

1. The best practices I used when developing the code was testing the code every few lines to narrow down where the errors were occurring. If I had wrote the entire program without testing, then it would be difficult to isolate the bugs. Commenting also helped me develop the code also and stay on task labeling code blocks. Also making the code block descriptive enough to help another person who reads the code. Lastly, I named the variable short but descriptive, so I was able to call the correct variable when the time came. In developing the flowchart, I used the lab as a guide in determining which flowchart item to place, labeling them, and having arrowing pointing in the right direction to show the flow of information and processes.

Other best practices I used were to make the flowchart balanced and beautiful. This helps readers gather information more visually and help to understand the flow of the chart. I focused on symmetry and flush lines that do not zigzag throughout the chart. Also, I attempted to make the flowchart and code as flat as possible. I did not want to develop complex nested statements like looping the customer selection and appending that to a list that called the dictionary for the price which is complex. Since the customer was only allowed two services this made the structure much easier. I felt that this was the most obvious way to run the code.

1. The approach I used to problem solve was analyzing the pseudocode in the directions of the lab 4.10 Automotive Service Invoice. The directions included the task to be done and it built upon the task before it. Adhering to the labs format to the exact amount of characters and spaces was helping but based on the pseudocode a person could develop code with the same fundamental requirements. In the flow chart each node represents code blocks so it helped in determine the separate nodes.

Further, based on feedback realized are sub-steps within the steps in the directions and broke those down even further to discover more input/output activities and included control structure activities that was in my code and displayed in the flow chart. Also, I used a decision in my flow chart to loop back to the services menu and ask the customer for input if they decided not to go through with the services plan. Then I created another decision to help the customer decide if they want to quit the process although from the flow chart they will have to enter two services of enter “-“ for both choices for a bill of $0 and no service. This removed the creating of a blank invoice for when the customer wants to leave.

1. I annotated the code in major blocks that seemed most important such as develop a menu. This would be a dictionary that consisted of a menu item and also prices linked to that item. I also added a note that state why I chose to make that menu the way I did which is to make the menu easily updateable in the future my only changing two elements. I also commented what I had added with using the ‘-‘ and associating that with a “No Service” option on the menu. I found using this algorithm structure and data structure of the dictionary most effective in reaching the solution.

# Develop a menu of keys and values (services:prices)

# Softcode menu for easy changes with a standard menu format

services = {'Oil change': 35, 'Tire rotation': 19, 'Car wash': 7, 'Car wax': 12, 'No service': 0}

print("Davy's auto shop services")

print("%s -- $%d" % ('Oil change', services['Oil change']))

print("%s -- $%d" % ('Tire rotation', services['Tire rotation']))

print("%s -- $%d" % ('Car wash', services['Car wash']))

print("%s -- $%d" % ('Car wax', services['Car wax']))

# Prompt user for first and second service

# Surround user prompt with if statement incase user selects "no service"

# Substitute '-' for 'No service' on the menu

first\_service = input("\nSelect first service: \n")

if first\_service == '-':

first\_service = "No service"

second\_service = input("\nSelect second service: \n")

if second\_service == '-':

second\_service = "No service"

# Begin Invoice

#Display itemized invoice

#Calculate invoice total

print("\n\nDavy's auto shop invoice\n")

if first\_service == "No service":

print("Service 1: No service")

else:

print("Service 1: %s, $%d" % (first\_service, services[first\_service]))

if second\_service == "No service":

print("Service 2: No service")

else:

print("Service 2: %s, $%d" % (second\_service, services[second\_service]))

print("\nTotal: $%d" % (services[first\_service] + services[second\_service]))

Part II.

After conducting further research, I am compelled to dive into the field of big data. Information surrounds us and as time passes more information in being either collected or missed. I can look out to the world and determine the data points they can produce. Big data connects to IoT and Cybersecurity. IoT can collect and produce a dataset that can be analyzed. How we interact to the world and our tools can now be visualized and interpreted. Big data is also connected to cybersecurity because this information is private and should be secured. In cyberspace data and be manipulated and used for personal gain. Big data is significant because it will provide us with more actionable information that will help us make better decisions and prevent mistakes.

I would describe big data to someone who is unfamiliar as “an extremely large set of information, collected to be interpreted to help make people make better decisions”. An example I would give is collecting weather information to predict when and if the next natural disaster will occur, so companies and disaster relief teams can gather supplies ahead of time and save lives. For example, Walmart has a weather center so they can track any issues with shipments and calculate possible delays. They used their system to deliver water to those who needed help after Hurricane Katrina. Using data, they were able to locate the closest Walmart locations and suppliers to deliver a certain amount of water to specific locations.

Future development for big data is data privacy and governance. Since data can be used to manipulate others, future development will be concentrated on creating standards to protect people from being taken advantage of. Development of best practices with other people’s information will adopt practices used in healthcare, HIPPA. In the previous paragraph I discussed an application of big data and how it can be used in society. Another application is studying health patterns. Is there an increase in eye problems as technology is being used more frequently and for longer durations? Using patterns in history we can determine is this is true. Another application is tracking traffic patterns on busy highways. Your GPS app already does this by telling you when there will be heavy traffic based on the data they collect and attempts to re-route you to the best path.

Some potential career opportunities in big data is titles like data analyst, data scientist, statistician, actuary, mathematician, and database administrator. Someone in big data would need to have business acumen. The ability to understand business operations and how data flows through the information system. They data professional would need to know what questions to ask and what information will be needed to get to the answer. Further, the data professional would need to know how to interpret data and communicate that data in a digestible way to managers and chief officers to use.

Part III.

An activity I found interesting and challenging was 7.15.1: Ch 8 Custom lab for SNHU Online - Program: Sorting movies (Lists and Dictionaries) (Python 3) Lab Activity. This activity took my full concentration and I learned to apply the skills I learned earlier to combine them to make a solution. Using variable, control structures, list, and dictionaries gave a full overview of how problem solving with programming really is.

When programming it was hard not to just jump into it and code because it seems so simple after just a few minutes of developing I got stuck and remembered that developing a plan of either a flow chart or pseudocode can greatly help with the development because I won’t be planning as I go. One of the best practices to quickly get started was developing pseudocode to get an idea for what I was going to program. Another practice I found helpful was to test the code with every live to make sure that an error could easily be found. This greatly reduced the number of times I had to search through my code just to find a small error that messes up the rest of the program. Another best practice I used was naming objects to be specific but not longwinded to make referencing object much easier. Like using plural in a list or dictionary when running loops, I would use a singular name structure to loop through the plural. (ex. “for each title in movies”, “for producer in producers”)

The problem-solving approach I used was developing micro-pseudocode for each step. This lab activity was particularly challenging because of the complexity and the amount of moving part involved in developing the solution. First to create the dictionary, then discovers that the keys needed to be unique to iterate over the keys. Then nesting loops in an if-then control structure. And within those control structures, turn certain information into list to iterate over them. It was important that I chunk the larger problem into smaller more manageable problems. Using the prompts and directions I was also able to envision what a flowchart would look like had one been needed.

I also tested the code frequently, examining what the code did based on test inputs, and then rewriting the code to get the correct output, then repeating if necessary. This made writing the code investigative and testing out if a certain syntax would work in python. The tool I used to quickly move from writing to testing is PyCharm. This is a great IDE that allowed me to quickly lest my code in a readable environment that displayed on a console. When writing the code in zybooks it does not halt to code to ask for an input but instead you must already have the input written in the text box. This made it difficult to understand the flow of the code.

Throughout the code I included general annotations that several main objectives based on the prompts above. Such as, # Build a dictionary, and #prompt user for a year. The first prompt asked for the users input to be between a certain number and if it wasn’t print “N/A”. This was a control stricture and annotated the beginning of a condition. The else if the condition was a bit tricky but when worded out it became easy to see the condition that needed to be met. (# for each title in movies, print if the year == users input) This was a basic search algorithm to return what meets the users input I chose because I saw it as the straightest path to display the result.

The structure I used for the menu options was a while loop with the other commands nested within. The ‘q’ option acted as the break and would allow the user to loop through the menu to select again. Then I used an if-then control structure to make a decision-tree for each option. The overall instructions I gave myself was “make a list”, “remove duplicates”, “for each year in the list, find each movies that corresponds to the year” (this began the outer loop since we wanted to iterate through each year in the list then iterate over the values in the dictionary that equaled that year in the list and print them. I believe this was the most efficient solution because each step is blocked, and no code was unnecessarily repeated. After completing the first search prompt is became easy to complete the menu option prompts since the same format for the code was used. The only thing I needed to do was figure out how to nest the loop.

# Build a dictionary containing the specified movie collection

movies = {

'Munich': ['2005', 'Steven Spielberg'],

'The Prestige': ['2006', 'Christopher Nolan'],

'The Departed': ['2006', 'Martin Scorsese'],

'Into the Wild': ['2007', 'Sean Penn'],

'The Dark Knight': ['2008', 'Christopher Nolan'],

'Mary and Max': ['2009', 'Adam Elliot'],

'The King\'s Speech': ['2010', 'Tom Hooper'],

'The Artist': ['2011', 'Michel Hazanavicius'],

'The Help': ['2011', 'Tate Taylor'],

'Argo': ['2012', 'Ben Affleck'],

'12 Years a Slave': ['2013', 'Steve McQueen'],

'Birdman': ['2014', 'Alejandro G. Inarritu'],

'Spotlight': ['2015', 'Tom McCarthy'],

'The BFG': ['2016', 'Steven Spielberg']

}

# Prompt the user for a year

user\_input = int(input('Enter a year between 2005 and 2016:\n'))

# Displaying the title(s) and directors(s) from that year

if 2005 > user\_input or user\_input > 2016:

print('N/A')

else:

for title in movies:

if movies[title][0] == str(user\_input):

print('%s, %s' % (title, movies[title][1]))

print()

# Display menu

menu\_prompt = 'MENU\n\

Sort by:\n\

y - Year\n\

d - Director\n\

t - Movie title\n\

q - Quit\n\

\n\

Choose an option:'

# Carry out the desired option: Display movies by year,

# display movies by director, display movies by movie title, or quit

user\_choice = input(menu\_prompt)

print('') # Begin control structure

while user\_choice != 'q':

if user\_choice == 'y':

sorted\_year\_list = sorted(movies.values()) # make a list of movies values

del sorted\_year\_list[1] # remove duplicates

del sorted\_year\_list[6]

for year in sorted\_year\_list: # for each year in the list, find each movies that corresponds to the year

print('%s:' % year[0])

for title in movies: # For each title(key) in movies(dict), if the years match, print the title, director

if movies[title][0] == year[0]: # same as above, slgiht different

print('\t%s, %s' % (title, movies[title][1]))

print('')

elif user\_choice == 'd':

sorted\_director\_list = list(sorted(movies.values()))

del sorted\_director\_list[4]

del sorted\_director\_list[12]

producers = []

for pair in sorted\_director\_list:

producers.append(pair[1])

for producer in sorted(producers):

print('%s:' % producer)

for title in movies:

if movies[title][1] == producer:

print('\t%s, %s' % (title, movies[title][0]))

print('')

elif user\_choice == 't':

list\_of\_titles = sorted(movies.keys())

for movie in list\_of\_titles:

print('%s:' % movie)

for title in movies:

if title == movie:

print('\t%s, %s' % (movies[title][1], movies[title][0]))

print('')

user\_choice = input(menu\_prompt)

print('')