

## Objectives

- Defining our own classes

## Lots of Different Ways to Do Things

- Input/Output files for Problem 2
- Writing numbers to file

## Lots of Different Ways to Do Things

- Input/Output files for Problem 2
  - Hardcode into file
  - Make a loop
  - Input from user
    - Change extension from “txt” to “dat”
    - Request output file
- Writing numbers to file
  - Use **str** constructor
  - Use format specifier

## Classes

- Made up of **data** and **methods**
- Methods are functions **within** a class
  - Putting a function inside a class makes it a **method**
- Methods are our API

## Defining a Card Class

- Create a class that represents a playing card
  - What do we need to represent a playing card?
  - What information do we need?



## Defining a Card Class

- Create a class that represents a playing card
  - Suite (hearts, diamonds, clubs, spades)
  - Rank
    - 2-10: numbered cards
    - 11: Jack
    - 12: Queen
    - 13: King
    - 14: Ace

Typically starts with  
a capital letter

- Syntax: **class** <class-name>:  
    <method definitions>

## Card Class (Partial)

```
class Card:
    """
    A class to represent a standard playing card. The ranks are ints:
    2-10 for numbered cards, 11=Jack, 12=Queen, 13=King, 14=Ace.
    The suits are strings: 'clubs', 'spades', 'hearts', 'diamonds'.
    """
    def __init__(self, rank, suit):
        """Constructor for class Card takes int rank and string suit."""
        self.rank = rank
        self.suit = suit

    def getRank(self):
        """Returns rank."""
        return self.rank

    def getSuit(self):
        """Returns suit."""
        return self.suit
```

Doc String

Methods with special meaning begin and end with two underscores

card.py

## Defining the Constructor

- `__init__` method is the **constructor**
  - In constructor, define **instance variables**
    - Data contained in every object
    - Also called **attributes** or **fields**
  - For our Card objects, the data is the **rank** and **suit**
    - First parameter of every method is **self** - pointer to the object that method acts on
- ```
def __init__(self, rank, suit):
    """Constructor for class Card takes int rank and string suit."""
    self.rank = rank
    self.suit = suit
```
- Instance variables

Oct 31, 2007

Sprenkle - CS111

8

## Using the Constructor

- In this case, constructor is called using
  - **Card(<rank>, <suit>)**
    - Do not pass anything for the **self** parameter
    - Python handles underneath, passing the parameter for us automatically
- Example:
  - **card = Card(2, "hearts")**
  - Underneath, Python passes **card** for us for **self** parameter

Oct 31, 2007

Sprenkle - CS111

9

## Another Special Method: `__str__`

- Returns a string that describes the object
  - Whenever you print an object, Python checks if you have defined the `__str__` method to see what should be printed
- ```
def __str__(self):
    """Returns a string describing the card as 'rank of suit'."""
    result = ""
    if self.rank == 11:
        result += "Jack"
    elif self.rank == 12:
        result += "Queen"
    elif self.rank == 13:
        result += "King"
    elif self.rank == 14:
        result += "Ace"
    else:
        result += str(self.rank)
    result += " of " + self.suit
    return result
```

Oct 31, 2007

Sprenkle - CS111

10

## Using the Card Class

```
def main():
    c1 = Card(14, "spades")
    print c1
    c2 = Card(13, "hearts")
    print c2
```

Invokes the `__str__` method

Outputs:

Ace of spades  
King of hearts

Oct 31, 2007

Sprenkle - CS111

11

## Example: Black Jack Value

- Add a method to the Card class called **blackJackValue** that returns the value of the card in the game of black jack.
  - Have Jack, Queen, and King be worth 10
  - Ace is worth 1
  - All the other cards have the value of their rank
- What is the method header?

Oct 31, 2007

Sprenkle - CS111

card2.py

12

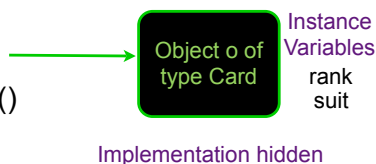
## Example: Rummy Value

- Add a method to the Card class called **rummyValue** that returns the value of the card in the game of Rummy

## Card API

## Card API

- Card(<rank>, <suit>)
- getRank()
- getSuit()
- blackJackValue()
- rummyValue()



## Using the Card class

- Now that we have the Card class, how can we use it?
- Can now make a **Deck** class
  - What data should a Deck contain?
  - How can we represent that data?
- First, write methods **\_\_init\_\_** and **\_\_str\_\_**
  - What do the function headers look like?

## Creating a Deck Class (Partial)

- List of Card objects

```
from card import *

class Deck:
    def __init__(self):
        self.cards = []
        for suit in ["clubs", "hearts", "diamonds", "spades"]:
            for rank in range(2, 15):
                self.cards.append(Card(rank, suit))

    def __str__(self):
        result = ""
        for c in self.cards:
            result += c.__str__() + "\n"
        return result
```

## Adding Deck Functionality

- Functionality:
  - Shuffle the cards
  - Deal one card
  - Number of cards remaining
- What do the method headers look like?
- Any special cases to check?

## Deck API

## Looking Ahead ...

- I'm going to a conference next Tuesday
  - Should have email access after I arrive
  - Ask questions before I leave!
- Guest lecturer for lab -- a different type of lab
- Wednesday: no class
  - finish up lab or study for second midterm
- Friday: midterm

## Midterm Topics

- Everything on first midterm
  - Keep using those building blocks
- Defining/using our own functions
- Creating/using modules
- Files - opening, reading, writing
- Lists - creating, processing, using
- Dictionaries - creating, processing, using
- Using an API
- Defining/using our own classes