Objectives

- Defining our own classes

Review

- When defining a function, how can we make a parameter have a *default value*?
- Compare some properties about dictionaries and lists
  - When should you use one over the other?

Abstractions

- Provide ways to think about program and its data
  - Get the jist without the details
- Examples we've seen
  - Functions and methods `encodeMessage(phrase, key)`
    - Used to perform some operation but we don't need to know how they're implemented
  - Dictionaries
    - Know they map keys to values
    - Don't need to know how the keys are organized/ stored in the computer's memory
  - Just about everything we do in this class…

Classes and Objects

- Provide an abstraction for how to organize and reason about data
- Example: `GraphWin` class
  - Had ***attributes*** (i.e., data or state) background color, width, height, and title
  - Each `GraphWin` object had these attributes
    - Each `GraphWin` object had its own values for these attributes
  - Used methods to modify the object's state.

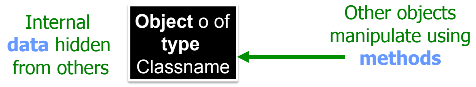
Defining Our Own Classes

- Often, we want to represent data or information that we do ***not*** have a way to represent using *built-in types* or *libraries*

- Classes provide way to *organize* and *manipulate* data
  - Organize: data structures used
    - E.g., ints, lists, dictionaries, other objects, etc.
  - Manipulate: methods

What is a Class?

- Defines a new ***data type***
- Defines the class's ***attributes*** (i.e., data) and ***methods***
  - Methods are like **functions** *within* a class and are the class's **API**



Defining a Card Class

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

Representing a Card object

- Every card has two attributes:
  - Suite (one of “hearts”, “diamonds”, “clubs”, “spades”)
  - Rank
    - 2-10: numbered cards
    - 11: Jack
    - 12: Queen
    - 13: King
    - 14: Ace

Defining a New Class

- Syntax:

Keyword

Typically starts with a capital letter

```
class <class-name>:
     <method definitions>
```

Card Class (Incomplete)

Doc String

```
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 the card’s rank."
        return self.rank

    def getSuit(self):
        "Returns the card’s suit."
        return self.suit
```

Methods

Methods are like *functions* defined in a *class*

card.py

Defining the Constructor

- `__init__` method is like the ***constructor***
- In constructor, define ***instance variables***
  - **Data** contained in every object
  - Also called **attributes** or **fields**
- Constructor **never *returns*** anything

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

Using the Constructor

```
def __init__(self, rank, suit):
```

- As defined, constructor is called using `Card(<rank>,<suit>)`
  - Do not *pass* anything for the `self` parameter
  - Python handles underneath, passing the parameter for us *automatically*

Object card of type Card

rank = ?
suit = ?

Using the Constructor

```
def __init__(self,
      rank, suit):
```

- As defined, 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")`
  - Creates a 2 of Hearts card
  - Python passes `card` as `self` for us

Object `card` of type `Card`

rank = 2
suit = "hearts"

Accessor Methods

- Need to be able to get information about the object

- Have `self` parameter
- Return data/ information

```
def getRank(self):
   "Returns the card's rank."
   return self.rank

def getSuit(self):
   "Returns the card's suit."
   return self.suit
```

- These will get called as `card.getRank()` and `card.getSuit()`
  - Python plugs `card` in for `self`

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
- `str(<object>)` also calls `__str__` method

```
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
```

Using the Card Class

Invokes the `__str__` method

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

Displays:

Ace of spades
King of hearts

Object `c1` of type `Card`

rank = 14
suit = "spades"

Object `c2` of type `Card`

rank = 13
suit = "hearts"

Example: Rummy Value

- Problem: Add a method to the `Card` class called **`rummyValue`** that returns the value of the card in the game of Rummy
- Procedure for defining a method (similar to functions)
  - What is the input?
  - What is the output?
  - What is the method header?
  - What does the method do?
- How do we call the method?

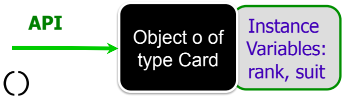
card2.py

Card API

- Based on what we've seen/done so far, what does the `Card` class's API look like?

Card API

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



API

Object o of type Card

Instance Variables: rank, suit

Implementation of methods is hidden

Defining a Card Class

- Create a class that represents a playing card
  - How can we represent a playing card?
  - What information do we need to represent a playing card?
- Do we ***need*** a class to represent a card?
  - Does any built-in data type naturally represent a card?

Using the `Card` class

- Now that we have the `Card` class, how can we use it?
- Let's write a simplified version of the game of War
  - Basically just part of a round
- What are the rules of War?

`war.py`

Using the `Card` class

- Now that we have the `Card` class, how can we use it?
- Can make a **`Deck`** class
  - What data should a `Deck` contain?
  - How can we represent that data?
- To start: write methods `__init__` and `__str__`
  - What do the method 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 xrange(2,15):
                self.cards.append(Card(rank, suit))

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

Initialize instance variable, `self.cards`

Creates and returns a string

Displays cards on separate lines

Deck API

- What methods should our `Deck` class provide?

Adding Deck Functionality

- Functionality:
  - Shuffle the cards
  - Deal one card
  - Number of cards remaining
- What do the method headers look like?
- What should they return?
- How do we implement them?

Deck API

- `Deck()` ← Constructor
- `shuffle()`
- `draw()`
- `deal(num_players, num_cards)`
- `numRemaining()`
- `isEmpty()`
- `__str__()`

Algorithm for Creating Classes

1. Identify need for a class
2. Identify state or attributes of a class/an object in that class
   - Write the constructor (`__init__`) and `__str__` methods
3. Identify methods the class should provide
   - How will a user call those methods (parameters, return values)?
     - Develop API
   - Implement methods

This Week

- Lab 9
  - Practice: Dictionary, defining classes, writing files
  - Processing data
- Broader Issue: environmental monitoring using sensor networks