CS2510: Modern Programming Languages
Assignment 3
Hand-In Deadline: 12 noon Friday 29 March 2019

It is essential that you provide the following details to mark your assignment. Failure to do so will result in assignment not being marked.

Student Name: Petar Petrov
Student ID: 51770258
Email Address: p.petrov.17@aberdeen.ac.uk

Java/Python

Question 1

```
class Record:
    def __init__(self, region, product, year, unit_price, units_sold,
sales):
        self.region = region
        self.product = product
        self.year = year
        self.unit_price = unit_price
        self.units_sold = units_sold
        self.sales = sales

    def __repr__(self):
        return "Region: {0}, Product: {1}, Year: {2}, UnitPrice: {3},
UnitsSold: {4}, Sales: {5}".format(self.region,

self.product,

self.year,

self.unit_price,

self.units_sold,

self.sales)

class RecordTable:

    def __init__(self, records=[]):
        """Accepts a list of records with which a new table will be
created, alternatively an
        empty table can be created by passing nothing and records can be
added in with the other methods"""
```

```
        self.records = []
        self.records.extend(records)

    def add_many_records(self, records):
        """Accepts a list of records to be added to the table"""
        self.records.extend(records)

    def add_one_record(self, record):
        """Accepts just a single record to be added to the table"""
        self.records.append(record)

    def clear_records(self):
        self.records.clear()

    def __repr__(self):
        temp = '\n'.join([repr(x) for x in self.records])
        return temp
```

Question 2

```
import copy

class Report:

    def __init__(self, table):
        self.table = table

    def report_years(self):
        distinct_years = set([x.year for x in self.table.records])
        final_report = {}
        for i in distinct_years:
            final_report[i] = [int(x.sales) for x in self.table.records if
x.year == i]
            final_report[i] = sum(final_report[i])
        return final_report

    def report_rp(self, variant):
```

```
        """Reports either region by year or product by year. For variant
== true, reports by region,
        variant == false, reports by product"""
        if variant == True:
            distincts = set([x.region for x in self.table.records])
        else:
            distincts = set([x.product for x in self.table.records])
        temp_storage = copy.deepcopy(self.table)    # saves a separate
copy instead of a reference
        reports = {}
        for i in distincts:
            self.table.clear_records()
            if variant == True:
                self.table.add_many_records([x for x in
temp_storage.records if x.region == i])
            else:
                self.table.add_many_records([x for x in
temp_storage.records if x.product == i])
            reports[i] = self.report_years()
        self.table = temp_storage
        return reports
```

Question 3

I have hardcoded the datasets so the file is a bit long and I won't paste the whole of it here. The relevant part:

```
print("Year Sum of Sales\n", report.report_years())
print("Region Year Sum of Sales\n", report.report_rp(True))
print("Product Year Sum of Sales\n", report.report_rp(False))

# print("Year Sum of Sales\n", report2.report_years())    # For the second
dataset
# print("Region Year Sum of Sales\n", report2.report_rp(True))    # For
the second dataset
# print("Product Year Sum of Sales\n", report2.report_rp(False)) # For the
second dataset
```

The first three sets of prints are on the first data set, uncomment the second set to print out the results from the second data set.

Haskell

Question 1

```
module SalesRecord where
```

```
type Region = [Char]
type Product = [Char]
type Year = Integer
type UnitPrice = Integer
type UnitsSold = Integer
type Sales = Integer

type Record = (Region, Product, Year, UnitPrice, UnitsSold, Sales)
```

```
module SalesData where
import SalesRecord

type Table = [Record]
```

Question 2

```
module SalesReport where
import SalesRecord
import SalesData
import Control.Monad


distinctYears :: Table -> [Year]
distinctYears [] = []
distinctYears ((_,_,year,_,_,_):records) =
    if year `elem` distinctYears records
        then distinctYears records
        else [year] ++ (distinctYears records)

distinctRegions :: Table -> [Region]
distinctRegions [] = []
distinctRegions ((region, _,_,_,_,_):records) =
    if region `elem` distinctRegions records
        then distinctRegions records
        else [region] ++ (distinctRegions records)

distinctProducts :: Table -> [Product]
distinctProducts [] = []
distinctProducts((_,product,_,_,_,_):records) =
    if product `elem` distinctProducts records
        then distinctProducts records
        else [product] ++ (distinctProducts records)

individualSalesForYear :: Table -> Year -> [Sales]
individualSalesForYear [] _ = []
individualSalesForYear ((_,_,recordYear,_,_,sales):records) year =
```

```
    if recordYear == year
        then [sales] ++ (individualSalesForYear records year)
        else individualSalesForYear records year

totalSalesForYear :: Year -> [Sales] -> (Year, Sales)
totalSalesForYear year salesList = (year, foldr (+) 0 salesList)


report1 :: Table -> [Year] -> [(Year, Sales)]
report1 [] [] = []
report1 table (year:years) =
    if null years
        then [totalSalesForYear year (individualSalesForYear table year)]
        else [totalSalesForYear year (individualSalesForYear table year)]
++ (report1 table years)


filterByRegion :: Table -> Region -> Table
filterByRegion [] _ = []
filterByRegion ((recordRegion,x,y,z,a,b):records) region =
    if recordRegion == region
        then [(recordRegion,x,y,z,a,b)] ++ (filterByRegion records region)
        else filterByRegion records region

filterByProduct :: Table -> Product -> Table
filterByProduct [] _ = []
filterByProduct ((x,recordProduct,y,z,a,b):records) product =
    if recordProduct == product
        then [(x,recordProduct,y,z,a,b)] ++ (filterByProduct records
product)
        else filterByProduct records product



report2 :: Table -> [Year] -> [Region] -> [(Region, [(Year, Sales)])]
report2 [] [] [] = []
report2 table years (region:regions) =
    if null regions
        then [(region, (report1 (filterByRegion table region) years))]
        else [(region, (report1 (filterByRegion table region) years))] ++
(report2 table years regions)

report3 :: Table -> [Year] -> [Product] -> [(Product, [(Year, Sales)])]
report3 [] [] [] = []
report3 table years (product:products) =
```

```
    if null products
        then [(product, (report1 (filterByProduct table product) years))]
        else [(product, (report1 (filterByProduct table product) years))]
++ (report3 table years products)
```

Question 3

Same situation here, pasting the whole file is unnecessary. Relevant bits:

```
records = table 1
years = distinctYears records
regions = distinctRegions records
products = distinctProducts records

firstReport = report1 records years
secondReport = report2 records years regions
thirdReport = report3 records years products


records1= table 2
years1 = distinctYears records1
regions1 = distinctRegions records1
products1 = distinctProducts records1


firstReport1 = report1 records1 years1
secondReport1 = report2 records1 years1 regions1
thirdReport1 = report3 records1 years1 products1
printer xs = forM_ xs print
```

As I assume you will run the file with an interpreter, calling firstReport, secondReport and thirdReport will return the relevant output for the first data set. Append a 1 on the end of them (e.g. call firstReport1) for the second data set. The printer function will print each element of the output list on a new line instead of the default everything on one line. Sample use:
printer firstReport

Question 4

Okay, first off: readability. Python is objectively superior here. I admit that me being new to Haskell and purely functional programming languages in general, negatively affects my ability to read it. However one of Python's core design philosophies per 'The Zen of Python' is 'explicit is better than implicit'. The language is designed to be as readable by humans as possible and it is hailed as one of the easiest languages to pick up. Personally I didn't really care about those aspects of the language since my first attempts in programming were in C and C++, however I came to really appreciate the simplicity of Python after Haskell. I felt like I was decoding some Elvish runes that have come out of Lord of The Rings. I bet that if I were to revisit this same code that I'm submitting right now next year, I would need at least a few hours and a lot of Googling to understand

what is happening in my Haskell files while it'd take me at most 30 minutes for the ones in Python.

Writability:

This one is a bit more subjective as I have vastly more experience in Python and other object-oriented languages compared to Haskell. I believe the hardest thing while writing the Haskell code was trying get out of the imperative and object-oriented mindset and get into the functional. Things that are very simple in Python like iterating over the elements of a list, took me hours and literally kept me up at night to figure out in Haskell. At the end I had an 'epiphany', one might say, that made it much more easier designing and writing my Haskell programs – I just had to think *backwards*! What I mean is that while writing Python, things usually go like this(Note: *very* simplified): I have some input data and I want some output data, what do I need to do to the input to get the output? I'd take some input do some stuff on it, working my way towards my desired output. While what I did for Haskell is: Okay I need this output data, what input do I need to get it? Here I'd have some desired output and I'd work on functions that would create the input I needed for the desired output.

Example from my code: in the file salesReport.hs, the first function created while working on Task1 was actually report1 – the function that provides the final output. However in order for that function to work I needed to derive some filtered input from the database and so I created individualSalesForYear and totalSalesForYear, which produced the required input for report1.

After this 'breakthrough' of mine in just 2 hours I had done more work than I had for the previous day. I'd still say that Python is superior here, but not as much. I feel like that if you spend enough time with Haskell and functional programming, things would start to 'click' and it definitely has the potential to be a fun language to write in. (Note: For me personally Python is not a very high bar to set when it comes to 'fun to write in', actually I find writing in Python quite tedious, but Haskell started to get quite enjoyable as I finally got around to actually writing it and not figuring out how to do things)

Reliability:

Haskell gets this one. It was *relentless*. GHCi caught every single mistake in the code and for once a language has error messages that are actually helpful. Aside from that, the absence of variables as we usually know them in imperative languages might make the language harder to read and write in, but also can save so much headaches by not making you worry about scoping and wondering if a variable has an actual value or just a reference. I spent 15 minutes debugging a part of my Python program wondering why wouldn't it work when I remembered that Python would never make an actual copy of an object and would reference it unless you explicitly force it not to.