Aggregations using reduce

Let us understand how to perform global aggregations using `reduce`.

- We can use `reduce` on top of `iterable` to return aggregated result.
- It takes aggregation logic and iterable as arguments. We can pass aggregation logic either as regular function or lambda function.
- `reduce` returns objects of type `int`, `float` etc. It is typically of type elements in the collection that is being processed.
- Unlike `map` and `filter` we need to import `reduce` from functools.

```
%run 02_preparing_data_sets.ipynb
```

```
orders[:10]
```

```
len(orders)
```

```
order_items[:10]
```

```
len(order_items)
```

```
orders[:10]
```

```
order_items[:10]
```

Task 1

Use orders and get total number of records for a given month (201401).

- Filter the data.
- Perform row level transformation by changing each record to 1.
- Use reduce to aggregate.

```
orders[:10]
```

```
order = '1,2013-07-25 00:00:00.0,11599,CLOSED'
```

```
order.split(',')
```

```
order.split(',')[1]
```

```
order.split(',')[1][:7]
```

```
order.split(',')[1][:7].replace('-', '')
```

```
int(order.split(',')[1][:7].replace('-', ''))
```

```
orders_filtered = filter(
    lambda order: int(order.split(',')[1][:7].replace('-', '')) == 201307,
    orders
)
```

```
orders_mapped = map(
    lambda order: 1,
    orders
)
```

Contents

```
from functools import reduce
reduce?
```

```
reduce(
    lambda tot, ele: tot + ele,
    orders_mapped
)
```

Task 2

Use order items data set and compute total revenue generated for a given product_id.

- Filter for given product_id.
- Extract order_item_subtotal for each item.
- Aggregate to get the revenue for a given product id.

```
order_items[:10]
```

```
order_item = '1,1,957,1,299.98,299.98'
```

```
order_item.split(',')
```

```
order_item.split(',')[2]
```

```
int(order_item.split(',')[2])
```

```
float(order_item.split(',')[4])
```

```
items_for_product = filter(
    lambda order_item: int(order_item.split(',')[2]) == 502,
    order_items
)
```

```
list(items_for_product)[:10]
```

```
items_for_product = filter(
    lambda order_item: int(order_item.split(',')[2]) == 502,
    order_items
)
item_subtotals = map(
    lambda order_item: float(order_item.split(',')[4]),
    items_for_product
)
```

```
list(item_subtotals)[:10]
```

```
items_for_product = filter(
    lambda order_item: int(order_item.split(',')[2]) == 502,
    order_items
)
item_subtotals = map(
    lambda order_item: float(order_item.split(',')[4]),
    items_for_product
)
reduce(
    lambda total_revenue, item_revenue: total_revenue + item_revenue,
    item_subtotals
)
```

Note

We can also aggregate using functions such as `add`, `min`, `max` etc to get the aggregated results.

```
from operator import add
items_for_product = filter(
    lambda order_item: int(order_item.split(',')[2]) == 502,
    order_items
)
item_subtotals = map(
    lambda order_item: float(order_item.split(',')[4]),
    items_for_product
)
reduce(
    add,
    item_subtotals
)
```

```
items_for_product = filter(
    lambda order_item: int(order_item.split(',')[2]) == 502,
    order_items
)
item_subtotals = map(
    lambda order_item: float(order_item.split(',')[4]),
    items_for_product
)
reduce(
    min,
    item_subtotals
)
```

Task 3

Use order items data set and get total number of items sold as well as total revenue generated for a given product_id.

```
t1 = (1, 200.0)
```

```
t2 = (2, 300.0)
```

```
res = (0, 0.0)
```

```
res = (res[0] + t1[0], res[1] + t1[1])
```

```
res
```

```
res = (res[0] + t2[0], res[1] + t2[1])
```

```
res
```

```
items_for_product = filter(
    lambda order_item: int(order_item.split(',')[2]) == 502,
    order_items
)
```

```
list(items_for_product)[:10]
```

```
items_for_product = filter(
    lambda order_item: int(order_item.split(',')[2]) == 502,
    order_items
)
item_details = map(
    lambda order_item: (int(order_item.split(',')[3]), float(order_item.split(',')[4])),
    items_for_product
)
```

```
list(item_details)[:10]
```

```
items_for_product = filter(
    lambda order_item: int(order_item.split(',')[2]) == 502,
    order_items
)
item_details = map(
    lambda order_item: (int(order_item.split(',')[3]), float(order_item.split(',')[4])),
    items_for_product
)
reduce(
    lambda tot, ele: (tot[0] + ele[0], tot[1] + ele[1]),
    item_details
)
```


```
items_for_product = filter(
    lambda order_item: int(order_item.split(',')[2]) == 502,
    order_items
)
item_details = map(
    lambda order_item: (int(order_item.split(',')[3]), float(order_item.split(',')[4])),
    items_for_product
)
reduce(
    lambda tot, ele: (tot[0] + ele[0], tot[1] + ele[1]),
    item_details
)
```


Task 4

Create a collection with sales and commission percentage. Using that collection compute total commission amount.
If the commission percent is None or not present, treat it as 0.

- Each element in the collection should be a tuple.
- First element is the sales amount and second element is commission percentage.
- Commission for each sale can be computed by multiplying commission percentage with sales (make sure to divide commission percentage by 100).
- Some of the records does not have commission percentage, in that case commission amount for that sale shall be 0

```
transactions = [(376.0, 8),
(548.23, 14),
(107.93, 8),
(838.22, 14),
(846.85, 21),
(234.84,),
(850.2, 21),
(992.2, 21),
(267.01,),
(958.91, 21),
(412.59,),
(283.14,),
(350.01, 14),
(226.95,),
(132.7, 14)]
```

```
type(transactions)
```

```
transactions[:6]
```

```
sale = transactions[0]
```

```
type(sale)
```

```
commission_amount = round(sale[0] * (sale[1] / 100), 2)
```

```
commission_amount
```

```
sale = (234.84,)
```

```
commission_amount = round(sale[0] * (sale[1] / 100), 2) # errors out
```

```
len(sale)
```

```
commission_pct = sale[1] / 100 if len(sale) == 2 else 0
```

```
commission_pct
```

```
transactions_fixed = map(
    lambda sale: sale[0] * (sale[1] / 100 if len(sale) == 2 else 0),
    transactions
)
```

```
list(transactions_fixed)
```

```
transactions_fixed = map(
    lambda sale: sale[0] * (sale[1] / 100 if len(sale) == 2 else 0),
    transactions
)
reduce(
    lambda tot, ele: round(tot + ele, 2),
    transactions_fixed
)
```

Note

Using `map` function call as argument.

```
reduce(
    lambda tot, ele: round(tot + ele, 2),
    map(
        lambda sale: sale[0] * (sale[1] / 100 if len(sale) == 2 else 0),
        transactions
    )
)
```

```
round(
    reduce(
        add,
        map(
            lambda sale: sale[0] * (sale[1] / 100 if len(sale) == 2 else 0),
            transactions
        )
    ), 2
)
```

By Durga Gadiraju