講義メモ

・p.30「標準書式指定文字を使った書式の指定」から

提出フォロー：課題：アレンジ演習 **p.30 text05.cs** 改良

・アレンジ演習で追加した部分を掛け算を用いて下記のように改良しよう
  ・最後のWriteLineの後で、xの値を100、yの値を1に変更して、再度、同じ書式で表示しよう
  ・xの値の10倍、yの値の10倍を、再度、同じ書式で表示しよう
  ・xの値の100倍、yの値の100倍を、再度、同じ書式で表示しよう

作成例

```
//提出課題：アレンジ演習p.30 text05.cs
using System; //System名前空間の利用を宣言
class Text05 //クラスの定義
{ //クラス定義の開始
    public static void Main() //開始時に動作するメソッドMain
    { //メソッド定義の開始
        int x = 10, y = 123456789;
        Console.WriteLine("12345678901234567890123456789O"); //目盛りを表示
        Console.WriteLine("{0, 10}", "abc"); //"       abc"(前に空き7個)
        Console.WriteLine("{0, 5}", "def"); //"  def"(前に空き2個)
        Console.WriteLine("{0, 0}", "ghi"); //"ghi"(桁数指定は無視)
        Console.WriteLine("{0, 10}{1, 10}", "あ", "い"); //"         あ         い"
        Console.WriteLine("{0, -10}{1, -10}", "あ", "い"); //"あ         い         "
        Console.WriteLine("x = {0, 5}, y = {1, 3}", x, y); //"x =    10, y = 123456789"
        x = 100; y = 1; //【以下追加】
        Console.WriteLine("x = {0, 5}, y = {1, 3}", x, y); //"x =   100, y =   1"
        Console.WriteLine("x = {0, 5}, y = {1, 3}", x * 10, y * 10); //"x =  1000, y =  10"
        Console.WriteLine("x = {0, 5}, y = {1, 3}", x * 100, y * 100); //"x = 10000, y = 100"
    } //メソッド定義の終了
} //クラス定義の終了
```

p.30 標準書式指定文字を使った書式の指定

・C#が用意している標準書式指定文字を用いると、表示形式を効率的に指定出来て便利
・書式：{番号, 桁数:標準書式指定文字}
  例：{0, 10:X} // 0番の値を最小10桁で16進数表示
・なお、表2.1にある標準書式指定文字の中にはあまり利用されていないものもあるので、丸暗記は不要
・便利なのは「C」の通貨形式（日本語環境では前に「￥」をつけて3桁カンマ区切り）や、「P」のパーセント形式（小数値を与えるとパーセント形式で表示）や、「X」の16進数変換、「F」＋小数点以下の表示桁数指定(かつ四捨五入)など
・なお「E」で表示される「1.230000E+002」は指数表記といい、1.23×10^{2}を意味する。非常に大きな/小さな値や非常に0に近い値の表記に使われる
  例：1234567890123456789 → $1.2345678901234567890\times10^{18}$ → 1.2345678901234567890E+018
  例：0.000000000123456789 → $1.2345678901234567890\times10^{-18}$ → 1.2345678901234567890E-018
・Visual Studioでは「￥」は「\」として表示されるが、同じ文字であり、環境によって異なる

p.31 text06.cs

```
//p.31 text06.cs
using System; //System名前空間の利用を宣言
```

```
class Text06 //クラスの定義
{ //クラス定義の開始
    public static void Main() //開始時に動作するメソッドMain
    { //メソッド定義の開始
        Console.WriteLine("D{0, 10:D}", 123);        //"D        123" 10進数表示
        Console.WriteLine("X{0, 10:X}", 123);        //"X         7B" 16進数表示
        Console.WriteLine("E{0, 10:E}", 123.456);    //"E1.234560E+002" 指数表記
        Console.WriteLine("F{0, 10:F}", 123.456);    //"F    123.46" 小数点以下2桁四捨五入
        Console.WriteLine("F{0, 10:F5}",123.456);    //"F 123.45600" 小数点以下5桁四捨五入
        Console.WriteLine("C{0, 10:C}", 123456);     //"C  ¥123,456" 通貨3桁カンマ区切り
        Console.WriteLine("N{0, 10:N}", 1234.4568); //"N  1,234.46" 3桁カンマ区切り+F
        Console.WriteLine("P{0, 10:P}", 0.05);       //"P      5.00%" 小数点以下2桁パーセント
        Console.WriteLine("G{0, 10:G}", 123.456);    //"G   123.456" 通常、Fと同じ
    } //メソッド定義の終了
} //クラス定義の終了
```

アレンジ演習 **p.31 text06.cs**

・追加として、3.14259265を、小数点以下1桁、2桁、3桁までにして1行で表示しよう
　実行結果は「3.1 3.14 3.143」となればOK

作成例

```
//アレンジ演習 p.31 text06.cs
using System; //System名前空間の利用を宣言
class Text06 //クラスの定義
{ //クラス定義の開始
    public static void Main() //開始時に動作するメソッドMain
    { //メソッド定義の開始
        Console.WriteLine("D{0, 10:D}", 123);        //"D        123" 10進数表示
        Console.WriteLine("X{0, 10:X}", 123);        //"X         7B" 16進数表示
        Console.WriteLine("E{0, 10:E}", 123.456);    //"E1.234560E+002" 指数表記
        Console.WriteLine("F{0, 10:F}", 123.456);    //"F    123.46" 小数点以下2桁四捨五入
        Console.WriteLine("F{0, 10:F5}",123.456);    //"F 123.45600" 小数点以下5桁四捨五入
        Console.WriteLine("C{0, 10:C}", 123456);     //"C  ¥123,456" 通貨3桁カンマ区切り
        Console.WriteLine("N{0, 10:N}", 1234.4568); //"N  1,234.46" 3桁カンマ区切り+F
        Console.WriteLine("P{0, 10:P}", 0.05);       //"P      5.00%" 小数点以下2桁パーセント
        Console.WriteLine("G{0, 10:G}", 123.456);    //"G   123.456" 通常、Fと同じ
        //【追加】"3.1 3.14 3.143"を表示（桁数は0で良い）
        Console.WriteLine("{0, 0:F1} {0, 0:F} {0, 0:F3}", 3.14259265);
    } //メソッド定義の終了
} //クラス定義の終了
```

p.32 カスタム書式指定文字を使った書式の設定

・書式のパターンを指定するための表記法の一つがプレースホルダで、この表記に用いるのがカスタム書式指定文字
・プレースホルダには、ゼロプレースホルダと、桁プレースホルダがある

・ゼロプレースホルダは指定したゼロの数より値が短い場合に、ゼロで埋める方式で、例えば、古くからあるゲームのハイスコアの表示形式である「000,012,345」というような表示に便利
・書式：{番号, 桁数:ゼロプレースホルダ}
　例：{0, 0:000.00} // 0番の値が000.00より短ければ前後に0を入れて6桁にする。例：1.2 → “001.20”
・桁プレースホルダは指定した＃の数より値が短い場合に、空白で埋める方式
・書式：{番号, 桁数:桁プレースホルダ}
　例：{0, 0:###.##} // 0番の値が###.##より短ければ前後に空白を入れて6桁にする。例：1.2 → “ 1.2 “
・ゼロプレースホルダと桁プレースホルダは混在が可能
・プレースホルダの整数部に「,」を入れると、3桁カンマ区切りになる
　例：{0, 0:0,0} // 0番の値を3桁カンマ区切りにする。例：1234567 → “1,234,567”
・また「0」「#」以外の文字を混在させて埋め込むことも可能
・なお「#」をカスタム書式指定文字ではなく、埋め込む文字にしたい場合は ‘#’とすればOK

p.33 text07.cs

```
//p.33 text07.cs
using System; //System名前空間の利用を宣言
class Text07 //クラスの定義
{ //クラス定義の開始
    public static void Main() //開始時に動作するメソッドMain
    { //メソッド定義の開始
        Console.WriteLine("{0, 10:0000.00}", 1.2); //"   0001.20"
        Console.WriteLine("{0, 10:####.##}", 1.2); //"       1.2 "
        Console.WriteLine(); //1行開ける
        Console.WriteLine("{0, 10:0,0}", 123456); //"   123,456"
        Console.WriteLine("{0, 10:#,#}", 123456); //"   123,456" ※上と同じ
        Console.WriteLine(); //1行開ける
        Console.WriteLine("{0, 10:0,00000000.00}", 123456); //"000,123,456.00"
        Console.WriteLine("{0, 10:#,########.##}", 123456); //"   123,456"
        Console.WriteLine(); //1行開ける
        Console.WriteLine("{0:(000)####-####}", 1132104566); //"(011)3210-4566"
        Console.WriteLine("{0, 20:'##'0.00}", 123456); //"         ##123456.00"
    } //メソッド定義の終了
} //クラス定義の終了
```

アレンジ演習 **p.33 text07.cs**

・追加として、1100030005を「011-0003-0005」と表示させよう

作成例

```
//アレンジ演習 p.33 text07.cs
using System; //System名前空間の利用を宣言
class Text07 //クラスの定義
{ //クラス定義の開始
    public static void Main() //開始時に動作するメソッドMain
    { //メソッド定義の開始
        Console.WriteLine("{0, 10:0000.00}", 1.2); //"   0001.20"
        Console.WriteLine("{0, 10:####.##}", 1.2); //"       1.2 "
        Console.WriteLine(); //1行開ける
        Console.WriteLine("{0, 10:0,0}", 123456); //"   123,456"
        Console.WriteLine("{0, 10:#,#}", 123456); //"   123,456" ※上と同じ
```

```
        Console.WriteLine(); //1行開ける
        Console.WriteLine("{0, 10:0,00000000.00}", 123456); //"000,123,456.00"
        Console.WriteLine("{0, 10:#,########.##}", 123456); //"    123,456"
        Console.WriteLine(); //1行開ける
        Console.WriteLine("{0:(000)####-####}", 1132104566); //"(011)3210-4566"
        Console.WriteLine("{0, 20:'##'0.00}", 123456); //"         ##123456.00"
        //【追加】
        Console.WriteLine("{0:0##-####-####}", 1100030005); //"011-0003-0005"
    } //メソッド定義の終了
} //クラス定義の終了
```

p.34 ユーザの入力を知る：文字列型 **string** について

・整数用のint型と同様に、文字列用の型としてstringがある。
・string型の変数を宣言すると、文字列を格納できる。
・文字列とは0文字以上の文字の並びで、プログラムの中に書く時は「”」で挟んで記述する

p.34 ユーザの入力を知る：**Console.ReadLine**

・コンソールに出力する Console.WriteLine/Writeの反対に、コンソールから入力するが Console.ReadLine
・Console.WriteLine/Writeとは異なりカッコ内（引数という：後述）は不要で、string型の文字列変数に代入する形式
・主な書式： string型の文字列変数 = Console.ReadLine();
・実行すると、コンソールが入力待ち状態になるので「何を待っているのか」がわかるように、ガイドとなる文字列を事前に表示すると良い
・この時、Console.WriteLinedではなく、Console.Write を使うと「お名前：」というように、入力ガイドになって便利

p.35 readline01.cs

```
//p.35 readline01.cs
using System; //System名前空間の利用を宣言
class ReadLine01 //クラスの定義
{ //クラス定義の開始
    public static void Main() //開始時に動作するメソッドMain
    { //メソッド定義の開始
        string name; //文字列型変数の宣言
        Console.Write("あなたのお名前は何ですか--- "); //改行しない
        name = Console.ReadLine(); //コンソールから入力した文字列を変数に代入
        Console.WriteLine("あなたのお名前は{0}さんですか", name); //変数値を表示
    } //メソッド定義の終了
} //クラス定義の終了
```

アレンジ演習 **p.35 readline01.cs**

・追加で「では、苗字は何ですか--- 」と表示して、別の文字列変数にコンソールから入力しよう
・そして「つまり、〇〇 〇〇 さんですね」と、苗字と名前の両方を表示しよう

作成例

```
//アレンジ演習 p.35 readline01.cs
using System; //System名前空間の利用を宣言
class ReadLine01 //クラスの定義
{ //クラス定義の開始
```

```
    public static void Main() //開始時に動作するメソッドMain
    { //メソッド定義の開始
        string name; //文字列型変数の宣言
        Console.Write("あなたのお名前は何ですか--- "); //改行しない
        name = Console.ReadLine(); //コンソールから入力した文字列を変数に代入
        Console.WriteLine("あなたのお名前は{0}さんですか", name); //変数値を表示
        //【以下追加】
        string fname; //文字列型変数の宣言(苗字用)
        Console.Write("では、苗字は何ですか--- "); //改行しない
        fname = Console.ReadLine(); //コンソールから入力した文字列を変数に代入
        Console.WriteLine("つまり、{0} {1} さんですか", fname, name); //2変数値を表示
    } //メソッド定義の終了
} //クラス定義の終了
```

p.36 練習問題：作り方 **ex02.cs**

・プログラム名、クラス名は自由（講師は ex02.cs とします）
・画面に表示する内容は指定されていないので、無表示で入力待ちにして構わない
・入力されたら、その内容を表示するだけでOK

作成例1

```
//p.36 練習問題：ex02.cs
using System; //System名前空間の利用を宣言
class Ex02 //クラスの定義
{ //クラス定義の開始
    public static void Main() //開始時に動作するメソッドMain
    { //メソッド定義の開始
        string line; //1行入力用の文字列型変数の宣言
        line = Console.ReadLine(); //コンソールから入力した文字列を変数に代入
        Console.WriteLine(line); //変数値を表示
    } //メソッド定義の終了
} //クラス定義の終了
```

作成例2：文字列型変数を**ReadLine**で得た文字列で初期化できる

```
//p.36 練習問題：ex02.cs
using System; //System名前空間の利用を宣言
class Ex02 //クラスの定義
{ //クラス定義の開始
    public static void Main() //開始時に動作するメソッドMain
    { //メソッド定義の開始
        string line = Console.ReadLine(); //コンソールから入力した文字列で変数を初期化
        Console.WriteLine(line); //変数値を表示
    } //メソッド定義の終了
} //クラス定義の終了
```

作成例3：**ReadLine**で得た文字列を直接**WriteLine**できる

```
//p.36 練習問題：ex02.cs
using System; //System名前空間の利用を宣言
```

```
class Ex02 //クラスの定義
{ //クラス定義の開始
    public static void Main() //開始時に動作するメソッドMain
    { //メソッド定義の開始
        Console.WriteLine(Console.ReadLine()); //コンソールから入力した文字列を表示
    } //メソッド定義の終了
} //クラス定義の終了
```

第3章 変数とデータ型

p.37 3.1 変数の初期化

・すでに説明したとおり、変数の宣言と同時に初期値を代入することを変数の初期化という
・初期化に用いた値を初期値という
・初期化では、値のみではなく、他の変数の値や、計算式や、値を返す処理（メソッド）を指定できる
例：

```
int i1 = 10; //整数値のよる初期化
string s1 = ”ABC”; //文字列値による初期化
int i2 = 10 + 20; //式による初期化（加算されて30になる）
int i3 = i2 + 30; //変数を含む式による初期化（加算されて60になる）
string s2 = i3 + s1; //型の異なる変数を含む式による初期化（連結されて”60ABC”になる）
string s3 = Console.ReadLine(); //コンソールから入力した文字列で初期化（s3の値は実行まで不定）
```

ミニ演習 **mini037.cs**

・上記の例を試そう
・初期化する都度、その変数の値を表示して確認しよう

作成例

```
//ミニ演習 mini037.cs
using System; //System名前空間の利用を宣言
class Mini037 //クラスの定義
{ //クラス定義の開始
    public static void Main() //開始時に動作するメソッドMain
    { //メソッド定義の開始
        int i1 = 10; //整数値のよる初期化
        Console.WriteLine("i1 : {0}", i1); //値を表示
        string s1 = "ABC"; //文字列値による初期化
        Console.WriteLine("s1 : {0}", s1); //値を表示
        int i2 = 10 + 20; //式による初期化（加算されて30になる）
        Console.WriteLine("i2 : {0}", i2); //値を表示
        int i3 = i2 + 30; //変数を含む式による初期化（加算されて60になる）
        Console.WriteLine("i3 : {0}", i3); //値を表示
        string s2 = i3 + s1; //型の異なる変数を含む式による初期化（連結されて”60ABC”になる）
        Console.WriteLine("s2 : {0}", s2); //値を表示
        string s3 = Console.ReadLine(); //コンソールから入力した文字列で初期化（s3の値は実行まで不定）
        Console.WriteLine("s3 : {0}", s3); //値を表示
    } //メソッド定義の終了
} //クラス定義の終了
```

p.38 **3.1** 変数の初期化：組み込み型とユーザ定義型

・int型や、string型はC#が提供する型として、あらかじめ組み込まれているので、組み込み型という
・なお、後述する列挙型(p.66)や、7章以降で説明するクラスをなどを用いると、プログラマが型を作ることができ、そのような型をユーザ定義型という

p.38 **3.1** 変数の初期化：ローカル変数

・変数は定義や初期化を行った場所によって、扱いが変わる
・Mainなどのメソッドの中で定義や初期化を行った変数をローカル変数という
・ローカル変数は初期化するか、最初の値を代入するまで使えない（値が不定なので）
・この場合「変数 '〇' は割り当てられていますが、その値は使用されていません」というエラーになる
・ローカル変数は定義や初期化を行った時点以降、メソッド内でのみ有効

p.39 定数

・変数の初期化の前に「const 」を記述すると、定数の初期化となる
・定数となった変数は値の変更が禁止される
・定数には代入できず、すると文法エラーになる
・定数は主に、値に名前を付けたり、ゲームの調整値（速度や画面サイズのようなパラメータ）の記述に用いる
  例：const string gamename = ”テトリス”; //ゲーム名を定数で定義しておけば後でいろいろ使える
  例：const int fps = 10; //FPS値を定数で定義して用いれば、初期値の変更でゲーム速度を調整できる
・定数は必ず初期化すること

ミニ演習 **mini039.cs**

・int型の定数として TAX を10で初期化しよう（税率10%）
・これを用いて、500円の商品の税込み価格を表示しよう
  例「500円 税込み 550円」

次回予告：

・p.40「値型と参照型」から