ΕΙΣΑΓΩΓΗ ΣΤΗ PYTHON

Η python πρόκειται για διερμηνευόμενη γλώσσα, τα αρχεία της οποίας τελειώνουν σε .py, ενώ ένα πρόγραμμα γραμμένο σε python δεν χρειάζεται κάποια στανταρ δομή!

ΜΕΤΑΒΛΗΤΕΣ: Οι μεταβλητές στη python είναι εύκολες στο χειρισμό τους, διότι δεν χρειάζονται δήλωση και δεν έχουν τύπο!! Μια μεταβλητή η οποία έχει δεχτεί ακέραια τιμή, μπορεί στην αμέσως επόμενη εντολή να δεχθεί string ως τιμή!

Χρήσιμες συναρτήσεις:

- int(): Μετατρέπει την παράμετρο σε ακέραια τιμή.
- float(): Μετατρέπει την παράμετρο σε τιμή κινητής υποδιαστολής.
- str(): Μετατρέπει την παράμετρο σε string
- type(): Επιστρέφει τον τύπο της μεταβλητής που δέχεται ως παράμετρο.

Για παράδειγμα, έχουμε το επόμενο πρόγραμμα:

```
A="1234.55"
B=int(A)  #θα παρει τιμή 1234
C=float(A) #θα πάρει τιμή 1234.55
print A,B,C #εκτύπωση '1234.55' 1234 1234.55
```

Όπως βλέπουμε μια ανάθεση τιμής γίνεται με το σύμβολο =, όπως και σε άλλες γλώσσες. Ακόμη, ένα σχόλιο ξεκινάει με το σύμβολο #, ενώ με την εντολή print μπορούμε να εκτυπώσουμε στην οθόνη.

ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ: Η εισαγωγή από το πληκτρολόγιο μπορεί να γίνει με την συνάρτηση raw_input(), η οποία επιστρέφει πάντα ένα string. Με τις συναρτήσεις int() και float() μπορούμε να μετατρέψουμε το string σε ακέραιο ή δεκαδικό όταν ζητάμε αντίστοιχη είσοδο.

Για παράδειγμα, έχουμε τα επόμενα προγράμματα:

```
print "Πως σε λένε?"
name=raw_input() #έστω δίνουμε Άλεξ
print "Γειά σου",name #εκτυπώνει: Γειά σου Άλεξ
```

```
print "Δώσε έναν ακέραιο:",
number=(int)raw_input() #έστω δίνουμε 10
print "'Έδωσες",number #εκτυπώνει: Έδωσες 10
```

Η raw_input() διαθέτει εσωτερικά μια μορφή της εντολής print, και έτσι μπορούμε να την χρησιμοποιούμε με παράμετρο το μήνυμα που θέλουμε ώστε να έχουμε είσοδο.

Για παράδειγμα, έχουμε το επόμενο πρόγραμμα:

```
number=(int)raw_input("Δώσε έναν ακέραιο:")
print "Έδωσες",number
```

Σχόλια:

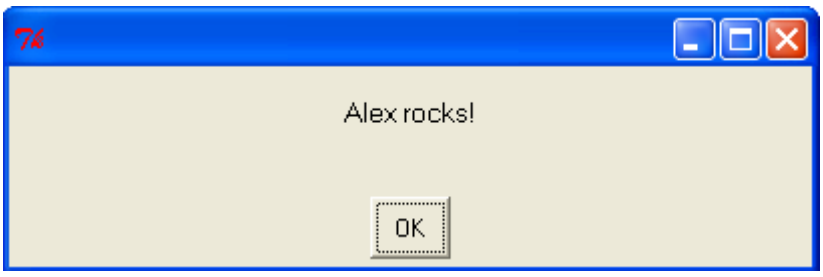
- Η print χωρίζει τα διάφορα αντικείμενα που εμφανίζει με κόμμα. Αυτόματα το κόμμα προσθέτει ένα κενό και δεν αλλάζει γραμμή!
- Τα διπλά με τα μονά εισαγωγικά σε ένα string δεν έχουν καμία διαφορά!

ΓΡΑΦΙΚΕΣ ΔΙΕΠΑΦΕΣ ΧΡΗΣΤΗ: Χρησιμοποιούμε ένα βασικό πρόγραμμα το οποίο ονομάζεται easyqui.py και έτσι, αν θέλουμε να δημιουργήσουμε γραφικές διεπαφές χρήστη πρέπει να το συμπεριλάβουμε με την εντολή import.

Κουτιά Μηνυμάτων: Χρησιμοποιούμε την μέθοδο msgbox(message_string) του easygui. Για παράδειγμα έχουμε το εξής κομμάτι κώδικα:

```
import easygui
easygui.msgbox("Alex rocks!")
```

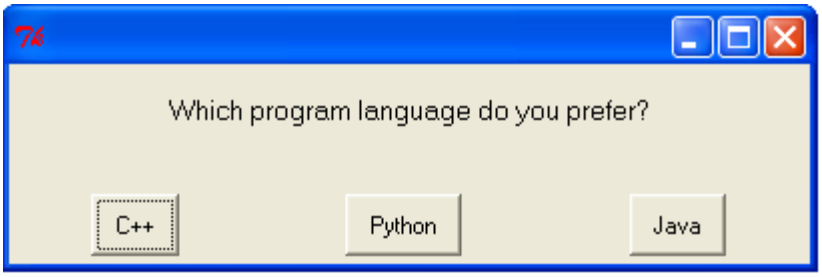
τότε ως έξοδο θα έχουμε:



Κουτιά Κουμπιών: Χρησιμοποιούμε τη μέθοδο buttonbox(message_string,choices=[]). Για παράδειγμα έχουμε το εξής κομμάτι κώδικα:

```
import easygui
L=easygui.buttonbox("Which program language do you prefer?",
                    choices=['C++','Python','Java'])
easygui.msgbox("You picked "+L)
```

τότε ως έξοδο θα έχουμε:

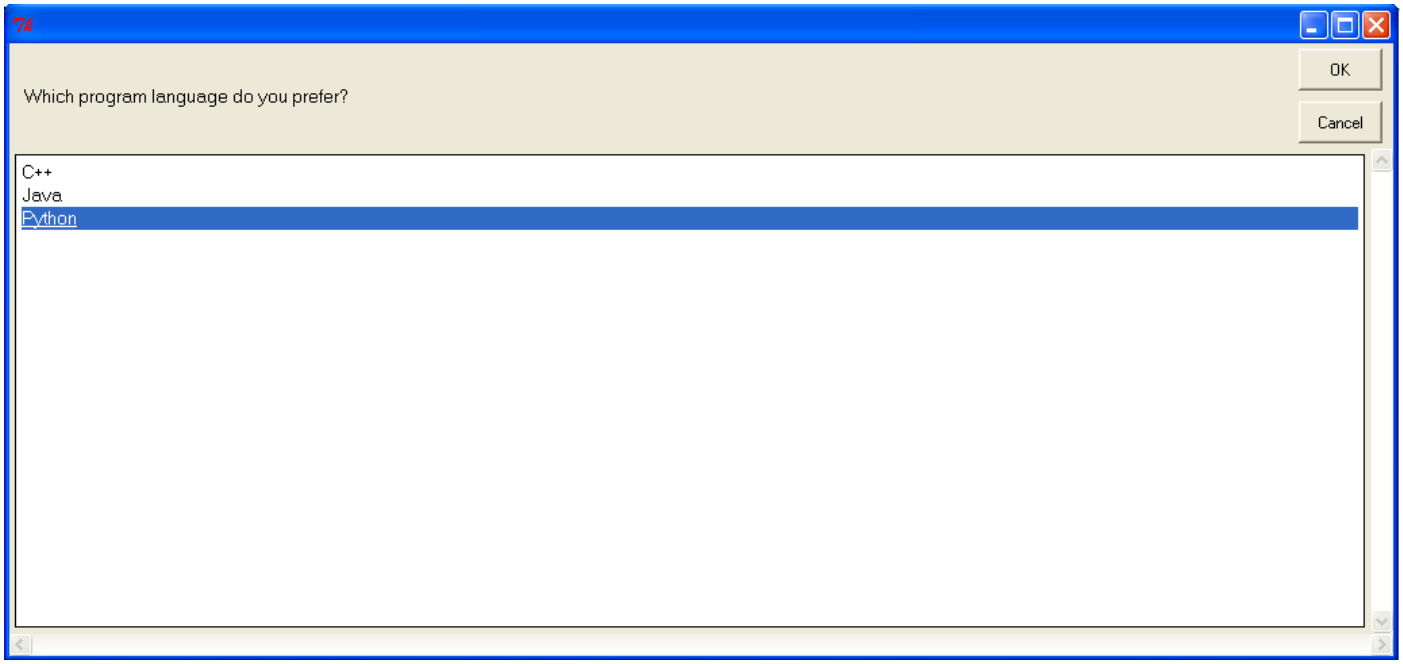


Και έστω ότι διαλέγουμε το δεύτερο κουμπί, δηλαδή Python, τότε θα έχουμε ως έξοδο:

Κουτιά Επιλογής: Χρησιμοποιούμε την μέθοδο choicebox(message_string,choices=[]). Για παράδειγμα έχουμε το εξής κομμάτι κώδικα:

```
import easygui
L=easygui.choicebox("Which program language do you prefer?",
                    choices=['C++','Java','Python'])
easygui.msgbox("You picked "+L)
```

τότε θα έχουμε:

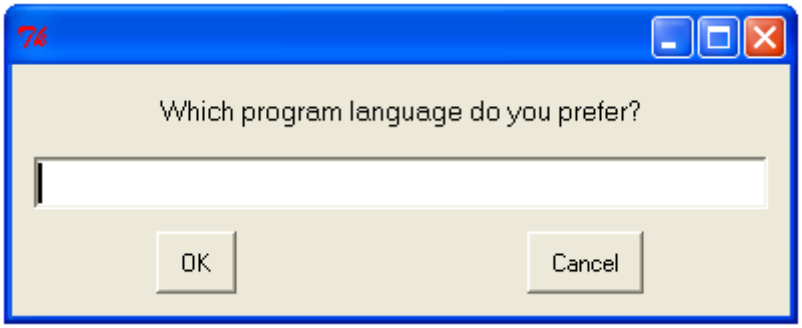


και αν επιλέξουμε και πάλι την python, θα έχουμε και πάλι ως έξοδο το κουτί μηνύματος που μας ειδοποιεί ότι επιλέξαμε την python.

Κουτιά Εισόδου: Χρησιμοποιούμε τη μέθοδο enterbox(message_string), η οποία δημιουργεί ένα πλαίσιο στο οποίο ο χρήστης δεδομένα. Για παράδειγμα, έχουμε το εξής κομμάτι κώδικα:

```
import easygui
L=easygui.enterbox("Which program language do you prefer?")
easygui.msgbox("You entered "+L)
```

τότε θα έχουμε:



Και έστω ότι γράφουμε python, τότε θα έχουμε ως έξοδο:

Αυτά είναι λίγα από τα έτοιμα guis που μπορούμε να φτιάξουμε με την python, θα δούμε και άλλα με το πρόγραμμα pygame.py.

ΔΟΜΗ ΕΠΙΛΟΓΗΣ: Η επιλογή στη python γίνεται μόνο με την εντολή if, η οποία έχει το εξής γενικό συντακτικό:

```
if condition:
    commands
elif condition:
    commands
else:
    commands
```

Στην python, οι λογικοί τελεστές είναι εξαιρετικά απλοί. Για του λόγου το αληθές, το λογικό ΝΑΙ είναι το and, το λογικό Ή είναι το or και το λογικό ΟΧΙ είναι το not!

Για παράδειγμα, το επόμενο πρόγραμμα ελέγχει έναν αριθμό που δίνει ο χρήστης από το πληκτρολόγιο για το αν είναι άρτιος ή περιττός:

```
number=(int)raw_input("Δώσε έναν ακέραιο")
if number%2==0:
   print "Είναι άρτιος!"
else:
   print "Είναι περιττός!"
```

Ακόμη, θα μπορούσαμε να το κάνουμε και με γραφική διεπαφή χρήστη, με τον εξής κώδικα:

```
import easygui
number=(int)easygui.enterbox("Δώσε έναν ακέραιο")
if number%2==0:
   easygui.msgbox("Είναι άρτιος!")
else:
   easygui.msgbox("Είναι περιττός!")
```

Ένα ακόμη παράδειγμα είναι ο έλεγχος ενός αριθμού για το αν θετικός και έπειτα να διακρίνεται αν είναι άρτιος ή περιττός:

```
number=(int)raw_input("Δώσε έναν ακέραιο")
if number>0 and number%2==0:
   print "Είναι άρτιος!"
elif number>0 and number%2!=0:
   print "Είναι περιττός!"
else:
   print "Δεν είναι θετικός!"
```

ΔΟΜΗ FOR: Η επαναληπτική εντολή for (επανάληψη όταν γνωρίζουμε τον αριθμό των επαναλήψεων) στην python έχει την εξής μορφή:

```
for counter in [K1,K2,...,KN]:
   commands
```

όπου $K_1,K_2,...,K_N$ είναι συγκεκριμένοι αριθμοί, τους οποίους δέχεται ως τιμή ο μετρητής των επαναλήψεων counter. Φυσικά, υπάρχει και η συνάρτηση range(), η οποία δημιουργεί ένα διάστημα τιμών για τον μετρητή και έτσι έχουμε και τις εξής δύο εκδοχές της for:

```
for counter in range(start,end+1):
   commands
```

```
for counter in range(start,end+1,step):
    commands
```

Η πρώτη μορφή δηλώνει ότι ο μετρητής θα πάρει τιμές στο διάστημα [start,end] με βήμα 1 ενώ η δεύτερη μορφή δηλώνει ότι το βήμα θα καθορίζεται από την παράμετρο step.

Ακόμη, μπορούμε να έχουμε και μια ακόμη μορφή της range() η οποία δέχεται μόνο μια παράμετρο η οποία καθορίζει το άνω όριο του διαστήματος, δηλαδή την end+1, ενώ το κάτω όριο είναι πάντα το 0:

```
for counter in range(end+1):
    commands
```

Για παράδειγμα, το επόμενο πρόγραμμα υπολογίζει το άθροισμα Σ=1+2+3+4+5 και με τους τρεις τρόπους:

```
sum=0
for counter in [1,2,3,4,5]:
    sum=sum+counter
print sum
```

```
sum=0
for counter in range(1,6):
    sum=sum+counter
print sum
```

```
sum=0
for counter in range(6):
    sum=sum+counter
print sum
```

```
sum=0
for counter in range(5,0,-1):
    sum=sum+counter
print sum
```

Στη τρίτη εκδοχή του προγράμματος, το γεγονός ότι η range() ξεκινάει από το 0 δεν μας επηρεάζει, αφού το 0 στην άθροιση είναι ουδέτερος αριθμός. Ακόμη, στην τελευταία εκδοχή για να δείξουμε την χρήση της range() με την παράμετρο step, πάμε ανάποδα στο διάστημα [1,5]!

Η εντολή for, μπορεί ακόμη να χρησιμοποιηθεί και με strings! Δηλαδή αντί να πούμε στον μετρητη να πάρει τιμές από ένα διάστημα αριθμών, μπορεί να πάρει χαρακτήρες από ένα string.

Για παράδειγμα, έχουμε το επόμενο πρόγραμμα:

```
for letter in "Hi Alex!":
    print letter,
```

Θα εκτυπώσει με την σειρά τους χαρακτήρες του string "Hi Alex" αφήνοντας κενό ανάμεσα σε κάθε γράμμα, δηλαδή: H i A l e x !

ΔΟΜΗ WHILE: Η επαναληπτική εντολή while (επανάληψη όταν δεν γνωρίζουμε τον αριθμό των επαναλήψεων) έχει το εξής βασικό συντακτικό:

```
while condition:
    commands
```

Για παράδειγμα, το επόμενο πρόγραμμα δέχεται συνέχει strings μέχρις ότου δεχθεί το σωστό pin για να συνεχίσει:

```
pin="@1eX"
code=raw_input("Δώσε το pin σου")
while code!=pin:
    print "Λάθος pin!Διορθώστε:"
    code=raw_input()
print "Σωστό pin,συνεχίστε!"
```

ΔΙΑΚΟΠΤΕΣ: Έχουμε τις εντολές break και continue με τον γνωστό τρόπο χρήσης.

ΛΙΣΤΕΣ: Μια λίστα στην python κάνει την δουλειά των πινάκων, οι οποίοι πολύ απλά δεν υπάρχουν! Μια λίστα πριν χρησιμοποιηθεί θα πρέπει να δημιουργηθεί. Αυτό μπορεί να γίνει με δύο βασικούς τρόπους:

Δημιουργία μιας κενής λίστας:

my_list=[]

Δημιουργία λίστας με αρχικές τιμές:

my_list=[K_1,K_2,...,K_N]

Οι λίστες στην python μπορούν να περιέχουν οτιδήποτε, δηλαδή μπορούν να έχουν τόσο μια ακέραια τιμή όσο και ένα string ή μια άλλη λίστα!

Μπορούμε να πάρουμε τιμές από μια λίστα ακριβώς όπως θα κάναμε και με έναν πίνακα. Για παράδειγμα το επόμενο τμήμα κώδικα:

```
my_list=[1,2,3,4,5]
print my_list #Εκτύπωση [1,2,3,4,5]
print my_list[0] #Εκτύπωση 1
for i in range(6):
    print my_list[i], #Εκτύπωση 1,2,3,4,5
```

Ακόμη, μπορούμε να αναφερθούμε και τμηματικά στα στοιχεία μιας λίστας, όπως για παράδειγμα στο επόμενο τμήμα κώδικα:

```
my_list=[1,2,3,4,5]
print my_list[1:4] #Εκτύπωση [2,3,4]
print my_list[:2] #Εκτύπωση [1,2]
print my_list[2:] #Εκτύπωση [3,4,5]
print my_list[:] #Εκτύπωση [1,2,3,4,5]
```

Στην ουσία το [Κ:Λ] λειτουργεί όπως και η συνάρτηση range(), δηλαδή ξεκινάει μεν από το Κ αλλά πάει μέχρι το Λ-1! Ακόμη, αυτού του είδους οι μερικές αναφορές στην ουσία δημιουργούν αντίγραφα της λίστας, δεν την εκτυπώνουν απλά.

Η εισαγωγή δεδομένων σε μια λίστα μπορεί να γίνει με τις εξής τρεις μεθόδους:

- append(K): Εισάγει την παράμετρο στο τέλος της λίστας.
- extend([K_1,K_2,...,K_N]): Εισάγει την παραμετρική λίστα στο τέλος της λίστας.
- insert(p,K): Εισάγει την παράμετρο Κ στην θέση p της λίστας.

Για παράδειγμα, έχουμε στο επόμενο πρόγραμμα δημιουργούμε αρχικά μια κενή λίστα και σιγά σιγά της προσθέτουμε δεδομένα:

```
my_list=[] #Κενή λίστα
my_list.append(1) #Η λίστα: [1]
my_list.extend([2,3,4]) #Η λίστα: [1,2,3,4]
my_list.insert(0,0) #Η λίστα: [0,1,2,3,4]
```

Η διαγραφή δεδομένων από μια λίστα μπορεί να γίνει με δύο μεθόδους και μια εντολή:

- pop(): Έχουμε δύο εκδόσεις. Η πρώτη δεν δέχεται παραμέτρους και απλά διαγράφει το τελευταίο στοιχείο της λίστας, το οποίο και επιστρέφει. Η δεύτερη δέχεται μια παράμετρο η οποία είναι η θέση του στοιχείου που θα διαγράψει και θα επιστρέψει.
- remove(K): Διαγράφει το αντικείμενο Κ από τη λίστα. Το Κ πρέπει να υπάρχει στη λίστα.
- del: Διαγράφει ένα συγκεκριμένο στοιχείο (αναφορά της λίστας σε μια συγκεκριμένη θέση).

Για παράδειγμα έχουμε το επόμενο κομμάτι κώδικα, το οποίο με χρήση των παραπάνω μεθόδων σιγά σιγά διαγράφει στοιχεία από μια λίστα:

```
my_list=[0,1,2,3,4,5,6,7,8]
print my_list.pop() #Εκτυπώνει: 8, Λίστα: [0,1,2,3,4,5,6,7]
print my_list.pop(1) #Εκτυπώνει:1, Λίστα: [0,2,3,4,5,6,7]
my_list.remove(4) #Η λίστα: [0,2,3,5,6,7]
del my_list[0] #Η λίστα: [2,3,5,6,7]
```

Η αναζήτηση δεδομένων σε μια λίστα μπορεί να γίνει με την εντολή in. Για παράδειγμα, έστω ότι στο επόμενο κομμάτι κώδικα θέλουμε να διαγράψουμε την τιμή 1 με την μέθοδο remove(), τότε:

```
my_list=[1,2,3,4,5]
if 1 in my_list:
   my_list.remove(1)
```

Ακόμη, με την μέθοδο index(object), μπορούμε να βρούμε την θέση που έχει το object στην λίστα. Ωστόσο πρέπει αναγκαστικά, όπως και στην remove(), το αντικείμενο να υπάρχει στη λίστα. Έτσι, έχουμε το επόμενο κομμάτι κώδικα:

```
my_list=[1,2,3,4,5]
if 4 in my_list:
   print my_list.index(4) #Εκτυπώνει 3
```

Η ταξινόμηση μιας λίστας μπορεί να γίνει είτε με τη μέθοδο sort() η οποία μεταβάλλει την ίδια τη λίστα είτε με την συνάρτηση sorted() η οποία δημιουργεί ένα ταξινομημένο αντίγραφο της λίστας. Για παράδειγμα έχουμε το εξής κομμάτι κώδικα:

```
my_list=[2,1,4,5,3]
new_list=sorted(my_list)
print my_list,new_list #Εκτυπώνει [1,2,3,4,5] [2,1,4,5,3]
my_list.sort() #Εκτυπώνει [1,2,3,4,5]
print my_list
```

Μπορούμε να δημιουργήσουμε και σταθερές λίστες, δηλαδή λίστες που αρχικοποιούνται και έπειτα δεν μπορούμε να μεταβάλλουμε το περιεχόμενο τους. Αυτό γίνεται ως εξής:

```
my_list=(1,2,3,4,5)
```

Τέλος, μπορούμε να δημιουργήσουμε και λίστα λιστών (σαν δισδιάστατο πίνακα)!! Ο παρακάτω κώδικας είναι ένα τέτοιο παράδειγμα:

```
N=[1,2,3,4,5]
M=[6,7,8,9,10]
my_list=[N,M] #Ακόμη και απ'ευθείας my_list=[[1,2,3,4,5],[6,7,8,9,10]]
for L in my_list:
    print L #Εκτυπώνει διαδοχικά: [1,2,3,4,5] και [6,7,8,9,10]
print my_list[0] #Εκτυπώνει [1,2,3,4,5]
print my_list[0][3] #Εκτυπώνει 4, το στοιχείο στην 1 γραμμή, 4 στήλη
```

ΣΥΝΑΡΤΗΣΕΙΣ: Μια συνάρτηση στην python ορίζεται με το διακριτικό def και έχει το εξής γενικό συντακτικό:

```
def function_name(parameters):
    commands
```

Η επιστροφή κάποιου δεδομένου από μια συνάρτηση μπορεί να γίνει με την κλασική εντολή return.

Η εμβέλεια των μεταβλητών είναι η εξής: Μέσα σε μια συνάρτηση έχουμε τοπική εμβέλεια. Έξω από μια συνάρτηση, κάθε μεταβλητή είναι δημόσια. Αν θέλουμε να χρησιμοποιήσουμε την δημόσια έκδοση μιας μεταβλητής μέσα στο σώμα μιας συνάρτησης, τότε θα χρησιμοποιήσουμε το προσδιοριστικό global.

Για παράδειγμα, έχουμε το επόμενο πρόγραμμα που υπόλογίζει το μέγιστο μεταξύ τριών αριθμών:

```
def max(a,b,c):
    m=a
    if b>m:
      m=b
    if c>m:
      m=c
    return m

print max(3,2,5)
```

ΚΛΑΣΕΙΣ ΚΑΙ ΑΝΤΙΚΕΙΜΕΝΑ: Μέσα σε μια κλάση της python ορίζουμε μόνο τις εσωτερικές μεθόδους της και όχι τα πεδία δεδομένων που θα έχει (αφού η python δεν θέλει ορισμούς)! Έτσι, ο ορισμός μιας κλάσης μπορεί να γίνει ως εξής:

```
class class_name:
    def method1(parameters):
      commands;
    ...
    def methodN(parameters):
      commands;
```

Μια μέθοδος πρέπει πάντα να έχει μια παράμετρο με όνομα self (ή και this), η οποία όταν καλούμε την μέθοδο δεν την περνάμε (περνιέται αυτόματα)! Η self αναφέρεται στο αντικείμενο που καλεί την μέθοδο της κλάσης.

Για παράδειγμα, έχουμε το επόμενο πρόγραμμα το οποίο ορίζει μια μπάλα:

```
#Ορισμός της κλάσης ball:
class ball:
    def bounce(self):
        if self.direction=='down':
            self.direction='up'

my_ball=ball() #Δημιουργία αντικειμένου
#Θέτουμε τιμές στα αυθαίρετα πεδία του:
my_ball.direction='down'
my_ball.color='green'
my_ball.size='middle'

print "Μόλις δημιούργησα μια μπάλα με:"
print "Κατεύθυνση:",my_ball.direction
print "Χρώμα:",my_ball.color
print "Μέγεθος:",my_ball.size

print "Τώρα της κάνω αναπήδηση!"
my_ball.bounce()
print "Νέα κατεύθυνση:",my_ball.direction
```

Για την απόδωση τιμής σε κάποια αυθαίρετα πεδία δεδομένων (ό,τι θέλουμε) μπορούμε να χρησιμοποιήσουμε δημιουργό με όνομα __init__():

```
#Νέος ρισμός της κλάσης ball:
class ball:
    def __init__(self,direction,color,size):
        self.direction=direction
        self.color=color
        self.size=size

    def bounce(self):
        if self.direction=='down':
            self.direction='up'

#Πλέον η δημιουργία αντικειμένου γίνεται έτσι:
my_ball=ball('down','green','middle')
```

Η python διαθέτει ειδικές μεθόδους. Οι ειδικές μέθοδοι είναι μέθοδοι που συμπεριλαμβάνονται αυτόματα με την δημιουργία μιας κλάσης αντικειμένων. Εξαιρετικά χρήσιμες τέτοιες μέθοδοι είναι οι: __init__() και __str__().

Η μέθοδος __str__() ορίζει τι ακριβώς θα εκτυπωθεί στην οθόνη όταν θέλουμε να χρησιμοποιήσουμε ένα αντικείμενο με την εντολή print. Για παράδειγμα έχουμε το επόμενο κομμάτι κώδικα:

```
class ball:
    def __init__(self,direction,color,size):
        self.direction=direction
        self.color=color
        self.size=size

    def __str__(self):
        return "Hi,i am a "+self.color+" "+self.size+" ball!"

my_ball=ball('down','green','middle')
print my_ball #Εκτύπωση: Hi,i am a green middle ball!
```

Σχόλιο: Οι κλάσεις στην python δεν διαθέτουν δημόσια, ιδιωτικά ή προστατευμένα δεδομένα (δεν έχει νόημα άλλωστε αφού δεν έχουμε δηλώσεις). Όλα είναι δημόσια→Προσοχή!

Χαρακτηριστικό των αντικειμενοστρεφών εφαρμογών είναι ο πολυμορφισμός! Πολυμορφισμός σημαίνει ότι μπορούμε να έχει ίδια ονόματα αλλά άλλες διαδικασίες. Για παράδειγμα, το επόμενο τμήμα κώδικα δημιουργεί δύο κλάσεις οι οποίες περιέχουν και οι δύο μια συνάρτηση με το ίδιο όνομα, αλλά ωστόσο επιτελούν διαφορετική λειτουργία:

```
class triangle:
    def __init__(self,width,height):
        self.width=width
        self.height=height

    def area(self):
         return self.width*self.height/2.0

class square:
    def __init__(self,size):
        self.size=size

    def area(self):
         return self.size*self.size

my_triangle=triangle(2,3)
my_square=square(2)
print my_triangle.area(),my_square.area() #Εκτύπωση: 3.0 4
```

Η κληρονομικότητα στην python είναι εξαιρετικά απλή. Για παράδειγμα έστω ότι έχουμε μια κλάση σημείων στο επίπεδο και θέλουμε φτιάξουμε μια κλάση για σημεία στον χώρο. Τότε θα έχουμε το εξής τμήμα κώδικα:

```
class point2d:
    def __init__(self,x,y):
       self.x=x
       self.y=y

    def get_x(self):
       return self.x

    def get_y(self):
       return self.y

class point3d(point2d):
    def __init__(self,x,y,z):
       point2d.__init__(self,x,y)
       self.z=z

    def get_z(self):
        return self.z

p=point3d(1,2,3)
print p.get_x(),p.get_y(),p.get_z() #Εκτύπωση: 1 2 3
```

MODULES: Τα Modules είναι ξεχωριστά τμήματα κώδικα, δηλαδή ξεχωριστά αρχεία με κώδικα γραμμένο σε python (σαν τα αρχεία επικεφαλίδας στη C). Για παράδειγμα έχουμε το πρόγραμμα:

```
#αρχείο max_p.py
def max(a,b,c):
    m=a
    if b>m:
      m=b
    if c>m:
      m=c
    return m
```

```
#αρχείο main.py
import max_p,min_p
print max_p.max(1,2,3)
print min_p.min(1,2,3)
```

```
#αρχείο min_p.py
def min(a,b,c):
    m=a
    if b<m:
      m=b
    if c<m:
      m=c
    return m
```

Φυσικά, και τα τρια αρχεία θα πρέπει να αποθηκευτούν στον ίδιο φάκελο για να έχουν άμεση επικοινωνία το ένα με το άλλο (βασικά το main με τα άλλα δύο). Αν θέλουμε, μπορούμε να φορτώσουμε από ένα module μόνο συναρτήσεις που θέλουμε, όπως στο εξής πρόγραμμα:

```
#αρχείο max_p.py
def max2(a,b)
    if a>b:
      return a
    else:
      return b

def max3(a,b,c):
    m=a
    if b>m:
      m=b
    if c>m:
      m=c
    return m

def maxL(my_list):
   m=my_list[0]
   for number in my_list:
       if number>m:
         m=number
   return m
```

```
#αρχείο main1.py
from max_p import max3
print max3(1,2,3)
```

```
#αρχείο main2.py
from max_p import *
print max2(10,0)
print max3(1,2,3)
M=[9,1,2,10,5]
print maxL(M)
```

Το main1 εισάγει μόνο τη συνάρτηση max3() από το module max_p, ενώ το main2 εισάγει όλες τις συναρτήσεις του. Παρατηρούμε ότι όταν εισάγουμε με αυτόν τον τρόπο κάποιες ή όλες τις συναρτήσεις (ή και κλάσεις) από ένα module δεν χρειάζεται να αναφερόμαστε σε αυτό με το όνομα του.

Μερικά βασικά modules είναι τα time και random τα οποία μας βοηθάνε στο να χειριζόμαστε τον χρόνο του συστήματος και να δημιουργούμε τυχαίους αριθμούς αντίστοιχα.

Για παράδειγμα έχουμε τα επόμενα προγράμματα:

```
import time
for M in range(10):
   time.sleep(1)
   print M
```

```
import random
print random.randint(0,100)
print random.random()*10
```

Το πρώτο πρόγραμμα μετράει 10 δευτερόλεπτα. Η συνάρτηση sleep(secs) του module time κάνει το πρόγραμμα να περιμένει secs δευτερόλεπτα.

Το δεύτερο πρόγραμμα δημιουργεί τυχαίους αριθμούς. Η συνάρτηση randint(K,L) δημιουργεί τυχαίους ακέραιους στο διάστημα [K,L], ενώ η random() τυχαίους δεκαδικούς στο διάστημα [0,1]. Πολλαπλασιάζοντας με το 10 κάνουμε το διάστημα [0,10].

ΕΙΣΑΓΩΓΗ ΣΤΑ ΓΡΑΦΙΚΑ

Για να δημιουργήσουμε γραφικά θα χρησιμοποιήσουμε το module pygame!

Δημιουργία Παραθύρου: Για να δημιουργήσουμε ένα απλό παράθυρο με το pygame θα πρέπει να γράψουμε το εξής κομμάτι κώδικα:

```
import pygame,sys
pygame.init() #Δημιουργεί το παράθυρο
screen=pygame.display.set_mode([640, 480]) #Θέτει στο παράθυρο διαστάσεις 640x480
#Το while-loop τρέχει συνέχεια το παράθυρο μέχρι να θελήσουμε να κάνουμε κάτι
while True:
#Το for-loop αναζητάει ενέργειες του χρήστη σε μια προκαθορισμένη λίστα του pygame
#όπου αποθηκεύονται όλες οι ενέργειες
    for event in pygame.event.get():
        if event.type == pygame.QUIT: #pygame.QUIT είναι το κουμπί X για κλείσιμο
            sys.exit() #Το σύστημα κλείνει το παράθυρο που έχουμε δημιουργήσει
```

Το Αντικείμενο Εμφάνισης: Το αντικείμενο εμφάνισης στον κώδικα μας είναι το screen και είναι εξαιρετικά χρήσιμο. Έχει δύο μορφές: Η πρώτη είναι αυτή που τρέχει ήδη στην οθόνη και η δεύτερη είναι αυτή που θα τρέξει στην οθόνη αν αλλάξουμε κάτι, όπως να ζωγραφίσουμε ένα κύκλο ή να βάλουμε μια εικόνα.

Για να είναι πιο ομαλή η μεταβολή από τη μία μορφή στη άλλη χρησιμοποιούμε τη μέθοδο flip() της κλάσης display του pygame κάθε φορά που μεταβάλουμε το αντικείμενο εμφάνισης.

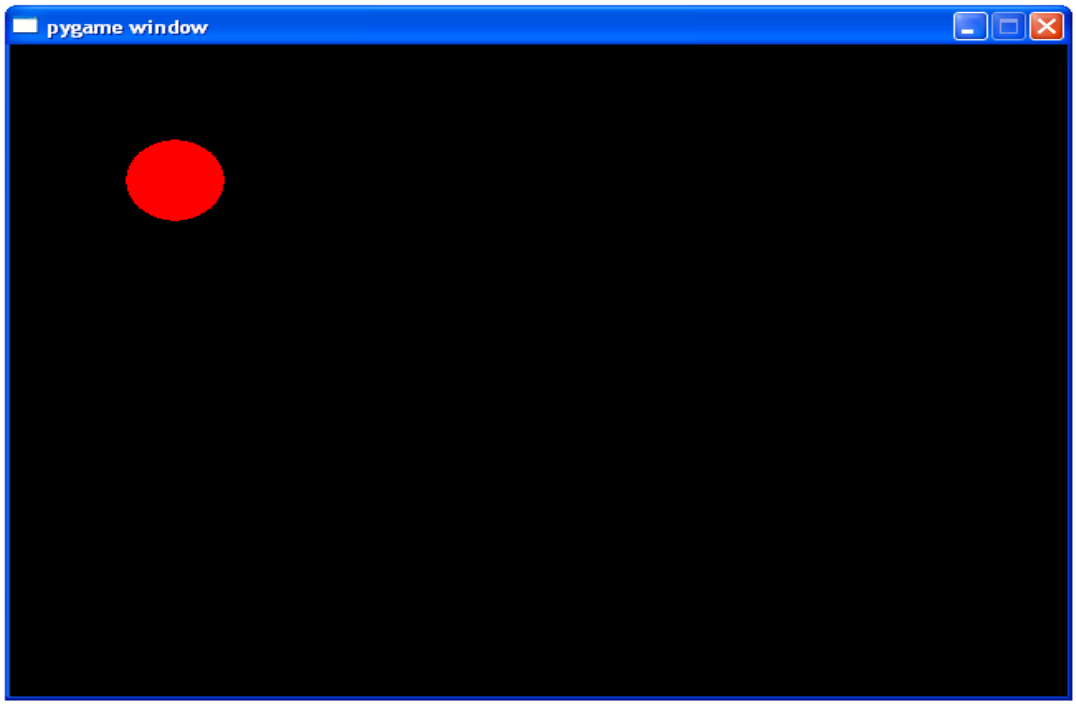
Δημιουργία Ενός Κύκλου: Για να δημιουργήσουμε ένα κύκλο χρησιμοποιούμε την μέθοδο circle() της κλάσης draw του pygame στην οποία θα πρέπει να δώσουμε πληροφορίες για:

- Την επιφάνεια όπου θα ζωγραφιστεί ο κύκλος, δηλαδή το αντικείμενο εμφάνισης, screen.
- Τι χρώμα θα έχει ο κύκλος σε RGB, έστω ότι θα είναι κόκκινος: [255,0,0]
- Σε ποια θέση του screen θα δημιουργηθεί, έστω 100 pixels δεξιά και 100 pixels κάτω από την πιο αριστερή γωνία του screen: [100,100]
- Τι μέγεθος θα έχει ο κύκλος, έστω 30 rad σε pixels.
- Το πάχος της γραμμής που θα δημιουργήσει τον κύκλο: Αν είναι 0 τότε γεμίζει ολόκληρος ο κύκλος με χρώμα.

Έτσι, το επόμενο πρόγραμμα δημιουργεί έναν κύκλο με τις πιο πάνω ιδιότητες:

```
import pygame,sys
pygame.init()
screen=pygame.display.set_mode([640, 480])
pygame.draw.circle(screen,[255,0,0],[100,100],30,0) #Δημιουργία του κύκλου
pygame.display.flip() #Αλλάζει την μορφή του αντικειμένου εμφάνισης screen
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
```

Έτσι, αν τρέξουμε το παραπάνω πρόγραμμα θα πάρουμε το εξής αποτέλεσμα:



Χρώματα: Ο συνδυασμός [255,0,0] για να πάρουμε το κόκκινο είναι λίγο περίεργος. Το σύστημα χρωμάτων που χρησιμοποιούμε είναι το RGB (Red Green Blue). Δηλαδή το κάθε ένα από τα βασικά χρώματα μπορεί να πάρει μια τιμή από 0 έως και 255 ενώ όλα τα άλλα χρώματα προκύπτουν από συνδυασμό των τριών βασικών. Δηλαδή σίγουρα είναι:

- Κόκκινο: [255,0,0]
- Πράσινο: [0,255,0]
- Μπλε: [0,0,255]
- Μαύρο: [0,0,0]
- Άσπρο: [255,255,255]

Για να μην χρειάζεται όμως να γνωρίζουμε συνδυασμούς αριθμών μπορούμε να χρησιμοποιήσουμε τον πίνακα THECOLORS που υπάρχει στην κλάση colors του pygame. Τότε θα έχουμε:

```
import pygame,sys
from pygame.color import THECOLORS
pygame.init()
screen=pygame.display.set_mode([640, 480])
pygame.draw.circle(screen,THECOLORS["red"],[100,100],30,0) #Δημιουργία του κύκλου
pygame.display.flip() #Αλλάζει την μορφή του αντικειμένου εμφάνισης screen
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
```

Θέση Στο Παράθυρο: Η θέση στο παράθυρο ξεκινάει από το [0,0] στην πιο αριστερή γωνία του παραθύρου (πάνω αριστερά) και πάει προς τα κάτω και δεξιά αυξάνοντας.

- Φανταστείτε αυτό το μέγεθος να αλλάζει τυχαία, τότε θα φαινόταν πως ο κύκλος (μπάλα) κινείται στο παράθυρο!

Μέγεθος Σχημάτων: Σε έναν κύκλο το μόνο που θέλουμε για να έχουμε το μέγεθος του είναι η ακτίνα του σε rad. Ωστόσο, αν θέλουμε να ζωγραφίσουμε κάτι διαφορετικό όπως είναι ένα ορθογώνιο, τότε θα πρέπει να δώσουμε το μήκος και πάχος του.

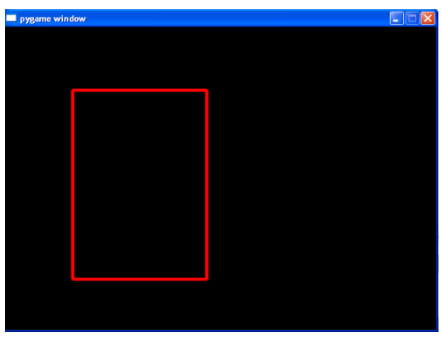
Για το ορθογώνιο, το pygame διαθέτει μια εσωτερική κλάση για να τα επεξεργάζεται ως αντικείμενα. Έτσι, με ένα τέτοιο μπορούμε να καθορίσουμε και την θέση που θα έχει στο παράθυρο αλλά και τις διαστάσεις του: [left,top,width,height] και έτσι η δημιουργία ενός ορθογωνίου γίνεται ως εξής:

```
pygame.draw.rect(screen,THECOLORS["red"],[100,100,300,20],0)
```

Πάχος Γραμμής: Είναι το πάχος της γραμμής που σχηματίζει το σχήμα που επιθυμούμε. Αν δεν είναι ίσο με 0 τότε δεν θα είναι γεμμάτο! Για παράδειγμα, έστω:

```
pygame.draw.rect(screen,THECOLORS["red"],[100,100,300,20],5)
```

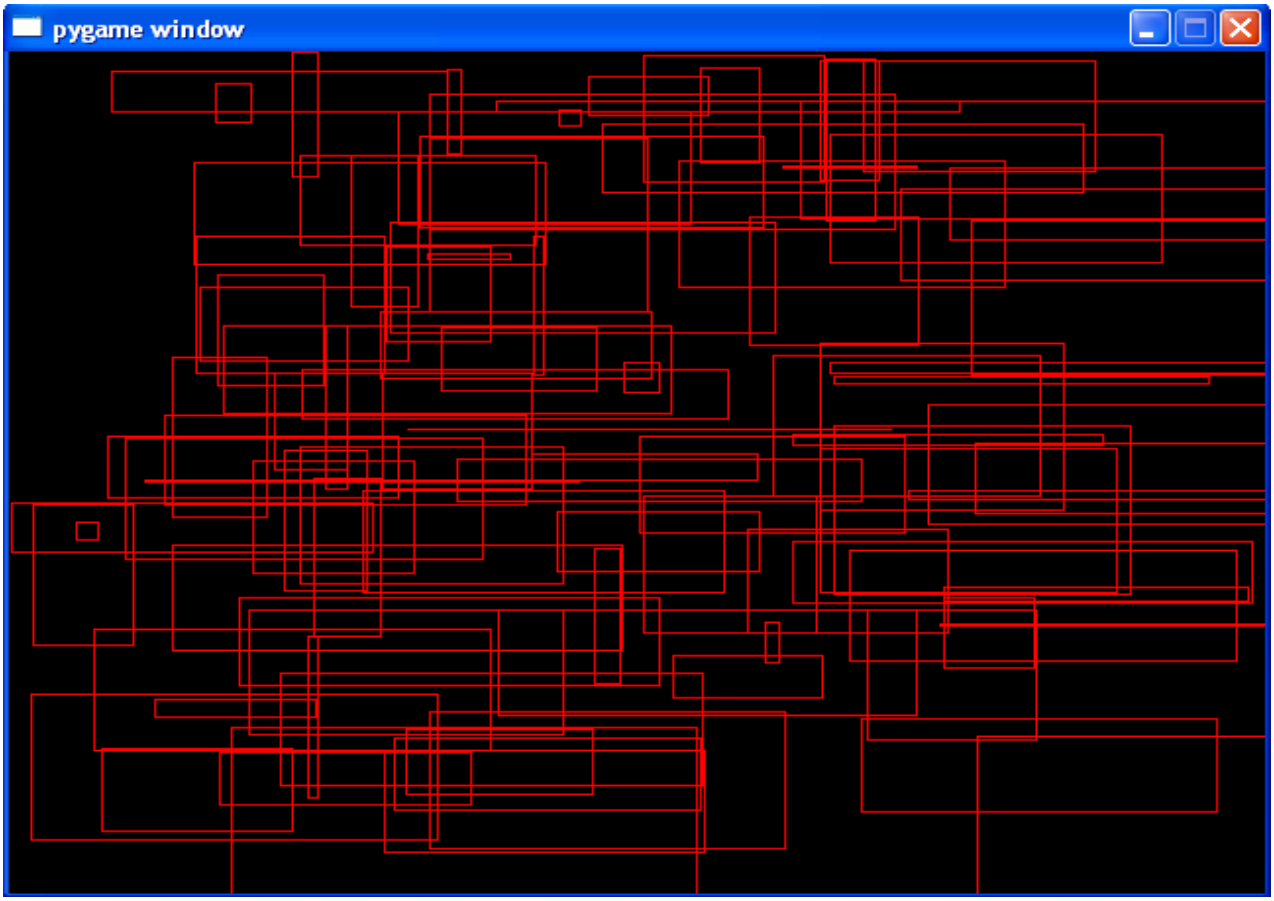
Θα έχουμε:



Ένα Παράδειγμα: Μπορούμε να δημιουργήσουμε πολλά ορθογώνια σχήματα σε διάφορα (τυχαία) σημεία του παραθύρου μας με διάφορα (τυχαία) μεγέθη και έτσι θα έχουμε ένα έργο τέχνης!! Για παράδειγμα:

```
import pygame,sys,random
from pygame.color import THECOLORS
pygame.init()
screen=pygame.display.set_mode([640, 480])
for i in range(100):
    width=random.randint(0,250)
    height=random.randint(0,100)
    top=random.randint(0,400)
    left=random.randint(0,500)
    pygame.draw.rect(screen,THECOLORS["red"],[left,top,width,height],1)
pygame.display.flip()
while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()
```

Και έτσι, θα έχουμε το εξής αποτέλεσμα:



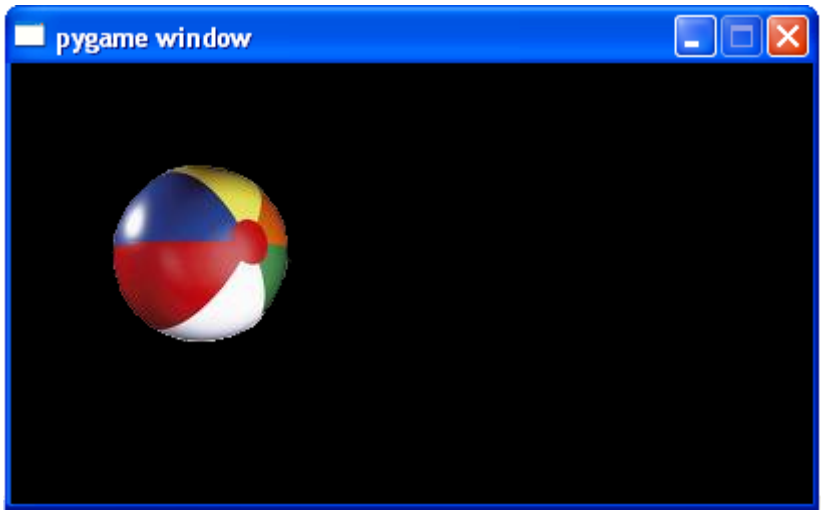
Εικόνες: Το να ζωγραφίζουμε εμείς οι ίδιοι σχήματα είναι ένα τρόπος για να δημιουργήσουμε γραφικά. Ωστόσο είναι πιο εύκολο να χειριζόμαστε έτοιμες εικόνες (Άλλωστε τα γραφικά δεν είναι η δουλειά ενός προγραμματιστή!).

Θα χρησιμοποιήσουμε την εικόνα μιας μπάλας με όνομα αρχείου beach_ball.png. Η εικόνα αυτή θα πρέπει να βρίσκεται στον ίδιο φάκελο με το ίδιο το πρόγραμμα που θα την χρησιμοποιήσει ώστε να μπορεί να την βρει εύκολα. Η είκονα είναι η:

Το επόμενο πρόγραμμα τοποθετεί στο παράθυρο της εφαρμογής μας την παραπάνω εικόνα:

```
import pygame,sys,random
pygame.init()
screen=pygame.display.set_mode([400, 220])
my_ball=pygame.image.load("beach_ball.png") #Φορτώνει την εικόνα στο my_ball
screen.blit(my_ball,[50,50]) #Αντιγράφει την μπάλα στη θέση 50,50 του παραθύρου
pygame.display.flip()
while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()
```

Τότε θα έχουμε:

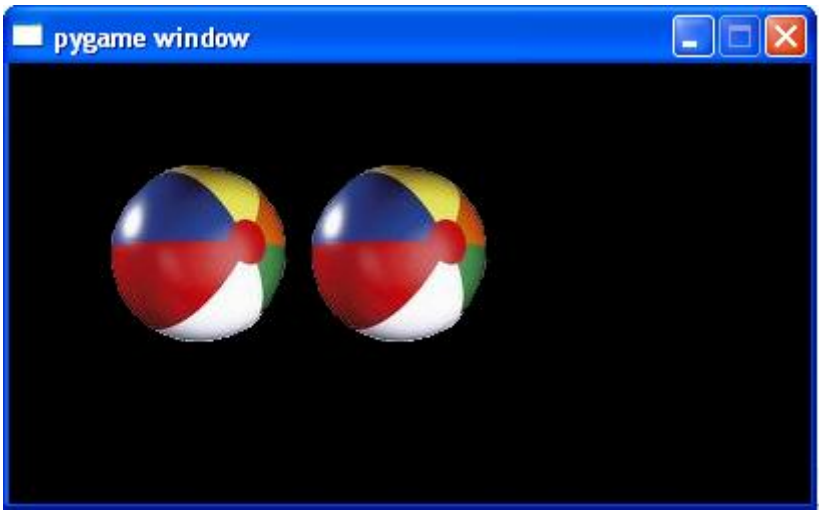


Τώρα μπορούμε να κάνουμε την μπάλα να κινηθεί!!!

Το μόνο που έχουμε να κάνουμε είναι να την αντιγράψουμε σε μια άλλη θέση του παραθύρου προσθέτοντας και μια χρονοκαθυστέρηση ώστε να φαίνεται στον χρήστη ότι κινείται! Αυτό μπορεί να γίνει εύκολα με το παρακάτω κομμάτι κώδικα:

```
import pygame,sys,random
pygame.init()
screen=pygame.display.set_mode([400, 220])
my_ball=pygame.image.load("beach_ball.png")
screen.blit(my_ball,[50,50])
pygame.display.flip()
pygame.time.delay(2000) #Προσθέτουμε χρονοκαθυστέρηση
screen.blit(my_ball,[150,50]) #Αντιγραφή της μπάλας στη θέση 150,50 του παραθύρου
pygame.display.flip()
while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()
```

Έτσι, αν τρέξουμε το παραπάνω πρόγραμμα, στην αρχή θα δούμε την μπάλα στη θέση (50,50) και μετά από λίγες χρονικές στιγμές αυτή θα εμφανιστεί και στη θέση (150,50) σαν να έχει κινηθεί:



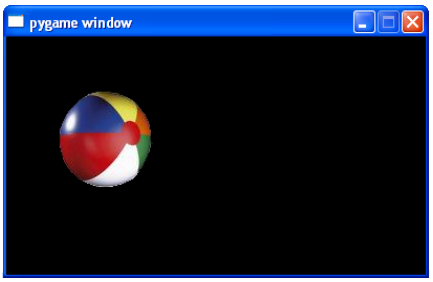
Για να φτιάξουμε πραγματικά animations, θα πρέπει όταν μεταφέρουμε το αντικείμενο (στην περίπτωση μας η μπάλα) σε μια καινούργια θέση και να διαγράψουμε την προηγούμενη έκδοση του!

Όσον αφορά τα γραφικά, στην πραγματικότητα δεν μπορούμε να διαγράψουμε ένα αντικείμενο! Για να διαγράψουμε ένα αντικείμενο απλά ζωγραφίζουμε από πάνω του το ίδιο χρώμα με το φόντο!

Η επόμενη γραμμή κώδικα μπορεί να τοποθετηθεί αμέσως μετά τη μετακίνηση της μπάλας στη νέα της θέση. Αυτό που κάνει είναι να ζωγραφίσει πάνω από τη παλιά μπάλα ένα ορθογώνιο με μαύρο:

```
pygame.draw.rect(screen,THECOLORS["black"],[50,50,90,90],0)
```

Έτσι αν τρέξουμε πλέον το πρόγραμμα θα έχουμε διαδοχικά:



Στην περίπτωση μας η διαγραφή έγινε με απλό βάψιμο του παραθύρου με μαύρο. Αν όμως η μπάλα είχε ως φόντο κάποιο περίεργο σχέδιο τι θε γινόταν? Μια λύση είναι να βαφτεί ολόκληρο το παράθυρο και έπειτα να γίνει η μεταφορά της μπάλας.

Αν έχετε τρέξει το παραπάνω πρόγραμμα θα έχετε δει ότι η μεταφορά της μπάλας από το πρώτο σημείο στο δεύτερο είναι λιγάκι απότομη! Ας την κάνουμε πιο ομαλή:

```
import pygame,sys,random
from pygame.color import THECOLORS
pygame.init()
screen=pygame.display.set_mode([400, 220])
my_ball=pygame.image.load("beach_ball.png")
x=50
y=50
screen.blit(my_ball,[x,y])
pygame.display.flip()
for counter in range(1,100): #Συνεχόμενη κίνηση
    pygame.time.delay(20) #Μειώνουμε την χρονοκαθυστέρηση
    pygame.draw.rect(screen,THECOLORS["black"],[x,y,90,90],0) #Διαγραφή
    x=x+5 #Συνέχεια μετακινείται στην ευθεία
    screen.blit(my_ball,[x,y])
    pygame.display.flip()
while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()
```

Έτσι η μπάλα κινείται από την αρχική της θέση μέχρι το τέλος του παραθύρου και στη συνέχεια εξαφανίζεται διότι το x ξεπερνάει την μέγιστη τιμή του που είναι 400. Έχουμε δύο επιλογές ή θα κάνει κύκλο ή θα αναπηδά στο δεξί (και μετά στο αριστερό) άκρο του παραθύρου!

Ας δούμε πρώτα την αναπήδηση: Η μπάλα είναι περίπου 90pixels, τα οποία μετράνε από την αριστερή της πλευρά. Έτσι όταν θα κάνει αναπήδηση από τα αριστερά θα προσέχουμε αν x==0, ενώ όταν θα κάνει αναπήδηση από τα δεξιά θα προσέχουμε αν x==310.

Αντί να κάνει τόσα βήματα η μπάλα όσα καθορίζει το for-loop, ας το καθορίζει ο χρήστης με το να επιλέξει να κλείσει την εφαρμογή. Έτσι θα βάλουμε τον κώδικα στο while-loop. Η x θα αλλάζει με μια συγκεκριμένη ταχύτητα speed (έστω 5 όπως και πριν). Έτσι:

```
import pygame,sys,random
from pygame.color import THECOLORS
pygame.init()
screen=pygame.display.set_mode([400, 220])
my_ball=pygame.image.load("beach_ball.png")
x=50
y=50
speed=5

screen.blit(my_ball,[x,y])
pygame.display.flip()
while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()

        pygame.time.delay(20)
        pygame.draw.rect(screen,THECOLORS["black"],[x,y,90,90],0)
        x=x+speed
        #Αν χτυπήσουμε κάποιο άκρο του παραθύρου η κατεύθυνση αλλάζει
        if x>screen.get_width()-90 or x<0:
          speed=-speed
        screen.blit(my_ball,[x,y])
        pygame.display.flip()
```

Τώρα, μπορούμε να περάσουμε την κίνηση αυτή και στις δύο διαστάσεις (κίνηση και στον y-άξονα), άρα θα πρέπει να έχουμε μια ταχύτητα ακόμη y_speed. Έτσι λοιπόν, θα έχουμε το εξής πρόγραμμα:

```
import pygame,sys,random
from pygame.color import THECOLORS
pygame.init()
screen=pygame.display.set_mode([400, 220])
my_ball=pygame.image.load("beach_ball.png")
x=50
y=50
x_speed=5
y_speed=5

screen.blit(my_ball,[x,y])
pygame.display.flip()
while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()

        pygame.time.delay(20)
        pygame.draw.rect(screen,THECOLORS["black"],[x,y,90,90],0)
        x=x+x_speed
        y=y+y_speed
        if x>screen.get_width()-90 or x<0:
          x_speed=-x_speed
        if y>screen.get_height()-90 or y<0:
          y_speed=-y_speed
        screen.blit(my_ball,[x,y])
        pygame.display.flip()
```

Αν θέλουμε τώρα όταν η μπάλα χάνεται απλά να κάνει κύκλο τότε θα πρέπει να προσθέσουμε το παρακάτω κομμάτι κώδικα:

```
if x>screen.get_width():
   x=0
```

στο σημείο όπου ελέγχουμε αν χτυπάμε κάποιο από τα άκρα του παραθύρου στα πιο πάνω προγράμματα όπου κάνουμε αναπηδήσεις. Προσέξτε ότι πλέον δεν γίνεται αναπήδηση και έτσι η μπάλα θα πρέπει να εξαφανίζεται γι'αυτό και πάμε μέχρι width() και όχι width()-90!

Sprites: Τα sprites μας βοηθάνε στο να θυμόμαστε πόσα και ποια γραφικά αντικείμενα κινούνται στο παράθυρο μας. Είναι εξαιρετικά χρήσιμα διότι μας βοηθάνε να θυμόμαστε τι υπάρχει και κάτω από κάθε γραφικό αντικείμενο ώστε να επιδιορθώσουμε το παράθυρο σε περιπτώσεις διαγραφών.

Το pygame διαθέτει εσωτερικά το sprite module για τον χειρισμό όλων αυτών των ανεξάρτητων γραφικών αντικειμένων. Το κάθε sprite αποτελείται από δύο βασικά στοιχεία:

- Μια εικόνα: Είναι τα γραφικά που υπάρχουν στο παράθυρο.
- Ένα ορθογώνιο: Είναι το περιτύλιγμα της εικόνας ώστε να ξέρουμε τι θα βάψουμε όταν θα κάνουμε διαγραφή.

Το sprite module διαθέτει εσωτερικά τη κλάση Sprite την οποία ωστόσο δεν χρησιμοποιούμε απ' ευθείας αλλά δημιούργουμε μια υποκλάση της. Έστω λοιπόν το επόμενο πρόγραμμα:

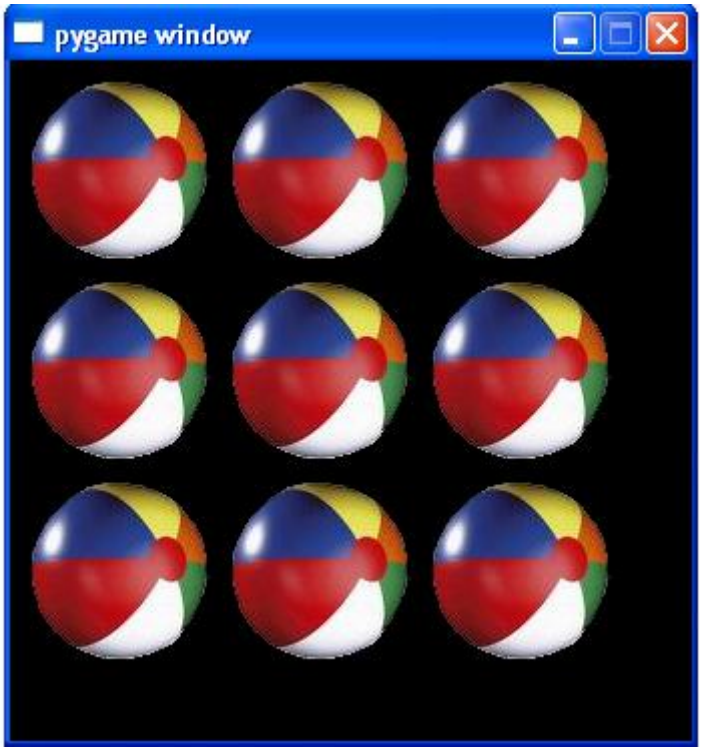
```
import pygame,sys
class Beach_Balls(pygame.sprite.Sprite):
    def __init__(self,image_file,location):
        pygame.sprite.Sprite.__init__(self) #Καλεί τον δημιουργό της υπερκλάσης
        self.image=pygame.image.load(image_file) #Φορτώνει μια εικόνα
        self.rect=self.image.get_rect() #Δημιουργεί το ορθογώνιο
        self.rect.left,self.rect.top=location #Θέτει τις αρχικές συντεταγμένες

size=width,height=340,340
screen=pygame.display.set_mode(size)
img_file="beach_ball.png"
balls=[]
for i in range(3):
   for j in range(3):
       location=[j*100+10,i*100+10]
       ball=Beach_Balls(img_file,location)
       balls.append(ball)

for ball in balls:
   screen.blit(ball.image,ball,rect)
pygame.display.flip()

while True:
       for event in pygame.event.get():
           if event.type == pygame.QUIT:
               sys.exit()
```

Έτσι, αν τρέξουμε το παραπάνω πρόγραμμα θα έχουμε ως έξοδο:



Πλέον όλες αυτές τις μπάλες μπορούμε να τις χειριστούμε μέσω της λίστας balls με έναν ενιαίο τρόπο! Τώρα, ας αρχίσουμε να κάνουμε τις μπάλες να κινούνται με μια μέθοδο της κλάσης μας:

```
def move(self):
    self.rect=self.rect.move(self.speed)
    if self.rect.left<0 or self.rect.left>width:
      self.speed[0]=-self.speed[0]
    if self.rect.top<0 or self.rect.top>height:
      self.speed[1]=-self.speed[1]
```

Εσωτερικά η κλάση Sprites έχει μια μέθοδο με όνομα move η οποία δέχεται ως παράμετρο τη ταχύτητα της μπάλας και στις δύο διαστάσεις και έτσι η ταχύτητα speed είναι στην ουσία μια λίστα δύο στοιχείων. Θα πρέπει να αλλάξουμε τον ορισμό του δημιουργού ώστε να αρχικοποιούμε και τη ταχύτητα:

```
def __init__(self,image_file,location,speed):
   pygame.sprite.Sprite.__init__(self)
   self.image=pygame.image.load(image_file)
   self.rect=self.image.get_rect()
   self.rect.left,self.rect.top=location
   seld.rect.speed=speed
```

Έτσι, αν αλλάξουμε γενικά τον κώδικα του προγράμματος όπως παρακάτω θα κινούνται οι μπάλες με τυχαία ταχύτητα η κάθε μία:

```
import pygame,sys
from random import *
class Beach_Balls(pygame.sprite.Sprite):
    def __init__(self,image_file,location,speed):
        pygame.sprite.Sprite.__init__(self)
        self.image=pygame.image.load(image_file)
        self.rect=self.image.get_rect()
        self.rect.left,self.rect.top=location
        seld.rect.speed=speed

    def move(self):
        self.rect=self.rect.move(self.speed)
        if self.rect.left<0 or self.rect.left>width:
           self.speed[0]=-self.speed[0]
        if self.rect.top<0 or self.rect.top>height:
          self.speed[1]=-self.speed[1]

size=width,height=340,340
screen=pygame.display.set_mode(size)
img_file="beach_ball.png"
balls=[]
for i in range(3):
   for j in range(3):
       location=[j*100+10,i*100+10]
       speed=[choice([-2,2]),choice([-2,2])]
       ball=Beach_Balls(img_file,location,speed)
       balls.append(ball)

for ball in balls:
   screen.blit(ball.image,ball,rect)
pygame.display.flip()

while True:
       for event in pygame.event.get():
           if event.type == pygame.QUIT:
               sys.exit()
       pygame.time.delay(20)
       screen.fill([0,0,0])
       for ball in balls:
           ball.move()
           screen.blit(ball.image, ball.rect)
           pygame.display.flip()
```

Αν τρέξουμε το παραπάνω πρόγραμμα θα δούμε ότι η μία μπάλα διαπερνάει την άλλη! Για να αποφύγουμε κάτι τέτοιο μπορούμε να κάνουμε ανίχνευση σύγκρουσης! Αντί για λίστες, στο επόμενο πρόγραμμα θα χρησιμοποιήσουμε τις ομάδες (groups) που μας δίνει το pygame και για να κάνουμε τον κώδικα μας πιο εύκολο θα σπάσουμε σε συναρτήσεις το κύριως πρόγραμμα ως εξής:

```
import pygame,sys
from random import *
class Beach_Balls(pygame.sprite.Sprite):
    #Ο ορισμός της κλάσης είναι ίδιος με πριν

def animate(group):
   screen.fill([0,0,0])
   for ball in group:
      group.remove(ball) #Αφαιρεί την μπάλα από το sprite
      if pygame.sprite.spritecollide(ball,group,False): #Έλεγχος για σύγκρουση
         ball.speed[0]=-ball.speed[0]
          ball.speed[1]=-ball.speed[1]
      group.add(ball) #Ξαναεισάγει την μπάλα στο sprite
      ball.move()
      screen.blit(ball.image, ball.rect)
   pygame.time.delay(20)
   pygame.display.flip()

size=width,height=340,340
screen=pygame.display.set_mode(size)
img_file="beach_ball.png"
group=pygame.sprite.Group() #Δημιουργία της ομάδας sprite
for i in range(3):
   for j in range(3):
       location=[j*100+10,i*100+10]
       speed=[choice([-2,2]),choice([-2,2])]
       ball=Beach_Balls(img_file,location,speed)
       group.add(ball)

while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:sys.exit()
        animate(group)
```

Το παραπάνω πρόγραμμα δημιουργεί 9 μπάλες και τις βάζει σε μια ομάδα. Κάθε φορά που υπάρχει σύγκρουση είτε με το παράθυρο είτε μεταξύ τους αλλάζει η κατεύθυνση τους.

Ωστόσο μπορείτε να παρατηρήσετε ότι υπάρχουν κάποια προβλήματα. Αυτό συμβαίνει διότι πρώτα κινείται μια μπάλα, μετά ελέγχεται για σύγκρουση με κάποια άλλη μπάλα και μετά η επόμενη μπάλα στην ομάδα κινείται κτλ. Καλό είναι πρώτα να κινούνται όλες οι μπάλες και μετά να ελέγχονται όλες οι συγκρούσεις ως εξής:

```
def animate(group):
    screen.fill([0,0,0])
    for ball in group:
        ball.move()
    for ball in group:
        group.remove(ball)
        if pygame.sprite.spritecollide(ball,group,False):
            ball.speed[0]=-ball.speed[0]
             ball.speed[1]=-ball.speed[1]
        group.add(ball)
        screen.blit(ball.image, ball.rect)
    pygame.time.delay(20)
    pygame.display.flip()
```

Μέτρηση Χρόνου: Σε όλα τα προγράμματα μας χρησιμοποιούσαμε το pygame.time.delay() για να ελέγξουμε την καθυστέρηση με την οποία θα εμφανίζεται κάθε γραφικό στην καινούργια του θέση.

Ωστόσο αυτό που μας χαλάει είναι ότι δεν ξέρουμε πόσο είναι ο συνολικός χρόνος που τρέχει ένα frame (frame είναι ένα animation=γραφικό που κινείται) ο οποίος υπολογίζεται ως το άθροισμα της χρονοκαθυστέρησης και του χρόνου που χρειάζεται το loop.

Αν υποθέσουμε ότι το loop τρέχει σε 15ms και η χρονοκαθυστέρηση μας είναι 20ms, τότε ο συνολικός χρόνος είναι 35ms. Ένα δευτερόλεπτο έχει 1000ms, άρα έχουμε 1000/35=28,57. Δηλαδή έχουμε περίπου 29 loops (ή frames) ανά δευτερόλεπτο (fps).

Αντί λοιπόν, να ελέγχουμε μόνο το ένα μέρος του χρόνου που θα κάνει ένα frame, καλύτερο είναι να γνωρίζουμε τον συνολικό, δηλαδή να τον θέτουμε εμείς! Αυτό γίνεται με την δημιουργία ενός αντικειμένου της κλάσης pygame.time.Clock() ως εξής:

```
clock=pygame.time.Clock()
clock.tick(30)
```

Το παραπάνω τμήμα κώδικα δημιουργεί ένα στιγμιότυπο της κλάσης και ορίζει θα τρέχει 30 fps.

Σχόλιο: Η κάρτα γραφικών παίζει σημαντικό ρόλο στο πόσο χρόνο θα κάνει ένα frame, έτσι τα 30 fps παίζουν για σύγχρονους επεξεργαστές και όχι για αρκετά παλιούς όπου θα ήταν στα 10 fps.

Για να είμαστε σίγουροι για το πόσα fps μπορεί να τρέξει η κάρτα γραφικών μπορούμε να θέσουμε clock.tick(200) (δηλαδή υπερβολικά γρήγορα) και στη συνέχεια με clock.get_fps() μπορούμε να μετρήσουμε τα πόσα fps πιάνει ο υπολογιστής μας για τα συγκεκριμένα γραφικά!

Αν εμείς θέλουμε να τρέχει 30 fps ωστόσο από την παραπάνω δοκιμή μας μας βγήκε 20 fps για παράδειγμα, τότε δεν χρειάζεται να συμβιβαστούμε στα 20 fps. Απλά μπορούμε να αλλάξουμε την ταχύτητα με την οποία κινείται το κάθε γραφικό μας στους άξονες (x_speed, y_speed)! Για να τη μεταβάλλουμε εκτελούμε τον εξής κώδικα:

```
speed=speed*(30/20)
```

Το παρακάτω πρόγραμμα είναι ένα παράδειγμα χρήσης του clock() για μέτρηση του χρόνου:

```
import pygame,sys
from random import *
class Beach_Balls(pygame.sprite.Sprite):
    #Ο ορισμός της κλάσης είναι ίδιος με πριν

def animate(group):
    #Ο ορισμός της συνάρτησης είναι ίδιος με πριν μόνο που λείπει η pygame.time.delay(20)

size=width,height=340,340
screen=pygame.display.set_mode(size)
img_file="beach_ball.png"
clock=pygame.time.Clock() #Δημιουργία του ρολογιού
group=pygame.sprite.Group()

for i in range(3):
   for j in range(3):
       location=[j*100+10,i*100+10]
       speed=[choice([-2,2]),choice([-2,2])]
       ball=Beach_Balls(img_file,location,speed)
       group.add(ball)

while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
               frame_rate=clock.get_fps()  #Υπολογίζει το πραγματικό fps
               print "Πραγματικό fps=",frame_rate
               sys.exit()
        animate(group)
        clock.tick(30) #Ορισμός των 30 fps
```