# Week 4 - Python Applications & String

We can create a variety of applications with Python. In week 4, we saw how to use Python to implement some Unix tools (uniq, cat, grep, wc). Those might appear to be boring to you because you don't use uniq in your daily life. How about creating an application that you really need every day? You can come up with your own ideas, but here we'll start with a money management application.

Let's start from the very beginning. Write a program that lets a user input the initial amount of money and one record of expense or income. The program will report the balance.

$ python3 pymoney.py

How much money do you have? 1000

Add an expense or income record with description and amount:  
breakfast -50

Now you have 950 dollars.

## Required Steps

1. Use **input()** function to accept the inputs.
2. Use **str.split()** method to deal with the input record.
3. Use **int()** function to convert the value.
4. Calculate the balance.
5. Use string formatting and **print()** to report the balance to the user.

## Notes

* The amount input by the user could be negative (representing expense) or positive (representing income). The "+" sign is optional for positive numbers. If you use the **int()** function, both strings "95" and "+95" are converted to the integer value 95.
* You might get two strings returned from the **split()** method. The first one is the description (e.g. "breakfast") and the second one is the amount of expense or income (e.g. "-50"). For now, your program doesn't need to handle the first string. Just convert the second string to integer and do the remaining operations.

## Related Knowledge

* input() and print() functions
* Operators
* Integers
* Variables
* Strings
* str.split() method
* String formatting

# Week 5 - Sequences, Sets, Dictionary

One record is never enough. Let's make it accept multiple records.

$ python3 pymoney.py

How much money do you have? 1000

Add some expense or income records with description and amount:

desc1 amt1, desc2 amt2, desc3 amt3, ...  
breakfast -50, lunch -70, dinner -100, salary 3500

Here's your expense and income records:

breakfast -50

lunch -70

dinner -100

salary 3500

Now you have 4280 dollars.

## Required Steps

1. Modify the prompt strings.
2. Split the input string by commas then spaces and build a data structure like [('breakfast', -50), ('lunch', -70), ...].
3. List the records and report the balance to the user.
4. Add some comments for your code.

## Related Knowledge

* str.split() method
* Conversion to list or tuple
* List comprehension
* str.join() method
* sum() function

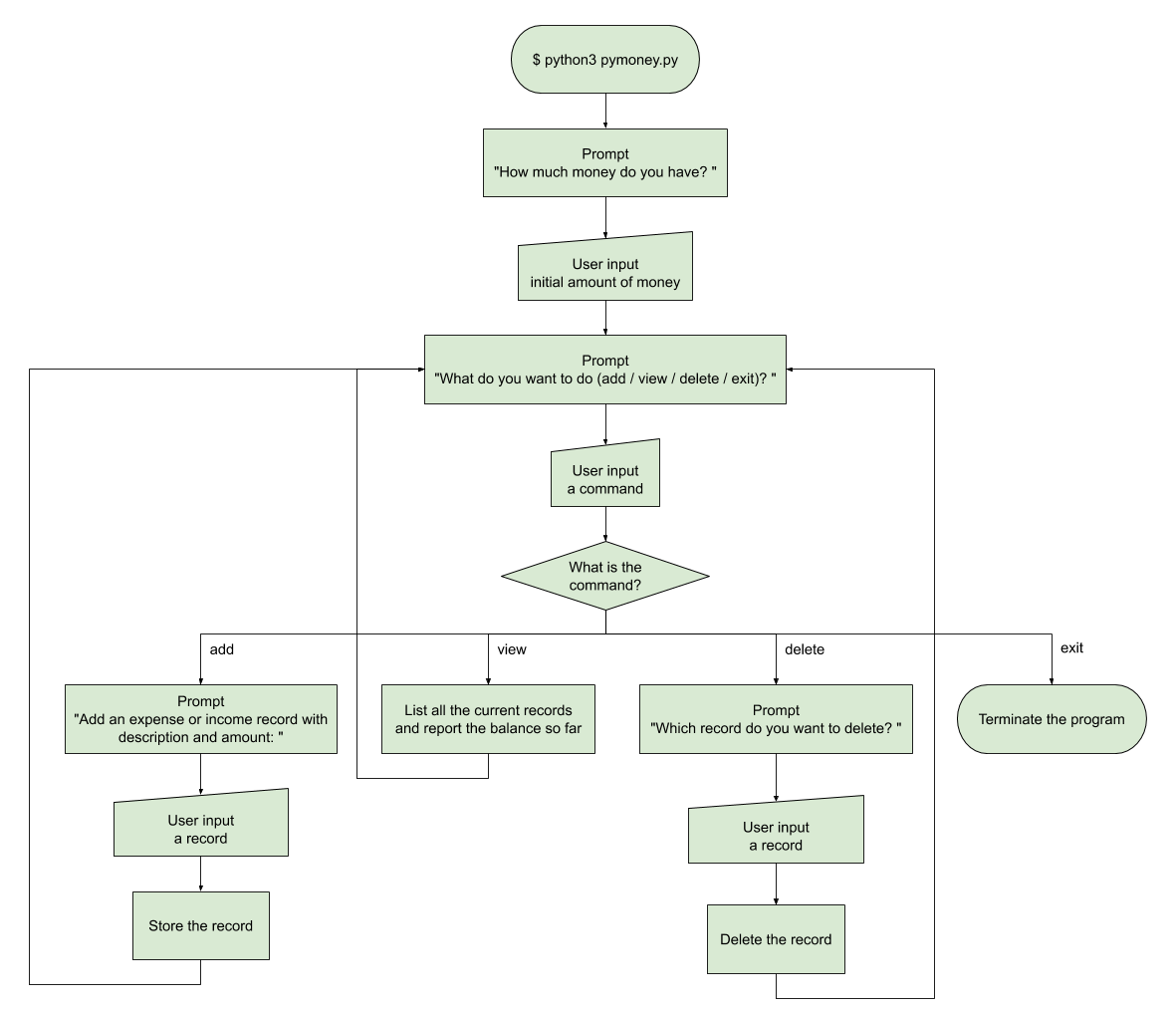
# Week 6 - Control Constructs

Currently our application only has one flow: user entering initial amount of money, user entering the records, program listing the records, and program reporting the balance.

Most of the time, the user should be able to decide what to do next. Let's provide 4 basic commands which the user can execute at any time.

* Add a record
* View the records and the balance
* Delete a record
* Exit the application

After asking for the initial amount of money, the program should keep prompting the user for one of the above commands until the user chooses to exit. The following flow chart illustrates the overall flow of the program.



$ python3 pymoney.py

How much money do you have? 1000

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

Add an expense or income record with description and amount:

breakfast -50

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

Add an expense or income record with description and amount:

lunch -70

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

Add an expense or income record with description and amount:

dinner -100

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

Here's your expense and income records: you may design your own printing format

Description Amount

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

breakfast -50

lunch -70

dinner -100

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

Now you have 780 dollars.

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

Add an expense or income record with description and amount:

breakfast -50

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

Add an expense or income record with description and amount:

salary 3500

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

Here's your expense and income records: you may design your own printing format

Description Amount

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

breakfast -50

lunch -70

dinner -100

breakfast -50

salary 3500

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

Now you have 4230 dollars.

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

Which record do you want to delete? design your own way to specify the "breakfast -50" record between "dinner -100" and "salary 3500"

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

Here's your expense and income records: you may design your own printing format

Description Amount

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

breakfast -50

lunch -70

dinner -100

salary 3500

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

Now you have 4280 dollars.

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

## Required Steps

1. Prompt the user for the initial amount of money.
2. Prepare a data structure (e.g. list of tuples, dictionary, class, etc.) to store the records.
3. Create a **while** loop. In the **while** loop,
   1. Prompt the user for a command.
   2. Create a **if-elif-else** statement to handle different commands.
      1. Handle the "add" command.
      2. Handle the "view" command. Try to print the records in a neat format.
      3. Handle the "delete" command. You should also remove the amount of this record when calculating the balance (e.g. 50 is added back to the balance after deleting "breakfast -50").
      4. Handle the "exit" command if necessary.
   3. Leave the **while** loop if the command is "exit".
4. Add appropriate comments for your code.

## Think and Solve

You might run into a question when designing the "delete" command: how should the user specify which record to delete?

Apparently by saying "breakfast" or "breakfast -50" is not enough because there are possibly more than one records with the same description and amount. It's not feasible to assume that the user is deleting the first, last, or all records as "breakfast -50".

As a software developer, you need to come up with a solution and implement it in your code.

## Related Knowledge

* while loop
* if-elif-else statements
* list, tuple, dictionary and their methods
* List comprehension
* enumerate() function
* str.split() and str.join() methods
* string formatting

# **Week 7 - Exceptions & Files**

You might notice that no record is left after the user exits from the application then starts it again. That's because all the variables in Python are stored in the memory (記憶體). Once a program terminates, it releases the memory space it allocated. Our application is far from useful. You can't ask the user to run the application for days, months, or even years. Even if a user is willing to run it forever, computers power off accidentally from time to time.

However, things stored in files stay there no matter how many times the computer restarts, unless the hard disk (硬碟) is broken. This leads to our solution: **write the records to a file before the program stops and read from it when the program restarts**.

$ python3 pymoney.py

How much money do you have? 1000

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

Add an expense or income record with description and amount:

breakfast -50

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

Add an expense or income record with description and amount:

lunch -70

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

Add an expense or income record with description and amount:

salary 3500

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

Here's your expense and income records:

Description Amount

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

breakfast -50

lunch -70

salary 3500

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

Now you have 4380 dollars.

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

$ python3 pymoney.py

Welcome back!

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

Here's your expense and income records:

Description Amount

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

breakfast -50

lunch -70

salary 3500

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

Now you have 4380 dollars

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

Add an expense or income record with description and amount:

breakfast -50

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

Which record do you want to delete? design your own way to specify "lunch -70"

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

Add an expense or income record with description and amount:

dinner -120

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

$ python3 pymoney.py

Welcome back!

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

Here's your expense and income records:

Description Amount

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

breakfast -50

salary 3500

breakfast -50

dinner -120

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

Now you have 4280 dollars.

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

Since your code is getting longer and more complicated, many things can go wrong and crash your program. Besides some mistakes you might make in your code, more errors can occur as the user inputs something out of your expectation. Now that we included file operations in our program, we also have to consider things happen outside the program: files created by the program being modified or deleted outside the program, for example.

Here we list some exceptions that may occur in our current program:

* (1) When prompted for the initial amount of money, the user inputs a string that cannot be converted to integer.
* (2) When prompted for a command (add / view / delete / exit), the user inputs a string that is not one of the four above.
* When prompted to add a record,
* (3) the user inputs a string that cannot be split into a list of two strings, or
* (4) the second string after splitting cannot be converted to integer.
* When prompted to delete a record,
* (5) the user inputs in an invalid format in respect of your design, or
* (6) the specified record does not exist.
* When loading the records from the file,
* (7) the file does not exist,
* (8) no line is in the file,
* (9) the first line cannot be interpreted as initial amount of money (i.e. cannot be converted to integer), or
* (10) any of the other lines cannot be interpreted as a record (i.e. cannot be split into a list of two strings or the second string after splitting cannot be converted to integer).

$ python3 pymoney.py

How much money do you have? abc

Invalid value for money. Set to 0 by default.

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

Invalid command. Try again.

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

Add an expense or income record with description and amount:

salary3000

The format of a record should be like this: breakfast -50.

Fail to add a record.

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

Add an expense or income record with description and amount:

salary 3500

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

Add an expense or income record with description and amount:

breakfast -abc

Invalid value for money.

Fail to add a record.

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

Here's your expense and income records:

Description Amount

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

salary 3500

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

Now you have 3500 dollars.

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

Which record do you want to delete? invalid in respect to your design

Invalid format. Fail to delete a record.

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

Which record do you want to delete? specify a record that doesn't exist

There's no record with xxxxxxxx. Fail to delete a record.

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

(modify the content of records.txt to make it invalid)

$ python3 pymoney.py

Invalid format in records.txt. Deleting the contents.

How much money do you have? 1000

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

Add an expense or income record with description and amount:

breakfast -50

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

Here's your expense and income records:

Description Amount

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

breakfast -50

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

Now you have 950 dollars.

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

## **Required Steps**

1. Before the program terminates, write the initial amount of money and records into a file named **'records.txt'** in a **with-as** statement.
2. Use the **write()** method to write the initial amount of money into the file.
3. Convert the data structure storing the records into a list of strings and use the **writelines()** method to write the list of strings into the file.
4. Put new line characters properly so that you can read the file by **readline()** and **readlines()** later.
5. At the beginning of the program, try to open 'records.txt' by calling the **open()** function in a **try-except** statement.
6. If the file exists, use the **readline()** method to read the first line, which is the initial amount of money, into a variable. Then use the **readlines()** method to read the records and build the data structure you used before (e.g. list of tuples, dictionary, class, etc.) to store them. Remember to **close()** the file after reading.
7. If the file doesn't exist (i.e. a **FileNotFoundError** is raised), prompt the user for the initial amount of money and initialize the variables needed in the coming operations.
8. Handle 10 possible exceptions apart from the already handled **FileNotFoundError**.
9. Use **try-except** (and probably **try-except-finally**) statements.
10. Specify the error type if possible. (e.g. except **ValueError**:)
11. Use **sys.stderr.write()** to report some error messages to the user if necessary.
12. Prompt the user again or set related variables directly, depending on the situation.
13. Add appropriate comments for your code.

## **Notes**

* In the 10 exceptions we list above, "the file does not exist" is not really an unexpected condition. It inevitably happens when this application runs for the first time. Therefore, it doesn't make sense to report an error using **sys.stderr.write()**. You can simply assume that there is no record at all and prompt the user for the initial amount of money.
* In some cases, you may use the **if-else** statement instead, if it's more reasonable (and more convenient) than using the **try-except** statement.

## **Related Knowledge**

* File operations
* open() function
* readline() and readlines() method
* write() and writelines() method
* close() method
* with-as statements
* Stand error output (sys.stderr)
* Exception handling
* try-except and try-except-finally statements
* Common error types

# **Week 8 - Functions**

Over the past weeks, the program we build has grown to almost or over one hundred lines of code. Let's move the detailed works into functions and make the main flow clearer.

Here are the functions we'll extract:

* initialize: read the file 'records.txt' or prompt for initial amount of money.
* add: prompt for a record and add it into the record list (or other data structure you're using).
* view: print the records.
* delete: prompt for a record to delete and delete it from the record list.
* save: write the records to the file 'records.txt'.

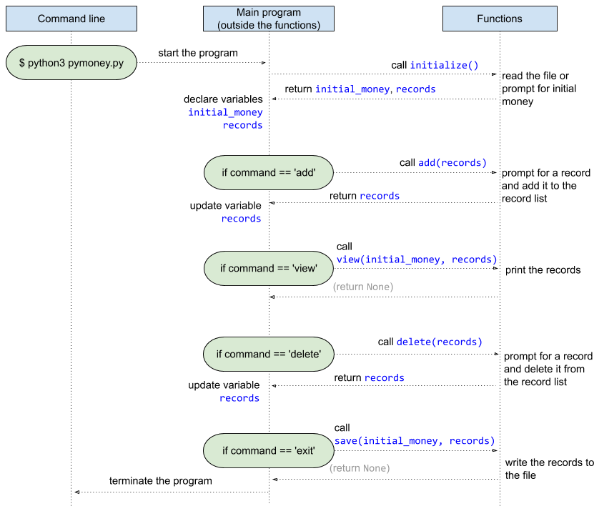
After defining the functions, the remaining part (not in functions) of your code should be like this:

| import sys    # The 5 function definitions here    initial\_money, records = initialize()  while True:  command = input('\nWhat do you want to do (add / view / delete / exit)? ')  if command == 'add':  records = add(records)  elif command == 'view':  view(initial\_money, records)  elif command == 'delete':  records = delete(records)  elif command == 'exit':  save(initial\_money, records)  break  else:  sys.stderr.write('Invalid command. Try again.\n') |
| --- |

Here we want to make the functions independent of other variables defined in the global scope. That is, none of the functions should access **initial\_money** or **records** directly. Instead, needed variables should be passed into the functions as parameters and reassigned as the returned value from the functions.

| **O** | **X** |
| --- | --- |
| def foo(L):  s = input()  L.append(s)  return L    L = ['abc', 'def']  L = foo(L) | def foo():  s = input()  L.append(s)    L = ['abc', 'def']  foo() |
| The function foo does not know there is a variable L outside. It only sees the passed-in parameter L and simply returns it after modification. | Yes, this works the same as the code at the left. But now foo is dependent on the outside variable L, which is not what we want here. |

This diagram illustrates the data flow between the main program and the functions:



## **Required Steps**

1. Identify the variables that are used and maintained throughout the main program. Generally you should find only **initial\_money** and **records** (maybe under other names). You may have other ones but make sure they are necessary. Eliminate unnecessary ones and keep those kind of variables as few as possible.
2. Define the 5 functions: **initialize**, **add**, **view**, **delete**, and **save**.
3. Define the functions with proper formal parameters and move the code originally in **if-elif** into respective functions.
4. Adjust the code inside the functions so that they're only using parameters and not accessing outside variables.
5. Return values for the caller (i.e. main program) to update related variables.
6. Rearrange the remaining code as the example in purple border above.
7. Call the functions and pass needed variables as parameters.
8. Declare or update related variables after a function returns.

## **Related Knowledge**

* Functions, parameters, and return values
* Scope of identifiers