# Week 9 - Plug-in Functions & Recursion

After restructuring our code, let's introduce a new little feature to this application -- categorizing the records.

You're going to provide some categories and subcategories, such as

* expense
  + food
    - meal
    - snack
    - drink
  + transportation
    - bus
    - railway
* income
  + salary
  + bonus

There is a new command "view categories" for the user, which list all the provided categories hierarchically. Once they know the available categories, they can specify a category when adding a record. When the user "view" all the records, the table also shows the categories for each record.

Then we provide another new command, "find", for the user. They can input a category name, and the program should find all records in the specified category or in a subcategory under that one. For example, if the user is finding "food", then all the records with category "food", "meal", "snack", and "drink" should be listed out.

$ python3 pymoney.py

How much money do you have? 1000

What do you want to do (add / view / delete / view categories / find / exit)? view categories

- expense

- food

- meal

- snack

- drink

- transportation

- bus

- railway

- income

- salary

- bonus

What do you want to do (add / view / delete / view categories / find / exit)? add

Add some expense or income records with category, description, and amount (separate by spaces):

cat1 desc1 amt1, cat2 desc2 amt2, cat3 desc3 amt3, ...

meal breakfast -50

... (add more records with different categories)

What do you want to do (add / view / delete / exit)? add

Add some expense or income records with category, description, and amount (separate by spaces):

cat1 desc1 amt1, cat2 desc2 amt2, cat3 desc3 amt3, ...

clothing pants -350

The specified category is not in the category list.

You can check the category list by command "view categories".

Fail to add a record.

What do you want to do (add / view / delete / view categories / find / exit)? view

Here's your expense and income records:

Category Description Amount

=============== ==================== ======

meal breakfast -50

drink coffee -100

food bread -80

food sausage -40

meal lunch -120

railway MRT -45

transportation taxi -170

bus 902 -20

income lottery 50

salary part-time 1200

snack cake -35

meal lunch -70

===========================================

Now you have 1520 dollars.

What do you want to do (add / view / delete / view categories / find / exit)? find

Which category do you want to find? food

Here's your expense and income records under category "food":

Category Description Amount

=============== ==================== ======

meal breakfast -50

drink coffee -100

food bread -80

food sausage -40

meal lunch -120

snack cake -35

meal lunch -70

===========================================

The total amount above is -495.

What do you want to do (add / view / delete / view categories / find / exit)? find

Which category do you want to find? transportation

Here's your expense and income records under category "transportation":

Category Description Amount

=============== ==================== ======

railway MRT -45

transportation taxi -170

bus 902 -20

===========================================

The total amount above is -235.

What do you want to do (add / view / delete / view categories / find / exit)? find

Which category do you want to find? income

Here's your expense and income records under category "income":

Category Description Amount

=============== ==================== ======

bonus lottery 50

salary part-time 1200

===========================================

The total amount above is 1250.

What do you want to do (add / view / delete / view categories / find / exit)? exit

## Required Steps

1. Define a function **initialize\_categories**.
   1. In the function, define a multi-level list  
      ['expense', ['food', ['meal', 'snack', 'drink'], 'transportation', ['bus', 'railway']], 'income', ['salary', 'bonus']]  
      (you may add more subcategories as you want)  
      and simply return it.
   2. Call **initialize\_categories** before the while loop and assign the returned value to a variable **categories**.
2. Define a recursive function **view\_categories**.
   1. In the function, handle the base case and recursive case.
      1. Print a category with proper indentation in the base case.
      2. Iterate through the list and call **view\_categories** in the recursive case.
   2. Add a condition **command == 'view categories'** in the **if-elif** construct and call **view\_categories** with **categories** as the parameter.
3. In the **add** function, prompt the user to specify a category for a record in addition to description and amount.
   1. Adjust your data structure for a record to store the category in addition to description and amount.
   2. Modify the code of **view** to add the categories of the records to the printed table.
   3. Modify the code of file reading and file writing to add the categories of the records to the file.
   4. Go through your code to make other necessary modifications due to this change.
4. Check if the specified category is in the predefined list **categories** when adding a record.
   1. Define a recursive function **is\_category\_valid(category, categories)** that returns **True** if **category** is in **categories** and **False** otherwise.
   2. Call **is\_category\_valid** in the **add** function to prevent a record from being added if the specified category is not valid.
   3. You might have to pass the global variable **categories** as a parameter into the **add** function so that it can be passed into **is\_category\_valid** in the **add** function.
5. Define a **find** function.
   1. In the **find** function, prompt for a category name to find.
   2. (If you find it this one too difficult, you could look for the hint below.)  
      Define a function **find\_subcategories** (may be recursive) that takes a category name to find and the predefined list **categories** as parameters, and returns a non-nested list containing the specified category and all the subcategories under it (if any).
      1. For example, **find\_subcategories('expense', categories)** returns  
         ['expense', 'food', 'meal', 'snack', 'drink', 'transportation', 'bus', 'railway']
      2. **find\_subcategories('transportation', categories)** returns  
         ['transportation', 'bus', 'railway']
      3. **find\_subcategories('salary', categories)** returns ['salary']
      4. **find\_subcategories('not-in-the-list', categories)** returns []
   3. In the **find** function, call **find\_subcategories** with the category name input by the user.
   4. Use the built-in **filter** function with lambda expression to filter the records whose category is in the list returned from **find\_subcategories**.
   5. Print out the filtered records and report the total amount of money of the listed records.
   6. Add a condition **command == 'find'** in the **if-elif** construct and call the **find** function with **records** and **categories** as the parameters.
6. Add at least one line of triple-quote docstring to each function you defined.
7. Make sure all of the variable and function names are in snake\_case.

## Hint for find\_subcategories

Basically there are 2 steps in this function: find the target category and flatten the subcategories under it.

1. The first step is similar to the **rec\_find** function in the lecture of week 9.

| def find\_subcategories(category, categories):  if type(categories) == list:  for i, v in enumerate(categories):  p = find\_subcategories(category, v)  if p == True:  return (i,)  if p != False:  return (i,) + p  return categories == category |
| --- |

Instead of returning a tuple of indices (the two lines in red highlight), it should return a list of the target category and its subcategories.

| def find\_subcategories(category, categories):  if type(categories) == list:  for v in categories:  p = find\_subcategories(category, v)  if p == True:  # if found, return the flatten list including itself  # and its subcategories  index = categories.index(v)  return flatten(categories[index:index + 2])  if p != False:  # p is a list returned from flatten  return p  return categories == category  def flatten(L):  # return a flat list that contains all element in the nested list L  # for example, flatten([1, 2, [3, [4], 5]]) returns [1, 2, 3, 4, 5] |
| --- |

Also some boundary conditions are (1) the target category does not have subcategories and (2) the target category is not found.

| def find\_subcategories(category, categories):  if type(categories) == list:  for v in categories:  p = find\_subcategories(category, v)  if p == True:  index = categories.index(v)  if index + 1 < len(categories) and \  type(categories[index + 1]) == list:  return flatten(categories[index:index + 2])  else:  # return only itself if no subcategories  return [v]  if p != []:  return p  return True if categories == category else []  # return [] instead of False if not found  def flatten(L):  # return a flat list that contains all element in the nested list L  # for example, flatten([1, 2, [3, [4], 5]]) returns [1, 2, 3, 4, 5] |
| --- |

1. The second step is to define the **flatten** function. Here we give you a reference answer, but you are encouraged to try it yourself.

| def flatten(L):  if type(L) == list:  result = []  for child in L:  result.extend(flatten(child))  return result  else:  return [L] |
| --- |

## Related Knowledge

* Lambda expressions
* filter() function
* Docstring
* Python style guide
* Recursion
* Recursive finding
* Indentation