講義メモ
・p.114「配列の書き方を覚えよう」

提出フォロー：アレンジ演習：chap3_5_2f

・乱数で1から9を5回得て、５本の横棒グラフを表示しよう
例：
7:■■■■■■■
2:■■
6:■■■■■■
5:■■■■■
4:■■■■
ヒント：1から9の乱数を得るには
  System.Random r = new System.Random(); //乱数クラスのオブジェクトを生成(１度でOK)
   :
  int n = r.Next(9) + 1; //1から9までのどれかになる

作成例

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

public class chap3_5_2f : MonoBehaviour {
    System.Random r = new System.Random(); //乱数クラスのオブジェクトを生成
    void Start() {
        for (int i = 0; i < 5; i++) { //5回繰返す
            int n = r.Next(9) + 1; //1から9までのどれかになる
            string s = n + ":"; //連結用の文字列を用意
            for (int j = 0; j < n; j++) { //n回繰り返す
                s += "■"; //文字列に■を連結
            } //内側のforブロック(繰返し内容)の終わり
            Debug.Log(s); //出来上がった文字列を表示
        } //外側のforブロック(繰返し内容)の終わり
    }
    void Update() {

    }
}
```

p.114 配列の書き方を覚えよう

・配列：同じ意味で同じ型のデータをまとめて扱える仕組みの一つで、添字（インデックス、指標）と呼ばれる番号を用いる軽量・検索が高速な仕掛け。
・生成書式： 型[] 配列名 = {値①, 値②, …};
・例： string[] name = {"リムル", "シュナ", "シオン"};
・配列を構成する変数を要素といい、配列名[添字]で扱う
・C#では添字は0スタートなので、配列名[0]が先頭要素になる
・配列は生成時の要素の数＝要素数を変更できない
・なお、末尾の要素は、配列名[要素数 - 1]になり、配列名[要素数]は存在しない
  ※ プログラム中で配列名[要素数]や配列名[負の数]にアクセスすると実行時エラーで異常終了する

p.115 chap3_6_1

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

```
using UnityEngine;

public class chap3_6_1 : MonoBehaviour {
    void Start() {
        string[] dirs = { "東", "西", "南", "北" };
        Debug.Log(dirs[1]); //配列dirsの要素[1]＝"西"を表示
    }
    void Update() {

    }
}
```

p.115 配列を作って利用する

・配列の添字は式や変数で与えることができる
・例：
```
string[] name = {"リムル", "シュナ", "シオン"}; int i = 2; Debug.Log(name[i]);
```

アレンジ演習：p.115 chap3_6_1

・dirs[1]ではなく、int型パブリック変数pを用いて、dirs[p]を表示しよう
・パブリック変数pの値は0から3とする（負の数か4以上にして異常終了することも試そう）

作成例

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

public class chap3_6_1 : MonoBehaviour {
    public int p = 0;
    void Start() {
        string[] dirs = { "東", "西", "南", "北" };
        Debug.Log(dirs[p]); //配列dirsの要素[p]を表示
    }
    void Update() {

    }
}
```

参考：実行時エラーの対処

・実行時エラーは文法エラーではないので、Visual Studioのエディタでは発生がわからない
　※ 実行時エラーが起こることが明らかな場合はエラーや警告が表示されることもある
・実行時エラーの発生場所はUnity側に表示されるメッセージで確認できる
・書式：
　[発生時刻]発生したエラーを示す例外クラス名:メッセージ
　実行クラス名.メソッド名(…)(at Assets/ソースファイル名:行番号)
・例：
```
[12:50:02]IndexOutOfRangeException: Index was outside the bounds of the array.
chap3_6_1.Start () (at Assets/chap3_6_1.cs:9)
```

p.116 配列の要素を置き換える

・配列の要素は単独の変数と同様に扱える
・よって、値を代入することで、格納されていた値を置き換えることが可能

p.116 string.Joinメソッド

・書式： string.Join(区切り文字列, 配列名)
・C#が提供するメソッドで、配列の全要素を連結した結果の文字列を返す
・この時、区切りに指定した文字列を要素間に挿入してくれる

p.116 chap3_6_2

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

public class chap3_6_2 : MonoBehaviour {
    void Start() {
        string[] dirs = { "東", "西", "南", "北" };
        dirs[0] = "真東"; //配列dirsの要素[0]に代入（書き換え）
        Debug.Log(string.Join(" ", dirs)); //配列dirsの全要素を連結表示(空白区切
り)
    }
    void Update() {

    }
}
```

アレンジ演習：p.115 chap3_6_2

・int型パブリック変数pと、string型パブリック変数sを用いて、dirs[p]にsを代入しよう
　例： p ← 2、s ← 真南
・パブリック変数pの値は0から3とする

作成例

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

public class chap3_6_2 : MonoBehaviour {
    public int p = 0;
    public string s = "";
    void Start() {
        string[] dirs = { "東", "西", "南", "北" };
        dirs[p] = s; //配列dirsの要素[p]に文字列sを代入（書き換え）
        Debug.Log(string.Join(" ", dirs)); //配列dirsの全要素を連結表示(空白区切
り)
    }
    void Update() {

    }
}
```

p.117 要素を入れずに配列を作成する

・p.114の書式は「配列の初期化」と呼ばれるもので、初期値の数により要素数が決まる
・この書式では要素数が大きい配列には不向きなので、要素数だけを指定して、初期値を与えずに空で生成する書式を用いる
・書式： 型[] 配列名 = new 型[要素数];

※ これは「配列の宣言」である「型[] 配列名;」と「配列の生成」である「配列名 = new 型[要素数];」を合わせたものなので、型を2回指定している。
・例： string[] name = new string[3]; //name[0]、name[1]、name[2]が生成される

p.117 new演算子

・newは「新しい」ではなく「生成せよ」のイメージで、メモリ上に領域を確保して利用できるようにする処理に汎用的に用いる
・配列の生成においても「指定した型×要素数分の領域」を確保することを示している。

p.117 chap3_6_3

・このままでは実行しても何も表示されないので、chap3_6_2の9行目を、末尾に入れると良い

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

public class chap3_6_3 : MonoBehaviour {
    void Start() {
        string[] dirs = new string[4]; //要素数4の空の配列を生成
        dirs[0] = "東"; //配列dirsの要素[0]に文字列を代入
        dirs[1] = "西"; //配列dirsの要素[0]に文字列を代入
        Debug.Log(string.Join(" ", dirs)); //配列dirsの全要素を連結表示(空白区切り)
    }
    void Update() {

    }
}
```

ミニ演習：フィボナッチ数列 mini117

・フィボナッチ数列とは要素[n]の値が要素[n - 2]と要素[n - 1]の和になっているもので、各種の統計処理やシミュレーションに利用される
・要素数5のフィボナッチ数列を作成しよう。要素[0]と要素[1]の値は1とする
・配列の型はintで良い
・できた数列をカンマ区切りで連結表示しよう
・実行結果：1,1,2,3,5

作成例

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

public class mini117 : MonoBehaviour {
    void Start() {
        int[] fib = new int[5]; //要素数5のフィボナッチ数列を生成
        fib[0] = fib[1] = 1; //配列fibの要素[0]と要素[1]に1を代入
        fib[2] = fib[0] + fib[1]; //前２要素の和を代入
        fib[3] = fib[1] + fib[2]; //前２要素の和を代入
        fib[4] = fib[2] + fib[3]; //前２要素の和を代入
        Debug.Log(string.Join(",", fib)); //配列fibの全要素を連結表示(カンマ区切り)
    }
    void Update() {
```

```
    }
}
```

p.118 foreach文

・配列などの複数のデータを持つ構造に用いる専用の繰返し文で「拡張for文」ともいう
・配列の場合の書式： foreach（型 作業変数名 in 配列名）{ 繰返し内容 }
・作業変数名は自由で、その型としては通常は配列の型を指定する
　※ 作業変数の型は配列の型で決めることができるので、自動指定を意味する「var」にもできる
・繰返し内容では、配列から１要素ずつが作業変数にコピーされて渡される
・よって、繰返し内容では、要素の代わりに作業変数を用いる。添字は用いない
・繰返し内容では作業変数への代入はできない（エラーになる）

p.118 chap3_7_1

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

public class chap3_7_1 : MonoBehaviour {
    void Start() {
        string[] dirs = { "東", "西", "南", "北" };
        foreach (string d in dirs) { //配列dirsの全要素について作業変数dを用いて繰返す
            Debug.Log(d + "方向");
        }
    }
    void Update() {

    }
}
```

ミニ演習：フィボナッチ数列 mini117 続き

・出来上がった数列の要素値の合計をforeachで得て表示しよう

作成例

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

public class mini117 : MonoBehaviour {
    void Start() {
        int[] fib = new int[5]; //要素数5のフィボナッチ数列を生成
        fib[0] = fib[1] = 1; //配列fibの要素[0]と要素[1]に1を代入
        fib[2] = fib[0] + fib[1]; //前２要素の和を代入
        fib[3] = fib[1] + fib[2]; //前２要素の和を代入

        fib[4] = fib[2] + fib[3]; //前２要素の和を代入
        Debug.Log(string.Join(",", fib)); //配列fibの全要素を連結表示(カンマ区切り)
        int sum = 0; //【以下追加】合計用
        foreach(var work in fib) { //数列の全要素について作業変数を用いて繰返す
            sum += work; //合計に足し込む
        }
```

```
        Debug.Log("計＝" + sum); //合計を表示
    }
    void Update() {

    }
}
```

p.119 for文を使ってインデックスを指定する

・foreachでは表現できない処理を行いたい場合や、繰返しの中で配列の要素値を変更したい場合などは、for文を用いると良い
・for文のカウンタ用の変数を添字として用いると、可読性が高くなる
・利用書式例： for(int i = 0; i < 要素数; i++) { 配列[i]の処理 }
・よって、foreachで表現できる処理はすべて上記の書式に置き換えることが可能
・また、繰返しの中で添字を用いたい場合も、for文で記述すると良い

p.119 chap3_7_2

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

public class chap3_7_2 : MonoBehaviour {
    void Start() {
        string[] dirs = { "東", "西", "南", "北" };
        for (int cnt = 0; cnt < 4; cnt++) { //要素数の分、変数cntを用いて繰返す
            Debug.Log(dirs[cnt] + "方向");
        }
    }
    void Update() {

    }
}
```

アレンジ演習：chap3_7_2

・添え字を付けて「0＝東方向」「1＝西方向」「2＝南方向」「3＝北方向」と表示しよう
・これはforeachでは難しいが、forでは簡単

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

public class chap3_7_2 : MonoBehaviour {
    void Start() {
        string[] dirs = { "東", "西", "南", "北" };
        for (int cnt = 0; cnt < 4; cnt++) { //要素数の分、変数cntを用いて繰返す
            Debug.Log(cnt + "＝" + dirs[cnt] + "方向"); //【変更】
        }
    }
    void Update() {

    }
}
```

補足：配列の要素数

・配列の要素数は、配列名.Lengthで得られる。よって、for文で扱うときには

  for(int i = 0; i < 配列名.Length; i++) { 配列[i]の処理 }
  としておけば、配列の要素数が変わっても、書き換える必要がなくなるので便利
※ 配列の要素数は、必ず配列名.Lengthで得ることというルールにしている場合もある

アレンジ演習：chap3_7_2

・配列の要素数を配列名.Lengthで得るようにしよう
・すると、配列の末尾に要素"中央"を加えても、処理の変更は不要になることを確認しよう

作成例

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

public class chap3_7_2 : MonoBehaviour {
    void Start() {
        string[] dirs = { "東", "西", "南", "北", "中央" };
        for (int cnt = 0; cnt < dirs.Length; cnt++) { //要素数の分、変数cntを用
いて繰返す
            Debug.Log(cnt + "＝" + dirs[cnt] + "方向");
        }
    }
    void Update() {

    }
}
```

ミニ演習：フィボナッチ数列 mini117 続き

・数列の要素数を10にして、要素[2]以降の値のセットをfor文による繰返しで行うようにしよう
・ヒント： for(int i = 2; i < fib.Length; i++) { fib[i] = fib[i - 2] + fib[i - 1]; }
・実行結果： 1,1,2,3,5,8,13,21,34,55
・合計は、そのままforeachで得ると良い

提出：ミニ演習：フィボナッチ数列 mini117 続き

次回予告：p.120 chap3_8_1（配列とforeachによる２重ループ）から