Critical Thinking Questions:

1. Examine the following code below, that defines a new 2-dimensional list datastructure.

```
0 class Matrix:
1     __slots__ = ['_matx']
2     def __init__(self, m):
3         self._matx = m
6     if __name__ == '__main__':
7         m = Matrix([[0,0], [1,0],[0,1], [1,1]])
```

a. What are the instance attribute(s) of a Matrix object?

_______________________________________________________________________

b. On what line do we instantiate a new Matrix object?_________

c. How many arguments do we instantiate this new Matrix object with?_________

d. What is stored in m._matx at the end of this code?

_______________________________________________________________________

e. What does the __init__(self, m) method do?

_______________________________________________________________________

FYI: Python specifies a series of special methods, which, when defined within a class are called implicitly. When we instantiate a new object, that calls the special method, __init__(..).
2. Examine the following code below, that adds line 8 to our previous Matrix class:

```python
class Matrix:
    __slots__ = ['_matx']
    def __init__(self, m):
        self._matx = m

if __name__ == '__main__':
    m = Matrix([[0,0], [1,0],[0,1], [1,1]])
    print("Num Cells in Matrix:", len(m))
```

a. What does the programmer hope will be printed after line 8?

b. This code will generate the following error, "TypeError: object of type 'Matrix' has no len()", why do you think that is?

c. If we add the following lines after line 3, the error is resolved. Why might that be?

```python
def __len__(self):
    return len(self._matx) * len(self._matx[0])
```

d. What type of value should be returned by __len__() (Hint: What type of value is len('hello')?)

e. If we changed line 5 to "return 99", what might line 8 output?

f. Why is len(m) preferable to m.__len__()? Both lines do the exact same thing!
3. Examine the following code, a new example!

```python
class Currency:
    __slots__ = ['_usd']
    def __init__(self, m):
        self._usd = m
    def __str__(self):
        return "Money money money, MONEY"

if __name__ == '__main__':
    myMoney = Currency(5)
    print(myMoney)
```

a. What are the instance attribute(s) of a `Currency` object?

b. On what line do we instantiate a new `Currency` object? 

c. What is stored in `myMoney`'s instance attributes at the end?

d. What does line 11 output?

e. Line 12 outputs "Money money money, MONEY", what method was called?

FYI: `print()` calls `str()` which implicitly calls the special method `__str__()`.  

f. What type of value does `__str__()` return?

g. Rewrite the special method `__str__(self)` so that it provides a meaningful, human-interpretable representation of the `Currency` object:

h. Write a line to call this special method:
4. Examine the following code, a modification of the previous example with lines 6-9 and lines after 12 added.

```python
class Currency:
    __slots__ = ['_usd']
    def __init__(self, m):
        self._usd = m
    def __str__(self):
        return '$' + str(self._usd)
    def __eq__(self, other):
        if not isinstance(other, Currency):
            return False
        return self._usd == other._usd

if __name__ == '__main__':
    myMoney = Currency(5)
    print(myMoney) # Prints '$5'
    print(myMoney == 5) # Prints False
    print(myMoney == Currency(5)) # Prints True
```

**FYI:** `isinstance(obj, ClassType)` returns True if and only if `obj` is an object of type `ClassType`.

a. What class method of `Currency` returns boolean values?

b. What is the type of the objects printed on lines 13 & 14?

c. What method might be being called when we use the `==` operator in lines 13 & 14?

d. For line 14, `print(myMoney == Currency(5))`, what do the arguments in `__eq__(self, other)`’s function header represent?

```python
def __eq__(self, other):
    if not isinstance(other, Currency):
        return False
    return self._usd == other._usd
```

e. What would be output for the following lines, if we added them to the end of the code?

```python
15 print(myMoney == Currency(70))
16 print(myMoney == Matrix([0]))
```

f. Write a new line 17 implicitly using `Currency`'s `__eq__()` method that would output ‘True’:

```python
print(myMoney == Currency(5)) # Prints True
```

g. Write a `__lt__`(self, other) method for `Currency` that will return `False` if other is not a `Currency` object, `True` if other represents a dollar amount less than self’s dollar amount, and `False` otherwise:

```python
def __lt__(self, other):
    if not isinstance(other, Currency):
        return False
    return self._usd < other._usd
```
5. Match up special methods on the left-hand column with the code that implicitly calls them in the right-hand column (make educated guesses using special method names and parameters!):

<table>
<thead>
<tr>
<th>Special Method</th>
<th>Called By</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. <strong>len</strong>(self)</td>
<td>m = Matrix()</td>
</tr>
<tr>
<td>b. <strong>str</strong>(self)</td>
<td>len(m)</td>
</tr>
<tr>
<td>c. <strong>iter</strong>(self)</td>
<td>mylist[22] = 5</td>
</tr>
<tr>
<td>d. <strong>bool</strong>(self)</td>
<td>mylist[22]</td>
</tr>
<tr>
<td>e. <strong>and</strong>(self, other)</td>
<td>m**2</td>
</tr>
<tr>
<td>f. <strong>add</strong>(self, other)</td>
<td>m * 2</td>
</tr>
<tr>
<td>g. <strong>mul</strong>(self, other)</td>
<td>m + 2</td>
</tr>
<tr>
<td>h. <strong>pow</strong>(self, other)</td>
<td>m &lt; 2</td>
</tr>
<tr>
<td>i. <strong>contains</strong>(self, item)</td>
<td>m &lt;= 5</td>
</tr>
<tr>
<td>j. <strong>getitem</strong>(self, key)</td>
<td>m &gt; 5</td>
</tr>
<tr>
<td>k. <strong>setitem</strong>(self, key, value)</td>
<td>m &gt;= 5</td>
</tr>
<tr>
<td>l. <strong>init</strong>(self)</td>
<td>m == 5</td>
</tr>
<tr>
<td>m. <strong>eq</strong>(self, other)</td>
<td>m and True</td>
</tr>
<tr>
<td>n. <strong>lt</strong>(self, other)</td>
<td>if m:</td>
</tr>
<tr>
<td>m. <strong>le</strong>(self, other)</td>
<td>22 in m</td>
</tr>
<tr>
<td>p. <strong>gt</strong>(self, other)</td>
<td>str(m)</td>
</tr>
<tr>
<td>q. <strong>ge</strong>(self, other)</td>
<td>for item in m</td>
</tr>
</tbody>
</table>

Confirm your responses by checking the python3 documentation:

https://docs.python.org/3/reference/datamodel.html#special-method-names
Application Questions: Use Python to check your work

1. a. Create a class, **MyList**, which has a list as an instance attribute. Define the special method, **__bool__** to return **False** if the list is empty, and **True** otherwise. Write some example lines of code that would call this **__bool__**(.) method *implicitly* (no underscores!).

```python
class MyList:
    __slots__ = ['thelist']

def __bool__(self):
    # your code here
```

1. b. Add a **__gt__**(self, other) special method to the previously defined class, **MyList**. It returns **True** if self is greater than other. How you define “greater than” is up to you!

```python
def __gt__(self, other):
    # your code here
```

1. c. Add a **__setitem__**(self, key, value) method that sets the item at index, key, to be value:

```python
def __setitem__(self, key, value):
    # your code here
```
2. Write a class, `Currency`, that has the instance attribute `_usd`. Implement an `__add__(self, other)` method that adds the value stored in `self._usd` to the value in `other` and returns the sum. Be sure to include a few example lines of code that calls this special method on your Currency objects.

```python
class Currency:
    __slots__ = ['_usd']

    def __add__(self, other):
        pass
```

3. Review lab assignments and Homeworks for more applications of special methods. We cover special methods repeatedly throughout the semester in labs, homeworks, and lecture.