CS 231
Naomi Nishimura

Module `grids.py`

1 Introduction

Just like the Python list and the Python dictionary provide ways of storing, accessing, and modifying data, a *grid* can be viewed as a way of storing, accessing, and modifying data. Because Python does not have built-in support for grids, I have supplied a module for use in the course.

At times you may be writing pseudocode that uses grid operations. To use a grid function, simply translate from dot notation and put the name in all capitals. For example, instead of `grid.enter(i, j, item)`, write ENTER*(grid, i, j, item)*.

2 Module `grids.py`

A grid stores `r` × `c` data items arranged in `r` rows and `c` columns. For a grid `grid`, the number of rows is `grid.rows` and the number of columns is `grid.cols`. We describe the row and column in which a data item can be found as its **position** in the grid; the values for the row range from `0` to `r` - 1 and the values for the column range from `0` to `c` - 1. To access and modify the data items, use the methods described below.

Grids can store any type of data. When you use `make_grid`, data items are entered as strings. A grid storing strings such as `"2"` and `"45"` can be converted to a grid storing integers by using the method `convert_grid_int`. To use a grid to store an image, each data item is a string of length one; a compact representation of the image can be formed using the method `create_image`.

2.1 Creating a grid from a file

The module contains the function `make_grid`, which consumes a string (the name of a file) and produces an object of type Grid.

A file should contain the following information, in this order:

- the number of rows in the grid (on one line)
- the number of columns in the grid (on one line)
- string entries for each row (one line each)

Each grid will be created with string data items. To convert a grid to one containing integers, use the method `convert_grid_int`, described below.

2.2 Creating a text file for a new grid

The module `grids.py` contains the function `make_grid_text_file`, which consumes two integers (`num_rows` and `num_cols`) and two strings (`content` and `file_name`) and produces a text file that can be used to create a grid. You may find this function useful when creating graphs to use when testing your code. This function is adapted from a function for graphs that was generously provided by Adam Hunter, who took the course in Spring 2018 and wished to make life easier for future students.

The four inputs should contain the following information:

- `num_rows` is a positive integer, the number of rows in the grid
- `num_cols` is a positive integer, the number of columns in the grid
- `content` is a string of characters of length the product of `num_rows` and `num_cols`, consisting of the contents of each row concatenated without spaces
- `file_name` is a suffix for the name of the file

The file will represent a grid as described; the name of the file will be `"testgrid"` concatenated with `file_name`.

For example, `make_grid_text_file(3, 4, "000011110000", "3-1")` will create a file with the name `testgrid3-1.txt` that contains the following data:

```
3
4
0 0 0 0
1 1 1 1
0 0 0 0
```

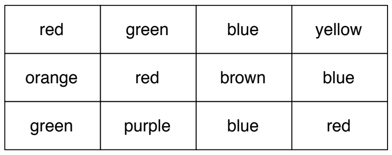
2.3 Methods

In the operations below, entering a data item replaces any data item that might have previously been stored in the same location. In an empty grid, the value `None` is stored in each location. Throughout, it is assumed that the grid `grid` has `r` rows and `c` columns.

The method `convert_grid_int` is used only for grids in which each data item is a string form of an integer (such as `"1"` or `"25"`), and the method `create_image` is used only for grids containing strings, ideally of length one.

The file `griduse.py` gives an example of the methods being used.

Because the module is designed to allow you to implement code with grids without considering the details of how the grid is implemented, the worst-case costs listed in the table are not intended to reflect the actual costs of this particular implementation. Accessing an attribute `grid.rows` or `grid.cols` can be accomplished in $\Theta(1)$ time.



red	green	blue	yellow
orange	red	brown	blue
green	purple	blue	red

Figure 1: Sample grid 1

Method	What it does	Cost
`Grid(r,c)`	creates a new empty grid of `r` rows and `c` columns, where $r > 0$ and $c > 0$	$\Theta(rc)$
`repr(grid)`	produces a string representation of the grid `grid`	$\Theta(rc)$
`grid.access(i, j)`	produces the data item in row `i` and column `j`	$\Theta(1)$
`grid.access_row(i)`	produces the data items in row `i` as a Python list	$\Theta(c)$
`grid.access_col(j)`	produces the data items in column `j` as a Python list	$\Theta(r)$
`grid.enter(i,j,item)`	enters `item` in row `i` and column `j`	$\Theta(1)$
`grid.enter_row(i,itemlist)`	enters the Python list of data items `itemlist` in row `i`	$\Theta(c)$
`grid.enter_col(j,itemlist)`	enters the Python list of data items `itemlist` in column `j`	$\Theta(r)$
`grid.convert_grid_int()`	mutates `grid` to replace strings with equivalent integers	$\Theta(rc)$
`grid.create_image()`	produces a string giving a compact representation of `grid`	$\Theta(rc)$
`grid_a == grid_b`	produces `True` if `grid_a` and `grid_b` store the same values and `False` otherwise	$\Theta(rc)$

3 Using the module to write code

3.1 Copying grids

If you wish to make a copy of a grid, import the `copy` module and use `copy.deepcopy`.

4 Sample grids

Sample grids have been provided for you in the files `samplegrid1.txt`, `samplegrid2.txt`, `samplegrid3.txt` and sample images have been provided in the files `sampleimage1.txt` and `sampleimage2.txt`. For your convenience, they have been illustrated here, with images shown using `create_image`. Code that you write for assignments should work for any grid, not just the samples provided.

10	3	6	23
4	67	45	9
78	1	34	2
65	27	7	8
45	2	12	69

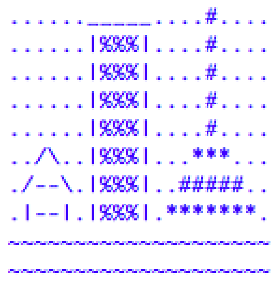
Figure 2: Sample grid 2

a0	a1	a2	a3	a4	a5	a6	a7	a8	a0
b0	b1	b2	b3	b4	b5	b6	b7	b8	b9
c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
d0	d1	d2	d3	d4	d5	d6	d7	d8	d9
e0	e1	e2	e3	e4	e5	e6	e7	e8	e9
f0	f1	f2	f3	f4	f5	f6	f7	f8	f9
g0	g1	g2	g3	g4	g5	g6	g7	g8	g9
h0	h1	h2	h3	h4	h5	h6	h7	h8	h9
i0	i1	i2	i3	i4	i5	i6	i7	i8	i9
j0	j1	j2	j3	j4	j5	j6	j7	j8	j9

Figure 3: Sample grid 3

```
0000000000
0100100000
0100110000
0111111000
0000111000
0000011100
1000001100
0100000000
0010000000
0010011000
0010011000
0010000000
0100000000
1000000000
0000000000
0000000000
0011111000
0011111000
0000000000
0000000000
```

Figure 4: Sample image 1



```
......_____....#....
......|%%%|....#....
......|%%%|....#....
......|%%%|....#....
......|%%%|....#....
../\..|%%%|...***...
./--\.|%%%|..#####..
.|--|.|%%%|.*******.
~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~
```

Figure 5: Sample image 2