**CS 1120 Computer Science II (with Python),** Fall 2020  
Department of Computer Science, Western Michigan University  
Instructor: Wassnaa Al-Mawee, Lab TA: Utkarsha Prashant Kotkar

Andrew Kroll

**SOFTWARE LIFE CYCLE REPORT FOR LAB ASSIGNMENT 1**

**PHASE 1: SPECIFICATION (“What do we build?)**

Create a Python application which generates two two-dimensional array with the user specified dimensions between 5 and 20 rows (inclusive), 5 and 20 columns (inclusive), without columns and rows being equal. Fill one two-dimensional array with random values from 0 to 15 (inclusive). Calculate values for the second array based on the values of all neighbors for each cell in the random generated array. Print out the values from both the random and calculated arrays.

**PHASE 2: DESIGN**

**2.1 Modules (Classes) and Their Structure (Class Hierarchy)**

**LA1Main Class:**

Contains the **main** method which calls methods to get user input for row and column counts. Creates an instance of ProcessList. Calls methods in ProcessList object.

**ProcessList Class:**

Contains methods:

* **\_\_init\_\_**
  + Accepts row and column number parameters
  + Initializes row and column variables based on parameter input
  + Defines two zero-filled two-dimensional arrays with the given number of rows and columns, one for random numbers, one for computed numbers.
* **randomly\_fill\_list**
  + Fill the random value array with randomly generated numbers from 0 to 15 (inclusive).
* **compute\_list\_values**
  + Fill the computed value array with the sum of each cell’s neighbors from the random value array.
* **print\_list**
  + Prints the values of the random value array and computed value array, padded appropriately so each column lines up.
  + Getter/setter values methods for class attributes.

**2.2 Pseudocode for the Modules (Classes)**

**File LA1Main.py:**

# Method main: Main application method.

# Define placeholder variables for row and column counts

# Request user input for row count. Validate input. Continue requesting if invalid.

# Request user input for column count. Validate input. Continue requesting if invalid.

# Initialize ProcessList object with row and column counts

# Fill random value array in ProcessList object

# Fill computed value array in ProcessList object

# Display both random value array and computed value arrays

**File ProcessList.py:**

# Fields: rows, cols, random\_numbers, calculated\_numbers

# Constructor(rows, cols): Initialize ProcessList object

# Set object’s rows value based on passed value

# Set object’s column values based on passed value

# Define random\_numbers as a 0-filled array with the dimensions rows, cols

# Define calculated\_numbers as a 0 filled arrway with the dimensions rows, cols

# Method randomly\_fill\_list: Generate random numbers between 0 and 15 for each cell

# Iterate through row values

# Iterate through column values

# Set cell value to random number between 0 and 15

# Method compute\_list\_values: Set each cell’s value to the sum of its neighbors

# Iterate through row values

# Iterate through column values

# Define neighbor\_sum = 0 for calculating the sum of neighbor cells

# Iterate through neighboring cells

# add neighboring cell values to neighbor\_sum

# update cell’s value in computed\_numbers with neighbor\_sum

# Method print\_list: Display both random and computed arrays

# Print message indicating this is the array of random numbers

# Iterate through rows in array

# iterate through columns in array

# print value of cell with proper spacing

# print new line

# Print message indicating this is the computed array

# Iterate through rows in array

# Iterate through columns in array

# print value of cell with proper spacing

# print new line

# Method get\_rows: Return the object’s set rows value

# Method get\_cols: Return the object’s set cols value

# Method get\_random\_number(row, col): Return the cell at row, col’s value in random number array

# Method set\_random\_number(row, col, val): Update the cell’s value at row, col in random number array

# Method get\_computed\_number(row, col): Return the cell at row, col’s value in computed number array

# Method set\_computed\_number(row, col, val): Update the cell’s value at row, col in computed number array

**PHASE 3: RISK ANALYSIS (“What can go wrong, and how bad can it be?”)**

None.

**PHASE 4: VERIFICATION (“Are the algorithms correct?”)**

All input is validated. All “algorithms” are functional.

**PHASE 5: CODING**

**5a) Code Refinement #1 (class structure with pseudocode only; pseudocode is used as comments)**

**File LA1Main.py:**

# Method main: Main application method.

# Define placeholder variables for row and column counts

# Request user input for row count. Validate input. Continue requesting if invalid.

# Request user input for column count. Validate input. Continue requesting if invalid.

# Initialize ProcessList object with row and column counts

# Fill random value array in ProcessList object

# Fill computed value array in ProcessList object

# Display both random value array and computed value arrays

**File ProcessList.py:**

# Fields: rows, cols, random\_numbers, calculated\_numbers

# Constructor(rows, cols): Initialize ProcessList object

# Set object’s rows value based on passed value

# Set object’s column values based on passed value

# Define random\_numbers as a 0-filled array with the dimensions rows, cols

# Define calculated\_numbers as a 0 filled arrway with the dimensions rows, cols

# Method randomly\_fill\_list: Generate random numbers between 0 and 15 for each cell

# Iterate through row values

# Iterate through column values

# Set cell value to random number between 0 and 15

# Method compute\_list\_values: Set each cell’s value to the sum of its neighbors

# Iterate through row values

# Iterate through column values

# Define neighbor\_sum = 0 for calculating the sum of neighbor cells

# Iterate through neighboring cells

# add neighboring cell values to neighbor\_sum

# update cell’s value in computed\_numbers with neighbor\_sum

# Method print\_list: Display both random and computed arrays

# Print message indicating this is the array of random numbers

# Iterate through rows in array

# iterate through columns in array

# print value of cell with proper spacing

# print new line

# Print message indicating this is the computed array

# Iterate through rows in array

# Iterate through columns in array

# print value of cell with proper spacing

# print new line

# Method get\_rows: Return the object’s set rows value

# Method get\_cols: Return the object’s set cols value

# Method get\_random\_number(row, col): Return the cell at row, col’s value in random number array

# Method set\_random\_number(row, col, val): Update the cell’s value at row, col in random number array

# Method get\_computed\_number(row, col): Return the cell at row, col’s value in computed number array

# Method set\_computed\_number(row, col, val): Update the cell’s value at row, col in computed number array

**5b) Code Refinement #2 (still incomplete program: class and constructor/method structure with pseudocode only; pseudocode is used as comments)**

**File LA1Main.py:**

# Method main: Main application method.

# Define placeholder variables for row and column counts

# Request user input for row count. Validate input. Continue requesting if invalid.

# Request user input for column count. Validate input. Continue requesting if invalid.

# Initialize ProcessList object with row and column counts

# Fill random value array in ProcessList object

# Fill computed value array in ProcessList object

# Display both random value array and computed value arrays

**File ProcessList.py:**

# Fields: rows, cols, random\_numbers, calculated\_numbers

# Constructor(rows, cols): Initialize ProcessList object

# Set object’s rows value based on passed value

# Set object’s column values based on passed value

# Define random\_numbers as a 0-filled array with the dimensions rows, cols

# Define calculated\_numbers as a 0 filled arrway with the dimensions rows, cols

# Method randomly\_fill\_list: Generate random numbers between 0 and 15 for each cell

# Iterate through row values

# Iterate through column values

# Set cell value to random number between 0 and 15

# Method compute\_list\_values: Set each cell’s value to the sum of its neighbors

# Iterate through row values

# Iterate through column values

# Define neighbor\_sum = 0 for calculating the sum of neighbor cells

# Iterate through neighboring cells

# add neighboring cell values to neighbor\_sum

# update cell’s value in computed\_numbers with neighbor\_sum

# Method print\_list: Display both random and computed arrays

# Print message indicating this is the array of random numbers

# Iterate through rows in array

# iterate through columns in array

# print value of cell with proper spacing

# print new line

# Print message indicating this is the computed array

# Iterate through rows in array

# Iterate through columns in array

# print value of cell with proper spacing

# print new line

# Method get\_rows: Return the object’s set rows value

# Method get\_cols: Return the object’s set cols value

# Method get\_random\_number(row, col): Return the cell at row, col’s value in random number array

# Method set\_random\_number(row, col, val): Update the cell’s value at row, col in random number array

# Method get\_computed\_number(row, col): Return the cell at row, col’s value in computed number array

# Method set\_computed\_number(row, col, val): Update the cell’s value at row, col in computed number array

**5c) Code Refinement #3 (complete program – with complete fields/properties, code for constructor/methods)**

**LA1Main.py:**

*import* ProcessList  
  
  
*def* main():  
 rows = get\_rows() *# Get the number of rows from the user* cols = get\_cols(rows) *# Get the number of columns from the user* process\_list = ProcessList.ProcessList(rows, cols) *# Generate process\_list  
 # object* process\_list.randomly\_fill\_list() *# Fill the random numbers array with  
 # random numbers* process\_list.compute\_list\_values() *# Compute values for computed values  
 # array* process\_list.print\_list() *# Display both the random and computed arrays  
  
  
def* get\_rows() -> *int*:  
 rows = 0  
 *while not* 5 <= rows <= 20: *# Continue asking user for input until between  
 # 5 and 20 (inclusive)  
 try*:  
 rows = *int*(*input*("Enter number of rows in range [5, 20]: "))  
 *if not* 5 <= rows <= 20: *# Alert user if outside range  
 print*("Invalid input.")  
 *except ValueError*: *# Alert user if not an integer  
 print*("Input must be an integer.")  
 *return* rows  
  
  
*def* get\_cols(rows: *int*) -> *int*:  
 cols = 0  
 *while not* 5 <= cols <= 20 *or* cols == rows: *# Continue asking user for  
 # input until between 5 and 20 (inclusive), not equal to rows  
 try*:  
 cols = *int*(*input*("Enter number of cols in range [5, 20], "  
 "except {}: ".format(rows)))  
 *if not* 5 <= cols <= 20 *or* rows == cols: *# Alert user if outside  
 # range  
 print*("Invalid input.")  
 *except ValueError*: *# Alert user if not an integer  
 print*("Input must be an integer.")  
 *return* cols  
  
  
main()

**ProcessList.py:**

*import* random  
  
  
*class* ProcessList:  
  
 *def \_\_init\_\_*(*self*, rows: *int*, cols: *int*): *# Initializes the ProcessList  
 # object  
 self*.rows = rows *# Defines the number of rows  
 self*.cols = cols *# Defines the number of columns  
 self*.random\_numbers = [[0 *for* col *in range*(cols)]  
 *for* row *in range*(rows)] *# Define random\_numbers  
 # variable as a two dimensional array of 0's with rows rows and cols  
 # cols  
 self*.computed\_numbers = [[0 *for* col *in range*(cols)]  
 *for* row *in range*(rows)] *# Define  
 # computed\_numbers variable as a two dimensional array of 0's with rows  
 # rows and cols cols  
  
 def* randomly\_fill\_list(*self*): *# Fills the random\_numbers array with  
 # randomly generated numbers between 0 and 15  
 for* row *in range*(*self*.get\_rows()): *# Iterate through row values  
 for* col *in range*(*self*.get\_rows()): *# Iterate through col values  
 self*.set\_random\_number(row, col, random.randint(0, 15))  
 *# Set the cell to a random number between 0 and 15 (inclusive)  
  
 def* compute\_list\_values(*self*): *# Fills the computed\_numbers array with  
 # the sum of all neighboring cells from random\_numbers array  
 for* row *in range*(*self*.get\_rows()): *# Iterate through row values  
 for* col *in range*(*self*.get\_rows()): *# Iterate through col values* neighbor\_sum = 0 *# Used to temporarily hold the sum of all  
 # neighboring cells  
 for* n\_row *in range*(*max*(0, row-1), *min*(*self*.get\_rows(), row+2)):  
 *# Iterate through neighboring rows  
 for* n\_col *in range*(*max*(0, col-1), *min*(*self*.get\_rows(),  
 col+2)):  
 *# Iterate through neighboring cols  
 if* n\_row != row *or* n\_col != col:  
 *# Prevent adding current cell to neighbor cell sum* neighbor\_sum += *self*.get\_random\_number(n\_row,  
 n\_col)  
 *self*.set\_computed\_number(row, col, neighbor\_sum)  
 *# Set computed cell value  
  
 def* print\_list(*self*): *# Displays both the random\_numbers and  
 # computed\_numbers arrays with appropriate padding.  
 print*("Initial list with random numbers:")  
 *for* row *in range*(*self*.rows): *# Iterate through row values  
 for* col *in range*(*self*.cols): *# Iterate through col values  
 print*(" {:>3n}".format(*self*.get\_random\_number(row, col)),  
 end='') *# Print cell aligned to right with 3 digit  
 # padding  
 print*() *# Ends the current row's line  
 print*("Computed list:")  
 *for* row *in range*(*self*.rows): *# Iterate through row values  
 for* col *in range*(*self*.cols): *# Iterate through col values  
 print*(" {:>3n}".format(*self*.get\_computed\_number(row, col)),  
 end='') *# Print cell aligned to right with 3 digit  
 # padding  
 print*() *# Ends the current row's line  
  
 def* get\_rows(*self*) -> *int*: *# Gets the set number of rows  
 return self*.rows  
  
 *def* get\_cols(*self*) -> *int*: *# Gets the set number of cols  
 return self*.cols  
  
 *def* get\_random\_number(*self*, row: *int*, col: *int*) -> *int*:  
 *# Gets the targeted cell value from random\_numbers array  
 return self*.random\_numbers[row][col]  
  
 *def* set\_random\_number(*self*, row: *int*, col: *int*, val: *int*):  
 *# Sets the targeted cell value in random\_numbers array  
 self*.random\_numbers[row][col] = val  
  
 *def* get\_computed\_number(*self*, row: *int*, col: *int*) -> *int*:  
 *# Gets the targeted cell value from computed\_numbers array  
 return self*.computed\_numbers[row][col]  
  
 *def* set\_computed\_number(*self*, row: *int*, col: *int*, val: *int*):  
 *# Sets the targeted cell value in computed\_numbers array  
 self*.computed\_numbers[row][col] = val

**PHASE 6: TESTING (“Did we build it correctly?”)**

All input is validated upon entry. Non-decimal input is rejected, as well as decimal input outside specification.

Test output:

Enter number of rows in range [5, 20]: 0

Invalid input.

Enter number of rows in range [5, 20]: 100

Invalid input.

Enter number of rows in range [5, 20]: test

Input must be an integer.

Enter number of rows in range [5, 20]: 10

Enter number of cols in range [5, 20], except 10: 10

Invalid input.

Enter number of cols in range [5, 20], except 10: test

Input must be an integer.

Enter number of cols in range [5, 20], except 10: 0

Invalid input.

Enter number of cols in range [5, 20], except 10: 15

Initial list with random numbers:

11 2 10 15 1 15 13 3 3 3 0 0 0 0 0

4 14 13 4 10 3 0 8 9 10 0 0 0 0 0

15 12 5 14 5 2 6 7 10 12 0 0 0 0 0

2 14 15 0 5 12 2 13 1 2 0 0 0 0 0

14 6 0 3 11 9 3 10 2 13 0 0 0 0 0

2 8 8 13 2 13 12 7 10 13 0 0 0 0 0

0 2 5 15 12 12 7 4 9 9 0 0 0 0 0

14 9 15 4 2 13 6 4 5 7 0 0 0 0 0

5 4 4 12 11 10 7 8 1 1 0 0 0 0 0

12 2 15 1 12 15 6 0 1 12 0 0 0 0 0

Computed list:

20 52 48 38 47 27 29 33 33 22 0 0 0 0 0

54 72 76 73 59 52 57 51 56 37 0 0 0 0 0

46 82 86 57 50 43 47 49 62 32 0 0 0 0 0

61 69 54 58 56 43 62 41 69 38 0 0 0 0 0

32 63 67 54 57 60 78 50 69 28 0 0 0 0 0

30 37 52 56 88 68 65 57 67 43 0 0 0 0 0

35 61 74 61 74 67 71 60 59 44 0 0 0 0 0

20 49 55 76 89 67 65 47 43 25 0 0 0 0 0

41 76 62 64 69 72 62 30 38 26 0 0 0 0 0

11 40 23 54 49 46 40 23 22 3 0 0 0 0 0

Process finished with exit code 0

**PHASE 7: REFINING THE PROGRAM (“Add bells and whistles to the program”)**

Nothing to add.

**PHASE 8: PRODUCTION**

Uploaded, along with this document, to the dropbox.

**PHASE 9: MAINTENANCE**

If any maintenance is required, I may do so upon receiving feedback.