講義メモ
・p.54「入力結果をちょっと加工して表示する」から

提出フォロー：アレンジ演習： p.46 chap1_8_2 続き2

・税率を100分率で入力できるようにしよう
・例えば、8と入力したら価格を1.08倍した結果になること
・ヒント： 売値＝価格×(100＋税率)÷100

作成例

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

public class Chap1_8_2 : MonoBehaviour {
    public double kakaku = 100, rate; //パブリックな実数型変数の初期化と宣言
    void Start() {
        double urine = kakaku * (100 + rate) / 100; //実数型変数の計算式に
よる初期化
        Debug.Log(urine); //実数型変数の値を出力
    }
    void Update() {
    }
}
```

補足：計算順序と誤差

・基本的に誤差が出やすい除算は最後に行うと良い
・先に誤差が出た状態で乗算を行うと誤差が増えてしまう
・また、整数演算では先に除算を行うと計算結果が大きく損なわれることがある（後述）

p.54 入力結果をちょっと加工して表示する

・C#では演算子に複数の作用を持たせることができる。
・これをオーバーロード（多重定義）という
・どの作用が行われるかは演算の対象の型によって決まる
・2項＋演算子の場合：
  ① 双方が加算可能（数値）なら加算
  ② どちらかまたは両方が文字列なら連結
  ③ ①でも②でもなければエラー（ただし、プログラムで追加定義が可能）
・複数の2項＋演算子を使った式の場合、途中段階でどの型になるかで動作が変わるので注意

p.54 Chap1_10_2

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

```
public class Chap1_10_2 : MonoBehaviour {
    public string text; //パブリック変数の宣言
    void Start() {
        Debug.Log( "入力は" + text ); //文字列変数の値を連結して出力
    }
    void Update() {

    }
}
```

アレンジ演習:Chap1_10_2 ①

・2項＋演算子のどちらかが文字列なら連結であることを確認しよう
① 文字列＋実数　例:"円周率は" + 3.14 を表示
② 実数＋文字列　例: double pi = 3.14; pi + "が円周率" を表示

作成例

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

public class Chap1_10_2 : MonoBehaviour {
    public string text; //パブリック変数の宣言
    void Start() {
        Debug.Log( "入力は" + text ); //文字列変数の値を連結して出力
        Debug.Log( "円周率は" + 3.14 ); //文字列＋実数で連結して出力
        double pi = 3.14; //実数型変数を初期化
        Debug.Log( pi + "が円周率" ); //実数変数＋文字列で連結して出力
    }
    void Update() {

    }
}
```

アレンジ演習:Chap1_10_2 ②

・複数の2項＋演算子を使った式の場合、途中段階でどの型になるかで動作が変わることを確認しよう
① 実数＋実数＋文字列は、実数の和と文字列の連結　例:1.1 + 2.04 + "は円周率" を表示
② 実数＋文字列＋実数は、実数と文字列の連結に実数を連結　例: 1.5 + "個の重さは" + 2.5 を表示
③ よって、文字列＋実数＋実数は、文字列と実数の連結に実数を連結してしまう。カッコを用いて優先順位を調整することで)
 double x = 1.5, y = 2.2;
 について「和は3.7」と表示する処理を追加しよう。

作成例

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

public class Chap1_10_2 : MonoBehaviour {
    public string text; //パブリック変数の宣言
    void Start() {
        Debug.Log( "入力は" + text ); //文字列変数の値を連結して出力
        Debug.Log( "円周率は" + 3.14 ); //文字列＋実数で連結して出力
        double pi = 3.14; //実数型変数を初期化
        Debug.Log( pi + "が円周率" ); //実数変数＋文字列で連結して出力
        Debug.Log( 1.1 + 2.04 + "は円周率" ); //実数の和と文字列の連結
        Debug.Log( 1.5 + "個の重さは" + 2.5 ); //実数と文字列の連結に実数を連
結
        double x = 1.5, y = 2.2; //実数型変数を初期化
        Debug.Log( "和は" + (x + y) ); //加算後の実数を文字列に連結
    }
    void Update() {

    }
}
```

アレンジ演習：p.46/47 Chap1_8_2/3をベースに

・「●円で税率●%なので●円」と表示しよう

作成例

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

public class Chap1_8_2 : MonoBehaviour {
    public double kakaku = 100, rate; //パブリックな実数型変数の初期化と宣言
    void Start() {
        double urine = kakaku * (100 + rate) / 100; //実数型変数の計算式に
よる初期化
        Debug.Log(kakaku + "円で税率" + rate + "%なので" + urine + "円");
//【変更】連結出力
    }
    void Update() {
    }
}
```

作成例：別解

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

```
using UnityEngine;

public class Chap1_8_2 : MonoBehaviour {
    public double kakaku = 100, rate; //パブリックな実数型変数の初期化と宣言
    void Start() {
        double urine = kakaku * (100 + rate) / 100; //実数型変数の計算式による初期化
        string why = kakaku + "円で税率" + rate + "%なので"; //【追加】理由の文字列
        Debug.Log(why + urine + "円"); //【変更】連結出力
    }
    void Update() {
    }
}
```

参考: Unity UIについて

・通常のアプリケーションにおいて、実行中に利用者（プレイヤー）が値を与えるには、そのためのユーザインタフェース(UI)が必要
・特にグラフィック系のアプリでは、キーボードを利用して入力するのは不適切な場合が多いし、入力のチェックの問題もある
・よって、専用のユーザインタフェースツールを用いて、アプリの作りに合わせたUI作り込むのが一般的
・Unityの場合、Unityが用意しているUnity UIを用いることができる
 https://docs.unity3d.com/ja/2022.3/Manual/UIInteractionComponents.html

p.56 組み込み型とは

・doubleは実数を、stringは文字列を扱うためのデータ型で、C#があらかじめ提供しているので、組み込み型ともいう
・参考:4章以降で学習するクラスを用いると、プログラマが型を定義できる。これをユーザ定義型という
・実数型:正式には浮動小数点数型といい、doubleとfloatがある
 ① double:倍精度浮動小数点数型・1個64ビット。実数値のデフォルト型
 ② float :単精度浮動小数点数型・1個32ビット。メモリを節約できるのでUnity向き
・整数型ー符号有:負の数も扱える
 ① int:汎用整数型・１個32ビット。整数値のデフォルト型。約-21億から21億まで。
 ② long:長整数型・１個64ビット。intの範囲を超える整数値用
 ③ short:短整数型・１個16ビット。メモリを節約できる。-32768～32767
 ④ sbyte:符号付きバイト型・１個8ビット。メモリをさらに節約できる。-128～127
・整数型ー符号無:0と整数のみなので、正の数の範囲が上記①～④の倍になる
 ① uint:符号無汎用整数型・１個32ビット。0から約42億まで。
 ② ulong:符号無長整数型・１個64ビット。uintの範囲を超える整数値用
 ③ ushort:符号無短整数型・１個16ビット。メモリを節約できる。0～65535
 ④ byte:バイト型・１個8ビット。メモリをさらに節約できる。0～255
・各型の範囲はC#が提供する定数で得られる。最大値は「型.MaxValue」最小値は「型.MinValue」

ミニ演習:mini056

・各型の最大値「型.MaxValue」最小値「型.MinValue」を確認しよう

作成例

```
using UnityEngine;
public class mini056 : MonoBehaviour {
    void Start(){
        Debug.Log("double Max:" + double.MaxValue + " Min:" +
double.MinValue);
        Debug.Log("float  Max:" +  float.MaxValue + " Min:" +
float.MinValue);
        Debug.Log("int    Max:" +    int.MaxValue + " Min:" +
int.MinValue);
        Debug.Log("long   Max:" +   long.MaxValue + " Min:" +
long.MinValue);
        Debug.Log("short  Max:" +  short.MaxValue + " Min:" +
short.MinValue);
        Debug.Log("sbyte  Max:" +  sbyte.MaxValue + " Min:" +
sbyte.MinValue);
        Debug.Log("uint   Max:" +   uint.MaxValue + " Min:" +
uint.MinValue);
        Debug.Log("ulong  Max:" +  ulong.MaxValue + " Min:" +
ulong.MinValue);
        Debug.Log("ushort Max:" + ushort.MaxValue + " Min:" +
ushort.MinValue);
        Debug.Log("byte   Max:" +   byte.MaxValue + " Min:" +
byte.MinValue);
    }
    void Update() { }
}
```

p.56 組み込み型とは（つづき）

・char：文字型。1文字分の情報で16ビット。文字リテラル(「'」で1文字を囲んだもの)を代入できる
・string：文字列型。0文字以上の文字の並びの情報で大きさはない。文字列リテラル(「"」で0文字以上を囲んだもの)を代入できる
・bool：論理型。1ビット分の情報。真偽値リテラルであるtrueとfalseを代入できる(後述)
　※ 2進数の1と0に該当するが、C/C++のように0や1で代用することはできない

参考：その他の組み込み型と.NET型

・誤差を極力減らしたい場合（例：金銭計算）はdouble型の代わりにdecimal型を用いることができる。1個128ビットで、decimal型実数を代入して用いるので、特殊用途。
・各型には、C#の基盤である.NETフレームワークで規定している型名「.NET型」があり、エラーメッセージなどに用いられる。基本的には「System.型の名の先頭を大文字にしたもの」だが、下記の例外がある
int ⇒ System.Int32、long ⇒ System.Int64、short ⇒ System.Int16、float ⇒ System.Single
uint ⇒ System.UInt32、ulong ⇒ System.UInt64、ushort ⇒ System.UInt16、bool ⇒ System.Boolean

p.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(正確は小数点以下を取り除いた値)になる(四捨五入ではない)

ミニ演習:mini057

・パブリック変数で実数を受け取ってint型の変数にキャストして代入し、その値を表示しよう

作成例

```
using UnityEngine;
public class mini057 : MonoBehaviour {
    public double num;
    void Start() {
        int i = (int)num; //得た実数値をint型にキャストして代入
        Debug.Log(i);
    }
    void Update() { }
}
```

p.56 型を変換する(つづき)

・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

参考:数字列から整数/実数型への変換

・数字のみの文字列を整数に変換するにはC#が提供している int.Parse(文字列)を使うことができる
 例: string s = "365"; int i = int.Parse(s); //整数365になる

提出:ミニ演習:mini057

次回予告:p.57 Chap1_11_1から