

## Objectives

- Defining our own classes

## Review: Dictionaries

- What is a dictionary in Python?
- What is the syntax for creating a new dictionary?
- How do we access a key's value from a dictionary?
  - What happens if there is no mapping for that key?
- How do we create a key → value mapping in a dictionary?
- How can we iterate through a dictionary?

# ABSTRACTIONS

Nov 15, 2017

Sprenkle - CSCI111

3

## Abstractions

- Provide ways to think about program and its data
  - Get the jist without the details
- Examples we've seen
  - Functions and methods `encodeFile(filename, 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...

Nov 15, 2017

Sprenkle - CSCI111

4

## 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 (API) to modify the object's state, get information about attributes

Nov 15, 2017

Sprenkle - CSCI111

5

## 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

Nov 15, 2017

Sprenkle - CSCI111

6

## What is a Class?

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



Nov 15, 2017

Sprenkle - CSCI111

7

## 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?



Nov 15, 2017

Sprenkle - CSCI111

8

## 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

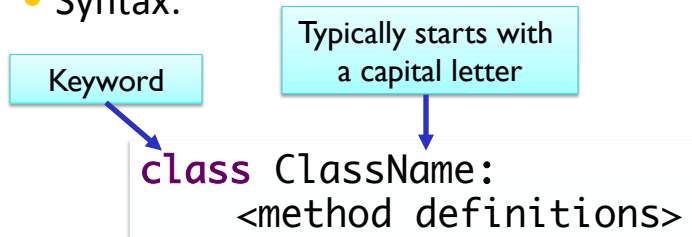
Nov 15, 2017

Sprenkle - CSCI111

9

## Defining a New Class

- Syntax:



Nov 15, 2017

Sprenkle - CSCI111

10

## 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

Nov 15, 2017

Sprenkle - CSCI111

card.py

11

## 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*

Nov 15, 2017

Sprenkle - CSCI111

card.py

12

## Defining the Constructor

- `__init__` method is like the **constructor**
- In constructor, define **instance variables**

- **Data** contained in every object
- Also called **attributes** or **fields**

Convention:  
named with `_`

- 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

Nov 15, 2017

Sprenkle - CSCI111

13

## Review

- How do we use the constructor for an object?

Nov 15, 2017

Sprenkle - CSCI111

14

## 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 for us
    - Passes the parameter *automatically*

Object **card**  
of type **Card**

```
_rank = ?  
_suit = ?
```

Nov 15, 2017

Sprenkle - CSCI111

15

## 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, 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"
```

Nov 15, 2017

Sprenkle - CSCI111

16



## Review

- How do we call a method on an object?

## 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
```

```
card = Card(..., ...)
```

- These methods 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 the object's `__str__` method is defined
  - Prints result of calling `__str__` method
- `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
```

Nov 15, 2017

Sprenkle - CSCI111

19

## Using the Card Class

Invokes the  
`__str__` method

```
def main():  
    c1 = Card(14, "spades")  
    print(c1)  
    c2 = Card(2, "hearts")  
    print(c2)
```

Displays:

Ace of spades  
2 of hearts

Object **c1** of  
type Card

`_rank = 14`  
`_suit = "spades"`

Object **c2** of  
type Card

`_rank = 2`  
`_suit = "hearts"`

Nov 15, 2017

Sprenkle - CSCI111

20

## Example: Rummy Value

- **Problem:** Add a method to the `Card` class called `getRummyValue` 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 signature/header?
  - What does the method do?
- How do we call the method?

Nov 15, 2017

Sprenkle - CSCI111

`card2.py`

21

## Card API

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

Nov 15, 2017

Sprenkle - CSCI111

22

## Card API



Implementation of methods is hidden

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

Nov 15, 2017

Sprenkle - CSCI111

23

## Defining a Card Class

(not covered in 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?
  - What are the tradeoffs to those approaches?



Nov 15, 2017

Sprenkle - CSCI111

24

## Using the Card class

- Having the `Card` class means that we can represent a `Card` in code

Now that we have the `Card` class,  
how can we **use** it?

## 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 a round of War?

## Review

```
from graphics import *  
  
win = GraphWin("Picture")  
win.setBackground("black")
```

```
from card import *  
  
c = Card(7, "diamonds")  
print(c.getRank())
```

- Same programming as before
- Just defining our own classes

Nov 15, 2017

Sprenkle - CSCI111

27

## 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?

Nov 15, 2017

Sprenkle - CSCI111

28

## Creating a Deck Class (Partial)

- List of Card objects

```
from card import *  
  
class Deck:  
    def __init__(self):           Initialize instance variable,  
        self._cardList = []     self._cardList  
        for suit in ["clubs", "hearts", "diamonds", "spades"]:  
            for rank in range(2,15):  
                myCard = Card(rank, suit)  
                self._cardList.append(myCard)  
  
    def __str__(self):          Creates and returns a string  
        deckRep= ""  
        for c in self._cardList: ← Displays cards on  
            deckRep += str(c) + "\n" ← separate lines  
        return deckRep
```

No doc strings due to space

Nov 15, 2017

Sprenkle - CSCI111

29

## Looking Ahead

- Lab 9 due Friday
- Broader Issue: Friday

Nov 15, 2017

Sprenkle - CSCI111

30