Lab 3 Functions, arithmetic expressions, and scalar product (09.16.19) CS103 Fall 2019

author john k johnstone jkj at uab dot edu
course CS103 Fall 2019

version Fall 2019

---

materials

- lab03_19fa103.pdf (this document)
- lab03_19fa103.py (with functions you will implement)

lab partner

Please work with the lab partner that you chose or were assigned in Lab02.

purpose

In a preparatory phase, you will make sure your computer is ready to work on HW2. The TA's will give a demo beforehand.

In the heart of this lab, you will build Python functions related to *scalar product*.
This will allow you to:

- practice writing docstrings,
- practice calling your functions on test data,
- practice writing functions,
- practice your arithmetic operators,
- practice your manipulation of tuples, and
- learn some valuable tools related to scalar product.

note I am delaying the introduction of git through Udacity to allow you to focus on writing more Python code and on preparing your computer properly.

preparing for Lab03

1. set up your screen with Canopy terminal on left and Canopy editor on right
2. download the Canvas materials and put in your 19fa103/lab/lab03 directory

The TA will give a demo. Sometime during lab, a TA will check off these tasks with each of you.

preparing for HW2

1. download the HW2 materials and put them in your 19fa103/hw/hw2 directory
2. run the HW2 tester (in a terminal) on the original HW2 script: `python tester_hw2_19fa103.py`
3. solve the practice problem and run the HW2 tester again (you should pass some tests)

The TA will give a demo. Sometime during lab, a TA will check off these tasks with each of you.

preamble to scalar product exercises

Addition and multiplication are popular arithmetic operators in computer science, due to their efficiency. They are more efficient than other operators like division and square root.

Scalar product is a valuable operator that is built out of additions and multiplications. It transforms vectors into scalars.

For a reminder about what scalars and vectors are, I have added some definitions and examples below.

Here is an example of two vectors and the angle between them:

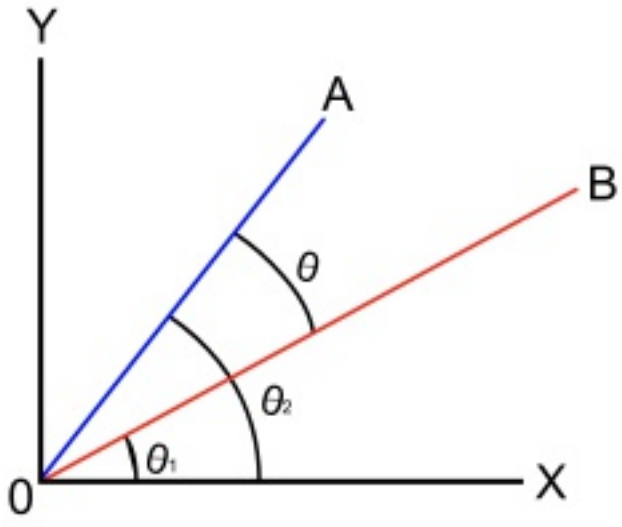

Figure 1: two vectors A and B, and the angle θ between them (thanks to Wikimedia Commons)

For example, a triangle ABC naturally defines three vectors: from the first vertex to the second vertex (AB) from the second vertex to the third vertex (BC) from the third vertex to the first vertex (CA).

scalar something with magnitude; examples of scalars are integers, rational numbers, and reals (in Python, ints and floats)

vector something with direction and magnitude; an example of a vector is a translation (a relative motion from here to there along a straight line, such as pushing a box), the normal vector at a point of a surface (the vector perpendicular to the surface at that point), and forces in physics (e.g., force of gravity: direction is toward the center of the earth, magnitude is the gravitational constant)

representation of a vector in 2-space (x,y) where x and y are scalars

reference wikipedia page on scalar product

Exercises (with your lab partner)

Note: there are more exercises here than you can complete in lab.
You are only expected to complete two or three of them today.
You can use the others for practice during the rest of the week.

In each exercise, you write a docstring, then design test calls, then write the function. Note that writing the docstring is a great way to start to better understand the function and its underlying content (e.g., normalization below), so *start by writing a docstring.* Designing test calls is a second way to better understand the computation, so *write the test calls second.* Then you are ready to attack the code of the function.

Let $v = (v_1, v_2)$ and $w = (w_1, w_2)$ be vectors in 2-space.
The **scalar product** of v and w, whose notation is $v \cdot w$, is defined as follows:
$v \cdot w := v_1 * w_1 + v_2 * w_2$
Scalar product is also called inner product and dot product.

- **scalar product** of 2 vectors in 2-space, using four float parameters
  (the function scalarProduct1)
  write a docstring; then write some test function calls; then implement the body of the function

  some test data:
  $(1., 1.) \cdot (2, 3) = 5$ (or, as a function call: scalarProduct1 (1.,1.,2.,3.))
  $(1., 1.) \cdot (2, 0) = 2$
  $(1., 1.) \cdot (0, 2) = 2$
  $(2., 3.) \cdot (4, 5) = 23$

- implement **scalar product** of 2 vectors in 2-space, using two tuple parameters
  (the function scalarProduct2)
  same drill: docstring/tests/function body

- use scalar product to calculate the **length** of a vector in 2-space:
  $|v| = \sqrt{v \cdot v}$
  (the function euclideanLength)
  make sure that you call your earlier scalar product function, rather than calculating this from scratch

- use scalar product to **normalize** a vector in 2-space:
  $\hat{v} = \frac{v}{|v|}$
  the function should return the unit vector $\hat{v}$ as a tuple
  (the function normalize)
  to normalize a vector means to turn it into a unit vector;
  a unit vector is a vector of length 1;
  the unit vector associated with a vector v will be notated $\hat{v}$;
  remember that tuples are immutable (see lecture 06, which also shows how to initialize a tuple)
  computation is often simplified when you use unit vectors;
  (fact: a popular graphics library called OpenGL really likes unit vectors; OpenGL graphics code, say for lighting a scene, will fail if you don't use unit vectors)

- compute the **angle** θ between two vectors using scalar product:
  (the function angle)
  $\cos\theta = \frac{v \cdot w}{|v||w|}$, so $\theta = \text{acos}(\frac{v \cdot w}{|v||w|})$
  this computation simplifies when unit vectors are used:
  $\cos\theta = \hat{v} \cdot \hat{w}$, so $\theta = \text{acos}(\hat{v} \cdot \hat{w})$
  so you have two options, normalize first (using the normalization function), or use the first formula (which would call your length function);
  in either case, you need to use the math.acos function in the math library
  python 3 documentation about math.acos
  recall arc cosine from your trigonometry course: it is the inverse of the cosine function;
  often in computations, you can work directly with $\cos\theta$ rather than θ, saving the expensive acos

- use scalar product to **project** a vector v onto another vector w;
  the function should return the projection of v on w
  (the function project)
  $\text{proj}(v, w) = c\hat{w}$, where $c = v \cdot \hat{w}$
  remember to first normalize w to $\hat{w}$
  Wikipedia page on projection

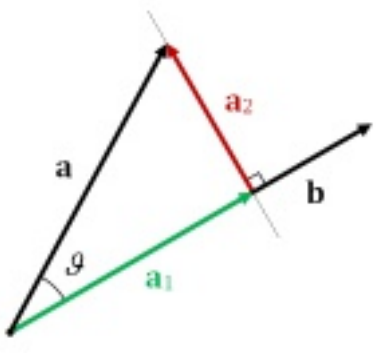

Figure 2: image of projecting a onto b, yielding a_1 (thanks to Wikimedia Commons)

Deliverables

B attendance (which includes
checkoff of the elements in `preparing for Lab03`,
checkoff of the elements in `preparing for HW2`)
A all the elements for B;
docstring/3-test-calls/code for the functions scalarProduct2 and normalize

Challenges

Why do computer scientists like polynomials so much? (It is related to today's lab.)

Implement these functions in 3-space.

Compute the **area** of a triangle ABC using the above functions and the classic triangle area formula $\text{area}(ABC) = \frac{bh}{2}$, where b is a base of the triangle and h the height from this base.

Foreshadowing test your implementations by comparing them to the associated functions in the numpy module (you should not use numpy in any of the above exercises)