PyCells

Multidimensional Data Structures for Python

pycells_mds

User Manual

Mambetali DevTech

Version 0.2.0
November 13, 2025

Contents

1 Overview

`pycells_mds` is a lightweight Python engine that brings spreadsheet-like functionality into Python using SQLAlchemy ORM, NumPy-based formulas, Excel-like expressions, lists (sheets), tables, groups and Redis-powered cursors.

Each cell stores:

- cell name (e.g., `A1`),
- raw data or formula,
- computed value,
- optional group,
- optional style,
- belonging to a specific sheet and table.

2 Installation

```
pip install pycells_mds
```

3 Initialization

```
from pycells_mds.session import init_db
from pycells_mds.core import PyCells

db = init_db({"engine": "sqlite", "path": "demo.db"})
pc = PyCells()
```

4 User Management

```
user = pc.safe_register_user("demo", "1234", "demo@example.com")
user_id = user.id

# login
uid = pc.login("demo", "1234")
```

5 Tables and Sheets

5.1 Create or get a table

```
tbl = pc.ctable("Finance", user_id)
```

5.2 Create or get a sheet

```
sheet = pc.get_or_create_list("Finance", "Main", user_id)
```

6 Working with Cells

6.1 Write data

```
pc.write("Finance", "Main", "A1", "10", user_id)
pc.write("Finance", "Main", "A2", "20", user_id)
```

6.2 Write formulas

```
pc.write("Finance", "Main", "A3", "=A1+A2", user_id)
pc.write("Finance", "Main", "B1", "=SUM(A1:A3)", user_id)
```

6.3 Read computed value

```
val = pc.read("Finance", "Main", "A3", user_id)
print(val)    # 30.0
```

7 Supported Formulas

The engine supports:

- arithmetic: `+ - * /` *ranges* : `A1:A10`
- Excel-like functions:

```
SUM, MAX, MIN, AVERAGE
ABS, ROUND, POWER
IF
LEFT, RIGHT, CONCAT, TEXTJOIN
UPPER, LOWER, TRIM
TODAY, NOW, DATE, YEAR, MONTH, DAY
ETEXT (formatting)
```

Example:

```
pc.write("Finance", "Main", "C1", "=IF(A1>5,␣'YES',␣'NO')", user_id)
```

8 Recalculation

```
pc.recalc("Finance", "Main", user_id)
```

9 Groups

9.1 Create group

```
sheet = pc.get_or_create_list("Finance", "Main", user_id)
sheet.add_group("Totals", ["A1","A2","A3"], style="color:red;")
```

9.2 Update group style

```
sheet.update_group_style("Totals", "background:␣yellow;")
```

9.3 Get group cells

```
cells = sheet.get_group_cells("Totals")
print([c.name for c in cells])
```

10 Cursor Manager (Redis)

`CursorManager` tracks active cursor selection of cells. It uses Redis for instant state storage and SQL for history.

10.1 Activate cursor

```
from pycells_mds.managers import CursorManager

CursorManager.set_cursor(
    user_id=user_id,
    table_id=sheet.model.table_id,
    list_id=sheet.model.id,
    cells=["A1","A2"]
)
```

10.2 Get active/previous cursor

```
active = CursorManager.get_active(user_id)
previous = CursorManager.get_prec_cursor(user_id)
```

11 Project Structure

```
pycells_mds/
          core.py
          models.py
          wrappers.py
          session.py
          users.py
          managers/
                cursor_manager.py
          README.md
```

12 Conclusion

`pycells_mds` provides a full Python spreadsheet engine with tables, sheets, formulas, groups, and Redis-backed cursor logic.

For more information: https://pycells.com