Announcements

- Homework 5 is due Wednesday 3/4 @ 11:59pm
Announcements

- Homework 5 is due Wednesday 3/4 @ 11:59pm
  - Homework/Project party Tuesday 3/3 5pm–6:30pm in 2050 VLSB
Announcements

• Homework 5 is due Wednesday 3/4 @ 11:59pm
  - Homework/Project party Tuesday 3/3 5pm–6:30pm in 2050 VLSB
• Quiz 2 is due Thursday 3/5 @ 11:59pm
Announcements

• Homework 5 is due Wednesday 3/4 @ 11:59pm
  ▪ Homework/Project party Tuesday 3/3 5pm–6:30pm in 2050 VLSB
• Quiz 2 is due Thursday 3/5 @ 11:59pm
• Project 3 is due Thursday 3/12 @ 11:59pm
Announcements

• Homework 5 is due Wednesday 3/4 @ 11:59pm
  • Homework/Project party Tuesday 3/3 5pm–6:30pm in 2050 VLSB
• Quiz 2 is due Thursday 3/5 @ 11:59pm
• Project 3 is due Thursday 3/12 @ 11:59pm
• Midterm 2 is on Thursday 3/19 7pm–9pm
Announcements

• Homework 5 is due Wednesday 3/4 @ 11:59pm
  ▪ Homework/Project party Tuesday 3/3 5pm–6:30pm in 2050 VLSB
• Quiz 2 is due Thursday 3/5 @ 11:59pm
• Project 3 is due Thursday 3/12 @ 11:59pm
• Midterm 2 is on Thursday 3/19 7pm–9pm
• Hog strategy contest winners will be announced on Wednesday 3/4 in lecture
String Representations
String Representations
String Representations

An object value should behave like the kind of data it is meant to represent.
String Representations

An object value should behave like the kind of data it is meant to represent.

For instance, by producing a string representation of itself.
String Representations

An object value should behave like the kind of data it is meant to represent

For instance, by producing a string representation of itself

Strings are important: they represent language and programs
String Representations

An object value should behave like the kind of data it is meant to represent.

For instance, by producing a string representation of itself.

Strings are important: they represent language and programs.

In Python, all objects produce two string representations:
String Representations

An object value should behave like the kind of data it is meant to represent.

For instance, by producing a string representation of itself.

Strings are important: they represent language and programs.

In Python, all objects produce two string representations:

* The `str` is legible to humans.
String Representations

An object value should behave like the kind of data it is meant to represent.

For instance, by producing a string representation of itself.

Strings are important: they represent language and programs.

In Python, all objects produce two string representations:

- The `str` is legible to humans.
- The `repr` is legible to the Python interpreter.
String Representations

An object value should behave like the kind of data it is meant to represent

For instance, by producing a string representation of itself

Strings are important: they represent language and programs

In Python, all objects produce two string representations:
• The `str` is legible to humans
• The `repr` is legible to the Python interpreter

The `str` and `repr` strings are often the same, but not always
The repr String for an Object
The repr String for an Object

The `repr` function returns a Python expression (a string) that evaluates to an equal object.
The **repr** String for an Object

The **repr** function returns a Python expression (a string) that evaluates to an equal object:

```
repr(object) -> string
```

Return the canonical string representation of the object. For most object types, `eval(repr(object)) == object`. 
The repr String for an Object

The `repr` function returns a Python expression (a string) that evaluates to an equal object:

```python
repr(object) -> string
```

Return the canonical string representation of the object. For most object types, `eval(repr(object)) == object`.

The result of calling `repr` on a value is what Python prints in an interactive session.
The repr String for an Object

The `repr` function returns a Python expression (a string) that evaluates to an equal object.

```
repr(object) -> string
```

Return the canonical string representation of the object. For most object types, `eval(repr(object)) == object`.

The result of calling `repr` on a value is what Python prints in an interactive session.

```python
>>> 12e12
```

The repr String for an Object

The `repr` function returns a Python expression (a string) that evaluates to an equal object:

```
repr(object) -> string
```

Return the canonical string representation of the object. For most object types, `eval(repr(object)) == object`.

The result of calling `repr` on a value is what Python prints in an interactive session:

```
>>> 12e12
12000000000000.0
```
The repr String for an Object

The `repr` function returns a Python expression (a string) that evaluates to an equal object

```
repr(object) -> string
```

Return the canonical string representation of the object.
For most object types, `eval(repr(object)) == object`.

The result of calling `repr` on a value is what Python prints in an interactive session

```python
>>> 12e12
12000000000000.0
>>> print(repr(12e12))
```
The repr String for an Object

The `repr` function returns a Python expression (a string) that evaluates to an equal object.

```
repr(object) -> string
```

Return the canonical string representation of the object. For most object types, `eval(repr(object)) == object`.

The result of calling `repr` on a value is what Python prints in an interactive session:

```
>>> 12e12
12000000000000.0
>>> print(repr(12e12))
12000000000000.0
```
The `repr` String for an Object

The `repr` function returns a Python expression (a string) that evaluates to an equal object.

```python
repr(object) -> string
```

Return the canonical string representation of the object. For most object types, `eval(repr(object)) == object`.

The result of calling `repr` on a value is what Python prints in an interactive session:

```python
>>> 12e12
12000000000000.0
>>> print(repr(12e12))
12000000000000.0
```

Some objects do not have a simple Python-readable string.
The `repr` String for an Object

The `repr` function returns a Python expression (a string) that evaluates to an equal object.

```
repr(object) -> string
```

Return the canonical string representation of the object. For most object types, `eval(repr(object)) == object`.

The result of calling `repr` on a value is what Python prints in an interactive session:

```
>>> 12e12
12000000000000.0
>>> print(repr(12e12))
12000000000000.0
```

Some objects do not have a simple Python-readable string:

```
>>> repr(min)
'<built-in function min>'
```
The str String for an Object

Human interpretable strings are useful as well:
The str String for an Object

Human interpretable strings are useful as well:

```python
>>> import datetime
```
The **str** String for an Object

Human interpretable strings are useful as well:

```python
>>> import datetime
>>> today = datetime.date(2014, 10, 13)
```
The str String for an Object

Human interpretable strings are useful as well:

```python
>>> import datetime
>>> today = datetime.date(2014, 10, 13)
>>> repr(today)
```

```python
6
```
The str String for an Object

Human interpretable strings are useful as well:

```python
>>> import datetime
>>> today = datetime.date(2014, 10, 13)
>>> repr(today)
'datetime.date(2014, 10, 13)'
```
The str String for an Object

Human interpretable strings are useful as well:

```python
>>> import datetime
>>> today = datetime.date(2014, 10, 13)
>>> repr(today)
'datetime.date(2014, 10, 13)'
>>> str(today)
'2014-10-13'
```
The str String for an Object

Human interpretable strings are useful as well:

```python
>>> import datetime
>>> today = datetime.date(2014, 10, 13)
>>> repr(today)
'datetime.date(2014, 10, 13)'
>>> str(today)
'2014-10-13'
```
The str String for an Object

Human interpretable strings are useful as well:

```python
>>> import datetime
today = datetime.date(2014, 10, 13)
>>> repr(today)
'datetime.date(2014, 10, 13)'
>>> str(today)
'2014-10-13'
```

The result of calling `str` on the value of an expression is what Python prints using the `print` function:
The str String for an Object

Human interpretable strings are useful as well:

```python
>>> import datetime
today = datetime.date(2014, 10, 13)
>>> repr(today)
'datetime.date(2014, 10, 13)'
>>> str(today)
'2014-10-13'
```

The result of calling `str` on the value of an expression is what Python prints using the `print` function:

```python
>>> print(today)
2014-10-13
```
Human interpretable strings are useful as well:

```python
>>> import datetime
>>> today = datetime.date(2014, 10, 13)
>>> repr(today)
'datetime.date(2014, 10, 13)'
>>> str(today)
'2014-10-13'
```

The result of calling `str` on the value of an expression is what Python prints using the `print` function:

```python
>>> print(today)
2014-10-13
```
The str String for an Object

Human interpretable strings are useful as well:

```python
>>> import datetime
today = datetime.date(2014, 10, 13)
>>> repr(today)
'datetime.date(2014, 10, 13)'
>>> str(today)
'2014-10-13'
```

The result of calling `str` on the value of an expression is what Python prints using the `print` function:

```python
>>> print(today)
2014-10-13
```
Polymorphic Functions
Polymorphic Functions
Polymorphic Functions

Polymorphic function: A function that applies to many (poly) different forms (morph) of data
Polymorphic Functions

Polymorphic function: A function that applies to many (poly) different forms (morph) of data.

`str` and `repr` are both polymorphic; they apply to any object.
**Polymorphic Functions**

Polymorphic function: A function that applies to many (poly) different forms (morph) of data

`str` and `repr` are both polymorphic; they apply to any object.

`repr` invokes a zero-argument method `__repr__` on its argument.
Polymorphic Functions

Polymorphic function: A function that applies to many (poly) different forms (morph) of data

`str` and `repr` are both polymorphic; they apply to any object

`repr` invokes a zero-argument method `__repr__` on its argument

```python
>>> today.__repr__()
'datetime.date(2014, 10, 13)'
```
Polymorphic Functions

Polymorphic function: A function that applies to many (poly) different forms (morph) of data

`str` and `repr` are both polymorphic; they apply to any object

`repr` invokes a zero-argument method `__repr__` on its argument

```python
>>> today.__repr__()
'datetime.date(2014, 10, 13)'
```

`str` invokes a zero-argument method `__str__` on its argument
Polymorphic Functions

Polymorphic function: A function that applies to many (poly) different forms (morph) of data.

`str` and `repr` are both polymorphic; they apply to any object.

`repr` invokes a zero-argument method `__repr__` on its argument:

```python
>>> today.__repr__()
'datetime.date(2014, 10, 13)'
```

`str` invokes a zero-argument method `__str__` on its argument:

```python
>>> today.__str__()
'2014-10-13'
```
Implementing `repr` and `str`
Implementing repr and str

The behavior of `repr` is slightly more complicated than invoking `__repr__` on its argument:
Implementing repr and str

The behavior of repr is slightly more complicated than invoking __repr__ on its argument:

- An instance attribute called __repr__ is ignored! Only class attributes are found
Implementing repr and str

The behavior of `repr` is slightly more complicated than invoking `__repr__` on its argument:

- An instance attribute called `__repr__` is ignored! Only class attributes are found
- *Question*: How would we implement this behavior?
Implementing repr and str

The behavior of `repr` is slightly more complicated than invoking `__repr__` on its argument:

- An instance attribute called `__repr__` is ignored! Only class attributes are found
- *Question*: How would we implement this behavior?
Implementing `repr` and `str`

The behavior of `repr` is slightly more complicated than invoking `__repr__` on its argument:
- An instance attribute called `__repr__` is ignored! Only class attributes are found
- **Question**: How would we implement this behavior?

The behavior of `str` is also complicated:
Implementing **repr** and **str**

The behavior of **repr** is slightly more complicated than invoking **__repr__** on its argument:

- An instance attribute called **__repr__** is ignored! Only class attributes are found
- **Question**: How would we implement this behavior?

The behavior of **str** is also complicated:

- An instance attribute called **__str__** is ignored
Implementing `repr` and `str`

The behavior of `repr` is slightly more complicated than invoking `__repr__` on its argument:
- An instance attribute called `__repr__` is ignored! Only class attributes are found
- *Question:* How would we implement this behavior?

The behavior of `str` is also complicated:
- An instance attribute called `__str__` is ignored
- If no `__str__` attribute is found, uses `repr` string
Implementing `repr` and `str`

The behavior of `repr` is slightly more complicated than invoking `__repr__` on its argument:

- An instance attribute called `__repr__` is ignored! Only class attributes are found
- *Question*: How would we implement this behavior?

The behavior of `str` is also complicated:

- An instance attribute called `__str__` is ignored
- If no `__str__` attribute is found, uses `repr` string
- *Question*: How would we implement this behavior?
Implementing repr and str

The behavior of `repr` is slightly more complicated than invoking `__repr__` on its argument:
- An instance attribute called `__repr__` is ignored! Only class attributes are found
- *Question*: How would we implement this behavior?

The behavior of `str` is also complicated:
- An instance attribute called `__str__` is ignored
- If no `__str__` attribute is found, uses `repr` string
- *Question*: How would we implement this behavior?
- `str` is a class, not a function
Implementing `repr` and `str`

The behavior of `repr` is slightly more complicated than invoking `__repr__` on its argument:

• An instance attribute called `__repr__` is ignored! Only class attributes are found
• *Question*: How would we implement this behavior?

The behavior of `str` is also complicated:

• An instance attribute called `__str__` is ignored
• If no `__str__` attribute is found, uses `repr` string
• *Question*: How would we implement this behavior?
• `str` is a class, not a function

(Demo)
Interfaces


Interfaces

**Message passing:** Objects interact by looking up attributes on each other (passing messages)
**Interfaces**

**Message passing:** Objects interact by looking up attributes on each other (passing messages)

The attribute look-up rules allow different data types to respond to the same message
**Interfaces**

*Message passing:* Objects interact by looking up attributes on each other (passing messages).

The attribute look-up rules allow different data types to respond to the same message.

A **shared message** (attribute name) that elicits similar behavior from different object classes is a powerful method of abstraction.
Interfaces

Message passing: Objects interact by looking up attributes on each other (passing messages)

The attribute look-up rules allow different data types to respond to the same message.

A shared message (attribute name) that elicits similar behavior from different object classes is a powerful method of abstraction.

An interface is a set of shared messages, along with a specification of what they mean.
Interfaces

**Message passing:** Objects interact by looking up attributes on each other (passing messages)

The attribute look-up rules allow different data types to respond to the same message.

A *shared message* (attribute name) that elicits similar behavior from different object classes is a powerful method of abstraction.

An interface is a set of shared messages, along with a specification of what they mean.

**Example:**
Interfaces

**Message passing:** Objects interact by looking up attributes on each other (passing messages)

The attribute look-up rules allow different data types to respond to the same message.

A *shared message* (attribute name) that elicits similar behavior from different object classes is a powerful method of abstraction.

An interface is a set of shared messages, along with a specification of what they mean.

**Example:**

Classes that implement `__repr__` and `__str__` methods that return Python- and human-readable strings implement an interface for producing string representations.
Property Methods
Property Methods

Often, we want the value of instance attributes to stay in sync.
Property Methods

Often, we want the value of instance attributes to stay in sync

```python
>>> f = Rational(3, 5)
```

```
3
---
5
```
Property Methods

Often, we want the value of instance attributes to stay in sync

```python
>>> f = Rational(3, 5)
>>> f.float_value
0.6
```

```
3
---
5
```
Property Methods

Often, we want the value of instance attributes to stay in sync

```python
>>> f = Rational(3, 5)
>>> f.float_value
0.6
>>> f.numer = 4
```
Property Methods

Often, we want the value of instance attributes to stay in sync:

```python
>>> f = Rational(3, 5)
>>> f.float_value
0.6
>>> f.numer = 4
>>> f.float_value
0.8
```

```python
4
```

```python
5
```
Property Methods

Often, we want the value of instance attributes to stay in sync.

```python
>>> f = Rational(3, 5)
>>> f.float_value
0.6
>>> f.numer = 4
>>> f.float_value
0.8
```

No method calls!
Property Methods

Often, we want the value of instance attributes to stay in sync

```python
>>> f = Rational(3, 5)
>>> f.float_value
0.6
>>> f.numer = 4
>>> f.float_value
0.8
>>> f.denom -= 3
```

No method calls!
Property Methods

Often, we want the value of instance attributes to stay in sync

```python
>>> f = Rational(3, 5)
>>> f.float_value
0.6
>>> f.numer = 4
>>> f.float_value
0.8
>>> f.denom -= 3
>>> f.float_value
2
```

No method calls!
Property Methods

Often, we want the value of instance attributes to stay in sync

```python
>>> f = Rational(3, 5)
>>> f.float_value
0.6
>>> f.numer = 4
>>> f.float_value
0.8
>>> f.denom -= 3
>>> f.float_value
2.0
```

The @property decorator on a method designates that it will be called whenever it is looked up on an instance
Property Methods

Often, we want the value of instance attributes to stay in sync

```python
>>> f = Rational(3, 5)
>>> f.float_value
0.6
>>> f.numer = 4
>>> f.float_value
4
>>> f.denom -= 3
>>> f.float_value
2
```

The @property decorator on a method designates that it will be called whenever it is looked up on an instance

It allows zero-argument methods to be called without an explicit call expression
Property Methods

Often, we want the value of instance attributes to stay in sync

```python
>>> f = Rational(3, 5)
>>> f.float_value
0.6
>>> f.numer = 4
>>> f.float_value
0.8
>>> f.denom -= 3
>>> f.float_value
2.0
```

The `@property` decorator on a method designates that it will be called whenever it is looked up on an instance.

It allows zero-argument methods to be called without an explicit call expression.

(Demo)
Example: Complex Numbers
Multiple Representations of Abstract Data
Multiple Representations of Abstract Data

Rectangular and polar representations for complex numbers
Multiple Representations of Abstract Data

Rectangular and polar representations for complex numbers
Multiple Representations of Abstract Data

Rectangular and polar representations for complex numbers

(1, 1)
Multiple Representations of Abstract Data

Rectangular and polar representations for complex numbers

(1, 1)
Multiple Representations of Abstract Data

Rectangular and polar representations for complex numbers

(1, 1)

(\sqrt{2}, \frac{\pi}{4})
Multiple Representations of Abstract Data

Rectangular and polar representations for complex numbers

Most programs don't care about the representation
Multiple Representations of Abstract Data

Rectangular and polar representations for complex numbers

Most programs don't care about the representation

Some arithmetic operations are easier using one representation than the other
Implementing Complex Arithmetic
Implementing Complex Arithmetic

Assume that there are two different classes that both represent Complex numbers
Implementing Complex Arithmetic

Assume that there are two different classes that both represent Complex numbers

\[ 1 + \sqrt{-1} \]
### Implementing Complex Arithmetic

Assume that there are two different classes that both represent Complex numbers

<table>
<thead>
<tr>
<th>Number</th>
<th>Rectangular representation</th>
<th>Polar representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 + \sqrt{-1}$</td>
<td>ComplexRI($1, 1$)</td>
<td>ComplexMA($\sqrt{2}$, $\pi/4$)</td>
</tr>
</tbody>
</table>
Implementing Complex Arithmetic

Assume that there are two different classes that both represent Complex numbers:

<table>
<thead>
<tr>
<th>Number</th>
<th>Rectangular representation</th>
<th>Polar representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 + \sqrt{-1}$</td>
<td>ComplexRI(1, 1)</td>
<td>ComplexMA(sqrt(2), pi/4)</td>
</tr>
</tbody>
</table>

Perform arithmetic using the most convenient representation.
Implementing Complex Arithmetic

Assume that there are two different classes that both represent Complex numbers

<table>
<thead>
<tr>
<th>Number</th>
<th>Rectangular representation</th>
<th>Polar representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 + \sqrt{-1}$</td>
<td>ComplexRI(1, 1)</td>
<td>ComplexMA($\sqrt{2}$, $\pi/4$)</td>
</tr>
</tbody>
</table>

Perform arithmetic using the most convenient representation

```python
class Complex:
```
Implementing Complex Arithmetic

Assume that there are two different classes that both represent Complex numbers

<table>
<thead>
<tr>
<th>Number</th>
<th>Rectangular representation</th>
<th>Polar representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 + \sqrt{-1}$</td>
<td>ComplexRI(1, 1)</td>
<td>ComplexMA(sqrt(2), pi/4)</td>
</tr>
</tbody>
</table>

Perform arithmetic using the most convenient representation

```python
class Complex:
    def add(self, other):
        return ComplexRI(self.real + other.real,
                          self.imag + other.imag)
```
Implementing Complex Arithmetic

Assume that there are two different classes that both represent Complex numbers

<table>
<thead>
<tr>
<th>Number</th>
<th>Rectangular representation</th>
<th>Polar representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 + \sqrt{-1}$</td>
<td><strong>ComplexRI</strong>($1, 1$)</td>
<td><strong>ComplexMA</strong>($\sqrt{2}$, $\pi/4$)</td>
</tr>
</tbody>
</table>

Perform arithmetic using the most convenient representation

```python
class Complex:
    def add(self, other):
        return ComplexRI(self.real + other.real, self.imag + other.imag)
    def mul(self, other):
        return ComplexMA(self.magnitude * other.magnitude, self.angle + other.angle)
```
Complex Arithmetic Abstraction Barriers
**Complex Arithmetic Abstraction Barriers**

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
</table>

### Complex Arithmetic Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use complex numbers to perform computation
## Complex Arithmetic Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use complex numbers to perform computation</td>
<td>whole data values</td>
<td></td>
</tr>
</tbody>
</table>
Complex Arithmetic Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use complex numbers</td>
<td>whole data values</td>
<td>x.add(y), x.mul(y)</td>
</tr>
</tbody>
</table>
## Complex Arithmetic Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use complex numbers to perform computation</td>
<td>whole data values</td>
<td>x.add(y), x.mul(y)</td>
</tr>
</tbody>
</table>
## Complex Arithmetic Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use complex numbers to perform computation</td>
<td>whole data values</td>
<td>x.add(y), x.mul(y)</td>
</tr>
</tbody>
</table>

Add complex numbers
## Complex Arithmetic Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use complex numbers to perform computation</td>
<td>whole data values</td>
<td>x.add(y), x.mul(y)</td>
</tr>
<tr>
<td>Add complex numbers</td>
<td>real and imaginary parts</td>
<td></td>
</tr>
</tbody>
</table>
### Complex Arithmetic Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use complex numbers to perform computation</td>
<td>whole data values</td>
<td>x.add(y), x.mul(y)</td>
</tr>
<tr>
<td>Add complex numbers</td>
<td>real and imaginary parts</td>
<td>real, imag, ComplexRI</td>
</tr>
</tbody>
</table>
## Complex Arithmetic Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use complex numbers to perform computation</td>
<td>whole data values</td>
<td><code>x.add(y), x.mul(y)</code></td>
</tr>
<tr>
<td>Add complex numbers</td>
<td>real and imaginary parts</td>
<td><code>real, imag, ComplexRI</code></td>
</tr>
<tr>
<td>Multiply complex numbers</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Complex Arithmetic Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use complex numbers to perform computation</td>
<td>whole data values</td>
<td>x.add(y), x.mul(y)</td>
</tr>
<tr>
<td>Add complex numbers</td>
<td>real and imaginary parts</td>
<td>real, imag, ComplexRI</td>
</tr>
<tr>
<td>Multiply complex numbers</td>
<td>magnitudes and angles</td>
<td></td>
</tr>
</tbody>
</table>
Complex Arithmetic Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use complex numbers to perform computation</td>
<td>whole data values</td>
<td>x.add(y), x.mul(y)</td>
</tr>
<tr>
<td>Add complex numbers</td>
<td>real and imaginary parts</td>
<td>real, imag, ComplexRI</td>
</tr>
<tr>
<td>Multiply complex numbers</td>
<td>magnitudes and angles</td>
<td>magnitude, angle, ComplexMA</td>
</tr>
</tbody>
</table>
## Complex Arithmetic Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use complex numbers to perform computation</td>
<td>whole data values</td>
<td>x.add(y), x.mul(y)</td>
</tr>
<tr>
<td>Add complex numbers</td>
<td>real and imaginary parts</td>
<td>real, imag, ComplexRI</td>
</tr>
<tr>
<td>Multiply complex numbers</td>
<td>magnitudes and angles</td>
<td>magnitude, angle, ComplexMA</td>
</tr>
</tbody>
</table>
### Complex Arithmetic Abstraction Barriers

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Treat complex numbers as...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use complex numbers to perform computation</td>
<td>whole data values</td>
<td>x.add(y), x.mul(y)</td>
</tr>
<tr>
<td>Add complex numbers</td>
<td>real and imaginary parts</td>
<td>real, imag, ComplexRI</td>
</tr>
<tr>
<td>Multiply complex numbers</td>
<td>magnitudes and angles</td>
<td>magnitude, angle, ComplexMA</td>
</tr>
</tbody>
</table>

*Implementation of the Python object system*
Implementing Complex Numbers
An Interface for Complex Numbers
An Interface for Complex Numbers

All complex numbers should have \texttt{real} and \texttt{imag} components.
An Interface for Complex Numbers

All complex numbers should have real and imag components

All complex numbers should have a magnitude and angle
An Interface for Complex Numbers

All complex numbers should have real and imag components.

All complex numbers should have a magnitude and angle.

All complex numbers should share an implementation of add and mul.
An Interface for Complex Numbers

All complex numbers should have real and imag components

All complex numbers should have a magnitude and angle

All complex numbers should share an implementation of add and mul
An Interface for Complex Numbers

All complex numbers should have real and imag components

All complex numbers should have a magnitude and angle

All complex numbers should share an implementation of add and mul

(Demo)
The Rectangular Representation

The @property decorator allows zero-argument methods to be called without the standard call expression syntax, so that they appear to be simple attributes
The Rectangular Representation

```python
class ComplexRI:

The @property decorator allows zero-argument methods to be called without the standard call expression syntax, so that they appear to be simple attributes.
The Rectangular Representation

class ComplexRI:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

The @property decorator allows zero-argument methods to be called without the standard call expression syntax, so that they appear to be simple attributes.
The Rectangular Representation

class ComplexRI:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

@property
def magnitude(self):
    return (self.real ** 2 + self.imag ** 2) ** 0.5

The @property decorator allows zero-argument methods to be called without the standard call expression syntax, so that they appear to be simple attributes
The Rectangular Representation

class ComplexRI:

def __init__(self, real, imag):
    self.real = real
    self.imag = imag

@property
def magnitude(self):
    return (self.real ** 2 + self.imag ** 2) ** 0.5

The @property decorator allows zero-argument methods to be called without the standard call expression syntax, so that they appear to be simple attributes.
The Rectangular Representation

class ComplexRI:

    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    @property
    def magnitude(self):
        return (self.real ** 2 + self.imag ** 2) ** 0.5

    @property
    def angle(self):
        return atan2(self.imag, self.real)

The @property decorator allows zero-argument methods to be called without the standard call expression syntax, so that they appear to be simple attributes
The Rectangular Representation

class ComplexRI:

    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

@property
    def magnitude(self):
        return (self.real ** 2 + self.imag ** 2) ** 0.5

@property
    def angle(self):
        return math.atan2(self.imag, self.real)

The @property decorator allows zero-argument methods to be called without the standard call expression syntax, so that they appear to be simple attributes
class ComplexRI:

    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    @property
    def magnitude(self):
        return (self.real ** 2 + self.imag ** 2) ** 0.5

    @property
    def angle(self):
        return math.atan2(self.imag, self.real)

    def __repr__(self):
        return 'ComplexRI({0:g}, {1:g})'.format(self.real, self.imag)


The @property decorator allows zero-argument methods to be called without the standard call expression syntax, so that they appear to be simple attributes
The Polar Representation
The Polar Representation

class ComplexMA:
The Polar Representation

class ComplexMA:

    def __init__(self, magnitude, angle):
        self.magnitude = magnitude
        self.angle = angle
The Polar Representation

class ComplexMA:

    def __init__(self, magnitude, angle):
        self.magnitude = magnitude
        self.angle = angle

@property
def real(self):
    return self.magnitude * cos(self.angle)
class ComplexMA:

    def __init__(self, magnitude, angle):
        self.magnitude = magnitude
        self.angle = angle

    @property
    def real(self):
        return self.magnitude * cos(self.angle)

    @property
    def imag(self):
        return self.magnitude * sin(self.angle)
class ComplexMA:
    
def __init__(self, magnitude, angle):
        self.magnitude = magnitude
        self.angle = angle

@property
def real(self):
    return self.magnitude * cos(self.angle)

@property
def imag(self):
    return self.magnitude * sin(self.angle)

def __repr__(self):
    return 'ComplexMA({0:g}, {1:g} * pi)'.format(self.magnitude, self.angle / pi)
Using Complex Numbers

Either type of complex number can be either argument to add or mul:
Using Complex Numbers

Either type of complex number can be either argument to \texttt{add} or \texttt{mul}:

```python
class Complex:
    def add(self, other):
        return ComplexRI(self.real + other.real,
                         self.imag + other.imag)
    def mul(self, other):
        return ComplexMA(self.magnitude * other.magnitude,
                         self.angle + other.angle)
```
Using Complex Numbers

Either type of complex number can be either argument to add or mul:

```python
class Complex:
    def add(self, other):
        return ComplexRI(self.real + other.real,
                         self.imag + other.imag)
    def mul(self, other):
        return ComplexMA(self.magnitude * other.magnitude,
                         self.angle + other.angle)

>>> from math import pi
```
Using Complex Numbers

Either type of complex number can be either argument to `add` or `mul`:

```python
from math import pi

>>> ComplexRI(1, 2).add(ComplexMA(2, pi/2))
```
Using Complex Numbers

Either type of complex number can be either argument to add or mul:

```python
class Complex:
    def add(self, other):
        return ComplexRI(self.real + other.real,
                          self.imag + other.imag)
    def mul(self, other):
        return ComplexMA(self.magnitude * other.magnitude,
                         self.angle + other.angle)

>>> from math import pi
>>> ComplexRI(1, 2).add(ComplexMA(2, pi/2))
ComplexRI(1, 4)
```
Using Complex Numbers

Either type of complex number can be either argument to `add` or `mul`:

```python
class Complex:
    def add(self, other):
        return ComplexRI(self.real + other.real,
                         self.imag + other.imag)
    def mul(self, other):
        return ComplexMA(self.magnitude * other.magnitude,
                         self.angle + other.angle)
```

```python
>>> from math import pi
>>> ComplexRI(1, 2).add(ComplexMA(2, pi/2))
ComplexRI(1, 4) ................................................................. 1 + 4i
```
Using Complex Numbers

Either type of complex number can be either argument to `add` or `mul`:

```python
class Complex:
    def add(self, other):
        return ComplexRI(self.real + other.real,
                         self.imag + other.imag)
    def mul(self, other):
        return ComplexMA(self.magnitude * other.magnitude,
                         self.angle + other.angle)
```

```python
>>> from math import pi
>>> ComplexRI(1, 2).add(ComplexMA(2, pi/2))
ComplexRI(1, 4)                                          1 + 4·√−1
>>> ComplexRI(0, 1).mul(ComplexRI(0, 1))
```
Using Complex Numbers

Either type of complex number can be either argument to \texttt{add} or \texttt{mul}:

```python
class Complex:
    def add(self, other):
        return ComplexRI(self.real + other.real,
                         self.imag + other.imag)
    def mul(self, other):
        return ComplexMA(self.magnitude * other.magnitude,
                         self.angle + other.angle)
```

```python
>>> from math import pi
>>> ComplexRI(1, 2).add(ComplexMA(2, pi/2))
ComplexRI(1, 4) ................................................................. 1 + 4 \cdot \sqrt{-1}
```

```python
>>> ComplexRI(0, 1).mul(ComplexRI(0, 1))
ComplexMA(1, 1 * pi)
```
Using Complex Numbers

Either type of complex number can be either argument to \texttt{add} or \texttt{mul}:

```python
from math import pi

class Complex:
    def add(self, other):
        return ComplexRI(self.real + other.real,
                         self.imag + other.imag)
    def mul(self, other):
        return ComplexMA(self.magnitude * other.magnitude,
                         self.angle + other.angle)

>>> from math import pi
>>> ComplexRI(1, 2).add(ComplexMA(2, pi/2))
ComplexRI(1, 4) .............................................. \(1 + 4 \cdot \sqrt{-1}\)
>>> ComplexRI(0, 1).mul(ComplexRI(0, 1))
ComplexMA(1, 1 \cdot \pi) .................................. \(-1\)
```