Data file parser in memory

Problem definition

You are given two type of files as input, these files consist of a datafile and a format specification. Given these two type of files, parse the datafile and return the data in a usable format.

Problem details

Data files exist in a `data/` directory relative to your application and specification files exist in a `specs/` directory relative to your application.
Specification files will have filenames equal to the file type they specify and extension of `.csv`. So `fileformat1.csv`would be the specification for files of type `fileformat1`.
Data files will have filenames based on their specification file name, followed by an underscore, followed by the drop date and an extension of `.txt`. For example, `fileformat1_2007-10-15.txt` would be a data file to be parsed using `specs/fileformat1.csv`, which arrived on 10/15/2007.
Format files will be csv formated with columns "column name", "width", and "datatype".

- "column name" will be the name of keys in the dictionary
- "width" is the number of characters taken up by the column in the data file
- "datatype" is the expected data type stored in the dictionary

Data files will be flat text files with lines matching single records in the dictionary. Lines are formatted as specified by their associated specification file.

Example

This is an example file pair; other files may vary in structure while still fitting the structure of the problem details (above):
`specs/testformat1.csv`:

```
"column name",width,datatype
name,10,TEXT
valid,1,BOOLEAN
count,3,INTEGER
```

Here we have a specification that describes 3 columns:

- The first 10 characters labeled "name" of type TEXT
- The next 1 character labeled "valid" of type BOOLEAN ('1' = True, '0' = False)
- The last 3 characters labeled "count" of type INTEGER

`data/testformat1_2015-06-28.txt`:

```
Diabetes  1  1
Asthma    0-12
Stroke    1103
```

Processing this data file results in the following dictionary in the case of Python:

```
output = [
    {'name': 'Diabetes', 'valid': True, 'count': 1},
    {'name': 'Asthma', 'valid': False, 'count': -12},
    {'name': 'Stroke', 'valid': True, 'count': 103},
]
```

Signature

The following is the expected function signature should you choose to write this program in Python. You are free to choose any other programming language of your choice, the function should still follow the logical signature given here.

```
def parse_flatfile(datafilename, formatfilename):
    # TODO(you): implement
    output = [{'?':'?',...},...]
    return output
```

Expectations

You will need to show your code running in the interview on your own machine. That also means the interviewers will review your code but will not run it.

Here are some basic assumptions:

1. Choice of programming language is left up to your discretion
2. You should implement the conversions for the data types: `TEXT`, `BOOLEAN`, and `INTEGER` into primitive datatype in your programming language
3. You should be able to handle any specification file that matches the problem description (not just the given example)
4. Files can be assumed to use UTF-8 encoding
5. You are free to use any library/opensource projects whenever necessary and reasonable. We might question your choice during the interview, however.

You are expected to include the followings in your submission:

1. Finished method with the given signature AND a driver program that calls that method. Please output the content of your dictionary-equivalent data structure to stdout for readability (Any stdout console printing would work).
2. Unit tests
    a. you are expected to design your code such that core logics are unit-testable
    b. you are expected to write unit tests for different cases of the core logics. You do not have to use any specific unit test frameworks but should include instructions to run your tests should you use any. You may even write a simple driver program for us to run your unit test cases with mockup data.
3. Error Handling
    a. Your code should handle exceptions and errors gracefully
    b. Think of the edge cases and potential exceptions you might encounter. You will have a chance to talk about your choice for handling such errors during the interview