Critical Thinking Questions:

1. Examine the sample code below.

```
# robot.py

class EvilRobot:
    morality = 'evil'
    __slots__ = ['name']
    def __init__(self, nm):
        self.name = nm

>>> from robot import EvilRobot
>>> er1 = EvilRobot('Herbert')
>>> er1.name
'Herbert'
>>> er2 = EvilRobot('Pearl')
>>> er1.morality
'evil'
>>> er2.morality
'evil'
>>> EvilRobot.morality
'evil'
```

a. What attributes does EvilRobot have?

________________________________________________________________________

b. How do you know those are its attributes?

________________________________________________________________________

c. What type of object is er1?

________________________________________________________________________

d. On what line is er1's name attribute on the left-hand side of an assignment operator?

________

e. On what line is morality on the left-hand side of an assignment operator?

________
f. How do the location of the assignments differ between `name` and `morality`?:

_______________________________________________________________________


g. What are lines c & f & h doing?:

_______________________________________________________________________


h. Why is the output on lines g & i the same?:

_______________________________________________________________________


i. How does line j differ from lines f & h?:

_______________________________________________________________________


FYI: **Class Attributes** are attributes that have shared values for all instances of that class. Class attributes are defined inside a class definition but outside any method. **Instance Attributes** are attributes that can have different values for different instances.

j. Is `name` an instance or class attribute? How do you know?:

_______________________________________________________________________


k. Add a line of python to add `mission` as a class attribute of Evil Robot:

_______________________________________________________________________


2. Examine the sample code below which continues from the above example.

```
class EvilRobot:
    morality = 'evil'
    __slots__ = ['name']
    def __init__(self, nm):
        self.name = nm

# ...continues from above ^^
m >>> er1.name = 'Potato'
n >>> er1.name
o 'Potato'
p >>> er2.name
q 'Pearl'
r >>> EvilRobot.morality = 'bad'
s >>> er1.morality
t 'bad'
u >>> er2.morality
v 'bad'
```

<table>
<thead>
<tr>
<th>robot.py</th>
<th>Interactive python</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 class EvilRobot:</td>
<td># ...continues from above ^^</td>
</tr>
<tr>
<td>1    morality = 'evil'</td>
<td>m &gt;&gt;&gt; er1.name = 'Potato'</td>
</tr>
<tr>
<td>2    <strong>slots</strong> = ['name']</td>
<td>n &gt;&gt;&gt; er1.name</td>
</tr>
<tr>
<td>3    def <strong>init</strong>(self, nm):</td>
<td>o 'Potato'</td>
</tr>
<tr>
<td>4        self.name = nm</td>
<td>p &gt;&gt;&gt; er2.name</td>
</tr>
<tr>
<td></td>
<td>q 'Pearl'</td>
</tr>
<tr>
<td></td>
<td>r &gt;&gt;&gt; EvilRobot.morality = 'bad'</td>
</tr>
<tr>
<td></td>
<td>s &gt;&gt;&gt; er1.morality</td>
</tr>
<tr>
<td></td>
<td>t 'bad'</td>
</tr>
<tr>
<td></td>
<td>u &gt;&gt;&gt; er2.morality</td>
</tr>
<tr>
<td></td>
<td>v 'bad'</td>
</tr>
</tbody>
</table>

| a. What value is stored in `er1.name` after line m?: | ________________________________ |
| b. What value is stored in `er2.name` after line m?: | ________________________________ |
| c. What value is stored in `er1.morality` after line r?: | ________________________________ |
| d. What value is stored in `er2.morality` after line r?: | ________________________________ |
| e. What might be displayed if we printed `EvilRobot.morality`?: | ________________________________ |
| f. How do lines m & r differ? | ________________________________ |
e. Why do er1 and er2 have the same morality after line r, but they do not have the same name after line m?

f. Write a line of python to change all EvilRobots' morality to 'no good':

FYI: The values stored in Class Attributes can be referenced by placing the class name prior to the dot notation followed by the variable name. This is in contrast to instance attributes, which are referenced by placing the instance prior to the dot notation.

3. Examine the sample code below which is from a different script.

goodbot.py

```python
0 class Robot:
1    __slots__ = ['name']
4    def introduce(self):
5        return 'I AM ' + self.name.upper()

6 class GoodRobot(Robot):
7    morality = 'good'
8    __slots__ = []
9
9 if __name__ == '__main__':
10    gr1 = GoodRobot()
11    gr1.name = 'Fifi'
12    print(gr1.name)
```

a. What type of object is gr1?

b. What slots does GoodRobot have?

c. If we added the line gr1.serialNum=20200410 to line 13, what would happen?

d. What attribute is being modified on line 11?

e. What slots does Robot have?

f. What is different about the class declaration on line 6 that we haven't seen before?

g. Line 12 will print 'Fifi', without an error. Why might that be?

FYI: A child-class or sub-class is a new class created by inheriting from a parent-class or super-class. Inheritance is the ability to define a new class that is a modified version of a previously defined class.

h. If we added the line gr1.introduce() to line 13 'I AM FIFI' would be displayed. What method was likely called, and what class does it belong to?
4. Examine the sample code below which is a slightly modified version of the previous script.

```python
class Robot:
    __slots__ = ['name']
    def __init__(self, nm):
        self.name = nm
    def introduce(self):
        return 'I AM ' + self.name.upper()

class GoodRobot(Robot):
    morality = 'good'
    __slots__ = []

if __name__ == '__main__':
    gr2 = GoodRobot('Stan')
    print(gr2.name)
```

a. What type of object is `gr2`? _______________________

b. What class does `GoodRobot` inherit from? _______________________

c. What is the parameter we are attempting to initialize `gr2` with? _______________________

d. Does `GoodRobot` have an initializer method with a name parameter? _________

e. Does `Robot` have an initializer method with a name parameter? _________

g. Does `GoodRobot` have name slot? _________

h. Does `Robot` have name slot? _________

f. Line 15 will display 'Stan'. What is stored in `gr2.name`? _______________________

g. In line 14, what special method is being called? Which class does it belong to? _______________________

---

FYI: Child-classes will inherit the attributes, methods, and special methods of their parent-class.

Application Questions: Use the Python Interpreter to check your work

1. a. Create a class, `NeutralRobot`, that inherits from the `Robot` class above. This class should have `morality` as a class attribute, and a `purpose` (i.e., 'sit and watch the grass grow', 'long walks on the beach', etc.) instance attribute:
b. Write a \_str\_(self) function to display a NeutralRobot's name, morality, and purpose:

c. Write an if \_name\_=='\_main\_' function to instantiate a new NeutralRobot, assign it name, morality, and purpose and then print the NeutralRobot.