Assignment 1

In this assignment, you will develop an ASCII-based function plotter. The objective of this assignment to practice array indexing and slicing, conditionals, loops and other concepts introduced in the first two weeks of the bootcamp. The plotter you will develop can be useful for visualizing the shape of any function on text-based environments (e.g., command-line applications) but is not designed to replace plotting utilities such as matplotlib.

Figure 1, Figure 2 and Figure 3 display examples of the expected plots. Plots can optionally display a grid (e.g., Figure 1), or just the data points. Each plot features *x* and *y* axes with minimum and maximum values as labels. Moreover, a plot may include a title and a legend at the top and at the bottom, respectively.

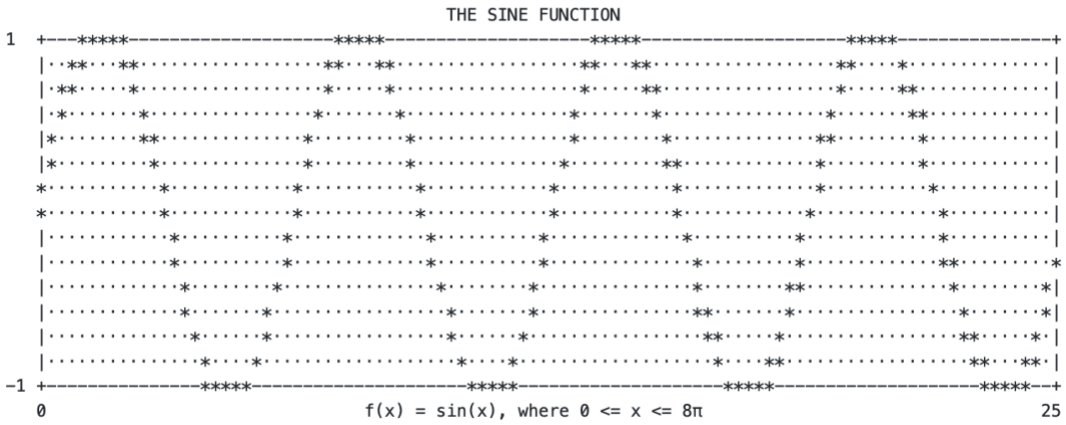

Figure 1. Example plot using the sine function

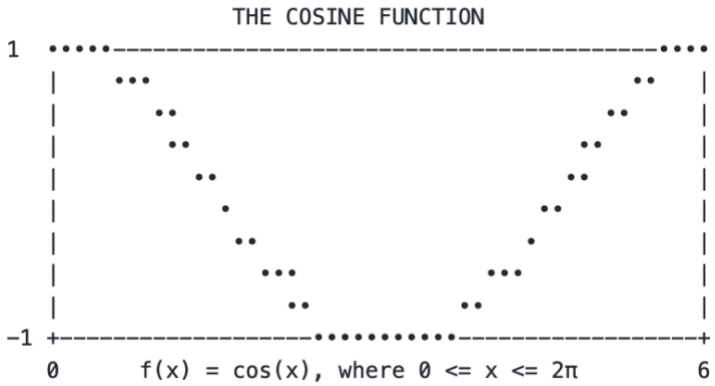

Figure 2. Example plot using the cosine function

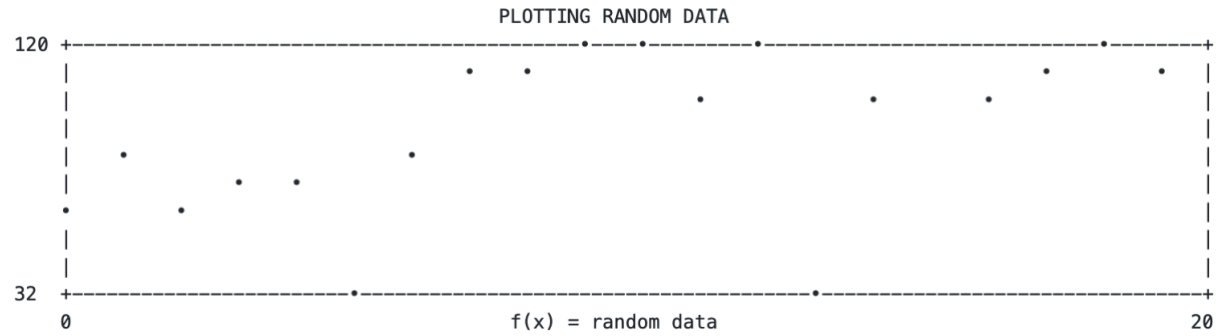

Figure 3. Example plot using randomly generated data

Objectives

- Learn top-down design using function decomposition
- Learn how to create parameterized functions
- Learn how to instantiate, index, slice, append and concatenate multidimensional arrays
- Learn how to use conditionals
- Learn how to use for loops for iteration
- Learn how to use relational and arithmetic expressions

Programming Instructions

Assignment 1 consists of seven parts, designed to develop skills for writing functions and loops, as well as manipulate multidimensional arrays. Each part requires at least one Python function, and your final submission will consist of one syntactically correct Python program. Each part is worth 1 mark, for a total of seven marks.

Part I: Printing a multidimensional array to the standard output

Create a Python function to print a multidimensional array to the standard output. The function's signature is expected to be as follows:

```
def print_chart(canvas: NDArray) -> None:
```

NDArray is a type provided by numpy, and can be imported as follows:

```
from numpy.typing import NDArray
```

The *canvas* parameter is a 2D character array containing each ASCII character in a plot (e.g., Figure 3). You can use the function *print_chart* to test your implementation of subsequent parts of this assignment. The following 2D array exemplifies the internal representation of an empty plot, without axes and labels, and its expected output:

Python array:

```
[["+", "-", "-", "-", "+"],
 ["|", " ", " ", " ", "|"],
 ["|", " ", " ", " ", "|"],
 ["+", "-", "-", "-", "+"]]
```

Expected output:

```
+---+
|   |
|   |
+---+
```

Hints

- There are multiple ways to turn an array into a string. You may use nested *for* or *while* loops, the string method *join*, or a combination of both. In any case, use a method to print the characters to the standard output, such as *print* or *pprint*.

Part II: Initializing the plot's main canvas (the cartesian plane)

Create a function for initializing a 2D array based on given dimensions *width* and *height*. Moreover, fill the array with appropriate ASCII characters as follows:

- The corners of the plot are represented with a plus (+)
- The left and right borders are represented with a pipe (|)
- The top and bottom borders are represented with a dash (-)
- Any other cell is represented with an empty space ( ) or a middle dot (·), depending on whether the plot should display a grid

For example:

Without a grid (4x10):

```
+--------+
|        |
|        |
+--------+
```

With a grid (4x5):

```
+---+
|···|
|···|
+---+
```

Hints

- The display of a grid is an optional parameter.
- Use numpy's multidimensional arrays to initialize the canvas. The following line creates an array of string characters of width *w* and height *h*:

```
numpy.empty((h, w), dtype="str")
```

- Use slicing as much as possible to fill the cells of the canvas

Part III: Normalizing and scaling the data points

Create a function to normalize and scale the data points as follows:

- Apply min-max normalization to each axis (separately)
- For each value in the x-axis, multiply it by the width of the canvas and then apply the floor function to the result
- For each value in the y-axis, multiply it by the height of the canvas and then apply the floor function to the result

Each axis should now contain integer values that can be used as indices within the canvas.

Hints

- The data points are represented as two 1D-array, one for values in the x-axis and another one for values in the y-axis.
- Numpy arrays provide easy access to min and max values from an array. Turn any given array into a numpy array using `np.asarray(x)`, where *x* is an instance of `List[Any]`.

Part IV: Drawing the data points on the canvas

Create a function to draw the data points on the canvas. That is, for each point (*x*, *y*), set the value of the corresponding cell with either a star (*) or a middle dot (•), depending on whether the plot should display a grid (e.g., Figure 1 and Figure 2, respectively).

For example, in a 4x10 canvas, the points (5, 2) and (6, 3) would look like this:

```
+-----•--+
|     •  |
|        |
+--------+
```

Hints

- Each data point (*x*, *y*) can be used as an index (*column*, *row*) in the canvas.
- Since the top row of the canvas represents the minimum value (i.e., 0), flip the *y* values.

Part V: Adding the x and y axes

Create a function to extend the main canvas to accommodate additional cells on the left and bottom sides. The left side corresponds to the y-axis and will contain the minimum and maximum values from the normalized and scaled *y* values. The bottom side corresponds to the x-axis and will contain the minimum and maximum values from the normalized and scaled *x* values. Moreover, the x-axis optionally contains a legend describing the plotted function. Place the legend in the middle of the axis.

The following are examples of the y and x axes, before concatenating them to the canvas.

The y-axis (4x3)

```
 1
 
-1
```

The canvas (4x10):

```
+---------+
|    •    |
|         |
+---------+
```

The x-axis (1x13)

```
  0 label  1
```

Hints

- The plot's legend is an optional parameter.
- Use numpy's functions *hstack* and *vstack* to concatenate two arrays horizontally and vertically, respectively.
- You can convert an integer into a string array using constructors *str* and *list*. For example, `list(str(-10))` returns `["-", "1", "0"]`
- Use array slicing to set various array cells at the same time (i.e., with another array).

Part VI: Adding a title

Create a function to extend the canvas to accommodate additional cells at the top, where the plot title, if any, will be placed. The title must be centered with respect to the canvas.

Hints

- The plot's title is an optional parameter.
- Use numpy to create a 1D array of width *w*:

```
numpy.empty(w, dtype="str")
```

- Use numpy's *vstack* to concatenate the canvas and the title's array vertically
- Use array slicing to set the cells of the array

Part VII: Verifying the depicted function shape with matplotlib

Use matplotlib to validate that that your plots are correct. Create an additional function to display a *matplotlib* plot, depicting the same *x* and *y* data passed to your plots. Figure 4 displays an example of such a plot.

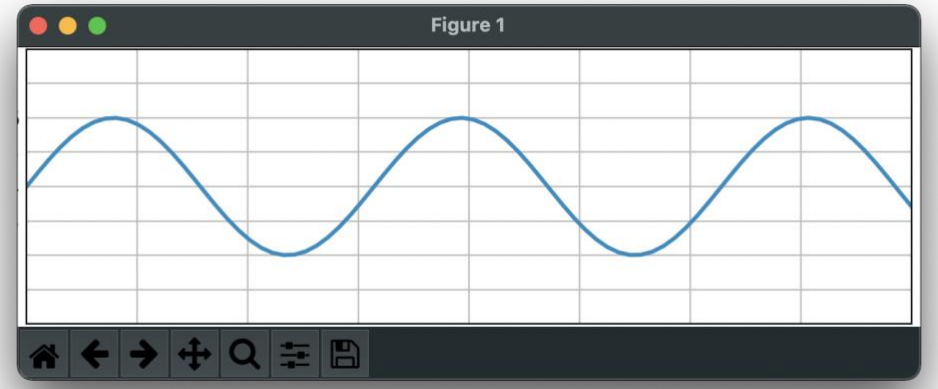

Figure 4. Example plot created with matplotlib

Submission

Use the functions you created in parts *I* through *VII* in a single function called *plot*. This function receives the following parameters:

- *x*: A list containing values for the x-axis
- *y*: A list containing values for the y-axis (same length as *x*)
- *display grid*: Whether to display the grid within the cartesian plane (default value: False)
- *height*: The plot's height, without including title and axes (default value: 15)
- *width*: The plot's width, without including title and axes (default value: 100)
- *title*: The chart's title (default value: empty string)
- *legend*: The function's legend (default value: empty string)

Use the following snippets to test your functions:

```
# Example 1
scale = 0.1
n = int(8 * math.pi / scale)
x = [scale * i for i in range(n)]
```

```
y = [math.sin(scale * i) for i in range(n)]
plot(x, y,
     display_grid=True,
     height=15,
     title="The sine function",
     legend="f(x) = sin(x), where 0 <= x <= 8π")


# Example 2
scale = 0.1
n = int(2 * math.pi / scale)
x = [scale * i for i in range(n)]
y = [math.cos(scale * i) for i in range(n)]
plot(x, y,
     width=50,
     title="The cosine function",
     legend="f(x) = cos(x), where 0 <= x <= 2π")


# Example 3
y = [ord(c) for c in 'ASCII Plotter example']
n = len(y)
x = [i for i in range(n)]
plot(x, y,
     title="Plotting Random Data",
     legend="f(x) = random data")
```