**Technical Design Document**

Brayden Fox

06/12/2025

**Description:**

The program asks the user to enter their list of monthly expenses, and when finished, analyzes the expenses, displaying the total, highest, and lowest values. The total is calculated using the `reduce` method.

**Functions (in call order):**

1. **Function Name:** `main()`

**Description:** Contains the main program.

**Parameters:** None.

**Variables:**

* **`exp`:** stores data type contained in the `expenses` list; only used for linter type hints.
* **`\_TYPE, \_AMT, \_FMTD`:** stores integer indexes into `exp` type (a tuple).
* **`expenses`:** a list storing data entered by the user.
* **`dollar\_fmt`:** format string for dollar amounts.
* **`is\_entering`:** Boolean variable is false when the user elects to stop entering data.
* **`line`:** stores the raw line of text entered by the user.
* **`exp\_type, exp\_amount`:** stores the formatted contents after `line` is split by a comma delimiter.
* **`exp\_formatted`:** stores the formatted dollar amount so that a loop does not need to be created or used later to do the same.
* **`type\_col\_width, fmtd\_col\_width`:** stores the column widths calculated from the data gathered from the user (in unit chars).
* **`expenses\_total`:** stores the result of the `reduce()` function which calculates the total expense amount.

**Stepwise Logic:**

* Types, constants, and the `expenses` list are initialized.
* Program prompt and instructions are printed and the main while loop is engaged.
* A line of raw text is gathered from the user and stores in `line`.
* If the line is not empty, then:
  + In a try-except block, the line is split by comma delimiter and the resulting list is unpacked into `exp\_type, exp\_amount`.
  + `exp\_type` is stripped of leading/trailing whitespace, while `exp\_amount` attempts to parse to a float.
  + If the parsing fails, a `ValueError` is raised, and the user is advised of the unrecognized entry, given a number of example entries, and is prompted to re-enter.
  + Otherwise, the data is added to the `expenses` list, along with the string formatted dollar amount.
* Otherwise, if the line is empty:
  + The user has elected to end data entry, `is\_entering` is set to `False`, and the next iteration of the main while loop does not engage.
* After data entry, the length of the `expenses` list is determined.
* If it is zero, no data was entered, and the program forgoes the expense analysis.
* Otherwise, the program runs through the string elements of `expenses` that are to be displayed in columns of a table, determining the column widths in unit chars.
  + It does so using a lambda key expression to index into the appropriate elements of the tuples contained in `expenses`.
* The `expenses` list is sorted in ascending order. Again, a lambda key expression is used to access the correct element of the stored tuples.
* The function `format\_row(t, f)` is defined, where `t` accepts the column “type” and `f` accepts the column “formatted amount,” returning a formatted string based on the computed column widths from earlier.
  + The function avoids seemingly arbitrary format strings throughout the code (which would otherwise all be the same format strings, just accepting different parameters).
* Using the aforementioned `format\_row()` function, the expenses table is printed to the console.
* The `expenses\_total` is calculated using the `reduce()` method of the `functools` library.
  + How the lambda function passed to `reduce()` works is slightly complicated.
  + Because the function call to `reduce()` is given an initial value, `0`, the initial value of the first lambda parameter, `x0`, will be `0`; while the initial value of the second parameter, `x1`, will be the first element of the `expenses` list (a tuple).
  + Since `x1` is a tuple of the `exp` type, the lambda function indexes it at the index `\_AMT` to get a numeric amount, which is summed with the current `x0`.
  + At each iteration, `x0` will be the current numeric sum while `x1` is the current tuple from `expenses`, until all elements from `expenses` are exhausted and the `reduce()` function returns.
* Lastly, the program prints the highest/lowest amounts along with their names. Because the `expenses` list is sorted in ascending order by amount, it simply prints a format string containing the last/first elements, respectively.

**Return(s):** None.

1. **Function Name:** `format\_row()`

**Description:** produced a formatted string for printing the table result.

**Parameters:**

* **`t`:** accepts the column expense “type” string.
* **`f`:** accepts the column expense “formatted amount” string.

**Variables:** None.

**Stepwise Logic:** Using the column widths calculated in the higher scope, it creates and immediately returns the formatted string.

**Return(s):** A formatted string.

**Logical Structure:**

1. If the Python module is called directly, the `main()` function is called and the main program begins.
2. When the `main()` function returns, execution stops.

**Repository:** <https://github.com/fox-2-4/COP2373>

**Output Screenshot(s):**

