單元 1：一維等速運動

- 物理觀念：

$$位移 = 速度 \times 時間$$

- 程式：模擬物體的一維等速運動。
- 你可以下載程式碼後，執行安裝好的 VIDLE，打開下載的程式碼後，按 F5 後按 ok 執行。
  注意：在 Mac 中使用 Python 時，須將程式內的中文部分全部移除。
  執行時，可以按住滑鼠右鍵，移動滑鼠，來改變視角。若同時按住滑鼠的左右兩鍵，移動滑鼠，則可以改變視距。

```
########################################程式開始#####################################
from visual import *

#----------------------------------------------------------------------------
#    1. 參數設定
#----------------------------------------------------------------------------
v = 0.03            #木塊速度  = 0.03 m/s
dt = 0.001          #畫面更新的時間間隔，單位為 s
t = 0               #模擬所經過的時間  ，單位為 s，初始值為 0

#----------------------------------------------------------------------------
#    2. 畫面設定
#----------------------------------------------------------------------------
scene = display(title='1', width=800, height=800, x=0, y=0, center=(0,0.06,0), background=(0.5,0.6,0.5))
floor = box(pos=(0,-(0.005)/2,0), length=0.3, height=0.005, width=0.1)
cube = box(pos=(0, 0.05/2, 0), length=0.05, height=0.05, width=0.05)

#----------------------------------------------------------------------------
#    3. 物體運動部分
#----------------------------------------------------------------------------
while(cube.pos.x < 0.10):
      rate(1000)
      t += dt
      cube.pos.x += v*dt

print t
```

- 程式說明：

```
from visual import *
```

在 Python 裏，除主程式外，也可以使用其他稱為模組的程式來增加功能，使用時以關鍵字 from … import … 載入，visual 指的就是 vpython 模組，載入後就可以任意使用 vpython 的功能。

1. 變數設定

```
v = 0.03            #木塊速度  = 0.03 m/s
dt = 0.001          #畫面更新的時間間隔，單位為 s
t = 0               #模擬所經過的時間，單位為 s，初始值為 0
```

為模擬物理，須先設定所要模擬物理現象的相關物理常數和參數，在此每一常數和參數的數字表示的就是他們帶有 SI 單位的物理量。#之後的文字不是程式而是註解，是寫下來幫助我們或他人在讀此段程式時，了解此段程式碼作用的文字。

2. 畫面設定

```
scene = display(title='1', width=800, height=800, x=0, y=0, center=(0,0,0), background=(0.5,0.6,0.5))
```

此行程式會開一個專屬視窗，並以 scene 為名字，代表此視窗，而**視窗在開啟時，右方為模擬世界中的+*x* 軸、向上為+*y* 軸、射出紙面為+*z* 軸**。其餘性質說明如下：

display() 告訴 vpython 開一個新視窗，()裡面放可修改的參數。如：

title 為這個視窗的名稱。

width 與 height 是視窗的寬與高，單位是顯示器的像素。

x, y 是此視窗在顯示器的位置，單位是像素，x=y=0 表示視窗的最左上位置會在螢幕的左上角。

center 是視窗內模擬世界的中心，可以想像成此視窗內的世界是透過相機所看到的畫面，改變 center 就是改變相機視野的中心。

background 是視窗內的背景顏色，三個數字代表的是三原色(紅綠藍)的強度(由 0 至 1)。

```
floor = box(pos=(0,-0.005)/2,0), length=0.3, height=0.005, width=0.1, axis=(1,0,0))
cube = box(pos=(0, 0.05/2, 0), length=0.05, height=0.05, width=0.05)
```

在畫面加上地板，在此以 box 物件，畫一個非常扁平的 box，代表地板，取名為 floor。

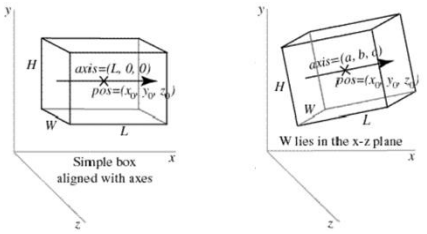
再在地板上畫一方塊 cube，因交接處在 y=0，所以 floor 的中心位置會在 pos=(0, -0.005/2, 0) 地方，而 cube 的中心位置會在 pos=(0, 0.05/2, 0)。

box() 產生一個長方體的物件，()裡面是可修改的參數。

pos 代表此物件中心所在位置的三維向量。

axis 設定長方體的方向，無指定時，預設為模擬世界中的+x 軸。

length, height, width 為長(平行 x 軸)、高(y 軸)、寬(z 軸)，如下圖所示。



接著宣告變數 cube 為一個 box，作為模擬中移動的木塊。

3. 物體運動部分

```
while (cube.pos.x < 0.10):
    rate(1000)
    t += dt
    cube.pos.x += v*dt

print t
```

while 是 Python 語言中的迴圈指令，只要它後面的條件式一直成立(這裡的條件式是 cube.pos.x < 0.10)，其**下方縮排的附屬程式碼就會持續重複執行，直到條件不成立為止（縮排指的就是 while 下面有好幾行程式碼的起頭位置都“排列整齊”，且比 while 還後面的排列方式）**，這種

持續執行一段程式的方式，稱為迴圈。在此，條件式 cube.pos.x < 0.10，表示的是當 cube 的位置的 x 值小於 0.10 時，while 的附屬程式就會持續重複執行，直到此條件不成立為止。在 Python 中，A.B 就是“A 的 B”的意思，物件在產生後，可以用 “A.B”的方式來指定或使用 A 的 B 參數。

```
rate(1000)
```

在 while 迴圈中，可以用 rate 來調整執行速度的快慢，在此 while 下的附屬程式碼，會以每秒 1000 次的頻率被執行。

```
t += dt
```

這行指令可以將變數 t，增加 dt 的數值，也可寫成 t = t + dt。所以在這段程式中，每執行一次 while 迴圈內的附屬程式，t 都會增加 dt。如此， t 就表示在模擬世界裏所經歷的時間。一開始設定 dt=0.001 (s)，配合在這裡 rate 的設定，迴圈每秒執行 1000 次，就是模擬世界中的每 1 秒，也是真實世界中的$1000 \times 0.001 = 1$秒。如果，要慢動作或快動作看執行的結果，可以調整 rate 內的數值。另外 dt 的數值要依所模擬物理事件的數量級來設定，設的太小則模擬的進展會很慢，設的太大則模擬會不準確， dt 通常設為要模擬物理事件時間數量級的10^{-3}倍，因此 dt 設為 0.001。

```
cube.pos.x += v*dt
```

此行也是這個程式所要模擬物理的描述方程式，即每隔一小段時間，就讓木塊位置的 x 分量，增加此段時間的位移(即速度乘以時間)。這一行也可寫成 cube.pos.x = cube.pos.x + v*dt

```
print t
```

如果我們想知道木塊到達 0.10 m 時，所花費的總時間，可以用 print 將 t 的數值顯示出來。**注意，print 不在 while 的縮排之中，而在 while 之後，意即等到 while 迴圈結束後，才會執行。**

- 練習

1. 改變 v 的數值，並用你的手錶或碼錶計時，看看到木塊到 0.10m 時，所花的時間是否等於 0.10 / v。
2. 設定 v 的數值，並改變程式碼中 while (cube.pos.x < 0.10): 中的數值 0.10 為你想要的數值，執行程式，看看結果，有什麼不同。
3. 更改 display()中，參數 x、y 與 center 的值，或 background 中的數值，看發生了什麼差異。
4. 改變 rate()括號中的值，並用你的手錶或碼錶計時，看看到模擬停止時，不同數值在實際時間所造成的差異。
5. 將這一行程式碼 cube = box(pos=(0, 0.05/2, 0), length=0.05, height=0.05, width=0.05)改為 cube = box(pos=(0, 0.05/2, 0), length=0.05, height=0.05, width=0.05, material= materials.wood) 後，執行程式，你會發現木塊上有了木質條紋。

單元 2：自由落體和觸地反彈

- 物理觀念：

速度 = 加速度 × 時間

- 程式：

```
#######################################程式開始#####################################
from visual import *

#   1. 參數設定
a = -9.8                    #加速度值，在 x、z 方向為 0，在 y 方向為 g=-9.8 m/s^2
vy = 0                      #球的 y 方向初速
size = 0.2                  #球的半徑
h = 10.0                    #球的初始高度
dt = 0.001                  #畫面更新的時間間隔，單位為 s
t = 0                       #模擬所經過的時間 ，單位為 s，初始值為 0

#   2. 畫面設定
scene = display(center = (0, h/2, 0), background=(0.5,0.6, 0))
floor = box(pos=(0,-0.005/2,0), length=15, height=0.005, width=5)
ball = sphere(pos =(0, h, 0), radius=size, color=color.blue)

#   3. 描述物體的運動

while (True):
    rate(1000)
    vy += a*dt
    ball.pos.y += vy*dt

    if ball.pos.y <= size :
        vy = abs(vy)
```

- 程式說明：之後的程式說明，只會針對此單元新出現的指令、程式邏輯或用法做說明。之前已學過的，除非必要就不再多做說明。

1. 參數設定：設定此次模擬所需要的物理參數和物理係數，採用 SI 單位。其中

```
a = -9.8                    #加速度值，在 x、z 方向為 0，在 y 方向為 g=-9.8 m/s^2
vy = 0                      #球的 y 方向初速
size = 0.2                  #球的半徑
h = 10.0                    #球的初始高度
```

設定在 y 方向的加速度值為 -9.8 m/s^2。因為是自由落體，所以 y 方向的初速設為 0。並將其餘的相關條件設定好，如球的半徑和球的初始高度。

2. 畫面設定

```
scene = display(center=(0,h/2,0), background=(0.5,0.6,0)
```

關於 display 和 box 的用法，請參考單元 1，和單元 1 不同的是，這裏我們有許多參數並沒有設

定；當我們不設定這些參數時，VPython 會使用其內建的數值，大多數的情形下，這些內建的參數數值，對我們的模擬來說，都很適當。在此使用預先宣告好的變數 h 來設定 display 的參數 center，不寫(0,0.25,0)，而寫作(0,h/2,0)。這樣的作法有一個非常大的好處，即根據同一參數的程式段落可能非常多且散布在各處，如果要更改此參數值時，就不需到處尋找，只要更改程式一開始的參數值即可。

```
ball = sphere(pos=(0,h,0), radius=size, color=color.blue)
```

在此以 sphere()產生一個球體，並在程式中以 ball 作為其變數名稱。
pos 為球心初始所在位置的座標，即 x=0，y=h，z=0。
radius 可設定此球體的半徑。
此物件一樣也可以改變顏色。

3. 物體運動部分

```
while (True):
    rate(1000)
    vy += a*dt
    ball.pos.y += vy*dt
```

while 的條件式是 True，表示此條件式永遠為真，所以 while **下方縮排的附屬程式，會一直執行，直到你把視窗關掉為止**。因為設定了加速度 a，當每一次執行 vy += a*dt，就會讓 vy(物體的速度)依此物理公式，增加在這一小段時間(dt)內由加速度所造成的增加量，而下一行 ball.pos.y += vy*dt，則會讓 ball 位置的 y 分量，增加在這一小段時間內由速度所造成的位移。
以下程式碼，則是判斷 ball 在接觸地面時，使其向上反彈。

```
    if ball.pos.y <= size :
        vy = abs(vy)
```

要判斷 ball 是否接觸到地面，就須使用 python 中的邏輯判斷指令 if，其基本用法如框內所示，**若 if 後面的條件式成立，則會執行冒號 “:” 之後縮排的附屬程式碼，若不成立，則不會執行且直接跳過此段附屬程式碼**。這裡的條件式是 ball.pos.y <= size，意思是當 ball 的球心位置的 y 分量小於或等於其半徑時，即 ball 已接觸地板。這裏設定的是完全彈性碰撞，所以在碰撞後，ball 的運動速度會由負變正，因此使用函數 abs()，即取絕對值的意思，使 ball 的速度變成正值，也就是向上。注意，此行式子 vy = abs(vy)屬於 while 迴圈內的 if 的附屬程式，所以需要縮排兩次。

- 更多說明：

1. 關係運算子與布林(Boolean)代數

在 if 的條件式中出現<=，在單元 1 中，while 的條件式中出現<，這都是所謂的關係運算子，其他還有

x == y	# x 等於 y
x != y	# x 不等於 y
x > y	# x 大於 y
x < y	# x 小於 y
x >= y	# x 大於或等於 y
x <= y	# x 小於或等於 y

它會判斷 x 與 y 的關係是否如所指定關係運算子的描述，是的話其結果就是 True，不是的話會結果為 False。True 跟 False 又稱為布林(Boolean)代數。

- 練習：

1. 顯示每一次接觸地面時的時間 t
2. 如果，在地面發生的碰撞，不是彈性碰撞而有動能耗損，假設每次碰撞速度的絕對值是碰撞前的 0.9 倍，程式要如何修改，才能模擬這樣的情形？
3. 如果這個球有一個橫向初速(例如 vx = 0.5m/s)，那麼程式要怎麼改寫，才可以描述這樣的情形？
4. 接續 3，如果我們設定球心的初始位置在 x=-7.5，y=size，z=0，並且讓球的初速為 vy=5，vx=5，我們會觀察到什麼現象？如果我們要讓球在離開 floor 的範圍前停下來，while 的條件式要做什麼修改？

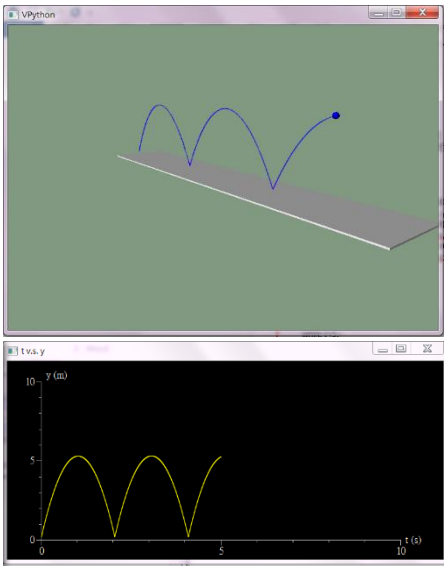
單元 3：三維運動

- 物理觀念：

將位移、速度、加速度等物理量由一維推廣到三維

- 範例程式：

```
from visual import *
from visual.graph import *

#   1. 變數設定
size = 0.2                  # 球的半徑  0.2 m
g = vector(0,-9.8,0)        # 重力加速度
t = 0
dt = 0.001

#   2. 畫面設定
scene = display(width=800, height=600, center=(15,0,0), background=(0.5,0.6,0.5))
gd1 = gdisplay(x=800, y=0, title='t v.s. y', xtitle='t (s)', ytitle='y (m)', ymax=10, xmax=10)
ty = gcurve(gdisplay=gd1, color=color.yellow)
floor = box(pos=(15,-0.05,0), length=30, height=0.1, width=5)
ball = sphere(radius=size, color=color.blue, make_trail= True)

#   3. 初始條件
ball.pos = vector(0, size, 0)
ball.v = vector(5, 10, 0)

#   4. 物體運動部分
while t<5:
    rate(1000)
    ty.plot( pos = (t, ball.pos.y) )
    t += dt
    if ball.pos.y <= size:
        ball.v.y = abs(ball.v.y)

    ball.pos += ball.v*dt
    ball.v += g*dt
```



- 程式說明：

此程式除了模擬拋體運動和留下軌跡外，並繪製此拋體的高度對時間的作圖(t-y 圖)。在執行模擬時，按住滑鼠右鍵移動滑鼠可以改變視角，按住滑鼠兩鍵移動滑鼠可以改變視野大小。

```
from visual.graph import *
```

visual.graph 模組的功能為繪圖，其相關繪圖功能，稍後會敘述。

1. 參數設定

```
g = vector(0,-9.8,0)                # 重力加速度
```

之前在設定加速度、速度時，都是設定為變數，而 vpython 另有一個方便實用的向量物件。其用法和特性就如同一般在科學與工程上使用的向量，例如 v = vector(10,20,30)，則其三個方量即為 v.x = 10、v.y = 20、v.z = 30。關於向量更多的應用，在之後會做更多的介紹。

2. 畫面設定

```
gd1 = gdisplay(x=800, y=0, title='x v.s. y', xtitle='x (m)', ytitle='y (m)',ymax=5, xmax=30)
ty = gcurve(gdisplay=gd1, color=color.cyan)
```

這是跟作圖有關的程式碼，其中兩個較常用的物件介紹如下：
gd1=gdisplay(…) 產生一個繪圖的視窗物件，設定該視窗的參數，並命名為 gd1。
title 在視窗標題欄顯示的名稱。
xtitle, ytitle，x 和 y 軸顯示的名稱。
width, height，視窗的寬度與高度，單位是像素，預設值分別是 800 與 400。
x, y 調整視窗在螢幕上的位置，單位是像素。
xmax, xmin, ymax, ymin 指定圖中 x 和 y 軸的最大與最小值，如果未指定，則會隨資料變動。

ty=gcurve() 產生一個會依據輸入的資料在 gdisplay 中畫曲線圖的物件。
gdisplay=gd1，指定該曲線圖出現在名為 gd1 的 gdisplay 視窗中。

```
ball = sphere(radius=size, color=color.blue, make_trail= True)
```

Vpython 中所有的繪圖物件都可留下軌跡，只要設定物件的參數時加上 make_trail=True 即可。

3. 初始條件

```
ball.pos = vector(0, size, 0)
ball.v = vector(5,10,0)
```

ball.pos = vector(0, size, 0) 設定球的初始位置在 x=0, z=0, 而 y=size 表示球心是在 floor 上方以球的半徑為高度的地方，即球的下緣是接觸 floor 的上表面。
ball.v = vector(5,10,0) 設定球的初速有三個分量，分別是朝 x 的速度分量是 5 m/s，朝 y 的速度分量是 10 m/s，朝 z 的分量是 0。

4. 物體運動部分

在這裏 while 的條件式是 t<5，所以作的是時間 t < 5 s 的模擬。

```
    ty.plot( pos=(t, ball.pos.y) )
```

這行程式碼就是前面在介紹 gcurve 時提到輸入資料並繪圖的部分了，將 (t, ball.pos.y)即時間 t

和球位置的 y 分量設定給取名為 ty 的 gcurve 中的 pos，即可在圖中加入一個新的球位置的 y 分量對於時間 t 的資料點，因為在 while 迴圈中每次執行時都會產生新的資料點，所以每次執行時也同時會在 gcurve 加上一筆新的資料，並同時繪出。

- 練習

1. 可以將 ball 在 floor 上的反彈由彈性碰撞，改為非彈性碰撞(如同你在單元 2 中練習所做的)看會發生什麼事情？這個模擬是不是越來越像真實世界中球的運動。
2. 改變球的初位置和初速度，觀察一下這兩個變數對於拋體運動的影響各是什麼？
3. 請將 ball 速度的 y 分量對於時間 t 作圖。
4. 請將 ball 速度的 x 分量對於時間 t 作圖。
5. 請將 ball 位置的 x 分量對於時間 t 作圖。

- 進階練習

1. 先在 pos=(30, 3, 0)的地方，畫一道牆，其長高寬各為 length =0.001、height = 6、width = 5。之後將程式修改為，當球接觸到此道牆時會反彈回來。提示：有兩個地方要修改，一個是 while 裏的條件式，另一個是在 while 的附屬程式裏，要判斷 ball 是否有接觸到牆，有的話要做適當的處理。

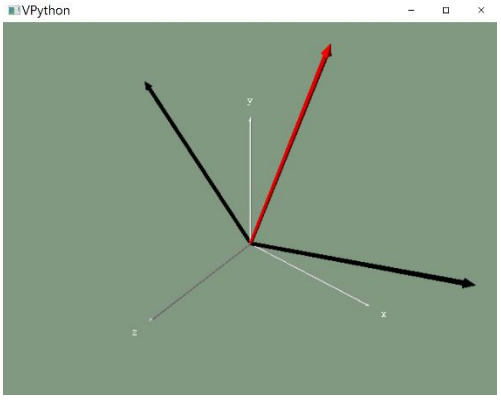
單元 4：力(向量)的合成

- 物理觀念：

力(向量)的合成

- 範例程式：執行程式後，觀察力(向量)的合成，可以按住滑鼠的右鍵並移動滑鼠，以改變視角，或按住滑鼠的左右兩鍵並移動滑鼠，以改變視距。

```
from visual import *

#   1. 設定初始值、list
vectors = [vector(1.5,0.5,0), vector(-1,1,0)]
arrows = []
final_vector = vector(0,0,0)

#   2. 畫面設定
scene = display(width=1000, height=1000, background=(0.5,0.6,0.5))

x_axis = arrow(axis=(1,0,0), shaftwidth=0.01)
y_axis = arrow(axis=(0,1,0), shaftwidth=0.01)
z_axis = arrow(axis=(0,0,1), shaftwidth=0.01)
label(pos=(1.1,0,0), text='x', box = False)
label(pos=(0,1.1,0), text='y', box = False)
label(pos=(0,0,1.1), text='z', box = False)

#   3. 計算和畫出各向量
for vec in vectors:
    arrows.append(arrow(color=color.black, shaftwidth=0.02))
    arrows[-1].axis = vec
    final_vector += vec

final_arrow = arrow(axis= final_vector, shaftwidth=0.02, color=color.red)
print final_vector
```



- 程式說明：

1. 設定初始值、list

```
vectors = [vector(1.5, 0.5, 0), vector(-1, 1, 0)]
```

此處 vectors 屬於一種稱為 list 的資料型態，可以儲存並處理多項資料。在 list 中(被 [ ] 框住)的資料稱為元素(element)或項目(item)，存在 list 中的元素沒有限定種類，可以是任何種類的資料或物件等，在這裡 vectors 中存了兩個 vector。創造一個 list 最簡單的方式就是用中括號 [ ]，在其中放入想存的元素，以逗號分隔。你可以想像，list 就是一個清單，裏面有一串序列的物件。你可以用索引(index)來定位其中的任何元素，每個元素所對應的索引，就是該元素與 list 中第一個元素的索引值的偏移量(offset)，如 list 中的第一個元素的索引值為 0，第二個元素的索引值為 1、第三個元素的索引值為 2…以此類推。例如

A = [1, 2, 3, 4, 5]

代表我們設定一個 list 叫做 A，其中 A[0]的值是 1，A[1]值是 2，以下依此類推。

另外當索引是 -1 時，則指的是最後一個元素，如 A[-1]的值就是 5。

在程式當中，如果要在已存在的 list 增加元素，可利用 append 指令，其用法範例如下：

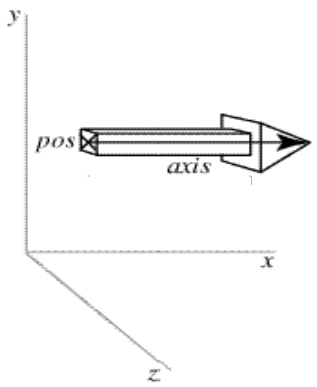
A.append(6)
如此 A 會有六個元素，依序為 1、2、3、4、5、6。而此時，A[-1]的值就是 6。
不含任何元素的 list 也可以使用，如 B=[ ]，或主程式中的 arrows 都是空的 list。

2. 畫面設定

在這裡畫三個箭頭，分別表示 x、y、z 軸的方向。

```
x_axis = arrow(axis=(1,0,0), shaftwidth=0.01)
y_axis = arrow(axis=(0,1,0), shaftwidth=0.01)
z_axis = arrow(axis=(0,0,1), shaftwidth=0.01)
```

物件 arrow 可以讓箭頭出現在畫面上，常用來表示物理量值與方向
x_axis=arrow() 產生一個稱為 x_axis 的箭頭物件。
pos 指定箭頭平端的位置，若未指定，則其內定的值為 vector(0, 0, 0)。
axis 指定箭頭的長度向量，由箭頭的平端指向箭頭的尖端，如圖所示。
shaftwidth 箭頭的寬度，它的預設值是0.1 × axis 向量的量值。



注意，我們也可以先產生物件，之後再設定它的參數，作法如下：

```
x_axis = arrow()
x_axis.pos = (0, 0, 0)
x_axis.axis = (1, 0, 0)
x_axis.shaftwidth = 0.01
```

其實，在 VPython 的物件，如之前學過的 display、box、sphere 等都可以使用這種方法，先產生物件，然後在需要的地方，再指定或改變寫的參數。

我們也為這 x、y、z 三個軸向，分別標上名稱

```
label(pos=(1.1,0,0), text='x', box = False)
label(pos=(0,1.1,0), text='y', box = False)
label(pos=(0,0,1.1), text='z', box = False)
```

當你想要在畫面上作標註文字時，可以使用 label()，其中的
pos 表示標籤所在的位置。
text 則是標籤的內容。
box 表示要不要加框，如果是 True，則會在文字外加個框，False 則不會。

3. 計算和畫出各向量

```
for vec in vectors:
    arrows.append(arrow(color=color.black, shaftwidth=0.02))
    arrows[-1].axis = vec
    final_vector += vec
```

計算合力(合向量)則要把 vectors 中的每個力(向量)加總，以下介紹 python 的一種迴圈方式

for A in B:

意思是將 list B 中的每一個元素，按照順序指定給 A，然後將下方縮排的附屬程式碼執行一次，當把 list B 的全部元素，依照這樣的方式輪過一遍後，就結束整個迴圈。

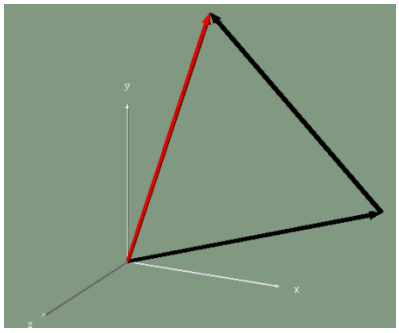
如上所述，append(element)可以放在 list 之後，以增加一個新的元素(element)到 list 的最後面。在此，arrows 原來是一個空的 list，在這段 for 迴圈中，vec 會依序被設定為 vector(1.5, 0.5, 0)和 vector(-1, 1, 0)，然後，迴圈的附屬程式會先產生一個黑箭頭，其寬度為 0.02，並將其加到名為 arrows 的 list 裏，再將其(arrows[-1] 代表 arrow 這個 list 裏最後一個元素，也就是剛剛才產生的箭頭) 長度向量(axis)設定為 vec。接著，以 final_vector += vec，逐次將 final_vector 增加 vec 的向量值，當迴圈全部執行結束後，final_vector 的值就是此二向量的加總。

```
final_arrow = arrow(axis= final_vector, shaftwidth=0.02, color=color.red)
print final_vector
```

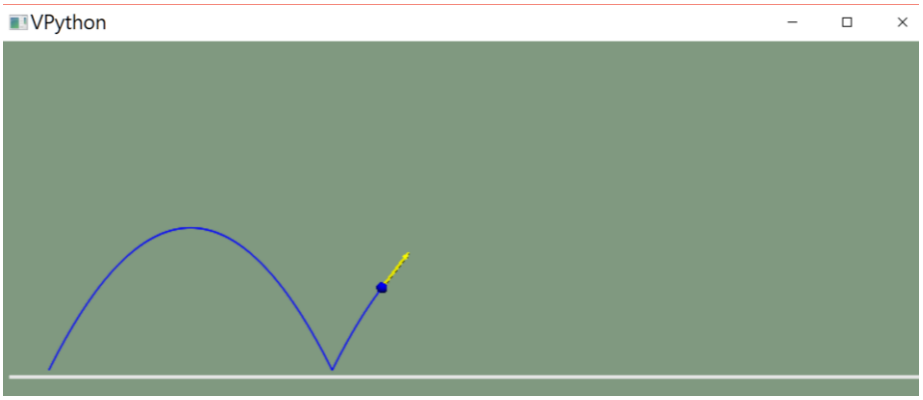
程式最後，再將合力(合向量)的箭頭畫出來，並將值印出來。

- 練習：

1. 請在 vectors 中，增加 1 個或多個力(或向量)，再以此程式做多個力(或向量)的合成。
2. 改寫程式，看你是否可以用一個向量的尾(尖)端作為另一個向量的平端的始，畫出向量和，如右圖，紅色向量是兩個黑色向量的和。



3. 請在單元 3 裏的程式，加適當的程式碼，使得在任何時刻，球上都有一個箭頭(arrow)，並且此箭頭的長度正比於球的速度，且方向平行於球的速度方向。



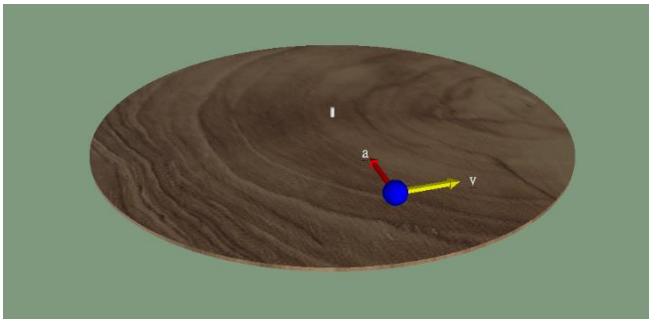
單元 5：等速率圓周運動

- 物理觀念：

  施一向心加速度於有切線初速的物體上，使其做等速率圓周運動

- 範例程式：

```
from visual import *

#   1. 畫面設定
scene = display(width=1000, height=1000, background=(0.5,0.6,0.5))
table = cylinder(pos=(0,-0.03,0), axis=(0,-0.01,0), radius=0.7, material=materials.wood)
center = cylinder(pos=(0, -0.03, 0), axis = (0, 0.03, 0), radius = 0.007)
ball = sphere(pos=(-0.5,0,0), radius=0.03, color=color.blue)

##1

#   2. 設定參數、初始值
ball.v = vector(0, 0, 0.5)
r = abs(ball.pos)
print "velocity = ", abs(ball.v)
print "period = ", 2*pi*r/ abs(ball.v)
dt = 0.001


#   3. 運動部分
while True:
    rate(1000)

    ball.a = - (abs(ball.v)**2 / r) * (ball.pos/r)
    ball.v += ball.a*dt
    ball.pos += ball.v*dt

    ##2
```



- 程式說明：

1. 畫面設定

畫出模擬圓周運動的球(ball)，桌面(table)和桌面中心(center)。

```
table = cylinder(pos=(0,-0.03,0), axis=(0,-0.01,0), radius=0.7, material=materials.silver)
```

為了畫圓桌，我們使用可以產生圓柱的新物件 cylinder()

pos 是圓柱一端的中心位置，以三維座標表示。

axis 指的從 pos 指到圓柱另一端的向量，以設定圓柱的方向。

radius 設定圓柱的半徑。

length 設定圓柱的長度。

material 設定材質。

2. 設定參數、初始值

```
ball.v = vector(0, 0, 0.5)
r = abs(ball.pos)
print "velocity = ", abs(ball.v)
print "period = ", 2*pi*r/ abs(ball.v)
```

設定球初速為 vector(0, 0, 0.5)，並將此速度向量的絕對值以指令 print “velocity = ”, abs(ball.v) 顯示出來，這裏 abs( )會計算出在括號中的變數的絕對值，之後再計算出此圓周運動的週期。r 則是球繞中心作圓周運動的半徑，所以周期是 2*pi*r/abs(ball.v)。

3. 運動部分

```
ball.a = - (abs(ball.v)**2 / r) * (ball.pos/r)
```

我們知道等速率圓周運動的向心加速度為$a = v^2/r$，其方向為向著圓心，故可以將其改寫為 ball.a = - (abs(ball.v)**2 / r) * (ball.pos/r)，注意，這裏 ball.a 和 ball.pos 都是向量，有個負號是因為這是向心加速度，所以加速度方向和位置向量是相反的。算出加速度後，再接著算出速度和位移，就可以得到圓周運動的模擬。

- 練習：

1. 將下列 4 行程式加到原始程式中##1 的下方

```
a_arrow = arrow(shaftwidth = 0.01, color = color.red)
v_arrow = arrow(shaftwidth = 0.01, color = color.yellow)
a_label = label(text='a', height=15, opacity = 0, box= False)
v_label = label(text='v', height=15, opacity = 0, box= False)
```

將下列 4 行程式加到原始程式中##2 的下方(注意程式碼縮排的位置)

```
    a_arrow.pos, a_arrow.axis = ball.pos, ball.a/3
    v_arrow.pos, v_arrow.axis = ball.pos, ball.v/3
    a_label.pos = a_arrow.pos + a_arrow.axis*1.2
    v  label.pos = v  arrow.pos + v  arrow.axis*1.2
```

執行程式後，你會發現在模擬的畫面上，會有加速度 a 和速度 v 的箭頭和標示。觀察一下，看看這些功能，是如何由上述幾行程式來達成的。

注意：在這段程式裏，我們有一個新用法，就是 a_arrow.pos, a_arrow.axis = ball.pos, ball.a/3 會直接讓等號右邊的各數值，依序指定到等號左邊的變數中。所以在使用時要小心，等號兩邊的數值或變數個數要相同。

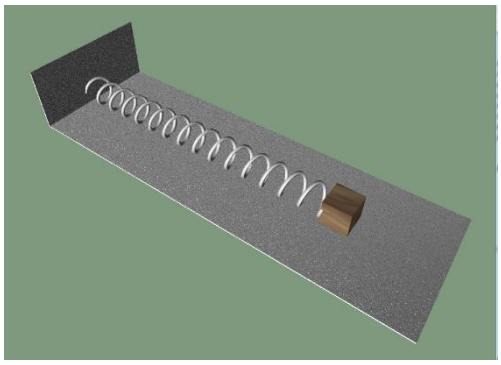
2. 改變球的初速、或球的初始位置，觀察模擬的結果會如何改變。
3. 用手表計時，讓球繞中心轉 5 圈以上，以得到平均週期，此週期和理論計算結果是否相同？

單元 6：虎克定律和簡階運動

- 物理觀念：

利用彈簧施力於木塊，使物體作簡階運動

- 範例程式：



```
from visual import *
#    1. 參數設定
m = 0.5                    #木塊質量 0.5 kg
k = 10.0                   #彈簧的彈性性數  10 N/m
v0 = 2                     #木塊的初速  2 m/s
dt = 0.001

#    2. 畫面設定
scene = display(width=1000, height=1000, background=(0.5,0.6,0.5))
bottom = box(length=3, height=0.01, width=1, material=materials.silver)
wall = box(length=0.01, height=0.5, width=1, material=materials.silver)

square = box(length=0.2, height=0.2, width=0.2, material=materials.wood)          #正立方木塊
spring = helix(pos=(-bottom.length/2,0,0), radius=0.06, coils=15, thickness = 0.03)          #以螺旋畫出彈簧

bottom.pos = (0,-square.height/2,0)
wall.pos = (-bottom.length/2,0.5/2-square.height/2,0)
square.vx = v0

#    3. 運動部分
while True:
    rate(1000)
    square.a = -(k/m)*square.pos.x                    #彈簧的加速度，= ( k / m ) *  物體在+x 的位移量
    square.vx += square.a*dt
    square.pos.x += square.vx*dt
    spring.length = (square.pos.x-square.length/2)-spring.pos.x          #求出彈簧的長度
```

- 程式說明：

1. 參數設定

設定要模擬現象的物體質量、彈簧彈性系數，物體的初速等。

2. 畫面設定

畫出各項物體，並設定這些物體的位置和大小，木塊的初始位置(square.pos)在程式中未設定，所以電腦預設為 vector(0, 0, 0)，這是 Vpython 的內建設定，如果畫出一個物體而沒有給定位置，則會設為預定值 vector(0, 0, 0)；另設定木塊在 x 方向的初速(square.vx)為 v0。

```
spring = helix(pos=(-bottom.length/2,0,0), radius=0.06, coils=15, thickness = 0.03 )
```

Vpython 中有一個物件叫 helix，可以產生螺旋狀的物體，我們利用這個形狀來畫彈簧

helix() 中可使用的參數有

pos 代表螺旋物件一端所在的位置(注意：不是中心)，以三維座標表示。

axis 表示從 pos 指到螺旋物件另一端的方向向量。
radius 設定螺旋物件的半徑長。
length 設定螺旋物件的長度。
coils 設定螺旋物件的匝數。
thickness 設定螺旋物件線的寬度。

3. 運動部分

木塊位置的 x 座標(square.pos.x)，就是和原點位置相比較的位移量，也是彈簧的伸長量，此量乘以(-k)，即是彈簧的作用力，此力正比於伸長量，且方向和彈簧的伸長量相反。此力作用於質量為 m 的木塊，可使木塊有加速度 square.a = -(k/m)*square.pos.x。

```
spring.length = (square.pos.x-square.length/2)-spring.pos.x
```

因為彈簧的長度會隨物體運動而改變，所以一端的位置固定在牆上，長度則依物體的運動的隨時改變

- 練習：

1. 請以手碼表或手表計時，測量此木塊來回一次所需的時間，即為此簡階運動的週期。多測量幾次，再和課本中學到的彈簧的簡階運動作比較。
2. 可以改變木塊的質量、彈簧的彈性係數、木塊的初速，並觀察這些改變對週期的影響。

- 進階練習：

1. 你可以加一段程式碼來計算木塊來回一次的周期嗎？提示：你先想想看，木塊到達最右邊頂點時，位移、速度、加速度會有什麼特性？從這裏出發，並參考單元 2 的觸地反彈的處理方式即可。
2. 我們知道空氣阻力和速度方向相反，且多數情形下和速度成正比，假設此系數為 100.0，在計算加速度的程式碼 square.a = -(k/m)*square.pos.x 中，加入空氣阻力的影響後，觀察模擬結果有何變化。如果係數改為 300 呢或更大的數值呢？
3. 改寫程式，畫一天花板，將彈簧改為懸掛在天花板下，同時考慮重力，模擬一個垂直的簡階運動，觀察其週期和水平的簡階運動是否有所不同？

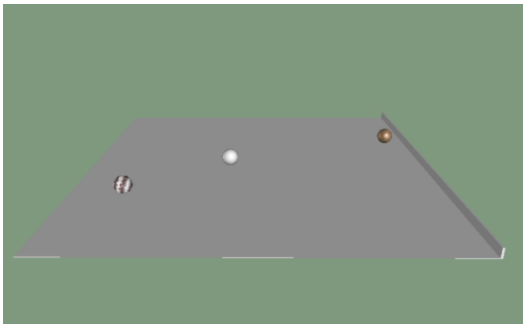
單元 7：動量

- 物理觀念：

比較有相同動量但不同質量的物體速度的快慢

- 範例程式：

```
from visual import *

#   1. 參數設定
density ={'wood':400.0, 'plastic':900.0, 'marble':2600.0}          #物質密度 單位: kg/m**3
size = 0.05                  #球半徑 0.05 m
L = 1.00                     #地板長
dt = 0.001                   #兩連續畫面間之時間間隔
V = (4/3)*pi*(size)**3   #體積
P = 0.1                      #初始動量 kg*m/s

#   2. 畫面設定
scene = display(width=800, height=800, background=(0.5,0.6,0.5))
bottom = box(pos=(0,-size,0), length=2*L, height=0.001, width=2)
wall = box(pos=(L,-size/2,0), length=0.01, height=size, width=2)

#   3. 球的設定
balls = {}
balls['wood'] = sphere(pos=(-L,0,-0.40), radius=size, material=materials.wood)
balls['plastic'] = sphere(pos=(-L,0,0), radius=size, material=materials.plastic)
balls['marble'] = sphere(pos=(-L,0,0.40), radius=size, material=materials.marble)

for material in balls.keys():
    balls[material].m = V*density[material]
    balls[material].v = P/balls[material].m

#   4. 運動
while True:
    rate(1000)
    for material in balls.keys():
        balls[material].pos.x += balls[material].v *dt
        balls[material].rotate(axis=(0,0,1), angle=-balls[material].v*dt/size/pi)
        if balls[material].pos.x >= L-size :
            balls[material].v = 0
```



- 程式說明：

1. 參數設定

```
density ={'wood':400.0, 'plastic':900.0, 'marble':2600.0}          #物質密度 單位: kg/m**3
```

單元 4 用 list 儲存一個序列的資料，這裏我們使用一個稱為 dictionary 新的資料類型。dictionary 與 list 差別在它的索引值(indices)可以是任何種類(list 的索引只可以是 0 或正整數)，可以把 dictionary 想像成一個 list 但是其索引值 (以下稱為關鍵字，keyword) 可以是任意的，每一個關鍵字都會對應到一個值 (item)。創造 dictionary 的方式不只一種，上述框中為其中一種，先宣告 dictionary 的變數名，以{}代表 dictionary 的內容，每項的內容用逗號分開，每一項會包含一個冒號(:)，冒號左邊是此項的關鍵字，冒號右邊是此項的值。如上列方框，名為 density 的

dictionary 中，關鍵字 ’wood’ 對應項的值是 400.0，’plastic’ 對應項的值是 900.0。如果我們利用 print 指令來顯示的話，用法就是 print density['wood']，這裏 density['wood']，此的就是 wood 的密度。下面還會介紹另一種 dictionary 的設定方法。

2. 畫面設定

開視窗，畫地板和擋板

3. 球的設定

```
balls = {}
balls['wood'] = sphere(pos=(-L,0,-0.40), radius=size, material=materials.wood)
balls['plastic'] = sphere(pos=(-L,0,0), radius=size, material=materials.plastic)
balls['marble'] = sphere(pos=(-L,0,0.40), radius=size, material=materials.marble)
```

在這裏，我們介紹另一種產生 dictionary 的方法，我們先令 balls 為一個沒有儲存任何東西，空的 dictionary。接著，我們以 sphere 產生一個球體，其 material 為 materials.wood，並令其為 balls['wood'] 的值，並以相同的方法，產生另兩個球體，分別令他們為 balls['plastic'] 和 balls['marble']。此時，balls['wood']、balls['plastic'] 和 balls['marble']分別代表三個球體，關於球體的任何操作，都可以用於 balls['wood']、balls['plastic'] 和 balls['marble']。

```
for material in balls.keys():
    balls[material].m = V*density[material]
    balls[material].v = P/balls[material].m
```

接下來，我們以 for 指令來操作 balls 中的每一個球體，這裏是以 balls.keys()來取得 balls 中的每一個關鍵字(keyword)，以 for 來指定變數 material 依序為 balls 中的關鍵字，然後以其下的兩行，來得到 balls['wood']、balls['plastic'] 和 balls['marble'] 的質量，與利用動量除以質量來指定他們各自的初速。

4. 運動

在 while 迴圈中，我們再以相同的 for 指令，去計算每一個球在新的時刻的位置，並以

```
balls[material].rotate(axis=vector(0,0,1), angle=-balls[material].v*dt/size/pi)
```

方框中附屬於球體的方法 rotate，去轉動球，球的轉動角度是以其移動的位移來決定，其轉軸則設為 vector(0, 0, 1)。

- 練習：

1. 按碼表計時每個球體在走完 1m 的距離所需的時間，計算其速度，看是否與利用動量和質量算出來的速度相等。
2. 增加一個球體，指定其材質，設定其密度，並執行程式。

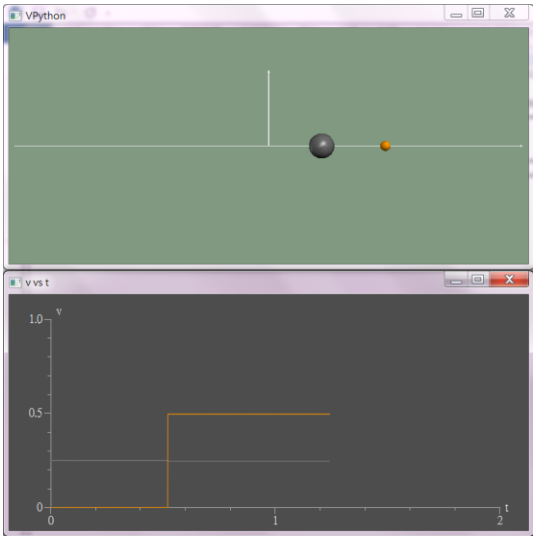
單元 8：彈性碰撞

- 物理觀念：

一維空間的彈性碰撞

- 範例程式：

```
from visual import *
from visual.graph import *

#   1.  畫面設定
scene1 = display(width=600, height=400, background=(0.5,0.6,0.5), y=0)
arrow1 = arrow(display=scene1, pos=(-1,0,0), axis=(2,0,0), shaftwidth=0.005)
arrow4 = arrow(display=scene1, pos=(0,0,0), axis=(0,0.3,0), shaftwidth=0.005)

gd1 = gdisplay(x=600, y=0, title='v vs t', xtitle='t', ytitle='v', ymax=1, xmax=2, background=(0.3,0.3,0.3))
vt1 = gcurve(gdisplay=gd1, color=(0.5, 0.5, 0.5))
vt2 = gcurve(gdisplay=gd1, color=color.orange)
vts= (vt1, vt2)

#   2.  物體設定
ironball = sphere(display=scene1, radius=0.05, pos=(-0.2,0,0), color=(0.5, 0.5, 0.5), material=materials.rough)
ironball.m, ironball.v = (pi*4/3*(0.05)**3)*7.9*1e3, 0.25

ping_pong = sphere(display=scene1, radius=0.02, pos=(0.1,0,0), color=color.orange)
ping_pong.m, ping_pong.v = 2.6*10e-3, 0
balls=(ironball, ping_pong)

#   3.  定義函數
def collide(v1,v2,m1,m2):
    v1f = v1*(m1-m2)/(m1+m2) + v2*2*m2/(m1+m2)
    v2f = v1*2*m1/(m2+m1) + v2*(m2-m1)/(m2+m1)
    return v1f, v2f

#   4.  物體運動
dt = 0.001
t = 0
while t < 2:
    rate(200)
    t += dt

    for (ball, vt) in zip(balls, vts):
        ball.pos.x += ball.v*dt
        vt.plot(pos=(t,ball.v))

    if abs(ironball.pos-ping_pong.pos) < (ironball.radius+ping_pong.radius) and ironball.v > ping_pong.v:
        ironball.v, ping_pong.v = collide(ironball.v,ping_pong.v,ironball.m,ping_pong.m)
```



- 程式說明：

1. 畫面設定

開模擬視窗，並畫出座標軸。另開 vt 圖視窗，設定兩碰撞物體各自的速度-時間曲線物件，並將兩個曲線物件，一起放入叫 vts 的 tuple 裏，tuple 是 python 一個常用的資料類型，例如 a = (0.05, 2) 就是一個 tuple，tuple 由小括號和其中的元素(元素可以是資料類型或物件)所組成，

元素個數不限，也不一定要屬於相同的類型或物件，本程式裏，vts= (vt1, vt2) 是由兩個曲線物件所組成的 tuple。tuple 的元素超過一個的話，小括號可省略，如 a = 0.05, 2 或 vts = vt1, vt2。

2. 物體設定

```
ironball = sphere(display=scene1, radius=0.05, pos=(-0.1,0,0), color=(0.5, 0.5, 0.5), material=materials.rough)
ironball.m, ironball.v = (pi*4/3*(0.05)**3)*7.9*1e3,    0.25

ping_pong = sphere(display=scene1, radius=0.02, pos=(0.1,0,0), color=color.orange)
ping_pong.m, ping_pong.v = 2.6*10e-3, 0
balls=(ironball, ping_pong)
```

方框內第一行程式畫出一個球體(ironball)，作為碰撞系統中的鐵球。第二行設定鐵球的性質，當等號兩邊都是 tuple 時，會啟用一種特別的指定方式(稱為 tuple assignment)，將等號右邊 tuple 內的資料，依序指定給等號左邊 tuple 內的變數，方框內第二行就是把半徑為 0.05m 的鐵球的質量算出來，指定給 ironball.m，把鐵球初速 ironball.v 設定為 0.25 m/s。其下兩行程式，則畫出乒乓球(ping_pong)，並指定其質量和初速。最後將 ironball 和 ping_pong 一起放入 balls 的 tuple 裏，方便之後使用。

3. 定義函數

```
def collide(v1,v2,m1,m2):
    v1f = v1*(m1-m2)/(m1+m2) + v2*2*m2/(m1+m2)
    v2f = v1*2*m1/(m2+m1) + v2*(m2-m1)/(m2+m1)
    return v1f, v2f
```

函數(function)是將一部分的程式碼命名，以進行特定的運算，會使用 function 有幾個原因，其中最常見的有：(一)有些程式碼需要重複使用，將這些程式碼事先寫好並命名，等要使用的時候直接呼叫函數即可，可以減少程式碼；(二)將複雜的程式，化簡成幾個區塊，各別寫成函數，再按照該區塊的功能，給予一個名字，以便於整體程式的撰寫、閱讀和維護。

撰寫函數並不困難，第一行定義函數，先打 def，空格，再按照你要賦予這個函數的功能，給這個函數一個名字(這裡是 collide)，後面接小括弧，小括弧內列出需要用到的參數，這裏是 (v1, v2, m1, m2)，分別代表物體 1 和物體 2 的速度和質量，若有多個參數則用逗號分開，最後加上冒號。冒號以下，也就是第二行以後的文字需要縮排，表示這部分的程式碼，是函數 collide 的附屬程式碼，也就是 collide 被呼叫時，所要執行的程式碼。而這段程式碼會計算兩物體碰撞後的速度，指定給變數 v1f、v2f。我們是直接使用物理課本中的公式 $v_{1f} = \frac{m_1-m_2}{m_1+m_2} v_1 + \frac{2m_2}{m_1+m_2} v_2$，和 $v_{2f} = \frac{2m_1}{m_1+m_2} v_1 + \frac{m_2-m_1}{m_1+m_2} v_2$。函數程式中的最後一行是 return，目的在將執行此函數後的結果回傳，在這裡會回傳 v1f、v2f 兩個值。所以這一個函數的目的就是，輸入 v1、v2、m1、m2，計算並回傳 v1f、v2f，後面會有如何使用此函數的說明。

4. 物體運動模擬主程式

在主程式中，我們設定這個模擬世界一開始的時間是 t=0，讓 dt=0.001 而在 while 迴圈裏，rate(200)，表示這模擬是以真實世界的 0.2 倍速在做模擬。t += dt，則每更新一次畫面，模擬的總時間就增加 dt，這個 t 就是“速度-時間圖”的時間軸 t。

```
for (ball, vt) in zip(balls, vts):
    ball.pos.x += ball.v*dt
    vt.plot( pos=(t,ball.v) )
```

這個方框裏，我們可以用一個叫 zip 的函數做“拉拉鏈”的動作，它會將 balls 和 vts 兩個 tuple 中各別的元素，按順序放在一起，產生一個新的 list，這個 list 的元素，是由 balls 和 vts 中的元素依照順序合成的新 tuple，換句話說，zip(balls, vts) 得到的結果就是一個 list，它的內容為 [(ironball, vt1), (ping_pong, vt2)]，就好像在拉拉鏈時，將左右兩邊的拉鏈，由上而下將各齒依序結合在一起。我們以 for 做迴圈執行，先指定 (ball, vt) = (ironball, vt1) 執行 for 的附屬程式，這一段附屬程式碼會先計算鐵球的新位置，然後將此時刻的時間(t)和此時刻的鐵球速度(ball.v)以 pos=(t, ball.v) 加到對應的速度時間曲線上。執行完後，再令 (ball, vt) = (ping_pong, vt2)，執行同一段程式碼，對乒乓球做同樣的事情。

```
if abs(ironball.pos-ping_pong.pos) < (ironball.radius+ping_pong.radius) and ironball.v > ping_pong.v:
    ironball.v, ping_pong.v = collide(ironball.v, ping_pong.v, ironball.m, ping_pong.m)
```

接著，我們要處理的是兩球間的碰撞，當兩球的球心距離小於他們的半徑和、且鐵球的速度比乒乓球還快時，表示鐵球已追上乒乓球，即發生碰撞，這時就以 ironball.v, ping_pong.v = collide(ironball.v, ping_pong.v, ironball.m, ping_pong.m) 呼叫我們之前寫好的碰撞速度處理程式

```
def collide(v1,v2,m1,m2):
    v1f = v1*(m1-m2)/(m1+m2) + v2*2*m2/(m1+m2)
    v2f = v1*2*m1/(m2+m1) + v2*(m2-m1)/(m2+m1)
    return v1f, v2f
```

將兩邊互相對照，python 在呼叫碰撞速度處理程式(collide)時，會按照順序將參數值設定給被呼叫的函數，如 ironball.v 會指定給 v1，ping_pong.v 指定給 v2，ironball.m 指定給 m1，ping_pong.m 給 m2，並加以執行後，最後將算出來的 v1f 和 v2f，以指令 return 利用 tuple assignment 的方式，設定給 ironball.v，和 ping_pong.v，也就是 ironball.v 會被 collide 更新，設定為算出來 v1f 的值，ping_pong.v 會被更新，設定為算出來 v2f 的值。因為 while 迴圈的條件設定為 t<2，所以整個運動過程的模擬，會執行虛擬世界中 2 秒，直到 t 不小 2 時才停止。

- 練習：

1. 在鐵球碰撞到乒乓球前，鐵球初速不為 0，而乒乓球初速為 0，碰撞後，鐵球速度幾乎不變，而乒乓球速度變為鐵球速度的 2 倍，從碰撞方程式，可以看出為什麼會有這樣的結果嗎？
2. 把鐵球和乒乓球的位置和初速互換，再將 while 中判斷兩球是否碰撞的條件，鐵球速度大於乒乓球改為乒乓球的速度大於鐵球，就是改以乒乓球去撞靜止鐵球，你觀察到什麼結果？
3. 把乒乓球改為和鐵球一樣的質量(變數名不變沒關係，只要改他的質量數值即可)，執行模擬程式，你觀察到什麼結果？

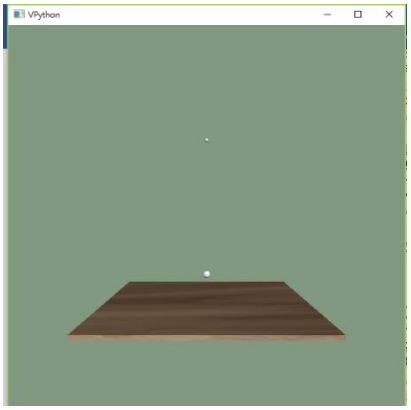
單元 8 附加實作：兩自由落體球間的彈性碰撞

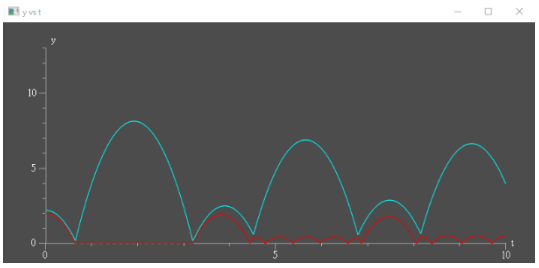
- 物理觀念：

兩相鄰自由落體間的彈性碰撞所造成的能量轉移

- 範例程式：

```
from visual import *
from visual.graph import *

#   1.  參數設定
g = -9.8
dt, t = 0.0001, 0

#   2.  畫面設定
scene = display(width=600, height=600, background=(0.5,0.6,0.5), center=(0,2,0), range=5)
gd = gdisplay(x=600, y=0, title='y vs t', xtitle='t', ytitle='y', ymax=13, xmax=10, background=(0.3,0.3,0.3))
bottom = box(pos=(0,-0.2,0), length=5, height=0.1, width=5, material=materials.wood)
yts = ( gcurve(gdisplay=gd, color=color.cyan), gcurve(gdisplay=gd, color=color.red) )          #兩球高度時間圖

#   3.  球的設定
ball1 = sphere(radius=0.05, color=color.white, v=0, m=0.2)
ball2 = sphere(pos=(0,2,0), radius=0.1, color=color.white, v=0, m=0.6)
ball1.pos=(0,2+ball1.radius+ball2.radius+0.05,0)
balls = (ball1, ball2)

#   4.  函數定義
def collide(v1,v2,m1,m2):
    v1f = v1*(m1-m2)/(m1+m2) + v2*2*m2/(m1+m2)
    v2f = v1*2*m1/(m2+m1) + v2*(m2-m1)/(m2+m1)
    return v1f, v2f

#   5.  運動部分
while t<=10:
    rate(2000)
    t += dt
    for (yt, ball) in zip(yts, balls):
        ball.v += g*dt
        ball.pos.y += ball.v*dt
        yt.plot( pos = (t, ball.pos.y))
        if ball.pos.y <= 0:
            ball.v = abs(ball.v)
    if abs(ball1.pos-ball2.pos) <= ball1.radius + ball2.radius :
        ball1.v, ball2.v = collide(m1=ball1.m, v1=ball1.v, v2=ball2.v, m2=ball2.m)
```





- 程式說明：

利用之前單元(含物理和程式兩方面)所學過的內容來做模擬，執行時你會看到當兩物體做自由落體運動時，能量的交互傳遞情形。整個程式中除最後一行外的用法，都在之前的單元學過，而最後一行關於呼叫函數的方式，則和單元 8 有所不同，單元 8 的呼叫寫法是 ball1.v, ball2.v = collide(ball1.v, ball2.v, ball1.m, ball2.m)，會依序將 ball1.v 指定給 v1，ball2.v 給 v2，ball1.m 給 m1，ball2.m 給 m2，這種方法是依照位置來設定，所以叫 call by position，而在這裏是以參數名稱來指定，所以叫做 calll by keyword。你可以看到，此程式中產生物件的方式，如 ball1 = sphere(radius=0.05, color=color.white, v=0, m=0.2)，也是一種 call by keyword 的概念。

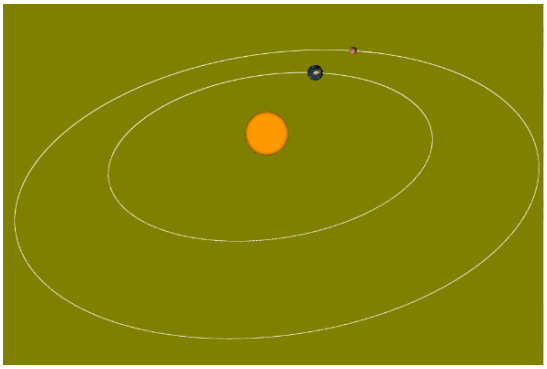
單元 9：行星公轉

- 物理觀念：

太陽和行星之間的萬有引力，做為行星公轉的向心力

- 範例程式：

```
from visual import*

#     1. 參數設定和畫面準備
G=6.673E-11
mass = {'earth': 5.97E24, 'mars':6.42E23}       # 以 dictionary 存放地球、火星的質量
d_at_perihelion = {'earth': 1.495E11, 'mars':2.279E11}          # 近日點時，地球或火星到太陽的距離
v_at_perihelion = {'earth': 2.9783E4, 'mars':2.4077E4}          # 在近日點時，地球或火星的移動速度

scene = display(width=1000, height=1000, background=(0,0,0))
sun = sphere(pos=vector(0,0,0), radius = 2.1E10, color = color.orange, material = materials.emissive)
scene.lights = [local_light(pos=(0,0,0), color=color.white)]

#     2. 行星物件產生類別
class planet_c(sphere):
      m_sun = 1.99E30         # 太陽質量
      def a(self):
            return - norm(self.pos) * G * self.m_sun / mag2(self.pos)

#     3. 產生地球和火星
earth = planet_c(pos = vector(d_at_perihelion['earth'],0,0),radius = 9.5E9,
                        material = materials.earth, make_trail = True, retain = 365 *24)
earth.rotate(angle = pi/2, axis=(1, 0, 0))
earth.m, earth.v = mass['earth'], vector(0, v_at_perihelion['earth'], 0)
mars = planet_c(pos = vector(d_at_perihelion['mars'],0,0), radius = 4.9E9,
                        material= materials.bricks, make_trail = True, retain = 700 * 24)
mars.m, mars.v = mass['mars'], vector(0, v_at_perihelion['mars'], 0)
planets = [earth, mars]

#     4. 運動部分
dt= 60*60
while True:
      rate(6*24)
      for planet in planets:
            planet.v += planet.a()* dt
            planet.pos += planet.v * dt
```



- 程式說明：

1. 參數設定和畫面準備

設定地球、火星等要模擬天體的參數。

開視窗、畫太陽，另外再用下列的程式碼控制光源，讓模擬世界的光源發自原點 vector(0, 0, 0)，也就是太陽的中心位置，讓光看起來是由太陽發出來的。為了在模擬中可以看到太陽，在此設定太陽的半徑是真實數值的 30 倍。

```
scene.lights = [local_light(pos=(0,0,0), color=color.white)]
```

2. 產生行星的類別

在看這一段程式碼之前，我們先來看看，在我們要模擬的世界中，一個行星和之前模擬過的球有什麼差別，兩個一樣都是球體，有位置、大小、材質等特性，其不同處在行星多了一些特性，如行星會因太陽的引力作用而有加速度，如果我們可以把之前使用的球體物件加以擴充，那麼程式寫起來，會更有結構性和可讀性。

到目前為止，我們已經使用過可以產生物件的幾個 vpython 內建的類別(類別在 python 裏稱為 class)，像是 box、sphere，vector…等，我們以這些類別產生物件(如球、地板、彈簧，力的向量等)，現在我們要創造一個新的類別，並以此類別產生物件。建立新的類別，要以關鍵字 class，空一格後，加上我們為這個類別的命名作宣告，括弧內則是要這個類別所要繼承性質的

```
class planet_c(sphere):
    m_sun = 1.99E30        #  太陽質量
    def a(self):
        return - norm(self.pos) * G * self.m_sun / mag2(self.pos)
```

已存在類別。所以方框中第一行程式，會創造一個新類別 planet_c，它繼承了 sphere 類別。類別繼承是 python 很有用的工具，有一點複雜，在此我們只說明其最簡單的概念，因為 planet_c 繼承了 sphere，所以之後用 planet_c 創造出的物件，除了有 planet_c 自己的參數和函數外(如這裏的 m_sun 和 a)，也有 sphere 類別的所有特性、參數和函數。在這裏 a 是 planet_c 專有的函數，這種屬於 class 的函數稱為方法(method)，其作用和之前學的函數(function)一樣，但是這些方法只專屬於這個類別所產生出來的物件。這種以物件和類別為主體的程式，就稱為"物件導向程式"(Object-Oriented Programming，簡稱 OOP)。

在 planet_c 中只有一個方法，def a(self)，在括弧內唯一的參數是 self，它代表呼叫這個方法的物件，例如在 5.運動部分，以 planet.a()來呼叫此方法，則此時的 self 就會被指定為這個 planet；注意，類別中的方法，也可以有其他的參數，用法就像是之前學的函數一樣，但是在所有的方法中，self 必須出現且永遠要放在第一個參數位置，來代表呼叫這個方法的物件。在 def a(self)裏，計算的是太陽對行星(self)施予萬有引力所產生的加速度 $a = G\frac{M_{sun}}{r^2}$ ，所以方向是由行星的位置向著太陽，而太陽的位置在原點，所以在 a()這個方法中，會以 norm(self.pos)來計算 planet 位置向量的單位向量，前面加個負號，表示這個向量方向是由行量朝向原點，也就是太陽的球心，G*self.m_sun/mag2(self.pos)則計算出加速度的量值，其中 self.m_sun 就是代表 class 宣告中的 m_sun，而 mag2(self.pos)則計算 self.pos 量值的平方。

3. 產生地球和火星

```
earth = planet_c(pos = vector(d_at_perihelion['earth'],0,0), radius = 9.5E9,
                 material = materials.earth, make_trail = True, retain = 365 *24)
earth.rotate(angle = pi/2, axis=(1, 0, 0))
earth.m, earth.v = mass['earth'], vector(0, v_at_perihelion['earth'], 0)
```

第一行，宣告 earth 是由 planet_c 類別所產生的物件，其中我們讓地球的初始位置設定在近日點，為了在模擬中可以看到地球，讓地球的半徑設成真實量值的 1500 倍，另外以 retain 讓軌跡持續留在 365*24 個模擬畫面中，配合之後的 dt=60*60 秒=1 小時，即每個畫面之間的時間

差是 1 小時，所以軌跡留下來的時間在模擬世界中會剛好是一個地球年。
第二行是將地球的北極轉北方。第三行，則是設定地球在近日點的速度。
火星的部分，也用相同的方式處理。

4. 運動部分

設定 rate(6*24)，表示真實世界的時間每秒鐘執行 6*24 次畫面更新，配合 dt=1 小時，我們可以知道在真實世界中的 1 秒鍾會等於模擬世界中的 6 天。

```
planet.v += planet.a()* dt
```

方框中是以呼叫物件方法的方式，來得到行星的加速度，並以此來計算這一段時間內，行星速度的變化量。雖然在呼叫方法的括號中沒有任何參數，但是在呼叫過程中，planet 的值會被指定給在類別 planet_c 中的方法 def a(self)中的參數 self。

- 練習：

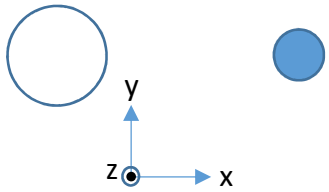
1. 請在 class planet_c 中加一個方法 def k_energy(self)，計算行星的動能，並且在迴圈 while True 之前，將地球和火星的動能以下面程式碼列印出來：
   print 'earth kinetic energy = ', earth.k_energy()
   print 'mars kinetic energy = ', mars.k_energy()
2. 在 earth.rotate(angle = pi/2, axis=(1, 0, 0)) 的下一行加入下面的程式碼
   earth.rotate(angle = 23.5*pi/180, axis = (0, 1, 0))
   在 planet.pos += planet.v * dt 後，且以一樣的縮排，加以下程式碼
   planet.rotate(angle = 2*pi/24, axis = (sin(23.5*pi/180), 0, cos(23.5*pi/180)))
   如此則可以模擬地球和火星的自轉和四季(附註：地球和火星的自轉軸傾角差不多，且太陽日的長度也差不多相等。)

如何作專題：以月球軌道的進動為例

```
from visual import*
G=6.673E-11
mass = {'earth': 5.97E24, 'moon': 7.36E22, 'sun':1.99E30}
earth_orbit = {'r': 1.495E11, 'v': 2.9783E4}
moon_orbit = {'r': 3.84E8, 'v': 1.022E3}
radius = {'earth': 6.371E6*10, 'moon': 1.317E6*10, 'sun':6.95E8*10}        #10 times larger for better view
theta = 5.145*pi/180.0
```

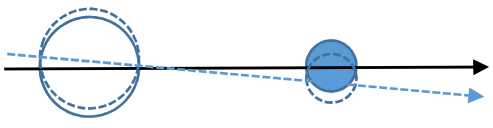
以上程式碼提供你做月球軌道的進動模擬所需要的參數，其中mass代表的是地球、月球、和太陽的質量，earth_orbit['r']和earth_orbit['v']分別代表的是“地月系統”(即地球加月球的系統)公轉繞太陽軌道的半徑和速度，moon_orbit['r']和moon_orbit['v']代表的是月球公轉繞地球軌道的半徑和速度，theta代表地球公轉軌道面和月球公轉軌道面之間的夾角；另外，為了比較好觀察，在這些參數裏，地球、月球、和太陽的球體半徑（radius）被刻意放大10倍。

1. 以 sphere()建構兩個物件 earth 和 moon，設定質量 earth.m = mass['earth'] 和 moon.m = mass['moon']，設定位置 earth.pos = vector(0, 0, 0)，moon.pos = vector(moon_orbit['r'], 0, 0)，再設定月球的軌道初始速度 moon.v = vector(0, 0, -moon_orbit['v'])。在這裏（如右圖）我們的設定是月球繞地球的軌道面在 x-z 平面，月球在地球的右方（+x 方向）以初速向內（-z 方向）繞地球作公轉。接著，在 while 的迴圈裏，地球的位置不作改變，計算地球施予月球的萬有引力，月球因為這個萬有引力所獲得的加速度，和因為這加速度所改變的速度與位置，你將可以看到月球繞著地球作公轉。 你可在開啟視窗後（scene = display (…)的下一行），加一行改變視角的設定 scene.forward = vector(0, -1, 0)，改為由上而下觀察月球的公轉。



2. 然而，我們知道在 1 部分的模擬不完全正確，因為地球也受到萬有引力的作用而會移動，所以 1 的程式要略加修改，以計算月球施予地球的萬有引力，以及地球因為此萬有引力所獲得的加速度，和因為此加速度所改變的速度與位置。模擬後，你會看到地球也會作一個較小的圓運動，這是因為地球的質量遠大於月球，所以地月系統的質心會落在地球裏面，而月球和地球事實上是各自以地月系統的質心為圓心在做公轉運動。但是，除此之外，如果你由上而下觀察月球的公轉還會看到月地系統向 -z 方向移動，這是因為整個系統在初始時，就已經有一個不為零的往 -z 方向的動量。請修改一下你的初始條件，(即 earth.pos, earth.v, moon.pos, moon.v)使得地月系統的質心落在原點，且不隨時間改變。

3. 因為之後要設定地月系統繞太陽的公轉軌道在 x-z 平面上，所以需要改變月球和地球的初始位置，使得月球繞地球的公轉軌道平面會和 x-z 平面夾 theta 的角度。改好之後，在模擬時，你就會看到月球以有傾角的軌道面，繞地球公轉。在這裏可以把 scene.forward = vector(0, -1, 0) 這行程式碼刪掉或註記(在前方加#，不執行此行程式碼)，從側方來觀察月球的公轉運動。



另外，為了改以地球為中心來看月球的公轉，我們可以在計算地球位置後的下一行加一行程式碼 scene.center = earth.pos。這樣看到的就是地球位於視窗中心不動，而月球以傾斜的軌道繞地球公轉。

4. 現在加上太陽，以 sphere()建構物件 sun，設定質量 sun.m = mass['sun']並設定太陽的位置在原點，即 sun.pos = vector(0, 0, 0)。將地球和月球的初始位置作調整，分別再加上“地月系統”的質心位

置 vector(earth_orbit['r'], 0, 0)，也將地球和月球的初始速度作調整，分別加上“地月系統”的質心速度 vector(0, 0, -earth_orbit['v'])。另外在 while 迴圈內，計算地球和月球所受萬有引力時，把太陽所產生對地球和月球的萬有引力，分別加上去。
當你執行模擬時，一開始你會看不到地球和月球，這是因為太陽比較，或和太陽與地球間距離的比較，地球和月球非常的小，所以請以滑鼠調近視距，這是你會看到月球繞著地球作公轉。請觀察這個模擬一段時間，你會發現月球公轉的傾角會改變，數一下，要隔多少年，傾角才會變回原來的角度？這就是月球的進動。

附加 1 ：你要如何寫一段程式碼，讓電腦來自動計算這個進動周期？

附加 2：當地球在月球和太陽中間時，會有月蝕(月全蝕或月偏蝕)，當月球在地球和太陽中間時會有日蝕，你可以由上述程式碼再加個幾行程式，來判斷月蝕或日蝕發生的時機嗎？

附加 3 ：將地球、月球的半徑改回真實的大小，再加幾行程式碼，模擬一個衛星，在地球赤道的上方作運行。

附加 4 ：將 3 做一些更改(人造衛星初速)，模擬人造衛星，由地球飛向月球(不降落)，繞過月球，再飛回地球，你會發現，這個人造衛星初速和方向的設定非常重要。

附加 5：將火星加上去，找到一個適當的初速，可以讓地球上出發的太空船，可以抵達火星。