

Name: Garrett DuBow	Lab Time F 1400
----------------------------	------------------------

Names of people you worked with:
<ul style="list-style-type: none">•

Websites you used:
<ul style="list-style-type: none">• https://www.geeksforgeeks.org/python-math-library-isclose-method/• https://docs.scipy.org/doc/numpy/reference/generated/numpy.argmax.html• https://realpython.com/numpy-array-programming/• https://www.geeksforgeeks.org/numpy-sum-in-python/• https://stackoverflow.com/questions/13682628/use-savefig-in-python-with-string-and-iterative-index-in-the-name• https://matplotlib.org/3.1.1/gallery/statistics/hist.html• https://www.w3schools.com/python/numpy_array_shape.asp• https://numpy.org/doc/stable/reference/generated/numpy.in1d.html•

Approximately how many hours did it take you to complete this assignment (to nearest whole number)?	9
--	---

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.

Garrett DuBow

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*. Functional iterators are also not allowed.

****Student assumed conditional blocks are not loops**

***Use numpy**

The following operations will be deducted: for, while, comprehensions (like [x for x in range(10)]), map(), and filter().

Assignment Grading Checkpoints (12 points total)

1. `numpy_close()` works for base case (1 point) and mismatched dimensions case (1 point). [2 points total]*
2. `simple_minimizer()` produces the correct values. [3 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. `is_transformation_matrix` returns the correct values. [2 points total]*
5. `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]*
6. Your code does not use any loops or functional iterators. [1 point off for each loop detected, no lower bound]*

Standard Grading Checkpoints (3.5 points total)

- [0.5 points] Turn in a properly filled-in PDF to Gradescope.
- [1 point]* Code passes PEP8 checks with 10 errors max.
- [2 point] Code passes TA-reviewed style checks for cleanliness, layout, readability, etc.

* The autograder will give you immediate feedback if it's right.
Hand in all files to Gradescope.

Problem 2 Minimization

Write a function `simple_minimizer` which takes in four arguments:

- A function reference `func` which represents a 1-D function, e.g. $f(x) = x^2$. You can assume that it also will work on a Numpy array, so that if you do `func(x_values)`, you will get a corresponding array of y values.
- A `start` and `end`, floats which represent the region to search for a minimum. (Throw a `ValueError` if `start > end`.)
- An optional param `num` which defaults to 100.

It should then evaluate `func` at `num` evenly-spaced points between `start` and `end` (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)`.

Example:

```
my_func = lambda x: x**2
simple_minimizer(my_func, -1.75, 2.25, num=5)
# Should return (0.25, 0.0625)
```

Comments for grader/additional information (if any)

Problem 3 Dice roll simulation

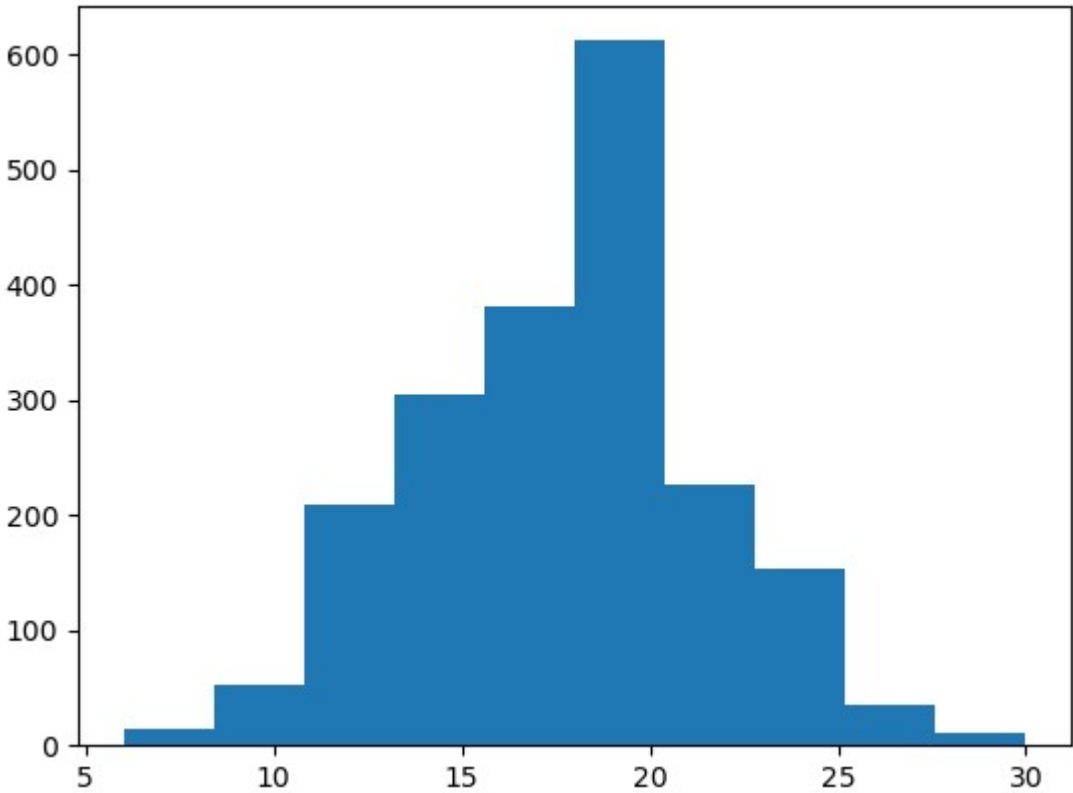
Write a function `simulate_dice_rolls` which takes in two params, `num_rolls` and `iterations`. It should return a 1-D Numpy array of length `iterations`, where each item is the sum of throwing a fair 6-sided die `num_rolls` times.

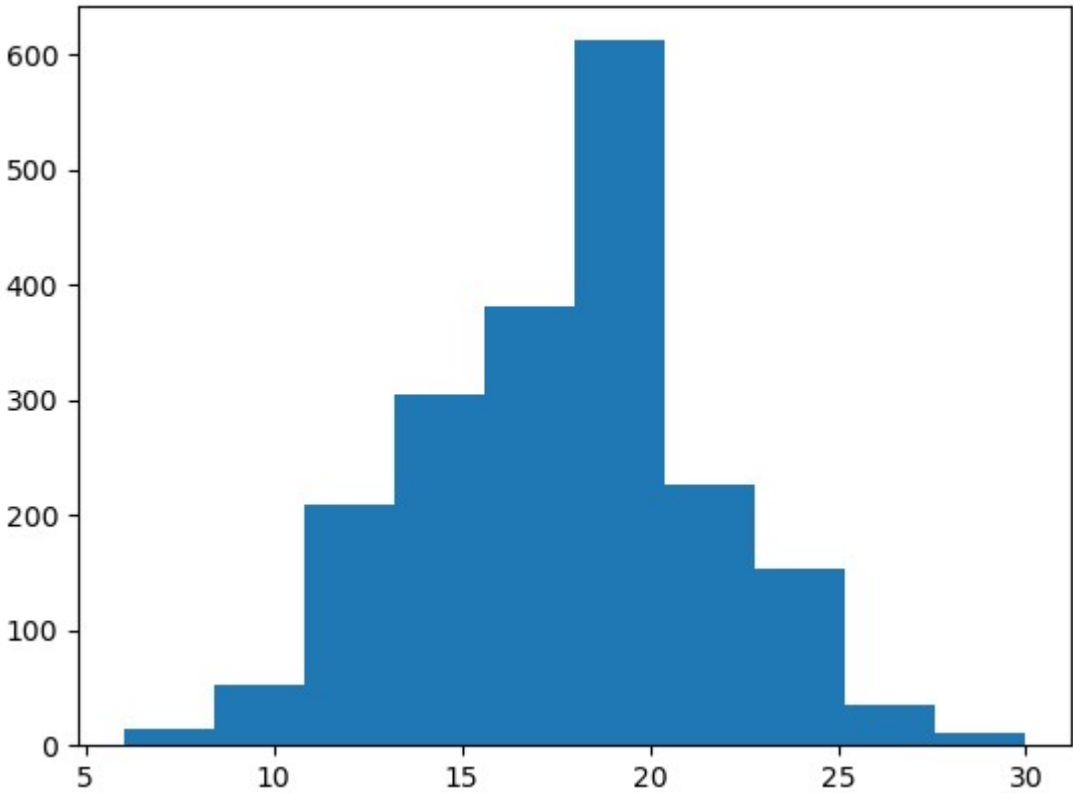
It should also automatically output a histogram of the rolls to the file

`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 `iterations = 2000` and paste the plot here.

Comments for grader/additional information (if any)

<code>dice_5_rolls_2000.png</code>




Problem 4 Transformation matrix

Recall that a transformation matrix T is a 4x4 matrix which can be used to represent the position and orientation of a coordinate frame relative to a base frame. A transformation matrix has the following form:

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

p is a 3x1 matrix in \mathbb{R}^3 , and R is a 3x3 *rotation* matrix. All rotation matrices have the property that the *transpose* of R is equal to its *inverse*, i.e. $R^T R = I$, where I is the 3x3 identity matrix.

Write a function `is_transformation_matrix` that takes in a 4x4 Numpy array and returns True if it's a valid transformation matrix, and False otherwise.

Example:

```
tf_valid = np.array([
    [0, 0, -1, 4],
    [0, 1, 0, 2.4],
    [1, 0, 0, 3],
    [0, 0, 0, 1]
])

tf_invalid = np.array([
    [1, 2, 3, 1],
    [0, 1, -3, 4],
    [0, 1, 1, 1],
    [-0.5, 4, 0, 2]
])

print(is_transformation_matrix(tf_valid)) # True
print(is_transformation_matrix(tf_invalid)) # False
```

Comments for grader/additional information (if any)

