


Maxscript(기초 입문편)

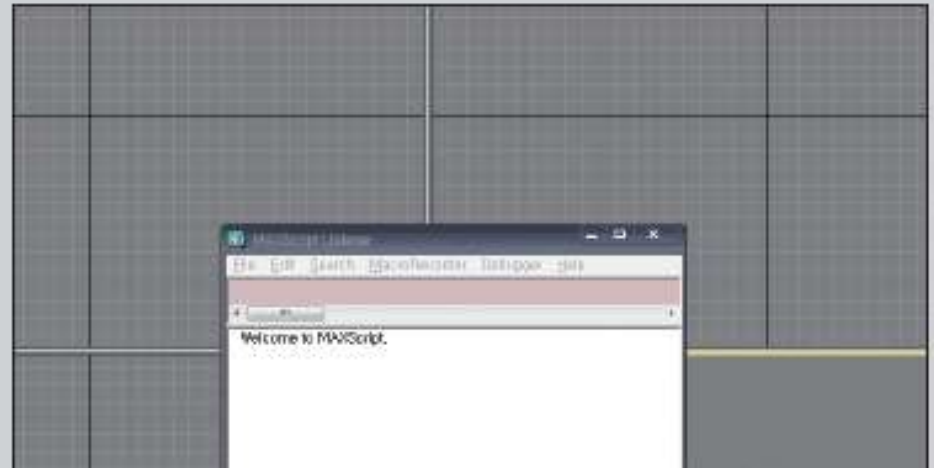
MaxScript로 할 수 있는 일

- MaxScript로 모델링, 애니메이션, 메터리얼, 렌더링 등을 제어하고 제작할 수 있다.
- Maxscript Listener에서 커맨드 라인 입력으로 즉시 제어할 수 있다.
- 사용자가 UI(User Interface)를 직접 디자인하고 패키지 할 수 있다.
- User Interface를 Object, Modifiers, Materials, Textures, Render Effects, Atmospheric Effect에 확장하거나 덮어씌울 수 있다.
- Objects, Modifiers, Render Effect 등 Script로 Plug-in을 제작할 수 있다.

MaxScript Listener의 기초

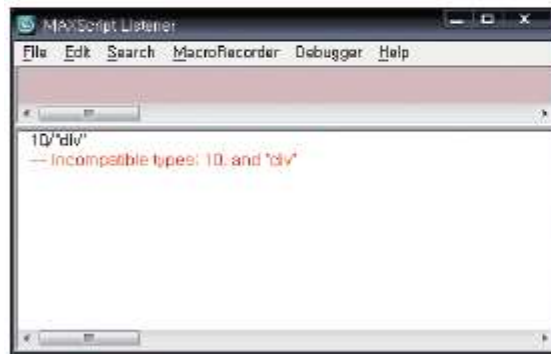
3ds Max에서 Scripting > MaxScript Listener나 을 눌러서 MaxScript Listener(이하 Listener)를 열어봅시다.

그림에서 보듯이 두 개의 영역으로 구성된 작은 창이 뜹니다. Listener는 언뜻 보면 별로 하는 일이 없어 보이는 작은 창이지만 Script를 만드는 데 매우 중요한 역할을 합니다. Listener가 하는 일은 주로 출력을 담당하며 중요한 내용을 보여주거나 변수의 내용을 알려주기도 하고 간단한 계산기 역할도 합니다. 또한 Script의 버그가 존재할 때 디버깅해볼 수도 있고 자신이 실행한 내용을 그대로 표시하기도 합니다.

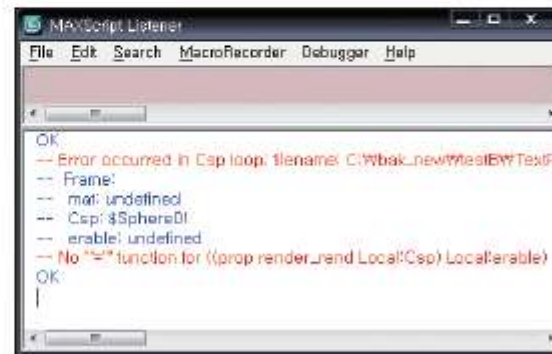




Listener의 또 다른 기능은 에러 메시지 출력입니다. 명령어를 잘못 입력했거나 Script의 오류가 있다면 즉시 에러 메시지를 출력합니다. 그림의 빨간색 글자는 Script 에러가 발생한 경우입니다. 아직 Script에 익숙하지 않기 때문에 이와 같은 메시지를 자주 보게 될 것입니다. 이 빨간색 에러 메시지를 자세히 살펴보면 에러의 발생 원인을 알 수 있습니다.

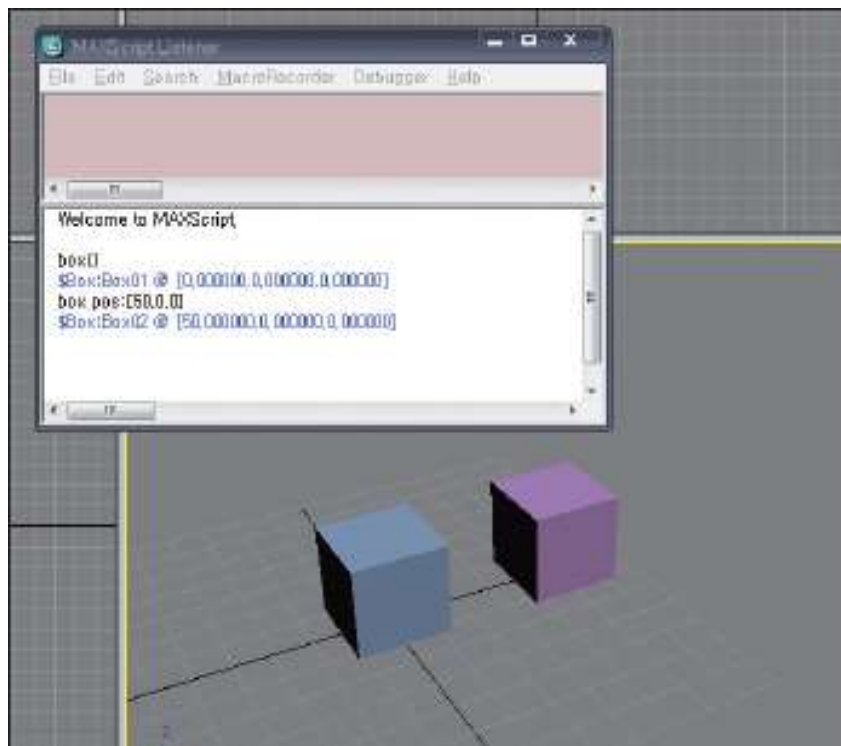


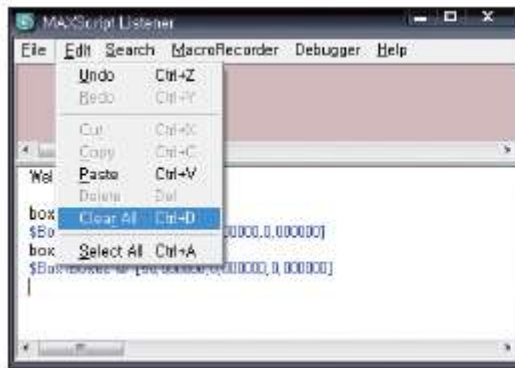
Script 에러 발생



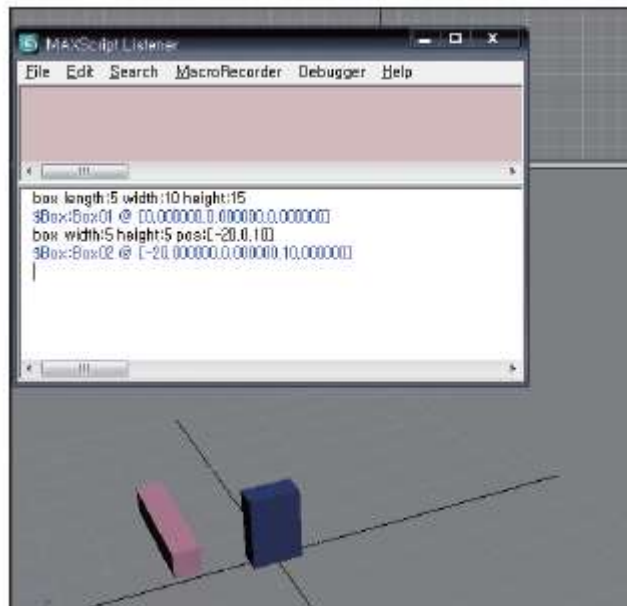
Script를 본격적으로 배우기 전에 Listener에 익숙해져야 하므로 계속해서 Listener에서 할 수 있는 것을 계속 입력해보겠습니다.

```
box ()  
box pos:[50,0,0]
```



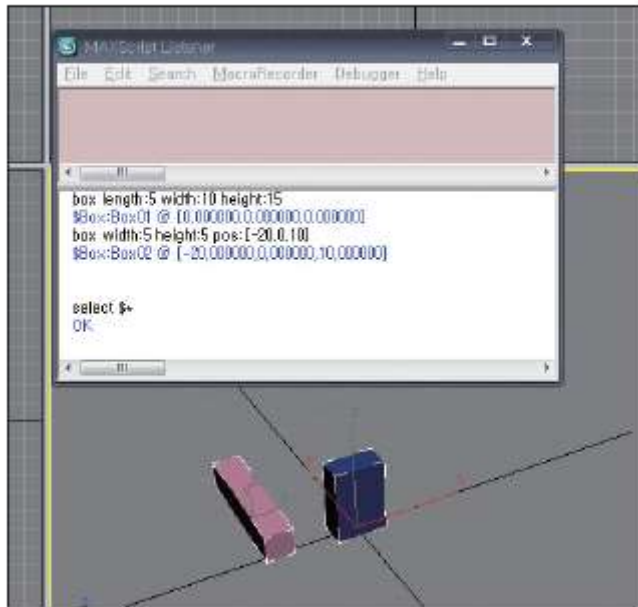


메뉴에서 'Clear All'을 실행하면 화면이 깨끗해집니다.



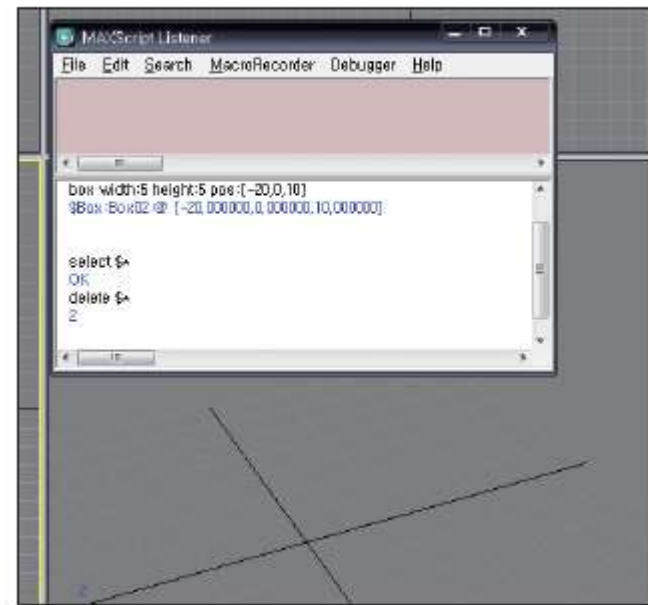
```
box length:5 width:10 height:15
box width:5 height:5 pos:[-20,0,10]
```

만들어진 Box를 선택하겠습니다.



Select \$*

선택한 후에 지우는 방법도 간단합니다.



Delete \$*

Select나 Delete 명령어 뒤에 \$*가 붙는데 이것은 전체 오브젝트를 가리키는 Collection을 뜻

```
sphere()
```

```
sphere pos:[50,0,0] radius:15
```

```
dummy pos:[100,0,0]
```

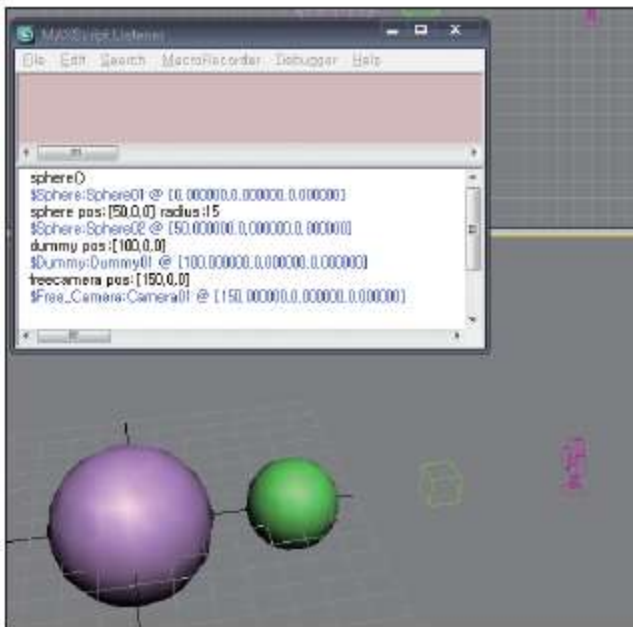
```
freecamera pos:[150,0,0]
```

-- sphere를 만든다

-- sphere를 [50,0,0]에 radius가 15인 크기로 만든다.

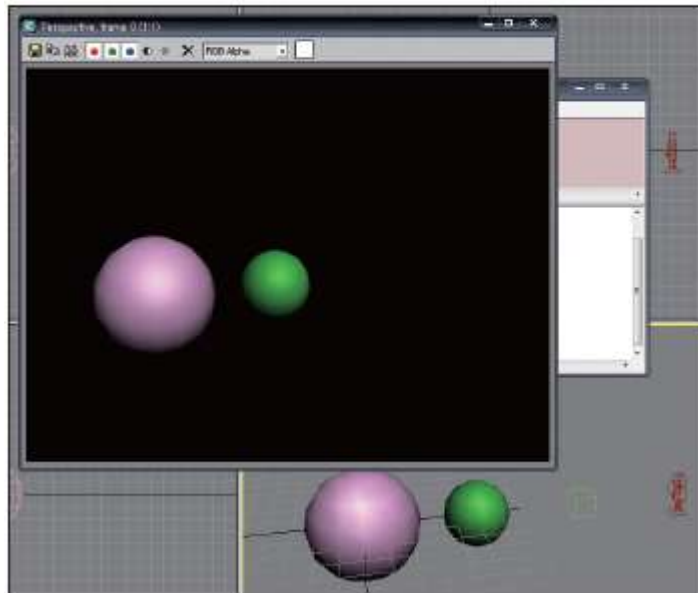
-- dummy를 [100,0,0] 위치에 만든다.

-- freecamera를 [150,0,0] 위치에 만든다.



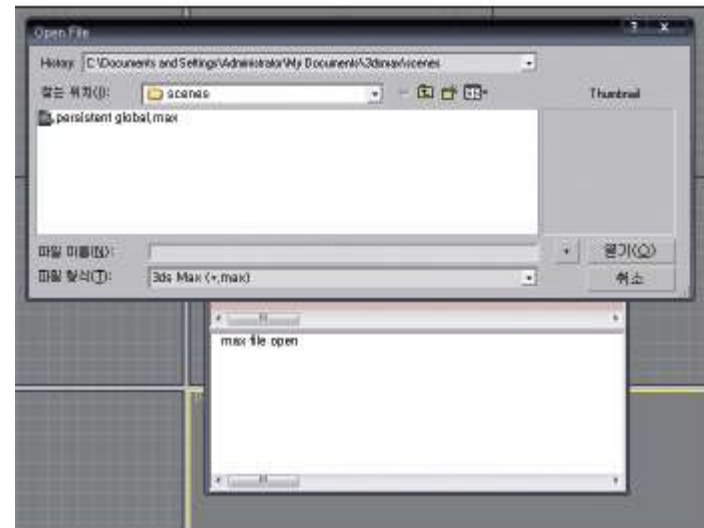
Max quick render

-- render 버튼을 누른 효과



Max file open

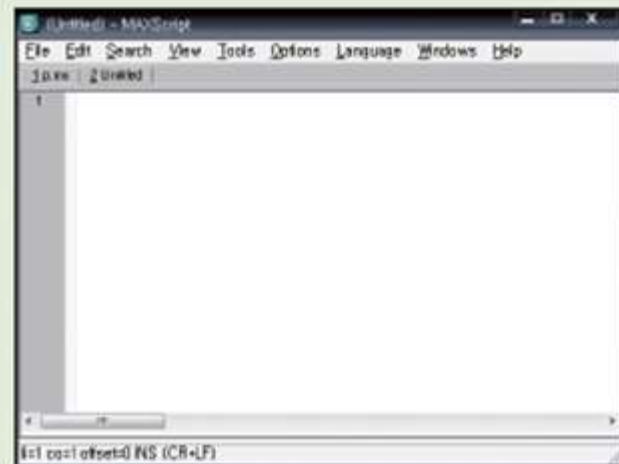
-- 파일 오픈창을 연다.



한 줄 이상의 Script를 작성하려면?

한 줄 이상의 Script를 작성하려면 Max Script Editor를 사용해야 합니다.

〈Script Editor〉를 이용하려면 Listener에서 〈**Ctrl+N**〉
을 누르거나 상단 메뉴의 **Scripting > NewScript**
Editor를 선택하여 실행할 수 있습니다.

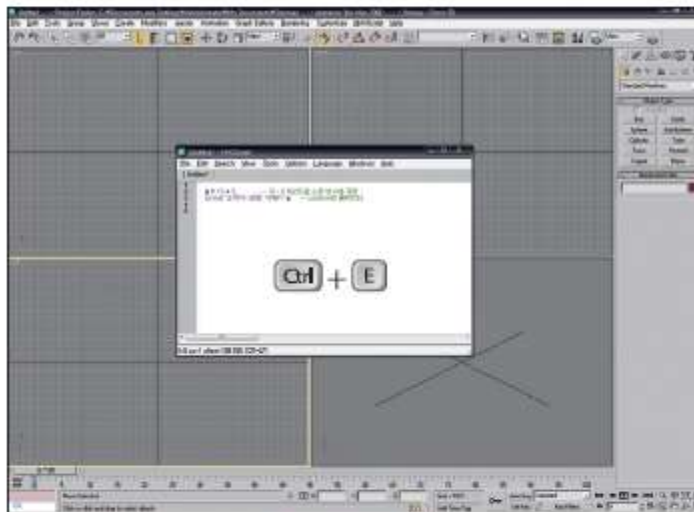


MaxScript Editor

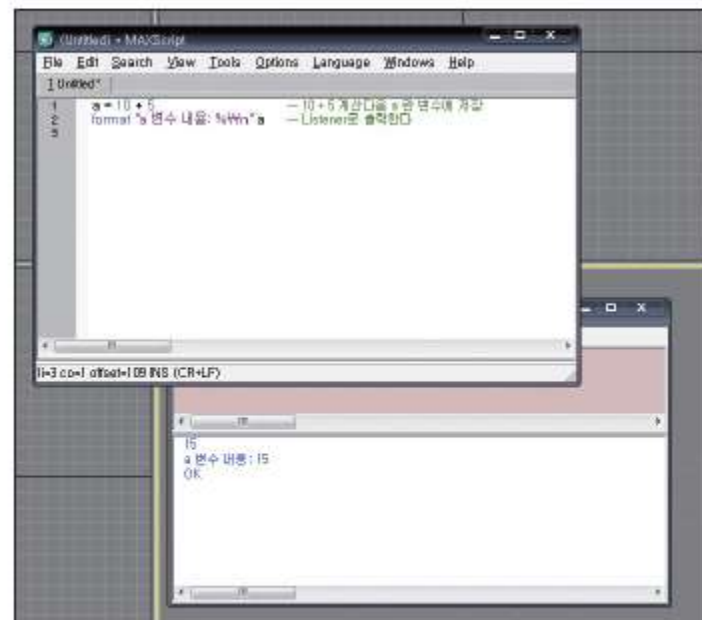
Editor를 열었으면 간단한 Script를 작성해 보겠습니다.

```
a = 10 + 5          -- 10 + 5 계산한 후 a란 변수에 저장
format "a 변수 내용: %\n" a  -- Listener로 출력한다.
```

Ctrl + **E**를 누르거나 메뉴의 Tools > Evaluate All를 누르면 Script가 즉시 실행합니다.



Script Editor에서 **Ctrl** + **E**를 눌러 실행합니다.



실행 결과

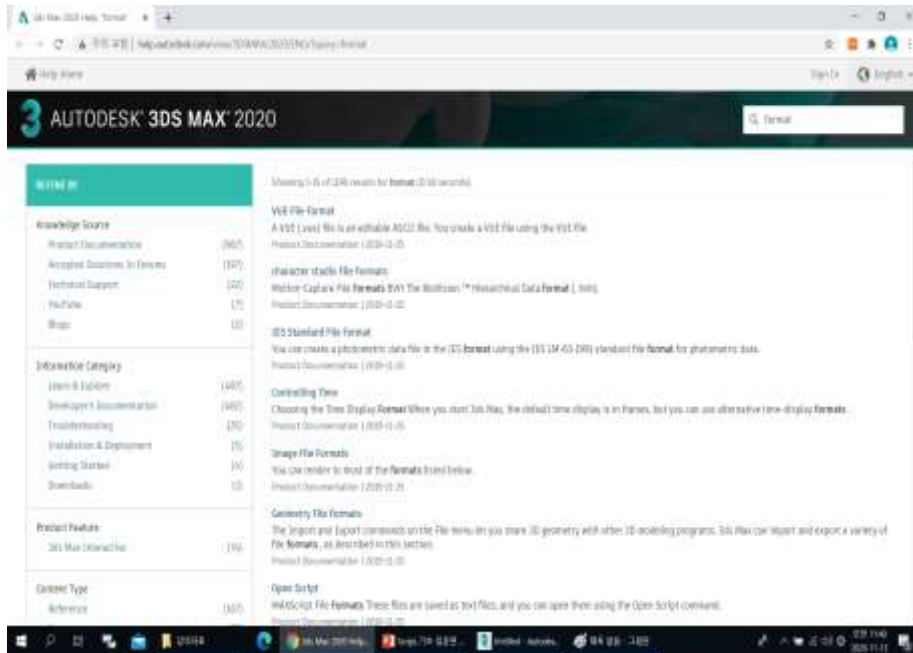
format이라는 함수가 나오는데 print와 비슷하게 Listener로 출력해주는 기능을 합니다.

MaxScript Editor 단축키

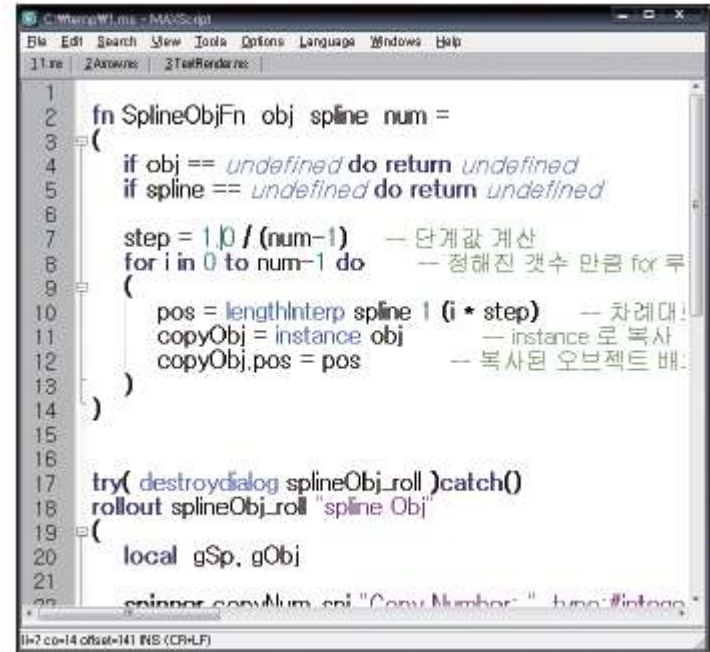
MaxScript Editor에서 자주 쓰이는 단축키를 알아봅시다.
매우 자주 쓰이는 단축키들이니 잘 알아두는 것이 좋습니다.

Ctrl + E	Script를 실행한다.
Shift + Enter	한 줄 단위로 실행한다.
Ctrl + S	코드를 저장한다.
Ctrl + Shift + S	Script를 다른 이름으로 저장한다.
Ctrl + N	Script를 새로 작성한다.
Ctrl + O	Script를 open 한다
Ctrl + W	Script를 close 한다.
Ctrl + Z	undo
Ctrl + Y	redo
Ctrl + C	문자열 복사
Ctrl + V	문자열 붙여넣기
Ctrl + X	문자열 cut
Ctrl + F	찾기
Ctrl + H	replace
Ctrl + F2	북마크 toggle
F2	북마크로 이동
Ctrl + Q	주석문으로 만든다 (toggle)

Editor에서 **F1**을 누르면 Maxscript Reference가 열립니다.



글자 크기가 너무 작다면 **Ctrl** + 마우스 휠을 이용하면 크기를 조정할 수 있습니다.

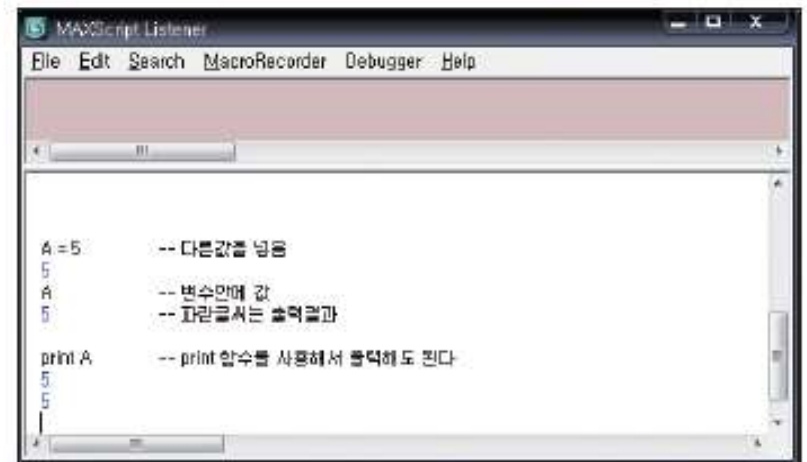
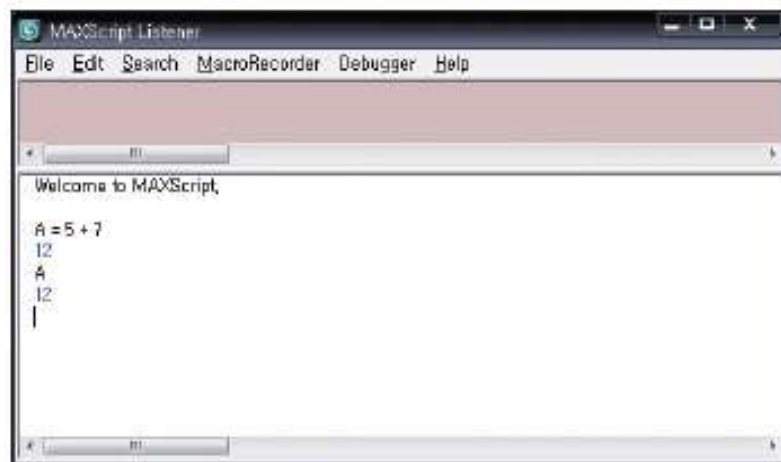


글자 크기의 조절

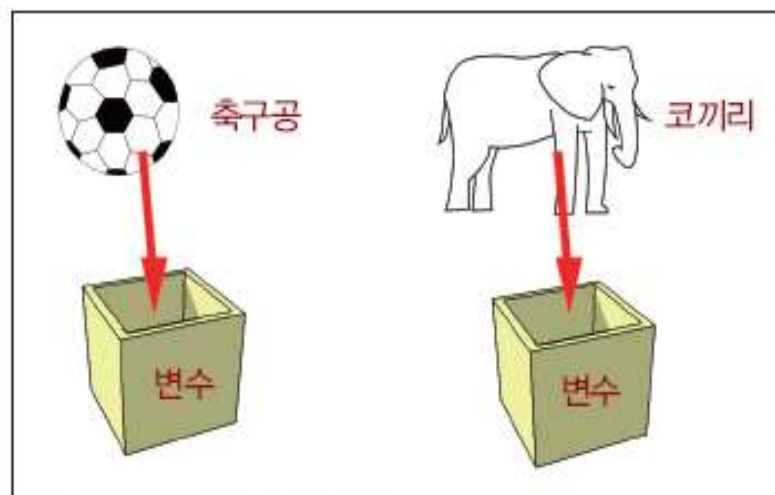
변수의 대입

변수를 최초로 사용하려면 먼저 어떤 값을 넣어야 합니다. 변수에 어떤 값을 대입할 경우에 대입한 덩치만큼 메모리 공간이 생기며 그제서야 변수로서 효력이 생깁니다. 만약 대입한 변수의 다른 값을 넣었다면 먼저 대입한 값은 지워지고 새로운 값이 들어갑니다. 그리고 변수 생성에 C언어나 자바처럼 데이터 타입을 결정하지 않아도 됩니다.

```
my = 15          -- my 변수에 15를 대입
A = 5 + 7        -- 5 와 7을 더한 값을 A란 변수에 저장(대입)하라는 의미
x = y = z = 0    -- x, y, z 에 0을 대입
```



MaxScript의 변수는 다른 언어와는 다르게 어떠한 데이터형도 넣을 수 있습니다. 그림에서 의미하는 것과 같이 변수는 정수형(Integer), 실수형(Float), 스트링(String), 함수(Function), 롤아웃(Rollout), 오브젝트(object) 등 무엇이든지 어떤 타입과 관계없이 들어갈 수 있습니다. 변수는 선언과 동시에 어떤 값을 대입하면 자동으로 타입이 결정됩니다. 또한 정수형(Integer)에서 스트링(String)과 같이 다른 데이터 값으로 새롭게 덮여져서 전혀 다른 타입으로 변경될 수 있습니다.



변수에 어떤 타입의 값도 넣을 수 있다.

변수의 연산

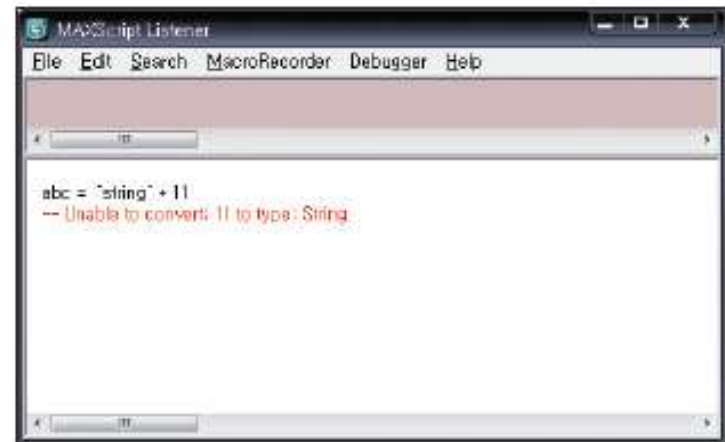
산술 연산자의 종류는 다음과 같습니다.

```
<math_operand> + <math_operand>      -- 더하기  
<math_operand> - <math_operand>      -- 빼기  
<math_operand> * <math_operand>      -- 곱하기  
<math_operand> / <math_operand>      -- 나누기  
<math_operand> ^ <math_operand>      -- 거듭제곱  
<math_operand> as <class>            -- 형변환
```

```
10/3.0                                -- 나누기  
2 ^ 5                                 -- 거듭제곱  
11 as float                           -- 정수형을 실수형으로 형 변환
```


다른 형끼리 연산하면 어떻게 될까요?

그림의 결과처럼 바로 에러 메시지가 빨간 글자로 출력됩니다.



복합 연산자

```
<destination> += <expr>
<destination> -= <expr>
<destination> *= <expr>
<destination> /= <expr>
```

```
a=0
a +=1      -- a = a+1 와 같다.
```

```
1          -- 출력 결과
```

```
b=10
b -=2      -- b = b-2 와 같다.
```

```
8          -- 출력 결과
```

```
c=11.0
c *=2      -- c = c*2 와 같다.
```

```
22.0       -- 출력 결과
```

```
d=100.0
d /=3      -- d = d/3 와 같다.
```

```
33.3333    -- 출력 결과
```

형(Type) 변환하기

```
10 as float          -- 정수형을 실수형으로 변환
10.5 as integer       -- 실수형을 정수형으로 변환

55 as string          -- 정수형을 스트링으로 변환
[10,20,30] as color    -- Point3를 color로 변환

10.0                  -- 출력 결과
10
"55"
(color 10 20 30)
```

예약어(Reserved Keywords)

변수 이름을 정할 때는 몇 가지 주의할 점이 있습니다. 특수문자를 포함하거나 예약어 같은 경우 불가능한 변수 명들이 있습니다.

```
aa?                  -- 특수 문자가 들어간 경우
0abc                 -- 변수 명 앞에 숫자가 들어간 경우
e                   -- 예약어
for = 10             -- 예약어
print = 20           -- 함수명
```

Maxscript에서 사용되는 예약어는 다음과 같습니다.

```
About, and, animate, as, at,  
By, case, catch, collect, continue, coordsys,  
do, else, exit,  
fn, for, from, function,  
global, if, in, local,  
macroscript, mapped, max,  
not, of, off, on, or, parameters,  
persistent, plugin, rcmenu, return, rollout,  
set, struct, then, throw, to, tool, try,  
undo, utility, when, where, while, with
```

예약어 외에 미리 정의된 변수(Predefined Global Variables) 등도 있습니다.

On, off, true, false	-- Boolean이라고 불리운다.
Pi, e	-- 수학 상수
Red, green, blue, yellow ...	-- color 값을 의미
X_asix, y_asix, z_asix	-- 축을 의미한 값
Ok	-- 함수나 Expressions 등에서 가끔 나오는 결과값
Undefined	-- 미리 정의되어 있지 않은 값

변수 명으로 사용 가능여부 Script Listener에서 직접 입력하여 확인할 수 있습니다. Listener가 Undefined라고 출력하면 아무 것도 사용하지 않는 빈 값이란 뜻으로 사용이 가능한 변수 명입니다.



MaxScript의 변수는 대/소문자 구분을 하지 않습니다.

만약에 애니메이션을 할 때 누르는 "Auto key" 버튼을 누른다고 하면

```
animButtonState = true
```

라고 간단히 입력하면 됩니다.

이외에 자주 사용되는 3ds Max의 시스템 변수를 몇 가지 알아봅시다.

backgroundColor	-- (Rendering → Environment)의 background 값
ambientColor	-- (Rendering → Environment)의 ambientColor 값
frameRate	-- frameRate 값
keyboard.shiftPressed	--  가 눌러져 있는지를 true/false 보여준다.
keyboard.controlPressed	--  가 눌러져 있는지를 true/false 보여준다.
keyboard.altPressed	--  가 눌러져 있는지를 true/false 보여준다.
keyboard.escPressed	--  가 눌러져 있는지를 true/false 보여준다.
localTime	-- 현재 시간을 보여준다.
maxFileName	-- 현재 열려있는 맥스 파일 이름을 보여준다.
renderHeight	-- 렌더링 Height 값
renderWidth	-- 렌더링 Width 값
rendOutputFilename	-- 렌더링 output file 이름

변수의 범위(Scope of Variables)

지금까지 Listener에서 입력하여 직접 만든 변수는 전역변수(Global Variable)입니다. 전역변수는 코드 전체에 걸쳐서 영향을 발휘하며 한번 선언되면 3ds Max가 종료되기 직전까지 남아있는 변수를 말합니다. 하지만 전역변수는 다른 Script의 전역변수와 겹치면 먼저 생성되었던 변수는 사라지는 단점이 있습니다. 심각하면 다른 전역변수와 충돌로 3ds Max를 시스템 다운될 수도 있습니다.

이런 경우에는 지역변수(Local Variable)를 사용하면 해결할 수 있습니다. 전역변수와 지역변수의 차이는 영역제한에 있으며 사용법은 전역변수이면 Global을, 지역변수이면 Local을 변수 앞에 붙여주면 됩니다.

```
Global foo                                -- 전역변수 선언
Global a,b,c
Global car = 10                           -- 변수 선언과 동시에 10으로 초기화
Airplane = "fly"                          -- Global 키워드를 안 붙인 경우

(
    local abc = 10                        -- 지역 변수 선언
    print abc
)
```

Global이나 local 등의 키워드를 생략하면 자동으로 Script는 전역/지역 변수를 할당하게 됩니다.

```
helicopter = 1          -- 괄호 밖의 최상위 레벨에 있으니 자동으로 Global
(
    airplane = 2         -- 괄호나 함수, rollout 등에 있으면 자동으로 Local
)
```

배열의 기초

배열(Array)은 한꺼번에 대량의 데이터를 순차적으로 처리하거나 보관할 때 사용합니다.

예를 들어 100개의 오브젝트가 있고 이들에 순차적으로 매터리얼(Material)을 적용해야 했을 때 하나씩 일일이 적용하는 것보다 차례대로 오브젝트를 배열에 넣고 루프문(for 문, while 문)을 돌리면 순차적으로 쉽게 처리할 수 있습니다.

배열은 아래와 같은 방식으로 만들 수 있습니다. 배열을 의미하는 #() 안에 어떤 값을 연속적으로 넣으면 되고 값과 값 사이에는 쉼표 ',' 로 구분합니다.

```
#( <expr> { , <expr> } )
#() - 빈 배열
```


배열을 사용하려면 초기화해야 합니다. 배열 이름은 변수 이름을 만드는 것과 동일합니다.

```
myArray = #()          -- myArray란 배열을 만든다.
```

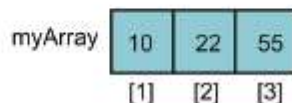
myArray에 단순히 초기화만 했을 뿐이고, 사용하려면 아래와 같이 배열 인덱스에 값을 넣어야 합니다.

```
myArray[1] = 10        -- 1번 배열에 10을 넣는다.  
myArray[2] = 22        -- 2번 배열에 22를 넣는다.  
myArray[3] = 55        -- 3번 배열에 55를 넣는다.
```

```
format "myArray: %\n" myArray  
myArray: #(10, 22, 55)  -- 출력 결과
```

```
myArray = #(10,22,55)
```

위의 myArray를 그림으로 표현해 보면 다음과 같습니다.



이 배열은 다음과 같이 인덱스 값에 의해 액세스가 가능하고 읽고/쓰기를 할 수 있습니다.

```
myArray = #(10,22,55)
myArray[2] = 2500          -- 배열의 두 번째에 2500을 넣는다.
a = myArray[1]             -- 배열의 첫 번째를 a 변수에 대입
format "a:%  myArray: %\n" a  myArray    -- a 와 myArray 출력
format "myArray[10]:% \n" myArray[10]    -- 배열의 열 번째 값을 출력(아무것도 넣지 않았으므로 undefined)

a:10  myArray: #(10, 2500, 55)           -- 출력 결과

myArray[10]:undefined
```

주의할 점은 MaxScript의 배열은 항상 0이 아니라 1부터 시작합니다.

배열의 크기를 알아내려면 .count를 붙이면 됩니다.

```
myArray.count

3          -- 출력 결과
```

배열은 서로 다른 형(Type)을 자유롭게 넣을 수 있습니다.

```
aaa = #()  
aaa[1] = 15          -- 정수형  
aaa[2] = 123.5       -- 실수형  
aaa[3] = [10,20,30]  -- point3  
aaa[4] = "my car"     -- string  
aaa[5] = #()         -- 이중 배열  
aaa[6] = print        -- 함수  
aaa[7] = aaa[1] + 2   -- aaa[1]+2 연산
```

배열에 순서대로 값을 넣지 않아도 상관없습니다.

```
man = #()  
man[1] = 20  
man[5] = 55  
format "man: %\n" man  
format "man size: %\n" man.count  
  
-- 출력 결과  
  
man: #(20, undefined, undefined, undefined, 55)  
man size: 5
```

배열은 루프문인 for 문과 함께 사용하면 효과적입니다. for 문의 자세한 사용법은 제어문에서 다시 배우겠습니다.

```
ar = #()

for i in 1 to 20 do
    ar[i] = random 1 10
format "ar: %\n" ar

-- 출력 결과
ar: #(2, 9, 3, 4, 2, 7, 3, 1, 1, 3, 6, 5, 5, 7, 10, 10, 5, 10, 8, 3)
```

배열끼리는 더하기 연산이 가능합니다.

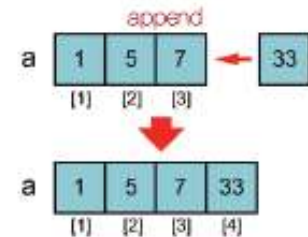
```
#(1,2) + #(5,3)
#("a","b","35") + #(1,2,3)

#(1, 2, 5, 3)
#("a", "b", "35", 1, 2, 3)
```

배열 함수

append <array> <value>

배열의 마지막 요소에 값을 집어넣습니다. 결과적으로 배열을 리턴합니다.



```
(  
    a = #(1,5,7)  
    append a 33          -- a배열에 추가로 33을 넣는다.  
    format "a: %\n" a  
)  
  
a: #(1, 5, 7, 33)      -- 출력 결과
```

deleteItem <array> <number>

배열에서 지정된 인덱스의 요소를 지웁니다. 한 개의 사이즈가 줄어들며 결과적으로 배열을 리턴합니다.

```
(  
    a = #(1,5,7)  
    deleteitem a 2      -- a배열에서 2번째 아이템을 지운다.  
    format "a: %\n" a  
)  
  
a: #(1, 7)              -- 출력 결과
```

```
findItem <array> <value>
```

배열에서 지정한 값을 찾습니다. 결과적으로 인덱스를 리턴합니다.



```
(  
    a = #(1,5,7)  
    f = finditem a 7          -- a배열에서 7을 찾는다.  
    format "find index: %\n" f  
)
```

```
find index: 3          -- 출력 결과
```

```
insertItem <value> <array> <integer>
```

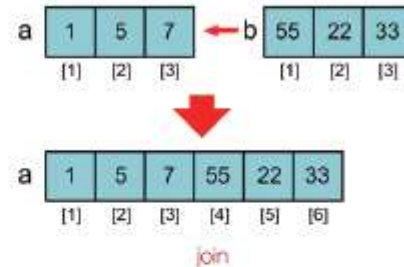
배열에 지정된 값을 추가로 지정한 인덱스에 넣습니다. 배열이 하나 증가되며 성공하면 OK를 리턴합니다.

```
(  
    a = #(1,5,7)  
    insertItem "myBaby" a 2    -- a배열 2번째에 'myBaby'를 삽입한다.  
    format "a: %\n" a  
)
```

```
a: #(1, "myBaby", 5, 7)          -- 출력 결과
```

join <array> <collection>

배열에 collection을 합침(append)니다.



```
(  
    a = #(1,5,7)  
    b = #(55,22,33)  
    join a b  
    format "a: %\n" a  
)
```

-- a배열과 b배열을 합친다.

a: #(1, 5, 7, 55, 22, 33) -- 출력 결과

sort <array>

```
(  
    a = #(7,5,1,55,22,11,3,9)  
  
    sort a  
    format "a: %\n" a  
)
```

-- a배열을 정렬한다.

a: #(1, 3, 5, 7, 9, 11, 22, 55) -- 출력 결과

제어문

if 문

```
if <expr> then <expr> [else <expr> ]  
if <expr> do <expr>
```

연산자는 아래와 같습니다.

관계 연산자	>	좌항이 우항보다 크다.
	<	우항이 좌항보다 크다.
	==	같다.
	!=	같지 않다.
	>=	좌항이 우항보다 크거나 같다.
	<=	우항이 좌항보다 크거나 같다.
논리 연산자	and	좌항과 우항이 전부 만족해야 True
	or	좌항과 우항 중 어느 한쪽만 만족하면 True
	not	우항이 true면 false 리턴, 우항이 false면 True를 리턴


```
a = 10
if a < 20 do print "small"      -- 조건이 True이므로 'Small' 출력
"small"                        -- 실행 결과
```

```
a = 10
if a == 50 do print "ok~"      -- 조건이 False이므로 'Ok'를 출력하지 않는다.
undefined                      -- 실행 결과
```

if 문은 뒤에 else를 붙일 수 있는데 이때는 do 대신 then을 사용해야 합니다.

```
a = 30
if a<20 then
    print "A"                  -- 조건이 True이면 A로 분기
else
    print "B"                  -- 조건이 False이면 B로 분기
"B"                            -- 실행 결과
```

```
(
    a = 50
    if a<10 then print "A"
    else if a<30 then print "B"
    else if a<60 then print "C"
    else print "nothing"
)
"C"                            -- 실행 결과
```

If의 조건에 만족하는(True) 결과가 2줄 이상이면 반드시 괄호를 사용해야 합니다. 물론 앞의 예제처럼 한 줄이면 괄호는 사용하지 않아도 됩니다.

```
(
    text1 = "someday"
    if text1 == "someday" do
        print "A"                                -- 괄호가 없는 경우 다음 줄까지 영향을 받는다.

    if text1 == "someday" do
        (                                          -- 두 줄 이상인 경우에는 괄호를 넣는다.
            text1 += ", goodday"
            print text1
            ok
        )
    )
)
```

-- 실행 결과

"A"

"someday, goodday"

OK

if 문은 and, or, not의 논리 연산자를 조합해서 사용할 수 있습니다.

```
if 11 == 12 and 5 == 5 then
    print "Yes"
else
    print "No"
```

"No"

-- 실행 결과

```
if 11==12 or 5==5 then
    print "Yes"
else
    print "No"
```

"Yes"

-- 실행 결과

if 문은 마지막에 결과를 리턴하기도 합니다.

```
str1="dog"
str2="ani"
D = if str1 != str2 do
(
    print "my dog !!"
    55
)

format "D: %\n" D
```

-- 실행 결과

"my dog !!"

D: 55

for 문

for 문은 제어문 중에서 가장 강력하고 많이 쓰이는 방법입니다. for 문은 반복문이라고도 불리는데 '몇 번째부터 몇 번째까지 반복해서 수행해라'라고 명령할 수 있습니다. for 문의 사용법은 다음과 같습니다.

```
for <var_name> ( in | = ) <sequence> ( do | collect ) <expr>
```

여기서

```
for i in 1 to 10 do
```

대신,

```
for i = 1 to 10 do
```

이렇게 사용해도 됩니다.

```
for i in 1 to 10 do          -- 1부터 10까지 i를 출력한다.
  format "% " i
1 2 3 4 5 6 7 8 9 10 OK      -- 실행 결과
```

이번에는 1부터 100까지 더하는 Script를 작성해보겠습니다.

```
add=0
for i in 1 to 100 do        -- 1부터 100까지 돌면서 add에 숫자를 증가시킨다.
  add += i
format "1부터 100까지 합: %\n" add
1부터 100까지 합: 5050      -- 실행 결과
```

기본적으로 for 문은 순서대로 1씩 증가하면서 수행하지만 조건식을 수정하는 방법으로 몇 칸씩 뛰어넘거나 거꾸로 루프를 돌릴 수 있습니다.

```
for i in 1 to 10 by 2 do      -- 1부터 10까지 2칸씩 수행한다.  
    format "% " i
```

```
1 3 5 7 9 OK                -- 실행 결과
```

```
for i in 10 to 1 by -1 do    -- 10부터 1까지 거꾸로 수행  
    format "% " i
```

```
10 9 8 7 6 5 4 3 2 1 OK     -- 실행 결과
```

exit를 사용하면 도중에 루프를 탈출할 수 있습니다.

for 문을 사용하여 루프를 끝까지 돌릴 필요가 없는 경우에 사용합니다.

```
for i in -10 to 100 do      -- -10부터 100까지 루프  
(  
    if i > 0 do exit         -- 0보다 크면 for 문을 탈출한다.  
    format "% " i  
)
```

for 문은 배열과 함께 사용되면 매우 효과적인 능력을 발휘합니다.

이때 collection이 배열이라면 배열 인덱스는 0이 아니라 1부터 시작해야 합니다.

```
myarray = #(1,3,5,9,22)
for i in 1 to myarray.count do
  format "% " myarray[i]
```

1 3 5 9 22 OK -- 실행 결과

for 문을 collect와 함께 사용하면 for 문의 마지막 라인을 배열에 모을 수 있는 기능이 있습니다.

```
ar = for i in 1 to 20 where mod i 3 == 0 collect
      i
format "ar: %\n" ar
```

ar: #(3, 6, 9, 12, 15, 18) -- 실행 결과

Collect의 사용이 매우 흥미로운데 간단한 문법으로 코드를 짧고 보기 쉽게 만들어 줍니다.

Collect 사용을 확실히 이해하기 위해 다른 예제를 살펴보겠습니다.

```
myarray = #(55,33,22,11,3)
myCollect = for i in myarray collect
(
  m = i * i
  m
  -- 마지막 라인이 collect 하게 된다.
)
```

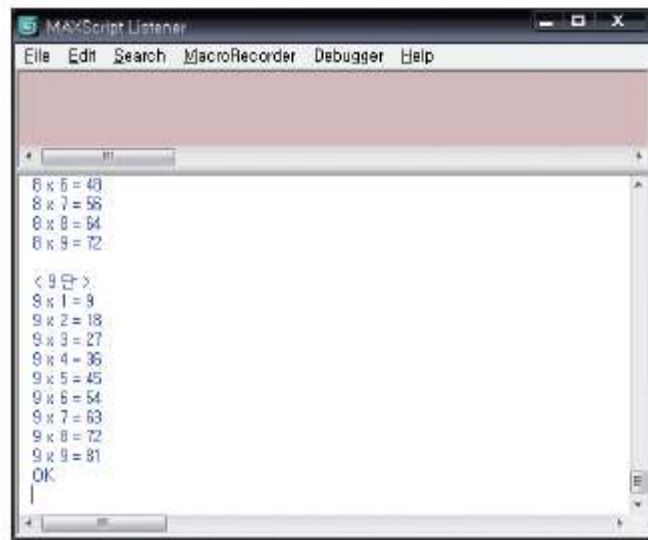
```
format "myCollect --> %\n" myCollect
```

myCollect --> #(3025, 1089, 484, 121, 9) -- 실행 결과

예제 01

간단하게 구구단을 출력하는 프로그램을 만들어 보겠습니다.

먼저 구구단을 만들려면 첫 번째 숫자와 두 번째 숫자를 곱하여 구구단이 만들어지기 때문에 이중 for 문으로 하는 것이 가장 좋습니다. 그러므로 첫 번째 for 문은 첫 번째 숫자를 증가시키는 일을 담당하고, 두 번째 for 문은 두 번째 숫자를 증가시키는 일을 담당해서 작성하면 되는 것입니다.



구구단의 출력 결과

```
minDan = 1
maxDan = 9

for i in minDan to maxDan do
(
    format "\n"
    format "< % 단 >\n" i
    for j in 1 to 9 do
        format "% x % = %\n" i j (i*j)
)
```

while 문

while 문은 for 문과 같이 일을 반복하여 수행하는 역할을 합니다. 하지만, while 문은 어느 특정 조건이 만족할 때까지 무한정 반복합니다. 그래서 while 문은 종료 조건을 만들어 주는 것이 중요합니다. while 문은 for 문보다 사용 빈도는 낮지만 때때로 요긴하게 사용됩니다.

while 문의 사용 방법은 다음과 같습니다.

```
do <expr> while <expr>
while <expr> do <expr>
```

```
a = 0
while a < 10 do                -- a가 10보다 작으면 계속 반복 수행
(
    a += 1
    format "%d " a
)
```

```
1 2 3 4 5 6 7 8 9 10 OK
```

-- 실행 결과

예제 02

Box를 만들어 층층이 쌓아 피라미드를 만드는 Script를 만들어 보겠습니다.

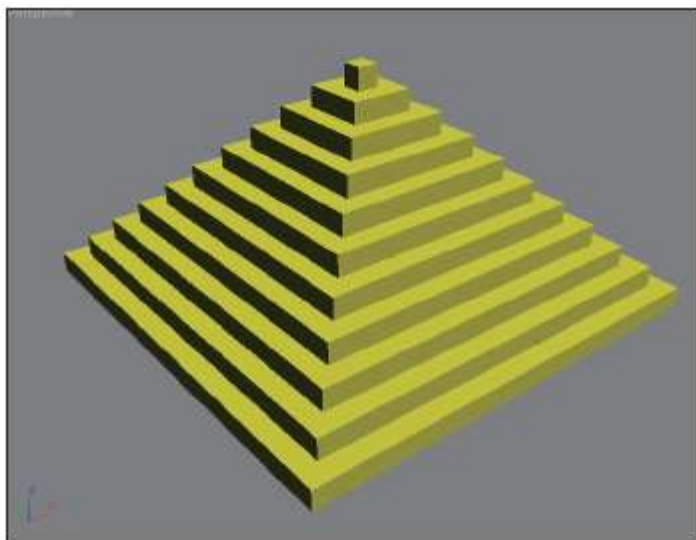
가장 먼저 맨 아래 기준이 되는 넓이를 결정하고 box를 만듭니다. 넓이를 줄여가면서 또다시 box를 계속 만듭니다. 점점 넓이가 작아지는 box를 만들다가 0보다 작아지면 프로그램이 종료되는 프로세스입니다.

이 Script는 맨 위에 baseWidth, baseHeight 변수만 바꾸면 피라미드 크기가 변합니다.

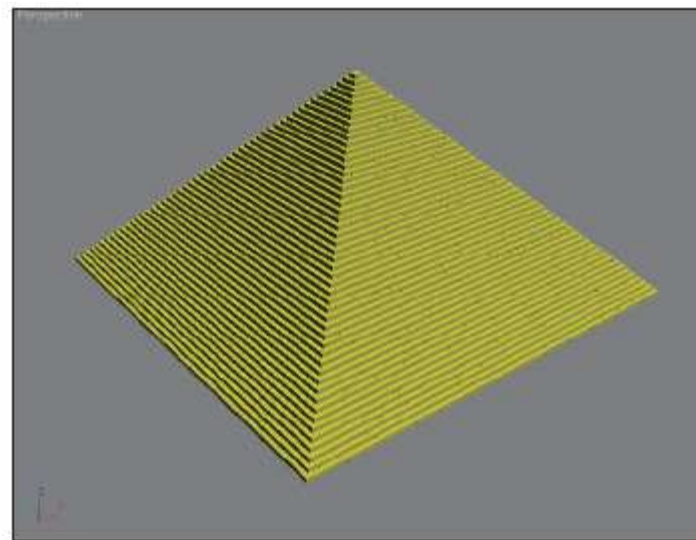
```
baseWidth = 210          -- 맨 아래 box 넓이
baseHeight = 10          -- box 높이

w = baseWidth
count = 0
hPos = 0
while w > 0 do
(
    -- 루프 중 [Esc] 누르면 탈출
    if keyboard.escPressed do exit

    -- box 생성(색상은 노랑색)
    box width:w length:w height:baseHeight pos:[0,0,hPos] wirecolor:yellow
    w = w - baseHeight * 2      -- 현재 box의 넓이를 계산
    count +=1                  -- 계속 증가하는 인덱스
    hPos = count * baseHeight  -- 현재 box의 높이를 계산
)
```



baseWidth=210 Script 실행 결과



baseWidth=1000 Script 실행 결과

주석문(Comments)

```
clearListener()      -- script Listener를 깨끗이 한다.  
a1 = 10              -- 젤 처음에 만든 변수  
b2 = 20              -- 두 번째 변수
```

파일 경로(Path)의 표현

```
dirA = "C:\\\\Program Files\\\\Autodesk\\\\3ds Max 2013 \\\3dsmax.exe"  
dirB = "C:/Program Files/Autodesk/3ds Max 2013 /3dsmax.exe"
```

그 밖에 코딩에 필요한 것들

다음 두 예제는 완전히 같습니다. 한번 살펴봅시다.

```
a = 10.0
d = 20.0
val = a / d
```

-- 첫 번째 예제

```
a = 10.0; d = 20.0; val = a / d
```

-- 두 번째. 위와 동일한 결과

만약 코딩할 때 한 줄이 너무 길어서 다음 줄로 내리고 싶다면 어떻게 해야 할까요?
바로 그 라인 끝에 `\n`을 붙이면 됩니다.

```
str = "good day~"
format "wtwthello world ~ \n\nwtwt%\nwt ~%\n\n" pi w
str
```

-- 출력 결과

```
hello world ~
```

```
3.14159
```

```
~good day~
```

수학 함수들

abs <number>

절대값을 구합니다.

ceil <number>

숫자를 올림합니다.

floor <number>

숫자를 내림합니다.

mod <number1> <number2>

<number1>과 <number2>를 나누어서 나머지를 구합니다.

atan2 <number> <number>

삼각 함수입니다. 각도는 degree를 사용합니다.

acos <number>

삼각 함수입니다. 각도는 degree를 사용합니다.

cos <number>

삼각 함수입니다. 각도는 degree를 사용합니다.

asin <number>

삼각 함수입니다. 각도는 degree를 사용합니다.

cosh <number>

삼각 함수입니다. 각도는 degree를 사용합니다.

atan <number>

삼각 함수입니다. 각도는 degree를 사용합니다.

exp <number>

지수 함수입니다.

sin <number>

삼각 함수입니다. 각도는 degree를 사용합니다.

sqrt <number>

숫자를 제곱근(루트) 합니다.

log <number>

로그입니다.

random <number> <number>

<number>와 <number> 사이에서 랜덤 값을 구합니다.

pow <number> <number>

숫자를 거듭제곱합니다.

random 0 100

함수의 기초

함수의 사용 형식은 다음과 같습니다.

```
[mapped] (function | fn) <name> { <parameter> } = <expr>
```

<parameter>

<name>

<name>: [<operand>] --keyword parameter

```
function add a b = -- add 함수
```

```
(
```

```
    val = a + b
```

```
)
```

```
v = add 10 20 -- add 함수 호출
```

```
30
```

```
-- 실행 결과
```

Function 단어는 줄여서 fn으로 많이 쓰입니다.

```
-- 평균을 구하는 함수
fn average  nums =
  (
    -- 배열의 숫자를 전부 더하려고 만든 변수(float 에 유의하자)
    add = 0.0

    for i in nums do
      add += i

    result = add / nums.count
    result
  )

av1 = average #(40,120)
av2 = average #(50,22,100,125)

80.0
74.25
```

-- 첫 번째 실행 결과

-- 두 번째 실행 결과

만약 함수 중간 부분에서 리턴해야 한다면 리턴 키워드와 해당하는 값을 붙여서 리턴하면 됩니다. 리턴 문은 값을 반환한다는 의미도 있지만 해당 함수를 더 이상 수행하지 않고 되돌아간다는 의미도 있습니다.

```
fn abc val =  
(  
    if mod val 2 == 0 do          -- val이 짝수이면 true를 리턴  
        return true  
    false                        -- 위에 조건이 만족하지 않았을 경우  
)  
abc 10                          -- abc 함수를 호출  
abc 11                          -- abc 함수를 호출  
  
true                            -- 첫 번째 실행 결과  
false                          -- 두 번째 실행 결과
```


만약 여러 개의 리턴 값을 가지고 싶다면 다음과 같이 값을 배열에 담아서 리턴하는 방법을 추천합니다.

```
-- 문자를 하나씩 배열로 분리시키는 함수
fn splitString str =
(
    strArray = #()                                -- 결과를 담을 배열

    for i in 1 to str.count do                    -- for문을 돌며 문자를 분리
        append strArray str[i]

    strArray                                        -- 배열을 최종적으로 리턴
)

sa = splitString "the book"                       -- 함수 호출

#("t", "h", "e", " ", "b", "o", "o", "k")         -- 실행 결과
```

직접 만든 함수를 미리 정의해놓고 3ds Max를 필요할 때마다 불러 재활용하는 것이 가능할까요?

물론 가능합니다. 함수로만 이루어진 ms 스크립트 파일을 startup에 넣으면 됩니다.
(예 : C:\WProgram Files\Autodesk\3ds Max 2021\WScripts\Startup)

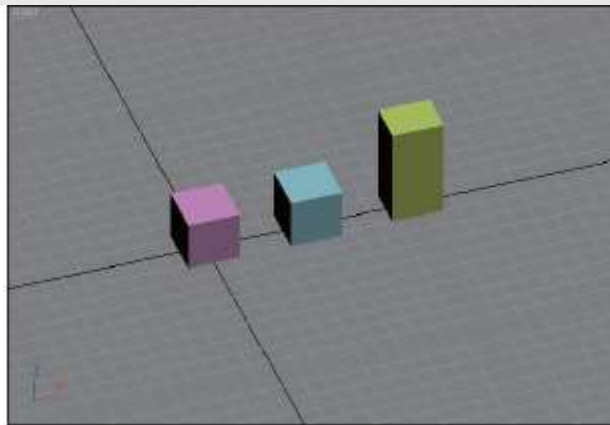
옵션 파라미터(Keyword parameters)

사용법은 파라미터 뒤에 ':' 를 붙이고 기본 값을 넣어두면 됩니다. 이 파라미터는 사용하지 않으면 미리 정의된 기본 값이 자동으로 들어가게 됩니다.

```
fn make_name str first:"big_" last:"_car" =  
(  
    first + str + last  
)  
  
make_name "red"  
make_name "green" first:"small_"  
make_name "blue" first:"my_" last:"_airplane"  
  
"big_red_car"           -- 첫 번째 실행 결과  
"small_green_car"       -- 두 번째 실행 결과  
"my_blue_airplane"      -- 세 번째 실행 결과
```

많이 사용하지는 않지만 함수 function 앞에 mapped를 붙일 수 있는데 이것은 해당 collection에 자동으로 루프를 돌리는 효과를 발휘합니다. Mapped를 사용하게 되면 for 문을 한 번 돌릴 것을 절약시켜 주는 것과 같습니다.

```
box()  
box pos:[50,0,0]  
box pos:[100,0,0] length:20 height:50  
  
mapped fn rand_color x =  
    x.wireColor = random (color 0 0 0) (color 255 255 255)  
  
rand_color objects -- 모든 오브젝트에 wirecolor를 랜덤으로 바꾼다.
```



문자열(String)

Copy

```
<string>copy <string>
```

문자열을 복사합니다. 복사된 문자열은 리턴됩니다.

```
str = "dream come true"  
newStr = copy str
```

Append

문자열과 문자열을 합쳐줍니다. 배열의 append와 같은 기능입니다.

```
ss = "oh! my "  
append ss "love"
```

"oh! my love"

-- 실행 결과

Findstring

```
<integer>findString <string> <search_string>
```

문자열에서 특정 문자를 찾아주는 함수입니다. 이때 찾은 위치가 리턴됩니다.

```
str = "have a good time"  
find = findstring str "good"
```

8 -- 실행 결과

Execute

```
exe = "box pos:[0,0,10] "  
execute exe                    -- box가 [0,0,10] 위치에 생성된다.
```

