Introduction

EasyCashFlow, or ECF for short, is a programming language whose main purpose is to aid in the learning of how cashflow diagrams work. After taking the course Engineering Economics, we realized that a small programming language where you could manipulate cash flow diagrams would be useful. ECF's main function is to help people visualize how earning and losing money in specified periods of time looks and how these values change as they are moved to the present and the future.

Language Tutorial

First, the user must run EasyCashFlow.py, which is located inside the EasyCashFlow folder, from the command line. As soon as it runs, the user should be ready to start making inputs. ECF's main and only data type is cfd, which represents cash flow diagrams. To create cashflow diagrams the user must type, for example:

```
cfd NAME = create(LENGHT)
```

where NAME is the desired name for the cashflow diagram and LENGTH is the desired integer value for the length of the cashflow diagram. This can be done multiple times to create different cashflow diagrams. If one wishes to obtain the inputted LENGTH of said cash flow, you can use the command `length(NAME)` and obtain this value. This is the first command the user must do, because if not, there a no cashflows in the system to manipulate and therefor nothing to do.

After already having one or more cfds registered in the system, the user is ready to start adding values to different time periods, manipulating these values and displaying the cashflow diagram.

At any point after creating the cashflow diagram, the user can choose to display it using one simple command:

```
display(NAME)
```

where NAME is the name of a cashflow diagram in the system. It can be displayed with or without values added.

To add a positive value to a cfd the user must type, for example:

```
earn(VALUE, TIME,CFD_NAME)
```

Where VALUE is the amount of cash to be added, TIME is the time period (no greater than the length of the cashflow) and CFD_NAME is the name of the cashflow where the earn action will take place. The user can also add negative values with the same format, but typing lose instead of earn.
They can also remove a value by resetting a specified TIME to 0 using the command

```
remove(TIME, NAME)
```

where TIME is the specified period of time and NAME is the name of the cashflow diagram in the system.

After adding positive and/or negative values to the cashflow, the user can choose to modify the cashflow by moving values in specified times to the present within a cashflow. The user must type:

```
pw(INTEREST, TIME,NAME)
```

where INTEREST is any interest in decimal form, TIME is the time where the desired value is specified at and NAME is the name of the cashflow diagram. This command moves the value in a specified time to the present based on a specific user inputted interest. The values can also be moved to the future by typing fw instead of pw, but using the same format.
Another way to create cashflows is to combine two already existing ones. For example, the user can type:

```
cfd NAME = combine(NAME1,NAME2)
```

where NAME is the desired name for the new cashflow and NAME1 and NAME2 are names of preexisting cashflow diagrams. You can also subtract NAME2 from NAME1 using the subtract command, which has the same format, but the user types subtract instead of combine

LANGUAGE REFERENCE MANUAL

DATA TYPE:

cfd - a cashflow diagram

COMMANDS:

create(INTEGER) - creates a new cashflow diagram where INTEGER is any positive integer.

```
Ex: cfd cf_name = create(8)
```

length(ID) - displays the length of a specified cashflow diagram where ID is the name of the cashflow

```
Ex: length(cf_name)
```

display(ID) - displays a cashflow diagram in system, where ID is the name of a cashflow.

Ex: display(cf_name)

earn(VALUE, TIME, ID) - add positive values to a specified time in a cashflow diagram. VALUE is an integer or float that represents the amount of money earned, TIME is the period in which one wants to add money and ID is the name of the cashflow in the system.

Ex: earn(10, 2, cf_name)

lose(VALUE, TIME, ID) - add negative values to a specified time in a cashflow diagram. VALUE is an integer or float that represents the amount of money earned, TIME is the period in which one wants to lose money and ID is the name of the cashflow in the system.

Ex: lose(20.5, 3, cf_name)

remove(TIME, ID) - remove a value from a specified time in a cashflow diagram. TIME is the period in which one wants to reset to 0 and ID is the name of the cashflow in the system.

Ex: remove(3, cf_name)

clear(ID) - remove all values from a cashflow diagram. ID is the name of the cashflow diagram in the system to be cleared.

pw(INTEREST, TIME, ID) - move a specified value in a specified time to the present (to time = 0). INTEREST is a float or integer value representing an interest, TIME is the specified period that the user wishes to move to the present and ID is the name of the cashflow diagram in the system.

Ex: pv(0.8,3,cf_name)

fw(INTEREST, TIME, ID) - move a specified value in a specified time to the future (to time = length). INTEREST is a float or integer value representing an interest, TIME is the specified period that the user wishes to move to the present and ID is the name of the cashflow diagram in the system.

Ex: fv(1.2, 3, cf_name)

combine(ID1, ID2) - adds the values in two cashflow diagrams to form a brand new cashflow diagram. Must declare a new cfd variable along with it. ID1 is the name of a cashflow in the system and ID2 is the name of another cashflow in the system.

Ex: cfd cf_name3 = combine(cf_name1,cf_name2)

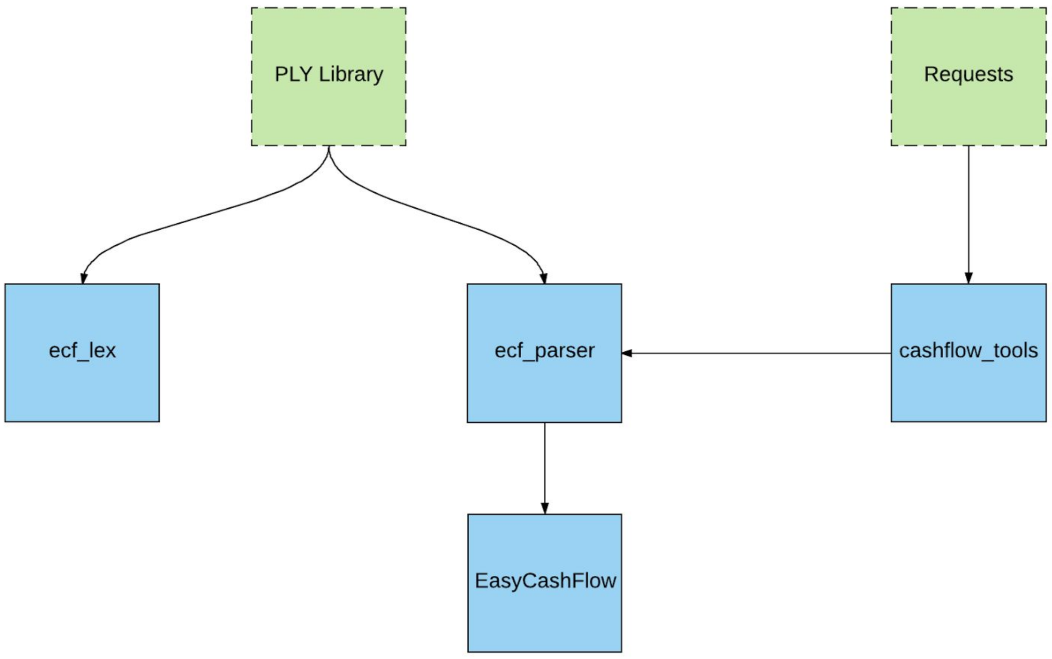
subtract(ID1, ID2) - subtracts the values in two cashflow diagrams to form a brand new cashflow diagram (ID1 - ID2).

Must declare a new cfd variable along with it. ID1 is the name of a cashflow in the system and ID2 is the name of another cashflow in the system.

Ex: cfd cf_name3 = subtract(cf_name1,cf_name2)

LANGUAGE DEVELOPMENT

TRANSLATOR ARCHITECTURE:



The execution of EasyCashFlow starts from the module with the same name, EasyCashFlow.py. Through that module, the program runs constantly as the user keeps inputting commands… or until he inputs the "quit" command.

Development Tools:

Python version 3.6 - The most recent version of python was selected due to its high compatibility and libraries.

JetBrains Pycharm Community Edition 2016.3.2 - The selected IDE for developing the program.

Anaconda 4.3.0 - Anaconda is the project interpreter that we used for our project. We chose anaconda since it provides many libraries that we used in our code (numpy, etc..)

Github - A web based git repository service. utilized to publish and facility the mergings of our individual work.

Test Methodology

The inputs used to test the program where the following:

```
cfd cfd1 = create(10)
cfd cfd2 = create(12)
earn(10.5, 2, cfd1)
earn(2, 2, cfd1)
lose(20, 3, cfd2)
earn(5, 3, cfd2)
earn(20, 2, cfd2)
display(cfd1)
display(cfd2)
cfd cfd3 = combine(cfd1,cfd2)
cfd cfd4 = subtract(cfd1,cfd2)
display(cfd3)
display(cfd4)
remove(2,cfd3)
display(cfd3)
pw(2, 2, cfd1)
fw (1.2, 2, cfd2)
display(cfd1)
display(cfd2)
```

Though more programs were utilized during testing to ensure both lexer and parser worked correctly, these particular inputs helped us test the final product. Each display(cfd) command can show us what is going on with the cashflow diagram at specific moments. With various display methods after key functionalities, we can analyze if what the program was calculating was correct or not. Any errors the user might do in his inputs would not be accepted by the program.

Conclusion

During the process of developing the EasyCashFlow programming language some roadblocks were found. For example, as we developed the intermediate code we found out that we would have to make the cash flows vertically instead of horizontally as specified in the proposal of this project. Also we had some bugs with subtract and combine functions. When we tried to combine a negative cashflow with a positive cashflow it would return a negative cash flow even though the positive values were greater than the negative ones. By debugging we found out that its was a problem with some sign convection.This project helped us become familiar with how programming languages are created.Also, we became more proficient with Python which is a skill that will definitely help us in the future as software developers. Finally, as a group we improve team skills such as communicating collaborating, and time management. It was a complete learning experience that has given us the skills and experience to tackle our future projects throughout our college career.