1 Classes in Python

```
1  class Card:
2      # Define the suits
3      DIAMONDS = 1
4      CLUBS = 2
5      HEARTS = 3
6      SPADES = 4
7      SUITS = {
8          CLUBS: 'Clubs',
9          HEARTS: 'Hearts',
10         DIAMONDS: 'Diamonds',
11         SPADES: 'Spades'
12     }
13
14     # Define the names of special cards
15     VALUES = {
16         1: 'Ace',
17         11: 'Jack',
18         12: 'Queen',
19         13: 'King'
20     }
21
22     def __init__(self, suit, value):
23         # Save the suit and card value
24         self.suit = suit
25         self.value = value
26
27     def __lt__(self, other):
28         # Compare the card with another card
29         # (return true if we are smaller, false if
30         # we are larger, 0 if we are the same)
```

```
31          if self.suit < other.suit:
32              return True
33          elif self.suit > other.suit:
34              return False
35
36          if self.value < other.value:
37              return True
38          elif self.value > other.value:
39              return False
40
41          return 0
42
43      def __str__(self):
44          # Return a string description of ourself
45          if self.value in self.VALUES:
46              buf = self.VALUES[self.value]
47          else:
48              buf = str(self.value)
49          buf = buf + ' of ' + self.SUITS[self.suit]
50
51          return buf
```

We start off by defining some class constants to represent each suit, and a lookup table that makes it easy to convert them to the name of each suit. We also define a lookup table for the names of special cards (Ace, Jack, Queen and King).

The constructor, or `__init__` method, accepts two parameters, the suit and value, and stores them in member variables.

The special `__cmp__` method is called whenever Python wants to compare a Card object with something else. The convention is that this method should return a negative value if the object is less-than the other object, a positive value if it is greater, or zero if they are the same. Note that the other object

passed in to be compared against can be of any type but for simplicity, we assume it's also a Card object.

The special `__str__` method is called whenever Python wants to print out a Card object, and so we return a human-readable representation of the card.

Here is the class in action:

```
1  >>> card1 = Card(Card.SPADES, 2)
2  >>> print(card1)
3  2 of Spades
4  >>> card2 = Card(Card.CLUBS, 12)
5  >>> print(card2)
6  Queen of Clubs
7  >>> print(card1 > card2)
8  True
```

2 Standard Python Class Methods

Python classes have many standard methods, such as `__init__` which we saw above, that gets called when a new instance of the class is created. These are some of the more commonly-used ones:

- `__del__`: Called when an instance is about to be destroyed, which lets you do any clean-up e.g. closing file handles or database connections
- `__repr__ and __str__`: Both return a string representation of the object, but `__repr__` should return a Python expression that can be used to re-create the object. The more commonly used one is `__str__`, which can return anything.
- `__cmp__`: Called to compare the object with another object. Note that this is only used with Python 2.x. In Python 3.x, only rich comparison methods are used. Such as `__lt__`.
- `__lt__, __le__, __eq__, __ne__, __gt__ and __ge__`: Called to compare the object with another object. These will be called if defined, otherwise Python will fall-back to using `__cmp__`.

- `__hash__`: Called to calculate a hash for the object, which is used for placing objects in data structures such as sets and dictionaries.
- `__call__`: Lets an object be "called"e.g. so that you can write things like this: obj(arg1,arg2,...).

Python also lets you define methods that let an object act like an array (so you can write things like this: obj[2] = "foo"), or like a numeric type (so you write things like this: print(obj1 + 3*obj2).