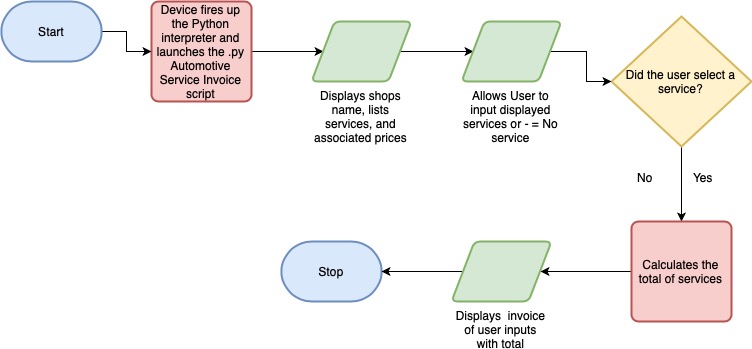
**Final Project Part 1**

**Flowchart for Automobile Service Invoice:**



**Automobile Service Invoice Code:**

# Creates service option variables with associated prices.

# **Best Practice 1**: Annotated my code in sequences or chunks.

print('Davy\'s auto shop services')

# **Best Practice 2:** Ignored at this point, by not keeping variables consistently lower case.

# **Problem-Solving 1:** Practiced with lists and dicts to test output, ended up scraping my original concept and using plan old variables with values best on program needed results.

Oil\_Change = 35

Tire\_Rotation = 19

Car\_Wash = 7

Car\_Wax = 12

no\_service = '-'

# Displays output of services and associated costs.

# **Problem-Solving 2:** Referenced past “Zybooks” course material to accomplish the below substitution methods in order to eliminate spaces required by the program.

print('Oil change -- $%d' % (Oil\_Change))

print('Tire rotation -- $%d' % (Tire\_Rotation))

print('Car wash -- $%d' % (Car\_Wash))

print('Car wax -- $%d' % (Car\_Wax))

print(")

# Allows User to input selected services

**# Best Practice 3:** Focused on using single quotes when statements where simple and double when complex. I tried to be consistent and definitely need to work on this concept.

service\_1 = str(input("Select first service: "))

print('\n')

service\_2 = str(input("Select second service: \n"))

print('')

print('')

# Displays an Invoice to the Customer with previously selected services, associated costs, and re-assigns variables with ints.

# **Best Practices 4:** Focused on 4 column indentation while writing if statements.

# **Problem-Solving 3:** These if statements took a major transition using different concepts from the previous code written and trial and error.

# **Best Practices 5:** This block of code taught me to stay one step ahead, because I originally wrote it a different way, but when it came to the next step, I couldn’t get the total to simply add.

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

if service\_1 == 'Oil change':

print("Service 1:", "%s, $%d" % (service\_1, Oil\_Change))

service\_1 = 35

if service\_1 == 'Tire rotation':

print("Service 1:", "%s, $%d" % (service\_1, Tire\_Rotation))

service\_1 = 19

if service\_1 == 'Car wash':

print("Service 1:", "%s, $%d" % (service\_1, Car\_Wash))

service\_1 = 7

if service\_1 == 'Car wax':

print("Service 1:", "%s, $%d" % (service\_1, Car\_Wax))

service\_1 = 12

if service\_1 == '-':

print("Service 1: No service")

service\_1 = 0

if service\_2 == 'Oil change':

print("Service 2:", "%s, $%d\n" % (service\_2, Oil\_Change))

service\_2 = 35

if service\_2 == 'Tire rotation':

print("Service 2:", "%s, $%d\n" % (service\_2, Tire\_Rotation))

service\_2 = 19

if service\_2 == 'Car wash':

print("Service 2:", "%s, $%d\n" % (service\_2, Car\_Wash))

service\_2 = 7

if service\_2 == 'Car wax':

print("Service 2:", "%s, $%d\n" % (service\_2, Car\_Wax))

service\_2 = 12

if service\_2 == '-':

print("Service 2: No service\n")

service\_2 = 0

# Prints total based on user service selections

total\_services = service\_1 + service\_2

print('Total: $%d' % (total\_services))

**Programming Best Practice & Problem Solving:**

When writing this code, I kept referring back to the best practices this course has taught. Some of the items I focused on were comments and where to place them. I based my placement decision based on the best practice to code and correct in small chunks. It seemed rather intuitive to place the comments where the code begins a new process. I also focused on 4 column indentation instead of using tab while writing if statements. One item, I did not follow through with was keeping variable names lowercase throughout my code.

This program took a lot of trial and error to complete. I kept referencing past elements that were vague or fuzzy in my knowledge bank. I tried multiple approaches with items, along the programs sequences, that seemed incompatible in one way or another with the overall results. Dictionary versus a list was a great example that I tested with, while trying to figure out how to get the prices to print for the user to see, without unwanted commas or colons. I ended up scrapping the concept and going with plan old variables with values assigned. I ultimately learned that code evolves, and it is best to keep the next sequence in your mind.

**Control Structure Activity:**

# This loop will allow the inner if and else statements to keep iterating while the appropriate condition is met. **# Best Practices 1:** I ensured that throughout this code 4 spaces were utilized to ensure proper indentation techniques. **# Problem Solving 1:** I conducted extensive external research looked at numerous examples on Stack Overflow and freecodecamp’s programming basics for Python tutorial videos. These resources assisted in providing clarity to various elements throughout this process. I chose the while statement, because when writing the below if and else statements the code would only run once without it. I believe that the while control structure provides the most efficient way of solving the iteration process, and makes the program intelligent.

while True:

# Allows the user to input a value and stores it in the variable user\_input. **# Best practices 2**: Utilized simple but efficient uniform naming scheme in addressing variables. **# Problem Solving 2:** This control structure technique is the most efficient in storing the user inputted values. I tried to do this variable and a stand-alone print statement to manipulate the string formatting that resulted in additional code.

user\_input = input('Enter input string: \n')

# Checks to see if the user inputted value is not equal to q and allows the program to iterate to the next sequence unless True. **# Best Practices 3:** Ensured the use of single quotes with simple strings and double with the more complex.

if(user\_input != 'q'):

# Replaces the specific values of the user\_input variable with the specified value. **# Best Practices 4:** I ensured that all code followed uniform spacing between the values and operators. **# Problem Solving 3:** The tool I used to ensure the variable stored had the proper formatting was the good old, print statement.

user\_input = user\_input.replace(' ', '')

# This if else statement checks to see if a comma was used in the user input and if that condition was met, splits the user inputted values at their comma. This if statement also prints the user inputted values at position 0 and 1. The else statement is in case the conditions are not met in the if statement. This condition also informs the user of a syntax format error. **# Problem Solving 4:** I had to review the previous chapter 3.2 to better understand the use of the correct index arguments, so I could achieve the desired print output. Ex: [0] vs [0:1].

if ',' in user\_input:

user\_input = user\_input.split(',')

print ('First word: ' + user\_input[0])

print ('Second word: ' + user\_input[1])

print("")

print("")

else:

print('Error: No comma in string.')

# Terminates the program if the user inputs the string q.

else:

break

**Challenging Project: Sorting Movies**

# Creates a dictionary with nested lists. **# Best Practices 1:** Ensured the dictionary’s curly brackets are located in the proper places. **# Problem Solving 1:** I created my dictionary with a single nested list but couldn’t get the second movie title and director to print using any index range. I used various tools to aid in resolving this issue like the print statement and the index function.

movies\_dict = {

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

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']]

};

# This prompts the user for a year and assigns the input to user\_year. **# Best Practices 1: I** used standard naming syntax and simple naming conventions when addressing variables throughout the program.

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

# This block check the variable user\_input and prints accordingly based on key/value index. **# Best Practices 2:** Ensured spacing between operators and operands were uniform. **# Problem Solving 2:** This may not have been the most efficient solution, but it is very simple to understand and eliminates unnecessary nested blocks of code that are complex. The downfall is that it only works with the current dictionary keys and values. If you were to add different years then the code would need to be modified accordingly. In retrospect the use of a for loop to iterate over the dict would have been a dynamic approach to this solution.

if (user\_year < 2005) or (user\_year > 2016):

print('N/A\n')

elif user\_year == 2005:

print('%s, %s\n' % (movies\_dict[2005][0][0], movies\_dict[2005][0][1]))

elif user\_year == 2006:

print('%s,' % movies\_dict[2006][0][0], movies\_dict[2006][0][1])

print('%s, %s\n' % (movies\_dict[2006][1][0], movies\_dict[2006][1][1]))

elif user\_year == 2007:

print('%s, %s\n' % (movies\_dict[2007][0][0], movies\_dict[2007][0][1]))

elif user\_year == 2008:

print('%s, %s\n' % (movies\_dict[2008][0][0], movies\_dict[2008][0][1]))

elif user\_year == 2009:

print('%s, %s\n' % (movies\_dict[2009][0][0], movies\_dict[2009][0][1]))

elif user\_year == 2010:

print('%s, %s\n' % (movies\_dict[2010][0][0], movies\_dict[2010][0][1]))

elif user\_year == 2011:

print('%s,' % movies\_dict[2011][0][0], movies\_dict[2011][0][1])

print('%s, %s\n' % (movies\_dict[2011][1][0], movies\_dict[2011][1][1]))

elif user\_year == 2012:

print('%s, %s\n' % (movies\_dict[2012][0][0], movies\_dict[2012][0][1]))

elif user\_year == 2013:

print('%s, %s\n' % (movies\_dict[2013][0][0], movies\_dict[2013][0][1]))

elif user\_year == 2014:

print('%s, %s\n' % (movies\_dict[2014][0][0], movies\_dict[2014][0][1]))

elif user\_year == 2015:

print('%s, %s\n' % (movies\_dict[2015][0][0], movies\_dict[2015][0][1]))

else:

print('%s, %s\n' % (movies\_dict[2016][0][0], movies\_dict[2016][0][1]))

# This is a small but subtle line of code that allows the below section of this program to run continuously while the condition is True. **#Problem Solving 3:** I choose to use this control structure based on my past project and it seemed the most efficient way to conduct the desired program’s operation.

while True:

# Displays the menu for the user to interact with and assigns there input in the variable.

movie\_menu\_input = input('MENU\nSort by:\ny - Year\nd - Director\nt - Movie title\nq - Quit\n\nChoose an option:\n')

#Establishes the condition and specifies the input character to terminate the program. **# Best Practices 3:** I ensured to use the proper 4 spaces for each nested line of code involving If statements and for loops.

if movie\_menu\_input != 'q':

# If the user enters the string y the movies will be displayed based on year. **# Problem Solving 4:** I had to adjust the print statement several times to obtain the correct spacing for the output. I also realized that a single list would not work and had to go back to the dictionary and readjust the nested lists accordingly.

if movie\_menu\_input == 'y':

# This for loop iterates through the movies\_dict keys and assigns their names as values to the variable year.

for year in sorted(movies\_dict.keys()):

print(year, end = ':\n')

# This for loop iterates through the movies\_dict list values and assigns them to the variable movie. You can access individual values by their proper index positions.

for movie in movies\_dict[year]:

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

print('')

# This block of code allows the movies to be sorted by the director. **# Problem Solving 5:** I researched a lot of different methods for this control structure to work. I didn’t actually realize up until this point how stupid a computer really was. Some of the approaches that I decided to take were creating another dictionary or a simple list to accomplish this task, but the issue was the unique values that the year dict expressed. I even found some function called, lambda which I abandoned, mainly because my head was going to explode if I had to grasp another Python concept. **# Best Practices 4:** I initially was giving my for loop variables a unique namebut abandoned this approach, because it was complicating the code writing process. In most cases it’s best to keep things simple.

if movie\_menu\_input == 'd':

# Creates an empty dictionary called directors\_dict.

directors\_dict = {}

# This for loop iterates through the movies\_dict keys and assigns their names as values to the variable key.

for key in sorted(movies\_dict.keys()):

# A nested for loop that assigns the movies\_dict values associated with the keys to the variable movie and creates a variable named director with the values of the variable movie index 1.

for movie in movies\_dict[key]:

director = movie[1]

# A Nested if statement that states if there is any value assigned to the variable director contained within the directors\_dict then the keys are equal to the variable director and the title and original key are the new associated values contained within nested lists within that specific order.

if director in directors\_dict:

directors\_dict[director].append([movie[0],key])

else:

directors\_dict[director] = [[movie[0],key]]

# A for loop that assigns the keys of directors\_dict to the variable key and prints the values accordingly.

for key in sorted(directors\_dict.keys()):

print(key, end = ':')

print('')

# A for loop that takes the values assigned to the keys and assigns them to the variable director and prints according to the required index position.

for director in directors\_dict[key]:

print("\t"+ str(director[0])+ ", "+str(director[1]))

print('')

# This block allows the movies to be sorted by the year. This section operates in the same manner as the most previous annotated code block above. The only difference is variable names and key value assignments based on the programs desired operations.

if movie\_menu\_input == 't':

titles\_dict = {}

for key in sorted(movies\_dict.keys()):

for movie in movies\_dict[key]:

title = movie[0]

if title in titles\_dict:

titles\_dict[title].append([movie[1],key])

else:

titles\_dict[title] = [[movie[1],key]]

for key in sorted(titles\_dict.keys()):

print(key, end = ':')

print('')

for title in titles\_dict[key]:

print("\t"+str(title[0])+", "+str(title[1]))

print('')

# This simple line allows the program to stop once the user enters the string q.

else:

break