Tartalom

DP24a - 2. FP gyakorlat, 2024-09-10

```
Mix.install([
{:"Elixir.LastEr", path: "./", app: false, compile: false},
{:"Elixir.LastEx", path: "./", app: false, compile: false},
{:"Elixir.Nth", path: "./", app: false, compile: false}
])
```

Rekurzív számítások

Legnagyobb közös osztó (greatest common divisor) euklideszi algoritmussal

```
defmodule Gcd do
 @spec gcd(a :: integer(), b :: integer()) :: d :: integer()
 # a és b legnagyobb közös osztója d
 def gcd(a, b) do
 ...
 end
end
(Gcd.gcd(96, 42) === 6) |> IO.inspect
(Gcd.gcd(90, 45) === 45) |> IO.inspect
```

Súgó

Ha $a = b \cdot q + r$, akkor $gcd(a, b) = gcd(b, r)$, ahol a, b, q és r egész számok.

Írjon több változatot, kivonással és maradékos osztással (rem/2). Gondoljon arra, hogy a függvény első paramétere nagyobb is, kisebb is lehet a másodiknál.

A Ludolph-féle szám, azaz π közelítése a Leibniz-sorral

```
defmodule Pi do
 @spec pi(i :: integer) :: pi :: float
 # A pi i-edik közelítő értéke pi
 def pi(i), do: ...
 ...
end
(abs(Pi.pi(10000000) - :math.pi) < 1.0e-6)  |> IO.inspect
```

Segédfüggvényt definiálhat.

Súgó

$\pi/4 = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7}$ ... Javasoljuk, hogy használjon segédfüggvényt.

Egész szám tetszőleges számrendszerbe konvertálása

```
defmodule Dec2rad do
 @spec dec2rad(i :: integer, r :: integer) :: ds :: [integer]
 # Az i egész szám r alapú számrendszerbe konvertált, decimális számként
 # megadott számjegyeinek listája ds
 def dec2rad(i, r), do: ...
```

```
  ...
end
(Dec2rad.dec2rad(13, 2) === [1, 1, 0, 1]) |> IO.inspect
(Dec2rad.dec2rad(721, 17) === [2, 12, 7]) |> IO.inspect
```

Segédfüggvényt definiálhat.

Súgó

Az r számrendszerbeli $d_4d_3d_2d_1$ szám Horner-elrendezés szerinti alakja: $((d_4 \cdot r + d_3) \cdot r + d_2) \cdot r + d_1$

Szülinapi gyertyák

Informatikusék szülinapot ünnepelnek. Nagy a család, egyszerre többet is. Már a tortánál tartanak, amikor kiderül, hogy gyertyát – azt nem vettek. Sebaj, van a fiók mélyén pár alig használt számjegy-gyertya, csak az a baj, hogy ezekkel tízes számrendszerben nem lehet kirakni a szülinapok összegét. Ez se baj, kirakják más számrendszerben.

Segítsen nekik, írjon olyan függvényt Dec2rad.dec2rad/2 felhasználásával, amelyiknek az első paramétere a megünnepelt szülinapok összege, a második pedig egy olyan egészlista, amelyben a fiókban talált számjegy-gyertyák értéke van felsorolva.

```
defmodule Candles do
 @type candles() :: [integer()]
 @spec candles(yrs::integer(), cndls::candles()) :: sols::[{rdx::integer(), solcndls::candles()}]
  # a sols lista solcndls listáinak azok a felhasznált gyertyák az elemei,
  # amelyekkel az adott rdx alapú számrendszerben a yrs szülinapszám kirakható
  # a cndls értékű gyertyák felhasználásával
 def candles(yrs, cndls), do: ...
 ...
(Candles.candles(76, [13,9,7,3]) == [{21, [3, 13]}, {23, [3, 7]}]) |> IO.inspect
(Candles.candles(19, [9,7,1]) == [{10, [1, 9]}, {12, [1, 7]}]) |> IO.inspect
```

Segédfüggvényt definiálhat.

Összetettebb feladatok listák feldolgozására

Mátrix első oszlopában lévő elemek összegyűjtése

Egy $n * m$ $(n, m >= 0)$ méretű mátrixot listák listájaként, sorfolytonosan ábrázolunk. Írjon olyan rekurzív függvényt, amelyik egy listába gyűjti a mátrix első oszlopában lévő elemeket, az eredeti sorrendet megőrizve. Használjon mintaillesztést, ahol csak lehet!

```
defmodule FirstCol do
 @spec first_col(xss :: [[any()]]) :: zs :: [any()]
 def first_col(xss) do
 ...
 end
end
xss = [[A,B,C],[:a,:b,:c],[1,2,3],[3.2,2.1,1.0]]
(FirstCol.first_col(xss) === [A, :a, 1, 3.2]) |> inspect() |> IO.puts()
(FirstCol.first_col([]) === []) |> inspect() |> IO.puts()
```

Listák első elemeinek összegyűjtése

Egy listában különböző hosszúságú listák vannak, ezek üresek is lehetnek. Írjon olyan rekurzív függvényt, amelyik egy listába gyűjti a nemüres listák első elemeit, az eredeti sorrendet megőrizve. Használjon mintaillesztést, ahol csak lehet!

```
defmodule FirstElem do
 @spec first_elem(xss :: [[any()]]) :: zs :: [any()]
 def first_elem(xss) do
 ...
 end
end
xss = [[A,B,C],[:a,:b,:c],[1,2,3],[3.2,2.1,1.0]]
(FirstElem.first_elem(xss) === [A, :a, 1, 3.2]) |> inspect() |> IO.puts()
xss = [[A,B],[:a,:b,:c,:d],[1],[],[3.2,2.1,1.0]]
```

```
(FirstElem.first_elem(xss) === [A, :a, 1, 3.2]) |> inspect() |> IO.puts()
(FirstElem.first_elem([]) === []) |> inspect() |> IO.puts()
```

Módosítsa úgy a megoldását, hogy üres lista esetén a lista feje helyett a nil atomot rakja be az eredménylistába.

```
defmodule FirstElem do
 @spec first_elem(xss :: [[any()]]) :: zs :: [any() | nil]
 def first_elem(xss) do
 ...
 end
end
xss = [[A,B],[:a,:b,:c,:d],[1],[],[3.2,2.1,1.0]]
(FirstElem.first_elem(xss) === [A,:a,1,nil,3.2]) |> inspect() |> IO.puts()
```

Elfajult mátrix első oszlopának transzponálása

Elfajult mátrixnak nevezünk egy olyan mátrixot, amelynek soraiban különböző számú elemek lehetnek. Egy ilyen mátrixot is listák listájaként, sorfolytonosan ábrázolunk. Írjon olyan rekurzív függvényt, amelyik egy elfajult mátrix első oszlopát az eredménymátrix első sorává transzformál, a további sorokban pedig a bemenő mátrix sorai vannak a transzformált első elemek kivételével, az eredeti sorrend megtartásával. Ha egy sor üres, akkor első eleme helyett a nil atom kerüljön be az eredménymátrixba. Használjon mintaillesztést, ahol csak lehet!

```
defmodule FirstColTranspose do
 @spec first_col_transpose(xss :: [[any()]]) :: zss :: [[any()]]
 def first_col_transpose(xss) do
 ...
 end
end
xss = [[A,B,C],[:a,:b,:c],[1,2,3],[3.2,2.1,1.0]]
zss = [[A, :a, 1, 3.2],[B,C],[:b,:c],[2,3],[2.1,1.0]]
(FirstColTranspose.first_col_transpose(xss) === zss) |> inspect() |> IO.puts()
xss = [[A,B],[:a,:b,:c,:d],[1],[3.2,2.1,1.0]]
zss = [[A, :a, 1, 3.2],[B],[:b,:c,:d],[],[2.1,1.0]]
(FirstColTranspose.first_col_transpose(xss) === zss) |> inspect() |> IO.puts()
(FirstColTranspose.first_col_transpose([]) === [[]]) |> inspect() |> IO.puts()
xss = [[A,B],[:a,:b,:c,:d],[1],[],[3.2,2.1,1.0]]
zss = [[A,:a,1,nil,3.2],[B],[:b,:c,:d],[],[],[2.1,1.0]]
(FirstColTranspose.first_col_transpose(xss) === zss) |> inspect() |> IO.puts()
```

Lehet olyan változatot is írni, amelyik csak egyszer halad végig a bemenő mátrixon. Rajta, írjon egyet! Vagy többet! :-)

A következő feladatokban az előző gyakorlaton megírt függvényeket kell alkalmaznia (LastEr.last/1, LastEx.last/1), Nth.nth/2. Megteheti, hogy ezeket bemásolja ide egy cellába, de a lefordított kódot magát is importálhatja. Ehhez a következőket kell tennie:

- Nyissa meg az előző gyakorlat megoldásait is tartalmazó .livemd fájlt.
- Mentse el a fájlt .exs formátumban:
  - Klikkeljen a notebook-lap jobb felső részén látható *három függőleges pontra.*
  - Nyissa meg az *export* menüpontot.
  - Válassza ki az *IEx session* lehetőséget.
  - Kattintson a *Download source* ikonra.
- A letöltött fájlt másolja be abba a mappába, ahol ennek a gyakorlatnak a .livemd fájlja van.
- Fordítsa le a fájlt az elixirc fordítóval. Ha a fordító hibát jelez, javítsa.
- Ha a fordítás sikeres, az eredménye több .beam kiterjesztésű fájl lesz – minden modul külön fájlba kerül.
- E notebook-lap tetején van egy cella *(Notebook dependencies and setup)*, ebbe másolja be az alábbi sorokat:

  Mix.install([{:"Elixir.LastEr", path: "./", app: false, compile: false}, {:"Elixir.LastEx", path: "./", app: false, compile: false}, {:"Elixir.Nth", path: "./", app: false, compile: false}])
- Klikkeljen a *Setup* gombra. Most már hivatkozhat ebből a notebookból az importált modulokra. Ezt a módszert használhatja egyéb idegen modulok importálására is.

Listák utolsó elemeinek listája

Írjon olyan rekurzív függvényt az előző gyakorlaton megírt LastEr.last/1 függvény felhasználásával, amelyik paraméterként egy listák listáját kapja meg, és a belső listák utolsó eleméből álló listát adja eredményül. Ha egy belső lista üres, hagyja figyelmen kívül. Ha az összes belső lista üres, vagy a függvényt magát egy üres listára alkalmazzuk, az eredmény az üres lista legyen.

```
defmodule LastOfListsEr do
 @spec last_of_lists(xss :: [[any()]]) :: zs :: [any()]
 # xss utolsó elemeinek listája zs
 def last_of_lists(xss) do
 ...
 end
end
IO.puts(LastOfListsEr.last_of_lists([]) == [])
IO.puts(LastOfListsEr.last_of_lists([[]]) == [])
IO.puts(LastOfListsEr.last_of_lists([[], ~c""]) == [])
IO.puts(LastOfListsEr.last_of_lists([~c"apple", ~c"peach", ~c"", ~c"orange"]) == ~c"ehe")
```

Súgó

A LastEr.last/1 függvény visszatérési értékeit mintaillesztéssel azonosítsa egy case szerkezetben a rekurzív hívásokhoz.

Írja meg a függvény egy változatát az előző gyakorlaton szintén megírt LastEx.last/1 függvény alkalmazásával.

```
defmodule LastOfListsEx do
 @spec last_of_lists(xss :: [[any()]]) :: zs :: [any()]
 # xss utolsó elemeinek listája zs
 def last_of_lists(xss) do
 ...
 end
end
IO.puts(LastOfListsEx.last_of_lists([]) == [])
IO.puts(LastOfListsEx.last_of_lists([[]]) == [])
IO.puts(LastOfListsEx.last_of_lists([[], ~c""]) == [])
css = [~c"now", ~c"bye", ~c"", ~c"hell", ~c"cell", ~c""]
(LastOfListsEx.last_of_lists(css) == ~c"well") |> IO.puts()
```

Mire kell ügyelnie a case kifejezésben ebben a változatban, amire a LastEr.last/1 függvény alkalmazásakor nem kellett? Itt miért igen, ott miért nem?

Súgó

A minták sorrendjére a LastEx.last/1 függvény visszatérési értékeinek mintaillesztéssel való azonosításakor.

Listák utolsó előtti elemeinek listája

Írjon olyan rekurzív függvényt az előző gyakorlaton megírt LastEx.last/1 vagy Nth.nth/2függvény felhasználásával, amelyik listák listáját kapja paraméterként, és a belső listák utolsó előtti eleméből álló listát adja eredményül. Ha egy belső lista üres, hagyja figyelmen kívül. Ha az összes belső lista üres, vagy a függvényt magát egy üres listára alkalmazzuk, az eredmény az üres lista legyen.

```
defmodule LastButOnesEx do
 @spec last_but_ones(xss :: [[any()]]) :: zs :: [any()]
 # xss utolsó előtti elemeinek listája zs
 def last_but_ones(xss) do
 ...
 end
end
IO.puts(LastButOnesEx.last_but_ones([]) == [])
IO.puts(LastButOnesEx.last_but_ones([[]]) == [])
IO.puts(LastButOnesEx.last_but_ones([[], ~c""]) == [])
IO.puts(LastButOnesEx.last_but_ones([~c"bye", ~c"tiles", ~c"", ~c"list"]) == ~c"yes")
```