Intro to programming II

http://www.codeskulptor.org/#user40_bTGbTBPjOoM6s4G.py
http://www.codeskulptor.org/#user40_o7bBVOT4q6LAXbY.py

Week 3

Event Driven Programming (EDP)

In EDP we set the stage of how the program needs to reacts to events and then enter a loop that waits until there is event to handle. This type of programs follow this template:

```
import simplegui

# global variables
counter = 0
start_value = 0

# helper functions
def increment():
    global counter
    counter = counter + 1

# event handlers
def tick():
    increment()
    print(counter)

def button_press():
    global counter
    counter = start_value

def set_start_value(value):
    global start_value
    start_value = int(value)
    button_press()

# create a frame
frame = simplegui.create_frame("test",
100, 150)

# handler registration
timer = simplegui.create_timer(1000, tick)
frame.add_button('Reset', button_press)
fld = frame.add_input('start value',
set_start_value, 50)
fld.set_text( str(start_value) )

# start frame and timers
frame.start()
timer.start()
```

Local variables

They are created when a function is called, they exist while the function runs, and are deleted when the function terminates. They can only be used inside the function. Variables received as arguments are local:

```
def foo(local_1, local_2):
    local_3 = " and "
    print( local_1 + local_3 + local_2)

foo("Spongebob", "Patrick")
```

Global variables

They exist in the 'main' program, outside any function. Once they are created they exist until the program finishes; all functions can see them:

```
global_1 = "Mickey"
global_2 = "Daisy"
global_3 = "Pluto"

def foo():
    global global_2, global_4
    global_1 = "Donald"   # this is a local
    global_2 = "Minnie"   # this is a global
    global_4 = global_3

foo()
msg = global_1 + ", " + global_2
msg = msg + ' and ' + global_4
print(msg)
```

We use the keyword `global` inside a function to modify a global that already exists (e.g., `global_2`), or to create a new global (e.g., `global_4`).

All functions can read the value of a global, e.g., we can set `global_4` to `global_3` without pointing out that global_3 is a global. If we try to modify a global inside a function without declaring it as such, Python assumes that what we are doing is creating a local variable that happens to have the same name as a global, e.g., `global_1`.