Lecture 15: Inheritance and Interfaces

7/17/2014
Guest Lecturer: Marvin Zhang

Some (a lot of) material from these slides was borrowed from John DeNero.
Announcements
Announcements

- Project 3, Ants, is out! Due Sunday 7/27
Announcements

- Project 3, Ants, is out! Due Sunday 7/27
- Homework 7 released! Due Saturday 7/19
Announcements

- Project 3, Ants, is out! Due Sunday 7/27
- Homework 7 released! Due Saturday 7/19
- Homework party tonight, 7/17, 6-10pm
Announcements

• Project 3, Ants, is out! Due Sunday 7/27
• Homework 7 released! Due Saturday 7/19
• Homework party tonight, 7/17, 6-10pm
• 61A Hackathon tomorrow, 7/18, 5pm-12am
Announcements

- Project 3, Ants, is out! Due Sunday 7/27
- Homework 7 released! Due Saturday 7/19
- Homework party tonight, 7/17, 6-10pm
- 61A Hackathon tomorrow, 7/18, 5pm-12am
- Mid-semester survey due tonight, 11:59pm
Inheritance
Inheritance

• Powerful idea in Object-Oriented Programming
Inheritance

- Powerful idea in Object-Oriented Programming
- Way of relating similar classes together
Inheritance

• Powerful idea in Object-Oriented Programming

• Way of relating similar classes together

• Common use: a specialized class inherits from a more general class
Inheritance

• Powerful idea in Object-Oriented Programming

• Way of *relating* similar classes together

• Common use: a *specialized* class inherits from a more *general* class

```python
class <new class> (<base class>):
    ...
```
Inheritance

• Powerful idea in Object-Oriented Programming

• Way of relating similar classes together

• Common use: a specialized class inherits from a more general class

    class <new class>(<base class>):
        ...

• The new class shares attributes with the base class, and overrides certain attributes
Inheritance

• Powerful idea in Object-Oriented Programming

• Way of relating similar classes together

• Common use: a specialized class inherits from a more general class

    class <new class>(<base class>):
        ...

• The new class shares attributes with the base class, and overrides certain attributes

• Implementing the new class is now as simple as specifying how it’s different from the base class
Inheritance Example
Inheritance Example

class Account:
    """A bank account."""
    ...

Inheritance Example

class Account:
    """A bank account."""
    ...

    • Bank accounts have:
Inheritance Example

class Account:
    """A bank account."""
    ...

    • Bank accounts have:
        • an account holder
Inheritance Example

class Account:
    """A bank account."""
    ...

    • Bank accounts have:

        • an account holder

        • a balance
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%
Inheritance Example

class Account:
    """A bank account."""
    ...

    • Bank accounts have:
      • an account holder
      • a balance
      • an interest rate of 2%

    • You can:
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
Inheritance Example

class Account:
    """A bank account."""
    ...

    - Bank accounts have:
        - an account holder
        - a balance
        - an interest rate of 2%

    - You can:
        - deposit to an account
        - withdraw from an account
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
  • withdraw from an account

• A CheckingAccount is a specialized type of Account.
Inheritance Example

```python
class Account:
    """A bank account."""
    ...

- Bank accounts have:
  - an account holder
  - a balance
  - an interest rate of 2%

- You can:
  - deposit to an account
  - withdraw from an account

- A CheckingAccount is a specialized type of Account.

- Checking accounts have:
```
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
  • withdraw from an account

• A CheckingAccount is a specialized type of Account.

• Checking accounts have:
  • an account holder
Inheritance Example

class Account:
    """A bank account."""
    ...

- Bank accounts have:
  - an account holder
  - a balance
  - an interest rate of 2%

- You can:
  - deposit to an account
  - withdraw from an account

- A CheckingAccount is a specialized type of Account.

- Checking accounts have:
  - an account holder
  - a balance
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
  • withdraw from an account

• A CheckingAccount is a specialized type of Account.

• Checking accounts have:
  • an account holder
  • a balance
  • an interest rate of 1%
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
  • withdraw from an account

• A CheckingAccount is a specialized type of Account.

• Checking accounts have:
  • an account holder
  • a balance
  • an interest rate of 1%
  • a withdraw fee of $1
Inheritance Example

class Account:
   """A bank account."""

   ...  

   • Bank accounts have:  
     • an account holder  
     • a balance  
     • an interest rate of 2%  

   • You can:  
     • deposit to an account  
     • withdraw from an account

   • A CheckingAccount is a specialized type of Account.  

   • Checking accounts have:  
     • an account holder  
     • a balance  
     • an interest rate of 1%  
     • a withdraw fee of $1  

   • You can:
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
  • withdraw from an account

• A CheckingAccount is a specialized type of Account.

• Checking accounts have:
  • an account holder
  • a balance
  • an interest rate of 1%
  • a withdraw fee of $1

• You can:
  • deposit to a checking account
**Inheritance Example**

```python
class Account:
    '''A bank account.'''
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
  • withdraw from an account

• A CheckingAccount is a specialized type of Account.

• Checking accounts have:
  • an account holder
  • a balance
  • an interest rate of 1%
  • a withdraw fee of $1

• You can:
  • deposit to a checking account
  • withdraw from a checking account (but there’s a fee!)```
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
    • an account holder
    • a balance
    • an interest rate of 2%

• You can:
    • deposit to an account
    • withdraw from an account

• A CheckingAccount is a specialized type of Account.

• Checking accounts have:
    • an account holder
    • a balance
    • an interest rate of 1%
    • a withdraw fee of $1

• You can:
    • deposit to a checking account
    • withdraw from a checking account (but there’s a fee!)
Inheritance Example

```python
class Account:
    """A bank account."""
...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
  • withdraw from an account

• A CheckingAccount is a specialized type of Account.

• Checking accounts have:
  • an account holder
  • a balance
  • an interest rate of 1%
  • a withdraw fee of $1

• You can:
  • deposit to a checking account
  • withdraw from a checking account (but there’s a fee!)
```
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
  • withdraw from an account

• A CheckingAccount is a specialized type of Account.

• Checking accounts have:
  • an account holder
  • a balance
  • an interest rate of 1%
  • a withdraw fee of $1

• You can:
  • deposit to a checking account
  • withdraw from a checking account (but there’s a fee!)
class Account:
    """A bank account."""
    ...

- Bank accounts have:
  - an account holder
  - a balance
  - an interest rate of 2%

- You can:
  - deposit to an account
  - withdraw from an account

- A CheckingAccount is a specialized type of Account.

- Checking accounts have:
  - an account holder
  - a balance
  - an interest rate of 1%
  - a withdraw fee of $1

- You can:
  - deposit to a checking account
  - withdraw from a checking account (but there's a fee!)
Attribute Look Up
Attribute Look Up

To look up a name in a class:
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')
```
To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
```

Attribute Look Up
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
>>> tom.interest
```

```
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
>>> tom.interest  # Found in CheckingAccount
0.01
```
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
>>> tom.interest  # Found in CheckingAccount
0.01
>>> tom.deposit(20)
```
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
>>> tom.interest  # Found in CheckingAccount
0.01
>>> tom.deposit(20)  # Found in Account
20
```
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
>>> tom.interest  # Found in CheckingAccount
0.01
>>> tom.deposit(20)  # Found in Account
20
>>> tom.withdraw(5)
```
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes *are not copied* into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
>>> tom.interest               # Found in CheckingAccount
0.01
>>> tom.deposit(20)            # Found in Account
20
>>> tom.withdraw(5)            # Found in CheckingAccount
14
```
Designing for Inheritance
Designing for Inheritance

• Don’t repeat yourself! Use existing implementations
Designing for Inheritance

• Don’t repeat yourself! Use *existing implementations*

• Reuse overridden attributes by accessing them through the *base class*
Designing for Inheritance

- Don’t repeat yourself! Use existing implementations
- Reuse overridden attributes by accessing them through the *base class*
- Use attribute look up through *instances* if possible
Designing for Inheritance

• Don’t repeat yourself! Use existing implementations

• Reuse overridden attributes by accessing them through the base class

• Use attribute look up through instances if possible

class CheckingAccount(Account):
    withdraw_fee = 1
    interest = 0.01

def withdraw(self, amount):
    return Account.withdraw(self, \
        amount + self.withdraw_fee)
Designing for Inheritance

- Don’t repeat yourself! Use existing implementations
- Reuse overridden attributes by accessing them through the base class
- Use attribute look up through instances if possible

```python
class CheckingAccount(Account):
    withdraw_fee = 1
    interest = 0.01
    def withdraw(self, amount):
        return Account.withdraw(self, \
                                amount + self.withdraw_fee)
```
Designing for Inheritance

- Don’t repeat yourself! Use existing implementations
- Reuse overridden attributes by accessing them through the base class
- Use attribute look up through instances if possible

```python
class CheckingAccount(Account):
    withdraw_fee = 1
    interest = 0.01
    def withdraw(self, amount):
        return Account.withdraw(self, \
            amount + self.withdraw_fee)
```
Designing for Inheritance

- Don’t repeat yourself! Use *existing implementations*
- Reuse overridden attributes by accessing them through the *base class*
- Use attribute look up through *instances* if possible

```python
class CheckingAccount(Account):
    withdraw_fee = 1
    interest = 0.01

def withdraw(self, amount):
    return Account.withdraw(self, \
                            amount + self.withdraw_fee)
```
Designing for Inheritance

- Don’t repeat yourself! Use existing implementations
- Reuse overridden attributes by accessing them through the base class
- Use attribute look up through instances if possible

```python
class CheckingAccount(Account):
    withdraw_fee = 1
    interest = 0.01

def withdraw(self, amount):
    return Account.withdraw(self, amount + self.withdraw_fee)
```
Designing for Inheritance

- Don’t repeat yourself! Use existing implementations
- Reuse overridden attributes by accessing them through the base class
- Use attribute look up through instances if possible

```python
class CheckingAccount(Account):
    withdraw_fee = 1
    interest = 0.01
    def withdraw(self, amount):
        return Account.withdraw(self, amount + self.withdraw_fee)
```
Inheritance vs Composition
Inheritance vs Composition

- Inheritance: relating two classes through specifying similarities and differences
Inheritance vs Composition

• Inheritance: relating two classes through specifying similarities and differences
  
  • Represents “is a” relationships, e.g. a checking account is a specific type of account
Inheritance vs Composition

- Inheritance: relating two classes through specifying similarities and differences
  - Represents “is a” relationships, e.g. a checking account is a specific type of account

- Composition: connecting two classes through their relationship to one another
Inheritance vs Composition

• Inheritance: relating two classes through specifying similarities and differences

  • Represents “is a” relationships, e.g. a checking account is a specific type of account

• Composition: connecting two classes through their relationship to one another

  • Represents “has a” relationships, e.g. a bank has a collection of bank accounts
Inheritance vs Composition (demo)

- Inheritance: relating two classes through specifying similarities and differences
  - Represents “is a” relationships, e.g. a checking account is a specific type of account

- Composition: connecting two classes through their relationship to one another
  - Represents “has a” relationships, e.g. a bank has a collection of bank accounts
Multiple Inheritance
Multiple Inheritance

- In Python, a class can inherit from multiple base classes
Multiple Inheritance

• In Python, a class can inherit from multiple base classes

• This exists in many *but not all* object-oriented languages
Multiple Inheritance

- In Python, a class can inherit from multiple base classes
- This exists in many but not all object-oriented languages
- This is a tricky and often dangerous subject, so proceed carefully!
Multiple Inheritance

- In Python, a class can inherit from multiple base classes

- This exists in many *but not all* object-oriented languages

- This is a tricky and often dangerous subject, so proceed carefully!

```python
class SavingsAccount(Account):
    deposit_fee = 2
    def deposit(self, amount):
        return Account.deposit(self, \
                                amount - self.deposit_fee)
```
Multiple Inheritance Example
Multiple Inheritance Example

• Bank executive wants the following:
Multiple Inheritance Example

- Bank executive wants the following:
  - Low interest rate of 1%
Multiple Inheritance Example

- Bank executive wants the following:
  - Low interest rate of 1%
  - $1 withdrawal fee
Multiple Inheritance Example

- Bank executive wants the following:
  - Low interest rate of 1%
  - $1 withdrawal fee
  - $2 deposit fee
Multiple Inheritance Example

• Bank executive wants the following:
  • Low interest rate of 1%
  • $1 withdrawal fee
  • $2 deposit fee
  • A free dollar for opening the account!
Multiple Inheritance Example

- Bank executive wants the following:
  - Low interest rate of 1%
  - $1 withdrawal fee
  - $2 deposit fee
  - A free dollar for opening the account!

```python
class BestAccount(CheckingAccount, SavingsAccount):
    def __init__(self, account_holder):
        self.holder = account_holder
        self.balance = 1  # best deal ever
```
Multiple Inheritance Example

- Account
- CheckingAccount
- SavingsAccount
- BestAccount
Multiple Inheritance Example

```python
>>> such_a_deal = BestAccount('Marvin')

>>> such_a_deal.balance  # instance attribute
1

>>> such_a_deal.deposit(20)  # SavingsAccount
19

>>> such_a_deal.withdraw(5)  # CheckingAccount
13
```
Multiple Inheritance Example

```python
>>> such_a_deal = BestAccount('Marvin')
>>> such_a_deal.balance  # instance attribute
1
```
Multiple Inheritance Example

```python
>>> such_a_deal = BestAccount('Marvin')
>>> such_a_deal.balance  # instance attribute
1
>>> such_a_deal.deposit(20)  # SavingsAccount
19
```
Multiple Inheritance Example

```python
>>> such_a_deal = BestAccount('Marvin')
>>> such_a_deal.balance  # instance attribute
1
>>> such_a_deal.deposit(20)  # SavingsAccount
19
>>> such_a_deal.withdraw(5)  # CheckingAccount
13
```
Complicated Inheritance
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

![Diagram of genetic inheritance](diagram.png)
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

```
some guy  Gramma  Gramps  Grandpop  Grandmom
          /       |       /        /
         /        |       /         \
        /         |       /          \
       /          |       /           
      /           |       /             
     /            |       /               
    /             |       /                 
   /              |       /                   
  /               |       /                     
 /                 |     You                     
```

---

Mom  Dad
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

![Diagram of inheritance relationships]
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

![Genealogy Chart]

- Some guy
- Gramma
- Gramps
- Grandpop
- Grandmom
- Half Aunt
- Mom
- Dad
- Half Cousin
- You
- Some other guy
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

`some guy`  `Gramma`  `Gramps`  `Grandpop`  `Grandmom`  `some other guy`  `Double Half Aunt`  `Mom`  `Dad`  `Double Half Uncle`  `Double Half Cousin`  `You`
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

Moral of the story: inheritance (especially multiple inheritance) is complicated and weird. Use it carefully!
Break
Interfaces
Interfaces
Interfaces

• Boundary that allows communication between different components by specifying the rules for communication
Interfaces

- Boundary that allows communication between different components by specifying the rules for communication
- E.g. hardware-software interfaces, user interfaces, API’s, etc.
Interfaces

• Boundary that allows communication between different components by specifying the rules for communication

• E.g. hardware-software interfaces, user interfaces, API’s, etc.

• In OOP, interfaces are defined by what the object has to implement (attributes, methods, etc.)
Two (Three) Examples
Two (Three) Examples

- Magic methods and Python protocols
Two (Three) Examples

- Magic methods and Python protocols
  - the string representation protocol
Two (Three) Examples

• Magic methods and Python protocols
  • the string representation protocol
  • the sequence protocol
Two (Three) Examples

- Magic methods and Python protocols
  - the string representation protocol
  - the sequence protocol
- API’s and the YouTube API
Python Magic Methods
Python Magic Methods

• Special methods surrounded by double underscores (e.g., \_\_init\_\_) that add “magic” to your classes
Python Magic Methods

- Special methods surrounded by double underscores (e.g., `__init__`) that add "magic" to your classes
- Used to implement several interfaces (called protocols) in Python
Python Magic Methods

• Special methods surrounded by double underscores (e.g., `__init__`) that add “magic” to your classes

• Used to implement several interfaces (called protocols) in Python
  
  • `__str__` and `__repr__`: the string representation protocol
Python Magic Methods

• Special methods surrounded by double underscores (e.g., `__init__`) that add “magic” to your classes

• Used to implement several interfaces (called protocols) in Python
  
  • `__str__` and `__repr__`: the string representation protocol
  
  • `__len__` and `__getitem__`: the sequence protocol
Python Magic Methods

• Special methods surrounded by double underscores (e.g., `__init__`) that add “magic” to your classes

• Used to implement several interfaces (called protocols) in Python

  • `__str__` and `__repr__`: the string representation protocol
  
  • `__len__` and `__getitem__`: the sequence protocol
  
  • `__iter__` and `__next__`: the iterator protocol
Python Magic Methods

- Special methods surrounded by double underscores (e.g., `__init__`) that add “magic” to your classes
- Used to implement several interfaces (called protocols) in Python
  - `__str__` and `__repr__`: the string representation protocol
  - `__len__` and `__getitem__`: the sequence protocol
  - `__iter__` and `__next__`: the iterator protocol
- We’ll look at the first two - the last will be talked about in depth next lecture!
Protocols

- Protocols are what Python (and many other languages) call *interfaces for object-oriented programming*
Protocols

• Protocols are what Python (and many other languages) call *interfaces for object-oriented programming*

• Sometimes, they’re just called interfaces (e.g., Java)
Protocols

- Protocols are what Python (and many other languages) call *interfaces for object-oriented programming*

- Sometimes, they’re just called interfaces (e.g., Java)

- To implement a protocol, objects typically need to have a certain set of attributes. In Python, these attributes are usually a collection of *magic methods*
String Representation
String Representation

• Python has two functions to produce string representations of objects: `str` and `repr`
String Representation

- Python has two functions to produce string representations of objects: `str` and `repr`.
- The “repr” string is legible to the Python interpreter, while the “str” string is legible to humans.
String Representation

• Python has two functions to produce string representations of objects: `str` and `repr`

• The “repr” string is legible to the Python interpreter, while the “str” string is legible to humans

• The “repr” string is what Python displays in an interactive session, and the “str” string is what Python prints using the `print` function
String Representation

• Python has two functions to produce string representations of objects: \texttt{str} and \texttt{repr}

• The “\texttt{repr}” string is legible to the \texttt{Python} \texttt{interpreter}, while the “\texttt{str}” string is legible to \texttt{humans}

• The “\texttt{repr}” string is what Python displays \textit{in an interactive session}, and the “\texttt{str}” string is what Python prints \texttt{using the print function}
Implementing `str` and `repr`
Implementing `str` and `repr`

- Implementing the “`repr`” string for an object requires defining the `__repr__` magic method for the corresponding class.
Implementing str and repr

- Implementing the “repr” string for an object requires defining the `__repr__` magic method for the corresponding class.

- Implementing the “str” string for an object requires defining the `__str__` magic method for the corresponding class.
Implementing str and repr

- Implementing the “repr” string for an object requires defining the `__repr__` magic method for the corresponding class.

- Implementing the “str” string for an object requires defining the `__str__` magic method for the corresponding class.

- It’s a bit more subtle than this, but we won’t go into details.
Implementing str and repr  (demo)

• Implementing the “repr” string for an object requires defining the __repr__ magic method for the corresponding class

• Implementing the “str” string for an object requires defining the __str__ magic method for the corresponding class

• It’s a bit more subtle than this, but we won’t go into details
Sequences
Sequences

- Python has many built-in sequence types: lists, tuples, ranges, strings, etc.
Sequences

- Python has many built-in sequence types: lists, tuples, ranges, strings, etc.
- Python also has a protocol for defining custom sequence classes
Sequences

- Python has many built-in sequence types: lists, tuples, ranges, strings, etc.

- Python also has a protocol for defining custom sequence classes

- Defining custom sequences is as easy as implementing the \_\_len\_\_ and \_\_getitem\_\_ magic methods
Sequences

- Python has many built-in sequence types: lists, tuples, ranges, strings, etc.

- Python also has a protocol for defining custom sequence classes

- Defining custom sequences is as easy as implementing the `__len__` and `__getitem__` magic methods

  - `__len__` is called by the `len` function, and `__getitem__` is used in sequence indexing
Sequences

- Python has many built-in sequence types: lists, tuples, ranges, strings, etc.

- Python also has a protocol for defining custom sequence classes

- Defining custom sequences is as easy as implementing the `__len__` and `__getitem__` magic methods

  - `__len__` is called by the `len` function, and `__getitem__` is used in sequence indexing
Note about Magic Methods
Note about Magic Methods

- Magic methods, when used properly, allow for very versatile, powerful, and integrated classes and objects
Note about Magic Methods

- Magic methods, when used properly, allow for very versatile, powerful, and integrated classes and objects.

- We only scratched the surface of the magic methods that exist in Python. For a more in-depth discussion, check out the following link:
Note about Magic Methods

- Magic methods, when used properly, allow for very versatile, powerful, and integrated classes and objects.

- We only scratched the surface of the magic methods that exist in Python. For a more in-depth discussion, check out the following link:

  http://www.rafekettler.com/magicmethods.html
API’s
API’s

- Application Programming Interfaces (API’s) are interfaces that define how different software components (i.e., applications) should interact
API’s

• Application Programming Interfaces (API’s) are interfaces that define how different software components (i.e., applications) should interact

• API’s take the form of libraries containing functions and classes, or remote function calls, i.e. queries for some specific data
API’s

• Application Programming Interfaces (API’s) are interfaces that define how different software components (i.e., applications) should interact

• API’s take the form of libraries containing functions and classes, or remote function calls, i.e. queries for some specific data

• API’s are incredibly important in the real world - almost every application depends on some other application
YouTube API
YouTube API

- The API for YouTube allows programs to retrieve and play videos, fetch search results, collect related videos, etc.
YouTube API

- The API for YouTube allows programs to retrieve and play videos, fetch search results, collect related videos, etc.

- The YouTube API is an *interface* for working with the YouTube application
YouTube API

• The API for YouTube allows programs to retrieve and play videos, fetch search results, collect related videos, etc.

• The YouTube API is an interface for working with the YouTube application

• We’ll look at an example of a program built using this API: ytadventure.com
How the YouTube API Works
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)
How the YouTube API Works

The YouTube API is accessed through a set of *remote function calls* (URL’s that return some specific data)
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data).

My App
(YT Adventure)
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

I’d like the video with the cat please

My App (YT Adventure)
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

My App (YT Adventure)

I’d like the video with the cat please

YouTube API
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

I’d like the video with the cat please

My App
(YT Adventure)
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data).

My App (YT Adventure)

I’d like the video with the cat please

gdata.youtube.com/feeds/api/videos/fA860GBFCg8
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

I’d like the video with the cat please

My App
(YT Adventure)
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

I’d like the video with the cat please

Sure thing

My App (YT Adventure)

YouTube API
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

My App (YT Adventure) ──────────────────── YouTube API ──────────────────── YouTube

I’d like the video with the cat please

Sure thing
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

My App (YT Adventure)

YouTube API

Sure thing
How the YouTube API Works

The YouTube API is accessed through a set of *remote function calls* (URL’s that return some specific data)

My App
(YT Adventure)

YouTube API

I’d like the video with the cat please

Sure thing
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

I’d like the video with the cat please

Sure thing

I’d like more related videos about cats please

My App
(YT Adventure)
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

My App
(YT Adventure)

I’d like the video with the cat please

Sure thing

I’d like more related videos about cats please
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

I’d like the video with the cat please

Sure thing

I’d like more related videos about cats please
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

My App
(YT Adventure)

YouTube API

I’d like the video with the cat please

Sure thing

I’d like more related videos about cats please
How the YouTube API Works

The YouTube API is accessed through a set of *remote function calls* (URL’s that return some specific data)

I’d like the video with the cat please

Sure thing

I’d like more related videos about cats please

Ugh fine
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

I’d like the video with the cat please

Sure thing

I’d like more related videos about cats please

Ugh fine
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

I’d like the video with the cat please

Sure thing

I’d like more related videos about cats please

Ugh fine
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

I’d like the video with the cat please

Sure thing

My App
(YT Adventure)

I’d like more related videos about cats please

Ugh fine
How the YouTube API Works

The YouTube API is accessed through a set of *remote function calls* (URL’s that return some specific data)

My App
(YT Adventure)

YouTube API

Sure thing

I’d like the video with the cat please

I’d like more related videos about cats please

Ugh fine
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

I’d like the video with the cat please

Sure thing

I’d like more related videos about cats please

Ugh fine
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data).

I’d like the video with the cat please

Sure thing

I’d like more related videos about cats please

Ugh fine
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

I’d like the video with the cat please

Sure thing

I’d like more related videos about cats please

Ugh fine
How the YouTube API Works

The YouTube API is accessed through a set of *remote function calls* (URL’s that return some specific data).

I’d like the video with the cat please

Sure thing

My App (YT Adventure)

I’d like more related videos about cats please

Ugh fine
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data).

I’d like the video with the cat please

Sure thing

I’d like more related videos about cats please

Ugh fine
How the YouTube API Works

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

My App
(YT Adventure)
How the YouTube API Works

The YouTube API is accessed through a set of *remote function calls* (URL’s that return some specific data)

My App
(YT Adventure)

---

YouTube API

I’d like the video with the cat please

Sure thing

I’d like more related videos about cats please

Ugh fine

This is, of course, drastically simplified - check out the actual API for more details and actual code!

https://developers.google.com/youtube/v3/getting-started
How the YouTube API Works (demo)

The YouTube API is accessed through a set of remote function calls (URL’s that return some specific data)

I’d like the video with the cat please
Sure thing

My App (YT Adventure)

I’d like more related videos about cats please

YouTube API

This is, of course, drastically simplified – check out the actual API for more details and actual code!

https://developers.google.com/youtube/v3/getting-started
Interface Wrap-up
Interface Wrap-up

• Interfaces are a broad concept, and it can be hard to wrap your head around what it really means
Interface Wrap-up

• Interfaces are a broad concept, and it can be hard to wrap your head around what it really means.

• The thing to remember is that interfaces are always about defining the rules for communication.
Interface Wrap-up

- Interfaces are a broad concept, and it can be hard to wrap your head around what it really means.

- The thing to remember is that interfaces are always about defining the rules for communication.

- Python protocols are interfaces for Python objects, as they allow communication with custom classes and objects through specific magic methods.
• Interfaces are a broad concept, and it can be hard to wrap your head around what it really means

• The thing to remember is that interfaces are always about defining the rules for communication

• Python protocols are interfaces for Python objects, as they allow communication with custom classes and objects through specific magic methods

• API’s are interfaces for applications, as they allow communication with the application through a library and/or remote function calls
Summary
Summary

• Inheritance allows for *abstraction* and *implementing relationships* in object-oriented programming
Summary

- Inheritance allows for *abstraction* and *implementing relationships* in object-oriented programming.

- Interfaces allow for *systematic and meaningful communication* by defining how to communicate, not only in OOP but many other areas of computer science.
Summary

• Inheritance allows for *abstraction* and *implementing relationships* in object-oriented programming

• Interfaces allow for *systematic and meaningful communication* by defining how to communicate, not only in OOP but many other areas of computer science

• Learning these ideas well is one of the keys to becoming a great programmer