Name: Evan Leglar	**Lab Time** Tuesday 12:00 PM

Names of people you worked with:
• Aidan

Websites you used:
• https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.norm.html • https://numpy.org/ •

Approximately how many hours did it take you to complete this assignment (to nearest whole number)?	4

The Rules: Everything you do for this lab should be your own work. Don't look up the answers on the web, or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.

By writing or typing your name below you affirm that all of the work contained herein is your own, and was not copied or copied and altered.

Evan Leglar

Note: Failure to sign this page will result in a 50 percent penalty. Failure to list people you worked with may result in no grade for this lab. Failure to fill out hours approximation will result in a 10-percent penalty.

BEFORE YOU BEGIN

1) Make a new project for this lab
2) Make np_exercises.py
3) Browse through the Numpy quickstart page and familiarize yourself with some of the stuff you can do with Numpy. Pay particular attention to how it avoids using for loops in places you might have used them before.

Learning Objectives:

You should be able to do the following:

- Use vectorized numpy features to avoid using loops
- Compare numeric arrays
- Determine minimum/maximum values and the positions using numpy
- Draw large samples of random numbers in one line of code
- Filter and sort arrays using numpy

Thoughts (come back and read these after you've read the problems):

1. Numpy is full of features which make your life convenient. This assignment asks you to take full advantage of them, which means that you will need to figure out how to actually do that. For instance, previously when you sampled random numbers, you would import a function from random, make an empty list, and keep appending to that list after calling your random number generator. You should think about what "the Numpy way" is to do that.

2. Note that with Numpy arrays, if you try to do array_a == array_b, this does not return a value of True or False, but rather an array of True and False values. This means that the following lines of code will get you into trouble:

```
array_a = np.array([1,2])
if array_a == array_a:
    print('Hooray')
```

```
ValueError: The truth value of an array with more than one
element is ambiguous. Use a.any() or a.all().
```

   If you see this error, it's because Numpy arrays cannot be treated as Booleans like regular Python lists. As the error says, you should use the `any()` or `all()` methods, which return a single Boolean value corresponding to whether any/all of the values in the array are True.

Grading Checkpoints

For this assignment, you should not use loops! You should be using vectorized operations.

1. `numpy_close()` works for base case (1 point), mismatched dimensions case (1 point), and does NaN checking (1 point). [3 points total]*
2. `minimize_pow()` produces the correct values. [2 points total]*
3. `simulate_dice_rolls()` exhibits the expected behavior (1 point)* and produces a reasonable histogram for `simulate_dice_rolls(5, 2000)` (1 point). [2 points total]
4. `nearest_neighbors()` returns the correct set of neighbors (2 points)* and sorts them by distance in ascending order of distance (1 point) [3 points total]*
5. Your code does not use any loops. [3 points total, 1 point off for each loop detected, no lower bound]*
6. **Extra credit:** `nearest_neighbors()` works on arbitrary N x D dimensional arrays. [1 point total] *

* The autograder will give you immediate feedback if it's right.

Hand in all files to Gradescope.

Problem 1 Numpy Equals

For this assignment, create a new file called **np_exercises.py**. Put all functions in there.

Write a function called `numpy_close` which takes in two Numpy arrays `a` and `b`, as well as a third optional argument `tol` which defaults to 1e-8. It should return True if the arrays have the same shape and the absolute difference of each corresponding pair of elements less than `tol`.

- Your function should be able to work on arrays of arbitrary dimensions (they could be 1-D, 2-D, 8-D, etc.)
- You should also make sure that your code returns False for any pair of arrays in which one of the values is NaN.

Comments for grader/additional information (if any)

Problem 2 Optimize

Consider the function f(x) = x^x. **Write a function `minimize_pow`** which takes in a single argument, `n`. It should then evaluate f(x) at n evenly-spaced points between 0 and 1 (endpoint *inclusive*) and **return two values**: The x corresponding to the **minimum** value of f(x), followed by the actual minimum value. For example, if out of all the values you tested the minimum was f(0.5) = 2.4, you would return (0.5, 2.4).

Comments for grader/additional information (if any)

Problem 3 Dice roll simulation

Write a function `simulate_dice_rolls` which takes in two params, `num_rolls` and `iterations`. It should return a 1-D Numpy array of length `iterations`, where each item is the sum of throwing a fair 6-sided die `num` times.

It should also automatically output a histogram of the rolls to the file `dice_{0}_rolls_{1}.png`, where {0} is `num_rolls` and {1} is `iterations`. (No need for any labels, title, legends, etc.)

Run your function for num_rolls = 5 and rolls = 2000, and **upload dice_5_rolls_2000.png as a separate file** to Gradescope. **Do not paste it here.** (The autograder will warn you in Part 0 if you didn't upload it.)

Comments for grader/additional information (if any)

Problem 4 Nearest neighbors

Write a function `nearest_neighbors` which takes in three arguments: a N x 3 Numpy array, a 1-D array of length 3 `point`, and a distance threshold `dist`. It should return an M x 3 Numpy (M <= N) array of all points in the array which are within Euclidean distance `dist` of `point`.

The output points should be sorted by the distance from the corresponding point, with closest elements coming first. (You will receive partial credit if the correct points are output but not sorted.)

Example:

```
array = np.array([[1, 1, 1], [2, 3, 5], [0, 1, 1], [1.5, 1, 1], [10, 9,
9]])
target_pt = np.array([0, 1, 1])
nearest_neighbors(array, target_pt, 3.0)

# Should return [[0, 1, 1], [1, 1, 1], [1.5, 1, 1]]
```

Extra credit: Modify your function so that it is compatible with an N x D Numpy array (i.e. to return all D-dimensional points in the array whose D-dimensional Euclidean norm is less than `dist`.)

Comments for grader/additional information (if any)