wilan057
chrvu878
2020-05-06

Lab report 1 - Big Data Analytics TDDE31

1) The output should contain the following information:
Year, temperature

```
(u'1975', 36.1)  (u'1990', -35.0)
(u'1992', 35.4)  (u'1952', -35.5)
(u'1994', 34.7)  (u'1974', -35.6)
(u'2014', 34.4)  (u'1954', -36.0)
(u'2010', 34.4)  (u'1992', -36.1)
(u'1989', 33.9)  (u'1975', -37.0)
(u'1982', 33.8)  (u'1972', -37.5)
(u'1968', 33.7)          .
        .                .
        .                .
        .        (u'1967', -45.4)
(u'1960', 29.4)  (u'1987', -47.3)
(u'1950', 29.4)  (u'1978', -47.7)
(u'1998', 29.2)  (u'1999', -49.0)
(u'1951', 28.5)  (u'1966', -49.4)
(u'1965', 28.5)
(u'1962', 27.4)
```

Max temperature *Min temperature*

wilan057
chrvu878
2020-05-06

2) The output should contain the following information:
Year, month, (distinct counts)

```
((1980, 10), 7079)
((2008, 6), 102900)
((1956, 2), 3)
((1999, 3), 1262)
((1981, 9), 37266)
((2009, 5), 60867)
((2002, 8), 126073)
((1990, 12), 2)
((1961, 5), 16470)
((1950, 4), 352)
((1991, 11), 138)
        .
        .
        .
((1964, 3), 54)
((1988, 11), 1)
((1952, 7), 13412)
((2005, 8), 113950)
((1989, 8), 67793)
((1961, 12), 8)
((1968, 7), 52838)
((2010, 9), 74816)
((1954, 1), 3)
((1962, 9), 24977)
((1958, 5), 9076)
```

Count of instances above 10

```
((1980, 10), 249)
((2008, 6), 316)
((1956, 2), 2)
((1999, 3), 148)
((1981, 9), 335)
((2009, 5), 308)
((1961, 5), 268)
((2002, 8), 322)
((1989, 1), 23)
((1990, 12), 2)
((1965, 9), 358)
        .
        .
        .
((2000, 7), 320)
((1980, 3), 3)
((1988, 11), 1)
((1952, 7), 115)
((1978, 9), 358)
((1953, 4), 104)
((1961, 12), 7)
((1958, 5), 127)
((2010, 9), 316)
((1954, 1), 3)
((1994, 9), 299)
```

Distinct count of instances above 10

wilan057
chrvu878
2020-05-06

3) The output should contain the following information:
Year, *month*, *station number*, *average monthly temperature*

```
((2008, 7, 158740), 13.720967741935485)
((1993, 12, 63340), 1.7806451612903227)
((1999, 4, 85490), 5.6949999999999985)
((2003, 10, 107440), 5.096774193548387)
((1973, 12, 76160), -1.170967741935484)
((1960, 9, 82650), 10.590000000000002)
((1968, 7, 73660), 14.649999999999999)
((2008, 7, 104580), 16.111290322580643)
((1966, 7, 188830), 10.606451612903225)
((1993, 8, 62400), 14.670967741935485)
((1988, 3, 106100), -3.4209677419354843)
                .
                .
                .
((1999, 5, 86340), 9.229032258064516)
((1970, 4, 65130), 3.3049999999999997)
((1985, 3, 63500), -0.15645161290322585)
((1965, 5, 63530), 9.283870967741935)
((1966, 10, 64620), 8.024193548387096)
((1980, 6, 64330), 16.308333333333334)
((1962, 8, 83230), 13.580645161290322)
((1977, 11, 75100), 2.653333333333333)
((2000, 2, 65450), 1.3517241379310345)
((1966, 8, 103050), 13.933870967741937)
((1975, 9, 96120), 12.313333333333333)
```

Average monthly temperature per station

4) Empty output - no stations fulfill the conditions.

wilan057
chrvu878
2020-05-06

5) The output should contain the following information:
Year, *month*, *average monthly precipitation*

```
((u'2012', u'09'), 72.75)
((u'1995', u'05'), 26.00000000000002)
((u'1997', u'09'), 43.56666666666667)
((u'2011', u'08'), 86.26666666666665)
((u'2007', u'04'), 21.249999999999996)
((u'2007', u'06'), 108.95)
((u'1993', u'04'), 0.0)
((u'2011', u'10'), 43.75000000000001)
((u'2014', u'10'), 72.13749999999999)
((u'1996', u'09'), 57.46666666666667)
((u'2002', u'05'), 72.13333333333334)
                .
                .
                .
((u'2000', u'10'), 110.29999999999997)
((u'2012', u'03'), 8.549999999999999)
((u'2011', u'07'), 94.91666666666664)
((u'2013', u'02'), 25.52500000000002)
((u'2000', u'07'), 135.8666666666666)
((u'1997', u'03'), 9.55)
((u'2000', u'05'), 25.31666666666668)
((u'2005', u'11'), 32.600000000000016)
((u'2001', u'08'), 69.96666666666668)
((u'1996', u'04'), 8.1)
((u'2011', u'05'), 37.85)
```

Average monthly precipitation in Östergötland region

wilan057
chrvu878
2020-05-06

Appendix

Exercise1

Picture:

```python
from pyspark import SparkContext
# Definition for calculating the maximum value
def max_temperature(a,b):
    if a>=b:
        return a
    else:
        return b
# Definition for calculating the minimum value
def min_temperature(a,b):
    if a<=b:
        return a
    else:
        return b
sc = SparkContext(appName = "exercise 1")
# Read file from hadoop
temperature_file=sc.textFile("BDA/input/temperature-readings.csv")
# Split features of the file separated by a ';'
lines = temperature_file.map(lambda line: line.split(";"))
# Map key as year and value as temperature
year_temperature = lines.map(lambda x: (x[1][0:4], float(x[3])))
# Filter relevant data
year_temperature = year_temperature.filter(lambda x: int(x[0])>=1950 and int(x[0])<=2014)
# Find max and min temperature for each key (year)
max_temperatures = year_temperature.reduceByKey(max_temperature)
min_temperatures = year_temperature.reduceByKey(min_temperature)
# Sort the result by descending temperature
max_temperaturesSorted = max_temperatures.sortBy(ascending = False, keyfunc=lambda k: k[1])
min_temperaturesSorted = min_temperatures.sortBy(ascending = False, keyfunc=lambda k: k[1])
# Save the file to a specified folder at hadoop
max_temperaturesSorted.saveAsTextFile("BDA/output/max_temperature")
min_temperaturesSorted.saveAsTextFile("BDA/output/min_temperature")
```

Text:

```
from pyspark import SparkContext
# Definition for calculating the maximum value
def max_temperature(a,b):
    if a>=b:
        return a
    else:
```

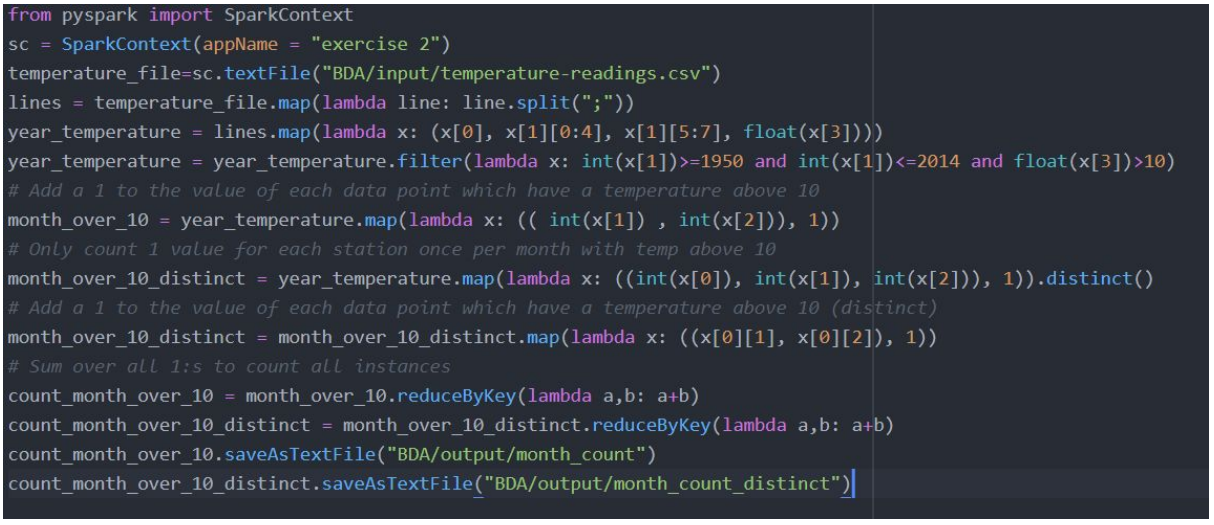
wilan057
chrvu878
2020-05-06

```
        return b
# Definition for calculating the minimum value
def min_temperature(a,b):
    if a<=b:
        return a
    else:
        return b
sc = SparkContext(appName = "exercise 1")
# Read file from hadoop
temperature_file=sc.textFile("BDA/input/temperature-readings.csv")
# Split features of the file separated by a ';'
lines = temperature_file.map(lambda line: line.split(";"))
# Map key as year and value as temperature
year_temperature = lines.map(lambda x: (x[1][0:4], float(x[3])))
# Filter relevant data
year_temperature = year_temperature.filter(lambda x: int(x[0])>=1950 and int(x[0])<=2014)
# Find max and min temperature for each key (year)
max_temperatures = year_temperature.reduceByKey(max_temperature)
min_temperatures = year_temperature.reduceByKey(min_temperature)
# Sort the result by descending temperature
max_temperaturesSorted = max_temperatures.sortBy(ascending = False, keyfunc=lambda
k: k[1])
min_temperaturesSorted = min_temperatures.sortBy(ascending = False, keyfunc=lambda k:
k[1])
# Save the file to a specified folder at hadoop
max_temperaturesSorted.saveAsTextFile("BDA/output/max_temperature")
min_temperaturesSorted.saveAsTextFile("BDA/output/min_temperature")
```

wilan057
chrvu878
2020-05-06

Exercise 2

Picture:



```
from pyspark import SparkContext
sc = SparkContext(appName = "exercise 2")
temperature_file=sc.textFile("BDA/input/temperature-readings.csv")
lines = temperature_file.map(lambda line: line.split(";"))
year_temperature = lines.map(lambda x: (x[0], x[1][0:4], x[1][5:7], float(x[3])))
year_temperature = year_temperature.filter(lambda x: int(x[1])>=1950 and int(x[1])<=2014 and float(x[3])>10)
# Add a 1 to the value of each data point which have a temperature above 10
month_over_10 = year_temperature.map(lambda x: (( int(x[1]) , int(x[2])), 1))
# Only count 1 value for each station once per month with temp above 10
month_over_10_distinct = year_temperature.map(lambda x: ((int(x[0]), int(x[1]), int(x[2])), 1)).distinct()
# Add a 1 to the value of each data point which have a temperature above 10 (distinct)
month_over_10_distinct = month_over_10_distinct.map(lambda x: ((x[0][1], x[0][2]), 1))
# Sum over all 1:s to count all instances
count_month_over_10 = month_over_10.reduceByKey(lambda a,b: a+b)
count_month_over_10_distinct = month_over_10_distinct.reduceByKey(lambda a,b: a+b)
count_month_over_10.saveAsTextFile("BDA/output/month_count")
count_month_over_10_distinct.saveAsTextFile("BDA/output/month_count_distinct")
```

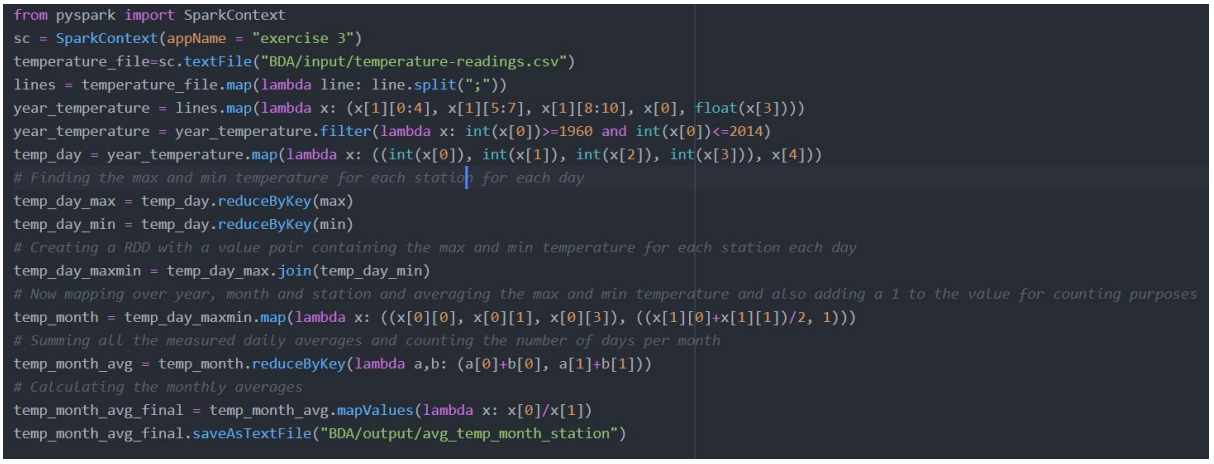
Text:

```
from pyspark import SparkContext
sc = SparkContext(appName = "exercise 2")
temperature_file=sc.textFile("BDA/input/temperature-readings.csv")
lines = temperature_file.map(lambda line: line.split(";"))
year_temperature = lines.map(lambda x: (x[0], x[1][0:4], x[1][5:7], float(x[3])))
year_temperature = year_temperature.filter(lambda x: int(x[1])>=1950 and int(x[1])<=2014
and float(x[3])>10)
# Add a 1 to the value of each data point which have a temperature above 10
month_over_10 = year_temperature.map(lambda x: (( int(x[1]) , int(x[2])), 1))
# Only count 1 value for each station once per month with temp above 10
month_over_10_distinct = year_temperature.map(lambda x: ((int(x[0]), int(x[1]), int(x[2])),
1)).distinct()
# Add a 1 to the value of each data point which have a temperature above 10 (distinct)
month_over_10_distinct = month_over_10_distinct.map(lambda x: ((x[0][1], x[0][2]), 1))
# Sum over all 1:s to count all instances
count_month_over_10 = month_over_10.reduceByKey(lambda a,b: a+b)
count_month_over_10_distinct = month_over_10_distinct.reduceByKey(lambda a,b: a+b)
count_month_over_10.saveAsTextFile("BDA/output/month_count")
count_month_over_10_distinct.saveAsTextFile("BDA/output/month_count_distinct")
```

wilan057
chrvu878
2020-05-06

Exercise 3

Picture:



```
from pyspark import SparkContext
sc = SparkContext(appName = "exercise 3")
temperature_file=sc.textFile("BDA/input/temperature-readings.csv")
lines = temperature_file.map(lambda line: line.split(";"))
year_temperature = lines.map(lambda x: (x[1][0:4], x[1][5:7], x[1][8:10], x[0], float(x[3])))
year_temperature = year_temperature.filter(lambda x: int(x[0])>=1960 and int(x[0])<=2014)
temp_day = year_temperature.map(lambda x: ((int(x[0]), int(x[1]), int(x[2]), int(x[3])), x[4]))
# Finding the max and min temperature for each station for each day
temp_day_max = temp_day.reduceByKey(max)
temp_day_min = temp_day.reduceByKey(min)
# Creating a RDD with a value pair containing the max and min temperature for each station each day
temp_day_maxmin = temp_day_max.join(temp_day_min)
# Now mapping over year, month and station and averaging the max and min temperature and also adding a 1 to the value for counting purposes
temp_month = temp_day_maxmin.map(lambda x: ((x[0][0], x[0][1], x[0][3]), ((x[1][0]+x[1][1])/2, 1)))
# Summing all the measured daily averages and counting the number of days per month
temp_month_avg = temp_month.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1]))
# Calculating the monthly averages
temp_month_avg_final = temp_month_avg.mapValues(lambda x: x[0]/x[1])
temp_month_avg_final.saveAsTextFile("BDA/output/avg_temp_month_station")
```

Text:

```
from pyspark import SparkContext
sc = SparkContext(appName = "exercise 3")
temperature_file=sc.textFile("BDA/input/temperature-readings.csv")
lines = temperature_file.map(lambda line: line.split(";"))
year_temperature = lines.map(lambda x: (x[1][0:4], x[1][5:7], x[1][8:10], x[0], float(x[3])))
year_temperature = year_temperature.filter(lambda x: int(x[0])>=1960 and int(x[0])<=2014)
temp_day = year_temperature.map(lambda x: ((int(x[0]), int(x[1]), int(x[2]), int(x[3])), x[4]))
# Finding the max and min temperature for each station for each day
temp_day_max = temp_day.reduceByKey(max)
temp_day_min = temp_day.reduceByKey(min)
# Creating a RDD with a value pair containing the max and min temperature for each station each day
temp_day_maxmin = temp_day_max.join(temp_day_min)
# Now mapping over year, month and station and averaging the max and min temperature and also adding a 1 to the value for counting purposes
temp_month = temp_day_maxmin.map(lambda x: ((x[0][0], x[0][1], x[0][3]), ((x[1][0]+x[1][1])/2, 1)))
# Summing all the measured daily averages and counting the number of days per month
temp_month_avg = temp_month.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1]))
# Calculating the monthly averages
temp_month_avg_final = temp_month_avg.mapValues(lambda x: x[0]/x[1])
temp_month_avg_final.saveAsTextFile("BDA/output/avg_temp_month_station")
```

wilan057
chrvu878
2020-05-06

Exercise 4

Picture:

```
from pyspark import SparkContext
sc = SparkContext(appName = "exercise 4")
temperature_file=sc.textFile("BDA/input/temperature-readings.csv")
precipitation_file=sc.textFile("BDA/input/precipitation-readings.csv")
lines_temperature = temperature_file.map(lambda line: line.split(";"))
lines_precipitation = precipitation_file.map(lambda line: line.split(";"))
temperature = lines_temperature.map(lambda x: (x[0], float(x[3])))
precipitation = lines_precipitation.map(lambda x: (x[0], float(x[3])))
# Finding the max temp and max precipitation for each station
temp_max = temperature.reduceByKey(max)
prec_max = precipitation.reduceByKey(max)
# Creating a RDD with a value pair containing the max temperature and max precipitation for each station
joined_max = temp_max.join(prec_max)
# FIltering all instances where the temperature and precipitation is within a specified range
filtered_data = joined_max.filter(lambda x: float(x[1][0])>=25 and float(x[1][0])<=30 and float(x[1][1])>=100 and float(x[1][1])<=200)
filtered_data.saveAsTextFile("BDA/output/max_temp_prec")
```

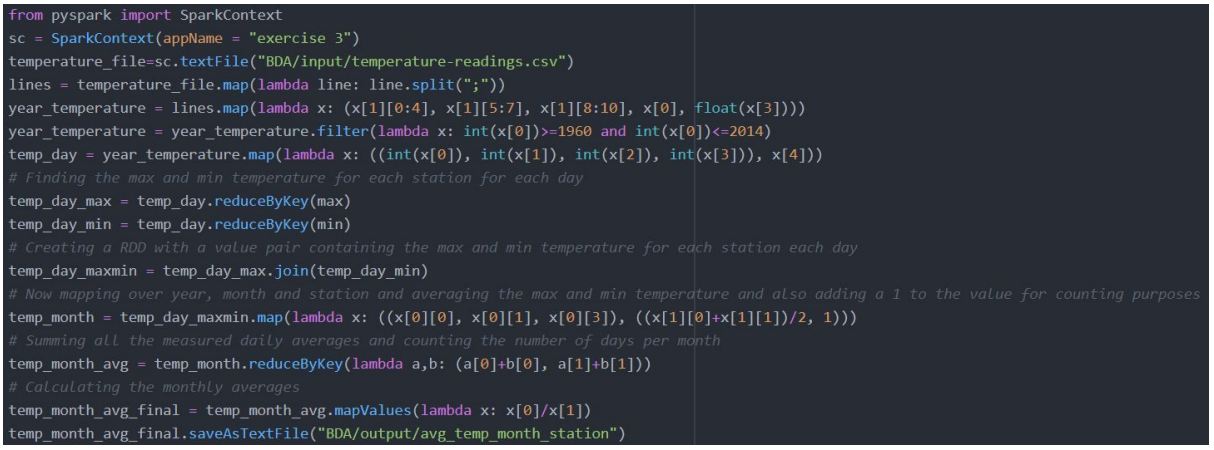
Text:

```
from pyspark import SparkContext
sc = SparkContext(appName = "exercise 4")
temperature_file=sc.textFile("BDA/input/temperature-readings.csv")
precipitation_file=sc.textFile("BDA/input/precipitation-readings.csv")
lines_temperature = temperature_file.map(lambda line: line.split(";"))
lines_precipitation = precipitation_file.map(lambda line: line.split(";"))
temperature = lines_temperature.map(lambda x: (x[0], float(x[3])))
precipitation = lines_precipitation.map(lambda x: (x[0], float(x[3])))
# Finding the max temp and max precipitation for each station
temp_max = temperature.reduceByKey(max)
prec_max = precipitation.reduceByKey(max)
# Creating a RDD with a value pair containing the max temperature and max precipitation for each station
joined_max = temp_max.join(prec_max)
# FIltering all instances where the temperature and precipitation is within a specified range
filtered_data = joined_max.filter(lambda x: float(x[1][0])>=25 and float(x[1][0])<=30 and float(x[1][1])>=100 and float(x[1][1])<=200)
filtered_data.saveAsTextFile("BDA/output/max_temp_prec")
```

wilan057
chrvu878
2020-05-06

Exercise 5

Picture:



```
from pyspark import SparkContext
sc = SparkContext(appName = "exercise 3")
temperature_file=sc.textFile("BDA/input/temperature-readings.csv")
lines = temperature_file.map(lambda line: line.split(";"))
year_temperature = lines.map(lambda x: (x[1][0:4], x[1][5:7], x[1][8:10], x[0], float(x[3])))
year_temperature = year_temperature.filter(lambda x: int(x[0])>=1960 and int(x[0])<=2014)
temp_day = year_temperature.map(lambda x: ((int(x[0]), int(x[1]), int(x[2]), int(x[3])), x[4]))
# Finding the max and min temperature for each station for each day
temp_day_max = temp_day.reduceByKey(max)
temp_day_min = temp_day.reduceByKey(min)
# Creating a RDD with a value pair containing the max and min temperature for each station each day
temp_day_maxmin = temp_day_max.join(temp_day_min)
# Now mapping over year, month and station and averaging the max and min temperature and also adding a 1 to the value for counting purposes
temp_month = temp_day_maxmin.map(lambda x: ((x[0][0], x[0][1], x[0][3]), ((x[1][0]+x[1][1])/2, 1)))
# Summing all the measured daily averages and counting the number of days per month
temp_month_avg = temp_month.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1]))
# Calculating the monthly averages
temp_month_avg_final = temp_month_avg.mapValues(lambda x: x[0]/x[1])
temp_month_avg_final.saveAsTextFile("BDA/output/avg_temp_month_station")
```

Text:

```
from pyspark import SparkContext
sc = SparkContext(appName = "exercise 3")
temperature_file=sc.textFile("BDA/input/temperature-readings.csv")
lines = temperature_file.map(lambda line: line.split(";"))
year_temperature = lines.map(lambda x: (x[1][0:4], x[1][5:7], x[1][8:10], x[0], float(x[3])))
year_temperature = year_temperature.filter(lambda x: int(x[0])>=1960 and int(x[0])<=2014)
temp_day = year_temperature.map(lambda x: ((int(x[0]), int(x[1]), int(x[2]), int(x[3])), x[4]))
# Finding the max and min temperature for each station for each day
temp_day_max = temp_day.reduceByKey(max)
temp_day_min = temp_day.reduceByKey(min)
# Creating a RDD with a value pair containing the max and min temperature for each station each day
temp_day_maxmin = temp_day_max.join(temp_day_min)
# Now mapping over year, month and station and averaging the max and min temperature and also adding a 1 to the value for counting purposes
temp_month = temp_day_maxmin.map(lambda x: ((x[0][0], x[0][1], x[0][3]), ((x[1][0]+x[1][1])/2, 1)))
# Summing all the measured daily averages and counting the number of days per month
temp_month_avg = temp_month.reduceByKey(lambda a,b: (a[0]+b[0], a[1]+b[1]))
# Calculating the monthly averages
temp_month_avg_final = temp_month_avg.mapValues(lambda x: x[0]/x[1])
temp_month_avg_final.saveAsTextFile("BDA/output/avg_temp_month_station")
```