**Pseudocode: 3.11 paint the wall**

Start

OBTAIN the measurements for the wall’s height and its width in feet

Write wall\_Height value from the measurement OBTAINED for the wall’s height.

Write wall\_Width value from the measurement OBTAINED for the wall’s width.

CALCULATE the square feet for the area of the wall by multiplying the wall’s height with its width

Write wall\_Area value from the calculation

ASSUME one gallon of paint covers 350 square feet

CALCULATE the amount of paint needed by dividing the wall’s square feet by 350 square feet

Write paint\_Needed value from the calculation

Round up the amount of paint needed to find the amount of can(s) needed

Write can\_Needed value after rounding up

After getting the value of can(s) needed, go to Home Depot’s paint section

Once at the paint section, call wife and ask what paint color she wants the wall to be (Input)

Find Home Depot’s set price for the requested color (Dictionary)

Grab the amount of 1-gallon paint can(s) for the color requested to get

Pay the amount of 1-gallon paint can(s) multiplied by the set price for the requested color

Stop

**INPUT-OUTPUT:**

PRACTICES:

The best practices I used for this exercise were the following: Beautiful is better than ugly, simple is better than complex, sparse is better than dense, now is better than never,

INPUT:

1. 12
2. 15
3. Red

OUTPUT:

Enter wall height (feet):

Enter wall width (feet):

Wall area: 180.0 square feet

Paint needed: 0.5142857142857142 gallons

Cans needed: 1 can(s)

Choose a color to paint the wall:

Cost of purchasing red paint: $35

CODING:

1. import math
2. paintColors = {
3. 'red': 35,
4. 'blue': 25,
5. 'green': 23
6. }
7. #(1): Prompt user to input wall's width
8. # Calculate and output wall area
9. wallHeight = float(input('Enter wall height (feet): \n'))
10. wallWidth = float(input('Enter wall width (feet): \n'))
11. wall\_Area = wallHeight \* wallWidth
12. print('Wall area:',wall\_Area,'square feet')
13. # FIXME (2): Calculate and output the amount of paint in gallons needed to paint the wall
14. paint\_Needed = wall\_Area / 350
15. print('Paint needed:', paint\_Needed, 'gallons')
16. #(3): Calculate and output the number of 1 gallon cans needed to paint the wall, rounded up to nearest integer
17. cans\_Needed = math.ceil(paint\_Needed)
18. print('Cans needed: %d can(s)' % (cans\_Needed))
19. # FIXME (4): Calculate and output the total cost of paint can needed depending on color
20. userColor = input('\nChoose a color to paint the wall: \n')
21. price = paintColors[userColor]
22. totalPrice = int(price) \* cans\_Needed
23. print('Cost of purchasing %s paint: $%d' % (userColor, totalPrice))

Problem-solving approaches:

The problem-solving techniques I used to assist me with coding were going back throughout my notes to remember the use of certain expressions and wordings. Ex: **math.ceil** to raise the can needed to a non-decimal number. Another problem-solving technique I would use was a trial and error technique. I would input false information and run the program to see the results of the output. Even though this technique sounds like I go against the Zen of Python practice: “In the face of ambiguity, refuse the temptation to guess”, I do not believe it does because I am not guessing with random information instead I am testing a hypothesis of a certain coding I believe is right and the python error program will me a specific change to my coding. The tools I used was the run program tool and submit program to discover my errors in my coding for the lab activities.

**Controlled Structure:**

#Make a Dictionary

prices = {'-': 0,

'Oil change': 35,

'Tire rotation': 19,

'Car wash': 7,

'Car wax': 12

}

first\_service = ""

second\_service = ""

total = 0

print("Davy's auto shop services")

print('Oil change -- $35')

print ('Tire rotation -- $19')

print ('Car wash -- $7')

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

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

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

print('\n')

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

# Change inputs to dictionary items

if first\_service in prices:

first\_service == first\_service

else:

#No Service must have a numerical value, or it will cause an error

first\_service == '-'

if second\_service in prices:

second\_service == second\_service

else:

second\_service = '-'

if first\_service == 'Oil change':

print('Service 1: Oil change, $35')

elif first\_service == 'Tire rotation':

print('Service 1: Tire rotation, $19')

elif first\_service == 'Car wash':

print('Service 1: Car wash, $7')

elif first\_service == 'Car wax':

print('Service 1: Car wax, $12')

else:

print('Service 1: No service')

if second\_service == 'Oil change':

print('Service 2: Oil change, $35')

elif second\_service == 'Tire rotation':

print('Service 2: Tire rotation, $19')

elif second\_service == 'Car wash':

print('Service 2: Car wash, $7')

elif second\_service == 'Car wax':

print('Service 2: Car wax, $12')

else:

print('Service 2: No service')

# find Sum for the total

total = prices[first\_service] + prices[second\_service]

print('\nTotal: $%d' % (total))

**1A.** The best practices I used were the following:

* In the face of ambiguity, refuse the temptation to guess.
* If the implementation is hard to explain, it’s a bad idea. If the implementation is easy to explain, it’s a good idea.
* Simple is better than complex. Complex is better than complicated.
* Explicit is better than implicit
* Beautiful is better than Ugly.

**1B**. I used the problem-solving pseudocode to help me finish this code. Since I had two services, it helped me make sure I handled the first service and then the second service. I then brought both services together by adding each one’s end value to each other to receive the total for the service. I added comment characters into my algorithm in order to leave notes for some of the major key points of my algorithm.

**1C**: As you can see in my input and output beneath this statement, my algorithm is simple, easy to understand, and it completes the task required. I chose the if and else structure because it allowed the program to be ready for different inputs from the customer and the program had set outputs for each type of input.

INPUT: Oil change

Dragon

OUTPUT:

Davy's auto shop services

Oil change -- $35

Tire rotation -- $19

Car wash -- $7

Car wax -- $12

Select first service:

Select second service:

Davy's auto shop invoice

Service 1: Oil change, $35

Service 2: No service

Total: $35

**MY CHOICE:**

while True:

input\_strings = input('Enter input string: ')

print("\n".rstrip("\n"))

#’q’ will terminate the loop

if(input\_strings !='q'):

input\_strings = input\_strings.replace(' ', '')

#Other requirements are considered inside the data structure.

if ',' in input\_strings:

input\_strings = input\_strings.split(',')

print ('First word: ' + input\_strings[0])

print ('Second word: ' + input\_strings[1])

print('\n')

else:

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

else:

#Force a stop in the loop because of ‘q’ input

break

**3A:** The best practices I used were the following:

* In the face of ambiguity, refuse the temptation to guess.
* If the implementation is hard to explain, it’s a bad idea. If the implementation is easy to explain, it’s a good idea.
* Simple is better than complex. Complex is better than complicated.
* Beautiful is better than Ugly
* Sparse is better than dense

**3B:** I used the problem-solving flowchart to help me finish this code. Since I multiple outcomes for certain requirements, it helped me make sure I was able to branch off certain statements because of the multiple final outputs. I added comment characters into my algorithm in order to leave notes for some of the major key points of my algorithm.

**3C:** My algorithm had multiple situations that had different outcomes depending on the user’s input. The statements were complex, but the coding was short, and it completes the task required. The first structure I used was while True structure to create a loop for the coding. I then placed multiple if-else structures inside of the while True structure because there were multiple final outputs, depending on the requirements, of the user’s input. I then used the “break” statement to force an immediate exit of the loop if the user’s input was “q”. The “break” statement prevented any infinite loop for the coding because it gave the user a way out.