Name:_____ **Partner:** _____

Python Activity 27: Special Methods

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.

If you encounter any issues/typos, please let Iris know! Questions? Ask Iris or the POGILing forum

Critical Thinking Questions:

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

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]])
```

   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) * len(self._matx[0])
```

_____

_____

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__()`.

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!

Currency.py

```
0 class Currency:
1     __slots__ = ['_usd']
2     def __init__(self, m):
3         self._usd = m
4     def __str__(self):
5         return "Money money money, MONEY"

10 if __name__ == '__main__':
11    myMoney = Currency(5)
12    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.

Currency.py

```
0 class Currency:
1     __slots__ = ['_usd']
2     def __init__(self, m):
3          self._usd = m
4     def __str__(self):
5          return '$' + str(self._usd)
6     def __eq__(self, other):
7          if not isinstance(other, Currency):
8               return False
9          return self._usd == other._usd

10 if __name__ == '__main__':
11   myMoney = Currency(5)
12   print(myMoney) # Prints '$5'
13   print(myMoney == 5) # Prints False
14   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?

`def __eq__(`_____`,` _____`):`

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

```
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': _____

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:

_____

_____

_____

_____

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!):

Special Method	Called By
a. `__len__(self)`	`m = Matrix()`
b. `__str__(self)`	`len(m)`
c. `__iter__(self)`	`mylist[22] = 5`
d. `__bool__(self)`	`mylist[22]`
e. `__and__(self, other)`	`m**2`
f. `__add__(self, other)`	`m * 2`
g. `__mul__(self, other)`	`m + 2`
h. `__pow__(self, other)`	`m < 5`
i. `__contains__(self, item)`	`m <= 5`
j. `__getitem__(self, key)`	`m > 5`
k. `__setitem__(self, key, value)`	`m >= 5`
l. `__init__(self)`	`m == 5`
m. `__eq__(self, other)`	`m and True`
n. `__lt__(self, other)`	`if m:`
m. `__le__(self, other)`	`22 in m`
p. `__gt__(self, other)`	`str(m)`
q. `__ge__(self, other)`	`for item in m`

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!).

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

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!

```
def __gt__(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.