Odd Semester (2024)

A close up of a sign

Description automatically generated

**Assignment Cover Letter**

**(Individual Work)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |  | | | |  | | | |  | | | |
| **Student Information**: **Surname** | | | | | | | **Given Names**    **Leon** | | | | **Student ID Number**  **2440032392** | | | |  | | | |
|  | | **Jayakusuma** | | |  | | | |  | | | |  | | | |  | | |
|  |  |  | |  | | | |  | | | |  | | | |  | | |
| **Course Code** | **: COMP6502** |  | |  | | | | **Course Name** | | | | **: Introduction to Programming** | | | |  | | |
| **Class** | **: L1AC** |  | |  | | | | **Name of Lecturer(s)** | | | | **: Jude Joseph Lamug Martinez** | | | |  | | |
|  |  |  | |  | | | |  | | | |  | | | |  | | |
| **Major** | **: Computer Science** |  | |  | | | |  | | | |  | | | |  | | |
| **Title of Assignment**  (if any) | : **Matrix Calculator** | |  | | |  | | | |  | | | |  | | | |
| **Type of Assignment**    **Submission Pattern** | **: Final Project** |  | |  | | | |  | | | |  | | | |  | | |
| **Due Date** | **: 14-01-2020** |  | |  | | | | **Submission Date** | | | | **: 14-01-2019** | | | |  | | |

# Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

# Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

**“Matrix Calculator”**

**Name : Leon Jayakusuma**

**ID : 2440032392**

# Project Specification & Solution Design

## Definition

Many people are simply demotivated when studying math when asked to do extremely long calculations that are very prone to error. In my experience, what most people find interesting in math are the intuition and interpretation of the logical formulation that explains some general condition. The purpose of this software is to help ease the mental burden of calculations especially for those taking a linear algebra course and help them enjoy the intuition and meaning of the formulas instead. Another purpose of this software is to also help with the experimentation and or exploration of linear algebra.

There are mainly 3 features that are available in this software: numeric calculations, unary matrix operations, and binary matrix operations. Numeric calculations are your everyday arithmetic calculations, clear entry, delete, and number (0-9, Euler’s number, Pi) input. Unary matrix operations are operations that require 1 matrix as input. Binary matrix operations are operations that require 2 matrices as input.

## Program Design

Matrix Calculator is implemented by having a graphical user interface that runs on python. Tkinter module is used for the GUI. Meanwhile, Numpy and Scipy modules are used for the matrix calculations.

Matrix Calculator is designed to be user-friendly (easy to use by end users). I cannot imagine a scenario where an end user is able to learn how to use the software easily and fast if there is no graphical user interface. Also, people in math are already burdened with many thoughts about their assignments to prove theorems, etc. Therefore, it is highly unlikely for them to use a software that is hard to use (requires a considerable amount of time to learn). That is why I use graphical user interface.

Originally, I planned to also include features such as prime functionality (finding a random prime number, finding all primes less than a number, etc.) and calculus helper but I realized that it will be too complex and will need a huge amount of time to implement the algorithms myself and / or might be too hard to debug especially as my project grows.

# Implementation

Classes Diagram

1. Object Classes : Classes that are intended to instantiate objects for further manipulations.

Graphical user interface, diagram

Description automatically generated

* HoverButton: a wrapper for Tkinter Button widget, makes button able to change color on hover.
* Matrix: a wrapper for list of entry widgets to make it easier to manipulate and access the properties (width and length)
* MatrixWindow: a wrapper for Tkinter Toplevel widget.

1. Graphical user interface

   Description automatically generatedOperation Classes : Classes that are used to wrap operations into a class due to a need to access some object attribute repeatedly.

* UnOpsClassic: a wrapper for arithmetic unary operations
* BinOpsClassic: a wrapper for arithmetic binary operations
* UnOps Matrix: a wrapper for matrix unary operations
* BinOps Matrix: a wrapper for matrix binary operations

DIRECTORY ORGANIZATION

![A picture containing text

Description automatically generated]()Root directory contents Classes content

![Graphical user interface, application

Description automatically generated]()

Configurations and Functions contents Objects contents

![Text, letter

Description automatically generated]()![Text

Description automatically generated]()

## Running and installing Matrix\_Calculator

The installation of Matrix\_Calculator is simple. You can clone the software with

Git clone <https://github.com/LeonJ142857/I2P/tree/master/Matrix_Calculator>. Afterwards, mark the following folders as sources root in your IDE : root folder, Classes, Configurations and Functions, Objects, Classes / Object, Classes / Operation.

## Implementation of Matrix\_Calculator

Matrix\_Calculator is currently implemented using Tkinter, Numpy, and Scipy. The following are short explanations about the usage of each module in this software:

1. Tkinter

Tkinter is a module that comes prepackaged with python during initial installation. It is a module to create GUI applications for beginners thus have so much resource over the internet. Basically all objects that appear on the screen are called widgets. The widgets that I use are Tk, Frame, Button, Entry, Text, and Label. There is also a method called grid which basically displays the widgets that I have initialized into grid positions of the master object that I want (either a Tk window, Toplevel window or Frame).

Initially, the program will display a classical calculator interface. The window for this interface is the root window. In there, I gridded the buttons to the root window. I also create an entry object and grid it to the topmost row and make it span the whole column of the root window.

![A picture containing diagram

Description automatically generated]()

Each button runs a unique command (function object to be called) whenever clicked. For exam

ple, this is the command for the ‘9’ button.

This function is defined as a method of the UnOpsClassic class.

![Text, letter

Description automatically generated]()

…



…

![Graphical user interface, text

Description automatically generated]()

What this means is that whenever the button ‘9’ is clicked, the command(function reference) ‘lambda: un\_ops\_classic.ins\_val(9)’ will be called. I use lambda because lambda creates a function object and the command keyword only accepts a function object / reference, not the function itself. un\_ops\_classic.ins\_val(9) means insert argument 9 to the number parameter defined in the ins\_val method of un\_ops\_classic, an instance of UnOpsClassic class.

First, it will get the value of whatever is inside the io\_space, a Tkinter entry object, and store it in curr. Afterwards, it will clear the io\_space. Then we insert the number if the value is Euler’s number or pi. Else, we append the number 9 (because we insert curr concatenated with the string-cast value of the number that we want to insert, in this case 9 and curr contains the string of the previous value).

Here a picture to clarify what I mean:

This is right before I click the button ‘9’

![Table

Description automatically generated]()

This is after I clicked the button ‘9’.

![Table

Description automatically generated]()

Compare with these examples:

This is Euler’s number’s approximation up to 32 significant figures.

![Table

Description automatically generated]()

This is after I clicked 9

![Table

Description automatically generated]()

The result is different because I think people will rarely want to directly modify the value of Euler’s number or pi. So, instead of appending them at the end, I just remove the special numbers and insert the value of a digit instead.

All the buttons for inserting digits work this way too.

1. Numpy and / or Scipy

I use Numpy and / or Scipy to help with the matrix operations. I implemented them both in UnOpsMatrix and BinOpsMatrix classes. The way the operation works for unary matrix operations is this way : First, I initialize a Matrix object. The Matrix object has an attribute called entry\_list which is a list of Tkinter Entry objects. After that, I plug the entry\_list as an argument to each respective calculation commands for each matrix operation buttons. Inside each respective function, I used Numpy method numpy.array with list comprehension which creates an array of numbers of which Numpy supports matrix calculations with. With each respective command call of matrix operation button, I stored the result in a locally defined variable using numpy/scipy.(insert operation name). And then I insert the locally defined variable’s value to the output\_space Text widget. The difference between the Text widget and the entry widget is that the Text widget’s height is adjustable. And I need a sizeable output space for matrix operations compared to classical operations because matrices take up more space than numbers.

# Evidence of working program

This is when you open the program initially

![Table

Description automatically generated]()

insert 989

![Table

Description automatically generated]()

press multiply (\*)

![Table

Description automatically generated]()

insert 2

![Table

Description automatically generated]()

Pressed equal

![Table

Description automatically generated]()

Pressed backspace / delete (X) 3 times

![Table

Description automatically generated]()

Pressed M UnOps and inserted the entry numbers manually(without buttons) and then pressed DET for determinant.

![Table

Description automatically generated]()

![Table

Description automatically generated]()

Changed the size manually by deleting ‘3 x 3’ and typing ‘4 x 4’ then pressed the create m x n matrix button

![Table

Description automatically generated]()

Clicked M UnOps again or close and the window titled “Classic Calculator > Matrix Unary Calculator disappear.

![Table

Description automatically generated]()

Clicked M BinOps

![Text, table

Description automatically generated]()

Inserted 1 manually to every entry and pressing ADD.

![Table

Description automatically generated with medium confidence]()