**Compiler-es és Interpreter-es nyelvek**

A programozási nyelvek lehetnek:

* compiler-es
* interpreter-es

**Fordító program (compiler):** Egy program, amely valamilyen programozási nyelven megírt programot gépi kódúvá fordít. Minél magasabb szintű egy nyelv annál bonyolultabb a fordítás.

**Értelmező (interpreter):** Egy program, amely valamilyen programozási nyelven megírt programot utasításonként értelmez és végre is hajt.

**A programkészítés és futtatás menete compileres nyelvek (pl C, C++, Pascal, Delphi, stb) esetén:**

1. program megírása egy programozási nyelven, ez az úgynevezett forrás program
2. forrás program lefordítása a fordító program segítségével gépi kódú programmá
3. a gépi kódú program futtatása

**A programkészítés és futtatás menete interpreteres nyelvek esetén (Basic, script nyelvek:**

1. program megírása egy programozási nyelven, ez az úgynevezett forrás program
2. forrás program futtatása egy futtató program (interpreter) segítségével

vagy (pl. JAVA):

1. program megírása egy programozási nyelven, ez az úgynevezett forrás program
2. forrás program lefordítása egy köztes kódra (ezt a fordítót is szokás compiler-nek nevezni)
3. köztes kód futtatása egy futtató program segítségével

A compiler-es nyelvek előnye: gyorsabb, hátránya: a lefordított kód csak ugyanolyan architectúrájú gépen futtatható, mint amelyiken fordították.

interpreter: Az értelmező, értelmező program (angolul interpreter) olyan program (ritkábban beépített hardver), ami képes arra, hogy az általa felismert nyelven megfogalmazott utasításokat bemenő adatként kezelje, és a futtató gép saját utasításkészletének megfelelő utasítások sorozatává alakítsa át, majd ezeket az utasítássorozatokat azonnal futtassa is.

Míg egy fordítóprogram a forrásprogramokat utasításonként a futtató gép által végrehajtható (gépi kódú) utasítások sorozatává alakítja át – fordítja – azaz a forrásprogramból a futtatásra kész forma teljes egészében előáll, addig az értelmező a forrásprogramot anélkül is végrehajthatja – azonnal – hogy a teljes forrásprogramot beolvasná.

Compiler: A fordítóprogram (angolul compiler) olyan számítógépes program, amely valamely programozási nyelven írt programot képes másik nyelv szabályaira lefordítani, például gépi kódra, köztes kódra, vagy egy másik programozási nyelv kódjára.

Szkript nyelvek:

szkriptnyelvek azok a programnyelvek, melyek segítségével a szoftverek képességei kiterjeszthetőek, funkcióik bővíthetőek, vagy kombinálhatóak.

Lua:

adattípusok:

* **nil:** Egyetlen értéke van, a nil. Legfőbb jellegzetessége, hogy különbözik bármely más típustól. Ahogy majd látni fogjuk, a globális változók alapértelmezett értéke nil az első értékadás előtt, és egy globális változót törölhetünk, ha értékét nilre állítjuk.
* **boolean:**Két értéket vehet fel, a true-t és a false-ot, amik a szokásos boolean értékek. Vigyázzunk a nullával és az üres stringekkel, mert más szkript nyelvekkel ellentétben az értékük egy feltételben: igaz.
* **number**Az egyetlen, alapesetben dupla pontosságú lebegőpontos számtípus, de a Lua tetszõleges számreprezentációval fordítható. A szokásos műveletek vannak értelmezve rajta. Number típusú értékek megadását lásd a numerikus literáloknál.
* **string**A szokásos string típus. Tetszőleges 8 bites értéket tartalmazhat, beleértve a \0 -t is. Megadását lásd a karakteres literáloknál.
* **function**Függvény típus. A függvényeket tárolhatjuk változókban, átadhatjuk őket paraméterként, legyenek akár Lua vagy C függvények.
* **userdata**A userdata típus segítségével tetszőleges C típusú adatot tárolhatunk Lua változókban. Csak az értékadás és az egyenlőség-vizsgálat műveletek vannak előre definiálva hozzá, de C-ben megírt függvényekkel saját műveleteket lehet hozzárendelni. Az ilyen értékeket létrehozni és módosítani is csak C-ből lehet, így biztosított az integritásuk.
* **table**Asszociatív tömb típus (tábla). Olyan tömbökről van szó, amelyek indexelhetőek bármely típussal (kivéve nil-lel).A tárolt értékek típusai szintén tetszőlegesek lehetnek (a nil kivételével). Ez a fő strukturáló mechanizmus a Luában: segítségével nem csak tömbök, de rekord és más adatszerkezetek is leírhatóak.
* **thread**Végrehajtási szálat reprezentáló típus.

A Lua egyetlen összetett típusa és egyúttal típuskonstrukciós eszköze a table típus.

Matematikai könyvtárak: math.sin();

math.cos(). stb….

-Az alábbi elágazásban a lokális változó példája látható.Az alábbi kiírás nil eredményt ad, hisz ott hozzuk létre a változót, hisz a lokális változó a blokk végén megszünik.Ha kivennénk a local kulcsszót, akkor a végén 10-et írna ki.

age = 13

if age < 16 then

local lovalVar = 10;

elseif age > 20 then

print('alma')

end

print(lovalVar)

-Logikai kifejezések megfeleltetése

!= ~=

&& and

|| or

! not

canVote = age > 18 ? true : false (értékadás->ha age nagyobb mint 18, akkor true értéket, ha kisebb akkor false értéket ad a változónak)

-Random számgenerálás:

math.randomseed(os.time())

io.write("random numbuer : ",math.random(),"\n")

io.write("random numbuer : ",math.random(10),"\n")

io.write("random numbuer : ",math.random(5,100),"\n")

-Értékadás:Luaban az értékadás érdekes.Lehetőség van többszörös értékadásra.

v1,v2,v3 = 234,564,123

i = 3

i, a[i] = i+1, 20

A többszörös értékadások szimultán módon hajtódnak végre. A következő kód például beállítja a[3] értékét 20-ra, anélkül, hogy megváltoztatná a[4] értékét: Ennek oka, hogy először az értékadás jobb oldala értékelődik ki, és csak ezután hajtjuk végre magát az értékadást.Először a jobb oldal kiértékelödik, majd utána a bal oldalnak tolja át.

A következő sor megcseréli x és y változó értékeit: x,y=y,x

-Elágazás: A Luában csak kétirányú elágazásra van lehetőség, melynek formája a szokásos terminológiát követi:

if *kifejezés* then

*blokk*

else

*blokk*

end

Egymásba ágyazott elágazásoknál az else ág hovatartozásának kérdését az end utasítások alapján lehet eldönteni. Ugyancsak egymásba ágyazott elágazásoknál lehet hasznos, hogy nem kell kitenni a befejező end-et, hanem használható az elseif utasítás is:

if *kifejezés* then

*blokk*

elseif *kifejezés* then

*blokk*

...

else

*blokk*

end

Ezzel az utasítással a többirányú elágazásokhoz hasonló konstrukció hozható létre, ugyanis a nyelv case-utasítást nem tartalmaz, de vigyázat: ez szemantikailag csak egymásba ágyazott elágazásoknak felel meg!

-Ciklusok:

i = 1

while(i <=10)do

print(i)

i = i + 1

if(i == 8) then

break

end

end

Nincs continou elem, nincs i++ sem ebben a nyelvben.

-for ciklus:

1 és 10 között 2 vel lépegetve.A harmadik paraméter a 2-es az opcionális.

for i = 1,10,2 do

print(i)

end

-Iterációs for:Az alábbi példában végigiterálunk egy táblán.

months={'alma','korte','szilva','banan','most','asd'}

for key,value in pairs(months) do

io.write(value," ,")

end

A pairs metódusnév lehet tetszőlegesen más is.Alapértelmezés szerint a keyek számok(az indexelés tetszőleges számtól kezdődik)

-Tábla nyelvi elem:

Vtable = {} /\*létrehozás\*/

/\*értékadás\*/

for i=1,10 do

Vtable[i] = i

end

io.write(Vtable[2])

io.write(#Vtable) /\*tábla nagyságánakmkiiratása\*/

table.insert(Vtable,1,10010101) /\*elem beszurása, ez lista funkció, minden utóbbi elemt arrébtol\*/

io.write(table.concat(Vtable,",")) /\* tábla stringgé konvertálása, és az elemek vesszővel elválasztása\*/

table.remove(Vtable,1)

az 1 indexű elemet eltávolítja a Vtable nevezetű táblából

-Függvények Luaban:

function provider()

tab = {}

for i = 0,10 do

tab[i] = i+1000

end

return tab,10 /\*látható, hogy több érték is visszaadható visszatérérsi értékként\*/

end

vtab,szam = provider()

for key,value in pairs(vtab) do

io.write(value," ,")

end

io.write("szam: ",szam)

-Másik fajta függvény , nem előre meghatározott oaraméterszámmal:

function summ(...) /\*a 3 pont jelenti, javaban is hasonló\*/

local sum = 0

for key,value in pairs{...} do

sum = sum + value

end

return sum

end

szam = summ(1,2,3,4,5,6,7,8,9)

io.write(szam)

**Table(asszociatív tömb típus):**Asszociatív tömbként funkcionál. Nincs fix méret, dinamikusan nő. Indexelhető stringgel számmal, kivéve nullel.

A table az egyetlen adatstruktúrát megvalósítható eszköz luaban.

Viszont ezzel lehet tömböt, listát, sort vermet is készíteni.

a = {} --> Létrehoz egy táblát, és az a változó a referencia rá.

k = "x"

a[k] = 10 -- new entry, with key="x" and value=10

a[20] = "great" -- new entry, with key=20 and value="great"

print(a["x"]) --> 10

k = 20

print(a[k]) --> "great"

a["x"] = a["x"] + 1 -- increments entry "x"

print(a["x"]) --> 11

-A következőkben a táblára való hivatkozást vizsgáljuk:

a = {}

a["x"] = 10

b = a -- b ugyan arra a táblára mutat mint a

print(b["x"]) --> 10

b["x"] = 20

print(a["x"]) --> 20

a = nil -- mostmár csak b mutat a táblára

b = nil -- mostmár semmi sem mutat arra a táblára

Ha megszűnik az összes referencia egy táblára, akkor a Lua szemétgyűjtő kisöpri a memóriából.

-A következő példában láthatjuk, ha nem inicializálunk benne valamit, akkor annak az értéke nil marad:

a = {} -- empty table

-- create 1000 new entries

for i=1,1000 do a[i] = i\*2 end

print(a[9]) --> 18

a["x"] = 10

print(a["x"]) --> 10

print(a["y"]) --> nil

a[”y”] nem volt inicializálva!!!!!!!!!

-Egy kis szintaxis: A Lua támogatja a táblák más szintaxisú használatát is:

a.x = 10 -- same as a["x"] = 10

print(a.x) -- same as print(a["x"])

print(a.y) -- same as print(a["y"])

-a.x with a[x] a kezdők sokszor elbasszák:

a = {}

x = "y"

a[x] = 10 -- belerakja a 10 értéket az a[”y”] -ba

print(a[x]) --> 10 -- a[”y”] értéke 10

print(a.x) --> nil -- a[”x”] ugyan az mint a.x , és ugyebár ezeknek nem lett érték adva ezért nil.

print(a.y) --> 10 -- value of field "y"

-Konzolról 10 adat bekérése, majd kiíratása:

a = {}

for i=1,10 do

a[i] = io.read()

end

Fent a konzolos bekérést láthatjuk.

-- print the lines

for i,line in ipairs(a) do

print(line)

end

A basic Lua könyvtár magába foglalja az ipairs-t.

Az ipairs segítségével végigiterálhatunk egy tömbön, arra viszont vigyázni kell, hogy a tömbnek ott tekinti a végét, ha az első nil elemhez ér. Tehát ha van egy 10 elemű tömböm(ami ugyebár táblaként van megvalósítva), akkor ha a[11]-re hivatkozna, mivel az nem volt inicializálva, annak az értéke nil. Az indexelés sajnos 1-től kezdődik.

-Indexelési problémák: Ha a 0 mint number , és „0” mint string szerepel, akkor mi történik?

a[0] = 10;

a["0"] = 20;

print(a[0]) 10

print(a["0"]) 20

Az alábbi példán az látható, hogy i változó mint változó, és mint értéke is felhasználható indexelésre.

i = 10; j = "10"; k = "+10"

a = {}

a[i] = "one value"

a[j] = "another value"

a[k] = "yet another value"

print(a[j]) --> another value

print(a[k]) --> yet another value

print(a[tonumber(j)]) --> one value

print(a[tonumber(k)]) --> one value

**-Globális változók:** A globális változóknak nincs szükségük deklarációra.

print(b) --> nil

b = 10

print(b) --> 10

Nil értéket kapnak alapból értékül. Általában nincs szükség arra, hogy globális változót töröljünk, ha rövid életű változót szeretnénk, akkor használjunk lokális változókat. Viszont ha törölni szeretnél egy globális változót, akkor nil értéket adj neki.

A Lua nyelv kis-nagybetű érzékeny.