Project Report

IFT 458 - PD 4

Spring 2018

Ashley Mendoza

Vincent Li

Dr. Kuitche

March 16, 2018

Introduction

In this project deliverable part 4 we build on our project deliverable part 3 by adding data validation using regex and database access. First, we will begin by creating functions of validations needed and applying them to the proper user-inputs. Next, we wrote a SQL script to create the database and relevant tables on the Cygwin terminal.Then we wrote a function that connects to the database and another that inserts the data given in the User Registration Form, MDS Form or TestResults.csv to populate the tables. Once all the tables were populated, we added a function that would retrieve and display the data from the database in tabular format as directed in the description problem of pd4. Additionally, we have written our code to modularize it by making a main menu in the main function that allows a user to choose which action to perform and rewriting the actions, making all actions functions themselves. The following report describes the steps taken by providing screenshots of the input and required outputs.

Description of Work

First, we will begin by creating functions of validations needed and applying them to the proper user-inputs. Figure 1.1 demonstrates the functions written for user validation. These functions simply take the user input as a parameter and match it to the regular expression. If the input matched, the function returns the value, otherwise it returns false. Figure 1.2 and Figure 1.3 show how those functions were implemented into the forms to validate the inputs. Here, the user inputs a value, then calls the appropriate validation function. While the function returns false, the user is prompted to re-enter the value in the correct form. Once the function returns true, it will append the value to the datalist (the list of user input for the form that is later zipped with a list of keys to create a dictionary).

Figure 1.1: Validator Functions

```
35
36  ##################################### VALIDATORS ########################################
37  def validateEmail(enteredEmail):
38      validator = r"(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)"
39      if len(enteredEmail) > 7 and len(enteredEmail)<=40:
40          if re.match(validator, enteredEmail) is not None:
41              return enteredEmail
42          else:
43              print ("Invalid. Email format: user@example.com")
44              return False
45      else:
46          print ("Invalid. Email length must be longer than 7 characters and no longer than 40 characters.")
47          return False
48
49  def validatePhone(enteredPhone):
50      validator = r"^(\d{3}-\d{3}-\d{4})"
51      if re.match(validator, enteredPhone) is not None:
52          return enteredPhone
53      else:
54          print ("This is an invalid phone format: ###-###-####")
55          return False
56
57  def validateUserName(name):
58      validator = "^[A-Za-z0-9]+$"
59      if len(name) <= 40:
60          if re.match(validator, name) is not None:
61              return name
62          else:
63              print ("Username must contain only letters and numbers.")
64              return False
65      else:
66          print ("Username must be less than 40 characters.")
67          return False
68
69  def validatePassword(pword):
70      validator = "^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{4,20}$"
71      if re.match(validator, pword) is not None:
72          return pword
73      else:
74          print("Password must be: \n" + "- Min: 4 characters \n" + "- Max: 20 characters \n" + "- Include 1 upper case letter
75          return False
76
77  def validateDate(enteredDate):
78      validator="^(19|20)\d\d[- /.](0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])$"
79      if len(enteredDate) > 4:
80          if re.match(validator, enteredDate) is not None:
81              return enteredDate
82          else:
83              print("This is an invalid date.")
84              return False
85      else:
86          print("This is an invalid date.")
87          return False
```

```
89  def validateCompany(ctype):
90      if  ctype == "Test Lab" or ctype == "Manufacturer" :
91              return ctype
92
93      else:
94          print ("Invalid. Company Type must be either Test Lab or Manufacturer.")
95          return False
96
97  def validateAddress(addr):
98      validator = r"\d+\s[A-z]+\s[A-z]+"
99      if len(addr)<=40:
100         if re.match(validator, addr) is not None:
101             return addr
102         else:
103             print ("Invalid. Format: 1234 ExampleName Ave")
104             return False
105     else:
106         print ("Invalid. Must be no longer than 40 characters.")
107         return False
108
109 def validateString(value):
110     validator = "^[a-zA-Z]+$"
111     if len(value) <= 40:
112         if re.match(validator, value) is not None:
113             return value
114         else:
115             print ("Invalid. Must contain only letters.")
116             return False
117     else:
118         print ("Invalid. Cannot exceed 40 characters.")
119         return False
120
121 def validateFloat(value):
122     validator = r"[0-9]+\.+[0-9]+$"
123     if re.match(validator, value) is not None:
124         return value
125     else:
126         print("Invalid. Not float format.")
127         return False
128
129 def validateInt(value):
130     validator = "[0-9]+$"
131     if re.match(validator, value) is not None:
132         return value
133     else:
134         print("Invalid. Not an integer.")
135         return False
136
```

```
137  def validateDimension(value):
138      validator = "^[0-9]+x[0-9]+$"
139      if re.match(validator, value) is not None:
140          return value
141      else:
142          print("Invalid. Format: ##x## (No spaces)")
143          return False
144
145  def validateAlphaNum(value):
146      validator = r"^[a-zA-Z]+\s[0-9]+$"
147      if re.match(validator, value) is not None:
148          return value
149      else:
150          print("Invalid. Format: Name Number")
151          return False
152
153  def validateRating(value):
154      validator = r"^[0-9]+\/[0-9A-Z0-9]+$"
155      if re.match(validator, value) is not None:
156          return value
157      else:
158          print("Invalid. Example Format: 10/10SQ050 ")
159          return False
160
161  def validateSub(value):
162      validator = r"^[A-Z]+\/[0-9]+\.+[0-9]+[a-zA-Z]+$"
163      if re.match(validator, value) is not None:
164          return value
165      else:
166          print("Invalid. Example Format: TPT/0.35mm ")
167          return False
168
169  def validatebox(value):
170      validator = r"^[A-Z]+\-[A-Z0-9A-Z]+$"
171      if re.match(validator, value) is not None:
172          return value
173      else:
174          print("Invalid. Example Format: PV-RH050BV")
175          return False
176
```

Figure 1.2: Using validation functions - MDS form

```
#REGISTER PV MODULE MDS FORM : this function gets input and returns the data in a  dictionary
def addFV():
    #list of input fields
    keys = ['Manufacturer', 'Location', 'Contact','Address','Email', 'Phone', 'Model Number', 'Module lxw', 'Module Weight',
    #empty list which will hold user input
    datalist = []

    ClearMDS()
    man = raw_input("Manufacturer: ")
    while validateString(man) is False:
        man = raw_input("Re-enter Manufacturer: ")
    datalist.append(man)

    ClearMDS()
    loc = raw_input("Location: ")
    while validateString(loc) is False:
        loc = raw_input("Re-enter Location: ")
    datalist.append(loc)

    ClearMDS()
    cont = raw_input("Contact: ")
    while validateString(cont) is False:
        cont = raw_input("Re-enter Contact: ")
    datalist.append(cont)

    ClearMDS()
    addr = raw_input("Address: ")
    while validateAddress(addr) is False:
        addr = raw_input("Re-enter Address: ")
    datalist.append(addr)

    ClearMDS()
    email = raw_input("Email: ")
    while validateEmail(email) is False:
        email = raw_input("Re-enter Email: ")
    datalist.append(email)

    ClearMDS()
    phone = raw_input("Phone: ")
    while validatePhone(phone) is False:
        phone = raw_input("Re-enter phone: ")
    datalist.append(phone)

    ClearMDS()
    mnum = raw_input("Model Number: ")
    while validateInt(mnum) is False:
        mnum = raw_input("Re-enter Model Number: ")
    datalist.append(mnum)

```

```
244     ClearMDS()
245     mlxw = raw_input("Module total length x width (cmxcm): ")
246     while validateDimension(mlxw) is False:
247         mlxw = raw_input("Re-enter Module total length x width (cmxcm): ")
248     datalist.append(mlxw)
249
250     ClearMDS()
251     mwgt = raw_input("Module weight(kg): ")
252     while validateFloat(mwgt) is False:
253         mwgt = raw_input("Re-enter Module weight(kg): ")
254     datalist.append(mwgt)
255
256     ClearMDS()
257     icarea = raw_input("Individual Cell Area(cm^2): ")
258     while validateFloat(icarea) is False:
259         icarea = raw_input("Re-enter Individual Cell Area(cm^2): ")
260     datalist.append(icarea)
261
262     ClearMDS()
263     ctech = raw_input ("Cell Technology: ")
264     while validateString(ctech) is False:
265         ctech = raw_input ("Re-enter Cell Technology: ")
266     datalist.append(ctech)
267
268     ClearMDS()
269     cmanpt = raw_input("Cell Manufacturer and Part#: ")
270     while validateAlphaNum(cmanpt) is False:
271         cmanpt = raw_input("Re-enter Cell Manufacturer and Part#: ")
272     datalist.append(cmanpt)
273
274     ClearMDS()
275     cmanloc = raw_input("Cell Manufacturing Location: ")
276     while validateString(cmanloc) is False:
277         cmanloc = raw_input("Re-enter Cell Manufacturing Location: ")
278     datalist.append(cmanloc)
279
280     ClearMDS()
281     totcell = raw_input("Total number of cells: ")
282     while validateInt(totcell) is False:
283         totcell = raw_input("Re-enter Total number of cells: ")
284     datalist.append(totcell)
285
286     ClearMDS()
287     cseries = raw_input("Number of cells in series: ")
288     while validateInt(cseries) is False:
289         cseries = raw_input("Re-enter Number of cells in series: ")
290     datalist.append(cseries)
291
292     ClearMDS()
293     serstg = raw_input("Number of series strings: ")
294     while validateInt(serstg) is False:
295         serstg = raw_input("Re-enter Number of series strings: ")
296     datalist.append(serstg)
```

```
298     ClearMDS()
299     bydid = raw_input("Number of bypass diodes: ")
300     while validateInt(bydid) is False:
301         bydid = raw_input("Re-enter Number of bypass diodes: ")
302     datalist.append(bydid)
303
304     ClearMDS()
305     bdrateA = raw_input("Bypass diode rating(A): ")
306     while validateRating(bdrateA) is False:
307         bdrateA = raw_input("Re-enter Bypass diode rating(A): ")
308     datalist.append(bdrateA)
309
310     ClearMDS()
311     juntemp = raw_input("Bypass diode max junction temp(C): ")
312     while validateInt(juntemp) is False:
313         juntemp = raw_input("Re-enter Bypass diode max junction temp(C): ")
314     datalist.append(juntemp)
315
316     ClearMDS()
317     sfratingA = raw_input("Series Fuse Rating(A): ")
318     while validateInt(sfratingA) is False:
319         sfratingA = raw_input("Re-enter Series Fuse Rating(A): ")
320     datalist.append(sfratingA)
321
322     ClearMDS()
323     matsup = raw_input("Interconnect material and supplier model no.: ")
324     while validateAlphaNum(matsup) is False:
325         matsup = raw_input("Re-enter Interconnect material and supplier model no.: ")
326     datalist.append(matsup)
327
328     ClearMDS()
329     dimen = raw_input("Interconnect dimensions(mm x mm): ")
330     while validateDimension(dimen) is False:
331         dimen =  raw_input("Re-enter Interconnect dimensions(mm x mm): ")
332     datalist.append(dimen)
333
334     ClearMDS()
335     suptype = raw_input("Superstrate Type: ")
336     while validateString(suptype) is False:
337         suptype = raw_input("Re-enter Superstrate Type: ")
338     datalist.append(suptype)
339
340     ClearMDS()
341     supmanpt = raw_input("Superstrate Manfacturer and part#: ")
342     while validateAlphaNum(supmanpt) is False:
343         supmanpt = raw_input("Re-enter Superstrate Manfacturer and part#: ")
344     datalist.append(supmanpt)
345
346     ClearMDS()
347     subtype = raw_input("Substrate Type: ")
348     while validateSub(subtype) is False:
349         subtype = raw_input("Re-enter Substrate Type: ")
350     datalist.append(subtype)
```

```
352     ClearMDS()
353     submanpt = raw_input("Substrate Manufacturer and part#: ")
354     while validateAlphaNum(submanpt) is False:
355         submanpt = raw_input("Re-enter Substrate Manufacturer and part#: ")
356     datalist.append(submanpt)
357
358     ClearMDS()
359     frametype = raw_input("Frame Type and Material: ")
360     while validateString(frametype) is False:
361         frametype = raw_input("Re-enter Frame Type and Material: ")
362     datalist.append(frametype)
363
364     ClearMDS()
365     framead = raw_input("Frame adhesive: ")
366     while validateAlphaNum(framead) is False:
367         framead = raw_input("Re-enter Frame adhesive: ")
368     datalist.append(framead)
369
370     ClearMDS()
371     encaptype = raw_input("Encapsulant Type: ")
372     while validateSub(encaptype) is Flase:
373         encaptype = raw_input("Re-enter Encapsulant Type: ")
374     datalist.append(encaptype)
375
376     ClearMDS()
377     encapmanpt = raw_input("Encapsulant Manufacturer and part#: ")
378     while validateAlphaNum(encapmanpt) is False:
379         encapmanpt = raw_input("Encapsulant Manufacturer and part#: ")
380     datalist.append(encapmanpt)
381
382     ClearMDS()
383     junboxtype = raw_input("Junction box type: ")
384     while validatebox(junboxtype) is False:
385         junboxtype = raw_input("Re-enter Junction box type: ")
386     datalist.append(junboxtype)
387
388     ClearMDS()
389     junboxmanpt = raw_input("Junction box manufacturer and part#: ")
390     while validateAlphaNum(junboxmanpt) is False:
391         junboxmanpt = raw_input("Re-enter Junction box manufacturer and part#: ")
392     datalist.append(junboxmanpt)
393
394     ClearMDS()
395     junboxpot = raw_input("Junction box potting material, if any: ")
396     while validateString(junboxpot) is False:
397         junboxpot = raw_input("Re-enter Junction box potting material, if any: ")
398     datalist.append(junboxpot)
399
400     ClearMDS()
401     junboxadh = raw_input("Junction box adhesive: ")
402     while validateAlphaNum(junboxadh) is False:
403         junboxadh = raw_input("Re-enter Junction box adhesive: ")
404     datalist.append(junboxadh)
```

```
406     ClearMDS()
407     junboxuse = raw_input("Is junction box intended for use with Conduit?: ")
408     while validateString(junboxuse) is False:
409         junboxuse = raw_input("Re-enter - Is junction box intended for use with Conduit?: ")
410     datalist.append(junboxuse)
411
412     ClearMDS()
413     cabcontype = raw_input("Cable & Connector Type: ")
414     while validateAlphaNum(cabcontype) is False:
415         cabcontype = raw_input("Re-enter Cable & Connector Type: ")
416     datalist.append(cabcontype)
417
418     ClearMDS()
419     maxsysvol = raw_input("Max system voltage(V): ")
420     while validateInt(maxsysvol) is False:
421         maxsysvol = raw_input("Max system voltage(V): ")
422     datalist.append(maxsysvol)
423
424     ClearMDS()
425     voc = raw_input("Voc(V): ")
426     while validateFloat(voc) is False:
427         voc = raw_input("Re-enter Voc(V): ")
428     datalist.append(voc)
429
430     ClearMDS()
431     isc = raw_input("Isc(A): ")
432     while validateFloat(isc) is False:
433         isc = raw_input("Re-enter Isc(A): ")
434     datalist.append(isc)
435
436     ClearMDS()
437     vmp = raw_input("Vmp(V): ")
438     while validateFloat(vmp) is False:
439         vmp = raw_input("Re-enter Vmp(V): ")
440     datalist.append(vmp)
441
442     ClearMDS()
443     imp = raw_input("Imp(A): ")
444     while validateFloat(imp) is False:
445         imp = raw_input("Re-enter Imp(A): ")
446     datalist.append(imp)
447
448     ClearMDS()
449     pmp = raw_input("Pmp(W): ")
450     while validateInt(pmp) is False:
451         pmp = raw_input("Re-enter Pmp(W): ")
452     datalist.append(pmp)
453
454     ClearMDS()
455     ff = raw_input("FF(%): ")
456     while validateInt(ff) is False:
457         ff = raw_input("Re-enter FF(%): ")
458     datalist.append(ff)
459
460     os.system('clear')
461
462     #zip to combine both lists into a dictionary
463     return dict(zip(keys, datalist))
```

Figure 1.3: Using validation functions - User Registration form

```
#REGISTER USER: this function gets input and returns the data in a  dictionary
def addUser():
    #list of input fields
    keys = ['Username','Password','First Name', 'Middle Name', 'Last Name', 'Company Name', 'Company Type', 'Address', 'Offi
    #empty list which will hold user input
    datalist = []

    ClearUser()
    uname = raw_input("Username: ")
    while validateUserName(uname) is False:
        uname = raw_input("Re-enter Username: ")
    datalist.append(uname)

    ClearUser()
    pword = raw_input("Password: ")
    while validatePassword(pword) is False:
        pword = raw_input("Re-enter Password: ")
    datalist.append(pword)

    ClearUser()
    fname = raw_input("First Name: ")
    while validateString(fname) is False:
        fname = raw_input("Re-enter First Name: ")
    datalist.append(fname)

    ClearUser()
    mname = raw_input("Middle Name: ")
    while validateString(mname) is False:
        mname = raw_input("Re-enter Middle Name: ")
    datalist.append(mname)

    ClearUser()
    lname = raw_input("Last Name: ")
    while validateString(lname) is False:
        lname = raw_input("Re-enter Last Name: ")
    datalist.append(lname)

    ClearUser()
    cname = raw_input("Company Name: ")
    while validateString(cname) is False:
        cname = raw_input("Re-enter Company Name: ")
    datalist.append(cname)

    ClearUser()
    ctype = raw_input("Company Type(Test Lab or Manufacturer): ")
    while validateCompany(ctype) is False:
        ctype = raw_input("Re-enter Company Type(Test Lab or Manufacturer): ")
    datalist.append(ctype)
```

```
514     ClearUser()
515     addr = raw_input("Address: ")
516     while validateAddress(addr) is False:
517         addr = raw_input("Re-enter Address: ")
518     datalist.append(addr)
519
520     ClearUser()
521     ophone = raw_input("Office phone number: ")
522     while validatePhone(ophone) is False:
523         ophone = raw_input("Re-enter Office phone number: ")
524     datalist.append(ophone)
525
526     ClearUser()
527     cphone = raw_input("Cell phone number: ")
528     while validatePhone(cphone) is False:
529         cphone = raw_input("Re-enter Cell phone number: ")
530     datalist.append(cphone)
531
532     ClearUser()
533     email = raw_input("Email Address: ")
534     while validateEmail(email) is False:
535         email = raw_input("Re-enter Email Address: ")
536     datalist.append(email)
537
538     os.system('clear')
539
540     #zip to combine both lists into a dictionary
541     return dict(zip(keys, datalist))
```

Next, we wrote a SQL script to create the database and relevant tables on the Cygwin terminal. Figure 1.4 shows the SQL script used to create the database. Figure 1.5 shows the successful output of the script.

Figure 1.4: SQL script

```
DROP database mw;
Create database mw;
use mw;

CREATE TABLE IF NOT EXISTS Manufacturer(
    Manu_name varchar(40),
    registered_country varchar(40),
    PRIMARY KEY (Manu_name));

CREATE TABLE IF NOT EXISTS User(
    username varchar(40),
    password varchar(40),
    fname varchar(40),
    mname varchar (40),
    lname varchar (40),
    address varchar (40),
    officePhone varchar (12),
    cellphone varchar (12),
    email varchar (40));

CREATE TABLE IF NOT EXISTS testlab(
    Lab_name varchar(40) NOT NULL,
    address varchar(40),
    PRIMARY KEY (Lab_name));

CREATE TABLE IF NOT EXISTS test_results(
    Lab_name varchar(40),
    Test_date date,
    reportingCondition varchar(40),
    NOCT float,
    ISC float,
    VOC float,
    PMP float,
    FF float,
    VMP float,
    IMP float,
    FOREIGN KEY (Lab_name) REFERENCES testlab(Lab_name)
    );
```

```
CREATE TABLE IF NOT EXISTS Product(
    ManufacturedDate date,
    modelNumber varchar(40) NOT NULL,
    Length float,
    Width float,
    Weight float,
    Cell_Area float,
    Cell_Technology varchar(40),
    Total_num_cell int,
    Num_of_cell_series int,
    Num_of_series int,
    Num_of_diodes int,
    Series_fuse_rating float,
    Interconnect_material varchar(40),
    Interconnect_supplier varchar(40),
    Superstrate_type varchar(40),
    Superstrate_manu varchar(40),
    Substrate_type varchar(40),
    Substrate_manu varchar(40),
    Frame_material varchar(40),
    Frame_adhesive varchar(40),
    Encapulant_type varchar(40),
    Encapsulant_manu varchar(40),
    Junction_box_type varchar(40),
    Junction_box_manu varchar(40),
    Junction_box_adhesive varchar(40),
    Cable_type varchar(40),
    Connector_type varchar(40),
    Max_system_voltage float,
    ISC float,
    VOC float,
    IMP float,
    VMP float,
    FF float,
    PMP float,
    PRIMARY KEY(modelNumber)
);
```

Figure 1.5: Successful output of SQL Script.

```
MariaDB [(none)]> source ~/createDatabase.sql
Query OK, 5 rows affected (0.14 sec)

Query OK, 1 row affected (0.00 sec)

Database changed
Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.04 sec)

Query OK, 0 rows affected (0.04 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.03 sec)

MariaDB [mw]>
```

Then we wrote a function within a class that connects to the database and another that inserts the data given in the User Registration Form, MDS Form or TestResults.csv to populate

the tables. Once all the tables were populated, we added a function that would retrieve and display the data from the database in tabular format as directed in pd4. FIgure 1.6 shows the class we created. Figure 1.7 verifies the functionality of the code by showing the open connection in the python interactive interface. Figure 1.8 shows the code used to populate the database and Figure 1.9 show the database for proof of concept.

Figure 1.6: Connection Code

```
3   class DBModule(object):
4
5       '''
6       def __init__(self):
7           pass
8       '''
9
10      def create_db(self):
11          pass
12
13      def connect_db(self, user='root', db='mw'):
14          con = MySQLdb.connect(user=user, db=db, passwd="")
15          return con
```

```
586     #create the connection to the database
587     mydb = myClasses.DBModule()
588     con = mydb.connect_db()
589     cur = con.cursor()
590
```

Figure 1.7: Successful Open Connection

```
>>> MySQLdb.connect(user='root', passwd='', db='mw')
<_mysql.connection open to 'localhost' at 600422890>
>>>
```

Figure 1.8: Populating the Database - Insert data

```
    #create user object
    user1 = myClasses.User(uname, pword, fname, mname, lname, addr, ophone, cphone, email)


    ###create object user dictionary##
    var1 = {'username':user1.getUsername(), 'password':user1.getPassword(), 'fname':user1.getFirstName(), 'mname':user1.getM


    emptylist = []
    emptylist.append(var1)

####################populate the database.####################

    # Now we can create the connection to the database
    mydb = myClasses.DBModule()
    con = mydb.connect_db("root", "mw")
    cur = con.cursor()

    # Connection established and cursor object created

    for i in range(len(emptylist)):
        mydb.insert_data("Users", cur, emptylist[i])

    con.commit()
    cur.close()
    con.commit()
    con.close()

    return user1
```

```
801   # function will populate Database
802   def Populate (MDS, UREG):
803
804       #instantiate the contact person using the User Class
805       uname = UREG.get('Username')
806       pword = UREG.get('Password')
807       fname = UREG.get('First Name')
808       mname = UREG.get('Middle Name')
809       lname = UREG.get('Last Name')
810       addr = UREG.get('Address')
811       ophone = UREG.get('Office Phone Number')
812       cphone = UREG.get('Cell Phone Number')
813       email = UREG.get('Email Address')
814
815       #create user object
816       user1 = myClasses.User(uname, pword, fname, mname, lname, addr, ophone, cphone, email)
817
818
819       ###create object user dictionary##
820       user_row ={}
821       user_row = {'username':user1.getUsername(), 'password':user1.getPassword(), 'fname':user1.getFirstName(), 'mname':user1.getMiddleName(), 'lna
822
823       userlist = []
824       userlist.append(user_row)
825
826
827       # Now we can create the connection to the database
828       mydb = myClasses.DBModule()
829       con = mydb.connect_db("root", "mw")
830       cur = con.cursor()
831
832       ####################populate the user database.####################
833       for i in range(len(userlist)):
834           mydb.insert_data("Users", cur, userlist[i])
835           con.commit()
836
837       #initialize empty list
838       man_row = {}
839       manlist =[]
840       test_row = {}
841       testlist = []
842
843       # check if user entered company type Manufacturer
844       if UREG.get('Company Type') == "Manufacturer":
845
846           #instantiate the manufacturer contact person
847           mname = MDS.get('Manufacturer')
848           country = MDS.get('Location')
849
850           #create manufacturer object
851           manufacturer1 = myClasses.manufacturer(mname, country, User)
852
853           ##create object Manufacturer dictionoary
854           man_row = {'Manu_name':manufacturer1.getName(), 'registered_country':manufacturer1.getCountry()}
855           manlist.append(man_row)
856
857           ##populate Manufacturer table
858           for i in range(len(manlist)):
859               mydb.insert_data("Manufacturer", cur, manlist[i])
860           con.commit()
861
862       #check if user entered company type test Lab
863       elif UREG.get('Company Type') == "Test Lab":
864
```

```
        #instantiate the test lab
        labname = UREG.get('Company Name')
        address = UREG.get('Address')

        #create test lab object
        lab = myClasses.TestLab(labname, address, User)

        ##create object Test Lab dictionary
        test_row = {'Lab_name':lab.getName(), 'address':lab.address()}
        testlist.append(test_row)

        #populate TestLab table
        for i in range(len(testlist)):
            mydb.insert_data("testlab", cur, testlist[i])
        con.commit()


    #######Populate Prodduct table.
    #instantiate the product
    mnum = MDS.get('Model Number')
    mname = UREG.get('First Name')
    mdate = 'Date'
    length = MDS.get('Module lxw')
    wdh = MDS.get('Module lxw')
    wgt = MDS.get('Module Weight')
    cellarea = MDS.get('Individual Cell Area')
    celltec = MDS.get('Cell Technology')
    numcell = MDS.get('Total number of cells')
    numcellseries = MDS.get('Number of cells in a series')
    numstring = MDS.get('Number of series strings')
    numbypass= MDS.get('Number of bypass diodes')
    fuserating = MDS.get('Series fuse rating')
    intermat = MDS.get('Innterconnect material')
    intersup = MDS.get('Cell Manufacturer')
    suptype= MDS.get('Superstrate Type')
    supman = MDS.get('Superstrate Manufacturer')
    subtype = MDS.get('Substrate Type')
    subman = MDS.get('Substrate Manufacturer')
    framemat = MDS.get('Frame Type')
    frameadh = MDS.get('Frame adhesive')
    entype = MDS.get('Encapsulant Type')
    enman = MDS.get('Encapsulant Manufacturer')
    jbtype = MDS.get('Junction Box Type')
    jbman = MDS.get('Junction box manufacturer')
    jbad = MDS.get('Junction box adhesive')
    cabtype = MDS.get('Cable & Connector type')
    contype = MDS.get('Cable & Connector type')
    maxsys = MDS.get('Maximum system voltage')
    rvoc = MDS.get('voc')
    risc = MDS.get('isc')
    rvmp = MDS.get('vmp')
    rimp = MDS.get('imp')
    rpmp = MDS.get('pmp')
    rff = MDS.get('ff')

    #create product object
    product1 = myClasses.Product(mnum, mname, mdate, length, wdh, wgt, cellarea, celltec, numcell, numcellseries, numstring, numbypass, fuseratin

    ###create object product dictionary##
    product_row = {'ManufacturedDate':product1.getManufacturingDate(), 'modelNumber':product1.getModelNumber(), 'Length':product1.getLength(), 'W

    productlist = []
    productlist.append(product_row)
```

```
    ####################populate the Product table ####################
    for i in range(len(productlist)):
        mydb.insert_data("Product", cur, productlist[i])


    ## Populate the test_results table
    resultlist=[]
    result_row={}
    for i in dict_list:
        test = myClasses.TestResults(i)
        result_row = {'Lab_name':test.getDataSource(), 'Test_date':test.getTestDate(), 'reportingCondition':test.getReportingConditon(), 'NOCT':t
        resultlist.append(result_row)

    # Test Results table
    for i in range(len(resultlist)):
        mydb.insert_data("test_results", cur, resultlist[i])

    #close connectionn
    con.commit()
    cur.close()
    con.commit()
    con.close()

    os.system('clear')
```

Figure 1.9: 'MW' Database - 'Users' Table Output (3 users are registered)

```
MariaDB [mw]> SELECT * FROM Users;
+----------+-------------+--------+-------+---------+----------------+----------
-------+--------------+------------------+
| username | password    | fname  | mname | lname   | address        | officePho
ne     | cellphone    | email            |
+----------+-------------+--------+-------+---------+----------------+----------
-------+--------------+------------------+
| asfd     | dsafAM123   | asdf   | asfd  | asdf    | 123 df sdf     | <bound me
thod U | 123-123-1231 | afsd@asf.ed      |
| fasd     | asdfAM324   | adsf   | asdf  | adsf    | 123 adas das   | 123-123-1
321    | 123-123-1233 | adsf@asf.edu     |
| agmendo4 | PASSword123 | Ashley | G     | Mendoza | 1234 Street st | 123-123-1
234    | 123-123-1234 | agmendo4@asu.edu |
+----------+-------------+--------+-------+---------+----------------+----------
-------+--------------+------------------+
3 rows in set (0.00 sec)

MariaDB [mw]>
```

Once all the tables were populated, we added a function that would retrieve and display the data from the database in tabular format as directed in the description problem of pd4. The code to accomplish this is shown in Figure 1.10.

Figure 1.10: Code to retrieve data and print in tabular format

```
673  def query(qry):
674      mydb = myClasses.DBModule()
675      con = mydb.connect_db("root", "mw")
676      cur = con.cursor()
677
678      query_str = qry
679      cur.execute(query_str)
680
681      # fetch all of the rows from the query
682      data = cur.fetchall()
683
684      # close the cursor object
685      cur.close()
686      # close the connection
687      con.close()
688
689      for row in data :
690          return row[0]
691
```

```
692  #function will print product information
693  def productInformation():
694      #print "----------Product Information-----------"
695      #print ""
696      #print "Manufacturer Name: " + str(pvl.getManufacturer())
697      #print "Contact Name: " + str(ul.getFirstName())
698      #print "Contact Email: " + str(ul.getEmail())
699      #print "Model Number: " + str(pvl.getModelNumber())
700      #print "Cell Technology: " + str(pvl.getCellTechnology())
701      #print "System Voltage: " +  str(pvl.getmaxsysvoltage())
702      #print "Rated Power (PMP): " + str(pvl.getratedpmp())
703
704
705      qry = 'SELECT Manufacturer from Product'
706      manufacName = query(qry)
707
708      qry = 'SELECT contactPerson from Manufacturer where Manu_name="'+manufacName+'"'
709      contactperson = query(qry)
710
711      qry = 'SELECT email from Users where username="'+contactperson+'"'
712      contactemail = query(qry)
713
714      qry = 'SELECT Cell_Technology from Product'
715      celltech = query(qry)
716
717      qry = 'SELECT PMP from Product'
718      rpmp = query(qry)
719
720      qry = 'SELECT AVG(ISC) from TestResults where testSequence="Baseline"'
721      baseline_isc = query(qry)
722
723      qry = 'SELECT AVG(VOC) from TestResults where testSequence="Baseline"'
724      baseline_voc = query(qry)
725
726      qry = 'SELECT AVG(PMP) from TestResults where testSequence="Baseline"'
727      baseline_pmax = query(qry)
728
729      qry = 'SELECT AVG(ISC) from TestResults where testSequence="TC200"'
730      tc_isc = query(qry)
731
732      qry = 'SELECT AVG(VOC) from TestResults where testSequence="TC200"'
733      tc_voc = query(qry)
734
735      qry = 'SELECT AVG(PMP) from TestResults where testSequence="TC200"'
736      tc_pmax = query(qry)
737
738      qry = 'SELECT AVG(ISC) from TestResults where testSequence="Damp Heat"'
739      dh_isc = query(qry)
740
741      qry = 'SELECT AVG(VOC) from TestResults where testSequence="Damp Heat"'
742      dh_voc = query(qry)
743
744      qry = 'SELECT AVG(PMP) from TestResults where testSequence="Damp Heat"'
745      dh_pmax = query(qry)
746
747      qry = 'SELECT AVG(ISC) from TestResults where testSequence="HF10"'
748      hf_isc = query(qry)
749
750      qry = 'SELECT AVG(VOC) from TestResults where testSequence="HF10"'
751      hf_voc = query(qry)
752
753      qry = 'SELECT AVG(PMP) from TestResults where testSequence="HF10"'
754      hf_pmax = query(qry)
755
```

```
756     tcdrop = str(round(tc_pmax / baseline_pmax * 100 - 100,4))+"%"
757     dhdrop = str(round(dh_pmax / baseline_pmax * 100 - 100,4))+"%"
758     hfdrop = str(round(hf_pmax / baseline_pmax * 100 - 100,4))+"%"
759     os.system('clear')
760
761     print '\nManufacturer: \t\t'+manufacName
762     print 'Contact name: \t\t'+contactperson
763     print 'Contact Email: \t\t'+contactemail
764     print 'Cell Technology: \t'+celltech
765     print 'Rated Power: \t\t'+str(rpmp)
766     print '\n           Baseline    TC200       Damp Heat   HF10\n'
767     print 'Average Isc:   \t\t'+str(round(baseline_isc,4))+'\t\t'+str(round(tc_isc,4))+'\t\t'+str(round(dh_isc,4))+'\t\t'+str(round(hf_isc,4))
768     print 'Average Voc:   \t\t'+str(round(baseline_voc,4))+'\t\t'+str(round(tc_voc,4))+'\t\t'+str(round(dh_voc,4))+'\t\t'+str(round(hf_voc,4))
769     print 'Average Pmax:  \t\t'+str(round(baseline_pmax,4))+'\t'+str(round(tc_pmax,4))+'\t\t'+str(round(dh_pmax,4))+'\t\t'+str(round(hf_pmax,4))
770     print '\nAverage Pmax'
771     print 'drop from \t\t0%'+'\t\t'+tcdrop+'\t'+dhdrop+'\t'+hfdrop
772     print 'Baseline avg:'
773
774     raw_input("\nPress Enter to return to menu.")
775     os.system('clear')
776
```

User Manual

To execute these scripts, ensure that the .zip file provided is downloaded and all files are kept in the same folder. This folder should be added to the Cygwin/Home/User folder so that they can be accessible through the Cygwin terminal. First run the SQL file to create the database by typing `source ~/createDatabase.sql`. Now that the database is created, the user can successfully connect to it and populate it. To run the main file, type `python pd4.py`. This will bring the user to the main menu shown in Figure 1.11. Select an option 1-4 or 5 to quit.

Figure 1.11: PD4 Main Menu

```
----------------------------MAIN MENU----------------------------

Select an option below:

1. New User Registration
2. MDS Form
3. Get Registered Product Information
4. Test Result Data
5. Quit

-----------------------------------------------------------------
Select an option 1 - 4 to continue or 5 to quit:
```

Conclusion

In conclusion, completing this portion of the project allowed us to review the concepts learned in IFT 394 (IFT383) and served mostly as a refresher for creating regular expressions and connecting mySQL through python object oriented. We did, however, master developing regexs and learned how to connect to different servers, multiple ways to insert data to populate the database and ultimately how to retrieve the data. Additionally, we learned how to populate the database using a dictionary. We also modularized our data, which helped us have a better

understanding of functions and their importance of reusability. The most challenging task we faced was properly populating the database. It was complicated dealing with various data types when extracted from the dictionary. We did overcome this hurdle by reviewing online python documentation. This code can be improved by allowing the user to fill multiple forms. This would be done by have a list of multiple dictionaries. Additionally, the code could verify the user exists before allowing the user to fill an MDS form. These features will be included in the future.