18 - Inheritance

How to use inheritance

Inheritance allows you to extend an existing class. This is great when you do not have access to the class itself or you simply want to add new features to it.

The class you are extending is called the superclass. The class you are creating the subclass.

To demonstrate inheritance we will design a class for a playing card.

Ex1: cards.py

```python
# File: cards.py

# ***** Class: Deck *****
class Card:
    ''' Represents a standard playing card.

    Attributes:
    rank: int, suit: int
    '''

    # Define class attributes. These are associated with the class itself.
    # The suits are by ascending order in Bridge.
    suits = ['Clubs', 'Diamonds', 'Hearts', 'Spades']

    # The ranks are simple mappings 1-13. The first element is None, # since no rank has value 0.
    ranks = [None, 'Ace', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King']

    def __init__(self, rank=2, suit=0):
        # Defaults to 2 of clubs.
        self.rank = rank
        self.suit = suit

    def __str__(self):
        return '%s of %s' % (Card.ranks[self.rank], Card.suits[self.suit])
```

Last updated: March 27, 2020 at 7:58 AM
You could then create a card, e.g. Queen of Hears as:

```python
def __lt__(self, other):
    # Arbitrarily compare by suit then rank.
    return (self.suit, self.rank) < (other.suit, other.rank)

queenOfHearts = Card(12, 2);
print(queenOfHearts) # Queen of Hearts
```

The next class is for representing a deck of cards. Here we use composition, since a deck has cards. This is a has-a relationship between Deck and Card.

```python
# ***** Class: Deck *****
class Deck:
    '''Represents a deck of 52 cards.

    Attributes:
    cards: list
    '''
    def __init__(self):
        '''Creates a deck of 52 cards.''
        # Create the list.
        self.cards = []

        # Populate the list.
        for suit in range(4):
            for rank in range(1,14):
                self.cards.append(Card(rank, suit))

    def __str__(self):
        '''The list of card descriptions.
        l = []
        for card in self.cards:
            l.append(str(card))

        # Join each element with the newline and return resultant string.
        return '\n'.join(l)

    def pop_card(self):
        '''Removes the last card from the cards list.''
```
# pop() removes last card, i.e. deals from the bottom of the deck.
return self.cards.pop()

def add_card(self, card):
    '''Appends the card to the end of the cards list.'''
    self.cards.append(card)

def shuffle(self):
    '''Randomly shuffles the cards list.'''
    random.shuffle(self.cards)

def sort(self):
    '''Sorts the cards list.''
    # The sort method uses the __lt__ method defined in the Card class.
    self.cards.sort()

def deal(self, hand, count):
    '''Deals out count cards representing a hand.''
    for _ in range(count):
        hand.add_card( self.pop_card() )

The third class we will add is the one that represents a Hand.

# ***** Class: Hand *****
class Hand(Deck):
    '''Represents a hand of playing cards.

    Arguments:
    cards: list - the list of playing cards
    label: str - a label for the hand
    '''

    # The subclass Hand inherits from Deck.
    # A Hand is like a Deck in that both have cards and the ability to
    # add or remove hands to name a few.

    # Hand inherits the __init__ from Deck, but that initializer is not
    # quite what we need. We don't have 52 cards in our hand afterall.
    # So, overwrite the parent class's __init__.
    def __init__(self, label=''):
```python
self.cards = []
self.label = label

qoh = Card(12, 2)
print(qoh)

d = Deck()
print(f'Original deck:
{d}')
print('---------------------')
d.shuffle()
print(f'Shuffled deck:
{d}')
print('---------------------')
d.sort()
print(f'Sorted deck:
{d}')
h = Hand('Player 1')
d.deal(h, 5)
print(h.label)
print(h)
```

`p1: dealHands.py`