ETGG3802	Lab2: **Python/C GameObject "class"**	Points: **100 points**
Assigned: **1/19/2016**	Due: **8:00am on 1/29/2016** (we might do other things in class – don't procrastinate!)	

Overview:

- This is (almost surely) that last individual lab. After this, we'll move on to group work!
- In this lab, we'll make a Python / C "wrapper" around our GameObject class.
- We will then replace the SimplePlayer (C++) class with a custom script in which we create a ssurge.GameObject-derived class that overrides a few select methods.
- In the group project portion of the class, we'll expand the functionality of the Python/C GameObject class.

Tasks:

1. Start by reading through https://docs.python.org/3.5/extending/newtypes.html, especially section 2.1 – 2.1.1 (you may have already read parts of this in ETGG3801)
    - You might be able to use your / my solution to Lab19 from ETGG3801 (esp. the "testClass" portions) as a guide.
2. (**30 points**) Make a (python/C) ssurge.GameObject class that mimics our C++ GameObject class functionality
    - The user should be able to re-define base-class methods.
    - See the test program below for an idea of *what* methods need to appear in the GameObject class.
    - You should ensure that the C++ and Python objects both have access to the corresponding twin (in the other 'universe' (Python/C <-> C++)
        - "Python Capsules" are the way I'll implement the Python/C => C++ connection
        - Just store a PyObject * for the C++ => Python/C connection
3. (**40 points**) Add a top-level ssurge function called createGameObject. Also, make sure that if the C++ GameObject is destroyed, its script "twin" is also destroyed.
4. (**30 points**) Add the minimal number of methods / attributes to get the output shown in the next task
5. At this point, you can test your game object by calling something like this (it also shows the minimal ssurge.GameObject methods I'm expecting you to implement)

```
import ssurge

ssurge.log("str(dir(ssurge)) =\n" + str(dir(ssurge)))
# ['GameObject', '__doc__', '__loader__', '__name__', '__package__', '__spec__',
#     'createGameObject', 'getActionDown', 'getAxis', 'log', 'setBackgroundColor']

x = ssurge.createGameObject("blah", 15)
ssurge.log(str(dir(x)))
# ['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
#  '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__', '__ne__',
#  '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
#  '__str__', '__subclasshook__', 'loadMesh', 'mCapsule', 'rotateWorld', 'scale',
#  'translateWorld']

x.loadMesh("ninja.mesh")                # Creates a mesh component
x.translateWorld(0, 0, 0)
x.scale(0.05, 0.05, 0.05)
x.rotateWorld(90.0, 1,0,0)              # 90 degrees around the x-axis
```

6. (**+10 points**) Create a Python / C dictionary of all game object *instances*. Also, add a top-level Python / C ssurge function to retrieve a game object by name.
7. (**+30 points**) Callbacks
    - [You don't need to do anything to allow this – it happens automatically from what we've done up to this point] Create a class (in Python) to replace the SimplePlayer (C++) class. You can use something like this:

```
# simple_player.py
import ssurge

class SimplePlayer(ssurge.GameObject):
    def initialize(self, args):
        self.mSpeed = 2.0
        self.mDirection = [0, -1, 0]
        self.mRotationSpeed = 120.0
        self.loadMesh("ninja.mesh")
```

```
def update(self, dt):
      offset = [ssurge.getAxis(0) * dt * self.mSpeed, \
                ssurge.getAxis(1) * dt * self.mSpeed, 0]
      self.translateWorld(offset[0], offset[1], offset[0])
      self.pointTowards(offset[0], offset[1], offset[0])
```

- Now, include a mechanism (in the GOM) to retrieve a reference to a ssurge.GameObject-derived (Python/C) class and create an instance. For example:

```
<node name="SimplePlayer" …>
      <script src="simple_player.py">
</node>
```

- In the C++ update of a game object, if that GameObject is “script-aware”, call it’s (Python) update method (if it’s overloaded that method, as we did in the above script) and pass it dt.
- Remove the SimplePlayer class from our C++ project. You should now see the ninja instead of the monkey head – it should move according to player input (it won’t do the turning-towards…yet).