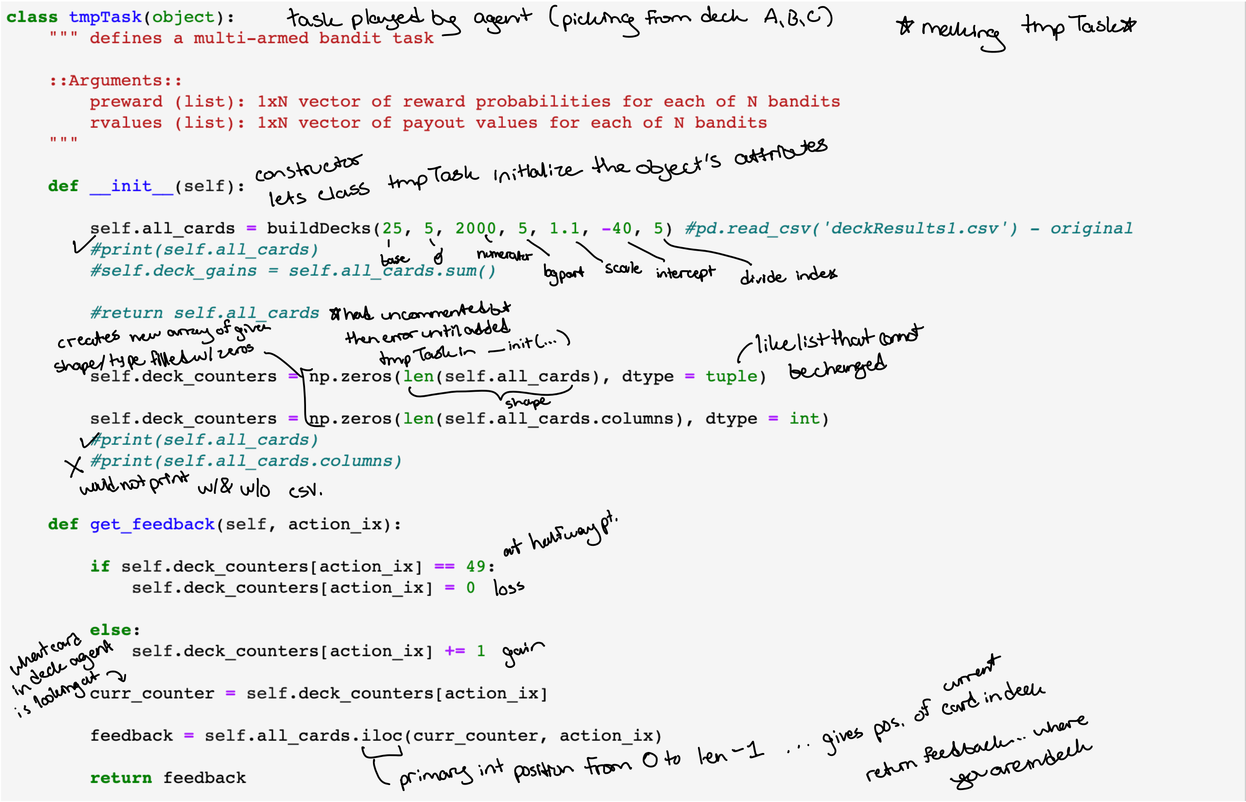
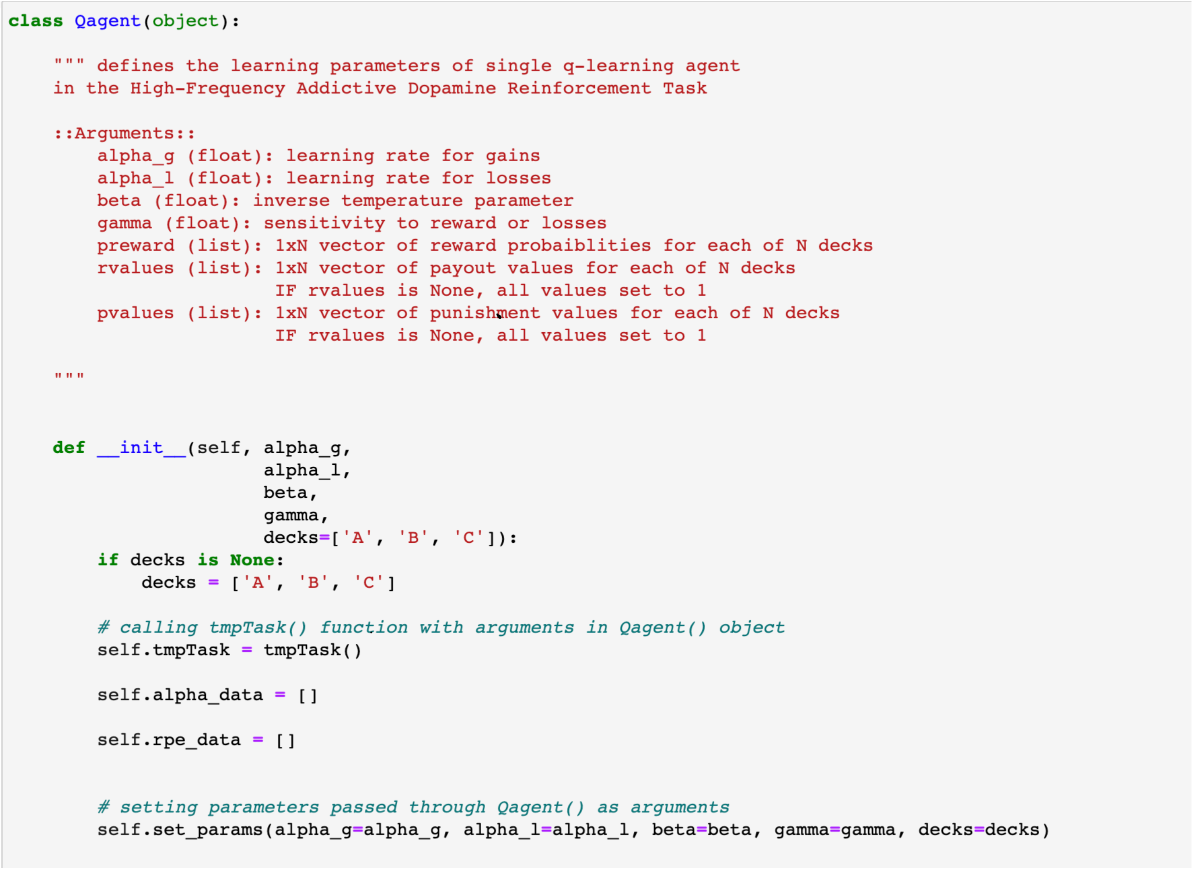


```
class tmpTask(object):      task played by agent (picking from deck A,B,C)      ★ making tmp Task★
    """ defines a multi-armed bandit task

    ::Arguments::
        preward (list): 1xN vector of reward probabilities for each of N bandits
        rvalues (list): 1xN vector of payout values for each of N bandits
    """
                               constructor
    def __init__(self):         lets class tmpTask initialize the object's attributes

        self.all_cards = buildDecks(25, 5, 2000, 5, 1.1, -40, 5) #pd.read_csv('deckResults1.csv') - original
     ✓  #print(self.all_cards)            base σ  numerator  lgpart  scale  intercept  divide index
        #self.deck_gains = self.all_cards.sum()

        #return self.all_cards ★had uncommented but
   creates new array of given      then error until added
   shape/type filled w/ zeros        tmpTask in __init(...)
        self.deck_counters = np.zeros(len(self.all_cards), dtype = tuple)   like list that cannot
                                              shape                              be changed
        self.deck_counters = np.zeros(len(self.all_cards.columns), dtype = int)
     ✓  #print(self.all_cards)
     X  #print(self.all_cards.columns)
     would not print w/& w/o csv.

    def get_feedback(self, action_ix):
                                                       at halfway pt.
        if self.deck_counters[action_ix] == 49:
            self.deck_counters[action_ix] = 0  loss

        else:
   what card   self.deck_counters[action_ix] += 1  gain
   in deck agent
   is looking at
        curr_counter = self.deck_counters[action_ix]
                                                                               current
        feedback = self.all_cards.iloc(curr_counter, action_ix)   ... gives pos. of card in deck
                                                                       return feedback... where
        return feedback          primary int position from 0 to len-1            you are in deck
```



```
class Qagent(object):

    """ defines the learning parameters of single q-learning agent
    in the High-Frequency Addictive Dopamine Reinforcement Task

    ::Arguments::
        alpha_g (float): learning rate for gains
        alpha_l (float): learning rate for losses
        beta (float): inverse temperature parameter
        gamma (float): sensitivity to reward or losses
        preward (list): 1xN vector of reward probaiblities for each of N decks
        rvalues (list): 1xN vector of payout values for each of N decks
                        IF rvalues is None, all values set to 1
        pvalues (list): 1xN vector of punishment values for each of N decks
                        IF rvalues is None, all values set to 1

    """


    def __init__(self, alpha_g,
                       alpha_l,
                       beta,
                       gamma,
                       decks=['A', 'B', 'C']):
        if decks is None:
            decks = ['A', 'B', 'C']

        # calling tmpTask() function with arguments in Qagent() object
        self.tmpTask = tmpTask()

        self.alpha_data = []

        self.rpe_data = []


        # setting parameters passed through Qagent() as arguments
        self.set_params(alpha_g=alpha_g, alpha_l=alpha_l, beta=beta, gamma=gamma, decks=decks)
```