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)

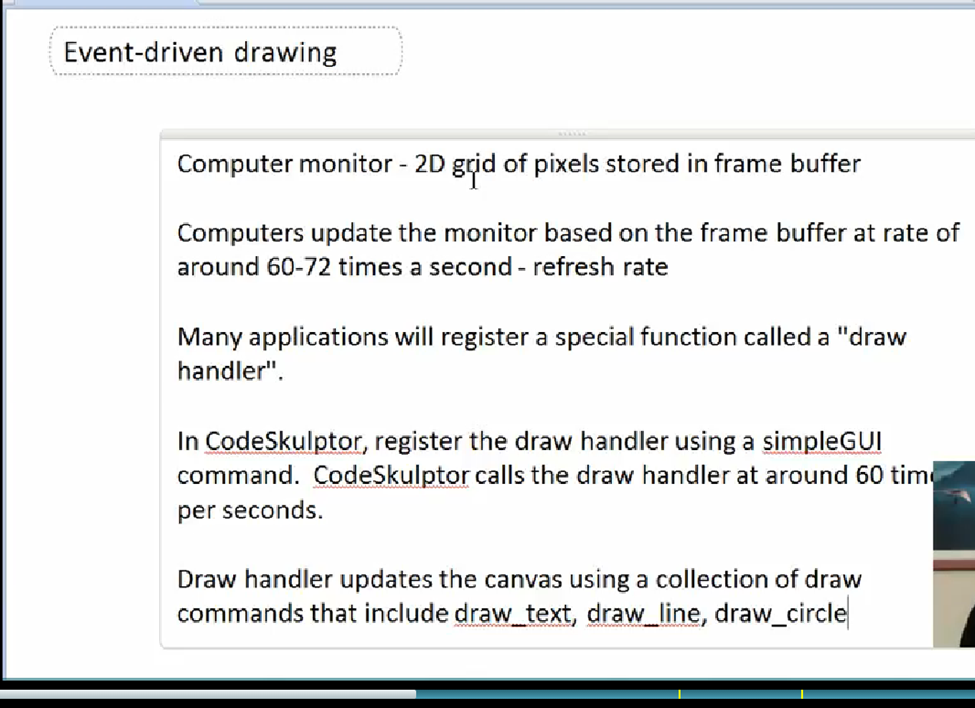
f.add\_button("Div", div, 100)

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

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

# get frame rolling

f.start()



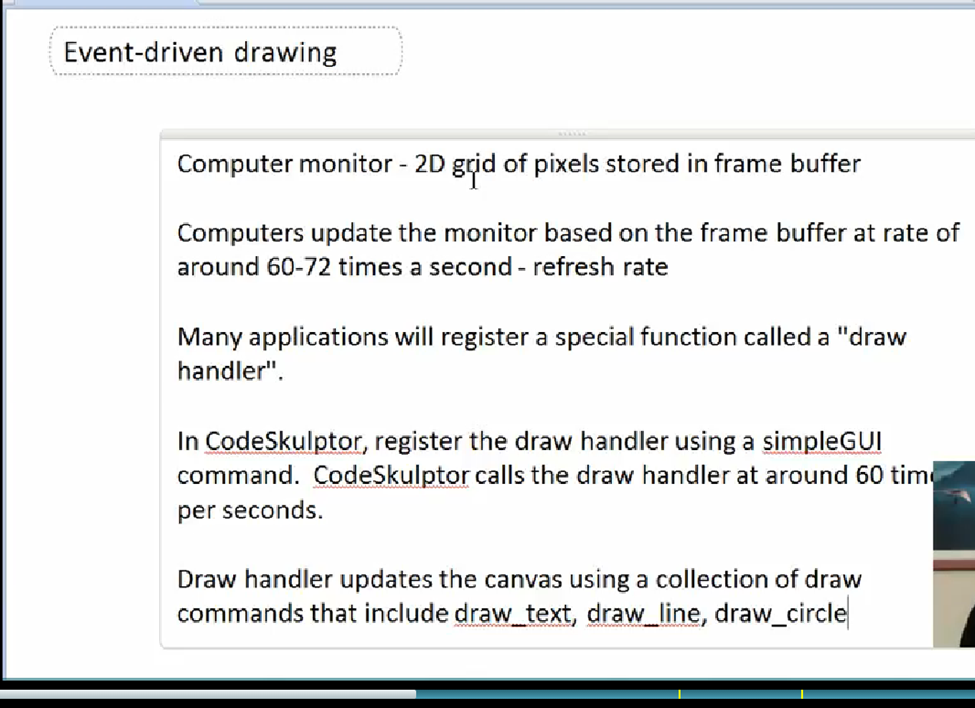
-Refresh rate is around 60 frames/sec

-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()

Mouse Input.

In order to put a mouse input that works with an interactive application say for example a simple program that indicates where in the canvas there was a click with a circle which moves (obviously) to the [x,y] coordinates where the click happened in the canvas and turning the ball green when the click was located within the radius of the circle, the appropriate mouse handler must be written, the example mentioned is as follows:

# Examples of mouse input

import simplegui

import math

# intialize globals

WIDTH = 450

HEIGHT = 300

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

BALL\_RADIUS = 15

ball\_color = "Red"

# helper function for the event in which the click is located within the radius of the circle takes a vector between the

# position of the ball and the position of the click and takes the module of that vector

def distance(p, q):

return math.sqrt( (p[0] - q[0]) \*\* 2 + (p[1] - q[1]) \*\* 2)

# define event handler for mouse click, if distance between position of the click and center of the circle is less than its # radius modify color to green, else modify the position of the circle to be the position of the click.

#It is important to note that the mouse position is a TUPLE so for each mous click a new tuple is created

def click(pos):

global ball\_pos, ball\_color

if distance(pos, ball\_pos) < BALL\_RADIUS:

ball\_color = "Green"

else:

ball\_pos = list(pos)

ball\_color = "Red"

def draw(canvas):

canvas.draw\_circle(ball\_pos, BALL\_RADIUS, 1, "Black", ball\_color)

# create frame

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

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

# register event handler

frame.set\_mouseclick\_handler(click)

frame.set\_draw\_handler(draw)

# start frame

frame.start()

List Methods.

List methods include statements such as ‘in’ and the function ‘index()’.

The ‘in’ statement serves the purpose of determining if certain element is anywhere in the list, as well as for running through the list through iterations (for) and the index function serves the purpose of telling in which place of the list the element is found which can be very useful when modifying the list.

Others list methods like the functions append() and pop().

Append(), as we know adds another element to the list.

Pop() by default takes out the last place of the list, but it takes arguments as to which of the elements of the list will be removed, it takes the place on the list in which the element in this certain place will be removed.

Example:

#Simple task list

import simplegui

tasks = []

# Handler for button

def clear():

global tasks

tasks = []

# Handler for new task

def new(task):

tasks.append(task)

# Handler for remove number

def remove\_num(tasknum):

n = int(tasknum)

if n > 0 and n <= len(tasks):

tasks.pop(n-1)

# Handler for remove name

def remove\_name(taskname):

if taskname in tasks:

tasks.remove(taskname)

# Handler to draw on canvas, for loop to run through the list and making the draw text modify the y position through # each iteration

def draw(canvas):

n = 1

for task in tasks:

pos = 30 \* n

canvas.draw\_text(str(n) + ": " + task, [5, pos], 24, "White")

n += 1

# Create a frame and assign callbacks to event handlers

frame = simplegui.create\_frame("Task List", 600, 400)

frame.add\_input("New task:", new, 200)

frame.add\_input("Remove task number:", remove\_num, 200)

frame.add\_input("Remove task:", remove\_name, 200)

frame.add\_button("Clear All", clear)

frame.set\_draw\_handler(draw)

# Start the frame animation

frame.start()

Example for creating in each click a circle in the position of the mouse

# Examples of mouse input

import simplegui

import math

# intialize globals

width = 450

height = 300

ball\_list = []

ball\_radius = 15

ball\_color = "Red"

# helper function

def distance(p, q):

return math.sqrt((p[0] - q[0]) \*\* 2 + (p[1] - q[1]) \*\* 2)

# define event handler for mouse click, draw

def click(pos):

ball\_list.append(pos)

# if distance(ball\_pos, pos) < ball\_radius:

# ball\_color = "Green"

# else:

# ball\_pos = [pos[0], pos[1]]

# ball\_color = "Red"

def draw(canvas):

for ball\_pos in ball\_list:

canvas.draw\_circle(ball\_pos, ball\_radius, 1, "Black", ball\_color)

# create frame

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

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

# register event handler

frame.set\_mouseclick\_handler(click)

frame.set\_draw\_handler(draw)

# start frame

frame.start()

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

Modified versión of the previous program that changes the color of any previously drawn circle

# Examples of mouse input

import simplegui

import math

# intialize globals

width = 450

height = 300

ball\_list = []

ball\_radius = 15

# helper function

def distance(p, q):

return math.sqrt((p[0] - q[0]) \*\* 2 + (p[1] - q[1]) \*\* 2)

# define event handler for mouse click, in each click there is going to be a

# run through each of the elements of the list that contains the ball positions

# if the distance between the click and the previous ball position is less than

# its radius it will mean that the ball has been clicked and will change color

# so the color on the main list is modified for that certain ball in some position

# if the flag isn't changed that will mean that their was a click outside the ball

# radius, so the new ball will be appended to the main list of balls

def click(pos):

changed = False

for ball in ball\_list:

if distance([ball[0], ball[1]], pos) < ball\_radius:

ball[2] = "Green"

changed = True

if not changed:

ball\_list.append([pos[0], pos[1], "Red"])

def draw(canvas):

for ball in ball\_list:

canvas.draw\_circle([ball[0], ball[1]], ball\_radius, 1, "Black", ball[2])

# create frame

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

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

# register event handler

frame.set\_mouseclick\_handler(click)

frame.set\_draw\_handler(draw)

# start frame

frame.start()

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

Modified version of the previous example but now removing each circle by clicking

# Examples of mouse input

import simplegui

import math

# intialize globals

width = 450

height = 300

ball\_list = []

ball\_radius = 15

ball\_color = "Red"

# helper function

def distance(p, q):

return math.sqrt((p[0] - q[0]) \*\* 2 + (p[1] - q[1]) \*\* 2)

# define event handler for mouse click, draw

def click(pos):

remove = []

n=0

for ball in ball\_list:

if distance(ball, pos) < ball\_radius:

remove.append(ball)

if remove == []:

ball\_list.append(pos)

else:

for ball in ball\_list:

if ball is remove[0]:

ball\_list.pop(n)

n=n+1

def draw(canvas):

for ball in ball\_list:

canvas.draw\_circle([ball[0], ball[1]], ball\_radius, 1, "Black", ball\_color)

# create frame

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

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

# register event handler

frame.set\_mouseclick\_handler(click)

frame.set\_draw\_handler(draw)

# start frame

frame.start()

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

Some other useful example functions for modifying lists bellow:

# Iterating over lists

def count\_odd(numbers):

count = 0

for num in numbers:

if num % 2 == 1:

count += 1

return count

def check\_odd(numbers):

for num in numbers:

if num % 2 == 1:

return True

return False

def remove\_odd(numbers):

for num in numbers:

if num % 2 == 1:

numbers.remove(num)

def remove\_odd2(numbers):

remove = []

for num in numbers:

if num % 2 == 1:

remove.append(numbers.index(num))

for idx in remove:

numbers.pop(idx)

def remove\_odd3(numbers):

remove = []

for num in numbers:

if num % 2 == 1:

remove.append(num)

for num in remove:

numbers.remove(num)

def remove\_odd4(numbers):

newnums = []

for num in numbers:

if num % 2 == 0:

newnums.append(num)

return newnums

def remove\_last\_odd(numbers):

n=0

last\_odd = 0

for num in numbers:

if num % 2 == 1:

last\_odd = num

for num in numbers:

if num==last\_odd:

ind=n

n=n+1

numbers=numbers.pop(ind)

def run():

numbers = [1, 7, 2, 34, 8, 7, 2, 5, 14, 22, 93, 48, 76, 15, 7]

print numbers

remove\_last\_odd(numbers)

print numbers

run()

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

Object Oriented Programming.

Each type of object has some methods to it, the built-in methods for a list object are for example methods like str\_name.append(something), or str\_name.sort() or any other method (previously called function) that is included within the type list.

As well as there are methods and types which are built-in the python language, there is the capability in the language for building your own types and methods within them.

A new type is declared in Python and in many other object oriented programming language as class, a new class will always begin with capital letter.

Example:

# for this new class we name it Character, within this new class we define it’s methods, which are no # other than functions defined with double underscores, and with the fundamental difference that this #functions will only be able to operate in objects of the same class. As we can see, each method takes at #least one parameter which is self, in this parameter we can have , the functions with no underscores #are called behaviors.

class Character:

def \_\_init\_\_(self, name, initial\_health):

self.name = name

self.health = initial\_health

self.inventory = []

def \_\_str\_\_(self):

s = "Name: " + self.name

s += " Health: " + str(self.health)

s += " Inventory: " + str(self.inventory)

return s

def grab(self, item):

self.inventory.append(item)

def get\_health(self):

return self.health

def example():

me = Character("Bob", 20)

print str(me)

me.grab("pencil")

me.grab("paper")

print str(me)

print "Health:", me.get\_health()

example()

Example of OOP Program used to simulate the movement of particles.

# Particle class example used to simulate diffusion of molecules

import simplegui

import random

# global constants

WIDTH = 600

HEIGHT = 400

PARTICLE\_RADIUS = 5

COLOR\_LIST = ["Red", "Green", "Blue", "White"]

DIRECTION\_LIST = [[1,0], [0, 1], [-1, 0], [0, -1]]

# definition of Particle class

class Particle:

# initializer for particles

def \_\_init\_\_(self, position, color):

self.position = position

self.color = color

# method that updates position of a particle

def move(self, offset):

self.position[0] += offset[0]

self.position[1] += offset[1]

# draw method for particles

def draw(self, canvas):

canvas.draw\_circle(self.position, PARTICLE\_RADIUS, 1, self.color, self.color)

# string method for particles

def \_\_str\_\_(self):

return "Particle with position = " + str(self.position) + " and color = " + self.color

# draw handler

def draw(canvas):

for p in particle\_list:

p.move(random.choice(DIRECTION\_LIST))

for p in particle\_list:

p.draw(canvas)

# create frame and register draw handler

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

frame.set\_draw\_handler(draw)

# create a list of particles

particle\_list = []

for i in range(251):

p = Particle([WIDTH / 2, HEIGHT / 2], random.choice(COLOR\_LIST))

particle\_list.append(p)

# start frame

frame.start()

When you build an OOP you must always begin by constructing all the types that you’ll need.

Example from quiz 6b (First working object type ever made by me!)

class BankAccount:

def \_\_init\_\_(self, initial\_balance):

#Creates an account with the given balance. Note that when you use a variable in a method that is going #to be used in other methods you’ll need to place the self. In front of it

self.balance=initial\_balance

self.cont=0

def deposit(self, amount):

"""Deposits the amount into the account."""

self.balance=amount+self.balance

def withdraw(self, amount):

"""

Withdraws the amount from the account. Each withdrawal resulting in a

negative balance also deducts a penalty fee of 5 dollars from the balance.

"""

self.balance=self.balance-amount

if self.balance<0:

self.balance=self.balance-5

self.cont=self.cont+5

def get\_balance(self):

"""Returns the current balance in the account."""

return self.balance

def get\_fees(self):

"""Returns the total fees ever deducted from the account."""

return self.cont

The power of OOP Relies on the fact that when you make any object a part of a class Python will keep track of any indefinite number of objects and its values, just as it would precisely do with the built in Types of objects such as lists or dictionaries and that is exactly how all new types of objects making code recycling much more easier.

* Class overload: in some OOP languages exists the capability of creating various \_\_init\_\_ methods which will respond to the class assignation in relation to the amount of parameters that the methods are capable of handling. In python this capability does not exist.

For example:

class Overload:

def \_\_init\_\_(self, value1):

pass

def \_\_init\_\_(self, value1, value2):

pass

value1=Overload(1)

value2=Overload(2,1)

In this previous example Python would give us a Type error it would only be able to take the first init value. But Python is flexible with methods, so if you need to use various parameters in a same method you would start the parameter in a predetermined value and if you wish you can use that parameter when needed. As in the following example.

class Overload:

def \_\_init\_\_(self, one, two=0):

"""Example of method that takes one required argument and one optional argument."""

pass

Overload(1) # Implicitly, we leave the second argument as its default value, 0.

Overload(1,2)

This happens too in built in methods for types just like range() which can be used in this way: range(100) and creates a sequenced list of the numbers 0 to 99 or in this other way: range(50,100) which creates a list from 50 to 99, that first parameter of the method is set to zero.

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

Tiled Images.

To add tiled images and be able to use them:

# demo for drawing using tiled images

import simplegui

import random

# define globals for cards

RANKS = ('A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K')

SUITS = ('C', 'S', 'H', 'D')

r=random.choice(RANKS)

s=random.choice(SUITS)

# card sprite - 950x392

CARD\_CENTER = (36.5, 49)

CARD\_SIZE = (73, 98)

#Load Image

card\_image = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/cards.jfitz.png")

#LOC is location in which the image will be drawn in the image

LOC= (155,90)

#Lists for defining the domain in which a mouse click which will be the area that encloses the card

card\_loc = list(LOC)

card\_size = list(CARD\_SIZE)

half\_card\_size= [card\_size[0]/2 , card\_size[1]/2]

#List to store the initial range of the card approximating ‘x’ and ‘y’ from the left

CARD\_RANGE= [card\_loc[0]-half\_card\_size[0], card\_loc[1]-half\_card\_size[1]]

click=False

# define card class

class Card:

def \_\_init\_\_(self, suit, rank):

self.rank = rank

self.suit = suit

def draw(self, canvas, loc):

i = RANKS.index(self.rank)

j = SUITS.index(self.suit)

card\_pos = [CARD\_CENTER[0] + i \* CARD\_SIZE[0],

CARD\_CENTER[1] + j \* CARD\_SIZE[1]]

canvas.draw\_image(card\_image, card\_pos, CARD\_SIZE, loc, CARD\_SIZE)

# define draw handler

def draw(canvas):

if click is True:

one\_card = Card(s,r)

one\_card.draw(canvas, LOC)

def click(pos):

global r, s, click

click= True

position=pos

if position[0]>=CARD\_RANGE[0] and position[0]<=CARD\_RANGE[0]+card\_size[0]:

if position[1]>=CARD\_RANGE[1] and position[1]<=CARD\_RANGE[1]+card\_size[1]:

r=random.choice(RANKS)

s=random.choice(SUITS)

one\_card = Card(s,r)

# define frame and register draw handler

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

frame.set\_draw\_handler(draw)

frame.set\_mouseclick\_handler(click)

# create a card

frame.start()

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

Visualizing objects.

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

# Object creation and use

# Mutation with Aliasing

# This particular example illustrates how aliasing also works with object instances in which an object p is # created within the type Point1 and then r is not created but instead entangled within the same #instance of the type p so basically r and p are the same object.

class Point1:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def set\_x(self, newx):

self.x = newx

def get\_x(self):

return self.x

p = Point1(4, 5)

q = Point1(4, 5)

r = p

p.set\_x(10)

print p.get\_x()

print q.get\_x()

print r.get\_x()

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

# Object shared state

# Mutation of shared state

# in this example a list is created, this same list will be modified in the class so the objects p and q which # are created with the coordinates list will be sharing the same instance of the class thus if the list #coordinates is changed within the class, which will happen, both objects will be changed because they #share the same list

class Point2:

def \_\_init\_\_(self, coordinates):

self.coords = coordinates

def set\_coord(self, index, value):

self.coords[index] = value

def get\_coord(self, index):

return self.coords[index]

coordinates = [4, 5]

p = Point2(coordinates)

q = Point2(coordinates)

r = Point2([4, 5])

p.set\_coord(0, 10)

print p.get\_coord(0)

print q.get\_coord(0)

print r.get\_coord(0)

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

# Objects not sharing state

# By simply creating a copy of whatever list is passed to the class in order to create an object within the # class you avoid the previous situation from happening.

class Point3:

def \_\_init\_\_(self, coordinates):

self.coords = list(coordinates)

def set\_coord(self, index, value):

self.coords[index] = value

def get\_coord(self, index):

return self.coords[index]

coordinates = [4, 5]

p = Point3(coordinates)

q = Point3(coordinates)

r = Point3([4, 5])

p.set\_coord(0, 10)

print p.get\_coord(0)

print q.get\_coord(0)

print r.get\_coord(0)

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

Building Spaceship.

# Partial example code for Spaceship

# The following is an example for two of the classes that will be used for building spaceship

import simplegui

class ImageInfo:

def \_\_init\_\_(self, center, size, radius = 0, lifespan = None, animated = False):

self.center = center

self.size = size

self.radius = radius

if lifespan:

self.lifespan = lifespan

else:

self.lifespan = float('inf')

self.animated = animated

def get\_center(self):

return self.center

def get\_size(self):

return self.size

def get\_radius(self):

return self.radius

def get\_lifespan(self):

return self.lifespan

def get\_animated(self):

return self.animated

# art assets created by Kim Lathrop, may be freely re-used in non-commercial projects, please credit Kim

# ship image

ship\_info = ImageInfo([45, 45], [90, 90], 35)

ship\_image = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/lathrop/double\_ship.png")

# sound assets purchased from sounddogs.com, please do not redistribute

ship\_thrust\_sound = simplegui.load\_sound("http://commondatastorage.googleapis.com/codeskulptor-assets/sounddogs/thrust.mp3")

# Ship class

class Ship:

def \_\_init\_\_(self, pos, vel, angle, image, info):

self.pos = [pos[0],pos[1]]

self.vel = [vel[0],vel[1]]

self.thrust = False

self.angle = angle

self.angle\_vel = 0

self.image = image

self.image\_center = info.get\_center()

self.image\_size = info.get\_size()

self.radius = info.get\_radius()

def draw(self,canvas):

canvas.draw\_circle(self.pos, self.radius, 1, "White", "White")

def update(self):

pass

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

Set Sounds.

# simple music player, uses buttons and sounds

# note that .ogg sounds are not supported in Safari

import simplegui

# define callbacks

def play():

"""play some music, starts at last paused spot"""

music.play()

def pause():

"""pause the music"""

music.pause()

def rewind():

"""rewind the music to the beginning """

music.rewind()

def laugh():

"""play an evil laugh

will overlap since it is separate sound object"""

laugh.play()

def vol\_down():

"""turn the current volume down"""

global vol

if vol > 0:

vol = vol - 1

music.set\_volume(vol / 10.0)

volume\_button.set\_text("Volume = " + str(vol))

def vol\_up():

"""turn the current volume up"""

global vol

if vol < 10:

vol = vol + 1

music.set\_volume(vol / 10.0)

volume\_button.set\_text("Volume = " + str(vol))

# create frame - canvas will be blank

frame = simplegui.create\_frame("Music demo", 250, 250, 100)

# set up control elements

frame.add\_button("play", play,100)

frame.add\_button("pause", pause,100)

frame.add\_button("rewind",rewind,100)

frame.add\_button("laugh",laugh,100)

frame.add\_button("Vol down", vol\_down,100)

frame.add\_button("Vol up", vol\_up,100)

# initialize volume, create button whose label will display the volume

vol = 7

volume\_button = frame.add\_label("Volume = " + str(vol))

# load some sounds

music = simplegui.load\_sound("http://commondatastorage.googleapis.com/codeskulptor-assets/Epoq-Lepidoptera.ogg")

laugh = simplegui.load\_sound("http://commondatastorage.googleapis.com/codeskulptor-assets/Evillaugh.ogg")

# make the laugh quieter so my ears don't bleed

laugh.set\_volume(.1)

frame.start()

Terminal Velocity model for the ship:

acc=10.0

fr=acc\*.1

vel=0

t=1

while True:

if t==1:

vel=acc+vel

t=t+1

continue

if acc>.000000000001:

acc=acc-fr

fr=acc\*.1

vel=acc+vel

else:

break

print acc

print vel

print 'terminal velocity is aprox: ' , vel

Modification of the bouncing ball that makes the ball appear in the other side of the frame via modular math.

def draw(canvas):

# Update ball position

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

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

if ball\_pos[0]>WIDTH:

ball\_pos[0]=ball\_pos[0]%WIDTH

elif ball\_pos[1]>HEIGHT:

ball\_pos[1]=ball\_pos[1]%HEIGHT

elif ball\_pos[0]<0:

ball\_pos[0]=ball\_pos[0]%WIDTH + WIDTH

elif ball\_pos[1]<0:

ball\_pos[1]=ball\_pos[1]%HEIGHT + HEIGHT

# Draw ball

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

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

Sprites.

Two dimensional image or animation integrated into a larger scene, usually treated as a graphical overlay

in 1970/1980's, doing 2D graphics was computationally expensive. Sprites were 2D images provided to

special hardware accelerators that overlaid the images onto the display.

Now, sprites are logical entities used to organize/represent images that add visual complexity to a game

Sprite sheets consisted a collection of sprites organized as a single image. Note that the individual sprites need not be regularly spaced on the sprite sheet.

We will prefer to load sprites as individual images to provide more flexibility in modifying the art assets for Spaceship and RiceRocks.

Color and Transparency

RGB model - three red, green, blue channels

Stored channel values as numerical intensities in the range 0-255

HTML string - "rgb(255, 0, 0)" - equivalent to "Red"

http://www.w3schools.com/html/html\_colors.asp

Color - up to now, "White", "Black", "Red"

Challenge- would like to draw irregular shapes (like an asteroid or spaceship) that lie in

rectangular images

Add alpha channel to RGB model - channel stores transparency

HTML string - "rgba(255, 0, 0, 0.5)" - (1 is opaque, 0 is transparent)

Create image with transparent alpha channel in Photoshop, GIMP, paint.net, etc.

PNG image format is popular choice

Transparency - up to now, always opaque

Sprite Class.

# Sprite class emo

import simplegui

import math

# helper class to organize image information

class ImageInfo:

def \_\_init\_\_(self, center, size, radius = 0, lifespan = None, animated = False):

self.center = center

self.size = size

self.radius = radius

if lifespan:

self.lifespan = lifespan

else:

self.lifespan = float('inf')

self.animated = animated

def get\_center(self):

return self.center

def get\_size(self):

return self.size

def get\_radius(self):

return self.radius

def get\_lifespan(self):

return self.lifespan

def get\_animated(self):

return self.animated

# load ship image

asteroid\_info = ImageInfo([45, 45], [90, 90], 40)

asteroid\_image = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/lathrop/asteroid\_blue.png")

# Sprite class

class Sprite():

def \_\_init\_\_(self, pos, vel, ang, ang\_vel, image, info, sound = None):

self.pos = [pos[0],pos[1]]

self.vel = [vel[0],vel[1]]

self.angle = ang

self.angle\_vel = ang\_vel

self.image = image

self.image\_center = info.get\_center()

self.image\_size = info.get\_size()

self.radius = info.get\_radius()

self.lifespan = info.get\_lifespan()

self.animated = info.get\_animated()

self.age = 0

if sound:

sound.rewind()

sound.play()

def draw(self, canvas):

#canvas.draw\_circle(self.pos, self.radius, 1, "Red", "Red")

canvas.draw\_image(self.image, self.image\_center, self.image\_size, self.pos, self.image\_size, self.angle)

def update(self):

self.angle += self.angle\_vel

self.pos[0] += self.vel[0]

self.pos[1] += self.vel[1]

def draw(canvas):

# draw ship and sprites

a\_rock.draw(canvas)

# update ship and sprites

a\_rock.update()

# initialize frame

frame = simplegui.create\_frame("Sprite demo", 800, 600)

# initialize ship and two sprites

a\_rock = Sprite([400, 300], [0.3, 0.4], 0, 0.1, asteroid\_image, asteroid\_info)

# register handlers

frame.set\_draw\_handler(draw)

# get things rolling

frame.start()

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

Examples of Code repetition and ways to avoid it.

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

# Incomplete code from Pong

# Repeated code

def draw(c):

global paddle1\_pos, paddle2\_pos

paddle\_width = 80

if paddle\_width/2 <= paddle1\_pos + paddle1\_vel <= width - paddle\_width/2:

paddle1\_pos += paddle1\_vel

if paddle\_width/2 <= paddle2\_pos + paddle2\_vel <= width - paddle\_width/2:

paddle2\_pos += paddle2\_vel

c.draw\_line([width/2, 0],[width/2, height], 1, "White")

c.draw\_line([4, paddle1\_pos-paddle\_width/2], [4, paddle1\_pos+paddle\_width/2], 4, "White")

c.draw\_line([width-4, paddle2\_pos-paddle\_width/2], [width-4, paddle2\_pos+paddle\_width/2], 4, "White")

...

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

# Incomplete code from Pong

# Avoiding repetition with functions

paddle\_width = 80

def paddle\_move(paddle\_num):

if paddle\_width/2 <= paddle\_pos[paddle\_num] + paddle\_vel[paddle\_num] <= width - paddle\_width/2:

paddle\_pos[paddle\_num] += paddle\_vel[paddle\_num]

def paddle\_draw(c, paddle\_num):

c.draw\_line([paddle\_loc[paddle\_num], paddle\_pos[paddle\_num] - paddle\_width/2],

PADDLE\_THICKNESS, "White")

def draw(c):

paddle\_move(0)

paddle\_move(1)

c.draw\_line([width / 2, 0],[width / 2, height], 1, "White")

paddle\_draw(c,0)

paddle\_draw(c,1)

...

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

# Incomplete code from Pong

# Avoiding repetition with classes and methods

class Paddle:

def \_\_init\_\_(self, loc, pos, vel):

self.loc = loc

self.pos = pos

self.vel = vel

self.width = 80

def move(self):

if self.width/2 <= self.pos + self.vel <= width - self.width/2:

self.pos += self.vel

def draw(c, self):

c.draw\_line([self.loc, self.pos-self.width / 2], PADDLE\_THICKNESS, "White")

def draw(c):

paddle1.move()

paddle2.move()

c.draw\_line([width / 2, 0],[width / 2, height], 1, "White")

paddle1.draw(c)

paddle2.draw(c)

...

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

# Incomplete code from Pong

# Long if/elif chain

def keydown(key):

global paddle1\_vel, paddle2\_vel

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

paddle2\_vel -= 2

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

paddle2\_vel += 2

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

paddle1\_vel -= 2

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

paddle1\_vel += 2

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

# Incomplete code from Pong

# Avoiding long if/elif chain with dictionary mapping values to actions

def paddle1\_faster():

global paddle1\_vel

paddle1\_vel += 2

def paddle1\_slower():

global paddle1\_vel

paddle1\_vel -= 2

def paddle2\_faster():

global paddle2\_vel

paddle2\_vel += 2

def paddle2\_slower():

global paddle2\_vel

paddle2\_vel -= 2

inputs = {"up": paddle2\_slower,

"down": paddle2\_faster,

"w": paddle1\_slower,

"s": paddle1\_faster}

def keydown(key):

for i in inputs:

if key == simplegui.KEY\_MAP[i]:

inputs[i]()

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

# Illustration of a dictionary mapping values to functions

def f():

print "hi"

d = {0: f}

d[0]()

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

# Incomplete code from Pong

# Avoiding long if/elif chain with dictionary mapping values to action arguments

inputs = {"up": [1, -2],

"down": [1, 2],

"w": [0, -2],

"s": [0, 2]}

def keydown(key):

for i in inputs:

if key == simplegui.KEY\_MAP[i]:

paddle\_vel[inputs[i][0]] += inputs[i][1]

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

# Sample of using tiled image

# Has some uses of "magic" (unexplained) constants.

# demo for drawing using tiled images

import simplegui

# define globals for cards

RANKS = ['A', '2', '3', '4', '5', '6', '7', '8', '9', 'T', 'J', 'Q', 'K']

SUITS = ['C', 'S', 'H', 'D']

# card sprite - 950x392

CARD\_CENTER = (36.5, 49)

CARD\_SIZE = (73, 98)

card\_image = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/cards.jfitz.png")

# define card class

class Card:

def \_\_init\_\_(self, suit, rank):

self.rank = rank

self.suit = suit

def draw(self, canvas, pos):

i = RANKS.index(self.rank)

j = SUITS.index(self.suit)

card\_pos = [CARD\_CENTER[0] + i \* CARD\_SIZE[0],

CARD\_CENTER[1] + j \* CARD\_SIZE[1]]

canvas.draw\_image(card\_image, card\_pos, CARD\_SIZE, pos, CARD\_SIZE)

# define draw handler

def draw(canvas):

one\_card.draw(canvas, (155, 90))

# define frame and register draw handler

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

frame.set\_draw\_handler(draw)

# createa card

one\_card = Card('S', '6')

frame.start()

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

# Sample of using tiled image

# Naming constants and calculating other constants from those

# demo for drawing using tiled images

import simplegui

# define globals for cards

RANKS = ['A', '2', '3', '4', '5', '6', '7', '8', '9', 'T', 'J', 'Q', 'K']

SUITS = ['C', 'S', 'H', 'D']

# card sprite - 950x392

CARD\_SIZE = (73, 98)

card\_image = simplegui.load\_image("http://commondatastorage.googleapis.com/codeskulptor-assets/cards.jfitz.png")

FRAME\_SIZE = (300, 200)

# define card class

class Card:

def \_\_init\_\_(self, suit, rank):

self.rank = rank

self.suit = suit

def draw(self, canvas, pos):

i = RANKS.index(self.rank)

j = SUITS.index(self.suit)

card\_loc = [(.5 + i) \* CARD\_SIZE[0],

(.5 + j) \* CARD\_SIZE[1]]

canvas.draw\_image(card\_image, card\_loc, CARD\_SIZE, pos, CARD\_SIZE)

# define draw handler

def draw(canvas):

one\_card.draw(canvas, (FRAME\_SIZE[0] / 2, FRAME\_SIZE[1] / 2))

# define frame and register draw handler

frame = simplegui.create\_frame("Card draw", FRAME\_SIZE[0], FRAME\_SIZE[1])

frame.set\_draw\_handler(draw)

# createa card

one\_card = Card('S', '6')

frame.start()

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

# Incomplete code from Pong

# Magic unnamed constants, repeated code, long expressions

width = 600

height = 400

def ball\_init():

if random.randrange(0,2) == 0:

return [300,200], [3 + 3 \* random.random(), 8 \* (random.random() - 0.5)]

else:

return [300,200], [-(3 + 3 \* random.random()), 8 \* (random.random() - 0.5)]

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

# Incomplete code from Pong

# Constants named and computed, repetition avoided, expressions broken into

# named pieces

width = 600

height = 400

def ball\_init():

pos = [width/2, height/2]

vel\_x = 3 + 3 \* random.random()

vel\_y = 8 \* (random.random() - 0.5)

if random.randrange(0,2) == 1:

vel\_x = -vel\_x

return pos, [vel\_x, vel\_y]

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

Sets.

A set is a data type that stores mutliple values with no order and doesn’t allow duplicates but it does allow modifications.

# Examples of Sets

instructors = set(['Rixner', 'Warren', 'Greiner', 'Wong'])

print instructors

inst2 = set(['Rixner', 'Rixner', 'Warren', 'Warren', 'Greiner', 'Wong'])

print inst2

print instructors == inst2

for inst in instructors:

print inst

instructors.add('Colbert')

print instructors

instructors.add('Rixner')

print instructors

instructors.remove('Wong')

print instructors

#instructors.remove('Wong')

#print instructors

print 'Rixner' in instructors

print 'Wong' in instructors

Output of the program.

set(['Rixner', 'Warren', 'Greiner', 'Wong'])

set(['Rixner', 'Warren', 'Greiner', 'Wong'])

True

Rixner

Warren

Greiner

Wong

set(['Rixner', 'Warren', 'Greiner', 'Wong', 'Colbert'])

set(['Rixner', 'Warren', 'Greiner', 'Wong', 'Colbert'])

set(['Rixner', 'Warren', 'Greiner', 'Colbert'])

True

False

# Examples of Sets 2 using the set1.difference\_update(set2) method for removing from the all elements # in set2 that belong to set1

instructors = set(['Rixner', 'Warren', 'Greiner', 'Wong'])

print instructors

def get\_rid\_of(inst\_set, starting\_letter):

remove\_set = set([])

for inst in inst\_set:

if inst[0] == starting\_letter:

remove\_set.add(inst)

inst\_set.difference\_update(remove\_set)

get\_rid\_of(instructors, 'W')

print instructors

Output of the function.

set(['Rixner', 'Warren', 'Greiner', 'Wong'])

set(['Rixner', 'Greiner'])