1. CoronaSDK?

이 부분은 CoronaSDK의 장점..? 특징을 나열하는 부분이예요

- 무료 버전으로 개발 및 판매가 가능하다
- 생산성이 뛰어나다
- 퍼포먼스가 뛰어나다
- 다양한 디바이스로 연동이 가능하다
- 쉬운 Lua 기반이다
- 광범위한 기능을 구현할 수 있다
- Corona Enterprise, Cards로 여러 네이티브 라이브러리와 연동이 가능하다
- 매일 매일 라이브러리가 업데이트 된다

  공식 웹사이트 : http://coronalabs.com

  Docs.CoronaLabs : http://docs.coronalabs.com

2. Hello, World! 출력

```
display.newText("Hello, World!")
print("Hello, World!")
```

3. 루아의 몇 가지 TIP

비교적 쉬운 언어예요:)
루아는 대소문자를 구별합니다.

```
local hi = 1; -- 'hi'와 'Hi'는
local Hi = 10; -- 다른 변수입니다.
```

'--' 와 '--[[ ]]--' 를 사용해서 주석을 달 수 있어요

```
-- 한 줄 주석입니다.
--[[
여러 줄에 걸친 주석입니다.
]]--
```

코드에 마지막에 세미콜론을 붙여도 되고, 붙이지 않아도 되어요.

```
print("Hi") -- 하지만 대부분 붙이지 않는답니다.
print("Hi");
```

4. 변수와 타입

루아의 변수는 다이나믹(Dynamic) 타입이라 어떤 값이라도 변수에 담을 수 있어요!

루아에서 변수는 '지역 변수'와 '전역 변수'로 구분되어요.

```
a = 10 — globa : 전역변수는 선언할 필요 없이, 바로 값을 할당해버리면 된다.
local b = 15 — local :일반적으로 함수나 코드 블록에서 호출된다.
```

지역 변수와 전역 변수에 대해서 이해가 조금 더 빠르도록 코드를 하나 더 준비했습니다!

```
a = "a"
do
  local b = "b"
  c = "c"
end
print( "1 : ".. a )
print( "2 : ".. toString(b) )
print( "3 : ".. c )
```

```
-- 결과
1 : a
2 : nil
3 : c
```

위에서 print(b)가 아닌, print(tostring(b))를 사용한 이유는
b의 값이 nil(값 없음)이기 때문에 프로그램 오류가 발생하기 때문입니다.

전역 변수를 쓰면 코드의 어느 곳에서나 사용할 수 있기 때문에 편리해 보여요.
하지만 대부분의 프로그래머들은 전역 변수를 최소한으로 사용하고, 지역 변수를 쓰라고 조언해주셔요.
왜 그럴까요?

- 다른 코드에서 접근하지 못하도록 막기 위해서! (함수를 이용해 읽기, 쓰기만 가능하도록)
- 메모리를 절약하기 위해서!

예시를 하나 보여드릴게요.

```
display.newImage( “Icon.png”, 0, 0 )
```

display.newImage()는 이미지 파일을 읽어와서 화면에 보여주는 함수예요.
나중에 뒷부분에 가면 사용법에 대해서 제대로 배우겠지만,
변수의 사용 예시를 위해서 잠깐 꺼내왔어요.

우리는 사진을 화면에 표시했어요!
그런데 이 사진의 위치를 바꾸고 싶으면 어떻게 해야 할까요?

코드 아레에 “display.newImage(“icon.png” , 200 , 200) “를 추가하면 될까요?
막상 코드를 추가하면 사진의 위치가 바꾸는 게 아니라, 새로운 이미지가 추가되어 버리네요.

그래서 변수를 사용합니다!

```
img.x = 100
img.y = 100
img.width = 200
```

이렇게 변수를 사용하면 편리할 때가 많이 있어요.

+)
루아의 변수는 타입이 정해져 있지 않아서
실제 데이터도 타입이 없구나!하고 생각할 수 있어요.
하지만 그렇지 않아요!
루아의 데이터들도 타입이 있어요.

데이터 타입은 type() 함수로 알아볼 수 있어요!

```
print( type(“Hello, World!”) -- string
print( type(10.4*3) -- number
print( type( print ) ) -- function
a = { 1, 2, 3, 4, 5 }
print( type( a ) ) -- table
local b
print( type( b ) ) -- nil
```

각 타입마다 설명을 드릴게요.

1) nil (값 없음)
변수에 nil 값이 할당되면 아무런 값이 없다는 것을 뜻해요.
그런데 왜 굳이 이런 걸 만들었을까요?
여러 이유가 있지만, 가장 큰 이유는 "메모리 관리" 때문이예요.
루아에는 '가비지 콜렉터' 라는 메모리 관리자가 있는데,
메모리가 부족할 경우 필요 없는 데이터를 메모리에서 삭제해요.
변수에 nil 값을 할당하면 기존에 연결된 데이터는 메모리가 부족할 경우 완전히 삭제되어요.
nil은 메모리 최적화를 위해 꼭 필요하답니다

2) Boolean (참 / 거짓)
Boolean은 true와 false라는 두 가지 값만을 가져요.

```
a, b = true, false
```

3) String (문자열)
말 그래도 문자(글자)예요.
보통은 따옴표로 표현하지만, 여러 표현 방법이 있답니다

```
print("This is my string")
print('This is "your" String')
print([[Is this "your" 'string'?]]
```

결과.
This is my string.
This is "your" string.
Is this "your" 'string'?

4) Number (숫자)
숫자래요.

```
local a = 123
local b = 1.23
```

그 밖에도 테이블, 함수, 유저 데이터 등이 있어요.
나머지는 뒤에 나오면 알려드리도록 하져.

5. 연산자

변수의 값 등을 비교하거나 계산하기 위해 연산자를 사용할 수 있어요.

1) 산술 연산자

연산자	기능	예시
+	더하기	print(1+3) -- 4
-	빼기	print(3-1) -- 2
*	곱하기	print(2*4) -- 8
/	나누기	print(4/2) -- 2
%	나머지	print(5%3) -- 15
^	제곱	print(2^5) -- 32

2) 관계 연산자

연산자	기능	예시
a<b		print(1<3) -- true
a>b		print(1>3) -- false
a<=b		print(1<=3) -- true
a>=b		print(1>=3) -- false
a==b	둘이 같은가	print(1==3) -- false
a~=b	둘이 다른가	print(1~=3) -- true

3) 논리 연산자

연산자	기능	예시
and	그리고	a = 10 print(a > 0 and a < 10) -- false print(0 and 10) -- 10
or	또는	a = 20 print(a < 0 or a > 10) -- true print(1 or 2) -- 1
not	아니다	print(not nil) -- true print(not true) -- false print(not 3) -- false

4) 연결 연산자

연산자	기능	예시
..	연결	print("Hello, ".."World".."!")

5) 길이 연산자

연산자	기능	예시
#	길이 반환	print(#"Hello") -- 5 print(#"안녕") -- 6 a = { 1, 2, 3, 4, 5 } print(#a) -- 5

6. 조건문

우리는 말을 할 때 ‘ ~하면 A하고, 아니면 B 해’ 라는 말을 쓰기도 해요.
프로그래밍 언어도 동일해요.
영어로 옮기면 ‘if ~ then A else B end’ 이랄까요.

```
if 조건 then
    실행문
elseif 조건 then
    실행문
else
    실행문
end
```

루아에서의 조건문의 형식입니다.
우리의 지식과 비교해 보면 같으면서도 다른 형식이예요.

우리의 빠른 적응을 위해 활용을 해볼게요!
아래의 코드는 임의의 수가 홀수이면 홀수, 짝수이면 짝수라고 실행하는 코드입니다.
빈 칸을 열심히 채워서 코드를 완성해 주세요.

```
math.randomseed( os.time( ) ) -- 시간마다 다른 난수가 나오도록 하는 함수
local num = math.random( 1, 100 ) --
--[[
이 주석문 다음부터 채워주시면 됩니다.
]]--
```

답은 공개하지 않을거예요!
건투를 빌어요.

7. 반복문

어떠한 일을 반복하는 것을 말해요.
정해진 횟수만큼 할 수도 있고, 어떤 조건이 될 때까지 실행할 수도 있고, 무한히 반복할수도 있어요.

1) for문

정해진 횟수만큼 반복합니다!

```
for 시작값, 끝값, 증감 do
  실행문
end
```

```
for i = 1, 10, 1 do
  print( i )
end
```

결과.
12345678910

2) while문

해당 조건을 만족하는 한 계속 반복합니다!

```
while 조건 do
  실행문
end
```

```
while i <= 5 do
  print( i )
  i = i + 1
end
```

결과.
12345

3) repeat문

한 번 실행 후 조건을 검사하는 반복문!

```
repeat
  실행문
until 조건
```

```
a = true
repeat
  print("CONTINUE")
until a == true
print("END")
```

결과.
CONTINUE
END

8. 함수

주어진 값을 이용해 내부적으로 처리한 후 결과를 돌려주는 명령어!랄까요

```
function 함수명(매개변수)
  실행문
end
```

```
function plus( a, b )
  return a+b
end
print( add(3,2) )
```

```
결과.
5
```

함수의 특징을 알려드리지요.

1) 변수처럼 global과 local로 선언 가능합니다.

```
local fun = function()
end
```

2) 변수에 할당할 수 있어요. 추가로 매개변수에 함수를 넘겨줄 수도 있어요.

```
local fun = function()
end
```

3) 두 개 이상의 값을 돌려줄 수 있고, 타입이 달라도 되어요. 심지어 함수를 반환하기까지도..

```
local function get()
  return 1, “some”, 0.5
end
local a, b, c = get()
print( a, b, c )
```

```
결과.
1      some      0.5
```

4) 함수는 호출하는 코드보다 먼저 선언되어야 해요

```
hi()
local function hi()
end
```

9. 모듈

변수와 함수의 집합!

1) string 모듈

문자열과 관련된 함수들의 집합!

함수	기능	예시
string.byte()	문자를 문자 코드로 변환	print(string.byte("A")) -- 65
string.char()	문자 코드를 문자로 변환	print(string.char(65)) -- A
string.find()	해당 문자의 시작과 끝 위치 알아내기	print(string.find("Hello Corona World", "Corona") -- 7, 12
string.gsub()	문자열 바꾸기	local s = "Hello, Lua" s = string.gsub(s, "Lua", "Corona") print(s) -- Hello, Corona
string.upper()	모두 대문자로	s = string.upper(s) print(s) -- HELLO, CORONA
string.lower()	모두 소문자로	s = string.lower(s) print(s) -- hello, corona

2) math 모듈

수학과 관련된 함수들의 집합.

인수에 값만 넣으면 되어요.

함수	설명
math.abs()	절댓값 반환
math.acos()	a = cosθ 일 때, a값을 넣으면 θ(Radian) 값 반환
math.asin()	a = sinθ 일 때, a값을 넣으면 θ(Radian) 값 반환
math.atan()	a = tanθ 일 때, a값을 넣으면 θ(Radian) 값 반환
math.ceil()	소수점 올림 반환
math.cos()	cos 값 반환
math.deg()	Radian을 Degree로 변환한 값 반환
math.floor()	소수점 내림 반환
math.fmod()	나머지(%) 반환
math.log()	자연 로그 반환
math.log10()	상용 로그 반환
math.max()	가장 큰 값 반환
math.min()	가장 작은 값 반환
math.modf()	정수부와 소수부를 구분하여 나머지 반환
math.pi	원주율 상수
math.pow()	a를 n제곱한 값 반환
math.rad()	Degree를 Radian으로 변환한 값 반환
math.random()	무작위 난수 반환
math.randomseed()	무작위 난수의 발생 순서 설정.
math.round()	주어진 값의 반올림 반환
math.sin()	주어진 값의 sin 값 반환
math.sqrt()	주어진 값의 제곱근 반환
math.tan()	주어진 값의 tan 값 반환

2) os 모듈

운영체제와 관련된 함수들의 집합

– 날짜 관련 함수 : os.date()

```
local date = os.date("*t")

print( date.year, date.month, date.day )
print( date.hour, date.min, date.sec )
```

결과.

2016 8 15

15 23 16

format	설명
*t	현재 시간 구하기
%a	요일 (Mon, Tue, ...)
%j	365일 중 몇 번째? (001 - 365)
%A	요일 (Monday, Tuesday, ...)
%m	월의 숫자 표현 (01 - 12)
%b	달의 약식 표현 (Jan, Feb, ...)
%M	분의 표현 (00 - 59)
%B	달의 전체 표현 (January, ...)
%p	오전, 오후 표현 (AM, PM)
%c	날짜, 시간 표현 (08/15/16 15:20:45)
%S	초의 표현 (00 - 59)
%d	일의 표현 (01 - 31)
%U	1년 중 몇 번째 주 (00 - 53)
%H	시간 표현 (00 - 24)
%w	요일의 숫자 표현 (일요일 0, 토요일 6)
%I	시간 표현 (00 - 12)
%Y	년도의 표현 (1999, 2016)

– 시간 간의 차이 알아내기 : os.difftime()

```
local t = os.time()
local function on_Timer(e)
    print( os.difftime( os.time(), t ) )  -- print( os.time() - t )
end
timer.performWithDelay( 2000, on_Timer, 2 )
```

결과.

2

4

– 프로그램 종료하기 : os.exit()

실행되자마자 프로그램이 종료된다!

3) 모듈 만들기?!

```
-- Hi.lua 파일을 만든 후 입력합니다.
 local Hi = {}

 Hi.plus = function ( a, b )
        print( a + b )
 end

 Hi.minus = function ( a, b )
        print( a - b )
 end

 Hi.times = function ( a, b )
        print( a * b )
 end

 return Hi
```

```
-- main.lua 파일에 입력합니다.
hello = require "Hi"

hello.plus( 1, 3 )

hello.minus( 10 , 3 )

hello.times( 2, 4 )
```

이런 방식으로 사용해요.
만약 Hi.lua라는 파일을 MyModules라는 폴더에 보관했다면 아래와 같이 입력하세요.

```
hello = require “MyModules.Hi”
```

비슷한 기능의 모듈이 많을 경우에 따로 폴더를 만들어서 사용하는 것이 좋아요.

10. 코루틴

많은 사람들이 코루틴을 쓰레드라고 설명해요.
하지만 루아의 코루틴은 다른 언어의 쓰레드와는 달라요!
가장 큰 차이점은 동시에 실행되지 않는 다는 것이죠!
그래서 코루틴을 '함수 내부 코드의 순차적 실행 방법'이라고 이해하는 것이 좋을 거예요

```
local b = true
local function f1( )
  while b do
    print("OK")
  end
  print("END")
end

local function f2( )
  b = false
end

local c1 = coroutine.creat(fn1)
local c2 = coroutine.creat(fn2)

coroutine.resume(c1)
coroutine.resume(c2)
```

만약 아래의 두 줄이 실행되었다면 f1의 while문이 중지되었을거예요!
하지만 c2는 실행되지 않았습니다!
c2를 실행시키고 싶은데... 방법이 없을까요?
아닙니다. 방법은 있어요~
바로 coroutine.yield() 함수를 쓰는 것입니다.
yield는 양보하다라는 뜻을 가지고 있는데, 현재 루틴을 **잠시 중단**하고 다음 루틴이 실행되도록 해요.

f1 함수를 아래처럼 수정해 주세요.

```
local function f1( )
  while b do
    print("OK")
    coroutine.yield()
  end
  print("END")
end
```

자! 이렇게 수정을 해 주면, END가 출력될까요?

왜인지 END는 출력되지 않았어요!
아마 설명을 자세히 들으신 분들은 "잠시 중단"이라는 단어를 보셨을 거예요.
말 그대로 coroutine.yield( ) 함수는 print("OK")까지만 실행 한 뒤 잠시 중단된 상태예요.
따라서 다시 이어서 시작해야 while문에서 빠져나오고 print("END")를 할 수 있죠!
그래서 우리는 마지막 명령줄 다음에 **coroutine.resume(c1)**을 추가해야 해요.

```
local b = true
local function f1( )
  while b do
    print(“OK”)
  end
  print(“END”)
end

local function f2( )
  b = false
end

local c1 = coroutine.creat(fn1)
local c2 = coroutine.creat(fn2)

coroutine.resume(c1)
coroutine.resume(c2)
coroutine.resume(c1)
```

완성된 코드입니다!
한 번에 이해하기가 쉽지 않을 수 있어요!
하지만 당연한 부분이니 여러 번 읽어봐 주세요!
팁은 **“함수 내부 코드의 순차적 실행 방법”** 을 염두에 두시면 되어요.

아래의 코드는 이해를 돕기 위한 또 다른 코드입니다:)

```
local function fn( )
  print(1)
  coroutine.yield( )
  print(2)
  coroutine.yield( )
  print(3)
  coroutine.yield( )
end

local co = coroutine.create(fn)
coroutine.resume(co)
coroutine.resume(co)
coroutine.resume(co)
```

11. 테이블

어느덧 루아 기초도 끝이 나려고 해요.
테이블은 정말 간단하면서도 유용해요.
숫자 1부터 시작하는 숫자 인덱스를 이용해 배열처럼 사용해도 괜찮고,
키를 이용해 해시나 사전처럼 사용할 수도 있어요.

```
local table = { }
table[1] = 1
table[“five”] = 5
print( table[1] + table[“five”] )
```

결과.
6

테이블에는 모든 데이터 타입을 저장할 수 있어요.

```
local function hi( )
  print("Did you call me?")
end

local tab = { a = 3, b = "String", c = hi, d = math }
print( tab.c() )
print( tab.d.random() )
```

결과.
Did you call me?
0.091402935880612 (임의의 난수)

테이블은 각종 함수를 통해 다양한 조작을 할 수 있어요!
테이블의 함수들에 대해 살짝 살펴볼게요.

함수	기능	예시
table.concat( )	요소들을 문자로 연결	local tab = { "Apple", "Banana", "Cake", "Cookie" } print(table.concat(tab, ", ")) -- Apple, Banana, Cake, Cookie
table.copy( )	테이블 복사	local tab1 = { 1, 2, 3 } local tab2 = { 4, 5, 6 } print(table.concat(table.copy(tab1, tab2), ", ")) -- 1, 2, 3, 4, 5, 6
table.indexOf( )	몇 번째 요소인가?	local tab = { 1, 3, 5, 7, 9 } print(table.indexOf(tab, 5)) -- 3
table.insert( )	요소 추가	local tab = { “Lua”, “is”, “easy” } print(tab[3]) -- easy table.insert(tab, 3, “very”) print(table.concat(tab, “ ”)) -- Lua is very easy print(tab[4]) -- easy
table.remove( )	요소 제거	local tab = { 1, 3, 5, 7, 9 } table.remove(tab, 2) print(tab[2]) -- 5 print(table.concat(tab, ", ")) -- 1, 5, 7, 9
table.sort( )	요소 정렬	local tab = { 3, 1, 5, 9, 7 } table.sort(tab) print(table.concat(tab, ", ")) -- 1, 3, 5, 7, 9

만약 내림차순으로 정렬하고 싶다면,

```
local function compare ( a, b )
  return a > b
end

local tab = { 3, 1, 5, 9, 7 }
table.sort( tab, compare )
print( table.concat( tab, ", " ) )
-- 9, 7, 5, 3, 1
```

끝났어요! 수고했어요!