Vectors

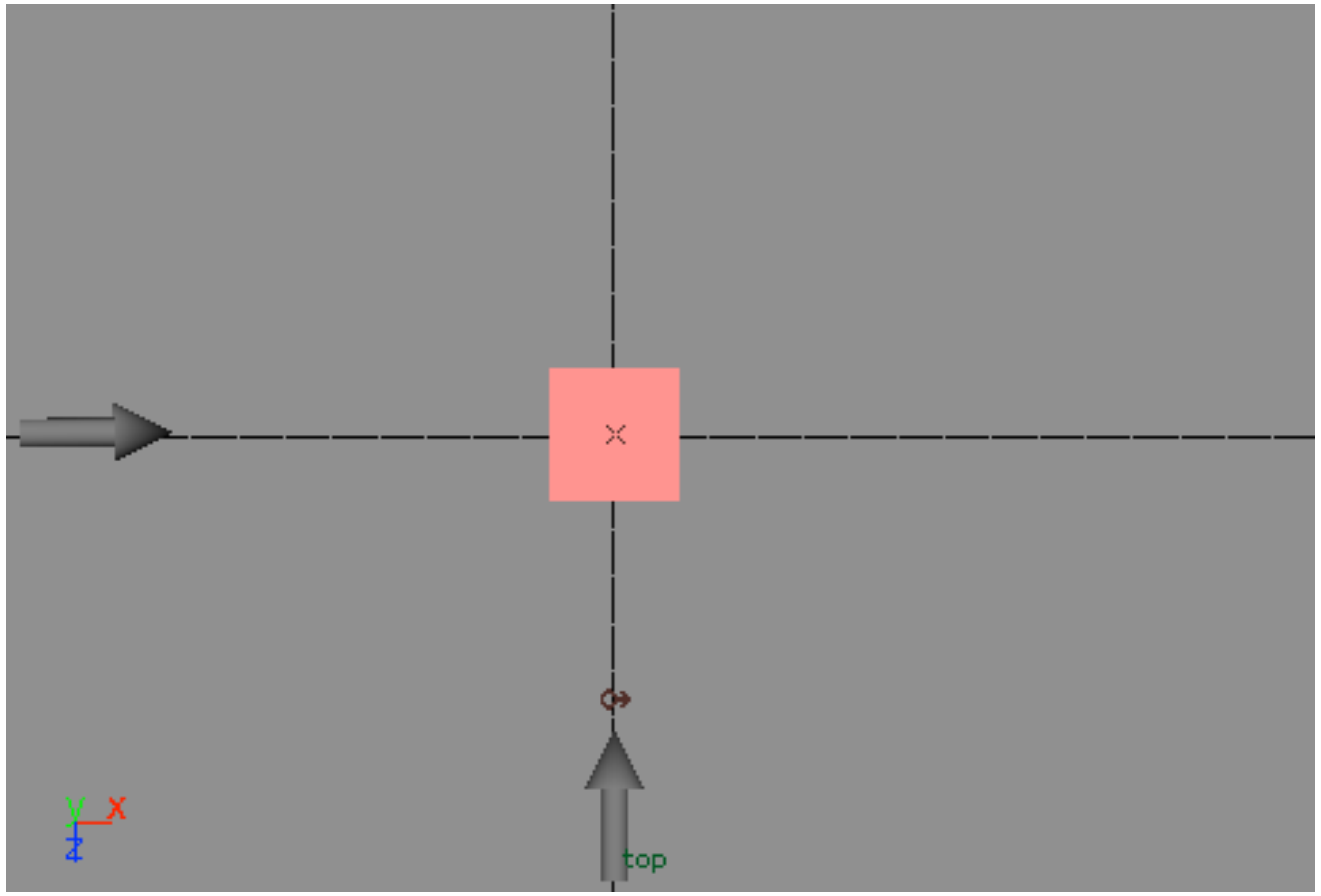
Scalars

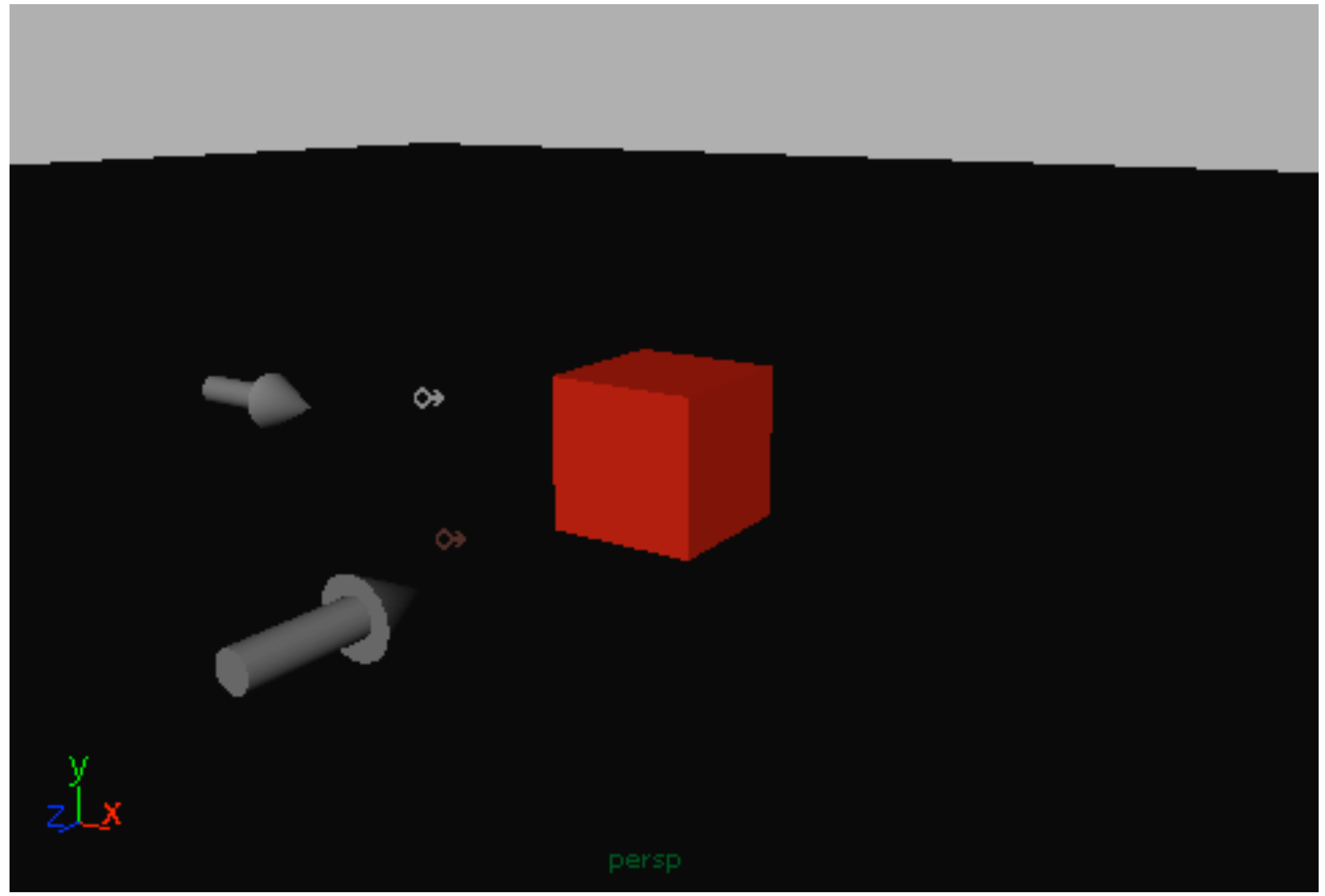
- We often employ a single number to represent quantities that we use in our daily lives such as weight, height etc.
- The magnitude of this number depends on our age and whether we use metric or imperial units.
- Such quantities are called **scalars**.
- In computer graphics scalar quantities include height, width, depth, brightness, number of frames, etc.

Vectors

- There are some things that require more than one number to represent them: wind, force, velocity and sound.
- These cannot be represented accurately by a single number.
- For example, wind has a magnitude and a direction.
- The force we use to lift an object also has a value and a direction.
- The velocity of a moving object is measured in terms of its speed (Km per hour) and a direction such as north west.
- Sound, too, has intensity and a direction.
- These quantities are called vectors.

Vector Example



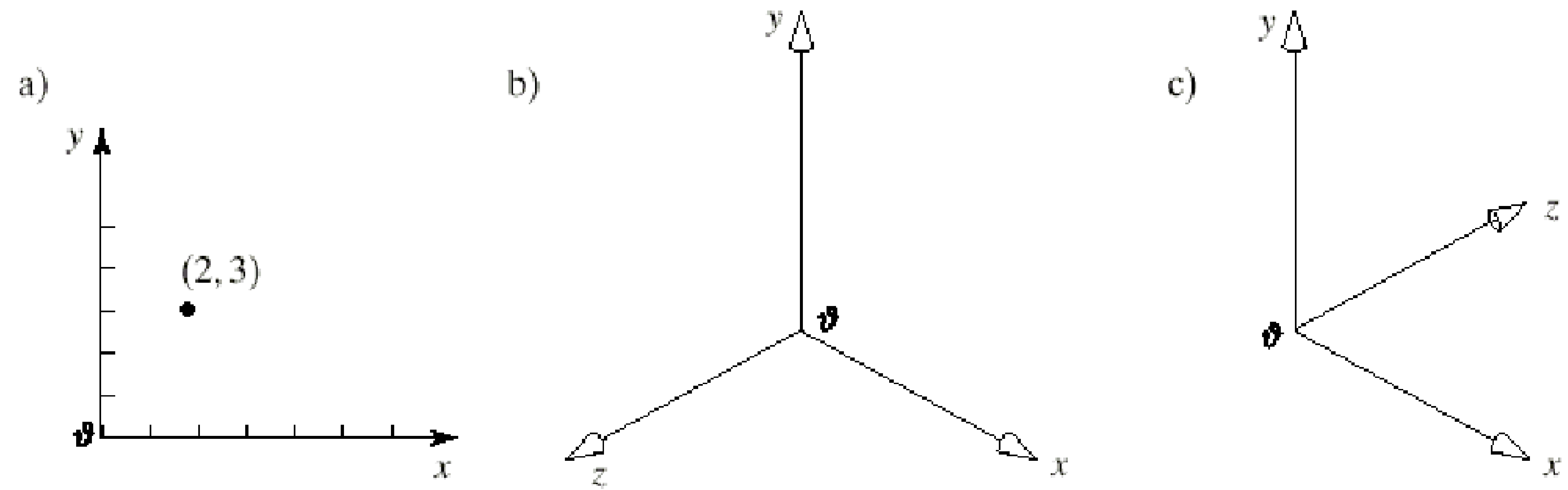


Gravity and Wind

Vectors in scripting

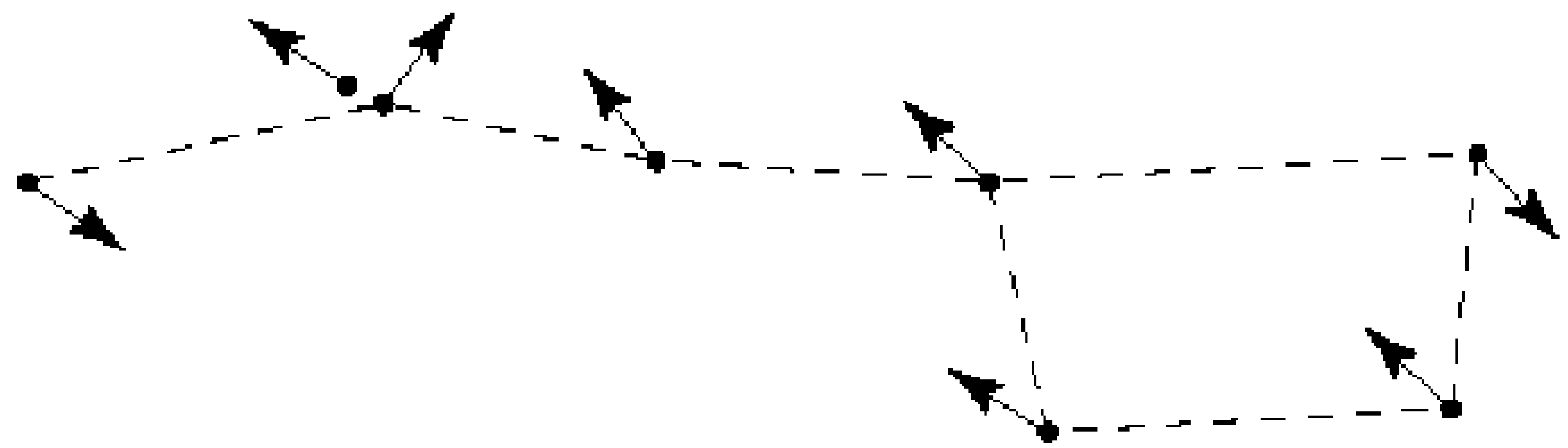
- Most programming languages have no direct support for Vectors and Matrices.
- Usually we either write our own system or use a 3rd party one.
- As Vectors and Matrices are fundamental to 3D graphics most graphics package API give use their own system for doing mathematics with them.
- Additionally to this the numpy system (http://www.scipy.org/) gives us the ability to create Vectors (array()) and matrices (matrix())

Introduction to Vectors



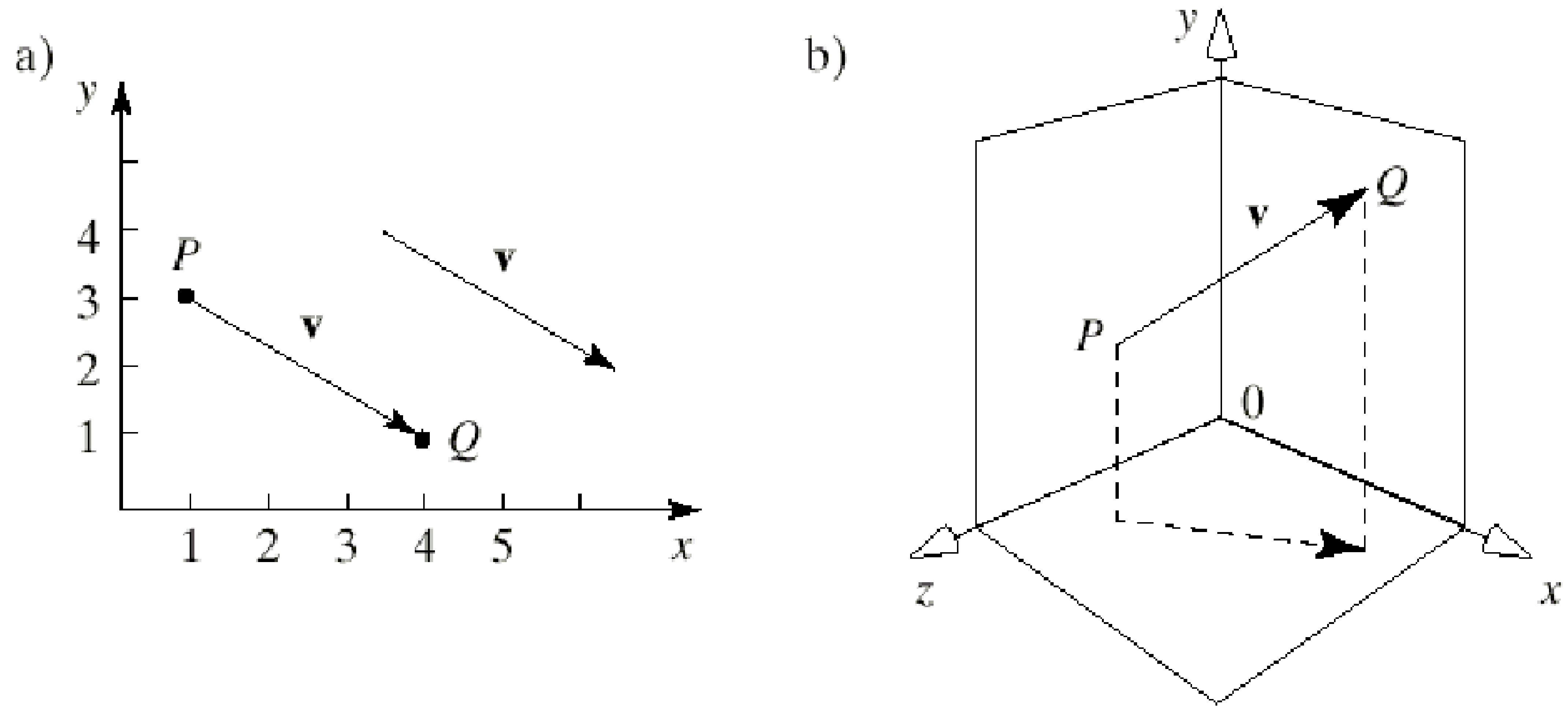
- All points and vectors we use are defined relative to some co-ordinate system
- Each system has an origin ϑ and some axis emanating from ϑ
- a) shows a 2D system whilst b) shows a right handed system and c) a left handed system
- In a right handed system, if you rotate your right hand around the Z axis by sweeping from the positive x-axis around to the positive y-axis your thumb points along the positive z axis.
- Right handed systems are used for setting up model views
- Left handed are used for setting up cameras

Simple Vectors



- Vector arithmetic provides a unified way to express geometric ideas algebraically
- In graphics we use 2,3 and 4 dimensional vectors however most operations are applicable to all kinds of vectors
- Viewed geometrically, vectors are objects having **length** and **direction**
- They represent various physical entities such as force, displacement, and velocity
- They are often drawn as arrows of a certain length pointing in a certain direction.
- A good analogy is to think of a vector as a displacement from one point to another

More Vectors



- Fig a) shows in a 2D co-ordinate system two points P=(1,3) and Q=(4,1)
- The displacement from P to Q is a vector v having components (3,-2), calculated by subtracting the co-ordinates of the points individually.
- Because a vector is a displacement, it has a size and a direction, but no inherent location.
- Fig b) shows the corresponding situation in 3 dimensions : v is the vector from point P to point Q.

Vectors

- The difference between two points is a vector **v**=Q-P
- Turning this around, we also say that a point Q is formed by displacing point P by vector **v**; we say the **v** "offsets" P to form Q
- Algebraically, Q is then the sum: Q=P+**v** also the sum of a point and a vector is a point P+**v**=Q
- Vectors can be represented as a list of components, i.e. an n-dimensional vector is given by an n-tuple $\mathbf{w} = [w_1 \ w_2 \ldots w_n]$
- For now we will be using 2D and 3D vectors such as $r = [3.4 - 7.78]$ and $t = [\ 33 \quad 142.7 \quad 89.1\ ]$ however later we will represent these as a column matrix as shown below

$$r = \begin{bmatrix} 3.4 \\ -7.78 \end{bmatrix} \text{ and } t = \begin{bmatrix} 33 \\ 142.7 \\ 89.1 \end{bmatrix}$$

Operations With Vectors

- Vectors permit two fundamental operations;
  - Addition
  - Multiplication with Scalars
- The following example assumes **a** and **b** are two vectors, and s is a scalar

$$\text{If a=}\left[\begin{array}{ccc} 2 & 5 & 6 \end{array}\right] \text{ and } \mathbf{b} = \left[\begin{array}{ccc} -2 & 7 & 1 \end{array}\right]$$

we can form two vectors :-

$$\mathbf{a} + \mathbf{b} = \left[\begin{array}{ccc} 0 & 12 & 7 \end{array}\right] \text{ and } 6\mathbf{a} = \left[\begin{array}{ccc} 12 & 30 & 36 \end{array}\right]$$

Vectors and Scalars

- A scalar is a single number e.g. 7
- A vector is a group of e.g. [4,5,3]
- Scalar and Vector addition

$$2 + 5 = 7 \text{ and } [2\ 3\ 5] + [2\ 7\ 2] = [4\ 10\ 7]$$

- Scalar product and scalar product (dot product) of two vectors

$$7 \times 8 = 56 \text{ and } [1\ 2\ 3] \cdot [2\ 4\ 6] = [1 \times 2 + 2 \times 4 + 3 \times 6] = 28$$

VectorScalar.py

```python
#!/usr/bin/python
from math import *
from numpy import *

def inputVector() :
  d=raw_input("enter vector x,y,z...n >")
  f=d.split(",")
  Vector=array(f,dtype=float32)
  return Vector



Vector=inputVector()
s=float(raw_input("enter a scalar >"))
print Vector ," * ", s, "= ",Vector * s
```

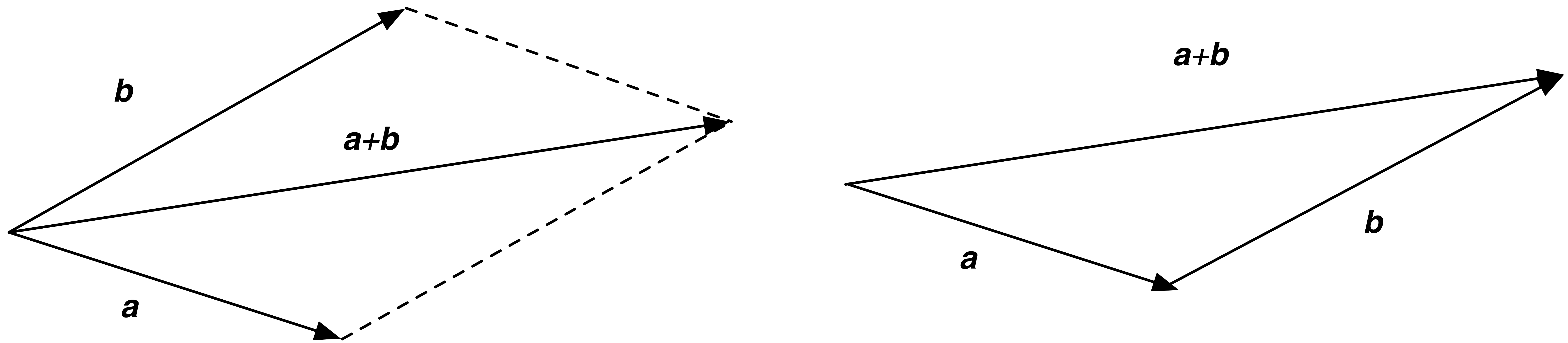
```
[jmacey@neuromancer:Lecture4]$./VectorScalar.py
enter vector x,y,z...n >2,4,5
enter a scalar >0.5
[ 2.  4.  5.] * 0.5 = [ 1.   2.   2.5]
[jmacey@neuromancer:Lecture4]$./VectorScalar.py
enter vector x,y,z...n >4,7,2,9,33
enter a scalar >2.6
[  4.   7.   2.   9.  33.] * 2.6 = [ 10.39999962  18.19999886   5.19999981  23.39999962  85.79999542]
```

VectorMultiplication.py

```python
#!/usr/bin/python
from math import *
from numpy import *

def inputVector() :
  d=raw_input("enter vector x,y,z >")
  f=d.split(",")
  Vector=array(f,dtype=float32)
  return Vector


Vector1=inputVector()
Vector2=inputVector()

print Vector1 ," . ", Vector2 , "= ",dot(Vector1,Vector2)
```

```
[jmacey@neuromancer:Lecture4]$./VectorMultiplication.py
enter vector x,y,z >2,3,4
enter vector x,y,z >1,2,3
[ 2.  3.  4.] . [ 1.  2.  3.] =  20.0
[jmacey@neuromancer:Lecture4]$./VectorMultiplication.py
enter vector x,y,z >2.4,0.2,10
enter vector x,y,z >2.5,0.9,1.5
[  2.4000001   0.2        10.        ] . [ 2.5         0.89999998  1.5        ] =  21.18
```

Vector Addition



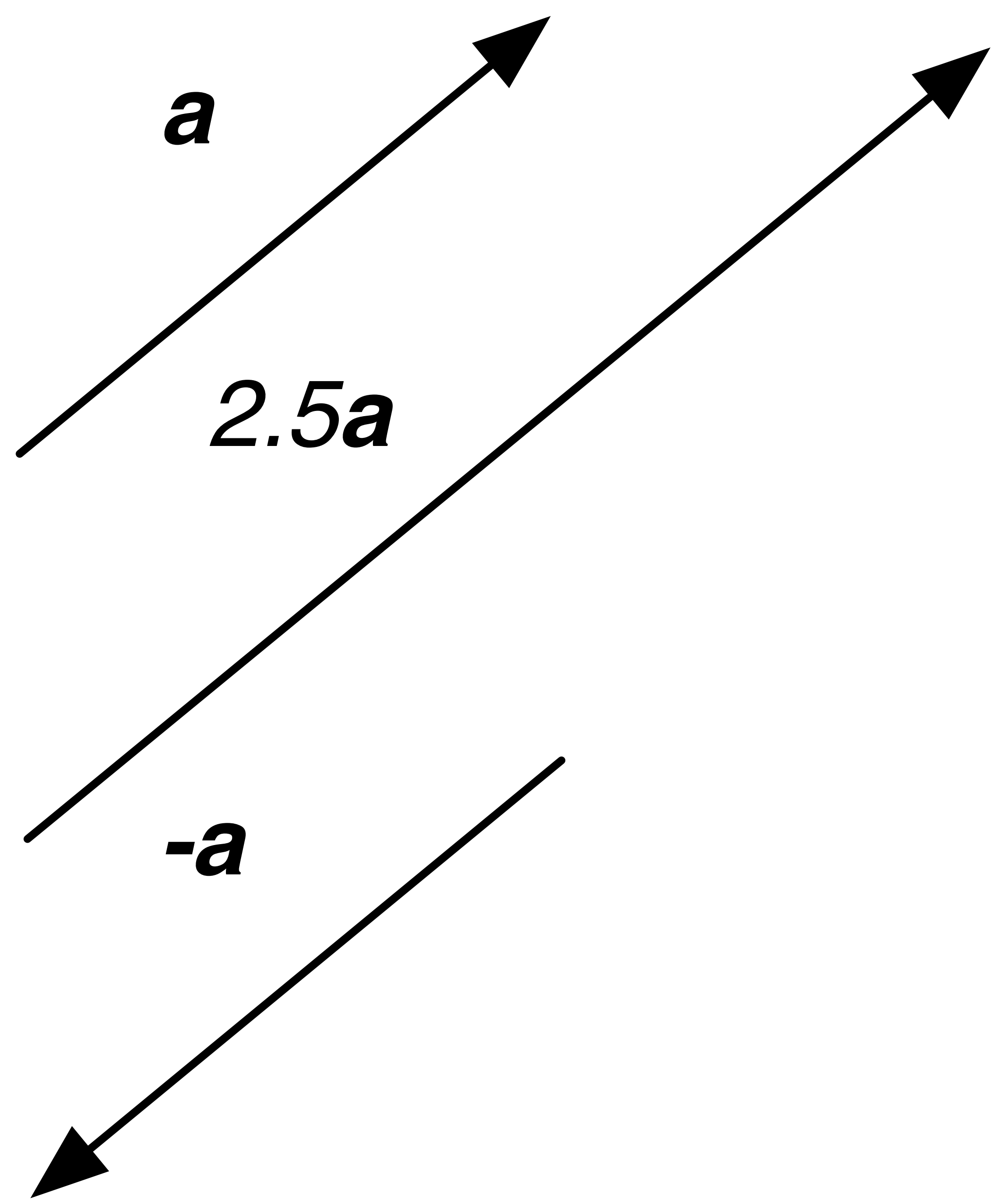
- A) shows both vectors starting at the same point, and forming two sides of a parallelogram.
- The sum of the vectors is then a diagonal of this parallelogram.
- B) shows the vector **b** starting at the head of **a** and draw the sum as emanating from the tail of **a** to the head of **b**

VectorAddition.py

```python
#!/usr/bin/python
from math import *
from numpy import *

def inputVector() :
  d=raw_input("enter␣vector␣x,y,z␣>")
  f=d.split(",")
  Vector=array(f,dtype=float32)
  return Vector

Vector1=inputVector()
Vector2=inputVector()

print Vector1 ,"␣+␣", Vector2 , "=␣",Vector1+Vector2
```

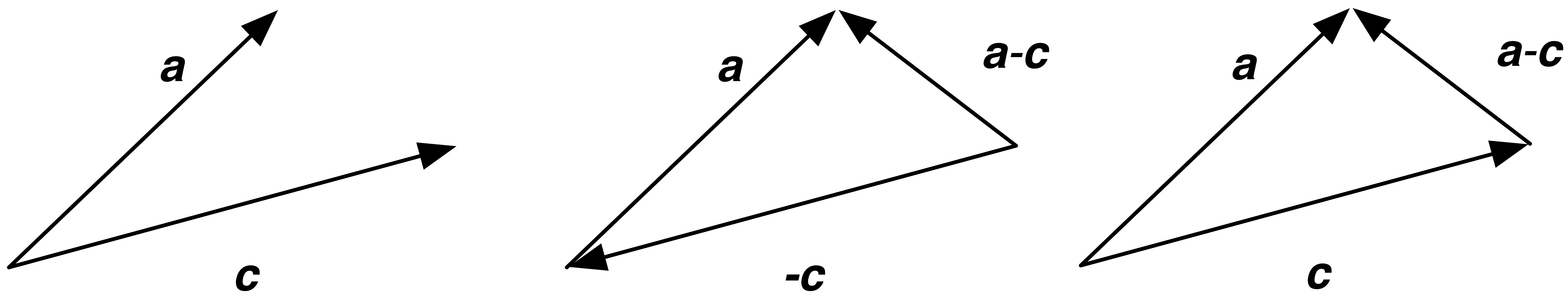
```
[jmacey@neuromancer:Lecture4]$./VectorAddition.py
enter vector x,y,z >2,3,4
enter vector x,y,z >3,2,1
[ 2.  3.  4.] + [ 3.  2.  1.] = [ 5.  5.  5.]
[jmacey@neuromancer:Lecture4]$./VectorAddition.py
enter vector x,y,z >2,3,4,5,6
enter vector x,y,z >6,5,4,2,3
[ 2.  3.  4.  5.  6.] + [ 6.  5.  4.  2.  3.] = [ 8.  8.  8.  7.  9.]
```

Vector Scaling



- The above figure shows the effect of multiplying (scaling) a vector **a** by a scalar s=2.5
- This now makes the vector **a** 2.5 times as long.
- When s is negative the direction of s**a** is opposite of **a**
- This is shown above with s=-1

Vector Subtraction



- Subtraction follows easily once adding and scaling have been established
- For example **a-c** is simply **a**+(-**c**)
- The above diagram shows this geometrically

For the vectors a= $\begin{bmatrix} 1 & -1 \end{bmatrix}$ and $\mathbf{c} = \begin{bmatrix} 2 & 1 \end{bmatrix}$
The value of a-c= $\begin{bmatrix} -1 & -2 \end{bmatrix}$
as
a+(-c)= $\begin{bmatrix} 1 & -1 \end{bmatrix} + \begin{bmatrix} -2 & -1 \end{bmatrix} = \begin{bmatrix} -1 & -2 \end{bmatrix}$

VectorSubtraction.py

```python
#!/usr/bin/python
from math import *
from numpy import *

def inputVector() :
  d=raw_input("enter␣vector␣x,y,z␣>")
  f=d.split(",")
  Vector=array(f,dtype=float32)
  return Vector

Vector1=inputVector()
Vector2=inputVector()

print Vector1 ,"␣-␣", Vector2 , "=␣",Vector1-Vector2
print Vector1 ,"␣+␣-", Vector2, "=␣",Vector1+(-Vector2)
```

```
[jmacey@neuromancer:Lecture4]$./VectorSubtraction.py
enter vector x,y,z >2,1
enter vector x,y,z >-1,1
[ 2.  1.]  -  [-1.  1.] =  [ 3.  0.]
[ 2.  1.]  + - [-1.  1.] =  [ 3.  0.]
[jmacey@neuromancer:Lecture4]$./VectorSubtraction.py
enter vector x,y,z >3,4,5,6
enter vector x,y,z >8,2,4,3
[ 3.  4.  5.  6.] - [ 8.  2.  4.  3.] =  [-5.  2.  1.  3.]
[ 3.  4.  5.  6.] + - [ 8.  2.  4.  3.] =  [-5.  2.  1.  3.]
```

Linear Combinations of Vectors

- To form a linear combination of two vectors **v** and **w** (having the same dimensions) we scale each of them by same scalars, say *a* and *b* and add the weighted versions to form the new vector *a***v**+*b***w**

DEFINITION : A linear combination of the m vectors $\mathrm{v}_1, \mathbf{v}_2, ..., \mathbf{v}_m$ is a vector of the form

$\mathrm{w}=\mathrm{a}_1\mathbf{v}_1 + a_2\mathbf{v}_2 + ... + a_m\mathbf{v}_m$
where $\mathrm{a}_1, a_2.....a_m$ are scalars

- For example the linear combination 2[3 4 -1]+6[-1 0 2] forms the vector [0 8 10]

- Two special types types of linear combinations, "affine" and "convex" combinations, are particularly important in graphics.

LinearCombination.py

```python
#!/usr/bin/python
from math import *
from numpy import *

def inputVector() :
  d=raw_input("enter vector x,y,z >")
  f=d.split(",")
  Vector=array(f,dtype=float32)
  return Vector

Vector1=inputVector()
a=float(raw_input("Enter Scalar a >"))
Vector2=inputVector()
b=float(raw_input("Enter Scalar b >"))

print a,"*",Vector1 ," + ",b,"*",Vector2 , "= ",a*Vector1+b*Vector2
```

```
[jmacey@neuromancer:Lecture4]$./LinearCombination.py
enter vector x,y,z >2,3,5
Enter Scalar a >2
enter vector x,y,z >3,2,5
Enter Scalar b >2.5
2.0 * [ 2.  3.  5.] + 2.5 * [ 3.  2.  5.] = [ 11.5  11.   22.5]
```

Affine Geometry

- In geometry, affine geometry is geometry not involving any notions of origin, length or angle, but with the notion of subtraction of points giving a vector.
- It occupies a place intermediate between Euclidean geometry and projective geometry.

Affine Combinations of Vectors

- A linear combination of vectors is an affine combination if the coefficients $a_1, a_2 ...... a_m$ add up to unity.
- Thus the linear combination in combination in the previous equation is affine if $a_1 + a_2 + ... + a_m = 1$
- For example 3**a**+2**b**-4**c** is an affine combination of **a**,**b**,**c** but 3**a**+**b**-4**c** is not
- The combination of two vectors **a** and **b** are often forced to sum to unity by writing one vector as some scalar *t* and the other as *(1-t)*, as in

$$(1\text{-}t)a+(t)b$$

Convex Combinations of Vectors

- A **convex combination** arises as a further restriction on an affine combination.
- Not only must the coefficients of the linear combinations sum to unity, but each coefficient must also be non-negative, thus the linear combination of the previous equation is convex if

$$\mathrm{a}_1 + a_2 + ... + a_m = 1$$
$$\text{and}$$
$$\mathrm{a}_i \geq 0, \ \text{for } i = 1, ...m$$

Convex Combinations of Vectors

- As a consequence, all $\mathbf{a}_i$ must lie between 0 and 1
- Accordingly 0.3**a**+0.7**b** is a convex combination but 1.8**a**-0.8**b** is not
- The set of coefficients $\mathrm{a}_1, a_2......a_m$ is sometimes said to form a **partition of unity**, suggesting that a unit amount of "material" is partitioned into pieces.

Magnitude of a Vector

- If a vector $\mathbf{w}$ is represented by the n-tuple $\begin{bmatrix} w_1 & w_2 \ldots & w_n \end{bmatrix}$ how may it's magnitude (equivalently, its length and size) be computed?

- We denote the magnitude by $|\mathbf{w}|$ and define it as the distance from the tail to the head of the vector.

- Using the Pythagorean theorem we obtain

$$|\mathbf{w}| = \sqrt{w_1^2 + w_2^2 + ... + w_n^2}$$

- For example the magnitude of $\mathbf{w} = \begin{bmatrix} 4 & -2 \end{bmatrix}$ is $\sqrt{4^2 + -2^2} = \sqrt{20}$

- A vector of zero length is denoted as 0

- Note that if $\mathbf{w}$ is the vector from point A to point B , then $|\mathbf{w}|$ will be the distance from A to B

Magnitude.py

```python
#!/usr/bin/python
from math import *
from numpy import *

def inputVector() :
  d=raw_input("enter␣vector␣x,y,z...n␣>")
  f=d.split(",")
  Vector=array(f,dtype=float32)
  return Vector

def Magnitude(Vector) :
  sum =0
  for v in Vector[:] :
    sum +=v*v
  return sqrt(sum)

Vector=inputVector()

print "Length␣of␣" ,Vector, "=␣",linalg.norm(Vector)
print "Length␣of␣" ,Vector, "=␣",Magnitude(Vector)
```

```
[jmacey@neuromancer:Lecture4]$./Magnitude.py
enter vector x,y,z...n >1,2,3
Length of  [ 1.  2.  3.] =  3.74166
Length of  [ 1.  2.  3.] =  3.74165738677
[jmacey@neuromancer:Lecture4]$./Magnitude.py
enter vector x,y,z...n >2,5,4
Length of  [ 2.  5.  4.] =  6.7082
Length of  [ 2.  5.  4.] =  6.7082039325
```

Unit Vectors

- It is often useful to scale a vector so that the result has unity length.
- This type of scaling is called **normalizing** a vector, and the result is know as a **unit vector**
- For example the normalized version of a vector $\mathbf{a}$ is denoted as $\hat{\mathbf{a}}$
- And is formed by scaling $\mathbf{a}$ with the value $1/|\mathbf{a}|$ or $\hat{\mathbf{a}} = \frac{\mathbf{a}}{|\mathbf{a}|}$
- We will use normalized vectors for many calculations in Graphics, such as rotations, normals to a surface and some lighting calculations.

Normalize.py

```python
#!/usr/bin/python
from math import *
from numpy import *

def inputVector() :
  d=raw_input("enter vector x,y,z...n >")
  f=d.split(",")
  Vector=array(f,dtype=float32)
  return Vector

def Magnitude(Vector) :
  sum =0
  for v in Vector[:] :
    sum +=v*v
  return sqrt(sum)

def Normalize(Vector) :
  return Vector / Magnitude(Vector)

Vector=inputVector()

print "Normalized version of " ,Vector, "= ",Vector/linalg.norm(Vector)
print "Normalized version of " ,Vector, "= ",Normalize(Vector)
```

```
[jmacey@neuromancer:Lecture4]$./Normalized.py
enter vector x,y,z...n >2,4,5
Normalized version of  [ 2.  4.  5.] =  [ 0.2981424   0.59628481  0.74535602]
Normalized version of  [ 2.  4.  5.] =  [ 0.2981424   0.59628481  0.74535602]
```

The Dot Product

- The dot product of two vectors is simple to define and compute
- For a two dimensional vector $\begin{bmatrix} a_1 & a_2 \end{bmatrix}$ and $\begin{bmatrix} b_1 & b_2 \end{bmatrix}$, it is simply the scalar whose value is $a_1b_1 + a_2b_2$
- Thus to calculate the dot product we multiply the corresponding components of the two vectors and add the the results
- For example for two vectors $\begin{bmatrix} 3 & 4 \end{bmatrix}$ and $\begin{bmatrix} 1 & 6 \end{bmatrix}$ the dot product $= 27$
- And $\begin{bmatrix} 2 & 3 \end{bmatrix}$ and $\begin{bmatrix} 9 & -6 \end{bmatrix}$ =0
- The generalized version of the dot product is shown below

The dot product d of two n dimensional vectors $\mathbf{v} = \begin{bmatrix} v_1 & v_2 \ldots & v_n \end{bmatrix}$ and $\mathbf{w} = \begin{bmatrix} w_1 & w_2 \ldots & w_n \end{bmatrix}$ is denoted as $\mathbf{v} \bullet \mathbf{w}$ and has the value

$$d = \mathbf{v} \bullet \mathbf{w} = \sum_{i=1}^{n} v_i w_i$$

Properties of the Dot Product

- The dot product exhibits four major properties as follows

  1. **Symmetry:** $\mathbf{a} \bullet \mathbf{b} = \mathbf{b} \bullet \mathbf{a}$
  2. **Linearity:** $(\mathbf{a} + \mathbf{c}) \bullet \mathbf{b} = \mathbf{a} \bullet \mathbf{b} + \mathbf{c} \bullet \mathbf{b}$
  3. **Homogeneity:** $(s\mathbf{a}) \bullet \mathbf{b} = s(\mathbf{a} \bullet \mathbf{b})$
  4. $|\mathbf{b}|^2 = \mathbf{b} \bullet \mathbf{b}$

- The dot product is **commutative** that means the order in which the vectors are combined does not matter.

- The final expression asserts that taking the dot product of a vector with itself yields the **square of the length** of the vector. This is usually expressed in the following form $|\mathbf{b}| = \sqrt{\mathbf{b} \bullet \mathbf{b}}$

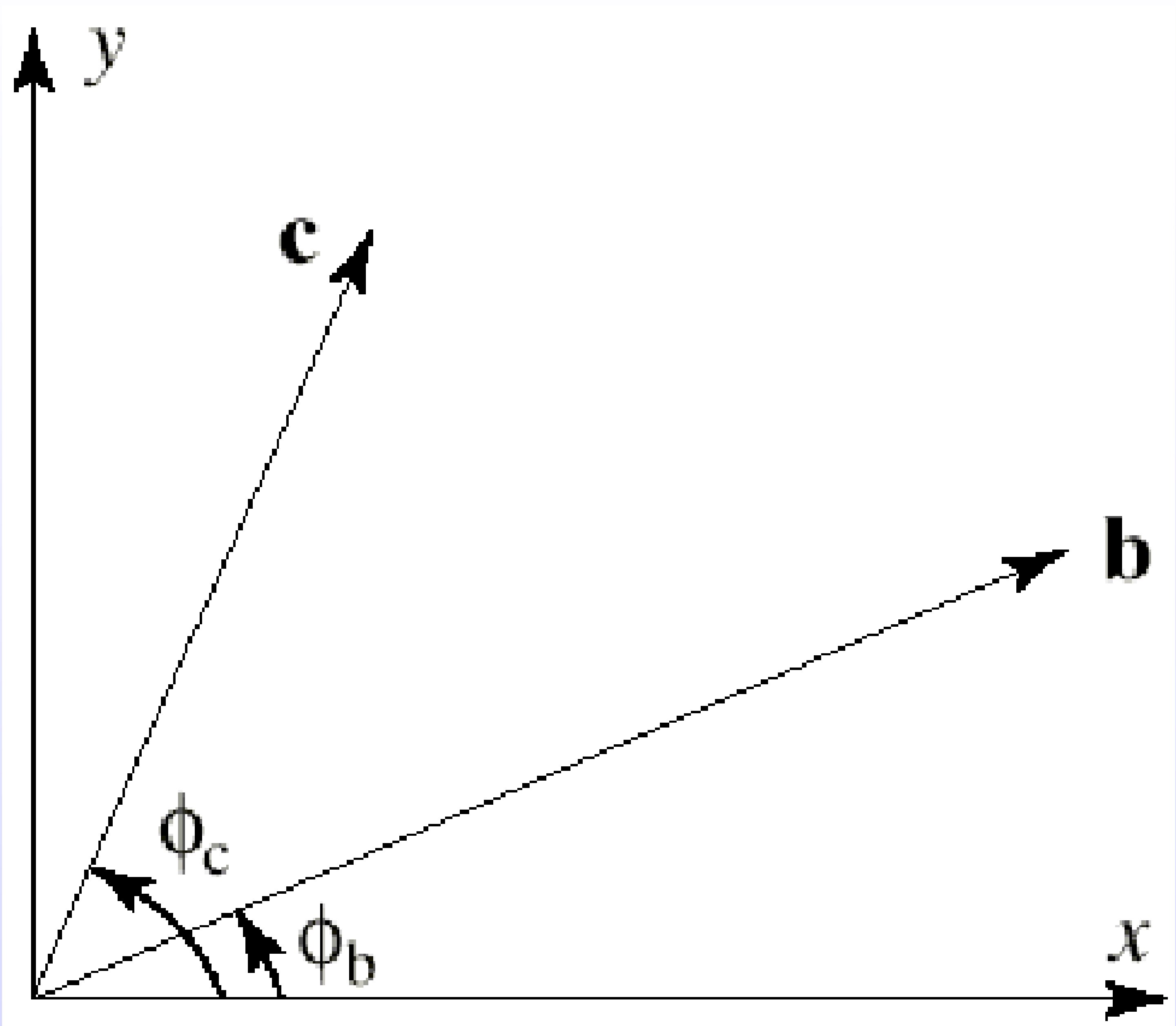
DotProduct.py

```python
#!/usr/bin/python
from math import *
from numpy import *

def inputVector() :
  d=raw_input("enter␣vector␣x,y,z...n␣>")
  f=d.split(",")
  Vector=array(f,dtype=float32)
  return Vector

def Magnitude(Vector) :
  sum =0
  for v in Vector[:] :
    sum +=v*v
  return sqrt(sum)


Vector1=inputVector()
Vector2=inputVector()
Vector3=inputVector()
s=float(raw_input("Enter␣a␣Scalar␣>"))

print "Symmetry"
print Vector1,".",Vector2,"=",dot(Vector1,Vector2)
print Vector2,".",Vector1,"=",dot(Vector2,Vector1)
print "Linearity␣"
print "(",Vector1,"+",Vector3,").",Vector2,"=",dot((Vector1+Vector3),Vector2)
print "(",Vector1,".",Vector3,")+(",Vector2,".",Vector3,")=",dot(Vector1,Vector2)+dot(Vector3
    ,Vector2)
print "Homogeneity"
print "(",s,"*",Vector1,").",Vector2,"␣=␣",dot(s*Vector1,Vector2)
print s,"*(",Vector1,".",Vector2,")=␣",s*dot(Vector1,Vector2)
print "Magnitude␣Squared"
mag=Magnitude(Vector1)
print "Magnitude(",Vector1,")=",mag*mag,"=",Vector1,".",Vector2,"=",dot(Vector1,Vector1)
```

```
[jmacey@neuromancer:Lecture4]$./DotProduct.py
enter vector x,y,z...n >2,3,4
enter vector x,y,z...n >3,2,1
enter vector x,y,z...n >6,5,2
Enter a Scalar >2
Symmetry
[ 2.  3.  4.] . [ 3.  2.  1.] = 16.0
[ 3.  2.  1.] . [ 2.  3.  4.] = 16.0
Linearity
( [ 2.  3.  4.] + [ 6.  5.  2.] ). [ 3.  2.  1.] = 46.0
( [ 2.  3.  4.] . [ 6.  5.  2.] )+( [ 3.  2.  1.] . [ 6.  5.  2.] )= 46.0
Homogeneity
( 2.0 * [ 2.  3.  4.] ). [ 3.  2.  1.]  =  32.0
2.0 *( [ 2.  3.  4.] . [ 3.  2.  1.] )=  32.0
Magnitude Squared
Magnitude( [ 2.  3.  4.] )= 29.0 = [ 2.  3.  4.] . [ 3.  2.  1.] = 29.0
```

The angle between two vectors

- The most important application of the dot product is in finding the angle between two vectors or between intersecting lines.
- The figure shows the vectors **b** and **c** which lie at angles ϕ_b and ϕ_c relative to the x axis.



From basic trigonometry we get $\mathbf{b} = (|\mathbf{b}|\, cos\phi_b, |\mathbf{b}|\, sin\phi_b)$ and $\mathbf{c} = (|\mathbf{c}|\, cos\phi_c, |\mathbf{c}|\, sin\phi_c)$

Thus the dot product of **b** and **c** is
$\mathbf{b} \bullet \mathbf{c} = |\mathbf{b}|\,|\mathbf{c}|\, cos\phi_c cos\phi_b + |\mathbf{b}|\,|\mathbf{c}|\, sin\phi_b sin\phi_c = |\mathbf{b}|\,|\mathbf{c}|\, cos(\phi_c - \phi_b)$,

so for any two vectors **b** and **c**,

$$\mathrm{b}\bullet\mathbf{c} = |\mathbf{b}|\,|\mathbf{c}|\, cos(\theta),$$

where θ is the angle from **b** to **c.**
Hence, $\mathbf{b} \bullet \mathbf{c}$ varies as the cosine of the angle from **b** to **c**.

Normalised Vector angles

- To obtain a more compact form the normalized vectors are usually used
- Therefore both sides are divided by $|\mathbf{b}|\,|\mathbf{c}|$and the unit vector notation is used so $\hat{\mathbf{b}} = \frac{\mathbf{b}}{|\mathbf{b}|}$ to obtain $cos(\theta) = \hat{\mathbf{b}} \bullet \hat{\mathbf{c}}$
- So the cosine of the angle between two vectors **b** and **c** is the dot product of the normalized vectors.

Example

find the angle between two vectors $\mathbf{b} = \begin{bmatrix} 3 & 4 \end{bmatrix}$ and $\mathbf{c} = \begin{bmatrix} 5 & 2 \end{bmatrix}$

$|\mathbf{b}| = \sqrt{3^2 + 4^2} = 5$ and $|\mathbf{c}| = \sqrt{5^2 + 2^2} = 5.3851$

so that $\hat{\mathbf{b}} = \begin{bmatrix} \frac{3}{5} & \frac{4}{5} \end{bmatrix}$ and $\hat{\mathbf{c}} = \begin{bmatrix} \frac{5}{5.3851} & \frac{2}{5.3851} \end{bmatrix}$

This gives us $\hat{\mathbf{b}} = \begin{bmatrix} 0.6 & 0.8 \end{bmatrix}$ and $\hat{\mathbf{c}} = \begin{bmatrix} 0.9285 & 0.3714 \end{bmatrix}$

The dot product $\hat{\mathbf{b}} \bullet \hat{\mathbf{c}} = 0.5571 + 0.296 = 0.8542 = cos(\theta)$

hence θ =31.326^o from the inverse cosine

this can then be expanded to work for 3 or 4 dimensions

AngleBetween.py

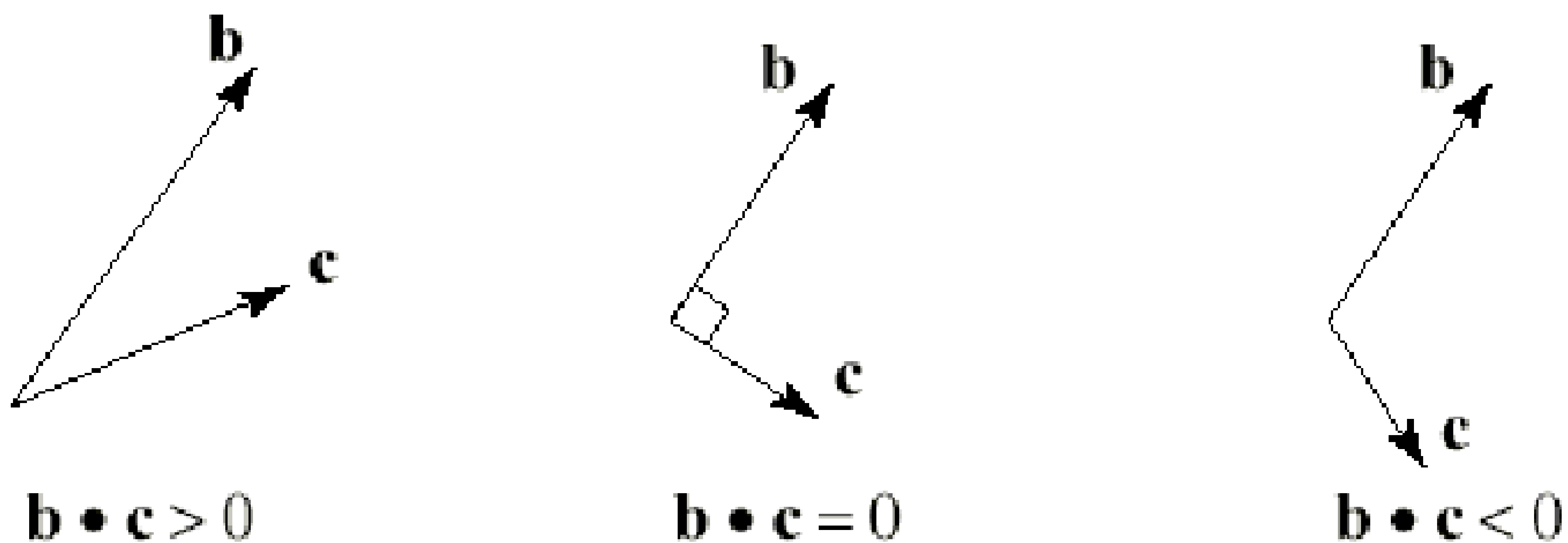
```python
#!/usr/bin/python
from math import *
from numpy import *

def inputVector() :
  d=raw_input("enter␣vector␣x,y,z...n␣>")
  f=d.split(",")
  Vector=array(f,dtype=float32)
  return Vector

def Magnitude(Vector) :
  sum =0
  for v in Vector[:] :
    sum +=v*v
  return sqrt(sum)

def Normalize(Vector) :
  return Vector / Magnitude(Vector)

Vector1=inputVector()
Vector2=inputVector()

Vector1=Normalize(Vector1)
Vector2=Normalize(Vector2)
ndot=dot(Vector1,Vector2)
print "Normalized␣vectors␣",Vector1,Vector2
angle = acos(ndot)
print "Angle␣between␣=␣",degrees(angle)
```

```
[jmacey@neuromancer:Lecture4]$./AngleBetween.py
enter vector x,y,z...n >3,4
enter vector x,y,z...n >5,2
Normalized vectors  [ 0.60000002  0.80000001] [ 0.92847669  0.37139067]
Angle between =  31.3286907294
[jmacey@neuromancer:Lecture4]$./AngleBetween.py
enter vector x,y,z...n >0,1,0
enter vector x,y,z...n >0,0,1
Normalized vectors  [ 0.  1.  0.] [ 0.  0.  1.]
Angle between =  90.0
[jmacey@neuromancer:Lecture4]$./AngleBetween.py
enter vector x,y,z...n >0,1,0
enter vector x,y,z...n >0.5,0.5,0
Normalized vectors  [ 0.  1.  0.] [ 0.70710677  0.70710677  0.        ]
Angle between =  45.0000009806
```

The sign of b•**c** and perpendicularity

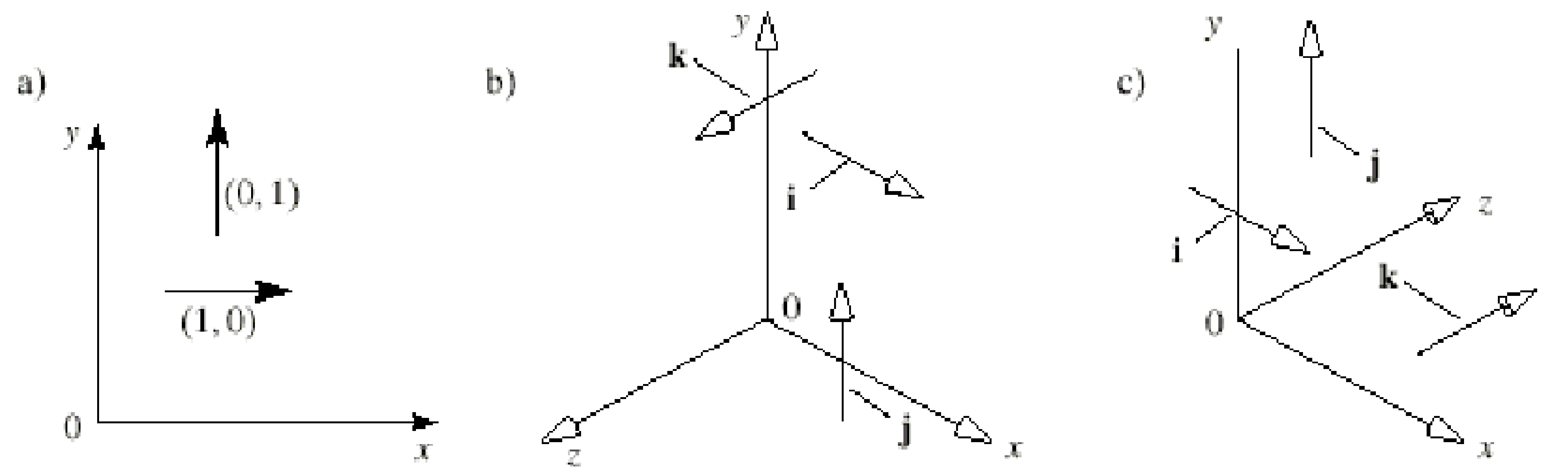
- $cos(\theta)$ is **positive** if $|\theta|$is less than 90^o, **zero** if $|\theta|$equals 90^o, and **negative** if $|\theta|$exceeds 90^o.

- Because the dot product of two vectors is proportional to the cosine of the angle between them, we can observe immediately that two vectors (of any non zero length) are



less than	90^o apart	if $\mathbf{b} \bullet \mathbf{c} > 0$;
exactly	90^o apart	if $\mathbf{b} \bullet \mathbf{c} = 0$;
more than	90^o apart	if $\mathbf{b} \bullet \mathbf{c} < 0$;

The standard unit Vector



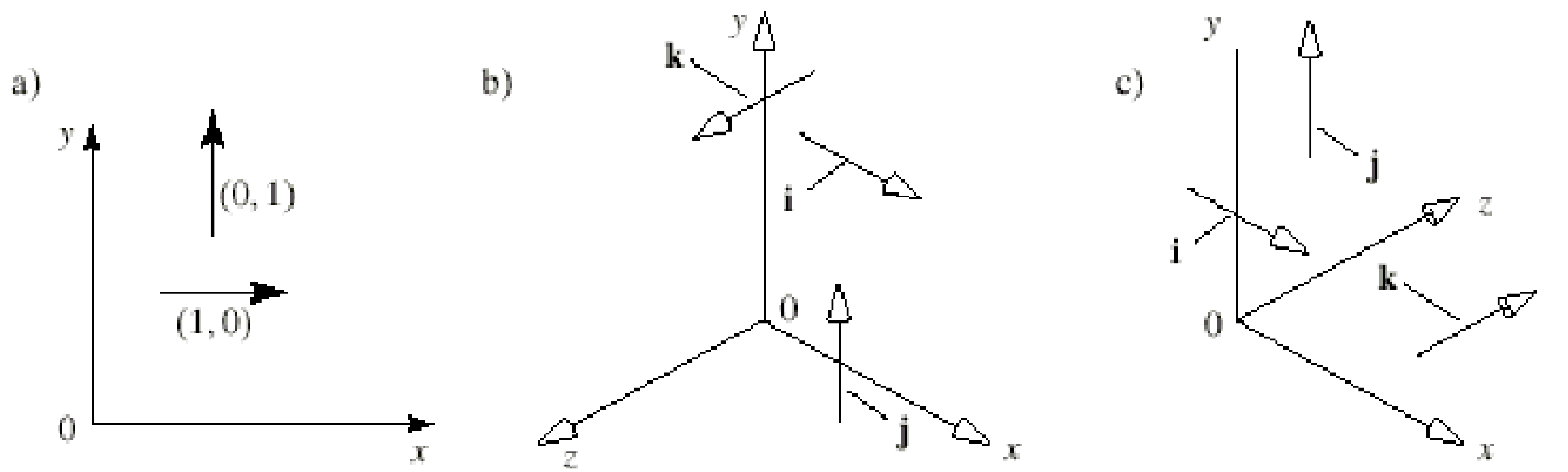
- The case in which the vectors are 90^o apart, or **perpendicular** is of special importance

Definition : Vectors **b** and **c** are perpendicular if $\mathbf{b} \bullet \mathbf{c} = 0$

- Other names for "perpendicular" are **orthogonal** and **normal** and they are used interchangeably.

- The most familiar examples of orthogonal vectors are those aimed along the axes of 2D and 3D coordinate systems as shown

The standard unit Vector



- The vectors **i**, **j** and **k** are called standard unit vectors and are defined as follows

$$\mathbf{i} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \mathbf{j} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \text{ and } \mathbf{k} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

Cartesian Vectors

- Let us define three Cartesian unit vectors i, j, k that are aligned with the x, y, z axes:

$$i = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \; j = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \; k = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

- Any vector aligned with the x-, y- or z-axes can be defined by a scalar multiple of the unit vectors i, j, k.

- A vector 10 units long aligned with the x-axis is 10i.

- A vector 20 units long aligned with the z-axis is 20k.

- By employing the rules of vector addition and subtraction, we can define a vector **r** by adding three Cartesian vectors as follows:

$$\mathbf{r} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$$

Cartesian Vectors

$$\mathbf{r} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$$

- This is equivalent to writing

$$\mathbf{r} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Therefore the magnitude of r is $|\mathbf{r}| = \sqrt{a^2 + b^2 + c^2}$

- A pair of Cartesian vectors such $\mathbf{r}$ and $\mathbf{s}$ can be combined as follows

$$\mathbf{r} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$$
$$\mathbf{s} = d\mathbf{i} + e\mathbf{j} + f\mathbf{k}$$
$$\mathbf{r} \pm s = [a \pm d]\mathbf{i} + [b \pm e]\mathbf{j} + [c \pm f]\mathbf{k}$$

Cartesian Vectors Example

$$\mathbf{r} = 2\mathbf{i} + 3\mathbf{j} + 4\mathbf{k} \text{ and } \mathbf{s} = 5\mathbf{i} + 6\mathbf{j} + 7\mathbf{k}$$

$$\mathbf{r} + \mathbf{s} = 7\mathbf{i} + 9\mathbf{j} + 11\mathbf{k}$$

$$|\mathbf{r} + \mathbf{s}| = \sqrt{7^2 + 9^2 + 11^2} = \sqrt{251} = 15.84$$

The cross product of two vectors

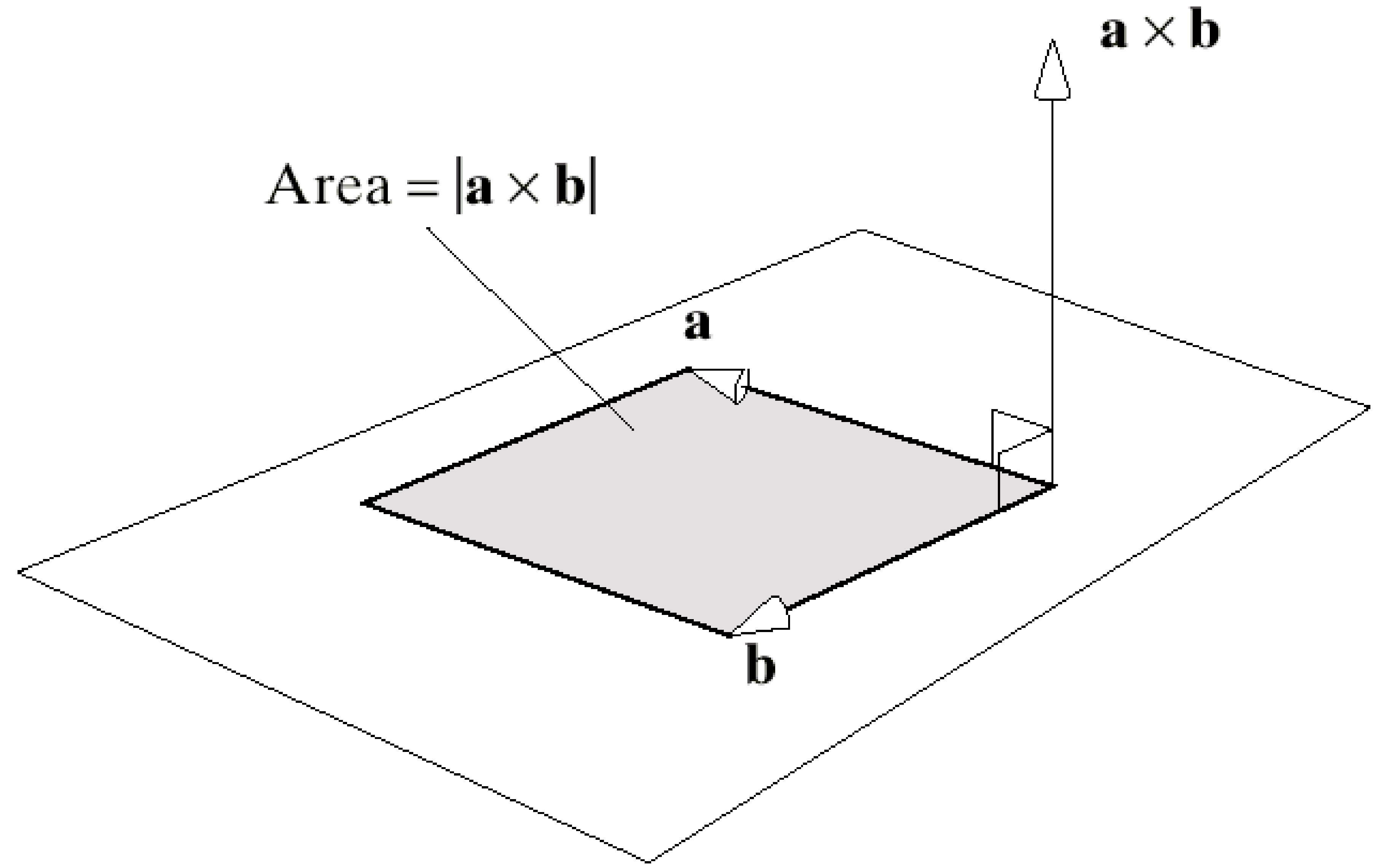
- The **cross product** (also called the **vector product**) of two vectors is another vector.
- It has many useful properties but the most useful is the fact that it is perpendicular to both of the given vectors
- Given the 3D vectors $\mathbf{a} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} b_x & b_y & b_z \end{bmatrix}$ their cross product is denoted as $\mathbf{a} \times \mathbf{b}$
- It is defined in terms of the standard unit vectors **i**, **j** and **k** as

$$\mathbf{a} \times \mathbf{b} = [a_y b_z - a_z b_y]\mathbf{i} + [a_z b_x - a_x b_z]\mathbf{j} + [a_x b_y - a_y b_x]\mathbf{k}$$

This form is usually replace using the determinant as follows

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

Geometric interpretation of the Cross Product



- By definition the cross product $\mathbf{a} \times \mathbf{b}$ of two vectors is another vector and has the following properties

- $\mathbf{a} \times \mathbf{b}$ is perpendicular (orthogonal) to both $\mathbf{a}$ and $\mathbf{b}$

- The length of $\mathbf{a} \times \mathbf{b}$ equals the area of the parallelogram determined by $\mathbf{a}$ and $\mathbf{b}$ this area is equal to $|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}|\,|\mathbf{b}|\,sin(\theta)$ where θ is the angle between $\mathbf{a}$ and $\mathbf{b}$ measured from $\mathbf{a}$ to $\mathbf{b}$ or $\mathbf{b}$ to $\mathbf{a}$, whichever produces an angle less than 180^o

CrossProduct.py

```python
#!/usr/bin/python
from math import *
from numpy import *

def inputVector() :
  d=raw_input("enter␣vector␣x,y,z...n␣>")
  f=d.split(",")
  Vector=array(f,dtype=float32)
  return Vector

def Magnitude(Vector) :
  sum =0
  for v in Vector[:] :
    sum +=v*v
  return sqrt(sum)

Vector1=inputVector()
Vector2=inputVector()

print Vector1,"x",Vector2,"␣=␣",cross(Vector1,Vector2)
print "Area␣of␣Vectors␣=␣",Magnitude(cross(Vector1,Vector2))
```

```
[jmacey@neuromancer:Lecture4]$./CrossProduct.py
enter vector x,y,z...n >0,0,1
enter vector x,y,z...n >1,0,0
[ 0.  0.  1.] x [ 1.  0.  0.]  =  [ 0.  1.  0.]
[jmacey@neuromancer:Lecture4]$./CrossProduct.py
enter vector x,y,z...n >0,1,0
enter vector x,y,z...n >1,0,0
[ 0.  1.  0.] x [ 1.  0.  0.]  =  [ 0.  0. -1.]
Area of Vectors =  1.0
[jmacey@neuromancer:Lecture4]$./CrossProduct.py
enter vector x,y,z...n >2,3,4
enter vector x,y,z...n >4,3,2
[ 2.  3.  4.] x [ 4.  3.  2.]  =  [ -6.  12.  -6.]
Area of Vectors =  14.6969384567
```

References

- Basic Algebra and Geometry. Ann Hirst and David Singerman. Prentice Hall 2001
- Computer Graphics With OpenGL, F.S. Hill jr, Prentice Hall
- "Essential Mathematics for Computer Graphics fast" John VinceSpringer-Verlag London
- "Geometry for Computer Graphics: Formulae, Examples and Proofs" John Vince Springer-Verlag London 2004
- "Engineering Mathematics", K. A. Stroud, Macmillan 3rd Edition 1987
- http://en.wikipedia.org/wiki/Affine_geometry