EXCELerator

Release v0.8

Nov 20, 2019

Contents:

Superseded by fuzzytable, which includes an improved data model and csv compatibility.

- Docs: fuzzytable.readthedocs.io
- PyPI: pypi.org/project/fuzzytable
- github: github.com/jonathanchukinas/fuzzytable

```
pip install fuzzytable
```

CHAPTER 1

EXCELerator documentation:

1.1 Readme

1.1.1 !! EXCELerator is Deprecated !!

Superseded by fuzzytable, which includes an improved data model and csv compatibility.

- Docs: fuzzytable.readthedocs.io
- PyPI: pypi.org/project/fuzzytable
- github: github.com/jonathanchukinas/fuzzytable

```
pip install fuzzytable
```

1.2 Anatomy of a Project

1.2.1 Excel Worksheet

- **HEADER**
  - HEADER ROW
- VALUES

1.2.2 Tabular Data

(As interpreted by `excelerator.TableReader`)

- **FIELD**
  - FIELD NAME

  - HEADER NAME
  - COLUMN NUMBER
  - DATA
- RECORD

1.3 API Reference

1.3.1 TableParser class

1.3.2 WorksheetParser class

Todo: This class has yet to be implemented.

1.3.3 FieldParser class

1.3.4 Normalizing Data

Basic Use

Let's say you have this very simple table:

Name	Birthday
John Doe	1995
123	2003OCt
Jane Doe	1964
River Song	2019

You want to ensure the first field (Name) returns only strings and the Birthday column returns integers.

```
from excelerator import TableReader
from excelerator import normalize as n

tr = TableReader(
    path='path/to/excel.xlsx',
    sheetname='names and birthdays',
    fields='Name Birthday'.split(),
    normalize=[n.STRING(), n.INTEGER()],
)
fields = tr.get_fields()
```

`fields['Name']` returns `['John Doe', '123', 'Jane Doe', 'River Song']`

`fields['Birthday']` returns `[1995, 2003, 1964, 2019]`

Note: Here's a common "gotcha": Make sure to instantiate the normalization classes. That is, `normalize=[n.STRING(), n.INTEGER()]` instead of `normalize=[n.STRING, n.INTEGER]`

Create Custom Normalizing Classes

But let's say we don't want the full string from `Names`, but just the first name.

We could subclass either `NormalizeBase` or one of its subclasses. Let's subclass `STRING`.

```
# Continuing our code from above...

class FirstString(n.STRING)

    norm_func = n.STRING().normalize
    # Note the lack of parentheses after normalize
    # We do this here instead of in the normalize method
    # so that n.STRING gets instantiated only once.

    def normalize(self, value):
        # This is the function that gets called to norm your data.
        strings = norm_func(value).split()
        return strings[0]

tr.normalize = [n.FirstString(), n.INTEGER()]
fields = tr.get_fields()
```

`fields['Name']` returns `['John', '123', 'Jane', 'River']`

Pretty easy, right?

Classes

1.3.5 Exceptions

Lists of these functions can be passed as the `normalize` argument of TableReader.

1.4 Change Log

1.4.1 v0.8

- library deprecated; superseded by fuzzytable.

1.4.2 v0.7

- terminology change: `TableParser`, `FieldParser`, `WorksheetParser`

1.4.3 v0.6

- **TableReader:**
  - moved `path` and `sheetname` parameters from `get_fields` and `get_records` methods to `TableReader()`
  - converted normalize functions to classes.
  - updated documentation

1.4.4 v0.5

- **TableReader**
  - `read_from()` method renamed to `get_fields()`
  - new method: `get_records()`

1.4.5 v0.4

- **excelerator.normalize:**
  - `n.STRING()`
  - `n.INTEGER()`
  - `n.INTEGER_LIST()`
- add `normalize` parameter to TableReader

1.4.6 v0.3

- add `header_row_num` param to `TableReader`

CHAPTER 2

Indices and tables

- genindex
- modindex
- search