Assignment 1

Basic Spreadsheet

**Data Structures and Algorithm**

# Introduction

In this assignment you will implement a simple spreadsheet program using a two dimensional python list *(explained in a section below)*. The user will issue some basic commands from the command line to operate on the spreadsheet.

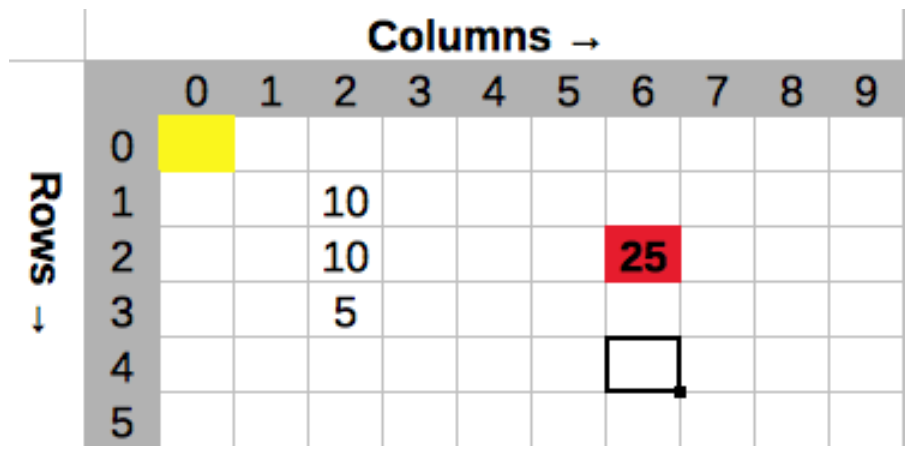
You are also given “a1\_starter.zip” which has a single “a1.py” python file. “a1.py” contains bare bones skeletal code which you will complete in this assignment.

***Important: You are encouraged to add any extra helper methods of member variables. But DO NOT modify the names of the classes and methods or the autograder will not work.***

## Internal representation

Internally the spreadsheet is represented as a *“rows of columns”*. Each row has a number of

columns. The upper left cell, colored yellow, has a position (0,0). The red cell containing “25” has a position of (2,6) because it is lying in 3rd row and 7th column:



Note that while specifying the position the row is mentioned first followed by column: *(row, col)*

There is also a “cursor” *(shown as a black rectangle in cell (4,6) triangle above)* which can move to any cell in the spreadsheet. You can freely move it around all over the document. If the user wants to insert or delete text then he first has to move the cursor to the desired position and then perform appropriate operations.

***NOTE: The above figure is only a visual representation of the spreadsheet. You do not have to implement any GUI whatsoever. All of the operations will be command line based.***

# Project Requirements

## Spreadsheet Working

The way the user enters data in this spreadsheet is through a series of commands in command prompt or console. Each command and its arguments are given in a “*single line*”. There are only a few commands:

|  |  |
| --- | --- |
| **Command** | **Arguments** |
| CreateSheet | row , col |
| Goto | row, col |
| Insert | data |
| Delete | - |
| ReadVal | - |
| Select | row, col |
| GetSelection | - |
| Sum | row, col |
| Mul | row, col |
| Avg | row, col |
| Max | row, col |
| PrintSheet | - |
| Quit | - |

### **CreateSheet**

Open up a1.py skeletal file. Following code is already filled in the Spreadsheet constructor. DO NOT modify these variables:

|  |
| --- |
| class Spreadsheet:  def \_\_init\_\_(self):  # WARNING: DO NOT CHANGE THE NAMES OR DELETE ANY VARIABLE  self.sheet = None # 2D array of Values  self.rows = 0 # number of ROWS in this spreadsheet   self.cols = 0 # number of COLUMNS in this spreadsheet  self.cursor = [0,0] # cursor's current position  self.selection = [None, None, None, None] # invalid selection |

The “sheet” data member represents the two dimensional list of values. Values can be integers or floats. “rows” and “cols” represent the total number of rows and columns in this spreadsheet. “cursor” member variable is a combination of two values represented in a list form. First value is the row number which the cursor is on. Second is the column number.

If you scroll down a bit you’ll see the “CreateSheet()” method. This method creates a 2D list and initializes it to the “self.sheet” member variable. It also sets “self.row” and “self.col” accordingly.

|  |
| --- |
| def CreateSheet(self, rows, cols):  #====================  # Add your code here:  raise NotImplementedError |

Complete the above method. This method will be called by the CreateSheet command. Again, commands are given by the user on the console when the program is running.

The CreateSheet command is used to create a new empty spreadsheet. It takes two arguments i.e. Max number of rows and Max number of columns per row:

>> CreateSheet 100 200

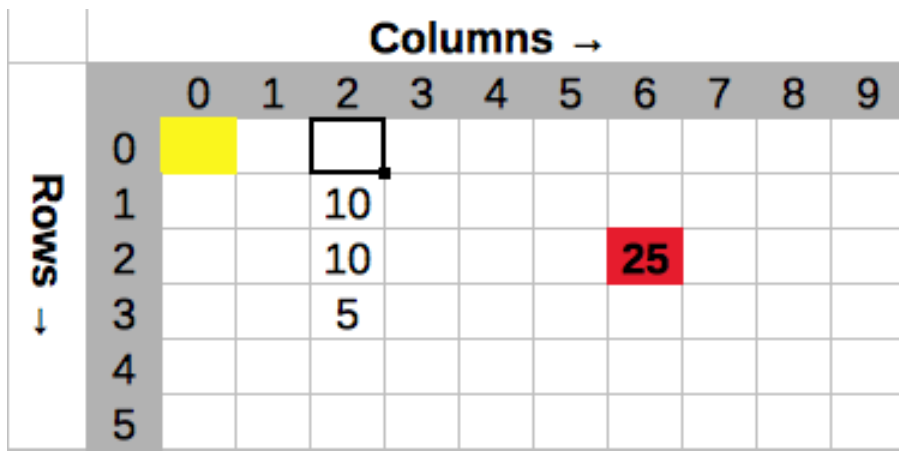
The above command will create a new sheet having 100 rows and 200 columns. Note that if you run any other command before creating a new sheet then it should display an error.

### **Goto**

The Goto command moves the cursor around. It takes two arguments i.e. row and column. The command:

>> Goto 0 2

Sets the cursor at the cell at row 0 and column 2:



For this command to work you will have to implement the Goto method:

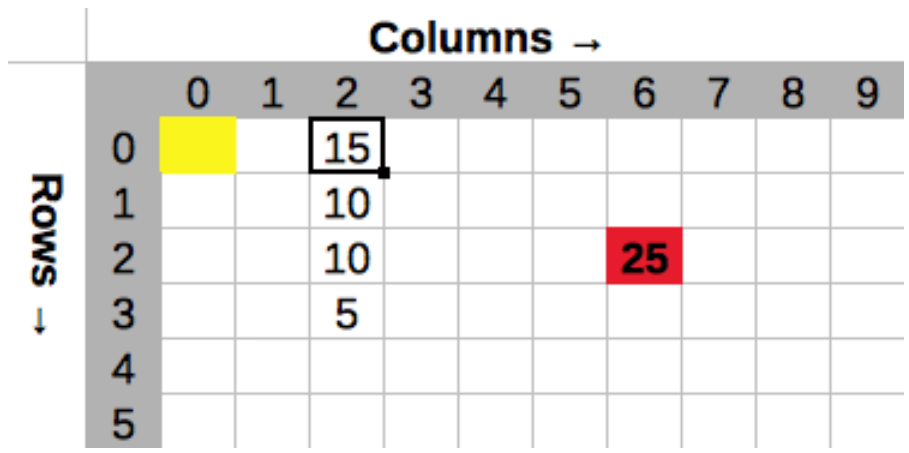
|  |
| --- |
| def Goto(self, row, col):   raise NotImplementedError |

### **Insert**

The Insert command takes a single argument: the data or a number to be entered. The command will then insert data at the cell pointed by the cursor. Suppose that the cursor was at position (0,2):

>> Insert 15

The insert command will write 15 at (0,2) cell pointed by cursor:



In this sheet assume that you can insert only numbers. The cursor position is not affected due to this command.

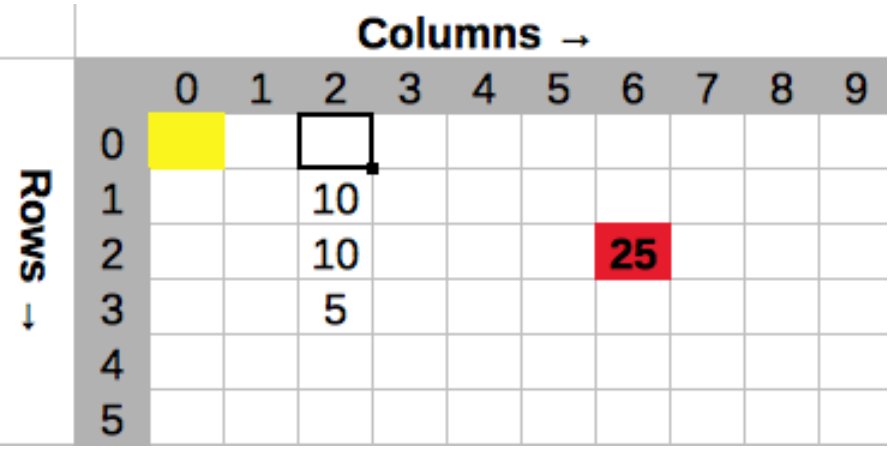
For the insert command to work you will have to implement the Insert() method. Insert takes a value to be inserted as a parameter. There is no return value.

|  |
| --- |
| def Insert(self, val):   raise NotImplementedError |

### **Delete**

Delete is the opposite of Insert. The Delete command erases the contents of the cell pointed by the cursor and resets it to empty. Again suppose that the cursor was already at (0,2):

>> Delete



The cursor position is not affected due to this command.

For the delete command to work you will have to implement the Delete() method:

|  |
| --- |
| def Delete(self):   raise NotImplementedError |

### **ReadVal**

The ReadVal command reads the contents of the cursor and prints the data on the screen. Suppose that cursor was at (1,2) this command will print 10:

>> ReadVal

10

The cursor position is not affected due to this command.

For the ReadVal command to work you will have to implement the ReadVal() method. It doesn’t take any parameters but it does return a value stored in the cursor cell:

|  |
| --- |
| def ReadVal(self):   raise NotImplementedError |

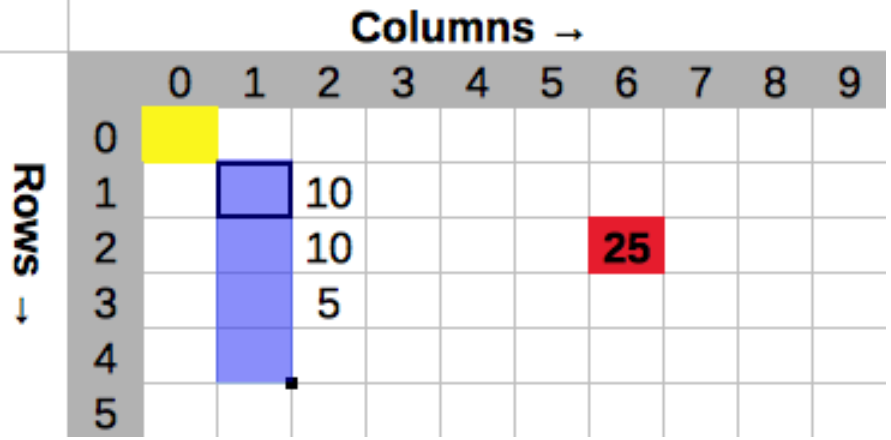
### **Select**

The Select command selects a rectangular region on the spreadsheet. “*The selection rectangle*” is defined by the upper left corner and the lower right corner. Upper right corner is the current cursor position. Lower right corner is given as an argument to the Select command. For example:

>> Goto 1 1

>> Select 4 1

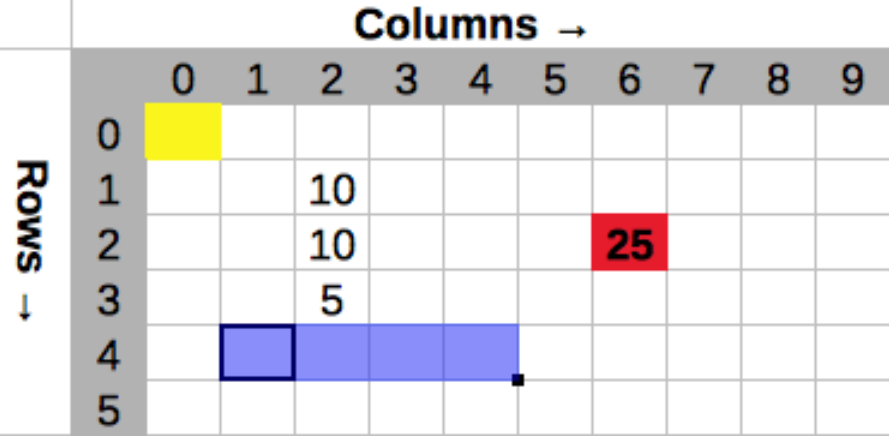
will select the following column *(the blue selection rectangle)*:



Assume that the cursor is already at position (4,1). The following command:

>> Select 4 4

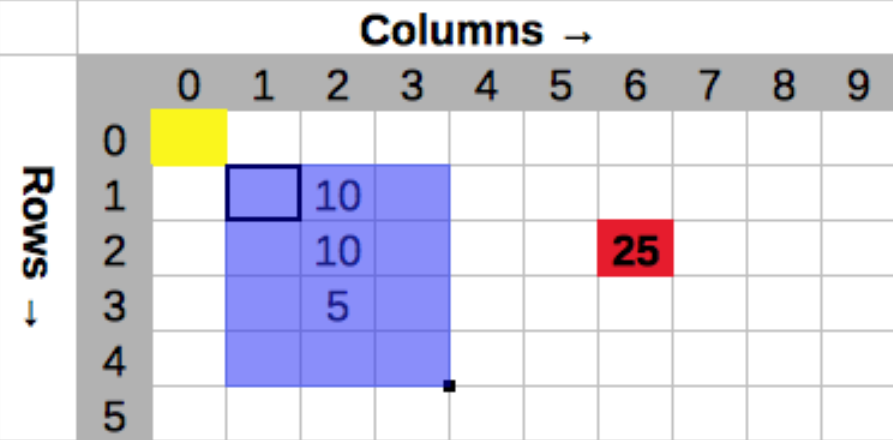
will select a portion of a row:



and we can also select multiple rows and columns:

>> Goto 1 1

>> Select 4 4



And as expected the following selection:

>> Goto 2 2

>> Select 2 2

will select a single cell.

***Note:*** *Once you execute the Select command it will remember its upper left corner of the selection rectangle even if the user changes the cursor using the Goto command afterwards. For example:*

>> Goto 1 1

>> Select 3 1

>> Goto 9 9

The selection rectangle in this case is still from (1,1) to (3,1). To change the selection rectangle you must call Select command again.

For the Select command to work you will have to implement the Select() method. This method takes two parameters i.e “row” and a “col” and modifies the “self.selection” member variable. It returns nothing.

|  |
| --- |
| def Select(self, row, col):   raise NotImplementedError |

### **GetSelection**

The GetSelection command prints the current selection in a comma separated format:

>> GetSelection

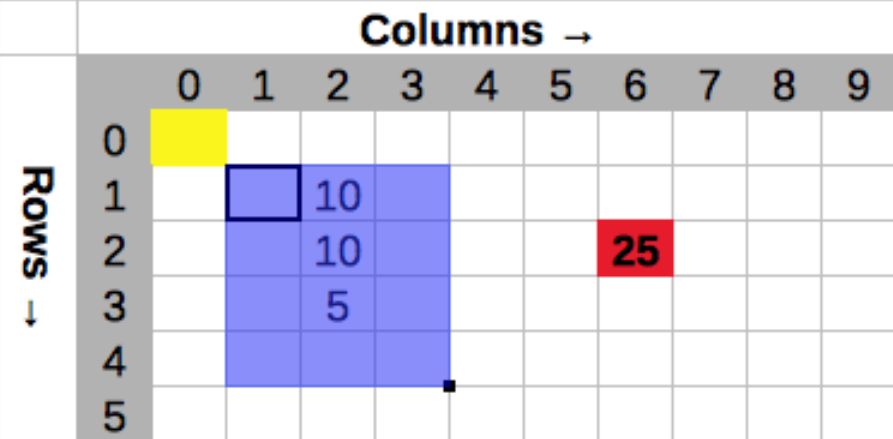
(1, 1, 4, 4)

For the GetSelection command to work you will have to implement the GetSelection() method. This method doesn’t take any parameters. It returns the current selection as a tuple of 4 values:

|  |
| --- |
| def GetSelection(self):   raise NotImplementedError |

### **Sum**

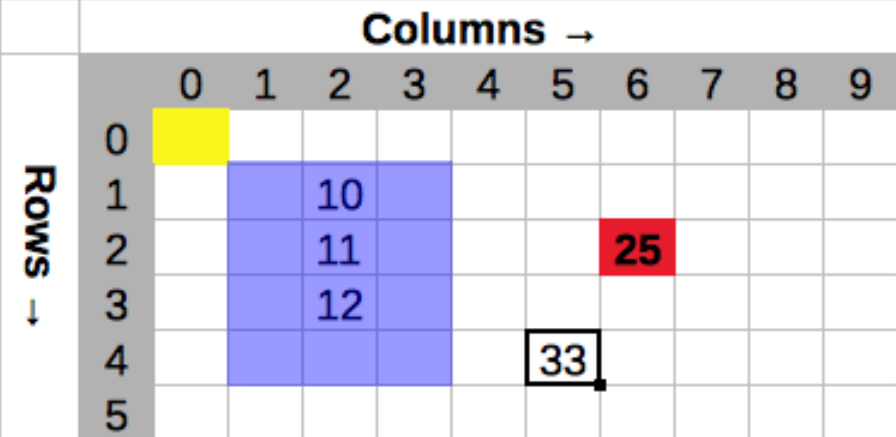
The Sum command adds all of the cells that fall under a current selection. Sum takes two arguments which are the coordinates for the position of the final cell. For example if we have the following selection:



Then:

>> Sum 4 5

Will add up all the numbers under the current selection *(ignoring empty cells of course)* and store the result at given location (4,5):



If a cell is empty then ignore its value in the final calculation.

If there is no valid selection then use only one cell that is under the cursor.

For the Sum command to work you will have to implement the Sum() method. This method takes two parameters i.e row and col of the destination cell to store the result. It returns nothing:

|  |
| --- |
| def Sum(self, row, col):   raise NotImplementedError |

### **Mul, Avg, Max**

The Mul, Avg, Max commands work exactly like the above Sum command. Each of these commands operate on a selection. They take two arguments which is the address of the final result cell. If you’ve successfully implemented Sum then implementing these commands will be trivial.

>> Mul 4 5

>> Avg 4 6

>> Max 4 7

>> Goto 4 5

>> ReadVal

33

>> Goto 4 6

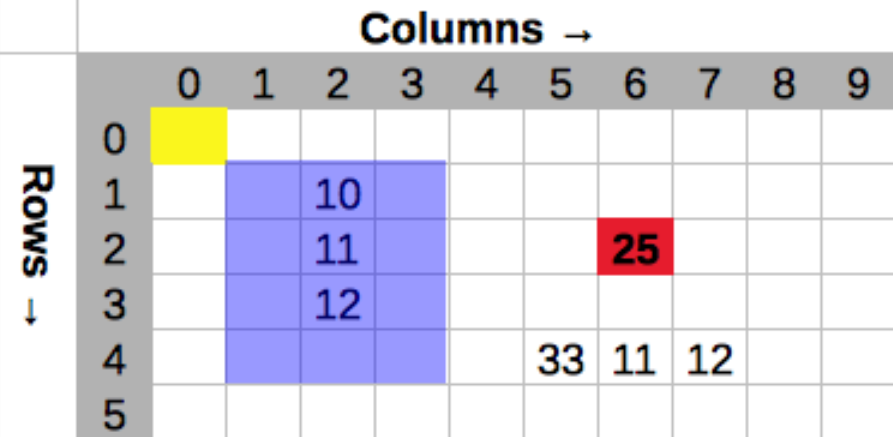
>> ReadVal

11

>> Goto 4 7

>> ReadVal

The following figure shows conceptual in-memory result of the above commands:



Just like the Sum command above, you’ll have to implement Mul(), Max() and Avg() methods for these commands to work properly.

### **PrintSheet**

This command prints the contents of the sheet on the screen. Of course if the sheet is big then the output is going to be a mess. But you can use this command as a debugging tool for small spreadsheets.

### **Quit**

The Quit command gracefully exits the program.

## Program Execution

Run your program as follows:

>> python a1.py

The program will print out a greeting message and then it will wait for the user to enter a series of commands. As soon as the user enters the command it will execute it and then displays the next prompt in a new line:

Welcome to DS SpreadSheet Program

Enter commands at the prompt

>> CreateSheet 10 10

Sheet successfully created !

>> Goto 1 2

>> Insert 10

>> Goto 2 2

>> Insert 11

>> Goto 3 2

>> Insert 12

>> Goto 1 2

>> ReadVal

10

>> Select 4 3

>> Sum 2 6

>> Goto 2 6

>> ReadVal

25

>> Goto 4 6

>> Avg 4 6

>> ReadVal

11

>> Quit

## Bonus

Implement following new commands:

|  |  |
| --- | --- |
| **Command** | **Arguments** |
| Undo | - |
| Redo | - |
| Save | file name |
| Load | file name |

### Explanation

* **Undo:** This command will undo the most recent operation. *Hint: use a separate stack. Whenever you execute a command, push it onto the stack along with the state of the cell that command is modifying.*
* **Redo:** We know that stack is built on top of a list. When popping the stack, if instead of doing list.pop(), we maintain a ‘top’ pointer and move that back and forth, we can easily implement the redo.
* **Save:** Saves the contents of the spreadsheet to a file.
* **Load:** Reads the contents of the spreadsheet from a file.

# Marks Distribution

**Total: *[ 180 marks ]***

| **Command** | **Marks** |
| --- | --- |
| CreateSheet | *20 marks* |
| Goto | *10 marks* |
| Insert | *10 marks* |
| Delete | *10 marks* |
| ReadVal | *10 marks* |
| Select | *20 marks* |
| GetSelection | *10 marks* |
| Sum | *20 marks* |
| Mul | *5 marks* |
| Avg | *5 marks* |
| Max | *5 marks* |
| PrintSheet | *10 marks* |
| Quit | *5 marks* |
| Good Commenting,  Crash free | *40 marks* |
|  | ***180 marks*** |
| Undo | *20 marks* |
| Redo | *10 marks* |
| Save | *5 marks* |
| Load | *5 marks* |
|  | ***40 marks*** |
| Total | ***220 marks*** |

**NOTE: There shouldn’t be any syntax errors in your code.**

# Submission

Please submit it on Moodle.