

유니티

게임 제작 입문

프로그래밍 기초

프로그래밍 기초

Level 1

변수

변수의 연산

조건문

반복문

1. 변수

어떤 숫자나 문자열을 저장할 수 있는 상자

자료형 : 변수의 형태

ex) int - 정수형, float - 실수형, string - 문자열, bool - 참/거짓 판정

▼ 그림 1-25 변수의 형태

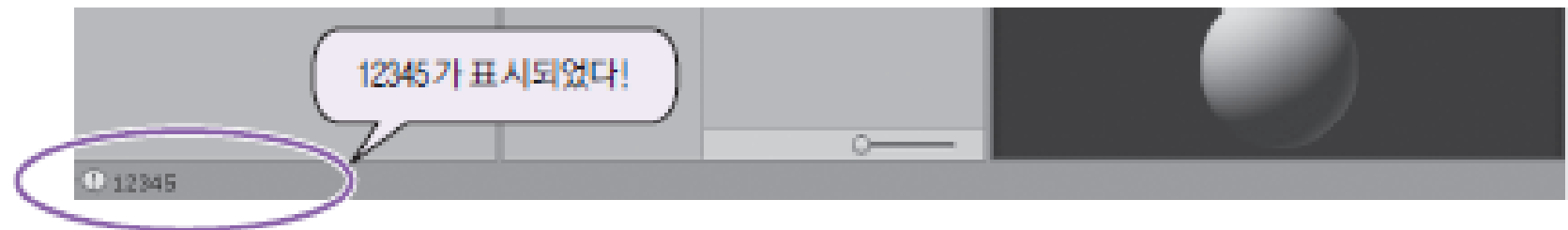


1-1. 변수 사용하기 (정수형)

studyScript.cs

```
void Start() {  
    int hako1;           // hako1이라는 int형 변수를 준비한다. 정수형  
    hako1 = 12345;       // hako1에 12345라는 값을 넣는다.  
    Debug.Log(hako1);    // hako1의 내용을 디버그 로그로 표시한다.  
}
```

♥ 그림 1-26 변수의 내용이 표시되었다



1-1. 변수 사용하기 (실수형)

studyScripts.cs

```
void Start() {  
    float hako2;      // hako2라는 float형 변수를 준비한다. 실수형  
    hako2 = 12.345f;   // hako2에 12.345f라는 값을 넣는다.  
    Debug.Log(hako2); // hako2의 내용을 디버그 로그로 표시한다.  
}
```

▼ 그림 1-27 소수점 값이 표시되었다



1-1. 변수 사용하기 (문자열)

studyScripts

```
void Start() {  
    string hako3;           // hako3이라는 string형 변수를 준비한다. 문자열  
    hako3 = "ABCDE";        // hako3에 "ABCDE"라는 값을 넣는다.  
    Debug.Log(hako3);       // hako3의 내용을 디버그 로그로 표시한다.  
}
```

▼ 그림 1-28 문자열이 표시되었다



2. 변수의 특성

변수의 내용은 원하는 대로 바꿀 수 있다.

```
studyScripts
void Start() {
    int hako1;
    hako1 = 1;
    Debug.Log(hako1); // 표시되는 값은 1.

    hako1 = 2;
    Debug.Log(hako1); // 표시되는 값은 2.

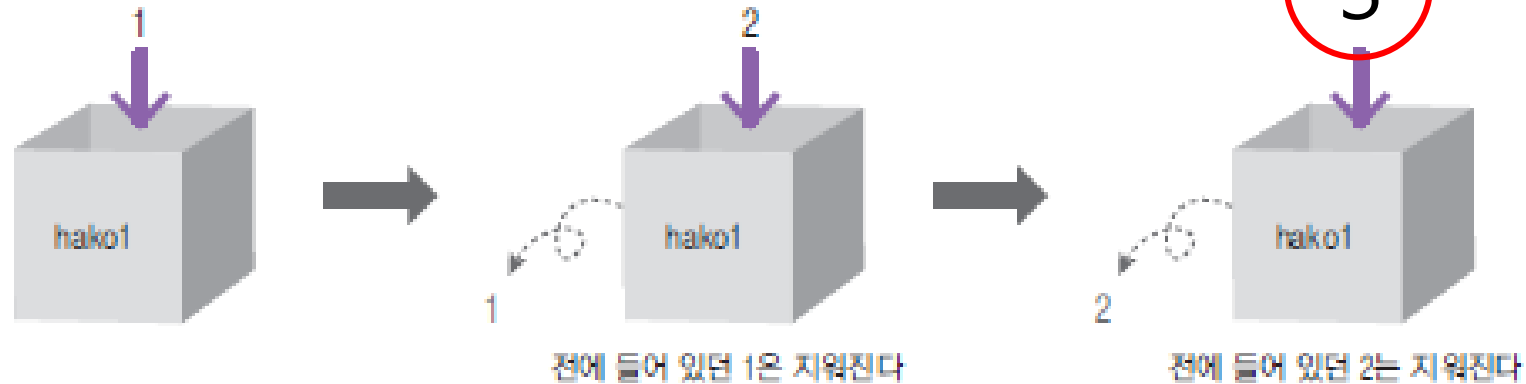
    hako1 = 3;
    Debug.Log(hako1); // 표시되는 값은 3.
}
```

위 코드를 실행했을 때 hako1에 저장된 값은?

2. 변수의 특성

변수의 내용은 원하는 대로 바꿀 수 있다. => **대입**

▽ 그림 1-29 덮어쓰기 되는 변수



2. 변수의 특성

변수에 변수를 대입할 수 있다

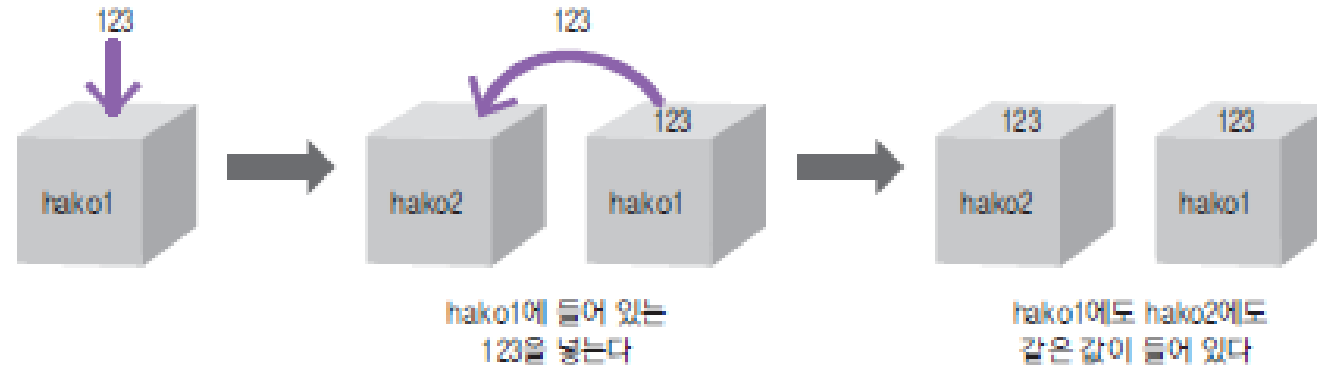
```
studyScripts
```

```
void Start() {  
    // 변수를 두 개 준비한다.  
    int hako1;  
    int kako2;  
  
    hako1 = 123;           // hako1에 숫자 123을 넣는다.  
    hako2 = hako1;         // hako2에는 hako1의 내용 123을 넣는다.  
    Debug.Log(hako2);     // () 안에 hako2를 지정하면 123이 표시된다.  
}
```

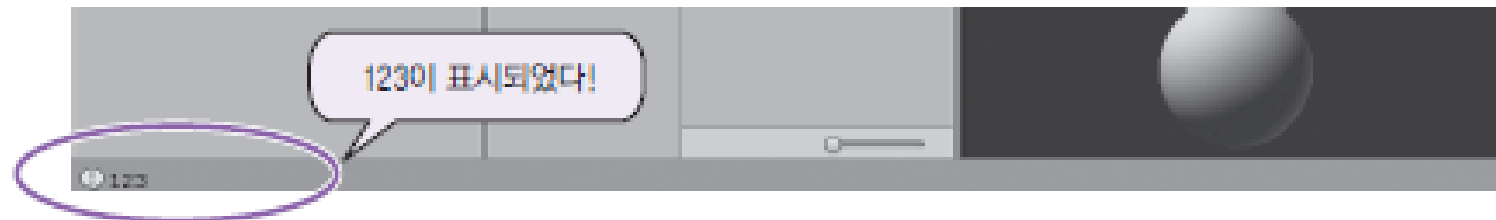
2. 변수의 특성

변수에 변수를 대입할 수 있다

♥ 그림 1-32 변수의 내용을 대입



♥ 그림 1-31 변수를 변수에 대입할 수도 있다



3. 변수의 연산

1) 사칙연산

$+$, $-$, $*$, $/$ 를 이용해 산술연산을 할 수 있다.

덧셈

studyScripts

```
void Start() {  
    int hako;  
    hako = 1 + 2; // 1+2의 계산 결과인 3이 들어간다.  
    Debug.Log(hako);  
}
```

뺄셈

studyScripts

```
void Start() {  
    int hako;  
    hako = 3 - 1; // 3-1의 계산 결과인 2가 들어간다.  
    Debug.Log(hako);  
}
```

3. 변수의 연산

1) 사칙연산

$+$, $-$, $*$, $/$ 를 이용해 산술연산을 할 수 있다.

곱셈

```
studyScripts
void Start() {
    int hako;
    hako = 2 * 3; // 곱셈에는 x가 아니라 *를 사용한다. // 2*3의 계산 결과인 6이 들어간다.
    Debug.Log(hako);
}
```

나눗셈



```
studyScripts
void Start() {
    int hako;
    hako = 10 / 2; // 나눗셈에는 ÷가 아니라 /를 사용한다. // 10÷2의 계산 결과인 5가 들어간다.
    Debug.Log(hako);
}
```



곱셈과 나눗셈 연산을 할 때는 자료형을 주의해야 한다.

3. 변수의 연산

2) 증감연산

`++`, `--`로 증가, 감소연산을 쉽게 할 수 있다.

<code>hako = 1;</code>		<code>hako = 1;</code>		<code>hako = 1;</code>
<code>hako++;</code>		<code>hako = hako + 1;</code>		<code>hako += 1;</code>

<code>hako = 1;</code>		<code>hako = 1;</code>		<code>hako = 1;</code>
<code>hako--;</code>		<code>hako = hako - 1;</code>		<code>hako -= 1;</code>

3. 변수의 연산

3) 누적연산

복합대입연산자를 통해 결과를 추가할 수 있다.

```
hako = 1;  
hako += 10;
```

11

```
hako = 5;  
hako -= 1;
```

4

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

40

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

4

4. 조건문

지정한 조건에 맞으면 프로그램 구문을 실행한다. **if문 사용**

```
studyScripts
```

```
void start() {
```

```
    int hako;
```

```
    hako = 1;
```

```
    if(hako == 1) { // hako의 값이 1이면.
```

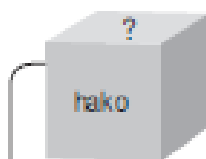
```
        Debug.Log(hako);
```

```
    }
```

```
}
```

=가 2개 사용되는 점에 주의

▼ 그림 1-33 if문



만약 hako의 내용이 1이라면 아래처럼 처리한다

디버그 로그에 hako의 내용을 표시한다

```
(!) 1
```

1이 아니면 출력되지 않는다.

4. 조건문

지정한 조건에 맞으면 프로그램 구문을 실행한다. **if문 사용**

```
studyScripts
```

```
void start() {
```

```
    int hako;
```

```
    hako = 1;
```

```
    if(hako == 1){ // hako의 값이 1이면.
```

```
        Debug.Log(hako);
```

```
    }
```

```
}
```

=가 2개 사용되는 점에 주의

hako == 1	hako가 1이면
hako >= 1	hako가 1 이상이면
hako <= 1	hako가 1 이하이면
hako != 1	hako가 1이 아니면

4. 조건문

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

studyScript.cs

```
void Start() {  
    int hako = 1;  
  
    if(hako == 1) { // hako의 값이 1이면.  
        Debug.Log("1이야");  
    } else { // 그렇지 않으면.  
        Debug.Log("1이 아니야");  
    }  
}
```

4. 조건문

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

studyScript.cs

```
void Start() {  
    int hako = 1;  
  
    if(hako == 1) {          // hako의 값이 1이면.  
        Debug.Log("1이야");  
    } else if(hako == 2) {  // 그렇지 않고 2라면.  
        Debug.Log("1이 아니라 2야");  
    } else {                // 모두 해당되지 않으면.  
        Debug.Log("1도 2도 아니라 3이야");  
    }  
}
```

else에 if를 더해서 여러 조건에
적용시킬 수 있다.

5. 반복문

조건에 일치한다면 같은 처리를 반복한다.

switch ~ case : 변수의 값을 보고 case 중 일치하는 것을 실행한다.

studyScript.cs

```
void Start() {
    int hako = 1;

    switch(hako) {
        case 0: // hako가 0일 때.
            Debug.Log("0이다");
            break; // switch문 밖으로 탈출.
        case 1: // hako가 1일 때.
            Debug.Log("1이다");
            break; // switch문 밖으로 탈출.
        case 2: // hako가 2일 때.
            Debug.Log("2다");
            break; // switch문 밖으로 탈출.
        default: // 위 조건 어디에도 해당하지 않을 때.
            Debug.Log("0, 1, 2 모두 아니다");
            break; // switch문 밖으로 탈출.
    } // ← break로 인해 이곳으로 탈출.
}
```

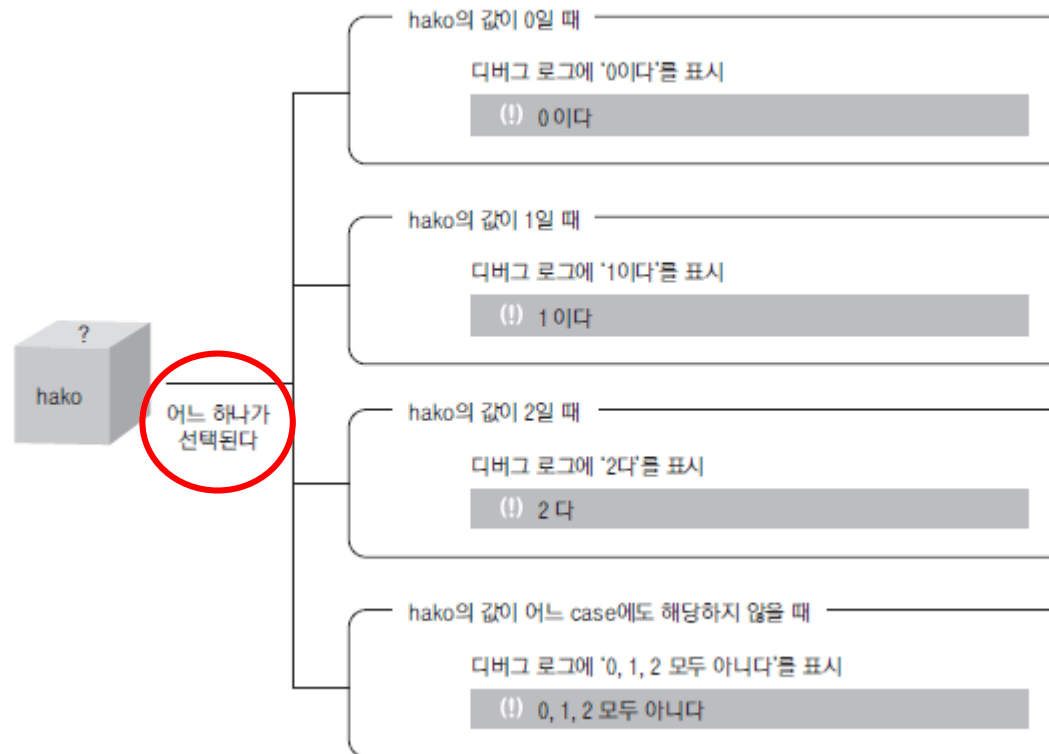
주의사항

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**

5. 반복문

switch ~ case : 변수의 값을 보고 case 중 일치하는 것을 실행한다.

♥ 그림 1-35 switch~case문



5. 반복문

for문 - 같은 처리를 반복할 때 사용

기본형

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

ex)

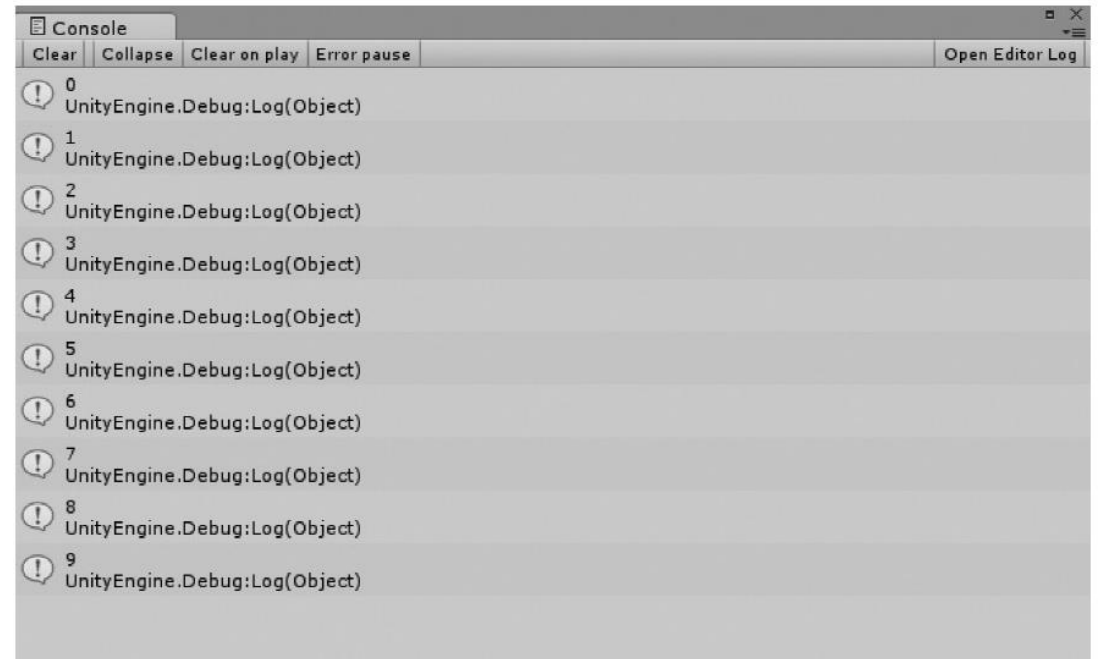
```
for(int i = 0; i<10; i++) {  
    Debug.Log(i);  
}
```

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

5. 반복문

for문 - 같은 처리를 반복할 때 사용

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



프로그래밍 기초 Level 2

배열

변수의 유효범위

메서드

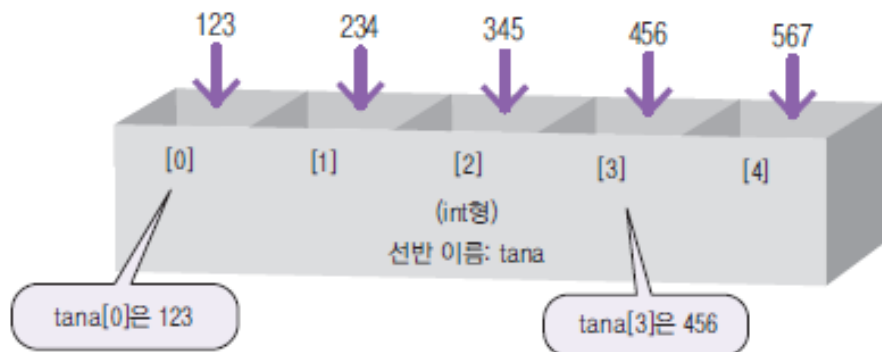
클래스

오류해결방법

입력받기

1. 배열

▼ 그림 1-38 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 0으로 시작

작성할 때는 변수형[] ex) int array[5]

1. 배열

배열 선언하는 방법 1. 선언과 동시에 값을 넣어준다.

studyScript.cs

```
void Start() {  
    int[] tana = { 123, 234, 345, 456, 567 };  
  
    // 0번지에서 요소의 개수 미만(최종 번지)까지 반복한다.  
    for( int i = 0; i < tana.Length; i++ ) {  
        Debug.Log(tana[i]); // 루프 변수를 [] 안에 지정한다.  
    }  
}
```

배열 선언하는 방법 2. 빈 배열을 선언 후 나중에 값을 넣어준다.

studyScript.cs

```
void Start() {  
    int[] tana = new int[5]; // int가 다섯 개 들어가는 배열을 작성한다.  
  
    Debug.Log(tana[0]); // 아직 값이 들어 있지 않으므로 0이 출력된다.  
    tana[0] = 32; // 0번지에 값을 넣는다.  
    Debug.Log(tana[0]); // 이번에는 32가 출력된다.  
}
```

1. 배열

배열의 특성 - 다른 배열의 값을 대입할 수도 있다.

studyScript.cs

```
void Start() {  
    int[] tana1 = { 123, 234, 345, 456, 567 };  
    int[] tana2 = new int[5];  
  
    tana2 = tana1;           // 배열 tana2에 배열 tana1을 통째로 대입한다.  
    Debug.Log(tana2[0]);    // 123이 출력된다.  
}
```

tana2 배열에는 tana1 배열의 값이 순서대로 저장되어 있을 것이다.

1. 배열

foreach : 배열의 모든 요소를 표시

studyScript.cs

```
void Start() {  
    int[] tana = { 123, 234, 345, 456, 567 };  
  
    // 0번지에서 요소의 개수 미만(최종 번지)까지 반복한다.  
    for( int i = 0; i < tana.Length; i++ ) {  
        Debug.Log(tana[i]); // 루프 변수를 [] 안에 지정한다.  
    }  
}
```

for문을 이용해서
배열의 값을 표시할 수 있다.

studyScript.cs

```
void Start() {  
    int[] tana = { 123, 234, 345, 456, 567 };  
  
    foreach( int i in tana ) { // i에는 tana의 모든 요소가 차례로 들어간다.  
        Debug.Log(i);  
    }  
}
```

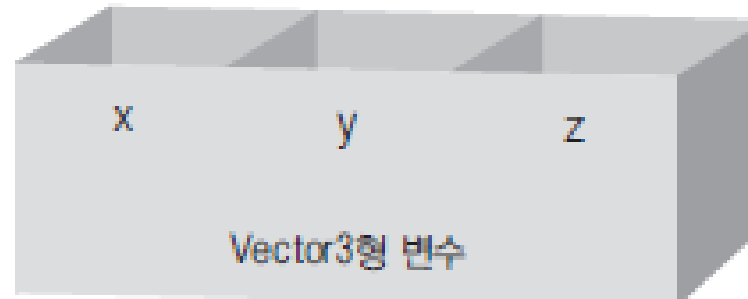
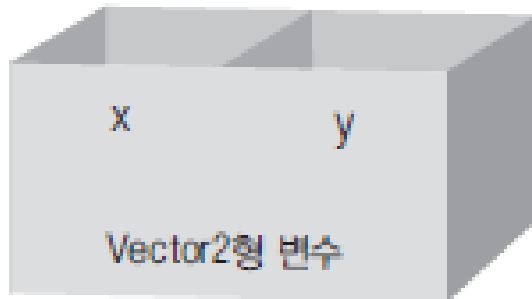
하지만 유니티에서는
foreach로 더 간편하게 모든 값을 표시할 수 있다

2. 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.

♥ 그림 1-39 Vector2와 Vector3



2. Vector형 변수

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

studyScript.cs

```
void Start() {  
    Vector3 pos;           // Vector3형 변수 pos를 만든다.  
  
    // pos에 이 스크립트가 설치된 게임 오브젝트의 위치 정보를 넣는다.  
    pos = this.gameObject.transform.position;  
  
    Debug.Log(pos);        // (0.0, 0.0, 0.0)처럼 x, y, z값이 출력된다.  
    Debug.Log(pos.y);      // 0처럼 y값이 출력된다.  
}
```

3. 변수의 유효범위

유효범위 : 변수가 참조할 수 있는 범위

▼ 그림 1-40 변수의 유효 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

3. 변수의 유효범위

studyScript.cs

```
using UnityEngine;
using System.Collections;

public class studyScript: MonoBehaviour {
    public int gomibako1 = 0;    // public인 변수.
    private int gomibako2 = 1;  // private인 변수.

    void Start() {
        Debug.Log(gomibako1);
        Debug.Log(gomibako2);
    }

    void Update() {
    }
}
```

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

▼ 그림 1-42 대합실의 휴지통인가. 역장실의 휴지통인가

public인 변수(누구나 볼 수 있고 사용할 수 있다)

private인 변수(밖에선 보이지 않고 사용할 수 없다)



4. 메서드

studyScript.cs

```
void Start() {
    int damage = hissatuwaza();
    Debug.Log(damage + "데미지!");
}
```

이 괄호 안에 start 메서드에서 인수를 받아 처리에 이용할 수 있다.

ex) int hissatuwaza(int a)

```
int hissatuwaza() {
    Debug.Log("몸통 박치기!");
    int power = 1200;
    return power; // 호출한 쪽으로 값을 반환한다.
}
```

int 데이터를 호출한 쪽에 반환하는 메서드
hissatuwaza()를 작성한다

반환값

Method - 위 코드에서 hissatuwaza와 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.

5. 클래스와 로컬 변수, 멤버 변수

선언되는 위치에 따라 로컬변수, 멤버변수로 나눌 수 있다.

```

studyScript.cs
public class studyScript: MonoBehaviour {
    void Start() {
    }

    int score = 1234;           // 멤버 변수.

    void Update() {
        int memo1 = 567;       // Update() 안에서만 유효한 로컬 변수.

        score++;
        memo1++;

        if(memo1 > 100) {        // memo1이 100보다 클 때만.
            int memo2 = 89;     // 이 if문 안에서만 유효한 로컬 변수.
            memo2++;
        }                       // 프로그램 처리가 여기에 도달한 시점에서 memo2는 사라진다

        Debug.Log(score);
        Debug.Log(memo1);       // 이곳은 if문 밖이므로 이미 memo2는 없다. 주석처리를 벗기면
        // Debug.Log(memo2);    // 존재하지 않는 변수를 참조하려 하므로 오류가 생긴다
    }                           // 프로그램 처리가 여기에 도달한 시점에서 memo1도 지워진다. score는 지워지지 않는다
}

```

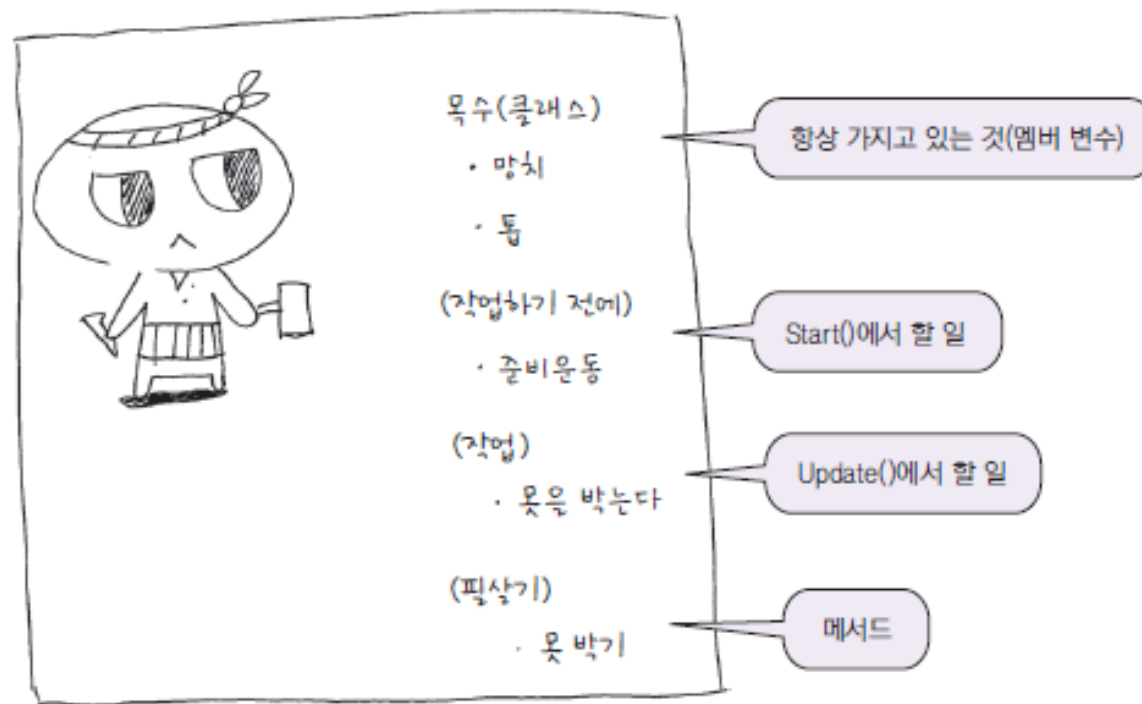
(=전역변수) 클래스가 사라지지 않는 이상
계속 사용할 수 있는 변수
즉, start 안에 선언되었으니 update 뿐 아니라
다른 메서드에서도 사용할 수 있다.

update() 메서드에서만 사용 가능하다.
다른 메서드에서 사용하고 싶다면
일일이 선언해주어야 한다.
이 지역에만 한정되어 있으니 로컬 변수!

5. 클래스와 로컬 변수, 멤버 변수

클래스 : 메서드와 변수를 하나로 묶은 것. 큰 울타리라고 생각하면 쉽다.

▼ 그림 1-44 목수(daiku) 클래스



5. 클래스와 로컬 변수, 멤버 변수

studyScript.cs

```
using UnityEngine;
using System.Collections;

public class oyaji {
    // oyaji 클래스(오야지라는 작업을 나타낸다)를 만든다. 자발적으로 움직이는
    // 클래스가 아니므로 뒤에 'MonoBehaviour'는 필요 없다
    public int hungry = 0; // 배고플 때 값.
    public int sleepy = 0; // 졸릴 때.

    public void akubi() { // 하품 메서드(필살기).
        Debug.Log("하품을 한다");
    }
}

public class studyScript: MonoBehaviour {
    // daiku 클래스(작업)를 만든다. 단 이번에는 studyScript로 대신한다
    private int tonkati = 1; // 망치를 하나 가지고 있음.
    private int kugi = 100; // 못을 1개 가지고 있음.

    // oyaji 클래스의 인스턴스를 'oyj'라는 이름으로 작성.
    private oyaji oyj = new oyaji();

    void Start() {
        // 생성자
        oyj.akubi(); // oyj의 하품하는 akubi 메서드 실행.
        Debug.Log("준비운동");
    }

    void Update() {
        kugiuchi();
    }

    void kugiuchi() { // 못 박기 메서드(필살기).
        Debug.Log("못을 박는다");
    }
}
```

* 클래스와 스크립트 명의 관계

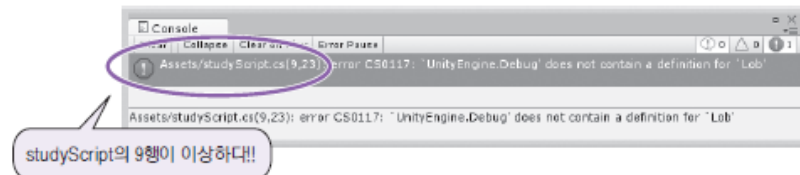
스크립트 이름과 같은 클래스는 꼭 존재하여야 한다.

6. 오류가 발생한 경우

1) 오류메시지를 참고하라

: 오류 발생위치를 알려준다. 오류가 난 곳을 찾아 수정하도록 한다.

▼ 그림 1-47 오류가 표시되면 오류가 발생한 곳을 알려준다



2) 수상한 곳에 Debug.Log를 설치한다

: 실행은 되지만 어딘가 이상할 때, 그 부분에 Debug.log를 끼워 넣은 후 제대로 돌아가는 지 확인한다.

```
int hako = 1;
void Start() {
    hako++;
    Debug.Log("check1: " + hako);
    if(hako > 2) {
        Debug.Log("2 이상이다");
        Debug.Log("check2: " + hako);
    }
    Debug.Log("check3: " + hako);
}
```

7. 키보드와 마우스 입력받기

1) 키보드 입력

“**Input.동작(키코드)**”의 형태로 사용한다.

Input.GetKey : 키가 눌린 상태

Input.GetKeyDown : 키가 눌린 순간

Input.GetKeyUp : 키에서 손이 떨어진 순간

Input.AnyKeyDown : 아무 키라도 눌려라!

▼ 표 1-1 키코드

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

활용 예)) **Input.GetKeyDown** : 키가 계속 눌린 상태

studyScript.cs

```
void Update() {
    if(Input.GetKeyDown(KeyCode.Space)) {    // 스페이스바가 눌린 상태라면.
        Debug.Log("space!");
    }
}
```

7. 키보드와 마우스 입력받기

2) 마우스 입력

키보드 입력과 같다. Key를 Mouse로 바꿔 사용하면 된다.

Input.GetMouseButton : 마우스가 눌러있는 동안

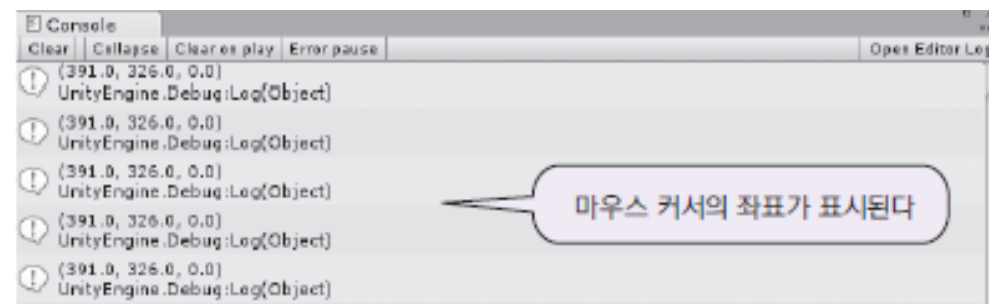
Input.GetMouseUp : 마우스에서 손이 떨어졌을 때

Input.AnyMouseDown : 클릭하는 순간

활용 예))

Input.MousePosition : 마우스 포인터의 위치를 구한다.

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



유니티 게임 제작 입문

다음 페이지에서 보아요