログラムで利用しない表示専用のコントロールは名前を変更しないことが多いです。

また、イベントプロシージャの行に警告が発生していますが、無視して進めます。この警告は、プロシージャの名前が大文字から始まっていないために表示されています。イベントプロシージャの名前はボタンの名前とイベントの種類の名前で自動生成されますので、コントロールの名前の先頭を小文字で設定した場合、プロシージャの名前も先頭が小文字になってしまいます。

将来的にはコントロールの名前の先頭を大文字にすることが一般的になるかもしれませんが、本テキストでは、今まで一般的に使われてきたように名前の先頭は小文字にします。

問題 3

【デザイン例】

【実行結果例】

番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	単価
②	Label	Text	数量
③	Label	Text	人数
④	TextBox	(Name)	txtTanka
⑤	TextBox	(Name)	txtSuu
⑥	TextBox	(Name)	txtNin
⑦	Button	(Name)	btnCalc
		Text	計算
⑧	Label	(Name)	lblKotae
		Text	一人当たり
⑨	Label	(Name)	lblHasuu
		Text	端数

コントロール	⑦	イベント	Click
④と⑤を掛け算し、その結果を⑥で割ったときの答えを⑧に、余りを⑨に表示させる。 【不正な入力は考慮しない】			

問題 4

【デザイン例】

【実行結果例】

番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	価格
②	Label	Text	税(%)
③	TextBox	(Name)	txtPrice
④	TextBox	(Name)	txtTax
⑤	Button	(Name)	btnCalc
		Text	計算
⑥	Label	(Name)	lblAns
		Text	計算結果

コントロール	⑤	イベント	Click
③に入力された数と④に入力された数を一度数値変換し、足し算した結果を⑥へ表示させる。小数点以下は切り捨てとする。切り捨ての計算は Math.Truncate() を利用すること。【不正な入力は考慮しない】			

使用例

Dim num As Integer = Math.Truncate(12.34)

上記の結果、変数 num には 12 が代入される。

課題 4（条件分岐）

Checkbox

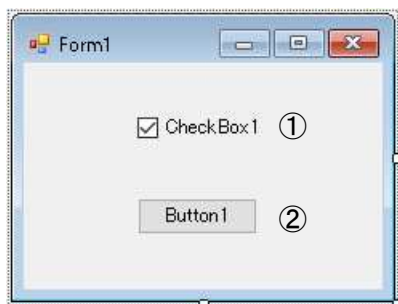
チェックボックスはそれぞれ独立した項目をユーザーに ON/OFF で指定させるためのコントロールです。

主なプロパティ	
Enabled	コントロールの有効／無効を設定できます。True で有効、False で無効になります。無効の場合は表示されますが操作できない状態になります。
Checked	チェックの状態を設定および取得できます。チェックが入っている場合は True、チェックが入っていない場合は False です。
主なイベントプロシージャ	
CheckedChanged	チェックの状態が変わったときに発生するイベントです。

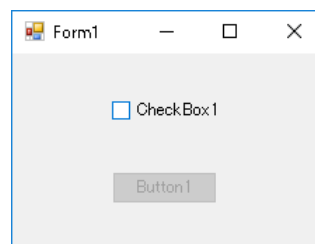
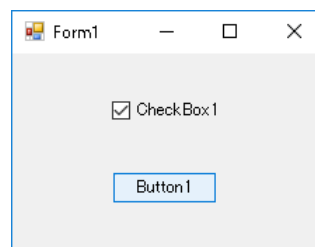
サンプル 4

チェックボックスの CheckedChanged イベントはコントロールをダブルクリックすることでソースコードエディタ上へ自動生成されます。また、コントロールの有効／無効を管理するプロパティは Enabled で、True で有効、False で無効になります。

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	CheckBox	Checked	True
②	Button	—	—

コントロール	①	イベント	CheckedChanged
チェックが ON のとき、②を有効にする。チェックが OFF のとき、②を無効にする。			
コントロール	②	イベント	Click
メッセージボックスで「Hello」と表示させる。			

```
Private Sub Button1_Click(sender As Object,  
                           e As EventArgs) Handles Button1.Click  
    MsgBox("Hello")  
  
End Sub  
  
Private Sub CheckBox1_CheckedChanged(sender As Object,  
                                     e As EventArgs) Handles CheckBox1.CheckedChanged  
  
    If CheckBox1.Checked = True Then  
        Button1.Enabled = True  
    Else  
        Button1.Enabled = False  
    End If  
  
End Sub
```

問題 5

【デザイン例】

【実行結果例】

番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	価格
②	Label	Text	税(%)
③	Label	Text	割引(%)
④	TextBox	(Name)	txtPrice
⑤	TextBox	(Name)	txtTax
⑥	TextBox	(Name)	txtDiscount
⑦	CheckBox	(Name)	chkDiscount
		Checked	True
		Text	割引
⑧	Button	(Name)	btnCalc
		Text	計算
⑨	Label	(Name)	lblAns
		Text	計算結果

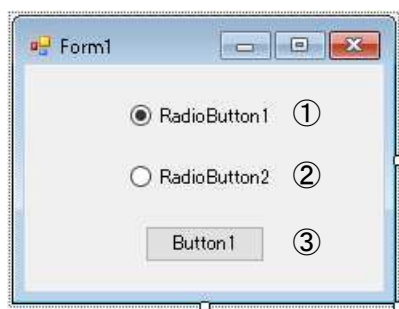
コントロール	⑦	イベント	CheckedChanged
チェックが ON のとき、⑥を有効にする。 チェックが OFF のとき、⑥を無効にする。			
コントロール	⑧	イベント	Click
価格④に対して税⑤をかけ、⑥の分だけ割り引いた結果を⑨へ表示させる。 ただし、⑦がチェックされていない場合は割引しない。【不正な入力は考慮しない】			

RadioButton

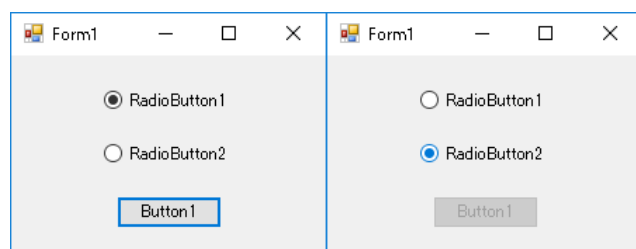
ラジオボタンはいくつかの項目のうち、ユーザーにひとつ指定させるためのコントロールです。RadioButton のデフォルトのイベントは CheckedChanged イベントです。これは「Click」ではないので、そのコントロール自身がクリックされなくても状態が変化すればイベントは発生します。

サンプル 5

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	RadioButton	Checked	True
②	RadioButton	—	—
③	Button	—	—

コントロール	①	イベント	CheckedChanged
チェックが ON のとき、③を有効にする。 チェックが OFF のとき、③を無効にする。			
コントロール	③	イベント	Click
メッセージボックスで「Hello」と表示させる。			

```
Private Sub RadioButton1_CheckedChanged(sender As Object,
                                         e As EventArgs) Handles RadioButton1.CheckedChanged

    Button1.Enabled = RadioButton1.Checked

End Sub

Private Sub Button1_Click(sender As Object,
                           e As EventArgs) Handles Button1.Click

    MsgBox("Hello")

End Sub
```

問題 6

【デザイン例】

Design view of Form1 showing the layout of controls with numbered callouts 1 through 11. The controls include labels for price, tax, discount, and calculation results, along with text boxes for input, checkboxes for discount, and radio buttons for tax calculation.

【実行結果例】

Four screenshots of Form1 showing the execution results for different input values and tax calculation options. The results are displayed in the '計算結果' (Calculation Result) label.

- Input: Price=1000, Tax=10, Discount=50, Discount checked, Tax checked. Result: 550.
- Input: Price=1000, Tax=10, Discount=50, Discount checked, Tax unchecked. Result: 500.
- Input: Price=1000, Tax=10, Discount=50, Discount unchecked, Tax checked. Result: 1100.
- Input: Price=1000, Tax=10, Discount=50, Discount unchecked, Tax unchecked. Result: 1000.

番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	価格
②	Label	Text	税 (%)
③	Label	Text	割引 (%)
④	TextBox	(Name)	txtPrice
⑤	TextBox	(Name)	txtTax
⑥	TextBox	(Name)	txtDiscount
⑦	CheckBox	(Name)	chkDiscount
		Checked	True
		Text	割引
⑧	Button	(Name)	btnCalc
		Text	計算
⑨	Label	(Name)	lblAns
		Text	計算結果
⑩	RadioButton	(Name)	radTax
		Checked	True
		Text	税込
⑪	RadioButton	Text	税抜

灰色の部分は前回から変更がない項目です。

コントロール	⑦	イベント	CheckedChanged
チェックが ON のとき、⑥を有効にする。 チェックが OFF のとき、⑥を無効にする。			
コントロール	⑧	イベント	Click
価格④に対して税⑤をかけ、⑥の分だけ割り引いた結果を⑨へ表示させる。 ただし、⑦がチェックされていない場合は割引しない。 さらに、ラジオボタンの状態によって税を含めるか含めないかを分けること。 各テキストボックスへの入力はすべて整数とする。【不正な入力は考慮しない】			

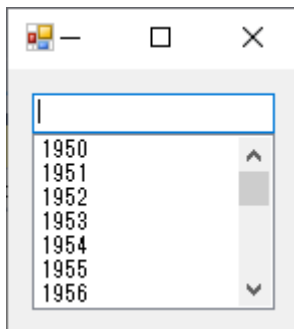
課題 5（繰り返し）

ComboBox

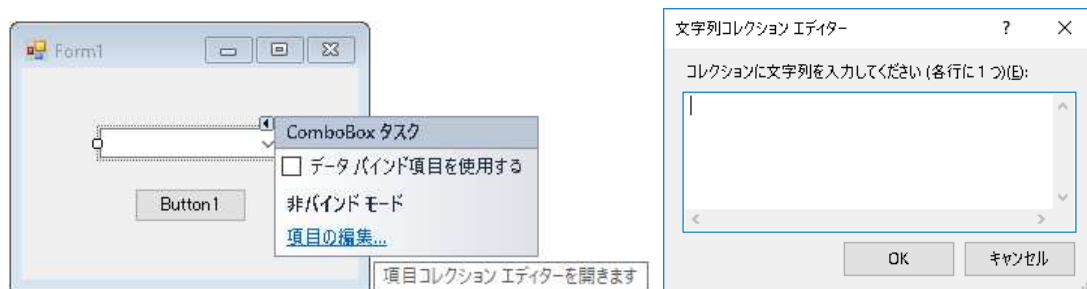
コンボボックスはいくつかの項目のうち、ひとつを指定させるためのコントロールです。ラジオボタンと同じような選択方法ですが、選択の候補が多かったり、順番になっていたりする場合はコンボボックスを利用します。アンケートなど、画面上に表示させたすべての候補の中から選ばせたいときや、ユーザーになるべく負荷をかけないようにする場合はラジオボタンが利用されます。（ラジオボタンは1クリックで選択できますが、コンボボックスは2クリックです）

コンボボックスの選択されている内容を取得するには Text プロパティを使用します。選択されている内容が、候補のうち何番目であるかを知るには SelectedIndex プロパティを使用します。SelectedIndex プロパティは最初の候補を 0 として、その後 1 ずつ増加して行きます。選択されていない場合は -1 です。

DropDownStyle プロパティでコンボボックスそのものの動作を変更できます。デフォルトで設定されている「DropDown」はコンボボックスコントロール内にテキストを入力できます。「DropDownList」は候補の中から選択するのみで、テキストを入力することはできません。「Simple」は下図のように、候補を折りたたまずに常に表示させる形式です。コントロールのサイズは候補の表示を含めた高さに設定します。



コンボボックスの選択の候補はあらかじめ準備しておく必要があります。下図のように、コンボボックスを配置した後に▶をクリックすると選択候補の設定が表示され、「項目の編集」をクリックすると、メモ帳のように1行が1候補となるような「文字列コレクションエディター」が表示されます。これに記入してもよいですが、連番などの場合はプログラムで作成したほうが良い場合もあります。



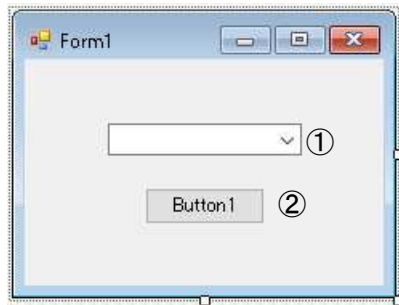
フォームの起動時にコントロールへ何かしらの設定をさせたい場合はフォームの Load イベントを利用します。このイベントはフォームが表示される直前に発生します。

ボタンコントロールをダブルクリックしてクリックイベントを自動作成させたように、フォーム上の何もコントロールが置いていない、フォームの土台のところでダブルクリックしてください。Load イベントのプロシージャが自動作成されます。

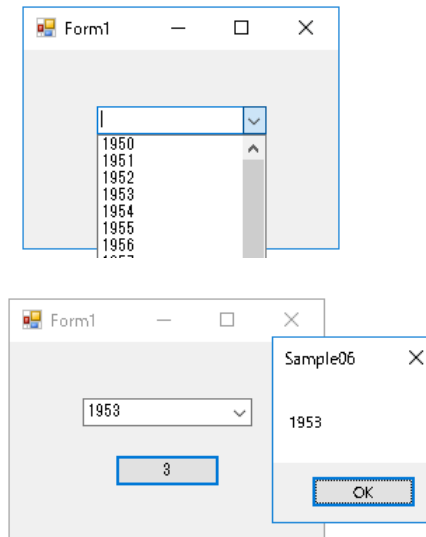
```
Private Sub Form1_Load(sender As Object,  
                        e As EventArgs) Handles MyBase.Load  
  
End Sub
```

サンプル 6

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	ComboBox	—	—
②	Button	—	—

コントロール	Form	イベント	Load
①の選択候補を 1950～2020 の数値とする			
コントロール	②	イベント	Click
選択した内容をメッセージボックスで表示する。			
選択した項目番号をボタンに表示する。			

```

Private Sub Form1_Load(sender As Object,
                        e As EventArgs) Handles MyBase.Load

    For cnt As Integer = 1950 To 2020
        ComboBox1.Items.Add( cnt )
    Next

End Sub

Private Sub Button1_Click(sender As Object,
                          e As EventArgs) Handles Button1.Click

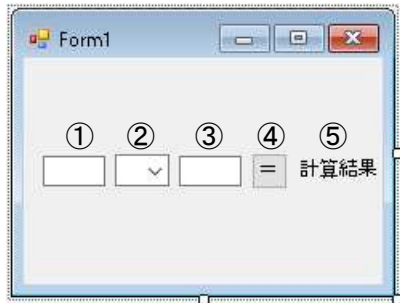
    Button1.Text = ComboBox1.SelectedIndex
    MsgBox( ComboBox1.Text )

End Sub

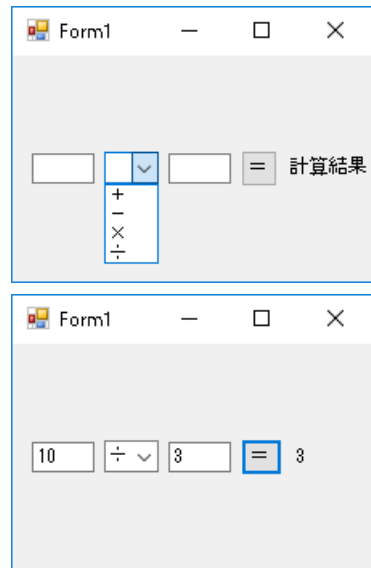
```

問題 7

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	TextBox	(Name)	txtNum1
②	ComboBox	(Name)	cmbCalc
③	TextBox	(Name)	txtNum2
④	Button	(Name)	btnCalc
		Text	=
⑤	Label	(Name)	lblAns
		Text	計算結果

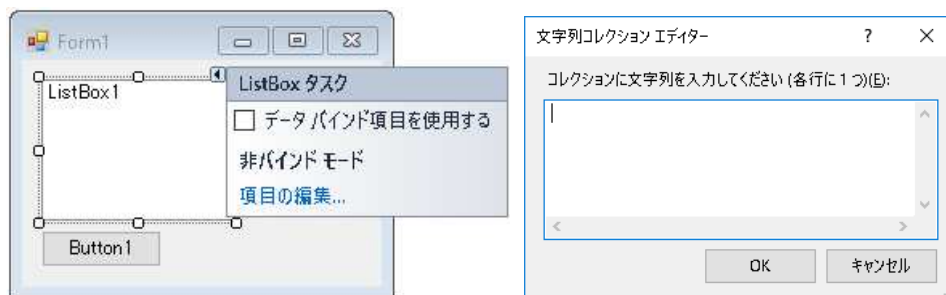
コントロール	Form	イベント	Load
②の候補を +, -, ×, ÷ の記号を選択候補とする。候補はひとつずつ登録する。			
コントロール	④	イベント	Click
①に入力された数と③に入力された数を一度数値変換し、②で選択された計算をした結果を⑤へ表示させる。ただし、割り算は答え（商）のみを求めるものとする。 【不正な入力は考慮しない】			

課題 6（配列）

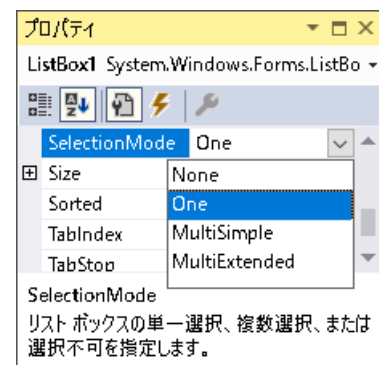
ListBox

リストボックスはいくつかの項目のうち、ユーザーにひとつまたは複数の項目を指定させるためのコントロールです。コンボボックスとチェックボックスの機能を合わせたような選択方法です。選択の候補が多かったり、順番になっていたりする場合はリストボックスを利用します。候補が多い場合はリストボックス内にスクロールバーが表示されます。

コンボボックスと同様、リストボックスの選択の候補もあらかじめ準備しておく必要があります。追加する方法もコンボボックスと同様、「文字列コレクションエディター」でも、プログラムでも追加可能です。



リストボックスで複数の項目を同時に選択させるにはプロパティの SelectionMode を変更します。プログラムでも変更可能です。MultiSimple と MultiExtended のどちらも複数選択が可能です。挙動が異なります。MultiExtended はマウスのドラッグで候補を範囲選択可能ですが、MultiSimple はクリックで一つずつしか選択できません。One の場合は一つだけ選択可能です。None の場合は選択できません。



None は主に複数行の文字の羅列を表示物として扱いたいときに使用します。テキストボックスの Multiline プロパティを True にしても同様の表示は可能ですが、テキストボックスは文字の編集が主な機能なので、ReadOnly プロパティを True にしても文字列への操作は可能で、コピーなどの操作が可能です。

選択項目が単一である場合は SelectedItem プロパティや Text プロパティでその項目を扱えます。

複数選択した場合の項目は SelectedItems プロパティで扱えます。SelectedItems プロパティ配列のような状態で、選択された情報がまとめられています。

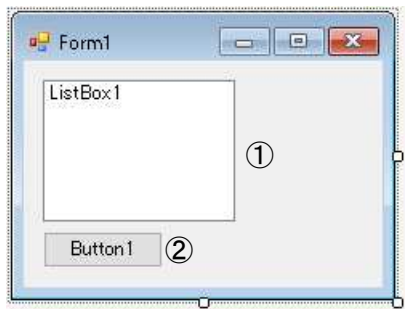
MessageBox.show(ListBox1.SelectedItems(1)) のように個別にアクセス可能です。

選択されている要素の数を取得する場合は SelectedItems プロパティ内の Count プロパティを利用します。

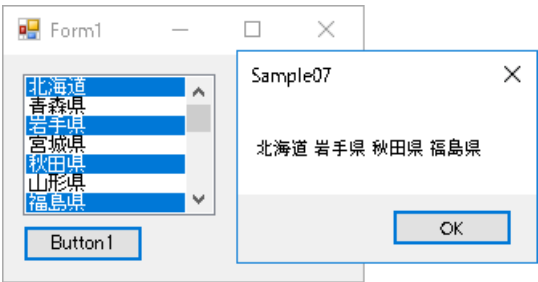
MessageBox.show(ListBox1.SelectedItems.Count) のように要素数を取得可能です。

サンプル 7

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	ListBox	—	—
②	Button	—	—

コントロール	Form	イベント	Load
①候補を都道府県名の一覧とする。都道府県名は配列で定義し、①への登録は繰り返しを利用する。			
コントロール	②	イベント	Click
①で選択された県名をメッセージボックスで表示させる。			

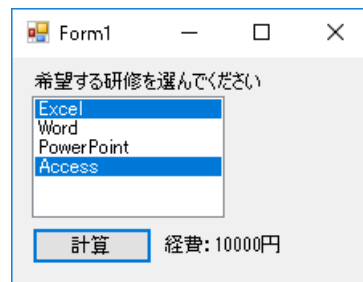
```
Private Sub Form1_Load(sender As Object,  
    e As EventArgs) Handles MyBase.Load  
  
    Dim tdfk() As String = {"北海道", "青森県", "岩手県", "宮城県",  
        "秋田県", "山形県", "福島県", "茨城県", "栃木県", "群馬県",  
        "埼玉県", "千葉県", "東京都", "神奈川県", "新潟県", "富山県",  
        "石川県", "福井県", "山梨県", "長野県", "岐阜県", "静岡県",  
        "愛知県", "三重県", "滋賀県", "京都府", "大阪府", "兵庫県",  
        "奈良県", "和歌山県", "鳥取県", "島根県", "岡山県", "広島県",  
        "山口県", "徳島県", "香川県", "愛媛県", "高知県", "福岡県",  
        "佐賀県", "長崎県", "熊本県", "大分県", "宮崎県", "鹿児島県",  
        "沖縄県"}  
  
    For Each ken As String In tdfk  
        ListBox1.Items.Add(ken)  
    Next  
  
    ListBox1.SelectionMode = SelectionMode.MultiSimple  
  
End Sub  
  
Private Sub Button1_Click(sender As Object,  
    e As EventArgs) Handles Button1.Click  
  
    Dim str As String = ""  
  
    For Each itm As String In ListBox1.SelectedItems  
        str &= itm & " "  
    Next  
  
    MsgBox(str)  
  
End Sub
```

問題 8

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	希望する研修を選んでください
②	ListBox	(Name)	lstOffice
③	Button	(Name)	btnCalc
		Text	計算
④	Label	(Name)	lblAns
		Text	経費 :

コントロール	Form	イベント	Load
②候補を“Excel”, “Word”, “PowerPoint”, “Access”とする。候補名は配列で定義し、②への登録は繰り返しを利用する。また、②は複数選択を可能とする。			
コントロール	③	イベント	Click
②で選択された項目数を得て、経費を計算して④へ表示させる。ただし、経費は1項目につき 5000 円とする。			

課題 7（関数）

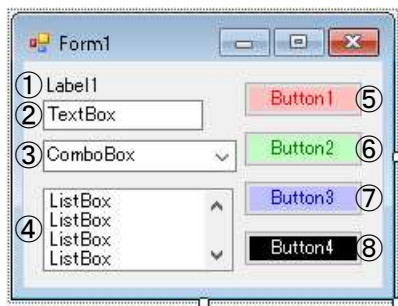
自作プロシージャ

フォームアプリケーションにおいて、自作メソッドを利用するのは、行う処理が似ているものをまとめるときが多いです。

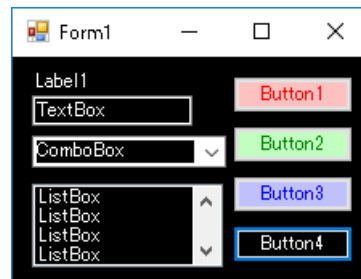
プロシージャを作成するときのコツは、似ている処理の中で、どこが違うのかを明確に把握することです。違う部分を変数にしておき、その変数への代入はプロシージャを呼び出す側が決めるようにしておけばよいのです。

サンプル 8

【デザイン例】



【実行結果例】



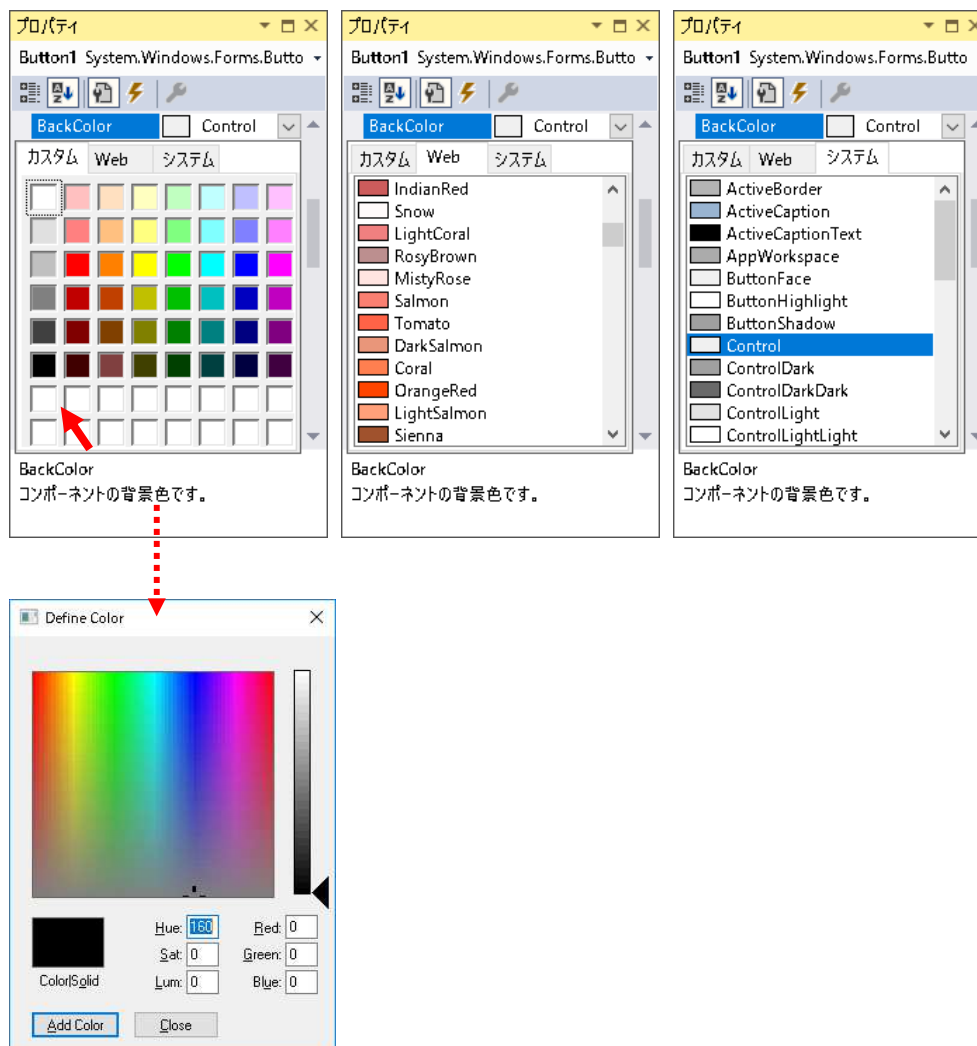
番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	(任意)
②	TextBox	Text	(任意)
③	ComboBox	(選択候補)	(任意)
④	ListBox	(選択候補)	(任意)
⑤	Button	BackColor	(任意)
		ForeColor	(任意)
⑥	Button	BackColor	(任意)
		ForeColor	(任意)
⑦	Button	BackColor	(任意)
		ForeColor	(任意)
⑧	Button	BackColor	(任意)
		ForeColor	(任意)

⑤、⑥、⑦、⑧の BackColor および ForeColor は同じ色にならないように設定してください。

コントロール	⑤	イベント	Click
⑤の BackColor を第一引数に、⑤の ForeColor を第二引数にして SetColor() メソッドを呼び出す。			
コントロール	⑥	イベント	Click
⑥の BackColor を第一引数に、⑥の ForeColor を第二引数にして SetColor() メソッドを呼び出す。			
コントロール	⑦	イベント	Click
⑦の BackColor を第一引数に、⑦の ForeColor を第二引数にして SetColor() メソッドを呼び出す。			
コントロール	⑧	イベント	Click
⑧の BackColor を第一引数に、⑧の ForeColor を第二引数にして SetColor() メソッドを呼び出す。			

プロシージャ	Sub	アクセス修飾子	Private
名前	SetColor	戻り値の型	なし
引数	Color 型 , Color 型		
<p>第一引数に指定された Color 型のデータを②, ③, ④, Form の BackColor へ設定する。</p> <p>第二引数に指定された Color 型のデータを②, ③, ④, Form の ForeColor へ設定する。</p> <p>Label, CheckBox, RadioButton は土台となるコントロール（フォーム）の BackColor と ForeColor を自動的に受け継ぐので、設定の必要はない。</p>			

BackColor および ForeColor はプログラムでも変更可能ですが、今回は視覚的に確認しながら設定してみましょう。下図のようにプロパティウィンドウから3種類の選択の方法があります。「システム」はWindowsの設定によるものです。「Web」は色を名前で選択します。「カスタム」は用意済みの色でも設定可能ですが、カラーパレットの色が設定されていない下2段を右クリックすることで Define Color ウィンドウが表示され、中間色を自分で作成できます。



この設定は Button 特有のものではなく、色が設定可能なコントロールは基本的に同じ操作で変更できます。

```
Private Sub SetColor(bcolor As Color, fcolor As Color)
```

```
    TextBox1.BackColor = bcolor  
    TextBox1.ForeColor = fcolor  
    ComboBox1.BackColor = bcolor  
    ComboBox1.ForeColor = fcolor  
    ListBox1.BackColor = bcolor  
    ListBox1.ForeColor = fcolor  
    Me.BackColor = bcolor  
    Me.ForeColor = fcolor
```

Label, CheckBox, RadioButton は
土台となるコントロール（フォー
ム）の BackColor と ForeColor を
自動的に受け継ぎます

```
End Sub
```

Form 自体を指定する場合は Me とします

```
Private Sub Button1_Click(sender As Object,  
                           e As EventArgs) Handles Button1.Click
```

```
    SetColor(Button1.BackColor, Button1.ForeColor)
```

```
End Sub
```

```
Private Sub Button2_Click(sender As Object,  
                           e As EventArgs) Handles Button2.Click
```

```
    SetColor(Button2.BackColor, Button2.ForeColor)
```

```
End Sub
```

```
Private Sub Button3_Click(sender As Object,  
                           e As EventArgs) Handles Button3.Click
```

```
    SetColor(Button3.BackColor, Button3.ForeColor)
```

```
End Sub
```

```
Private Sub Button4_Click(sender As Object,  
                           e As EventArgs) Handles Button4.Click
```

```
    SetColor(Button4.BackColor, Button4.ForeColor)
```

```
End Sub
```

問題 9

【デザイン例】

【実行結果例】

番号	コントロール名	プロパティ名	プロパティ値
①	Button	Text	名前
②	Button	Text	出身地
③	Button	Text	趣味

コントロール	①	イベント	Click
「名前」を引数にして自作メソッド「inputData()」を呼び出す			
コントロール	②	イベント	Click
「出身地」を引数にして自作メソッド「inputData()」を呼び出す			
コントロール	③	イベント	Click
「趣味」を引数にして自作メソッド「inputData()」を呼び出す			

使用例

Dim str As String = InputBox(表示文字列)

処理：ユーザーに入力を求めるフォームを表示する。入力されたデータは String として返される

プロシージャ	Sub	アクセス修飾子	Private
名前	InputData	戻り値の型	なし
引数	String 型		

引数で受け取った文字列を含むメッセージでインプットボックスを表示させる。入力された内容を引数として CheckData() 関数へ渡す。その結果が True の場合はメッセージボックスに入力した内容を表示させ、False の場合は「入力が不正です」と表示させる。

プロシージャ	Function	アクセス修飾子	Private
名前	CheckData	戻り値の型	Boolean
引数	String 型		

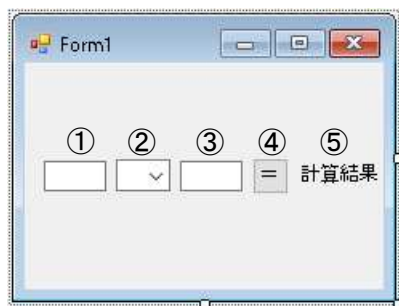
引数で受け取った文字列が空文字の場合は False を返す。それ以外の場合は True を返す。

サンプル 9

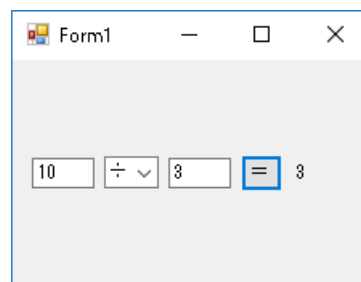
自作メソッドは、何度か利用する処理や似たような処理をまとめるだけではなく、一度しか利用されないとしても、処理をまとめてわかりやすくする場合にも利用されます。また、機能追加をする場合などは行の間にプログラムを何行も割り込ませるのではなく、自作メソッドの呼び出しを利用して、既存のプログラムになるべく影響を与えないようにします。

コンボボックスを利用した課題に機能を追加します。

【デザイン例】



【実行結果例】



番号	コントロール名	プロパティ名	プロパティ値
①	TextBox	(Name)	txtNum1
②	ComboBox	(Name)	cmbCalc
③	TextBox	(Name)	txtNum2
④	Button	(Name)	btnCalc
		Text	=
⑤	Label	(Name)	lblAns
		Text	計算結果

コントロール	Form	イベント	Load
②の候補を + , - , × , ÷ の記号を選択候補とする。候補はひとつずつ登録する。			
コントロール	④	イベント	Click
①に入力された数と③に入力された数を一度数値変換し、②で選択された計算をした結果を⑤へ表示させる。ただし、割り算は答え（商）のみを求めるものとする。			
ただし、入力に不備がある場合はメッセージボックスで入力を促し、計算はしない。数字の入力内容は Integer.TryParse() を利用してチェックし、演算記号は選択されているかどうかでチェックすること。			

使用例

Dim bool As Boolean= Integer.TryParse(data , num)

処理：第一引数のデータが第二引数に指定された変数へ代入できる場合は True を返し、代入できない場合は False を返す。変換できる場合、変数 num には変換後の値が格納される。変換できない場合は 0 が代入される。

第一引数：チェックする対象となるデータ

第二引数：第一引数のデータの代入を試みる変数


```

Private Sub Form1_Load(sender As Object,
                        e As EventArgs) Handles MyBase.Load

    cmbCalc.Items.Add("+")
    cmbCalc.Items.Add("-")
    cmbCalc.Items.Add("×")
    cmbCalc.Items.Add("÷")

End Sub

Private Function inputCheck(num1 As String,
                            symbol As String, num2 As String) As Boolean

    Dim num As Integer

    If Integer.TryParse(num1, num) = False Then
        MsgBox("1つ目の値の入力が不正です")
        Return False
    ElseIf Integer.TryParse(num2, num) = False Then
        MsgBox("2つ目の値の入力が不正です")
        Return False
    ElseIf symbol = "" Then
        MsgBox("計算記号の入力が不正です")
        Return False
    End If

    Return True

End Function

Private Sub btnCalc_Click(sender As Object,
                          e As EventArgs) Handles btnCalc.Click

    If inputCheck(txtNum1.Text, cmbCalc.Text, txtNum2.Text) = False Then
        Exit Sub
    End If

    Dim num1 As Integer = Integer.Parse( txtNum1.Text )
    Dim num2 As Integer = Integer.Parse( txtNum2.Text )

    Select Case cmbCalc.SelectedIndex
        Case 0
            lblAns.Text = num1 + num2
        Case 1
            lblAns.Text = num1 - num2
        Case 2
            lblAns.Text = num1 * num2
        Case 3
            lblAns.Text = num1 ÷ num2
    End Select

End Sub

```

灰色の部分は前回から変更がない項目です。

問題 10

以前の割り勘の問題に、入力内容をチェックする処理を追加しましょう。

【デザイン例】

Form1

単価 ① ④

数量 ② ⑤

人数 ③ ⑥

計算 ⑦ 一人当たり ⑧

端数 ⑨

【実行結果例】

Form1

単価

数量

人数

計算 割り勘 3,333円

端数 1円

番号	コントロール名	プロパティ名	プロパティ値
①	Label	Text	単価
②	Label	Text	数量
③	Label	Text	人数
④	TextBox	(Name)	txtTanka
⑤	TextBox	(Name)	txtSuu
⑥	TextBox	(Name)	txtNin
⑦	Button	(Name)	btnCalc
		Text	計算
⑧	Label	(Name)	lblKotae
		Text	一人当たり
⑨	Label	(Name)	lblHasuu
		Text	端数

灰色の部分は前回から変更がない項目です。

プロシージャ	Function	アクセス修飾子	Private
名前	CheckData	戻り値の型	Boolean
引数	String 型		
引数で受け取った文字列が、1 以上の数値に変換できる場合は True を返し、そうでない場合は False を返す。			

コントロール	⑦	イベント	Click
④、⑤、⑥の入力内容を CheckData() メソッドを利用してチェックする。3 箇所の入力内容が不正な場合はメッセージボックスでその旨を知らせる。不正な入力がない場合は④と⑤を掛け算し、その結果を⑥で割ったときの答えを⑧に、余りを⑨に表示させる。			

課題解答例

【 問題 1 】

```
Public Class Form1

    Private Sub Button1_Click(sender As Object,
                                e As EventArgs) Handles Button1.Click

        MsgBox("おはよう")

    End Sub

    Private Sub Button2_Click(sender As Object,
                                e As EventArgs) Handles Button2.Click

        MsgBox("こんばんは")

    End Sub

End Class
```

【 問題 2 】

```
Public Class Form1

    Private Sub Button1_Click(sender As Object,
                                e As EventArgs) Handles Button1.Click

        Dim num As Integer

        num = Integer.Parse( TextBox1.Text )
        Button1.Text = num * 2

    End Sub

End Class
```

【 問題 3 】

```
Private Sub btnCalc_Click(sender As Object,  
                           e As EventArgs) Handles btnCalc.Click  
  
    Dim tanka As Integer  
    Dim suu As Integer  
    Dim nin As Integer  
  
    tanka = Integer.Parse( txtTanka.Text )  
    suu = Integer.Parse( txtSuu.Text )  
    nin = Integer.Parse( txtNin.Text )  
  
    lblKotae.Text = "割り勘 " & Format(tanka * suu ¥ nin, "#,###") & "円"  
    lblHasuu.Text = "端 数 " & Format(tanka * suu Mod nin, "#,###") & "円"  
  
End Sub
```

【 問題 4 】

```
Private Sub btnCalc_Click(sender As Object,  
                           e As EventArgs) Handles btnCalc.Click  
  
    Dim price As Integer = Integer.Parse( txtPrice.Text )  
    Dim tax As Double = Integer.Parse( txtTax.Text ) / 100  
  
    lblAns.Text = Math.Truncate( price * (1 + tax) )  
  
End Sub
```

【 問題 5 】

```
Private Sub chkDiscount_CheckedChanged(sender As Object,  
                                       e As EventArgs) Handles chkDiscount.CheckedChanged  
  
    txtDiscount.Enabled = chkDiscount.Checked  
  
End Sub  
  
Private Sub btnCalc_Click(sender As Object,  
                           e As EventArgs) Handles btnCalc.Click  
  
    Dim price As Integer = Integer.Parse( txtPrice.Text )  
    Dim tax As Double = Integer.Parse( txtTax.Text ) / 100  
    Dim discount As Double  
  
    If chkDiscount.Checked = True Then  
        discount = Integer.Parse( txtDiscount.Text ) / 100  
        lblAns.Text = Math.Truncate(price * (1 + tax) * (1 - discount))  
    Else  
        lblAns.Text = Math.Truncate(price * (1 + tax))  
    End If  
  
End Sub
```

【 問題 6 】

```
Private Sub chkDiscount_CheckedChanged(sender As Object,  
                                     e As EventArgs) Handles chkDiscount.CheckedChanged
```

```
    txtDiscount.Enabled = chkDiscount.Checked
```

```
End Sub
```

```
Private Sub btnCalc_Click(sender As Object,  
                           e As EventArgs) Handles btnCalc.Click
```

```
    Dim price As Integer = Integer.Parse( txtPrice.Text )
```

```
    Dim tax As Double = 0
```

```
    Dim discount As Double = 0
```

```
    If chkDiscount.Checked = True Then
```

```
        discount = Integer.Parse( txtDiscount.Text ) / 100
```

```
    End If
```

```
    If radTax.Checked = True Then
```

```
        tax = Integer.Parse( txtTax.Text ) / 100
```

```
    End If
```

```
    lblAns.Text = Math.Truncate(price * (1 + tax) * (1 - discount))
```

```
End Sub
```

【 問題 7 】

```
Private Sub Form1_Load(sender As Object,  
                        e As EventArgs) Handles MyBase.Load  
  
    cmbCalc.Items.Add("+")  
    cmbCalc.Items.Add("-")  
    cmbCalc.Items.Add("×")  
    cmbCalc.Items.Add("÷")  
  
End Sub  
  
Private Sub btnCalc_Click(sender As Object,  
                           e As EventArgs) Handles btnCalc.Click  
    Dim num1 As Integer  
    Dim num2 As Integer  
  
    num1 = Integer.Parse( txtNum1.Text )  
    num2 = Integer.Parse( txtNum2.Text )  
  
    Select Case cmbCalc.SelectedIndex  
        Case 0  
            lblAns.Text = num1 + num2  
        Case 1  
            lblAns.Text = num1 - num2  
        Case 2  
            lblAns.Text = num1 * num2  
        Case 3  
            lblAns.Text = num1 ÷ num2  
    End Select  
  
End Sub
```

【 問題 8 】

```
Private Sub Form1_Load(sender As Object,  
                        e As EventArgs) Handles MyBase.Load  
  
    Dim office() As String = {"Excel", "Word", "PowerPoint", "Access"}  
  
    For Each ken As String In office  
        lstOffice.Items.Add(ken)  
    Next  
  
    lstOffice.SelectionMode = SelectionMode.MultiSimple  
  
End Sub  
  
Private Sub btnCalc_Click(sender As Object,  
                           e As EventArgs) Handles btnCalc.Click  
  
    Dim cnt As Integer = lstOffice.SelectedItems.Count  
  
    lblAns.Text = "経費：" & 5000 * cnt & "円"  
  
End Sub
```

【 問題 9 】

```
Private Function CheckData(str As String) As Boolean

    If str = "" Then
        Return False
    End If

    Return True

End Function

Private Sub InputData(item As String)

    Dim inp As String

    Do
        inp = InputBox(item & "を入力してください")

        If CheckData(inp) Then
            MsgBox(item & "を" & inp & "で登録しました")
            Exit Do
        Else
            MsgBox("入力が不正です")
        End If
    Loop

End Sub

Private Sub Button1_Click(sender As Object,
                           e As EventArgs) Handles Button1.Click

    InputData("名前")

End Sub

Private Sub Button2_Click(sender As Object,
                           e As EventArgs) Handles Button2.Click

    InputData("出身地")

End Sub

Private Sub Button3_Click(sender As Object,
                           e As EventArgs) Handles Button3.Click

    InputData("趣味")

End Sub
```

【 問題 10 】

```
Private Function CheckData(str As String) As Boolean

    Dim num As Integer

    If Integer.TryParse(str, num) Then
        If num > 0 Then
            Return True
        End If
    End If

    Return False

End Function

Private Sub btnCalc_Click(sender As Object,
                           e As EventArgs) Handles btnCalc.Click
    Dim tanka As Integer
    Dim suu As Integer
    Dim nin As Integer

    If CheckData(txtTanka.Text) And CheckData(txtSuu.Text) And
        CheckData(txtNin.Text) Then

        tanka = Integer.Parse( txtTanka.Text )
        suu = Integer.Parse( txtSuu.Text )
        nin = Integer.Parse( txtNin.Text )

        lblKotae.Text = "割り勘 " & Format(tanka * suu ¥ nin, "#,###") & "円"
        lblHasuu.Text = "端 数 " & Format(tanka * suu Mod nin, "#,###") & "円"

    End If

End Sub
```


最終課題

5つの文字列を管理するコンソールアプリケーションを作成してください。

「終了」が選択されるまで以下の機能の選択を繰り返してください。

機能名	機能内容
表示	現在登録されている文字列をすべて表示させる。
追加	追加する文字列を指定させる。すでに5つの文字列が追加されている場合はメッセージを表示して追加させない。
終了	上記機能の選択を終了する。

5つの文字列は配列で管理してください。

実行イメージは次のとおりです。

```
1:表示 2:追加 0:終了 2
追加する内容を入力してください
おはよう
```

```
1:表示 2:追加 0:終了 1
0: おはよう
```

(省略)

```
1:表示 2:追加 0:終了 1
0: おはよう
1: おやすみ
2: こんにちは
3:こんばんは
4: さようなら
```

```
1:表示 2:追加 0:終了 2
すでに5件登録済みです
```

```
1:表示 2:追加 0:終了 9
0~2を選択してください
```

```
1:表示 2:追加 0:終了 0
```

まずは、前述の機能が動作するプログラムを完成させましょう。完成後、余裕があれば下記の点を見直し、対応していないものに関しては改善してみましょう。

追加要件 1

下記のような変更の機能を追加しましょう。

機能名	機能内容
変更	変更する要素番号を指定させ、新たな文字列を入力させる。指定した要素番号に文字列が登録されていない場合はメッセージを表示する。

```

1:表示  2:追加  3:変更  0:終了  1
0 : おはよう
1 : おやすみ

1:表示  2:追加  3:変更  0:終了  3
変更する要素番号を入力してください
0
変更する内容を入力してください
ごきげんよう

1:表示  2:追加  3:変更  0:終了  1
0 : ごきげんよう
1 : おやすみ

1:表示  2:追加  3:変更  0:終了  3
変更する要素番号を入力してください
4
入力された番号の要素はありません

1:表示  2:追加  3:変更  0:終了  0

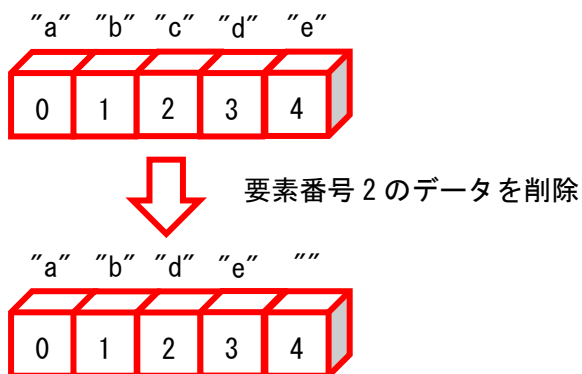
```

追加要件 2

下記のような削除の機能を追加しましょう。

機能名	機能内容
削除	削除する要素番号を指定させる。指定した要素番号に文字列が登録されていない場合はメッセージを表示する。

削除機能はデータを "" 空文字にするだけでなく、データを詰めてください。



```

1:表示 2:追加 3:変更 4:削除 0:終了 1
0: おはよう
1: おやすみ
2: こんにちは
3:こんばんは

1:表示 2:追加 3:変更 4:削除 0:終了 4
削除する要素番号を入力してください
2

1:表示 2:追加 3:変更 4:削除 0:終了 1
0: おはよう
1: おやすみ
2:こんばんは

1:表示 2:追加 3:変更 4:削除 0:終了 4
削除する要素番号を入力してください
4
入力された番号の要素はありません

1:表示 2:追加 3:変更 4:削除 0:終了 0
  
```

追加要件 3

- 追加、変更、削除、変更のそれぞれの動作をプロシージャで定義し、メインのプロシージャを簡潔にしましょう。
- 管理する文字列の数を変数で定義し、容易に変更できるようにしましょう。

模範解答例

```
Module Module1

    Sub Main()

        Dim arrayData(4) As String
        Dim cnt As Integer = 0
        Dim inpcmd As Integer

        Do
            Console.WriteLine("1:表示 2:追加 0:終了 ")
            inpcmd = Integer.Parse(Console.ReadLine())

            Select Case inpcmd
                Case 0
                    '何もしない
                Case 1
                    If cnt > 0 Then
                        For num As Integer = 0 To cnt - 1
                            Console.WriteLine(num & " : " & arrayData(num))
                        Next
                    End If
                Case 2
                    If cnt < 5 Then
                        Console.WriteLine("追加する内容を入力してください")
                        arrayData(cnt) = Console.ReadLine()
                        cnt += 1
                    Else
                        Console.WriteLine("すでに 5 件登録済みです")
                    End If
                Case Else
                    Console.WriteLine("0~2 を選択してください")
            End Select

            Console.WriteLine()

            Loop While inpcmd <> 0

        End Sub

    End Module
```

模範解答例（追加要件対応）

```

Module Module1

    Sub Main()

        Dim size As Integer = 5
        Dim arrayData(size - 1) As String
        Dim cnt As Integer = 0

        Dim inpcmd As Integer

        Do
            Console.WriteLine("1:表示 2:追加 3:変更 4:削除 0:終了 ")
            inpcmd = Integer.Parse(Console.ReadLine())

            Select Case inpcmd
                Case 0
                    '何もしない
                Case 1
                    CmdDsp(arrayData, cnt)
                Case 2
                    cnt = CmdAdd(arrayData, cnt, size)
                Case 3
                    CmdUpd(arrayData, cnt)
                Case 4
                    cnt = CmdDel(arrayData, cnt, size)
                Case Else
                    Console.WriteLine("0~4 を選択してください")
            End Select

            Console.WriteLine()

        Loop While inpcmd <> 0

    End Sub

    Function CmdAdd(list() As String,
                    cnt As Integer, size As Integer) As Integer

        If cnt < size Then
            Console.WriteLine("追加する内容を入力してください")
            list(cnt) = Console.ReadLine()
            cnt += 1
        Else
            Console.WriteLine("すでに" & size & "件登録済みです")
        End If

        Return cnt

    End Function

    Sub CmdUpd(list() As String, cnt As Integer)

        Dim inpidx As Integer

        Console.WriteLine("変更する要素番号を入力してください")
        inpidx = Integer.Parse(Console.ReadLine())

        If inpidx >= 0 And inpidx < cnt Then
            Console.WriteLine("変更する内容を入力してください")
            list(inpidx) = Console.ReadLine()
        Else
            Console.WriteLine("入力された番号の要素はありません")
        End If

    End Sub

End Module

```

```
End Sub

Function CmdDel(list() As String,
               cnt As Integer, size As Integer) As Integer

    Dim inpidx As Integer

    Console.WriteLine("削除する要素番号を入力してください")
    inpidx = Integer.Parse(Console.ReadLine())

    If inpidx >= 0 And inpidx < cnt Then
        For num As Integer = inpidx To (cnt - 1) - 1
            list(num) = list(num + 1)
        Next
        list(cnt - 1) = ""
        cnt -= 1
    Else
        Console.WriteLine("入力された番号の要素はありません")
    End If

    Return cnt
End Function

Sub CmdDsp(list() As String, cnt As Integer)
    If cnt > 0 Then
        For num As Integer = 0 To cnt - 1
            Console.WriteLine(num & " : " & list(num))
        Next
    End If
End Sub

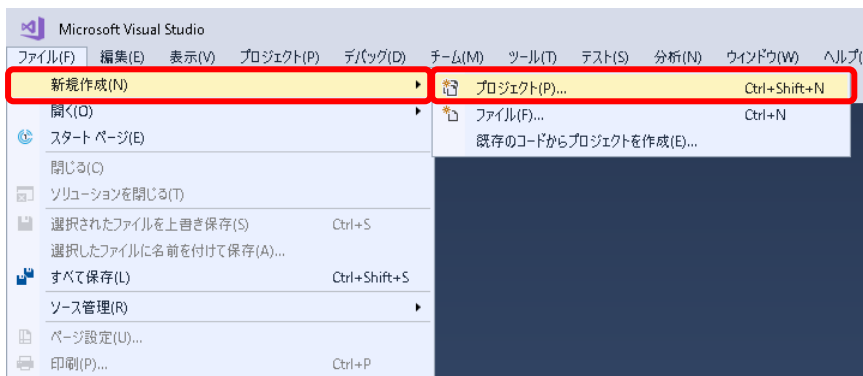
End Module
```


実行環境【Visual Studio】

プロジェクトを作成する

プロジェクトとは、これから作成して行く、いくつかのプログラムをまとめて管理するための土台のようなものです。

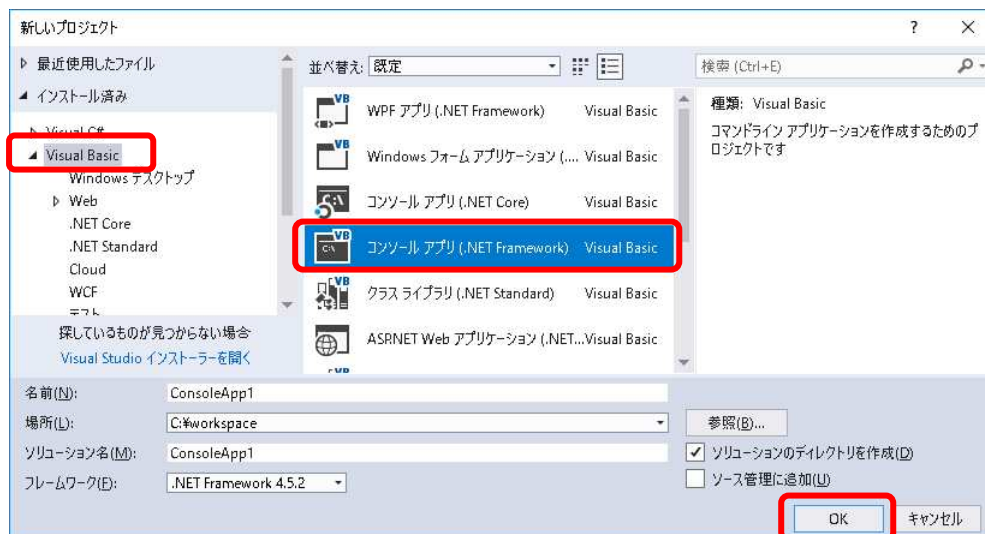
ツールバーの「ファイル」の「新規作成」から「プロジェクト」を選択します。



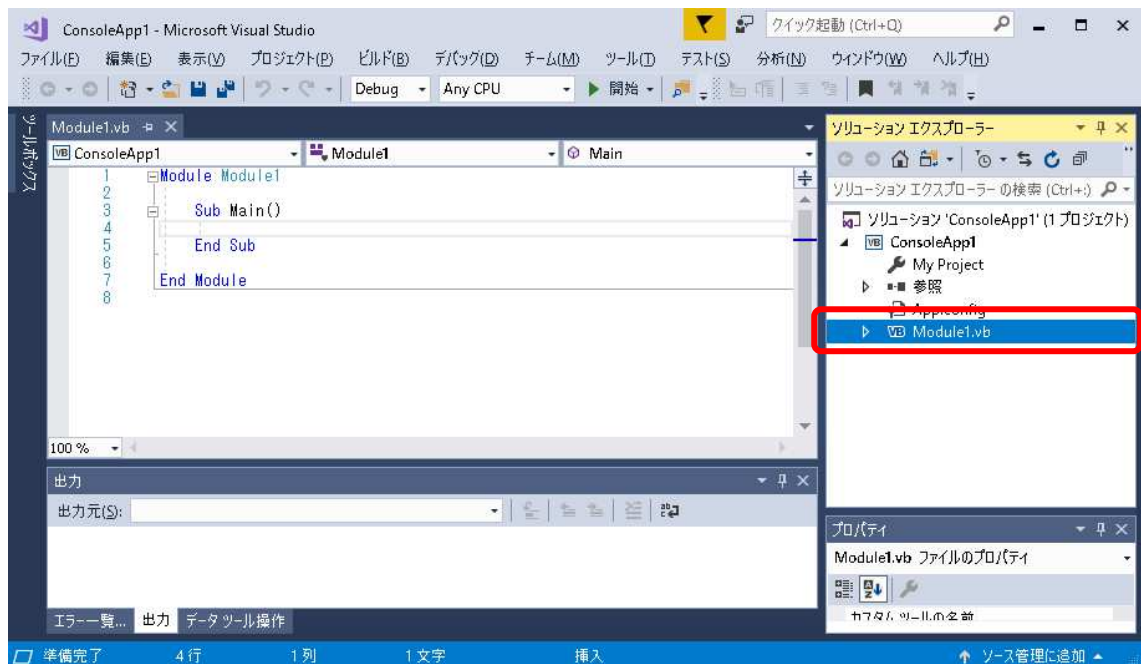
表示されたダイアログの左の部分から「Visual Basic」を選択し、中央の部分から「コンソールアプリ (.NET Framework)」を選択します。保存場所、ソリューション名、プロジェクトの名前は任意です。

(下図では場所を C:\workspace とし、ソリューション名とプロジェクトの名前はデフォルトのままにしています)

OK をクリックします。



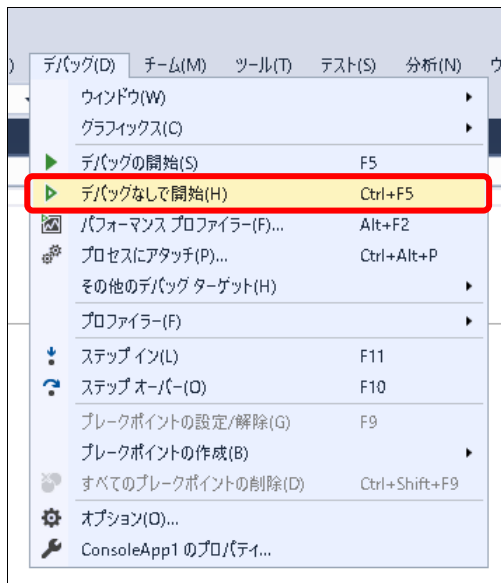
画面右上のソリューションエクスプローラーから、「Module1.vb」を選択すると、画面中央にソースコードのエディタが表示されます。



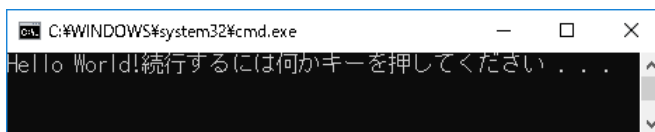
下記のように編集します。

```
1. Module Module1
2.
3.     Sub Main()
4.
5.         Console.WriteLine("Hello World!")
6.
7.     End Sub
8.
9. End Module
```

メニューバーの「デバッグ」から「デバッグなしで開始」を選択します。

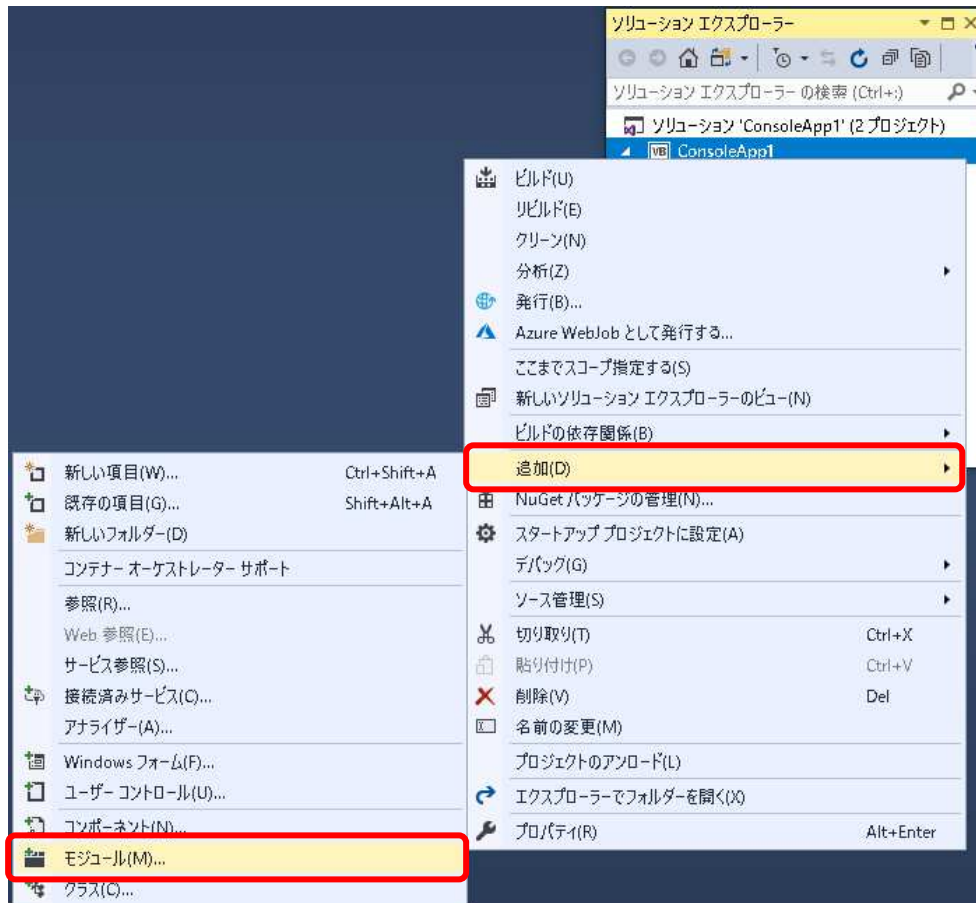


コンソールが起動し、実行結果が表示されます。

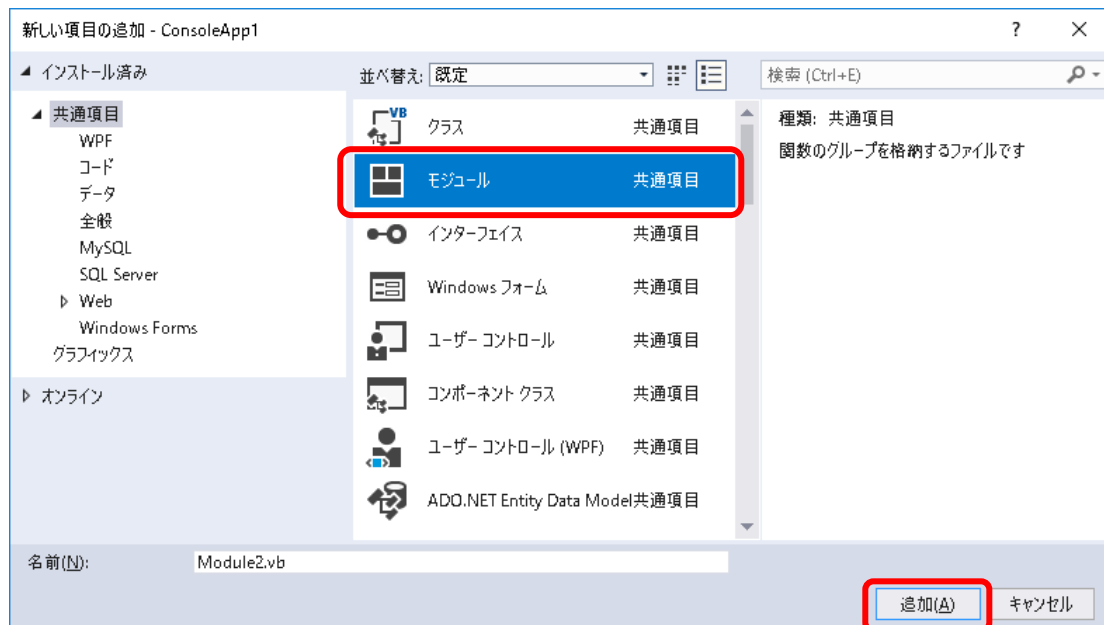


ソースファイルを追加する

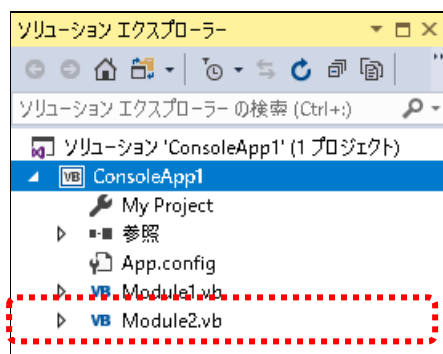
ソリューションエクスプローラーのプロジェクトを右クリックし、「追加」から「モジュール」を選択します。



モジュールを選択します。名前は任意です。「追加」をクリックします。



新しいモジュールが追加されます。



デザインの変更について

メインメニューのツールからオプションを選択すると、背景色や文字色を変更可能です。しかし、本学習環境での設定は変更しないようにお願いします。

この設定は、登録アカウントごとに行われ、別の PC にインストールされた Visual Studio でも、アカウントが同じであれば同じ設定になります。個人で扱う分にはどの PC でも自分で設定した内容になってくれるので便利ですが、本学習環境では複数人が同じ学習用のアカウントで利用しているので、一人が変更すると全員が変更されてしまいます。

プロジェクトの追加

通常業務ではほぼありませんが、本テキストのサンプルコードを実行確認して行く上で、プロジェクトを次々に作成することになります。

ソリューションエクスプローラーのソリューションを右クリックし、「追加」から「新しいプロジェクト」を選択します。

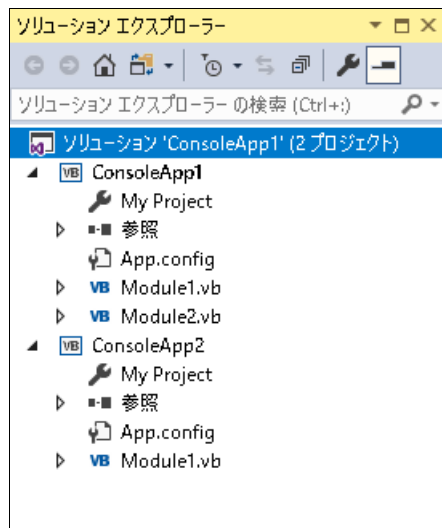


「新しいプロジェクトの追加」ダイアログの操作手順は1つ目と同様です。

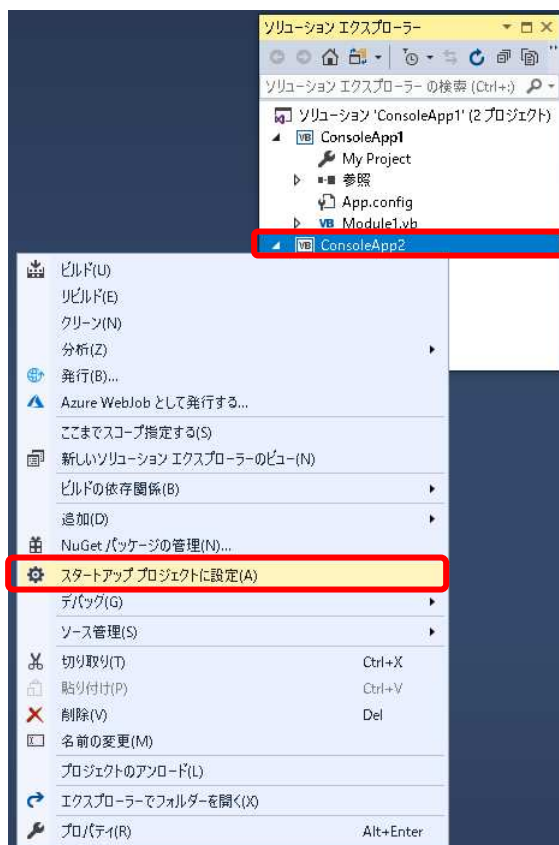
新しく作成したプロジェクトの Module1.vb に、先に作成したプロジェクトとは異なる結果が表示されるように実装します。

```
1. Module Module1
2.
3.     Sub Main()
4.
5.         Console.WriteLine("Hello New World!")
6.
7.     End Sub
8.
9. End Module
```

この時点でソリューションエクスプローラーは下図のようになります。この状態で実行しても新たに作成したほうのプログラムは動作せず、先に作成したプロジェクトのほうが動作してしまいます。それは、実行の対象が先に作成したプロジェクトのままになっているからです。



そこで、ソリューションエクスプローラー中の新たに作成したプロジェクトを右クリックし、「スタートアッププロジェクトに設定」を選択すると、実行をしたときに動作するプログラムに指定できます。



また、メニューバーの部分から実行対象とするプロジェクトを選択することでも変更可能です。



デバッグ手法

Visual Studio には様々なデバッグツールが用意されています。まずは本章で、どのような機能があるのかを確認していただき、演習や課題で練習し、実践で活用できるようにしましょう。

準備として、下記サンプルを作成し、実行確認を済ませておきましょう。

```

1.  Module Module1
2.
3.      Sub Main()
4.
5.          Dim num1 As Integer
6.          Dim num2 As Integer
7.          Dim num3 As Integer
8.
9.          num3 = num1 + num2
10.
11.          num1 = 5
12.          num2 = 6
13.
14.          Console.WriteLine(num1)
15.          Console.WriteLine(num2)
16.          Console.WriteLine(num3)
17.
18.      End Sub
19.
20. End Module

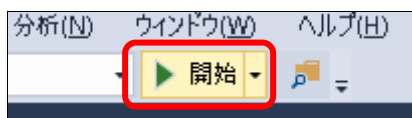
```

デバッグの開始とデバッグの停止

先ほどは「デバッグなしで実行」を選択しましたが、この項目では、デバッグを行いますので、「デバッグの開始」で実行します。メニューから選択してもかまいませんが、下記メニューアイコンをクリックするか、ショートカットでも可能です。また、同様にデバッグを停止させるメニューアイコンおよびショートカットを確認しておきましょう。

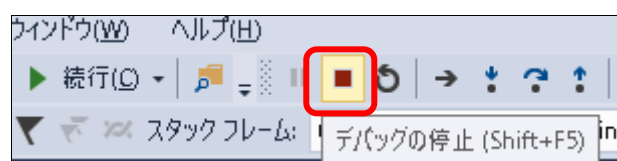
デバッグの開始

ショートカット : F5



デバッグの停止

ショートカット : Shift+F5

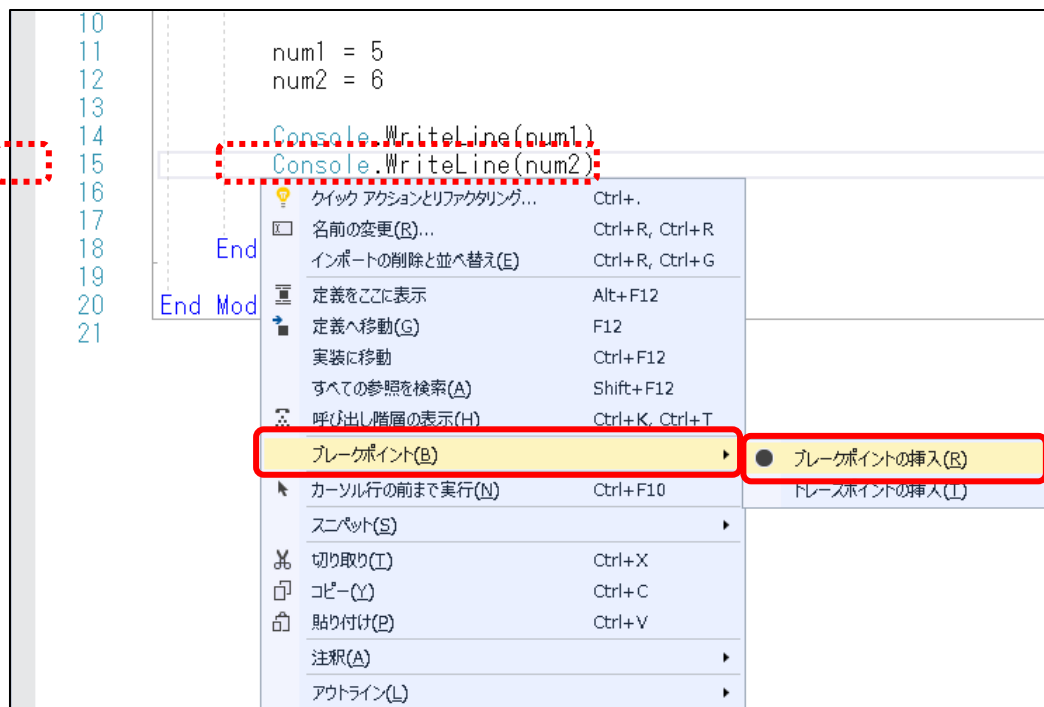


デバッグ中はメインメニューの表示が変化します。

ブレークポイント

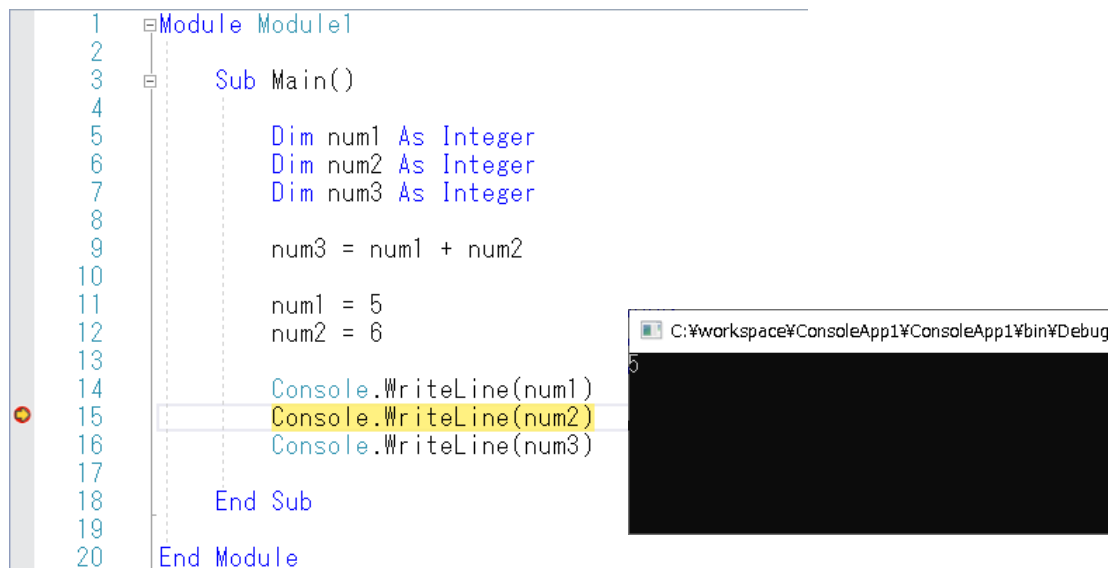
ブレークポイントを設定すると、プログラムの実行を一時停止させられます。今回は先のサンプルの 15 行目にブレークポイントを設定し、動作を一時的に停止させようとしています。

15 行目を右クリックし、「ブレークポイント」から「ブレークポイントの挿入」を選択するか、行番号の左側の灰色の部分をクリックすると、ブレークポイントを設定できます。ブレークポイントの解除は設定と同様の手順で行えます。



「デバッグの開始」をします。

ブレークポイント設定後に「デバッグの開始」をすると、ブレークポイントを設定した箇所でプログラムの実行が一時停止されます。



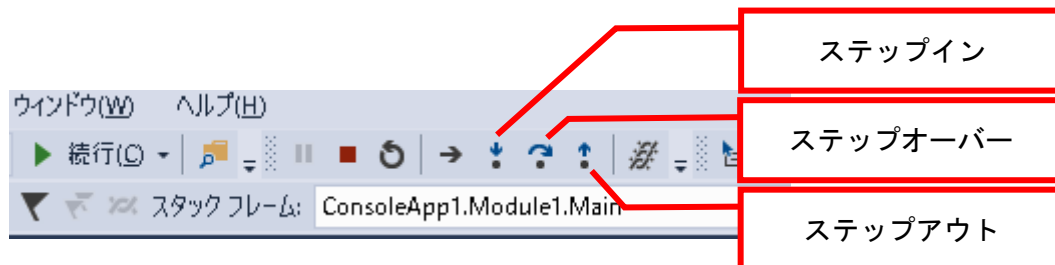
15 行目が黄色くなり、行番号の左には黄色い矢印が表示されます。この部分でプログラムが停止しているという表示です。この行はまだ実行していません。14 行目までは実行されていますが、15 行目はこれから実行しようとしている状態です。

num1 の「5」は表示されていますが、num2 の「6」は、まだ表示されていないことが確認できます。

このように、プログラムを一時停止させることができ、後述の手法で、そのときの様々な状態を調査できます。

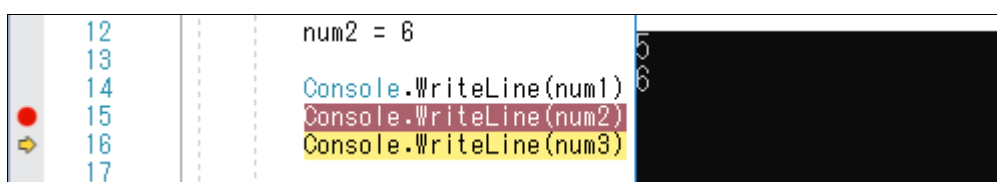
ステップ実行

プログラムを一時停止させた後、続きを一行ずつ確認しながら実行することをステップ実行といいます。ステップ実行には3種類あります。



ステップイン	停止状態にある行に関数の呼び出しが含まれている場合、その関数の中に入って確認を進めます。ソースコードが参照できない場合、コードは表示されません。
ステップオーバー	停止状態にある行を実行し、現在実行しているソースコード内の次の制御に移ります。
ステップアウト	現在停止状態にある関数の実行を進め、関数の呼び出し元に戻るまで実行をします。中断ではなく、その関数の終わりまで実行させ、呼び出し元にもどる機能です。

15行目で停止している状態から「ステップイン」または「ステップオーバー」を一度実行してください。



15行目が実行されて「6」と表示され、黄色の表示が16行目に移ります。

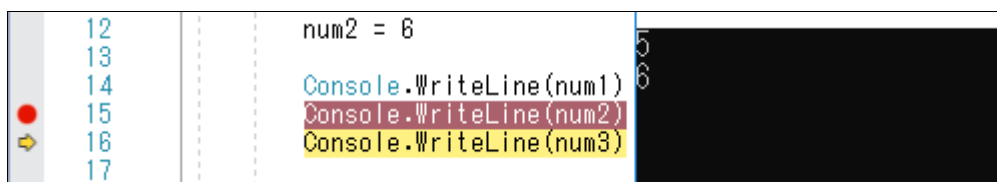
15行目は Console.WriteLine() メソッドの呼び出しですが、自作したものではないので「ステップイン」でこのメソッドの中は確認できません。今回は「ステップイン」と「ステップオーバー」のどちらでも同じ結果になります。

エディットコンティニュー

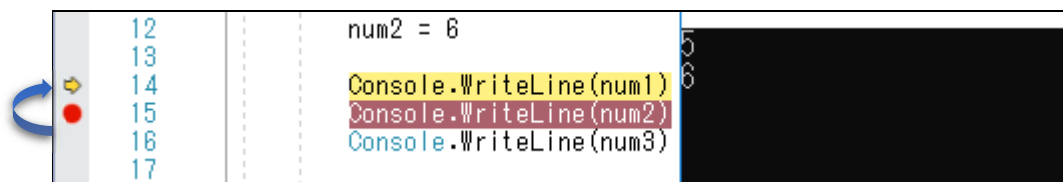
エディットコンティニューとは、プログラムの実行を一時停止中にコードを編集し、編集した内容で続行できる機能です。本来であれば、ソースコードを編集した後、再度コンパイルし、実行する必要がありますが、実行途中で編集が可能で、一度実行し終わった行をさかのぼって実行させることも可能です。

ただし、実行中の状態に大きな影響がある場合は続行が不可能になることがあります。

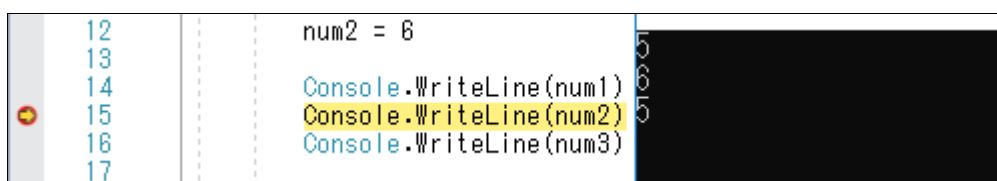
下図は 15 行目にブレークポイントを設定し、一度だけステップオーバーを実行した状態です。15 行目が実行され、num2 の「6」が表示されています。



この状態で、黄色い矢印をドロップして 14 行目へ移動させます。



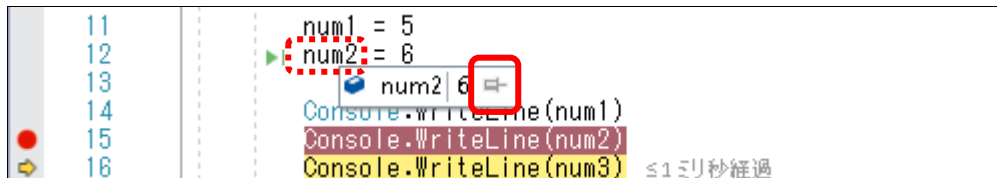
再度ステップオーバーを実行すると、14 行目が実行され、num1 の「5」が表示されました。



当然、一度出力された内容は変わりませんが、ちょっとした変更を確認するためだけに、新たに実行しなおす必要がないということです。

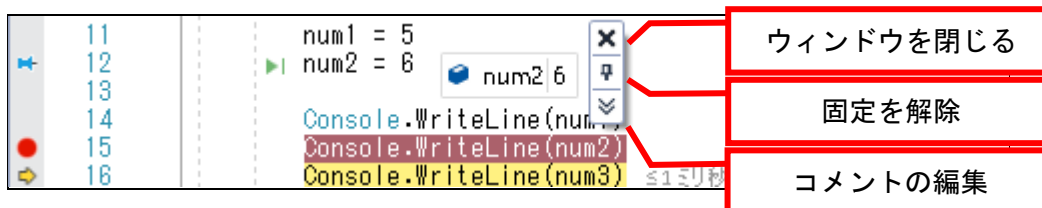
データヒント

一時停止中の関数内の変数にマウスポインタをあてると、現在の値を簡易的に表示させるウィンドウが表示されます。マウスポインタを変数から外すとこのウィンドウは消えてしましますが、ウィンドウ内の右のピンをクリックすると表示を継続させられます。

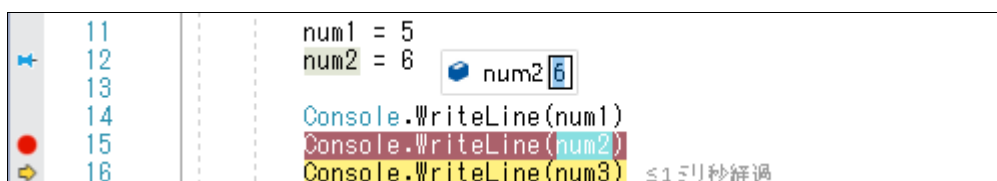


同一の変数であれば、ソース上のどの部分を参照しても同じです。プログラムが停止している現時点でのその変数の内容が表示されます。今回の場合は 12 行目の変数 num2 でも、15 行目の変数 num2 でも、どちらを参照しても中身は「6」であることが確認できます。

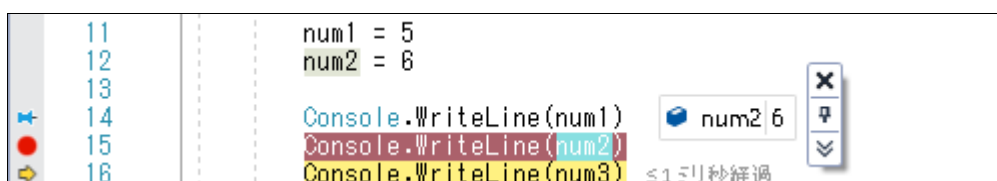
ウィンドウを固定すると、アイコンが増えます。



データ部分をクリックすることで直接データを変更できます。



また、このウィンドウは移動できます。



ローカルウィンドウ

デバッグ実行中、メニューバーの「デバッグ」から「ウィンドウ」の「ローカル」を選択します。



ローカルウィンドウはローカル変数を確認するためのものです。

ローカル		
名前	値	種類
num1	0	Integer
num2	0	Integer
num3	0	Integer

値の部分をダブルクリックすることで、データヒントと同様にデータ内容を変更できます。

ローカル		
名前	値	種類
num1	0	Integer
num2	0	Integer
num3	0	Integer

自動変数ウィンドウ

メニューバーの「デバッグ」から「ウィンドウ」の「自動変数」からでも同様のウィンドウを表示できます。ただし、自動変数ウィンドウに表示される変数はこれから実行しようとする行とその前の数行が対象です。下図であれば、これから実行しようとする行が12行目なので、自動変数ウィンドウには12行目と11行目で使用されている変数が表示されます。

```
4
5
6
7
8
9
10
11
12 num3 = num1 + num2
13
14 num1 = 5
15 num2 = 8
16
17 Console.WriteLine(num1)
18 Console.WriteLine(num2)
19 Console.WriteLine(num3)
20
21 End Sub
```

名前	値	種類
num1	5	Inte
num2	0	Inte
num3	0	Inte

名前	値	種類
num1	5	Inte
num2	0	Inte

変数内の値を確認するウィンドウの使い分け

関数内に変数が多い場合や確認したい部分に多くの変数がある場合は自動変数ウィンドウで絞り込んだほうが見やすいことがあります。

ローカル変数についてはローカルウィンドウに表示されるので、データヒントで表示する必要があまり無いかもしれません。データヒントはグローバル変数や、頻繁に確認したいローカル変数を対象に利用することが多いです。

トレースポイント・条件付きブレークポイント

次のデバッグ機能の確認のために下記サンプルを作成します。

```

3.      Sub Main()
4.
5.          For cnt As Integer = 1 To 10
6.              Console.WriteLine(cnt)
7.          Next
8.
9.      End Sub

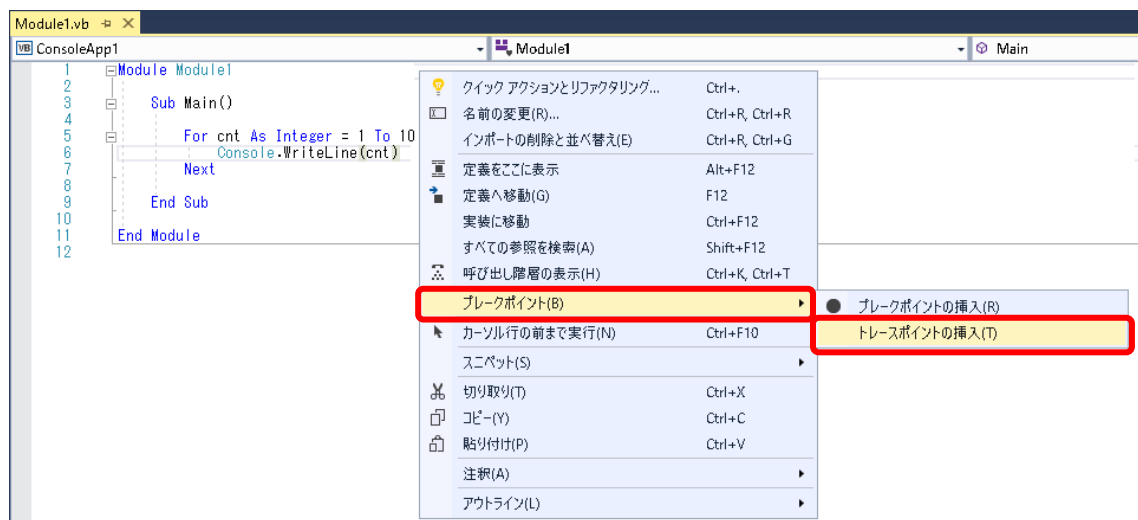
```

トレースポイント

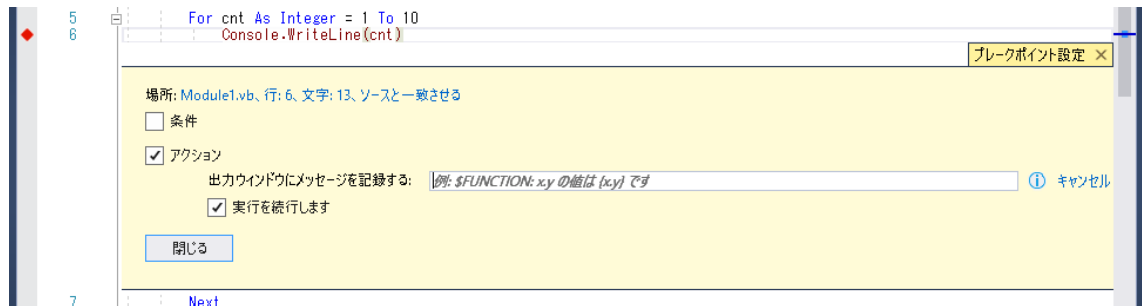
トレースポイントは、デバッグ用のメッセージを表示する行を設定できます。ブレークポイントと同じように、対象とする行を選択しますが、トレースポイントはプログラムを一時中断しません。そのポイントを通過したときに、あらかじめ設定したメッセージを表示させる機能です。

プログラムとしては、`Console.WriteLine()` メソッドなどでもメッセージを表示できますが、ソース上に記述しなければいけません。一度納品したプログラムなど、編集するべきではないソースに対してはトレースポイントが有効です。

対象となる行のところで右クリックをし、「ブレークポイント」から「トレースポイントの挿入」を選択します。今回は 6 行目に設定します。



ソースに割り込むような形で設定画面が表示されます。ブレークポイントとは異なり、対象となる行に赤い◆マークが表示されています。



「アクション」はメッセージの内容を設定します。その時点で宣言されている変数を含めることができます。

条件のチェックボックスを ON にすると、アクションが実行される条件を設定できます。



左のドロップダウンから、条件式／ヒットカウント／フィルターを選択します。隣のドロップダウンの内容は、前述のドロップダウンの選択状況によって変化します。

条件指定の種類と概要は以下のとおりです。

条件式	true の場合	cnt == 3 のようにプログラムの記述と同じ条件式で記述し、その結果が true の場合にアクションを実行する
	変更された場合	cnt や cnt/3 のように変数や数式を記述し、その結果に変化があった場合にアクションを実行する
ヒット カウント	=	トレースポイントを通過した回数をカウントし、その回数に対する条件を指定する。その条件が true の場合にアクションを実行する。
	の倍数の	
	>=	
フィルタ	true の場合	MachineName, ProcessId, ProcessName, ThreadId, ThreadName を指定して条件を生成する。その条件が true の場合にアクションを実行する。

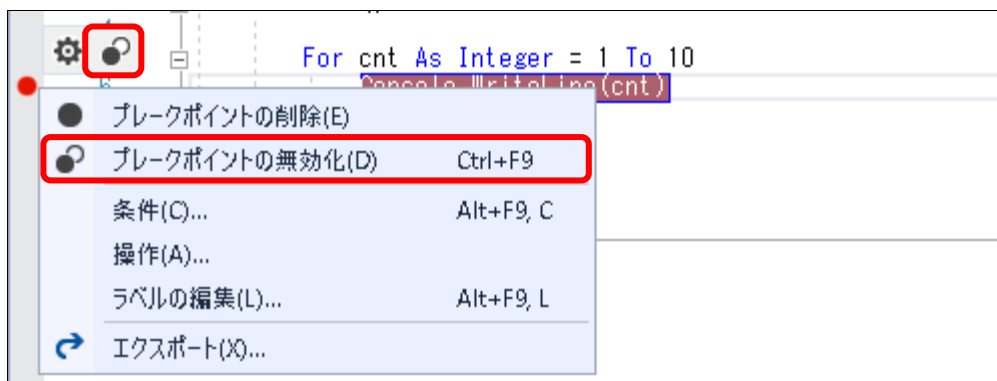
条件付きブレークポイント

ブレークポイントの赤い●マークを右クリックし、「条件」を選択すると、トレースポイントの設定と同じ画面が表示されます。

ブレークポイントとトレースポイントはアクションの違いで名前が異なるだけです。

また、これらは一時的に無効化できます。

マークの上にマウスを乗せると、小さいパネルが表示されます。マークの上で右クリックをするとコンテキストボックスが表示されます。



アクションの有無、条件の有無でアイコンが異なります。同時に、ブレークポイントを無効化したときのアイコンも異なります。

99		
100		ブレークポイント有効
101	◆	アクションあり条件あり
102	◆	アクションあり条件なし
103	+	アクションなし条件あり
104	●	アクションなし条件なし
105		
106		ブレークポイント無効
107	◆	アクションあり条件あり
108	◆	アクションあり条件なし
109	+	アクションなし条件あり
110	○	アクションなし条件なし
111		

無効化と削除は異なります。削除してしまうと条件やアクションはクリアされ、必要な場合は再入力が必要となりますので注意して扱しましょう。

呼び出し履歴

次のデバッグ機能の確認のために下記サンプルを作成します。

```
1.  Module Module1
2.
3.      Sub proc1()
4.          Console.WriteLine("Hello")
5.      End Sub
6.
7.      Sub proc2()
8.          proc1()
9.          Console.WriteLine("World")
10.     End Sub
11.
12.     Sub proc3()
13.         proc2()
14.         Console.WriteLine("!")
15.     End Sub
16.
17.     Sub Main()
18.
19.         proc3()
20.
21.     End Sub
22.
23. End Module
```

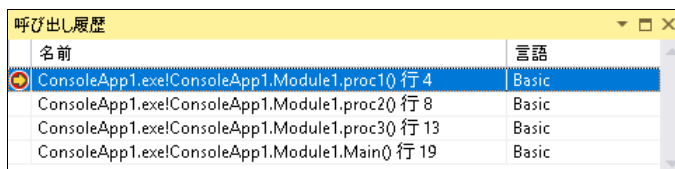
4 行目にブレークポイントを設定し、「デバッグ実行」をしてください。

実行が停止されたら、メニューバーの「デバッグ」から「ウィンドウ」の「呼び出し履歴」を選択します。

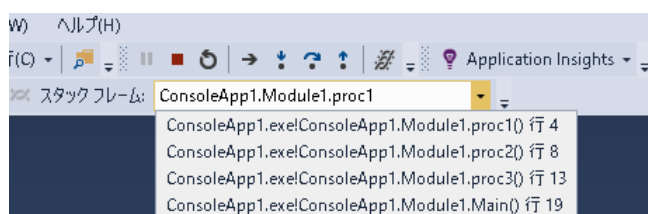


呼び出し履歴は、現在実行中のプロシージャが、どのプロシージャから呼び出されたのかを確認するためのものです。

リスト内をダブルクリックすると、選択した行へジャンプします。



画面上部にあるスタックフレームの部分をクリックすると、簡易的に呼び出し履歴を確認できます。クリックすると、選択した行へジャンプします。



初版発行日： 2014 年 03 月 30 日
最終更新日： 2021 年 02 月 12 日
著 作： 株式会社シンクスバンク
発 行 者： 株式会社シンクスバンク



学習サービスのさらなる向上を。
ISO29990 認証取得。

KENスクールは2015年8月「ISO 29990」を認証取得しました。
ISO29990は、ISO（国際標準機構）が学習サービスの「品質」を客観的に評価する国際規格で、認証取得後も学習サービスの質を維持・向上し続けていくことが常に求められます。

本書の一部または全部を、株式会社シンクスバンクから正式な許諾を得ずに、
いかなる方法（転載・転用・送信・上映等）においても無断で複写、複製する
ことは禁止されています。