# Pie-Num Language Proposal

Hadiah Venner (hkv2001)
Hana Fusman (hbf2113)
Ogochukwu Nwodoh(ocn2000)

**Motivation:**

Our motivation for our language is the functionality of NumPy which is a library for the Python programming language. NumPy adds support for large, multidimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. We want to create a static language that has some of the array manipulation power of NumPy. This would then allow us to write programs that involve manipulating arrays and matrices and doing complex mathematical calculations on them. This could be quite useful in a number of scientific and artistic fields such as machine learning, vector art and finance. We want to mimic and also enhance the ease of usage and flexibility in manipulating arrays that NumPy provides.

**Description of Language:**

Pie-Num is a static array manipulation language which implements some of the features of the Python library Numpy. It is an imperative language designed to have very intentional syntax, allowing the user to do complex array and matrix calculations without having to know exactly how these calculations are implemented.

The language contains functions for the creation of N-dimensional arrays and allows these arrays to be indexed and iterated over. This allows for matrix operations such as matrix multiplication, scalar multiplication, addition, subtraction and finding the inverse of a matrix. The language can be used to populate the arrays with strings as well as numbers. String operations such as string concatenation can be performed as well as various forms of statistical analysis such as the average and standard deviation can be calculated. The language will also have functions which make it possible to sort, search and count the elements of the arrays.

This functionality opens up a number of possibilities that we're hoping to implement. For instance, we're looking into being able to compute linear regression lines by finding the least-squares solution using matrix operations. And there is also the possibility of using matrix operations like scalar multiplication to tint and resize images.

**Summary of Parts of the Language**

- Array objects
    - N-dimensional array type which can be indexed using for example N integers.
        - The N-dimensional array
            - multidimensional container of items of the same type and size
        - Data type objects
            - Type of the data (integer, float, string etc.), Size of the data (how many bytes is in *e.g.* the integer), Byte order of the data (little-endian or big-endian)
        - Indexing
            - can be indexed x[obj] syntax
    - Iterating Over Arrays
        - To visit every element of an array, To visit the elements of an array in a specific order, To modify the array elements, (must specify either read-write or write-only mode)
- Routines
    - Array creation routines
        - Return a new array of given shape and type, without initializing entries.
        - Return a new array of given shape and type, filled with zeros.
        - Return a new array of given shape and type, filled with ones.
    - Array manipulation routines
        - Copies values from one array to another.
        - Gives a new shape to an array without changing its data
        - Join a sequence of arrays along an existing axis (concatenate).
        - Return a new array with sub-arrays along an axis deleted (delete).
        - Insert values along the given axis before the given indices.

        - Append values to the end of an array.
        - Return a new array with the specified shape
        - Reverse the order of elements in an array along the given axis.
        - Gives a new shape to an array without changing its data.
    - String operations
        - Return element-wise string concatenation for two arrays of str or unicode.
        - Return (a * i), that is string multiple concatenation, element-wise
        - Equal Return (x1 == x2) element-wise.
        - Not Equal Return (x1 != x2) element-wise.
        - Greater than or Equal Return (x1 >= x2) element-wise.
        - Less than or Equal Return (x1 <= x2) element-wise.
        - Greater Return (x1 > x2) element-wise.
        - Less Return (x1 < x2) element-wise.
    - Financial functions
        - Functions to calculate financial equations:
            - Future value, present value, IRR(Internal Rate of Return), Cash Flow series...
    - Linear algebra
        - Dot product of two arrays.
        - Matrix product of two arrays.
    - Mathematical functions
        - Sin, Cos, tan….
    - Matrix library
        - Interpret the input as a matrix.
        - Returns a matrix from an array-like object, or from a string of data.
        - Return a new matrix of given shape and type, without initializing entries.
        - Return a matrix of given shape and type, filled with zeros.

        - Matrix of ones.
    - Sorting, searching, and counting
        - Return a sorted copy of an array.
        - Counts the number of non-zero values in the array
    - Statistics
        - Min, Max, Percentile, Median, Average, STD,

**Language Syntax:**

- Special Symbols:

| **Symbol** | **Meaning** |
|---|---|
| ; | Denotes the end of a line of code |

- Primitive Data Type:

| **Type** | **Definition** |
|---|---|
| int | Integer value(32 bits) |
| float | Floating point value (32 bits) |
| boolean | true/false value(1 bits) |
| Short | two's complement integer (16 bits) |
| Long | two's complement integer (64 bits) |
| Double | floating point value (64 bits) |

- Arithmetic Operations:

| | |
|---|---|
| * | On two primitive data types this performs multiplication. On an array (1D or 2D) and a primitive data type this multiplies all values in the array by the primitive data type. Between two 1D arrays this calculates the dot product. Between two matrices this performs matrix multiplication between two matrices, matrix A and B. If matrix A's width is not equal to matrix B's height. |

| | |
|---|---|
| + | On two primitive data types this performs addition, on two 1D arrays this creates an array with the elements of both arrays. One two matrices with the same dimensions this creates a matrix with the elements of both matrices. This throws an error if done between matrices of different dimensions. This throws an error if done between a primitive data type and array. |
| - | On two primitive data types this performs subtraction, on two 1D arrays this creates an array with the elements of the matrix on the left hand side of the operator minus the elements on the right hand side of the operator. This throws an error if done between two matrices. |
| ** | Between two 1D arrays of length 3 this calculates the cross product. This throws an error if done between two matrices or 1D arrays that aren't both of length 3. |
| ^ | Creates a matrix that is the inverse of the matrix this is used on. Throws an error if done on a primitive data type, 1D array, or matrix that is not square. |

- Logical Operators (if applicable):
    - < strictly less than
    - <= less than or equal to
    - > strictly bigger than
    - >= bigger than or equal to
    - != not equal to
    - == if identical
- Comments:
    - # like in python
        - For example:
            - #this is a comment

- Function prototype/Function call:
    - returnType functionName(param1,param2 ...) { function body }
    - functionName(para1,para2 ...)
- Data Structures:
    - Arrays
- Control Flow:
    - if/else:
        - if (bool_expr) { Statement 1 }
        - elif (bool_expr) { Statement 2 }
        - else{ Statement 3 }
    - loops:
        - while (bool_expr) { Statement; }
        - for (val = a to b) { statement; }

**Example of Code:**

```
#declaring a matrix by using the array data structure
int array1[3][4];
int array2[4][3];

#filling the matrix with values
int m=0;
int n= 100;

for(int i=0; i<array.length(); i++) {
        for (int j=0; i<array[0].length();j++) {
                array1[i][j]=m++;
                array2[j][i]=n--;
        }
}

# This specifies that the value in an array is 6 in the first row and the fourth column :
int[0][3] = 6;

#matrix multiplication, creates a 3x3 square matrix
int array3 = array1 * array2;

#inverse of array3
int array4 = array3^;
```

```
#Indexing an array:
int k = array4[0][0];

#creating two arrays
int array5[3];
int array6[3];

#Iterating over every element in array:

for(int i = 0; i < array5.size(); i++)
{
        array5[i] =m++;
        array6[i]=n--;
}

#computing dot product
int b = array5 * array6;

#computing cross product
int array9 = array5 ** array6;

#Create an array with the elements of both arrays
array8 = array5 + array6;
```

Example Program that shows how to tint and resize an image in Pie-Num

```
#reads image into Pie-Num array
img = read_image(image.jpg);

#tint image by scaling each of the color channels
img_tinted = img * [1, .95, .9];

#resizes image to 200 x 200 pixels
img_tinted = resize_image(img_tinted, [200, 200]));

#save image
img_save  = (image.jpg, img_tinted);
```