

Python 程序设计 作业 2

注意事项:

- (1) 作业提交截止日期: 2020.06.13, 12:00pm, 迟交扣 20%, 缺交 0 分。
- (2) 提交方法: **blackboard**。
- (3) 提交要求: 作业答案+源代码, 打包上传, 命名: 学号_姓名_作业_2。
- (4) 答案包括问答题的解释, 代码题的解题思路、运行结果等。
- (5) 如同一题目需要多个 Python 文件执行, 请在文件名加 problem#, 如 problem1_file1.py, problem1_file2.py。
- (6) 禁止代码抄袭; 一经发现, 抄袭者和提供代码者统一 0 分处理!

1. 完成课程讲义中状态机的实现。

继承 inheritance

父类 ← 子类

```
class SM:
    start_state = None # default start state

    def transition_fn(self, s, x):
        '''s: the current state
        x: the given input
        returns: the next state'''
        raise NotImplementedError

    def output_fn(self, s):
        '''s: the current state
        returns: the corresponding output'''
        raise NotImplementedError

    def transduce(self, input_seq):
        ... {use transition_fn and output_fn}

class Accumulator(SM):
    start_state = 0

    def transition_fn(self, s, x):
        ...

    def output_fn(self, s):
        ...

class Binary_Addition(SM):
    start_state = (0,0) #(carry,digit)

    def transition_fn(self, s, x):
        ...

    def output_fn(self, s):
        ...
```

(1) 累加器 Accumulator

例子: 列表[-1, 2, 3, -2, 5, 6]中的数字相加, 我们得到和是 13。

```
output = Accumulator().transduce([-1,2,3,-2,5,6]) #output=[-1, 1, 4, 2, 7, 13]
```

(2) 二进制相加 Binary_Addition

例子: 011 和 001 相加我们得到 100。

```
output = Binary_Addition().transduce([(1,1),(1,0),(0,0)]) #output=[0, 0, 1]
```

(3) 反向器 Reverser

实现一个反向器, 逆向输出一个序列。输入的形式如下:

```
sequence1 + ['end'] + sequence2
```

其中 sequence1 是字符串列表, 'end'表示 sequence1 的终结, sequence2 为任意序列。反向器的作用如下: 当 sequence1 里的每个元素输入时, 反向器

输出 None。当['end']和 sequence2 的每个元素输入时，反向器依次反向输出 sequence1 中的元素；当没有元素输出时，则输出 None。

例子：

```
output = Reverser().transduce(['foo',' ', 'bar'] + ['end'] + list(range(5)))
```

```
#output = [None, None, None, 'bar', ' ', 'foo', None, None, None]
```

解释：输出中前三个 None 为输入 sequence1 时输出。从输入‘end’开始，sequence1 中的元素开始反向输出。sequence1 输出完毕后，继续输出 None（即输入 list(range(5))中的 2, 3, 4 时）。

更多测试例子：

```
output=Reverser().transduce(['foo', ' ', 'bar'] + ['end'] + ['end']*3 +list(range(2)))
```

```
#output = [None, None, None, 'bar', ' ', 'foo', None, None, None]
```

```
output = Reverser().transduce(list('the') + ['end'] + list(range(3)))
```

```
#output = [None, None, None, 'e', 'h', 't', None]
```

```
output=Reverser().transduce([] + ['end'] + list(range(5)))
```

```
#output = [None, None, None, None, None, None]
```

```
output=output=Reverser().transduce(list('nehznehS evol I') + ['end'] +  
list(range(15)))
```

```
#output = [None, None, None, None, None, None, None, None, None, None, None,  
None, None, None, None, None, 'I', ' ', 'I', 'o', 'v', 'e', ' ', 'S', 'h', 'e', 'n', 'z', 'h', 'e',  
'n', None]
```

实现要求：

- (1) 不同的状态机子类继承父类 SM；
- (2) 补充父类的 transduce 函数，子类不能重写 transduce 函数；
- (3) 补充每个子类的 transition_fn 和 output_fn 函数，其中累加器的实现代码如下：

```
class Accumulator(SM):
```

```
    start_state = 0
```

```
    def transition_fn(self, s, x):
```

```
        return s + x
```

```
    def output_fn(self, s):
```

```
        return s
```

建议：利用累加器的代码实现父类的 transduce 函数，再实现二进制相加和反向器的 transition_fn 和 output_fn 函数。

2. 纸牌游戏:

本次作业我们将设计和实现一个扑克牌游戏。游戏规则如下:

- (1) 游戏开始时, 扑克牌在两名玩家中平分 (扑克牌总共 52 张, 每位玩家分 26 张)。
- (2) 在每个回合中, 两个玩家都会展示手中的第一张牌。扑克牌等级较高的玩家将同时获得这两张牌, 并添加到自己的牌组中。然后玩家们重新调整手牌。
- (3) 若 (2) 中两位玩家的扑克牌等级一样 (即打平), 玩家们将展示手中的第二张牌。如果不再有平局, 扑克牌等级较高的玩家将同时获得这四张牌; 如果继续平局, 玩家们将展示手中的第三、四、.....张牌, 直到平局被打破。
- (4) 游戏持续到一个玩家收集完所有 52 张牌为止。若 (3) 中平局持续到一方无牌, 则还有手牌的玩家获胜; 若两个玩家都没有牌了, 则打平。

想一想, 为了实现上面这个游戏, 我们需要什么类和方法? 我们知道每张扑克牌都有一个花色和一个等级, 如红桃 K。因此, 创建一个“扑克牌”(Card)的类, 并包含这两个“属性”是有意义的。此外, 在我们的游戏中, “大”的扑克牌是指等级较高的牌; “打平”的牌是指等级相同的牌 (不管两张牌的花色)。所以我们需要自定义一些规则来衡量“大”、“小”、“相等”的牌。

每位玩家都有一“手”牌 (Hand), 是扑克牌的集合, 我们也可以将它写成一个类, 可以从一“手”牌中取出牌、或者洗牌。一“手”牌中如果包含所有的 52 张扑克牌, 叫牌堆 (Deck)。每位玩家 (Player) 有一“手”牌和名字。

最后我们需要一个类叫 Game, 实现游戏的逻辑。

作业任务:

补充提供的文件 card_game.py 中 Card, Hand, Deck, Player, Game 类的实现 (标注“补充代码”处)。测试代码以验证正确性。