Adatfeldolgozási megközelítések: a hagyományos módszereken túl

Zoom webinárium, 2021.10.26.

Előadó: Dr. Faragó Csaba

Áttekintés

	1. feladat	2. feladat	Műveletek
Hagyományos módszer	Python	Python	változók, for, while , if, …
Adatfolyam	Scala, Java, Python	Scala, Java, Python	filter, map, groupby, reduce, …
Adat keret	Numpy	SQL, Pandas, Spark, R	oszlop- és sorműveletek, merge, …

http://faragocsaba.hu/python-data-approaches

	1	2	Műv
Trad			
Stream			
DF			

1. feladat – hagyományos módszer – Python

Adott egy egész számokból álló tömb. A páros számok felét adjuk össze!

```
numbers = [4, 3, 6, 8, 5, 7]

def cyclish(numbers):
    result = 0
    for number in numbers:
        if number % 2 == 0:
            half_number = number // 2
            result += half_number
    return result

print(cyclish(numbers))
```

	1	2	Műv
Trad			
Stream			
DF			

1. feladat – folyam (stream)

Jellemzők:

- nincsenek változók
- megoldás: `return …`
- fluent API

Főbb folyam műveletek:

- szűrés (`filter`)
- átalakítás (`map`)
- redukálás (`reduce`)

1. feladat – folyam – Scala

	1	2	Műv
Trad			
Stream			
DF			

```
def streamish(numbers: List[Int]): Int = {
    return numbers
        .filter(x => x % 2 == 0)
        .map(x => x / 2)
        .reduce((x, y) => x + y)
}
```

```
def streamish(numbers: List[Int]): Int = {
    return numbers
        .filter(_ % 2 == 0)
        .map(_ / 2)
        .reduce(_ + _)
}
```

1. feladat – folyam – Java

	1	2	Műv
Trad			
Stream			
DF			

```
public static int streamish(List<Integer> numbers)
{
    return numbers.stream()
            .filter(x -> x % 2 == 0)
            .map(x -> x / 2)
            .reduce((x, y) -> x + y)
            .get();
}
```

1. feladat – folyam - Python

	1	2	Műv
Trad			
Stream			
DF			

```python
from functools import reduce

def streamish(numbers):
    return reduce(lambda x, y: x + y,
        map(lambda x: x // 2,
        filter(lambda x: x % 2 == 0, numbers)))
```

1. feladat – adatkeret – Numpy

	1	2	Műv
Trad			
Stream			
DF			

```python
import numpy as np

def numpyish(numbers):
    np_numbers = np.array(numbers)
    np_evens = np_numbers[np_numbers % 2 == 0]
    np_halves = np_evens // 2
    np_sum = np.sum(np_halves)
    return np_sum
```

	1	2	Műv
Trad			
Stream			
DF			

2. feladat

Tegyük fel, hogy az adatunk "táblázatszerű": két oszlopa van, az egyik gyümölcsnevekkel, a másik pedig egész értékekkel, az adott gyümölcs darabszámával. A feladat kiszűrni a negatív darabszámokat, majd összegezni az adott gyümölcsök darabszámát.

apple	3
banana	2
apple	-2
orange	4
banana	-3
apple	2
banana	5

- apple: 5
- banana: 7
- orange: 4

	1	2	Műv
Trad			
Stream			
DF			

2. feladat – hagyományos módszer – Python

```
fruit_pieces = [
    ('apple', 3),
    ('banana', 2),
    ('apple', -2),
    ('orange', 4),
    ('banana', -3),
    ('apple', 2),
    ('banana', 5),
]
```

```
def cyclish(fruit_pieces):
    result = {}
    for one_fruit_piece in fruit_pieces:
        fruit, pieces = one_fruit_piece
        if pieces > 0:
            if fruit in result:
                result[fruit] = result[fruit] + pieces
            else:
                result[fruit] = pieces
    return result
```

	1	2	Műv
Trad			
Stream			
DF			

2. feladat – folyam – Scala

```
def streamish(fruitPieces: List[(String, Int)]): Map[String, Int] = {
    return fruitPieces
        .filter((fruit, pieces) => pieces > 0)
        .groupMapReduce
            ((fruit, pieces) => fruit)
            ((fruit, pieces) => pieces)
            ((pieces1, pieces2) => pieces1 + pieces2)
}
```

```
def streamish(fruitPieces: List[(String, Int)]): Map[String, Int] =
    return fruitPieces
        .filter(_._2 > 0)
        .groupMapReduce(_._1)(_._2)(_ + _)
```

2. feladat – folyam – Java

	1	2	Műv
Trad			
Stream			
DF			

```
class FruitPieces {
    private String fruit;
    private int pieces;
    ...
}

import static java.util.stream.Collectors.*;
import java.util.*;

public static Map<Object, Integer> streamish(List<FruitPieces> fruitPieces) {
    return fruitPieces.stream()
            .filter(x -> x.getPieces() > 0)
            .collect(groupingBy(x -> x.getFruit(), summingInt(x -> x.getPieces())));
}
```

2. feladat – folyam – Python

	1	2	Műv
Trad			
Stream			
DF			

```python
from functools import reduce
from collections import Counter

def streamish(fruit_pieces):
    return dict(
        reduce(lambda x, y: x + y,
        map(lambda fp: Counter({fp[0]: fp[1]}),
        filter(lambda fp: fp[1] > 0, fruit_pieces))))
```

2. feladat – adatkeret – SQL

	1	2	Műv
Trad			
Stream			
DF			

```sql
SELECT fruit, SUM(pieces) FROM fruits WHERE pieces > 0 GROUP BY fruit;
```

2. feladat – adatkeret – Pandas

	1	2	Műv
Trad			
Stream			
DF			

```
import pandas as pd

def dataframish(fruit_pieces):
    fruit_pieces_df = pd.DataFrame(fruit_pieces, columns=['fruit', 'pieces'])
    fruit_pieces_filtered = fruit_pieces_df[fruit_pieces_df['pieces'] > 0]
    fruit_pieces_grouped = fruit_pieces_filtered.groupby(['fruit'], as_index=False).sum()
    return fruit_pieces_grouped.set_index('fruit').to_dict()['pieces']
```

2. feladat – adatkeret – Spark

	1	2	Műv
Trad			
Stream			
DF			

```
from pyspark.sql.functions import sum, col, asc

fruits_df\
  .select('fruit', 'pieces')\
  .where(col('pieces') > 0)\
  .groupBy('fruit')\
  .agg(sum('pieces').alias('pieces'))\
  .show()
```

2. feladat – adatkeret – R

	1	2	Műv
Trad			
Stream			
DF			

```r
fruits <- data.frame(
  fruit = c('apple', 'banana', 'apple', 'orange', 'banana', 'apple', 'banana'),
  pieces = c(3, 2, -2, 4, -3, 2, 5)
)

fruits_filtered = fruits[fruits$pieces > 0,]
fruits_aggregated = aggregate(pieces ~ fruit, fruits_filtered, sum)
print(fruits_aggregated)
```

	1	2	Műv
Trad			
Stream			
DF			

Klasszikus műveletek

- változók
- feltételkezelés (`if … else …`, `switch ... case ...`)
- ciklusok (`for`, `while`)
- segédfüggvények
- stb.

	1	2	Műv
Trad			
Stream			
DF			

Folyam műveletek

- **Nem végműveletek: a bemenet és a kimenet is folyam**
  - szűrés (`filter`),
  - átalakítás (`map`),
  - egy művelet végrehajtása a folyam minden elemén (`foreach`),
  - csoportosítás (`group`),
  - duplikátumok kiszűrése (`distinct`),
  - "kivasalás" (`flatten`; listák listájából listát készít),
  - rendezés (`sort`),
  - alfolyam képzése (`take`, `takeRight`, `takeWhile`),
  - stb.
- **Végműveletek: a bemenet folyam, a kimenet nem**
  - a folyam átalakítása listává vagy más nem folyam adatszerkezetté (`collect`, `copyToArray`),
  - redukálás (`reduce`; pl. az elemek összeadása, összeszorzása),
  - minimum vagy maximum elem megkeresése (`min`, `max`),
  - string készítése a folyam elemeiből (`mkString`)
  - egy adott elem megkeresése (pl. az első vagy utolsó előfordulás: `find`, `last`)
  - annak ellenőrzése, hogy egy adott feltétel igaz-e mindegyikre (`forall`), vagy legalább egy elemre (`exists`),
  - elemek megszámolása (`count`)
  - stb.

	1	2	Műv
Trad			
Stream			
DF			

Adatkeret műveletek

- **oszlopok kiválasztása**;
- **sorok szűrése** valamilyen oszlop érték (vagy értékek) alapján;
- **csoportosítás**, műveletek végrehajtása csoporton belül;
- **sorba rendezés**;
- **oszlopok hozzáadása**, beleértve a külső adatforrást és más oszlopokkal történő átalakítást;
- **adatkeretek összekapcsolása**, a hagyományos, 4 féle összekapcsoló művelettel (inner, left, right, outer),
- és még nagyon sok egyéb.

Köszönöm a figyelmet!