Függvények

A függvény egy önálló számítási egység.

Függvényhívás

A

```
typeof(42)
```

```
Int64
```

hívásnál, a terminológia szerint: meghívjuk a `typeof` függvényt a `42` argumentummal. A visszatérési érték a `typeof`-nál az argumentum típusa. Julia-ban **mindig** van visszatérési érték! Például a már használt `println`-nél:

```
ret=println("szia")
show(ret) # a println nem írja ki a semmit :-)
```

```
szia
nothing
```

Néhány függvényt nem a visszatérési értéke, hanem mellékhatása - pl. képernyőre írás - miatt használunk. Ezek a függvények (gyakran) `nothing` értéket adják vissza.

Figyeljük meg a következő hívásokat:

```
println(parse(Int, "112"))
println(parse(Float64, "1.12e-2"))
println(string(-112))
println(string(-1.12e-2))
```

```
112
0.0112
-112
-0.0112
```

Hiba esetén a következőt kapjuk:

```
parse(Int, "1.12e-2")
```

```
ArgumentError: invalid base 10 digit '.' in "1.12e-2"

Stacktrace:
 [1] tryparse_internal(::Type{Int64}, ::String, ::Int64, ::Int64, ::Int64, ::Bool) at ./pars
 [2] #parse#332(::Nothing, ::Function, ::Type{Int64}, ::String) at ./parse.jl:228
 [3] parse(::Type{Int64}, ::String) at ./parse.jl:228
 [4] top-level scope at In[18]:1
```

A következők hasznosak ha valós számból egészet akarunk csinálni:

```
a=-1.51
println( round(a) )
println( trunc(a) )
println( floor(a) )
println( ceil(a) )
a=1.51
println( round(a) )
println( trunc(a) )
println( floor(a) )
println( ceil(a) )
```

```
-2.0
-1.0
-2.0
-1.0
2.0
1.0
1.0
2.0
```

Matematikai függvények

Példa:

Adott $x > 0$ számhoz határozzuk meg azt a k egész számot melyre $2^k \leq x < 2^{k+1}$

Megoldás:

```
x=11.3
válasz=Int(trunc(log2(x))) # Int() egész típussá alakít
println(válasz)
x=16.00001
válasz=Int(trunc(log2(x)))
println(válasz)
x=15.999999
válasz=Int(trunc(log2(x)))
println(válasz)
```

```
3
4
3
```

Példa:

Adott egy derékszögű háromszög (c) átfogója és egyik befogója (a). Számold ki a területét!

Megoldás:

```
c=5
a=3
b=sqrt(c^2-a^2)
terület=0.5*a*b
println(terület)
c=11
a=10
b=sqrt(c^2-a^2)
terület=0.5*a*b
println(terület)
```

```
6.0
22.9128784747792
```

Példa:

Adott $y = ax + b$ egyenes esetén számoljuk ki az x-tengellyel bezárt szögét (fokokban)!

Megoldás:

```
a=2
szög=180/pi*atan(a)
println(szög)
a=1
szög=180/pi*atan(a)
println(szög)
a=-1
szög=180/pi*atan(a)
println(szög)
```

```
63.43494882292201
45.0
-45.0
```

Függvények létrehozása

Természetesen van lehetőség saját függvények létrehozására a következő szerkezettel:

```
function ír()
  println("Nincsen apám, se anyám")
  println("Se istenem, se hazám.")
end
ír()
```

```
Nincsen apám, se anyám
Se istenem, se hazám.
```

A függvényeink **kifejezések**, melyek kiértékelődnek az aktuális argumentumokkal és elvégzik az előírt tevékenységeket. Alapesetben a függvény a törzs utolsó kifejezésének értékét adja vissza - ez az ő kiértékelésének az eredménye - ami a `return` alkalmazásával felülbírálható. Jelen esetben ez a `nothing`. A függvények tetszőlegesen kombinálhatóak:

```
function ír2()
  ír()
  ír()
  2
end
ret = ír2()
println("ret=", ret)
```

```
Nincsen apám, se anyám
Se istenem, se hazám.
Nincsen apám, se anyám
Se istenem, se hazám.
ret=2
```

Ezen függvény visszatérési értéke 2: az utolsó kifejezés értéke a törzsben. Természetesen a függvények definíciójuk **után** használhatók:

```
y=f()
function f()
  prntln("error :-)")
end
```

```
UndefVarError: f not defined

Stacktrace:
 [1] top-level scope at In[54]:1
```

A végrehajtás menete

Amikor egy program által végrehajtott lépéseket akarjuk nyomonkövetni, az első utasítással kezdjük, majd sorba haladunk. A függvényhívásoknál viszont megszakad ez a lineáris sorrend. Úgy is képzelhetjük, hogy a végrehajtás az aktuális függvény törzsére ugrik. Ez megnehezítheti a működés végigkövetését, főleg ha a figyelembe vesszük, hogy a függvények más függvényeket is hívhatnak, azaz amikor elemzünk egy programot gyakran el kell térnünk a kód lineáris olvasásától.

Paraméterek és argumentumok

Néhány függvény hívásakor mindenképpen meg kell adnunk egy értéket híváskor:`sin(1.1)`, vagy esetleg kettőt: `parse(Int, "123123")`. Változókat feldolgozó függvények definiálása:

```
function ír(ezt)
  println(ezt)
  1
end
function ír2(eztIs)
  ír(eztIs)
  ír(eztIs)
  2
end
ír("Mikor születtem, a kezemben kés volt...")
ír2("Mikor születtem, a kezemben kés volt...")
ír(cos(pi))
par="hi hi "^2
ír2(par)
```

```
Mikor születtem, a kezemben kés volt...
Mikor születtem, a kezemben kés volt...
Mikor születtem, a kezemben kés volt...
-1.0
hi hi hi hi
hi hi hi hi

2
```

A definícióban szereplő `ezt, eztIs` változót (formális) **paraméter**nek nevezzük. A hívást úgy képzelhetjük hogy az aktuális paraméter, az **argumentum** értékével helyettesítődik a formális.

Példa:

Számoljuk ki egy számtani sorozat differenciáját és első n tagjának összegét, ha adott n, a_1, a_n.

Megoldás:

```
function sorozat(n, a1, an)
  d=(an-a1)/(n-1)
  s=0.5*(a1+an)*n
  d,s
end
k,ö = sorozat(3,1,3)
println(k," ",ö)
k,ö = sorozat(3,1,11)
println(k," ",ö)
```

```
1.0 6.0
5.0 18.0
```

A függvény formális paraméterei és a számításokhoz használt további változók: **lokális**ak - azaz csak a függvényen belül láthatók. Miután a függvény befejezi a működését, megsemmmisülnek.

Egy függvény több értéket is visszaadhat a fenti módon. (Pontosabban ez csak egy érték, az)

```
sorozat(3,1,3)
println(a1)
```

```
UndefVarError: a1 not defined

Stacktrace:
 [1] top-level scope at In[66]:2
```

Miért használjunk függvényeket?

1. felbonthatjuk a hosszú számolásokat, rövidebb, könnyebben karbantartható részekre
2. felesleges kódismétléseket szüntethetünk meg velük
3. a megírt függvényeket újrahasználhatjuk más feladatoknál

Feladatok

1 Feladat:

Írjunk olyan függvényt mely a paraméterül kapott x sztringet egy n-hosszú, az elején üres helyekkel feltöltött sztringbe alakítja.

Megoldás

2 Feladat:

Írjunk olyan függvényt mely egy paraméteres rácsnak megfelelő sztringet ad vissza.

```
println(rács(1,2,1,3))
+---+---+
|   |   |
+---+---+
println(rács(2,2,1,1))
+-+-+
| | |
+-+-+
| | |
+-+-+
println(rács(3,1,1,3))
+---+
|   |
+---+
|   |
+---+
|   |
+---+
```

A függvényt olyan alakban írjuk meg, hogy lehessen szabályozni az rácsok számát és méretét is! A fenti példában a második kettő paraméter a kis téglalapok méretét szabályozza.

Megoldás