

# 2022년 2학기 6주차 가상현실

---

성신여대 AI융합학부  
강사 이대영

# 강의 일정

---

주 별 수 업 내 용			
주/회차	수업내용	수업방법	교재진도/과제
1주 1회차	오리엔테이션	대면	
2주 2회차	가상현실의 구분과 역사	대면	
3주 3회차	가상공간과 구현에 대한 기술적 이해	대면	
4주 4회차	실제 가상현실콘텐츠의 활용사례	대면	
5주 5회차	사례분석을 통한 콘텐츠 구현을 위한 디자인 학습	대면	
6주 6회차	가상현실 콘텐츠 기획	대면	
7주 7회차	실제 제작을 위한 게임엔진 소개와 기본이용능력 학습	대면	
8주 8회차	이론 기반 서술형 시험	대면	

# Interface Design

---

## VR콘텐츠 디자인

- 목표에 따라 다른 콘텐츠
- 가상현실에 적합한 인터페이스 디자인 필요

# Human Interaction

---

## 행동 유도성

- 적절한 행동을 하게 만들기
- 가상환경의 오브젝트의 기능 부여 고민

# Human Interaction

---

## 기표

- 오브젝트의 용도, 구조, 작동, 행동 양식을 인지시키기
- 이미지만으로 미리 행동의미를 알려주어야

# Human Interaction

---

## 제약

- 행동에 대한 제약을 주기
- 논리적, 의미적, 문화적 제약
- 사용자의 행동계획에 도움

# Human Interaction

---

## 피드백

- 행동결과 및 상태에 대한 정보
- 상호작용에 대한 이해

# Human Interaction

---

## 매핑

- 사물 간의 상호작용
- 기존 상식적 물체 이용 혹은 사전 설명 필요



# Human Interaction

---

## 매핑

- 사물 간의 상호작용
- 기존 상식적 물체 이용 혹은 사전 설명 필요
- 컴플라이언스(compliance)
  - 감각 피드백을 입력기기와 매칭

# Human Interaction

---

## 매핑

- 공간 컴플라이언스(spatial compliance)
  - 위치(position) - 감각 피드백과 입력 기기의 위치를 동일한 장소에 배치
  - 방향(directional) - 가상 물체가 입력 기기조작과 똑같이 움직이는 것
  - 초기화(nulling) - 기기가 최초 배치로 돌아갈 때 가상현실 오브젝트도 최초 배치로
  - 즉각(temporal) - 행동이나 이벤트에 대한 감각 피드백이 즉시 동기화 되어야
  - 비공간(non-spatial) - 비실재하는 표현 방식, 제스처, 암시 등

# Human Interaction

---

## 직접 상호작용

- 오브젝트에 직접 상호작용
- 직접 조작하는 감각

## 준간접 상호작용

- 상호작용을 의미하는 중재도구, 오브젝트를 통한 상호작용

## 간접 상호작용

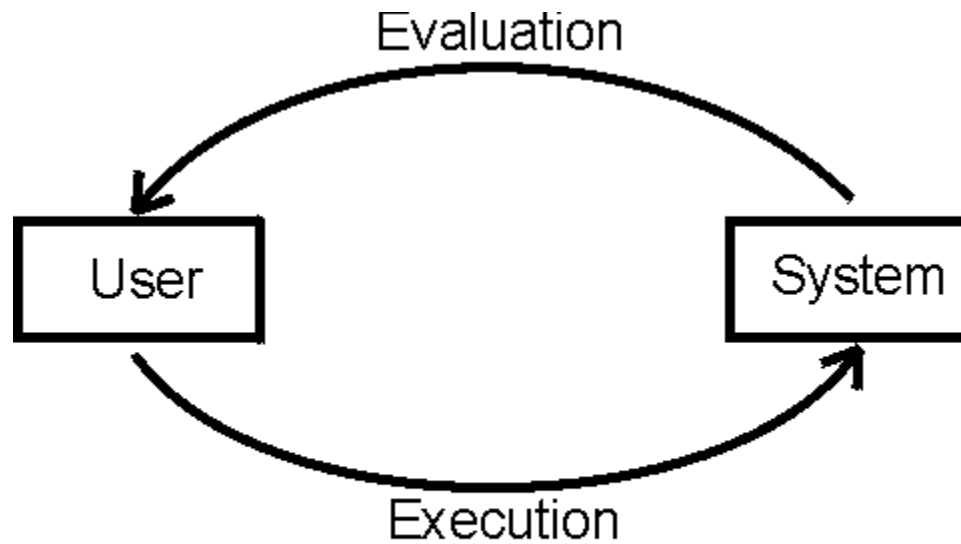
- 키보드 입력, 음성인식 등 입출력 사이의 인지전환이 요구되는 상호작용

# Human Interaction

---

## 상호작용 사이클

- 목표 형성
- 행동 수행
- 결과 평가

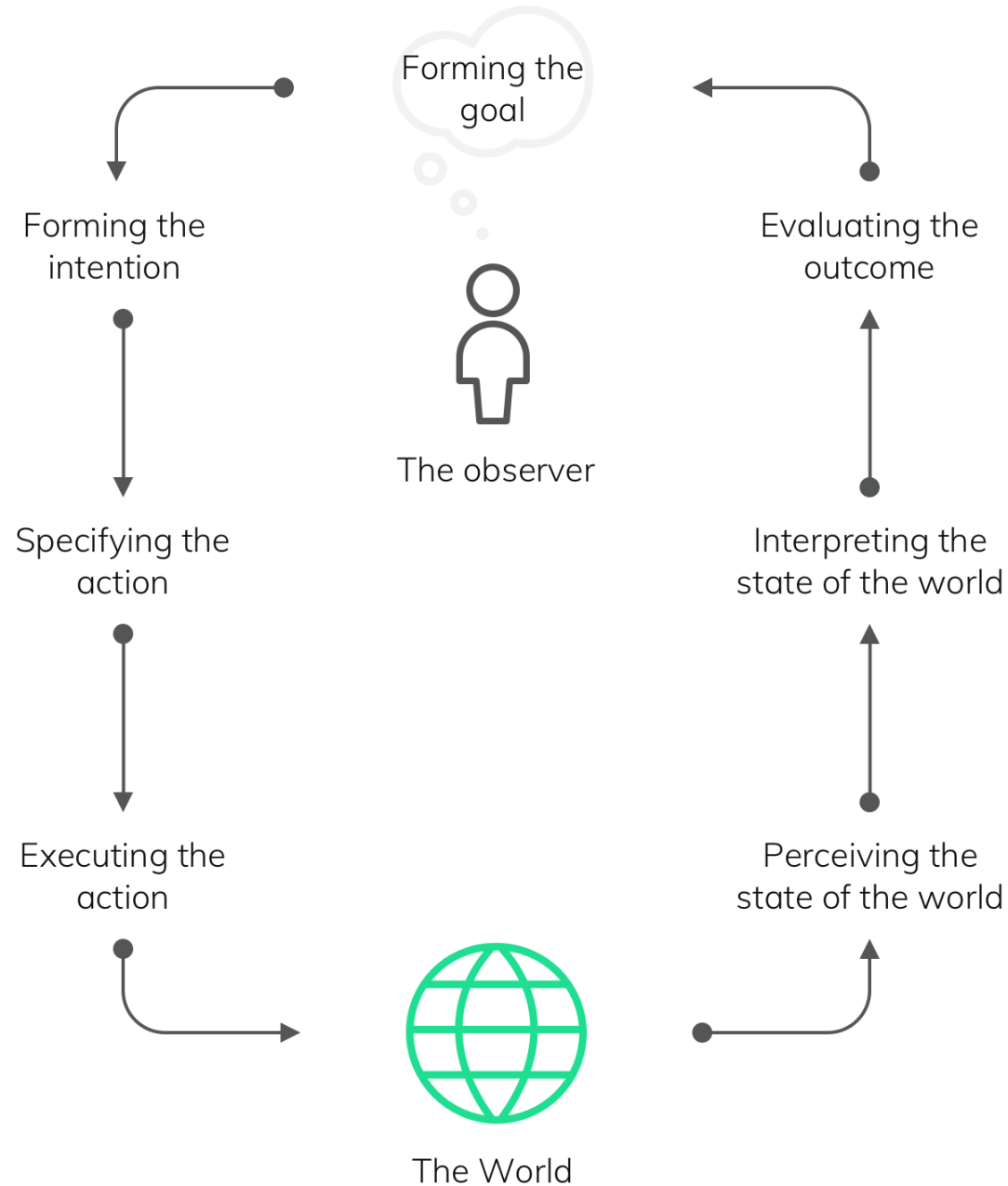


Norman's Interaction Cycle

# Human Action Cycle

상호작용 사이클

- 목표 형성
- 행동 수행
- 결과 평가



# Human Interaction

---

## 상호작용 충실도

- 생체 역학적 대칭
  - 가상 상호작용을 위한 신체의 움직임이 현실적 이슈에 대한 몸의 움직임과 닮은 정도
- 입력 정확도
  - 입력기기가 사용자의 행동을 포착하고 측정하는 정도
  - 정확도, 정밀도, 지연시간
- 컨트롤 대칭
  - 상호작용에 대응하는 현실 세계의 과제와 비교해 사용자가 상호작용을 통제할 수 있는 정도

# Human Interaction

---

## 고유감각

- 이용자, 즉 사람의 몸의 포즈 및 동작을 감지
- 시각정보 이외의 정보 활용가능

# Human Interaction

---

## 참고프레임

- 물체의 위치와 방향에 대한 기준좌표
  - 가상 세계 내에서의 프레임
  - 현실 세계 내에서의 프레임
- 서로 동기화 되지 않지만 콘텐츠와 개인인지, 멀미, 제약 등에 대해 고려해야



# Human Interaction

---

## 참고프레임

- 상체 참고 프레임
  - 몸 척추 축에서 전방
  - 머리, 팔, 손의 위치를 아는 고유감각
  - 일반적으로 머리를 중심으로 디자인
- 손 참고 프레임 - 컨트롤러, 손을 사용한 인터페이스 컨트롤
- 머리 참고 프레임 - 시선이 아닌 두 눈 사이의 점이 기준, HMD의 위치정보
- 눈 참고 프레임 - 시선의 위치, 시선의 방향 고려 필요

# Human Interaction

---

## 제스처

- 공간 정보 - 방향, 설명 등
- 상징적 정보 - 상징적 기호 전달
- 경로적 정보 - 정보 사이의 생각 표현
- 정서적 정보 - 감정의 표현

I



Gesture 1



Gesture 2



Gesture 3



Gesture 4



Gesture 5



Gesture 6



Gesture 7



Gesture 8

天

(

(

(

(



Gesture 9



Gesture 10



Gesture 11



Gesture 12



Gesture 13



Gesture 14



Gesture 15



Gesture 16



Gesture 17



Gesture 18



Writing



Handshake



Eye



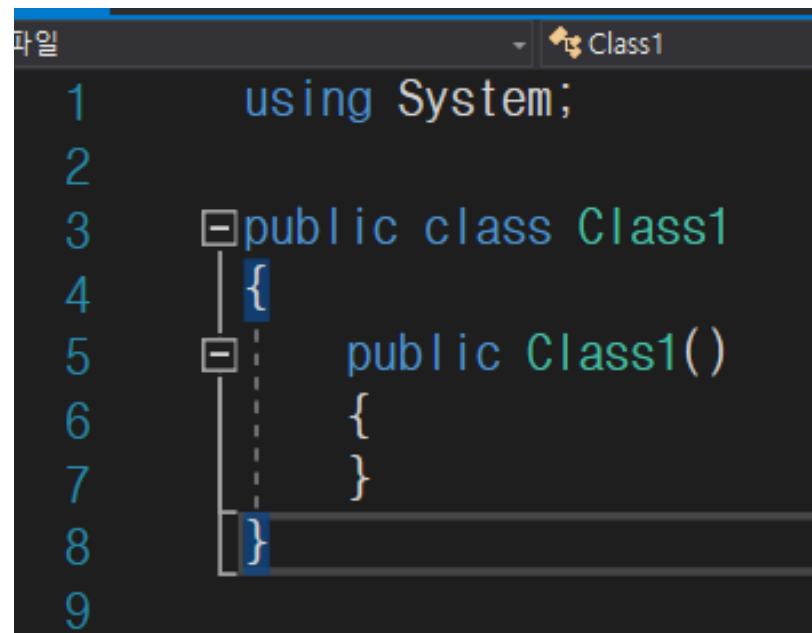
<https://youtu.be/KBSokruYsuM>

# Unity Coding

---

## 문장

- 문장의 나열을 통해 만들어지는 명령어를 쓰기 쉽게 구조화할 필요가 있음
- 클래스는 여러 개의 객체를 만들 수 있음
- C#의 경우 문장완료기호는 ‘;’를 사용



A screenshot of a C# code editor window. The title bar shows '파일' (File) and 'Class1'. The code is as follows:

```
1      using System;
2
3      public class Class1
4      {
5          public Class1()
6          {
7          }
8      }
```

The code is displayed on a dark background with syntax highlighting. Line numbers 1 through 9 are visible on the left. The class definition is enclosed in curly braces, and the constructor is also enclosed in curly braces. The code ends with a semicolon on line 8.

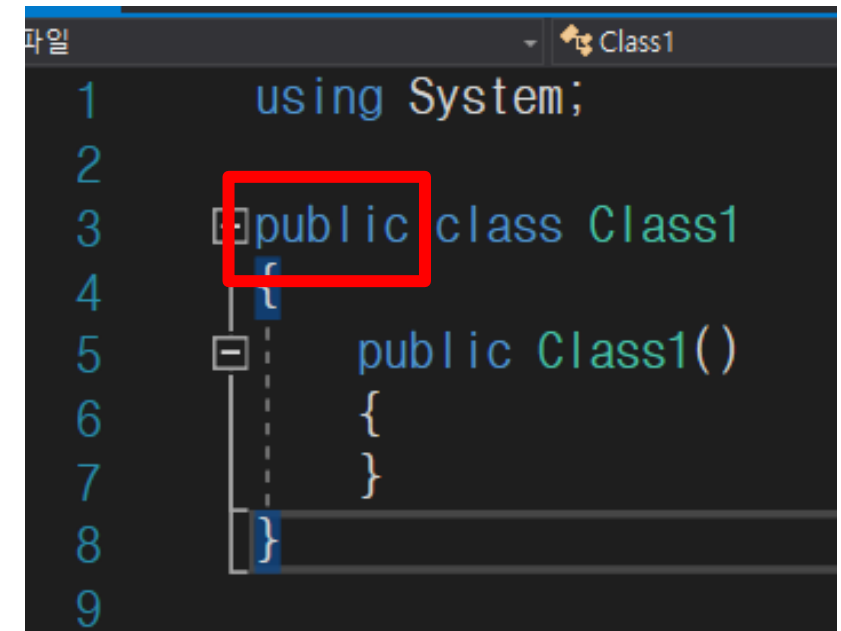
# Unity Coding

---

*public class Class1*

public/private:접근지정자(access modifier)

- class 외부에서 접근 가능 여부 결정
- public은 외부접근 허용, private는 class 내부에서만 허용
- 그 외 Internal, protected internal, protect가 있음
- 생략하면 private에 준함(enum제외)
- public의 반대개념, private은 class 내부에서만 사용 가능



```
파일 Class1
1 using System;
2
3 public class Class1
4 {
5     public Class1()
6     {
7     }
8 }
9
```

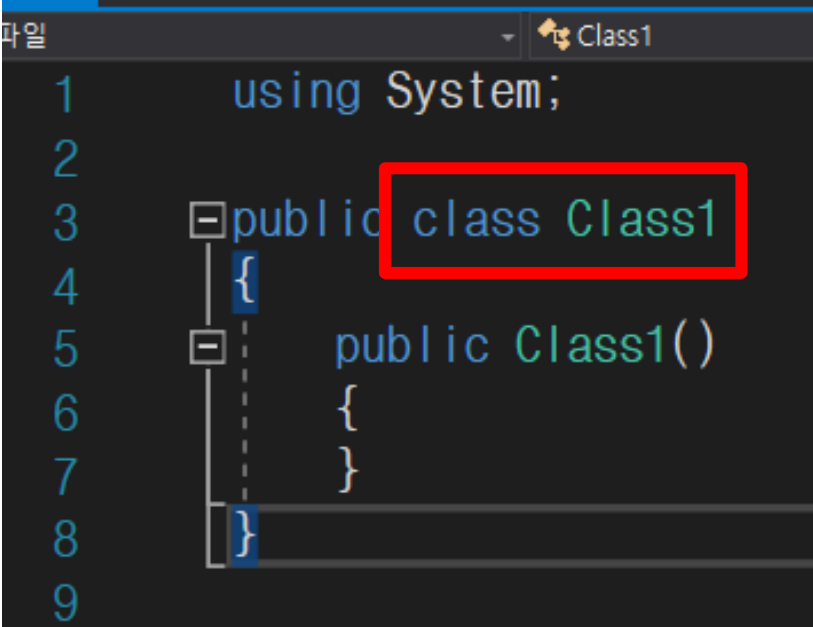
# Unity Coding

---

*public class Class1*

Class:사용자가 정의하게 되는 새로운 형식

- 클래스(Class)는 객체 지향 프로그래밍에서 특정한 객체를 생성하기 위해서 필드, 속성, 메소드 및 이벤트 등을 정의하는 틀
- Public class는 하나만 존재, 다른 class는 여러 개 가능
- Class 이름과 파일 이름(~.cs)은 동일해야 오류가 나지 않음
- 그룹화하여 자체 사용자 지정 형식을 만들 수 있는 구문
- 클래스는 참조 형식

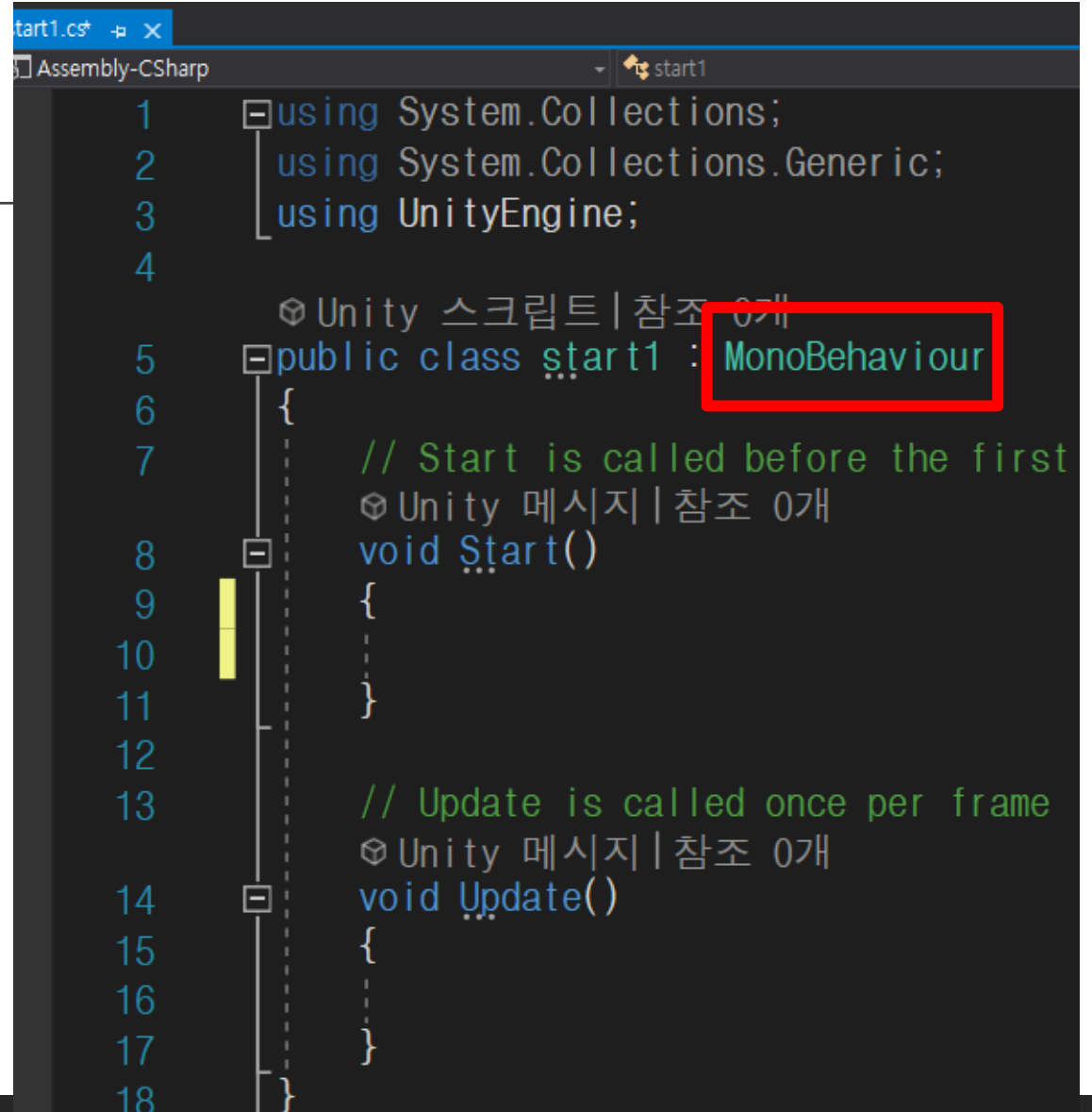


```
파일 Class1
1 using System;
2
3 public class Class1
4 {
5     public Class1()
6     {
7     }
8 }
9
```

# Unity Coding

## MonoBehaviour

- Unity 의 오브젝트에 사용되는 스크립트가 상속받는 클래스
- 미리 만들어 둔 명령어
- Unity 내에서 gameobject의 component 형태로 사용 가능



```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class start1 : MonoBehaviour
6  {
7      // Start is called before the first
8      void Start()
9      {
10         ...
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16         ...
17     }
18 }
```

The screenshot shows a C# script named 'start1' in a Unity environment. The script inherits from 'MonoBehaviour', which is highlighted with a red box. The script contains two methods: 'Start()' and 'Update()'. The 'Start()' method is commented as 'Start is called before the first' and the 'Update()' method is commented as 'Update is called once per frame'. The script is written in C# and uses the UnityEngine namespace.



# Unity Coding

---

```
int a = 1;
```

Keyword / identifier/ literal

예약어:keyword

- 미리 약속한 단어
- int(정수), float(실수), string(문자열), bool(참/거짓), object(사물) 등

# Unity Coding

---

```
int a = 1;
```

Keyword / identifier/ literal

정수(int)

- `int i = 10;`

실수(float)

- `float f = 3.14159f;`

문자열(string)

- `string str = "hello";`

논리값(bool)

- `bool isMoveable = true;`

# Unity Coding

---

종류	연산자	사용법	의미
이항 연산자	<code>+, -, *, /, %</code>	<code>a = b + c</code>	a에 b, c에 대한 계산값을 대입
복합 할당 연산자	<code>+=, -=, *=, /=, %=</code>	<code>a += b</code>	a에 b에 대한 계산을 실시하여 계산값을 a에 대입
논리 연산자	<code>&gt;, &lt;, &gt;=, &lt;=, ==, !=</code>	<code>a &gt; b</code>	a와 b의 비교
이항 논리 연산자	<code>&amp;&amp;,   , !</code>	<code>a &amp;&amp; b</code>	a와 b 포함관계
비트 이항 연산자	<code>&amp;,  , ^, &lt;&lt;, &gt;&gt;</code>	<code>a &gt;&gt; b</code>	비트단위 논리 연산
증감 연산자	<code>++, --</code>	<code>++a, a++</code>	Aa의 값에 1을 증가시키거나 감소시킴
삼항 연산자	<code>? :</code>	<code>(a &gt; b) ? a : b</code>	?앞의 내용에 대해 참이면 앞 : 거짓이면 : 뒤

# Unity Coding

---

```
int a = 1;
```

Keyword / identifier/ literal

예약어:keyword

- 미리 약속한 단어
- int(정수), float(실수), string(문자열), bool(참/거짓), object(사물) 등

식별자:identifier

- 변수의 이름

# Unity Coding

---

```
int a = 1;
```

Keyword / identifier/ literal

예약어:keyword

- 미리 약속한 단어
- int(정수), float(실수), string(문자열), bool(참/거짓), object(사물) 등

식별자:identifier

- 변수의 이름

리터럴:literal

- 변수의 고정된 값

# Unity Coding

---

```
int a ← 1;
```

Keyword / identifier/ literal

데이터를 편하게 사용하기 위해, 데이터를 하나의 변수variable를 만들어서 동일시  
=의 의미는 좌의 identifier에 우측 literal 값을 대입 시키는 것

\* 일반 수학에서의 '=' (동일)는 '=='으로 사용

# Unity Coding

---

```
int a = 1;
```

Keyword / identifier/ literal

C#의 경우 문장완료기호는 ‘;’를 사용

# Unity Coding

---

## void

- ‘비어’있는 항
- void 뒤의 문장을 항의 이름으로 정의
- 중괄호{ } 내부에 입력한 내용을 함수화
- 반환값(return)이 없는 함수
  - \* ( )안에 입력값을 넣을 수 있으며 이때는 반환값이 존재

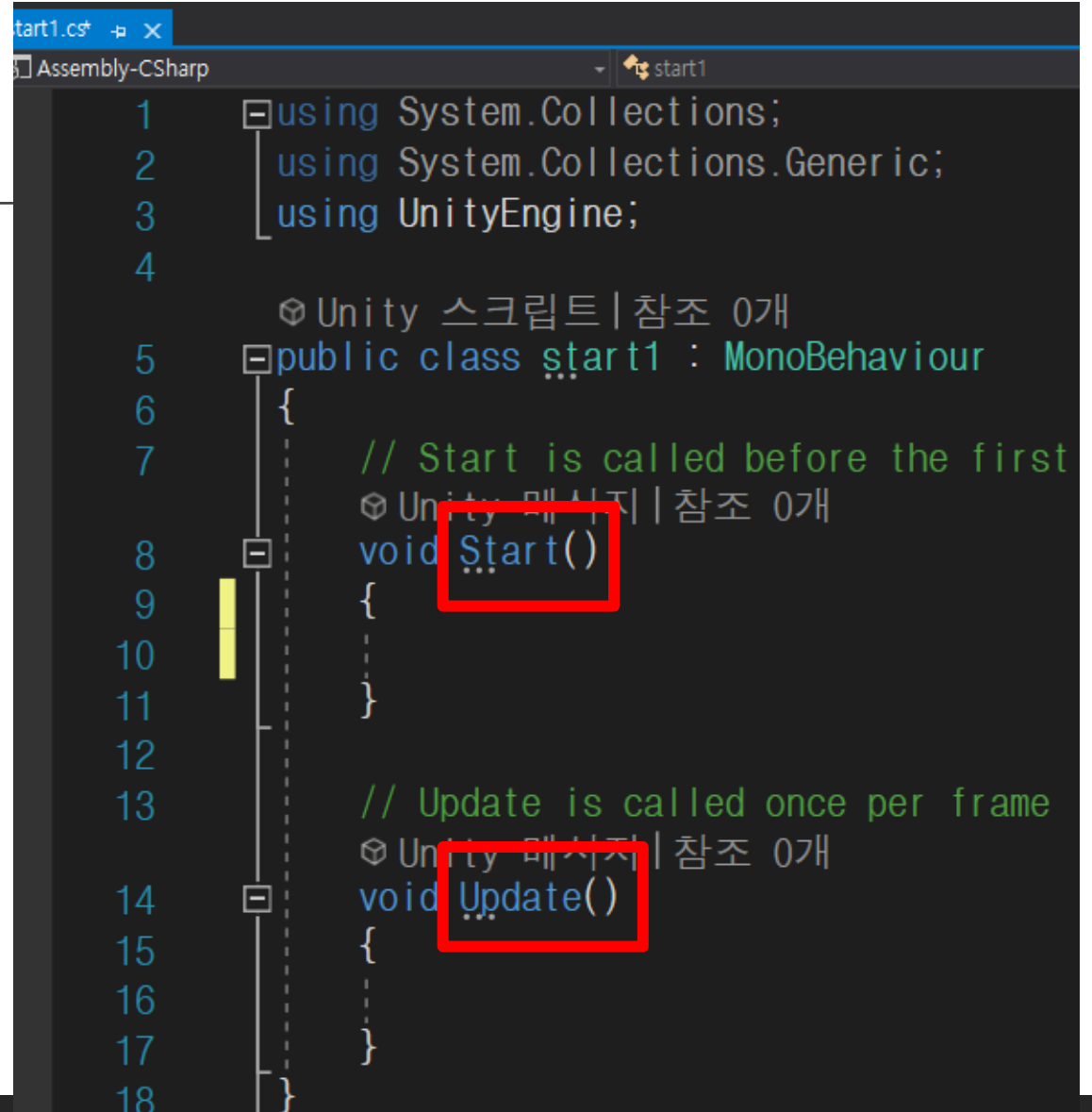
```
void Test()  
{  
-  
}
```



# Unity Coding

## 라이프사이클

- *Start* : Update 함수가 호출되기 전에 한번만 호출
- *Update* : 매 프레임마다 호출되는 가장 일반적인 Update 함수
- 그 외 Reset, Awake, OnEnable, FixedUpdate, LastUpdate, OnDisable 등

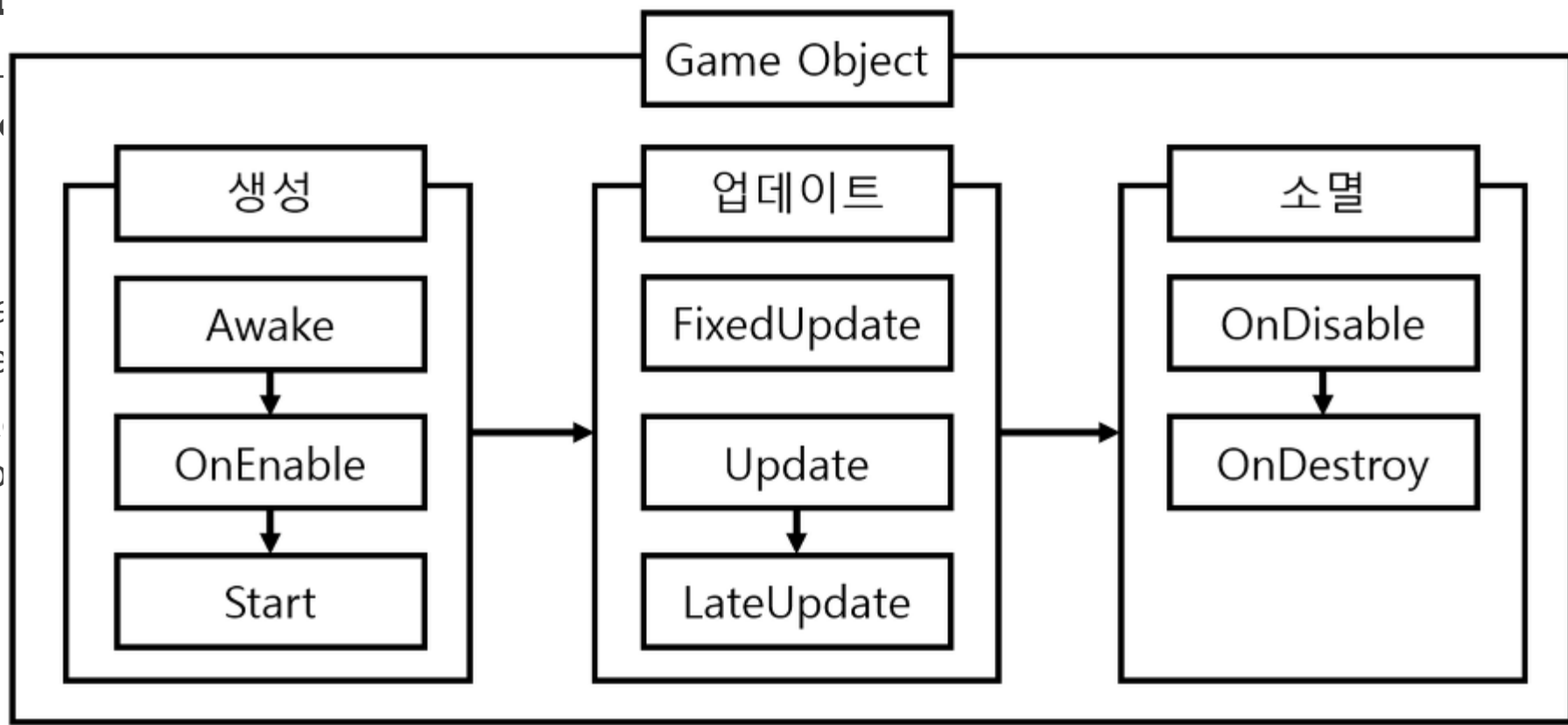


```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  Unity 스크립트 | 참조 0개
6  public class start1 : MonoBehaviour
7  {
8      // Start is called before the first
9      // Unity 메시지 | 참조 0개
10     void Start()
11     {
12     }
13
14     // Update is called once per frame
15     // Unity 메시지 | 참조 0개
16     void Update()
17     {
18     }
19 }
```

# Unity Coding

라이프사

- *Start* : 호출
- *Update* : Update
- 그 외 *Render* : LastUpdate



```
start1.cs x
Assembly-CSharp
start1
...
}
}
}
```

behaviour  
re the first  
e per frame

# Unity Coding

## { } 중괄호

- 함수(method, 코드들의 묶음)의 몸체
- 명령문의 그룹화

## [ ] 대괄호

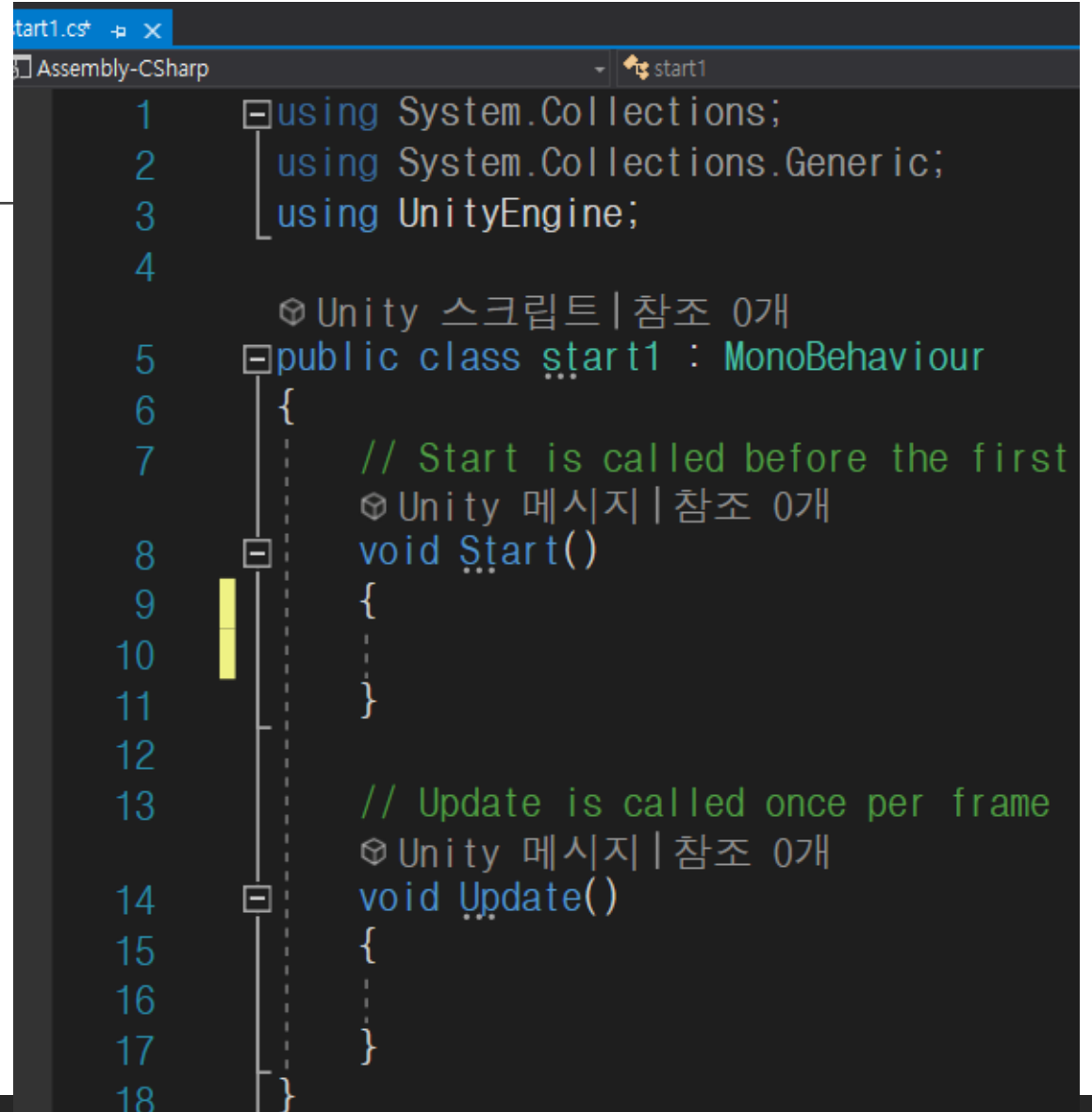
- 배열, 인덱서, 특성(메타데이터)

## ( ) 괄호

- 캐스트, 형변환 conversion

## < > 꺾은 괄호

- 특정한 유형의 인수를 불러옴



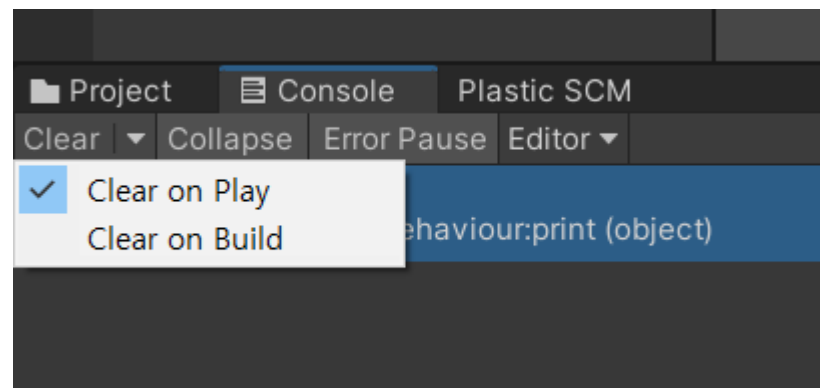
```
1  using System.Collections;
2      using System.Collections.Generic;
3      using UnityEngine;
4
5      Unity 스크립트 | 참조 0개
6  public class start1 : MonoBehaviour
7  {
8      // Start is called before the first
9      Unity 메시지 | 참조 0개
10     void Start()
11     {
12         ...
13     }
14
15     // Update is called once per frame
16     Unity 메시지 | 참조 0개
17     void Update()
18     {
19         ...
20     }
21 }
```

# Unity Coding

---

## 콘솔

- Clear on Play
  - 매번 테스트 할때마다 지우고 다시 작성
- Clear on Build
  - 매번 게임빌드 후에 지우고 다시 작성

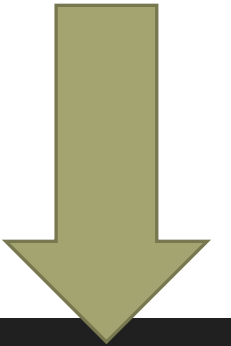


# Unity Coding

---

## 콘솔

- 디버그 메시지 보고 싶을 때 체크



# Unity Coding

---

void Start

- 시작할 때 한 번 실행

print

- 콘솔창에 메시지 출력

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Move : MonoBehaviour
6  {
7      int a = 1;
8      int b = 2;
9      int c;
10     // Start is called before the first frame update
11     void Start()
12     {
13         c = a + b;
14         print(c);
15     }
16 }
```

# Unity Coding

---

## 조건문

```
int abc = 1;
void Update()
{
    if(abc==1)//괄호 사이에 일치해야 할 조건 입력
    {
        //중괄호 사이에 조건이 맞는 경우 실행할 내용 입력;
    }
    else if ()//그 외의 다른 조건을 입력합니다
    {
    }
    else
    {
        //그 외의 아무 상황일때 실행
    }
}
```

# Unity Coding

---

## 키 입력

- <https://docs.unity3d.com/kr/2021.3/ScriptReference/KeyCode.html>



# KeyCode

enumeration

## Uni Description

Event.keyCode에 의해 반환되는 키 코드(KeyCode)들을 나타냅니다. 키보드의 실제키와 일치합니다.

### 키 입력

Key codes can be used to detect key down and key up events, using [Input.GetKeyDown](#) and [Input.GetKeyUp](#) :

```
using UnityEngine;
```

```
public class KeyCodeExample : MonoBehaviour
```

```
{
```

```
    void Update( )
```

```
    {
```

```
        if( Input.GetKeyDown( KeyCode.Space ) )
```

```
            Debug.Log( "Space key was pressed." );
```

```
        if( Input.GetKeyUp( KeyCode.Space ) )
```

```
            Debug.Log( "Space key was released." );
```

```
    }
```

```
}
```

키입력:누르는 순간 1회

# KeyCode

enumeration

## Uni Description

Event.keyCode에 의해 반환되는 키 코드(KeyCode)들을 나타냅니다. 키보드의 실제키와 일치합니다.

### 키 입력

Key codes can be used to detect key down and key up events, using [Input.GetKeyDown](#) and [Input.GetKeyUp](#) :

```
using UnityEngine;

public class KeyCodeExample : MonoBehaviour
{
    void Update( )
    {
        if( Input.GetKeyDown( KeyCode.Space ) )
            Debug.Log( "Space key was pressed." );

        if( Input.GetKeyUp( KeyCode.Space ) )
            Debug.Log( "Space key was released." );
    }
}
```

키입력:떼는 순간 1회

# KeyCode

enumeration

## Uni Description

Event.keyCode에 의해 반환되는 키 코드(KeyCode)들을 나타냅니다. 키보드의 실제키와 일치합니다.

### 키 입력

Key codes can be used to detect key down and key up events, using [Input.GetKeyDown](#) and [Input.GetKeyUp](#) :

```
using UnityEngine;

public class KeyCodeExample : MonoBehaviour
{
    void Update( )
    {
        if( Input.GetKeyDown( KeyCode.Space ) )
            Debug.Log( "Space key was pressed." );

        if( Input.GetKeyUp( KeyCode.Space ) )
            Debug.Log( "Space key was released." );
    }
}
```

키코드 : 키보드 등의 키  
이름

해당 링크에서 키 확인

# Unit 버튼 이름

키를 축으로 매핑할 때, 키 이름을 Inspector의 Positive Button 또는 Negative Button 속성에 입력해야 합니다.

## 키 입력

### 키(Keys)

키의 이름은 다음 규칙을 따릅니다:

- 일반 키 : "a", "b", "c"...
- 숫자 키 "1", "2", "3", ...
- 화살표 키 : "up", "down", "left", "right"
- 키패드 키 : "[1]", "[2]", "[3]", "[+]", "[equals]"
- 수정 키 : "right shift", "left shift", "right ctrl", "left ctrl", "right alt", "left alt", "right cmd", "left cmd"
- 마우스 버튼 : "mouse 0", "mouse 1", "mouse 2"...
- (임의의) 조이스틱 버튼 : "joystick button 0", "joystick button 1", "joystick button 2"...
- (특정) 조이스틱 버튼 : "joystick 1 button 0", "joystick 1 button 1", "joystick 2 button 0"...
- 특수 키 : "backspace", "tab", "return", "escape", "space", "delete", "enter", "insert", "home", "end", "page up", "page down"
- Function 키 : "f1", "f2", "f3"...

Input 키

키 입력을 연속으로 받음

스크립팅 인터페이스와 인스펙터 에서 키를 식별하는 데 사용되는 이름은 같습니다.

```
value = Input.GetKey ("a");
```

<https://docs.unity3d.com/kr/530/Manual/ConventionalGameInput.html>

# Unit

## 위치

# Transform

class in UnityEngine / Inherits from: [Component](#)

## Description

게임오브젝트의 위치, 회전 그리고 스케일(scale)을 나타냅니다.

씬의 모든 오브젝트는 트랜스폼을 가집니다. 오브젝트의 위치, 회전 그리고 스케일을 저장하고 다루기 위해서 사용합니다. 모든 트랜스폼은, 계층적으로 위치, 회전, 스케일을 적용할 수 있는, 부모 트랜스폼을 가질 수 있습니다. 이러한 계층은 계층 뷰에서 확인할 수 있습니다. 트랜스폼은 또한 enumerator를 지원하기 때문에 자식 트랜스폼에 대해 다음과 같이 루프를 이용할 수 있습니다:

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour
{
    void Example() {
        foreach (Transform child in transform) {
            child.position += Vector3.up * 10.0F;
        }
    }
}
```

오브젝트의 위치/회전/크기

# Unity C

## Transform.position

public Vector3 position;

위치

### Description

월드 공간에서 트랜스폼의 위치를 나타냅니다.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Example() {
        transform.position = new Vector3(0, 0, 0);
        print(transform.position.x);
    }
}
```

오브젝트의 위치

# Unity C

## Transform.position

public Vector3 position;

위치

### Description

월드 공간에서 트랜스폼의 위치를 나타냅니다.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour
{
    void Example() {
        transform.position = new Vector3(0, 0, 0);
        print(transform.position.x);
    }
}
```

오브젝트의 포지션 값  
Vector3(x, y, z)

# Unity Coding

## 벡터

- 2차원의 경우 z축을 사용하지 않음

Vector3.zero

## Vector3

struct in UnityEngine

### Description

3차원 벡터와 위치를 표현합니다.

이 구조체는 3차원 공간에서의 위치와 벡터를 표현하기 위해 사용됩니다. 또한 공통 벡터 연

함수들은 아래에서 확인할 수 있고, 벡터와 위치를 다루는 데, 다른 클래스가 사용될 수 있습  
키는 데 유용합니다.

### Static Variables

<u>back</u>	Vector3(0, 0, -1)를 사용하는 간단한 방법입니다.
<u>down</u>	Vector3(0, -1, 0)를 사용하는 간단한 방법입니다.
<u>forward</u>	Vector3(0, 0, 1)를 사용하는 간단한 방법입니다.
<u>left</u>	Vector3(-1, 0, 0)를 사용하는 간단한 방법입니다.
<u>one</u>	Vector3(1, 1, 1)를 사용하는 간단한 방법입니다.
<u>right</u>	Vector3(1, 0, 0)를 사용하는 간단한 방법입니다.
<u>up</u>	Vector3(0, 1, 0)을 사용하는 간단한 방법입니다.
<u>zero</u>	Vector3(0, 0, 0)을 사용하는 빠른 방법입니다.



# Unity Coding

---

## 벡터

- 한 축씩 사용 가능

## Variables

<u>magnitude</u>	벡터의 길이를 반환합니다. (읽기전용)
<u>normalized</u>	해당 벡터의 magnitude 가 1인 벡터를 반환합니다.(읽기전용)
<u>sqrMagnitude</u>	벡터의 길이의 제곱한 값을 반환합니다. (읽기전용)
<u>this[int]</u>	[0], [1], [2]를 각각 사용하여 x, y, z 요소에 접근합니다.
<u>x</u>	벡터의 X컴포넌트를 나타냅니다.
<u>y</u>	벡터의 Y컴포넌트를 나타냅니다.
<u>z</u>	벡터의 Z컴포넌트를 나타냅니다.

Vector3.x

# Unity Coding

## 벡터

- 벡터 값끼리 연산 가능

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Example() {
        print(new Vector3(1, 2, 3) - new Vector3(6, 5, 4));
    }
}
```

## Operators

operator -

한 벡터를 다른 벡터로부터 뺍니다.

operator !=

벡터가 다를 경우 true를 반환합니다.

operator \*

벡터에 숫자를 곱합니다.

operator /

벡터를 숫자로 나눕니다.

operator +

두 벡터를 더합니다.

operator ==

벡터가 같으면 true를 반환합니다.