Notes on the ‘Introduction to interactive programming in python’ course.

Structural Programming: It is that which runs in a linear way from some point that would be the start and to the end, every bit of the program is looking for input from the user.

Event Driven Programming: It initializes and then it starts waiting, the program waits until there is some event that triggers action, it is based normally on the use of handlers.

These events sublclasify in:

-Timer events

Happen Periodically over time

-Input events

Buttons

Text Box

-Keyboard "

Key up

Key down

-Mouse "

Click

Drag

Ex.

------------------------------------------------------------------------------------------------------------------------------------------

# Example of a simple event-driven program

# CodeSkulptor GUI module, this module was created by the professors of the class for creating interactive applications in python

import simplegui

# Event handler, this event handler as its name says will handle the main event in the program, which consists in just printing 'tick!' in form of a simple function with no input

def tick():

print "tick!"

# Register handler, Here the handler of the event is registered to occur in function of a function that comes with the simplegui which sets a timer for the event

timer = simplegui.create\_timer(1000, tick)

# Start timer

timer.start()

------------------------------------------------------------------------------------------------------------------------------------------

In this particular case, the program never stops printing tick!; which happens without the need of iterations, this is what event driven programming is about. The program keeps doing what it was told to do, and just waits until there is another event, which in this case, there is not.

Event Que.

The event queue is a list of events that are in the program that hen system runs with no particular order just depending on actions taken, in the moment all events are done running the system will keep waiting indefinitely. Every event runs from a handler, just one particular handler can be run at a time.

Local Variables & Global Variables.

Ex.

------------------------------------------------------------------------------------------------------------------------------------------

# global vs local examples

# num1 is a global variable

num1 = 1

print num1

# num2 is a local variable

def fun():

num1 = 2

num2 = num1 + 1

print num2

fun()

# the scope of global num1 is the whole program, num 1 remains defined

print num1

# the scope of the variable num2 is fun(), num2 is now undefined

# print num2

# why use local variables?

# give a descriptive name to a quantity

# avoid computing something multiple times

def fahren\_to\_kelvin(fahren):

celsius = (5.0 / 9) \* (fahren - 32)

print celsius

zero\_celsius\_in\_kelvin = 273.15

return celsius + zero\_celsius\_in\_kelvin

print fahren\_to\_kelvin(212)

# the risk/reward of using global variables

# risk - consider the software system for an airliner

# critical piece - flight control system

# non-critical piece - in-flight entertainment system

# both systems might use a variable called "dial"

# we don't want possibility that change the volume on your audio

# causes the plane's flaps to change!

# example

num = 4

def fun1():

global num

num = 5

def fun2():

global num

num = 6

# note that num changes, this is because of the use of the statement global inside the function which enables to reference a variable outside the function

print num

fun1()

print num

fun2()

print num

# global variables are an easy way for event handlers

# to communicate game information.

# safer method - but they required more sophisticated

# object-programming techniques

------------------------------------------------------------------------------------------------------------------------------------------

SimpleGUI: Explanation

SimpleGui is the module destined exclusively for the purpose of being able to run completely interactive aplications directly from the browser

Ej.

------------------------------------------------------------------------------------------------------------------------------------------

import simplegui

#We create a global variable containing the string that will print out on a message

message = "Welcome!"

# Handler for mouse click event

def click():

global message

message = "Good job!"

# Handler to draw on canvas that takes as one parameter the global variable message as well as the position, size of the font and color of it

def draw(canvas):

canvas.draw\_text(message, [50,112], 36, "Red")

# Create a frame and assign callbacks to event handlers

frame = simplegui.create\_frame("Home", 300, 200)

frame.add\_button("Click me", click)

frame.set\_draw\_handler(draw)

# Start the frame animation

frame.start()

------------------------------------------------------------------------------------------------------------------------------------------

Recommended Programming Structure.

Global variables

Helper Functions

Classes

Define event handlers

Create a frame

Register event handlers

Start Frame and Timers

Ej.

------------------------------------------------------------------------------------------------------------------------------------------

# SimpleGUI program template

# Import the module

import simplegui

# Define global variables (program state)

counter=0

# Define "helper" functions

def increment():

global counter

counter=counter+1

# Define event handler functions

# First function calls helper function increment and prints the global variable counter

def tick():

increment()

print counter

# Second Function is for the secondary button that resets the counter

def button():

global counter

counter=0

# Third function is for besides printing in the console the output of the program, printing in the canvas

def text(canvas):

canvas.draw\_text(str(counter),[150,112], 36, 'Blue')

# Create a frame with all its elements, the first button which is for executing the handler function tick independant to the timer, and the first one for reseting the counter

frame=simplegui.create\_frame('simplegui test', 300, 200)

frame.add\_button('Click me', tick)

frame.add\_button('Reset', button)

# Register event handlers, the first is for the counter to be printed in the canvas, the second is for the increments of the counter to occur every second

frame.set\_draw\_handler(text)

timer= simplegui.create\_timer(1000, tick)

# Start frame and timers

frame.start()

timer.start()

------------------------------------------------------------------------------------------------------------------------------------------

Frame Operations:

simplegui.create\_frame frame.set\_canvas\_background frame.start frame.get\_canvas\_textwidth frame.add\_label frame.add\_button frame.add\_input frame.set\_keydown\_handler

frame.set\_keyup\_handler frame.set\_mouseclick\_handler frame.set\_mousedrag\_handler frame.set\_draw\_handler

With the next example we get to know how to set an input on simplegui interactive frame

Simple Calculator Ex.

----------------------------------------------------------------------------------------------------------------------------------------------------

# calculator with all buttons

import simplegui

# initialize globals

store = 0

operand = 0

# event handlers for calculator with a store and operand

def output():

"""prints contents of store and operand"""

print "Store = ", store

print "Operand = ", operand

print ""

def swap():

""" swap contents of store and operand"""

global store, operand

store, operand = operand, store

output()

def add():

""" add operand to store"""

global store

store = store + operand

output()

def sub():

""" subtract operand from store"""

global store

store = store - operand

output()

def mult():

""" multiply store by operand"""

global store

store = store \* operand

output()

def div():

""" divide store by operand"""

global store

store = store / operand

output()

def enter(t):

""" enter a new operand"""

global operand

try:

operand = int(t)

except:

operand=float(t)

output()

# create frame

f = simplegui.create\_frame("Calculator",300,300)

# register event handlers and create control elements

f.add\_button("Print", output, 100)

f.add\_button("Swap", swap, 100)

f.add\_button("Add", add, 100)

f.add\_button("Sub", sub, 100)

f.add\_button("Mult", mult, 100)

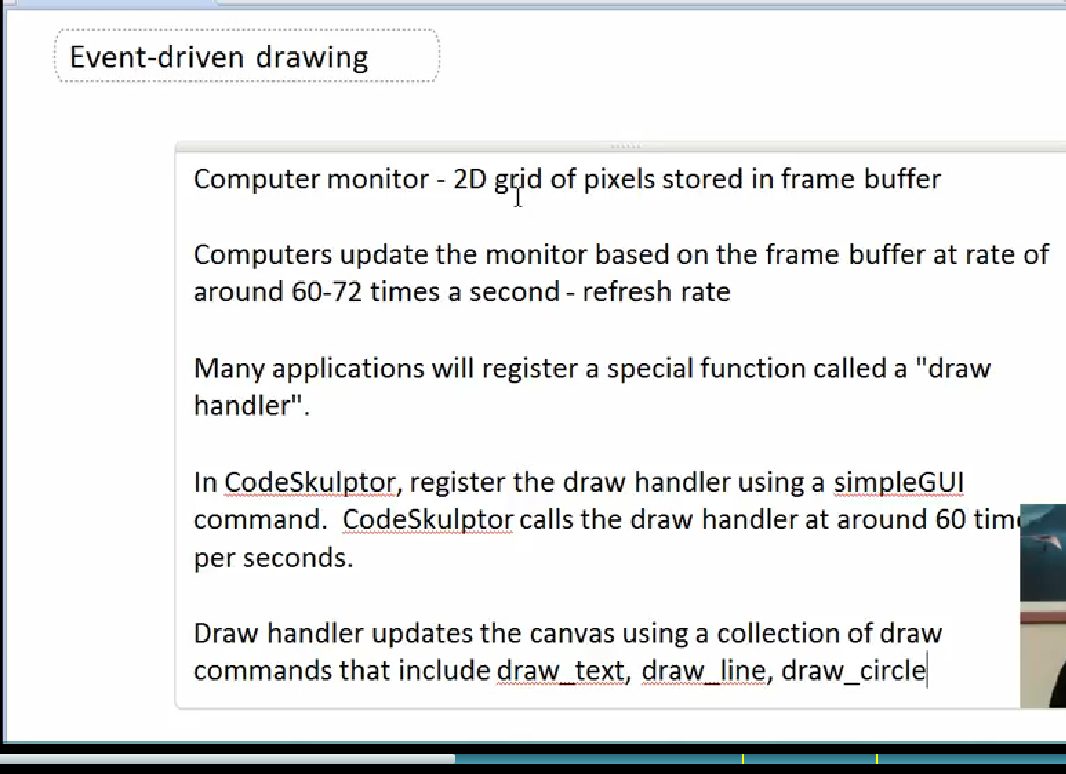
f.add\_button("Div", div, 100)

f.add\_input("Enter", enter, 100)

f.set\_canvas\_background('White')

# get frame rolling

f.start()



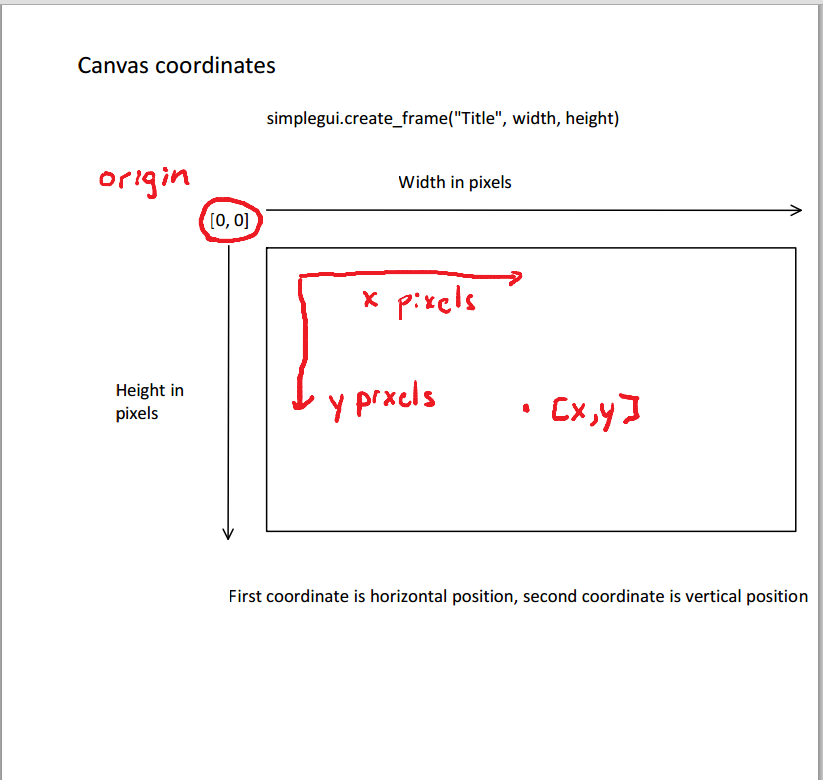
-Refresh rate is around 60 frames/sec

-Computer operating system requests that each application draw itself

-Each application has registered a special event handler called the "draw handler"

-In SimpleGUI, create and register a draw handler that draws on the canvas

-Use collection of draw operations defined in SimpleGUI



------------------------------------------------------------------------------------------------------------------------------------------

Example of a program that draws in simplegui a circle, a line, and a line of text:

# first example of drawing on the canvas

import simplegui

# define draw handler

def draw(canvas):

canvas.draw\_text("Hello!",[400, 325], 24, "White")

canvas.draw\_circle([400, 400], 30, 20, "Red")

canvas.draw\_line([400,500],[100,20],5,"Blue")

# create frame

frame = simplegui.create\_frame("Text drawing", 800, 650)

# register draw handler

frame.set\_draw\_handler(draw)

# start frame

frame.start()

Another example of canvas

# example of drawing operations in simplegui

# standard HMTL color such as "Red" and "Green"

# note later drawing operations overwrite earlier drawing operations

import simplegui

# Handler to draw on canvas

def draw(canvas):

# for the circle the parameters are: coordinates of the center[x,y], radius,

#width of the outer line,"color of the outer line","color of the inner circle"

canvas.draw\_circle([100, 100], 50, 2, "Red", "Pink")

canvas.draw\_circle([300, 300], 50, 2, "Red", "Pink")

# Line parameters: coordinates of the initial point [x,y], coordinates of the ending

# point, width of the line, 'color'

canvas.draw\_line([100, 100],[300, 300], 2, "Black")

canvas.draw\_circle([100, 300], 50, 2, "Lime", "Green")

canvas.draw\_circle([300, 100], 50, 2, "Lime", "Green")

canvas.draw\_line([100, 300],[300, 100], 2, "Black")

# Polygon parameters: [A coordinates, B coordinates, C coordinates, D coordinates],

# width of the outer lines, "color of the outer lines", "color of the interior"

canvas.draw\_polygon([[150, 150], [250, 150], [250, 250], [150, 250]], 2,

"Blue", "Aqua")

canvas.draw\_text("An example of drawing", [60, 385], 24, "Black")

# Create a frame and assign callbacks to event handlers

frame = simplegui.create\_frame("Home", 400, 400)

frame.set\_draw\_handler(draw)

frame.set\_canvas\_background("Yellow")

# Start the frame animation

frame.start()

Interactive drawing

# interactive application to convert a float in dollars and cents

import simplegui

# define global value

value = 3.12

# Handle single quantity

def convert\_units(val, name):

result = str(val) + " " + name

if val > 1:

result = result + "s"

return result

# convert xx.yy to xx dollars and yy cents

def convert(val):

# Split into dollars and cents

dollars = int(val)

cents = int(round(100 \* (val - dollars)))

# Convert to strings

dollars\_string = convert\_units(dollars, "dollar")

cents\_string = convert\_units(cents, "cent")

# return composite string

if dollars == 0 and cents == 0:

return "Broke!"

elif dollars == 0:

return cents\_string

elif cents == 0:

return dollars\_string

else:

return dollars\_string + " and " + cents\_string

# define draw handler

def draw(canvas):

canvas.draw\_text(convert(value), [60, 110], 24, "White")

# define an input field handler

def input\_handler(text):

global value

value = float(text)

# create frame

frame = simplegui.create\_frame("Converter", 400, 200)

# register event handlers

frame.set\_draw\_handler(draw)

frame.add\_input("Enter value", input\_handler, 100)

# start frame

frame.start()

Code for drawing a truck:

import simplegui

def draw(canvas):

canvas.draw\_circle([90,200], 20, 10, 'White', 'Blue')

canvas.draw\_circle([210,200], 20, 10, 'White', 'Blue')

canvas.draw\_line([50,180],[250,180], 40, 'Red')

canvas.draw\_line([55,170],[90,120], 5, 'Red')

canvas.draw\_line([90,120],[130,120],5, 'Red')

canvas.draw\_line([180,108],[180,160], 140, 'Red')

f=simplegui.create\_frame('Draw',300,300)

f.set\_draw\_handler(draw)

f.set\_canvas\_background('Yellow')

f.start()

# Simple "screensaver" program.

# Import modules

import simplegui

import random

# Global state

message = "Python is Fun!"

position = [50, 50]

width = 500

height = 500

interval = 2000

# Handler for text box

def update(text):

global message

message = text

# Handler for timer

def tick():

x = random.randrange(0, width)

y = random.randrange(0, height)

position[0] = x

position[1] = y

# Handler to draw on canvas

def draw(canvas):

canvas.draw\_text(message, position, 36, "Red")

# Create a frame

frame = simplegui.create\_frame("Home", width, height)

# Register event handlers

text = frame.add\_input("Message:", update, 150)

frame.set\_draw\_handler(draw)

timer = simplegui.create\_timer(interval, tick)

# Start the frame animation

frame.start()

timer.start()

The following is an example of changing labels in event driven programming

#####################

# Example of event-driven code, buggy version

import simplegui

size = 10

radius = 10

# Define event handlers.

def incr\_button\_handler():

"""Increment the size."""

global size

size += 1

label.set\_text("Value: " + str(size))

def decr\_button\_handler():

"""Decrement the size."""

global size

# Insert check that size > 1, to make sure it stays positive

size -= 1

if size<0:

size=size\*(-1)

elif size==0:

size=size+1

label.set\_text("Value: " + str(size))

def change\_circle\_handler():

"""Change the circle radius."""

global radius

radius = size

# Insert code to make radius label change.

labelt.set\_text("Radius: " + str(radius))

def draw(canvas):

"""Draw the circle."""

canvas.draw\_circle((100, 100), radius, 5, "Red", 'White')

# Create a frame and assign callbacks to event handlers.

frame = simplegui.create\_frame("Home", 200, 200)

# If you want a label to change you need to insert name\_of\_label.set\_text() function in each of the event #handlers

label = frame.add\_label("Value: " + str(size))

frame.add\_button("Increase", incr\_button\_handler)

frame.add\_button("Decrease", decr\_button\_handler)

labelt=frame.add\_label("Radius: " + str(radius))

frame.add\_button("Change circle", change\_circle\_handler)

frame.set\_draw\_handler(draw)

# Start the frame animation

frame.start()

Keyboard Events.

import simplegui

# initialize state

current\_key = ' '

# event handlers, these event handlers serve the purpose of indicating in the status bar if a key is #pressed of is not, the first handler handles the event of a key being pressed and changes the current #key value to a char with the chr() function just in case the key pressed is not a character which would #result in a traceback because the draw canvas can only get strings

def keydown(key):

global current\_key

current\_key = chr(key)

#the second event handler just changes back the value of current\_key to an empty string, the status bar #will print any key that was pressed and that it is now up, or not pressed

def keyup(key):

global current\_key

current\_key = ' '

#This third event handler just draws on the canvas

def draw(c):

# NOTE draw\_text now throws an error on some non-printable characters

# Since keydown event key codes do not all map directly to

# the printable character via ord(), this example now restricts

# keys to alphanumerics

if current\_key in "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789":

c.draw\_text(current\_key, [10, 25], 20, "Red")

# create frame

f = simplegui.create\_frame("Echo", 35, 35)

# register event handlers

f.set\_keydown\_handler(keydown)

f.set\_keyup\_handler(keyup)

f.set\_draw\_handler(draw)

# start frame

f.start()

# control the position of a ball using the arrow keys

import simplegui

# Initialize globals

WIDTH = 600

HEIGHT = 400

BALL\_RADIUS = 20

#Ball position at the center of the screen

ball\_pos = [WIDTH / 2, HEIGHT / 2]

# define event handlers

def draw(canvas):

canvas.draw\_circle(ball\_pos, BALL\_RADIUS, 2, "Red", "White")

# event handler for keyboard events up, down, right, left. Each time the rate of movement of the ball is #4 pixels per key down, each movement specified by either +-x or +-y

def keydown(key):

vel = 4

if key == simplegui.KEY\_MAP["left"]:

ball\_pos[0] -= vel

elif key == simplegui.KEY\_MAP["right"]:

ball\_pos[0] += vel

elif key == simplegui.KEY\_MAP["down"]:

ball\_pos[1] += vel

elif key == simplegui.KEY\_MAP["up"]:

ball\_pos[1] -= vel

# create frame

frame = simplegui.create\_frame("Positional ball control", WIDTH, HEIGHT)

# register event handlers

frame.set\_draw\_handler(draw)

frame.set\_keydown\_handler(keydown)

# start frame

frame.start()

Motion in programs.

# Ball motion with an explicit timer

import simplegui

# Initialize globals

WIDTH = 600

HEIGHT = 400

BALL\_RADIUS = 20

ball\_pos = [0,0]

init\_pos = [WIDTH / 2, HEIGHT / 2]

vel = [0, 3] # pixels per tick

time = 0

# define event handlers

def tick():

global time

time = time + 1

#Function that represents the accereration

def cronos():

vel[0]=vel[0]+1

vel[1]=vel[1]+1

def draw(canvas):

global time

global init\_pos

# calculate ball position adding the acceleration factor of a unit to the velocity vector per second

ball\_pos[0] = init\_pos[0] + time \* vel[0] + 0.5\* (time\*\*2)

ball\_pos[1] = init\_pos[1] + time \* vel[1] + 0.5\* (time\*\*2)

#the position of the ball wraps around

if ball\_pos[0]>=600 or ball\_pos[1]>=400:

time=0

ball\_pos[0]=(ball\_pos[0] + vel[0])%WIDTH

ball\_pos[1]=(ball\_pos[1] + vel[1])%HEIGHT

init\_pos=[ball\_pos[0],ball\_pos[1]]

# draw ball

canvas.draw\_circle(ball\_pos, BALL\_RADIUS, 2, "Red", "White")

# create frame

frame = simplegui.create\_frame("Motion", WIDTH, HEIGHT)

# register event handlers

frame.set\_draw\_handler(draw)

timer = simplegui.create\_timer(50, tick)

t= simplegui.create\_timer(1000, cronos)

# start frame

frame.start()

timer.start()

t.start()

# Ball motion with an implicit timer

import simplegui

# Initialize globals

WIDTH = 600

HEIGHT = 400

BALL\_RADIUS = 20

ball\_pos = [WIDTH / 2, HEIGHT / 2]

vel = [0, 1] # pixels per update (1/60 seconds)

# define event handlers

def draw(canvas):

# Update ball position

ball\_pos[0] += vel[0]

ball\_pos[1] += vel[1]

# Draw ball

canvas.draw\_circle(ball\_pos, BALL\_RADIUS, 2, "Red", "White")

# create frame

frame = simplegui.create\_frame("Motion", WIDTH, HEIGHT)

# register event handlers

frame.set\_draw\_handler(draw)

# start frame

frame.start()

Collisions and reflections.

# Ball motion with an implicit timer

import simplegui

# Initialize globals

WIDTH = 600

HEIGHT = 400

BALL\_RADIUS = 20

ball\_pos = [WIDTH / 2, HEIGHT / 2]

vel = [2, 2] # pixels per update (1/60 seconds)

# define event handlers

def draw(canvas):

# Update ball position, this position updates include the collision and reflection effect by modifying # the vertical component and maintaining the horizontal when the object collides with a horizontal #border of by modifying the horizontal component and maintaining the vertical component when the #object collides with a vertical border

if ball\_pos[0]>=WIDTH-BALL\_RADIUS-1:

vel[0]=vel[0]\*-1

elif ball\_pos[1]>=HEIGHT-BALL\_RADIUS-1:

vel[1]= vel[1]\*(-1)

elif ball\_pos[0]<=21:

vel[0]= vel[0]\*(-1)

elif ball\_pos[1]<=21:

vel[1]= vel[1]\*(-1)

ball\_pos[0] += vel[0]

ball\_pos[1] += vel[1]

# Draw ball

canvas.draw\_circle(ball\_pos, BALL\_RADIUS, 2, "Red", "White")

# create frame

frame = simplegui.create\_frame("Motion", WIDTH, HEIGHT)

# register event handlers

frame.set\_draw\_handler(draw)

# start frame

frame.start()

Velocity Control

# control the velocity of a ball using the arrow keys

import simplegui

import random

# Initialize globals

WIDTH = 600

HEIGHT = 400

BALL\_RADIUS = 20

ball\_pos = [WIDTH / 2, HEIGHT / 2]

vel = [0, 0]

# define event handlers, the conditionals will make the ball bounce of the canvas

def draw(canvas):

# Update ball position

ball\_pos[0] += vel[0]

ball\_pos[1] += vel[1]

if ball\_pos[0]>=WIDTH-BALL\_RADIUS-1:

vel[0]=-vel[0]

elif ball\_pos[1]>=HEIGHT-BALL\_RADIUS-1:

vel[1]=-vel[1]

elif ball\_pos[0]<=21:

vel[0]=-vel[0]

elif ball\_pos[1]<=21:

vel[1]=-vel[1]

# Draw ball

canvas.draw\_circle(ball\_pos, BALL\_RADIUS, 2, "Red", "White")

#each key stroke will modify the velocity vector to one direction, a little sound is added to make the ball #move more realistically and to make the control of the ball more chaotic

def keydown(key):

acc = 1

if key==simplegui.KEY\_MAP["left"]:

vel[0] -= acc-random.random()

elif key==simplegui.KEY\_MAP["right"]:

vel[0] += acc+random.random()

elif key==simplegui.KEY\_MAP["down"]:

vel[1] += acc+random.random()

elif key==simplegui.KEY\_MAP["up"]:

vel[1] -= acc-random.random()

print ball\_pos

# create frame

frame = simplegui.create\_frame("Velocity ball control", WIDTH, HEIGHT)

# register event handlers

frame.set\_draw\_handler(draw)

frame.set\_keydown\_handler(keydown)

# start frame

frame.start()

http://www.codeskulptor.org/#user38\_BlIgML7uf0\_3.py