New York University
School of Continuing and Professional Studies
Division of Programs in Information Technology

Introduction to Python
Exercises, Session 10

Ex. 10.1 Create a class called ThisClass with the statement class ThisClass(object): and create one method inside the class with the statement def report(self): Inside this method, the instance/object (which is labeled self) should call the special id() function to report its own reference id, i.e. print id(self). Create three instances using the constructor (for example, a = ThisClass()) and then call the report() method on each of them.

Lastly, print id() on each of your objects in the calling code, for example print id(a). Note that the id numbers are the same as those found when calling report().

Expected calls and output:

```
a = ThisClass()
b = ThisClass()
c = ThisClass()

a.report()          # 4299790736  [your ids will of
                    #              course differ]
b.report()          # 4299790544
c.report()          # 4299790800
print()               # [blank line]

print(id(a))         # 4299790736   (same as a.report() above)
print(id(b))         # 4299790544   (same as b.report() above)
print(id(c))         # 4299790800   (same as c.report() above)
```

Ex. 10.2 Create a class, TimeStamp(object): that can store the current timestamp in an instance attribute.

set_time(self): will set the timestamp. It can do this by setting the attribute in self this way (you will need to import datetime at the top of your script containing class TimeStamp):

```
self.t = str(datetime.datetime.now())
```

where t is the attribute label (you could call it whatever you prefer). (Of course, you'll also need to import datetime at the top of your module.)

get_time(self): this returns the timestamp. It simply returns the object attribute time, i.e. return self.t.

Expected calls and output (your timestamp will be different of course, but note when the object time is repeated and when it is different):

```
var1 = TimeStamp()
var2 = TimeStamp()
var1.set_time()
var2.set_time()
print(var1.get_time())    # 2013-04-07 15:19:31.762220
print(var2.get_time())    # 2013-04-07 15:19:31.956881
print()                     # [blank line]
var1.set_time()           # change timestamp for var1
print(var1.get_time())    # 2013-04-07 15:19:31.956941
                          # (diff from var1 above)
print(var2.get_time())    # 2013-04-07 15:19:31.956881
                          # (same as var2 above)
```

Ex. 10.3 Copy the above code, and this time replace the set_time() method with the constructor, __init__(self), which does the same work - sets the attribute to the current timestamp. Now the method has only an __init__(self) method and a get_time(self) method which returns the timestamp.

Expected calls and output (your timestamp will differ of course, but note when the object time is repeated and when it is different):

```
var1 = TimeStamp()
var2 = TimeStamp()
print(var1.get_time())    # 2013-04-07 15:19:31.762220
print(var2.get_time())    # 2013-04-07 15:19:31.956881
print()
print(var1.get_time())    # 2013-04-07 15:19:31.762220
                         # (same as previous var1)
print(var2.get_time())    # 2013-04-07 15:19:31.956881
                         # (same as previous var2)
```