

Assignment 13 – Classes

Object-Oriented Programming (OOP)

Object oriented programming groups related variables and functions into a data structure called an object.

Classes

Classes are templates or blueprints from which objects are made. All class names must start with an uppercase letter.

Objects

An object is a unique instance of a class made up of attributes (or properties) that define the object and methods which are used to modify the attributes as well as provide the functionality of the object.

Defining a Class

The Constructor

- The constructor is executed when a class is instantiated.
- Constructors must have at least 1 parameter, “*self*” but can have more, separated by commas.
- This constructor with the parameters is known as the class *signature*.
- Unlike some other programming languages, you can only have one constructor per class, there is no constructor overriding in python.

Methods

- To create a method in a class, use the keyword `def` followed by the name of the method and an optional list of parameters.
- Like the constructor, the method must take a reference to the object itself as the first parameter. By convention, this reference is named `self`.

Creating a Book Class

1. Create a new file book.py

```

1      #!/usr/bin/env python3
2
3
4      class Book:
5          def __init__(self, title, author, price, qty):
6              self.title = title
7              self.author = author
8              self.price = price
9              self.qty = qty
10
11         def get_discount_percent(self):
12             if self.qty > 100:
13                 return .50
14             elif self.qty > 50:
15                 return .25
16             elif self.qty > 25:
17                 return .15
18             else:
19                 return .10
20
21         def get_discount_amount(self):
22             return self.price * self.get_discount_percent()
23
24         def get_discounted_price(self):
25             return self.price - self.get_discount_amount()
26

```

Creating a Book Object

2. Create a new store.py file

```

1      #!/usr/bin/env python3
2
3      from book import Book
4
5      book1 = Book("The Green Mile", "Steven King", 19.95, 22)
6      book2 = Book("Ready Player One", "Ernest Cline", 14.95, 12)
7      book3 = Book("Harry Potter and the Sorcerer's Stone", "J.K. Rowling", 21.99, 46)
8
9      print("My Book Store")
10     print("Title:           {:s}".format(book1.title))
11     print("Author:          {:s}".format(book1.author))
12     print("Price:             {:.2f}".format(book1.price))
13     print("Discount Percent: {:.0f}%".format(book1.get_discount_percent() * 100))
14     print("Discount Amount:  {:.2f}".format(book1.get_discount_amount()))
15     print("Discounted Price: {:.2f}".format(book1.get_discounted_price()))
16

```

3. Run the Store File

Screen Capture #1 (1 points)

```
C:\Users\Saddleback\AppData\Local\Pro
My Book Store
Title:          The Green Mile
Author:         Steven King
Price:          19.95
Discount Percent: 10%
Discount Amount: 2.00
Discounted Price: 17.95

Process finished with exit code 0
```

Displaying Books

4. Continue working with the store.py file.
5. Code as follows:

```
1  #!/usr/bin/env python3
2
3  from book import Book
4
5
6  def print_book(books):
7      for i in range(len(books)):
8          book = books[i]
9          print("Title:          {:s}".format(book.title))
10         print("Author:         {:s}".format(book.author))
11         print("Price:          {:.2f}".format(book.price))
12         print("Discount Percent: {:.2f}%".format(book.get_discount_percent() * 100))
13         print("Discount Amount: {:.2f}".format(book.get_discount_amount()))
14         print("Discount Price: {:.2f}".format(book.get_discounted_price()))
15         print()
16     print()
17
18
19  def main():
20     print("My Book Store")
21     print()
22
23     books = (Book("The Green Mile", "Steven King", 19.95, 22),
24             Book("Ready Player One", "Ernest Cline", 14.95, 12),
25             Book("Harry Potter and the Sorcerer's Stone", "J.K Rowling", 21.99, 46))
26
27     print_book(books)
28
29
30  if __name__ == "__main__":
31     main()
```

6. Run:

Code Validation: book.py & store.py (3 points)

Screen Capture #2 (3 points)

```
C:\Users\Kelly\PycharmProjects\students\final\venv\Scripts
My Book Store

Title:          The Green Mile
Author:         Steven King
Price:          $19.95
Discount Percent: 10.00%
Discount Amount: $ 2.00
Discount Price: $17.95

Title:          Ready Player One
Author:         Ernest Cline
Price:          $14.95
Discount Percent: 10.00%
Discount Amount: $ 1.50
Discount Price: $13.45

Title:          Harry Potter and the Sorcerer's Stone
Author:         J.K Rowling
Price:          $21.99
Discount Percent: 15.00%
Discount Amount: $ 3.30
Discount Price: $18.69

Process finished with exit code 0
```

Object Composition

Object composition has a user-defined class made up of one or more user-defined classes

6. Create a new **deckofcards.py** file

a. Create a Card class as follows:

Note: You can pull a lot of this code from the Blackjack code lab from the deck.py file to save you some typing.

```

1  #!/usr/bin/env python3
2  import random
3
4
5  class Card:
6      def __init__(self, rank, suit):
7          self.rank = rank
8          self.suit = suit
9          self.value = self.get_value()
10
11     def get_value(self):
12         if self.rank == "Ace":
13             value = 11
14         elif self.rank == "Jack" or self.rank == "Queen" \
15             or self.rank == "King":
16             value = 10
17         else:
18             value = int(self.rank)
19         return value
20

```

7. Add a Deck class

```

21
22 class Deck:
23     def __init__(self):
24         self.deck = []
25
26         ranks = {"Ace", "2", "3", "4", "5", "6", "7", "8",
27                 "9", "10", "Jack", "Queen", "King"}
28         suits = {"Clubs", "Diamonds", "Hearts", "Spades"}
29         for suit in suits:
30             for rank in ranks:
31                 self.deck.append(Card(rank, suit))
32
33     def shuffle(self):
34         random.shuffle(self.deck)
35
36     def deal_card(self):
37         Card.card = self.deck.pop()
38         return Card.card
39

```

8. Add a Hand class

```
40
41 class Hand:
42     def __init__(self):
43         self.__cards = []
44
45     def add_card(self, card):
46         self.__cards.append(card)
47
48     def get_card(self, index):
49         return self.__cards[index]
50
51     def count(self):
52         return len(self.__cards)
53
54     def points(self):
55         points = 0
56         ace_count = 0
57         for card in self.__cards:
58             if card.rank == "Ace":
59                 ace_count += 1
60             points += card.value
61
62         # adjust points based on number of aces
63         if ace_count > 0 and points > 21:
64             points -= (ace_count * 10)
65         if ace_count > 1 and points <= 11:
66             points += 10
67         return points
68
```

9. Finally create a main method for testing.
10. Code as follows:

```
69
70 def main():
71     print("Cards - Tester")
72     print()
73
74     # create and shuffle deck
75     deck = Deck()
76     deck.shuffle()
77
78     # test hand
79     print("HAND")
80     hand = Hand()
81     for i in range(3):
82         hand.add_card(deck.deal_card())
83
84     for i in range(hand.count()):
85         card = hand.get_card(i)
86         print(card.rank + " of " + card.suit)
87
88     print("Hand points:", hand.points())
89
90
91 if __name__ == "__main__":
92     main()
93
```

11. Now Run the program to test:

Code Validation deckofcards.py (3 points)

Screen Capture #3 (3 points)

```
C:\Users\Saddleback\AppData\Local\Program
Cards - Tester

HAND
5 of Hearts
Jack of Spades
Jack of Diamonds
Hand points: 25

Process finished with exit code 0
```

Encapsulation

- Encapsulation is a fundamental concept of OOP
- Encapsulation can make your code more user-friendly and easy to use.
- By hiding properties and methods, encapsulation protects them from being used in ways that may make them not work properly.

12. Notice in our previous example, all variables are public...

```
class Card:
    def __init__(self, rank, suit):
        self.rank = rank
        self.suit = suit
        self.value = self.get_value()
```

13. Meaning they can be altered outside of the class and produce unexpected results.

14. Modify the main method as follows:

```
83
84     for i in range(hand.count()):
85         card = hand.get_card(i)
86
87         # modify the rank of a card
88         if i == 1:
89             card.rank = "Joker"
90
91         print(card.rank + " of " + card.suit)
92
93     print("Hand points:", hand.points())
94
```


15. Now Run the program see the results:

Screen Capture #4 (1 point)

```
C:\Users\Saddleback\AppData\Local\Programs\
Cards - Tester

HAND
2 of Spades
Joker of Spades
Jack of Spades
Hand points: 19

Process finished with exit code 0
```



Hiding Attributes

16. To protect against this, we'll make the card rank, value and suit all private so we have control of what those values get set to.

17. Update the Card class as follows:

```

1      #!/usr/bin/env python3
2      import random
3
4
5      class Card:
6          def __init__(self, rank, suit):
7              self.__rank = rank
8              self.__suit = suit
9              self.__value = self.get_value()
10
11         def get_value(self):
12             if self.__rank == "Ace":
13                 value = 11
14             elif self.__rank == "Jack" or self.__rank == "Queen" \
15                 or self.__rank == "King":
16                 value = 10
17             else:
18                 value = int(self.__rank)
19             return value
20

```

18. Now run the program again

- Notice we no longer have access to the rank variable thus preventing us from doing something the code does not expect

```

C:\Users\Saddleback\AppData\Local\Programs\Python\Python3
Cards - Tester
Traceback (most recent call last):

HAND
  File "C:/Users/Saddleback/PycharmProjects/test/deckofca
    main()
  File "C:/Users/Saddleback/PycharmProjects/test/deckofca
    print(card.rank + " of " + card.suit)
AttributeError: 'Card' object has no attribute 'rank'

Process finished with exit code 1

```

Accessing Hidden Attributes


19. Remove the code we added in step 14. If you run the program, we still have the same issue of not being able to access the rank and suit directly for display.

20. Update the Card class as follows:

```

19      return value
20
21      def get_rank(self):
22          return self.__rank
23
24      def get_suit(self):
25          return self.__suit
26

```




21. Update the points method in the hand class to use those getters:

```

60      def points(self):
61          points = 0
62          ace_count = 0
63          for card in self.__cards:
64              if card.get_rank() == "Ace":
65                  ace_count += 1
66              points += card.get_value()
67
68          # adjust points based on number of aces
69          if ace_count > 0 and points > 21:
70              points -= (ace_count * 10)
71          if ace_count > 1 and points <= 11:
72              points += 10
73          return points
74

```

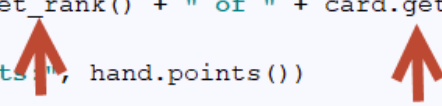


22. And finally update the main method:

```

84      # test hand
85      print("HAND")
86      hand = Hand()
87      for i in range(3):
88          hand.add_card(deck.deal_card())
89
90      for i in range(hand.count()):
91          card = hand.get_card(i)
92
93          print(card.get_rank() + " of " + card.get_suit())
94
95      print("Hand points ", hand.points())
96

```



23. Run once again to prove the card class still works.

The Pig Dice Game

I recommend trying to complete the game on your own but if you get stuck, code images for the pig dice game can be found on canvas.

Code Validation: All code files (3 points)

6. The output for your game should match the following:

Screen Capture #5 (3 points)

```
C:\Users\Saddleback\AppData\Local\Programs\Python\Python
Let's Play PIG!

* See how many turns it takes you to get to 20.
* Turn ends when you hold or roll a 1.
* If you roll a 1, you lose all points for the turn.
* If you hold, you save all points for the turn.

TURN 1
Roll or hold? (r/h): r
Die: 4
Roll or hold? (r/h): r
Die: 6
Roll or hold? (r/h): r
Die: 6
Roll or hold? (r/h): h
Score for turn: 16
Total score: 16

TURN 2
Roll or hold? (r/h): r
Die: 4
Roll or hold? (r/h): h
Score for turn: 4
Total score: 20

You finished in 2 turns!
Play again? (y/n):
```

Extra Credit

To get full points for each extra credit, you must include screen captures of the running output as well as the python (.py) code files. Extra credit is only available for assignments submitted on time.

Extra Credit #1 – Rectangle Calculator (+1 Extra Credit)

Create an object-oriented program that performs calculations on a rectangle.

```

Rectangle Calculator

Height:    10
Width:     20
Perimeter: 60
Area:      200
* * * * *
*
*
*
*
*
*
*
*
*
* * * * *

Continue? (y/n): y

Height:    5
Width:     10
Perimeter: 30
Area:      50
* * * * *
*
*
*
*
* * * * *

Continue? (y/n): n

Bye!

```

Specifications:

- Use a Rectangle class that provides attributes to store the height and width of a rectangle. This class should also provide methods that calculate the perimeter and area of the rectangle. In addition, it should provide a method that gets a string representation of the rectangle.
- When the program starts, it should prompt the user for height and width. Then, it should create a Rectangle object from the height and width and use the methods of that object to get the perimeter, area, and string representation of the object.

Extra Credit #2 – Travel Time Calculator (+1 Extra Credit)

Create an object-oriented program that creates a deck of cards, shuffles them, and deals the specified number of cards to the player.

```
Card Dealer

I have shuffled a deck of 52 cards.

How many cards would you like?: 7

Here are your cards:
Jack of Hearts
Jack of Diamonds
2 of Diamonds
6 of Spades
Jack of Spades
6 of Hearts
King of Diamonds

There are 45 cards left in the deck.

Good luck!
```

Specifications:

- Use a Card class to store the rank and suit for each card. In addition use a method to get a string representation for each card such as “Ace of Spades”, “2 of Spades”, etc.
- Use a Deck class to store the 52 cards in a standard playing deck (one card for each rank and suit):
 - Ranks: 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace
 - Suits: Clubs, Diamonds, Hearts, Spades
- The Deck class should include a method that shuffles the deck, a method that counts the number of cards in the deck, and a method that deals a card from the deck, which should reduce the count of the cards in the deck by 1.
- When the program starts, it should get a new deck of cards, shuffle them, and display a message that indicates the total number of cards in the deck. To shuffle the cards, you can use the shuffle function used in the Code Lab.
- The program should prompt the user for the desired number of cards. Then, it should deal the user the desired number of cards and display a message that indicates the number of cards left in the deck.