ICGE Module 4 Session 1

Object-oriented programming in Python

Imagine you want to simulate something:

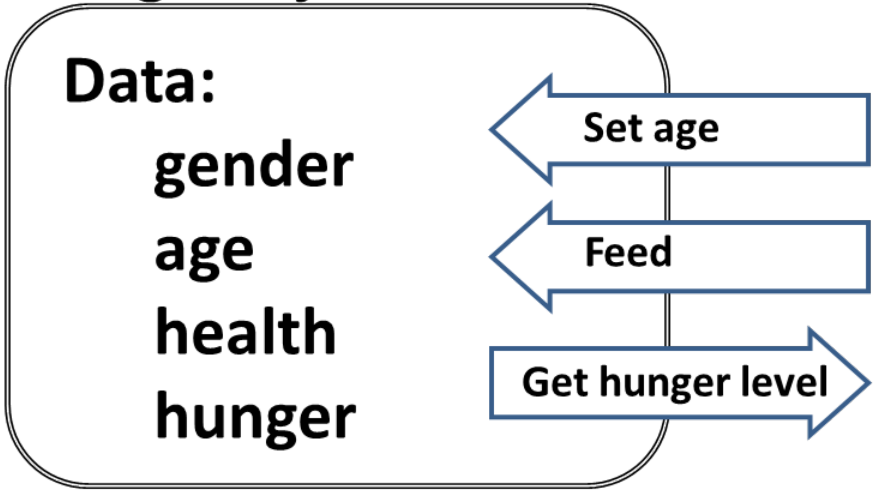
What will your program need to include?

- Variables to store the properties of each component (cards, frogs, etc.)
- Logic and math to change these variables (deal card, move frog, etc.)
- Steps to initialize and print out the properties of each component

What's the best way to organize these different pieces?

“Object-oriented” programming organizes your program around the natural “objects” involved

Frog “Object"



Data:

gender

age

health

hunger

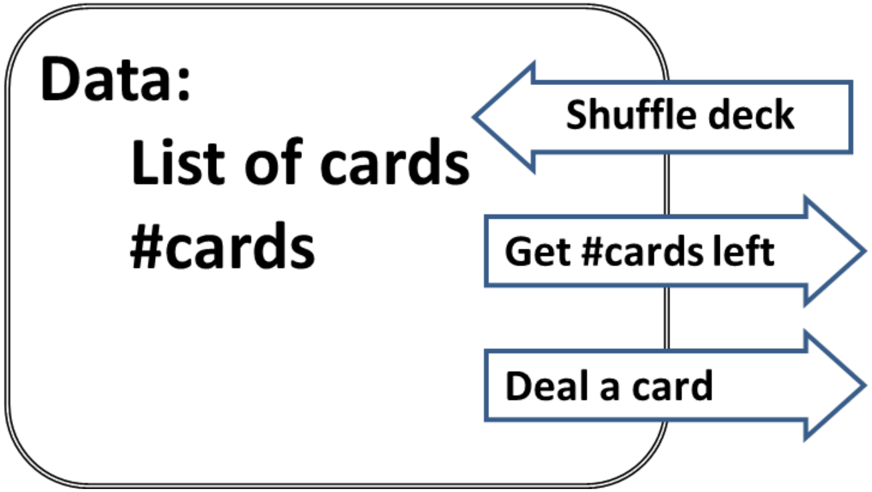
Set age

Feed

Get hunger level

Functions to operate on object

Deck “Object”



Data:

List of cards

#cards

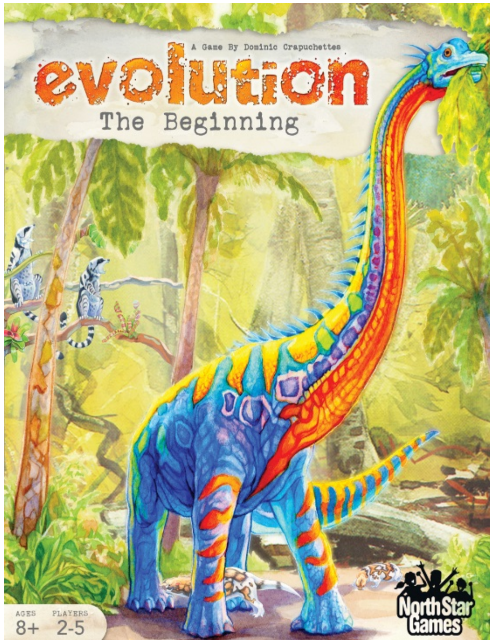
Shuffle deck

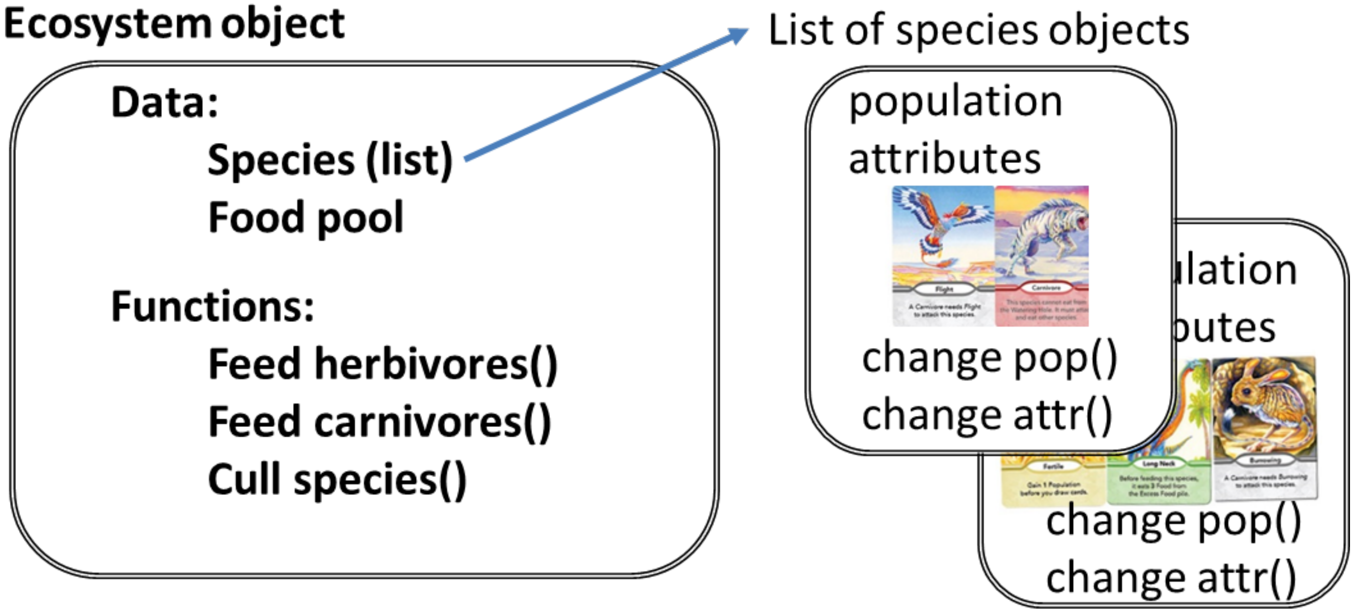
Get #cards left

Deal a card

Functions to operate on object

"OO" programming is an intuitive & fun approach to designing many types of simulation programs





Ecosystem object

Data:
Species (list)
Food pool

Functions:
Feed herbivores()
Feed carnivores()
Cull species()

List of species objects

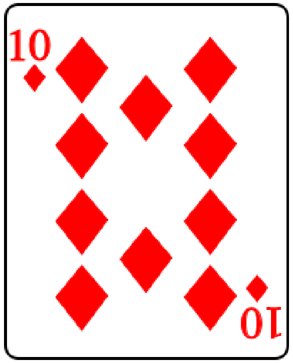
population
attributes
change pop()
change attr()

ılation
butes
change pop()
change attr()

Promised advantages of OO programming

- Simplifies programming by hiding the details of each component of the program
- Improved reliability since each class can be independently debugged
- Improved code reuse and sharing since you only need to remember the class "interface" and don't need to know the details of how the code is implemented

Let's try out two simple classes that implement a deck of playing cards and an individual card



Deck object		**Card object**	
Create deck	__init__()	Create card	__init__()
Shuffle deck	shuffle()	What type of card?	type()
Look at whole deck	printdeck()	What suit?	suit()
Deal a card	dealcard()	What is the card value? (depends on card game)	value()
How many cards left?	cardsleft()	Look at card	printcard()

Start idle, then open and run the file **`cards.py`**

Create a deck object and try some of its functions:

```
  adeck=deck()
  adeck.shuffle()
  adeck.printdeck()
  for i in range(15):
    acard=adeck.dealcard()
    print "acard:",acard.printcard()
  print "# left:",adeck.cardsleft()
  adeck.shuffle()
  adeck.printdeck()
  bdeck=deck()
  bdeck.printdeck()
```

Let's use this card "class" to build a simple card game and determine players' odds of winning

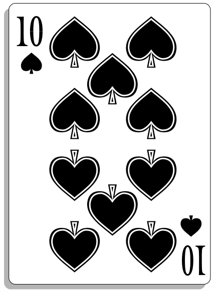
Rules:

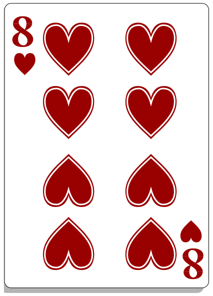
1. Player A gets 2 cards & Player B gets 1 card
2. Player A wins the hand if either card has a greater value than Player B's card
3. Play though entire deck and tally hands won

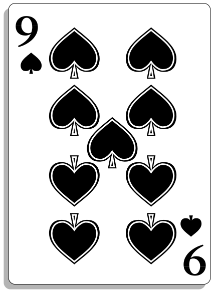
Player A

Player B

Hand 1:

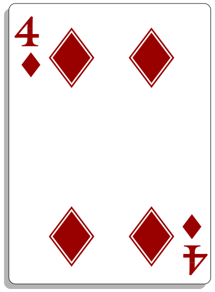


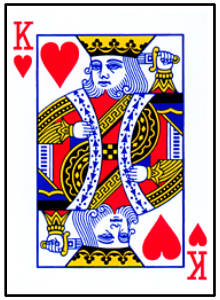


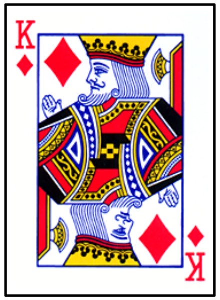


A wins

Hand 2:







B wins

Open a new window and enter the following code

Save the file with the name **game.py** in the same directory with the file **cards.py**

```
from __future__ import division
from cards import *
adeck=deck()
adeck.shuffle()
ascore=0
bscore=0
while adeck.cardsleft()>2:
    acard1=adeck.dealcard()
    acard2=adeck.dealcard()
    bcard=adeck.dealcard()
    if acard1.value()>bcard.value() or acard2.value()>bcard.value():
        ascore+=1
    else:
        bscore+=1
if ascore > bscore:
    print("Player A wins")
else:
    print("Player B wins")
```

Modification of program to run 10000 games and compute the fraction of time Player A wins

Program downloaded from CatCourses: **gameMC.py**

```
from __future__ import division
from cards import *
ntrials=10000
awins=0
for i in range(ntrials):
    adeck=deck()
    adeck.shuffle()
    ascore=0
    bscore=0
    while adeck.cardsleft()>2:
        acard1=adeck.dealcard()
        acard2=adeck.dealcard()
        bcard=adeck.dealcard()
        if acard1.value()>bcard.value() or acard2.value()>bcard.value():
            ascore+=1
        else:
            bscore+=1
    if ascore > bscore:
        awins+=1
print("Player A win percentage=",awins/ntrials)
```

The card values are set in the deck class and can be changed by editing the numerical values

Edit cards.py and look for following lines:

```
class deck:
    def __init__(self):
        self.deck=[]
        suits=['S','C','H','D']
        values={'A':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':
9,'10':10,'J':10,'Q':10,'K':10}
        types=['A','2','3','4','5','6','7','8','9','10','J','Q','K']
```

Player B wins when cards are equal, so giving more cards equal values will help this player. Edit the **cards.py** file and make this change (save your changes before rerunning **gameMC.py**)

```
values={'A':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':9,'9':9
,'10':10,'J':10,'Q':10,'K':10}
```

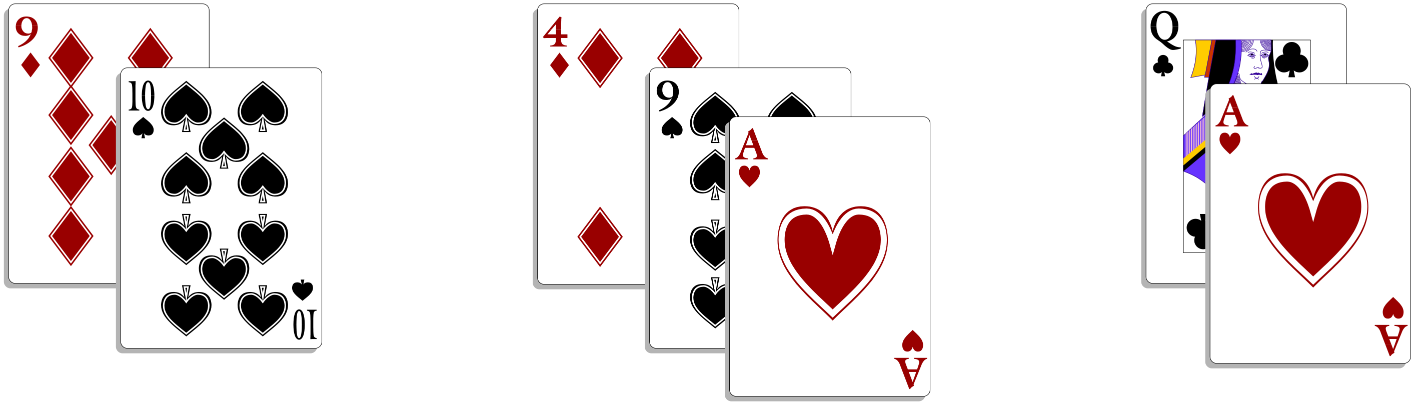
The most balanced version of the program I could find gave Player A a 50.5% chance of winning—can you do better?

Blackjack is a slightly more complex game where winning depends on the point value each hand

Goal: Get a set of cards totaling as close as possible to 21, without going over 21

Card values:

- 2, 3, 4, 5, 6, 7, 8, 10: Value of number
- J, Q, K: Count as 10
- A: Count as 1 or 11



Rules of blackjack (simplified)

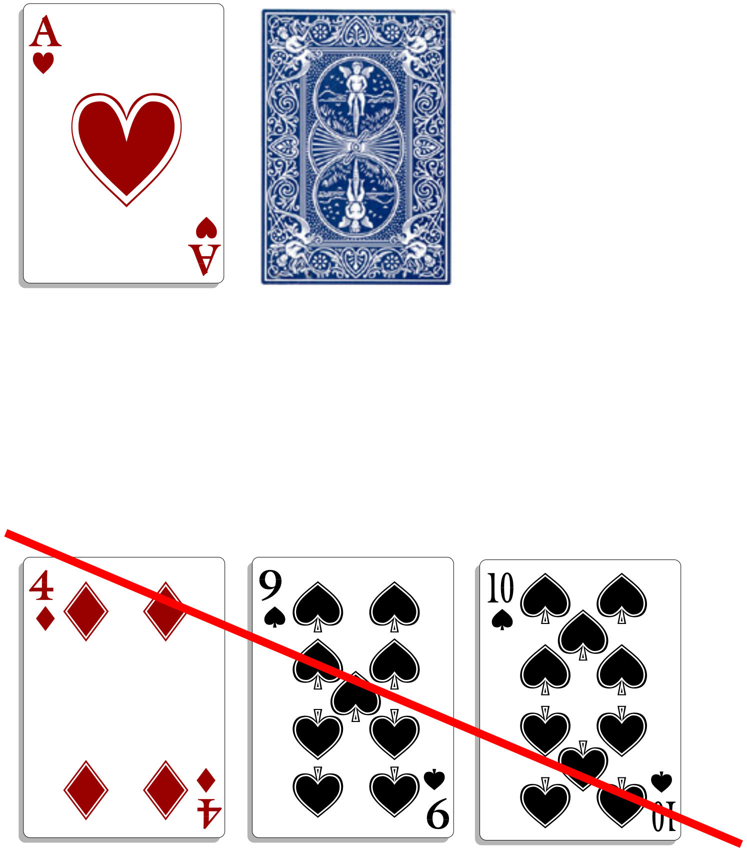
Players: 1 player and 1 dealer

Rules:

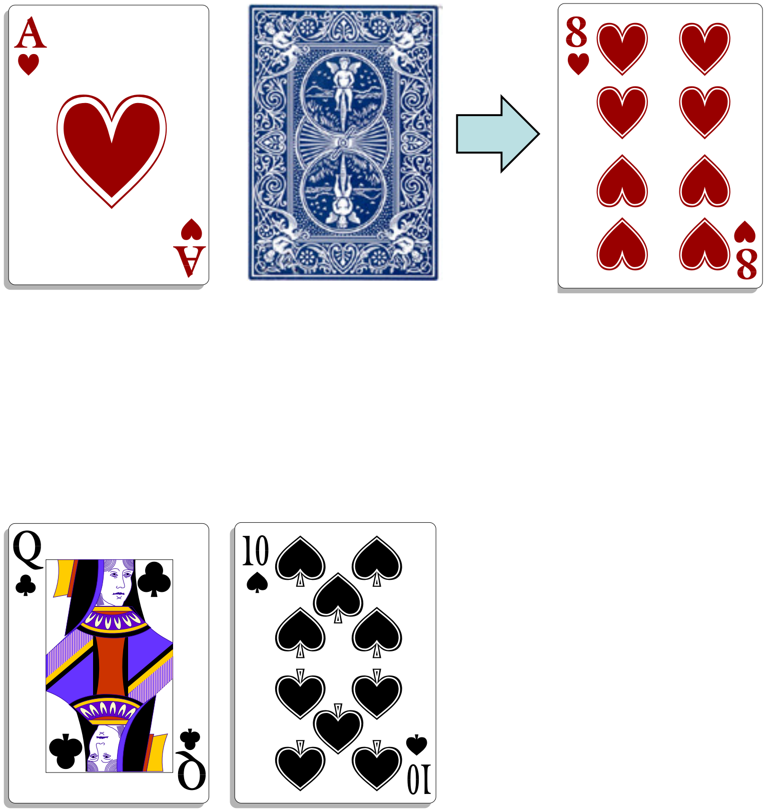
- Deal two cards to player & dealer with one of the dealer's cards face up
- Player goes first, requesting as many cards as he wants ("hits")
- If player goes over 21, he "busts" and dealer wins
- If player doesn't bust, dealer takes cards up to a cutoff of 17 or a bust
- Player & dealer compare scores; dealer wins in a tie

Two sample hands of Blackjack

DEALER



Player Busted !



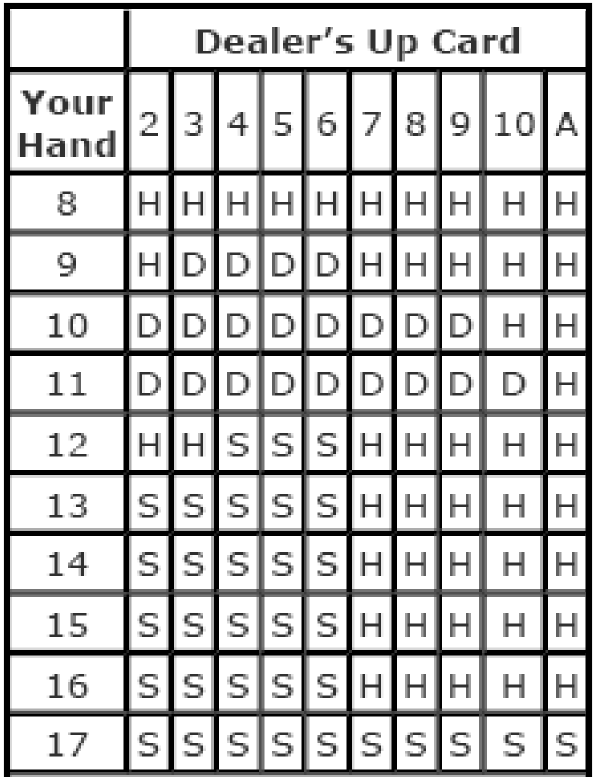
Player wins !

PLAYER

You can change the player's strategy and use Monte Carlo to test effectiveness

Things to change in strategy:

- Player's cutoff to take new card (recalling that dealer must "hold" at 17)
- How to use information about what cards the dealer is showing—Typically the higher the card the dealer is showing, more likely you will benefit by taking another card



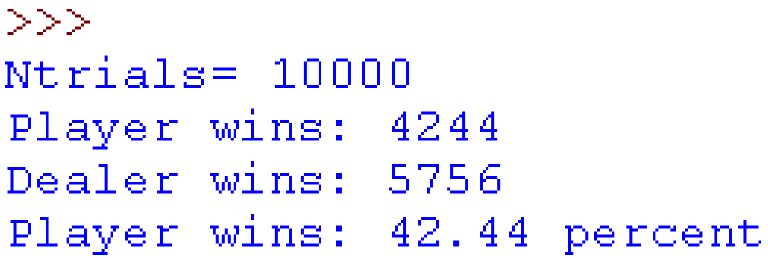
	Dealer's Up Card									
Your Hand	2	3	4	5	6	7	8	9	10	A
8	H	H	H	H	H	H	H	H	H	H
9	H	D	D	D	D	H	H	H	H	H
10	D	D	D	D	D	D	D	D	H	H
11	D	D	D	D	D	D	D	D	D	H
12	H	H	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
16	S	S	S	S	S	H	H	H	H	H
17	S	S	S	S	S	S	S	S	S	S

http://www.hitorstand.net/strategy.php

Program **blackjack.py** on CatCourses is a Monte Carlo simulation of the game

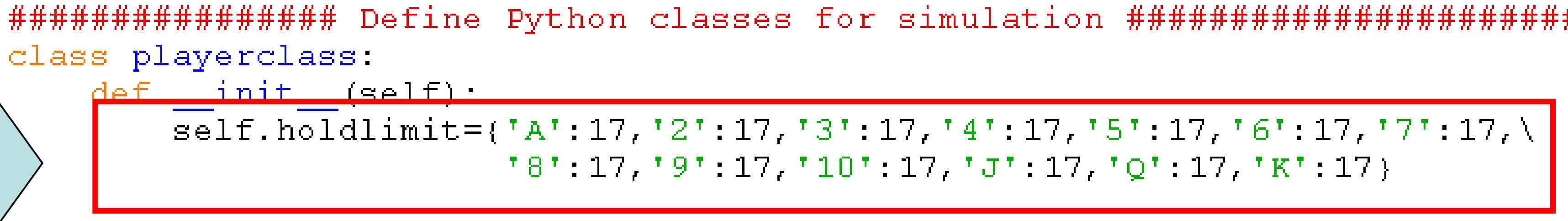
The program plays 10000 games of blackjack following the specified player strategy

Output:



```
>>>
Ntrials= 10000
Player wins:  4244
Dealer wins:  5756
Player wins:  42.44 percent
```

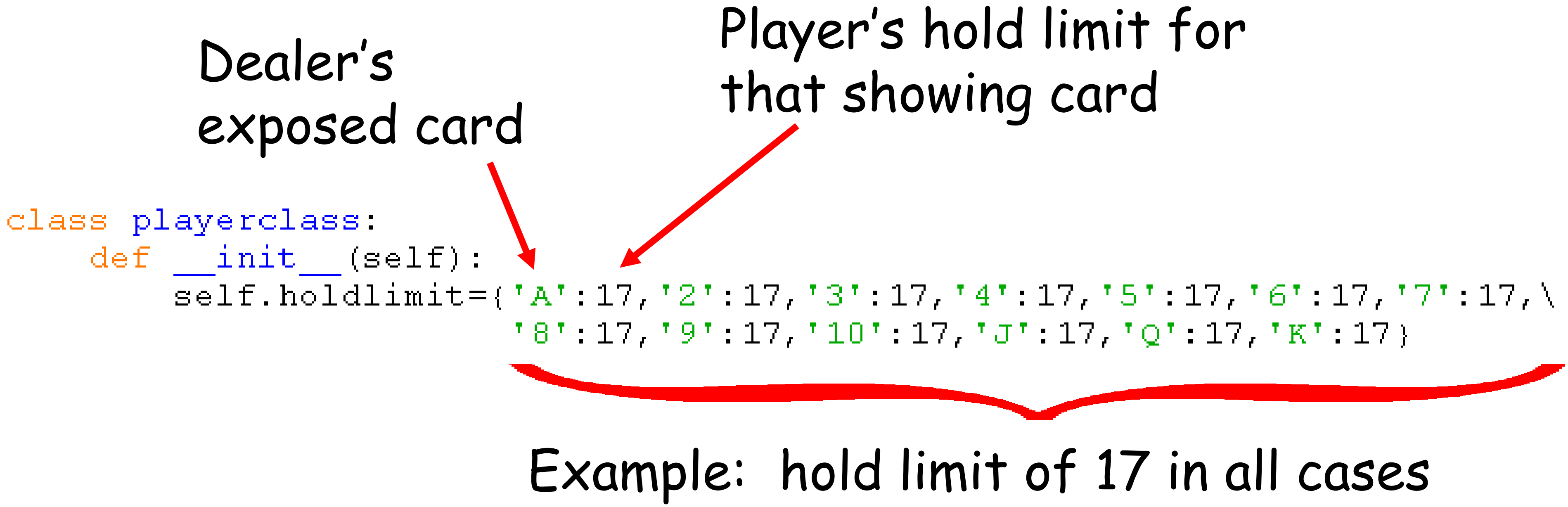
The player strategy can be modified by editing the **holdlimit** variable in the **playerclass**



```
################ Define Python classes for simulation ######################
class playerclass:
    def __init__(self):
        self.holdlimit={'A':17,'2':17,'3':17,'4':17,'5':17,'6':17,'7':17,\
                        '8':17,'9':17,'10':17,'J':17,'Q':17,'K':17}
```

You specify the player's strategy in terms of the hold value under different conditions

Code: **`blackjack.py`**

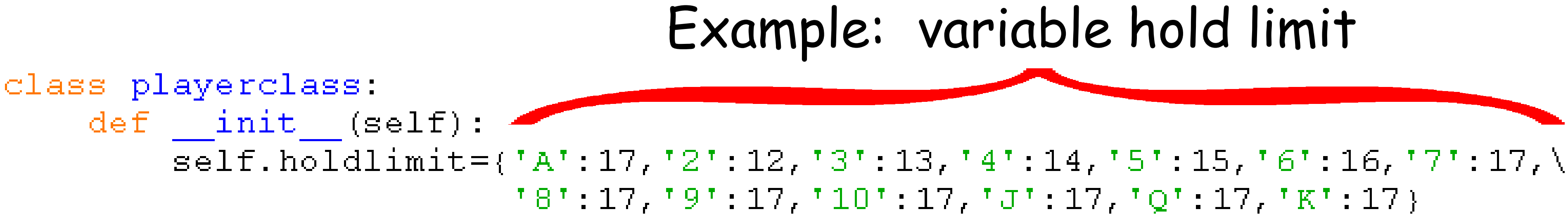


Dealer's exposed card

Player's hold limit for that showing card

```
class playerclass:
    def __init__(self):
        self.holdlimit={'A':17,'2':17,'3':17,'4':17,'5':17,'6':17,'7':17,\
                        '8':17,'9':17,'10':17,'J':17,'Q':17,'K':17}
```

Example: hold limit of 17 in all cases



Example: variable hold limit

```
class playerclass:
    def __init__(self):
        self.holdlimit={'A':17,'2':12,'3':13,'4':14,'5':15,'6':16,'7':17,\
                        '8':17,'9':17,'10':17,'J':17,'Q':17,'K':17}
```