**Semester Programming Portfolio**

**Objective 1: Define the data required by a Python program to solve a problem.**

* **Description in Own Words:**

In my words, defining data required by a python program to solve a problem mainly involves figuring out whether you should use integers, floats, strings, Boolean values, etc… Along with that, this involves identifying what data in a problem is necessary for finding the solution of the problem. This could also entail figuring out what type of data is being used in the problem in terms of whether or not it is from a file, a database, a spreadsheet, or if its user input data.

* **Understanding Grade (0-100):**

I would give myself an understanding grade of 95, as I would say I understand this concept almost fully.

* **Completion Evidence:**

I have completed this objective on almost every problem I have solved during my semester project, and here is an example. In this example I made sure that Inputs were floats, and made sure certain inputs were strings so the problem would flow properly and I could calculate a sales target, and track it over a 5 day week.

##Problem 4.1

##Using repetition with the for statement, write a Python program that prompts

##the user for a sales target, and then tracks progress toward that target over

##the course of a 5-day week.

##Prompting the user for the total sales target and converting it to a float

##for later calculations

sales\_target = float(input("Enter total sales target: "))

##Initializeing a variable to keep track of cumulative sales, starting at zero

cumulative\_sales = 0.0

##creating a for loop that loops through each day of the week (1 through 5)

##in order to track daily sales

for day in range(1, 6):

##Asking the user to enter the sales for the current day and convert it to a float

##format replaces crly brakcets with the value of the day, so if the day is 1

##it will put in day 1

daily\_sales = float(input('Enter day' + str(day)+ 'sales: '))

##Adding the daily sales to the cumulative sales to keep a running total

cumulative\_sales += daily\_sales

##Calculating the percentage of the sales target so far

percentage\_of\_target = (cumulative\_sales / sales\_target) \* 100

##Printing the cumulative sales and the percentage of the target achieved,

##using format for rounding to 2 decimals

print("Cumulative sales: $", format(round(cumulative\_sales, 2), ",.2f"),

"(", format(round(percentage\_of\_target, 2), ",.2f"), "% )")

* **Barriers Encountered:**

Some barriers I encountered when defining the data required by a python program to solve a problem were not fully understanding what type of data certain problems were asking for, which would result in me not being able to complete the problems. Along with that I sometimes had a hard time figuring out whether I needed to use strings, floats, or integers for the problems I was solving.

**Objective 2: Manipulate data in a Python program using the standard library, control structures, and object-oriented programming.**

* **Description in Own Words:**

Manipulating data with the standard library involves filtering, sorting, and transforming data on python with the basic functions that python supply’s you with. This means you aren’t using any functions or tools from other external python packages. Using control structures includes things such as if statements, and for/while loops to help with decision making when manipulating data. Finally, object-oriented programming is the subject that I would say I am least familiar with out of all of these, however I know it involves using methods within code to help combine different attributes.

* **Understanding Grade (0-100):**

I would grade my understanding of manipulating data in a python program using the standard library, control structures, and object-oriented programming an 80, as I still have a lot of learning that could be done to understand it better.

* **Completion Evidence:**

Here is an example of a problem where I used the standard library, control structures, and object-oriented programming to find out what type of shipping a company should use on specific shipments, and how much they should charge their customers.

##problem 3.2

##A delivery company charges $10 per package for standard shipping, and $15

##per package for express shipping.Write a program that prompts for the number

##of packages to ship as well as whether to ship these via regular or express,

##then calculates and displays the total delivery charge.

##creating a variable and defining price for standard shipping charge

standard\_shipping\_charge = 10

##creating a variable and defining price for express shipping charge

express\_shipping\_charge = 15

#Asking the user to input the number of packages to ship and storing the

##results and returns them as a number

number\_of\_packages = eval(input("Enter the number of packages to ship: "))

##Asking the user for the type of shipping (regular or express) and storing

##the result in variable shipping type

shipping\_type = input("Enter shipping type (regular/express): ")

##Creating a variable to hold the total delivery charge

total\_delivery\_charge = 0

##Checking if the shipping type is regular and calculating the charge accordingly

if shipping\_type.lower() == "regular":

total\_delivery\_charge = number\_of\_packages \* standard\_shipping\_charge

##using Else if statement if the shipping type is express

##it calculates the charge accordingly

elif shipping\_type.lower() == "express":

total\_delivery\_charge = number\_of\_packages \* express\_shipping\_charge

##Prints the total delivery charge

print("Total delivery charge: $", total\_delivery\_charge)

* **Barriers Encountered:**

Some barriers I encountered when manipulating data in a python program using the standard library, control structures, and object-oriented programming were that I found it difficult to learn and understand a lot of what is in the standard library as there are just so many things to learn. Along with that it was difficult for me to understand all of the concepts within object oriented programming and how that works.

**Objective 3: Store data in Python from files and databases**

* **Description in Own Words:**

Storing data in Python from files and databases involves learning how to interact with external files in Python, including data from text files, CSV files, databases, excel files, etc. This includes displaying the data from these files on your screen, and being able to add to, delete from, and filter data from these files and databases.

* **Understanding Grade (0-100):**

I would grade my understanding of storing data in python from files and databases a 95 as I understand it almost fully.

* **Completion Evidence:**

I have stored data in python from files and databases in a variety of different problems, here is an example of one. I used this to a display a company’s top employee and what their sales were.

##oppening the sales file, it opens by default to read as strings.

sales\_file = open("sales.txt")

##reading the file and turning it into a list of strings.

all\_records = sales\_file.readlines()

sales\_file.close()

##Initializing variables to find the employees with the highiest sales.

leading\_employee = ""

maximum\_sales = 0

##creating a for loop to process the data in the file.

for record in all\_records:

##seperating each line containing an emppoyees name and their sales by a space.

##splitting the line into two parts. Empoloyee name and sales.

parts\_of\_record = record.split()

employee\_name = parts\_of\_record[0]

##converting the sales into an integer.

employee\_sales = int(parts\_of\_record[1])

##Comparing this employee's sales to what the current maximum employee sales is

if employee\_sales > maximum\_sales:

##If this employee's has higher sales then the variables are updated.

maximum\_sales = employee\_sales

leading\_employee = employee\_name

##priting the employee who has the highest sales.

print("Employee with top sales:", leading\_employee, "with sales of", maximum\_sales)

* **Barriers Encountered:**

Some barriers I ran into when trying to store data in python from files and databases where different errors when trying to read in files due to me being in the wrong folder in spider, and troubles connecting with databases due to typos and connection errors.

**Objective 4: Create user-defined functions**

* **Description in Own Words:**

Creating User-defined functions in excel is when you design and create your own custom functions which you can implement into python. This involves defining functions, creating different arguments, and making set return values for the functions. This allows you to speed up your code writing for certain problems if the function you would like to use does not already exist.

* **Understanding Grade (0-100):**

I would grade my understanding of creating user defined functions a 98 as I understand it almost entirely.

* **Completion Evidence:**

Here is an example of a problem where I created my own user defined function in order to solve it.

##Write a function named calcCost, that takes two required input parameters,

##count and price. The function should also have one named parameter named

## discount. The signature of the function should be:

##def calcCost(count, price, discount=.1)

##The function should return:

##count \* price if the count is less than 5, or

##count \* price \* (1-discount) if the count is 5 or more

##Below the function, write test statements to determine and display the cost

##for 10 items priced at $15, using a discount of 20%.

##Defining the calcCost function with the parameters 'count' and 'price',

##also adding a parameter 'discount' with a default value of 0.1 (10%).

def calcCost(count, price, discount=0.1):

##Checking if the number of items (count) is less than 5.

if count < 5:

##If count is less than 5, no discount is applied and cost is calculated by

##multiplying 'count' and 'price' which is returned without applying a discount

return count \* price

else:

##If count is 5 or more, the discount is applied

##Calculating the discounted cost by first finding the total cost without

##discount by multiplying 'count' and 'price'. Then, apply the discount

##by multiplying with (1 - discount).

return count \* price \* (1 - discount)

##Testing the function to ensure it works as expected.

##using 10 as the count of items, $15 as the price per item, and a 20% discount.

cost = calcCost(count=10, price=15, discount=0.2)

##Printing the calculated cost.

##This displays the total cost after applying the discount for 10 items each

##priced at $15.

print("Cost = $", cost)

* **Barriers Encountered:**

Some barriers I encountered when creating user-defined functions were that I found it hard to create functions that were easily reusable, as I was not familiar with how to create functions. Along with that I had a hard time with understanding how to nest user defined functions within my problems.

**Objective 5: Use Python to aid in analyzing data.**

* **Description in Own Words:**

Using python to aid in analyzing data involves using a variety of different libraries in order to analyze data. This involves transforming data and completing basic statistical analysis on data. You can find different statistics on data sets such as the mean, median, mode, and other central measures of data. Along with that you are able to find things such as Variance, standard deviations, and how spread out or accurate data sets are. One of the main ways we used python to aid in analyzing data was though the use of pivot tables.

* **Understanding Grade (0-100):**

I would grade my understanding on using python to aid analyzing data a 90 because it is still kind of new to me, but I think that I have a solid understanding.

* **Completion Evidence:**

Here is an example of code we went over and class where we took data from a company over there sales and created a pivot table with the data in order to easily analyze it.

# Chapter 17, section 2

# \*\*\* requires datafile: sales.csv

import pandas as pd

sales = pd.read\_csv('sales.csv', parse\_dates=['time'])

sales.head()

sales.pivot\_table(values='count', index='product', aggfunc='sum'

pd.set\_option('display.precision', 1)

pivot = sales.pivot\_table(values='count', index='product', aggfunc='mean')

pivot.columns = ['Avg # per Transaction']

sales['quarter'] = sales['time'].dt.to\_period('Q')

sales.head()

sales.pivot\_table(values='count', columns='quarter', aggfunc='sum')

sales.pivot\_table(values='count', index='product', columns='quarter',

aggfunc='sum')

* **Barriers Encountered:**

Some barriers I encountered when using python to aid in analyzing data were that I oftentimes found it difficult to filter out data, so I was able to easily analyze it. Oftentimes datasets are not filtered and can be very difficult to work with. Along with that, I thought it was difficult to work with large sets of data that had lots of columns as it was hard to decide which data was the most important to use for analyzation.

**Objective 6: Use Python for data visualization.**

* **Description in Own Words:**

Using python for data visualization involves using different functions such as the plot function to visually display data with Python. This allows you to create a multitude of graphs and charts that allow you to analyze and read large sets of data easier. Some external libraries that allow for better data visualization include the seaborn library, and the matplotlib library.

* **Understanding Grade (0-100):**

I would give myself an understanding grade of 98 for using python for data visualization.

* **Completion Evidence:**

Here is an example we did in class where we took a bunch of peoples favorite Easter candys and plotted the results in a bar graph.

easter\_candy = {

'candy': [ 'Reeses Egg','Cadbury Egg','Peeps', 'Jelly Beans', 'Chocolate Bunny'],

'votes': [7, 5, 0, 5, 3]

}

import pandas as pd

dfEasterCandy = pd.DataFrame(easter\_candy)

dfCandy = dfEasterCandy['votes']

dfCandy.index = dfEasterCandy['candy']

dfCandy = dfCandy.sort\_values(ascending=False)

dfCandy.plot(kind= 'bar')

* **Barriers Encountered:**

Some barriers I encountered when Using python for data visualization included figuring out how to use the different libraries such as matplotlib and seaborn. Along with that I often times had difficulty figuring out how to make sure a set of data was on the x or y axis when plotting it. Learning how to edit the graphs and make them look more pleasing took a large amount of time as there is just so many different things you are able to do with visualizing data on Python.

**Objective 7: Use SQL to connect to a database and populate data into a Python program.**

* **Description in Own Words:**

Using SQL to connect to database and popular data into a python program includes establishing connections between python and SQL databases, executing different queries in order to modify data, and being able to integrate data from SQL into python so you can further analyze it.

* **Understanding Grade (0-100):**

I would grade my understanding of using SQL to connect to a database and populate data into a python program a 90 as I mostly understand it but there is still some learning I could do.

* **Completion Evidence:**

Here is an example of me using SQL to connect to a database and transform data into a python program.

##Importing sqlite 3 and pandas as db and pd

import sqlite3 as db

import pandas as pd

##creating the SQL queries and storing them in variables as strings

sql2 = "SELECT \* FROM tracks;"

sql3 = "SELECT \* FROM albums ORDER BY Title;"

sql4 = "SELECT EmployeeId, FirstName, LastName, Title FROM employees;"

sql5 = "SELECT \* FROM artists WHERE Name LIKE 'A%';"

sql6 = "SELECT \* FROM artists ORDER BY Name DESC;"

sql7 = "SELECT \* FROM tracks WHERE UnitPrice = 1.99;"

sql8 = "SELECT \* FROM tracks WHERE UnitPrice != 1.99;"

##Establishing connection with chinook database

db\_connection = db.connect('chinook.db')

##Fetching data using the cursor for first three queries

db\_cursor = db\_connection.cursor()

##Query 2: Extracting all details from the tracks table

db\_cursor.execute(sql2)

all\_tracks\_data = db\_cursor.fetchall()

all\_tracks\_cols = [desc[0] for desc in db\_cursor.description]

df\_all\_tracks = pd.DataFrame(all\_tracks\_data, columns=all\_tracks\_cols)

print("All Tracks:", df\_all\_tracks)

##Query 3: Retrieving all details from the albums table,

##with an alphabetical sort by title

db\_cursor.execute(sql3)

sorted\_albums\_data = db\_cursor.fetchall()

sorted\_albums\_cols = [desc[0] for desc in db\_cursor.description]

df\_sorted\_albums = pd.DataFrame(sorted\_albums\_data, columns=sorted\_albums\_cols)

print("Sorted Albums:", df\_sorted\_albums)

##Query 4: Getting EmployeeID, FirstName, LastName,

##and Title from the employees table

db\_cursor.execute(sql4)

employee\_details\_data = db\_cursor.fetchall()

employee\_details\_cols = [desc[0] for desc in db\_cursor.description]

df\_employee\_details = pd.DataFrame(employee\_details\_data, columns=employee\_details\_cols)

print("Employee Details:", df\_employee\_details)

db\_cursor.close()

##Executing remaining queries using read\_sql\_query from the pandas library

##Query 5: Fetching artists starting with "A"

df\_artists\_starting\_A = pd.read\_sql\_query(sql5, db\_connection)

print("Artists Starting with 'A':", df\_artists\_starting\_A)

##Query 6: Listing all artists in descending order by their name

df\_artists\_by\_desc\_name = pd.read\_sql\_query(sql6, db\_connection)

print("Artists by Descending Name:", df\_artists\_by\_desc\_name)

##Query 7: Retrieving all tracks priced at 1.99

df\_tracks\_at\_199 = pd.read\_sql\_query(sql7, db\_connection)

print("Tracks Priced at 1.99:", df\_tracks\_at\_199)

##Query 8: Finding tracks with a UnitPrice other than 1.99

df\_tracks\_not\_at\_199 = pd.read\_sql\_query(sql8, db\_connection)

print("Tracks Not Priced at 1.99:", df\_tracks\_not\_at\_199)

##Closing the database connection after executing all queries

db\_connection.close()

* **Barriers Encountered:**

Some barriers I encountered when Using SQL to connect to database and popular data into a python program were issues with integrating the SQL queries into python and figuring out how the fetch all with the cursor worked, and how to execute the functions. Along with that, understanding the different types of joins you can do within SQL was very hard to understand at times.