PHY1112: Assignment 4

> Be classy and roll with it

Assigned: January 30th, 2024

Due: February 6th, 2024

Learning Objectives

1. Making new methods for your classes
2. Practice importing functions from other modules

Grade Breakdown

|  |  |  |  |
| --- | --- | --- | --- |
| Part | 1 | 2 | Total |
| Points | 15 | 16 | 31 |
| Score |  |  |  |

**Question 1: Stay classy.**

1. Write a new custom class, Vector3, which is like the Vector2 custom class you wrote in lab, but now is three dimensional with x, y, and z components (i.e., data attributes).

Write \_\_str\_\_() and magnitude() methods for your Vector3 class, similar to what you did for your Vector2 class in the lab. The formula for the magnitude in three dimensions is .

Demonstrate they work with the following vector, copying a screenshot of your results into your assignment solutions document:

**(4 marks)**

**A screenshot of a computer program

Description automatically generated**

1. Write two special methods \_\_add\_\_() and \_\_sub\_\_() that perform, respectively, vector addition and subtraction between the first Vector3 instance and a second Vector3 instance. They are invoked by Python when the + and – operators are used with Vector3 objects.

Demonstrate that your \_\_add\_\_() and \_\_sub\_\_() methods work with the following vectors, copying a screenshot of your results into your assignment solutions document:

,

**(3 marks)**

**A screenshot of a computer program

Description automatically generated**

1. Write another method called inner\_product(), that performs an inner product between the first Vector3 instance and a second (sometimes called the dot product).

Demonstrate that it works using the and vectors above, copying a screenshot of your results into your assignment solutions document.

**(2 marks)**

**A screen shot of a computer program

Description automatically generated**

1. Using your inner\_product() and \_\_add\_\_() methods, show that the following is true:

.

Copy a screenshot of your results into your assignment solutions document. Use the and vectors as above, as well as

**(2 marks)**

**A computer screen with white and purple text

Description automatically generated**

1. Write another method that calculates the angle between two Vector3 objects and call it angle\_between(). Recall that the formula for the angle between two vectors and is:

where the dot operator indicates the inner product, and the | | indicates you are taking the magnitude. Hint: This will also require the math module.

What is the angle between   and ? Give the answer in both in radians and degrees, and copy a screenshot of your results in your assignment solutions document.

**(3 marks)**

A screenshot of a computer program

Description automatically generated

**(15 marks total, 1 for doc strings/file header/variable naming/comments)**

CODE

'''

Filename:       a4\_300349204.py

Author:         Patrick Geraghty

Date Created:   2024-02-01

Date Modified:  2024-02-01

Description:    Contains the Vector3 class, which initializes a 3D vector and contains methods to calculate the magnitude, addition, subtraction, inner product, and angle (rad and deg) between two vectors.

'''

# Begin by importing the math module. Import as m for brevity.

import math as m

# Define the Vector3 class.

*class* Vector3:

    # Initialize the class with x, y, and z coordinates.

*def* \_\_init\_\_(*self*, *x*, *y*, *z*):

        self.x = x

        self.y = y

        self.z = z

    # Define the \_\_str\_\_ method to return the vector as a string.

*def* \_\_str\_\_(*self*):

        return '(' + *str*(self.x) + ', ' + *str*(self.y) + ', ' + *str*(self.z) + ')'

    # Define the magnitude method to return the magnitude of the vector.

*def* magnitude(*self*):

        return (self.x\*\*2 + self.y\*\*2 + self.z\*\*2)\*\*0.5

    # Define the addition and subtraction methods to return the sum and difference of two vectors, respectively.

    # Addition = (x1+x2, y1+y2, z1+z2), Subtraction = (x1-x2, y1-y2, z1-z2)

*def* \_\_add\_\_(*self*, *other*):

        return Vector3(self.x + other.x, self.y + other.y, self.z + other.z)

*def* \_\_sub\_\_(*self*, *other*):

        return Vector3(self.x - other.x, self.y - other.y, self.z - other.z)

    # Define the inner\_product method to return the inner product of two vectors.

    # Inner product = x1\*x2 + y1\*y2 + z1\*z2

*def* inner\_product(*self*, *other*):

        return self.x \* other.x + self.y \* other.y + self.z \* other.z

    # Define the angle\_between\_rad method to return the angle between two vectors in radians.

    # Angle = arccos((a \* b) / (magnitude(a) \* magnitude(b)))

*def* angle\_between\_rad(*self*, *other*):

        return m.acos(self.inner\_product(other) / (self.magnitude() \* other.magnitude()))

    # Convert the angle between two vectors from radians to degrees.

    # Angle = degrees(angle\_between\_rad)

    # Why do math when you can just use the math module?

*def* angle\_between\_deg(*self*, *other*):

        return m.degrees(self.angle\_between\_rad(other))

**Question 2: Just roll with it.**

In this question, we will make use of the time and itertools modules to compute how long it takes to do task via nested loops versus via vectorization.

Consider four N-sideddice being rolled, where N is something that will vary so should be implemented as a variable you can change the value of each time you run your program.

1. Usingfour nested for loops, create a list of every possible combination of dice rolls. The entries of this list should be tuples, where the tuples each contain four numbers (that is, one number for each die that is rolled).

Report how many combinations there are for four 20-sided dice (that is N=20).

160000 combinations

**(3 marks)**

1. Inside your four for loops from part a, also create a list that contains as entries the sum value of the four dice for each combination. That is, add up the four numbers in each of your tuples and put all the sums in a new list.

**(2 marks)**

1. Write code to determine the average of all the sums, and run it for 20-sided dice. Report your answer including a screenshot.

Average sum of 42

**(1 mark)**

1. Now, do the same thing, but with vectorization. First, use itertools.product() to generate your list of all possible combination of dice rolls, where the entries in the list are tuples each containing the four numbers.

Report how many combinations there are for four 20-sided dice (that is N=20)

160000 combinations

**(3 marks)**

1. As in part b, using this list of tuples, create a list that contains as entries the sums of all the combinations, that is, the sum value of numbers in each tuple. Accomplish this using at most one for loop.[[1]](#footnote-1)

**(2 marks)**

1. Determine the average of all the sums with no loop. Again, use N=20 and put a screenshot of your results into your assignment solution document.

A screen shot of a computer program

Description automatically generated

**(1 mark)**

1. Use the time module to determine how long it took to run the calculations in the nested for-loop method (parts a,b,c) and how long it took to run the calculations with vectorization (parts d,e,f).

Run your code three times for N=10. You should now have three time measurements for the for nested loop method, and three time measurements for the vectorization method.

Use these to calculate an average run time for nested for loops, as well as an average run time for vectorization, and record them in the table below.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Repeat for N=20 and N=50, continuing to fill in your results in the table below.

For each N, also report the number of combinations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number of faces** | **Number of rolls** | **Number of combinations** | **Nested for loops average runtime** | **Vectorization average runtime** |
| 10 | 4 | 10000 | 0.0025302999953661733 | 0.0026337666689263037 |
| 20 | 4 | 160000 | 0.04353406666389977 | 0.04837156667296464 |
| 50 | 4 | 6250000 | 1.9704985333325264 | 1.88940270000118 |

Which method is faster? Comment on what you observed.

Itertools appears to be faster with large sets of data, as it only yielded a faster runtime when calculating for d50s. For both the d10s and d20s, the nested for loops yielded better results.

**(3 marks)**

**(16 marks total, 1 for doc strings/file header/variable naming/comments)**

CODE:

'''

Filename:       a4\_part2.py

Author:         Patrick Geraghty

Date Created:   2024-02-05

Date Modified:  2024-02-05

Description:

'''

# Import the itertools and time modules.

import itertools as it

import time as t

# Define a function to roll four n-sided dice using a for loop.

*def* for\_loop\_dice\_roller(*n*):

    '''

    (int) -> list, list

    Takes an integer n and returns a list of all possible combinations of four n-sided dice rolls.

    Precondition: n is an integer.

    '''

    # Create an empty list to store the combinations and a list to store the net sum of each combination.

    combinations = []

    net\_sum = []

    # Iterate through the range of n four times.

    for i in range(n):

        for j in range(n):

            for k in range(n):

                for l in range(n):

                    # Append each combination to the list.

                    combinations.append((i+1, j+1, k+1, l+1))

                    # Append the sum of each combination to the list.

                    net\_sum.append(i+j+k+l+4)

    # Return the list net\_sum and combinations.

    return net\_sum, combinations

# Define a general function to calculate the average of a list of net sums.

*def* average\_roll(*net\_sum*):

    '''

    (list) -> float

    Takes a list of net sums and returns the average net sum.

    Precondition: net\_sum is a list of integers.

    '''

    # Return the average of the net sums.

    return sum(net\_sum) / len(net\_sum)

# Define a function to roll four n-sided dice using the itertools module.

*def* iter\_dice\_roller(*n*):

    '''

    (int) -> list, list

    Takes an integer n and returns a list of all possible combinations of four n-sided dice rolls.

    Preconditions: n is an integer.

    '''

    # Create an empty list to store the combinations and a list to store the net sum of each combination.

    combinations = []

    net\_sum = []

    # Iterate through the range of n four times.

    for i in it.product(range(1, n+1), *repeat*=4):

        # Append each combination to the list.

        combinations.append(i)

        # Append the sum of each combination to the list.

        net\_sum.append(sum(i))

    # Return the list net\_sum and combinations.

    return net\_sum, combinations

# Test the functions.

# Initiate a for loop to test the functions for 10, 20, and 50-sided dice.

for n in [10, 20, 50]:

    # Record the time it takes to run the for loop functions.

    t0\_for = t.perf\_counter()

    for\_loop\_dice\_roller(n)

    average\_roll(for\_loop\_dice\_roller(n)[0])    # Note that the average\_roll function takes the first element of the tuple returned by for\_loop\_dice\_roller as that is the element which contains the net sum of each combination.

    t1\_for = t.perf\_counter()

    # Record the time it takes to run the itertools functions.

    t0\_iter = t.perf\_counter()

    iter\_dice\_roller(n)

    average\_roll(iter\_dice\_roller(n)[0])    # Note that the average\_roll function takes the first element of the tuple returned by for\_loop\_dice\_roller as that is the element which contains the net sum of each combination.

    t1\_iter = t.perf\_counter()

    # Print the stats for each n-sided dice.

    print(*f*'Stats for {n}-sided dice:')

    print(*f*'Total number of combinations: {len(iter\_dice\_roller(n)[1])}')

    print(*f*'For loop time: {t1\_for - t0\_for}')

    print(*f*'Itertools time: {t1\_iter - t0\_iter}')

    print()

1. Note that you can use list comprehension here to avoid an official loop altogether, but we will not be covering that officially in this course. [↑](#footnote-ref-1)