

인터랙티브 콘텐츠 기초

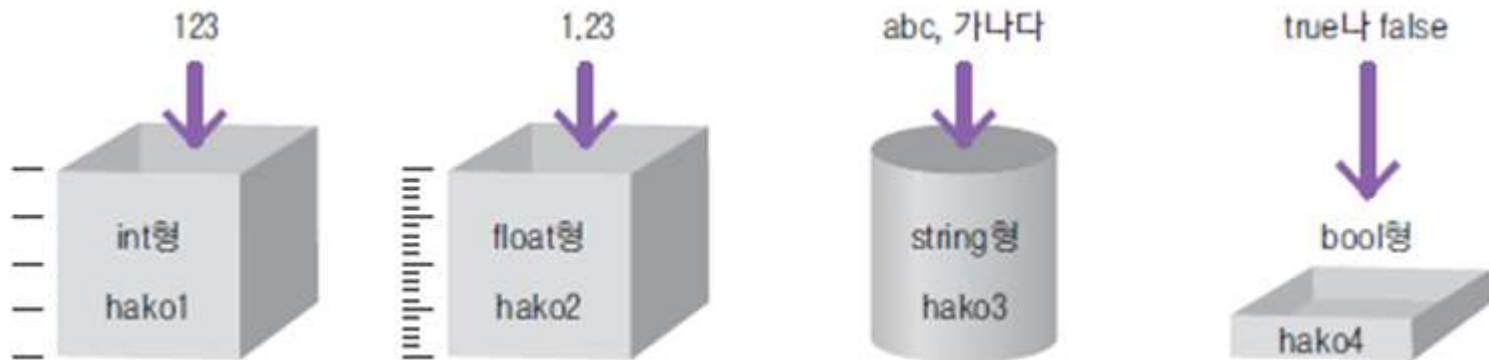
(Chapter 3)

Jin-Mo Kim

jinmo.kim@hansung.ac.kr

변수

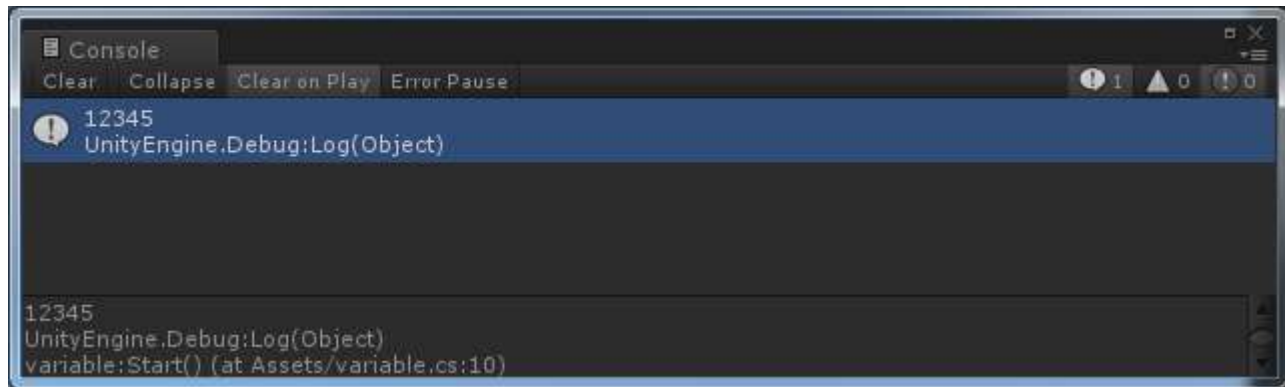
- 어떤 숫자나 문자열을 저장할 수 있는 상자
 - 자료형 : 변수의 형태
 - ex) int – 정수형, float – 실수형, string – 문자열, bool– 참/거짓 판정



변수 사용하기 (정수형)

- C# 스크립트

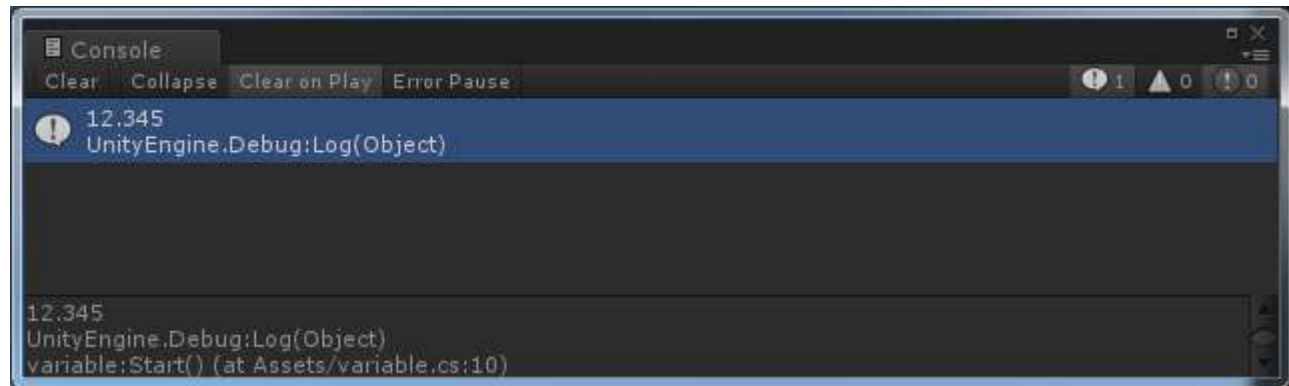
```
void Start () {  
    int num;  
    num = 12345;  
    Debug.Log (num);  
}
```



변수 사용하기 (실수형)

- C# 스크립트

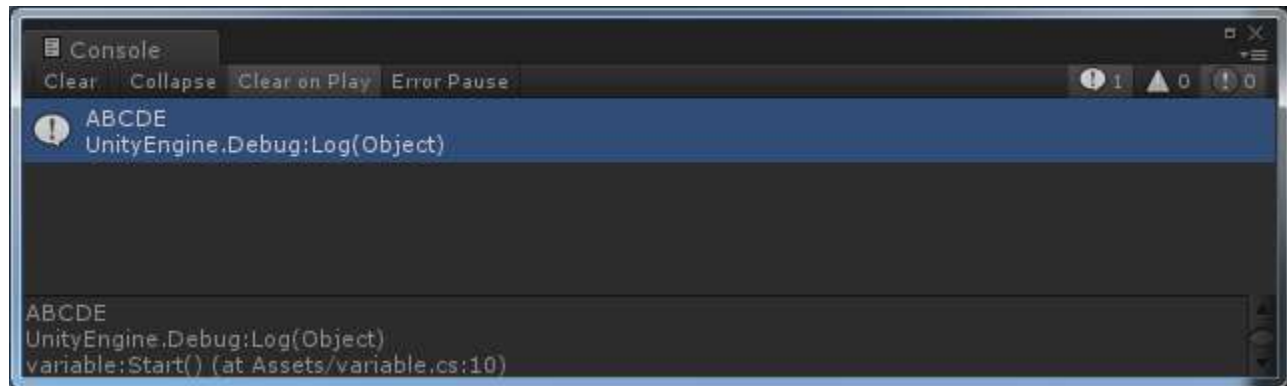
```
void Start () {  
    float num;  
    num = 12.345f;  
    Debug.Log (num);  
}
```



변수 사용하기 (문자열)

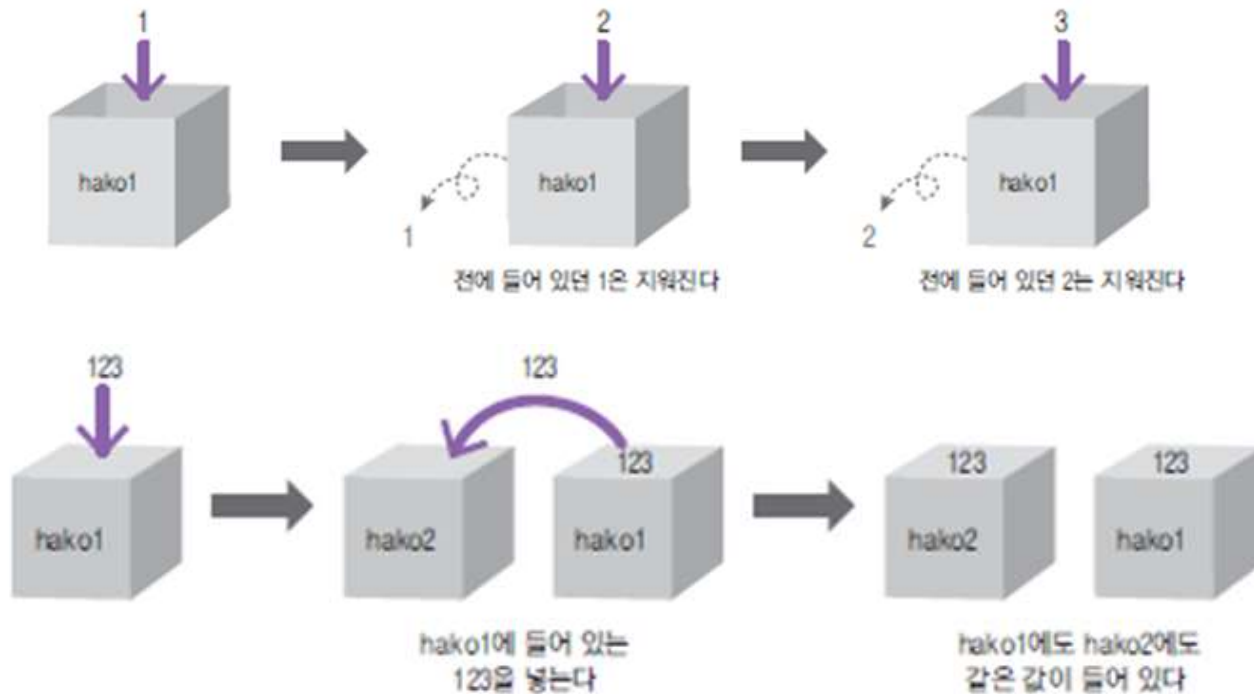
- C# 스크립트

```
void Start () {  
    string num;  
    num = "ABCDE";  
    Debug.Log (num);  
}
```



변수의 특성

- 변수의 내용은 원하는 대로 바꿀 수 있다.
- 변수에 변수를 대입할 수 있다



변수의 연산

- 사칙 연산
 - +, -, *, / 를 이용한 사칙연산
- 증감 연산
 - ++, -- 로 증가, 감소 연산
- 복합대입 연산자

```
hako = 1;      hako = 5;  
hako += 10;    hako -= 1;
```

11

4

```
hako = 10;  
hako *= 4;
```

40

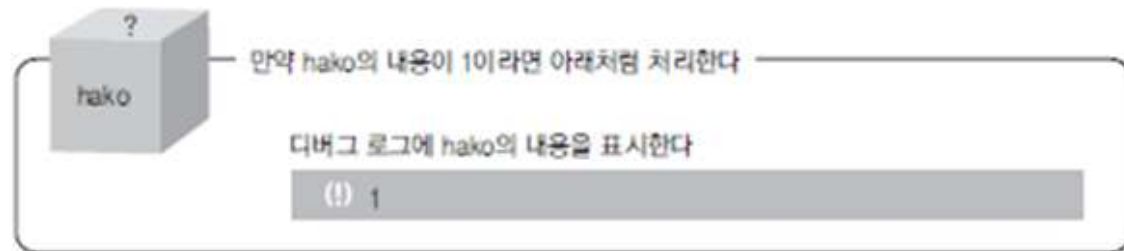
```
hako = 24;  
hako /= 6;
```

4

조건문

- 지정한 조건에 맞으면 프로그램 구문을 실행한다.
 - if문 사용

```
void Start () {  
    int num;  
    num = 1;  
  
    if (num == 1) {  
        Debug.Log (num);  
    }  
}
```



조건문

- 지정한 조건에 맞으면 프로그램 구문을 실행한다.
 - if문 사용

```
void Start () {  
    int num;  
    num = 1;  
  
    if (num == 1) {  
        Debug.Log (num);  
    }  
}
```

num == 1	num가 1이면
num >= 1	num가 1 이상이면
num <= 1	num가 1 이하이면
num != 1	num가 1이 아니면

조건문

- if ~ else
 - 조건이 맞지 않을 때, 실행할 수 있는 구문을 추가할 수 있다.

```
void Start () {  
    int num;  
    num = 1;  
  
    if (num == 1) {  
        Debug.Log ("1");  
    }else {  
        Debug.Log ("not 1");  
    }  
}
```

조건문

- if ~ else if ~ else
 - 조건이 맞지 않을 때, 실행할 수 있는 구문을 추가할 수 있다.

```
void Start () {  
    int num;  
    num = 1;  
  
    if (num == 1) {  
        Debug.Log ("1");  
    } else if (num == 2) {  
        Debug.Log ("2");  
    }  
    else {  
        Debug.Log ("not 1");  
    }  
}
```

조건문

- 조건에 일치한다면 같은 처리를 반복한다.
- switch ~ case
 - 변수의 값을 보고 case 중 일치하는 것을 실행한다.

```
void Start () {  
    int num = 1;  
    switch (num) {  
        case 0:  
            Debug.Log ("0");  
            break;  
        case 1:  
            Debug.Log ("1");  
            break;  
        default:  
            Debug.Log ("0도 1도 아님");  
            break;  
    }  
}
```

주의사항

1. case(값) 형식이 아니라 case 1
- case 다음에 공백 넣고 값
2. 값 뒤에 콜론(:)으로 구분
case 1 :
3. case 구문 마지막에는 **break**

반복문


- for문
 - 같은 처리를 반복할 때 사용

기본형

```
for(변수의 초기값; 실행조건; 변화값) {  
    실행될 처리  
}
```

ex)

```
for(int i = 0; i<10; i++) {  
    Debug.Log(i);  
}
```

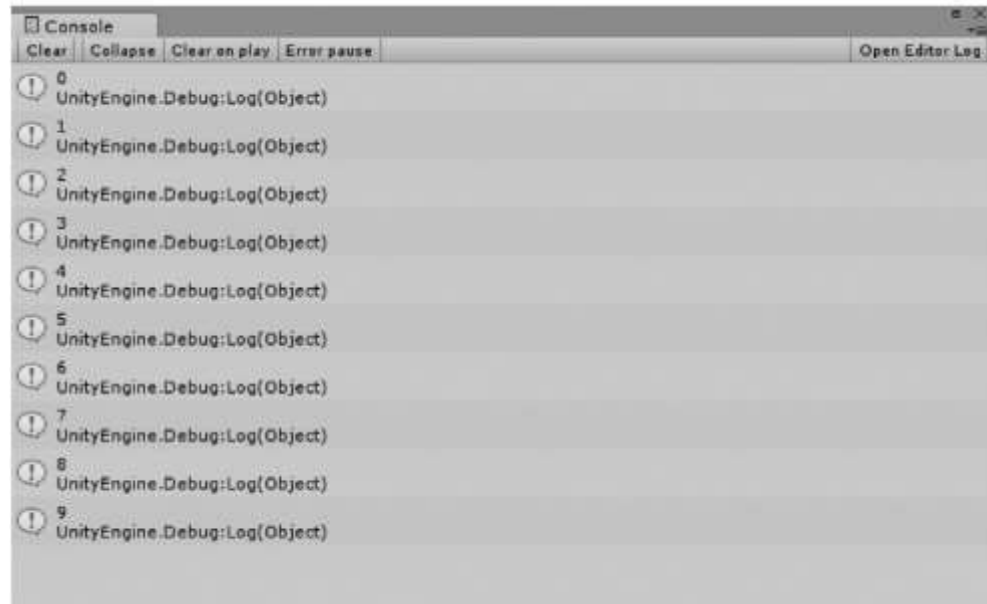


i의 초기값은 0이다.
i가 10보다 작으면 i의 값을 출력하고 i를 하나 증가 시켜라
i=10 이면 이 반복문은 실행되지 않는다.

반복문

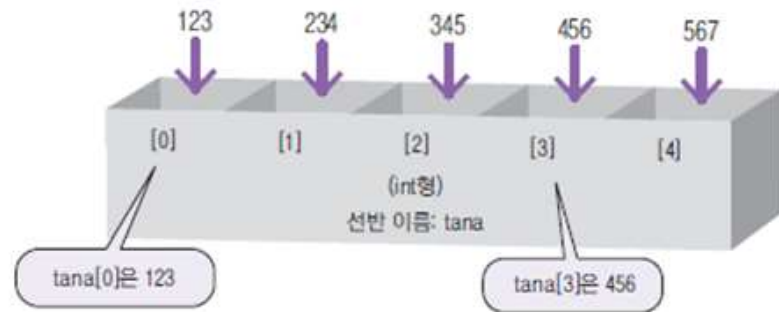
- for문
 - 같은 처리를 반복할 때 사용

```
for(int i = 0; i<10; i++) {  
    Debug.Log(i);  
}
```



배열

- 여러 개의 값을 넣을 수 있다.
- 값이 있는 곳에는 각각 주소가 부여된다.
- 주소로 저장된 값을 불러올 수 있다.
- 하나의 배열에는 하나의 변수형
- 배열의 가장 첫 번째 칸은 **0**으로 시작
- 작성할 때는 변수형[] ex) int array[5]



배열

- 배열 선언하는 방법 1.
 - 선언과 동시에 값을 넣어준다.

```
void Start () {  
    int[] arr = { 123, 234, 345 };  
  
    for (int i = 0; i < arr.Length; i++) {  
        Debug.Log (arr[i]);  
    }  
}
```

- 배열 선언하는 방법 2.
 - 빈 배열을 선언 후 나중에 값을 넣어준다.

```
void Start () {  
    int[] arr = new int[5];  
  
    Debug.Log (arr [0]);  
    arr [0] = 32;  
    Debug.Log (arr [0]);  
}
```


배열

- 배열의 특성
 - 다른 배열의 값을 대입할 수도 있다.

```
void Start () {  
    int[] arr1 = { 1, 2, 3, 4, 5 };  
    int[] arr2 = new int[5];  
  
    arr2 = arr1;  
    Debug.Log (arr2 [0]);  
}
```

배열

권장하지 않는다

- foreach
 - 배열의 모든 요소를 표시

```
void Start () {  
    int[] arr = { 1, 2, 3, 4, 5 };  
  
    foreach (int i in arr) {  
        Debug.Log (arr [i]);  
    }  
}
```

하지만 유니티에서는
foreach로 더 간편하게 모든 값을 표시할 수 있다

Vector형 변수

- vector2, vector3
 - 2D, 3D를 표현할 수 있는 자료형 중 하나
 - 캐릭터가 3D 공간(또는 2D) 어디에 있고, 어디에 진행하려 하는지,
 - 어느 방향으로 힘을 가하려 하는지 등 **캐릭터의 움직임을 나타내는 데 유용.**



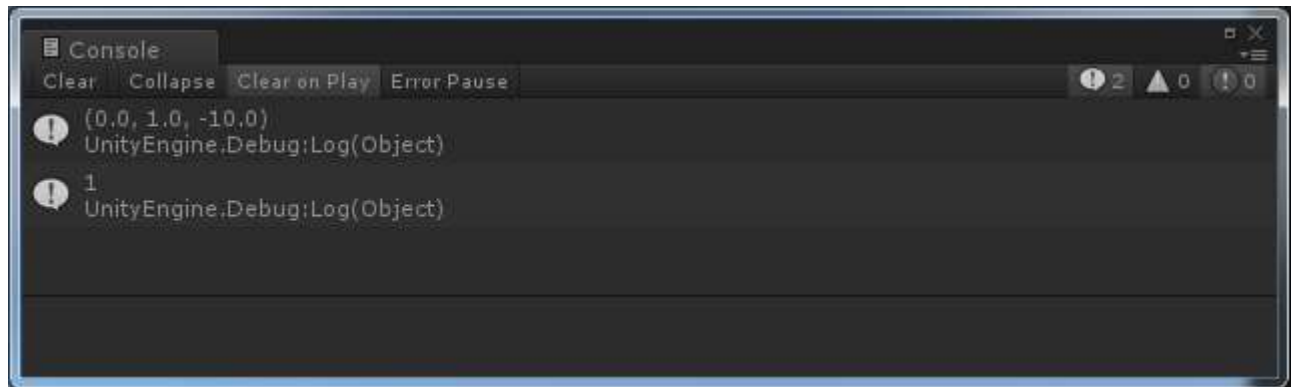
Vector형 변수

- vector2, vector3
 - 2D, 3D를 표현할 수 있는 자료형 중 하나

```
void Start () {  
    Vector3 pos;  
  
    pos = this.gameObject.transform.position;  
  
    Debug.Log (pos);  
    Debug.Log (pos.y);  
}
```

this는 코드는

적용받은 물체를 의미



변수의 유효범위

- 유효범위
 - 변수가 참조할 수 있는 범위

public인 변수(외부에서 보이고 사용할 수 있다)



private인 변수(외부에서 보이지 않고 사용할 수도 없다)



- public
 - 외부에서 보이고 외부에서 내용을 변경할 수 있는 변수
- Private
 - 외부에서 보이지 않고 외부에서 내용을 변경할 수도 없는 변수

변수의 유효범위

- 유효범위

```
public class VarRange : MonoBehaviour {  
  
    public int var1 = 0;  
    private int var2 = 1;  
  
    // Use this for initialization  
    void Start () {  
        Debug.Log (var1);  
        Debug.Log (var2);  
    }  
  
    // Update is called once per frame  
    void Update () {  
  
    }  
}
```

studyScript의 inspector를 들어가서
변수의 값을 바꿔보고,
어떤 값이 출력되는 지 확인해보자

메서드

- 메서드

```
// Use this for initialization
void Start () {
    int damage = method ();
    Debug.Log (damage + "데미지!");
}

int method(){
    Debug.Log ("공격");
    int power = 1200;
    return power;
}
```

Method - 위 코드에서 method와 같이 자주 사용하는 프로그램이나 동작을 한 덩어리로 묶어 등록한 것

위치는 start() 앞뒤 상관 없으며 Update()와 start()사이에 있어도 괜찮다.
병렬만 맞춰서 작성하면 위치는 상관없다.

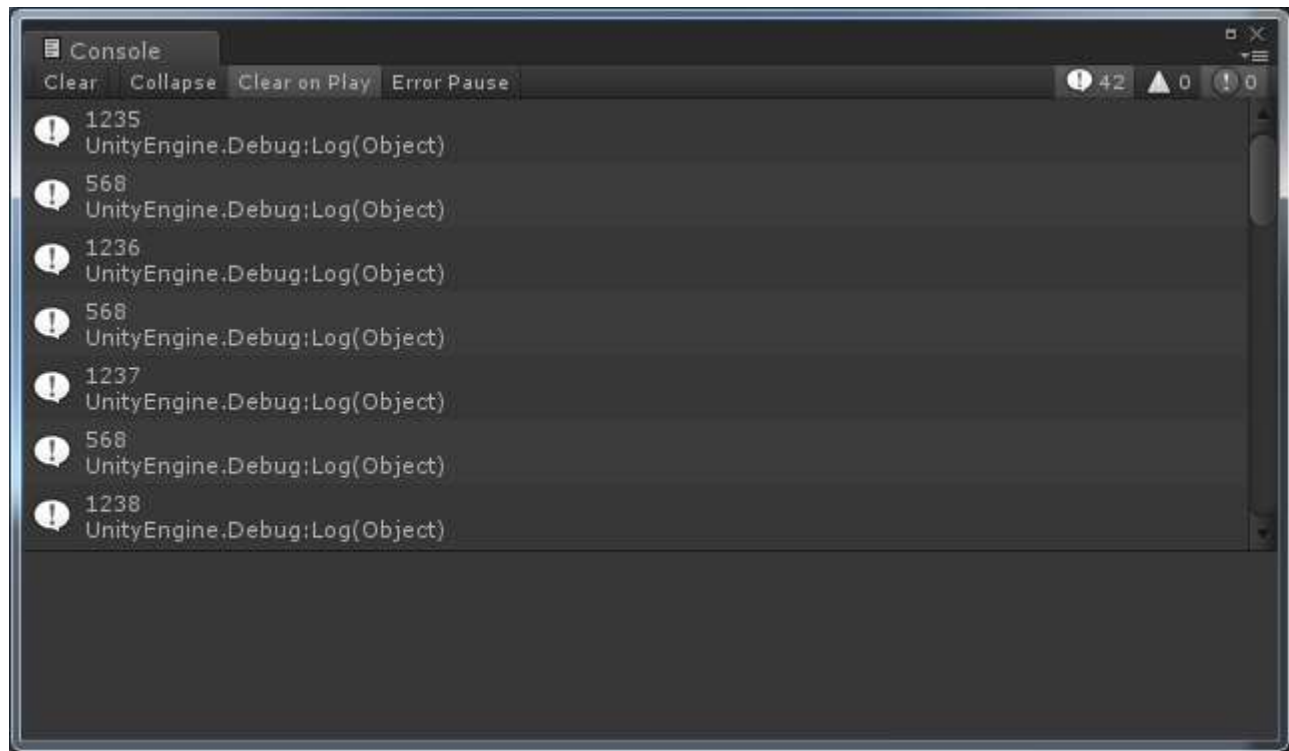
클래스와 로컬 변수, 멤버 변수

- 선언되는 위치에 따라 로컬변수, 멤버변수로 나눌 수 있다.

```
public class Variable : MonoBehaviour {  
    void Start () {  
  
    }  
  
    int score = 1234;                // 멤버 변수  
  
    // Update is called once per frame  
    void Update () {  
        int num = 567;              // Update() 안에서만 유효한 로컬 변수  
        score++;  
        num++;  
  
        if (num > 100) {  
            int result = 89;  
            result++;  
        }  
        Debug.Log (score);  
        Debug.Log (num);  
        //Debug.Log (result);  
    }  
}
```

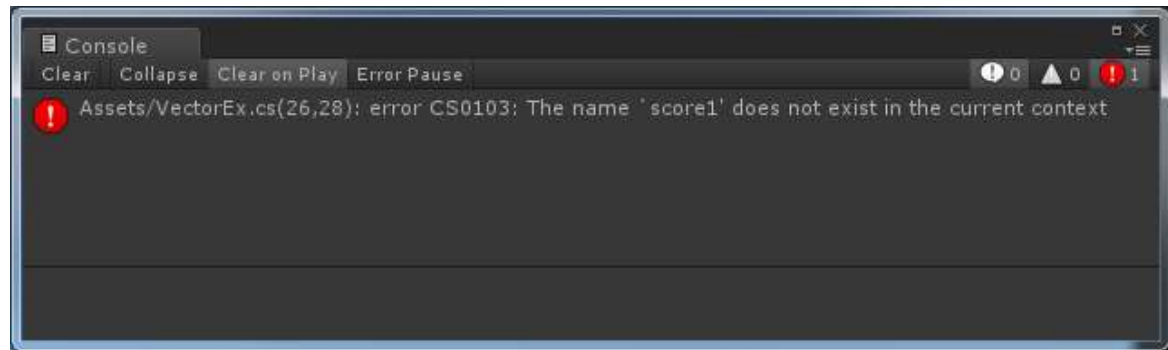

클래스와 로컬 변수, 멤버 변수

- 선언되는 위치에 따라 로컬변수, 멤버변수로 나눌 수 있다.



오류가 발생한 경우

- 오류메시지를 참고하라
 - 오류 발생위치를 알려준다. 오류가 난 곳을 찾아 수정하도록 한다.



- 수상한 곳에 Debug.Log를 설치한다
 - 실행은 되지만 어딘가 이상할 때, 그 부분에 Debug.log를 끼워 넣은 후 제대로 돌아가는 지 확인한다.

실행을 하지 않고도 미리 콘솔에서 빨간 에러를 확인할 수 있다

키보드와 마우스 입력

- 키보드 입력
 - “Input.동작(키코드)”의 형태로 사용한다.
 - Input.GetKey : 키가 눌린 상태 (누적)
 - Input.GetKeyDown : 키가 눌린 순간
 - Input.GetKeyUp : 키에서 손이 떨어진 순간
 - Input.AnyKeyDown : 아무 키라도 눌러라!

키코드	키
KeyCode.Space	Space bar
KeyCode.Return	Enter 키 (Return 키)
KeyCode.UpArrow	↑ 키
KeyCode.DownArrow	↓ 키
KeyCode.LeftArrow	← 키
KeyCode.RightArrow	→ 키
KeyCode.Escape	ESC 키
KeyCode.Backspace	Backspace 키

키코드	키
KeyCode.X	X 키
KeyCode.S	S 키
KeyCode.LeftShift	왼쪽 (Shift) 키
KeyCode.RightShift	오른쪽 (Shift) 키
KeyCode.LeftControl	왼쪽 (Ctrl) 키
KeyCode.RightControl	오른쪽 (Ctrl) 키
KeyCode.Alpha1	[1] 키
KeyCode.F1	F1 키

키보드와 마우스 입력 *cs_key.cs*

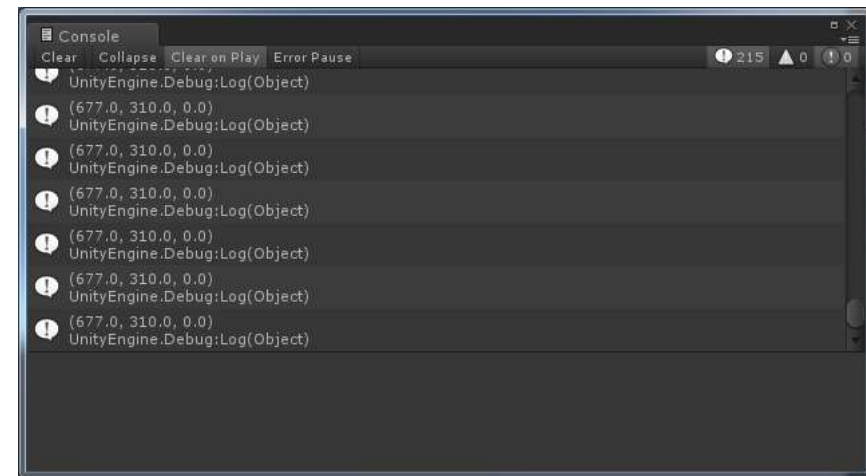
- 활용 예))
 - Input.GetKey : 키가 계속 눌린 상태

```
void Update () {  
    if (Input.GetKey (KeyCode.Space)) {  
        Debug.Log ("space!");  
    }  
}
```

키보드와 마우스 입력 *cs-key.cs*

- 마우스 입력
 - 키보드 입력과 같다. Key를 Mouse로 바꿔 사용하면 된다.
 - Input.GetMouseButton : 마우스가 눌러있는 동안
 - Input.GetMouseButtonUp : 마우스에서 손이 떨어졌을 때
 - Input.~~AnyMouseDown~~ : 클릭하는 순간
Get MouseButton Down
 - 활용 예))
 - Input.mousePosition : 마우스 포인터의 위치를 구한다.

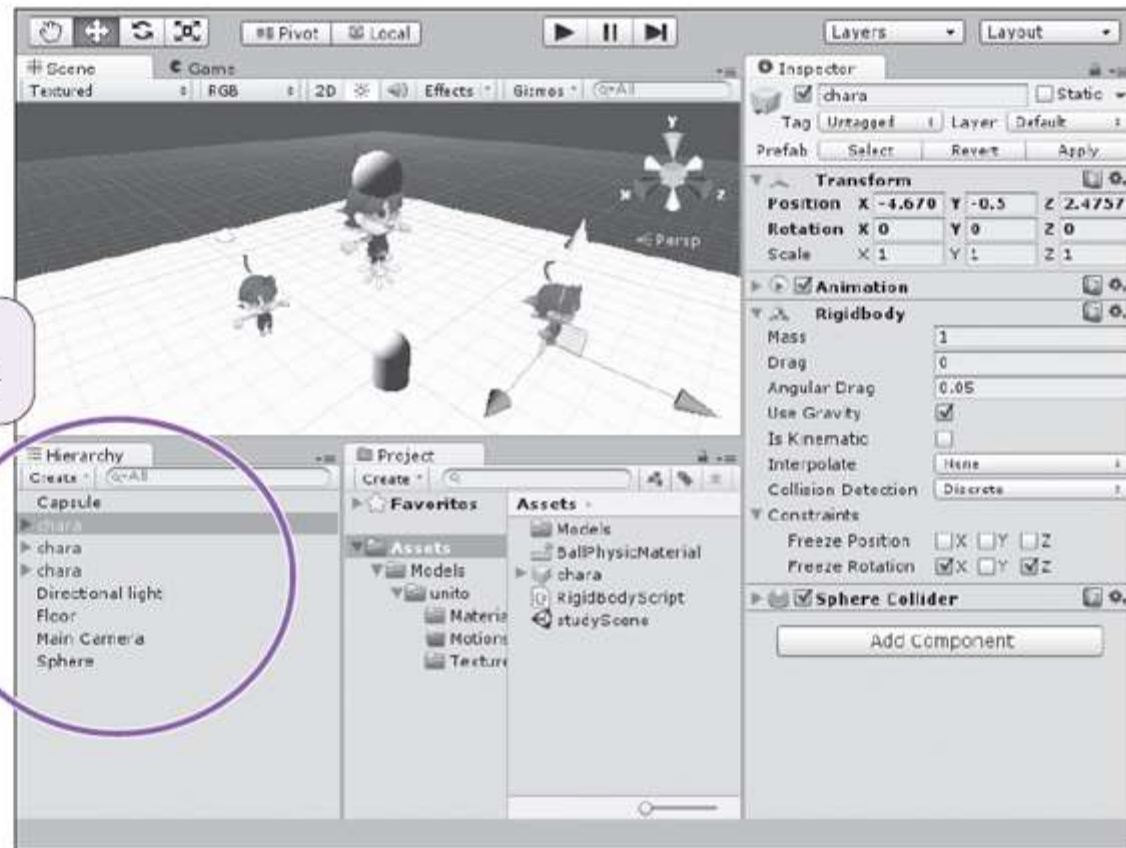
```
void Update () {  
    Debug.Log (Input.mousePosition);  
}
```



게임 오브젝트

- 게임 오브젝트

카메라도 조명도
모두 게임 오브젝트다

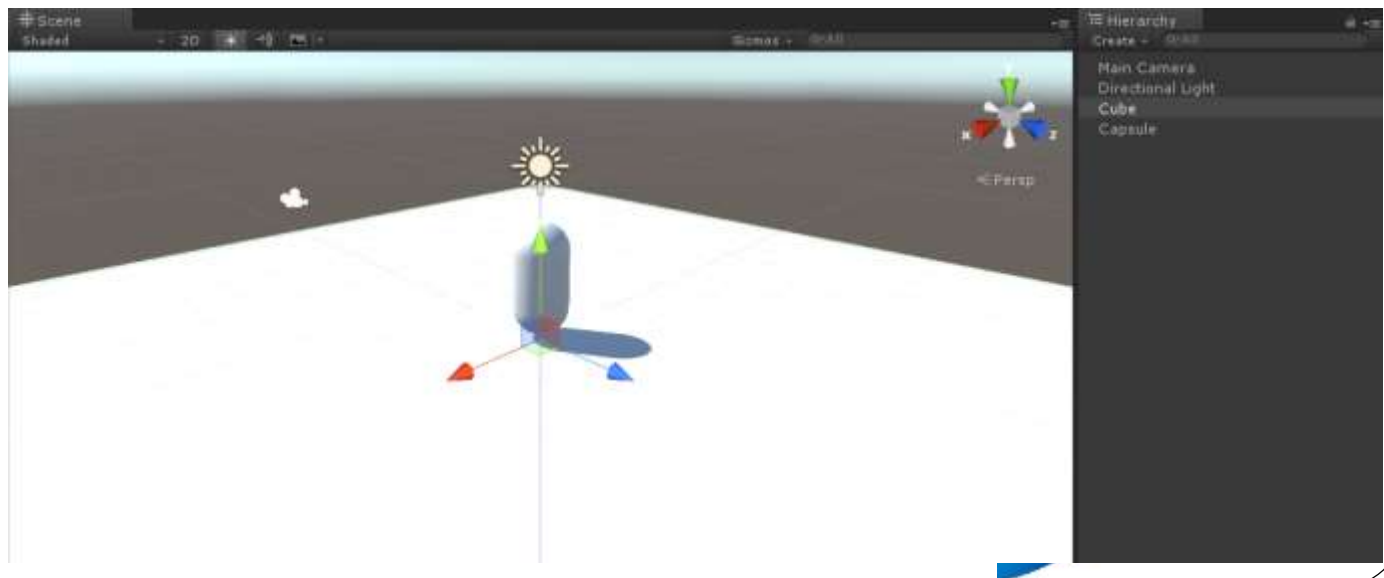


게임 오브젝트

- 게임 오브젝트
 - 유니티의 씬에 배치된 것은 모두 게임 오브젝트
 - 게임 상에서 보이지 않는 존재도 포함 (사운드 등)
 - 게임 오브젝트에 스크립트를 추가하여 동작시킨다.

실습

- 실습
 - 바닥
 - [GameObject]-[3D Object]-[Cube]
 - 조명
 - [GameObject]-[Light]-[Directional Light]
 - 캡슐 + 스크립트
 - [GameObject]-[3D Object]-[Capsule]
 - [Assets]-[Create]-[C# Script]



Transform 변경하기 *CS_player.cs*

- 오브젝트 안에 사용된 Inspector의 각도, 크기, 회전을 변경

```
void Update () {  
    if(Input.GetKeyDown(KeyCode.A)){  
        float rnd = Random.Range(0.0f, 0.5f);  
        this.transform.position = new Vector3(0.0f, 1.0f, rnd);  
    }
```

A키가 눌리면 거리가 조금씩 변한다.

```
    if(Input.GetKeyDown(KeyCode.B)){  
        float rnd = Random.Range(0.0f, 360.0f);  
        this.transform.rotation = Quaternion.Euler (rnd, 0.0f, 0.0f);  
    }  
}
```

B키가 눌리면 오브젝트를 회전시킨다.
X축 기준으로 임의의 각도만큼 회전.

Translate

CS_player.cs

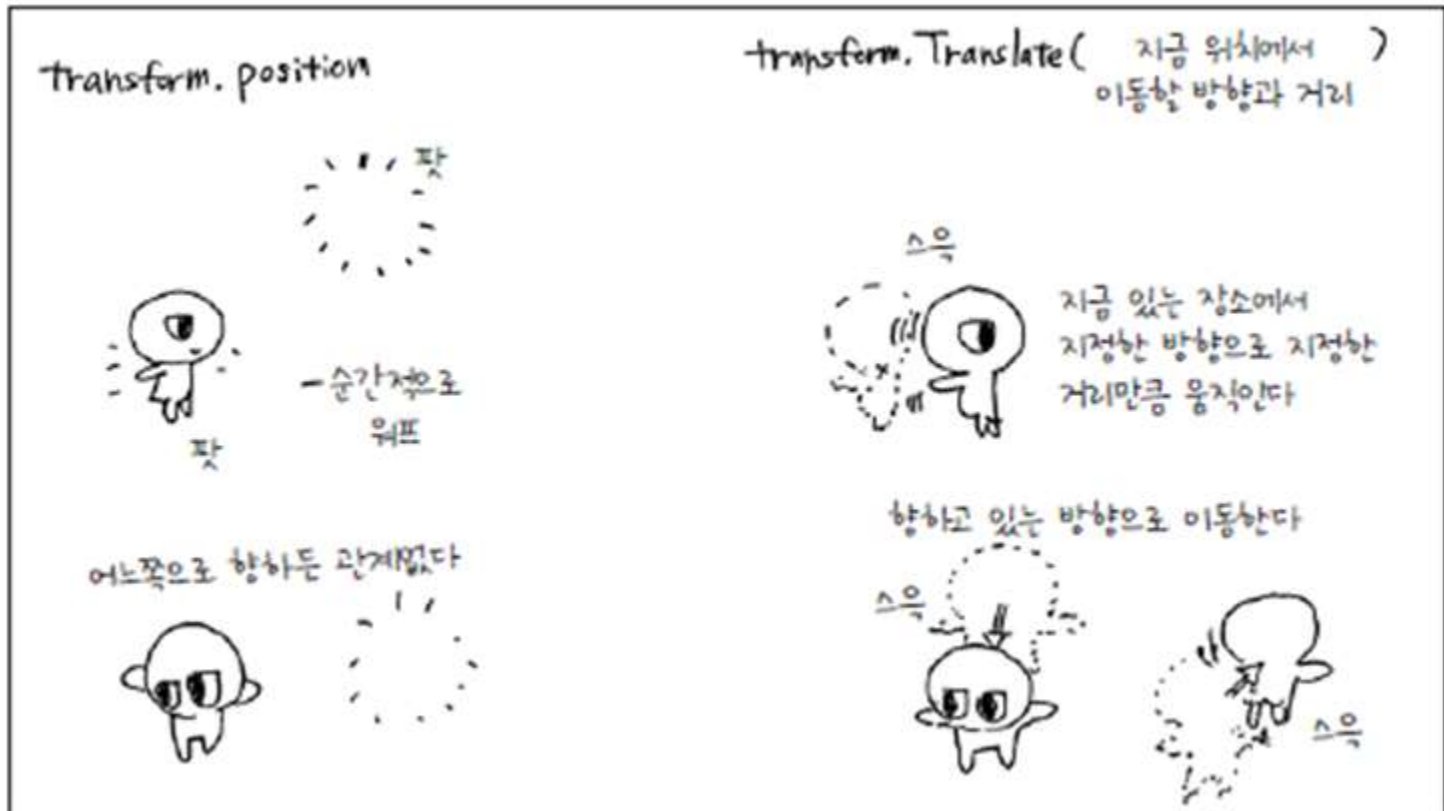
- 오브젝트 안에 사용된 Inspector의 각도, 크기, 회전을 변경

```
void Update () {  
    ...  
    if(Input.GetKey(KeyCode.UpArrow)){  
        this.transform.Translate (new Vector3 (0.0f, 0.0f, 3.0f * Time.deltaTime));  
    }  
}
```

오브젝트 자신이 현재 위치와 각도를 기준으로 조작한다.

- Transform과 다르게 단순히 이동할 거리만 지정. 오브젝트가 향한 방향으로 진행한다. 객체기준 0, 월드기준 X
- 절대위치를 단순한 표현 가능, 하지만 변하는 **방향에 맞춰 위치 변경하려면 재계산 필요**

Transform과 Translate



Time.deltaTime

Time.deltaTime 은
검사량에 따라서
업데이트 속도가 다른 것을
보완해준다

Time.deltaTime을 사용하지 않으면

빠른 기기에선 빨리 달린다



느린 기기에선 느리게 달린다



Time.deltaTime을 사용하면

빠른 기기나 느린 기기나
1초당 이동 속도는 같다



갱신 시간이 0.25초면
1회 이동 거리는 25m

갱신 시간이 0.5초면
1회 이동 거리는 50m

고사양

저사양

기기에 따라 갱신 횟수가 다르다. 그 갱신 횟수에 맞춘 이동 거리를 지정한다

Time.deltaTime

- 기기의 성능에 따라 게임의 갱신빈도가 다르다.
- 기기 별 성능의 차이를 메워주는 것

→ Time.deltaTime

- 초당 이동 거리와 회전량 지정
- 빠른 기기 : 1회 이동거리 ↓ 갱신 빈도 ↑
- 느린 기기 : 1회 이동거리 ↑ 갱신 빈도 ↓

Rotate

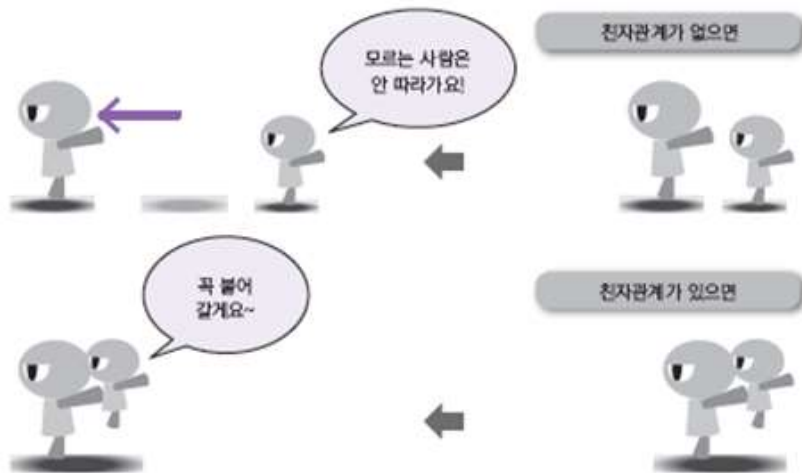
- 오브젝트가 향한 방향에서 얼마나 회전시킬 것인가.
 - 상대적인 회전

```
void Update () {  
    ... ..  
    if(Input.GetKey(KeyCode.R)){  
        this.transform.Rotate (90.0f * Time.deltaTime, 0.0f, 0.0f);  
    }  
    if(Input.GetKey(KeyCode.L)){  
        this.transform.Rotate (-90.0f * Time.deltaTime, 0.0f, 0.0f);  
    }  
}
```

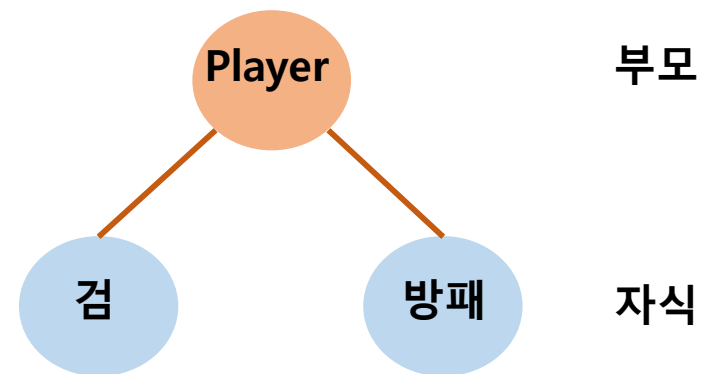
절대적 회전은 rotation. 잊지 말자!

오브젝트의 친자 관계

- 오브젝트는 서로의 부모-자식 관계가 될 수 있다.
 - 친자 관계로 엮어두면 게임 조작에 훨씬 편리하다. **parent 사용**

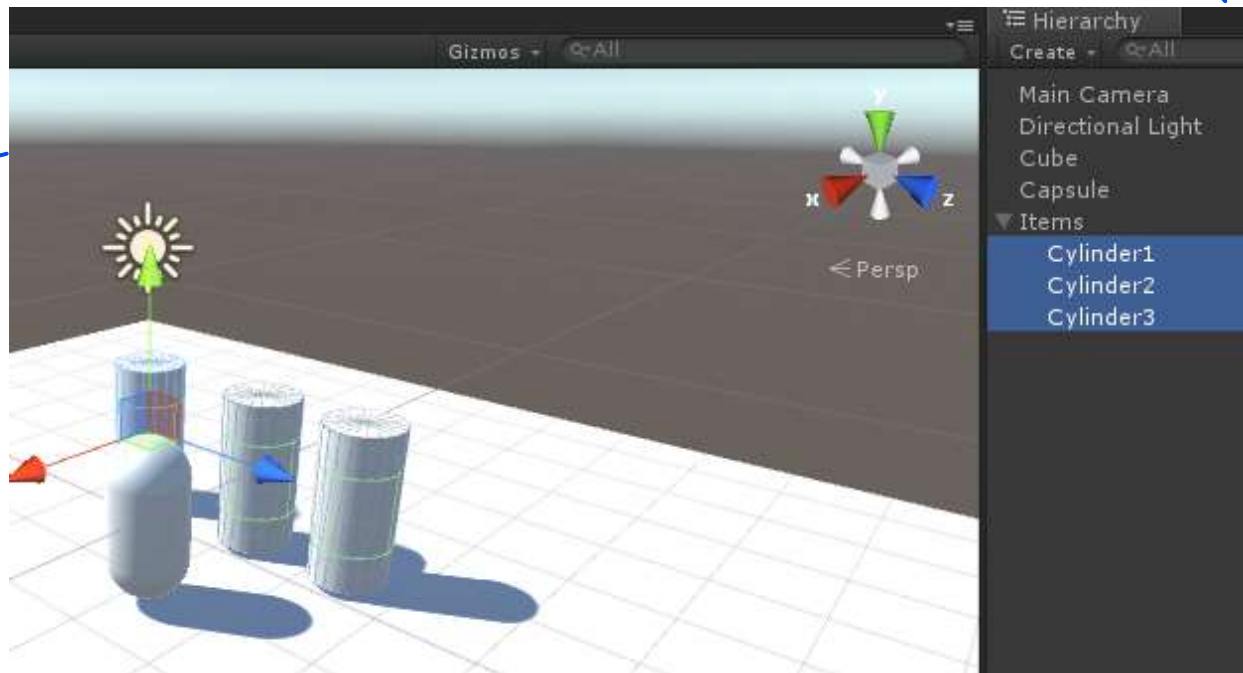


EX) 무기와 방패를 가진 캐릭터



오브젝트의 친자 관계

- 빈 게임 오브젝트를 폴더처럼 사용하기
 - (Item 창 만들 때 유용)
- GameObject → Create Empty 빈 오브젝트로 그룹화를 할 수 있다



참고로 빈오브젝트는
Transform 있다

Script 재사용

- 같은 처리를 여러 번 작성하는 수고를 덜 수 있다.

```
public class cubeScript : MonoBehaviour {  
    // Use this for initialization  
    void Start () {  
    }  
    // Update is called once per frame  
    void Update () {  
    }  
    public void bigsize(){  
        //x, y, z모든 방향으로 크기 3배  
        this.transform.localScale = new Vector3(3.0f, 3.0f, 3.0f);  
    }  
}
```

```
public class studyScript : MonoBehaviour {  
    // Update is called once per frame  
    void Update () {  
        if (Input.GetKey (KeyCode.G)) {  
            GameObject go = GameObject.Find ("Cube") as GameObject;  
            go.GetComponent<cubeScript> ().bigsize ();  
        }  
    }  
}
```

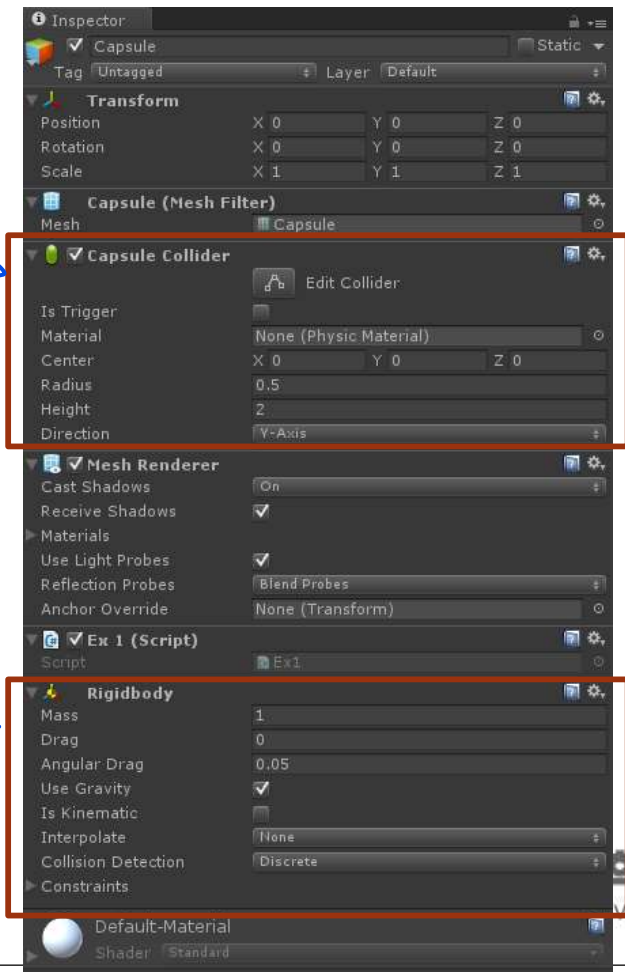
Rigidbody

- 게임 오브젝트에 물리적인 움직임을 부여하는 마법
- 리지드바디를 적용하면 오브젝트는 중력에 의해 낙하한다.
 - 즉, 중력 효과를 부여한다.

collider는 두 물체가 부딪혔는지 여부를 알아낸다.

rigid body를 Material에
붙여다 놓으면 된다

add component 해서
rigid body를 적용하면
중력이 적용된다



Physic Material

- 슈퍼마리오 게임을 생각해보자.
 - 단계마다 배경이 다른데, 벽돌 맵이 있는 가 하면 얼음 맵도 있다.
 - 이를 표현하기 위해 게임 오브젝트에서 **소재를 결정**할 수 있다.

소재는 파일쪽에서
생성할 수 있다

- GameObject → Create → Plane
 - 이름 : Floor
- Asset → Create → Physic Material
 - 이름 : BallPM
 - Bounciness : 1 탄성은 0 ~ 1, 1이 최댓값
 - Bounce Combine : Maximum

- Floor 에 Drag & Drop

Script에서 rigidbody 조작 *cs Physic.cs*

- 메서드 안에서 리지드바디를 조작할 수 있다.
 - '지정한 방향으로, 지정한 양의 힘을 가하는.'

```
void Update () {  
    if (Input.GetKey (KeyCode.UpArrow)) {  
        GetComponent<Rigidbody> ().AddForce (Vector3.forward * 300 * Time.deltaTime);  
    }  
    if (Input.GetKey (KeyCode.DownArrow)) {  
        GetComponent<Rigidbody> ().AddForce (Vector3.back * 300 * Time.deltaTime);  
    }  
    if (Input.GetKey (KeyCode.LeftArrow)) {  
        GetComponent<Rigidbody> ().AddForce (Vector3.left * 300 * Time.deltaTime);  
    }  
    if (Input.GetKey (KeyCode.RightArrow)) {  
        GetComponent<Rigidbody> ().AddForce (Vector3.right * 300 * Time.deltaTime);  
    }  
    if (Input.GetKey (KeyCode.U)) {  
        GetComponent<Rigidbody> ().AddForce (Vector3.up * 300 * Time.deltaTime);  
    }  
    if (Input.GetKey (KeyCode.D)) {  
        GetComponent<Rigidbody> ().AddForce (Vector3.down * 300 * Time.deltaTime);  
    }  
}
```

가속도 부여

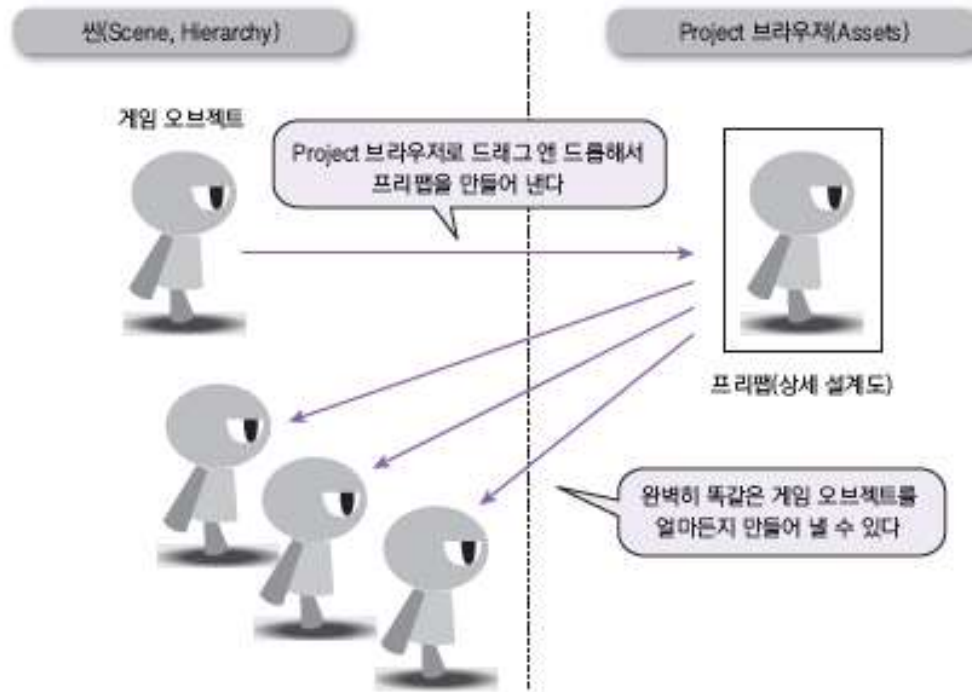
Script에서 rigidbody 조작

- 중력 제어
 - gravity 파라미터 이용

```
void Update () {  
    ... ..  
    if (Input.GetKeyDown (KeyCode.Keypad0)) {  
        Physics.gravity = Vector3.zero;  
    }  
    if (Input.GetKeyDown (KeyCode.Keypad8)) {  
        Physics.gravity = Vector3.up;  
    }  
    if (Input.GetKeyDown (KeyCode.Keypad2)) {  
        Physics.gravity = Vector3.down;  
    }  
}
```

프리팹(Prefab)

동일한 객체를 반복적으로 나타내야 할 때 쓴다 (ex. 총알 발사)



오브젝트를 반대로 파일쪽에 갖다놓으면 .prefab 으로 등록된다
(메모리에 잡아두고 여러번 반복하여 쓴다)

프리팹을 프로그램으로 배치하기

- 프리팹 복사를 통한 객체 생성
 - Sphere를 Prefab으로 설정
 - Floor 객체에 아래 스크립트를 등록

```
public GameObject prefab;
```

```
// Update is called once per frame
```

```
void Update () {
```

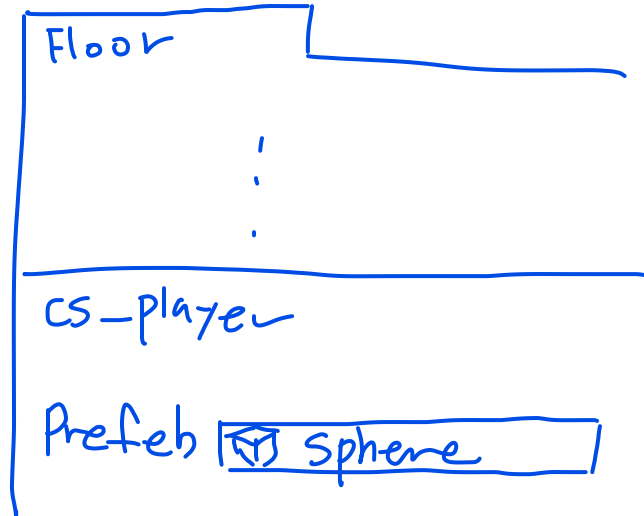
```
    if(Input.GetMouseButtonDown(0)){
```

```
        GameObject go = GameObject.Instantiate(prefab) as GameObject;
```

```
        go.transform.position = new Vector3(Random.Range(-2.0f, 2.0f), 1.0f, 1.0f);
```

```
    }
```

```
}
```



효과음 넣기

- 사운드에는 AudioSource가 필수!!

```
private AudioSource audio;  
public AudioClip clip;
```

```
// Use this for initialization
```

```
void Start () {  
    this.audio = this.gameObject.AddComponent<AudioSource> ();  
    this.audio.clip = this.clip;  
    this.audio.loop = false;  
}
```

```
// Update is called once per frame
```

```
void Update () {  
    if (Input.GetMouseButtonDown (0)) {  
        this.audio.Play ();  
    }  
}
```


OnGUI()

- 2D 표현
 - 사용 예) 체력 게이지, score.

```
public Texture2D icon = null;  
public static string mes_text = "test";
```

```
void OnGUI(){  
    GUI.DrawTexture (new Rect (Screen.width / 2, 64, 64, 64), icon);  
    GUI.Label (new Rect (Screen.width / 2, 128, 128, 32), mes_text);  
}
```

- 유니티에서 GameRoot 선택 – [GameRootScript] 컴포넌트에 새로 생긴 [Icon] 항목에
- 넣고 싶은 이미지를 드래그 앤 드롭

씬의 이동

- 게임에는 실행화면만 있는 것이 아니다.
- 타이틀화면이 있고, 결과를 나타내는 화면도 있다.
- 화면 간의 이동은 꼭 필요하다.
 - **먼저 씬을 저장하고 새로운 씬을 만들자.**

```
using UnityEngine.SceneManagement;
```

```
void Update () {  
    if (Input.GetMouseButtonDown(0)) {  
        //Application.LoadLevel (0);  
        SceneManager.LoadScene(0);  
    }  
}
```

