**INPUT / OUTPUT PSUDEOCODE**

START

OBTAIN wall height

OBTAIN wall width

wall area = wall height \* wall width

OUTPUT wall area

paint amount = wall area / 350

IF wall area <= 1 then

cans needed = 1

ELSE

cans needed = wall area // 350

ENDIF

OUTPUT cans needed

OBTAIN chosen color

Total = Cans needed \* paint price

OUTPUT total

END

**INPUT / OUTPUT**

import math

#Using a library makes calculating the total price a breeze

paintColors = {

'red': 35,

'blue': 25,

'green': 23

}

wallHeight = float(input('Enter wall height (feet): \n'))

wallWidth = float(input('Enter wall width (feet): \n'))

wallArea = float(wallWidth \* wallHeight)

print ('Wall area:', wallArea, 'square feet')

paintAmount = wallArea / 350

print ('Paint needed:', paintAmount, 'gallons',)

paintGallons = math.ceil(paintAmount)

print ('Cans needed: %d can(s)' %(paintGallons))

color = input('\nChoose a color to paint the wall: \n')

price = paintColors[color]

totalPrice = int(price) \* paintGallons

print ('Cost of purchasing', color, 'paint: $%s' % (totalPrice))

**Analysis**

In this program I tried to implement the best coding practices throughout. Beginning with the variable names, I attempted to make them simple and concise. Zybooks prompted the use of uppercase letters at the beginning of each word fallowing the first in order to sperate the words (ex. “wallHeight”). I find the best practice is to use underscores to separate multiword variables (ex. wall\_height). If I had felt I had the option, I would have implemented this technique because it is the easiest to read.

Another practice I used in this program was organization. I used blank lines throughout the program in order to make it less confusing to read. Though it is a simple program, if not organized well it could become quite a bear to read. By separating key parts of the program, it becomes much easier to understand because you can visually see how the program will execute.

One thing I would critique myself on after reviewing this program is not placing a space between “print” and it’s preceding “(“ throughout the program. I think the readability of this program would have been better had I done so. The “print” lines appear cluttered because everything is clumped together, which makes it hard to read.

When writing this program, I ran in to two problems. The first problem that I ran in to was figuring out how to produce the proper gallon count. For rooms that do not have a surface area divisible by 350 without producing a reminder, an additional gallon is required in order to completely cover the room. I tried implementing the remainder (%) in order to achieve this by adding 1 gallon to “paintGallons” if there was a remainder. I chose not to go this route because we had not learned if and else statements yet and I wanted to be sure and do it right. So, I googled a few things and came across Python’s mathematical functions site and found the function “math.ceil”. This function returns the ceiling value of a float and that was exactly what I needed in order to round up to the desired gallon count (9.2. math, 2019).

The second problem I had was figuring out how to print the total with the correct distance from the colon. I again resorted to google and came across a thread on stackoverflow.com. This thread gave examples of how to replace a number with “%d”. I then implemented this tactic in order to insert the colon immediately following the word “needed” but with a space between the colon and the total as Zybooks requested.

**ALGORITHM AND DATA STRUCTURE**

All of the variables used in the program were necessary in order to make the program easily understood and manipulatable. A library is used in order to correlate the paint colors with their prices which makes finding a total price a breeze. Logically this program makes sense because the first step someone takes when calculating the cost of paint to cover a room is to get the area of the room. Then, they find out how many gallons are needed to cover that area. Then they select the color that they want it painted and from there a price can be calculated once the price per gallon and amount of gallons needed has been found.

**CONTROL STRUCTURES**

START

while true

name = input

print (#name without spaces)

if name != q

name = name #without spaces

if “,” in name

name = name #separate at comma

print (first word)

print (second word)

print name #without spaces

print (“ “)

else

print (error)

END

while True:

name = input(str('Enter input string: '))

print ("name".rstrip("name"))

if (name !='q'):

name = name.replace(' ', '')

if ',' in name:

name = name.split(',')

print ('First word:', name[0])

print ('Second word:', name[1],)

print ("name".rstrip("name"))

print ('')

else:

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

else:

break

**Best practices**

For this program, I really tried to keep it as simple as I could. During my first attempt I had 3 variables. After reviewing my program, I realized that two of those variables were unnecessary. One stored the first word and the other stored the second word. I also made sure that the program was easy to read in both appearance and logic. Utilizing the methods presented to me in ZyBooks 6.3 and 6.4 really assisted in the simplicity and readability of the program.

**Problem Solving**

I was having an issue figuring out how to eliminate multiple spaces before and after the words. I knew to use the rstrip() method to eliminate single white spaces before and after words but I wasn’t sure how to get rid of multiple. I thought to do a “while” loop that would run the rstrip() method until all white spaces were gone but I didn’t like the way that looked after I finished. I then searched ZyBooks for other methods that could either do what I needed or be manipulated to do so. The cleanest way I found to accomplish the task was to use the replace() method to replace all spaces with no spaces. Surprisingly, it ended up working out very well, so I ran with it.

When I first attempted to write this program, I had multiple variables and if/else statements scattered about. This program completed most of the tasks but not all. I eventually realized I needed to begin with a loop because once one-word combination was entered that contained a comma my program would quit, and I needed the program to run until “q” was entered. After some frustrating trial and error, and a little help from Repl.it I found what I was missing.

**Algorithm and Data Structure**

This program is the most efficient solution because I simply made it a “while” loop. I kept it simple by utilizing many methods including: split(), replace(), and rstrip(). I also only ended up using one variable name in the whole program.

**YOUR CHOICE**

**Pseudocode**

Weight count

Count

Weight amount

Weight array

While count < wight amount

“enter weight”

Weight amount = Input

Wight array [count] = Weight amount

Count + 1

End while

Print weight array

Avg = weight array (sum) / 4

Print avg

Max = max(weight array)

Print max

“enter weight index number for conversion”

Print “weight in lbs”

Weight amount / 2.2

Print “weight in kg”

End

**Program**

#Max number of weights in the array

num\_weights = 4

#Began the array at 1 instead of 0 to make it visually easier to understand

cnt\_weights = 1

#Takes in the inputted weight by the user

weight = 0

#The array that all of the weights are held in

weights = []

#While loop that inserts all of the weights into the weights[] array

while cnt\_weights <= num\_weights:

print ('Enter weight %d: ' % cnt\_weights, end='\n')

weight = float(input())

weights.append(weight)

#adds one to the count to limit the loop to 4

cnt\_weights += 1

print ('')

print ('Weights:', weights)

#Finds the average of the four weights

avg\_weight = sum(weights) / 4

print ('Average weight:', avg\_weight)

#finds the max value in the weights[] array

weight = max(weights)

print ('Max weight:',weight)

print ('\nEnter a list index (1 - 4): ', end='\n')

#index via subtraction

weight = int(input()) – 1

print ('Weight in pounds:', weights[weight])

weight = weights[weight]

#Converts the selected weight to kilograms

weight /= 2.2

print ('Weight in kilograms:', weight)

print ('\nSorted list:', sorted(weights))

return

**BEST PRACTICES**

Readability is of the most important “Best Practices” in my opinion. Inevitably, any code that is written for a company will have to be read by a human no matter how well it was written due to the rate in which technology advances. In this program, I made sure to use comprehensible variable names and to include appropriate spacing between words and syntax terms. I also tried to keep this program as simple as I could. I ended up only using three variable names and a list in this program. Considering all the tasks that were asked of the program I feel that kept the program as concise and as basic as I could.

**PROBLEM SOLVING**

I only ran in to one problem when I was writing this program. I needed to figure out how to print the correct weight for the output “Weight in pounds”. Each weight is stored in the list *weights[],* list’s begin at the number zero, not number one. For this reason, the number inputted by the user is not equal to the element’s placement within the list. In order to display the correct element 1 needs to be subtracted from the user’s input.

I first tried subtracting one from *weight* and making that answer equal to *weight* so that it could be used to designate the placement within the *weights[]* list, but it didn’t end up working. So, I tried a few different approaches, and through the process of trial and error I finally found a solution. I ended up using this line; *weight = int(input()) – 1*. This line works because it simply takes the value that is inputted and subtracts the number one in a single line.

**ALGORITHM AND DATA STRUCTURE**

For each “FIXME” task, my goal was to be able to successfully execute each task with as few lines as possible. In order to do that I utilized many methods such as sum(), max(), and sorted() in this program. These methods really cut down on the number of things I had to type. One example of this is the sum() method. Instead of adding weights[0] + weights[1] + weights[2] + weights[3] to get the sum, I simply wrote sum(weights).

**REFERENCES**

1. 9.2. math - Mathematical functions¶. (n.d.). Retrieved May 25, 2019

2. Darkskydarksky. (n.d.). Print Combining Strings and Numbers. Retrieved May 25, 2019

3. Ranger, S. (2019, June 05). What is the IoT? Everything you need to know about the Internet of Things right now.

4. Repl.it. (n.d.). 11.9 zylab (strings) Parsing strings (Python 3). Retrieved June 8, 2019