Project Part 1

Contents

[Contents 1](#_Toc469071822)

[Purpose 2](#_Toc469071823)

[Assignment Background 2](#_Toc469071824)

[Assignment Statement 2](#_Toc469071825)

[Requirements: 3](#_Toc469071826)

[Code/Comment Format 5](#_Toc469071827)

[What to Deliver 5](#_Toc469071828)

[Notes 6](#_Toc469071829)

[Grading 6](#_Toc469071830)

# Purpose

This assignment is the first of a mini project where you are going to demonstrate your understanding of the modules of this class.

The mini project is about building a small ETL program in python. Each part will focus on different portions / stages of the ETL process.

# Assignment Background

Summarizing your acquired knowledge from modules 1-4, we are going to focus on the E part of the ETL process. For reference, ETL stands for Extract, Transform, Load. We are going to focus here on Extract.

# Assignment Statement

* Read CSV files
* Load the content of the CSV files into usable records
* Use polymorphism to hold data
* Use inheritance to process data

The file*\_\_init\_\_.py* is blank and should not be modified. You are required to supply *Last Name\_First Name\_Project\_Part1.py.*

# Requirements:

To extract the raw data (row) into a usable format that can be relied upon by the rest of your application, we will define the concept of a record.

1. Create class: AbstractRecord
   1. This class will contain 1 (instance) member: name
2. Create a record class for each of the files you want to load.  
   i.e. BaseballStatRecord and StockStatRecord.  
   And including the following functionality:
   1. inherit the AbstractRecord
   2. have an initializer method that takes the data you want to load as arguments:  
      For stocks:
      * Stock symbol (ticker)→ this should be stored in the “name” member
      * Company name (company\_name)
      * Exchange country (exchange\_country)
      * Stock Price (price)
      * Exchange Rate (exchange\_rate)
      * Shares Outstanding (shares\_outstanding)
      * Net Income (net\_income)
      * Market Value in USD (market\_value\_usd) – This value is calculated in step 4e
      * Price/Earnings Ratio (pe\_ratio) – This step is calculated in step 4e
   3. for baseball
      * player name → this should be stored in the “name” member
      * salary
      * G (Games played)
      * AVG (which is the batting average)
   4. For each record type, override \_\_str\_\_() (<https://docs.python.org/3/reference/datamodel.html#object.__str__> ) to return a string of the form: “<name of the record type> ( <value1>, <value2>, <...> )” using “str.format”.

*For floats please only display 2 decimal numbers (2 numbers after the comma)*

To load the data we are going to need a CSV reader.

1. Create 1 AbstractCSVReader class
   1. The class should have an initializer method taking the path to the file to be read
   2. The class should have the method: row\_to\_record(row)
      * Where “row” is a row from the CSV as a dictionary
      * This method should be implemented by simply raising NotImplementedError.
   3. The class should have the method: load() that returns a list of records. Load should:
      1. Use “with” to open the CSV files
      2. read each row from the file into a list
      3. call the row\_to\_record method and send the row as a parameter
      4. handle the BadData exception raised by row\_to\_record by skipping the record – For more on BadData Exception see step 5
      5. If no exception is raised: then the record should be added to the list of records.
      6. Once all records are loaded into the list, returns the list.
2. Create a CSV reader class for each of the files you want to load   
   *i.e.* *BaseballCSVReader and StocksCSVReader*
   1. The class should inherit the AbstractCSVReader
   2. Each class should implement its own row\_to\_record method. The input is a list of unvalidated data, it should validate the data, parse it, create new record and return the record created. (Hint: a tuple if a good structure to use for records)
   3. The validation depends on your concrete record:
      * Validation fails for any row that is missing any piece of information
      * Validation fails if the name (symbol or player name) is empty
      * Validation fails if any of the numbers (int or float) cannot be parsed (watch out of the division by zero!!)
   4. If validation fails: this method should raise a BadData exception (requirement #5)
   5. StocksCSVReader should have two calculations using the extracted records:
      * market\_value\_usd = Price \* ExchangeRate \* SharesOutstanding
      * pe\_ratio = Price \* SharesOutstanding / NetIncome
3. Create a BadData custom exception to handle record creation errors
4. From your main section ( <https://docs.python.org/3/library/__main__.html>) )
   1. load the CSV (e.g. BaseballCSVReader('path to my CSV').load())
   2. Print each record to the console. You are to use: print(record)

# Code/Comment Format

Good code includes well named variables that are consistent from the beginning to the end of the program. Naming of objects should be self-explanatory. For instance, iterator\_for\_noun\_list is much better than i.

Every program consists of a sequence of paragraphs, each of which has objectives, and which builds on the previous paragraphs. We are mostly interested in objectives that are valid at the end of the program so we can verify the program's design. The following is a preferred form for such paragraph headings. The # sign is adequate when the comment is a single line.

#This is an in-line comment – used to document the code for you, or anyone else, that intends

#To extend the code

In-line comments are helpful when one has to go back to the code 6 months later to make changes.

For doc strings, python allows the use of triple quotes. The triple quotes can be either single or double quotes. A doc sting is generally used as user documentation. It does not need to include details of the implementation of the program, but instead it provides documentation as how to use the API for the program (input, output etc.)

For example:

“””

This is an example of a doc string

It allows multiple lines within the string.

“””

‘’’

This is an example of a doc string

It allows multiple lines within the string.

‘’’

This becomes significant when using functions, classes etc. as the triple quotes help to self-document the parameters and return values of the function.

# What to Deliver

Supply

1. A .doc file with any remarks/comments you want to make. We can insert feedback into the file. Include screen shots of key output. Please label this Word file <Last Name>\_<First Name>\_Assignment \_2. For example, Smith\_Joe\_Assignment\_1.
2. The *Last Name\_First Name\_Assignment2.py* file (the *\_\_init\_\_.py* file is attached and should be left as is).

# Notes

* Assignments can be submitted once. If extenuating circumstances exist, contact your facilitator.
* Note the statement in the syllabus on timeliness of submissions (the gist being that all assignments must observe the deadlines).
* Start by identifying and ordering the objectives.
* There are no testing requirements for this assignment. However, it would be prudent to make sure your program does not crash and all input validation is performed correctly.

# Grading

Please note that points for each step can be deducted for poorly documented code. If you think something needs to be explained, add comments.

Step 1a – 2 pts

Step 2a – 2 pts

Step 2b – 2 pts

Step 2c – 2 pts

Step 2d – 2 pts

Step 3a – 2 pts

Step 3b – 2 pts

Step 3ci – 5 pts

Step 3cii – 10 pts

Step 3ciii – 5 pts

Step 3civ – 5 pts

Step 3cv – 5 pts

Step 3cvi – 5 pts

Step 4a – 2 pts

Step 4b – 20 pts – 10 pts for each implementation of row\_to\_record

Step 4c - 2 pts

Step 4d – 2 pts

Step 4e – 10 pts

Step 5 – 5 pts

Step 6 – 10 pts

5 pts load and print Baseball records

5 pts load and print Stock records