Objectives

- Defining our own classes

1

Review: Dictionaries

- What is a dictionary in Python?
  - What is it helpful for representing?
- What is the syntax for creating a new dictionary?
- How do we access a key's value from a dictionary? (2 ways)
  - What happens if there is no mapping for that key?
- How do we create a key → value mapping in a dictionary?
- How do we iterate through a dictionary?

- What does this code do?

```
if key not in dictionary :
    dictionary[key] = 1
else:
    count = dictionary[key] + 1
    dictionary[key] = count
```

- Compare lists and dictionaries
  - What are their structures and properties? How are they similar, different? When would you use one or the other?

2

1

Discussion: Comparing Lists and Dictionaries

- What are their structures? Properties?

- How are they similar?

- How are they different?

- When do you use one or the other?

3

Lists vs. Dictionaries

Lists	Dictionaries
integer positions (0, …) to any type of value	Map immutable keys (int, float, string) to any type of value
Ordered	Unordered
Slower to find a value (`in` or `find`)	Fast to find a value (use key)
Fast to print in order	Slower to print in order (by key)
Only as big as you make it	Takes up a lot of space (so can add elements in the middle)

4

Review: Equivalent Solutions
A Dictionary of Accumulators

```python
if key not in dictionary :
    dictionary[key] = 1
else:
    count = dictionary[key] + 1
    dictionary[key] = count
```

```python
if key not in dictionary :
    dictionary[key] = 1
else:
    dictionary[key] += 1
```

5

ABSTRACTIONS

6

3

Abstractions

- Provide ways to think about program and its data
  - Get the jist without the details
- Examples we've seen
  - Functions and methods
    `encryptFile(filename, key)`
    - 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 (API) to modify the object's state, get information about attributes

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

9
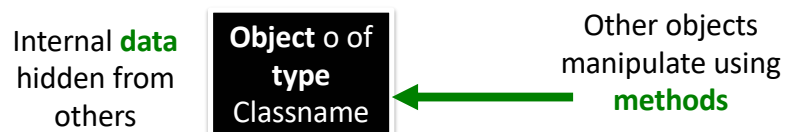
What is a Class?

- 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**

Internal **data** hidden from others

Object o of **type** Classname

Other objects manipulate using **methods**

Object o is an *instance* of Classname

10

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:
  - Suit (one of "hearts", "diamonds", "clubs", "spades")
  - Rank
    - 2-10: numbered cards
    - 11: Jack
    - 12: Queen
    - 13: King
    - 14: Ace

Pattern of Presentation Today

How We Define It

- The code we write so that someone else can use this code

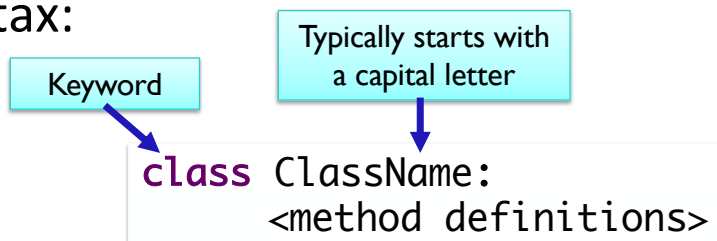**How Someone Calls/Uses It**

- How someone uses/leverages our code

13

Defining a New Class

- Sytnax:

Keyword

Typically starts with a capital letter

```
class ClassName:
        <method definitions>
```

14

Card Class (Incomplete)

```python
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
```

Method Doc String

Methods

March 20, 2023

card.py   15

15

Card Class (Incomplete)

Class Doc String

```python
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
```

Method Doc String

Methods

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

March 20, 2023

card.py   16

16

Defining the Constructor: `__init__`

- `__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`
- reference 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

Convention: named with _

17

Review

- How do we call/use the constructor for a class?

18

Using the Constructor

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

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

Object **card**
of type Card

_rank = ?
_suit = ?

19

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 *automatically* passes the `self` parameter for us
- Example:
  - `card = Card(2, "hearts")`
  - Creates a 2 of Hearts card
  - Python passes `card` as `self` for us
  - `card` is an instance of the `Card` class

Object **card**
of type Card

_rank = 2
_suit = "hearts"

20

10

Review

- How do we call a method on an object?

Accessor Methods

- To get information about the object

> - Must take **self** parameter
> - Return data/information

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

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

- Scenario: previously created object using
  `card = Card(…, …)`
  these methods would get called as `card.getRank()` and `card.getSuit()`
  - Python plugs **card** in for **self**
- Methods can access the instance variables (even though not defined in these methods) through **self**

Testing Accessor Methods

- Repeat:

  1. Create an object

  2. Call the accessor method and confirm it returns what is expected

```
c1 = Card(14, "spades")

# test the getSuit() method and constructor
test.testEqual(c1.getSuit(), "spades")

# test the getRank() method and constructor
test.testEqual(c1.getRank(), 14)
```

23

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

```python
def __str__(self):
    """Returns a string
    representing 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
```

24

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"
```

25

Testing `__str__` Method

- Repeat

1. Create an object

2. Call the method and confirm it returns what is expected

```
c1 = Card(14, "spades")

test.testEqual( str(c1), "Ace of spades")
```

Recall: str(…) automatically calls `__str__` method

26

Example: Card Color

- Problem: Add a method to the Card class called getCardColor that returns the card's suit's color ("red" or "black")
- (Partial) **procedure** for defining a method (similar to functions)
  - ➤ What is the input to the method?
  - ➤ What is the output from the method?
  - ➤ (Wait on defining the body of the method)
- How do we call the method?
- How can we test the method using test.testEqual function?
  - ➤ Provide some test cases

Example: Card Color

- Problem: Add a method to the Card class called getCardColor that returns the card's suit's color ("red" or "black")

- Procedure for defining a method (similar to functions)
  - ➤ What is the input to the method?
  - ➤ What is the output from the method?
  - ➤ What is the method signature/header?
  - ➤ What does the method do?

Example: Rummy Value

Left as a practice problem

- **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 to the method?
  - ➤ What is the output from the method?
  - ➤ What is the method signature/header?
  - ➤ What does the method do?
- How do we call the method?
- How can we test the method?

Looking Ahead

- **Prelab 9 for tomorrow**
  - ➤Engage in the object-oriented reading
- **Lab 9 due Friday**
- **Exam Friday**
  - ➤Defining classes will *not* be on exam
  - ➤Review on Wednesday