Lua를 다루는 법

목차

[오늘부터 게임개발] 시리즈에서는 **Lua**라는 프로그래밍 언어를 사용해서 게임을 개발합니다. 이 부록에서는 그 Lua를 다루는 방법을 다루고 있습니다.

여러분이 다른 공부를 할 때처럼 모든 것을 암기할 필요가 없습니다. 간단하게 살펴본 다음에, 나중에 필요할 때 다시 찾아서 살펴보는 과정을 여러 번 겪으면 자연스럽게 여러분에게 상식으로 자리잡게 될 것입니다.

★

0️⃣1️⃣ print()

print()란?

Solar2D Simulator에서 프로젝트를 생성하면 게임을 다운받아 플레이할 때 보여질 화면인 시뮬레이터 창과 개발자만 볼 수 있는 콘솔창 두 개의 창이 화면에 띄워지게 됩니다. print()는 그 중 콘솔창에 메시지를 출력할 수 있게 해주는 기능입니다.

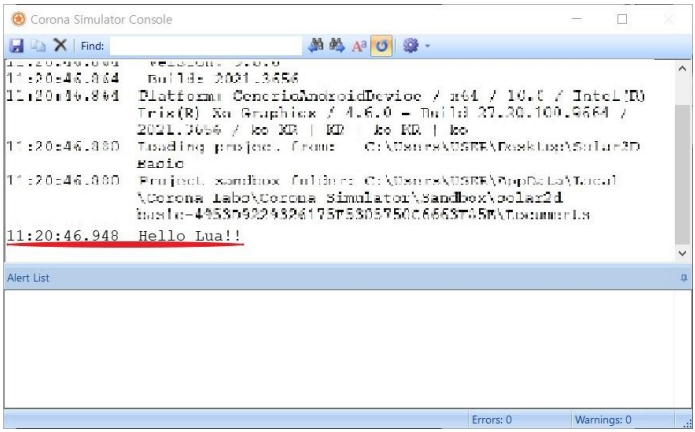
개발자만 볼 수 있는 창인 콘솔창에 메시지를 출력하기 때문에, print()는 게임을 만들고 있는 개발자만을 위한 기능이기도 합니다.

print() 사용 방법

```
print( “Hello Lua!!” )
```

print( ) 안에 출력하기를 원하는 문장을 “ ”(큰 따옴표) 혹은 ‘ ’(작은 따옴표)로 묶어 넣어주면 됩니다.

실행 결과



▲ 콘솔창에 “Hello Lua!!”를 출력한 모습

콘솔창에 알 수 없는 영어들이 많지만, 우리가 출력한 문장만 찾아보도록 합니다. 앞으로는 콘솔창 캡처 화면 없이 출력 결과만 담았습니다.

02 주석

주석이란?

주석은 코드 중간에 메모를 작성하기 위해서 사용합니다. 복잡하고 긴 코드를 작성하게 될 때나 다른 친구와 같이 게임을 만들 때 코드 중간중간 주석을 사용하면 코드를 한눈에 알아보기 쉬워집니다.

또한 작성한 코드를 주석으로 만들어주면, 해당 코드는 프로그램 실행 시 작동하지 않습니다. 주석처리도 프로그램 실행과는 전혀 상관이 개발자의 편의성을 위한 기능입니다.

주석 사용 방법

```
1| -- 이건 주석입니다.
2| -- 여기에 쓰는 코드는 실행되지 않습니다.
3| -- print("hello Lua!!")
4| print("hello Solar2D!!")
```

앞에 --만 달아주면 그 뒷부분은 주석으로 처리됩니다.

실행 결과

```
Hello Solar2D!!
```

주석처리된 3| 코드는 실행되지 않아 "hello Lua!!"가 출력되지 않았고, 4| 코드만 실행이 되어 "hello Solar2D"만 출력이 되는 것을 확인할 수 있습니다.

★★★
03 변수

변수란?

프로그래밍에서 변수란 변하는 어떤 값을 담을 수 있는 상자라고 볼 수 있습니다. 게임을 진행하면서 계속 변화하는 어떤 값(Ex. 체력, 점수 등)를 변수에 담아뒀다가 사용하게 됩니다. 변수 안에 값이 계속 바뀌더라도 우리는 코드를 작성할 때 그 값을 담는 변수만 지칭하면 우리가 원하는 값을 사용할 수 있습니다.

변수 사용 방법

```
1| local HP       -- 변수 선언
2| HP = 100       -- 변수에 값 대입
3| local MP = 100       -- 변수 선언과 동시에 값 대입
4| local bg = display.newImage("img/bg.png")
```

변수를 사용하려면 먼저 선언을 한 뒤에 사용해야 합니다. 1| 첫번째 줄 코드처럼 앞에 local을 붙이고 원하는 변수의 이름을 지정해주면 해당 이름으로 된 변수 상자가 생성이 되어, 뒤에서 변수를 사용할 수 있게 됩니다.

2| 코드처럼 변수를 선언한 뒤에 =을 사용해서 변수에 값을 넣어줄 수 있습니다. 여기서 =는 '오른쪽의 값을 왼쪽의 변수에 넣어라(값 대입)'라는 의미입니다.

3| 코드처럼 변수의 선언과 대입은 한 줄로 동시에 작성할 수 있습니다.

4| 코드는 본편에서 사용해봤던 display object를 생성해서 변수에 담아두는 코드입니다.

변수에 담을 수 있는 값의 종류

변수에 담을 수 있는 값의 종류로는 대표적으로 정수(-1, 0, 1 ..), 실수(소수)와 문자열("hello Lua!!"), 변수가 비어있음을 뜻하는 **nil**이 있습니다.

부울(true/false)도 있지만, 이는 따로 **06 true/false**에서 다룹니다.

★★★

04 테이블

테이블이란?

테이블은 번호표(index)를 가지는 변수들의 집합입니다. 같은 계열의 데이터들을 좀 더 편리하게 저장하고 관리하기 위해서 사용합니다.

index는 1부터 시작하는 정수로 사용하면 되고, **08 반복문**과 함께 사용하면 더 간편하게 코드를 작성할 수 있습니다.

테이블 사용하기

```
1| local score = { }     -- 테이블 변수 선언
2| score[ 1 ] = 95
3| score[ 2 ] = 25
```

테이블도 일종의 값이므로, 1| 코드처럼 테이블을 담을 변수를 먼저 선언해야 합니다. 그 다음 변수 이름[index] 형식으로 여러 개의 변수를 사용할 때마다 선언할 필요없이 사용할 수 있습니다.

★★★

05 연산자

연산자란?

연산자를 이용해 값을 변화시킬 수 있습니다. 연산자에는 우리가 익숙한 수학 연산자(+ - * / 등)과 관계 연산자, 논리 연산자 그 외에 몇 가지 연산자가 있습니다. 여기서는 간단한 수학 연산자와 기타 몇 가지 연산자만 살펴봅니다.

연산자의 종류

수학 연산자

더하기	+	20 + 8 == 28
빼기	-	20 - 8 == 12
곱하기	*	20 * 8 == 160
나누기	/	20 / 8 == 2.5
나머지	%	20 % 8 == 4
제곱	^	2^4 == 16

기타 연산자

..	문자열을 합쳐주는 연산자
#	테이블의 길이를 알려주는 연산자

연산자 사용하기

```
1| local N = {}
2| N[1] = 10
3| N[2] = 8
4|
5| print( "N[1] + N[2] = " .. N[1] + N[2] )
6| print( "N[1] - N[2] = " .. N[1] - N[2] )
7| print( "N[1] * N[2] = " .. N[1] * N[2] )
8| print( "N[1] / N[2] = " .. N[1] / N[2] )
9| print( "N[1] % N[2] = " .. N[1] % N[2] )
10| print( "테이블의 길이 = " .. #N )
```

실행 결과

```
N1 + N2 = 28
N1 - N2 = 12
N1 * N2 = 160
N1 / N2 = 2.5
N1 % N2 = 4
테이블의 길이 = 2
```

연산자를 사용해본 예시 코드입니다.

문자열과 값을 .. 연산자를 이용하여 한 줄로 출력했고, 테이블의 길이를 알려면 10| 코드처럼 # 연산자를 테이블 변수 앞에 붙여주면 알 수 있습니다.

★★★★

06 true/false

부울이란?

변수에 담을 수 있는 값의 종류 중 하나입니다. 부울 값은 true(참) 또는 false(거짓)의 값을 가지게 됩니다. 이 부울 값은 후에 알고리즘을 위한 **07 조건문**, **08 반복문**에서 사용하게 됩니다.

부울 사용하기

```
1| local bool1 = true
2| local bool2 = false
3| print( bool1 )
4| print( bool2 )
```

실행 결과

```
true
false
```

관계연산자와 논리연산자

부울 값은 보통 true/false라고 바로 입력해서 사용하기보다는 **07 조건문**, **08 반복문** 등에서 조건식에 주로 사용하게 됩니다. 관계연산자와 논리연산자를 이용해 조건식을 생성할 수 있고, 조건식은 해당 식이 참인지 거짓인지에 따라 true / false 값을 갖게 됩니다.

관계 연산자 (A, B는 정수 or 실수)

A > B	A가 B보다 크다
A < B	A가 B보다 작다
A >= B	A가 B보다 크거나 같다
A <= B	A가 B보다 작거나 같다
A == B	A와 B의 값이 같다
A ~= B	A와 B의 값이 같지 않다

논리 연산자 (A, B는 true or false)

A **and** B	A와 B가 모두 true면 **true**
A **or** B	A와 B 중 하나라도 true면 **true**
not A	A가 false면 **true**

부울 사용하기

```
1| local N = 10
2| local M = 20
3| print( N == M )
4| print( N > M or N < M )
```

실행 결과

```
false
true
```

★★★

07 조건문

조건문이란?

조건문은 해당 시점에 지정된 조건식이 true일 경우에 실행할 부분을 작성할 수 있게 해줍니다. 조건식은 if를 사용해서 작성하기 때문에 if문이라고도 합니다.

조건문 사용하기

+) 삽화 추가 예정

```
1| if (조건식) then
2|     -- 조건식이 true면 실행할 코드
3| end
```

```
1| local N = 10
2| local M = 10
3|
4| if ( N == M ) then
5|     print( "N과 M의 값이 같다." )
6| end
```

실행 결과

```
N과 M의 값이 같다.
```

조건식은 **if (조건식) then ~ end** 형식을 사용해서 작성합니다. then과 end 사이에 조건식이 true이면 실행할 코드들을 작성하면 됩니다.

```
1| if (조건식 A) then
2|     -- 조건식 A가 true일 때 실행할 코드
3| elseif (조건식 B) then
4|     -- 조건식 A가 false, 조건식 B가 true일 때 실행할 코드
5| else
6|     -- 조건식 A와 조건식 B가 모두 false일 때 실행할 코드
7| end
```

elseif를 사용해서 앞에 조건식이 false일 경우에 실행할 새로운 조건문을 작성할 수 있고, **else**를 이용해서 모든 조건식이 false일 때 실행할 코드들을 지정할 수 있습니다.

★★★

08 반복문

반복문이란?

반복문은 조건식이 성립하는 동안 반복해서 실행할 코드를 작성할 수 있게 해줍니다. 반복문은 while문과 for문 두 가지 종류가 있습니다.

while문

```
while (조건식) do
    -- 조건식이 true일 동안 실행할 코드
end
```

while문은 간단히 while (조건식) do ~ end 형식을 해서 작성합니다. Do ~ end에 조건식이 true일 때 반복할 코드를 작성해주면 됩니다.

for문

```
for i = start, end, add do
    -- i가 end값에 도달할 때까지 반복할 코드
end
```

+) 삽화 추가 예정

for문은 while문보다 조금 복잡합니다. 변수 i가 start에서 시작해서 add씩 더해서 end에 도달할 때까지 돌아가는 반복문입니다. 한번 코드를 돌때마다 i에 add 값만큼 더해지게 되고, 이때 add가 1이라면 for문을 작성할 때 생략해도 됩니다.

N번 반복하는 반복문 예시

```
local i = 1
while i <= N do
    -- N번 실행할 코드
    i = i + A
end
```

```
for i = 1, N do
    -- N번 실행할 코드
end
```

add값이 1이므로 생략

★★★★

09 함수

함수란?

함수는 특정 기능을 하도록 미리 짠 코드를 불러와서 사용할 수 있는 기능입니다. 게임을 개발하다가 필요한 기능을 편리하게 함수로 정의해서 그 기능이 필요할 때마다 그 함수를 불러와서 사용할 수 있습니다.

함수 선언하기

```
local function function_name ( … )
    -- body
end
```

함수 사용하기

```
function_name( … )
```

함수를 선언하기 위해서는 **local function**으로 시작해서 **함수 이름( )**을 작성해줍니다. 여기서 괄호 안에는 **매개변수**들을 작성할 수 있습니다. 매개변수는 함수를 실행할 때 전달할 수 있는 변수로, 함수를 사용할 때 괄호 안에 값을 넣어주면 매개변수의 이름으로 함수 내에서 사용 할 수 있습니다. 그리고 원하는 함수의 내용들을 작성한 뒤, **end**로 함수의 끝을 표기해주면 됩니다.

이렇게 함수를 선언한 뒤에는 **함수 이름( )** 형식으로 코드를 작성하여 함수를 불러와 실행할 수 있습니다.

함수를 선언할 때, 반환되는 값을 설정할 수 있습니다. 함수 안에 **return [값]** 형태로 코드를 작성해주면, 함수가 종료되면서 함수 자체가 **[값]**을 가지게 됩니다. 아래 코드들은 사용 예시입니다.

함수 선언하기

```
local function add10( N )
    return N + 10
end
```

함수 사용하기

```
local num = 20
local sum = add10( num )
print( sum )
```

+) 게임을 제작할 때 유용한 math.random()

math 라이브러리의 math.random( )를 이용하면 랜덤으로 생성된 값을 가져올 수 있습니다. math.random( )는 매개변수를 몇 개 넣어주냐에 따라 조금씩 다르게 사용할 수 있습니다.

math.random() 사용하기

```
1| math.random( )          -- 매개변수 없이 사용
2| math.random( n )        -- 매개변수 하나 사용
3| math.random( n, m )     -- 매개변수 두개 사용
```

1| 처럼 매개변수 없이 사용할 경우, [0, 1] 사이의 실수(소수)가 생성됩니다.
2| 처럼 매개변수를 하나 사용할 경우 [0, n] 사이의 정수 값이 생성됩니다.
3| 처럼 매개변수를 두개 사용할 경우 [n, m] 사이의 정수 값이 생성됩니다.

★★★

10 변수/함수의 사용 범위

코드 블록의 개념

07~09에서 학습한 조건문, 반복문, 함수들은 모두 **~end** 형식으로 이루어져 있습니다. 이 end의 의미는 해당 조건문, 반복문, 함수 안에 있는 코드들을 묶어주는 역할을 하며, 그 묶인 코드들을 **코드 블록**이라고 하겠습니다. 코드 블록 안에서 또다시 조건문, 반복문, 함수들이 쓰인다면 코드 블록 속에 코드 블록이 생길 수 있습니다.

```
1| local function factorial ( N )
2|     local num = 1
3|     for i = 1, N do
4|         num = num * 1
5|     end
6|     return num
7| end
```

local의 개념

앞에서 변수와 함수를 선언할 때 local를 붙여서 선언해주었습니다. 이때 local은 해당 변수와 함수가 선언된 코드 블록 안에서만 사용할 수 있음을 의미합니다. 이때 코드 블록 안에 또 다른 코드 블록에서도 local 변수와 함수를 사용할 수 있습니다.

잘못된 사용의 예시

```
1| local function addTwoNumbers ( N, M )
2|     local sum = N + M
3|     print( sum )
4| end
5|
6| print( sum )
```

```
1| local function addTwoNumbers ( N, M )
2|     local sum = N + M
3|     if sum < 10 then
4|         local secondSum = sum + 10
5|     end
6|     print( secondSum )
7| end
```