Southern New Hampshire University

7-1 Final Project Part 1

Collection of Problem-Solving Activities

Brandon Lombard

CS-200-Q2055 Computer Sci Role in Industry

Instructor: Gholam Shaykhian

12/12/2019

**Part 1: Input/output activity**

**Pseudocode**  
  
MEASURE wall height  
MEASURE wall width  
OBTAIN area of wall  
CALCULATE gallons of paint needed  
CALCULATE cans of paint needed  
DETERMINE color of the paint  
CALCULATE cost of purchasing the paint  
DISPLAY calculations

import math

# Dictionary of paint colors and cost per gallon  
paintColors = {  
 'red': 35,  
 'blue': 25,  
 'green': 23  
}

wallHeight = float(input('Enter wall height (feet): \n'))  
  
# FIXME (1): Prompt user to input wall's width  
wallWidth = float(input('Enter wall width (feet): \n'))  
  
# Calculate and output wall area  
wallArea = wallWidth \* wallHeight  
print('Wall area:', wallArea, 'square feet')

# FIXME (2): Calculate and output the amount of paint in gallons needed to paint the wall  
gallonsNeeded = float(wallArea / 350)  
print('Paint needed:', gallonsNeeded, 'gallons')

# FIXME (3): Calculate and output the number of 1 gallon cans needed to paint the wall, rounded up to nearest integer  
cansNeeded = math.ceil(gallonsNeeded)  
print('Cans needed:', cansNeeded, 'can(s)\n')

# FIXME (4): Calculate and output the total cost of paint can needed depending on color  
paintChoice = str(input('Choose a color to paint the wall: \n'))  
paintCost = paintColors[paintChoice] \* cansNeeded  
print('Cost of purchasing %s paint: $%d' % (paintCho

Both pseudocode and flow charts are necessary in creating an efficient computer program. I had debated which was the better option. However, for the “Painting a wall” program, I developed pseudocode because it is the most effective in my eyes. Both techniques are essential in planning your code process before you write your program, but pseudocode is quicker and better laid out because it is in a straight text format. I prefer not to mess around with a design program just to lay out my plans for developing code.

One best practice I used in my code is “readability counts.” This makes the code easier to read and follow, because if you gave random names in the code, one element could be mistaken for another, and using comments makes it easier to see what the code below it does. Another best practice I used was “beautiful is better than ugly.” In this, I properly spaced the code ton make it much easier to read and sort through. Finally, “consistent names” were also given so that you could easily identify what that line of code does. To complete this task, I had to figure out formulas to make this program properly work. To problem solve this, I used zybooks and also used Google to ensure I had the proper information. Zybooks helped me in any places where I forgot what code I should use for different pieces of this project. Also, I ran into some struggle in placing the code in a particular order, which I quickly remembered that the computer reads it in order, so rearranging it may help.

**Part 2: Control Structures Activity**

#Establish service types and cost of service.

#Best practice: relevant naming.

oil\_change = 35

tire\_rotation = 19

car\_wash = 7

car\_wax = 12

choice1\_cost = 0

choice2\_cost = 0

#Print service costs.

print("Davy's auto shop services")

print('Oil change -- $35')

print('Tire rotation -- $19')

print('Car wash -- $7')

print('Car wax -- $12\n')

#Input chosen services.

#Best practice: keeping code consistent by using strings.

first\_service = str(input('Select first service: \n\n'))

second\_service = str(input('Select second service: \n'))

print('\n')

#Print first chosen service.

#If-else statements are the most efficient data structure for this project because I can determine what is printed in the program by the user’s input.

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

if first\_service == 'Oil change':

print('Service 1: ', first\_service + ", $", oil\_change, sep='')

choice1\_cost = oil\_change

elif first\_service == 'Tire rotation':

print('Service 1: ', first\_service + ", $", tire\_rotation, sep='')

choice1\_cost = tire\_rotation

elif first\_service == 'Car wash':

print('Service 1: ', first\_service + ", $", car\_wash, sep='')

choice1\_cost = car\_wash

elif first\_service == 'Car wax':

print('Service 1: ', first\_service + ", $", car\_wax, sep='')

choice1\_cost = car\_wax

else:

print('Service 1: No service')

choice1\_cost = 0

#Print second chosen service.

#Using a sep= command so there isn’t any extra unnecessary spacing in my printed output.

if second\_service == 'Oil change':

#Problem solving: I originally had trouble combining the parts of the output together, but was able to use + and , to combine parts of the output together.

#Problem solving: I had spacing problems when outputting my code, but was able to use the sep= command to avoid this issue.

print('Service 2: ', second\_service + ", $", oil\_change, sep='')

choice2\_cost = oil\_change

elif second\_service == 'Tire rotation':

print('Service 2: ', second\_service + ", $", tire\_rotation, sep='')

choice2\_cost = tire\_rotation

elif second\_service == 'Car wash':

print('Service 2: ', second\_service + ", $", car\_wash, sep='')

choice2\_cost = car\_wash

elif second\_service == 'Car wax':

print('Service 2: ', second\_service + ", $", car\_wax, sep='')

choice2\_cost = car\_wax

else:

print('Service 2: No service')

choice2\_cost = 0

#Print total cost.

#Combining both costs with a plus symbol.

print('')

total\_cost = choice1\_cost + choice2\_cost

print('Total: $', total\_cost, sep='')

In the control structures activity, I used multiple best practices (as annotated) including keeping the code consistent with strings, annotating the code with comments, and using relevant naming. All of these best practices combines allowed me to design an efficient and easy to edit program. I also used some problem solving techniques as I encountered a few problems developing this code. I used google to research some pieces of code that helped me properly space the program, such as the “sep” command. Ultimately, my algorithm and data structure used mainly if-else statements because the user would input information regarding which service they would like to purchase (up to two). I also used input strings for the user to input their data. Then combined the strings on the relevant if-else statements. This was a fairly basic program and I didn’t run into many issues creating it, but it did teach me more in depth on how to create if-else statements and it helped with my assignments after it.

Out of the four topics available (big data, robotics, cybersecurity, and Internet of Things), I decided to choose cybersecurity. Cybersecurity is hugely important in preserving and securing information for big companies such as Facebook, Amazon, E-bay, and even the government. As technology is constantly growing and is unlikely to cease any time soon, people and organization’s information are constantly at risk of being stolen. Criminals are frequently coming up with new ways to manipulate and steal data, so cybersecurity will always be an important role in the internet and technology. An example of a well-known breach of information is the Experian website breach from 2013-2015 that affected 15 million customers. Hackers stole user’s names, dates of birth, street addresses, Social Security numbers, and driver’s license numbers(“Krebs on Security”).

Security hazards range from SQL injection attack, denial of service, session hijacking, and so on. With the vast array of possible online hazards, cybersecurity will always be an important asset in the safety of internet users. Not only do we have forward attacking hackers, but also hackers who manipulate people into unintentionally giving away their personal information. Hackers use a technique called phishing, where users will input their information

**Part 3: Activity I found interesting or challenging**

movie\_collection = {2005: [['Munich','Steven Spielberg']],

#Best Practice: organized code to ensure tidiness and that the code works properly.  
2006: [['The Prestige','Christopher Nolan'],['The Departed','Martin Scorsese']],  
2007: [['Into the Wild', 'Sean Penn']],  
2008: [['The Dark Knight', 'Christopher Nolan']],  
2009: [['Mary and Max', 'Adam Elliot']],  
2010: [["The King's Speech",'Tom Hooper']],  
2011: [['The Artist', 'Michel Hazanavicius'],['The Help', 'Tate Taylor']],  
2012: [['Argo', 'Ben Affleck']],  
2013: [['12 Years a Slave', 'Steve McQueen']],  
2014: [['Birdman', 'Alejandro G. Inarritu']],  
2015: [['Spotlight', 'Tom McCarthy']],  
2016: [['The BFG', 'Steven Spielberg']]}

#Best Practice: Relevant naming for the code, such as “year” and “movie\_collection” to ensure ease of use when looking back into code or if another developer wants to edit the code.  
year = int(input("Enter a year between 2005 and 2016:\n"))  
  
if(year<2005 or year>2016):  
 print("N/A")  
else:  
 movies = movie\_collection[year]  
 for movie in movies:  
 print(movie[0]+", "+movie[1])

var = 1  
print("")

while var==1:  
 print("MENU")  
 print("Sort by:")  
 print("y - Year")  
 print("d - Director")  
 print("t - Movie title")  
 print("q - Quit")  
 print("")  
 option = input("Choose an option:\n")  
 #Best practice: Keeping the code as simple as possible (not overexaggerating). I used if- else statements with basic for statements to keep the code simple and consistent.   
 if(option == 'y'):  
 for key in sorted(movie\_collection.keys()):  
 print(key,end=':')  
 print("")  
 for movie in movie\_collection[key]:  
 print("\t"+movie[0]+", "+movie[1])  
 #Best practice: Consistently spacing the program with print(“”) statements to allow a well-designed output.  
 print("")  
 elif(option == 'd'):  
 dirDict = {}  
 for key in sorted(movie\_collection.keys()):  
 for movie in movie\_collection[key]:  
 dire = movie[1]  
 if dire in dirDict:  
 dirDict[dire].append([movie[0],key])  
 else:  
 dirDict[dire] = [[movie[0],key]]

for key in sorted(dirDict.keys()):  
 print(key,end=':')  
 print("")  
 for dire in dirDict[key]:  
 print("\t"+ str(dire[0])+ ", "+str(dire[1]))  
 print("")

elif(option == 't'):  
 titleDict = {}  
 for key in sorted(movie\_collection.keys()):  
 for movie in movie\_collection[key]:  
 title = movie[0]  
 if title in titleDict:  
 titleDict[title].append([movie[1],key])  
 else:  
 titleDict[title] = [[movie[1],key]]

for key in sorted(titleDict.keys()):  
 print(key,end=':')  
 print("")  
 for title in titleDict[key]:  
 print("\t"+str(title[0])+", "+str(title[1]))  
 print("")

elif(option == 'q'):  
 break;  
 #Best practice: Using an else statement to make sure the code doesn’t have an error if the user enters an invalid option.  
 else:  
 print("Invalid option")

This section, 7.15 Ch 8 Custom lab – Sorting movies (Lists and Dictionaries), was the most difficult zyBooks activity thus far in my opinion. In encompassed many more difficult algorithms to develop and to plan. I used multiple best practices, as noted above in individual comments, ranging from organization of code, relevant naming, proper spacing, keeping the code simple, etc. Another simple but important best practice is to use an else statement, because if your user inputs a text that you didn’t prepare for, it would produce an error. An efficient code does not allow many errors, if any at all.

I ran into multiple problems when producing this code. I had to use an array of problem-solving techniques in order to properly create this program. It was difficult to properly work the dictionary and properly add a list to it. In this case, I went back into the zyBooks lessons and determined the best way to do so, and the best way to tie it to the “for” statements. I was also able to tie the strings and ints together with an “+” operand. After multiple tries, I was able to produce the proper format in order to efficiently print my code (with some help from google, too).

The algorithm and data structure I designed was the best option for my program in various ways. I used if-else statements to determine what the user inputs (with an else statement to catch any invalid entries). Then I used for statements to determine things such as the alphabetical order or order by year depending on what the user had input. I was able to use a “+” operand to combine different parts of the lists that were inside the dictionaries, in a specific order. This was the best algorithm in that it is efficient and clean, and does properly what the program is supposed to do.