Haskell 1.

Alapok

- tisztán **funkcionális nyelv**, **minden függvény** (a konstansok is)
- nincsenek hagyományos változók, az első értékadás után **nem módosíthatók**
  - elég jól elkerülhetők így a mellékhatások
  - könnyebben elemezhető a kód

Kiértékelési módok

- ***Lusta kiértékelés:*** A lusta kiértékelés során mindig a legkülső redex (redukálható kif.) kerül helyettesítésre, az argumentumokat csak szükség esetén értékeli ki. Ez a módszer mindig megtalálja a kezdeti kifejezés normál formáját.
- ***Mohó kiértékelés:*** A mohó kiértékelés az argumentumok kiértékelésével kezdődik, csak ezután hajtja végre a függvény alkalmazásának megfelelő redukciós lépést.

A Haskell nyelv a ***lusta*** kiértékelési stratégiát használja.

Példa a két kiértékelésre:

- *inc* és *duplaz:* egyargumentumú függvények
- *inc:* eggyel növeli argumentumának értékét,
- *duplaz:* megkétszerezi argumentumának értékét

Lusta kiértékelés

duplaz (inc 9)
(inc 9) + (inc 9)
(9 + 1) + (9 + 1)
10 + 10
20

Mohó kiértékelés

duplaz (inc 9)
duplaz (9 + 1)
duplaz 10
10 + 10
20

Atomi típusok és operátorok

- Minden típus nagy betűvel kezdődik!
- **Int** – egész típusok
  - Infix op.: **+, –, *, *div*** (egész osztás – függvény!), ***mod*** (egész osztás maradéka), **^**
  - Prefix op.: **–**, ***negate***
- **Float** – valós típus, lebegőpontos számok
  - Infix op.: **+, –, *, /** (törtosztás)
  - Prefix: ***negate, –, sin, cos, log, sqrt*** (gyökvonás)

- **Bool** – logikai típus, a ***True*** és ***False*** értékeket veheti fel.
  - Infix op.:
    - **&&**- logikai ÉS
    - || - logikai VAGY
  - Prefix op.: ***not***

Numerikus argumentumú infix relációs operátorok

- **==** - egyenlő
- **/=** - nem egyenlő
- **>** - nagyobb
- **>=** - nagyobb-egyenlő
- **<** - kisebb
- **<=** - kisebb-egyenlő

Komment

- - egy soros komment

{- több

soros

komment **-}**

Hugs interpreter

- Linux → System Tools → Terminál.
- Innen indítható a Hugs interpreter a ***hugs*** paranccsal.
- Használat:
  - ***hugs*** - értelmező indítása; kilépés: ***:quit (:q)***
  - hugs +s - a végrehajtás során elvégzett műveletek, pl. redukciós lépések számának kiíratása
  - hugs +t - a visszatérési érték típusának kiíratása
- Fontos parancsok:
  - fájl betöltése: ***:load*** **fájlnév** ***(:l)*** - paraméter nélkül kidob minden betöltött fájlt
  - utoljára betöltött fájl újratöltése: ***:reload (:r)***
  - kilépés: ***:quit (:q)*** vagy ***Ctrl+D***
  - parancsok listája: ***:?***

Haskell program

- Kiterjesztése: **.hs**
- Függvények definícióját fogja tartalmazni
- Megnyitjuk az interpretert, és betöltjük a fájl tartalmát:

  > ***:load*** *fajl_neve.hs*

- Ezután az állományunk bármely függvényét meghívhatjuk
- Függvény hívása:

  > ***fgv_neve param_1 param_2 … param_n***

Függvény definiálása

- **Paraméteres függvény:**

  függvény_azon :: arg1_típusa - >arg2_típ - > … - > argn_típusa - > visszatérés_típusa

  függvény_azon arg1 arg2 … = kifejezések

  A kifejezések kiértékelése határozza meg, hogy mi lesz a függvény visszatérési értéke.

- **Paraméter nélküli**:

  függyvény_azon :: visszatérés_típusa

Írjuk meg a duplaz és az add függvényt!

```
duplaz :: Int -> Int
duplaz x = x + x
```

```
add :: Int -> Int -> Int
add x y = x + y
```

Esetvizsgálatok
(hs_1.hs)

- Az **if** kifejezéssel: A visszatérési értéket az argumentumban érkező szám értékétől tesszük függővé. Ha az 0, térjünk vissza 1-gyel, egyébként hajtsuk végre az else ág rekurziós kifejezését.

```
faktorialis :: Int - > Int
faktorialis n =     if n = = 0
                    then 1
                    else n * ( faktorialis ( n – 1 ) )
```

- ***Őrökkel*** (Boolean Guards): Logikai kifejezéseket adunk meg. A függvény visszatérési értékét azon feltétel mögötti kifejezés határozza meg, amely teljesül. Az egyes feltétel-kifejezés párokat |-pal választjuk el.

```
faktorialis :: Int - > Int
faktorialis n| n = = 0     = 1
             | n  >  0     = n * ( faktorialis ( n – 1 ) )
```

- **Mintaillesztéssel**: Az egyes „mintákat” külön-külön sorokban definiáljuk és a függvény neve után adjuk meg. A visszatérési értéket azon minta mögötti kifejezés határozza meg, amelyikre illik az argumentum.

```
faktorialis :: Int - > Int
faktorialis 0       = 1
faktorialis n       = n * ( faktorialis (n - 1) )
```

- A **case** kifejezéssel: A hagyományos esetkiválasztásos szelekció Haskell-beli megfelelője. A teljesülő „eset” mögött megadott kifejezés határozza meg a visszatérési értéket. Jelen esetben vagy egy konstansérték vagy egy rekurzív kifejezés.

```
faktorialis :: Int - > Int
faktorialis n = case n of
    0     - > 1
    n     - > n  * (faktorialis (n - 1))
```

1. Határozzuk meg két szám átlagát! **(hs_2.hs)**

```
atlag::Double -> Double-> Double
atlag x y = (x + y ) / 2
```

2. Számítsuk ki egy másodfokú egyenlet gyökeit! **(hs_4.hs)**

```
roots (a,b,c) = if d < 0
                then error "sorry"
                else (x1, x2)
                  where
                    x1 = e + sqrt d / (2 * a)
                    x2 = e - sqrt d / (2 * a)
                    d = b * b - 4 * a * c
                    e = - b / (2 *a)
```