Exercise: String Functions and If Statements

Summary

Demonstrates how to use string functions to work with text, and also shows how to use conditional code within `if` blocks.

Input Data

The input data is the file ***transactions.txt***, which contains real estate transactions in Central New York from 2020 through 2024. It has nine columns separated by tabs (see the tips section for more about tab-separated files). For this assignment, we'll only use three of the columns: column 1, which contains the municipality where the property is located (recall that the first column, which here contains the street address, is 0 by Python's numbering convention), column 4, which contains the sale price, and column 5, which contains the sale date.

Deliverables

Your finished repository should have one new script, **counts.py**, that performs the steps below.

Instructions

Please write a script called `counts.py` that summarizes recent real estate transactions in three parts of Central New York via the steps below. Remember to use comments to indicate what blocks of code are doing. See `demo.py` for examples of some of the steps and the use of comments.

1. Import `json`.
2. Create a variable called `ifile` equal to the string "transactions.txt".
3. Create a tuple called `places` that consists of the following three strings: "Ithaca", "Manlius", and "Oswego". These will be the places of interest.
4. Create a second tuple called `years` that consists of the following four strings: "2021", "2022", "2023", and "2024". These will be the years of interest.
5. Next, we'll create a two-level dictionary to hold data extracted from the input file. It will group sale prices by municipality and year. Start by setting variable `trades` to an empty dictionary `{}`.
6. Then, start a `for` loop that uses `muni` as the running variable and loops over `places`. Within the loop, do the following:
    1. Create a new dictionary called `by_year` by using a dictionary comprehension that loops over `years` using `y` as the running variable, and creates key-value pairs with `y` as the key and an empty list as the value. That is, the left portion of the dictionary comprehension should be `y:[]`.
    2. Put the new dictionary in `trades` under the name of the current municipality by setting `trades[muni]` equal to `by_year`.
7. After the `for` loop, check the structure of the `trades` variable by using the `print` function to print the result of calling `json.dumps()` with two arguments: `trades` and `indent=4`. If all goes well, there should be an outer dictionary with place names, and each place should have an inner dictionary of years, each of which holds a blank list. The code below will use those lists to collect transaction prices by location and year.
8. Next, open `ifile`.
9. Loop through the lines of the file using `line` as the loop variable and doing the following to each line:

1. Remove any quotes in the line by setting `line` equal to the result of calling `.replace()` on `line` with two arguments: the string `'"'` (the old string to be replaced, here a double-quote character inside two single-quotes) and `''` (the new string to replace it with, here two single quotes with nothing between them).
2. Create a variable called `items` that is equal to the result of applying the `.split()` to `line` with argument `"\t"` (which represents a tab character).
3. Create a variable called `muni` equal to element 1 of `items`.
4. Create a variable called `price` equal to element 4 of `items`.
5. Create a variable called `date` equal to element 5 of `items`.
6. Next, we'll clean up the municipality name by removing parenthetical suffixes like "(town)" or "(village)". Find the location of the first open parenthesis in the municipality name, if there is one, by creating a variable called `suffix` that is equal to the result of applying `.find()` to `muni` with the argument `"("`.
7. Find returns -1 when it the search string isn't present so we only need to fix names when `suffix` is 0 or larger. Start an `if` block by testing whether `suffix` is greater than `-1`. Within the if-block, do the following:
    1. Use list subscripting to set variable `muni` equal to the substring of `muni` from the start through `suffix`.
    2. Set `muni` equal to the result of calling `.strip()` on `muni` to remove any extra spaces.
8. After the if-block, see if the municipality is in the target group by adding a line that creates a variable called `place_ok` that is equal to the result of applying `.startswith(places)` to `muni`.
9. Check that the year is in the target group by creating a variable called `year_ok` by applying `.endswith(years)` to `date`.
10. Check that the place and year are both in their target groups by creating a variable called `keep` that is equal to `place_ok and year_ok`. Note that in Python, `and` is a logical operation: it's true only when both `place_ok` and `year_ok` are true.
11. Start an `if` block by testing whether `keep` is `False`.
    1. Within the block, use the `continue` statement to cause the script to go on to the next line in the file without processing this line any further.
12. After the if-block, add a line setting `year` to the last 4 characters in `date`.
13. Set `price` equal to the result of calling the `int()` function on `price`.
14. Start an `if` block by testing whether `price` is greater than `10e3` (ten thousand). This will eliminate entries that aren't normal "arms-length" sales, such as transfers between family members or between individuals and trusts, which are often recorded at $0, $1 or some other small price that's largely symbolic.
    1. Within the block, use the `.append()` method to add `price` to the list stored in `trades[muni][year]`, which collects the prices of all transactions in `muni` that happened in `year`.

10. The instruction above concludes the `for` loop. Now we'll print out the results. Add a `print` statement that prints the heading "Summary of Transactions".
11. Start a `for` loop using `muni` as the running variable and looping over the result of calling `sorted()` on the keys from `trades`. Within the loop do the following.
    1. Print the value of `muni`.
    2. Set variable `by_year` to the value of `trades` for key `muni`. This will be a dictionary with one key per year, and the value at each key will be a list of the prices for the trades in the that year
    3. Start a `for` loop using `year` as the running variable and looping over the result of calling `sorted()` on the keys from `by_year`. Within the loop do the following:
        1. Set `sales` equal to the value of `by_year` for key `year`.
        2. Set `n` to the length of `sales`.
        3. Set `tot` to the result of summing `sales`.
        4. Set `avg` to the result of calling `round()` with arguments `tot/n` and `-3`. The `-3` tells `round()` to round to 3 digits to the *left* of the decimal point rather than than the right. That is, it rounds the average to the nearest thousand dollars.
        5. Write out the data via a `print` statement called with 6 arguments: `"   "` (a string with 3 spaces), `year`, `":"`, `n`, `"trades, average"`, and `avg`. If all has gone well, the result should be a nicely formatted summary of the data.

Submitting

Once you're happy with everything and have committed all of the changes to your local repository, please push the changes to GitHub. At that point, you're done: you have submitted your answer.

Tips

- Files with columns of variables separated by tabs are an alternative to a more common format where the columns are separated by commas (comma-separated-variable or CSV format). Tab-separated-variable format is convenient when some of the variables may have commas embedded in them. They can easily be read by Excel or Stata.