Mock Exam Problems (Homework06, Nov 16)

1 OOP and Inheritance (2021A, 18pts)

The teaching assistants (TAs) of SICP want to open a beverage store. Currently they sell normal drinks and milk beverage. But they encounter some problems when using Python to sell drinks, can you help them out? TAs already implement the `Drink` and `Milk` classes:

```
class Drink:
    menu = []

    def __init__(self, name):
        self.cups = 0
        self.name = name
        if name not in Drink.menu:
            Drink.menu.append(name)

    def make(self, amount):
        self.cups += amount

    def order(self, amount):
        if amount > self.cups:
            print('Insufficient cups')
        else:
            self.cups -= amount
            print('OK')

    def in_menu(name):
        return name in Drink.menu


class Milk(Drink):
    def __init__(self, name='Milk'):
        super().__init__(name)
```

1.1 Attributes (4pts)

Among the three attributes: (1) `menu`, (2) `cups`, and (3) `name`.
Which of them are class attributes, which of them are instance attributes?

1.2 What Would Python Display? (7pts)

TAs want to test their code! What would Python display when evaluating the following code in interactive console? Write your answer in each blank line.

```
>>> d = Drink('Coffee')
>>> m = Milk()
>>> m is d
____________________
```

```
>>> d.menu
____________________
>>> Milk.menu = ['WholeMilk', 'LowFatMilk']
>>> d.menu == m.menu
____________________
>>> d.menu = ['Cola']
>>> Drink.in_menu('Cola')
____________________
```

1.3 Inheritance and Override (7pts)

The bussiness is hot and there is a milk shortage. TAs decide to set purchase limitations on milk drinks: at most one cup in each order. If a customer orders two or more cups of milk, an error message should be displayed.

Please modify `Milk` class to **satisfy the behaviors described in doctest**. You don’t need to use all lines provided.

```
class Milk(Drink):
    """
    >>> drinks = [Drink('Coffee'), Milk()]
    >>> for d in drinks:
    ...     d.make(4)
    >>> for d in drinks:
    ...     assert d.cups == 4
    >>> for d in drinks:
    ...     d.order(1)
    OK
    OK
    >>> for d in drinks:
    ...     d.order(2)
    OK
    Insufficient milk
    >>> for d in drinks:
    ...     d.order(2)
    Insufficient cups
    Insufficient milk
    """
    def __init__(self, name='Milk'):
        super().__init__(name)


    ____________________________________________________________
        ________________________________________________________
        ________________________________________________________
        ________________________________________________________
        ________________________________________________________
```

1. (21 points) OOP and Inheritance (2020A)

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. Each line in the right-hand side of the table corresponds to each output.

Note that the interactive interpreter displays the value of a successfully evaluated expression, unless it is None. Assume that you have started Python3 and executed the code shown on the left. Expressions evaluated by the interpreter have a cumulative effect.

```
class Pet:
    life = 100
    def __init__(self, birth):
        self.dead = birth + self.life

class A:
    total = 0
    def __init__(self, c):
        self.count = self.total
        self.total = c + 1

class B(A):
    def __init__(self, c):
        self.count = self.total
        total = c + 1

class C(B):
    total = 10
    def __init__(self, a, b):
        self.count = a
        total = b
        B.__init__(self, 0)
```

Expression	Interactive Output

`p = Pet(2020)`
`q = Pet(2021)`
`p.dead`
`q.dead`	

`Pet.life = 50`
`p.life`
`q.life`	

`p.life = 80`
`p.life`
`q.life`	

`Pet.life = 70`
`p.life`
`q.life`	

`a = A(10)`
`a.count`
`a.total`
`A.total`	

`b = B(20)`
`b.count`
`b.total`
`B.total`	

`c = C(30, 40)`
`c.count`
`c.total`
`C.total`	