**02.06 Module Two Project**

Name: Daniel Robert Bealer Quintero

**Directions**

Now that you have an understanding of using functions and operations with numerical and non-numerical data, it's time to show your instructor and yourself what you can do with them! You will use the software development life cycle (SDLC) to create a working program of your choice that will perform calculations with user input. Your program can include any Python skills and functions you have learned up to this point.

**This assignment has five steps.**

**Step One: Planning & Analysis**

Read the options carefully and choose **ONE** as the basis of your project.

**Option 1 – Wish List**

Have your eye on a new gadget? A new game? Maybe some new clothes? Well, now's your chance to create a program that calculates the total cost of three items on your wish list, including tax and shipping.

Follow these steps to begin your planning:

1. Create a wish list by selecting at least three items you want.
2. Do your research. Find the online store(s) where you can purchase your wish list items, and record the price of each one.
3. Think about what user input is required for others to use your program.
4. Think about how you will write an equation to calculate the subtotal of your three items, the tax, and the total purchase cost with tax and shipping. **Note:** Use 6.5% tax and a $5.99 flat-rate shipping fee for your program.
5. The output must include the following: name of each item, item price, subtotal for items, total amount of tax, shipping fee, and total purchase cost with tax and shipping.

Use this table to organize your data:

|  |  |
| --- | --- |
| Item | Price |
| Mac Book Pro | 1999 |
| iPad Air | 790 |
| Apple Pencil Pro | 98 |
| Home Pod Mini | 99 |

STUDENT NOTE: The program has a product system where you input custom product and assign their categories and respective prices. Which means that the products shown in the table above will not be complied in the code, but you can set those same products in the program by manually inputting them once the program asks you how many products you want to make.

**Option 2 – Let's Eat**

Everybody loves eating at restaurants! If you agree, take this opportunity to create a program that will calculate the total cost of a meal that includes an appetizer, entrée, drink, and dessert, with 6.5% tax and a 20% tip.

Follow these steps to begin your planning:

1. List your favorite appetizer, entrée, drink, and dessert.
2. Do your research. Find the menu for your favorite restaurant, and record the prices of your chosen menu items.
3. Think about what user input would be required for others to use your program.
4. Think about how you will write an equation to calculate the subtotal of your meal and the total cost of your meal, plus 6.5% tax and a 20% tip.
5. Your program must output the following: name of appetizer, entrée, drink, and dessert, including prices; subtotal for meal; total amount of tax; total amount of tip; and total cost of the meal with tax and tip.

Use this table to organize your data:

|  |  |
| --- | --- |
| Menu Item | Price |
|  |  |
|  |  |
|  |  |
|  |  |

**Option 3 – Programmer's Adventure**

If you want to journey the road less traveled and create your own program option, go for it! Keep in mind, the goal of your program must be to calculate *something*, using the operations and functions you have learned up to this point. Your program must output the following:

* item names and values
* results of the calculations

**Possible Program Options:**

* *Mini vacay* – Calculate the total cost of a two-day one-night vacation, including airfare, lodging, and two meals per day
* *Coupon Mania* – Calculate the total percentage of savings on three grocery items with coupons.
* *Stock Supreme* – Calculate the total profit or loss of a stock of your choice over five days for a person who owns 1,000 shares.

**Step Two: Design**

It's time to design your program by writing pseudocode. Your outline must include the following elements:

* Input statements
  + Ask the user for at least three numeric values.
  + Show proper use of the int() and float() functions.
  + Calculations required to achieve correct output.
  + Use proper order of operations.
  + Use any appropriate math functions.
  + Output statements
  + Create clear and well organized output to share the data and results of the calculations.
  + Show proper use of the str() function.

Insert your pseudocode here:

Comment the project number, the date, my name, grade, and purpose of the program.

Then,

Set the important variables as global before starting any main functions in the code.

Set Global variables as:

- SystemName (For dynamically naming the program using a var),

- CurMD (For letting the program view in which mode it should be running in)

- spEVENT (For setting up special and non- interfering easter eggs) DEPRECATED

Afterwards,

Import important functions such as math

Set the animation function that will be used to animate the programs string output.

After,

Set the def ***cubeSequal()*** which will run most of the program’s code.

In def cubeSequal() \_ (Called cubeSequal, because it shares a small portion of the code found in the cube() of last project )

Import time (To set the time.sleep() function)

Print to the user that the program is starting and make sure to advert the user to not click on run program more than 1 time in a row, or the flush and other functions from the typeAnim() will misbehave.

Start the dialogue that explains the user a little bit about the program.

Use a product cart to save what the products data made from user’s input. DEPRECATED02

Next,

Add the productWizard(), that its purpose is to to handle the information of the products set by the user’s input. (e.g. Damaged Keyboard, Category 5, 3.99)

Add more dialogue, and set the most important global Arrays such as, *prs[]* The products themselves, *prCategories[]* the categories (which is completely independent and can be modified for addition of even more categories in the code.), *cost[]* which will contain the user’s input set for the price of their products, *totalCost[]* which will contain the total cost of each product plus taxes (also known as tax()), *tax()* The tax will be dependent to whatever the user’s input will set the tax’s percentage value to, *taxPRS[]* will contain the tax given to each product (such data will be made through the use of math functions such as /, \*, etc.), *shippingVal[]* DEPRECATED03 , and, *idPR[]* that would contain the category id selected and inputted by the user (e.g. User selects 3 – which would be Office).

Set important variables which purpose is to avoid the for loops and while loops to misbehave, plus, to keep tabs on which round the current loop would be in.

i()=0 purpose would be to set the start value of range - for the for loop, and *f()=9* was

DEPRECATED04.

Set the global variable that would be known as *iPR()* – its purpose would be to store the int value of the input which would ask for how much products the user would want to make.

Set the variable *lpZoom()* – its purpose to check on which round the for loop would be in .

After,

Start for loop that depends on *i()* and, *iPR* as its range.

Continue dialogue, which in this case, an input function that will ask for the user the products name (lpZoom() var, in this case, will show which product the user will be naming), and use the *typeAnim(.)* function to print to the user to in which category the user’s product would like to be in.

Start the child for loop that will enumerate the categories available in the program’s code. Such information will be printed in the following form: 0. Catgeory.

Finish for Loop to then,

Make a system that will set to the idPR() var, the users’ input set to select the category the user would like their product to be in.

Ask for which category the user would like to select (e.g. The user would input the int value of 3, 3 would match for Office.), such would be asked with the input function, the input data will be saved on the temCateg() variable (a temporal variable the temporary store the int value data).

End the child for loop

After,

in the for loop, create a failsafe system made for an IF function which purpose is to reset a portion of the code in order to: if the user inputs a wrong input, the program will as a failsafe revert to the last for loop. Inside the failsafe system, inform the user that they have inputted a wrong value and that the program will revert some of its data to comply the users’ mis-action.

After,

In the for loop,

Ask for how much would the user set the product’s price into, such will be asked using an input function and stored to cost[lpzoom] array (lpzoom will be used to index in the cost[] array, to save the price value in the correct position that later will be used to merge its data to align with the products’ data).

If the user has more products left to name: The program will continue the dialogue by using the typeAnim function, which would print and inform the user that the program is heading to ask the user the data for the next product.

End For Loop

Once the user has finished inputting all of the product’s data, the program will set the *curMD()* variable to = 2 (Which will be used to inform early to the program for it to proceed towards the next function).

After,

Continue the dialogue by informing the user to set the tax addition percentage value though the typeAnim() function.

Ask the user using the input function the tax percentage value, such value will be stored in the *tax()* variable as a float value.

Lastly,

In the ***cubeSequel()*** parent function, start a new child function *receiptSim()*, which purpose is to align most of the programs important data for then later print out a fictitious receipt which contains the user products’ data (e.g. the products name, cost and category).

Next,

Continue dialogue by welcoming the user to the next part of the program, the welcome print message will be made using the typeAnim() function.

After,

Set the priority global arrays and variables: *Cart[]* which will contain the cart index data DEPRECATED05, *ItemQ[]* which will contain the Item Quantity Data (This will store the amount of how many items of the same product the user would like to buy in its exact index.), *paymentString[]* is a string array which contains the exact output the user will view (paymentString[] will contain the product’s name, cost and category data), *cartMode()* variable will be used to avoid the program from confusing if the user wants to buy more products or not (e.g. If cartMode = 2, the ~~for~~ or **while** loop will break), *subtotal()* variable will carry the subtotal of all of the products not including taxes, and, *lp()* variable will count the amount of repetitions the while loop will run.

After,

Start a while loop, which will not stop until when cartMode() doesn’t equal to 2.

In the while loop,

Continue dialogue by asking the user what item they would like to buy, by printing through the typeAnim() function, and, with the same purpose by asking though an input function that will store the value in the *item()* var.

Start a child for loop that will showcase the items available for the user to buy (such will use a minimum range as iPR+1 and will count the array prs[], this to not exceed from the amount of items bought and the products available.)

End the child for loop.

After, in the while loop,

Continue the dialogue by asking how much of the same product the user would like to buy; this will be printed using the typeAnim function. Afterwards using the input function, the program will store the value inputted by the user previously in the *itemQD()* var (itemQD() will be a temporal variable, storing temporally the data inputted by the user which is about how many times the user would like to buy the same product.). Set *itemQ[item]* to *itemQD()* (this for setting the inputted data previously to the exact index in the array itemQD[] which will contain how much of the same products the user wants to buy, this array will contain each product repetitions payment data though an array.).

After, in the while loop,

Ask using the typeAnim() function and input function if the user wants to buy something else using the *userBoolean()* var (Such will store the inputted value which only have the Y / N input options, which will end or continue the while loop).

Before the Y/N segment,

Do the respective code that will measure and calculate the tax addition percentage cost to each product set by the user.

Set the global vars and local vars for the Tax calculation segment,

n() being a temporal var containing the *float(cost[item])* (Which purpose is to obtain the cost of the current product for future calculations), *nt()* var containing *float(tax)* (for converting the tax percentage to a float value), *taxC()* containing *n \* (nt / 100.00)* (which calculates the tax for the current product using the *cost \* tax/100 equation*). totalCost[item] containing float(taxC) + float(cost[item]) (which adds the value of TaxC() and the cost of the current product), *qd()* var containing *float(totalCost[item]) \* float(itemQ[item])* (multiplying the amount of times the user settled to buy the same product), *subtotal()* containing and appending the amount of *qd()* through += (appending in the while function, means in which round the previous amount of qd() var will be added to the new value of qd() and stored in subTotal() var), *ctd()* containing *int(idPR[item])* (converts to int the category value set for the current product) , psip() containing *str(itemQ[item]) + " " + str(prs[item]) + " | $" + str(qd) + " | "}* (Which will contain temporarily the product’s name and cost, all syntaxed in a way that should look similar to this: *Item $0.00* ), and, *paymentString[lp]* containing *psip + prCategories[ctd]* (concatenates the previous string value and adds to which category the current product is part of).

After the user selects whether to continue buying or not,

Start the receipt render segment,

Continue by setting *taxTotal()* var to *subTotal \* (tax / 100)* (which multiplies the subtotal with the tax percentage, the tax percentage is calculated by the same equation which is *n/100*), and, setting *total()* var to *taxTotal + subTotal* (adds the taxTotal() int var with the subTotal in var, which adds the total of the taxes pending to pay plus the subtotal off all the products set to buy.)

Finally, render the receipt and organize the data to Name, cost, category and subtotal, tax, and total.

Finalize,

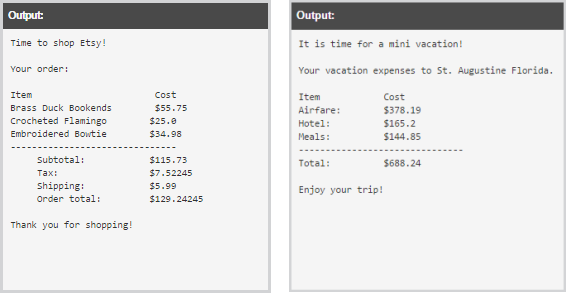
By ending all parent functions.

**Step Three: Coding**

Use the following guidelines to code your program:

* 1. To code the program, use the Python IDLE.
  2. Using comments, type a heading that includes your name, today's date, and a short description of the program.
  3. Follow the Python style conventions regarding indentation and the use of white space in your program.
  4. Use meaningful names for all variables.

**Example of expected output:**The output for your program should resemble the following screenshot. Your specific results will vary, depending on the choices you make and the input provided.



**Insert a copy of your code from the IDLE here**:

# Assignment 02.06 Project

# Date: Wed 27/01/2025 #2025 already!

# Student: modeoiz

# Grade: 11th

# Purpose: Calculating prices with a small twist.

# GLOBAL V A R S

global text #the text var for the typing animation

text = "test text texting"

global systemName

systemName = "Buy stuff in a Captalist Society Assistant by Simulating a Recepit Simulator!"

global curMD

curMD = 1

global spEvent #For the posibilty of special events once the user does an incorrect input

spEvent = 0

import math

def calVolume(side\_length): #calculates volume

return pow(side\_length, 3) #l^3

def calSurfArea(side\_length): #calculates well the surface area lol

return 6 \* pow(side\_length, 2)

#animates text well it is called first then it animated the text :D

def typeAnim(text, speed=0.1):

#imports time and this new system thing i NEVER knew about.

import time

import sys

for char in text: #char means characters

sys.stdout.write(char) #stdout for altering the output directly atleast its behaviour

sys.stdout.flush() #i searched what is flush and well its a really technicall thing ensuring that data is written from internet buffer to its destination correctly and immediately.

time.sleep(speed) #basically it will wait for the specified amount of time the code will set once the parent function is called.

def cubeSequel(): #Recognize the code? :D I reused some old code and made newer code.

import time

import math #very important for sqrt and other functions related to math :)

typeAnim("Once the text starts printing, please WAIT until that process finishes. If not IDLE will behave oddly...", speed=0.04)

time.sleep(1.0)

#D I A L O U G E

#Disclaimer

print("")

typeAnim("Program will store data only per session.", speed=0.03)

print("")

time.sleep(1.0)

typeAnim("Welcome to " + systemName , speed=0.03)

print("")

time.sleep(1.0)

x = 3

# S Y S T E M

#end of product storage

#def productCart():

#end of product cart

# c a l l

def productWizrd():

print("")

typeAnim("Let's make the products for your fake receipt :D", speed=0.04)

print("")

time.sleep(1.0)

typeAnim("...", speed=1.0)

#productStore Vars

global prs

prs = ["Free Gift (Receipt)","Product1","Product2", "Product3", "Product4", "Product5", "Product6", "Product7", "Product8", "Product9",]

global prCategories

prCategories = ["Other", "Tech & Software", "Food", "Office", "Drinks", "Licor", "Enterntainment", "Drug / Medicine", "Artesanal", "Unspecified"] #Categories Available, You can change this :D, And add how many ctageories you want!

global cost

cost = [0,0,0,0,0,0,0,0,0,0]

global totalCost

totalCost = [0,0,0,0,0,0,0,0,0,0]

global tax

tax = 2.7

global taxPRS

taxPRS = [0,0,0,0,0,0,0,0,0,0]

global shippingVal

shippingVal = 1.2

global idPR

idPR = ["0","0","0","0","0","0","0","0","0","0"] #IDs for each profuct containg each index that wil be compiled in to showcase its final values.

#Process of Creation of Products

i = 0

f = 9 #on what index it should end

global iPR

iPR = int(input("How many products you wish to make (Max 10) "))

time.sleep(0.5)

lpZoom = 0

for i in range(0, iPR):

lpZoom += 1

print("")

typeAnim("Name the product" , speed=0.02)

print("")

prs[lpZoom] = input("Name product " + str(lpZoom)) #:D Saves the product name to its corresponding oject index in the array

time.sleep(1.0)

temStr = prs[lpZoom]

typeAnim("Alright. In what category would your product, " , speed=0.02)

print(temStr)

typeAnim("be in?" , speed=0.04)

time.sleep(1.0)

print("")

time.sleep(1.0)

for index, val in enumerate(prCategories):

print(" {}: {}".format(index, val))

time.sleep(0.05)

temCatg = input("Select from the 5 categories mention below, by its corresponding number. eg: 2, for (Food)")

time.sleep(0.05)

failsafeA = len(prCategories)

failsafeA = int(failsafeA)

temCatgFS = int(temCatg) # converts catgory string into input for the if statement

#print(failsafeA , temCatgFS) #debug

if temCatgFS > failsafeA:

print("")

typeAnim("That isn't a real category :)" , speed=0.04)

print("")

time.sleep(1.0)

print("")

typeAnim("Reverting Data." , speed=0.03)

time.sleep(0.5)

print("")

typeAnim("...", speed=1.5)

time.sleep(0.1)

print("")

productWizrd()

typeAnim("Now give a price for your product." , speed=0.02)

time.sleep(1.0)

cost[lpZoom] = input("Set a price for your product, " + prs[lpZoom] + " Any decimal and natural int values are admitted.")

# A S S I G N ID for product

time.sleep(0.05)

print("")

idPR[lpZoom] = temCatg #itemCatg (Category)

#print(idPR[lpZoom]) #test to check if the id was succesfully set

print("")

#print(lpZoom, iPR) #DEBUG | To check the current loop turn or index

if lpZoom < iPR:

typeAnim("Alright, Let's head to fill the details of the next product." , speed=0.02)

time.sleep(1.0)

curMD = 2

#print(curMD) #more dubug print functions

print("")

typeAnim("Now," , speed=0.02)

time.sleep(0.5)

curMD == 2

if curMD == 1:

productWizrd()

# SETUP T A X E S C O S T

print("")

typeAnim("Set the percentage for the Tax Addition. (Just set the number, DO NOT add the % symbol.) A percentage like 1 would be the ideal tax." , speed=0.04)

print("")

time.sleep(1.0)

tax = float(input("Set the percentage for the Tax Addition. (Just set the number, DO NOT add the % symbol.) "))

def receiptSim():

#print(prCategories)

print("")

typeAnim(". . ." , speed=0.8)

print("")

time.sleep(1.0)

print("")

typeAnim("Welcome! " , speed=0.05)

time.sleep(1.0)

global cart

cart = [0,0,0,0,0,0,0,0,0,0]

global itemQ

itemQ = [0,0,0,0,0,0,0,0,0,0]

global paymentString

paymentString = ["Receipt (Free)","1","2","3","4","5","6","7","8","9"]

global cartMode

cartMode = 1 #1 for yes

global subTotal

subTotal = 0

global lp

lp = 1

#for i in range(0, 9):

while cartMode != 2: #loops until cartMode matches 2

#print("works") #debug thing

#IF STATEMENTS for channeling final modes

if cartMode == 1:

time.sleep(0.5)

print("")

iPRFix = iPR - 1

#for index, val in enumerate(prs):

print("")

typeAnim("What product would you like to buy?" , speed=0.05)

print("")

for index in range(min(iPR + 1, len(prs))): #in order to avoid th eprogram from reapiting the loop from how many items are or doing less than the items available.

iPRtem = prs[index] #ipr is temporal var fro the print statement below

print(" AVAILABLE {}: {}".format(index, iPRtem ) + " $"+str(cost[index]))

time.sleep(0.5)

item = int(input("Which item would you like to buy? (eg: 2 for Bread)"))

cart[item] = cart[item] + 1

#print(cart[item]) debug,check if works

print("")

typeAnim("Sounds like a great choice! How much " , speed=0.05)

print(prs[item] + ", ")

print("would you like to buy?")

time.sleep(1.0)

time.sleep(0.5)

itemQD = int(input("How much " + prs[item] + ", would you like to buy? (Write only the amount, eg: 2 for 2 apples))"))

itemQ[item] = itemQD #item Quantity Data

time.sleep(0.5)

print("")

typeAnim("Are there any more things you want to buy?" , speed=0.05)

time.sleep(1.0)

typeAnim(" [ Y / N ]" , speed=0.04)

global userBoolean

userBoolean = "U" #as for unknown

#T A X CALCUALTION | calculates item and adds tax amount

global taxC

n = float(cost[item]) #temporal var for cost and index item of array

nt = float(tax) # temporal var for float

taxC = n \* (nt / 100.00) #tax is calculated by dividing n with 100

taxPRS[item] = taxPRS[item] #saves the tax value to the array for later calculation :D

totalCost[item] = float(taxC) + float(cost[item])

# tempvars

qd = float(totalCost[item]) \* float(itemQ[item]) #quantity data wich multiplies the amount of times the user wants to buy the same product with the product price

subTotal += qd

ctd = int(idPR[item]) #category data

psip = str(itemQ[item]) + " " + str(prs[item]) + " | $" + str(qd) + " | " #payment string index preview

paymentString[lp] = psip + prCategories[ctd]

#print(paymentString[item]) #for dubug if the string that showcases everything works

time.sleep(0.5)

userBoolean = input("Are there any more things you want to buy? (Y for Yes, N for no)")

if userBoolean in ["N", "n"]: #breaks loop if user has bought everything they want.

cartMode = 2

break

elif userBoolean in ["Y","y"]:

cartMode = 1

#print("user pressed yes") #debug

lp += 1

#elif cartMode == 2:

#break

# FINALLY THE , R E C E I P T

#for num in cost: #adds every products cost into 1

# subTotal += cost(num)

#subTotal = float(sum(cost))

taxTotal = subTotal \* (tax / 100)

total = taxTotal + subTotal #adds both for the total of the receipt

print("")

time.sleep(1.0)

typeAnim("...", speed=1.2)

#important calculations (for taxes, subtotal)

time.sleep(0.5)

# receipt render

print(" ")

typeAnim("Your Order: ", speed=0.06)

print(" ")

print(" ")

typeAnim(" Item Cost Category ", speed=0.05)

print(" ")

for index in range(min(iPR + 1, len(prs))): #in order to avoid th eprogram from reapiting the loop from how many items are or doing less than the items available.

iPRtemB = paymentString[index] #ipr is temporal var fro the print statement below

print(" {}".format(iPRtemB) + " ")

print(" ")

#print(str(lp) + " " + str(iPR)) #debug var

typeAnim("-----------------------------------------------", speed=0.03)

print(" ")

print(" Subtotal: $" + str(subTotal))

print(" Tax: $" + str(taxTotal))

print(" Order Total: $" + str(total))

print(" ")

typeAnim("We appreciate You for shopping with us!" , speed=0.05)

print(" ")

receiptSim()

typeAnim("PROGRAM HAS ENDED" , speed=0.05)

print(" ")

typeAnim("\ (•◡•) / " , speed=0.05)

cubeSequel()

#hope you had fun testing the code!

**Step Four: Testing**

Run your code, and evaluate the output. Then, answer the following questions in the testing chart. Use two to three meaningful sentences to answer each question.

|  |  |
| --- | --- |
| Testing Question | Response |
| What bugs did you identify in your code? | I identified an excruciating amount of bugs in my code, proving the fact that codes are an often case.  One of the most painful bugs to fix was the for loops not working all that because of leaving the 2 break functions, which the second was made completely accidental.  Another worth to mention was the bug of the code not being able to add all of the product’s cost, which took also hours to fix. |
| How did you fix the bugs? | By running the program and checking where in the output the bug may be located, afterwards I link the erroneous output with a line of the actual code and try to fix it.  Not always the bugs got resolved on the first try, when that happens, I re-write some parts of the code in order to then later delete the malfunctioning code. If that didn’t work as a last resort I deprecate a part of the code completely and try to go around it (such happens usually because of some of the limitations found on the IDLE, which are easy to find alternatives to, like I used the while function because there wasn’t the possibility to import a wait until program) |

**Step Five: Maintenance**

Passionate programmers strive to improve their code! In two to three meaningful sentences, answer the following questions in the maintenance chart to consider the next steps of your program.

|  |  |
| --- | --- |
| Maintenance Question | Response |
| What design and functionality improvements could you make to your program? | Remove the unnecessary parts of the code and find better ways to optimize the overall program.  That would be achievable once I understand more how python functions. |
| How can you get feedback on ways to improve your program? | By showcasing the program to other individuals, usually by the different patterns of resonating, unconsciously the other individuals may find an obscure bug in your program.  This is found in so many examples, like when an OS has this specific bug that occurs once doing that specific thing.  Or in a simple program when the user tries a igger int value, or something related to testing the limits of the program by inputting unrealistic data (e.g. 99999 products, the cost of the Sun being 99999999999, I want to buy the product 8893 times, ect. ) |
| How can you expand your program into a new, better program in the future? | By optimizing the code and reusing it for different purposes, which is possible in so many ways. |
| What are potential bugs that users may possibly encounter if your program is expanded into a new program in the future? | That would depend in the way the program’s code was written, the overall new purpose of the program and the limitations found on the programming environment or language.  A potential bug could be in the input or the for loops or anything related to calculations. |