Return the Total Price Per Customer

- The function maps the price to the cust_id for each document and emits the cust_id and price.

```
var mapFunction1 = function() {
   emit(this.cust_id, this.price);
}
```

- The function reduces the valuesPrice array to the sum of its elements

```
var reduceFunction1 = function(keyCustId, valuesPrices) {
   return Array.sum(valuesPrices);
};
```

- Perform map-reduce on all documents in the orders collection using the mapFunction1 map function and the reduceFunction1 reduce function

```
var reduceFunction1 = function(keyCustId, valuesPrices) {
   return Array.sum(valuesPrices);
};
```

- Perform map-reduce on all documents in the orders collection using the mapFunction1 map function and the reduceFunction1 reduce function

```
db.orders.mapReduce(
   mapFunction1,
   reduceFunction1,
   { out: "map_reduce_example" }
)
```

- Query the map_reduce_example collection to verify the results
  `db.map_reduce_example.find().sort( { _id: 1 } )`

Aggregation Alternative

```
db.orders.aggregate([
   { $group: { _id: "$cust_id", value: { $sum: "$price" } } },
   { $out: "agg_alternative_1" }
])
```

```
db.agg_alternative_1.find().sort( { _id: 1 } )
```

Calculate Order and Total Quantity with Average Quantity Per Item

- For each item, the function associates the sku with a new object value that contains the count of 1 and the item qty for the order and emits the sku (stored in the key) and the value.

```
var mapFunction2 = function() {
    for (var idx = 0; idx < this.items.length; idx++) {
       var key = this.items[idx].sku;
       var value = { count: 1, qty: this.items[idx].qty };
       emit(key, value);
    }
};
```

- The function reduces the countObjVals array to a single object reducedValue that contains the count and the qty fields

```
var reduceFunction2 = function(keySKU, countObjVals) {
   reducedVal = { count: 0, qty: 0 };
   for (var idx = 0; idx < countObjVals.length; idx++) {
       reducedVal.count += countObjVals[idx].count;
       reducedVal.qty += countObjVals[idx].qty;
   }
   return reducedVal;
};
```

- Define a finalize function with two arguments key and reducedVal. The function modifies the reducedVal object to add a computed field named avg and returns the modified object

```
var finalizeFunction2 = function (key, reducedVal) {
  reducedVal.avg = reducedVal.qty/reducedVal.count;
  return reducedVal;
};
```

- Perform the map-reduce operation on the orders collection using the mapFunction2, reduceFunction2, and finalizeFunction2 functions

```
db.orders.mapReduce(
   mapFunction2,
   reduceFunction2,
   {
     out: { merge: "map_reduce_example2" },
     query: { ord_date: { $gte: new Date("2020-03-01") } },
     finalize: finalizeFunction2
   }
 );
```

- Query the map_reduce_example2 collection to verify the resultsdb.map_reduce_example2.find().sort({ _id: 1 })

Aggregation Alternative

```
db.orders.aggregate( [
  { $match: { ord_date: { $gte: new Date("2020-03-01") } } },
  { $unwind: "$items" },
  { $group: { _id: "$items.sku", qty: { $sum: "$items.qty" }, orders_ids: { $addToSet: "$_id" } }  },
  { $project: { value: { count: { $size: "$orders_ids" }, qty: "$qty", avg: { $divide: [ "$qty", { $size: "$orders_ids" } ] } } } },
  { $merge: { into: "agg_alternative_3", on: "_id", whenMatched: "replace",  whenNotMatched: "insert" } }
] )
```

```
db.agg_alternative_3.find().sort( { _id: 1 } )
```