CS61A NOTE7 OOP

Object-oriented programming (OOP) is a programming paradigm that allows us to treat data as objects, like we do in real life.For example, consider the **class** `Student`. Each of you as individuals is an **instance** of this class.

Details that all CS 61A students have, such as `name`, are called **instance variables**. Every student has these variables, but their values differ from student to student. A variable that is shared among all instances of `Student` is known as a **class variable**. For example, the `extension_days` attribute is a class variable as it is a property of all students.All students are able to do homework, attend lecture, and go to office hours. When functions belong to a specific object, they are called **methods**. In this case, these actions would be methods of `Student` objects.

面向对象编程允许我们将数据视为对象。例如，考虑 class 学生。每个人作为个体都是这个类的 instance。学生细节，如姓名，是**实例变量 instance variables**。每个学生都有这些变量，但它们的值因学生而异。一个在所有 "学生 "的实例中共享的变量被称为**类变量 class variable** 。extension_days 是一个 class variable，因为是所有学生的属性。当函数属于一个特定的对象时，它们被称为 **methods**，上课做作业等将是学生 objects 的 method。

- **class**: a template for creating objects 用来创建对象。学生
- **instance**: a single object created from a class 类建的一个对象。学生个体
- **instance variable**: a data attribute of an object, specific to an instance 对象数据属性，特定于某一实例。姓名
- **class variable**: a data attribute of an object, shared by all instances of a class 对象数据属性，一个类共享。政策
- **method**: a bound function that may be called on all instances of a class 类所有实例都能调用的绑定函数。上课交作业

Instance variables, class variables, and methods are all considered attributes of an object.这些都是对象属性

Q1:Student OOP

纯文本

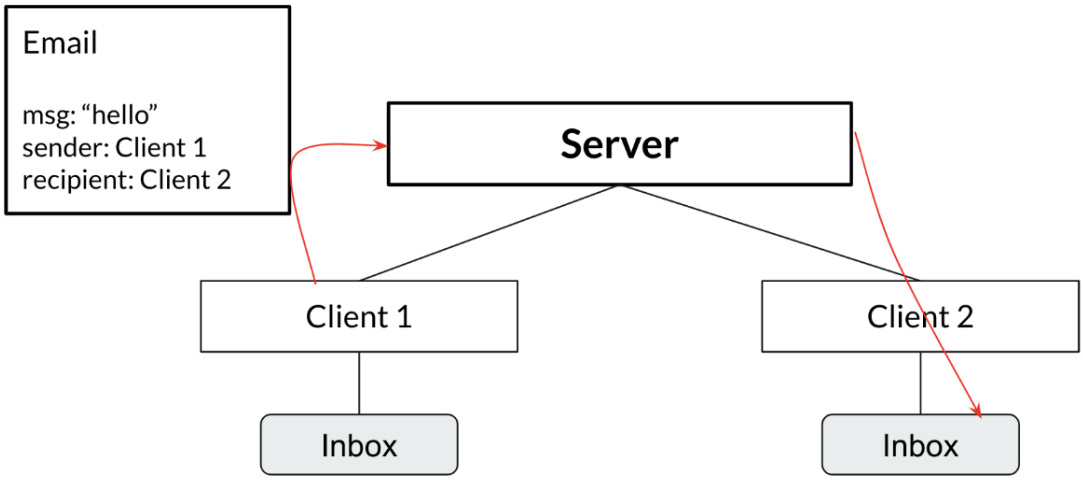
```
class Student:

    extension_days = 3 #类变量

    def __init__(self, name, staff): #__init__输入学生姓名与教授姓名
        self.name = name              # 初始自己姓名，实例变量，自己属性self...=
        self.understanding = 0        # 初始自己理解
        staff.add_student(self)       #staff原因:被教师加，而非学生自己有教师属性
        print("Added", self.name)     #相当于已经把学生加到教授底下

    def visit_office_hours(self, staff): #定义method
        staff.assist(self)                #别人的assist(self)
        print("Thanks, " + staff.name)

class Professor:

    def __init__(self, name):    #输入教授姓名，已把学生加到教授底下不再需要输入化学
        self.name = name         #初始教授姓名
        self.students = {}       #初始学生字典实例变量，self...={}建一个字典，不用

    def add_student(self, student):    #不是学生下加教授，是教授下加学生
        self.students[student.name] = student  #字典语法有[]

    def assist(self, student):            #定义method
        student.understanding += 1        #别人的

    def grant_more_extension_days(self, student, days): #定义method
        student.extension_days = days     #别人的

>>> callahan = Professor("Callahan")  #输入符合__init__的姓名,Professor其实也算
>>> elle = Student("Elle", callahan)  #传入两未知，顺便执行了staff.add_student
Added Elle  #print不用加引号

>>> elle.visit_office_hours(callahan) #输入staff，self学生放到前面
Thanks, Callahan

>>> elle.visit_office_hours(Professor("Paulette"))
Thanks, Paulette

>>> [name for name in callahan.students]
['Elle'] #是[]
```

```
>>> x = Student("Vivian", Professor("Stromwell")).name
Added Vivian #附加打印

>>> x
'Vivian'

>>> [name for name in callahan.students]
['Elle']

>>> elle.extension_days
3

>>> callahan.grant_more_extension_days(elle, 7) #输入学生天数，前缀教授.
>>> elle.extension_days  #前缀
7
>>> Student.extension_days
3
```

Q2: Email（难）

We would like to write three different classes (Server, Client, and Email) to simulate a system for sending and receiving emails. A Server has a dictionary mapping client names to Client objects, and can both send Emails to Clients in the Server and register new Clients. A Client can both compose emails (which first creates a new Email object and then sends it to the recipient client through the server) and receive an email (which places an email into the client's inbox).三个类，邮差，客户和邮件。邮差有客户字典(key 姓名 value 客户)，寄信函数，登记新客户(字典)函数。客户有发邮件(新建 email 对象 + 通过 server 发送给收件人)函数，收件 inbox list 函数

Emails will only be sent/received within the same server, so clients will always use the server they're registered in to send emails to other clients that are registered in the same server. 注册在同一 server 下的客户发邮件



To solve this problem, we'll split the section into two halves (students on the left and students on the right):

- Everyone will implement the Email class together 每人一起实现邮件类
- The first half (left) will implement the Server class 左边服务类
- The other half (right) will implement the Client class 右边客户类

纯文本

```
class Email:
    """
    Every email object has 3 instance attributes: the
    message, the sender name, and the recipient name.
    # 邮件对象有三个属性，信息，发信人，收信人
    >>> email = Email('hello', 'Alice', 'Bob')
    >>> email.msg
    'hello'
    >>> email.sender_name
    'Alice'
    >>> email.recipient_name
    'Bob'
    """

    def __init__(self, msg, sender_name, recipient_name):
        "*** YOUR CODE HERE ***"  #需要初始化，将给的参数传入对象
        self.msg=msg
        self.sender_name=sender_name
        self.recipient_name=recipient_name
```

```
class Server:
#邮差有客户字典(key姓名value客户)，寄信send函数，登记新客户(字典)函数
    """
    Each Server has one instance attribute: clients (which
    is a dictionary that associates client names with
    client objects)寄信人有一个属性：收信人(姓名字典)
    """
    def __init__(self): #没什么可以传入的，新客户字典初始化吧
        self.clients = {} #key是客户姓名，value是客户本身

    def send(self, email): #email给的client信息是客户姓名
        """
        Take an email and put it in the inbox of the client
        it is addressed to. #发邮件并放入收信客户(从email信息中得知)的收件箱。
        """
        client = self.clients[email.recipient_name] #赋值(收信人)而非加字典
        #右边是value是收信人本身，要server下的客户，提出为对象
        client.receive(email)

    def register_client(self, client, client_name):
        """
        Takes a client object and client_name and adds them to the
        clients instance attribute #对server来说登记客户对象&姓名，添加到客户属性
        """
        "*** YOUR CODE HERE ***"
        self.clients[client_name]=client #等级新客户字典

class Client: #发邮件(新建email对象+通过server发给收件人)函数，收件inbox list函数
    """
    Every Client has three instance attributes: name (which is
    used for addressing emails to the client), server
    (which is used to send emails out to other clients), and
    inbox (a list of all emails the client has received).每个客户有三个属性：
    name，server(用于向其他客户发送邮件)和收件箱(客户收到的邮件列表)。

    >>> s = Server()
    >>> a = Client(s, 'Alice') #客户姓名
    >>> b = Client(s, 'Bob')
    >>> a.compose('Hello, World!', 'Bob') #客户短信，接信人
    >>> b.inbox[0].msg
    'Hello, World!'
    >>> a.compose('CS 61A Rocks!', 'Bob')
    >>> len(b.inbox)
    2
```

```
    >>> b.inbox[1].msg
    'CS 61A Rocks!'
    """
    def __init__(self, server, name): #姓名server需传入
        self.inbox = []                #inbox下面直接[]list开始即可
        "*** YOUR CODE HERE ***"
        self.server = server
        self.name = name
        self.server.register_client(self, self.name)  #因为只给了姓名，用字典取

    def compose(self, msg, recipient_name):#作为寄信人，客户短信(email找)，接信
        """Send an email with the given message msg to the given recipient
        "*** YOUR CODE HERE ***"
        email = Email(msg, self.name, recipient_name)  #参数传入email
        self.server.send(email)  #自己对应的邮差发信

    def receive(self, email):#作为收信人(客户收到的邮件列表)
        """Take an email and add收信箱后加 it to the inbox of this client."""
        "*** YOUR CODE HERE ***"
        self.inbox.append(email) #list加一个
```

Q3: Keyboard

We'd like to create a `Keyboard` class that takes in an arbitrary number of `Button`s and stores these `Button`s in a dictionary. The keys in the dictionary will be `int`s that represent the position on the `Keyboard`, and the values will be the respective `Button`. Fill out the methods in the `Keyboard` class according to each description, using the doctests as a reference for the behavior of a `Keyboard`.创建键盘类，可以接收任意数量的 Button 并将这些 Button 存储在一个字典中。字典中的键将是代表键盘上位置的 ints，而值将是各自的 Button。根据每个描述填写键盘类中的方法，键盘行为参考 doctests。

纯文本

```
class Button:
    def __init__(self, pos, key):
        self.pos = pos
        self.key = key
        self.times_pressed = 0 #初始化，类似understanding，对button而言属性
```

```
class Keyboard:
    """A Keyboard stores an arbitrary number of Buttons in a dictionary.
    Each dictionary key is a Button's position, and each dictionary
    value is the corresponding Button.键盘将任意数量Button存在字典里。
    键是Button位置，值是相应的Button。
    >>> b1, b2 = Button(5, "H"), Button(7, "I")  #是*args
    >>> k = Keyboard(b1, b2)
    >>> k.buttons[5].key     #字典self.buttons[key]=，值self.buttons[pos].ke
    'H'
    >>> k.press(7)           #有press函数
    'I'
    >>> k.press(0) # No button at this position
    ''
    >>> k.typing([5, 7])     #有typing函数
    'HI'
    >>> k.typing([7, 5])
    'IH'
    >>> b1.times_pressed     #有self.times_pressed，一开始是0，后来+=1，不用def
    2
    >>> b2.times_pressed
    3
    """
    def __init__(self, *args):  #*args，函数以列表或者元组形式传参时使用*args
        self.button={}         #创建一个字典
        for button in arg:     #一个个button传入
            self.buttons[button.pos]=button #字典

    def press(self, pos):
        """Takes in a position of the button pressed, and
        returns that button's output."""输入pos，输出button值，考虑None情况
        if pos in self.button.keys(): #有空的情况
             b=self.button[pos] #用属性取出对象button，前文也是这个思路
             b.times_pressed+=1
             return b.key
        return ''  #反之前面不return的话

    def typing(self, typing_input):  #更多pos一起输入，要for循环
        """Takes in a list of positions of buttons pressed, and
        returns the total output."""
        total=''
        for pos in typing_input:
             total+=self.press(pos) #press对一个typing是对多个因此应用
        return total
```

Q4: Relay 接力

In a Math Olympiad style relay, team members solve questions while sitting in a line. Each team member's answer is calculated based on the answer from the team member sitting in front of them.队员答案根据坐在他们前面的队员答案来计算。

For example, suppose we have three team members, adder, adder2, and multiplier, with adder sitting at the very front, adder2 in the middle, and multiplier at the end. When we call the relay_calculate method from multiplier, we first apply the adder operation to the input x. Then, the answer from adder is passed into 传递 the adder2 operation 操作. Finally, the answer from adder2 is passed into the multiplier operation. The answer from multiplier is our final answer. Additionally, each team member has a relay_history method, which uses the fact that each team member has an instance variable history. relay_history returns a list of the answers given by each team member, and this is updated each time we call relay_calculate.每个团队成员都有 relay_calculate 和 relay_history 方法。relay_calculate 从 adder 到 adder2 到 multiplier 得出最终答案，relay_history 返回每个团队成员给出的答案列表，每次我们调用 relay_calculate 时都会更新。

纯文本

```
>>> adder = TeamMember(lambda x: x + 1) # team member at front
>>> adder2 = TeamMember(lambda x: x + 2, adder) # team member 2
>>> multiplier = TeamMember(lambda x: x * 5, adder2) # team member 3
>>> adder.relay_history() # relay history starts off as empty
[]
>>> adder.relay_calculate(5) # 5 + 1
6
>>> adder2.relay_calculate(5) # (5 + 1) + 2
8
>>> multiplier.relay_calculate(5) # (((5 + 1) + 2) * 5)
40
>>> multiplier.relay_history() # history of answers from the most recent re
[6, 8, 40]
>>> adder.relay_history()
[6]
```

纯文本

```
class TeamMember:
    def __init__(self, operation, prev_member=None):#以类想法add操作传入
        """前一个数字一开始已经初始
        A TeamMember object is instantiated by taking in an `operation`
        and a TeamMember object `prev_member`, which is the team member
        who "sits in front of" this current team member. A TeamMember also
        tracks追踪 a `history` list, which contains the answers given by
        each individual team member.输入操作和前一个数字，还追踪list,包含历史答案
        """
        self.history = []  #注意和字典不一样是[]
        "*** YOUR CODE HERE ***"
        self.operation = operation
        self.prev_member = prev_member

    def relay_calculate(self, x): #x是初始
        """
        The relay_calculate method takes in a number `x` and performs a
        relay by passing in `x` to the first team member's `operation`.
        Then, that answer is passed to the next member's operation, etc. un
        we get to the current TeamMember, in which case we return the
        final answer, `result`. 此函数输入x一步步算到最终答案
        """
        if self.prev_member==None: #一开始是前面没人None
            "*** YOUR CODE HERE ***"
            result=self.operation(x)
            self.history=[result] #字典是a[key]=value,list是+[]
        else:  #result更新(记忆前，得出后(加list))，prev_member更新
            "*** YOUR CODE HERE ***"
            pre_result=self.prev_member.relay_calculate(x) #现在坐的人的前面
            result=self.operation(pre_result)
            self.history=self.prev_member.histry+[result]
        return result

    def relay_history(self):
        """
        Returns a list of the answers given by each team member in the
        most recent relay the current TeamMember has participated in.
        """
        "*** YOUR CODE HERE ***"
        return self.history
```

Class Methods

Now we'll try out another feature of Python classes: class methods. A method can be turned into a class method by adding the classmethod decorator. Then, instead of receiving the instance as the first argument (`self`), the method will receive the class itself (`cls`). Python 类的另一个特性：类方法。通过添加 classmethod 装饰，一个方法可以变成一个类方法。该方法不再接收实例作为 self，而是接收类本身 cls。Class methods are commonly used to create "factory methods": methods whose job is to construct and return a new instance of the class.类方法通常被用来创建 "工厂方法"：这些方法的工作是构造并返回一个新的类实例。

For example, we can add a `robo_factory` 机器人狗 class method to our `Dog` class that makes robo-dogs:

```
纯文本
class Dog:
    def __init__(self, name, owner):
        self.name = name
        self.owner = owner
    @classmethod类方法
    def robo_factory(cls, owner): #调用时Dog不用作为cls传入，cls已绑定
        return cls("RoboDog", owner) #相当于Dog("RoboDog",owner)
```

Then a call to `Dog.robo_factory('Sally')` would return a new `Dog` instance with the name "RoboDog" and owner "Sally". Note that with the call above, we don't have to explicitly pass in the `Dog` class as the `cls` argument, since Python implicitly does that for us. We only have to pass in a value for `owner`. When the body of the `Dog.robo_factory` is run, the line `cls("RoboDog", owner)` is equivalent to `Dog("RoboDog", owner)` (since `cls` is bound to the `Dog` class), which creates the new `Dog` instance.调用 Dog.robo_factory('Sally') 将返回一个新的 Dog 实例，名字是 "RoboDog" ，主人是 "Sally"。在上面的调用中，我们不需要将狗作为 cls 参数传入， Python 为我们做过。我们只需要传入一个所有者的值。当 Dog.robo_factory 的主体运行时，cls("RoboDog", owner) 等价于 Dog("RoboDog", owner) (cls 被绑定到 Dog 类)，创建了新的 Dog 实例。

Q5: Own A Cat

Now implement the `cat_creator` method below, which takes in a string `owner` and creates a `Cat` named "**[owner]**'s Cat", where **[owner]** is replaced with the name in the `owner` string.创建猫方法传入字符串'主人'，主人代入"主人的猫"

> **Hint:** To place an apostrophe within a string, the entire string must be surrounded in double-quotes (`"DeNero's Dog"`)

纯文本

```
class Cat:
    def __init__(self, name, owner, lives=9): #类似pre=None
        self.is_alive = True   #⚠
        self.name = name
        self.owner = owner
        self.lives = lives

    def talk(self):
        return self.name + ' says meow!'

    @classmethod #别的传入的self是实例，这里传入的cls是一个大类
    def cat_creator(cls, owner):
        """
        Returns a new instance of a Cat.

        This instance's name is "[owner]'s Cat", with
        [owner] being the name of its owner.

        >>> cat1 = Cat.cat_creator("Bryce") #只传入主人string，得到一个新猫
        >>> isinstance(cat1, Cat)
        True
        >>> cat1.owner
        'Bryce'
        >>> cat1.name  #实例名字是主人's 猫
        "Bryce's Cat"
        >>> cat2 = Cat.cat_creator("Tyler")
        >>> cat2.owner
        'Tyler'
        >>> cat2.name
        "Tyler's Cat"
        """#以上知实例有两个属性owner和name——"主人's Cat"
        name = owner + "'s Cat"   #主人已经是string
        return cls(name, owner)   #传入cls新建一个实例，不用self
```

Lab7 Q3:Person

Modify the following `Person` class to add a `repeat` method, which repeats the last thing said. See the doctests for an example of its use.

纯文本

```
class Person:
    """Person class.

    >>> steven = Person("Steven")
    >>> steven.repeat()       # initialized person has the below starting r
    'I squirreled it away before it could catch on fire.'
    >>> steven.say("Hello")
    'Hello'
    >>> steven.repeat()
    'Hello'
    >>> steven.greet()
    'Hello, my name is Steven'
    >>> steven.repeat()
    'Hello, my name is Steven'
    >>> steven.ask("preserve abstraction barriers")
    'Would you please preserve abstraction barriers'
    >>> steven.repeat()
    'Would you please preserve abstraction barriers'
    """

    def __init__(self, name):
        self.name = name
        "*** YOUR CODE HERE ***"
        self.previous='I squirreled it away before it could catch on fire.'

    def say(self, stuff):
        "*** YOUR CODE HERE ***"
        self.previous=stuff #记录
        return stuff

    def ask(self, stuff):
        return self.say("Would you please " + stuff) #新的传入say的stuff

    def greet(self):
        return self.say("Hello, my name is " + self.name)
```

```
    def repeat(self):
        "*** YOUR CODE HERE ***"
        return self.previous
```

lab7 Q4: Smart Fridge 智能冰箱

The `SmartFridge` class is used by smart refrigerators to track which items are in the fridge and let owners know when an item has run out.追踪物品

The class internally uses a dictionary to store items, where each key is the item name and the value is the current quantity. 用字典存储物品，key 是物品名字值是现存量 The `add_item` method should add the given quantity of the given item and report the current quantity. You can assume that the `use_item` method will only be called on items that are already in the fridge, and it should use up the given quantity of the given item. If the quantity would fall to or below zero, it should only use *up to* the remaining quantity, and remind the owner to buy more of that item. `use_item` 只用在已存在的物品，且最多用到 0，并提醒人买更多

Finish implementing the `SmartFridge` class definition so that its `add_item` and `use_item` methods work as specified by the doctests.

纯文本

```
class SmartFridge:
    """
    >>> fridgey = SmartFridge()
    >>> fridgey.add_item('Mayo', 1)
    'I now have 1 Mayo'
    >>> fridgey.add_item('Mayo', 2)
    'I now have 3 Mayo'
    >>> fridgey.use_item('Mayo', 2.5)
    'I have 0.5 Mayo left'
    >>> fridgey.use_item('Mayo', 0.5)
    'Oh no, we need more Mayo!'
    >>> fridgey.add_item('Eggs', 12)
    'I now have 12 Eggs'
    >>> fridgey.use_item('Eggs', 15)
    'Oh no, we need more Eggs!'
    >>> fridgey.add_item('Eggs', 1)
    'I now have 1 Eggs'
    """
```

```
    def __init__(self):
        self.items = {} #字典，注意[]，items可以替换为dic[]

    def add_item(self, item, quantity):
        if item in self.items:
            self.items[item]+=quantity
        else:
            self.items[item]=quantity #顺便添加字典
        return f'I now have {self.items[item]} {item}'

    def use_item(self, item, quantity):
        self.items[item] -= min(quantity,self.items[item])
        if self.items[item]>0:
            return f'I have {self.items[item]} {item} left'
        else:
            return f'Oh no, we need more {item}!'
```

lab7 Q5:Cucumber

Cucumber is a card game. Cards are positive integers (no suits). Players are numbered from 0 up to `players` (0, 1, 2, 3 in a 4–player game).

In each `Round`, the players each `play` one card, starting with the `starter` and in ascending order 升序排列 (player 0 follows player 3 in a 4–player game). If the `card` played is as high or higher than the `highest` card played so far, that player takes control.卡片高于最高开始控制 The winner is the last player who took control after every player has played once.每人参与游戏之后最后控制的是赢家

纯文本

```
class CucumberGame:
    """Play a round and return all winners so far. Cards is a list of pairs
    Each (who, card) pair in cards indicates who plays and what card they p
    >>> g = CucumberGame()
    >>> g.play_round(3, [(3, 4), (0, 8), (1, 8), (2, 5)])
    >>> g.winners
    [1]
    >>> g.play_round(1, [(3, 5), (1, 4), (2, 5), (0, 8), (3, 7), (0, 6), (1
    It is not your turn, player 3
    It is not your turn, player 0
    The round is over, player 1
    >>> g.winners
```

```
    [1, 3]
    >>> g.play_round(3, [(3, 7), (2, 5), (0, 9)]) # Round is never complete
    It is not your turn, player 2
    >>> g.winners
    [1, 3]
    """

    def __init__(self):
        self.winners = []

    def play_round(self, starter, cards):
        r = Round(starter)
        for who, card in cards:
            try:
                r.play(who, card)
            except AssertionError as e:
                print(e)
        if r.winner != None:
            self.winners.append(r.winner)

class Round:
    players = 4

    def __init__(self, starter):
        self.starter = starter
        self.next_player = starter
        self.highest = -1
        self.winner = None

    def play(self, who, card):
        assert not self.is_complete(), f'The round is over, player {who}'
        assert who == self.next_player, f'It is not your turn, player {who}
        self.next_player = (who+1) % self.players #一直转，加winner到字典
        if card >= self.highest:
            self.control = who #不用self.winner是因为字典只能加一个
            self.highest = card
        if self.is_complete():
            self.winner=self.control #将这一个传回winner字典

    def is_complete(self):
        """ Checks if a game could end. """
        return self.next_player==self.starter and self.highest > -1
        #不能self.winner != None还没传进去
```

HW6 Q2: Vending Machine 自动售卖机

In this question you'll create a vending machine that only outputs a single product and provides change when needed.输出一个物品

Create a class called `VendingMachine` that represents a vending machine for some product. A `VendingMachine` object returns strings describing its interactions. Remember to match *exactly* the strings in the doctests -- including punctuation and spacing! 创建代表自动贩卖机的类，与 doctests 中 string 完全匹配

Fill in the `VendingMachine` class, *adding attributes and methods as appropriate*, such that its behavior matches the following doctests:

纯文本

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

    >>> v = VendingMachine('candy', 10)
    >>> v.vend()
    'Nothing left to vend. Please restock.'
    >>> v.add_funds(15)
    'Nothing left to vend. Please restock. Here is your $15.'
    >>> v.restock(2)
    'Current candy stock: 2'
    >>> v.vend()
    'Please add $10 more funds.'
    >>> v.add_funds(7)
    'Current balance: $7'
    >>> v.vend()
    'Please 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)
    'Nothing left to vend. Please restock. 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 ***"
    def __init__(self,product,price):
        self.product = product
        self.price = price
        self.stock=0
        self.balance=0

    def restock(self,n):
        self.stock+=n
        return f'Current {self.product} stock: {self.stock}'

    def add_funds(self,k):
        if self.stock==0:
            return f'Nothing left to vend. Please restock. Here is your ${k
        else:
            self.balance += k #balance是对于这个商品
            return f'Current balance: ${self.balance}'

    def vend(self):
        if self.stock==0:
            return f'Nothing left to vend. Please restock.'
        elif self.price>self.balance:
            return f'Please add ${self.price - self.balance} more funds.'
        elif self.price==self.balance:
            self.stock -= 1
            self.balance = 0
            return f'Here is your {self.product}.'
        elif self.price<self.balance:
            self.stock -= 1
            c=self.balance
            self.balance = 0
            return f'Here is your {self.product} and ${(c-self.price)} chan
```

hw6 Q6: Next Virahanka Fibonacci Object❓

Implement the `next` method of the `VirFib` class. For this class, the `value` attribute is a Fibonacci number. The `next` method returns a `VirFib` instance whose `value` is the next Fibonacci number. The `next` method should take only constant time.实现 next 方法，值属性是斐波那契数，next 方法花固定时间

Note that in the doctests, nothing is being printed out. Rather, each call to `.next()` returns a `VirFib` instance. The way each `VirFib` instance is displayed is determined by the return value of its `__repr__` method. 每次调用`.next()`返回 VirFib 实例，不通过打印而通过`__repr__`来显示值

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. 通过 next 里一个新实例来追踪之前数字

纯文本

```
class VirFib():
    """A Virahanka Fibonacci number.

    >>> start = VirFib()
    >>> start
    VirFib object, value 0
    >>> start.next()
    VirFib object, value 1
    >>> start.next().next()
    VirFib object, value 1
    >>> start.next().next().next()
    VirFib object, value 2
    >>> start.next().next().next().next()
    VirFib object, value 3
    >>> start.next().next().next().next().next()
    VirFib object, value 5
    >>> start.next().next().next().next().next().next()
    VirFib object, value 8
    >>> start.next().next().next().next().next().next() # Ensure start isn'
    VirFib object, value 8
    """

    def __init__(self, value=0):
        self.value = value
```

```
    def next(self):
        "*** YOUR CODE HERE ***"
        if self.value == 0:
            value = VirFib(1)
        else:
            value = VirFib(self.value + self.previous)
        value.previous = self.value
        return value

    def __repr__(self):
        return "VirFib object, value " + str(self.value)
```

lab8 Q5: Making Cards

To play a card game, we're going to need to have cards, so let's make some! We're gonna implement the basics of the `Card` class first.

First, implement the `Card` class' constructor in `classes.py`. This constructor takes three arguments:

- a string as the `name` of the card
- an integer as the `attack` value of the card
- an integer as the `defense` value of the card

Each `Card` instance should keep track of these values using instance attributes called `name`, `attack`, and `defense`.

You should also implement the `power` method in `Card`, which takes in another card as an input and calculates the current card's power. Refer to the Rules of the Game if you'd like a refresher on how power is calculated.

纯文本

```
class Card:
    cardtype = 'Staff'

    def __init__(self, name, attack, defense):
        """
        Create a Card object with a name, attack,
        and defense.
```

```
        >>> staff_member = Card('staff', 400, 300)
        >>> staff_member.name
        'staff'
        >>> staff_member.attack
        400
        >>> staff_member.defense
        300
        >>> other_staff = Card('other', 300, 500)
        >>> other_staff.attack
        300
        >>> other_staff.defense
        500
        """
        "*** YOUR CODE HERE ***"
        self.name = name
        self.attack = attack
        self.defense = defense

    def power(self, opponent_card):
        """
        Calculate power as:
        (player card's attack) - (opponent card's defense)
        >>> staff_member = Card('staff', 400, 300)
        >>> other_staff = Card('other', 300, 500)
        >>> staff_member.power(other_staff)
        -100
        >>> other_staff.power(staff_member)
        0
        >>> third_card = Card('third', 200, 400)
        >>> staff_member.power(third_card)
        0
        >>> third_card.power(staff_member)
        -100
        """
        "*** YOUR CODE HERE ***"
        return self.attack-opponent_card.defence
```

lab8 Q6: Making a Player

Now that we have cards, we can make a deck 做一副牌, but we still need players to actually use them. We'll now fill in the implementation of the `Player` class.

A `Player` instance has three instance attributes:

- `name` is the player's name. When you play the game, you can enter your name, which will be converted into a string to be passed to the constructor.
- `deck` is an instance of the `Deck` class. You can draw from it using its `.draw()` method.抽牌方法
- `hand` is a list of `Card` instances. Each player should start with 5 cards in their hand, drawn from their `deck`. Each card in the hand can be selected by its index in the list during the game. When a player draws a new card from the deck, it is added to the end of this list.一开始有 5 张牌，选择索引，抽一张牌从 deck 时加在 hand list

Complete the implementation of the constructor for `Player` so that `self.hand` is set to a list of 5 cards drawn from the player's `deck`. 建议一个 player.deck 的 5 张牌列表

Next, implement the `draw` and `play` methods in the `Player` class. The `draw` method draws a card from the deck and adds it to the player's hand. The `play` method removes and returns a card from the player's hand at the given index.

纯文本

```
class Player:
    def __init__(self, deck, name):
        """Initialize a Player object.
        A Player starts the game by drawing 5 cards from their deck. Each t
        a Player draws another card from the deck and chooses one to play.
        >>> test_card = Card('test', 100, 100)
        >>> test_deck = Deck([test_card.copy() for _ in range(6)])
        >>> test_player = Player(test_deck, 'tester')
        >>> len(test_deck.cards)
        1
        >>> len(test_player.hand)
        5
        """
        self.deck = deck
        self.name = name
        "*** YOUR CODE HERE ***"
        self.hand=[self.deck.draw() for i in range(5)]


    def draw(self):
        """Draw a card from the player's deck and add it to their hand.
        >>> test_card = Card('test', 100, 100)
```

```
        >>> test_deck = Deck([test_card.copy() for _ in range(6)])
        >>> test_player = Player(test_deck, 'tester')
        >>> test_player.draw()
        >>> len(test_deck.cards)
        0
        >>> len(test_player.hand)
        6
        """
        assert not self.deck.is_empty(), 'Deck is empty!'
        "*** YOUR CODE HERE ***"
        self.hand.append(self.deck.draw())

    def play(self, index):
        """Remove and return a card from the player's hand at the given IND
        >>> from cards import *
        >>> test_player = Player(standard_deck, 'tester')
        >>> ta1, ta2 = TACard("ta_1", 300, 400), TACard("ta_2", 500, 600)
        >>> tutor1, tutor2 = TutorCard("t1", 200, 500), TutorCard("t2", 600
        >>> test_player.hand = [ta1, ta2, tutor1, tutor2]
        >>> test_player.play(0) is ta1
        True
        >>> test_player.play(2) is tutor2
        True
        >>> len(test_player.hand)
        2
        """
        "*** YOUR CODE HERE ***"
        return self.hand.pop(index) #用pop的话直接输入index就行了，不用remove的
```

lab8 Q7: AIs: Resourceful Resources

In the `AICard` class, implement the `effect` method for AIs. An `AICard` will allow you to add the top two cards of your deck to your hand via `draw`ing from your deck. 通过 deck.draw() 加两个最前的卡

> Once you have finished writing your code for this problem, set `implemented` to `True` so that the text is printed when playing an `AICard`! *This is specifically for the* `AICard` *!* For future questions, make sure to look at the problem description carefully to know when to reassign any pre-designated variables.

```
纯文本
```

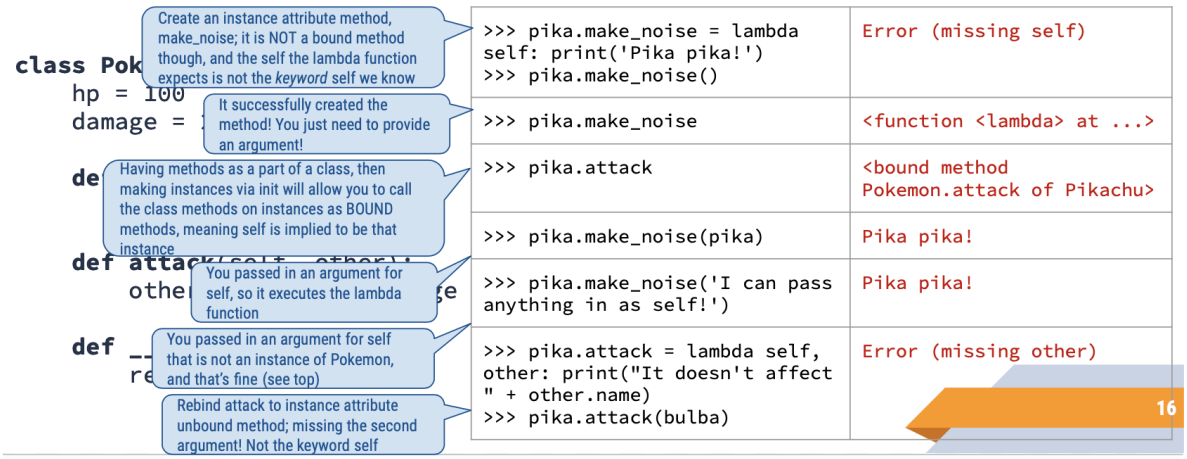
```
class AICard(Card):
    cardtype = 'AI'

    def effect(self, opponent_card, player, opponent):
        """
        Add the top two cards of your deck to your hand via drawing.
        Once you have finished writing your code for this problem,
        set implemented to True so that the text is printed when
        playing an AICard.

        >>> from cards import *
        >>> player1, player2 = Player(standard_deck.copy(), 'p1'), Player(s
        >>> opponent_card = Card("other", 500, 500)
        >>> test_card = AICard("AI Card", 500, 500)
        >>> initial_deck_length = len(player1.deck.cards)
        >>> initial_hand_size = len(player1.hand)
        >>> test_card.effect(opponent_card, player1, player2)
        AI Card allows me to draw two cards!
        >>> initial_hand_size == len(player1.hand) - 2
        True
        >>> initial_deck_length == len(player1.deck.cards) + 2
        True
        """
        "*** YOUR CODE HERE ***"
        implemented = False
        player.draw() #不能super().draw(共有)，引用的是card不是plaer
        player.draw() #类似猫叫两声
        implemented=True #按照题意
        # You should add your implementation above this.
        if implemented:
            print(f"{self.name} allows me to draw two cards!")
```

lab8 Q8: Tutors: Sneaky Search

In the `TutorCard` class, implement the `effect` method for Tutors. A `TutorCard` will add a copy of the first card in your hand to your hand, at the cost of automatically losing the current round. Note that if there are no cards in hand, a `TutorCard` will not add any cards to the hand, but must still lose the round.

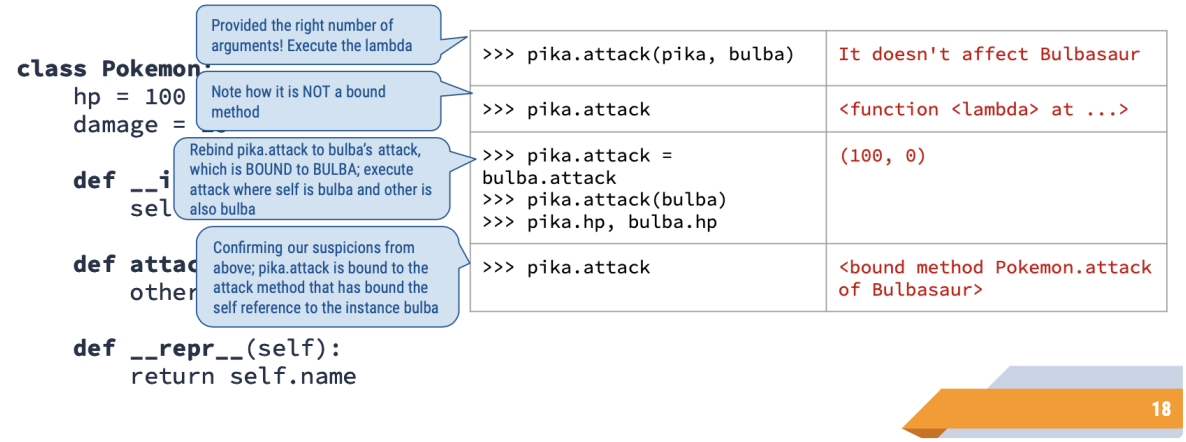
```
纯文本
class TutorCard(Card):
    cardtype = 'Tutor'

    def effect(self, opponent_card, player, opponent):
        """
        Add a copy of the first card in your hand
        to your hand, at the cost of losing the current
        round. If there are no cards in hand, this card does
        not add any cards, but still loses the round.  To
        implement the second part of this effect, a Tutor
        card's power should be less than all non-Tutor cards.

        >>> from cards import *
        >>> player1, player2 = Player(standard_deck.copy(), 'p1'), Player(s
        >>> opponent_card = Card("other", 500, 500)
        >>> test_card = TutorCard("Tutor Card", 10000, 10000)
        >>> player1.hand = [Card("card1", 0, 100), Card("card2", 100, 0)]
        >>> test_card.effect(opponent_card, player1, player2)
        Tutor Card allows me to add a copy of a card to my hand!
        >>> print(player1.hand)
        [card1: Staff, [0, 100], card2: Staff, [100, 0], card1: Staff, [0,
        >>> player1.hand[0] is player1.hand[2] # must add a copy!
        False
        >>> player1.hand = []
        >>> test_card.effect(opponent_card, player1, player2)
        >>> print(player1.hand) # must not add a card if not available
        []
        >>> test_card.power(opponent_card) < opponent_card.power(test_card)
        True
        """
        added = False
        if len(player.hand) > 0:
            player.hand.extend([player.hand[0].copy()])
            #extend(lst要[])/append(直接el)或直接+
            added = True
        # You should add your implementation above this.
        if added:
            print(f"{self.name} allows me to add a copy of a card to my han

    def power(self, opponent_card):
        return -float("inf")
```

I Choose You! Part 3 (solution)



第一行和最后一行易错：不再是之前绑定的函数，新定义了 lambda，lambda 传入 self 不是直接 self 的 pika，不传入的话会报错。

I Choose You! Part 4 (solution)



pika.attack 重被定义为 bulba 的 attck 方法，不再是 lambda 了，也不用传入两个了

马戏团

Practice Problem: Circus!

```
>>> cirque_soleil = Circus('Cirque Du Soleil', 'Montreal')
>>> acrobat = Performer('acrobat', 'fly through the air')
>>> acrobat.performances
    0
>>> cirque_soleil.recruit(acrobat)
>>> cirque_soleil.run_show()
    The acrobat will now fly through the air!
>>> acrobat.performances
    1
>>> tigers = Performer('tigers', 'jump through rings of
fire')
>>> cirque_soleil.recruit(tigers)
>>> len(cirque_soleil.performers)
    2
```

```
>>> cirque_soleil.run_show()
    The acrobat will now fly through the air!
    The tigers will now jump through rings of fire!
>>> print(acrobat.performances)
    2
>>> flying_circus  = TravelCircus('Flying Circus',
'Oakland')
>>> cirque_soleil.move('Boston')
>>> flying_circus.move('Denver')
>>> cirque_soleil.city
    'Montreal'
>>> flying_circus.city
    'Denver'
```

招募 1performer，run_show 写一句 perform 里的话，

招募两人，len 长度 2，提示马戏团下 performers 有一个[]而非字典，填在马戏图 init 下

由于[]，要 for i in []，run 方法下写 i.perform()，且 run 之后 performances+1 必然初始化 0 后面 +1，在 perform 里加

travel.move 方法输入 self 和 new city，self 匹配 travel(circus)，原 city 变为 new city，不匹配仍原来的

Circus: Skeleton

```
class Circus:
    can_move = False

    def __init__(self, name, city)
        self.name = name
        self.city = city
        self.performers = []

    def recruit(self, performer):
        self.performers.append(performer)

    def run_show(self):
        for p in self.performers:
            p.perform()

    def move(self, new_city):
        if can_move:
            self.city = new_city
```

```
class Performer:
    def __init__(self, role, act):
        self.role = role
        self.act = act
        self.performances = 0

    def perform(self):
        print(f"The {self.role} will now {self.act}!")
        self.performances += 1


class TravelCircus(Circus):
    can_move = True
```