MS in Data Science Challenge Exam
Jimmy Ng
June 18, 2018

(1) Python code

```
################################
########## Modification ##########
################################

# the FIRST SYNTAX ERROR is missing colon in the first line
def find_average(x, y):
        "'find_average will return the average of both a and b, which are floats'"

# the FIRST LOGICAL ERROR is subtle - only "a" and "b" are included in the function body
# however, only "x" and "y" are used for calculation here
# when the function is evaluated, the function will look up and search for the "x" and "y" variables in the global environment
# if "x" and "y" are not yet defined, the function will throw in an error
# on the other hand, "a" and "b" are just useless placeholders since they never get evaluated
# therefore, we should replace "a" and "b" with "x" and "y"

# the SECOND LOGICAL ERROR is missing parentheses in the return
# the result should be first adding the two numbers before dividing it into 2
        return (x + y) / 2


x = float( input("Please enter a number: ") )
y = float( input("Please enter another number: ") )

# the SECOND SYNTAX ERROR is missing comma in the function when calling it
average = find_average(x, y)
print(average)
```

MS in Data Science Challenge Exam

Math

1) Probability: Given a standard deck of cards, you draw a single card. What is the probability of drawing a 6 or a diamond?

$$\frac{4}{52} + \frac{13}{52} - \frac{1}{52} = \frac{16}{52} = \frac{4}{13}$$

2) Linear Algebra: What is the determinant of the following matrix?

A = [[1,4,3],
[3,7,1],
[2,0,3]]

$$1\begin{bmatrix} 7 & 1 \\ 0 & 3 \end{bmatrix} - 4\begin{bmatrix} 3 & 1 \\ 2 & 3 \end{bmatrix} + 3\begin{bmatrix} 3 & 7 \\ 2 & 0 \end{bmatrix}$$

$$(21-0) - 4(9-2) + 3(0-14)$$
$$21 - 4(7) + 3(-14)$$
$$21 - 28 - 42$$
$$= -49$$

3) Find a solution to the following linear equation using any method you feel most comfortable with and show all of your work.

4x - 4y + 5z = -34
6x - y = -6
-2x + 2y - 3z = 19

(See R code in separate sheet)

4) Calculus: Integrate x / (x^2 + 1) dx using substitution. Please show all of your work.

$$\int \frac{x}{x^2+1}\,dx$$

$$u = x^2 + 1$$
$$du = 2x\,dx$$
$$\frac{1}{2}\int \frac{du}{u}$$
$$\frac{1}{2}\ln|u| + c$$
$$\frac{1}{2}\ln|x^2+1| + c$$

(2) Math

3) Find a solution to the following linear equation using any method you feel most comfortable with and show all of your work.

```
if(!require(matlib)){install.packages("matlib"); require(matlib)}

A <- matrix(c(4, -4, 5, 6, -1, 0, -2, 2, -3),
        3,
        3,
        byrow = TRUE)

b <- c(-34, -6, 19)

matlib::showEqn(A, b)  # show matrices(A, b) as linear equations
# 4*x1 - 4*x2 + 5*x3  = -34
# 6*x1 - 1*x2 + 0*x3  =  -6
# -2*x1 + 2*x2 - 3*x3  =  19

solve(A, b)
# [1] -0.5  3.0 -4.0
# the answers of x, y and z are -0.5, 3.0 and -4.0 respectively
```

(3) SQL

1) There are several types of SQL statements, primarily **DDL** (Data Definition Language), **DML** (Data Manipulation Language), **TCL** (Transaction Control Language), and **DCL** (Data Control Language). Each of these has different commands for carrying out distinct functions.

For example, **DDL** is responsible for creating and restructuring database objects, such as creating or dropping tables, e.g.

```
--create a table
create table cbms.order_fact_staging (
        order_id string,
        datekey int,
        amount float
);

--drop a table
drop table cbms.order_fact_staging;
```

On the other hand, **DML** is used to manipulate data within objects. Some would consider DQL (Data Query Language) is part of DML. Thus, DML's commands would include "insert", "update", "delete", as well as "select", e.g.

```
--insert values in a table
insert into cbms.order_fact_staging (order_id, datekey, amount)
values ('e4x1031211229', 20180618, 150.5);

--query a table
Select *
From cbms.order_fact_staging
Where amount >100;
```

TCL is used to manage database transactions (as the name suggested), e.g.

```
--delete every order where amount is larger than 100
delete * from cbms.order_fact_staging
where amount >100;

--commit the previous deletion
commit;
```

Finally, **DCL** is used to control access to a database, e.g.

```
--grant user jng410 with permission to do anything with the table
grant ALL on cbms.order_fact_staging to jng410;
```

2a)

```
select *
from employee
where salary in (
        select max(salary)
        from employee
        where salary > (select max(salary) from employee)
) x
```

2b)

```
select department, avg(salary) as avg_salary
from employee
group by 1
```

3) SQL joins are used to combine rows from two or more tables based on a common field between them (such as primary key with foreign key), e.g.

Inner join: return all rows when there is at least one match in both tables
Left join: return all rows from the left table, and the matched rows from the right table
Right join: return all rows from the right table, and the matched rows from the left table
Full join: return all rows when there is a match in one of the tables
Cross join: produce a Cartesian product (all possible rows combinations) between two tables, no need to use on clause

For example,

```
--customer_fact left join with customer_blacklist in order to flag who is currently in the blacklist, i.e. whoever is flagged 1 is a "blacklisted" customer

select cf.customer_id
, case when b.customer_id is null then 0 else 1 end as flag
from customer_fact cf
left join customer_blacklist b on cf.customer_id = b.customer_id
```

(4) R

1a)

```
# create a data frame for student data, and then append a row at the end
Name <- c("Adam", "Peter", "Julia", "Ron")
Courses <- c("Math, Physics, Chemistry", "English, History, Sociology", "Physics, Botany, Chemistry", "Chemistry, Physics, Biology")

df <- data.frame(Name, Courses)
df2 <- data.frame(Name = "Stephanie", Courses = "Math, Geography, Chemistry")

# row bind the two data frames
df <- rbind(df, df2)
df
```

```
#      Name                Courses
# 1    Adam    Math, Physics, Chemistry
# 2   Peter English, History, Sociology
# 3   Julia  Physics, Botany, Chemistry
# 4     Ron Chemistry, Physics, Biology
# 5 Stephanie  Math, Geography, Chemistry
```

1b)

```
# add a new column to the df
df$Total_Score <- c(90, 65, 80, 75, 85)
df
```

```
#      Name                Courses        Total_Score
# 1    Adam    Math, Physics, Chemistry        90
# 2   Peter English, History, Sociology        65
# 3   Julia  Physics, Botany, Chemistry        80
# 4     Ron Chemistry, Physics, Biology        75
# 5 Stephanie  Math, Geography, Chemistry     85
```

2)

```
palindrome_check <- function(string) {
    if(!require(stringr)){install.packages("stringr"); require(stringr)}
    string <- stringr::str_to_lower(string)
    string <- stringr::str_trim(string)
    s <- stringr::str_split(string, "")[[1]]
    s.reverse <- s[length(s):1]
    all(s == s.reverse)
}

palindrome_check("Level ")
# [1] TRUE

palindrome_check("Jimmy")
# [1] FALSE
```

3)

```
word_check <- function(string1, string2){
    if(!require(stringr)){install.packages("stringr"); require(stringr)}
    string1 <- stringr::str_to_lower(string1)
    string1 <- stringr::str_trim(string1)
    string2 <- stringr::str_to_lower(string2)
    string2 <- stringr::str_trim(string2)
    string1 == string2
}

word_check("apple", "applE ")
# [1] TRUE

word_check("ORANGE", "applE")
# [1] FALSE
```