DP23a, NZH1, “B” csoport, 2023. okt. 13.

Személyi adatok

Biztos benne, hogy a “B” csoportban ül, és a megfelelő feladatsort oldja meg?

Másolja be egy Elixir-cellába az alábbi kódot, és írja be a kért adatokat: név, NEPTUN-kód, terem, sor és oszlop vagy ülés száma (a teremben kihirdetett elrendezés szerint; ha az ülések számozva vannak, a sor helyett az ülésszámot írja be, és az oszlop számát hagyja üresen).

```
defmodule Personal do
@nev ""
 @neptun ""
 @terem ""
 @sor_ules ""
 @oszlop ""

 def check() do
  unless @nev != "" and @neptun != "" and @terem != "" and @sor_ules != "" do
   "Írja be a kért adatokat!"
  else
   if @oszlop == "" do
    "Ha nem az ülés számát írta be a @sor_ules attribútumba," <>
    "írja be az oszlop számát is a @oszlop attribútumba!"
   else "Ellenőrizze még egyszer a beírt adatok helyességét."
   end
  end
 end
end
Personal.check()
```

```
defmodule Personal do
 @nev ""
 @neptun ""
 @terem ""
 @sor_ules ""
 @oszlop ""

 def check() do
  unless @nev != "" and @neptun != "" and @terem != "" and @sor_ules != "" do
   "Írja be a kért adatokat!"
  else
   if @oszlop == "" do
    "Ha nem az ülés számát írta be a @sor_ules attribútumba," <>
     "írja be az oszlop számát is a @oszlop attribútumba!"
   else
```

```
      "Ellenőrizze még egyszer a beírt adatok helyességét."
    end
  end
 end
end

Personal.check()
```

A beadásról

A zárthelyi megoldását az ETS-sel kell beadni.

1. A megoldás elmentéséhez kattintson a lap tetején a bal felső sarokban található három függőleges pontra, majd az *Export* menüpontra. Válassza ki az *IEX session* fület, majd kattintson a *Download source* ikonra. Az elmentett fájl (valószínűleg) a böngésző által használt Downloads/Letöltések mappába kerül. A fájlnak nem kell más nevet adnia, a beadáskor az ETS át fogja nevezni.
2. Lépjen be az ETS-be. Ha elfejeltette volna: a felugró ablakban az *etsuser* nevet és az *ets+2023#* jelszót kell megadnia. Ezután az ETS-be a Neptun-kódjával és a korábban megadott jelszavával tud belépni. A zárthelyi megoldását a *HF beadás* menüpont *Elixir nagyzéhá* sorában kell feltöltenie. A feltöltendő fájl neve bármi lehet, az ETS át fogja nevezni.

Korlátozások és ajánlások

Ne használja a következő könyvtári függények egyik változatát se a feladatok megoldásában:

Enum.at, Enum.with_index

A használatukat ellenőrizni fogjuk. Pontvesztéssel jár, ha nincs jobb ötlete, és mégis használná valamelyiket.

Javasoljuk (a Kernel modul függvényei mellett) a következő könyvtári függények valamelyik változatának használatát bizonyos feladatokban:

Enum.zip, Enum.split*, Enum.take*, Enum.uniq, Enum.find_index

1. feladat: egyklózos függvények

írjon egy-egy egyklózos, nemrekurzív függvényt az alábbi feladatok megoldására **for komprehenzióval.** Nem használhatja az Enum.at és az Enum.with_indexfügvényeket, és segédfüggvényt sem definiálhat. Javasoljuk az Enum.zip függvények valamelyikének használatát.

```
defmodule T1F1 do
 @spec f1(xs::[integer()]) :: rs::[integer()]
 # Az xs-ben lévő pozitív számok indexének listája rs, tetszőleges sorrendben.
```

```
 # Az indexelés 0-tól kezdődik. Az xs lehet üres, és lehet, hogy nincs benne
 # negatív szám.                                                    (3 pont)
 def f1(xs) do
   for ... , do: ...
 end
end
((T1F1.f1([1,4,-5,6,5,3,-12,11,7,4]) |> Enum.sort()) === [0,1,3,4,5,7,8,9]) |> IO.puts
(T1F1.f1([1,4])                                 === [0,1]) |> IO.puts
((T1F1.f1([-1,-4])                |> Enum.sort()) === []) |> IO.puts
((T1F1.f1([0,1,1])                |> Enum.sort()) === [1,2]) |> IO.puts

defmodule T1F2 do
 @spec f2(xs::[integer()]) :: rs::[integer()]
 # Az rs az xs minden olyan elemét tartalmazza, amelyik megegyezik az őt követő két
 # elem különbségének abszolút értékével. Az xs elemeinek száma háromnál kisebb,
 # akár 0 is lehet. Az rs elemeinek sorrendje tetszőleges.                (4 pont)
 def f2(xs) do
   for ... , do: ...
 end
end
(T1F2.f2([1,4,3])                          === [1]) |> IO.puts
((T1F2.f2([1,2,1,-1])          |> Enum.sort()) === [1,2]) |> IO.puts
(T1F2.f2([0,0])                            === []) |> IO.puts
((T1F2.f2([0,1,1,3,2,-1,-3,1,4]) |> Enum.sort()) === [0,1,2,3]) |> IO.puts

defmodule T1F3 do
 @spec f3(xs::[any()]) :: fs::[{x::any(), i::integer()}]
 # Az fs {x, i} párok listája, amelyekben x az xs lista egy eleme, i pedig az x jobbról
 # számítva első előfordulásának indexe xs-ben (indexelés jobbról 0-tól). xs lehet üres
 # is. Az fs-beli x értékek mind különbözőek, a párok fs-beli sorrendje tetszőleges.
 # Javasoljuk az Enum.uniq és az Enum.find_index függvények használatát.   (5 pont)
 def f3(xs) do
   for ... , do: ...
 end
end
((T1F3.f3([:a,:b,:c,:d,:b,:c,:d,:c,:d,:d]) |> Enum.sort())
  === [{:a,9},{:b,5},{:c,2},{:d,0}]) |> IO.puts
((T1F3.f3([:a,:a,:b,:a,:c,:b,:d,:c,:b,:d]) |> Enum.sort())
  === [{:a,6},{:b,1},{:c,2},{:d,0}]) |> IO.puts
((T1F3.f3('aaaaaaaaaaaa') |> Enum.sort())
  === [{?a,0}]) |> IO.puts
((T1F3.f3([1,3,3,0,1,3,8,3]) |> Enum.sort())
  === [{0,4},{1,3},{3,0},{8,1}]) |> IO.puts
(T1F3.f3([]) === []) |> IO.puts
(T1F3.f3([-1]) === [{-1,0}]) |> IO.puts
```

2. feladat: magasabb rendű és névtelen függvények

Az alábbi függvénydefinícióban a myfun szót cserélje le egy olyan segédfüggvényre, amellyel az adott függvény a specifikált módon működik. A segédfüggvény teszőleges könyvtári függvény, helyben definiált névtelen függvény vagy a kettő kombinációja lehet. A névtelen függvényt definiálhatja az fn jelöléssel vagy a & jellel kezdődő rövidített jelöléssel.

```
defmodule T2F1 do
 @spec f1(xs::[integer()]) :: n::integer()
 # Az xs lista maradéka, azaz az azonos elemekből álló, lehető leghosszabb
 # prefixuma nélküli része n hosszúságú. Feltételezheti, hogy az xs nem üres.    (3 pont)
 def f1(xs) do
  Enum.drop_while(xs, myfun) |> length()
 end
end
(T2F1.f1([6])             === 0) |> IO.puts
(T2F1.f1([6,6,6,5,5,5,6,6,6]) === 6) |> IO.puts
(T2F1.f1([2,2,2,7,-1,3])    === 3) |> IO.puts
(T2F1.f1([-1,-2,-2])       === 2) |> IO.puts
(T2F1.f1([3,2,1,0])        === 3) |> IO.puts
(T2F1.f1([1,1,1,1,1,1,1,-4]) === 1) |> IO.puts
```

3. feladat: Műveletvégzés bináris fa levelein

Egy bináris fa típusát így definiáljuk:

```
@type bintree() :: any() | {bintree(), bintree()}
```

Írjon olyan rekurzív függvényt mirrormap néven, amelynek két paramétere van: az első egy bináris fa, a második a fa összes levelére alkalmazandó függvény. A mirrormap/2 eredménye a függőleges tengelye mentén tükrözött és transzformált fa legyen. Írja meg a mirrormap olyan egyparaméteres változatait is a megfelelő segédfüggvény definiálásával, amelyek az alábbi példákban látható eredményt adják (myfun-t cserélje le a megfelelő segédfüggvényre). A segédfüggvény teszőleges könyvtári függvény, helyben definiált névtelen függvény vagy a kettő kombinációja lehet. A névtelen függvényt definiálhatja az fn jelöléssel vagy a & jellel kezdődő rövidített jelöléssel.

```
defmodule T3 do
 @type bintree() :: any() | {bintree(), bintree()}
 @spec mirrormap(bt::bintree(), f::(any() -> any())) :: rt::bintree()
 # rt a bt tükrözött másolata, melyben a levelek a bt leveleinek f-fel transzformált értéke (12 pont)
 def mirrormap do
  ...
 end


 # Minden levél azonos az eredetivel
```

```
  def map_identical(bt), do: mirrormap(bt, myfun)
  # A levél true, ha az eredeti integer() típusú levél értéke páratlan, egyébként false
  def map_parity(bt), do: mirrormap(bt, myfun)
  # A levél értéke az eredeti integer() típusú levél értékének harmada (egészosztással)
  def map_div_by_3(bt), do: mirrormap(bt, myfun)
  # A levél értéke az eredeti String.t() típusú levél értéke kapcsos zárójelek között
  def map_enclose(bt), do: mirrormap(bt, myfun)
  # A levél értéke az eredeti String.t() típusú levélben lévő szó "en" helyett "in"-nel
  def map_replace_string(bt), do: mirrormap(bt, myfun)
  # A levél értéke {} lesz, ha az erdeti levél értéke [] volt
  def map_replace_list(bt), do: mirrormap(bt, myfun)
end
t1 = {{{{3,6},{2,4}},{4,8}},{{5,10},{{1,2},{6,12}}}}
t2 = {{{{3,6},{2,4}},4},{5,{{1,2},6}}}
t3 = {{{{"itt","ott"},{"pont","font"}},{"porta","forte"}},{"szent",{"ment",{"lent","fent"}}}}
t4 = {{{{1,2},{}},{}},{{},{{},{}}}}
(T3.map_identical(t1) ===
 {{{{12, 6}, {2, 1}}, {10, 5}}, {{8, 4}, {{4, 2}, {6, 3}}}}) |> IO.inspect()
(T3.map_parity(t1) ===
 {{{{false, false}, {false, true}}, {false, true}},
  {{false, false}, {{false, false}, {false, true}}}}) |> IO.inspect()
(T3.map_div_by_3(t2) ===
 {{{2, {0, 0}}, 1}, {1, {{1, 0}, {2, 1}}}} )|> IO.inspect()
(T3.map_enclose(t3) ===
 {{{{"{fent}", "{lent}"}, "{ment}"}, "{szent}"},
  {{"{forte}", "{porta}"}, {{"{font}", "{pont}"}, {"{ott}", "{itt}"}}}}) |> IO.inspect()
(T3.map_replace_string(t3) ===
 {{{{"fint", "lint"}, "mint"}, "szint"}, {{"forte", "porta"},
   {{"font", "pont"}, {"ott", "itt"}}}}) |> IO.inspect()
(T3.map_replace_list(t4) ===
 {{{{}, {}}, {}}, {{}, {{}, {2, 1}}}}) |> IO.inspect()
```

4. feladat: Listakezelés rekurzív függvénnyel

Árkok

Egy egészlistában ároknak nevezzük a csupa negatív számból álló, legalább két elemű, egyik irányban sem kiterjeszthető, folytonos sorozatokat. Írjon olyan függvényt trenches néven, amelyik a paraméterként átadott listából kigyűjti az árkokat alkotó részlistákat, és ezek listáját adja eredményül, az eredeti sorrend megtartásával.

```
defmodule T4 do
  @spec trenches(xs::[integer()]) :: rss::[[integer()]]
  # Az rss az xs-ben előforduló árkok listája
  # Ároknak nevezzük a legalább két negatív egészből álló, egyik irányban
  # sem kiterjeszthető, folytonos sorozatot                    (8 pont)
```

```
 def trenches(xs) do
   ...
 end
end
(T4.trenches([1,2,3,-1,-1,4,4,-2,5,0,-6,-7,-8,9])
 === [[-1, -1],[-6, -7, -8]]) |> IO.inspect()
(T4.trenches('abcd' ++ Range.to_list(-1..-4 )++ 'x' ++ [0,0,0] ++ 'yyy')
 === [[-1, -2, -3, -4]]) |> IO.inspect()
(T4.trenches([0,0,0,0])       === []) |> IO.inspect()
(T4.trenches([1,2,3,4])       === []) |> IO.inspect()
(T4.trenches([])            === []) |> IO.inspect()
(T4.trenches([-5])           === []) |> IO.inspect
(T4.trenches([-4,-5])         === [[-4,-5]]) |> IO.inspect
(T4.trenches([-4,0,-6,0,-9])    === []) |> IO.inspect()
(T4.trenches([-4,0,-6,0,-9,-18]) === [[-9, -18]]) |> IO.inspect()
```

5. Bináris fa legkisebb értékű levele

Egy bináris fa típusát így definiáljuk:

```
@type bintree() :: any() | {bintree(), bintree()}
```

írjon függvényt minint néven egy bináris fában lévő legkisebb egész szám visszaadására! Ha a fában nincs egész szám, a függvény visszatérési értéke nil legyen. Segédfüggvényt definiálhat.

```
defmodule T5 do
 @type bintree() :: any() | {bintree(), bintree()}
 @spec minint(t :: bintree()) :: v :: integer()
 # v a legkisebb egész szám a t fában                    (8 pont)
 def minint(t) do
   ...
 end
end

t0a = 8
t0b = :a
t1a = {6, {5, 4}}
t1b = {:a, {5, 4}}
t1c = {:a, {:b, :c}}
t2a = {{7, 8}, 6}
t2b = {{:a, 9}, 6}
t3a = {4, {3, {5, {9, {13, {7, {11, 12}}}}}}}
t3b = {:a, {3, {5, {9, {15, {7, {11, 12}}}}}}}
t4a = {{{{{{{12, 11}, 7}, 16}, 9}, 5}, 3}, 4}
t4b = {{{{{{{:a, {}}, []}, 17}, 9}, 5}, 3}, 4}
t5a = {{{{{4, 3}, 8}, 9}, {6, {5, 10}}}, {11, {7, {1, 2}}}}
```

```
t5b = {{{{{:a, :b}, 8}, :c}, {6, {5, 10}}}, {12, {7, {1, 2}}}}

(T5.minint(t0a) === 8)   |> IO.inspect()
(T5.minint(t0b) === nil) |> IO.inspect()
(T5.minint(t1a) === 4)   |> IO.inspect()
(T5.minint(t1b) === 4)   |> IO.inspect()
(T5.minint(t1c) === nil) |> IO.inspect()
(T5.minint(t2a) === 6)   |> IO.inspect()
(T5.minint(t2b) === 6)   |> IO.inspect()
(T5.minint(t3a) === 3)   |> IO.inspect()
(T5.minint(t3b) === 3)   |> IO.inspect()
(T5.minint(t4a) === 3)   |> IO.inspect()
(T5.minint(t4b) === 3)   |> IO.inspect()
(T5.minint(t5a) === 1)   |> IO.inspect()
(T5.minint(t5b) === 1)   |> IO.inspect()
```