1. 변수

이번 첫 강좌에서는 GML 중 기초이고 제일 중요한 **변수**에 대해서 설명하겠습니다.

변수는 **과일들을 담는 상자**와 같습니다. 근데 이 상자에게는 이름을 붙입니다.
상자들을 주인 마음대로 구별하려고요!

초등학교 생활을 거치신 분은 더하기, 빼기, 곱하기, 나누기의 개념을 배웠을 겁니다.
우리는 그것을 이용해서 상자 안에 들어있는 과일 개수를 조절하도록 컴퓨터한테 시킬 수 있어요!
컴퓨터를 통해서 상자의 과일 개수들을 만지작만지작 할 수 있는 공식이 있습니다.

변수이름 = **숫자**;
[상자 이름] [과일 개수]

이것만 하면 **과일 개수가 몇 개 있는 상자**를 **창조**할 수 있습니다!
아 근데 유의할 점이 있습니다. 변수이름은 한글이 되면 안 되고, 영어와 _(언더바) 그리고 숫자를 적을 수 있는데,
1Apple_Box이렇게는 못쓰고 **Apple1_Box** 처럼 숫자를 맨 앞에 두면 안 됩니다.

근데 창조만 할 건가요? 이제 과일 개수를 조절해야죠.

```
변수이름 += 숫자; //더하기
변수이름 -= 숫자; //빼기
변수이름 /= 숫자; //나누기
변수이름 *= 숫자; //곱하기
```

간단하게는 이렇게 하지만, 아래와 같은 방법도 있습니다.

```
변수이름 = 변수이름 + 숫자; //더하기
변수이름 = 변수이름 - 숫자; //빼기
변수이름 = 변수이름 / 숫자; //나누기
변수이름 = 변수이름 * 숫자; //곱하기
```

즉 **변수이름** = **숫자**; 를 **변수이름** = **표현식**; 으로도 쓸 수 있다는 건데,
표현식에서 변수이름을 쓰면 컴퓨터가 변수이름에 저장되있는 숫자를 대입해줍니다.
오! 표현식이면 ()괄호도 되겠네요? 한번 저와 변수 창조를 해볼까요!

```
Apple = Apple_box*(500+64);
Apple_box = 100;
```

이렇게 한번 써보셨나요? 어……. 직접 해보면 에러가 나는데 왜 그럴까요!
GML은 **맨 위에**서 **아래로 순서대로 처**리합니**다**. 그리고 표현식에서 처리를 할 때 그 표현식 안에 있는 변수를 창조하지도 않았는데, 변수 이름을 넣는다면? 에러가 나겠죠. 이게 무슨 소린지 모르겠다면 좀 전에 말한 "**과일 개수가 몇 개 있는 상자**를 **창조**" 이걸 잘 보셔야 합니다. "**변수이름 = 숫자**" 같은 형태는 **변수를 창조하고, 값을 넣습니다**.

즉, 변수를 창조하지도 않았는데 그 변수를 표현식에서 불러들이면 그 변수는 존재하지도 않는데 뭘 가지고 오라는 걸까요? 그래서 게임메이커는 에러를 내뿜습니다.
그러므로 위의 코드를 바꾸면

```
Apple_box = 100;
Apple = Apple_box*(500+64);
```

이와 같이 하면 에러가 발생하지 않습니다!
아, 그리고 변수 이름을 표현식에 넣으면 실제로 컴퓨터가 받아들이는 것은 그 변수 이름의 값입니다.

그래서 위에 **Apple = Apple_box*(500+64);** 있는데, 이것을 컴퓨터가 받아들이는 것은
Apple = 100*(500+64); 라는 말입니다.

2. 변수의 주소

여러분도 집 주소가 있지만, 변수도 주소가 있습니다.
이번 시간에는 그 변수의 주소를 쓰는 방법을 배워봅시다!
배우실 때 직접 게임메이커 코드 창에서 쓰고 플레이 해보시면서 강좌를 봐주세요!

C언어 배워보신 분은 이 변수의 주소를 메모리 주소로 생각하지 말아주세요.
게임메이커에서 **변수의 주소**란 **각 인스턴스에 존재하는 id를 이용해 변수를 불러오는 경로**를 말합니다.

[PS. 인스턴스id란 각 인스턴스(오브젝트 설계도로 만들어진 객체)마다 다르게 존재하는 값입니다]

변수의 주소 표기법은 간단합니다.

인스턴스ID.변수이름

인스턴스는 오브젝트(설계도)로 만들어진 것(생산품)을 말합니다.
이 인스턴스에는 모두 id(주소값)이 존재합니다.
또한 이 id를 다른 인스턴스에서 이용해서 변수를 조작하거나 명령을 하는 등의 일을 할 수 있습니다.

이제 **인스턴스주소값**에는 무엇을 써서 주소를 알려주어야 하는가!
직접 **id**를 적으시거나, **변수**를 이용해서 적으시거나, **오브젝트 이름**을 적으시면 됩니다.
하지만 이 인스턴스주소값을 사용할 때 주의할 점 두가지가 있습니다.

- 오브젝트 이름을 쓸 경우 그 오브젝트 이름을 가진 인스턴스들 여러 개 중 한개만 생성된 것만 쓰게 됩니다.
- 인스턴스주소값을 사용했는데 그 인스턴스가 존재하지 않으면, 에러가 생기게 됩니다!

이럴 경우 변수 선언 순서를 바꾸거나, **instance_exists** 함수를 사용하는데 아직 함수 개념을 배우지 않았으니 패스

그러면 이것을 어떻게 써먹어야하나!
지난 시간에 이런 것들을 배웠습니다.

```
변수이름 = 숫자; //변수 창조 및 값 넣기
변수이름 += 숫자; //더하기
변수이름 -= 숫자; //빼기
변수이름 /= 숫자; //나누기
변수이름 *= 숫자; //곱하기
변수이름 = 변수이름 + 숫자; //더하기
변수이름 = 변수이름 - 숫자; //빼기
변수이름 = 변수이름 / 숫자; //나누기
변수이름 = 변수이름 * 숫자; //곱하기
```

이제 방금 배운 **인스턴스주소값.변수이름**을 아래처럼 쓰시면 됩니다.

```
인스턴스ID.변수이름 = 숫자; //변수 창조 및 값 넣기
인스턴스ID.변수이름 += 숫자; //더하기
인스턴스ID.변수이름 -= 숫자; //빼기
인스턴스ID.변수이름 /= 숫자; //나누기
인스턴스ID.변수이름 *= 숫자; //곱하기
인스턴스ID.변수이름 = 인스턴스ID.변수이름 + 숫자; //더하기
인스턴스ID.변수이름 = 인스턴스ID.변수이름 - 숫자; //빼기
인스턴스ID.변수이름 = 인스턴스ID.변수이름 / 숫자; //나누기
인스턴스ID.변수이름 = 인스턴스ID.변수이름 * 숫자; //곱하기
```

헉……. 실제로 이렇게 써보신 분은 깨달은 점이 있을 겁니다.
내가 진짜 인스턴스ID를 반복해서 써야하나?

해결방법은 **with문**이지만 12강에서 설명하겠습니다!

이제 실습을 해봅시다.

(상황: **obj_player, obj_system** 라는 오브젝트가 있다고 하고, 이 오브젝트로 인스턴스를 만들었는데 그 인스턴스주소값은 **111111**이고, 그 인스턴스주소값을 변수 **player_id** 에 저장했습니다. 근데 **obj_system** 인스턴스가 **obj_player** 인스턴스에서 **Apple** 변수의 값을 불러와 **Apple_Box** 변수에 저장해야합니다.)

상황 이해되셨나요? 이 상황에서 저장 방법은 아직 배운 것 중에서는 3가지 방법이 있습니다.
여러분도 스스로 작성해보시고 맞는지 확인해보시기 바랍니다.

1) Apple_Box = 111111.Apple;
2) Apple_Box = player_id.Apple;
3) Apple_Box = obj_player.Apple;

3. 배열

변수에는 액셀의 행과 열처럼 구성되는 배열이 존재합니다.
변수를 쓸 때 아래와 같은 경우가 있습니다.

```
apple_box0 = 1;
apple_box1 = 2;
...
```

이렇게 하면 나중에 반복문을 사용할 때 사용할 수 없습니다.

```
apple = 2;
apple_boxapple = 1;
```

이런다고 apple_boxapple이 apple_box2가 되는건 아니잖아요?
배열을 쓰면 비슷하게 쓸 수 있습니다.

```
apple = 2;
apple_box[apple] = 1;
```

배열의 형태는 이렇습니다.

```
변수이름[숫자1] = 숫자; //1차 배열 (예를 들면 좌표에서 x 좌표값만 존재하는 형태 [1],[3],[6])
변수이름[숫자1, 숫자2] = 숫자; //2차 배열 (예를 들면 좌표값 x, y를 포함한 형태 [1,2],[3,5],[6,4])
```

[ ] 안의 숫자 위치를 부르는 명칭이 없어서 **숫자1**, **숫자2**라고 하겠습니다.
위에 제가 쓴 주석을 보시는 것처럼, **변수이름[숫자1]** 이러한 형태를 **1차 배열**이라 하고, **변수이름[숫자1, 숫자2]**와 같은 형태를 **2차 배열**이라 합니다.

이제 특정 배열의 수를 바꾸는 방법을 설명해드리겠습니다.

이것은 for문을 이용한 배열 선언입니다.
for문은 반복문이라는 것인데, 10강에서 나옵니다!

```
for(i = 0; i <= 3; i += 1){
  for(j = 0; j <= 5; j += 1){
   Apple[i, j] = 5;
 }
}

Apple[2,3] = 2;
```

제가 [3,5]크기의 Apple 배열을 만들었습니다.
그리고 저는 그 중에서 Apple[2,3] 배열의 값을 2로 바꿨습니다.

4. 함수

게임메이커에서 함수는 없어서는 안 되는 것입니다.
그래서 게임메이커를 잘하려면 **도움말의 함수를 외워두는 것**이 최고로 좋아요!
지금부터 그 함수를 알아보죠.

게임메이커에서 함수는 게임메이커 안에 내장되어있는 함수들(컴퓨터에게 할 명령들)을 말합니다.
함수는 컴퓨터에게 할 명령이라고 했는데, 함수 사용 형태를 알아보고 정말 맞는지 간단한 실습을 해보죠!
함수 사용 형태는 이렇습니다.

<함수 이름>(<인자1>, <인자2>,,,);
//<>는 대입하라는 표시

뭔가 어렵게 쓴 듯하지만, 예를 들어보면 show_message("Hello Guys"); 이렇습니다.
show_message("~") 함수는 강좌에서 말했지만, 인자1의 값을 윈도우창으로 출력합니다.
위에 인자1, 인자2라고 쓰여 있는데, **인자**는 인자함을 말하는 것이 아니고, 컴퓨터에게 명령할 때 필요한 값을 전달해주는 것입니다.

그러면 게임메이커 직접 이렇게 써보고 적용해보세요!
일단 오브젝트를 만들고, 이벤트 추가-드로우(Draw)-드로우(Draw) 누르시고, 코드 액션을 추가해서 아래와 같은 코드를 작성해봅시다.

```
draw_text(16,16,"Hello Guys");
```

이렇게 쓰고, 룸을 만들고 오브젝트를 룸에 하나만 생성시키고 게임을 실행하면 Hello Guys 라는 글자를 보실 수 있을 겁니다!

이 draw_text(~) 함수의 사용법 알려드릴게요!

draw_text(X좌표, Y좌표, 문자열);

이 함수를 보면 궁금한 거 생길꺼에요!
문자열이라고 인자 이름을 제가 썼는데, 그러면 "Hello Guys" 말고 Hello Guys 라고 써야하는 것 아닐까요?
이렇게 생각하실 수 있는데, 프로그래밍에서는 그냥 Hello Guys라고 쓰면 변수로 오해할 수 있기에,
"텍스트" 큰따옴표로 텍스트를 감싸서 표현합니다.

함수 사용법을 알려드렸으니, 인자를 바꿔서 써보세요!
그리고 인자에는 숫자나 문자열 대신 변수를 쓸 수 있습니다!

```
apple = 5;
draw_text(16,16,apple);
```

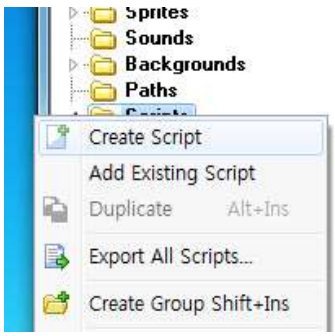
또한 문자열에는 앞으로 6강에서 배운 것을 활용할 수 있습니다!
혹시 제가 까먹을 수 있으니 잠깐 보고 넘어가시길. draw_text(16,16,"I have "+"three "+"apples");

5. 스크립트

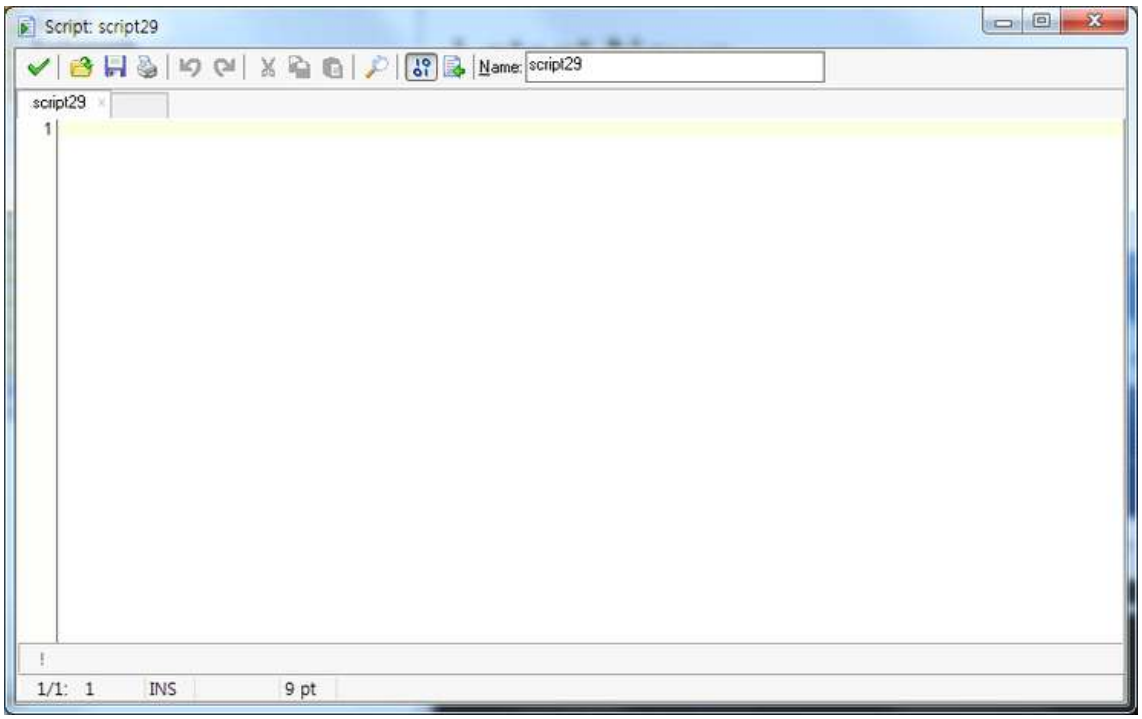
이번에는 직접 지난 강좌에 배운 함수를 만들 수 있는 방법을 배워보도록 하겠습니다.
먼저 설명을 듣기 전에 argument와 return을 기억하세요!
아 그리고, argument는 GML에서는 "논쟁"이란 뜻이 아니고 "인자"라는 뜻입니다!
인자의 뜻은 지난 강좌에서 설명했지만, 인자함을 말하는 것이 아니고,
컴퓨터에게 명령할 때 필요한 값을 전달해주는 것입니다.

또 용어에 대한 혼란이 올 수 있는데, **함수** = **스크립트** 입니다.

스크립트에는 argument, return이 중요하긴 해도 필요 없다면 쓰지 않아도 됩니다.
한번 컴퓨터에게 명령할 스크립트를 작성해보도록 할까요!



Scripts 폴더에 새로 스크립트를 만들어주시고,
그 만든 스크립트를 더블 클릭하시면



이러한 코드 창을 보실 수 있습니다!

이 코드창 위쪽에는 평소 코드창에서 보지 못했던 Name이라는 부분을 볼 수 있는데, 여기에 스크립트 이름(함수 이름)을 적으시면 됩니다.
하지만 show_message 같이 이미 게임메이커 지원하는 함수나 직접 만드신 스크립트와 이름이 중복되면,
에러납니다!

이제 직접 코드창을 채워보도록 할까요?
간단하지만 컴퓨터보고 사과 개수를 알려줄 테니 사과 개수를 출력하라고 해보도록 하죠!

스크립트 이름은 apple_count 라 하죠,

```
var apple = argument0;
show_message(apple);
```

이렇게 코드 내용을 적어보세요!
허헛 이해 안되시는 게 한 가지 있을 듯한데,

var apple = **argument0**;

여기서 var은 apple 변수를 지역 변수(하지만 임시 변수라는 용어를 주로 사용함)로 만들겠다는 것입니다.
지역 변수가 게임메이커에서 무엇이냐, 원래 의미는 중괄호{~} 사이에 선언한 변수를 의미하는데,
게임메이커에서는 다른 의미로 **코드 액션이나 스크립트가 종료될 때 메모리에서 사라지는 변수**를 의미합니다.

또 **argument0** 에 대해서 궁금하실 텐데, 인자를 쓸 때는 argument0, arugment1,,,argument15 총 16개의 인자를 지원합니다.
argument[0],, 이런 식으로 쓰기도 하는데, 이 배열 형태의 인자를 사용한 이유는 혹시나 그 인자가 필요 없는 경우가 스크립트를 작성하면서 생길 수 있기 때문입니다. 지금 이해 안가시더라도, 계속 스크립트를 쓰다보면 인자를 사용자 마음대로 쓰고 싶기도 하고 안 쓰고 싶을 경우가 있을 때가 있어서. 그때 이 말을 이해하실 수 있을 것 입니다.

이제 이 함수 사용은 지난번에 설명했으니 어서 스크립트를 써봅시다!

apple_count(**5**);

전 이런 식으로 써봤습니다. 이렇게 하고 게임을 실행하면 **5** 라고 뜹니다!
참 쉽죠?

이번엔 **return**에 대해 설명해드리겠습니다.
스크립트 apple_count를 수정해볼까요!

```
var apple = argument0;
return apple*2;
```

이 스크립트는 사과의 개수를 두 배 늘려 값을 말해주는 스크립트입니다.
return A; A에 반환해줄 값을 써주시면 됩니다.
그리고 아래와 같이 이 반환한 값을 받을 수 있습니다.

```
apples = apple_count(5); //apples에는 10이 저장됩니다.
show_message(apples); //한번 실험해보시라고 작성해봤습니다! 10이 나옵니다.
```

6. 실수와 문자열

이 두 가지의 개념은 간단합니다.
실수는 유리수, 무리수를 포함하는 수를 말합니다만, 게임메이커에서는 **유리수**만 표현 가능합니다. (ex / 1235, 101.2)
문자열은 "안녕하세요" 이런 식의 데이터를 말합니다. (ex / "Hello" / 'ASDF')

이 두 가지를 변수에 저장해보도록 합시다!

(**실수를 변수에 저장하기**)

```
apple = 5; //정수!
count = 5.235; //소수 표현 가능!
```
(무리수 사용은 불가능; 아무래도 언어 개발자들도 실수를 한 것 같습니다)

(**문자열을 변수에 저장하기**)

```
text = "Hello!"; //단어!
text1 = "Hello Guys!"; //두 단어! 그 이상도 저장가능
text2 = 'Hello Guys!'; //큰따옴표 대신 작은따옴표로 묶어도 됩니다.
text3 = "Hello "+"Guys!"; //이렇게 +를 이용해 문자열 데이터 저장 가능하고, 띄어쓰기도 데이터에 저장됩니다!

apples = 5; //실수 데이터
text4 = "I have "+apples+" apples"; //이건 틀렸습니다! 아래 자세히 설명하겠습니다.
```

변수 하나에 저장할 데이터를 통일해야합니다, 안 그러면 짜증나는 에러창이 뜹니다!
그래서 위에 text4에서 생긴 문제는, apples의 5는 문자열이 아닌 실수라는 것입니다.
이 문제를 어떻게 해결하느냐! 이 문제 해결을 위한 함수가 존재합니다.

string(val) //val에는 실수를 쓰면 문자열로 데이터를 바꿔 반환해줍니다.
real(val) //val에는 문자열을 쓰면 실수로 데이터를 바꿔 반환해줍니다.

그러면 아래와 같이 text4 = ~;를 수정할 수 있겠죠?

```
apples = 5;
text4= "I have "+string(apples)+" apples";
show_message(text4);
```

이 코드를 직접 해보시면 **I have 5 apples** 가 나옵니다!
4강에서 잠깐 알려줬지만, draw_text(16,16,**"I have "+"three "+"apples"**); 이제 이것이 가능한 것을 알 수 있습니다!

7. 조건문

if 라는 용어는 **"만약에"** 이라는 영어 뜻을 가지고 있습니다.
if는 말 그대로 코드에 작성하시면 됩니다.

```
apple = 0;
if (apple == 1){
  show_message("Oh Yeah");
}
```

if 다음에 괄호 (~) 이렇게 있는데 괄호는 안 쓰셔도 되긴 하지만, 쓰시는 게 보기 편하십니다!
처음 보는 것들은, **if**, ==, **{~}** 이겠죠.

(**==에 대한 설명**) A==B일 경우 1(true)를 반환하는 연산자입니다.
(**{~}에 대한 설명**) 중괄호로 코드를 묶는 것이라 보면 됩니다.
조건문, 앞으로 배울 반복문, with문의 필요한 것입니다!
이제 if를 설명해야겠군요. 이것은 코드를 해석하면 편합니다. 위의 if 부분의 코드를 한국어로 해석하면

```
apple = 0;
만약에 (apple == 1) 하면
  show_message("Oh Yeah");
하겠다.
```

아참! if 다음에 (~) 괄호의 값이 true**(숫자로는 1)** 일 경우에는 문 안으로 들어가서 show_message("Oh Yeah"); 를 실행합니다! 그래서 위에 설명한대로 A==B일 경우 **true**를 반환하니, 위의 코드에서 apple==1일 경우 true를 반환하게 되므로,
show_message(~)를 실행하겠지만, 제가 apple을 0으로 해놔서 show_message(~)는 실행되지 않습니다.

(~) 괄호 안에는 표현식을 작성할 수 있는데, (apple == 1)도 그 중 하나입니다.
이 표현식은 다음 8강에서 자세히 설명하겠습니다.

만약 그 조건이 아니면, 또 쓸 수 있는 문이 있습니다. **ELSE!** (**다른**)
아래와 같이 쓸 수 있습니다.

```
apple = 0;
if (apple==1){
  show_message("Oh Yeah");
}
else{
  show_message("Give me more apples");
}
```

만약 (apple==1)이 성립되지 않으면 show_message("Give me more apples"); 가 실행됩니다.
또, 다른 조건까지 포함하고 싶다면 아래를 보세요!

```
apple = 0;
if (apple==1){
  show_message("Oh Yeah");
}
else if (apple==2){
  show_message("Oh.. No..");
}
else{
  show_message("Give me more apples");
```

```
}
```

이 아이는 사과 두개를 먹기 싫어서 "Oh.. No.."라는 말을 하군요.

```
else if (~){
  ~
}
```

이렇게 쓰시면 됩니다.

이제 **SWITCH**!
이것은 형태 알려주는 것이 제일 좋을듯합니다.

```
apple = 0;
switch(apple){
 case 1:{
  show_message("Oh Yeah");
 break;
 }
 case 2:{
  show_message("Oh.. No..");
  break;
 }
 default:{
  show_message("Give me more apples");
  break;
 }
}
```

switch(데이터) 는 "변수의 어떤 경우" 라 생각하시면 됩니다.
case는 "특정한 경우"라는 뜻인데, **case 변하지_않는_데이터:{ ~ }** 이렇게 써주시면 됩니다.
case의 데이터는 변수를 통해 불러오는 것은 허용하지 않습니다. (ex / case apple:{~}) **X**
default는 case 외의 경우입니다. default는 안 써도 상관없고, case도 몇 개를 써도 상관없습니다.

근데 **break;** 처음 보셨죠? 이것은 switch 문을 나가겠다는 문입니다. (또한 with, for, while, repeat 문을 나갈 때도 씁니다. 나중에 이것에 대한 강좌 올릴 예정입니다) 만약 case 1: 안에 apple+=1; 이런 식으로 사과 개수를 하나 늘려 그다음 case 2: 로 들어갈 현상과, default: 로 들어갈 현상을 없애기 위해서 바로 swtich 문을 나가는 break; 가 필요합니다. break 없이 뭐 활용할 수 있을지는.. 골치 아픕니다!

게임메이커에서 지원하는 상수인 true false는 숫자 1과 0과 똑같습니다. 하지만 다른 프로그래밍 언어에서는 다르다는 점을 알아두셨으면 좋겠습니다.

8. 표현식

연산자는 더하기, 빼기, 나누기 등 여러 연산을 하기 위해 쓰는 것을 의미합니다.

이진 연산자

- &&, ||, ^^ : 논리 연산자 (&& = and, || = or, ^^ = xor)
- <, <=, ==, !=, >, >= : 비교 연산자 진(true(1))나 가짜(false(0))를 반환합니다.
- |, &, ^ : 비트 마다의 논리 연산자 (| = 비트 마다의 or, &= 비트 마다의 and, ^ = 비트 마다의 xor)
- <<, >>: 비트 시프트 연산자 (<< = 시프트 레프트, >>= 시프트 라이트)
- +, - : 덧셈, 뺄셈
- *, /, div, mod : 곱셈, 나누기, 정수 나누기, 나머지 구하기

단항 연산자

- !: true는 false로, false는 true로 반전합니다.
- -: 부호를 반전합니다.
- ~: 비트를 반전합니다.

이제 각 연산자의 설명을 하겠습니다.
** 연산할 때 1, 0의 의미는 1은 true를 의미하고, 0은 false를 의미합니다.

[논리 연산자]
A **&&** B : A와 B 둘 다 1 일 경우, 1이 나옵니다. 아닐 경우 0이 나옵니다. (&& 대신 and로도 쓸 수 있습니다)
A **||** B : A나 B, 둘 중 하나가 1 일 경우, 1이 나옵니다. 아닐 경우 0이 나옵니다. (|| 대신 or로도 쓸 수 있습니다)
A **^^** B : A나 B, 둘 중 하나의 값이 1이고, 나머지 하나의 값이 0일 경우, 1이 나옵니다. 아닐 경우 0이 나옵니다. (^^ 대신 xor로도 쓸 수 있습니다)

[비교 연산자]
A **<** B : A가 B보다 작을 경우, 1이 나옵니다. 아닐 경우 0이 나옵니다.
A **>** B : A가 B보다 클 경우, 1이 나옵니다. 아닐 경우 0이 나옵니다.
A == B : A와 B가 같을 경우, 1이 나옵니다. 아닐 경우 0이 나옵니다.
A **<=** B : A가 B보다 작거나 같을 경우, 1이 나옵니다. 아닐 경우 0이 나옵니다.
A **>=** B : A가 B보다 크거나 같을 경우, 1이 나옵니다. 아닐 경우 0이 나옵니다.
A **!=** B : A와 B가 같지 않을 경우, 1이 나옵니다. 아닐 경우 0이 나옵니다.

[비트 연산자] - 논리 연산자와 유사한 부분입니다만, 수끼리 비트단위로 하나하나 연산을 처리합니다! (*2진수)
A **&** B : A와 B의 비트 하나하나 비트단위로 두 비트 둘다 1일 경우 1로 바꾸고, 아닐 경우 0으로 바꿉니다.
A **|** B : A와 B의 비트 하나하나 비트단위로 두 비트 둘 중 하나가 1일 경우 1로 바꾸고, 아닐 경우 0으로 바꿉니다.
A **^** B : A와 B의 비트 하나하나 비트단위로 두 비트 둘 중 하나의 값이 1이고, 나머지 하나의 값이 0일 경우, 1로 바꾸고, 아닐 경우 0으로 바꿉니다.

[비트 시프트 연산자]
A **<<** B : A의 2진수 데이터를 왼쪽방향으로 B만큼 이동하고 범위 이상 나가버릴 경우 버려집니다.
ex) - A가 0100 0000 이고 B가 1일때, A <<B를 하면 A는 1000 0000 이 됩니다.
- A가 1000 0000 이고 B가 1일대, A <<B를 하면 A는 0000 0000 이 됩니다. 바깥으로 나가버릴 경우 버려집니다.
A **>>** B : A의 2진수 데이터를 오른쪽방향으로 B만큼 이동하고 범위 이상 나가버릴 경우 버려집니다.

[덧셈과 뺄셈... 연산자]
A **+** B : A에 B를 더한 값이 나옵니다.
A **-** B : A에 B를 뺀 값이 나옵니다.
A ***** B : A에 B를 곱한 값이 나옵니다.
A **/** B : A에 B를 나눈 값이 나옵니다.
A **div** B : A에 B를 나머지까지 고려하여 나눈 값이 나옵니다.
A **mod** B : A에 B를 나머지까지 고려하여 나눈 값이 아닌 그 나머지 값이 나옵니다.

[단항 연산자들]
!A : A가 1(true 값)일 경우 0이 나옵니다. A가 0(false 값)일 경우 1이 나옵니다. (!(A&&B) 이런 식으로도 쓸 수 있습니다)
-A : A가 양수일 경우 음수로, A가 음수일 경우 양수로 나옵니다.
~A : A의 비트를 반전한 값이 나옵니다.

이 연산자들을 한 표현식에 많이 쓸 수 있습니다.
제가 이 많은 연산자들을 응용해서 코드를 작성해서 쓰는 것은 조금 무리이기에 몇 가지만 사용해보도록 하겠습니다.

```
A = 500;
B = 250;
if (A<(B+251)){
 show_message("B Win");
} else{
 show_message("A Win");
}
```

직접 이 코드를 게임메이커에 써보니 결과는 무엇이 나왔나요?
B Win이 나왔습니다.

여러분도 직접 이 연산자들을 이용해 다양한 표현식을 만들어보시기 바랍니다!

9. 주석

```
//안녕하신가요, 이것이 바로 오늘 배울 주석이라는 것입니다.
```

주석은 코드를 설명하는 것인데, 형태는 두 가지가 있습니다.

```
//TEXT~~
```
이 형태는 한 줄만 주석을 쓰는 것입니다.

```
/*TEXT~
TEXT~
TEXT~*/
```
이 형태는 여러 줄을 주석으로 쓰는 것입니다.

실전에서는 이런 식으로 씁니다.

```
Apple = 2;
if (Apple <= 2){ //사과가 2개 이하일 경우
  show_message("더 가져와라");
} /*else asdfasd{
  show_message(adsfsgas)::;
}*/
```

/* */는 코드를 일부 작동 안하게 할 수도 있습니다. 위 코드에 /* */ 이 형태의 주석이 없었으면 확실히 에러가 나버리겠죠~!

10. 반복문

이번 강좌에서는 while, do-until, repeat, for문을 배워볼꺼에요!
배운 다음 바로 써먹어보길 바랍니다!

(while 문 사용하기)

while(**true**){
~코드
}

while 괄호의 값이 true일 경우 - **~코드** 실행 - 반복

하지만 위처럼 그대로 써먹다간 큰일 나요! ~코드가 무한 반복 실행이 일어나서.. while 괄호 안의 값의 조건을 언젠가는 false가 되도록 해야 합니다. 아니면 11강에서 배울 break문이나 exit 문을 사용하셔도 됩니다.

(do-until 문 사용하기)

do{
~코드
}**until**(**false**)

do의 ~코드 한번 실행 / until 괄호의 값이 false일 경우 - do의 **~코드** 실행 - 반복

이 반복문은 while 문과 비슷하지만, 괄호의 값이 false일 경우 반복하며,
do의 코드를 until 괄호의 값이 true라도 한번은 실행합니다.

(repeat 문 사용하기)

repeat(**반복 횟수**){
~코드
}

~코드를 repeat 괄호의 값만큼 반복합니다.

이 반복문은 쉬워서 더 설명이 필요가 없어 보입니다.

(for 문 사용하기)

for (i=0; i<10; i+=1){
~코드
}

이 반복문은 변수의 개념까지 들어가서 어려울 수 있습니다. 반복 실행 구조가 조금 복잡합니다!

1) (i=0;) 변수 초기화 부분 실행 **<변수 초기화식>**
2) (i<10;) 조건에 맞는지 확인 **<조건식>** -조건이 맞지 않는 경우 ~코드 실행하지 않고 바로 빠져나온다.
3) ~코드 실행

4) 조건에 맞는 경우 (i+=1)를 통해 변수 조정 **<증가식 또는 감소식>**

5) (i<10;) 조건에 맞는지 확인 **<조건식>** - 조건이 맞지 않는 경우 ~코드 실행하지 않고 바로 빠져나온다.
6) ~코드 실행

...반복...
이렇게 실행된다는 점을 유의하시고,
for문은 아래와 같이 작성하시면 됩니다.

for (**변수 초기화식; 조건식; 증감식**){
~코드
}

11. 유용한 문들

이번 강좌에서는 반복문에 잘 써먹을 수 있는 문들입니다!
break, continue, exit 배워봅시다!

(break 사용법)

switch 문에서도 사용했던 것이죠, **break의 기능**은 **반복문과 switch 문을 한번 빠져나가는 기능**입니다.

반복문에서는 가끔 예외적으로 빠져나가고 싶을 때가 있는데, break가 if문은 빠져나가지 않으니 if문을 이용해서 코드를 작성하시면 됩니다. 나중에, for문을 두개 겹치는 등으로 쓸 때 break 쓰면 무조건 for문 두개 겹친 것을 빠져나가버리진 않습니다.

```
apple = 0;
while(true){
 apple+=1;
 if (apple==10){break;} //이렇게 하면 while(true)라도 빠져나갈 수 있죠!
}
```

특별한 경우. 반복문 두개 빠져나가기

```
a = 0;

for(x=0;x<5;x+=1){
 for(y=0;y<5;y+=1){
  if (x==3)&&(y==2){a=1; break;} //한번 빠져나가기
 }
 if (a==1){break} //변수 이용해서 한번 더 빠져나가기
}
```

(continue 사용법)

반복문에서 한 부분을 건너뛸 때 사용합니다. 예시를 보는 것이 빠르겠군요!

```
apple = 0;
repeat(10){
 apple+=1;
 if apple==5{continue;} //이렇게 하면 apple이 5일때 다음 반복으로 건너뛰게 되므로, 아래 show_message가 실행되지 않게 됩니다.
 show_message("사과 "+string(apple)+"개 머겅");
}
```

(exit 사용법)

게임메이커 기준에서 exit는 한 코드 액션만 빠져나갑니다.
게임메이커 스튜디오 기준에서 exit는 이벤트 자체를 빠져나갑니다.
하지만 둘 다 스크립트에서 exit는 스크립트를 빠져나갑니다.

많이 안 쓰기도 하는 문인데, 위 break 사용할 때 특별한 경우. 반복문 두개 빠져나가기 같은 것에 귀찮으니 exit로 쉽게 빠져나가는 방법을 쓰기도 하는데, 좀 위험한 방법이고 단점도 많은 방법입니다!

12. 인스턴스와 변수

self.변수이름 : 로컬변수를 의미합니다. (자기 인스턴스만 사용할 수 있는 변수)
<self.는 생략해도 되지만 필요할 때가 있습니다>
global.변수이름 : 전역변수를 의미합니다. (어느 인스턴스에서나 사용할 수 있는 변수)
other.변수이름 : with문 안에 있을 때, 충돌 이벤트 안에 있을 때 의미가 달라지는데, 아래 설명해드리겠습니다.

with문은 **어떤 인스턴스(들)에게 이러한 코드 명령을 실행하라고 하는 문**입니다.
with문의 사용법은 아래와 같습니다.

```
with(인스턴스 id, 오브젝트 이름, 또는 all){ //all 은 모든 인스턴스들이 실행할 명령을 하려고 할 때 씁니다.
 ~with 괄호 안에 쓴 인스턴스(들)이 실행할 코드 명령들~
}
```

[이 코드의 오브젝트 이름을 **obj_player**라 할 때]

```
apple = 0;
wtih(obj_apple){
 apple_count = other.apple; //자기 인스턴스의 apple 변수를 가져옵니다.
 apple_count = obj_player.apple; //이런 방법도 되기는 하지만, obj_player라는 오브젝트가 여러 개 있을 경우 그 중 아무 오브젝트의 변수를 가져와버리기 때문에 게임메이커 함수를 쓰던지, other. 를 쓰는 것이 좋습니다.
}
```

[**충돌 이벤트**, 이 코드의 오브젝트 이름을 **obj_player**라 할 때]

```
with(other){ //충돌된 대상, 'other.변수이름'이 아닌 'other'만 쓸 경우 해당 인스턴스id를 가져옵니다.
 instance_destroy(); //충돌된 대상 제거
}
```

var 는 임시변수를 선언하는 것입니다.
임시변수는 코드 액션이 끝나면 삭제됩니다.
이것은 최적화할 때도 유용한데, 사용법을 알아봅시다!

```
var 변수이름; //하나만 작성해도 됨
var 변수이름1, 변수이름2, 변수이름3; //여러개도 콤마를 이용해 작성가능
```

게임메이커 8 버전에서는 임시변수 선언과 값을 넣어주는 동시에 하는 것이 안 되는데,
게임메이커 스튜디오 버전에서는 아래와 같이 가능합니다!

```
var 변수이름 = 100;
var 변수이름1 = 100, 변수이름2 = 123;
```

게임메이커 8이나 게임메이커 스튜디오나 이 var에서 중요한 것은 ;(세미콜론)을 변수 선언한 후, 끝에 붙여야합니다.

(**활용**) - PGM 카페 겜메 팁 중 일부분을 가져왔습니다. (멍멍이님이 작성해주셨습니다)

self 는 어떤 상황에서건 로컬 변수를 지칭할 때 쓰입니다. 즉,

```
var aaa;
aaa = 3
self.aaa = 5
```

라고 하게 되면, '임시 변수 aaa' 는 3이 되지만, '로컬 변수 aaa'가 5로 선언됩니다. 더불어서

```
globalvar bbb;
bbb = 7
self.bbb = 3
```

이런 식으로, globalvar 와 이름이 겹치더라도 self를 달아주면 구분이 가능합니다. 덤으로

```
globalvar aaa; var aaa;
aaa = 3
global.aaa = 4
self.aaa = 5
```

이렇게 하면, 임시 변수 aaa는 3, 전역 변수 aaa 는 4, 지역 변수 aaa 는 5가 됩니다.