講義メモ
・p.57 Chap1_11_1から

.56 型を変換する

・範囲の大きい同種の型に代入する場合は、自動的に暗黙の型変換が行われる
 例: int i = 5; long a = i; //暗黙の型変換でlongになる
・整数型から実数型に代入する場合は、自動的に暗黙の型変換が行われる
 例: int i = 5; double a = i; //暗黙の型変換でdoubleになる
・実数を整数型に代入する場合は、明示的な型変換(キャスト)が必要で「(型名)」を前置する
 ※ これをキャスト演算子ともいう
 例: double a = 3.14; int i = (int)a; //int型にキャストしてからなら代入可能で3になる
・実数リテラルを整数型にキャストすることも可能
 例: int i = (int)3.14; //int型にキャストしてからなら代入可能で3になる
・実数から整数型にキャストすると小数点以下切捨てになる(四捨五入ではない)
・文字列型以外の各型を文字列型に変換するには、""(0文字の文字列)に「+」で連結すると良い
 例: int i = 365; string s = "" + i; //文字列"365"になる

p.57 Chap1_11_1

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Chap1_11_1 : MonoBehaviour {
    void Start() {
        int seisuu = (int)3.5; //小数点以下切捨てでint変換
        string text = "" + 110; //文字列変換で"110"になる
        Debug.Log( seisuu + text  ); //加算ではなく連結で"3110"になる
    }
    void Update() {

    }
}
```

ミニ演習:mini057b

・3の整数の平均値を実数で表示したい
・public変数a、b、cを定義し、合計をdoubleにキャストしてから3で割って平均を得よう

作成例

```
using UnityEngine;
public class mini057b : MonoBehaviour {
    public int a = 1, b = 2, c = 3;
    void Start() {
        Debug.Log((double)(a + b + c) / 3);
    }
```

```
    void Update() {}
}
```

別解

```
using UnityEngine;
public class mini057b : MonoBehaviour {
    public int a = 1, b = 2, c = 3;
    void Start() {
        Debug.Log((a + b + c) / 3.0); //整数÷実数になる
    }
    void Update() {}
}
```

ミニ演習:mini057c

・西暦を受け取って何世紀か答えよう
・例:20世紀は1901年から2000年まで
・入力時にわかりやすいようにUnity側のパブリック変数の上にコメントを表示できる
・書式: [Header("コメント")] ※パブリック変数の定義のすぐ上に単独行で置くこと
・ヒント: 世紀 = (年 - 1) / 100 + 1

作成例

```
using UnityEngine;
public class mini057c : MonoBehaviour {
    [Header("西暦を入力してください")]
    public int a = 1901;
    void Start() {
        Debug.Log((a - 1) / 100 + 1 + "世紀です"); //計算後に連結
    }
    void Update() {}
}
```

参考:2つ以上のパブリック変数コメントも可能

```
using UnityEngine;
public class mini057c : MonoBehaviour {
    [Header("西暦を入力してください")]
    public int a = 1901;
    [Header("令和の和暦を入力してください")]
    public int b = 1;
    void Start() {
        Debug.Log((a - 1) / 100 + 1 + "世紀です"); //計算後に連結
        Debug.Log("西暦" + (b + 2018) + "年です"); //計算後に連結
    }
    void Update() {}
}
```

p.58 引数と戻り値

・Debug.Logはメソッドと呼ばれる処理を示す構文の一つ
・C言語などでは関数と呼んでいたが、C#/Javaなどのオブジェクト指向言語ではメソッドといいクラスの中におく
・メソッドは「O個以上の情報を受け取って何かを行いOまたは１個の情報を返す」仕組み
・Debug.Logの場合、１個の情報を受け取って何かを行いO個の情報を返す（何も返さない）
・この情報を受け取る仕組みを引数といい、メソッド名の後にカッコで示す
・引数のないメソッドもある
・１個の情報を返すメソッドでは、何型で返すのかを決める必要がある。これを戻り値型という
※ Debug.Logの「Debug」はLogメソッドが所属するクラス名（Unityが提供）
・平方根を返すMathf.Sqrtメソッドがあり、戻り値型はfloat、引数は１個でfloat型
 https://docs.unity3d.com/ja/2021.2/ScriptReference/Mathf.Sqrt.html
・このことを「戻り値型 メソッド名(引数型,…)」とも表現する 例：float Mathf.Sqrt(float)
・なお、引数型が実数(floatやdouble)の場合、整数を指定すると自動的に変換される
 例： Mathf.Sqrt(16) は Mathf.Sqrt(16.0)とみなされて4.0が返される
・数式にはメソッドの呼び出しを含むことができる
 例： float f = Mathf.Sqrt(16.0) + 1.0; //5.0になる
・引数でメソッドの呼び出しを含むことができる
 例： Debug.Log(Mathf.Sqrt(16.0)); //4を表示
※ Mathf.Sqrtの「Mathf」はSqrtメソッドが所属するクラス名でUnityが提供
※ なお、Unityが提供するクラスを利用するために「using UnityEngine;」を指定している

ミニ演習 mini059

・float型のパブリック変数で実数を受け取って、平方根を表示しよう

作成例

```
using UnityEngine;
public class mini059 : MonoBehaviour {
    [Header("実数を入力してください")]
    public float a = 0;
    void Start() {
        Debug.Log("平方根は" + Mathf.Sqrt(a)); //メソッドを呼んで連結
    }
    void Update() {}
}
```

p.59 複数の引数を渡す

・メソッドに渡すことのできる引数の型、数、順序は定義で決まっており、定義されていない引数の指定を行うとエラーになる
※ C#やUnityが提供するメソッドの引数の型、数、順序はオフィシャルドキュメントで確認できる
※ 同じ名前で引数の型、数、順序が異なるものを複数定義できる（オーバーロードという）
※ Mathf.Maxメソッドは引数をいくつでも指定できる特別なメソッドになっている（可変引数）
・複数の引数を指定できる場合はカンマ区切りで並べると良い
・Mathf.Maxの戻り値型はfloat、引数型はすべてfloatだが、整数値を指定でき、戻り値をint型変数に代入できる

※ カンマ区切りにおいてはカンマの後ろに空白を置くというチームルールにしていることがある

p.59 Chap1_12_1

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Chap1_12_1 : MonoBehaviour {
    void Start() {
        int value = Mathf.Max(10, 40, 20, 30); //最大値を得る
        Debug.Log( value );
    }
    void Update() {

    }
}
```

アレンジ演習：p.59 Chap1_12_1

・3つのfloat型実数をパブリック変数で受け取って最大値を表示するようにしよう

作成例

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Chap1_12_1 : MonoBehaviour {
    public float a, b, c;
    void Start() {
        float value = Mathf.Max(a, b, c); //最大値を得る
        Debug.Log( value );
    }
    void Update() {

    }
}
```

p.060 メソッドの「.」の前にあるものは何？

・前述のとおり、Debug.Logの「Debug」はLogメソッドが所属するクラス名で、これはUnityが提供するクラス
・前述のとおり、Mathf.Sqrtや、Mathf.Maxの「Mathf」はMaxメソッドが所属するクラス名で、これもUnityが提供するクラス
・メソッドは（C言語などの関数とは異なり）クラスに所属する
・なお、Start()やUpdate()もメソッドで、所属するクラス名がChap1_12_1ならば、フル名はChap1_12_1.Start()やChap1_12_1.Update()となる。
※ よって、実質的にはUnityシステムがクラス名.Start()を呼び出してくれている

※ なお、クラスが異なれば同名のメソッドを持つことができるので便利＆注意

p.061 静的メソッドとインスタンスメソッド

・クラスは多数のメソッドを持つ部品箱的なものもあるが、基本的にはメソッドとデータを持つ設計図のイメージでもある。
・汎用的に使われるメソッドは、部品箱的なクラスにおいて「クラス名.メソッド名」で呼び出せるように静的メソッドとして定義する。
・Debug.Logメソッド、Mathf.Sqrtメソッド、Mathf.Maxメソッドはすべて静的メソッド
※ 静的メソッドは定義において「static」をつけて区別する
・対して、メソッドとデータを持つ設計図のイメージになっているクラスでは、メソッドとデータをオブジェクトとしてまとめて扱いたいので「オブジェクト名.メソッド名」で呼び出す。
・このようなメソッドをインスタンスメソッドといい、こちらが標準的存在
※ 参考:インスタンスメソッドの例(C#のArrayListクラス)

```
ArrayList monsters = new ArrayList(); //モンスターを格納するリストを生成
monsters.Add("ヴェルドラ"); //monstersリストにAddメソッドで追加
monsters.Add("リムル"); //monstersリストにAddメソッドで追加
ArrayList players = new ArrayList(); //プレイヤーを格納するリストを生成
players.Add("私"); //playersリストにAddメソッドで追加
```

※ 上記のように本来のメソッドは「何に」をつけて示すもので、静的メソッドは例外的。
※ クラスはプログラマが定義でき、中にメソッドを記述するときに、インスタンスメソッドにするか、静的メソッドにするか判断できる

提出:アレンジ演習:p.59 Chap1_12_1

次回予告:p.62「エラーメッセージを読み解こう」から