Instructions

> Please download homework materials `hw06.zip` from our QQ group if you don't have one.

In this homework, you are required to complete the problems described in section 3. The starter code for these problems is provided in `hw06.py`.

Submission: As instructed before, you need to submit your work with Ok by `python ok --submit`. You may submit more than once before the deadline, and your score of this assignment will be the highest one of all your submissions.

Readings: You might find the following references to the textbook useful:

- Section 2.5

Required Problems

In this section, you are required to complete the problems below and submit your code to OJ website.

Remember, you can use `ok` to test your code:

```
$ python ok  # test all classes
$ python ok -q <class>  # test single class
```

Problem 1: Vending Machine (200pts)

Create a class called `VendingMachine` that represents a vending machine for some product. A `VendingMachine` object returns strings describing its interactions. Fill in the `VendingMachine` class, adding attributes and methods as appropriate, such that its behavior matches the following doctests:

```
class VendingMachine:
    """A vending machine that vends some product for some price.

    >>> v = VendingMachine('candy', 10)
    >>> v.vend()
    'Machine is out of stock.'
    >>> v.add_funds(15)
    'Machine is out of stock. Here is your $15.'
    >>> v.restock(2)
    'Current candy stock: 2'
    >>> v.vend()
    'You must add $10 more funds.'
    >>> v.add_funds(7)
    'Current balance: $7'
    >>> v.vend()
    'You must add $3 more funds.'
    >>> v.add_funds(5)
    'Current balance: $12'
    >>> v.vend()
    'Here is your candy and $2 change.'
    >>> v.add_funds(10)
    'Current balance: $10'
    >>> v.vend()
    'Here is your candy.'
    >>> v.add_funds(15)
    'Machine is out of stock. Here is your $15.'

    >>> w = VendingMachine('soda', 2)
    >>> w.restock(3)
    'Current soda stock: 3'
    >>> w.restock(3)
    'Current soda stock: 6'
    >>> w.add_funds(2)
    'Current balance: $2'
    >>> w.vend()
    'Here is your soda.'
    """
    "*** YOUR CODE HERE ***"
```

You may find Python string formatting syntax or f-strings useful. A quick example:

```
>>> ten, twenty, thirty = 10, 'twenty', [30]
>>> '{0} plus {1} is {2}'.format(ten, twenty, thirty)
'10 plus twenty is [30]'

>>> feeling = 'love'
>>> course = 61
>>> f'I {feeling} {course}A!'
'I love 61A!'
```

Problem 2: Cat (100pts)

Below is a skeleton for the `Cat` class, which inherits from the `Pet` class. To complete the implementation, override the `__init__` and `talk` methods and add a new `lose_life` method, such that its behavior matches the following doctests.

We may change the implementation of `Pet` while testing your code, so make sure you use inheritance correctly.

Hint: You can call the `__init__` method of Pet to set a cat's name and owner.

```python
class Pet:
    """A pet.

    >>> kyubey = Pet('Kyubey', 'Incubator')
    >>> kyubey.talk()
    Kyubey
    >>> kyubey.eat('Grief Seed')
    Kyubey ate a Grief Seed!
    """
    def __init__(self, name, owner):
        self.is_alive = True    # It's alive!!!
        self.name = name
        self.owner = owner

    def eat(self, thing):
        print(self.name + " ate a " + str(thing) + "!")

    def talk(self):
        print(self.name)


class Cat(Pet):
    """A cat.

    >>> vanilla = Cat('Vanilla', 'Minazuki Kashou')
    >>> isinstance(vanilla, Pet) # check if vanilla is an instance of Pet.
    True
    >>> vanilla.talk()
    Vanilla says meow!
    >>> vanilla.eat('fish')
    Vanilla ate a fish!
    >>> vanilla.lose_life()
    >>> vanilla.lives
    8
    >>> vanilla.is_alive
    True
    >>> for i in range(8):
    ...     vanilla.lose_life()
    >>> vanilla.lives
    0
    >>> vanilla.is_alive
    False
    >>> vanilla.lose_life()
    Vanilla has no more lives to lose.
    """
    def __init__(self, name, owner, lives=9):
        "*** YOUR CODE HERE ***"

    def talk(self):
        """ Print out a cat's greeting.
        """
        "*** YOUR CODE HERE ***"

    def lose_life(self):
        """Decrements a cat's life by 1. When lives reaches zero, 'is_alive'
        becomes False. If this is called after lives has reached zero, print out
        that the cat has no more lives to lose.
        """
        "*** YOUR CODE HERE ***"
```

Problem 3: Noisy Cat (100pts)

More cats! Fill in this implemention of a class called `NoisyCat`, which is just like a normal `Cat`. However, `NoisyCat` talks a lot -- twice as much as a regular `Cat`!

We may change the implementation of `Pet` and `Cat` while testing your code, so make sure you use inheritance correctly.

```
class NoisyCat: # Dose this line need to change?
    """A Cat that repeats things twice.

    >>> chocola = NoisyCat('Chocola', 'Minazuki Kashou')
    >>> isinstance(chocola, Cat) # check if chocola is an instance of Cat.
    True
    >>> chocola.talk()
    Chocola says meow!
    Chocola says meow!
    """
    def __init__(self, name, owner, lives=9):
        # Is this method necessary? If not, feel free to remove it.
        "*** YOUR CODE HERE ***"

    def talk(self):
        """Talks twice as much as a regular cat.
        """
        "*** YOUR CODE HERE ***"
```

Just for fun Problems

This section is out of scope for our course, so the problems below is optional. That is, the problems in this section **don't** count for your final score and **don't** have any deadline. Do it at any time if you want an extra challenge or some practice with higher order function and abstraction!

To check the correctness of your answer, you can submit your code to Contest 'Just for fun'.

Problem 4: Next Fibonacci Object (0pts)

Implement the `next` method of the `Fib` class. For this class, the `value` attribute is a Fibonacci number. The `next` method returns a `Fib` instance whose `value` is the next Fibonacci number. The `next` method should take only constant time.

You can assume we will neither create `Fib` instance with non-zero `value` directly nor edit the `value` of `Fib` instance outside the class.

> *Hint*: Keep track of the previous number by setting a new instance attribute inside next. You can create new instance attributes for objects at any point, even outside the `__init__` method.

```
class Fib:
    """A Fibonacci number.

    >>> start = Fib()
    >>> start.value
    0
    >>> start.next().value
    1
    >>> start.next().next().value
    1
    >>> start.next().next().next().value
    2
    >>> start.next().next().next().next().value
    3
    >>> start.next().next().next().next().next().value
    5
    >>> start.next().next().next().next().next().next().value
    8
    >>> start.value # Ensure start isn't changed
    0
    """
    def __init__(self, value=0):
        self.value = value

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