

INTERNET ACADEMY

Institute of Web Design & Software Services

C#1

インターネット・アカデミー

C# 目次

1. C# とは
2. プロジェクト
3. 変数とデータ型
4. 型変換
5. 基本的な演算子
6. 配列変数
7. ループ処理

C#とは

C#はマイクロソフトが開発した**オブジェクト指向**プログラミング言語の一種でWebアプリやゲーム開発など様々な分野で活用されている言語

メリット

- ・ マイクロソフトが開発した言語であるため使用しやすい開発環境が用意されている。
- ・ 文法がC++やJavaなど他のオブジェクト指向言語に近くこれら言語から移行しやすい
- ・ Web（ASP.NETなど）やゲームなど応用範囲が広い

C#とは

C#はマイクロソフトが開発した**オブジェクト指向**プログラミング言語の一種でWebアプリやゲーム開発など様々な分野で活用されている言語

現在主流のプログラムの設計手法の一つで、プログラムをオブジェクトという単位の部品で構成するという考え方

- ・マイクロソフトが開発した言語であるため使用しやすい開発環境が用意されている。
- ・文法がC++やJavaなど他のオブジェクト指向言語に近くこれら言語から移行しやすい
- ・Web（ASP.NETなど）やゲームなど応用範囲が広い

C#とは

C#はマイクロソフトが開発した**オブジェクト指向**プログラミング言語の一種でWebアプリやゲーム開発など様々な分野で活用されている言語

メリット

- ・ マイクロソフトが開発した言語であるため使用しやすい開発環境が用意されている。
- ・ 文法がC++やJavaなど他のオブジェクト指向言語に近くこれら言語から移行しやすい
- ・ Web（ASP.NETなど）やゲームなど応用範囲が広い

C#とは

C#はマイクロソフトが開発した**オブジェクト指向**プログラミング言語の一種でWebアプリやゲーム開発など様々な分野で活用されている言語

メリット

- ・ マイクロソフトが開発した言語であるため使用しやすい開発環境が用意されている。
- ・ 文法がC++やJavaなど他のオブジェクト指向言語に近くこれら言語から移行しやすい

C#はもともとC++言語の進化系として作られたので文法が近く、同系統のJava言語とも似ている点が多い

C#とは

C#はマイクロソフトが開発した**オブジェクト指向**プログラミング言語の一種でWebアプリやゲーム開発など様々な分野で活用されている言語

メリット

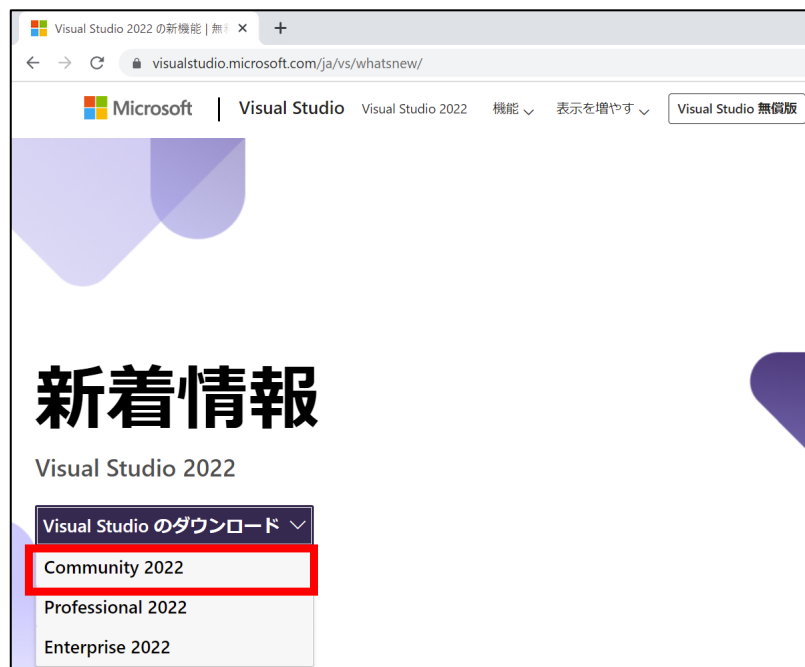
- ・ マイクロソフトが開発した言語であるため使用しやすい開発環境が用意されている。
- ・ 文法がC++やJavaなど他のオブジェクト指向言語に近くこれら言語から移行しやすい
- ・ **Webやゲームなど応用範囲が広い**

Web開発のためのフレームワーク（ASP.NET）やゲームエンジンであるUnityへの対応などで幅広い分野で活用されている

VisualStudio2022の入手

・ダウンロードとインストール

VisualStudioのダウンロードサイトにアクセスし無償利用可能な「community2022」をダウンロードしてインストーラを取得し起動する



ダブルクリックでインストール開始

VisualStudio2022の入手

・C#言語のインストール

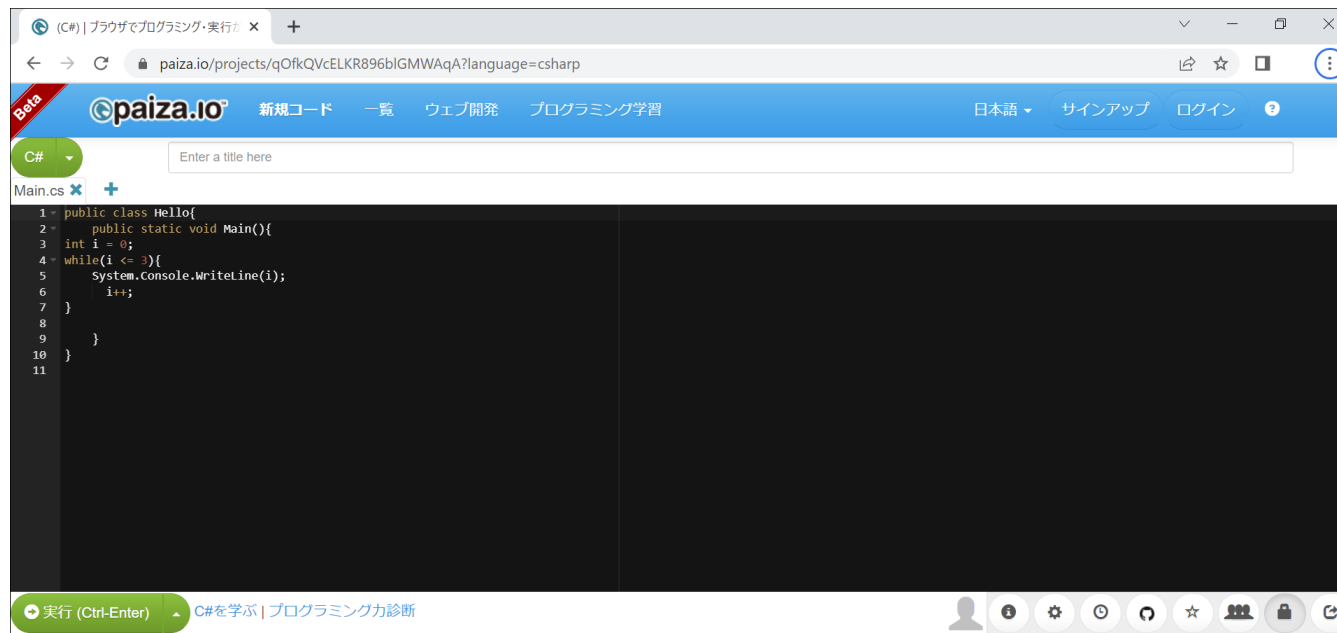
インストール完了後「VisualStudioInstaller」を起動し「.NET デスクトップ環境」を選択し「変更」をクリック



オンラインの開発環境 (Paiza)

- **Paiza.IO**

Webブラウザを利用した開発環境…無料で利用可能



<https://paiza.io/projects/qOfkQVcELKR896blGMWAqA?language=csharp>

プロジェクトの作成と構造

VisualStudio2022ではプロジェクトの単位でプログラムを管理するため最初にプロジェクトを作る必要がある

プロジェクト作成



プロジェクトの作成と構造

作成するプロジェクトの種類を選択する必要がある「**コンソールアプリ (.NETFramework)**」を選択し、「次へ」を選択する



プロジェクトの作成と構造

プロジェクト名を入力し「次へ」ボタンをクリックし次の画面で「作成」をクリックする

新しいプロジェクトを構成します

コンソール アプリ (.NET Framework) C# Windows コンソール

プロジェクト名(I)

ConsoleApp1 ①

場所(L)

C:\Users\Yuki\Documents\Visual Studio 2017\Projects\ConsoleApp1\ConsoleApp1

ソリューション(S)

ConsoleApp1

☐ ソリューションを生成する

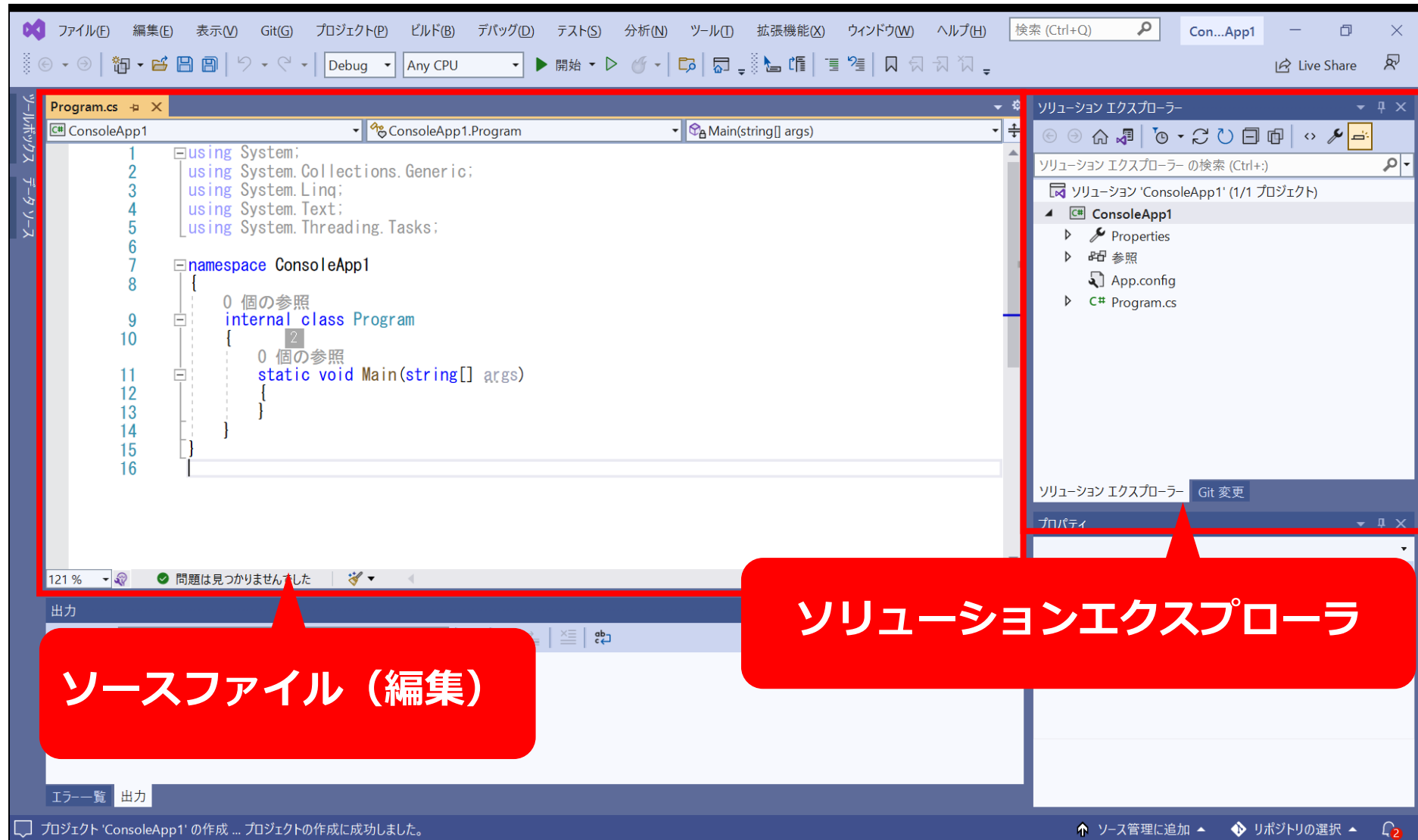
フレームワーク(F)

.NET Framework 4.7.2

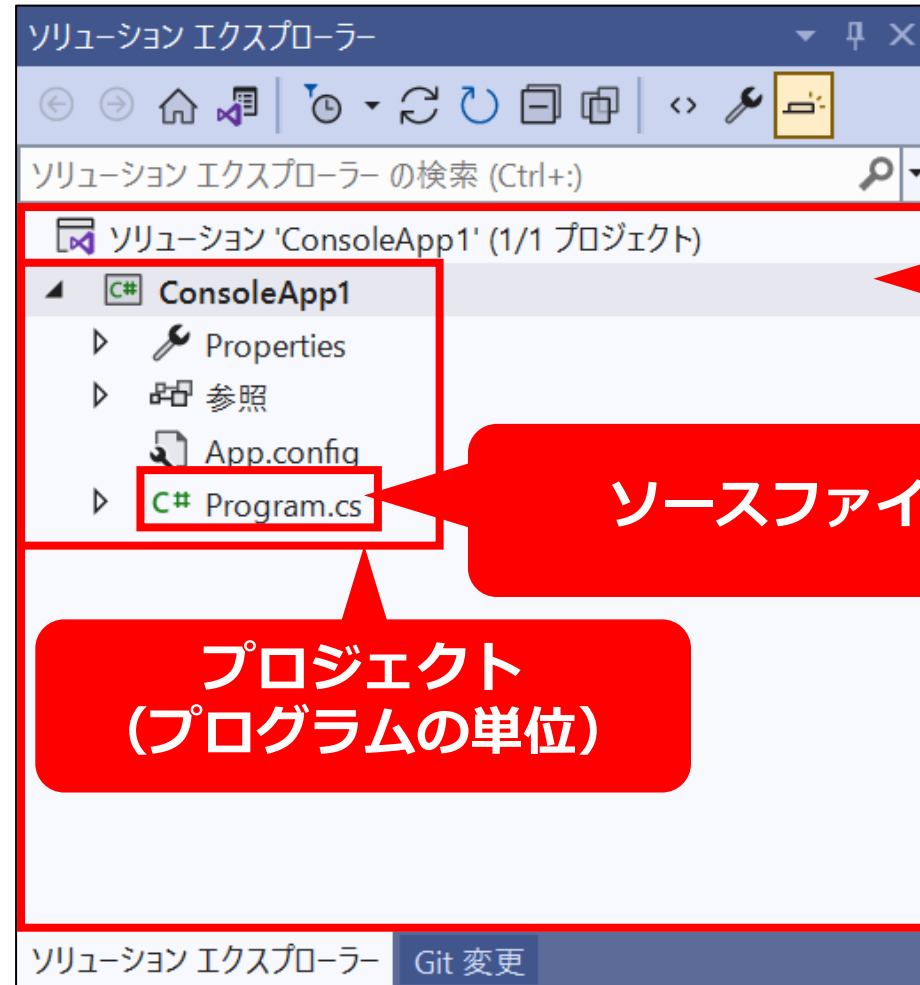
戻る(B) 作成(C) ②

プロジェクト名を入力

プロジェクトの作成と構造



プロジェクトの作成と構造

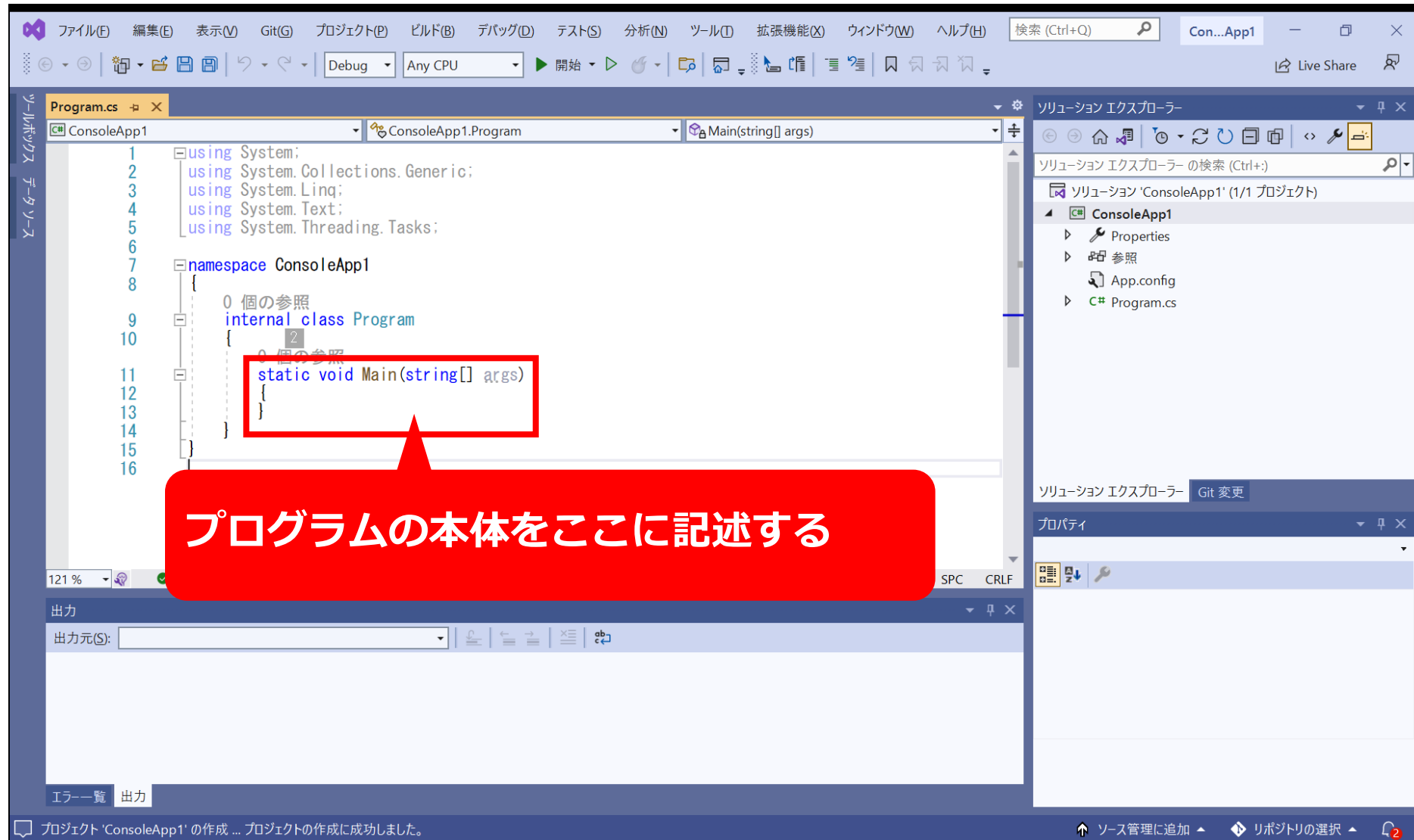


ソリューション
(1つ以上のプロジェクトを管理)

ソースファイル

プロジェクト
(プログラムの単位)

Mainメソッド



Mainメソッド

VisualStudio2022でのプログラムを入力する場合

▼記述例

```
Console.WriteLine("Hello, World!");
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp1
8  {
9      0 個の参照
10     internal class Program
11     {
12         0 個の参照
13         static void Main(string[] args)
14         {
15             Console.WriteLine("Hello, World!");
16         }
17     }
```

・プログラムは**Mainメソッド(※)**の内部に記述

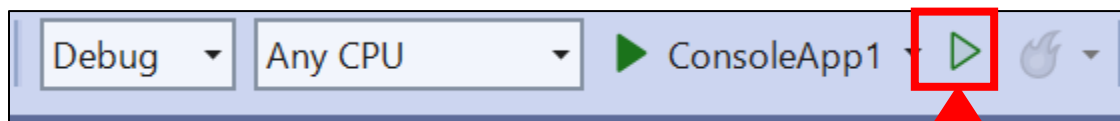
※**メソッド**…ここでは「処理」という風に理解しましょう。

プログラムの本体をここに記述する

Hello Worldの表示

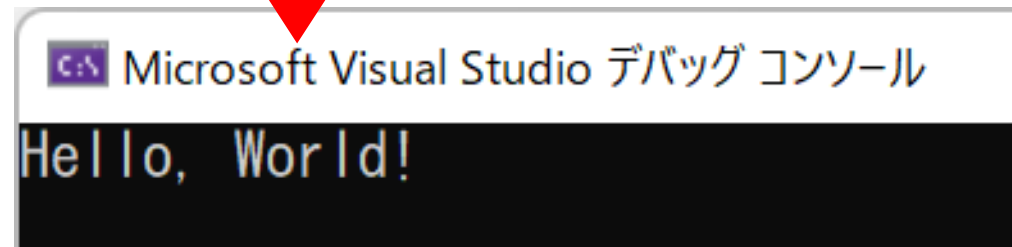
VisualStudio2022でプログラムをコンパイルし実行する

「デバッグなしで開始」 ボタンをクリック



[CTRL]+[F5]キーでも可

別ウィンドウに結果表示



！ ポイント

コンパイル=ソースファイルを実行可能なファイル（exeファイル）に変換すること

Mainメソッド

Paizaでプログラムを入力する場合

▼記述例

```
Console.WriteLine("Hello, World!");
```

```
C# Enter a title here
Main.cs
1 using System;
2
3 public class Hello{
4     public static void Main(){
5         Console.WriteLine('Hello,World');
6     }
7 }
8
```

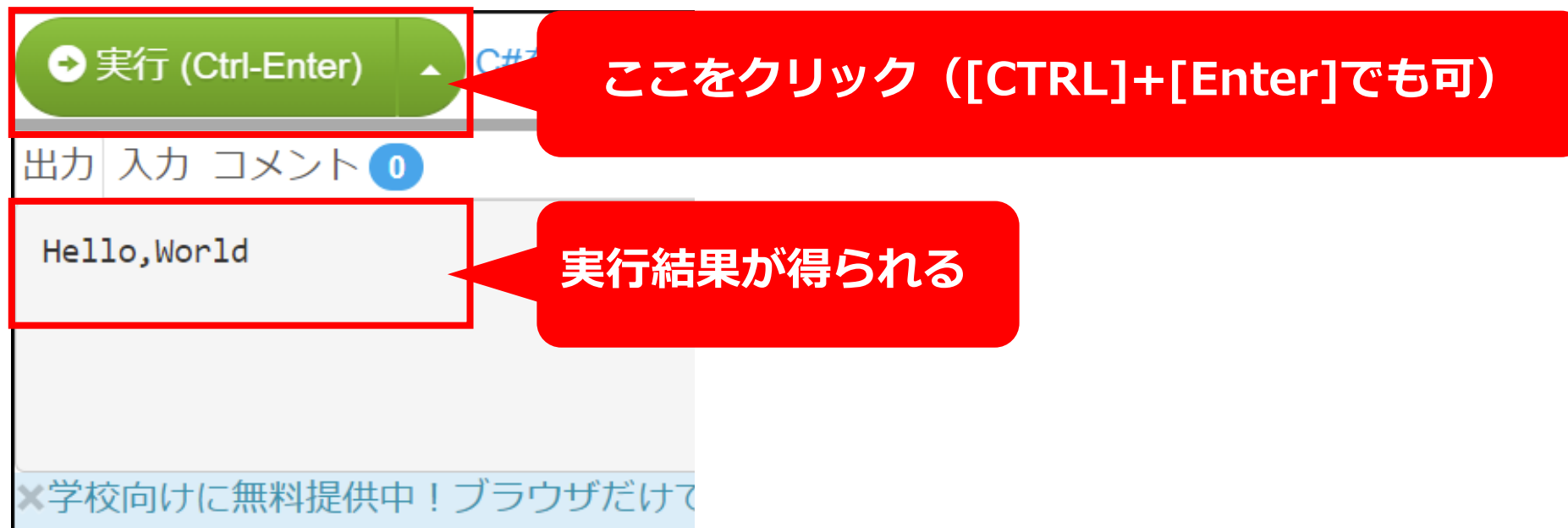
先頭に「using System;」と追加

プログラムの本体をここに記述する

Hello Worldの表示

Paizaでプログラムをコンパイルし実行する

「実行」ボタンをクリックすると実行される



Hello Worldの表示

C#で情報を画面に出力するにはConsole.WriteLineを用いる

```
Console.WriteLine(出力したい情報);
```

最後は処理の終わりを意味する「;」（セミicolon）をつける

▼記述例

```
Console.WriteLine("Hello, World!");
```

▼実行結果

```
Hello, World!
```

Hello Worldの表示

//を書くと、その後が実行されない**文（コメント）**を書くことができる

▼記述例

```
// コンソールにHelloWorldと表示して終了  
Console.WriteLine("Hello, World!");
```

1行のみのコメント=行コメント

▼実行結果

```
Hello, World!
```

コメントに記述した分は、実行結果に出力されない

Hello Worldの表示

`/*...*/`を書くと、複数行のコメント（**ブロックコメント**）が記述可能

▼記述例

```
/*
```

```
    HelloWorld
```

```
*/
```

```
Console.WriteLine("Hello, World!");
```

複数行のコメント=ブロックコメント

▼実行結果

```
Hello, World!
```

コメントに記述した分は、実行結果に出力されない

様々なコンソールへの出力

文字列を囲むクォーテーションは“(ダブルクォーテーション)
Console.Writeは改行なし、Console.WriteLineは改行ありで表示

▼記述例

```
Console.Write("ABC");    // 改行なし  
Console.WriteLine("DEF"); // 改行あり
```

▼実行結果

Writeで表示すると改行なしで次の文字列が表示される

ABCDEF

様々なコンソールへの出力

数値や数値による演算結果を表示する場合には“(ダブルクォーテーション)は必要ない

▼記述例

```
Console.Write(123);      // 改行なし  
Console.WriteLine(456); // 改行あり
```

▼実行結果

```
123456
```

様々なコンソールへの出力

数値や数値による演算結果を表示する場合には“(ダブルクォーテーション)は必要ない

▼記述例

```
Console.Write(123);      // 改行なし  
Console.WriteLine(456); // 改行あり
```

▼実行結果

```
123456
```

！ポイント

数値も“で囲んだ場合には文字列扱い (例 : “123”)

エスケープシーケンス

エスケープシーケンス

¥をつけて、' や " などの記号を文字として扱うようにするもの

エスケープシーケンス	意味
¥'	シングルクォーテーションそのもの
¥"	ダブルクォーテーションそのもの
¥¥	¥そのもの
¥n	改行
¥t	水平タブ
¥a	ベル(警告音)
¥b	バックスペース
¥f	改ページ
¥r	キャリッジリターン
¥v	垂直タブ
¥0	NULL
¥(改行)	文字列を途中で改行

エスケープシーケンス

エスケープシーケンス

¥をつけて、' や " などの記号を文字として扱うようにするもの

エスケープシーケンス	意味
¥'	シングルクォーテーションそのもの
¥"	ダブルクォーテーションそのもの
¥¥	¥そのもの
¥n	改行
¥t	水平タブ
¥a	ベル(警告音)
¥b	バックスペース
¥f	改ページ
¥r	キャリッジリターン
¥v	垂直タブ
¥0	NULL
¥(改行)	文字列を途中で改行

¥記号と文字を組み合わあせて、
特殊な意味を持ったものがある
(特に、改行と水平タブは利用頻
度が高い)

様々なコンソールへの出力

▼記述例

```
Console.WriteLine("¥tABC");      // タブの後にを表示  
Console.WriteLine("DEF¥nGHI");    // 途中で改行
```

「¥」をつけて直後の文字をエスケープ

▼実行結果

```
    ABC  
DEF  
GHI
```

A faint, light blue world map is visible in the background of the slide, centered behind the title text.

3.変数とデータ型

整数のデータ型

使用頻度が高いのがint型

短い名前 (エイリアス)	.NET クラス	型説明	範囲
byte	System.Byte	8ビット符号なし整数	0 ~ 255
sbyte	System.SByte	8ビット符号あり整数	-128 ~ 127
int	System.Int32	32ビット符号あり整数	-2,147,483,648 ~ 2,147,483,647
uint	System.UInt32	32ビット符号なし整数	0 ~ 4294967295
short	System.Int16	16ビット符号あり整数	-32,768 ~ 32,767
ushort	System.UInt16	16ビット符号なし整数	0 ~ 65535
long	System.Int64	64ビット符号あり整数	-922337203685477508 ~ 922337203685477507
ulong	System.UInt64	64ビット符号なし整数	0 ~ 18446744073709551615

実数のデータ型

使用頻度が高いのがdouble型

短い名前 (エイリアス)	.NET クラス	型説明	範囲
float	System.Single	単精度浮動小数点型	-3.402823e38 ~ 3.402823e38
double	System.Double	倍精度浮動小数点型	-1.79769313486232e308 ~ 1.79769313486232e308

・ float型とdouble型の表現方法の違い

float型 : 32.0f 0.0f -5.1f (数字の末尾に「f」をつける)

double型 : 32.0 0.0 -5.1 (数字の末尾に「f」はつけない)

文字型と文字列型

使用頻度が高いのがstring型

短い名前 (エイリアス)	.NET クラス	型説明	範囲
char	System.Char	単一Unicode 文字	テキストで使用される Unicode 記号1文字
string	System.String	文字列	char型Unicode 記号のコレクション

・ char型とstring型の表現方法の違い

char型 : 'A' '0' 'e' (シングルクォーテーションで囲む)

string型 : "ABCDEFGH" "A" "123" (ダブルクォーテーションで囲む)

その他特殊な型

使用頻度が最も高いのがbool型

短い名前 (エイリアス)	.NET クラス	型説明	範囲
bool	System.Boolean	論理型	true または false
decimal	System.Decimal	29 の有効桁数で 10 進 数を表現できる正確な小 数または整数型	$\pm 1.0 \times 10e - 28$ ～ $\pm 7.9 \times 10e28$
object	System.Object	他のすべての型の基本型	

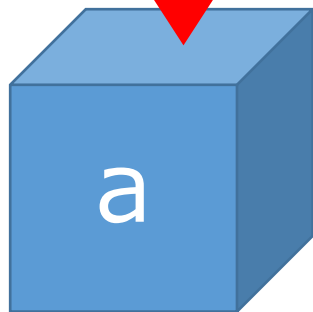
変数

データを一時的に入れておく箱のようなもの

1. 変数の宣言

…変数を作る

int型の変数を宣言

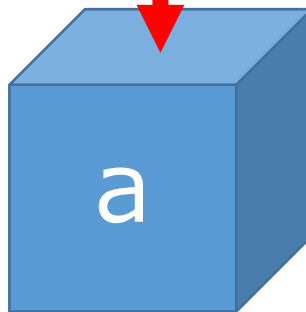


```
int a;
```

1. 変数の代入

…作った変数に値を入れる

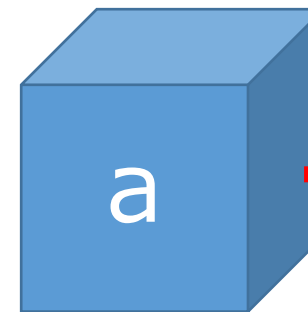
6



```
a = 6;
```

2. 参照

…代入した値を取得する



```
Console.WriteLine(a);
```

変数

変数名のルール

- ・ 半角英字、半角数字、_(アンダースコア) を使用できる
- ・ スペースやその他特殊文字は使用できない
- ・ 1文字目を数字にすることはできない
- ・ 大文字小文字の区別をする

例 1aaa = “あいう”;  1文字目が数字

a-1 = “あいう”;  -(ハイフン)の使用

変数の宣言と代入

▼記述例

int型の変数は整数の値を代入するための変数（文字列などは代入不可能）

```
int a; // 変数の宣言  
a = 6; // 代入  
Console.WriteLine(a); //参照
```

▼実行結果

6

変数の宣言と代入

▼記述例

変数の宣言と初期化を同時に行うことができる

```
int b = 3; // 宣言と初期化を同時に行う  
Console.WriteLine(b);  
Console.WriteLine("b={0}", b); // 文章に変数の値を埋め込んで出力
```

{0}の部分にbの値が埋め込まれる

▼実行結果

```
3  
b=3
```


変数の宣言と代入

▼記述例

文字列の変数はstring型で宣言する

```
string s = "abc"; // 宣言と初期化を同時に行う  
Console.WriteLine(s);  
Console.WriteLine("s={0}", s); // 文章に変数の値を埋め込んで出力
```

▼実行結果

```
abc  
s=abc
```

constキーワード

変数宣言の先頭にををつけると定数になり値を変更することはできない

▼記述例

const指定した変数は大文字にする慣習

```
const int NUMBER = 100;  
// NUMBER = 10;  
Console.WriteLine(NUMBER);
```

コメントを取るとエラーになる

▼実行結果

100

変数の練習

以下の処理を実行するC#プログラムを考えてみましょう

- ・ 変数numを作り、整数値123を代入する
- ・ 変数numの値を画面に出力する

▼正解コード

?

変数の練習

以下の処理を実行するC#プログラムを考えてみましょう

- ・ 変数numを作り、整数値123を代入する
- ・ 変数numの値を画面に出力する

▼正解コード

```
int num = 123;  
Console.WriteLine(num);
```


型変換とキャスト

型変換 ... データを異なる型に変換すること

キャスト ... 明示的に型変換を行うこと

- ・ **数値型の場合**

```
double d = 1.24;
```

```
int a = (int)d;
```

実数から整数へのキャスト

```
Console.WriteLine("a={0}", a); → 実行結果 : a=1
```

キャストが必要なケース

型変換の際にキャストを省略できるケースとできないケースがある

- ・ **整数→実数の型変換の場合**

```
int a = 5;
```

```
double d = a;
```

```
Console.WriteLine("d={0}", a);
```

→**実行結果：d=5**

実数→整数のようにデータの一部が損なわれる時キャストをしないとエラーになる

- ・ **実数→整数の型変換の場合**

```
double d = 1.24;
```

```
int a = d; // エラー
```

```
Console.WriteLine("a={0}", a);
```

実数→整数のようにデータの一部が損なわれる時キャストをしないとエラーになる

Parseメソッドによる型変換

文字列から数値への変換はキャストではできないのでParseメソッドを用いる

- ・ **文字列→整数の型変換の場合**

```
string s = "12345";  
int n = int.Parse(s); // 文字列から整数への変換  
Console.WriteLine(n); → 実行結果 : d=12345
```

- ・ **実数→整数の型変換の場合**

```
string s = "123.45";  
double d = double.Parse(s); // 文字列から実数への変換  
Console.WriteLine(d); → 実行結果 : 123.45
```


例外の発生

型変換の際にキャストを省略できるケースとできないケースがある

▼記述例

数値ではない文字列をParseで数値に変換

```
string s = "abcdef";
```

```
double d = double.Parse(s); // 文字列から実数への変換
```

```
Console.WriteLine("b={0}", b); // 文章に変数の値を埋め込んで出力
```

▼実行結果

「abcdef」は実数値に変換できないため例外発生

ハンドルされていない例外: System.FormatException: 入力文字列の形式が正しくありません。...

ToStringメソッドによる型変換

文字列以外の変数を文字列に変換するにはToStringメソッドを使う

- ・ **整数→文字列の型変換の場合**

```
int num = 10;  
string numStr = num.ToString();  
Console.WriteLine(numStr); → 実行結果 : 10
```

- ・ **実数→文字列の型変換の場合**

```
double d = 1.23;  
string numStr = d.ToString();  
Console.WriteLine(numStr); → 実行結果 : 1.23
```



5.基本的な演算子

演算子と式

数値計算に用いられる演算子は以下の種類がある

演算子	意味	使用例
+	足し算を行う演算子	$5 + 5$
-	引き算を行う演算子	$7 - 3$
*	掛け算を行う演算子	$7 * 3$
/	割り算を行う演算子	$7 / 3$
%	割り算のあまりの演算子	$7 \% 3$

演算子と式

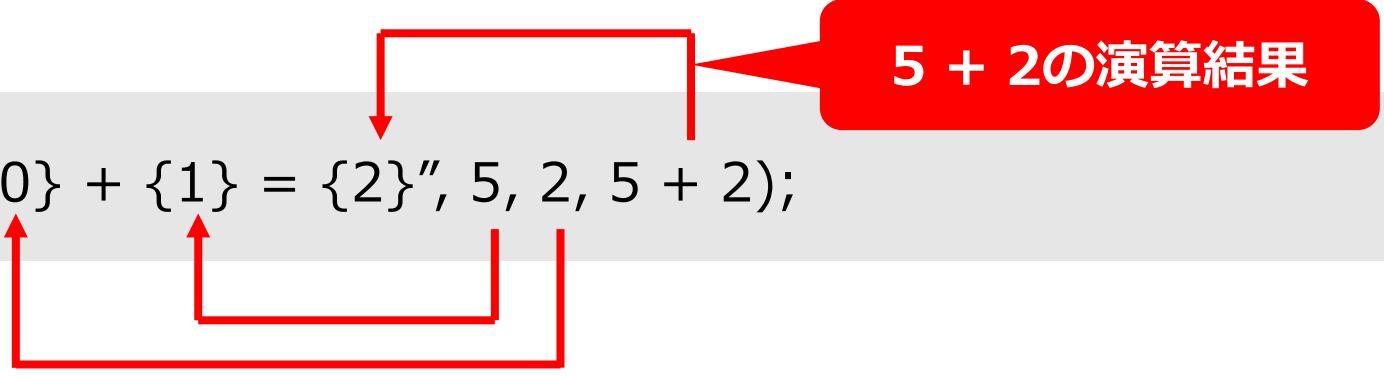
足し算の結果をConsole.WriteLineで出力

{0}、{1}、{2}…の部分に「,」の値が順番に入る

{2}には「5 + 2」は演算結果の「7」が入る

▼記述例

```
Console.WriteLine("{0} + {1} = {2}", 5, 2, 5 + 2);
```



5 + 2の演算結果

▼実行結果

5 + 2 = 7

演算子と式

足し算以外の演算の例

▼記述例

```
Console.WriteLine("{0} - {1} = {2}", 5, 2, 5 - 2);  
Console.WriteLine("{0} * {1} = {2}", 5, 2, 5 * 2);  
Console.WriteLine("{0} / {1} = {2} 余り {3}", 5, 2, 5 / 2, 5 % 2);
```

▼実行結果

5 - 2 = 3

5 * 2 = 10

5 / 2 = 2 余り 1

演算子の優先順位と ()

掛け算・割り算（余りも含む）は足し算・引き算に優先するが()で優先順変更可能

- ・ 乗算を含む () を使わない演算

```
Console.WriteLine(10 + 5 * 2); → 実行結果 : 20
```

掛け算優先

- ・ 乗算を含む () を使った演算

```
Console.WriteLine((10 + 5) * 2); → 実行結果 : 30
```

()内の演算を優先

変数と演算

変数による演算結果を変数に代入する

▼記述例

```
int a = 6,b = 3;  
double avg = (a + b) / 2.0;    // aとbの平均値を求める  
Console.WriteLine("{0}と{1}の平均値", a,b);
```

▼実行結果

6と3の平均値4.5

変数と演算

自分自身に自分自身の演算の結果を代入できる

▼記述例

```
int a = 2,b = 3;  
a = a + 2;    // aに2を足す  
b = b * 4;    // bに4をかける  
Console.WriteLine("a={0} b={1}", a,b);
```

▼実行結果

```
a=4 b=12
```

代入演算子

自分自身に自分自身の演算の結果を代入できる

▼記述例

```
int a = 2, b = 3;  
a += 2; // aに2を足す      ← 「a = a + 2;」と同じ意味  
b *= 4; // bに4をかける ← 「b = b * 4;」と同じ意味  
Console.WriteLine("a={0} b={1}", a, b);
```

▼実行結果

```
a=4 b=12
```

文字列の結合

文字列の結合

+ 演算子で、文字列同士を結合できる

▼記述例

```
string s1 = "ABC";  
string s2 = "DEF";  
string s3 = s1 + s2;  
Console.WriteLine(s3);
```

**s1 とs2の文字列を結合し
s3に代入する**

▼実行結果

ABCDEF

文字列の結合

文字列と数値の結合

+ 演算子で、文字列と数値を結合できる

▼記述例

```
int a = 2;  
string s = "a";  
System.Console.WriteLine(s + a);
```

▼実行結果

a2

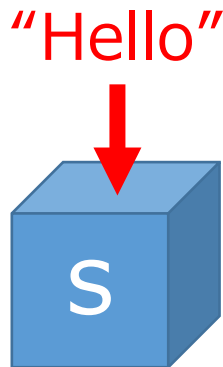
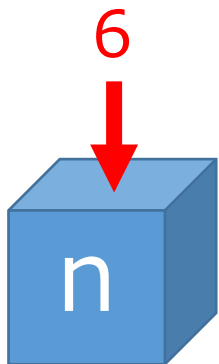
配列変数

一つの名前で複数の値を代入できる変数

・ 普通の変数

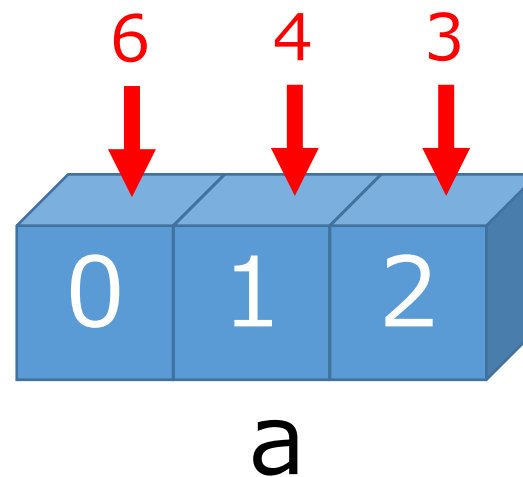
```
int n = 6;
```

```
string s = "Hello";
```



普通の変数は、1 変数に 1 つの値しか代入することができない。

・ 配列変数



配列変数は、1 つの変数に複数の値を代入することができる。

配列の作成と取得

配列を使用するときには配列変数の宣言を行う

値の代入・取得には**インデックス番号（添え字）**を用いる

▼記述例

```
double[] d = new double[3]; ← 長さ3の配列変数の宣言
```

```
d[0] = 1.2; ← 添え字は0から始まる
```

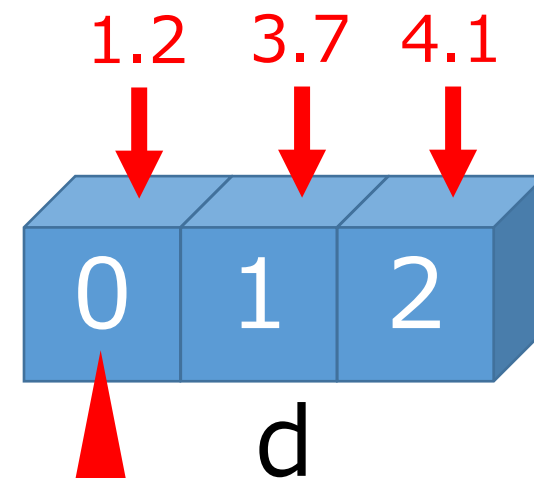
```
d[1] = 3.7;
```

```
d[2] = 4.1; ← 長さ3の場合は最後の添え字は2
```

```
Console.WriteLine(" {0} {1} {2}", d[0], d[1], d[2]);
```

▼実行結果

```
1.2 3.7 4.1
```



0 ~ 2 ... 添え字

配列の作成と取得

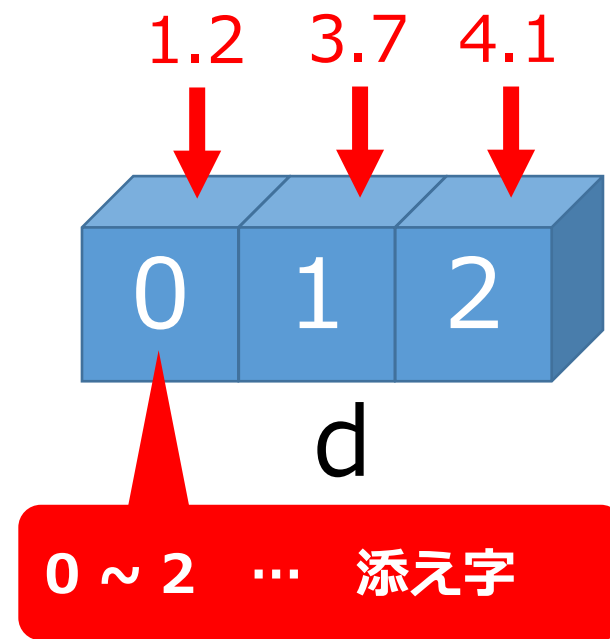
配列を宣言と初期値の代入を同時に行うことができる
{ }の中に値を「,」で区切って必要な数だけ記述する

▼記述例

```
double[] d = { 1.2, 3.7, 4.1 }; ← 宣言と代入を行う  
Console.WriteLine("{0} {1} {2}", d[0], d[1], d[2]);
```

▼実行結果

1.2 3.7 4.1



さまざまな種類の配列

▼記述例

```
int[] n = { 0, 2, 3, 5 };    ← 整数の配列  
Console.WriteLine("{0} {1} {2} {3}", n[0], n[1], n[2], n[3]);
```

▼実行結果

```
0 2 3 5
```

▼記述例

```
string[] s = { "abc", "def" }; ← 文字列の配列  
Console.WriteLine("{0} {1} {2}", s[0], s[1]);
```

▼実行結果

```
abc def
```

配列の長さの取得

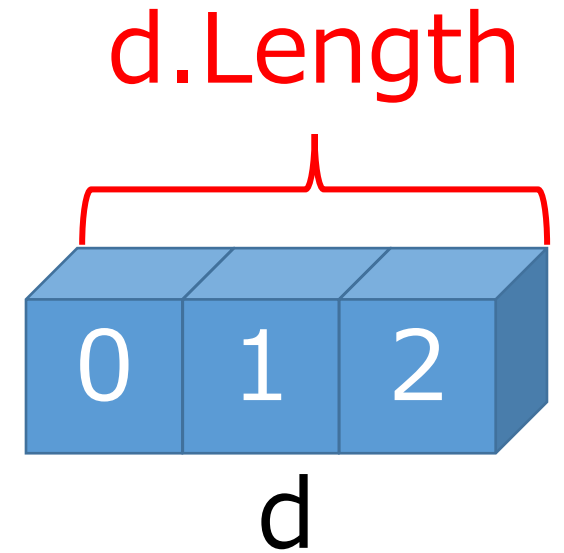
(配列変数名).Length ... 配列の長さを取得

▼記述例

```
double[] d = {1.2, 3.7, 4.1};  
Console.WriteLine(d.Length);
```

▼実行結果

3



A faint, light blue world map is visible in the background of the slide, centered behind the title text.

7.ループ処理

ループ処理 1 for文

繰り返し処理を行うためにfor文を使う

利用例 1 : 0から4までを出力する

▼記述例

```
for(int i = 0; i < 5; i++){  
    Console.WriteLine(i);  
}
```

▼実行結果

```
0  
1  
2  
3  
4
```

ループ処理 1 for文

繰り返し処理を行うためにfor文を使う

利用例 1 : 0から4までを出力する

▼記述例

```
for(int i = 0; i < 5; i++){  
    Console.WriteLine(i);  
}
```

iに1を足す処理

▼実行結果

```
0  
1  
2  
3  
4
```

ループ処理 1 for文

開始値

継続条件

```
for(int i = 0; i < 5; i++){  
    Console.WriteLine(i);  
}
```

i	停止条件
0	0<5
1	1<5
2	2<5
3	3<5
4	4<5
5	5<5

- ① 開始値の0を変数 i に代入し、処理を実行(変数 i の値0を出力)
- ② 変数 i の値を+1して、処理を実行(変数 i の値0を出力)
(②を繰り返す)
- ...
- ③ 変数 i の値を+1して継続条件満たさなくなったら、ループ終了(処理を実行しない)

ループ処理 1 for文

繰り返し処理を行うためにfor文を使う

利用例2：4から1までを出力する

▼記述例

```
for(int i = 4; i > 0; i--){  
    Console.WriteLine(i);  
}
```

iから1を引く処理

▼実行結果

```
4  
3  
2  
1
```

ループ処理 2 while文

繰り返し処理を行うためにwhile文を使う

利用例1：0から4までを出力する

▼記述例

```
int i = 0;

while(i < 5){

    System.Console.WriteLine(i);

    i++;

}
```

▼実行結果

```
4
3
2
1
```


ループ処理 2 while文

```
int i = 0;
while(i < 5){
    System.Console.WriteLine(i);
    i++;
}
```

開始値0

継続条件

変数iの値を+1

i	停止条件
0	0<5
1	1<5
2	2<5
3	3<5
4	4<5
5	5<5

- ① 開始値の0を変数 i に代入し、処理を実行(変数 i の値0を出力)
- ② 変数 i の値を+1して、処理を実行(変数 i の値0を出力)
(②を繰り返す)
- ...
- ③ 変数 i の値を+1して継続条件満たさなくなったら、ループ終了(処理を実行しない)

ループ処理 2 while文

繰り返し処理を行うためにwhile文を使う

利用例2：4から1までを出力する

▼記述例

```
int i = 4;

while(i > 0){

    System.Console.WriteLine(i);

    i--;

}
```

▼実行結果

```
4
3
2
1
```

配列の要素をループで順番に取得

繰り返し処理を行うためにfor文を使う

利用例 3 : リストの要素を1つずつ取り出す

▼記述例

```
double[] d = { 1.2, 3.7, 4.1 };  
  
for (int i = 0; i < d.Length; i++){  
    Console.WriteLine(d[i]);  
  
}
```

▼実行結果

```
1.2  
3.7  
4.1
```

配列の要素をループで順番に取得

繰り返し処理を行うためにfor文を使う

利用例 3 : リストの要素を1つずつ取り出す

▼記述例

配列変数

```
double[] d = { 1.2, 3.7, 4.1 };  
  
for (int i = 0; i < d.Length; i++){  
    Console.WriteLine(d[i]);  
}
```

▼実行結果

```
1.2  
3.7  
4.1
```

「iに初期値0を代入することにより、d[i]の値がd[0]、d[1]、d[2]と変化する」 ことにより、配列の値を1つずつ取得していく

演習問題 1 : 変数と演算

以下の処理を実行するC#プログラムを考えてみましょう

- ・ 整数型変数num1を用意し、10を代入する
- ・ 整数型変数num2を用意し、20を代入する
- ・ num1とnum2の合計を整数型変数sumに代入し出力

▼実行結果

30

演習問題 1 : 変数と演算

以下の処理を実行するC#プログラムを考えてみましょう

- ・ 整数型変数num1を用意し、10を代入する
- ・ 整数型変数num2を用意し、20を代入する
- ・ num1とnum2の合計を整数型変数sumに代入し出力

▼正解コード

```
int num1 = 10;  
int num2 = 20;  
int sum = num1 + num2;  
Console.WriteLine(sum);
```

演習問題 2 : ループ処理(for文)

以下の処理を実行するC#プログラムを考えてみましょう

- ・ 2から5までを画面に出力する

※変数*i*の値を1つずつ増やしていくこと

▼実行結果

2

3

4

5

演習問題 2 : ループ処理(for文)

以下の処理を実行するC#プログラムを考えてみましょう

- ・ 2から5までを画面に出力する

※変数*i*の値を1つずつ増やしていくこと

▼正解コード

```
for(int i = 2; i < 6; i++){  
    Console.WriteLine(i);  
}
```

演習問題 2 : ループ処理(for文)

以下の処理を実行するC#プログラムを考えてみましょう

- ・ 2から5までを画面に出力する

※変数*i*の値を1つずつ増やしていくこと

▼正解コ

開始値

継続条件

```
for(int i = 2; i < 6; i++){  
    Console.WriteLine(i);  
}
```

*i*の開始値を2、継続条件を「*i*<6」として*i*に1を足しながらループを継続させる

演習問題 3 : 配列とループ処理(for文)

以下の処理を実行するC#プログラムを考えてみましょう

- ・ 配列変数sportsに文字列「 "soccer"," tennis","basketball" 」を代入
- ・ 配列変数の値を1つずつ取り出し、変数画面に出力する

▼実行結果

soccer

tennis

basketball

演習問題 3 : 配列とループ処理(for文)

以下の処理を実行するC#プログラムを考えてみましょう

- ・ 配列変数sportsに文字列「 "soccer"," tennis","basketball" 」を代入
- ・ 配列変数の値を1つずつ取り出し、変数画面に出力する

▼正解コード

```
string[] sports = {"soccer", "tennis", "basketball"};  
for(int i = 0; i < sports.Length; i++){  
    Console.WriteLine(sports[i]);  
}
```

「配列変数の値を先頭から順に1つ取得し、Console.WriteLineで出力」を繰り返す

演習問題 4 : 配列とループ処理(for文)

以下の処理を実行するC#プログラムを考えてみましょう

- ・ 配列変数fruitsに文字列「"apple", "orange", "banana"」を代入
- ・ fruitsから、文字列データを一つ一つ取り出して、画面に出力するまた、その種類の数を出力する

▼実行結果

apple

orange

banana

3種類

演習問題 4 : 配列とループ処理(for文)

▼正解コード

```
string[] fruits = { "apple", "orange", "banana"};  
for(int i = 0; i < fruits.Length; i++){  
    Console.WriteLine(fruits[i]);  
}  
Console.WriteLine("{0}種類",fruits.Length);
```

ループ でfruitsのデータを取得

配列変数fruitsの長さで種類を取得

▼実行結果

apple
orange
banana
3種類

演習問題 5 : まとめ問題

以下の処理を実行するC#プログラムを考えてみましょう

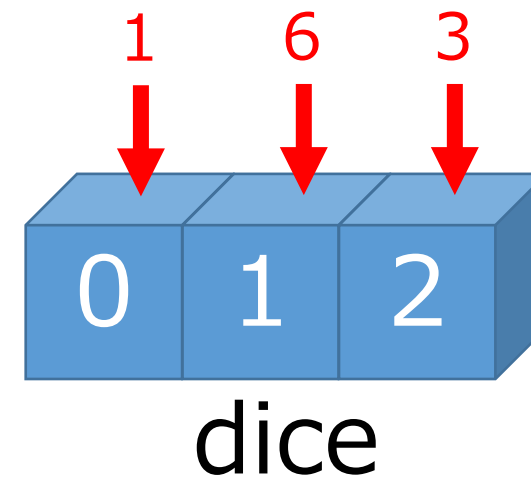
- ・サイコロの裏の目を出力するプログラムを作る
- ・配列変数diceを作り、1,6,3という3つの数字を代入
- ・for文を使って1つずつ数字を取得し、その数字(サイコロの目)の裏の目を出力

※サイコロは表と裏の目をあわせて、合計7になる

※実行結果は、6、1、4と出力される

▼正解コード

?



演習問題 5 : まとめ問題

以下の処理を実行するC#プログラムを考えてみましょう

- ・サイコロの裏の目を出力するプログラムを作る
- ・配列変数diceを作り、1,6,3という3つの数字を代入
- ・for文を使って1つずつ数字を取得し、その数字(サイコロの目)の裏の目を出力

※サイコロは表と裏の目をあわせて、合計7になる

※実行結果は、6、1、4と出力される

▼正解コード

```
int[] dice = { 1, 6, 3 };  
for(int i = 0; i < dice.Length ; i++){  
    Console.WriteLine(7-dice[i]);  
}
```

