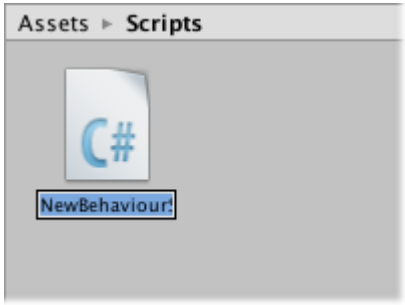
Unity3D Basic

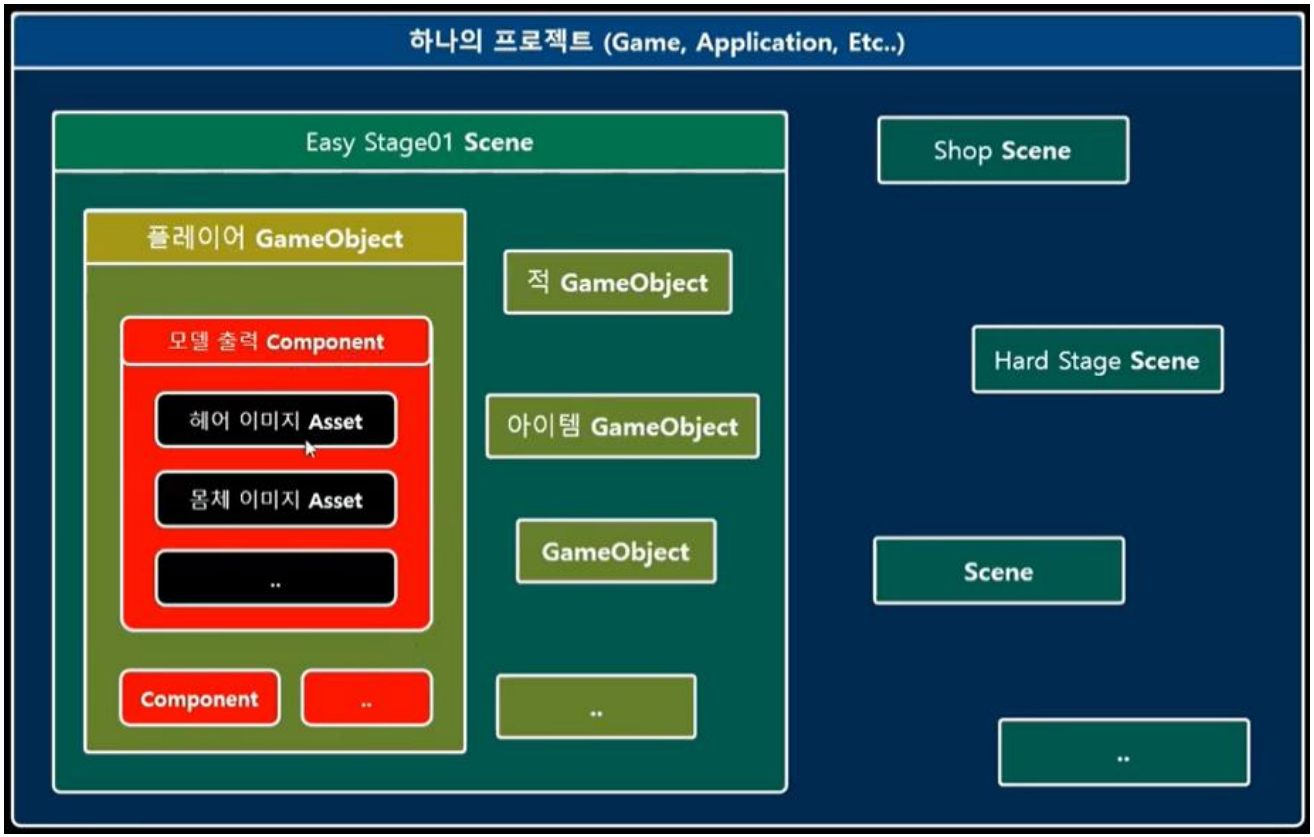
Unity Scripting

목차

1. Scene
2. Unity Scripting
   - Script
   - Basic Function



- 프로젝트 – 씬 – 게임 오브젝트 - 컴포넌트



- **프로젝트**(Project) : 하나의 게임, 콘텐츠, 애플리케이션을 의미
- **씬**(Scene) : 게임의 장면이나 상태를 저장하는 단위

  하나의 큰 게임은 씬 단위로 관리되며, (코드를 통해) 씬 간의 이동

  Intro 씬 → Menu 씬 → Stage 1 … N → GameOver 씬 → Ending 씬
- **게임 오브젝트**(GameObject) : 씬에 배치되는 하나의 물체를 지칭

  씬을 구성하는 최소 단위

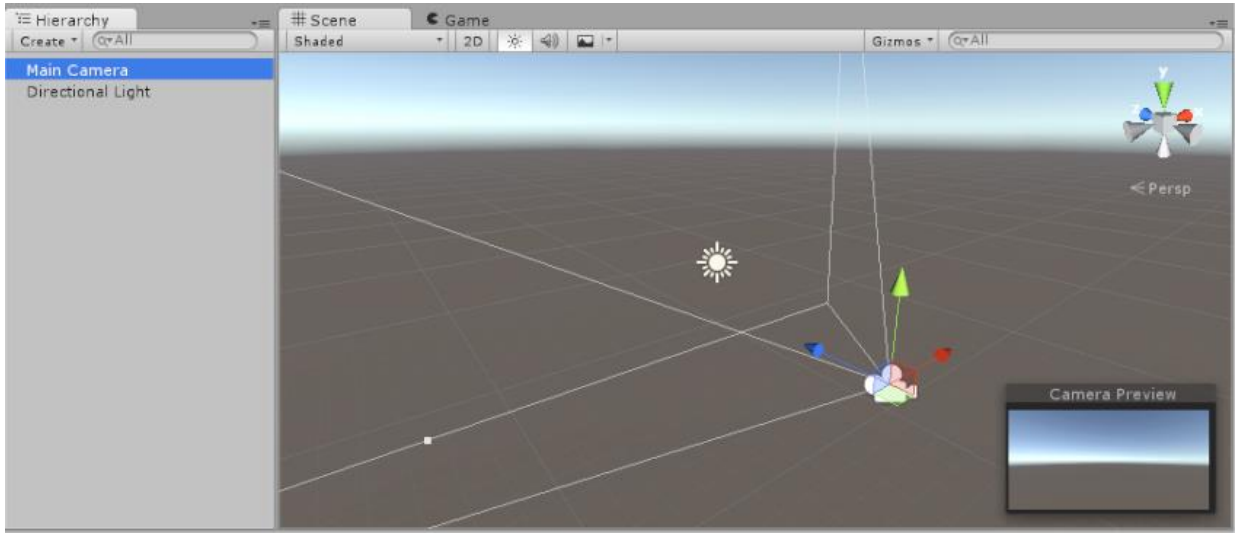
  (예) 화면에 나타나는 캐릭터, 타켓, 버튼, 환경 등

  게임오브젝트는 컴포넌트를 묶어서 관리, 접근할 수 있음
- **컴포넌트**(Component) : 독립적으로 기능을 수행

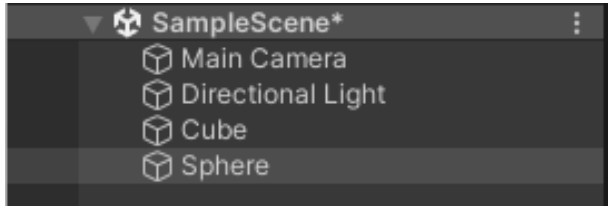
  (예) 화면에 그림을 그리는 것, 소리를 내는 것 등의 기능들을 컴포넌트라고 부름

  모든 게임오브젝트는 Position/Rotation/Scale를 제어하는 Transform 컴포넌트를 가짐

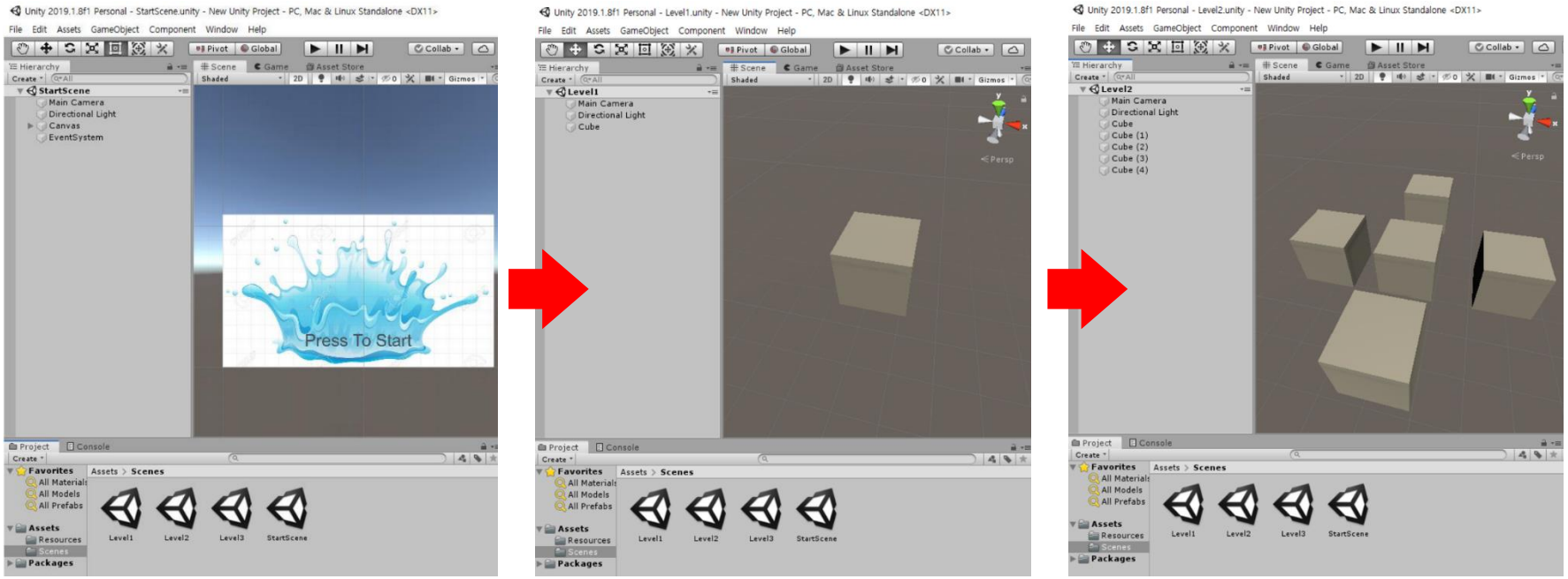
- 씬(Scene)
  - Unity에서 콘텐츠를 사용하여 작업하는 공간 (= 하나의 레벨/스테이지)
  - 게임, 애플리케이션의 전체 또는 일부를 포함하는 애셋 (파일 확장자는 .unity)



  - 씬 생성/저장 : 메뉴 [File – New Scene], [File – Save]를 클릭

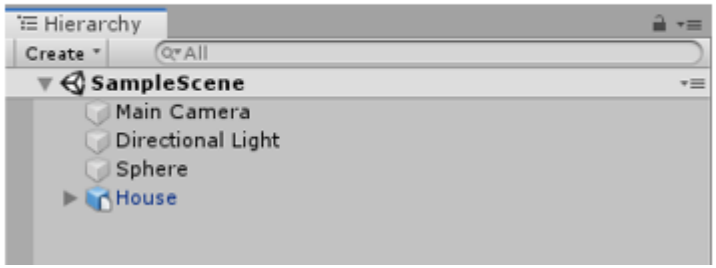


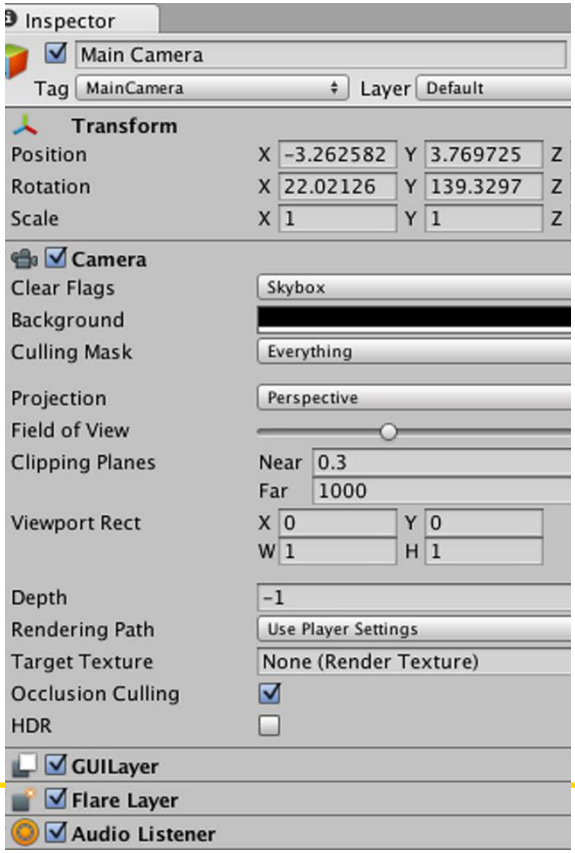
예) StartScene → Level1 → Level2 ..



- **게임 오브젝트** (Game object)
  - 하이러리키 창(Hierarchy window)에 있는 모든 것들이 게임 오브젝트
  - 캐릭터, 아이템부터 광원, 카메라, 특수효과에 이르기까지 게임에 존재하는 모든 오브젝트가 게임 오브젝트 – 속성(Property)를 가져야 게임 오브젝트 역할을 할 수 있음

Hierarchy 창



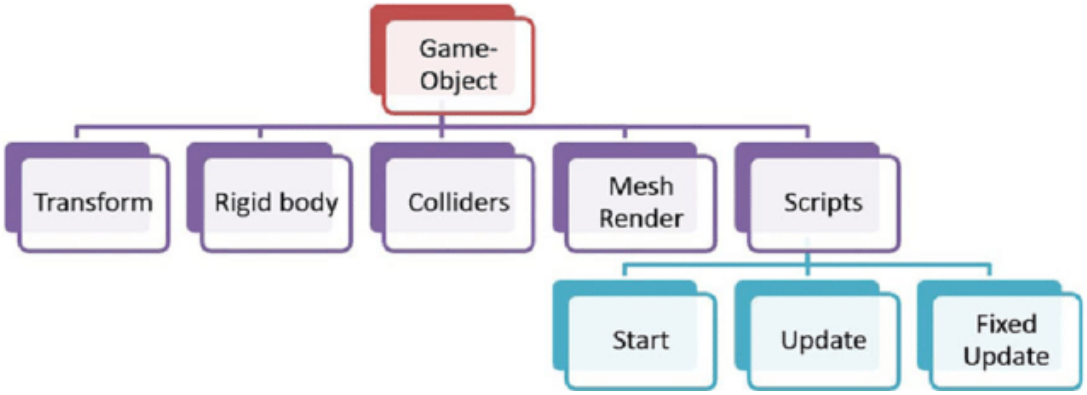


- 게임 오브젝트는 기능을 담고 있는 박스
  (예) 카메라는 빈 게임 오브젝트에 카메라 기능을 가지고 있는 박스

Q) 게임 오브젝트를 비활성화 하려면 ?

- **컴포넌트** (Component)
  - 게임 오브젝트의 구성요소

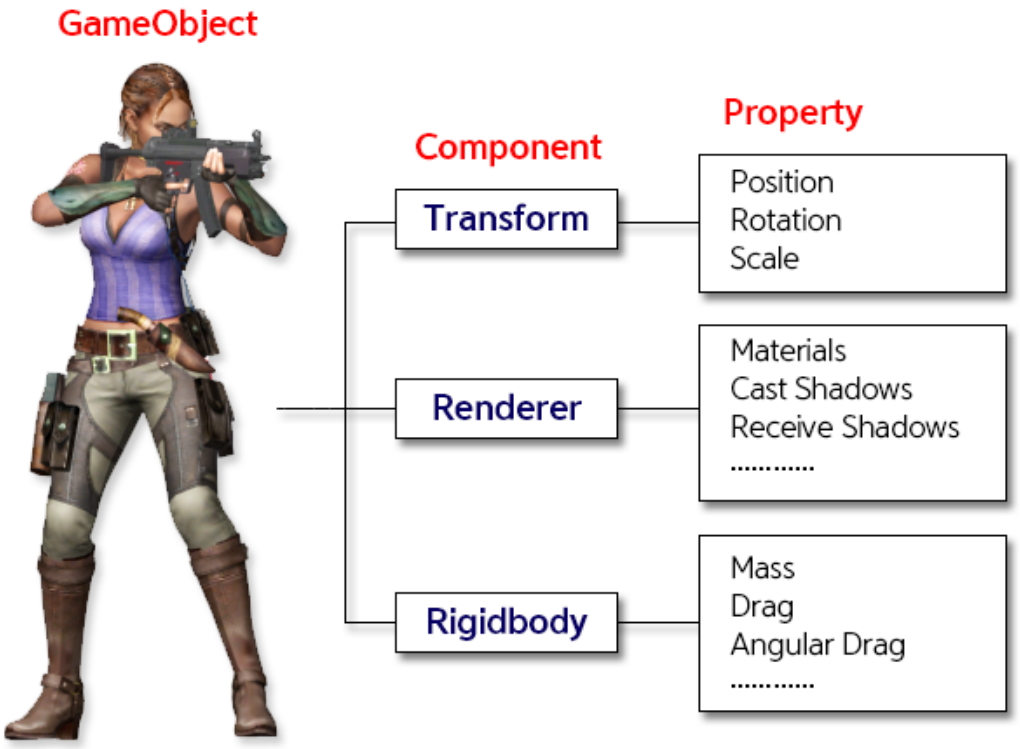
    게임 오브젝트는 여러 기능을 담고 있는 박스이며, 여기서의 각각의 기능이 컴포넌트가 됨



  - (예) 게임 오브젝트가 충돌할 수 있게 하는 콜라이더(Collider), 게임을 실행할 때에 볼 창을 위한 카메라(Camera), 사운드가 들리는 곳을 설정하기 위한 오디오 리스너(Audio Listener), 위치/회전/크기 등을 위한 트랜스폼(Transform), 물리적인 상호작용을 위한 리지드바디(Rigidbody), 사용자 정의 기능인 스크립트(Script) 등

속성(Property, 프로퍼티) : 오브젝트의 위치, 방향, 질량, 색상 등의 개별적인 값

컴포넌트(Component) : 서로 관련된 속성을 묶음



- **스크립트** (Script)
  - 사용자 정의 형태의 컴포넌트에 대한 설계 도면

    게임을 만들다 보면, 유니티에서 제공하는 기본 컴포넌트만으로는 모든 기능을 처리할 수 없음 → Script 를 사용하여 기능을 구현

▶ Start() 메소드

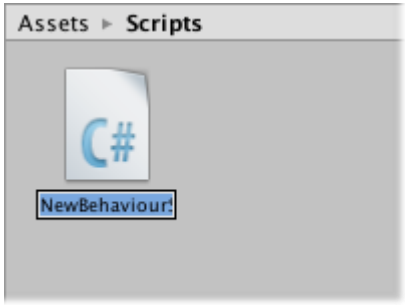
- 유니티가 자동으로 호출해주는 메서드 중 하나
- 게임이 시작할 때 딱 한 번만 호출
- Update() 메서드가 호출되기 직전에 호출
- 주로, 초기화와 같은 작업을 할 때 많이 사용

▶ Update() 메소드

- 유니티가 자동으로 호출해주는 메서드 중 하나
- 매 프레임마다 호출
- Frame이란?
  - N FPS = 1초마다 N개의 Frame이 수행된다는 의미
  - (예) 60 FPS이면 1초에 60번 Update 메서드가 호출

```
1 using UnityEngine;
2 using System.Collections;
3 public class NewScript : MonoBehaviour {
4     // Use this for initialization
5     void Start () {
6
7     }
8     // Update is called once per frame
9     void Update () {
10
11    }
12 }
13
```

1. Scene
2. Unity Scripting
   - Script
   - Basic Function



Unity Scripting

- Scripting in Unity is the programming side of game development
- Unity primarily uses the C# language
- C# is very similar to Java, and is ideal for game development because it is very object-oriented

  After all, everything we want to interact with is a GameObject

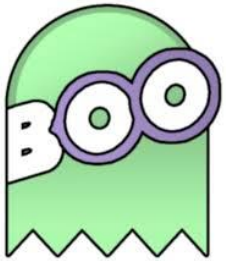
  Much easier to write code, if we can think in terms of objects
- Unity Scripting is primarily interacting with GameObject components
- Scripts are really just custom components

  When you create a script, you are creating your own component. You an give the component behavior, properties, fields, and values
- You add scripts to GameObject just like any other component

Supported Languages in Unity

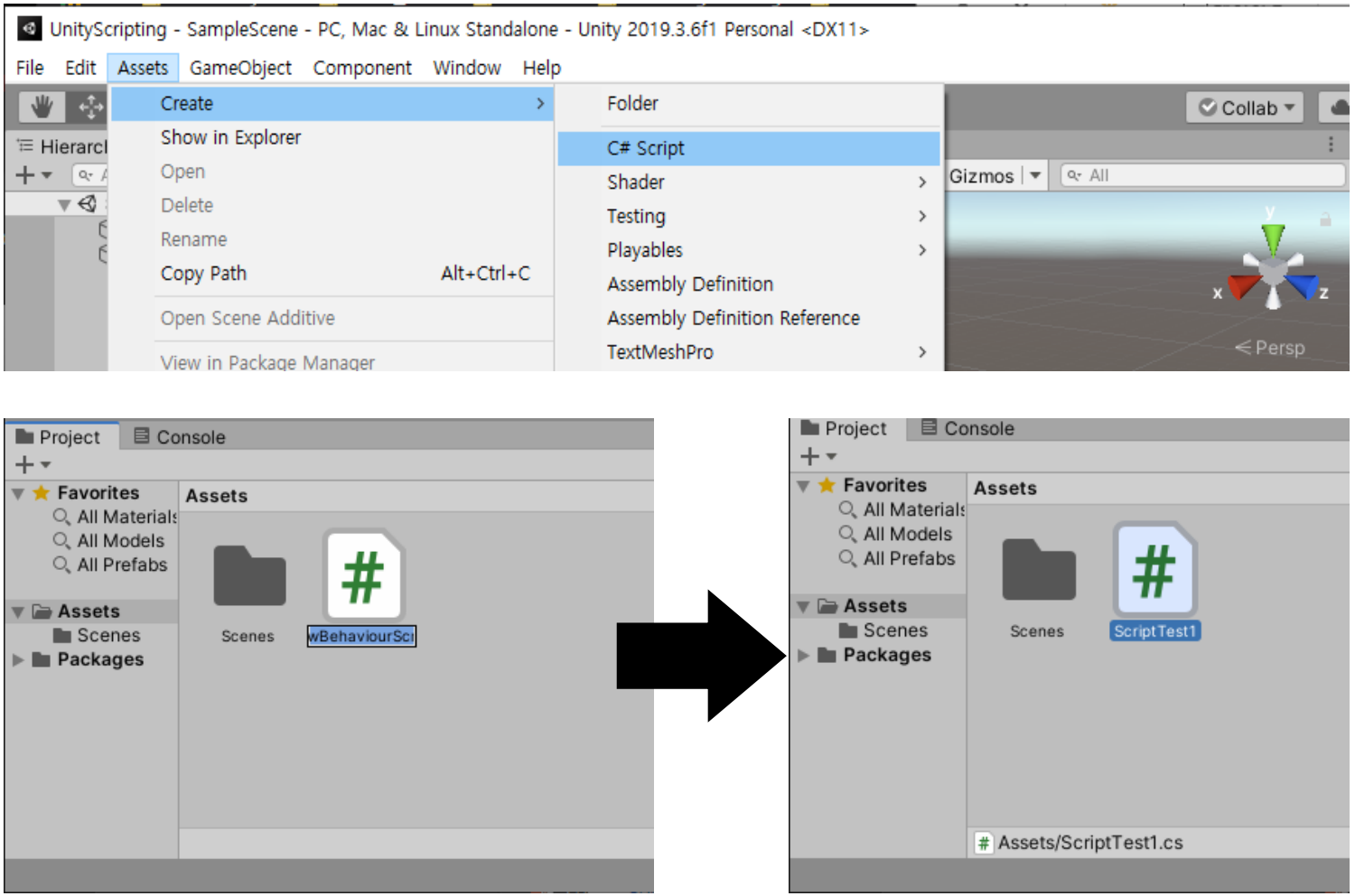
- 3 Languages are supported





  - Unity documentation provides examples for each type.
  - Javascript is often used in online Unity examples, but C# is now becoming first choice in current Unity tutorials .
  - Module using C# to align more with other modules in programs and commercial activities.

Creating Script Assets



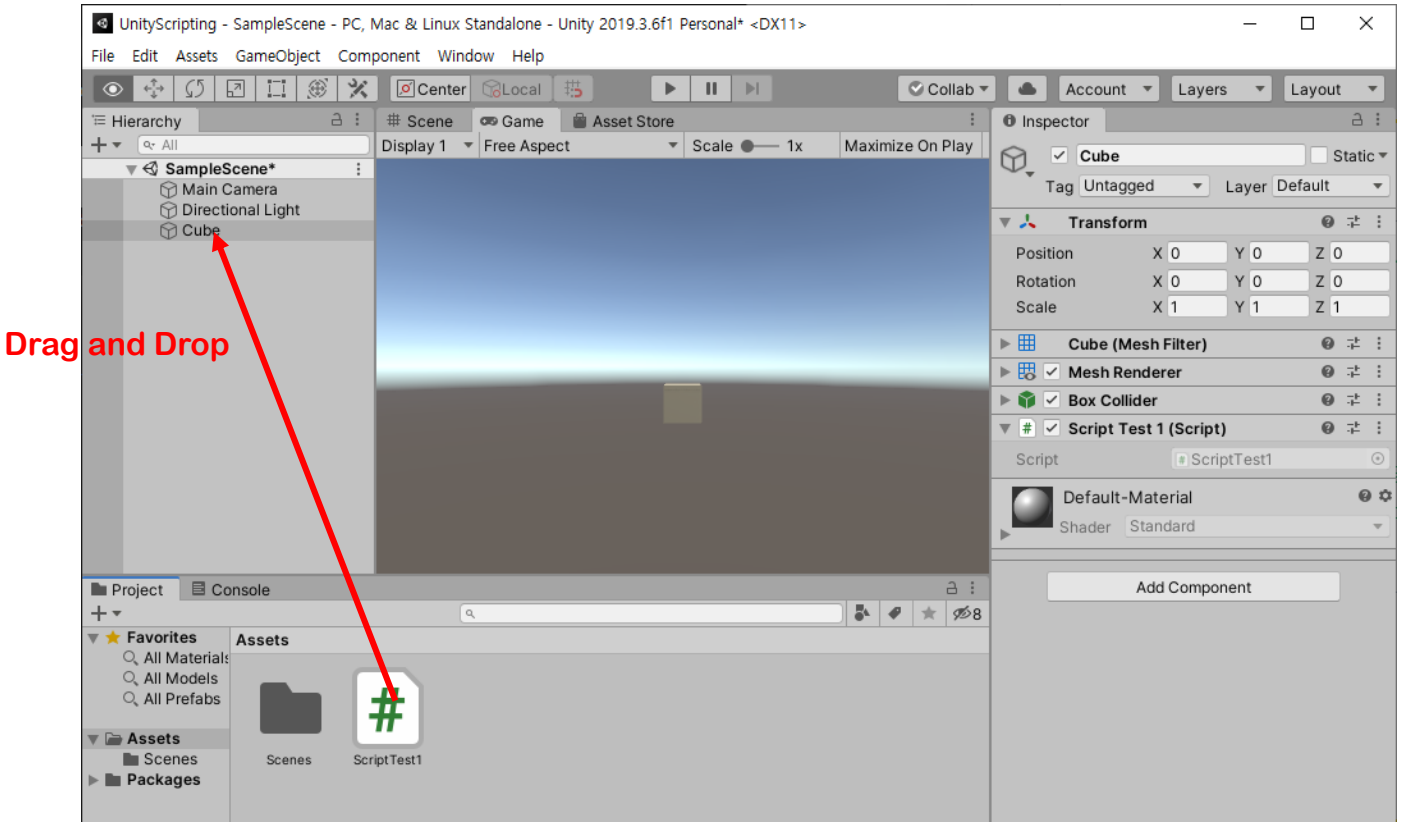
Default Script Structure

- C# Example

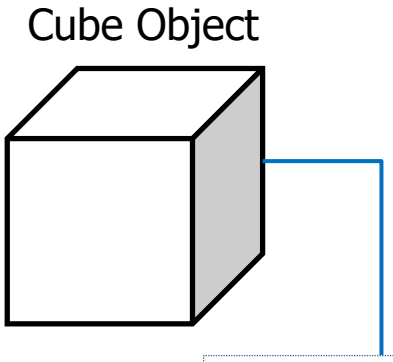
```
using UnityEngine;
using System.Collections;

public class NewBehaviourScript : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

– Attaching the Script to Object

  • 스트립트를 (연결하고자 하는) 객체로 드래그앤드롭



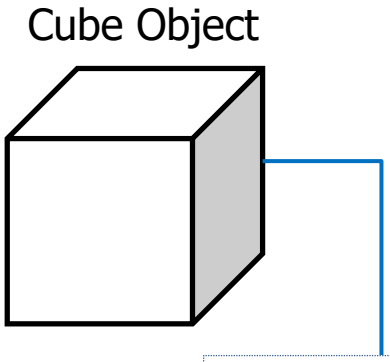
– Changing an Object's Properties via Scripts



ScriptTest2.cs

```
using UnityEngine;
using System.Collections;

public class ScriptTest2 : MonoBehaviour {
    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
        transform.position += new Vector3(0,0,1);
    }
}
```

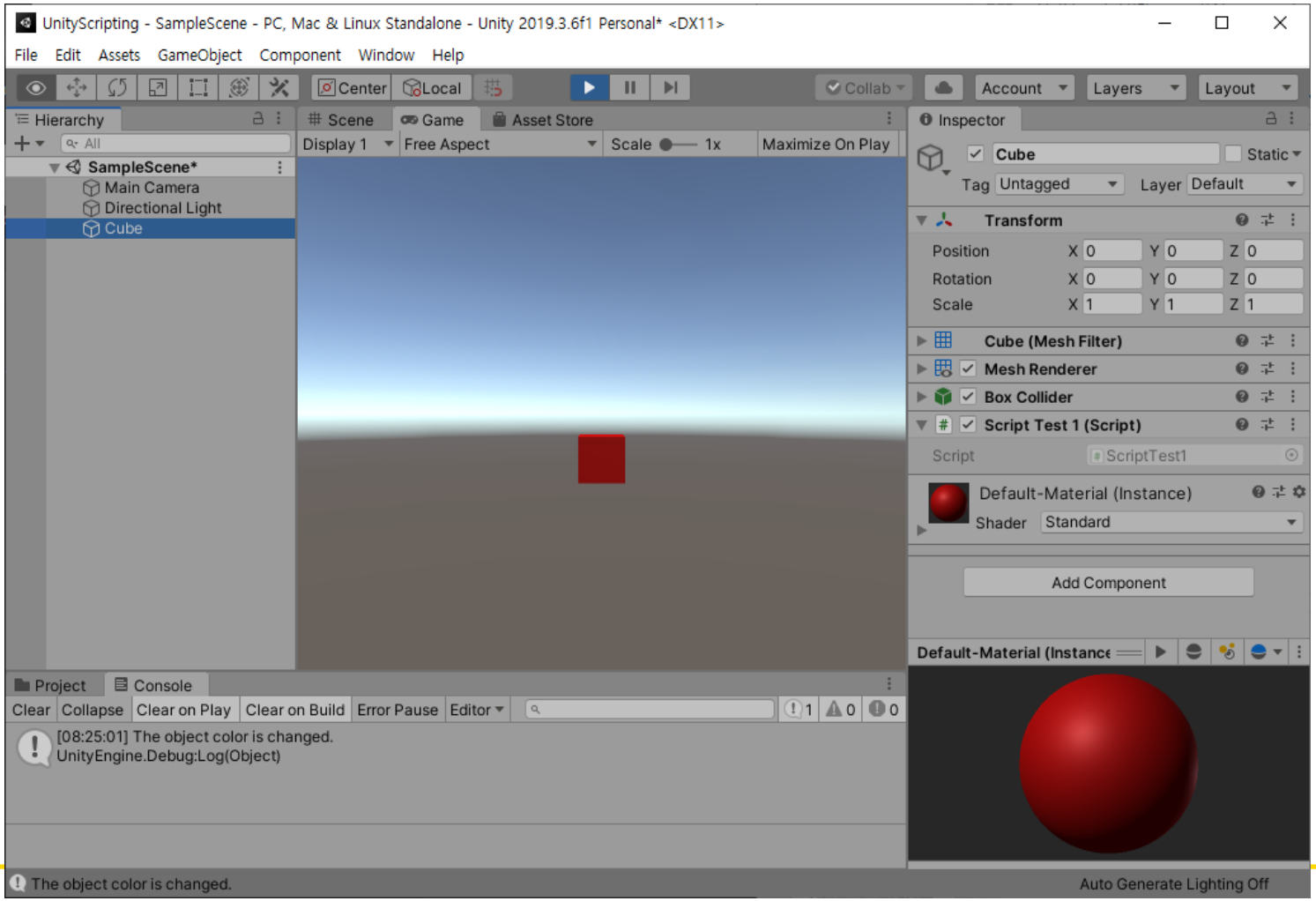
– Changing an Object's Properties via Scripts



ScriptTest1.cs

```
using UnityEngine;
using System.Collections;

public class ScriptTest1 : MonoBehaviour {
    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
        if(Input.GetKeyDown(KeyCode.R)){
            gameObject.GetComponent<Renderer>().material.color = Color.red;
            Debug.Log("The object color is changed.");
        }
    }
}
```

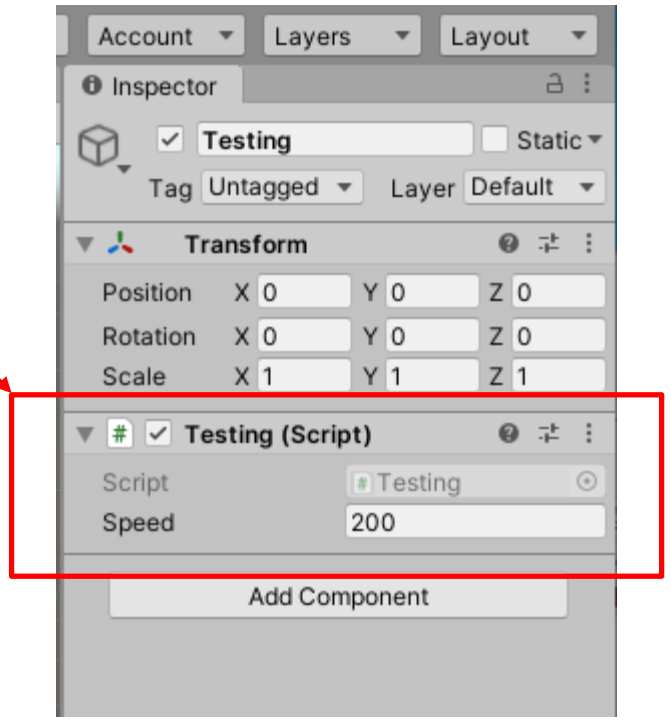
Play and press the key “R”



- public vs. private

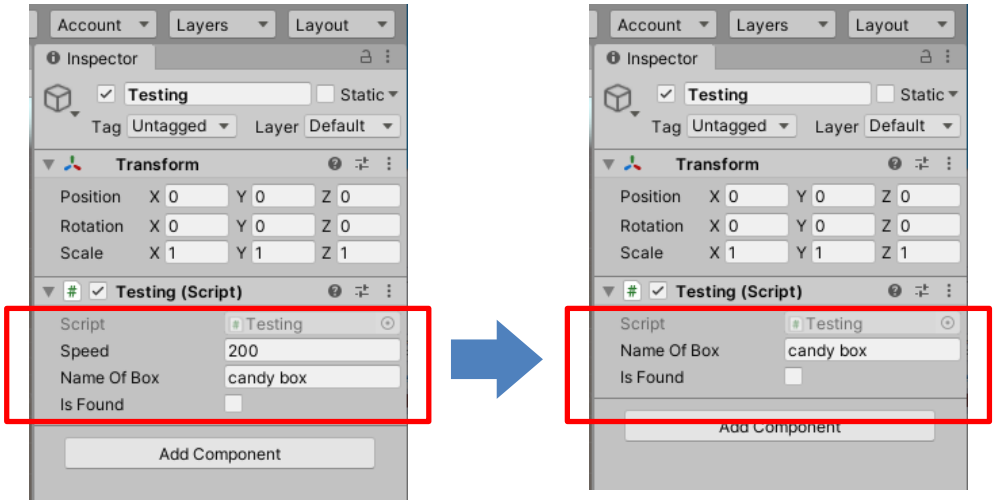
```
public int speed = 100;
private string nameOfBox = “candy box”;
private bool isFound = false;
```

– public 변수는 인스펙터에 보여짐



– public 변수 감추기

```
[System.NonSerialized]
public int speed = 200;
public string nameOfBox = “candy box”;
public bool isFound = false;
```



– private 변수 나타내기

```
[SerializeField] private int power = 100;
private int age = 20;
```

- 유니티 기본 내장 함수
  - Start()
    - 스크립트가 동작하는 동안 단 한번만 호출되는 유니티 기본 함수
    - 초기화 코드 실행에 활용
  - Update()
    - 게임의 매 프레임이 랜더링될 때마다 호출되는 유니티 기본 함수

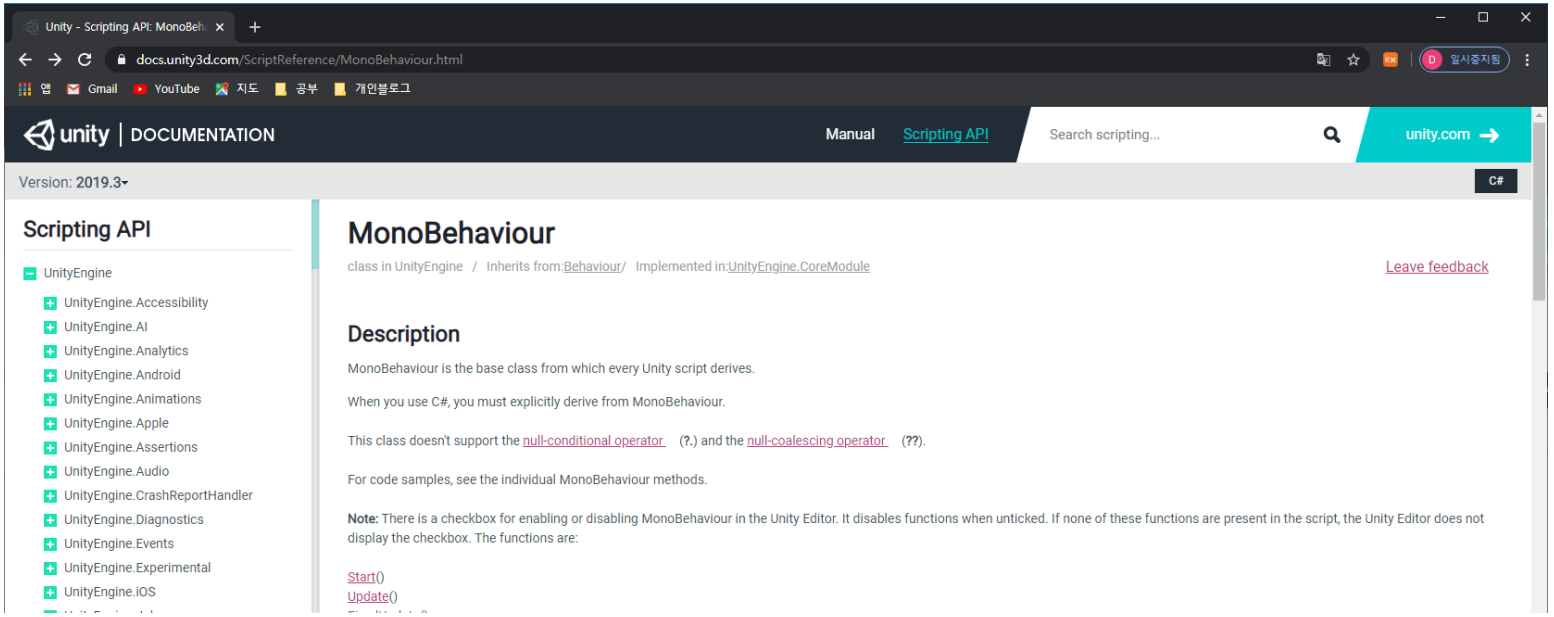
      (예, 초당 30 FPS인 게임에서는 1초에 Update() 함수가 30번 호출됨)
    - 지속적으로 처리되어야 하는 커맨드, 실시간으로 처리되는 게임 내의 변화를 처리할 때 사용

- On~ 계열 함수

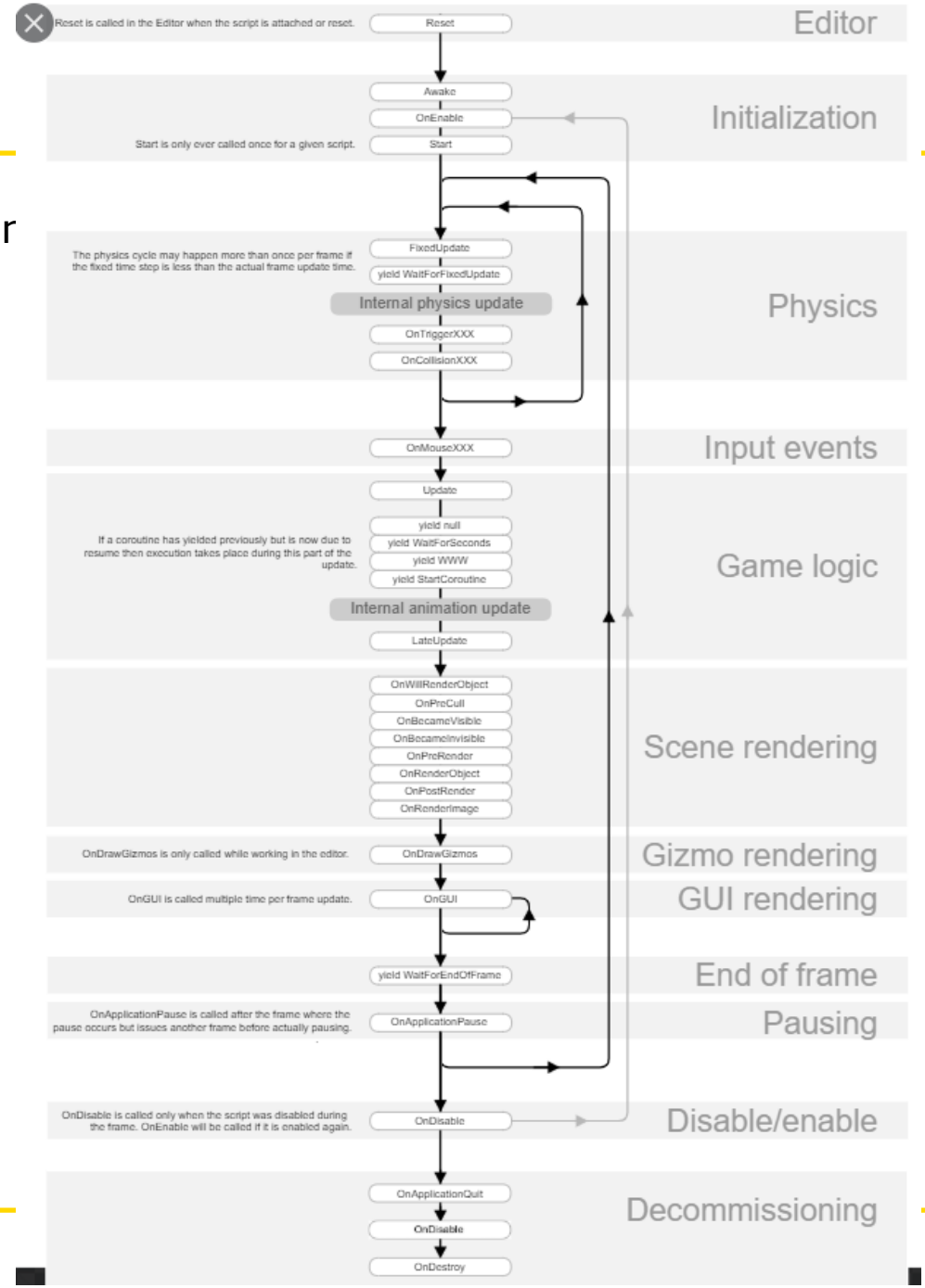

```
void OnMouseDown() {            // 마우스로 오브젝트를 클릭하고 떼는 순간
   Debug.Log(“Mouse click");
}

void OnMouseUp() {              // 마우스가 오브젝트를 클릭했을 때
   Debug.Log(“Mouse UP");       // (마우스를 눌렀다가 떼었을 때)
}

void OnMouseEnter() {           // 마우스가 오브젝트에 들어왔을 때
   Debug.Log(“Mouse Enter");
}

void OnMouseExit() {            // 마우스가 오브젝트 위에 머물다가 빠져 나왔을 때
   Debug.Log(“Mouse out");
}
```

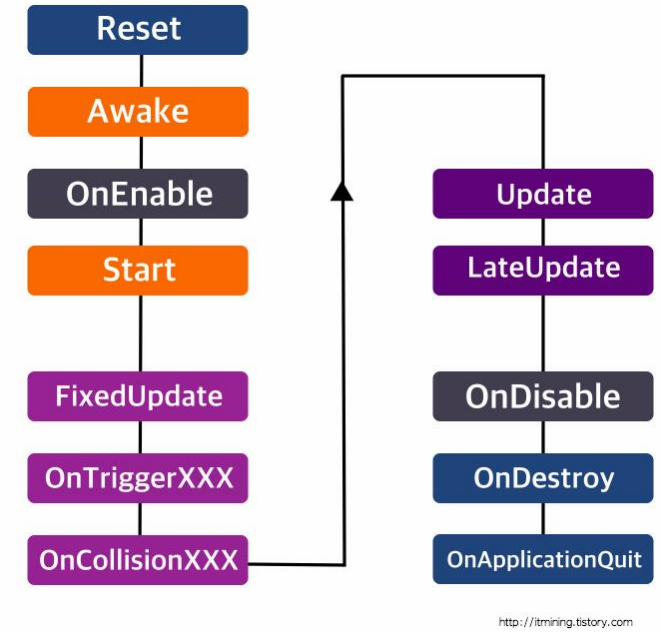
```
참조 0개
public class Testing : MonoBehaviour
{
```

- 미리 정의되어 있는 함수들 ?
  - “Testing” 클래스는 “MonoBehaviour” 클래스로 부터 상속받아 만들어짐
  - MonoBehaviour이 가지는 메소드를 확인



- Order of Execution for Ever Functions





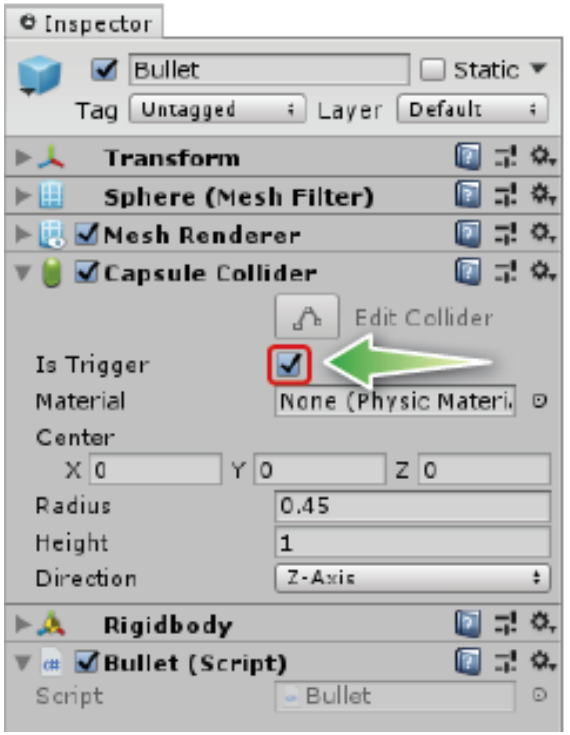
- Awake()
  - 스크립트가 비활성화 되어도 실행
  - 주로 게임의 상태 값 또는 변수 초기화에 사용
  - 1번만 실행, Start() 함수 실행 전에 실행
  - 코루틴(Couroutine) 사용이 불가능
- Start()
  - 1번만 실행, Update() 함수 실행 전에 실행
  - 스크립트가 활성화 되어야 실행
  - 코루틴(Couroutine) 사용 가능
- Update()
  - 매 프레임마다 호출
  - 정기적인 변경, 간단한 타이머, 입력 값 탐지, 카메라 이동 로직에 사용
  - 시간 간격이 동일하지 않음 (이전 프레임에서 오래 걸리면 다음 프레임에 딜레이가 발생)
- FixedUpate()
  - 규칙적인 시간 간격으로 호출 (호출 사이의 시간 간격이 동일)
  - 리지드바디 등 Physics 오브젝트에 영향을 주는 것은 FixedUpdate() 사용을 권장
- OnEnable()
  - 스크립트, 게임 오브젝타가 비활성화 → 활성화 할 때마다 호출
  - 이벤트 연경을 종료할 때 주로 사용
  - 코루틴 사용이 불가능
- OnGUI()
  - Legacy GUI 관련 함수 사용

- OnTrigger … (Collider other) Trigger 함수
- OnCollision … (Collision other) Collision 함수

- 충돌 판정 및 처리 : 게임에서 충돌 판정 및 처리는 매우 중요
- 충돌 이벤트는 Rigidbody가 있는 오브젝트에서만 발생

충돌 이벤트의 종류

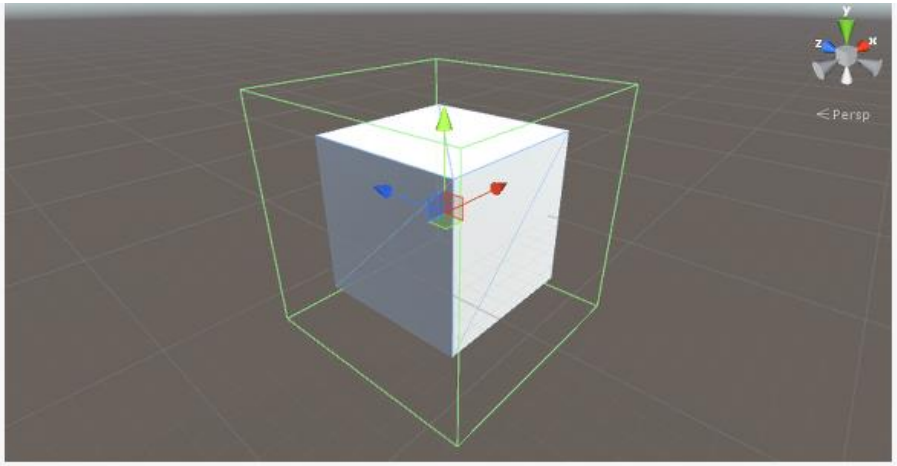
– 충돌이 발생할 때, 충격을 가하는 오브젝트가 반사되느냐 관통하는냐에 따라 발생하는 이벤트가 달라지며, 반사 및 관통 여부는 Collider의 Is Trigger 옵션으로 설정
  - Is Trigger ON : 오브젝트 관통, Trigger 이벤트 발생
  - Is Trigger OFF : 오브젝트에서 반사, Collision 이벤트 발생



- **충돌** : 2개의 오브젝트가 서로 접촉하는 상태
- 적어도 어느 하나가 Trigger On 이면 Trigger 이벤트가, 둘 다 Off면 Collision 이벤트가 발생한다
- 충돌 이벤트의 매개변수의 type
  - OnTrigger … (Collider other) Trigger 이벤트
  - OnCollision ... (Collision other) Collision 이벤트
- Collision의 반사는 절대적인 것이 아니라서 작은 물체가 빠른 속도로 이동하는 경우에는 가끔씩 오브젝트를 관통하기도 한다. 총알이나 화살 같은 발사체는 오브젝트를 관통하기도 하고 반사되기도 하므로, 일반적으로 Trigger를 On을 설정
- 프리팹으로 만든 Bullet도 Trigger을 On으로 설정해야 한다

– 충돌 처리를 위한 조건

- 충돌이 일어나기 위해서는 두 GameObject가 모두 Collider를 가지고 있어야 하며, 둘 중 하나는 Rigidbody를 가지고 있어야 한다
- 두 GameObject 중 하나만 움직인다면, 움직이는 GameObject가 Rigidbody를 가지고 있어야 한다



- 위 그림은 Collider를 잘 보이게 하기 위해 GameObject 보다 크기를 약간 키워 보임. 위에서 보이는 초록색 부분이 Collider로 실제로 충돌을 감지하는 영역임
- 유니티에서 제공하는 Object들에는 기본적으로 Collider가 들어가 있으며, Box Collider, Sphere Collider, Capsule Collider 등이 있음. 다른 Model을 불러와 작업한다면, Model에 알맞게 Collider를 설정해 줘야 올바른 충돌 처리를 할 수 있음

- Trigger

Trigger는 GameObject간의 물리적 연산을 하지 않고 충돌을 감지할 수 있다. 즉, 두 GameObject가 접촉했을때 서로 튕겨 나가지않고 그냥 통과하게 된다.

Trigger를 쓰기 위해서는 해당 Collider의 Is Trigger 항목을 체크해야 한다.

다음으로 스크립트에서 함수로 충돌 이벤트를 처리할 수 있다.

```
1 void OnTriggerEnter(Collider col) { }
2 void OnTriggerStay(Collider col) { }
3 void OnTriggerExit(Collider col) { }
```

다음과 같이 Enter, Stay, Exit 3가지 버전의 OnTrigger 함수를 만들 수 있다.
3가지 함수는 다음과 같은 특징을 나타낸다.
Enter - 충돌이 시작되는 순간 호출
Stay - 충돌이 되고있을 때 매 프레임마다 호출
Exit - 충돌이 끝날 때 호출

함수의 파라메터로 Collider 객체가 들어오며 col을 이용해 충돌한 GameObject에 대한 처리를 할 수 있다.

- Collision

Collision은 Trigger와 다르게 물리적인 연산을 하며 충돌을 감지한다.
주의할 것은 Rigidbody의 Kinematic 속성이 꺼져 있어야 충돌이 발생할 수 있다.

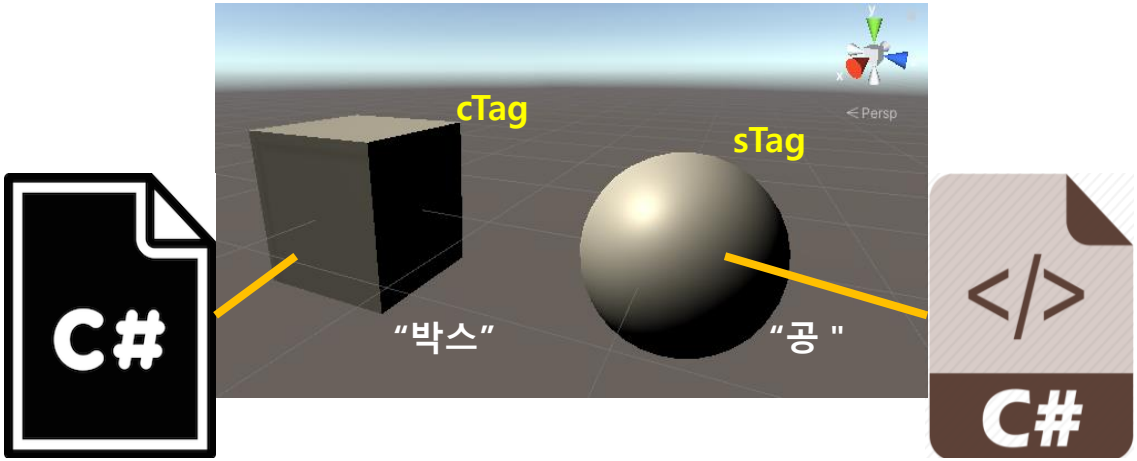
```
1 void OnCollisionEnter(Collision col) { }
2 void OnCollisionStay(Collision col) { }
3 void OnCollisionExit(Collision col) { }
```

Trigger와 동일하게 3가지 함수를 만들 수 있으며 특징 또한 동일하다.

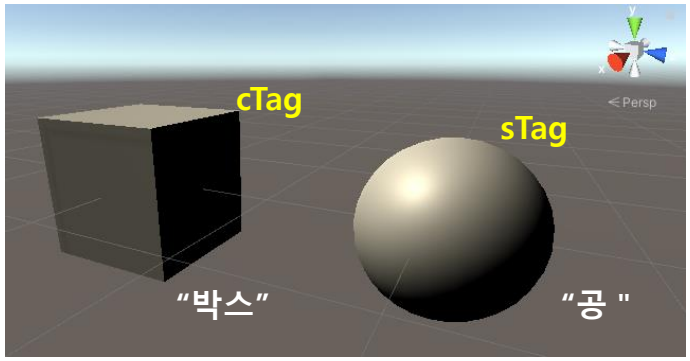
함수의 파라메터로 Collision 객체가 들어오며 col을 이용해 충돌한 GameObject에 대한 처리를 할 수 있다.

스크립트 간의 호출

- 다른 오브젝트에 연결된 스크립트를 실행
  - "박스"에 연결된 스크립트에서 "공"에 연결된 스크립트의 함수를 호출
    - 대상 오브젝트를 찾음
    - 해당 오브젝트에 연결된 스크립트 함수를 호출



- 다른 오브젝트로 접근
  - GameObject.Find() : 이름으로 찾기
  - GameObject.FindWithTag() : 태그로 찾기




```
void Start() {
   GameObject targetObj = GameObject.Find(“박스”);
   Debug.Log(targetObj.transform.position.x);

   GameObject targetObj2 = GameObject.FindWithTag(“sTag”);
   Debug.Log(targetObj2.transform.position.x);
}
```