Változók, kifejezések, utasítások

Értékadás

A programozási nyelvek nagyszerű képessége, hogy neveket rendelhetünk értékekhez:

```
egész=1
```

```
1
```

```
tört=1.1
```

```
1.1
```

```
karakter='1'
```

```
'1': ASCII/Unicode U+0031 (category Nd: Number, decimal digit)
```

```
sztring="1"
```

```
"1"
```

A `név = érték` kifejezés kiértékelése után a `név` sztringgel hivatkozhatunk az `érték`-re, azaz úgy használhatjuk, mintha az adott érték lenne. Ezt **értékadás**nak nevezzük, de a pontosabb kifejezés: hozzákötjük a **nevet** az **értékhez**. Ha mondjuk az $ax + b$ függvényt szeretnénk vizsgálni, akkor az

```
a=1
b=-1
```

```
-1
```

értékadások után, van értelme az

```
y1=a*1+b
y0=a*0+b
ym1=a*(-1)+b
```

```
-2
```

újabb értékadásoknak és továbbiakban használhatjuk az új neveket.

Nevek

Természetesen nem minden jelsorozat használható változónévként:

```
rossz@=12
```

```
syntax: extra token "@" after end of expression
```

```
1szám = 24
```

```
syntax: "1" is not a valid function argument name
```

```
szám1 = 24
```

```
24
```

```
_Értéke=pi
```

```
 = 3.1415926535897...
```

Láthatjuk hogy elég sokféle karaktert használhatunk változóink elnevezésére. Legegyszerűbb ha a betűvel vagy aláhúzással kezdődik és betűvel, számmal vagy aláhúzással folytatódik szabályhoz tartjuk magunkat. Ezenkívül vannak bizonyos tiltott nevek, kulcsszavak melyeket a rendszer használ:

---	---	---	---	---
abstract type	baremodule	begin	break	catch
const	continue	do	else	elseif
end	export	finally	for	function
global	if	import	importall	let
local	macro	module	mutable struct	primitive type
quote	return	try	using	struct
while	---	---	---	---

```
end = 123
```

```
syntax: unexpected "end"
```

A rendszer függvényneveinek magáncélú felhasználása erősen ellenjavallt, de lehetséges:

```
sin = 123
```

```
123
```

Bár ezután ne akarjuk a `sin` függvényt használni:

```
sin(pi)
```

```
MethodError: objects of type Int64 are not callable

Stacktrace:
 [1] top-level scope at In[41]:1
```

Kifejezések, utasítások

A *kifejezés*ek értékek, nevek és operátorok kombinációi, például az alábbiak mindegyike kifejezés:

```
sin=Base.sin # vissza kell állítani a sin-t, mert az előbb átdefiniáltuk
n=1
n=n+1
(sin(n)+n+1)^3
"Dörmögő Dömötör "^2
```

```
"Dörmögő Dömötör Dörmögő Dömötör "
```

Ha REPL-ben gépelsz egy kifejezést és nyelvtanilag hibátlan, akkor a rendszer kiírtékeli, és az értéket kiírja.

Szkript mód

Használhatjuk a Juliát szkript-módban is. Ez hosszabb lélekzetű feladatok megoldásánál hasznos. Valamilyen sima text-editorral hozzunk létre egy programot, mentsük el `példa01.jl` néven:

```
a=1
b=2
c=1
delta=sqrt(b^2-4*a*c) # sqrt: négyzetgyök függvény
x1=(-b+delta)/(2*a)
x2=(-b-delta)/(2*a)
```

Ezt a szkriptet a REPL-ből vagy a Jupyter-rendszerből az

```
include("példa01.jl")
```

módon futtathatjuk:

```
include("példa01.jl")
println(x1, " ", x2) # több paraméter vesszővel elválasztva
```

```
-1.0 -1.0
```

Ami megfelel az elvárásoknak, hiszen a szkriptben az $x^2 + 2x + 1 = 0$ egyenlet gyökeit számoltuk ki.

A kiértékelés sorrendje

Ha egy kifejezés több operátort is tartalmaz a kiértékelés sorrendje függ az operátorok erősségétől, precedenciájától. Ez a `PEMDAS` szabály, mely csökkenő erősséggel adja meg a precedenciákat:

zárójel	Parentheses
hatványozás	Exponentiation
szorzás - osztás	Multiplication - Division

| --- | --- |
| összeadás - kivonás | Addition - Subtraction |

Ha nem vagyunk biztosak a sorrendben, akkkor használjunk zárójelet!

Sztring műveletek

A sztringek közötti természetes művelet az utánafűzés, egymás melléírás, amit a `*` operátor valósít meg:

```
"Almás" * "rétes"
```

```
"Almásrétes"
```

Ha a két sztring egyenlő akkor használhatjuk a `^`-t:

```
"Hi "^3
```

```
"Hi Hi Hi "
```

Megjegyzések

A sorok `#` utáni részét figyelmen kívül hagyja rendszer, ide írhatjuk a számításokkal kapcsolatos emlékeztető, magyarázó szövegeinket:

```
a = 1 # ez lesz a főegyüttható
```

```
1
```

Ha jól választjuk a változóneveket, akkor kevesebb magyarázkodásra van szükség:

```
velocity = 10
time = 10
distance = velocity * time
```

```
100
```

Hibakeresés

A fejlesztés során felbukkanó hibák 3 fő csoportra bonthatóak:

1. nyelvtani, szintaktikai nem tartottuk be a nyelv szabályai, hamar kiderül
2. szemantikai, elvi a program nem az elvárt kimenetet adja bizonyos bemenetekre
3. futás közbeni, ”run-time” bizonyos bemenő adatokra kivételes, a program által le nem kezelt esemény következik be