JSON Assignment

Exercise
1

```
import json
with open('interface-data.json') as json_file:
    data = json.load(json_file)
print("Interface Status")
print("DN\tDescription\tSpeed\tMTU")
for interface in data:
    print("{}\t{}\t{}\t{}".format(
        interface['dn'],
        interface['descr'],
        interface['speed'],
        interface['mtu']
    ))
```

Exercise 2

```
import json
import statistics
with open('rows.json') as f:
    data = json.load(f)
airport_data = {}
for record in data['data']:
    airline = record[9]
    passengers = int(record[10])
    if airline in airport_data:
        airport_data[airline].append(passengers)
    else:
        airport_data[airline] = [passengers]
for airline in airport_data:
    average = round(statistics.mean(airport_data[airline]), 2)
    maximum = max(airport_data[airline])

    print(airline + ': Average = ' + str(average) + ', Max = ' + str(maximum))
```

Orders - accidental double-entry details, derived table

```
SELECT OrderDetails.OrderID, OrderDetails.ProductID, OrderDetails.Quantity,
Orders.OrderDate
      FROM (SELECT DISTINCT OrderID
            FROM OrderDetails
            WHERE Quantity >= 60) AS DerivedTable
      INNER JOIN OrderDetails
      ON DerivedTable.OrderID = OrderDetails.OrderID
      INNER JOIN Orders
      ON OrderDetails.OrderID = Orders.OrderID
      ORDER BY OrderDetails.OrderID;
```

Late orders vs. total orders - fix null

```
SELECT EmployeeID, COUNT(*) AS NumLateOrders,
      COALESCE(COUNT(*) / (SELECT COUNT(*) FROM Orders WHERE EmployeeID =
o.EmployeeID) * 100, 0) AS PctLateOrders
FROM Orders o
WHERE OrderDate > DueDate OR DueDate IS NULL
GROUP BY EmployeeID
ORDER BY PctLateOrders DESC;
```

Late orders vs. total orders - percentage

```
SELECT EmployeeID, COUNT(*) AS NumLateOrders,
    COALESCE(COUNT(*) / (SELECT COUNT(*) FROM Orders WHERE EmployeeID =
o.EmployeeID) * 100, 0) AS PctLateOrders
FROM Orders o
WHERE OrderDate > DueDate OR DueDate IS NULL
GROUP BY EmployeeID
ORDER BY PctLateOrders DESC;
```

Late orders vs. total orders - fix decimal

```
SELECT EmployeeID, COUNT(*) AS NumLateOrders,
    ROUND(COALESCE(COUNT(*) / (SELECT COUNT(*) FROM Orders WHERE EmployeeID
= o.EmployeeID) * 100, 0), 2) AS PctLateOrders
FROM Orders o
WHERE OrderDate > DueDate OR DueDate IS NULL
GROUP BY EmployeeID
ORDER BY PctLateOrders DESC;
```

Customer grouping - fix null

```
SELECT CustomerID,
  CASE
    WHEN SUM(Quantity * UnitPrice) BETWEEN 0 AND 1000 THEN '0 to 1,000'
    WHEN SUM(Quantity * UnitPrice) BETWEEN 1000 AND 5000 THEN '1,000 to 5,000'
    WHEN SUM(Quantity * UnitPrice) BETWEEN 5000 AND 10000 THEN '5,000 to 10,000'
    WHEN SUM(Quantity * UnitPrice) > 10000 THEN 'Over 10,000'
  END AS CustomerGroup
FROM Orders
JOIN OrderDetails USING (OrderID)
WHERE YEAR(OrderDate) = 2016
GROUP BY CustomerID
HAVING SUM(Quantity * UnitPrice) > 0
ORDER BY CustomerID;
```

Customer grouping with percentage

```
SELECT CustomerGroup, ROUND(COUNT(CustomerGroup) * 100.0 / (SELECT COUNT(*)
FROM Orders WHERE YEAR(OrderDate) = 2016), 2) AS Percentage
FROM (
  SELECT CustomerID,
    CASE
      WHEN SUM(Quantity * UnitPrice) BETWEEN 0 AND 1000 THEN '0 to 1,000'
      WHEN SUM(Quantity * UnitPrice) BETWEEN 1000 AND 5000 THEN '1,000 to 5,000'
      WHEN SUM(Quantity * UnitPrice) BETWEEN 5000 AND 10000 THEN '5,000 to
10,000'
      WHEN SUM(Quantity * UnitPrice) > 10000 THEN 'Over 10,000'
```

```
        END AS CustomerGroup
    FROM Orders
    JOIN OrderDetails USING (OrderID)
    WHERE YEAR(OrderDate) = 2016
    GROUP BY CustomerID
    HAVING SUM(Quantity * UnitPrice) > 0
) AS Customer_Grouping
GROUP BY CustomerGroup
ORDER BY COUNT(CustomerGroup) DESC;
```

Customer grouping-flexible

```
SELECT CustomerID,
    CASE
        WHEN SUM(Quantity * UnitPrice) <= (SELECT LowValue FROM
CustomerGroupThreshold) THEN 'Low'
        WHEN SUM(Quantity * UnitPrice) > (SELECT LowValue FROM
CustomerGroupThreshold)
            AND SUM(Quantity * UnitPrice) <= (SELECT MediumValue FROM
CustomerGroupThreshold) THEN 'Medium'
        WHEN SUM(Quantity * UnitPrice) > (SELECT MediumValue FROM
CustomerGroupThreshold)
            AND SUM(Quantity * UnitPrice) <= (SELECT HighValue FROM
CustomerGroupThreshold) THEN 'High'
        WHEN SUM(Quantity * UnitPrice) > (SELECT HighValue FROM
CustomerGroupThreshold) THEN 'Very High'
    END AS CustomerGroup
FROM Orders
JOIN OrderDetails USING (OrderID)
WHERE YEAR(OrderDate) = 2016
GROUP BY CustomerID
HAVING SUM(Quantity * UnitPrice) > 0
ORDER BY CustomerID;
```