Learning Objectives
Students will be able to:

Content:
- Define special methods in python
- Identify which special method is being called implicitly
- Explain how to call a special method implicitly

Process:
- Write code that calls a variety of special methods implicitly.
- Write code to implement special methods.

Prior Knowledge
- Python concepts from Activities 1-26. Classes. 28 (Arguments)

Folks, this is a brand new activity. If you encounter any issues/typos, please let Iris know!

Critical Thinking Questions:

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

```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]])
```

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:

```
Matrix.py

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]])
8    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?

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

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

e. Why is len(m) preferable to m.__len__()? Both lines do the exact same thing!

FYI: Many of the built-in python functions we’re familiar with are actually special methods that are implicitly calling methods defined within a class. For example, len() always implicitly calls __len__().
3. Match up special methods on the left-hand column with the code that implicitly calls them in the right-hand column:

<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; 5</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>print(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
4. 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"

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

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 10 output? ______________

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

```
_______________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________
```

i. How would you call this special method? _________________________________

j. What method is being called when we use the `==` operator? __________________

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

```
11 print(myMoney == Currency(5))
12 print(myMoney == Currency(70))
13 print(myMoney == Matrix([0]))
```

m. For line 11, `print(myMoney == Currency(5))`, what do the arguments in `__eq__(self, other)`'s function header represent?

```python
def __eq__(____________________, __________________):
```
Application Questions: Use Python to check your work

1. a. Create a class, MyClass, 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!).

class MyClass():
    __slots__ = ['thelist']
def __bool__(self):

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

def __lt__(self, other):

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

def __setitem__(self, key, value):

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.

class Currency():
    __slots__ = [‘_usd’]
    def __add__(self, other):
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________
        __________________________

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.