**Python Journal Template**

**Directions:** Follow the directions for each part of the journal template. Include in your response all the elements listed under the Requirements section. Prompts in the Inspiration section are not required; however, they may help you to fully think through your response.

Remember to review the Touchstone page for entry requirements, examples, and grading specifics.

**Name: William Lee**

**Date: 1/7/2022**

**Final Replit Program Share Link:**

**https://replit.com/@blee56/WorldlyHandmadePagerecognition#main.py**

**https://replit.com/@blee56/WorldlyHandmadePagerecognition#taxcalc.py**

Complete the following template. Fill out all entries using complete sentences.

PART 1: Defining Your Problem

**Task**

State the problem you are planning to solve.

**Requirements**

• Describe any input data you expect to use.

• Describe what the program will do to solve the problem.

• Describe any outputs or results the program will provide.

**Inspiration**

When writing your entry below ask yourself the following questions:

• Why do you want to solve this particular problem?

• What source(s) of data do you believe you will need? Will the user need to supply that data, or will you get it from an external file or another source?

• Will you need to interact with the user throughout the program? Will users continually need to enter data in and see something to continue?

• What are your expected results or what will be the end product? What will you need to tell a user of your program when it is complete?

<Write your journal entry response here>

One problem that I run into in my routine daily life is budgeting for federal taxes. In order to properly budget for my taxes I need to gather the effective federal tax rate for my family. The calculation is not as simple as multiplying your tax rate by your total salary because if your annual income spans multiple tax brackets you would have to tax the income you make in that bracket at the rate of that bracket until you exceed that bracket and move up into the next bracket with a higher rate. Due to this complexity I would like to build an application that calculates this information for me. A person’s tax rate depends on several inputs:

1. How the individual is filing their taxes? Are they filing as a single or are they married and filing jointly.

A. This determines your standard deduction. I will not get into itemizing but that would be a good thing for the future.

B. This information will be provided by the user.

2. The amount of money the person or family makes also plays a factor into the effective tax rate.

A. This determines your marginal tax rate and tax bracket.

B. This information will be provided by the user.

3. The tax brackets for each filing type and income. Will use a dictionary to hold this information.

A. This data will be a part of the program.

4. Standard deductions for both the single filing status and the married filing jointly status.

Some of the outputs of this program will be the following:

1. The amount of money owed in federal taxes.

2. The users average tax rate.

3. The users tax bracket.

PART 2: Working Through Specific Examples

**Task**

Write down clear and specific steps to solve a simple version of your problem you identified in Part 1.

**Requirements**

Complete the three steps below **for at least two distinct examples/scenarios**.

• State any necessary input data for your simplified problem.

• Write clear and specific steps in English (not Python) detailing what the program will do to solve the problem.

• Describe the specific result of your example/scenario.

**Inspiration**

When writing your entry below ask yourself the following questions:

• Are there any steps that you don’t fully understand? These are places to spend more time working out the details. Consider adding additional smaller steps in these spots.

• Remember that a computer program is very literal. Are there any steps that are unclear? Try giving the steps of your example/scenario to a friend or family member to read through and ask you questions about parts they don’t understand. Rewrite these parts as clearly as you can.

• Are there interesting edge cases for your program? Try to start one of your examples/scenarios with input that matches this edge case. How does it change how your program might work?

<Write your journal entry response here>

The application will allow the user to start over using a while loop to continue inputing information.

Gather the users filing type (single vs. married filing jointly). Ensure that they type the correct option. Using input validation techniques (try catch)

Gather the users total family income. If married filing jointly this will include their spouses income. Ensure that they type a numeric value. (Try catch)

Create a dictionary containing the tax rates for both filing types. Embedded in the code.

Function 1: Output the users amount due in federal taxes:

Inputs include total income and filing type

To begin we would subtract the standard deduction. Use the tax bracket thresholds to determine the amount of money to be taxed at the different rates. First test if the total income is less that the threshold for the second to last tax bracket, since the last bracket is 0. If this is true then the function will multiply the total income by the lowest tax rate depending on the filing type. If the first case is not true then we will determine the highest tax bracket using the user’s input for filing type and total income and comparing it against the tax bracket dictionary. Once the highest tax bracket is found we will subtract the key: value from the total income to get the amount within that tax bracket. This remaining amount will be multiplied by the key, which is the tax rate for this bracket. This amount will be stored in the value name tax\_amount. The remaining amount will then be subtracted from the income variable. This value will always match a tax bracket value. Before leaving this elif block we will store the key value in a variable called previous\_key\_value. Because of this I will build the next elif statement for when the income equals the key value.

Multiplying the amount within that bracket by the marginal tax rate of the bracket. We will subtract the new key value from the previous key value to obtain the taxable amount in this bracket, since starting value was higher than this bracket all of the income in this bracket will be taxed by the brackets rate. This taxable amount will be added to the tax\_amount variable. We will store this key value in the previous\_key\_value and continue to traverse all the way down the data structure until we reach the bottom where 0 will be added to the tax amount!

Function 2: Output the users amount due in state taxes:

Similar to the federal tax calculation but filters down off of the state selected by the user for that states individual tax brackets. The state being tested will be my state of residence Alabama, but TN is in there for testing too since they have a 0% state income tax.

Function 3: Output the users average tax rate:

Take the amount due in federal taxes divided by the total income. This will take the result of function 2 as an input. Return this amount.

Function 4: Output total taxes (state and federal):

Takes the output values from the function 1 and function 2 and adds them together.

PART 3: Generalizing Into Pseudocode

**Task**

Write out the general sequence your program will use, including all specific examples/scenarios you provided in Part 2.

**Requirements**

• Write pseudocode for the program in English but refer to Python program elements where they are appropriate. The pseudocode should represent the full functionality of the program, not just a simplified version. Pseudocode is broken down enough that the details of the program are no longer in any paragraph form. One statement per line is ideal.

**Help with writing pseudocode**

• Here are a few links that can help you write pseudocode with examples. Remember to check out part 3 of the Example Journal Template Submission if you have not already. Note: everyone will write pseudocode differently. There is no right or wrong way to write it other than to make sure you write it clearly and in as much detail as you can so that it should be easy to convert it to code later.

• https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/

• https://www.wikihow.com/Write-Pseudocode

**Inspiration**

When writing your entry below ask yourself the following questions:

• Do you see common program elements and patterns in your specific examples/scenarios in Part 2, like variables, conditionals, functions, loops, and classes? These should be part of your pseudocode for the general sequence as well.

• Are there places where the steps for your examples/scenarios in Part 2 diverged? These may be places where errors may occur later in the project. Make note of them.

• When you are finished with your pseudocode, does it make sense, even to a person that does not know Python? Aim for the clearest description of the steps, as this will make it easier to convert into program code later.

<Write your pseudocode here>

Import <filename> #filename containing classes

Dictionary containing tax brackets{

AL: {

Single: {rate: threshold, rate: threshold},

Married filing jointly: {rate: threshold, rate: threshold}},

TN: {

Single: {rate: threshold, rate: threshold},

Married filing jointly: {rate: threshold, rate: threshold}}

}

Function using parameters to output federal tax amount(filing type, income)

tax\_amount = 0

For key in dictionary:

If income > dictionary[key]:

remainder = income - dictionary[key]

tax\_amount = remainder \* key

income = income - remainder

Previouskeyvalue = dictionary[key]

Else:

Taxable amount = dictionary[key] - income

tax\_amount += taxable\_amount \* key

Function using parameters to output state tax amount(filing type, income, state)

For key in dictionary[state]:

If income > dictionary[key]:

remainder = income - dictionary[key]

tax\_amount = remainder \* key

income = income - remainder

Previouskeyvalue = dictionary[key]

Else:

Taxable amount = dictionary[key] - income

tax\_amount += taxable\_amount \* key

Function for average tax rate(tax\_amount, total\_income)

Return tax\_amount/total\_income

Function for total including sate and federal taxes:

total\_tax = State tax amount + federal tax amount

Return total\_tax

Def gather\_input

Gather input for the users filing type

Check if input matches available options

Gather input for the users income

Use exception handling to make sure the value is numeric

Gather input on the users state

Use conditional logic to make sure the value exists within the dictionary containing the state tax brackets

game\_over = false

while(game\_over){

Print menu of options

1. Calculate federal tax

A. Subtask to gather average tax rate

2. Calculate state tax

3. Calculate total tax

If federal tax:

filing type, total\_income = Gather\_inputs()

Federal tax function (filing type, total income)

Input(“Do you want to know your average tax rate?)

If yes:

print(average tax function(tax\_amount, total\_income)

Elif state tax:

Filing type, total\_income, state = gather\_inputs()

State tax function (filing type, total\_income, state)

Elif total tax:

Filing type, total\_income, state = gather\_inputs()

total\_tax = Federal tax function (filing type, total income) + State tax function (filing type, total\_income, state)

Elif Quit:

game\_over = True

Else:

print(“You didn’t select a valid option)

PART 4: Testing Your Program

**Task**

While writing and testing your program code, describe your tests, record any errors, and state your approach to fixing the errors.

**Requirements**

• For at least one of your test cases, describe how your choices for the test helped you understand whether the program was running correctly or not.

For each error that occurs while writing and testing your code:

• Record the details of the error from Replit. A screenshot or copy-and-paste of the text into the journal entry is acceptable.

• Describe what you attempted in order to fix the error. Clearly identify what approach was the one that worked.

**Inspiration**

When writing your entry below ask yourself the following questions:

• Have you tested edge cases and special cases for the inputs of your program code? Often these unexpected values can cause errors in the operation of your program.

• Have you tested opportunities for user error? If a user is asked to provide an input, what happens when they give the wrong type of input, like a letter instead of a number, or vice versa?

• Did the outcome look the way you expected? Was it formatted correctly?

• Does your output align with the solution to the problem you coded for?

<Record your errors and fixes here>

KeyError if filing type wasn’t spelled correctly. Built a validation method to ensure that the program would continue until an accurate value was input into the program.

TypeError if you did provide a numeric value. I used a function to check if the value was numeric. If it was we continue. If it wasn’t then it would throw them to a try catch block where they got one more chance before the program would throw a custom exception and quit the program.

def gather\_inputs(tax\_type):

valid\_entries = False

filing\_type = input("Please input your filing type either single or married filing jointly. Be sure to type your selection correctly. \n").lower()

while valid\_entries == False:

#Conditional logic for filing type

if filing\_type == "single" or filing\_type == "married filing jointly":

#Exception handling block for numeric value

try:

total\_income = float(input("Please provide your income in numeric form: "))

#Conditional logic to determine which calculator is being used.

if tax\_type == "state":

state = input("Enter the abbreviation for your state: ").upper()

valid\_selection = False

while valid\_selection == False:

#Ensuring that the data for the selected state is in the current data structure

if state in available\_states:

#exits the while loop

valid\_selection = True

else:

print("You did not enter state abbreviation that is in our system.")

#Keeps asking for the same input value until a valid selection is made

state = input("Please enter a different state abbreviation: ")

#exits the while loop determined by valid\_entries for state tax calculator

valid\_entries = True

elif tax\_type == "federal":

#exits the while loop for the federal tax calculator

valid\_entries = True

#Catches all scenarios where the user did not enter a number

except ValueError:

print("You did not enter a number. ")

#catches all scenarios where the user does not select single or married filing jointly

else:

print("Invalid filing type. Try again!")

filing\_type = input("Please input your filing type either single or married filing jointly. Be sure to type your selection correctly. \n").lower()

#The conditional varies the outputs depending on the calculator selected.

if tax\_type == "federal":

return filing\_type, total\_income

elif tax\_type == "state":

return filing\_type, total\_income, state

TypeError if you had a negative value in the standard deduction logic. Instead of converting the input for income to an int I changed the conversion to a float.

Error came from this.

if (self.filing\_type == "single"):

temp\_income = self.income - 13850

elif (self.filing\_type == "married filing jointly"):

temp\_income = self.income - 27700

#Dictionary traversal logic

for key in applicable\_fed\_tax\_brackets:

#Handles negative values after standard deduction logic

if(temp\_income < 0):

print("Your standard deduction eliminated your need to pay taxes. ")

break

I fixed it here within the gather\_inputs() function:

try:

total\_income = float(input("Please provide your income in numeric form: "))

#Conditional logic to determine which calculator is being used.

if tax\_type == "state":

state = input("Enter the abbreviation for your state: ").upper()

valid\_selection = False

while valid\_selection == False:

#Ensuring that the data for the selected state is in the current data structure

if state in available\_states:

#exits the while loop

valid\_selection = True

else:

print("You did not enter state abbreviation that is in our system.")

#Keeps asking for the same input value until a valid selection is made

state = input("Please enter a different state abbreviation: ")

#exits the while loop determined by valid\_entries for state tax calculator

valid\_entries = True

elif tax\_type == "federal":

#exits the while loop for the federal tax calculator

valid\_entries = True

#Catches all scenarios where the user did not enter a number

except ValueError:

print("You did not enter a number. ")

I had some errors in the logic of my tax amount function that wasn’t allowing it to traverse through all of the levels appropriately. I needed to have the next key prior to getting to it. So instead of using a dictionary where the values ascended I switched to a a dictionary descended.

self.taxbrackets\_des = {

"single": {.37: 578125, .34: 231250, .32: 182100, .24: 95375, .22: 44725, .12: 11000, .10: 0},

"married filing jointly": {.37: 693750, .34: 462500, .32: 364200, .24: 190750, .22: 89540, .12: 22000, .10: 0}

}

Initially, the dictionary was in ascending order. It was just easier to reverse it.

I tested the output of the Federal tax calculator piece of the project and found that my math wasn’t quite adding up. No exception was being thrown so I went through the code for the federal tax calculator line by line. Instead of having a ‘-‘ on one of the lines I had an “=“ sign which was causing the logical error. I caught the math error by comparing it to the online calculator I was using to test my math at https://www.bankrate.com/taxes/quick-tax-rate-calculator/The values do not match 1 for 1 with this website because they are using the 2022 numbers rather than the adjusted 2023 numbers.

PART 5: Commenting Your Program

**Task**

Submit your full program code, including thorough comments describing what each portion of the program should do when working correctly.

**Requirements**

• The purpose of the program and each of its parts should be clear to a reader that does not know the Python programming language.

**Inspiration**

When writing your entry, you are encouraged to consider the following:

• Is each section or sub-section of your code commented to describe what the code is doing?

• Give your code with comments to a friend or family member to review. Add additional comments to spots that confuse them to make it clearer.

<Copy your full program code here, including comments>

filename: main.py

# importing from class file

import taxcalc

# importing math module for round function.

import math

#Used to validate state options later

available\_states = ["AL", "TN"]

#Function to gather all of the necessary inputs and to type check these inputs until valid inputs are received

def gather\_inputs(tax\_type):

valid\_entries = False

filing\_type = input("Please input your filing type either single or married filing jointly. Be sure to type your selection correctly. \n").lower()

while valid\_entries == False:

#Conditional logic for filing type

if filing\_type == "single" or filing\_type == "married filing jointly":

#Exception handling block for numeric value

try:

total\_income = float(input("Please provide your income in numeric form: "))

#Conditional logic to determine which calculator is being used.

if tax\_type == "state":

state = input("Enter the abbreviation for your state: ").upper()

valid\_selection = False

while valid\_selection == False:

#Ensuring that the data for the selected state is in the current data structure

if state in available\_states:

#exits the while loop

valid\_selection = True

else:

print("You did not enter state abbreviation that is in our system.")

#Keeps asking for the same input value until a valid selection is made

state = input("Please enter a different state abbreviation: ")

#exits the while loop determined by valid\_entries for state tax calculator

valid\_entries = True

elif tax\_type == "federal":

#exits the while loop for the federal tax calculator

valid\_entries = True

#Catches all scenarios where the user did not enter a number

except ValueError:

print("You did not enter a number. ")

#catches all scenarios where the user does not select single or married filing jointly

else:

print("Invalid filing type. Try again!")

filing\_type = input("Please input your filing type either single or married filing jointly. Be sure to type your selection correctly. \n").lower()

#The conditional varies the outputs depending on the calculator selected.

if tax\_type == "federal":

return filing\_type, total\_income

elif tax\_type == "state":

return filing\_type, total\_income, state

#Menu options to be printed later

options = ["FEDTAX: Calculate federal tax amount and marginal tax rate",

"STATETAX: Calculate state tax amount",

"TOTALTAX: Calculate your total tax amount both federal and state",

"QUIT: To quit the application"]

# General intro to application

print("Welcome to the tax calculation app \n----------------------------------- ")

#While loop for the whole program. Lets the user continue to enter values

game\_over = True

while game\_over:

#Printing options

print("Please select from one of the following options.\n------------------------------------------------")

for option in options:

print(option)

print("Input the portion of each option in all caps.")

#Menu selector

selection = input("Enter your selection below: \n").upper()

#Different scenarios based on the option selected.

if selection == "FEDTAX":

#Calling function to gather required parameters for fedtaxcalc

filing\_type, total\_income = gather\_inputs(tax\_type="federal")

#Instantiation

fedtaxcalc = taxcalc.FederalTaxCalculator(filing\_type=filing\_type, total\_income=total\_income)

#calling method to receive federal tax amount

tax\_amount = fedtaxcalc.getFedTaxAndRate()

formatted\_tax\_amount = math.round(tax\_amount, 3)

print(f"Your total federal tax owed is ${formatted\_tax\_amount}.")

#Nested Logic for Average Tax Rate

selection\_two = input("Would you like to know your average tax rate? Enter Y for yes or N for no: ").upper()

#Used to break the while loop later

valid\_selection = False

#While loop to valid response

while valid\_selection == False:

if selection\_two == "Y":

average\_rate = fedtaxcalc.getAverageRate()

print(f"Your average tax rate is {average\_rate}%")

valid\_selection = True

elif selection\_two == "N":

valid\_selection = True

#Catches all values aside from Y or N

else:

print("You entered an invalid input")

#Logic if you just want to know the amount owed in state taxes

elif selection == "STATETAX":

#Calling method to gather necessary parameters for statetaxcalc

filing\_type, total\_income, state = gather\_inputs(tax\_type="state")

#Instantiation

statetaxcalc = taxcalc.StateTaxCalculator(filing\_type=filing\_type, total\_income=total\_income, state=state)

#object method to get the state tax amount

state\_tax\_amount = statetaxcalc.getStateTaxAmount()

format\_state\_amount = math.round(state\_tax\_amount, 3)

print(f"Your total state tax owed is ${format\_state\_amount}")

#Logic for the total value owed in state taxes

elif selection == "TOTALTAX":

#Using the state logic within gather\_inputs() because the state tax needs to be gathered

filing\_type, total\_income, state = gather\_inputs(tax\_type="state")

#I only needed to instantiate the statetaxcalculator because it inherits from the federal tax calculator

#Because it inherits from the federal tax calculator I can use its method to gather the fed tax.

statetaxcalc = taxcalc.StateTaxCalculator(filing\_type=filing\_type, total\_income=total\_income, state=state)

total\_taxes = statetaxcalc.getStateAndFedTaxes()

formatted\_taxes = round(total\_taxes, 2)

print(f'Your total tax amount including federal and state taxes is ${formatted\_taxes}.')

#Quit option allows the user to exit the while loop

elif selection == "QUIT":

game\_over = False

#Catches any entry that doesn't match the available options.

else:

print("You did not pick an option from the selection list. \n------------------------------------------------------------------------------------------------")

From taxcalc.py

class FederalTaxCalculator:

#sets up attributes based on parameters input into object instantiation

def \_\_init\_\_(self, filing\_type, total\_income):

#user inputted arguments

self.filing\_type = filing\_type

self.income = total\_income

#Used within several class methods

self.taxbrackets\_des = {

"single": {.37: 578125, .34: 231250, .32: 182100, .24: 95375, .22: 44725, .12: 11000, .10: 0},

"married filing jointly": {.37: 693750, .34: 462500, .32: 364200, .24: 190750, .22: 89540, .12: 22000, .10: 0}

}

#Used to store the users tax\_amount

self.tax\_amount = 0

#Function to return the amount owed in federal taxes taking a standard deduction into effect using the users filing type and income to filter.

def getFedTaxAndRate(self):

#filtering down dictionary based on filing type

applicable\_fed\_tax\_brackets = self.taxbrackets\_des[self.filing\_type]

previous\_key\_value = 0

tax\_amount = 0

temp\_income = 0

#standard deduction logic - standard deduction rates for 2023 gathered from irs.gov

if (self.filing\_type == "single"):

temp\_income = self.income - 13850

elif (self.filing\_type == "married filing jointly"):

temp\_income = self.income - 27700

#Dictionary traversal logic

for key in applicable\_fed\_tax\_brackets:

#Handles negative values after standard deduction logic

if(temp\_income < 0):

print("Your standard deduction eliminated your need to pay taxes. ")

break

#finds the highest tax bracket the user would have to pay for and the amount to pay for in this bracket.

elif(temp\_income > applicable\_fed\_tax\_brackets[key]):

#value after standard deduction - the threshold value for the tax bracket

remainder = temp\_income - applicable\_fed\_tax\_brackets[key]

#the remainder is the taxable amount within the tax bracket multiplied by the brackets rate

tax\_amount += remainder \* key

#temp\_incme and previous\_key\_value are used to set up the following traversals

temp\_income -= remainder

previous\_key\_value = applicable\_fed\_tax\_brackets[key]

#Based on the logic in the previous step the temp\_income should always meet the previous key value. This allows us to use the previous value and the current value

elif(temp\_income == previous\_key\_value):

#follows a similar flow from the previous logic

remainder = previous\_key\_value - applicable\_fed\_tax\_brackets[key]

tax\_amount += remainder \* key

temp\_income -= remainder

previous\_key\_value = applicable\_fed\_tax\_brackets[key]

self.tax\_amount = tax\_amount

return tax\_amount

#Outputs the average tax rate after applying the different rates to the different brackets of income for the user

def getAverageRate(self):

rate = (self.tax\_amount/self.income) \* 100

return rate

#Children class of FedTaxCalculator. Adding in the State to find the state income tax the user falls intos

class StateTaxCalculator(FederalTaxCalculator):

def \_\_init\_\_(self, filing\_type, total\_income, state):

super().\_\_init\_\_(filing\_type, total\_income)

self.state = state

self.statetaxbrackets\_des = {

"AL": {"single": {.05: 3000, .04: 500, .02: 0},

"married filing jointly": {.05: 6000.01, .04: 1000.01, .02:0}},

"TN": {"single": {1: 0}, "married filingjointly": {1:0}}

}

#Method returns the amount owed in state taxes.

def getStateTaxAmount(self):

applicable\_state\_tax\_brackets = self.statetaxbrackets\_des[self.state][self.filing\_type]

state\_tax\_amount = 0

state\_temp\_income = self.income \* 1

for key in applicable\_state\_tax\_brackets:

#Base case if below last threshold

if((self.income <= 500 and self.filing\_type == "single") or (self.income <= 1000 and self.filing\_type == "married filing jointly")):

state\_tax\_amount = self.income \* .02

#same logic as the federal tax calculator from here on out

elif(state\_temp\_income > applicable\_state\_tax\_brackets[key]):

remainder = state\_temp\_income - applicable\_state\_tax\_brackets[key]

state\_tax\_amount += remainder \* key

state\_temp\_income -= remainder

previous\_key\_value = applicable\_state\_tax\_brackets[key]

elif(state\_temp\_income == previous\_key\_value):

remainder = previous\_key\_value - applicable\_state\_tax\_brackets[key]

state\_tax\_amount += remainder \* key

state\_temp\_income -= remainder

previous\_key\_value = applicable\_state\_tax\_brackets[key]

return state\_tax\_amount

#Method returns the total amount owed in state and federal taxes

def getStateAndFedTaxes(self):

#gets fed tax amount from the parent class

fed\_taxes = self.getFedTaxAndRate()

#gets the state tax amount from the current class

state\_taxes = self.getStateTaxAmount()

total\_taxes = fed\_taxes + state\_taxes

return total\_taxes

PART 6: Your Completed Program

**Task**

Provide the Replit link to your full program code.

**Requirements**

• The program must work correctly with all the comments included in the program.

**Inspiration**

• Check before submitting your touchstone that your final version of the program is running successfully.

<Provide the link to your program here>

https://replit.com/@blee56/WorldlyHandmadePagerecognition#main.py

https://replit.com/@blee56/WorldlyHandmadePagerecognition#taxcalc.py