

4강

목표 Contents

01

Unity basic
유니티 기본 조작,

02

유니티 스크립팅
기본 문법 점검

01

Unity basic

유니티 기본 조작 복습



이번 시간

C# 복습

참고 문서

마이크로 소프트 C# 자습서

<https://docs.microsoft.com/ko-kr/dotnet/csharp/>



스크립트와 클래스에 대해 알아보시다.
일단 만들어봅시다.

스크립트

유니티의 스크립트파일(.cs)은 인스턴스화가 가능한 클래스를 하나씩 정의하고 있음.

클래스의 인스턴스화

- 특정 오브젝트에 스크립트를 컴포넌트로 추가해서 실제로 동작하도록 하는 과정
- 여러 오브젝트에 하나의 스크립트 파일을 추가해서 동작을 복제사용 가능

C#에서의 타입

값 형식(value) vs 참조 형식(reference)

값 형식의 변수에는 형식의 인스턴스가 포함
참조 형식 변수에는 개체에 대한 참조(reference)가 저장됨

<https://docs.microsoft.com/ko-kr/dotnet/csharp/language-reference/builtin-types/built-in-types>

클래스

클래스를 정의 한다 = C# 컴파일러에게 필요한 변수 함수 들에 대해 알려준다.

참조 형식에 해당된다.

참조형식의 변수를 선언하면 new 키워드를 통해 명시적으로(생성자) 개체를 만들 수도 있고 개체를 할당하지 않을 시 null

클래스의 인스턴스 생성자

class 또는 struct를 만들 때마다 해당 생성자가 호출됨.

C#

생성자 사용

```
class Coords
{
    public int x, y;

    // constructor
    public Coords()
    {
        x = 0;
        y = 0;
    }
}
```

클래스의 인스턴스 생성자

생성자 사용 (new)

C#

```
public class Taxi
{
    public bool IsInitialized;

    public Taxi()
    {
        IsInitialized = true;
    }
}

class TestTaxi
{
    static void Main()
    {
        Taxi t = new Taxi();
        Console.WriteLine(t.IsInitialized);
    }
}
```

C#에서의 변수와 데이터 타입

Int (정수형)

float(부동소수점)

Bool (참/거짓)

String(문자열)

Vector3(위치데이터) = (0.0.0) x,y,z

Public 으로 선언시 유니티 인스펙터에서 접근,편집이 가능,
뿐만 아니라 다른 클래스에서도 접근이 가능함.

<https://docs.microsoft.com/ko-kr/dotnet/csharp/language-reference/builtin-types/built-in-types>

C#에서의 변수와 데이터 타입

var 키워드.

Var키워드를 통해서 변수 또는 상수를 선언시, 컴파일러가 타입을 유추

```
var i = 10; // Implicitly typed.
```

```
int i = 10; // Explicitly typed.
```

변수 관련 키워드

Const 키워드

상수, 지역상수를 선언 시 사용.
선언 시에만 사용할 수 있는 키워드
컴파일 타임 상수 : 바꾸고 싶다면 다시 컴파일해야함...

유사품 : readonly (차이점 - readonly는 런타임상수도 가능하다. 생성자에서 따로 초기화가능)

변수 관련 키워드

Const 키워드

상수, 지역상수
선언 시에만
컴파일 타임 상수

유사품 : readonly

C#

```
public class ConstTest
{
    class SampleClass
    {
        public int x;
        public int y;
        public const int C1 = 5;
        public const int C2 = C1 + 5;

        public SampleClass(int p1, int p2)
        {
            x = p1;
            y = p2;
        }
    }

    static void Main()
    {
        var mC = new SampleClass(11, 22);
        Console.WriteLine($"x = {mC.x}, y = {mC.y}");
        Console.WriteLine($"C1 = {SampleClass.C1}, C2 = {SampleClass.C2}");
    }
}

/* Output
x = 11, y = 22
C1 = 5, C2 = 10
*/
```

생성자에서 따로 초기화가능)

열거형 (Enum)

변수에 가능한 범위값을 정해 담게 하는 구조
그 자체로는 변수는 아님

변수값의 제한하거나 유효성을 유지시키는데 도움되는 구조

배열

동적 배열
정적 배열

반복문

foreach / for / while

무한 루프 주의

함수

하나의 블록으로 묶인 코드

유니티 함수
사용자가 직접 만든 함수

메서드 시그니처

- public private protected internal 등의 선택적 액세스 수준을 나타내는 키워드
- Abstract, virtual 등의 선택적 한정자를 나타내는 키워드
- 메서드(함수) 이름
- 메서드 매개 변수

위 키워드들을 조합하여 메서드 시그니처를 구성.

메서드 시그니처

- public private p
- Abstract, virtual
- 메서드(함수) 이
- 메서드 매개 변

위 키워드들을 조합

C#

복사

```
using System;

abstract class Motorcycle
{
    // Anyone can call this.
    public void StartEngine() { /* Method statements here */ }

    // Only derived classes can call this.
    protected void AddGas(int gallons) { /* Method statements here */ }

    // Derived classes can override the base class implementation.
    public virtual int Drive(int miles, int speed) { /* Method statements here */ return 1; }

    // Derived classes can override the base class implementation.
    public virtual int Drive(TimeSpan time, int speed) { /* Method statements here */ return 0; }

    // Derived classes must implement this.
    public abstract double GetTopSpeed();
}
```

메서드(함수) 호출

C#

```
public class Example
{
    public static void Main()
    {
        // Call with an int variable.
        int num = 4;
        int productA = Square(num);

        // Call with an integer literal.
        int productB = Square(12);

        // Call with an expression that evaluates to int.
        int productC = Square(productA * 3);
    }

    static int Square(int i)
    {
        // Store input argument in a local variable.
        int input = i;
        return input * input;
    }
}
```

메서드 호출

- 인스턴스 메서드 이거나 정적 (static)
- 개체를 먼저 인스턴스화 한 뒤, 해당 개체의 메서드를 호출 할 수 있음.
- 인스턴스화 된 개체를 통해 정적(static) 메서드를 호출하면 오류!

internal

- 동일한 어셈블리 파일내에서만 액세스 가능함을 나타낸 한정자

C#

복사

```
// Assembly1.cs
// Compile with: /target:library
internal class BaseClass
{
    public static int intM = 0;
}
```

C#


복사

```
// Assembly1_a.cs
// Compile with: /reference:Assembly1.dll
class TestAccess
{
    static void Main()
    {
        var myBase = new BaseClass(); // CS0122
    }
}
```

internal


- 동일한 어셈블리 파일내에서만 액세스 가능함을 나타낸 한정자

C#

 복사

```
// Assembly2.cs
// Compile with: /target:library
public class BaseClass
{
    internal static int intM = 0;
}
```

C#

 복사

```
// Assembly2_a.cs
// Compile with: /reference:Assembly2.dll
public class TestAccess
{
    static void Main()
    {
        var myBase = new BaseClass(); // Ok.
        BaseClass.intM = 444;         // CS0117
    }
}
```

컴파일러 오류 CS0117

<https://docs.microsoft.com/ko-kr/dotnet/csharp/misc/cs0117>

데이터 형식이 없는 상태에서 참조하는 경우 오류

static

정적 클래스는 인스턴스화 할 수 없다

즉, new 연산자를 사용하여 클래스 형식의 변수를 만들 수 없음

- 정적 멤버만 포함합니다.
- 인스턴스화할 수 없습니다.
- 봉인되어 있습니다.
- 인스턴스 생성자를 포함할 수 없습니다.

static

예 : Math 클래스
삼각, 로그 및 기타 일반 수학 함수에 대한 상수 및 정적 메서드를 제공

```
public double GetHeight()
{
    double x = GetRightSmallBase();
    return Math.Sqrt(Math.Pow(m_rightLeg,2.0) - Math.Pow(x,2.0));
}
```

C#

 복사

```
public static class Math
```

static

정적 클래스는 인스턴스

예 : Math 클래스
삼각, 로그 및 기타 일

```
[SecuritySafeCritical]
[__DynamicallyInvokable]
[MethodImpl(MethodImplOptions.InternalCall)]
public static extern double Sin(double a);

/// <summary>지정된 각도의 탄젠트를 반환합니다.</summary>
/// <param name="a">라디안 단위의 각도입니다.</param>
/// <returns>
///     <paramref name="a" />의 접선입니다.
///     <paramref name="a" />가 <see cref="F:System.Double.NaN" />, <see cref="F:
/// </returns>
[SecuritySafeCritical]
[__DynamicallyInvokable]
[MethodImpl(MethodImplOptions.InternalCall)]
public static extern double Tan(double a);

/// <summary>지정된 각도의 하이퍼볼릭 사인을 반환합니다.</summary>
/// <param name="value">라디안 단위의 각도입니다.</param>
/// <returns>
///     <paramref name="value" />의 쌍곡선 사인입니다.
///     <paramref name="value" />가 <see cref="F:System.Double.NegativeInfinity"
/// </returns>
[SecuritySafeCritical]
[__DynamicallyInvokable]
[MethodImpl(MethodImplOptions.InternalCall)]
public static extern double Sinh(double value);

/// <summary>지정된 각도의 하이퍼볼릭 탄젠트를 반환합니다.</summary>
/// <param name="value">라디안 단위의 각도입니다.</param>
/// <returns>
///     <paramref name="value" />의 쌍곡선 접선입니다.
///     <paramref name="value" />가 <see cref="F:System.Double.NegativeInfinity"
///     값이 <see cref="F:System.Double.PositiveInfinity" />와 같으면 이 메서드는
///     <paramref name="value" />가 <see cref="F:System.Double.NaN" />과 같으면 (
/// </returns>
// ...
```

복사

상속과 클래스

상속이란 다른 클래스를 기능적으로 포함/흡수해 새로운 클래스를 만드는 개념.

원본 클래스는 기본 (base) 클래스 또는 원형 클래스, 등으로 불리움
이를 상속하여 파생되는 클래스를 서브 클래스 또는 파생 클래스, 등으로 부른다.

상속과 클래스

C#에서는 '단일' 상속만 허용됨.
한 클래스당, 하나의 기본클래스를 상속받을 수 있음.

(인터페이스는 여러 개 가능)

C#

```
abstract class Shape
{
    public abstract int GetArea();
}

class Square : Shape
{
    int side;

    public Square(int n) => side = n;

    // GetArea method is required to avoid a compile-time error.
    public override int GetArea() => side * side;

    static void Main()
    {
        var sq = new Square(12);
        Console.WriteLine($"Area of the square = {sq.GetArea()}");
    }
}

// Output: Area of the square = 144
```


상속과 오버라이드(재정의)

override 한정자

override 한정자는 기본 클래스에서 상속된 멤버들을 확장 또는 수정 하겠다는 의미

override 키워드에 의해 재정의된 메서드 = 재정의된 기본 메서드

정적 (static) 메소드는 오버라이드 불가

Virtual 또는 abstract

상속과 오버라이드

```
public class Employee
{
    public string name;

    // Basepay is defined as protected, so that it may be
    // accessed only by this class and derived classes.
    protected decimal basepay;

    // Constructor to set the name and basepay values.
    public Employee(string name, decimal basepay)
    {
        this.name = name;
        this.basepay = basepay;
    }

    // Declared virtual so it can be overridden.
    public virtual decimal CalculatePay()
    {
        return basepay;
    }
}
```

```
// Derive a new class from Employee.
public class SalesEmployee : Employee
{
    // New field that will affect the base pay.
    private decimal salesbonus;

    // The constructor calls the base-class version, and
    // initializes the salesbonus field.
    public SalesEmployee(string name, decimal basepay,
        decimal salesbonus) : base(name, basepay)
    {
        this.salesbonus = salesbonus;
    }

    // Override the CalculatePay method
    // to take bonus into account.
    public override decimal CalculatePay()
    {
        return basepay + salesbonus;
    }
}
```

```
static void Main()
{
    // Create some new employees.
    var employee1 = new SalesEmployee("Alice",
        1000, 500);
    var employee2 = new Employee("Bob", 1200);

    Console.WriteLine($"Employee1 {employee1.name} earned: {employee1.CalculatePay()}");
    Console.WriteLine($"Employee2 {employee2.name} earned: {employee2.CalculatePay()}");
}
```

상속과 관련된 키워드

virtual 키워드

파생클래스에서 재정의(override)를 허용한다는 의미

static, abstract, private, override 한정자와 함께 사용할 수 없음

```
public class Shape
{
    public const double PI = Math.PI;
    protected double x, y;

    public Shape()
    {
    }

    public Shape(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public virtual double Area()
    {
        return x * y;
    }
}

public class Circle : Shape
{
    public Circle(double r) : base(r, 0)
    {
    }

    public override double Area()
    {
        return PI * x * x;
    }
}
```

상속과 관련된 키워드

abstract 키워드

abstract class

: 클래스 자체가 인스턴스화 되지는 않고, 다른 파생 클래스의 기본(base)클래스 역할만 하겠다

abstract 멤버 (함수나 변수 등)

: 파생된 클래스에서 구현(implementation) 되어야만 하는 멤버

상속과 관련된 키워드

abstract 키워드

C#

```
abstract class Shape
{
    public abstract int GetArea();
}

class Square : Shape
{
    int side;

    public Square(int n) => side = n;

    // GetArea method is required to avoid a compile-time error.
    public override int GetArea() => side * side;

    static void Main()
    {
        var sq = new Square(12);
        Console.WriteLine($"Area of the square = {sq.GetArea()}");
    }
}

// Output: Area of the square = 144
```

상속과 관련된 키워드

abstract 키워드

```
abstract class BaseClass    // Abstract class
{
    protected int _x = 100;
    protected int _y = 150;
    public abstract void AbstractMethod();    // Abstract method
    public abstract int X    { get; }
    public abstract int Y    { get; }
}

class DerivedClass : BaseClass
{
    public override void AbstractMethod()
    {
        _x++;
        _y++;
    }

    public override int X    // overriding property
    {
        get
        {
            return _x + 10;
        }
    }

    public override int Y    // overriding property
    {
        get
        {
            return _y + 10;
        }
    }

    static void Main()
    {
        var o = new DerivedClass();
        o.AbstractMethod();
        Console.WriteLine($"x = {o.X}, y = {o.Y}");
    }
}
```

상속과 관련된 키워드

interface 키워드

일종의 계약.

인터페이스를 사용하는 클래스나 구조체는 인터페이스에 정의된 멤버를 구현

상속과 관련된 키워드

interface 키워드

일종의 계약.
인터페이스를 사용하는 클라

```
C#

interface IPoint
{
    // Property signatures:
    int X
    {
        get;
        set;
    }

    int Y
    {
        get;
        set;
    }

    double Distance
    {
        get;
    }
}

class Point : IPoint
{
    // Constructor:
    public Point(int x, int y)
    {
        X = x;
        Y = y;
    }

    // Property implementation:
    public int X { get; set; }

    public int Y { get; set; }

    // Property implementation
    public double Distance =>
        Math.Sqrt(X * X + Y * Y);
}

```

의된

```
class MainClass
{
    static void PrintPoint(IPoint p)
    {
        Console.WriteLine("x={0}, y={1}", p.X, p.Y);
    }

    static void Main()
    {
        IPoint p = new Point(2, 3);
        Console.Write("My Point: ");
        PrintPoint(p);
    }
}

// Output: My Point: x=2, y=3

```


상속과 관련된 키워드

struct 키워드

구조체 형식

Value type(값 형식)이며, 주로 데이터 관련 기능을 캡슐화하는 데에 쓰임 (클래스는 reference type)

C#

```
public struct Coords
{
    public Coords(double x, double y)
    {
        X = x;
        Y = y;
    }

    public double X { get; }
    public double Y { get; }

    public override string ToString() => $"({X}, {Y})";
}
```

상속과 관련된 키워드

Base 키워드

```
public class Person
{
    protected string ssn = "444-55-6666";
    protected string name = "John L. Malgraine";

    public virtual void GetInfo()
    {
        Console.WriteLine("Name: {0}", name);
        Console.WriteLine("SSN: {0}", ssn);
    }
}

class Employee : Person
{
    public string id = "ABC567EFG";
    public override void GetInfo()
    {
        // Calling the base class GetInfo method:
        base.GetInfo();
        Console.WriteLine("Employee ID: {0}", id);
    }
}

class TestClass
{
    static void Main()
    {
        Employee E = new Employee();
        E.GetInfo();
    }
}

/*
Output
Name: John L. Malgraine
SSN: 444-55-6666
Employee ID: ABC567EFG
*/
```

상속과 관련된 키워드

this 키워드
this 키워드는 클래스의 현재 인스턴스를 의미

상속과 관련된 키워드

this 키워드
this 키워드는 클래스의 현재 인스턴스를 의미

클래스의 인스턴스 생성자

C#

```
class Coords
{
    public int x, y;

    // constructor
    public Coords()
    {
        x = 0;
        y = 0;
    }
}
```

클래스의 정적 생성자

C#

```
class SimpleClass
{
    // Static variable that must be initialized at run time.
    static readonly long baseline;

    // Static constructor is called at most one time, before any
    // instance constructor is invoked or member is accessed.
    static SimpleClass()
    {
        baseline = DateTime.Now.Ticks;
    }
}
```

클래스의 정적 생성자

- 정적 생성자는 액세스 한정자를 사용하거나 매개 변수를 갖지 않습니다.
- 클래스 또는 구조체에는 한 개의 정적 생성자만 사용할 수 있습니다.
- 정적 생성자는 상속하거나 오버로드할 수 없습니다.
- 정적 생성자는 직접 호출할 수 없으며, CLR(공용 언어 런타임)을 통해서만 호출할 수 있습니다.
- 자동으로 호출됩니다.
- 정적 생성자는 인스턴스 생성자보다 먼저 실행

이벤트

본질은 함수. 독특한 방식의 함수.

유니티 내장 이벤트 함수 : start() update() 등

Void start () = 오브젝트가 새로 생성 되었을때.

Void update() = 매 프레임마다 한번씩

- 프레임 레이트 의존적

Extern 키워드

extern 한정자는 외부에서 구현되는 메서드를 선언하는 데 사용
abstract 및 extern 한정자를 함께 사용하여 같은 멤버를 수정할 수는 없다.

extern 한정자는 메서드가 C# 코드 외부에서 구현됨을 나타내고
abstract 한정자는 해당 클래스에서 메서드가 구현되지 않음을 나타냅니다.

C#

```
[DllImport("avifil32.dll")]  
private static extern void AVIFileInit();
```

Extern 키워드

extern 한정자는 외부에서 구현되는 메서드를 선언하는 데 사용
abstract 및 extern 한정자를 함께 사용하여 같은 멤버를 수정할 수는 없다.

extern 한정자는 메서드가 C# 코드 외부에서 구현됨을 나타내고
abstract 한정자는 해당 클래스에서 메서드가 구현되지 않음을 나타냅니다.

C#

```
[DllImport("avifil32.dll")]  
private static extern void AVIFileInit();
```

Extern 키워드

extern 한정자는 외부에서 구현되는 메서드를 선언하는 데 사용

abstract 및 `C# Player.cs` × `C# Component.cs` × `C# MonoBehaviour.cs` × `C# Behaviour.cs` × `C# Object.cs` × `C#`

extern 한정
abstract 한

```
/// <para>Returns the component with name type if the GameObject has one attached.  
/// </summary>  
/// <param name="type"></param>  
[FreeFunction(HasExplicitThis = true)]  
[MethodImpl(MethodImplOptions.InternalCall)]  
public extern Component GetComponent(string type);  
  
[TypeInferenceRule(TypeInferenceRules.TypeReferencedByFirstArgument)]  
public Component GetComponentInChildren(System.Type t, bool includeInactive)  
{  
    return this.gameObject.GetComponentInChildren(t, includeInactive);  
}
```

Extern 키워드

extern 한정
abstract 및

extern 한정
abstract 한

```
C# Player.cs × C# Component.cs × C# MonoBehaviour.cs × C# Behaviour.cs × C# Obj
/// <summary>
///   <para>Stops the first coroutine named methodName, or the coroutine
/// </summary>
/// <param name="methodName">Name of coroutine.</param>
/// <param name="routine">Name of the function in code, including corouti
[MethodImpl(MethodImplOptions.InternalCall)]
public extern void StopCoroutine(string methodName);

/// <summary>
///   <para>Stops all coroutines running on this behaviour.</para>
/// </summary>
[MethodImpl(MethodImplOptions.InternalCall)]
public extern void StopAllCoroutines();
```