OOP: exam-level questions

> If you need help reviewing OOP, take a look at these resources:
>
> - Albert's and Robert's slides (Pokemon examples!) (https://docs.google.com/presentation/d/1MgFiiLa1xtQ4LfGOFZ0McvifBh0F5RaLZ23B5E1cRyQ/edit)
> - Albert's and Robert's slides (inheritance) (https://docs.google.com/presentation/d/1KlUU5wxUvSig1q4ZXoe_ujSn9zSfp-ehzBvzaX19KPM/edit)

Each question has a "Toggle Solution" button -- click it to reveal that question's solution.

What would Python print?

For the following questions, use the following class definition:

```
class Account:
    """A class computer account. Each account has a two-letter ID
    and the name of the student who is registered to the account.
    """
    num_of_accounts = 0
    def __init__(self, id):
        self.id = id
        Account.num_of_accounts += 1

    def register(self, student):
        self.student = student
        print('Registered ' + student)

    @property
    def type(self):
        return type(self)
```

Question 1

```
>>> acc_aa = Account("aa")
>>> acc_aa.register("Peter Perfect")
______
>>> Account.register(self, "Jom Magrotker")
______
>>> Account.register(acc_aa, "Jom Magrotker")
______
>>> Account.register("Jom Magrotker")
______
```

Toggle Solution

Question 2

Use the `acc_aa` from the previous question.

```
>>> f1 = Account.register
>>> f1(acc_aa, "Peter Perfect")
______
>>> f2 = acc_aa.register
>>> f2(acc_aa, "Peter Perfect")
______
>>> f2("Peter Perfect")
______
```

Toggle Solution

Question 3

Use the `acc_aa` from the previous question.

```
>>> Account.register = lambda self: "WAT"
>>> Account.register(acc_aa, "Hello")
______
>>> Account.register("Hello")
______
>>> acc_aa.register("goodbye")
______
>>> acc_aa.register()
______
```

Toggle Solution

Question 4

Assume you have started a new Python interactive session with the original definition of the `Account` class.

```
>>> acc_aa = Account("aa")
>>> acc_aa.register("Peter Perfect")
______
>>> acc_zz = Account("zz")
>>> acc_aa.register = acc_zz.register
>>> acc_aa.register("Bozo")
______
>>> acc_aa.student
______
>>> acc_zz.student
______
```

Toggle Solution

Code-Writing questions

Question 5

In computer science, a circular buffer (http://en.wikipedia.org/wiki/Circular_buffer) a type of data structure that is used to store temporary, sequential data in a constant amount of space (this is commonly used to buffer data streams) in a first-in-first-out manner (the first element to be added is the first element to be removed). A circular buffer has the following properties:

- Each buffer has a fixed size of `n` elements (e.g. strings). This size is determined upon creation of the buffer. Note that the total number of elements that can be inserted into the buffer can exceed `n`, but the number of elements in buffer at *any given time* must be less than or equal to `n`.
- Each buffer has a `start` that keeps track of the earliest element that is currently in the buffer. Similarly, each buffer has an `end` that keeps track of the next available empty index in the buffer.
- The buffer has an `append` method, which adds a given element into the buffer. If the buffer is full, (i.e. the buffer already has `n` elements), do not add the element, and instead print "Buffer exceeded capacity".
- The buffer has a `remove` method, which removes the earliest element that is still in the buffer. If there are no elements in the buffer, print "Buffer is empty".

In order to implement the `append` and `remove` methods, you should have list of length `n` that stores the elements currently in the buffer. When you append the ith element, you should insert it into index (i mod n) of the list. Similarly, when you are removing the jth element, you should extract the element at index (j mod n) of the list. For more descriptions of the behavior, see the doctest.

```
class CircularBuffer:
    """Doctests:

    >>> buffer = CircularBuffer(3)
    >>> buffer.remove()
    Buffer is empty
    >>> buffer.append('a')
    >>> buffer.remove()
    'a'
    >>> buffer.remove()
    Buffer is empty
    >>> buffer.append('b')
    >>> buffer.append('c')
    >>> buffer.append('d')
    >>> buffer.append('e')
    Buffer capacity exceeded
    >>> buffer.remove()
    'b'
    >>> buffer.remove()
    'c'
    >>> buffer.remove()
    'd'
    >>> buffer.remove()
    Buffer is empty
    """
    def __init__(self, n):
        self.array = [None]*n   # list of length n
        self.n = n
        self.start = 0
        self.end = 0

    def append(self, elem):
        "*** YOUR CODE HERE ***"

    def remove(self):
        "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 6

Write a `Chef` class with the following qualities:

- Each `Chef` is initialized with a list of required ingredients. Each item in the list is added to a storage that is shared by all the `Chef`s with an initial stock of 2. If the item is already in the storage, do NOT add it in again.
- Each `Chef` can `fetch_ingredients` from a storage that is shared by all the `Chef`s. Each `Chef` only needs 1 of each ingredient.
- Each `Chef` can `serve`, where they put their finished food in a shared list of `finished` foods.

For finer details of implementation, see the doctest.

```
class Chef:
    """Doctests:

    >>> albert = Chef('quiche', ['egg', 'cheese', 'cream', 'salt'])
    >>> ramsay = Chef('steak', ['meat', 'bbq sauce', 'salt'])
    >>> ramsay.cook()
    'Not enogh ingredients!'
    >>> ramsay.serve()
    'No food to serve!'
    >>> ramsay.fetch_ingredients()     # 1 salt remaining
    "Fetched: ['meat', 'bbq sauce', 'salt']"
    >>> ramsay.cook()
    'Cooked steak!'
    >>> ramsay.serve()
    >>> Chef.finished
    ['steak']
    >>> albert.fetch_ingredients()     # 0 salt remaining
    "Fetched: ['egg', 'cheese', 'cream', 'salt']"
    >>> albert.cook()
    'Cooked quiche!'
    >>> albert.serve()
    >>> Chef.finished
    ['steak', 'quiche']
    >>> ramsay.fetch_ingredients()
    'No more salt!'
    """
    "*** YOUR CODE HERE ***"
```

Toggle Solution