C#プログラミング

C/C#演習

目次

１．プログラム概論

＜プログラミングの定義＞

初めに、何をもってプログラミングというのか、その定義について確認しておきましょう。

まずは、「コンピューター(パソコン)」についてお話しします。

コンピュータはさまざまな計算とその結果を保存することが出来る機械です。

１．プログラム概論

＜プログラミングの定義＞

プログラミングは、コンピューターに指示を出すことを言います。

１．プログラム概論

＜プログラミングの定義＞

この時に、私たち人間の言葉ではコンピューターは理解出来ません。そこで、コンピューターに理解出来る言葉を使って、コンピューターに指示を出す必要があります。

その言葉の総称を「プログラミング言語」といいます。

指示（プログラミング言語）

計算結果（OK/NG/画面表示等）

１．プログラム概論

＜プログラムの種類(代表)＞

スクリプト言語

言語名	用途・特徴
Python	AI、データ分析、Web開発で使用。 書き方に制限が多く、可読性が高い。
JavaScript (HTML)	Webブラウザ上で動く言語。 Webアプリ制作に不可欠、需要が高い。
Ruby	初心者にも分かりやすい言語。 Web開発に使用されている。
PHP	Webサーバー上で動く言語。 バックエンド側では主流。
Perl	Webサーバー上で動く言語。 PHPが出来て、化石状態。

コンパイル言語

言語名	用途・特徴
C	組み込み機器で多く使用な手続き言語。 とにかく実行が早い。
C++	CやJavaの拡張。ハイパフォーマンスな アプリ制作(ゲームなど)に強い。
Java	OSによらず動作する言語。基幹系の システムに多く採用。
C#	C++を扱いやすくした言語。 Windowsアプリやゲーム開発で採用。
Swift	iOSのアプリ開発で使用する言語。 Apple独自でとっつきにくいが、人気。

１．プログラム概論

●スクリプト言語について

以下の特徴があります。

・実行するには？

→プログラムを書いて、すぐに実行できる。

・実行速度

→1行ずつパソコンが解析しながら実行するため、遅い。

・デバッグ

→実行しながら都度、確認。

試しにHTML言語をメモ帳で作ってみましょう！

１．プログラム概論

●コンパイル言語について

以下の特徴があります。

・実行するには？

→プログラムを書いて、事前にコンパイル(機械語に変換)して事項。

・実行速度

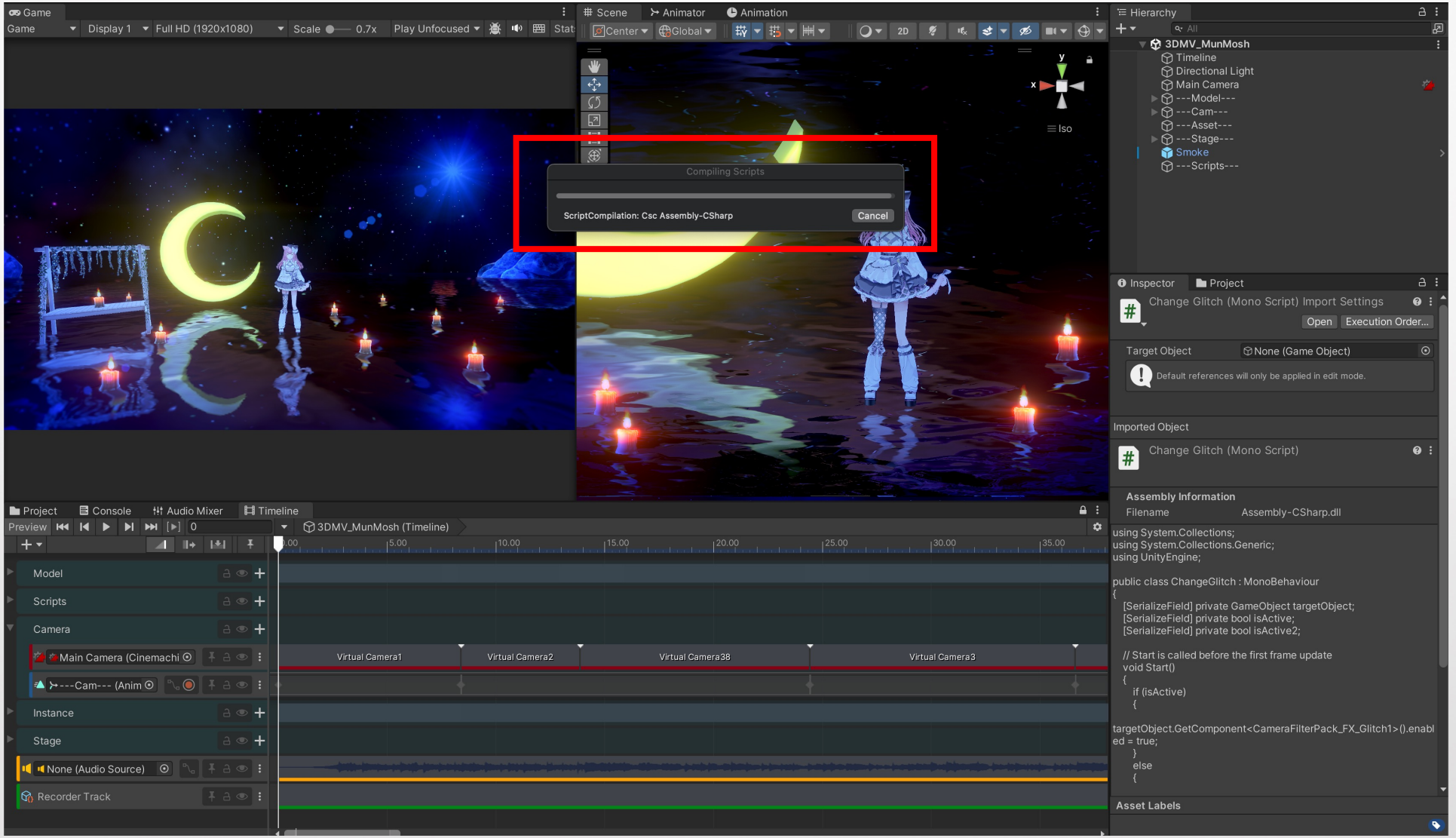
→事前にパソコンが理解できる機械語に変換しているので高速。

・デバッグ

→コンパイル時にチェック。

Unityでプログラムを変えた際に表示されるシークバーは裏でコンパイルしていたりします。

１．プログラム概論


Compiling Scripts
ScriptCompilation: Csc Assembly-CSharp
Cancel
Hierarchy
3DMV_MunMosh
Timeline
Directional Light
Main Camera
---Model---
---Cam---
---Asset---
---Stage---
Smoke
---Scripts---
Inspector
Project
Change Glitch (Mono Script) Import Settings
Open
Execution Order...
Target Object
None (Game Object)
Default references will only be applied in edit mode.
Imported Object
Change Glitch (Mono Script)
Assembly Information
Filename
Assembly-CSharp.dll
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class ChangeGlitch : MonoBehaviour
{
[SerializeField] private GameObject targetObject;
[SerializeField] private bool isActive;
[SerializeField] private bool isActive2;
// Start is called before the first frame update
void Start()
{
if (isActive)
{
targetObject.GetComponent<CameraFilterPack_FX_Glitch1>().enabled = true;
}
else
{
Asset Labels
Timeline
3DMV_MunMosh (Timeline)
Model
Scripts
Camera
Main Camera (Cinemachi
---Cam--- (Anim
Instance
Stage
None (Audio Source)
Recorder Track
Virtual Camera1
Virtual Camera2
Virtual Camera38
Virtual Camera3

１．プログラム概論

＜C#について＞

C#言語はMicrosoftが開発したオブジェクト指向のプログラミング言語です。

オブジェクト指向とはCのような手続き言語とは違い、作成するアプリケーションをオブジェクトと呼ばれる構造に分割、作成しそれらを組み合わせていく、という考え方になります。

基本的な文法はC言語をベースにして作られています。(C言語自体が現在の色々な言語のモデルケースとなっているので覚えるとコスパいいです！)

ちなみに正確には、

　C言語→C++言語→Java言語→C#言語

となります。

１．プログラム概論

この講義は、C#を学んでいきますが、実行環境はUnityを使用します。
Unityでは、C#を使って開発が出来ますので、楽しく頑張っていきましょう。
分からないことがあれば、話の途中でも手をあげて質問していただいて
構いません。

プログラミングは最初はチンプンカンプンだと思いますが理解が進み、自分の
考えたゲームを実現できるようになると、どんどん楽しくなっていくと思いますので、
自分自身の理解を深めるためにも、どんどん質問してください。

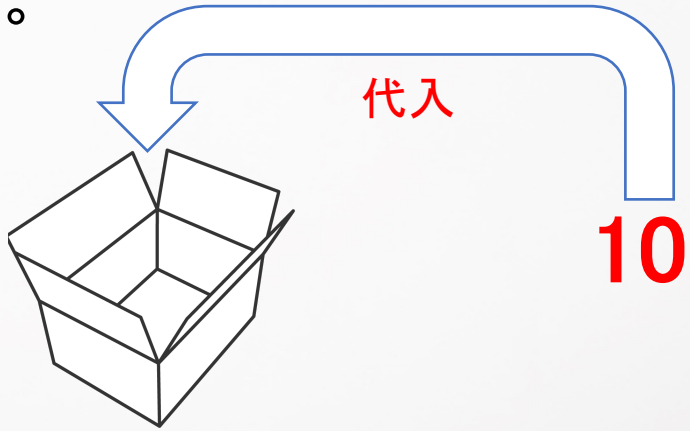
２．変数と演算子

１．変数の宣言と代入
２．変数の種類
３．四則演算子
４．比較演算子
５．論理演算子

２－１．変数の宣言と代入

変数とは、特定の値を設定して、繰り返し使えるようにしたものです。

よく、変数は「箱」に例えられることがあります。



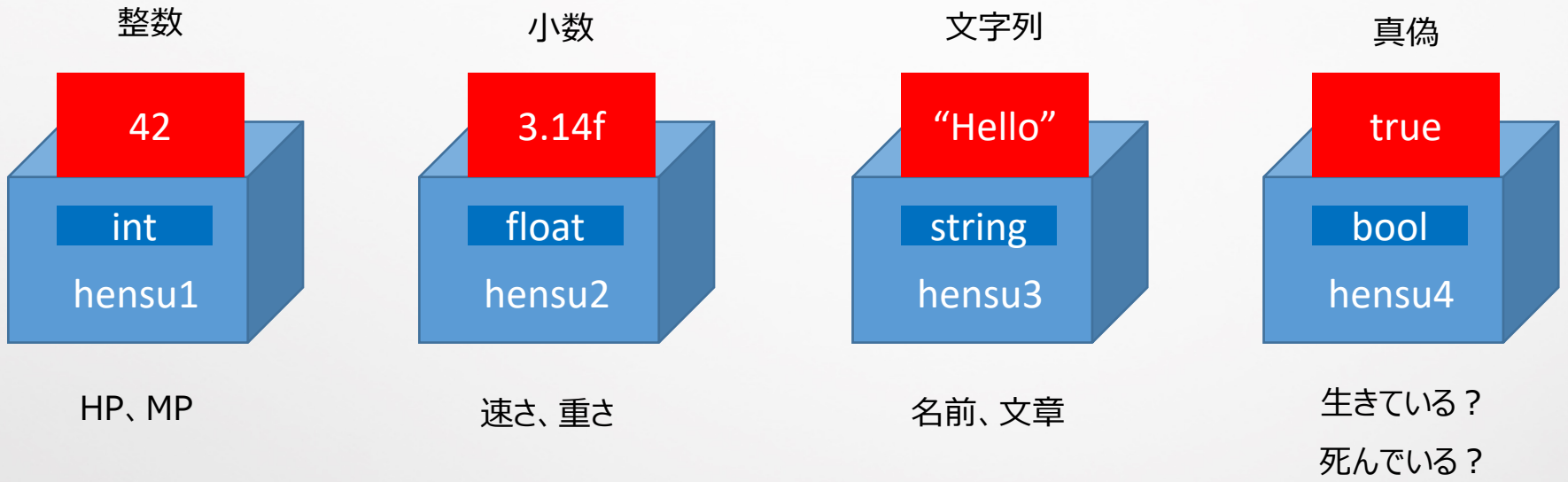
プログラムで書くと、こうなります。

```
int hensu = 10;
```

この時、hensuが10と同等(イコール)というわけではなく、hensuという箱に10を代入しているという意味になります。

2－2．変数の種類

変数にも以下のような種類があります！(正確には他にも型が存在しますが、ゲーム開発で使用するものは以下の通り)

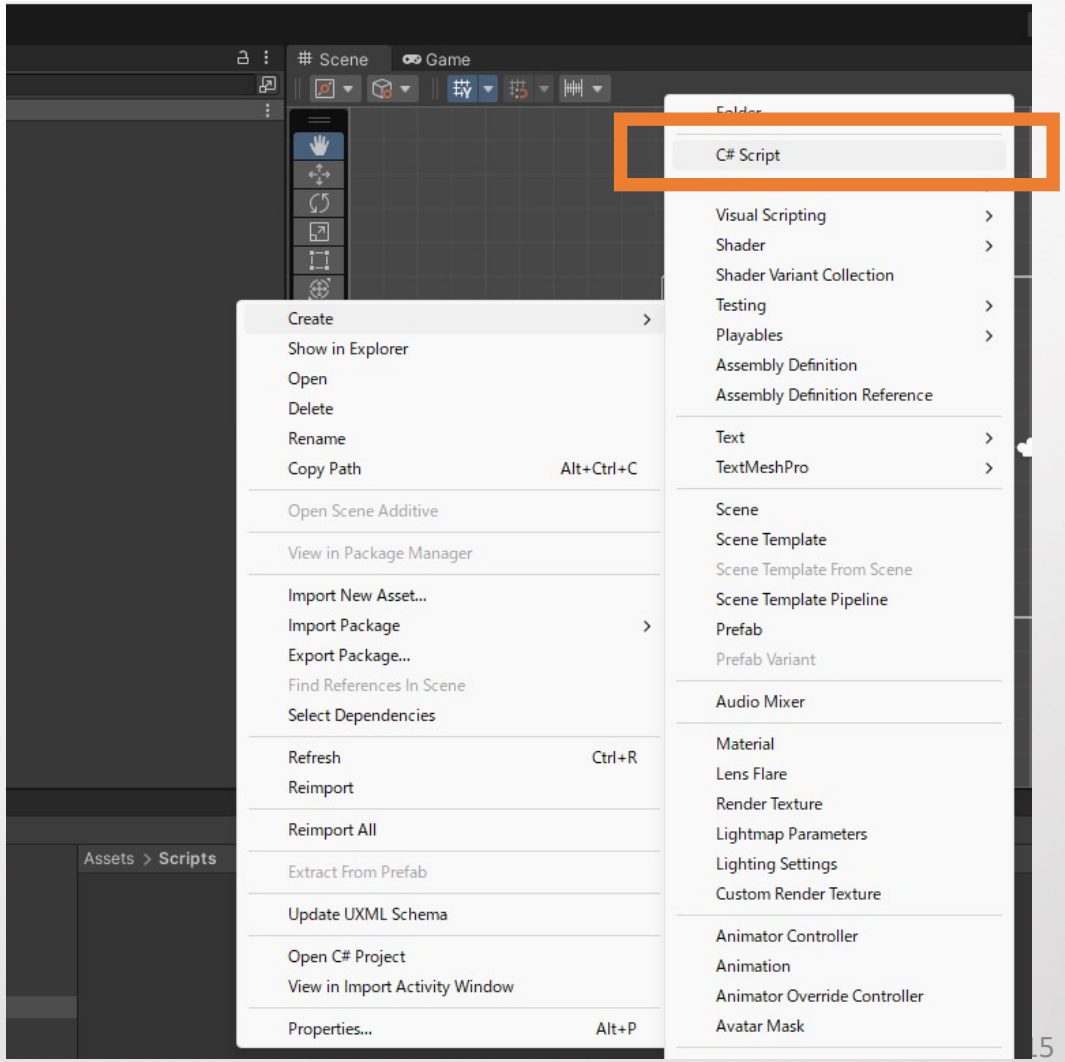


いつくかお約束ごとがあり、floatであれば末尾に「f」、stringは「” ”」を付けてその間に記載する、boolはtrueとfalseという文字のみ、となります。

2－2．変数の種類

Unityを立ち上げ、この講義専用のプロジェクトを立ち上げてください。
(2DのプロジェクトでOKです)

プロジェクトが立ち上がったら、C#を記載するスクリプトファイルを作成してください。
(名前はtest2_2.csとしてください)



2－2．変数の種類

実際に変数を作ってDebug.Logで表示してみましょう！

hensuという出力ではなく、5と出力されていればOKです！

```
public class test2_2 : MonoBehaviour
{
    // Start is called before the first frame update
    イベント関数 | 使用箇所 | オーバーライド | 拡張メソッド | 公開している API
    void Start()
    {
        int hensu = 5;
        Debug.Log(hensu);
    }

    // Update is called once per frame
    イベント関数 | 使用箇所 | オーバーライド | 拡張メソッド | 公開している API
    void Update()
    {

    }
}
```

2－2．変数の種類

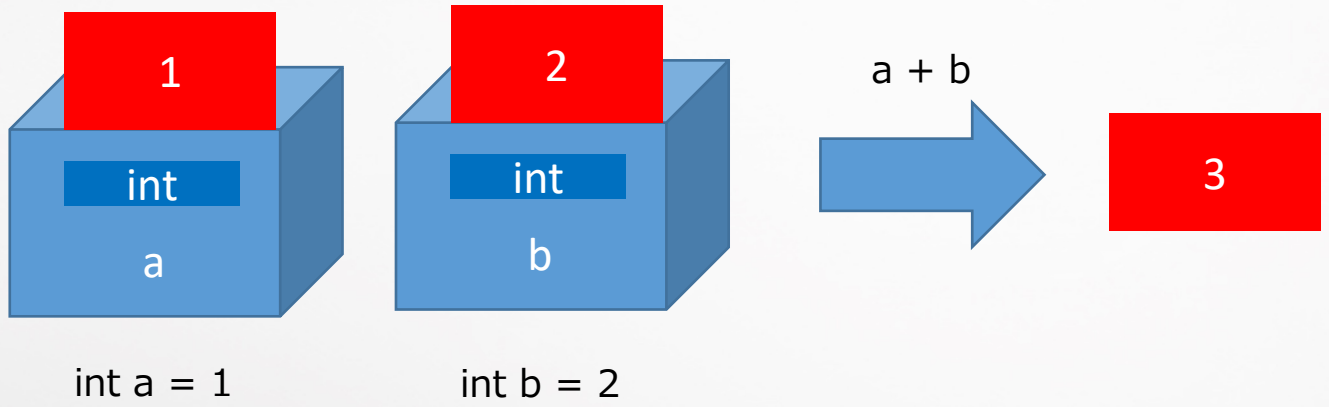
string、float、bool型についても変数を作成して表示し、Debug.Logで表示させてみてください。

stringの場合は値を入れる場合に
””(ダブルクォーテーション)で囲みます。
　string test1 = “テスト”;
floatの場合は、最後にfをつけます。
　float test2 = 3.14f;
bool型の場合はtrueとfalseの2択です。
　bool test3 = true;

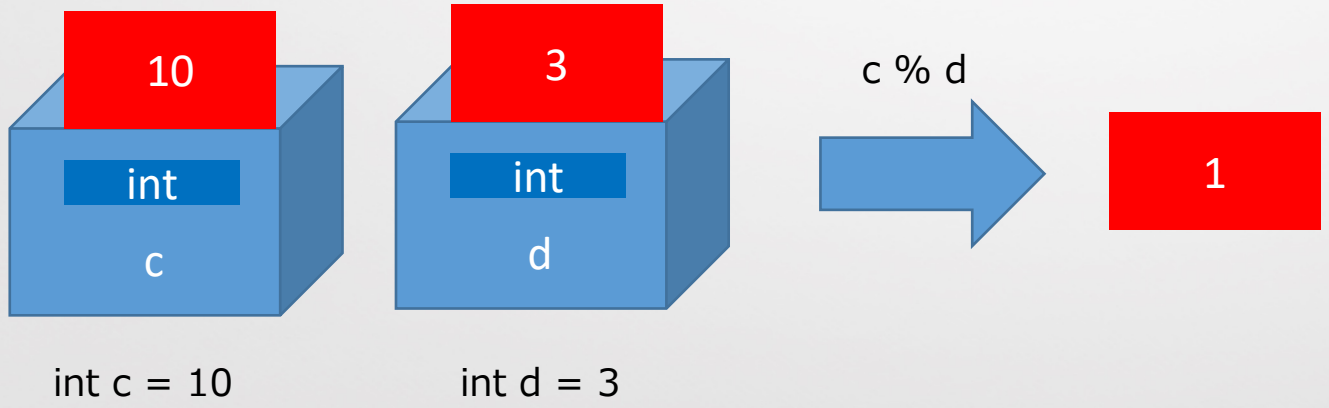
```
void Start()
{
    int hensu = 5;
    Debug.Log(hensu);

    string test1 = "テスト";
    Debug.Log(test1);

    float test2 = 3.14f;
    Debug.Log(test2);

    bool test3 = true;
    Debug.Log(test3);
}
```

2－3．四則演算子

変数同士は四則演算(+,-,*,/)することが出来ます。



こんなことも可能です！

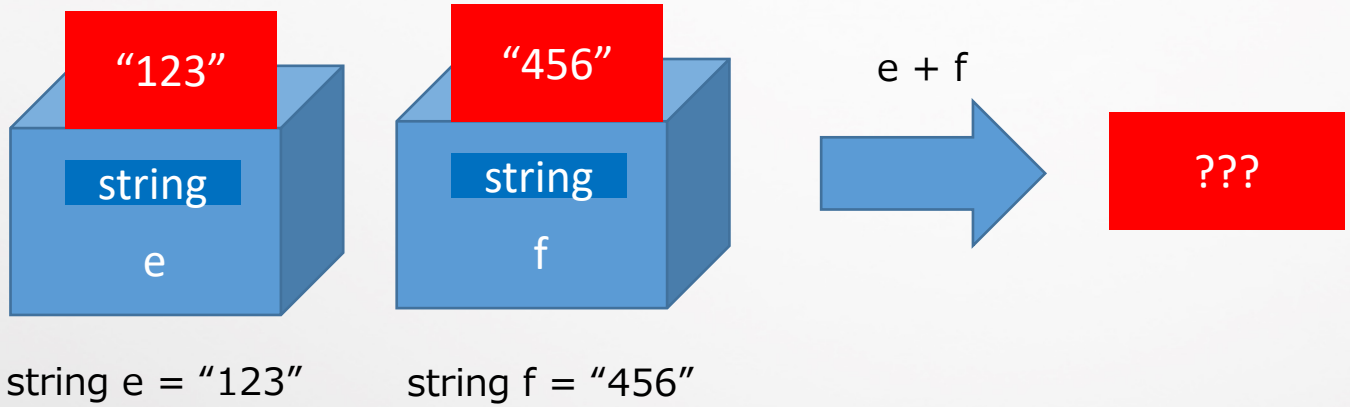


2－3．四則演算子

＜注意＞

変数は基本、同じ型同士のみ四則演算可能です。(bool除く)

では、以下の場合はどのようになるでしょうか？

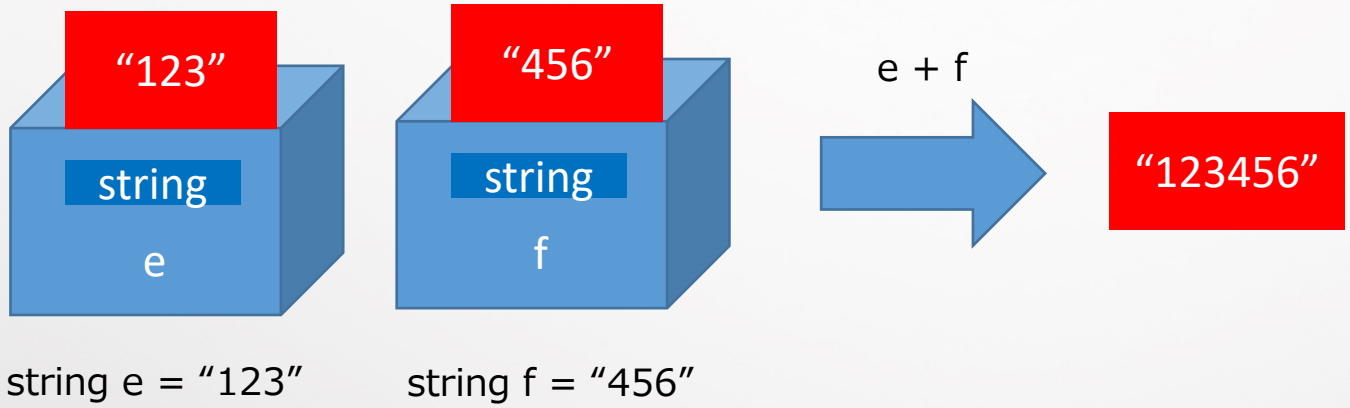


2－3．四則演算子

＜注意＞

変数は基本、同じ型同士のみ四則演算可能です。(bool除く)

では、以下の場合はどのようになるでしょうか？



例えば、

「プレイヤーの名前が入った変数」＋「は攻撃した！」

みたいな感じで使えそうですね。

2－3．四則演算子

＜注意＞

異なる変数の型の場合は、case(キャスト)をつけることで可能です。

```
int a = 10;
float b = 1.8f;
int c;

c = a + (int)b;
```

変数cには何が入るでしょうか、試してみましょう！

2－3．四則演算子

test2_3.csを作成し、実際にやってみましょう！

出来た人は、以下の問題にチャレンジしてみてください！

＜問題＞

float型の変数を2つ作って1.2と33.4を右記のように格納し、2つの変数を乗算してDebug.Logに表示させてください。

```
void Start()
{
    int n = 10;
    int m = 2;
    Debug.Log(n + m);

    n = 10;
    m = 3;
    Debug.Log(n % 3);

    //別の変数に入れることも出来る
    int x = n + 3;
    Debug.Log(x);

    string s = "123";
    string a = "45";
    Debug.Log(s + a);
}
```

2－3．四則演算子

＜解答例＞

```
float test1 = 1.2f;
float test2 = 33.4f;

Debug.Log(test1 * test2);
```

2－3．四則演算子

四則演算において、特殊な書き方も出来るのでやってみましょう！
(test2_3.csに続けて記載してください)

まずは、今までの知見から以下のプログラムを作成してください。

(1) int型の変数aaaに10を格納し、2を足すプログラムを作成。

(2) int型の変数bbbに5を格納し、1を足すプログラムを作成。

2－3．四則演算子

四則演算において、特殊な書き方も出来るのでやってみましょう！

(1) int型の変数aaaに10を格納し、2を足すプログラムを作成。

→ int aaa = 10;

aaa += 2;

(2) int型の変数bbbに5を格納し、1を足すプログラムを作成。

→ int bbb = 10;

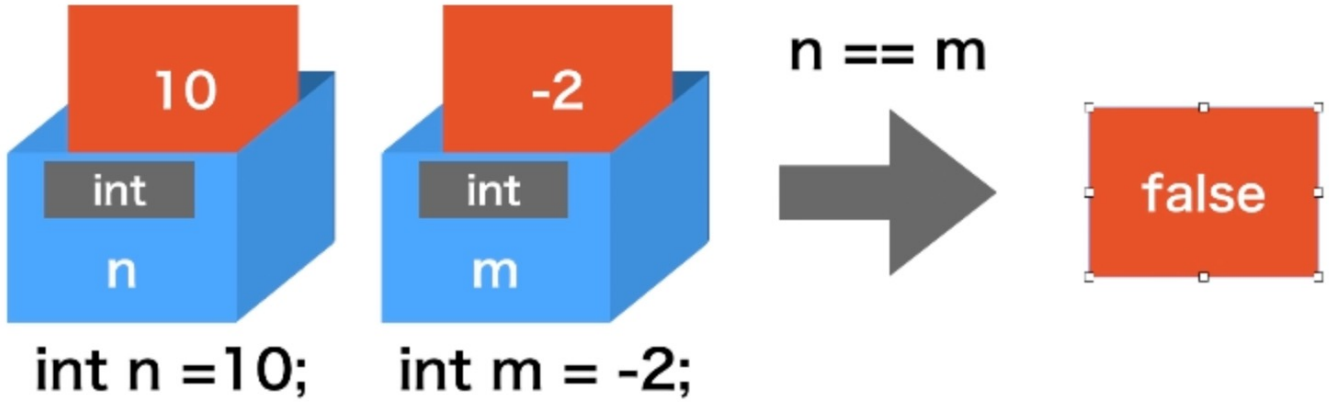
bbb++;

加算で実施しましたが、その他の計算でも可能です！

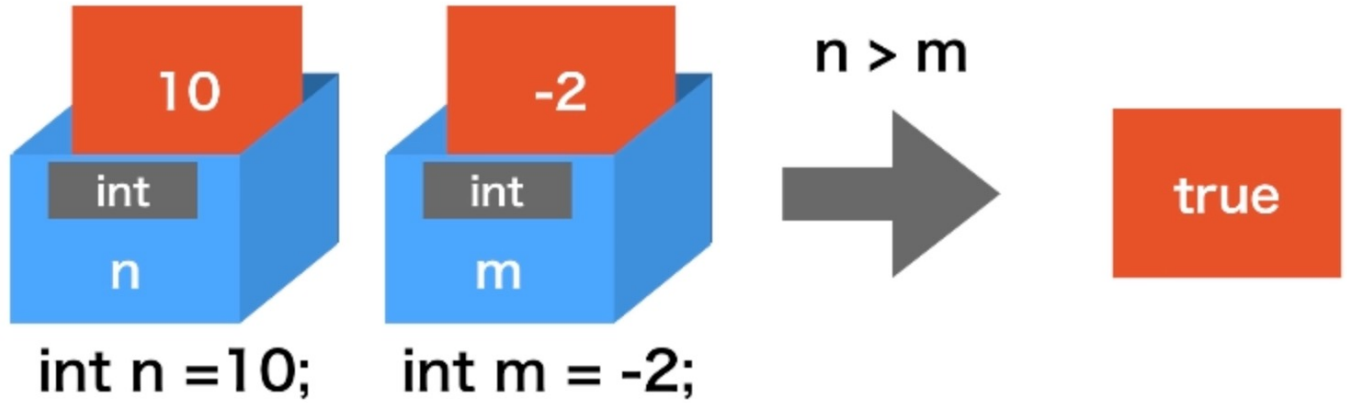
2－4．比較演算子

比較演算子：==, >, <, >=, <=, !=

- 「X == O」は
  XとOが同じか



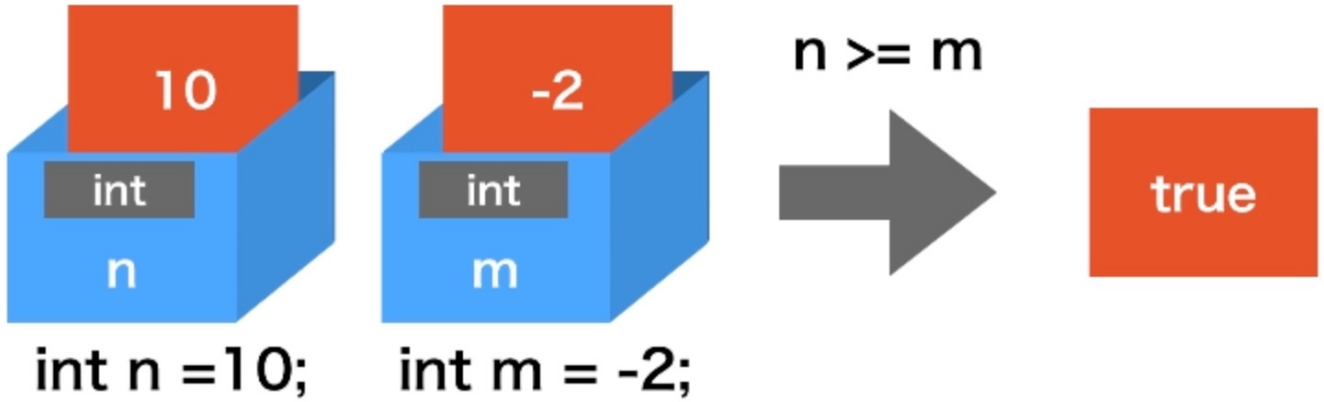
- 「>」



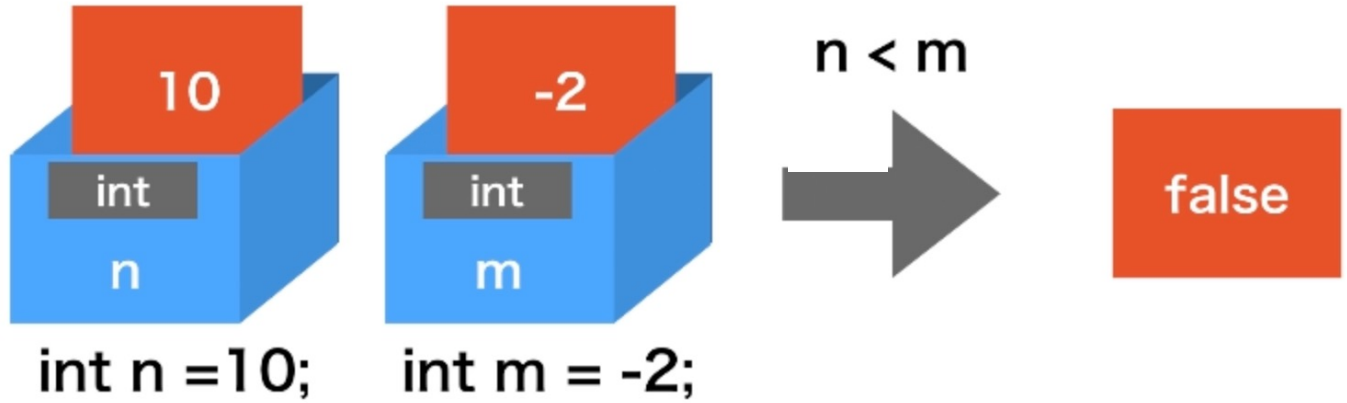
2－4．比較演算子

比較演算子：==, >, <, >=, <=, !=

- 「>=」



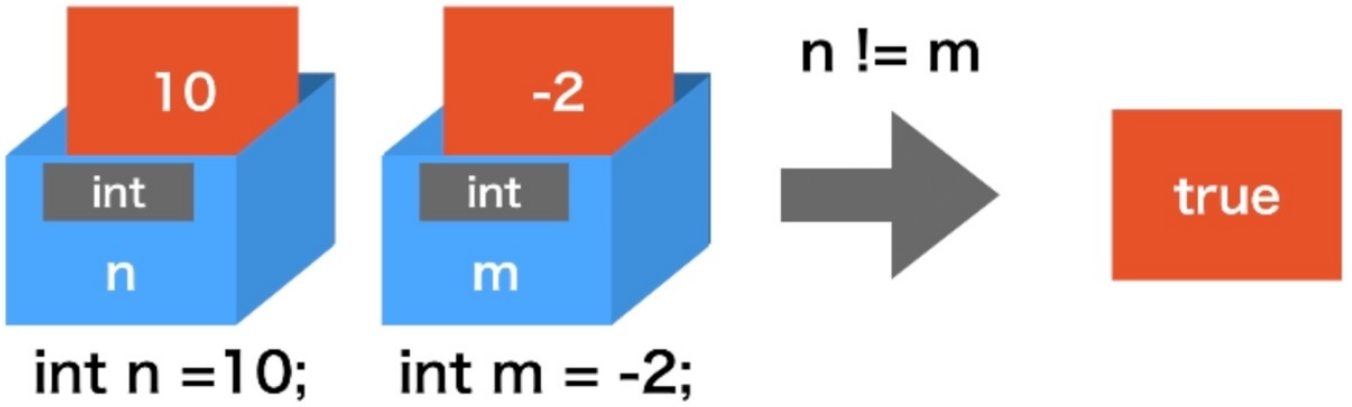
- 「<」



2－4．比較演算子

比較演算子：==, >, <, >=, <=, !=

- 「!=」は違うかどうか



２－４．比較演算子

test2_4.csを作成し、実際にやってみましょう！

プログラミングを理解するコツとして、
事前に結果がどのようになるのか
考えながら作ると良きです！！

```
void Start()
{
    int n = 10;
    int m = -2;

    Debug.Log(n == m);
    Debug.Log(n > m);
    Debug.Log(n >= m);
    Debug.Log(n < m);
    Debug.Log(n != m);
}
```

2－5．論理演算子

論理演算子(&&,||)：複数の条件を合わせるもの

&&（かつ）

&&	true	false
true	true	false
false	false	false

全部 trueならtrue

||（または）

||	true	false
true	true	true
false	true	false

1つでも trueならtrue

(m == n) && (s > a)　のように使う

2－5．論理演算子

test2_5.csを作成し、実際にやってみましょう！

出来た人は、以下の問題にチャレンジしてみてください！

＜問題＞

以下の変数の設定値でaとb、cとdを比較してください。

int型の変数 a=10,b=2,c=-4,d=-5

1 ：And条件で両方ともTrueになる

2 ：Or条件で片方だけTrueになる

```
void Start()
{
    int n = 10;
    int m = -2;
    int x = 5;
    int y = 7;

    Debug.Log(n >= m && x < y);
    Debug.Log(n >= m && x >= y);
    Debug.Log(n < m && x < y);
    Debug.Log(n < m && x >= y);

    Debug.Log(n >= m || x < y);
    Debug.Log(n >= m || x >= y);
    Debug.Log(n < m || x < y);
    Debug.Log(n < m || x >= y);
}
```

2－5．論理演算子

＜解答例＞

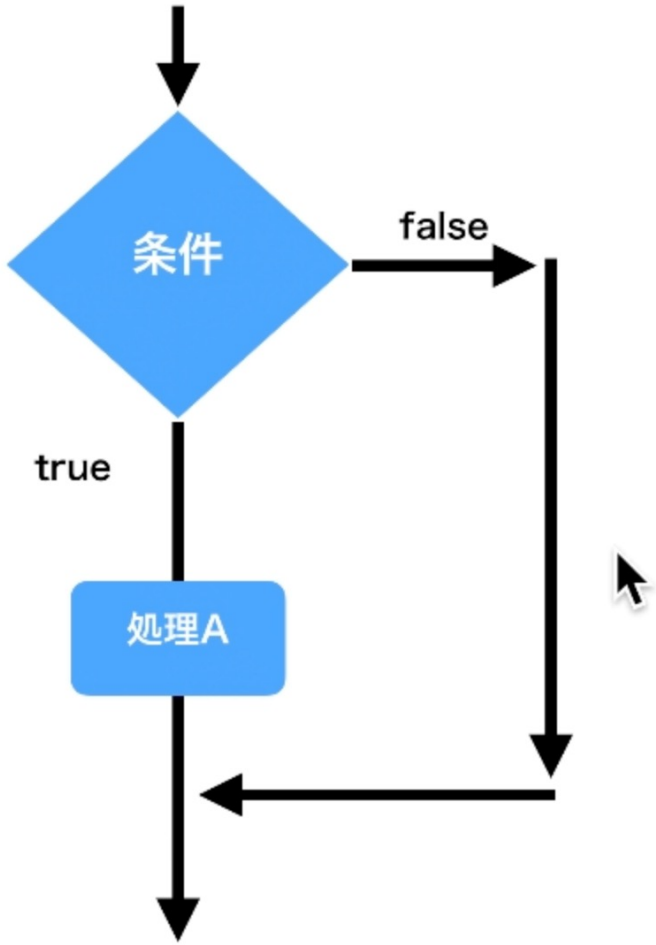
```
//以下の変数の設定値でaとb、cとdを比較してください。
//1：And条件で両方ともTrueになる
//2：Or条件で片方だけTrueになる
int a = 10;
int b = 2;
int c = -4;
int d = -5;

Debug.Log(a > b && c > d);
Debug.Log(a > b || c < d);
```

３．If文とSwitch文

１．If文について
２．If else文
３．else if文
４．Switch文

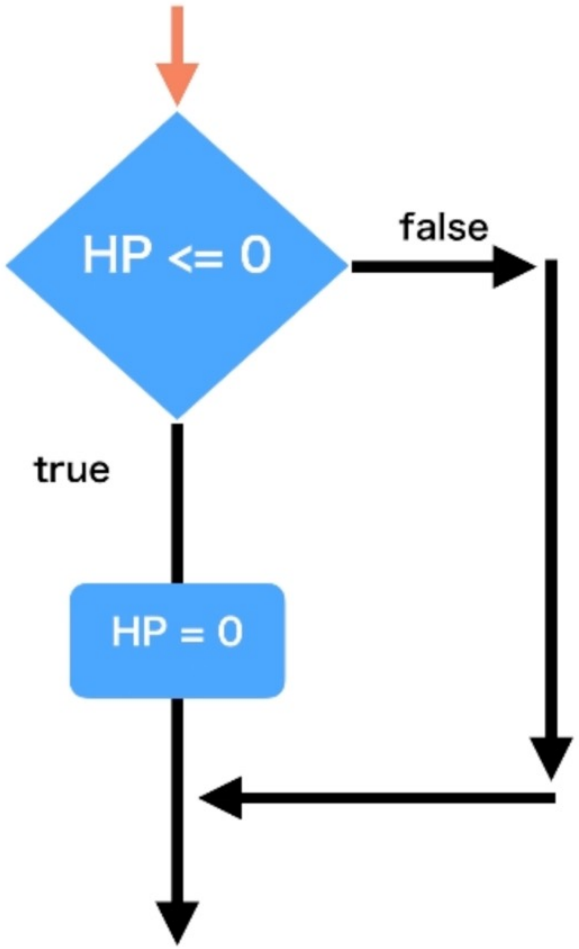
3－1．If文について

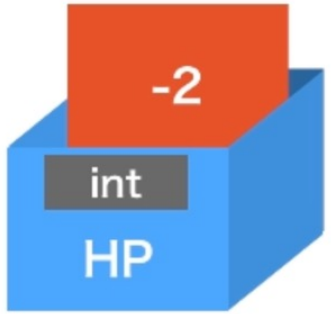
if：条件を満たしたときに処理を分岐できるもの



3－1．If文について

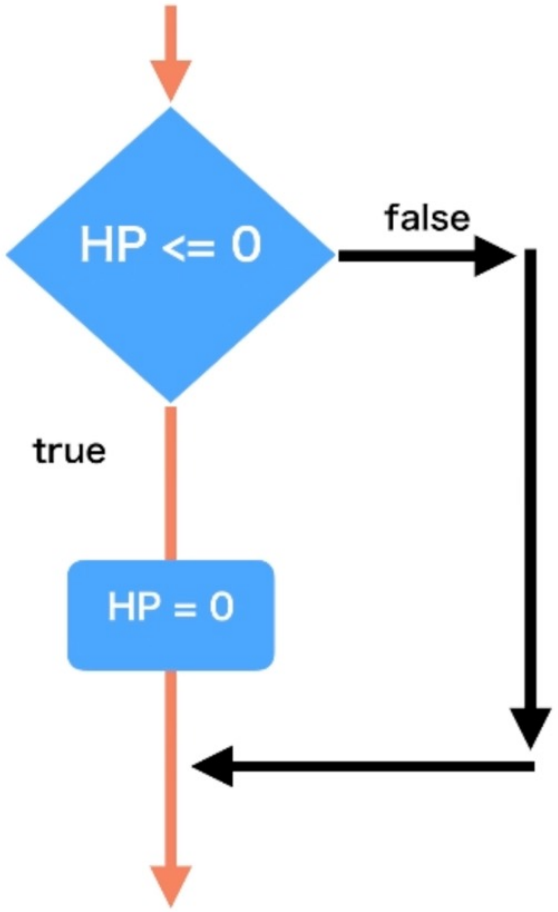
if：条件を満たしたときに処理を分岐できるもの

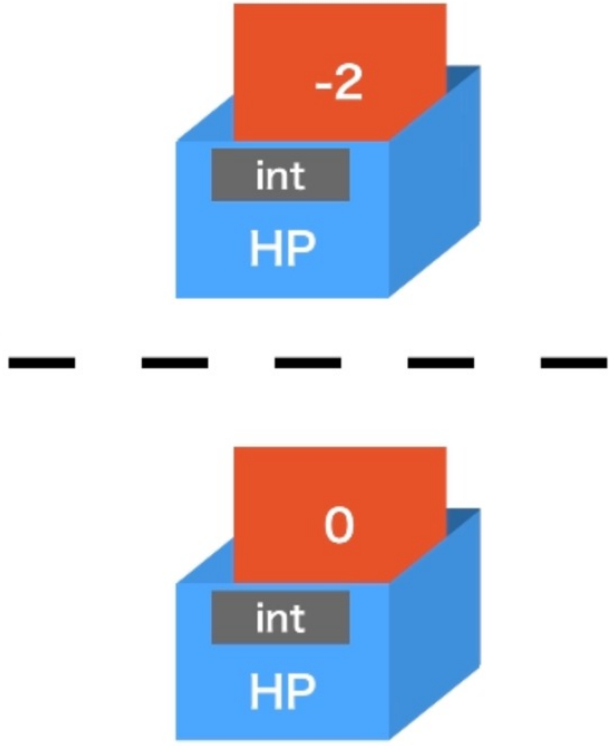




3－1．If文について

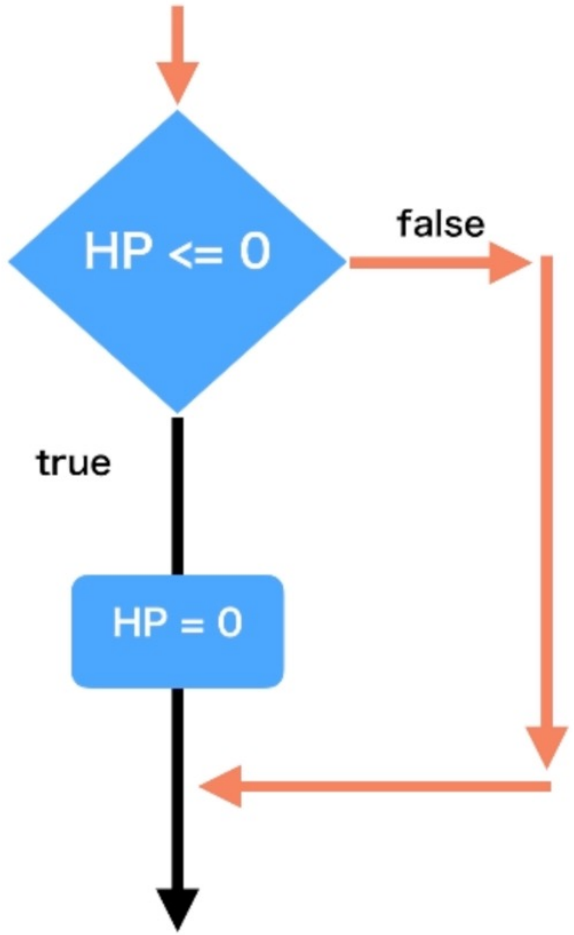
if：条件を満たしたときに処理を分岐できるもの

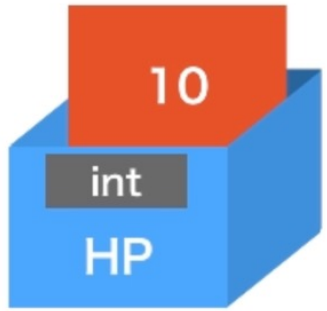




3－1．If文について

if：条件を満たしたときに処理を分岐できるもの





何もしない

3－1．If文について

記述方法は、以下のような形になります。
条件の欄に2項で実施した、比較演算子や論理演算子またはbool型が入ります。

```
if（条件）{
    // 条件がtrueのときに実行
}
```

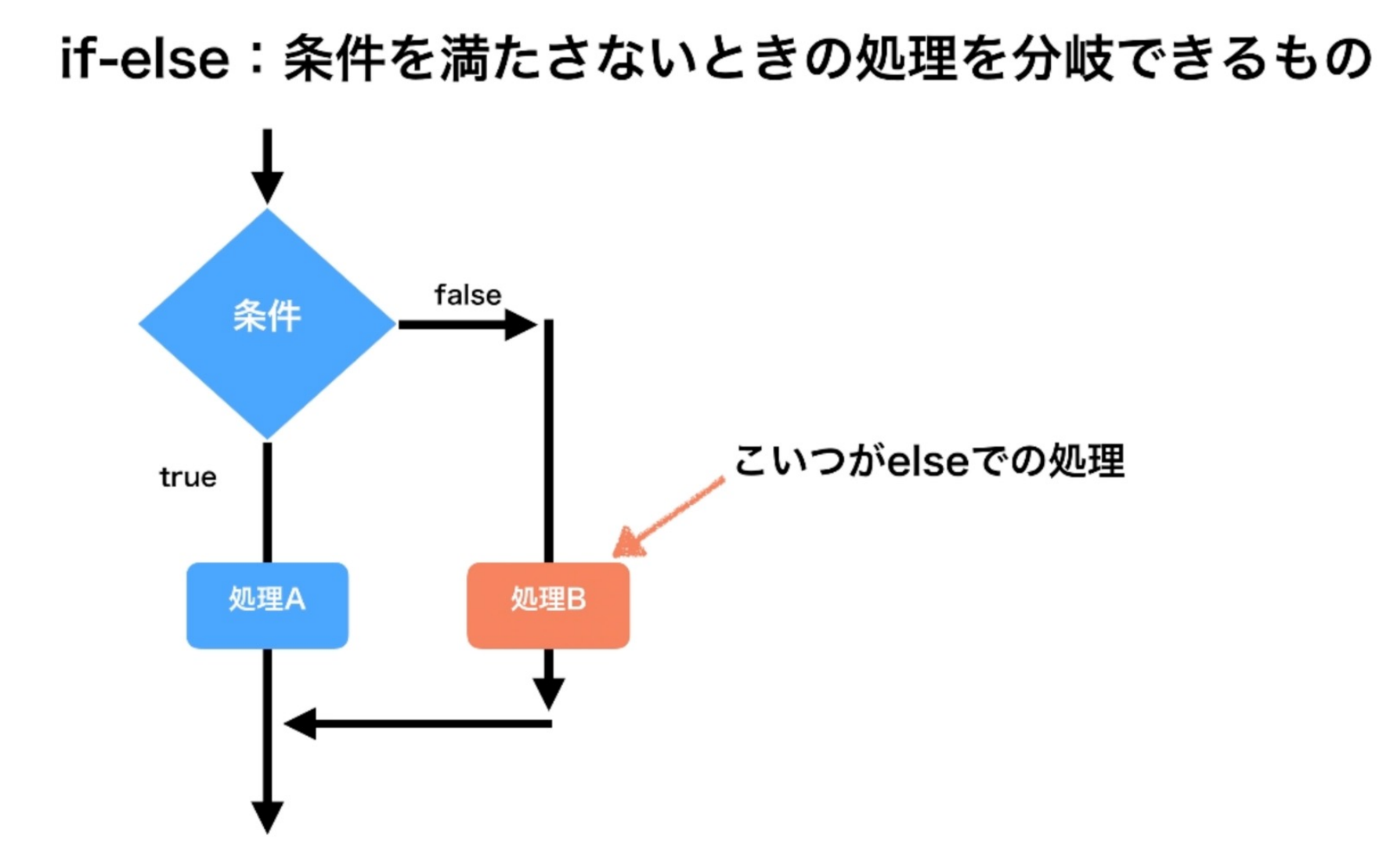
test3_1.csを作成し、2ページ前のフローチャートを上記の形式に沿って作成してみてください。

3－1．If文について

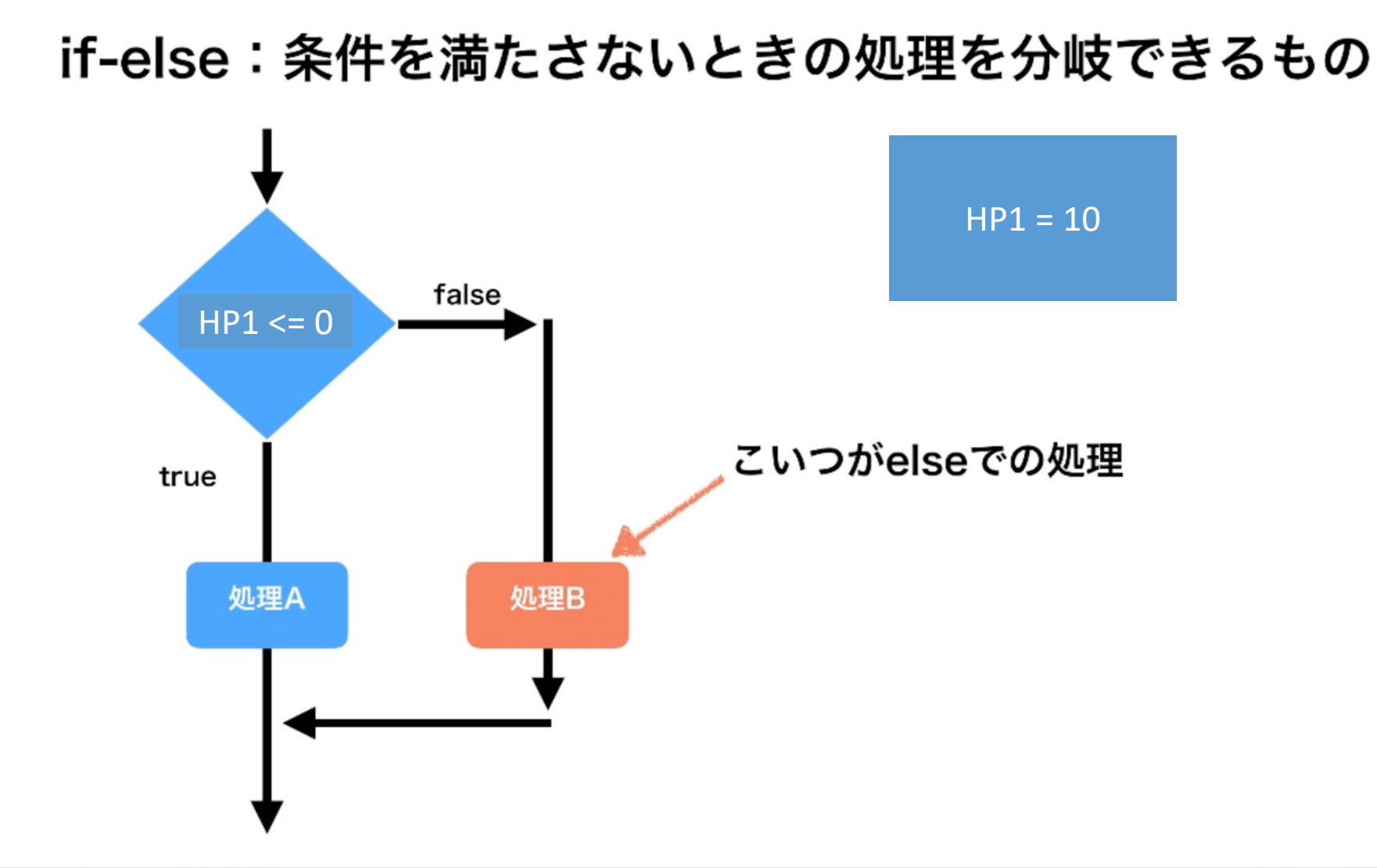
＜回答＞

```
void Start()
{
    int hp = -2;

    if (hp <= 0)
    {
        hp = 0;
    }
    Debug.Log(hp);
}
```

3－2．If else文


if-else：条件を満たさないときの処理を分岐できるもの
条件
false
true
処理A
処理B
こいつがelseでの処理

3－2．If else文


if-else：条件を満たさないときの処理を分岐できるもの
HP1 = 10
HP1 <= 0
false
true
こいつがelseでの処理
処理A
処理B

3－2．If else文

記述方法は、以下のような形になります。
条件の欄に2項で実施した、比較演算子や論理演算子またはbool型が入ります。

```
if（条件）{
    // 条件が true のときに実行される処理
} else {
    // 条件が false のときに実行される処理
}
```

test3_2.csを作成し、1ページ前のフローチャートを上記の形式に沿って作成してみてください。

3－2．If else文

<回答>

```
int hp1 = 10;

if (hp1 <= 0)
{
    Debug.Log("処理A");
}
else
{
    Debug.Log("処理B");
}
```

3－2．If else文

＜問題＞

①HPが30未満かつMPが50以上の場合、必殺技が使えることを示すDebug.Logを出力してください。
満たさない場合は、使えないことを示すDebug.Logを出力してください。

②制限時間(int time)が10秒以下またはHPが30以下かつMPが25未満の場合に超必殺技が使えることを示すDebg.Logを出力してください。
満たさない場合は、使えないことを示すDbug.Logを出力してください。

※hpとmpとtimeの変数にそれぞれの値を入れて、trueとなるパターンとfalseになるパターンの値を設定してみてください。

3－2．If else文

＜答え(trueのパターンのみ)＞

```
int hp = 25;
int mp = 60;

if (hp < 30 && mp >= 50) {
    Debug.Log("必殺技が使える！");
} else {
    Debug.Log("必殺技は使えません");
}
```

```
int time = 12;
int hp = 28;
int mp = 20;

if (time <= 10 || (hp <= 30 && mp < 25)) {
    Debug.Log("超必殺技が使える！");
} else {
    Debug.Log("超必殺技は使えません");
}
```

3－2．If else文

＜入れ子＞

以下のような場合、どのようにIf文を組み立てれば良いでしょうか。

HPが30以上の場合はMPを確認し、MPが20より多ければ使用可能と出力。MPが20以下の場合は使用不可と出力。

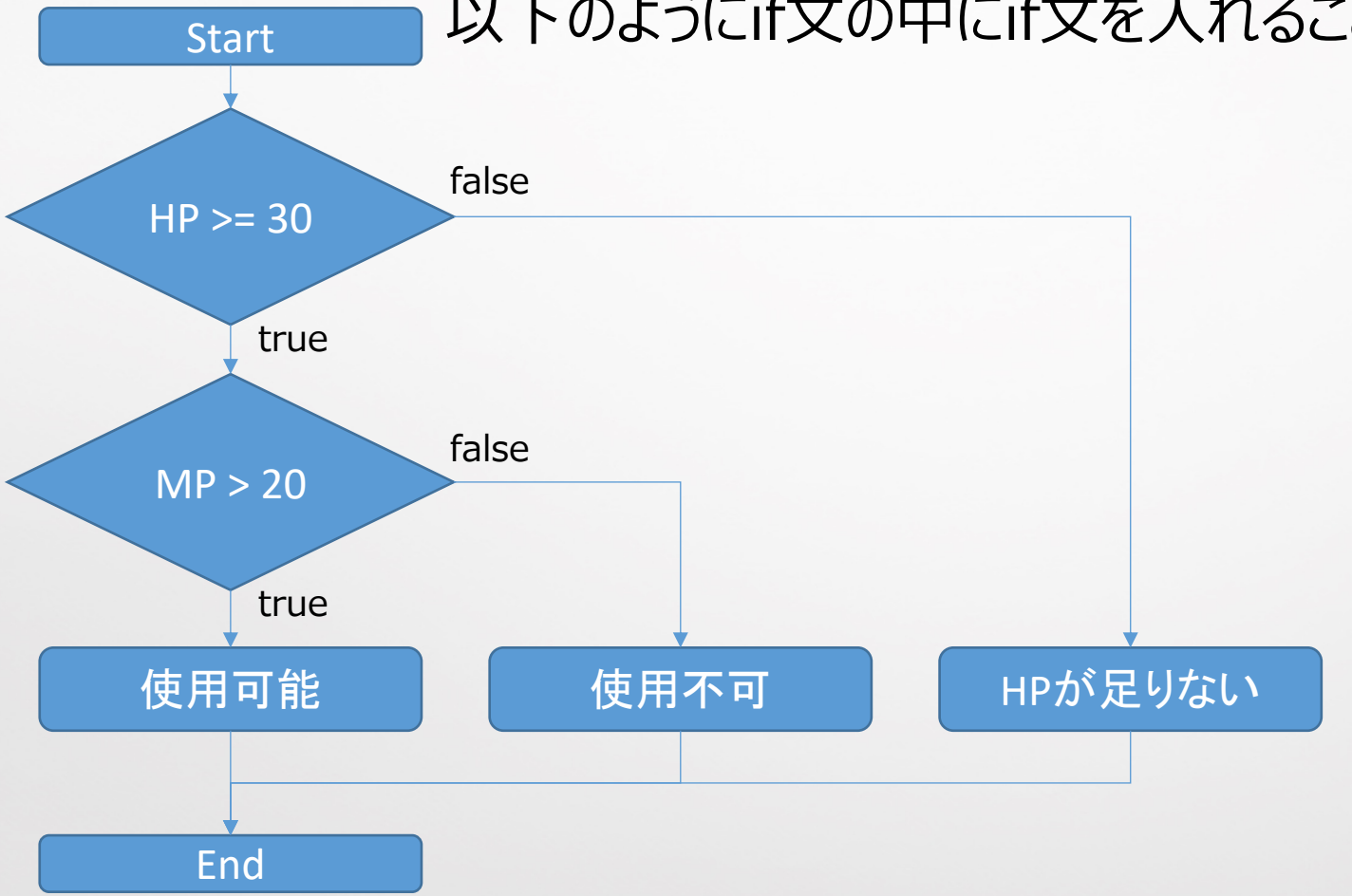
HPが30未満の場合はHPが足りないと出力してください。

分かりにくい場合は、フローチャートを書いてみたら分かりやすくなるかも？

3－2．If else文

＜フローチャート＞

以下のようにif文の中にif文を入れることも可能です！

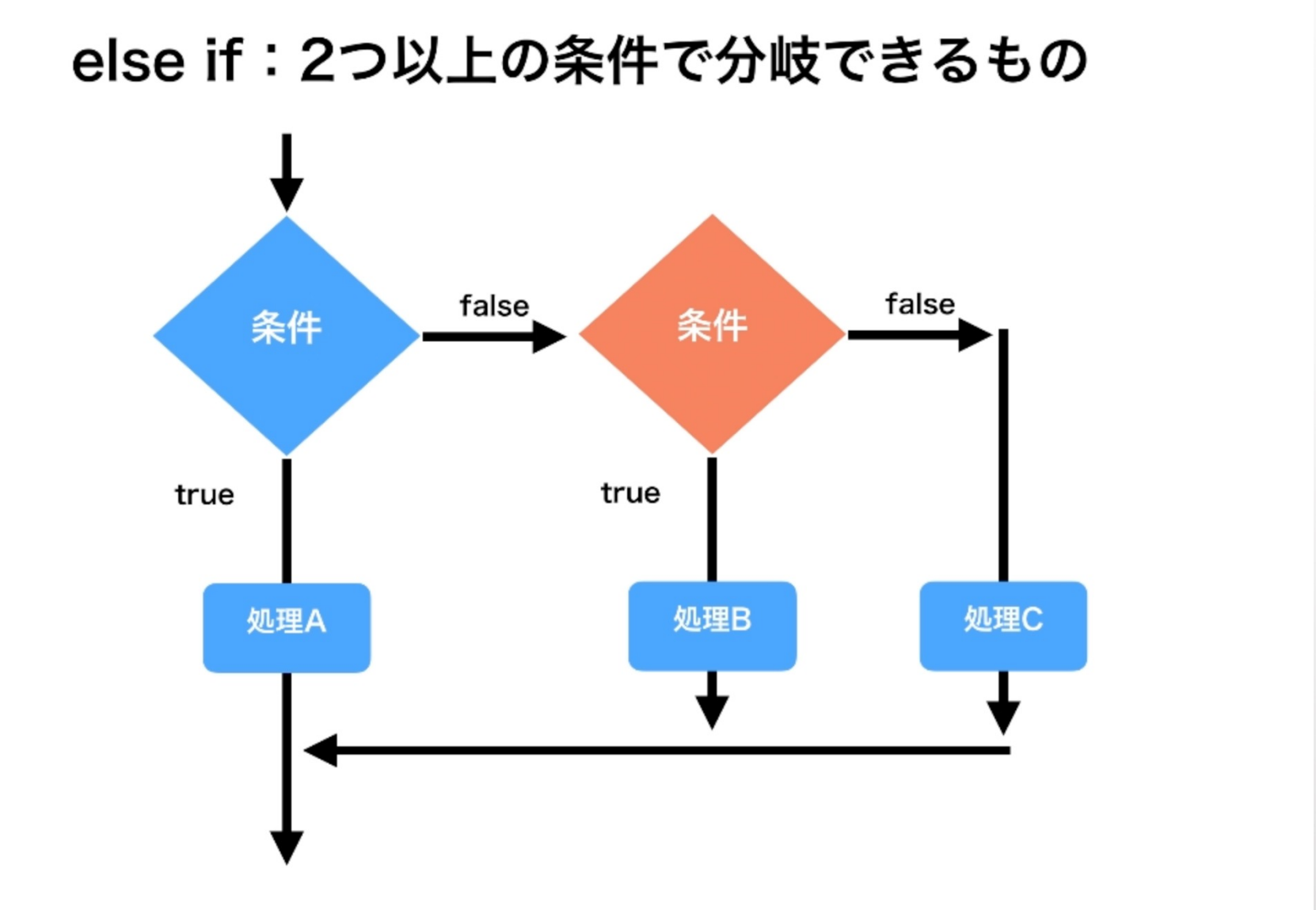


3－2．If else文

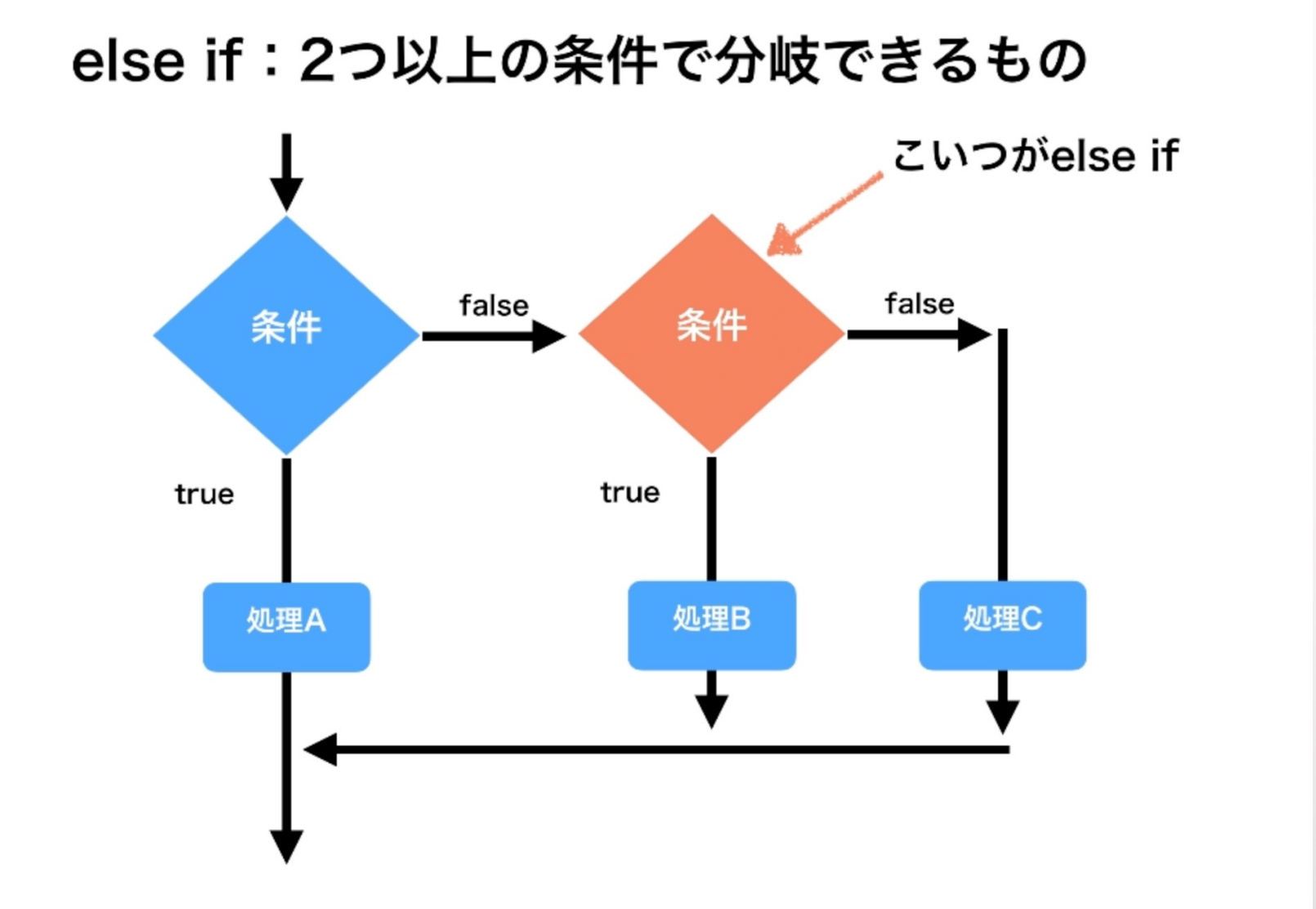
以下のように記述します、タブキーで1つ下げることを忘れないでくださいね！

```
int hp = 50;
int mp = 40;

if (hp >= 30) {
    if (mp >= 20) {
        Debug.Log("スキルを使用できる");
    } else {
        Debug.Log("MPが足りません");
    }
} else {
    Debug.Log("HPが足りません");
}
```

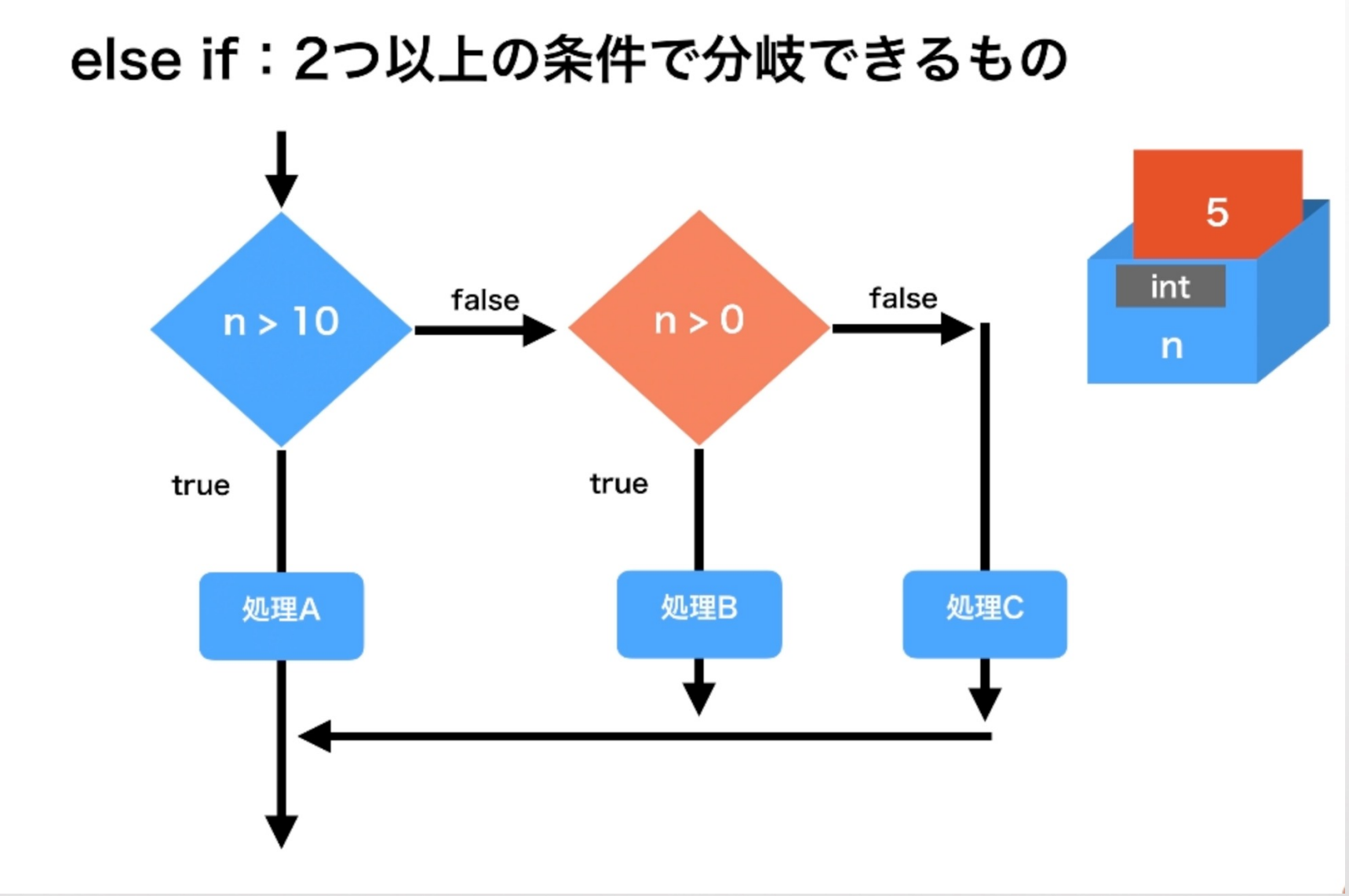
3－3．else if文


else if：2つ以上の条件で分岐できるもの
条件
false
条件
false
true
true
処理A
処理B
処理C

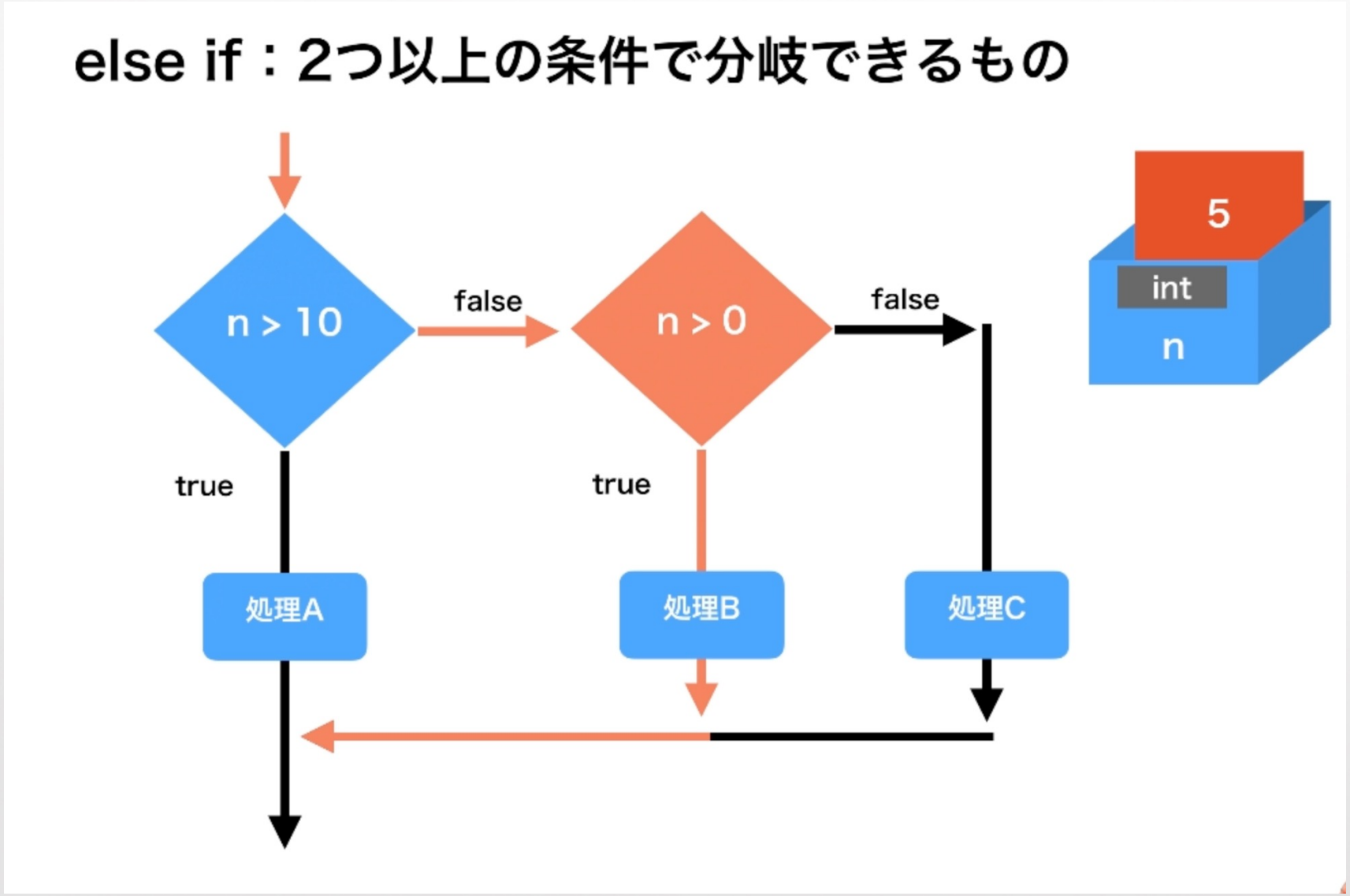
3－3．else if文


else if：2つ以上の条件で分岐できるもの
こいつがelse if
条件
false
条件
false
true
true
処理A
処理B
処理C

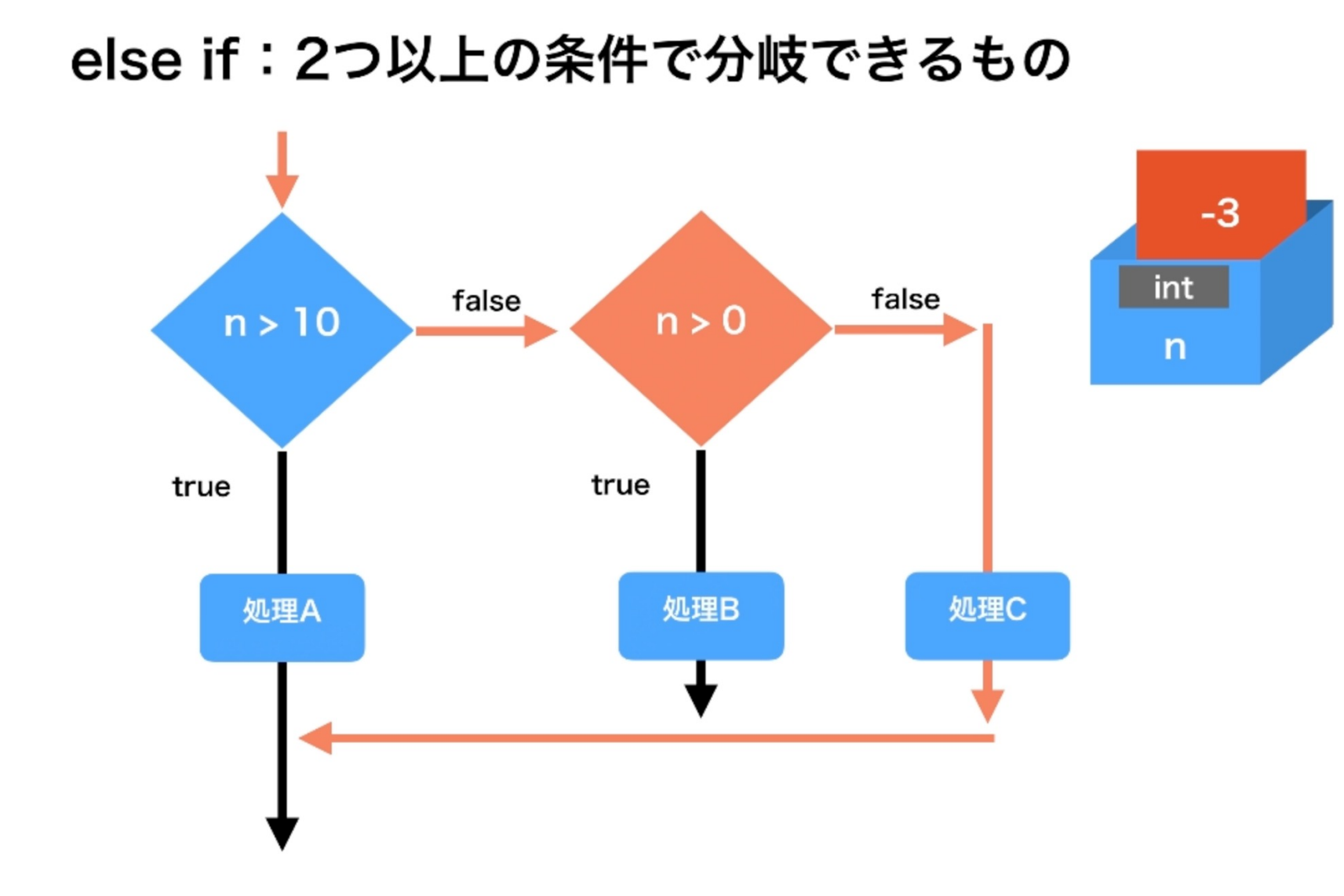
3－3．else if文


else if：2つ以上の条件で分岐できるもの
n > 10
false
n > 0
false
true
true
処理A
処理B
処理C
5
int
n

3－3．else if文


else if：2つ以上の条件で分岐できるもの
n > 10
false
n > 0
false
true
true
処理A
処理B
処理C
5
int
n

3－3．else if文


else if：2つ以上の条件で分岐できるもの
n > 10
false
n > 0
false
true
true
処理A
処理B
処理C
-3
int
n

3－3．else if文

記述方法は、以下のような形になります。
条件の欄に2項で実施した、比較演算子や論理演算子またはbool型が入ります。

```
if (条件1) {
    // 条件1が true のときの処理
} else if (条件2) {
    // 条件1が false で、条件2が true のときの処理
} else {
    // どちらの条件も false のときの処理
}
```

test3_3.csを作成し、1ページ前のフローチャートを上記の形式に沿って作成してみてください。

3－2．If else文

＜回答＞

```
void Start()
{
    int n = -3;

    if (n > 10)
    {
        Debug.Log("処理A");
    }
    else if (n > 0)
    {
        Debug.Log("処理B");
    }
    else
    {
        Debug.Log("処理C");
    }
}
```

3－3．else if文

＜問題＞

①int型でscoreという変数を作成し、以下のような出力を行なってください。

・90以上 → Sランク、70以上 → Aランク

・50以上 → Bランク、50未満 → Cランク

②以下のif文は何がダメ？

```
if (mass >= 20)
{
    Debug.Log("20Kg以上");
}
else if (mass >= 50)
{
    Debug.Log("50Kg以上");
}
else
{
    Debug.Log("軽い！");
}
```

3－3．else if文

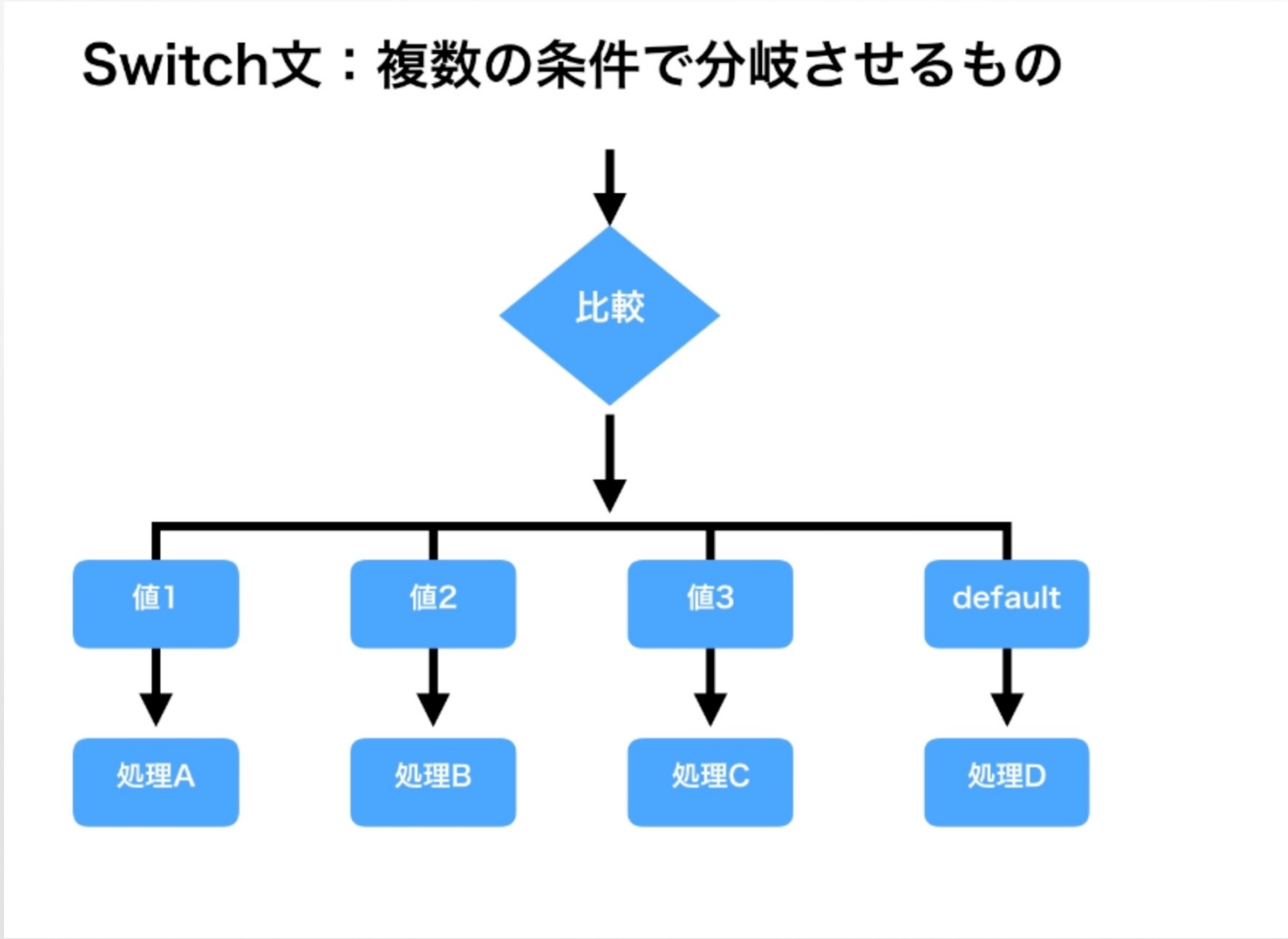
＜答え＞

```
int score = 75;

if (score >= 90) {
    Debug.Log("評価：Sランク");
} else if (score >= 70) {
    Debug.Log("評価：Aランク");
} else if (score >= 50) {
    Debug.Log("評価：Bランク");
} else {
    Debug.Log("評価：Cランク");
}
```

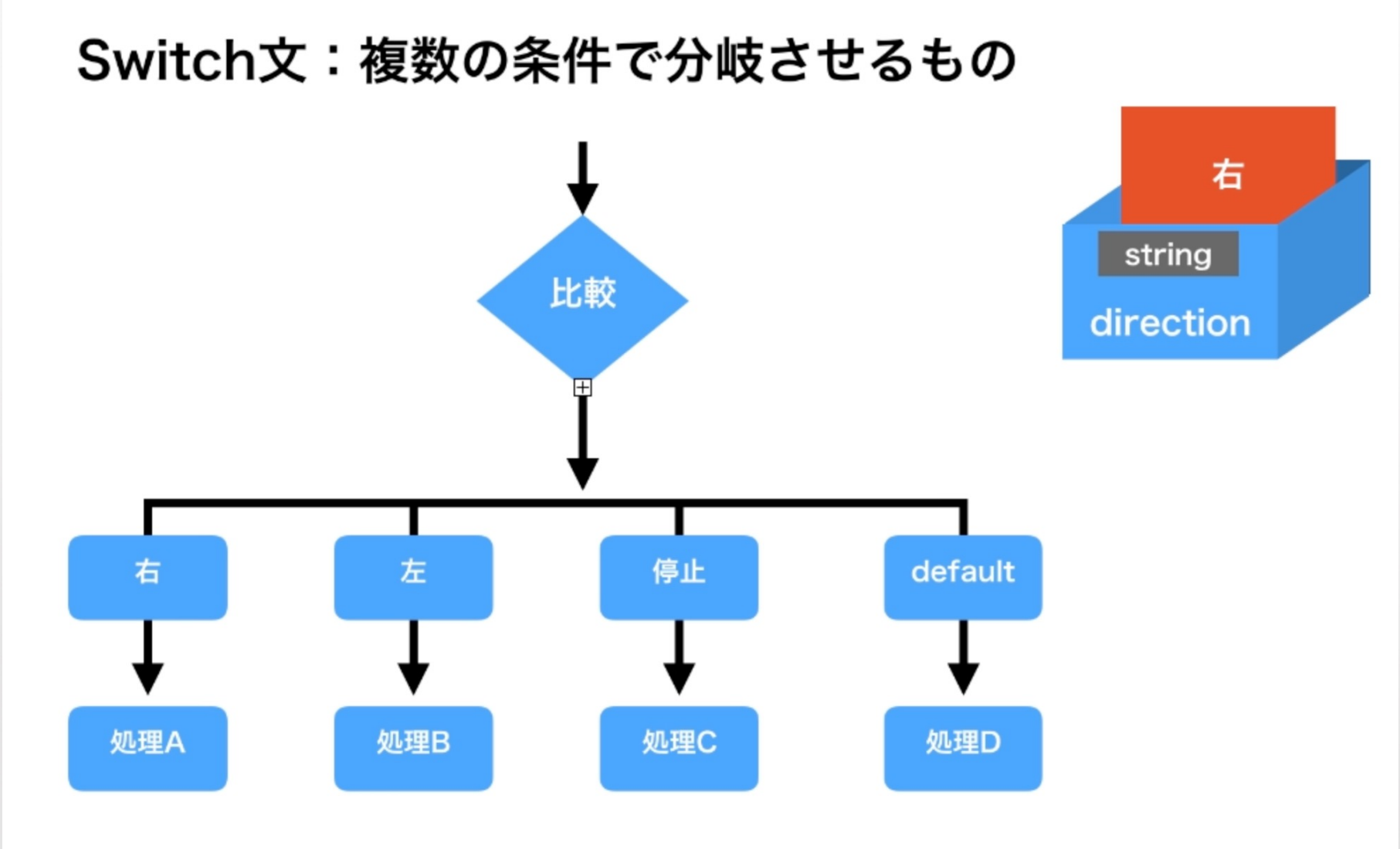
50kg以上でも、初めの条件に入ってしまう。

```
if (mass >= 20)
{
    Debug.Log("20Kg以上");
}
else if (mass >= 50)
{
    Debug.Log("50Kg以上");
}
else
{
    Debug.Log("軽い！");
}
```

3－4．Switch文



3－4．Switch文


Switch文：複数の条件で分岐させるもの
比較
右
左
停止
default
処理A
処理B
処理C
処理D
右
string
direction

3－4． Switch文

✅ if と switch の使い分け

項目	`if` 文	`switch` 文
🔁 比較対象	**範囲や複雑な条件**	**値が一致するか（＝）の比較**
🔢 比較できる型	数値・文字列・bool・論理式など	基本型（int, string, enumなど）
🔀 条件の種類	`>`, `<`, `&&`, `	
🧠 読みやすさ	複雑な条件の制御に向いている	**値ごとに処理を分けたい時に見やすい**
📦 拡張性	柔軟で複雑な処理も可能	シンプルな条件に限定される

3－4． Switch文

test3_4.csを作成し実際にやってみましょう！

出来た人は、以下の問題にチャレンジしてみてください！

＜問題＞

右のswitch文に「ダッシュ」というcaseを追加し、「処理E」と出力させてください

```
void Start()
{
    string direction;
    direction = "右";

    switch (direction)
    {
        case "右":
            Debug.Log("処理A");
            break;
        case "左":
            Debug.Log("処理B");
            break;
        case "停止":
            Debug.Log("処理C");
            break;
        default:
            Debug.Log("処理D");
            break;
    }
}
```

3－4．Switch文

＜答え＞

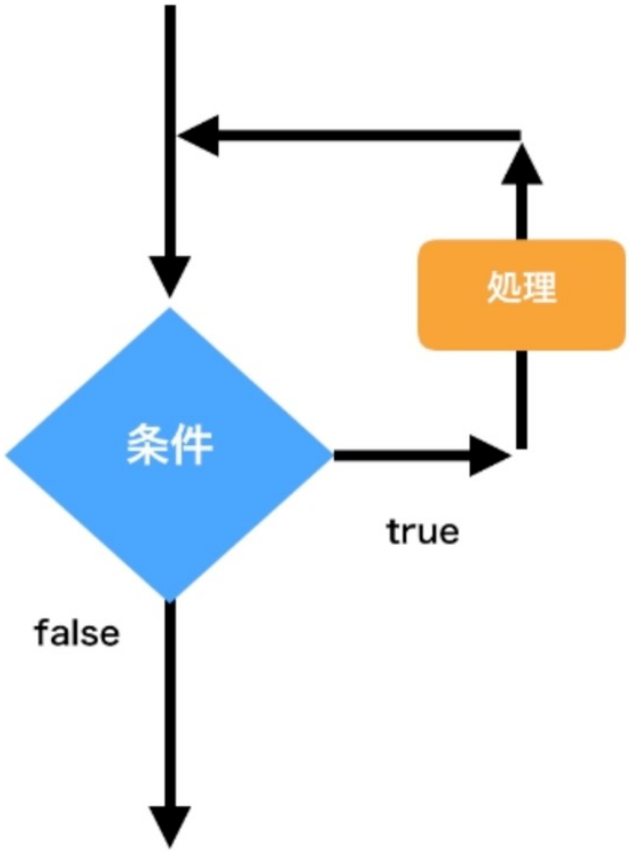
```
switch (direction)
{
    case "右":
        Debug.Log("処理A");
        break;
    case "左":
        Debug.Log("処理B");
        break;
    case "停止":
        Debug.Log("処理C");
        break;
    case "ダッシュ":
        Debug.Log("処理E");
        break;
    default:
        Debug.Log("処理D");
        break;
}
```

４．While文とfor文

１．while文
２．for文

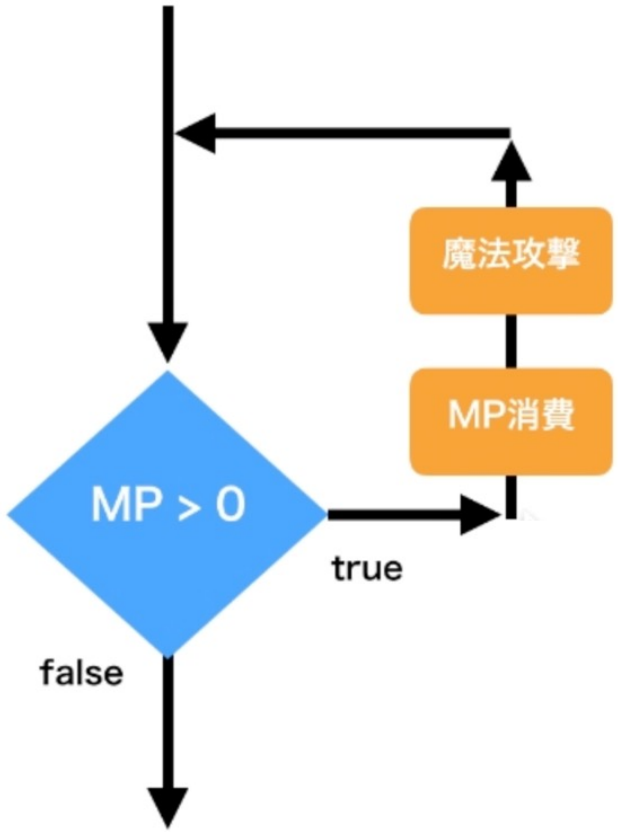
４－１．　while文

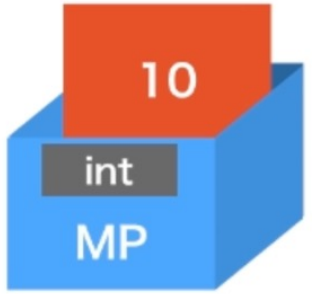
while：条件を満たしている間、処理を繰り返す



4－1．　while文

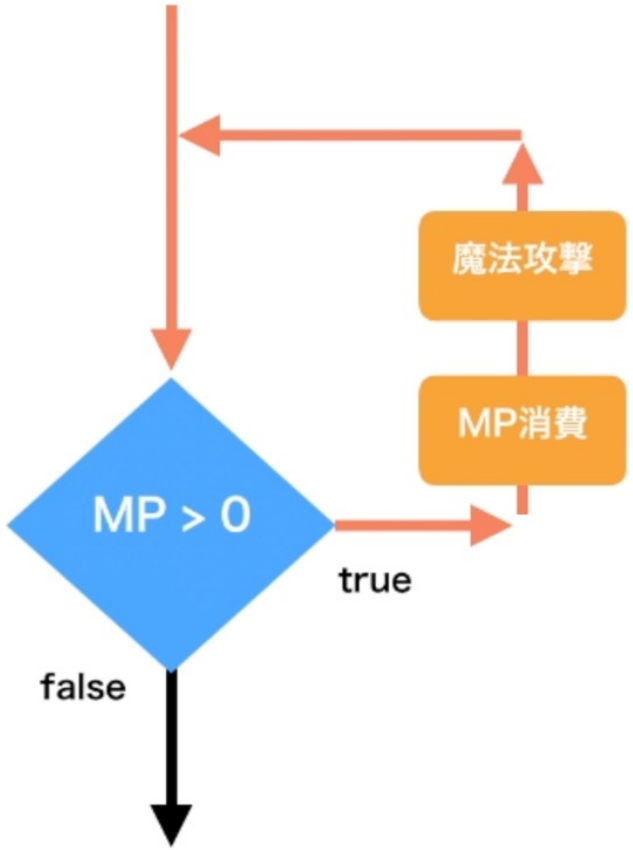
while：条件を満たしている間、処理を繰り返す

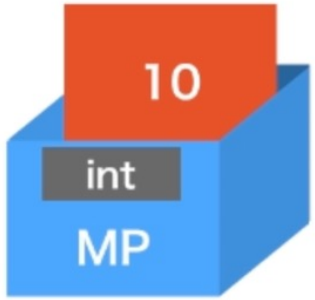




4－1．　while文

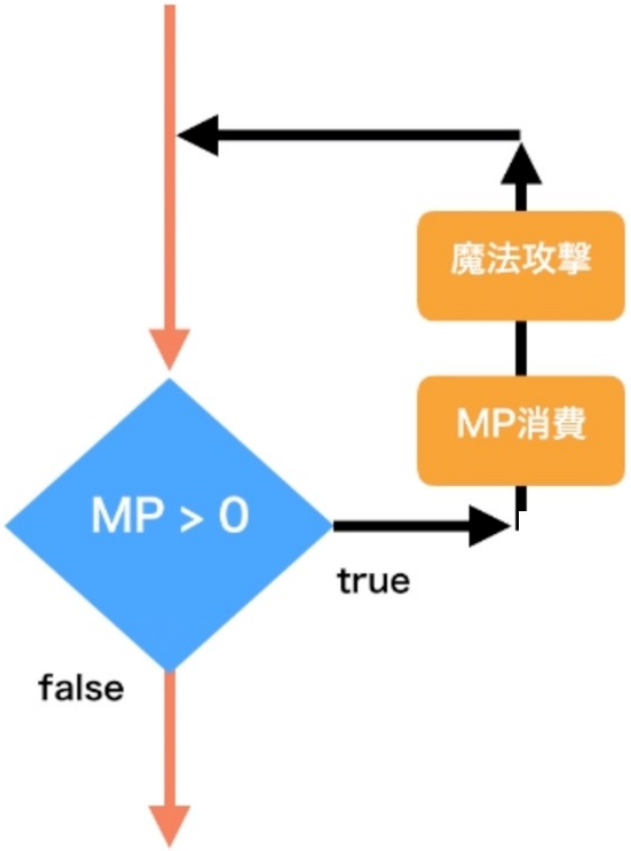
while：条件を満たしている間、処理を繰り返す

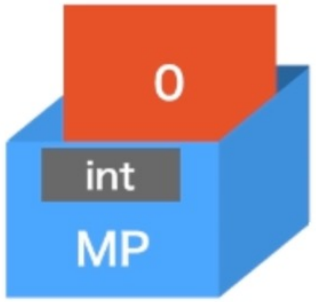




4－1．　while文

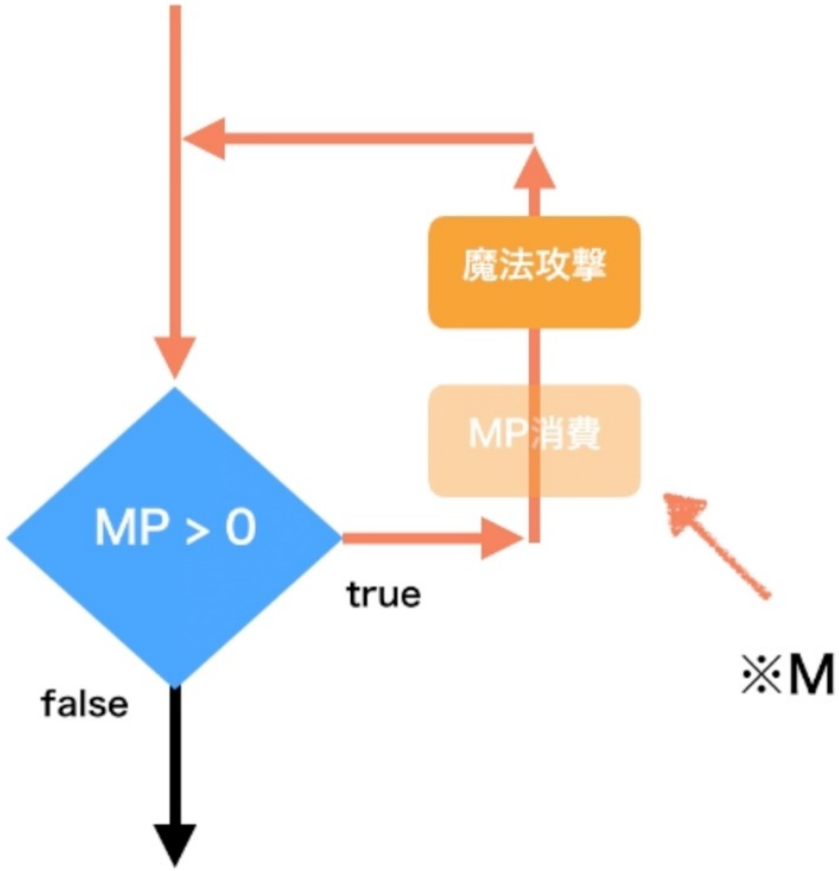
while：条件を満たしている間、処理を繰り返す

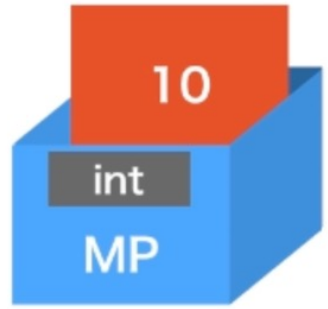




4－1．　while文

while：条件を満たしている間、処理を繰り返す





※MP消費がなければ無限ループ

４－１．　while文

記述方法は、以下のような形になります。
条件の欄に2項で実施した、比較演算子や論理演算子またはbool型が入ります。

```
while（条件式）
{
    // 条件式が true の間、繰り返し実行される処理
}
```

Test4_1.csを作成し、2ページ前のフローチャートを上記の形式に沿って作成してみてください。(mpは1ずつ減らしてください)

＜練習問題＞

0から5までカウントしてDebug.Logを出力してください。

４－１．　while文

＜答え＞

```
void Start()
{
    int mp = 10;

    while (mp > 0)
    {
        mp--;
        Debug.Log("魔法攻撃");
    }
}
```

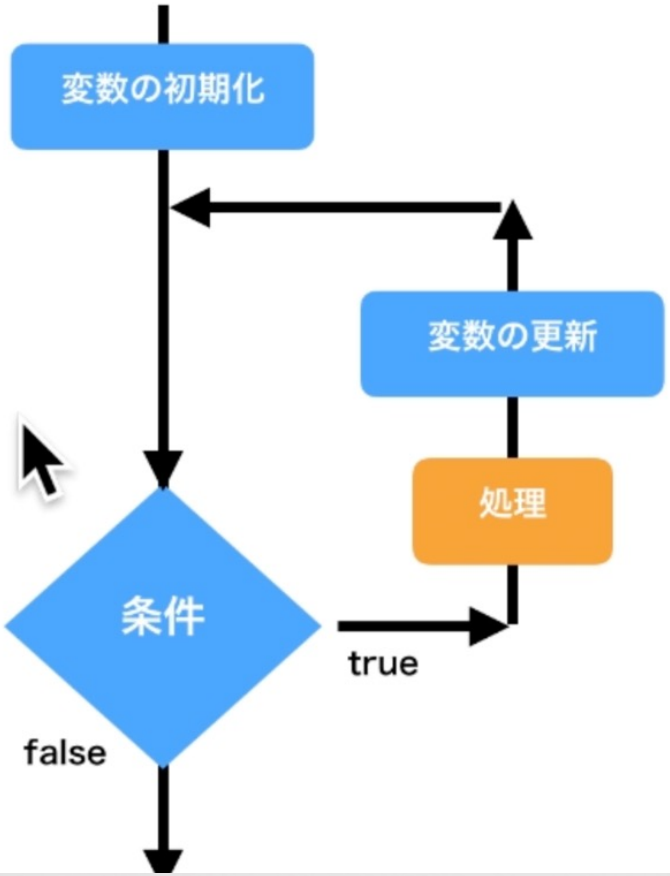
4－1．　while文

＜練習問題の答え＞

```
void Start()
{
    int count = 0;

    while (count < 6)
    {
        Debug.Log(count);
        count++;
    }
}
```

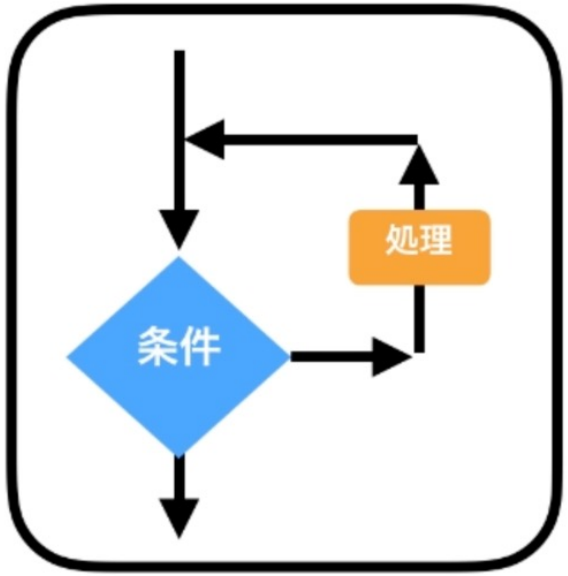
4－2． for文

for：特定の回数同じ処理を繰り返す



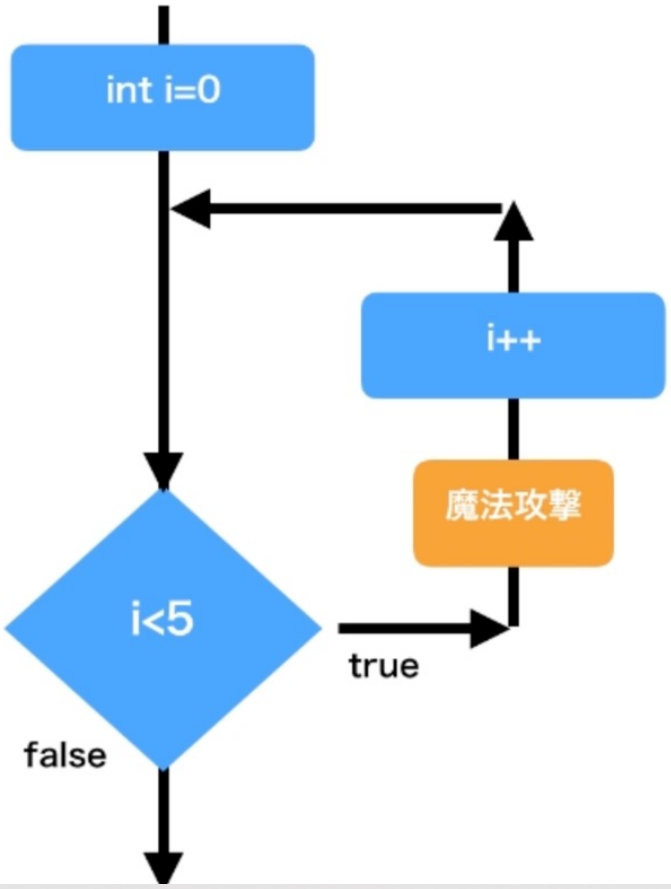
whileとの違い

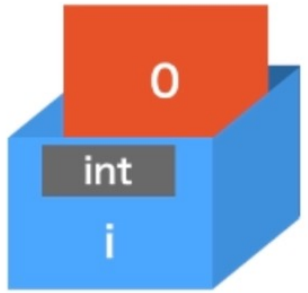
変数の初期化と更新がセット



4－2．　for文

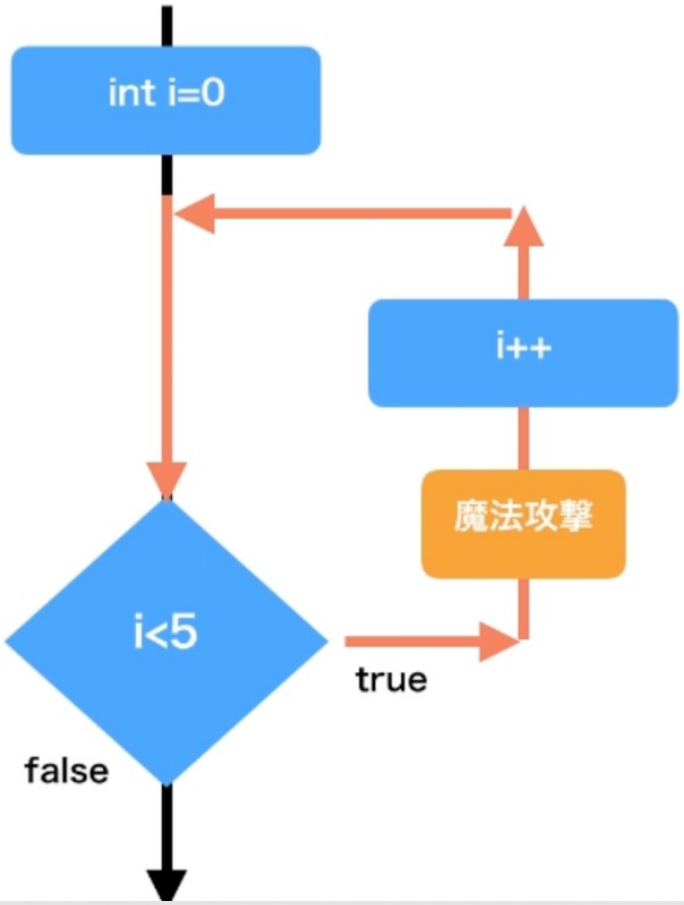
for：特定の回数同じ処理を繰り返す

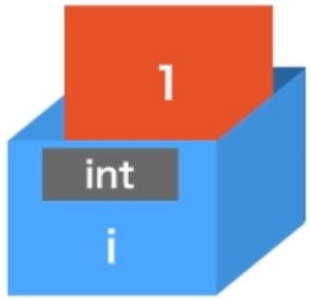




4－2．　for文

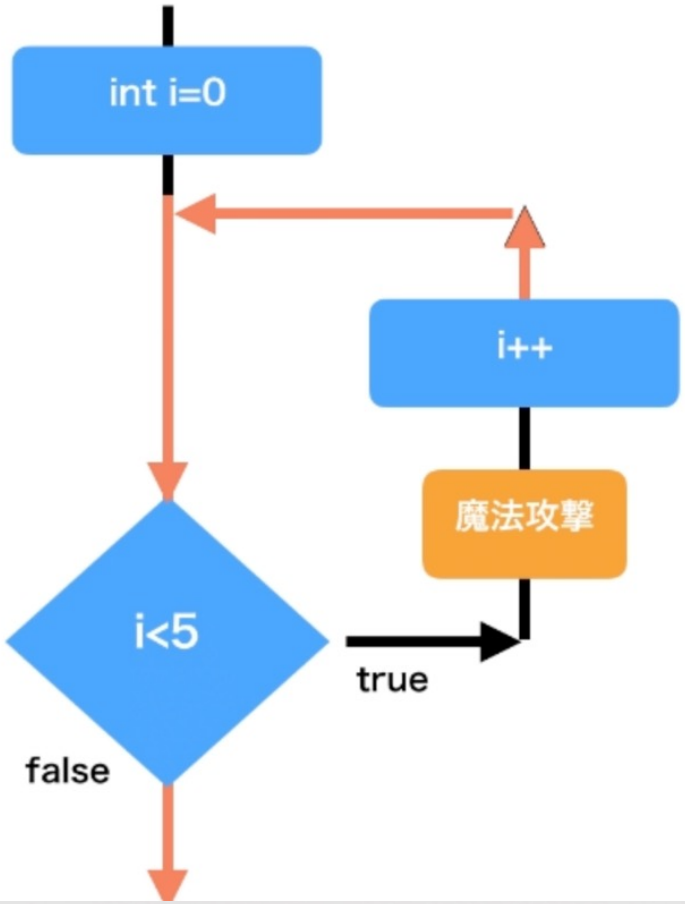
for：特定の回数同じ処理を繰り返す

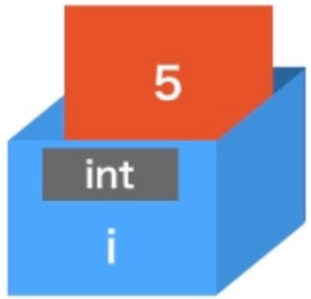




4－2．　for文

for：特定の回数同じ処理を繰り返す





４－２．　for文

記述方法は、以下のような形になります。

初期化、条件式、更新処理の3つが1行に入っているのが特徴になります。

```
for（初期化; 条件式; 更新）
{
    // 繰り返し実行される処理
}
```

Test4_2.csを作成し、2ページ前のフローチャートを上記の形式に沿って作成してみてください。

4－2．　for文

＜答え＞

```
void Start()
{
    for (int i = 0; i < 5; i++)
    {
        Debug.Log("魔法攻撃");
    }
}
```

出来た人は、以下の問題にチャレンジしてみてください！

＜問題＞

10回ループさせてください。

毎回魔法攻撃を出力、4回目のループだけ通常攻撃を出力

4－2．　for文

＜解答例＞

```
//10回ループさせてください。
//毎回魔法攻撃を出力、4回目のループだけ通常攻撃を出力
for (int i = 0; i < 10; i++)
{
    if (i != 3)
    {
        Debug.Log("魔法攻撃");
    }
    else
    {
        Debug.Log("通常攻撃");
    }
}
```

4－2．　for文

＜FizzBuzz問題にチャレンジしてみよう！＞

皆さんはFizzBuzzというゲームをご存知ですか？
プログラマなら大抵のかたは聞いたことがあるかと思いますが元は単純なパーティゲームで
– 2人以上のプレイヤーが1から順番に数字を発言していく
– 3で割り切れるときは「Fizz」を発言
– 5で割り切れるときは「Buzz」を発言
– 両方で割り切れるときは「FizzBuzz」を発言
– 間違えた人から脱落
細かな違いはあるでしょうが大体上記のようなルールのゲームです。

FizzBuzzとは？

1から100までのループ
Fizz/Buzz/FizzBuzz以外は数値をDebug.Logに表示

このFizzBuzzをプログラムとして画面に出力するコードを書くことが
FizzBuzzとはプログラマがプログラムを書くことができるか調べるクイズのようなものです。
当然ながらどのプログラム言語を選んでも解くことはできるのでプログラムの勉強を始めた方は一度解いてみてください。

4－2．　for文

＜解答例＞

```
using UnityEngine;
public class FizzBuzz : MonoBehaviour {
    public int start = 1;
    public int end = 100;
        void Start () {
        for (int i = start; i <= end; i++)
        {
            if (i % 15 == 0)
            {
                Debug.Log("FizzBuzz");
            }
            else if (i % 3 == 0)
            {
                Debug.Log("Fizz");
            }
            else if (i % 5 == 0)
            {
                Debug.Log("Buzz");
            }
            else
            {
                Debug.Log(i);
            }
        }
    }
}
```

5．発展問題

＜問題＞

ガチャのシステムを作ってみましょう！！

(1) ゲーム実行時に以下の「」のDebug.Logを10回出力してください。

確率は以下の通りです。

0.002%：「SSS」/ 0.098%：「SS」/ 0.9%：「S」/

19%：「A」/ 30%：「B」/ 50%：「C」

ランダムは以下の標準関数を使用してください。

(以下の例は0から8の9個の中からランダム)

```
void Start()
{
    int random = Random.Range(0, 9);
}
```

6．配列

＜配列とは＞

配列とは、同じ型のデータをまとめて管理する仕組みのこと。

複数の値を1つの変数で扱い時に使います。

🌟 書き方（基本形）：

csharp　コピーする　編集する

```
型[] 配列名 = new 型[要素数];
```

🔍 例：

csharp　コピーする　編集する

```
int[] scores = new int[5];
```

6．配列

＜配列に値を代入する＞

```
scores[0] = 80;
scores[1] = 90;
scores[2] = 75;
scores[3] = 60;
scores[4] = 85;
```

配列の要素には、配列名[インデックス]でアクセスします。インデックスは0から始まるので注意してください。(要素数が5の場合は、0,1,2,3,4の5つ)

宣言と同時に代入も可能です。

```
int[] scores = { 80, 90, 75, 60, 85 };
```

６．配列

＜Lengthプロパティ＞

配列名.Lengthでその配列に入っている要素の「数」を取得出来ます。

```
int[] numbers = { 10, 20, 30, 40, 50 };

Debug.Log(numbers.Length); // 出力：5
```

上記の例では、要素数は5なのでDebug.Logには5が返ります。

６．配列

＜問題＞

Test6.csを作成し、以下の実装を考えてみてください。

(1)以下の点数を配列に入れて、合計点を求めて表示してください。

Aさん：70、Bさん：85、Cさん：90

Dさん：60、Eさん：100

(2) (1)の点数の平均点を求めてください。

(3) (1)の最大点を求めてください。

6．配列

<解答>

```
int[] scores = { 70, 85, 90, 60, 100 };

// 合計点
int sum = 0;
for (int i = 0; i < scores.Length; i++)
{
    sum += scores[i];
}
Debug.Log("合計点: " + sum);

// 平均点
float average = (float)(sum / scores.Length);
Debug.Log("平均点: " + average);

// 最大点
int max = scores[0];
for (int i = 1; i < scores.Length; i++)
{
    if (scores[i] > max)
    {
        max = scores[i];
    }
}
Debug.Log("最高点: " + max);
```

7．List

List型について

- 配列とほぼ同じ
- 追加と削除がしやすい
- 要素の個数が決まっていないときに便利

7． List

＜使い方＞

Listの宣言と初期化

```
List<int> numbers = new List<int>(); // 空の List を作成
List<string> names = new List<string> { "Alice", "Bob", "Charlie" }; // 初期値を設定
```

要素の追加(addで追加)

```
List<int> numbers = new List<int>();
numbers.Add(10);  // 10を追加
numbers.Add(20);  // 20を追加
numbers.Add(30);  // 30を追加
```

要素の削除 (Removeは指定した数値、RemoveAtはインデックス)

```
numbers.Remove(20);  // 20を削除
numbers.RemoveAt(0); // 0番目の要素を削除
```

7． List

＜使い方＞

Listの要素数を取得。(配列のLengthと一緒)

```
int size = numbers.Count; // 要素数を取得
```

＜問題＞

Test7.csを作成し、以下の実装を考えてみてください。

List bを使って1000個配列を作ってください。格納するデータは0から1ずつ増やしていってください。

(例 b[0]は0、b[1]は1・・・)

```
List<int> b = new List<int>();
```

7．List

<解答>

```
List<int> b = new List<int>();
for (int i = 0; i < 1000; i++)
{
    b.Add(i);
}
```

８．メソッドとクラス

１．メソッド(関数)について
２．クラスについて

８－１．メソッド(関数)について

＜メソッドとは＞

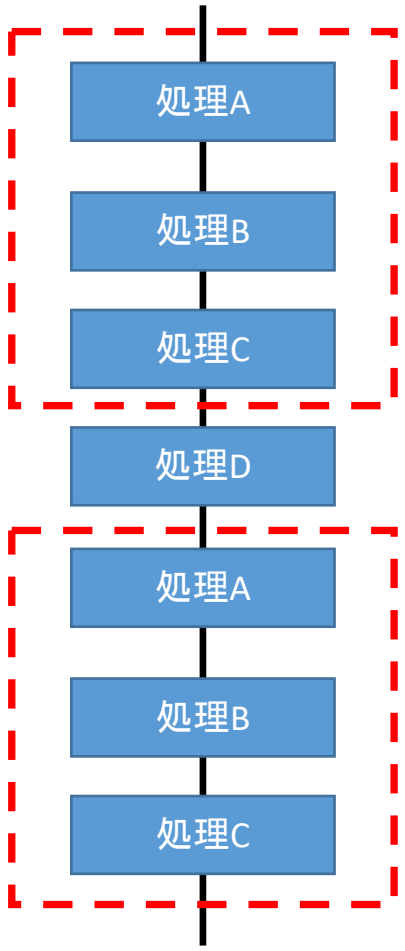
メソッドは、特定の操作や計算を行う関数です。
メソッドを使うことで、コードの再利用性が高まり、処理を分割して管理することができます。

大きく2つの使い方があります。

１：処理をまとめる
２：特定の機能を作成したい

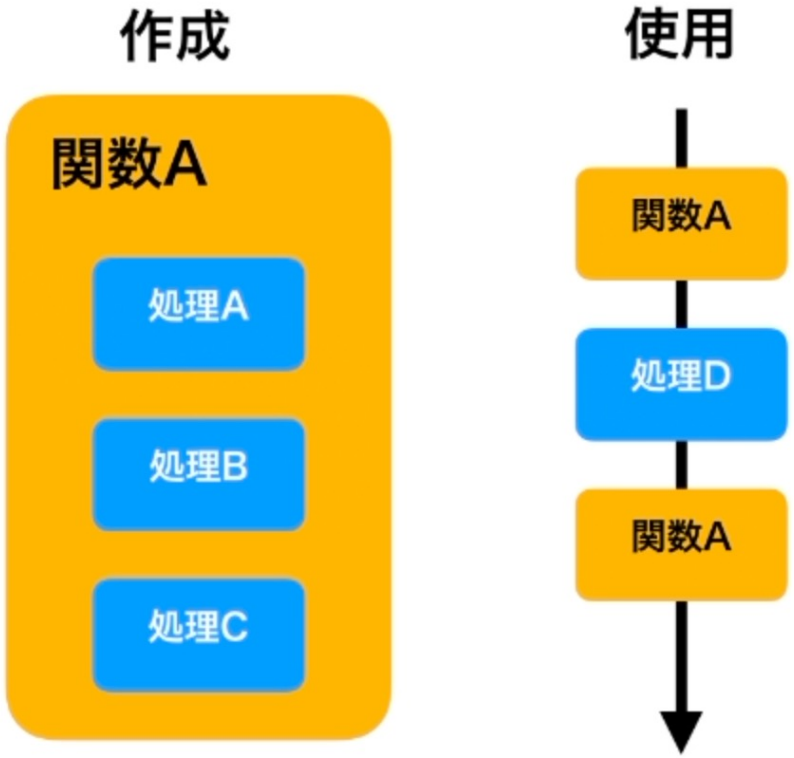
８－１．メソッド(関数)について

関数（メソッド）：処理をまとめたもの



８－１．メソッド(関数)について

関数：処理をまとめたもの



利点

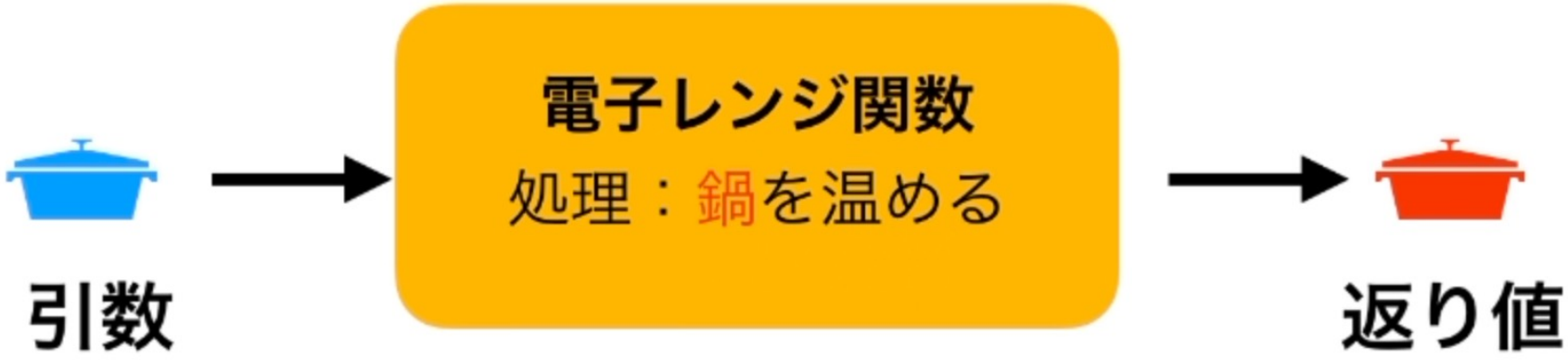
- 同じコードを書かなくていい
- 流れが分かりやすくなる

８－１．メソッド(関数)について

関数の種類：引数、返り値

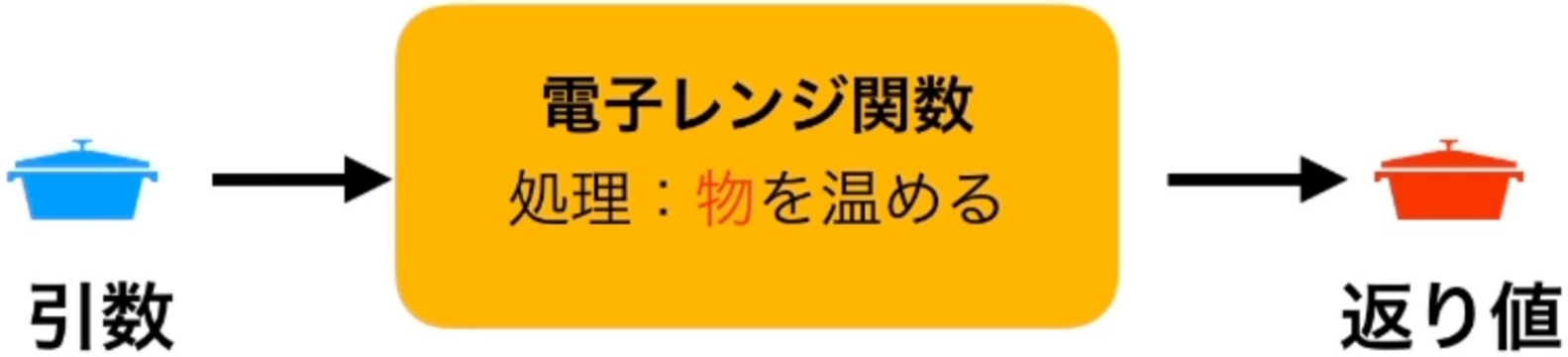
８－１．メソッド(関数)について

関数の種類：引数、返り値



８－１．メソッド(関数)について

関数の種類：引数、返り値



引数と返り値は作成の際に種類を指定

８－１．メソッド(関数)について

＜メソッドの構成＞

```
[アクセス修飾子] 戻り値の型 メソッド名(引数の型 引数名)
{
    // 処理内容
    return 戻り値;   // 必要なら戻り値を返す
}
```

アクセス修飾子：メソッドのアクセス範囲(後述)

戻り値の型：メソッドが返す値のデータ型(例えば int や string)
戻り値がない場合は void を使います。

メソッド名：メソッドを呼び出すための名前。

引数：メソッドに渡すデータ。引数がない場合もあります。

８－１．メソッド(関数)について

＜メソッドの例＞

```
// 引数を受け取り、合計を返すメソッド
public int AddNumbers(int num1, int num2)
{
    return num1 + num2;
}

// 戻り値がないメソッド（何も返さない）
public void SayHello()
{
    Debug.Log("こんにちは！");
}
```

AddNumbersメソッド：2つの整数を受け取り、合計を戻り値として返す。

SayHelloメソッド：void を使い何も返さず、「こんにちは！」と表示する。

右記のように呼び出す。

```
1   int result = AddNumbers(3, 5);  // 3 + 5 = 8
2   Debug.Log(result);              // 8 を表示
3
4   SayHello();  // こんにちは！と表示
5
```

８－１．メソッド(関数)について

＜問題＞

引数で2つの数値を渡してください。(どちらもint型)

2つの数値を割り切ることが出来たらTrue、割り切ることが出来ないなら

Falseを戻り値として返してください。

８－１．メソッド(関数)について

＜解答例＞

```
bool TestA(int a, int b)
{
    bool isResult;
    int value;

    isResult = false;
    value = a % b;

    if (value == 0)
    {
        isResult = true;
    }

    return isResult;
}
```

8－2．クラスについて

＜クラス＞

クラスとは設計図のことで、オブジェクト(UnityであればGameObject)は
その設計図から作られた実体のこと。
(クラス：たい焼きの型、オブジェクト：たい焼きの型から作られたたい焼き)

クラスには、フィールド(変数)とメソッド(関数)が存在します。

```
public class クラス名
{
    // フィールド（変数）
    // メソッド（関数）
}
```

8－2．クラスについて

Unityにおいては、実はあまりクラスを意識しなくても実装出来る仕組みとなっていますが、C#の一番重要なところなのでぜひ覚えてください！

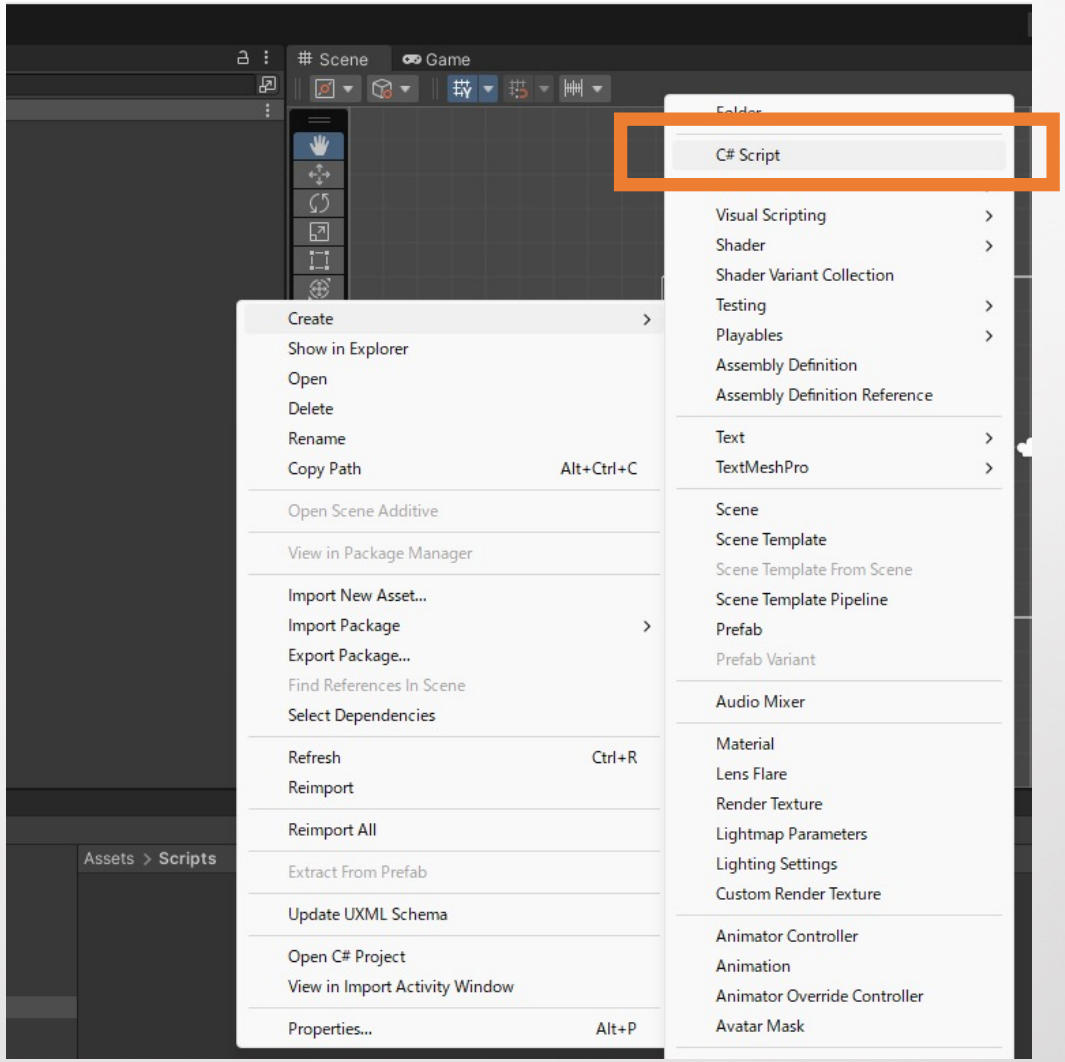
クラス：変数と関数をまとめたもの

“もの”を表現できる

8－2．クラスについて

ちなみにUnityだと、C#Scriptを作成する際に自動的にテンプレートを作成してくれますね！

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class NewBehaviourScript1 : MonoBehaviour
6  {
7      // Start is called before the first frame update
8      void Start()
9      {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
19
```



8－2．クラスについて

もちろん自分でも作成出来るため、
まずはTest8_2.csを作成してみましょう。

その後、Carというクラスを作りましょう！

```
public class Test8_2 : MonoBehaviour
{
    // 省略、何も書かなくても大丈夫
}
```

```
public class Car
{
    // フィールド（属性、状態）
    public string color;
    public int speed;

    private string handle;

    // メソッド（機能、動作）
    public void Drive()
    {
        Debug.Log("車が走っています！");
    }

    public void Stop()
    {
        Debug.Log("車が止まりました。");
    }

    private void CarCheck()
    {
        HandleCheck();
    }

    private void HandleCheck()
    {
        Debug.Log("ハンドルOKです");
    }
}
```

8－2．クラスについて

出来たら次に、Test8_2.csから
Carクラスを作成してみましょう！
この「作成する」、ということを
インスタンスと呼びます！

```
public class Test8_2 : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        // Car クラスのインスタンス（実体）を作成
        Car myCar = new Car();

        // フィールドに値を設定
        myCar.color = "赤";
        myCar.speed = 120;
        //myCar.handle = "NG";

        // メソッドを呼び出し
        myCar.Drive();  // 車が走っています！
        myCar.Stop();   // 車が止まりました。
        //myCar.HandleCheck();

        Debug.Log("車の色は " + myCar.color);
        Debug.Log("車のスピードは " + myCar.speed + "km/h");
    }
```

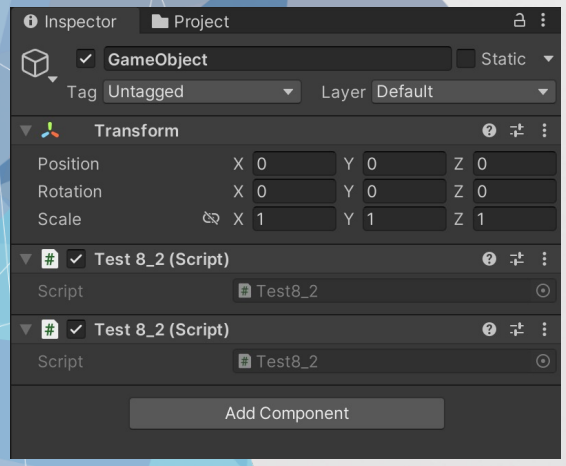
8－2．クラスについて

＜new＞

クラスの実体(インスタンス)を作るためのキーワードです。

Unityの場合は、GameObjectにアタッチした時点で裏でnewが呼ばれていて、インスタンスが行われています。

Unityの機能を使わない(StartやUpdateがいらない)クラスの場合は、
先ほどのCarクラスのようなnewを使った自前のクラスの作成も可能です。
(サーバーと通信するクラス、プレイヤーのデータを管理するクラスなど)



設計図から実体を作るので、以下のように大量生産も出来ますね！

```
Car car1 = new Car();
Car car2 = new Car();
```

8－2．クラスについて

＜アクセス修飾子＞

アクセス修飾子とは、アクセス制限を示した修飾子です。

public　　→　他のクラスからアクセス可能。

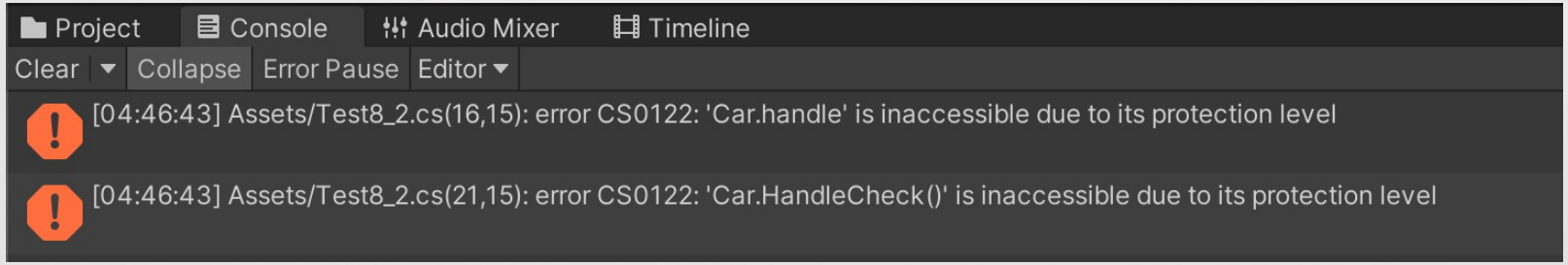
private　　→　自身のプログラムの中からしかアクセスできない。

(protected　→　継承したクラスのみアクセス可能。)

試しに、コメントアウトした以下の2つの“//”を削除するとエラーになると思います。

・myCar.handle = "NG";

・myCar.HandleCheck();



8－2．クラスについて

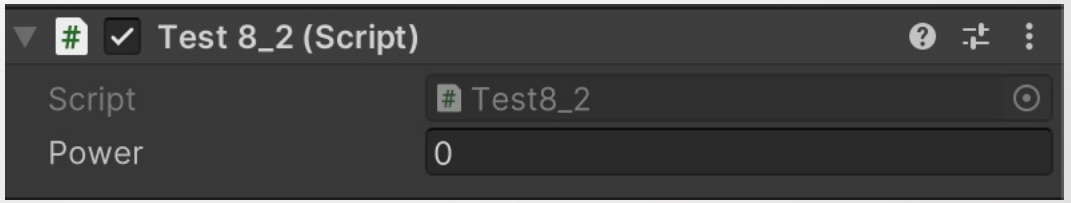
記述を略すと、private扱いになります。

int test = 10; ⇨ private int test = 10;

Unityの場合ですが、publicにするとUnity上から値を設定することが出来ます。ただ、扱いはprivateにしたいけどUnity上から値を設定したい場合は「SerializeField」という機能を使うと実装することが出来ます。

```
public class Test8_2 : MonoBehaviour
{
    public int power;
```

```
public class Test8_2 : MonoBehaviour
{
    [SerializeField] private int power;
```



8－2．クラスについて

＜問題＞

①test8_2のクラスで、Carクラスのhandle情報が知りたい。Carクラスのhandle情報を取得する実装をしてください。
(handleの変数自体をpublicにするのは無し)

②test8_2.csのファイル内にPlayerクラスを作成して以下の実装をし、test8_2クラスから名前とhpを設定して出力してください。

フィールド

・名前(string)、hp(int)でどちらもprivate

メソッド

・他のクラスから名前とhpを設定出来るメソッド

・他のクラスから設定した名前とhpをDebug.Logに出力するメソッド

8－2．クラスについて

＜解答＞

```
public string GetHandleInfo()
{
    return handle;
}
```

```
public class Test8_2 : MonoBehaviour
{
    [SerializeField] private int power;

    // Start is called before the first frame update
    void Start()
    {
        Player player1 = new Player();

        player1.SetName("プレイヤー");
        player1.SetHp(200);
        player1.ShowInfo();
    }
```

```
public class Player
{
    // フィールド（属性、状態）
    private string name;
    private int hp;

    // メソッド（機能、動作）
    public void SetName(string str)
    {
        name = str;
    }

    public void SetHp(int value)
    {
        hp = value;
    }

    public void ShowInfo()
    {
        Debug.Log(name);
        Debug.Log(hp);
    }
}
```

8－2．クラスについて

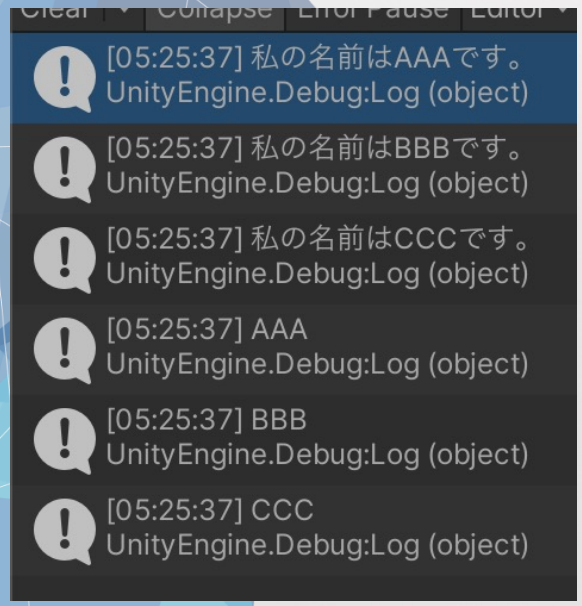
＜応用問題＞

StudentクラスとSchoolクラスを作成し、test8_3.csから生徒3人追加してそれぞれ自己紹介をして、その後全生徒の名前を出力してください。

①Studentはname(string)を持ち、自己紹介をするIntroduce()のメソッドを持ちます。(nameはprivate、Introduceはpublic)

②SchoolはList<Student>をフィールドに持ち、AddStudent(Student)メソッドで生徒を追加できる

③SchoolはShowAllStudents() メソッドで全生徒の名前を出力する



①のIntroduceは、私の名前は(設定した名前)です、と出力してください。

8－2．クラスについて

＜解答＞

```
public class Student
{
    private string name;

    public void SetName(string str)
    {
        name = str;
    }

    public void Introduce()
    {
        Debug.Log("私の名前は" + name + "です。");
    }

    public string GetName()
    {
        return name;
    }
}
```

```
public class School
{
    private List<Student> students = new List<Student>();

    public void AddStudent(Student student)
    {
        students.Add(student);
    }

    public void ShowAllStudents()
    {
        for(int i = 0; i < students.Count; i++)
        {
            Debug.Log(students[i].GetName());
        }
    }
}
```

```
public class Test8_2 : MonoBehaviour
{
    [SerializeField] private int power;

    // Start is called before the first frame update
    void Start()
    {
        // 学校のインスタンスを作成
        School mySchool = new School();

        // 学生を作成して追加
        Student s1 = new Student();
        Student s2 = new Student();
        Student s3 = new Student();
        s1.SetName("AAA");
        s2.SetName("BBB");
        s3.SetName("CCC");

        mySchool.AddStudent(s1);
        mySchool.AddStudent(s2);
        mySchool.AddStudent(s3);

        // 各生徒が自己紹介
        s1.Introduce();
        s2.Introduce();
        s3.Introduce();

        // 生徒一覧を表示
        mySchool.ShowAllStudents();
    }
```