Chúng ta sẽ tiếp tục 20 LC WS … bắt đầu từ những căn bản của Python , điều chúng ta chưa làm dc với C …
Cuốn ByteOfPython của Swaroop này rất chính quy , sẽ trang bị cho chúng ta nhiều kiến thức để tiếp tục lập trình game …
Mình đã thay đổi tên biến , hàm để minh họa sinh động gần gũi với gamedev hơn , và bổ sung một số đoạn ngoài sách …
Bạn hãy lên trang chủ của Python là www.python.org để down bản cài Python 2.5


#----------

Số trong Python có 4 loại
integer (số nguyên) như 2 , 99 ,1000 …, (chúng ta sẽ tập trung lấy số nguyên (…int) làm ví dụ)
longinteger (lớn hơn số nguyên) ,
floating point (số thực) như 2.32 ; 52,3E-4 , nghĩa là 52.3*10 mũ trừ 4
complex number (số ảo) như -5 + 4j ...

#----------

Xâu trong Python có 3 dạng
Dùng ( ' ) và ( " ) đều giữ nguyên các kí tự chứa trong cặp dấu đó
' gamedev ' và "gamedev" là như nhau …
Nếu sử dụng ba lần dấu nháy ( ''' ) hoặc ( """ ) sẽ tính cả xuống dòng

*'''gamedev.vn*
*Một diễn đàn lớn về gamedev… '''*

*""" gamedev.vn*
*Một diễn đàn lớn về gamedev… """*

Nếu bạn cần sử dụng dấu nháy trong xâu , bạn sẽ phải đánh thếm dấu xổ \'
' what\'s ' sẽ thành : *what's*
Bạn muốn cách dòng như khi nhấn tab thì hãy dùng \t
' ' xuống dòng như khi nhấn Enter thì hãy dùng \n
Bản thân dấu xổ muốn hiển thị cần dùng \\
Raw string dc dùng khi bạn muốn bỏ qua hiệu lực của \ trong xâu , bạn hãy dùng tiền tố r hay R trước xâu bình thường để chuyển sang Raw string…

r'ko dung dau \\'

thì Python(…P) sẽ hiển thị :

*ko dung dau* \\

… Raw string thường dc dùng khi nhập dữ liệu

Nếu bạn muốn dùng Unicode thì thêm tiền tố u hoặc U …

Xâu dc tạo ra sẽ ko thể bị thay đổi , chúng ta sẽ nghiên cứu sau…

Xâu có thể dc nối vào nhau

'gamedev' '.vn'

sẽ hiển thị *gamdev.vn*

#----------

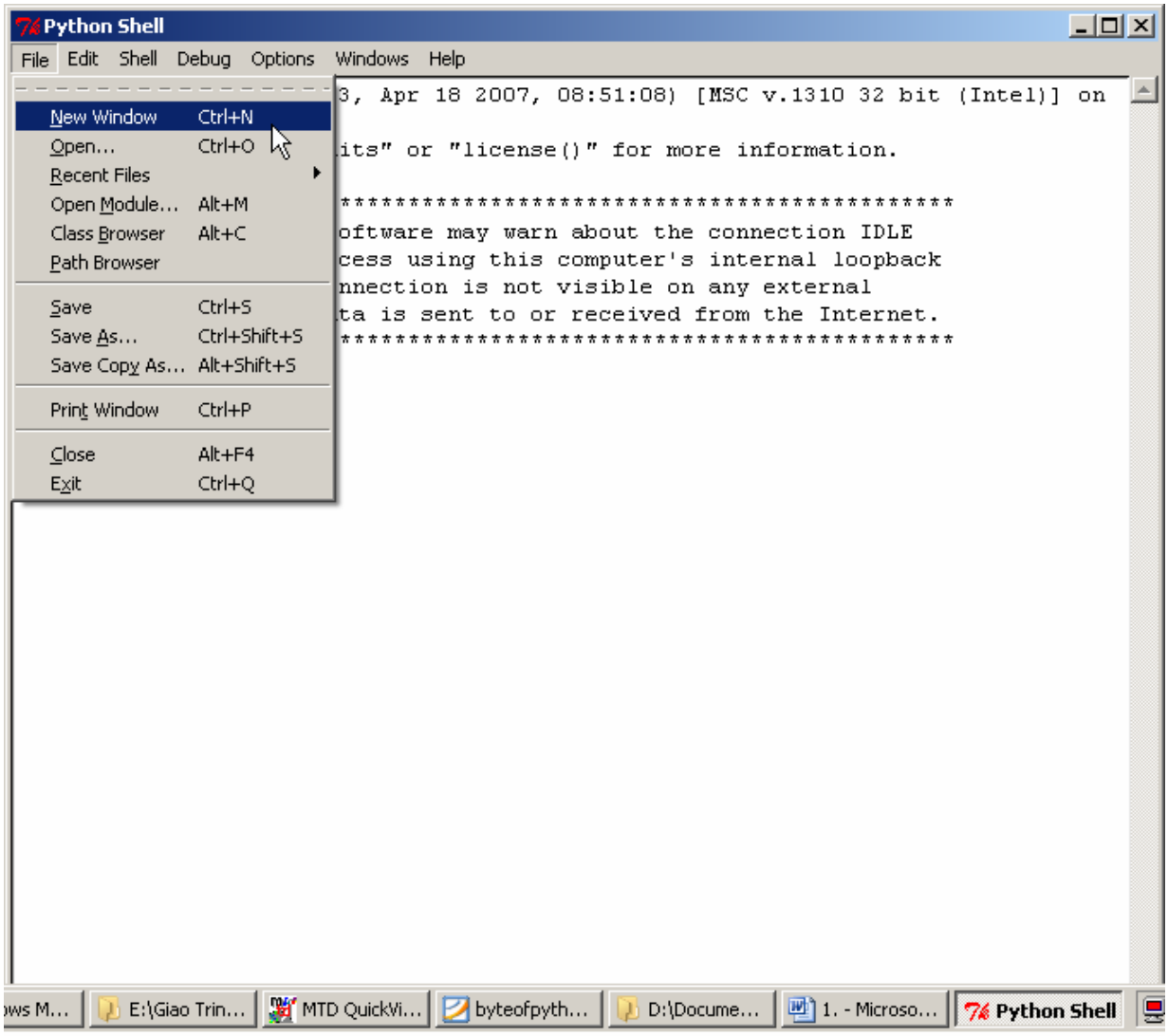
## Quy tắc đặt tên biến

-kí tự đầu tiên phải là chữ cái (hoa hoặc thường) hoặc dấu _
-phần còn lại có thể là chữ cái (hoa hoặc thường) , số , dấu _
-phân biệt chữ hoa và chữ thường

---

---

Bật IDLE lên

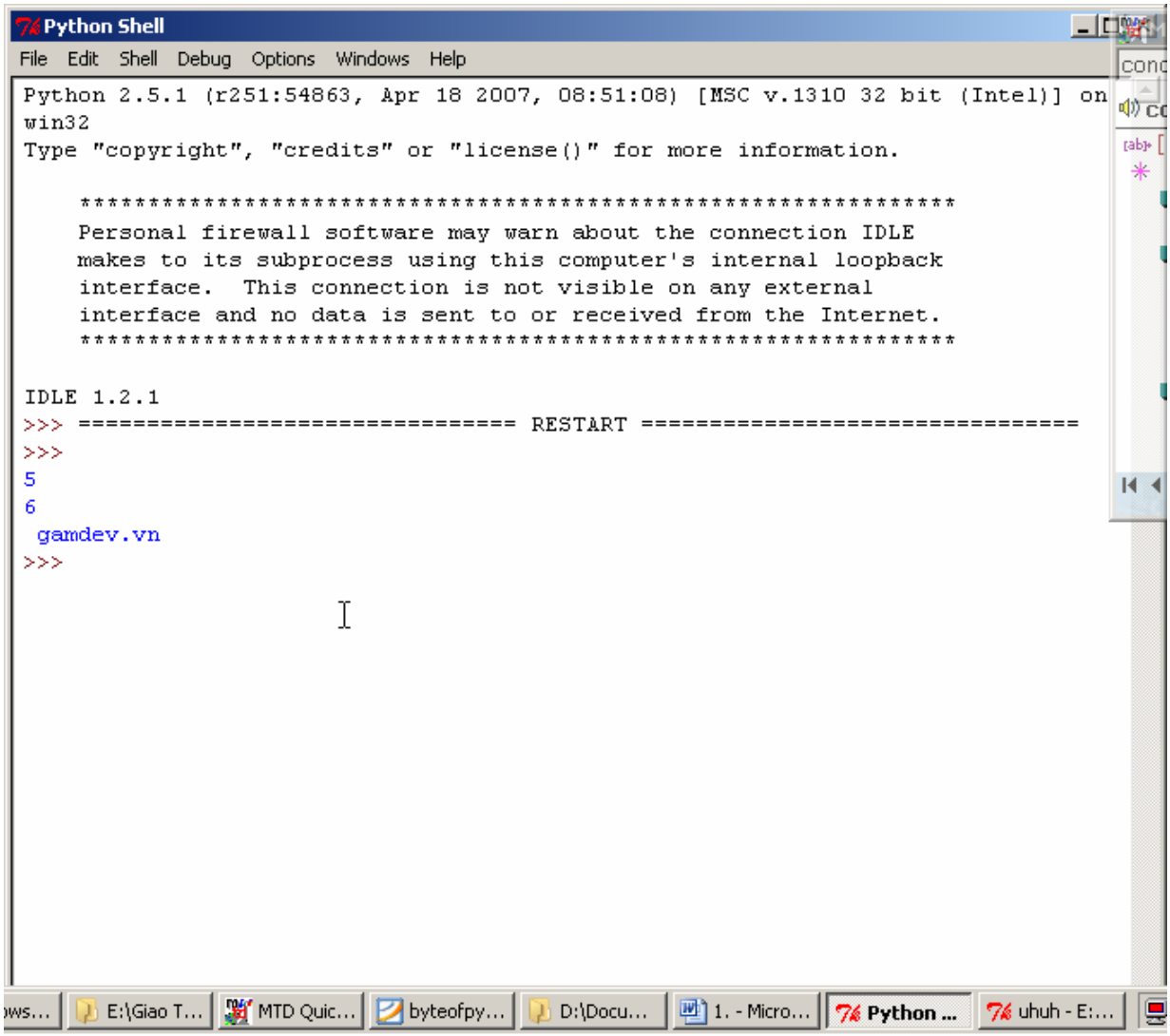



Hiện lên một cửa sổ và ct bắt đầu code

#P chú thích bằng dấu #

```
i =5        # gán i bằng 5
print i   # in i , print có nghĩa là in
i=i+1     # gán i bằng i cộng 1
print i   # in i
s = ' gamedev.vn '   # gán xâu gamdev.vn lưu trữ vào trong biến s
print s  # in s
```

Nhấn F5 để dịch , trong Python Shell sẽ hiển thị




Chúng ta ko cần phải khai báo biến từ đầu như C
Và cũng ko cần dấu ' ; ' để kết thúc câu lệnh
Để dễ đọc chúng ta nên viết mỗi dòng một câu lệnh
Nếu muốn viết nhiều câu lệnh trên một dòng thì dùng dấu ; như C

```
i =5 ; print i ;
```

thậm chí

```
i =5  ; print i
```

Nhưng ko nên làm vậy , HÃY VIẾT MỖI DÒNG MỘT CÂU LỆNH

Bạn có thể nối câu lệnh bằng dấu \

Như bạn viết

```
S = " Day la mot xau . \
Va day la mot xau dc noi tiep . "          # hãy để í dấu nháy kép
print S
```

Output (kết quả hiển thị , xuất ra) :

*Day la mot xau . Va day la mot xau dc noi tiep .*

Tương tự

```
print \
i
```

cũng tương đương với

```
print i
```

---

Bởi vì ko dấu ; đánh dấu khối lệnh nên P phải dùng indetation (vết khứa)

…các câu lệnh liên tiếp có cùng độ dài khoảng trống tính từ lề trái (kí tự space , dấu tab …) sẽ cùng nằm trong một khối lệnh …

Bây giờ bạn viết

```
i = 5
print " gia tri i la " , i
```

Output :

*gia tri i la 5*

Nhưng nếu bạn viết

```
i = 5
 print " gia tri i la " , i     # có thêm dấu cách
```

thì sẽ bị lỗi cú pháp ngay

#----------

## Toán tử :

+ : cộng
 - : trừ
* : nhân
** : mũ ( 3 **4 bằng 81)
/ : chia
(lưu í : số nguyên chia cho số nguyên sẽ ra số nguyên :  4/3=1
Nếu trong phép chia có thì ít nhất có một số là số thập phân mới cho ra kết quả là số thập phân…

4.0/3 hay
4/3.0 cho
1.33333333333333 )

// : chia lấy phần nguyên : 4 // 3.0 = 1.0 vì (3*1)+1=4 và 4 chia 3 bằng 1(phần nguyên) dư 1
%: chia lấy phần dư : 8%3 =2 vì 8 chia 3 bằng 2 dư 2
& : và
| : hoặc
!= : khác

Sau đây là thứ tự ưu tiên của các toán tử

| Operator |
|---|
| lambda |
| or |
| and |
| not x |
| in, not in |
| is, is not |
| <, <=, >, >=, !=, == |
| \| |
| ^ |
| & |
| <<, >> |
| +, - |
| *, /, % |
| +x, -x |
| ~x |
| ** |
| x.attribute |
| x[index] |
| x[index:index] |
| f(arguments ...) |
| (expressions, ...) |
| [expressions, ...] |
| {key:datum, ...} |
| `expressions, ...` |

Chúng ta sẽ chỉ tạm thời tham khảo bảng trên …

Nếu bạn đã học Toán chắc chắn phải biết trong biểu thức 2 + 3*5 thì chắc chắn phép tính nhân phải được thực hiện trước phép cộng , các toán tử cũng tương tự vậy , nó cũng có những thứ tự đã dc quy định , theo bảng này , toán tử ở dưới thấp hơn sẽ dc ưu tiên…

Bạn nên sử dụng cặp dấu ngoặc () để nhóm các biểu thức riêng rẽ với nhau như (2 + (3*5))…

#---

Các câu lệnh điều khiển

## Câu lệnh if (câu lệnh rẽ nhánh)

Mời bạn code các dòng lệnh dưới

```
number = 23

guess = int(raw_input("Nhap mot so nguyen:"))  # 'guess' nghĩa là đoán

if guess == number:             # chú í dấu hai chấm
        print 'Chuc mung ban da doan dung.'  # một khối lệnh mới bắt đầu khi nhấn tab
        print "(nhung ban se chang dc tang mot gia thuong nao ca )"

# kết thúc khối lệnh khi
# xuống dòng và lùi về đầu dòng

elif guess<number:

        print 'Khong, lon hon mot chut'  # một khối lệnh khác

else:
        print 'Khong, nho hon mot chut'

print 'OK!'  # câu lệnh này thuộc khối lệnh khác
```

Đây là một chương trình đoán số có sử dụng vòng if lồng nhau …

Chúng ta(..ct) cần thao tác với dữ liệu mà người dùng nhập vào nên chúng ta đã dùng hàm raw_input( )  . Trong dấu () là lời mời của chúng ta đối với người dùng để họ nhập số liệu , sau đó tất cả những gì người dùng nhập vào sẽ dc lưu vào bộ nhớ để xử lí , kết thúc việc nhập liệu bằng cách người dùng nhấn Enter
(ct sẽ nghiên cứu thêm sau)

Dữ liệu mà người dùng sẽ được thao tác ép kiểu sang dạng số nguyên (…int) để tiện so sánh, nên chúng ta dùng tiền tố int() …

Các bạn để ý rằng bắt đầu vòng **if** lồng nhau vẫn là **if** , các trường hợp tiếp theo là **elif** , kết thúc là **else**…
Để kết thúc vòng lặp phải xuống dòng lùi vào lề trái như cũ …

#---

## Vòng lặp while

Mời bạn code đoạn mã sau

```
number = 23

running=True      # gán running là đúng

while running:      # khi nào giá trị running vẫn đúng thì vòng lặp vẫn chạy

        guess=int(raw_input('Moi nhap so nguyen :'))
        if guess==number:
                print' Ban da doan dung'
                running=False      # khiến vòng lặp ngừng lại , running đã sai
        elif guess<number:
                print'Khong , lon hon mot chut.'
        else:                                   # chắc chắn là số này phải lớn hơn 23 vì
      chúng ta
                                                # đã loại đi hai trường hợp trên
                print'Khong , nho hon mot chut.'
else:                   # Else này của vòng lặp while chứ ko phải của if , khi kết thúc
vòng While
                # sẽ tự động thực hiện dòng lệnh này
        print 'Vong lap ket thuc.'

print'OK!'
```

Chúng ta đã bàn nhiều về vòng lặp While trong 20 WS LC nên chắc chắn sẽ ko có khó khăn gì cho bạn

## … dãy Phibônaxi…

```
a, b = 0, 1     # đây là phép gán đồng thời liên tiếp multiple assignment
# theo chiều từ trái qua phải , tương đương với a=0, b=1
while b < 10:   # khi nào b vẫn còn nhỏ hơn 10
    print b
    a, b = b, a+b # đây cũng là phép multiple assignment , nhưng phức tạp
# hơn…
```

KQ:

*1*
*1*
*2*
*3*
*5*
*8*
# dãy này có tính chất là một số bất kì nào cũng là tổng của hai số trước nó
# ví dụ 8=3+5
# nó bắt nguồn từ bài toán sinh sôi của một đàn thỏ trên một hòn đảo …
# trong thiết kế mĩ thuật , từ dãy số này chúng ta có thể tìm ra tỉ số vàng
# bằng cách lấy số trước chia cho số sau , dc khoảng 1,62 , đây cũng
# thường là tỉ lệ hợp lí của những vật hình chữ nhật như cửa sổ …

Đầu tiên , a =0 , b =1
Bước vào vòng lặp , b < 10 đúng nên tiếp tục in b =1
In b xong , gán a = b = 1 , rồi b=a+b = 0+1

#các bạn lưu í đây là phép gán đồng thời, nên trong b=a+b a vẫn chưa mang
#giá trị a=b=1, mà vẫn mang gía trị  ban đầu 0, sau khi kết thúc lệnh gán
#liên tiếp đồng thời này thì a mới mới bắt đầu mang giá trị a =b= 1 )

Kết thúc vòng lặp đầu tiên a=1 , b =1

Sang vòng lặp thứ hai
Kiểm tra thấy b vẫn nhỏ hơn 10 , tiếp tục in b =1
Gán a=b=1 , và b = a +b =1+1+2

Kết thúc vòng lặp hai ,a=1 , b=2

Sang vòng lặp 3 …
Gán a=b=2 , b = a+b=1+2=3
# lưu í đây là phép gán đồng thời , việc gán a=b
# và a=a+b là xảy ra cùng một lúc , ko dc nhầm là b=a+b=2+2

Cứ thế cho đến khi b<10 sai thì kết thúc vòng lặp …

#---

## Vòng lặp for

Trước hết chúng ta cần biết hàm range(1,5) sẽ cho ta một dãy số [1,2,3,4]
 # range có nghĩa là phạm vi

Nếu thêm tham số thứ ba , nó sẽ đóng vai trò bước nhảy trong hàm range
range(1,9,2) sẽ là [1, 3, 5, 7]

Ở đây hàm range có vai trò tính toán ra một dãy giá trị cho vòng lặp for dựa vào đó để hoạt động

Mới bạn code

```
for i in range(2,5):                    # 2 , 3, 4
        print i
else:
        print'Vong lap for ket thuc'
```

Output:

*2*
*3*
*4*
*Vong lap for ket thuc*

Ở đây , trong vòng lặp đầu tiên , theo thứ tự của dãy giá trị, i sẽ dc gán bằng 2 , sau đó đến lệnh in i ra , kết thúc vòng lặp đầu tiên
Tiếp tục sang vòng lặp thứ hai ,theo thứ tự của dãy giá trị ,i dc gán bằng 3 và in , kết thúc vòng lặp hai…
Sang vòng lặp ba , gán và in i =4 …
… đến dòng else của vòng lặp for ….

#---

## Câu lệnh break

# break có nghĩa là bẻ gẫy , ở đây nó sẽ mang nghĩa bóng là bẻ gẫy vòng lặp
#khi xuất hiện break , vòng lặp sẽ chấm dứt , hay bỏ qua những câu lệnh kế tiếp của khối lệnh

Mời bạn code

```
while True:         # vòng lặp vô hạn
        s = raw_input('Nhap chuoi ki tu')       # chúng ta ko dùng tiền tố int để ép kiểu về số
                                                # nguyên vì đơn giản là ko cần thiết
        if s== 'quit':      #  kiểm tra nếu người dùng muốn thoát , họ sẽ đánh chữ quit
                break       # chấm dứt vòng lặp vô hạn tại đây, ko đo độ dài xâu nữa, lưu í lề

        print"Do dai cua xau ki tu s la ", len(s)  # đo độ dài của xâu s , rồi ghi kết quả ngay sau xâu kí tự 'do dai …'


print"OK"
```

Đây là một chương trình đo độ dài xâu kí tự bạn vừa nhập vào , len(tên xâu kí tự cần đo) sẽ trả về số kí tự của xâu
# len là viết tắt của length – độ dài

#----------

## Câu lệnh continue (tiếp tục)

# Continue chỉ bỏ qua các câu lệnh tiếp theo của khối lệnh tính từ vị trí đặt continue , chứ ko thoát hẳn khỏi vòng lặp

Mời bạn code

```
while True:
        s=raw_input('Nhap xau ki tu co do dai lon hon 3 :')
        if s== 'quit':
                break
        if len(s)<=3:
                continue     # bỏ qua ko đếm kí tự nữa , quay về đầu khối lệnh
        else:
                print"Do dai cua xau ki tu s la ",len(s)   # lưu í dấu ' , '
```

Mời bạn tự kiểm tra kết quả …

#----------

## Hàm

Hàm là một đoạn mã có thể dc sử dụng đi sử dụng lại nhiều lần…

Chúng ta định nghĩa hàm bằng từ khóa def  # define – định nghĩa

```
def noiHello() :          # bắt đầu khai báo hàm noiHello
        print' HelloWorld! '

#đây là câu lệnh sẽ dc thực thi khi gọi hàm
# kết thúc khai báo hàm khi xuống dòng và lùi vào lề

print" Ham can loi goi ham de thuc thi "  # hàm cần một lời gọi hàm để thực thi

noiHello()   # đây là lời gọi hàm
```

out put :

*Ham can loi goi ham de thuc thi*
*HelloWorld!*

Cặp ngoặc đơn là nơi chứa tham số của hàm…
Hàm đơn giản trên ko cần truyền tham số …

Mời bạn code

```
def in_gia_tri_max(a,b):   # xuất hiện thêm hai tham số a và b
```

```
        if a>b:                 # nếu a lớn hơn sẽ in ra a
                print a,'la so lon hon'
        else:                   # nếu b lớn hơn sẽ in b
                print b,'la so lon hon'
# kết thúc phần khai báo hàm

in_gia_tri_max(3,4)

# gọi hàm đồng thời truyền tham số cho hàm theo thứ tự như khi ta khai
# báo hàm , tức là a tương ứng 3 , b tương ứng với b

#tiếp tục một ví dụ khác, chúng ta sẽ gán giá trị cho hai biến x, y

x=5
y=7
in_gia_tri_max(x,y)

# chúng ta cũng vừa gọi hàm lần thứ hai đồng thời truyền tham số cho hàm
# x tương ứng với a , y tương ứng với b
```

OutPut

*4 la so lon hon*
*7 la so lon hon*

Ở đây người ta gọi a , b là tham số - parameter
Các giá trị x,y và 3,4 được gọi là đối số - argument
Giá trị đối số sẽ dc gán cho tham số , các tham số sẽ dc thao tác trong hàm và trả về kết quả tính toán cho chúng ta …

---

## Biến cục bộ ( local variable)

Những biến này chỉ có hiệu lực trong phạm vi của hàm ,

Trong một hàm , bạn đặt tên một biến là a…
Trong hàm khác (hoặc vị trí nào đó trong file), bạn cũng lại đặt thêm một biến cũng tên là a , cũng ko sao , chẳng ảnh hưởng gì cả , vì chúng hoàn toàn ko hề liên quan tới nhau …

```
def in_gia_tri(x) :   # khai báo hàm để in giá trị x
        print 'x la',x       # x bây giờ là biến cục bộ , chỉ có hiệu lực trong hàm
in_gia_tri
        x=2    # đổi x thành 2
        print 'bien cuc bo x trong ham in_gia_tri da dc thay bang' , x
 # kết thúc khai báo hàm

x =50  # khai báo biến x ở ngoài hàm in_gia_tri
in_gia_tri(x)
```

```
print 'x o ngoai van giu nguyen gia tri nhu truoc khi dc truyen vao ham' , x
# x ở ngoài vẫn giữ nguyên giá trị như trước khi gọi hàm , nó ko bị đổi thành 2
```

OutPut

*x la 50*
*bien cuc bo x trong ham in_gia_tri da dc thay bang 2*
*x o ngoai van giu nguyen gia tri nhu truoc khi dc truyen vao ham 50*

---

## Global Statement (khai báo toàn cục)

Nếu chúng ta muốn biến của ct có hiệu lực ở bất kì mọi nơi (như ở trong bất kì hàm nào…)
Ct sẽ cần dùng khai báo global trước tên biến đó

Mời bạn code

```
def in_gia_tri()  :          # hàm nay ko cần truyền tham số
            global x       # khai báo biến toàn cục x
            print 'x la' , x
            x=2    # thay x bằng 2
            print 'Bien toan cuc x da bi thay bang ',x
# kết thúc khai báo hàm
x=raw_input('Moi ban nhap x :')   # nhập x , lưu í x này ko nằm trong hàm

in_gia_tri()  # gọi hàm
print 'Gia tri x bay gio la' , x
```

Output

*Moi ban nhap x :3*
*x la 3*
*Bien toan cuc x da bi thay bang  2*
*Gia tri x bay gio la 2*

---

## Default Argument Value  (đối số mặc định)

Đôi khi bạn muốn tham số của bạn mang sẵn một giá trị hằng số , đề phòng người dùng quên ko nhập vào , hãy sử dụng dấu bằng như sau

```
def in_xau_ki_tu (xau_ki_tu , so_lan=1):  # tham số có thể là một xâu kí tự
            print xau_ki_tu*so_lan

in_xau_ki_tu ('VietNam')   # bạn ko nhập đối số cho tham số so_lan ,so_lan tự động
=1
in_xau_ki_tu ('Gamedev.vn',5)
```

OutPut

*VietNam*
*Gamedev.vnGamedev.vnGamedev.vnGamedev.vnGamedev.vn*

Các bạn cần nhớ là tham số đầu tiên ko dc mang giá trị đối số mặc định

---

## Keyword Argument

Bạn đã biết cách truyền đối số cho tham số theo trật tự vị trí tương ứng , nhưng có một cách giúp bạn có thể truyền đối số cho tham số mà ko cần theo trật tự tham số như khi khai báo hàm

```
def in_gia_tri(a,b) :
        print 'so dau tien là a =', a , "so thu hai la b =", b
in_gia_tri(b=20,a=10)  # lưu í dấu bằng và thứ tự
```

OutPut

*so dau tien là a = 10 so thu hai la b = 20*

---

## Return Statement

Câu lệnh return sẽ giúp bạn trả về một giá trị tính toán sau khi gọi hàm

```
def so_sanh(a,b):
        if a>b :
                return a   # trả về giá trị là tham số a nếu a lớn hơn b
        else:
                return b   # trả về giá trị là tham số b nếu a ko lớn hơn b

print so_sanh(10,20)   # lệnh print sẽ in giá trị trả về của hàm so_sanh
```

Nếu bạn gán
x= so_sanh(10,20) thì x sẽ mang giá trị *20* ngay lập tức
Hàm ko chỉ có thể trả về giá trị bình thường mà còn có thể trả về giá trị của một biểu thức

```
def tong (a,b):
        return a+b
x= tong(1,2)
print x
```

None nghĩa là ko có gì (tương tự như null trong C)
Nếu khi viết hàm mà chúng ta quên ko dùng lệnh return thì P mặc định hàm có dòng return None

```
def mot_ham_rong () :   # khai báo một hàm rỗng
        pass            # trong P , pass dùng để chỉ một khối lệnh rỗng
```

```
# trong trường hợp này pass có tác dụng cho phép
# chúng ta bỏ qua ko cần thực thi một lệnh nào để
# tiếp tục thực hiện các câu lệnh tiếp theo
# pass chỉ nhằm mục đích ko gây ra lỗi cú pháp ,hàm cần ít nhất có một câu lệnh

print mot_ham_rong()
```

OutPut

*None*

---

## DocString (documentation – tài liệu)

Nó sẽ giúp bạn chú thích cách sử dụng hàm …

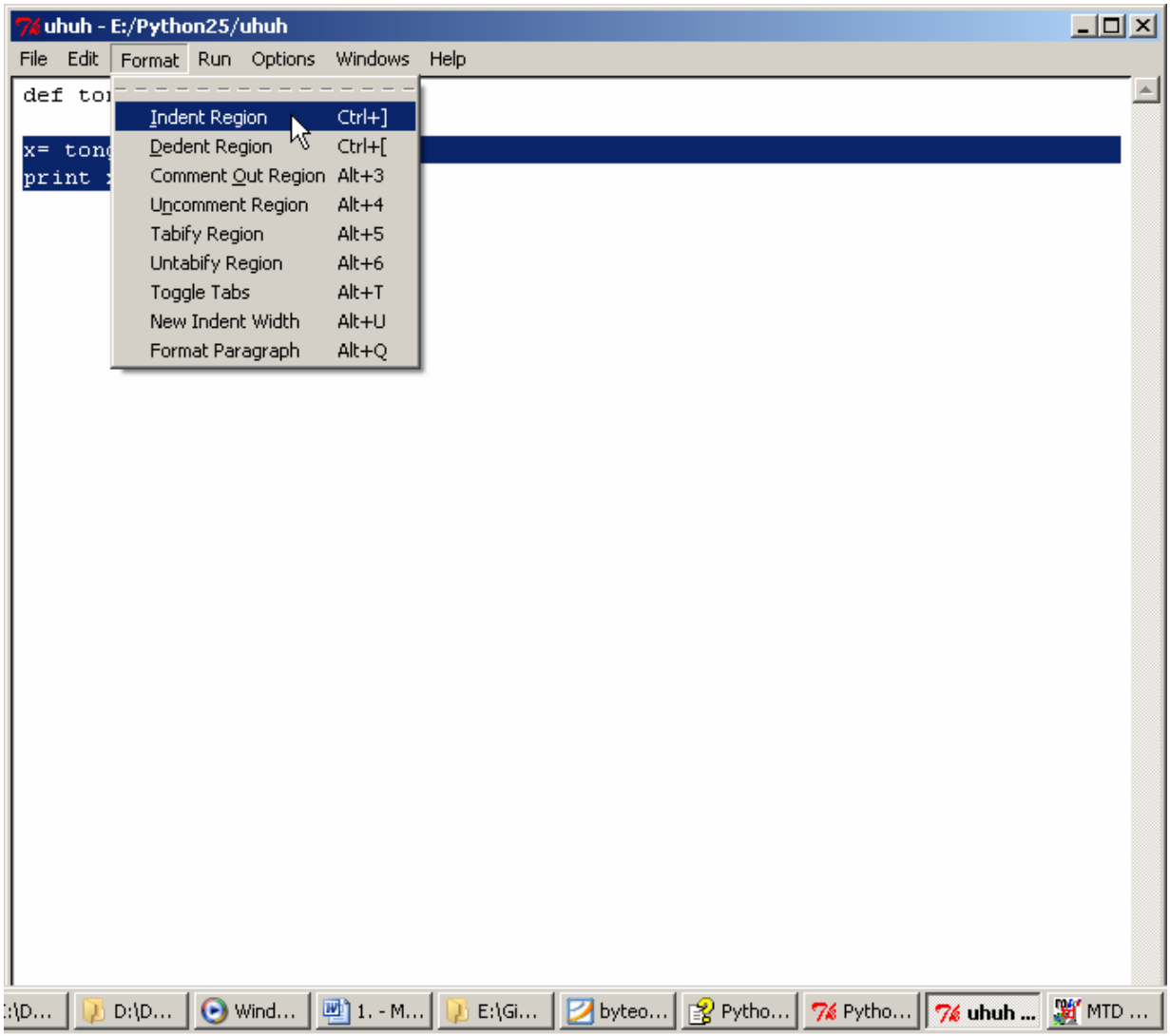
```
def so_sanh_so_nguyen(x,y) :
        ''' in ra gia tri lon nhat sau khi so sanh hai doi so.
        Neu doi so ko phai la so nguyen , tu dong chuyen ve so nguyen'''

        x=int(x)  # ép kiểu về số nguyên , nếu là kí tự hoặc số thực…
        y=int(y)
        if x>y :
                print x,'la so lon hon'
        else :
                print y,'la so lon hon'
        return x,y        # trả về giá trị x,y để kiểm tra
print so_sanh_so_nguyen(5.4,9.9)   # hai đối số ko phải là số nguyên
print so_sanh_so_nguyen.__doc__    # tên hàm.__doc__
```

OutPut:

*9 la so lon hon*
*(5, 9)*
*in ra gia tri lon nhat sau khi so sanh hai doi so.*
*Neu doi so ko phai la so nguyen , tu dong chuyen ve so nguyen*

Bạn có thể xem doc qua câu lệnh help(tên_hàm) # thoát bằng cách nhấn q

Tip: hãy để tự động lùi lề khi xuống dòng




#----------

# Module

Bạn đã có thể tái sử dụng một đoạn mã trong một file bằng cách sử dụng hàm , nhưng nếu bạn còn muốn sử dụng lại chúng trong nhiều file khác … Giải pháp chính là module …
Nó đơn giản chỉ là một file có đuôi là .py và chứa trong nó các hàm và biến mà bạn cần tái sử dụng nhiều lần …

---

Chúng ta bắt đầu bằng việc tạo một module

Mới bạn code rồi save thành file bat_dau.py vào thư mục cài đặt Python

```
def loi_chao():
        print 'Xin chao, module bat_dau xin gui loi chao den ban'
version='0.1'   # đây là biến chứa xâu kí tự chỉ phiên bản
```

Bạn đã tạo một module…

Code tiếp một file khác đặt tên là su_dung_module.py

```
import bat_dau   # báo cho P truy tìm module bat_dau để cài đặt

bat_dau.loi_chao()  # sử dụng hàm loi_chao trong module bat_đau

print 'Day la phien ban ', bat_dau.version  # sử dụng biến version trong module bat_dau
```

Nếu bạn ko muốn phải đánh cả chữ bat_dau.  thì hãy sử dụng

from (tên module) import (tên hàm hoặc biến)

Mời bạn code

```
from bat_dau import loi_chao,version

loi_chao()
print 'Day la phien ban ', version # bạn đã có thể sử dụng trực tiếp các hàm và biến trong module
```

---

Nếu bạn muốn kiểm tra xem trong module có nội dung gì hãy code

```
dir(tên hàm)

dir(bat_dau)
```

OutPut

*['__builtins__', '__doc__', '__file__', '__name__', 'loi_chao', 'version']*

```
#__name__ chứa tên của module

print bat_dau.__name__
```

OutPut

*'bat_dau'*

---

Nếu bạn muốn biết có thể đặt module ở đâu để P có thể tìm thấy , hãy code

```
import sys  # system – hệ thống
```

```
print sys.path      # path là đường dẫn

#----------------------------------------------------------------------
```

## DataStructure (cấu trúc dữ liệu) (…dts)

---

### List (danh sách)

Mời bạn code :

```
# đây là danh sách các đồ đạc trên người của tôi
do_dac= ['mu','giap','khien','giay']  # khai báo đồ_đạc là một list - danh sách, lưu ý dấu [ ]
print 'Toi dang co', len(do_dac), 'mon do tren nguoi.'

print 'Chung la'
for mon_do in do_dac :   # vòng lặp for sẽ  tuần tự tiến theo thứ tự các giá trị của list mon_do
        print mon_do,   # lưu ý dấu phẩy

print '\nToi vua cam them mot thanh kiem'# '\n' để xuống dòng

do_dac.append('kiem')  # append là nối thêm

print 'Cac mon do tren nguoi toi bay gio la',do_dac

print 'Toi se sap xep lai no'
do_dac.sort()      # sort là sắp xếp

print 'Sau khi sap xep , cac mon do tren nguoi toi bay gio la',do_dac

print 'Mon do thu nhat la', do_dac[0]   # P bắt đầu đếm từ 0
print 'Mon do thu nam la',do_dac[4]
mon_do_bi_vut=do_dac[0]
del do_dac[0]   #  delete xóa do_dac[0] ra khỏi list
print 'Toi da vut mon do ', mon_do_bi_vut
print 'Cac mon do tren nguoi toi gio la', do_dac
```

OutPut:

```
Toi dang co 4 mon do tren nguoi.
Chung la
mu giap khien giay
Toi vua cam them mot thanh kiem
Cac mon do tren nguoi toi bay gio la ['mu', 'giap', 'khien', 'giay', 'kiem']
Toi se sap xep lai no
Sau khi sap xep , cac mon do tren nguoi toi bay gio la ['giap', 'giay', 'khien', 'kiem', 'mu']
```

*Mon do dau tien la giap*
*Mon do cuoi cung la mu*
*Toi da vut mon do  giap*
*Cac mon do tren nguoi toi gio la ['giay', 'khien', 'kiem', 'mu']*

Như bạn thấy list có khả năng mutable (có thể biến đổi)
Và bạn cũng thấy list có ba dot method là sort , append ,del cho phép bạn chỉnh sửa lại list

Chúng ta vẫn tạm gọi những thủ tục này là dot method từ 20 WS LC

---

## Tuple

Nó cũng tương tự như List , nhưng nó ko thể bị chỉnh sửa …

```
# các bạn lưu ý dấu ngoặc đơn ()

do_dac= ('mu','giap','khien')
print 'Toi dang co so mon do la', len(do_dac)

do_dac_sau_khi_mua_them = ('kiem', 'giay', do_dac)

print 'So do dac cua toi bay gio la ', len (do_dac_sau_khi_mua_them )
print 'Chung bao gom' , do_dac_sau_khi_mua_them
print 'So do dac cua toi truoc day la' , do_dac_sau_khi_mua_them [2]
print 'Mon do cuoi cung trong so mon do cu cua toi la',
do_dac_sau_khi_mua_them [2][2]
```

OutPut

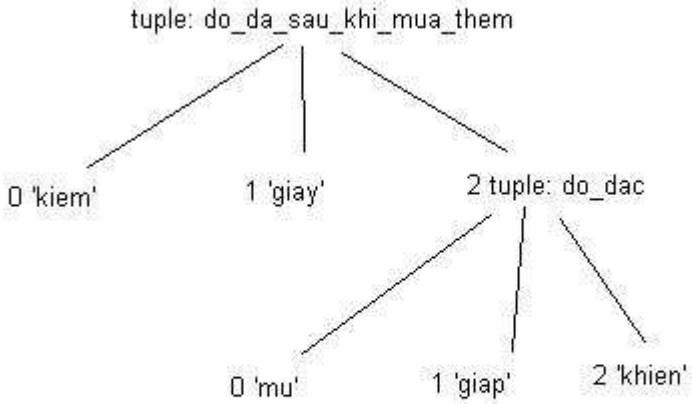
*Toi dang co so mon do la 3*
*So do dac cua toi bay gio la  3*
*Chung bao gom ('kiem', 'giay', ('mu', 'giap', 'khien'))*
*So do dac cua toi truoc day la ('mu', 'giap', 'khien')*
*Mon do cuoi cung trong so mon do cu cua toi la khien*

do_dac là một ví dụ về tuple …
Hàm len() cũng có thể đo dc độ dài của tuple , chứng tỏ tuple cũng là một chuỗi các phần tử có trật tự nhất định như list …

Chúng ta đã chuyển các món đồ từ tuple do_dac sang tuple do_dac_sau_khi_mua_them **…**
Nhưng ko giống list , tuple cũ dc chuyển sang tuple mới vẫn giữ nguyên cấu trúc của nó …

Vì thế chúng ta cần phải truy nhập vào các phần tử của tuple do_dac (bây giờ đã nằm trong lòng tuple do_dac_sau_khi_mua_them, nó đóng vai trò là phần tử thứ ba của tuple này) bằng toán tử [ ] ( gọi là toán tử index – index nghĩa là danh sách)

Trong ví dụ của chúng ta , do_dac_sau_khi_mua_them[2][2] chính là cách truy cập vào phần tử thứ ba của tuple do_dac_sau_khi_mua_them (chính là tuple do_dac) , sau đó tiếp tục truy cập vào phần tử thứ ba của tuple do_dac chính là khien …




(ND: xin lỗi , mình vẽ thiếu chữ c trong chữ đồ đạc sau khi mua thêm
Giống như C , P cũng bắt đầu đếm từ số 0… )

Nếu bạn muốn khai báo một tuple rỗng

```
Empty_tuple=()
```

Nhưng khi khai báo một tuple chỉ chứa một phần tử bạn lại cần thêm một dấu , sau phần tử duy nhất đó

```
Single_tuple=(gamedev , )
```

## Tuple và câu lệnh print

```
# bạn hãy chú í dấu % trong đoạn code sau

Tuoi = 18
Ten = 'eddyfosman'
print ' %s hien nay dang %d tuoi ' % (Ten,Tuoi)
print 'Tai sao %s lai nghien cuu Python ?' % Ten
```

OutPut

*eddyfosman hien nay dang 18 tuoi*
*Tai sao eddyfosman lai nghien cuu Python ?*

Dấu % dùng để định dạng cho output , như %s là định dạng cho string , %d cho integer …

Chúng ta đã từng dùng dấu , để thực hiện điều tương tự như đoạn code trên đã làm , nhưng đôi khi nó sẽ trở nên ko hiệu quả …

Các bạn để í thấy chúng ta đã sử dụng tuple (Ten,Tuoi) trên dòng print đầu tiên …

Tuân theo trật tự từ trái sang phải , %s sẽ dc thay bằng phần tử đầu tiên của tuple là Ten (rồi tự động ép kiểu về xâu) , %d sẽ dc thay bằng phần tử thứ hai của tuple là Tuoi (đồng thời cũng tự động chuyển về kiểu nguyên)

Việc định dạng này sẽ giúp cho chương trình của chúng ta tránh dc nhiều lỗi đáng tiếc như ko khớp kiểu dữ liệu …

Câu lệnh print thứ hai ko sử dụng tuple , đơn giản chỉ là định dạng cho xâu kí tự …

---

## Dictionary (từ điển)

Nó chính xác giống như một quyển từ điển … chúng ta sẽ tra tên của từ và tìm thấy nghĩa của chúng

Bạn sẽ kết hợp hai giá trị key (name-tên) và value (detail-đặc điểm chi tiết)
Bạn chỉ có thể dùng những đối tượng bất biến (immutable object) ko thể chỉnh sửa dc như xâu kí tự cho key , còn value thì có thể nhận cả đối tượng chỉnh sửa dc…..

Bạn sẽ khai báo dictionary bằng mẫu:

**Dictionary= { key1:value1 , key2:value2 …}**

Bạn sẽ ko thể tìm thấy thông tin chính xác nếu có hai key có tên trùng nhau

```
# lưu í dấu {} và dấu :
do_dac = { 'mu' : ' tang 10 % tri luc' ,          # xâu 'mu' đóng vai trò key ,
                                                  #'tang 10…' đóng vai trò value
        'giap' : 'tang 10 % phong thu' ,          # bạn nên xuống dòng để dễ quan sát
        'khien' : ' ki nang block don tan cong cua doi phuong',
        'kiem' : 'tang sat thuong vat li 20 %' ,
        'giay': "tang do nhanh nhen 15 %" }     # kết thúc khai báo dictionary

print 'cac mon do cua toi la %s' %[do_dac]

# bây giờ ct thêm thêm một món đồ nữa
do_dac ['nhan'] = 'tang khang phep 25 %'

# xóa một món đồ
del do_dac['khien']

print '\nTren nguoi toi co %d mon do '  %len(do_dac)

for mon_do,cong_dung in do_dac.items():
# chúng ta sẽ bàn về những dòng code ở dưới sau
```

```
        print 'Mon do %s co cong dung %s' % (mon_do,cong_dung)

if 'nhan' in do_dac:   # hoặc dùng dùng thủ tục do_dac.has_key('nhan')
        print "\nNhan co cong dung %s" % do_dac['nhan']
```

OutPut

*cac mon do cua toi la [{'mu': ' tang 10 % tri luc', 'khien': ' ki nang block don tan cong cua doi phuong', 'giap': 'tang 10 % phong thu', 'giay': 'tang do nhanh nhen 15 %', 'kiem': 'tang sat thuong vat li 20 %'}]*

*Tren nguoi toi co 5 mon do*
*Mon do giay co cong dung tang do nhanh nhen 15 %*
*Mon do giap co cong dung tang 10 % phong thu*
*Mon do mu co cong dung  tang 10 % tri luc*
*Mon do kiem co cong dung tang sat thuong vat li 20 %*
*Mon do nhan co cong dung tang khang phep 25 %*

*Nhan co cong dung tang khang phep 25 %*

Chúng ta vừa tạo một dictionary(…dict) tên là do_dac …

Giống như tuple và list , chúng ta có thể truy nhập vào các phần tử của dict bằng toán tử index … hoặc chỉnh sửa dict bằng các toán tử như del…

Chúng ta có thể cùng một lúc truy cập một cặp giá trị key – value bằng cách sử dụng thủ tục items của dict (trong ví dụ của chúng ta là do_dac.items ). Thủ tục này trả về một list các tuple , mỗi tuple lại là một cặp giá trị key- value (theo trật tự của dict). Trong vòng lặp for , chúng ta sử dụng phép gán đồng thời để gán key cho biến mon_do , value cho biến cong_dung , rồi tiếp tục in tuple (mon_do,cong_dung) trong mỗi vòng lặp…

Chúng ta có thể kiểm tra xem có tồn tại một key trong dict ko bằng toán tử in , theo mẫu
tên key in tên dict          # (in-trong)

---

## Sequence (chuỗi trình tự) (..seq)

List , tuple , string là những ví dụ của seq …
Hai đặc điểm nổi bật của seq là
 + có thể truy nhập vào từng phần tử của seq bằng toán tử index []
 + có thể dùng toán tử slice [tham số 1: tham số 2]  để tách seq   (slice- cắt thành từng lát..)
Tham số 1 để xác định vị trí bắt đầu slice , tham số 2 xác định ví trí kết thúc slice (nhưng P sẽ dừng slice lại trước khi đến dc vị trí kết thúc này)

```
mon_do=['mu', 'khien','giap']
print 'mon do thu 0  la ' ,mon_do[0]
```

```
print 'mon do thu 1  la ' ,mon_do[1]
print 'mon do thu 2  la ' ,mon_do[2]
print 'mon do thu -1 la ' ,mon_do[-1]  # các bạn lưu í dấu âm cũng dc [] chấp nhận
print 'mon do thu  -2 la ' ,mon_do[-2]

# dùng toán tử slice [ : ] với list
print 'cac mon do tu 0 den 2  la ' ,mon_do[0:2]     # hãy để í đến lời chú thích dc in và
dấu :
print 'cac mon do tu 1  den cuoi cung ' ,mon_do[1:]
print 'cac mon do tu 1 den -1  la ' ,mon_do[1:-1]
print 'cac mon do tu dau den cuoi la' , mon_do[:]  # slice toàn bộ các phần tử

# dùng toán tử slice [ : ] với xâu
dien_dan = 'gamedev.vn'
print 'ki tu 1 den 4 la ' , dien_dan[1:4]
print 'ki tu 2 den het la ' , dien_dan[2:]
print 'ki tu 1 den -1 la ' , dien_dan[1:-1]
print 'ki tu dau den het la ' , dien_dan[:]
```

OutPut

```
mon do thu 0  la  mu
mon do thu 1  la  khien
mon do thu 2  la  giap
mon do thu -1 la  giap
mon do thu  -2 la  khien
cac mon do tu 0 den 2  la  ['mu', 'khien']
cac mon do tu 1  den cuoi cung  ['khien', 'giap']
cac mon do tu 1 den -1  la  ['khien']
cac mon do tu dau den cuoi la ['mu', 'khien', 'giap']
ki tu 1 den 4 la  ame
ki tu 2 den het la  medev.vn
ki tu 1 den -1 la  amedev.v
ki tu dau den het la  gamedev.vn
```

Chúng ta dễ dàng nhận thấy trật tự vị trí của list mon_do

```
   mu   khien   giap
    0      1       2
   -3     -2      -1
```

Với toán tử slice , các con số có thể ko cần nhập vào , nhưng bắt buộc phải có dấu :
Với xâu kí tự , ta cũng dễ phát hiện ra

```
  g a m e de v  .  v  n
  0 1 2 3 4 5 6 7  8  9
-10 9 8 7 6 5 4 3  2  1
```

Tiếp tục thử nghiệm nếu bạn muốn , tương tự với tuple…

---

## Reference (tham chiếu)

Khi bạn tạo một đối tượng như list , tuple , … rồi gán nó cho một biến nào đó (chúng ta vẫn làm như vậy trong các phần trước) , biến đó chỉ "chỉ dẫn" (refer) đến đối tượng chứ ko hề đại diện (-represent) cho bản thân đối tượng. Biến " tên " của đối tượng này chỉ dẫn đến một vùng bộ nhớ trong máy tính nơi mà đối tượng đó dc lưu trữ (dưới dạng số nhị phân …). Người ta gọi đó là " binding of the name to the object "

Chúng ta nghiên cứu đoạn code sau

```
hom_do =['mu','khien','giap']   # các bạn nên quen dần rằng hom_do chỉ là một cái tên
# chỉ dẫn đến đối tượng
# list của chúng ta

hom_do_khac = hom_do  # hom_do_khac cũng chỉ là một cái tên cùng chỉ đến đối tượng list
# mà biến hom_do cũng chỉ đến

del hom_do[0]

# chúng ta sẽ kiểm tra hai hòm đồ
print 'hom do gom' , hom_do
print 'hom do khac gom ' , hom_do_khac
# bạn sẽ thấy , chúng cùng chỉ đến một list

# bây giờ chúng ta mới thực sự tạo một list mới từ list cũ , bằng cách
# dùng slice để copy toàn bộ list cũ
# vào list mới
hom_do_khac = hom_do [:]
# bây giờ chúng ta sẽ kiểm tra lại

del hom_do_khac[0]
print 'hom do bay gio gom' , hom_do
print 'hom do khac bay gio gom' , hom_do_khac

# thức sự chúng ta đã có hai list khác nhau , một cái dc chỉ dẫn bởi một biến riêng
```

OutPut

```
hom do gom ['khien', 'giap']
hom do khac gom  ['khien', 'giap']
hom do bay gio gom ['khien', 'giap']
hom do khac bay gio gom ['giap']
```

Từ bây giờ mong bạn cần cẩn thận khi thao tác với tên của các đối tượng…

---

## Hướng đối tượng  (Object-Oriented-Programming)

Phương thức lập trình với hàm và các khối lệnh gọi là hướng thủ tục …
Còn lập trình hướng đối tượng là gì ?

Đoạn dưới mình dựa theo blog của một người bạn …


Hướng đối tượng là một loại phương pháp dc sử dụng để xây dựng một ứng dụng phần mềm xoay quanh khái niệm đối tượng.
Phần mềm sẽ dc chia nhỏ thành nhiều đối tượng ,và để chúng tương tác với nhau…
Một đối tượng (object-…obj) đơn giản là một loại vật chất mà ta có thể cảm nhận bằng xúc giác thị giác… nói chung nó là những sự vật hiện tượng xung quanh chúng ta.

Một đối tượng có những đặc điểm sau:
-trạng thái của đối tượng (mặt tĩnh) : bao gồm thuộc tính và giá trị của thuộc tính
VD : thuộc tính của một đối tượng 'nhân vật' là sinh lực nhận giá trị từ 0 đến 100 …
-hành vi của đối tượng (mặt động) : được quy định bởi những phương thức , một đối tượng sẽ có những phương thức hành xử khác nhau …
VD : nhân vật có thể có những hành động như chạy , tấn công…
-mã số định dạng duy nhất của mỗi đối tượng , ko có hai đối tượng giống nhau
VD: tên của nhân , số khung xe…

Lớp (class) là gì ? một lớp dc định nghĩa như là một khai báo, một mẫu hay một bản thiết kế dc sử dụng để tạo ra các đối tượng.
VD: Lớp "nhân vật kiếm sĩ " sẽ tạo ra các đối tượng " kiếm sĩ "
# lớp có thể là bản thiết kế một chiếc ôtô, nó sẽ tạo ra nhiều đối tượng ôtô có #chung thuộc tính hình dạng

*Đối tượng là cái đã dc định hình cụ thể , còn lớp chỉ là những khái niệm chung chung.*
Lớp thì có những *thuộc tính* , còn đối tượng thì có những *giá trị thuộc tính* …
Đối tượng còn dc gọi là *instance of class*…

Thông điệp và các phương thức :
-Hành vi là cách mà một đối tượng ứng xử hay phản ứng lại dưới dạng các trạng thái (thuộc tính) của nó sẽ thay đổi và các thông điệp (message) của nó sẽ dc truyền đi.
-Thông điệp sẽ dc chuyển tiếp giữa các đối tượng… có thể nói hành vi dc hình thành trong quá trình nhận thông điệp và trả lời thông điệp …
-Phương thức là tập hợp các hành động thực hiện bởi một đối tượng.
Vd: " Kiếm sĩ " tấn công " Rồng " bằng cách gửi cho " Rồng " một thông điệp 'ta đánh mi'

"Rồng " phản ứng lại bằng phương thức : giảm thuộc tính sinh lực của mình xuống , đưa ra hành động bỏ trốn hoặc đánh trả …

Đặc trưng của hướng đối tượng :

- Mô hình hóa một cách thực tế
- Gắn kết các thuộc tính và hành vi lại với nhau
- Tái sử dụng (VD: sau khi xây dựng một lớp 'rồng' , sau này khi muốn phát triển thêm , ta ko phải làm lại từ đầu mà dựa vào lớp "rồng" đó để tạo thêm các đối tượng như "rồng lửa", "rồng đất" … )
- Dễ dàng thay đổi
- Tồn tại dưới nhiều hình thức khác nhau (VD: Đối tượng key blade của Sora trong KingdomHeart có thể dùng để tấn công quái vật (dạng vũ khí) hoặc mở khóa trong các màn chơi (dạng item)…)

Các hướng để lập trình hướng đối tượng
-OOA : Object Oriented Analysis : Phân tích hướng đối tượng với mục đích mô tả về một vấn đề , mô tả đó phải nhất quán , hoàn thiện , đọc dc dưới mọi hình thức khác nhau và nó cuối cùng nó phải dc kiểm tra tính khả thi trong thực tế - theo Mellor
-OOD: "" "" Design : thiết kế hướng đối tượng
-OOP: "" "" Programming : Lập trình hướng đối tượng

Chúng ta thường kết hợp hướng đối tượng và hướng thủ tục…

Quay về với Python…
(một số từ khóa ở trên sẽ ko còn thích hợp nữa , đó là OOP của C++ , nhưng bạn đừng quá lo lắng , vì bạn đã hiểu dc bản chất của hướng đối tượng rồi… )

---

Sau đây là một số thuật ngữ quan trọng , bạn cần nắm rõ để phân biệt tránh nhầm lẫn…

Obj có thể lưu trữ dữ liệu của mình bằng những biến thông thường…

-Những biến phục vụ cho obj và class gọi là field (-trường)
-Biến của obj gọi là instance variable (iv)
-"" "" class gọi là class variable (cv)
-Những hàm (function) thuộc về class mà obj có thể sử dụng gọi là method (-phương thức , trước đây mình dịch method là thủ tục vì nghĩ rằng lúc đó nói là phương thức sẽ khiến bạn khó hiểu , nay xin sửa lại )
-Tất cả field và method gọi chung là attribute (thuộc tính) của class đó

---

Chúng ta khai báo một class bằng từ khóa class

```
class nhan_vat :        # khai báo lớp nhân_vật
        def gioi_thieu(self) :      # khai báo một hàm trong class , nó trở thành một method
                                    # và còn có sự xuất hiện tham số self …

                print 'Xin chao , toi la mot nhan vat trong game'
```

```
# kết thúc khai báo một lớp

eddy = nhan_vat()    # tạo một đối tượng thuộc class nhan_vat

eddy.gioi_thieu()

# đối tượng eddy đã sử dụng method gioi_thieu của class nhan_vat
# bạn lưu ý thấy chúng ta ko cần truyền tham số ở đây
```

OutPut:
*Xin chao , toi la mot nhan vat trong game*

Method gioi_thieu khi dc gọi ko cần truyền tham số nào ,nhưng khi khai báo bất kì một method nào chúng ta vẫn cần khai báo thêm một tham số mặc định là self (-bản thân)

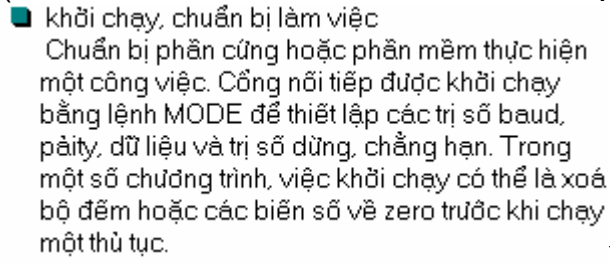
Self ở đây có tác dụng chỉ dẫn đến đối tượng đang thực thi method , nó giúp cho class biết dc đối tượng nào đang yêu cầu sử dụng method…
Nó cũng có nét tương đồng với con trỏ my,me trong LC…

Vì thế eddy.gioi_thieu() cũng tương đương với : nhan_vat.gioi_thieu(eddy)
(có thể tạm hiểu self dc thay bằng eddy)

Chúng ta sẽ bàn thêm sau…

Trong những chương trước , chúng ta thấy một số method đặc biệt có dạng __tên method__ , đó là những method mặc định của P , bây giờ chúng ta sẽ làm quen với method __init__

(init là viết tắt của chữ initialzation – các bạn có thể tham khảo thêm


khởi chạy, chuẩn bị làm việc
Chuẩn bị phần cứng hoặc phần mềm thực hiện một công việc. Cổng nối tiếp được khởi chạy bằng lệnh MODE để thiết lập các trị số baud, pảity, dữ liệu và trị số dừng, chẳng hạn. Trong một số chương trình, việc khởi chạy có thể là xoá bộ đếm hoặc các biến số về zero trước khi chạy một thủ tục.

)

Mời bạn code

```
class nhan_vat :
        def __init__(self, ten_nhan_vat, suc_manh):
 #  dòng (1)… đã có thêm hai tham số ngoài self
                self.ten = ten_nhan_vat
                self.atk = suc_manh   # atk viết tắt của chữ attack
        def gioi_thieu (self) :    # bất kì method nào khi khai báo cũng đều cần có tham số self
                print 'ten toi la ',self.ten , 'voi chi so suc manh la ',self.atk

eddy = nhan_vat('eddyfosman',1500)
# xâu 'eddyfosman' là đối số của tham số ten_nhan_vat …
```

```
eddy.gioi_thieu()
```

OutPut
*ten toi la  eddyfosman voi chi so suc manh la  1500*

Ở dòng (1) , khi khai báo hai tham số ten_nhan_vat và suc_manh thì đồng thời chúng ta cũng đã tạo thêm hai field mới…

Nếu tinh í bạn sẽ thấy , ct đã truyền đối số khi khởi tạo đối tượng eddy , nên chúng ta buộc phải dùng method __init__ ,đó cũng là một đặc điểm của __init__

Còn self.ten và self.atk cũng là field , trong đối tượng eddy , chúng chuyển thành iv : eddy.ten và eddy.atk,  ct vẫn thao tác phép gán trên các field như bình thường …

Chúng ta đã biết có hai loại field
- cv có thể dc truy cập bởi bất kì mọi obj của class
- iv thì ko thể … Ở mỗi obj khác nhau , nó lại là một biến khác , ko liên quan gì tới nhau , mặc dù chúng trùng tên …

Mời bạn code

```
class nhan_vat :
    ''' day la lop cac nhan vat trong game '''
    so_nhan_vat=0

# biến này là cv , có tác dụng đếm các nhân vật trong game ,
# có thể dc truy cập từ
# các obj của class


    def __init__(self , ten) :
        ''' Khoi tao du lieu cho nhan vat '''     # doc string
        self.ten=ten
        # các bạn lưu í  rằng chúng là hai biến khác nhau , một biến của obj , một biến
        # của class , nhưng bạn nên viết tên khác đi  , tránh nhầm lẫn
        print '(ten toi la %s)'  %self.ten
        nhan_vat.so_nhan_vat +=1   # tăng cv so_nhan_vat lên 1 sau khi tạo thêm một
đối tượng


    def __del__(self):   # đây cũng làm một method mặc định giúp bạn xóa đối tượng
        ''' nhan vat se roi khoi game '''
        print  '%s phai rut lui khoi chien truong.' % self.ten
        nhan_vat.so_nhan_vat -=1
        # kiểm tra lại số nhân vật
        if (0== nhan_vat.so_nhan_vat) :
            print 'Toi la nguoi cuoi cung , chung ta da thua ?'
        else:
            print 'Van con lai %d nhan vat.' % nhan_vat.so_nhan_vat
```

```
# thao tác với các đối tượng
eddy = nhan_vat('eddyfosman')
rikky = nhan_vat('rikkyfloren')
kairy = nhan_vat('kairylucifer')
kairy.__del__()
rikky.__del__()
eddy.__del__()
```

OutPut

*(ten toi la eddyfosman)*
*(ten toi la rikkyfloren)*
*(ten toi la kairylucifer)*
*kairylucifer phai rut lui khoi chien truong.*
*Van con lai 2 nhan vat.*
*rikkyfloren phai rut lui khoi chien truong.*
*Van con lai 1 nhan vat.*
*eddyfosman phai rut lui khoi chien truong.*
*Toi la nguoi cuoi cung , chung ta da thua ?*

Code ko hề quá phức tạp , bạn có thể dễ dàng hiểu dc nhờ những dòng chú thích …

---

## Inheritance (Thừa kế)

Nếu bạn muốn viết một chương trình quản lí các nhân vật trong game . Các nhân vật đều có những đặc điểm chung như tên , điểm sinh lực … Nhưng chúng cũng có những điểm khác biệt , Ví dụ : kiếm sĩ có điểm attack , rồng có điểm defend …

Bạn có thể tạo hai class riêng cho kiếm sĩ và rồng , nhưng nếu bạn muốn chèn thêm một thuộc tính chung mới , thì bạn sẽ phải code sửa lại cả hai class , và nếu có nhiều hơn hai class thì bạn sẽ phải lặp đi lặp lại nhiều lần một đoạn code …

Một cách tốt  hơn là chúng ta sẽ tạo một class chung gọi là nhan_vat , hai class kiem_si và rong sẽ cùng thừa kế class nhan_vat này

Và khi bạn chèn thêm một thuộc tính cho class nhan_vat , thì đồng thời hai class kia cũng dc chèn thêm thuộc tính đó …

Nhưng những thay đổi ở class kiem_si sẽ ko ảnh hưởng đến class rong
Chúng ta có thể coi các đối tượng của class kiem_si và rong là đối tượng của class nhan_vat , từ đó các obj của hai class này có thể sử dụng các method của class nhan_vat ( gọi là tính polymorphism-tính đa hình)

Class nhan_vat dc gọi là superclass , hay baseclass
Class rong, class kiem_si dc gọi subclass , derivedclass

```
class nhan_vat:   # khai báo một base class như bình thường
    ''' Mieu ta nhung dac diem chung cua tat ca cac nhan vat , do la ten
```

```
    va diem sinh luc'''

    def __init__(self,ten,sinh_luc):
        self.t_nhan_vat = ten              # bạn nên khai báo các field có tên khác nhau
        self.sl_nhan_vat = sinh_luc
    def gioi_thieu(self):
        print 'Toi la ' , self.t_nhan_vat, 'co' , self.sl_nhan_vat , 'diem sinh luc'

class kiem_si(nhan_vat):   #  khai báo tên của base class trong dấu ()
    ''' Mieu ta nhan vat kiem si , thua ke thuoc tinh ten va sinh luc cua
    Class nhan_vat , thuoc tinh rieng la atk'''
    def __init__(self, ten , sinh_luc , atk):

# chúng ta có thể các truy cập các field , method của base class bằng mẫu :
#   tên base class . tên method (self, tham số …)

        nhan_vat.__init__(self,ten, sinh_luc)

# bạn cần truyền đối số từ base class vào subclass như trên
# bây giờ có thể coi như class kiem_si cũng có hai dòng lệnh của class nhan_vat
# self.t_nhan_vat = ten
# self.sl_nhan_vat = sinh_luc

        self.atk_nhan_vat=atk  # đây là thuộc tính riêng của class kiem_si

    def gioi_thieu(self):
        nhan_vat.gioi_thieu(self)
        print 'Luc tan cong %d' % self.atk_nhan_vat

class rong(nhan_vat):
    '''Mieu ta nhan vat rong'''
    def __init__(self,ten,sinh_luc,defend):
        nhan_vat.__init__(self,ten,sinh_luc)
        self.df_nhan_vat=defend
    def gioi_thieu(self):
        nhan_vat.gioi_thieu(self)
        print 'Luc phong thu %d' %self.df_nhan_vat

eddy=kiem_si('eddyfosman',2000,1500)

dragon=rong('rong_vang',3000,2000)

# hai dòng code sau ko liên quan đến tính thừa kế…

print eddy  # hai dòng này giúp chúng ta chứng minh dc , eddy và dragon chỉ là những
cái tên…
print dragon
# …nó chỉ dẫn đến vị trí lưu trữ thông tin của hai đối tượng này trong bộ nhớ
# việc self chỉ dẫn đến tên của đối tượng ,
# mà tên của đối tượng cũng chỉ dẫn đến đối tượng , ở trong
```

```
# C gọi là con trỏ của con trỏ

cac_nhan_vat=[eddy,dragon]   # tạo một list

for i in cac_nhan_vat :   # biến i cũng đóng vai trò như một reference chỉ dẫn đến obj cần thao tác
    i.gioi_thieu()
```

OutPut

```
<__main__.kiem_si instance at 0x00D33260>
<__main__.rong instance at 0x00D332D8>
```

# có thể dịch là đối tượng của chúng ta đang thực sự nằm ở vị trí 0x00D33260 trong bộ nhớ,  thuộc class kiem_si trong module __main__

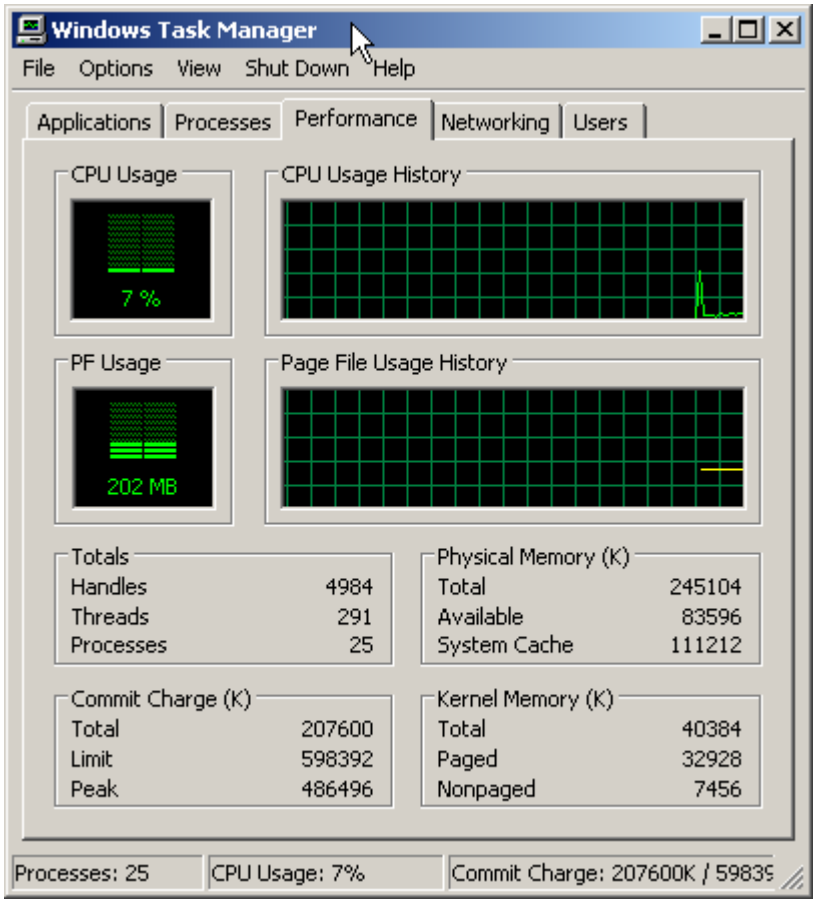
```
Toi la  eddyfosman co 2000 diem sinh luc
Luc tan cong 1500
Toi la  rong_vang co 3000 diem sinh luc
Luc phong thu 2000
```

Vậy là bạn dã trở thành một lập trình có ít nhiều kinh nghiệm với Python , nhưng bạn vẫn cần tham khảo them manual của Python…
Nếu bạn muốn tiếp tục phát triển game , mình xin nêu hai con đường mà mình cho là phù hợp :

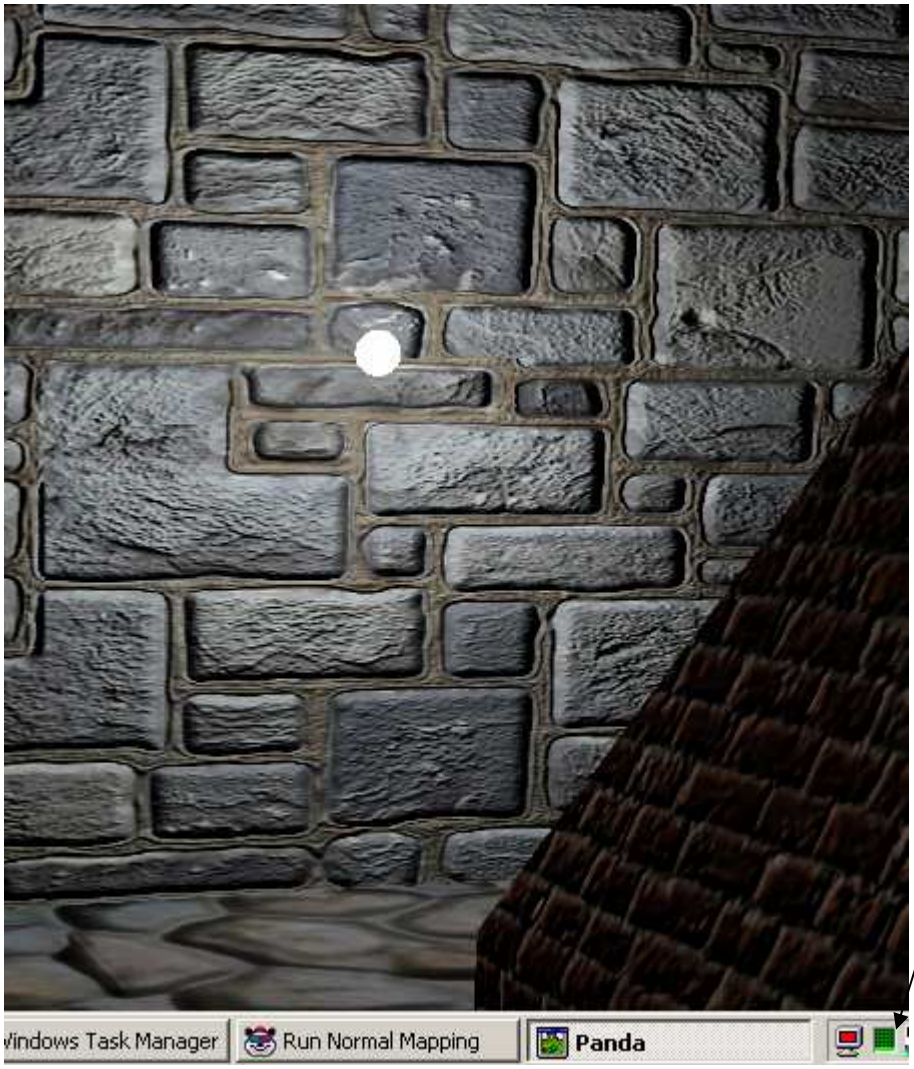
- Blender : Logic Block của B thật mạnh mẽ , bạn có thể làm một game mini bắn nhau theo phong cách CounterStrike mà ko cần đụng đến một dòng code , rất hợp với những người ko phải là coder  …
- Panda : đây là game engine của hãng hoạt hình WaltDisney dùng để làm một số game thương mại như ToonTownOnline, PirateOnline (Cướp biển vùng Caribe ), những mini game trên trang game của WaltDisney …Panda đã trở thành OpenSource khoảng 2 năm (nhưng WaltDissney vẫn đang dùng nó để làm game), ổn định ,dễ học vì dùng Python, là lựa chọn tốt nếu bạn muốn làm một game cực lớn (nhưng nó ko có khả năng vẽ như B nên bạn vẫn cần dùng B )

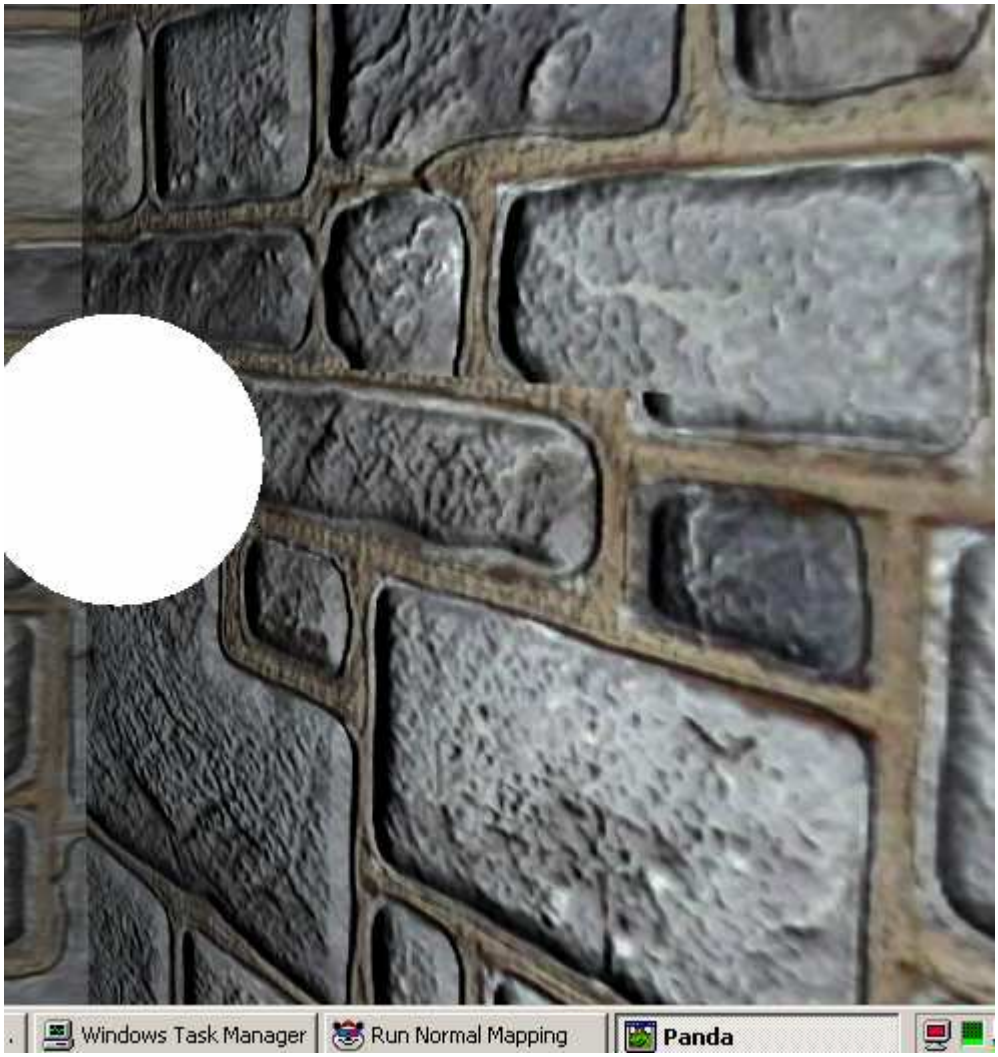
Trang chủ của Panda là www.panda3d.org , bạn hãy down Panda.exe 55MB và manual reference 10 MB
Trong thư mục cài đặt có thư mục Sample , bạn hãy thử nghiệm chúng  …

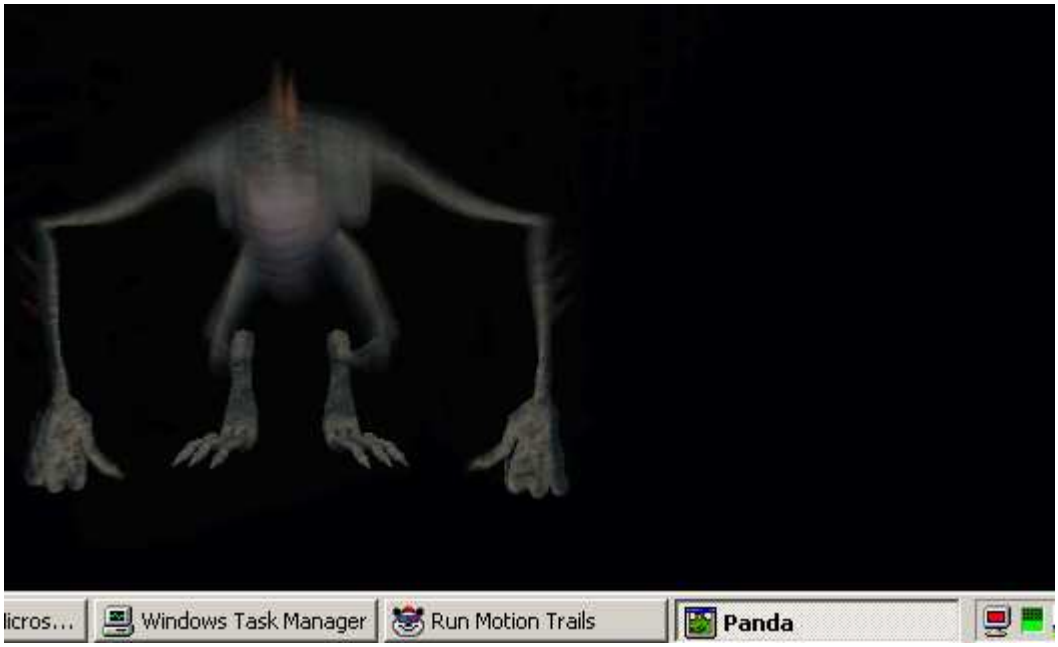
Khi test các demo của Panda ,bạn hãy mở



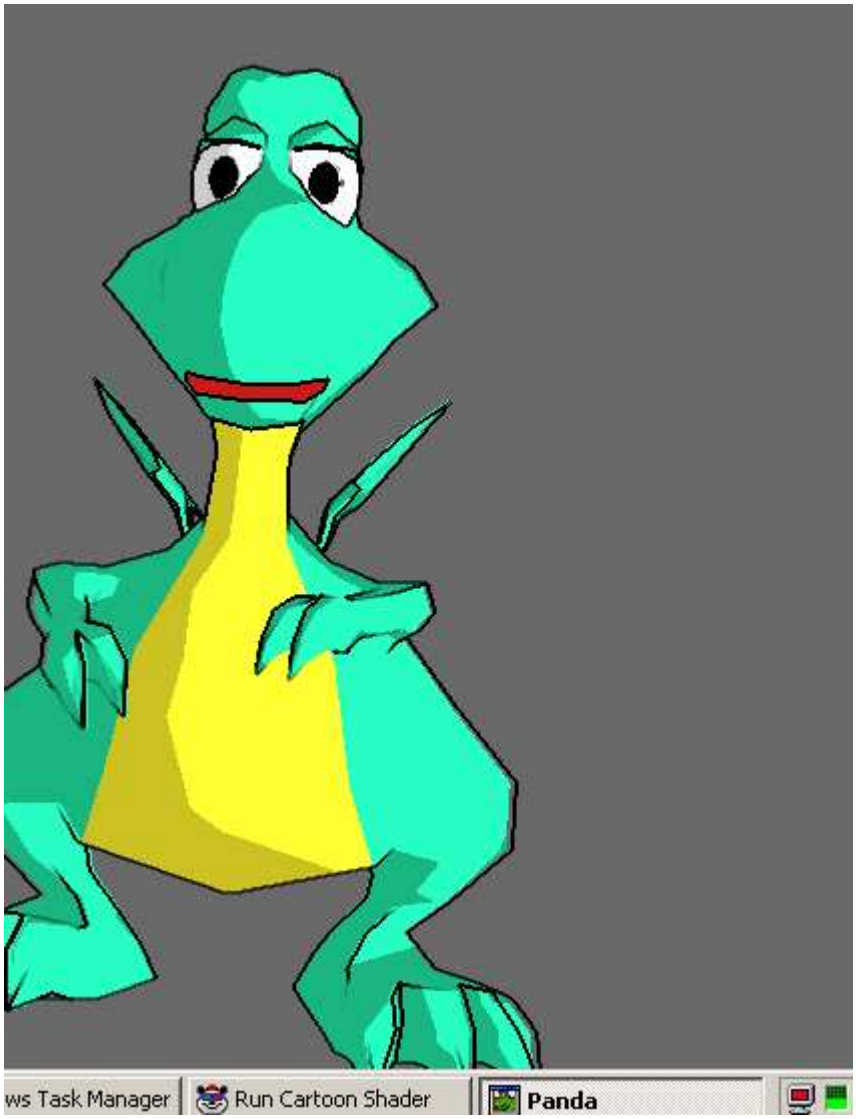

để kiểm tra xem độ tối ưu hóa CPU của Panda , bạn sẽ rất ngạc nhiên khi thấy CPU thường ko chạy quá 50 phần trăm


indows Task Manager
Run Normal Mapping
Panda


Windows Task Manager
Run Normal Mapping
Panda


Windows Task Manager
Run Motion Trails
Panda


cros...
Windows Task Manager
Run Motion Trails
Panda


ws Task Manager
Run Cartoon Shader
Panda


chỉ trừ trong demo feature này

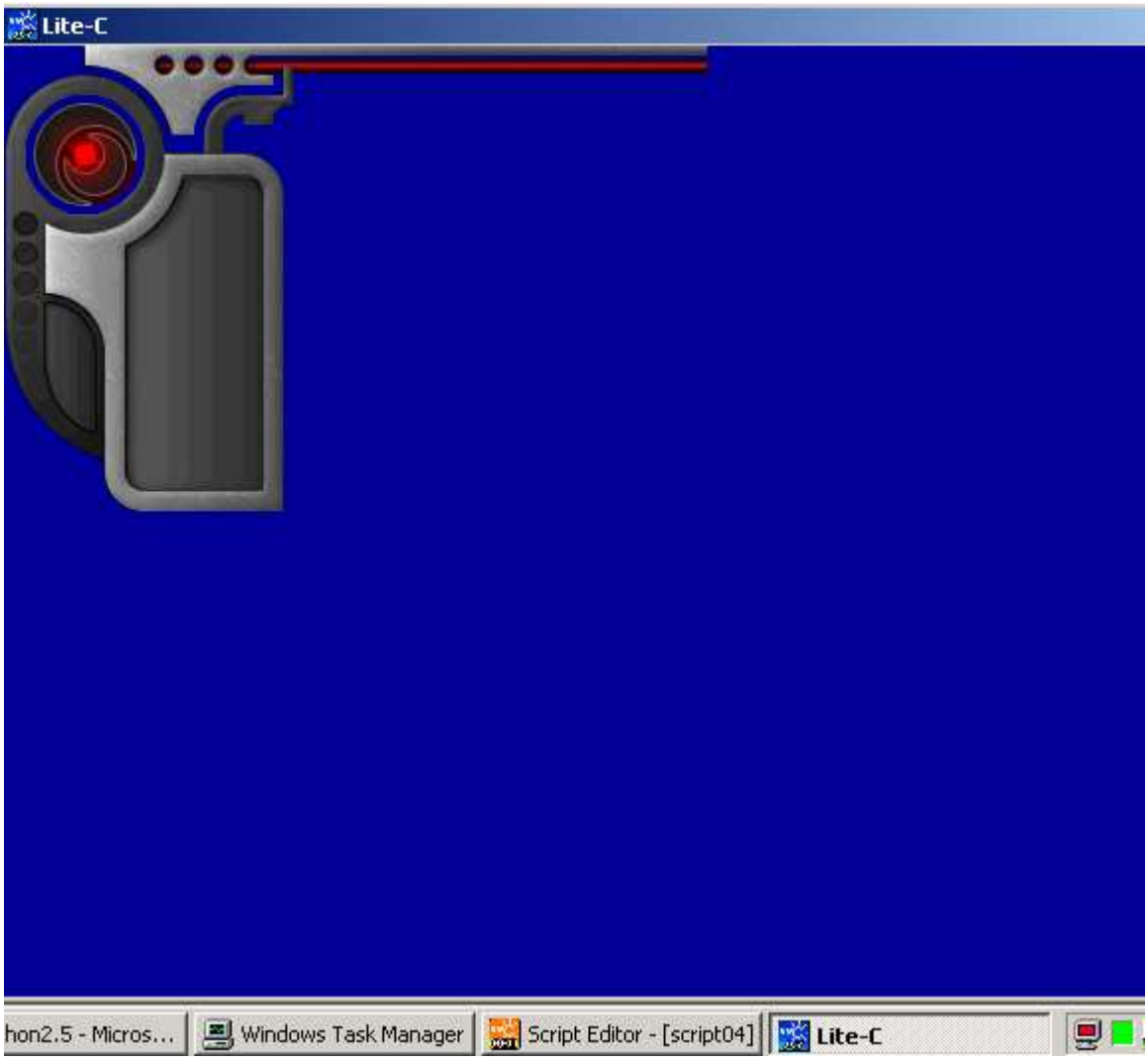

là Panda khiến CPU đôi lúc chạy lên 100 phần trăm , rồi sau đó lại giảm xuống …có thể do terrain rộng quá …

Và trong demo game AirBlade , khi chơi CPU cũng ko chạy quá 60 phần trăm , nhưng khi bạn nhấn nút continue sau khi game over lần thứ hai thì CPU đột ngột tăng lên 100 % , lúc nào cũng như vậy …


Điều đáng nói ở đây (và cũng là một trong những lí do mình chọn Panda) khi mình test các engine khác như Blender , IrrLitch , 3DGS … thì chúng đều chạy đến 100 phần trăm CPU dù chỉ là một demo rất đơn giản …
Đây là cấu hình máy của mình , share 16 MB RAM cho VGA

AMD Athlon(tm) 64 Processor
3500+
2.21 GHz, 240 MB of RAM

3DGS:


Lite-C
hon2.5 - Micros...
Windows Task Manager
Script Editor - [script04]
Lite-C


Kết quả của test này chẳng có ý nghĩa gì cả , mình ko muốn nói rằng Panda ổn định hơn 3DGS vì mỗi engine lại có một cơ chế tối ưu hóa khác nhau (như tốc độ update các thông số , thiết đặt max fps…), mà mình muốn chứng minh cho bạn thấy sự nghiêm túc của những người đã phát triển Panda …
Đồng thời Pnd dc mọi người đánh giá rất cao bởi nó dễ học và sử dụng …

Ease of Use: Once you understand some of the 'panda' ways of doing things, it really is quite easy to use. The tutorials/manual really help you get up on the learning curve/starting to produce right away

-easy to use
-easy to learn (thx to python)

I use panda3d for our game development because its powerful, easy to use and has great documenation, tutorials and a great community.

Fully integrated Python engine, very easy to use and speed seems more than adequate.

After using 3 other engines for some months (sdl, irrlicht, OGRE) I discovered panda3D and I think I finally found the perfect engine for my project

Panda3D is very user-friendly (python, good design, and very good documentation)

Ease of Use:

That's where Panda is really great. You'll have your first scene up and running in a few minutes.
The fact that Panda works perfectly with Python makes for quick production cycles. I can't think of
better engine/language combination to start developing your very first game. You could even start t
learn Panda and Python at the same time.
And if there's something you won't find in the manual, you'll find it in the forum.

Panda3D is just an amazing engine. It's cross-platform, it's not hard to program a game in a very sho
time, (also because it makes use of the easy Python language.)

i test the engine for a time now.. is very easy and have a short learnig curve... the develop time is
short because of python... the stability is grate and the performance to (for a binding of python in
c++)...

DevMaster.net
your source for game development

Nếu có điều kiện bạn hãy đến thăm forum của Panda tại panda3d.org

Bạn có thể tự thử nghiệm để so sánh Panda với các game engine khác …
EddyFosman
FOS là viết tắt của chữ Free Open Source