

Outline

Encapsulation

- data hiding
- polynomials in one variable

Inheritance

- base classes and derived classes
- points and circles

Polymorphism

- builtin functions

Wrapping and Delegation

Summary + Assignments

Encapsulation

- data hiding
- polynomials in one variable

Inheritance

- base classes and derived classes
- points and circles

Polymorphism

- builtin functions

Wrapping and Delegation

Summary + Assignments

MCS 260 Lecture 26
Introduction to Computer Science
Jan Vershelde, 26 October 2007

Outline

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Encapsulation

data hiding

Information hiding is important in modular design.

In object oriented programming,
we can hide the representation of an object.

Hidden data attributes are called *private*,
opposed to *public* (the default).

In Python, starting a name with `__`
makes a data attribute private.

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived
classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Encapsulation

data hiding

Information hiding is important in modular design.

In object oriented programming,
we can hide the representation of an object.

Hidden data attributes are called *private*,
opposed to *public* (the default).

In Python, starting a name with `__`
makes a data attribute private.

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived
classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Encapsulation

data hiding

Information hiding is important in modular design.

In object oriented programming,
we can hide the representation of an object.

Hidden data attributes are called *private*,
opposed to *public* (the default).

In Python, starting a name with `__`
makes a data attribute private.

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived
classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Encapsulation

data hiding

Information hiding is important in modular design.

In object oriented programming,
we can hide the representation of an object.

Hidden data attributes are called *private*,
opposed to *public* (the default).

In Python, starting a name with `__`
makes a data attribute private.

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived
classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Outline

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Polynomials in One Variable

the need for data hiding

Problem: design a class to manipulate polynomials.

Representing $2x^8 - 3x^2 + 7$:

1. as coefficient vector $c = [7, 0, -3, 0, 0, 0, 0, 0, 2]$,
 $c[i]$ is coefficient of x^i ;
2. as list of tuples $L = [(2, 8), (-3, 2), (7, 0)]$
 $(c, i) \in L$ represents cx^i .

Both representations have advantages and disadvantages.

Solution offered by *encapsulation*:

1. make representation of the object polynomial private;
2. access to the data only via specific methods.
3. resolve the problem of normalization, i.e.: $p == 0$?

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Polynomials in One Variable

the need for data hiding

Problem: design a class to manipulate polynomials.

Representing $2x^8 - 3x^2 + 7$:

1. as coefficient vector $c = [7, 0, -3, 0, 0, 0, 0, 0, 2]$,
 $c[i]$ is coefficient of x^i ;
2. as list of tuples $L = [(2, 8), (-3, 2), (7, 0)]$
 $(c, i) \in L$ represents cx^i .

Both representations have advantages and disadvantages.

Solution offered by *encapsulation*:

1. make representation of the object polynomial private;
2. access to the data only via specific methods.
3. resolve the problem of normalization, i.e.: $p == 0$?

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Polynomials in One Variable

the need for data hiding

Problem: design a class to manipulate polynomials.

Representing $2x^8 - 3x^2 + 7$:

1. as coefficient vector $c = [7, 0, -3, 0, 0, 0, 0, 0, 2]$,
 $c[i]$ is coefficient of x^i ;
2. as list of tuples $L = [(2, 8), (-3, 2), (7, 0)]$
 $(c, i) \in L$ represents cx^i .

Both representations have advantages and disadvantages.

Solution offered by *encapsulation*:

1. make representation of the object polynomial private;
2. access to the data only via specific methods.
3. resolve the problem of normalization, i.e.: $p == 0$?

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Polynomials in One Variable

the need for data hiding

Problem: design a class to manipulate polynomials.

Representing $2x^8 - 3x^2 + 7$:

1. as coefficient vector $c = [7, 0, -3, 0, 0, 0, 0, 0, 2]$,
 $c[i]$ is coefficient of x^i ;
2. as list of tuples $L = [(2, 8), (-3, 2), (7, 0)]$
 $(c, i) \in L$ represents cx^i .

Both representations have advantages and disadvantages.

Solution offered by *encapsulation*:

1. make representation of the object polynomial private;
2. access to the data only via specific methods.
3. resolve the problem of normalization, i.e.: $p == 0?$

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes
points and circles

Polymorphism

builtin functions

Wrapping and
Delegation

Summary +
Assignments

Polynomials in One Variable

the need for data hiding

Problem: design a class to manipulate polynomials.

Representing $2x^8 - 3x^2 + 7$:

1. as coefficient vector $c = [7, 0, -3, 0, 0, 0, 0, 0, 2]$,
 $c[i]$ is coefficient of x^i ;
2. as list of tuples $L = [(2, 8), (-3, 2), (7, 0)]$
 $(c, i) \in L$ represents cx^i .

Both representations have advantages and disadvantages.

Solution offered by *encapsulation*:

1. make representation of the object polynomial private;
2. access to the data only via specific methods.
3. resolve the problem of normalization, i.e.: $p == 0?$

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and
Delegation

Summary +
Assignments

Polynomials in One Variable

the need for data hiding

Problem: design a class to manipulate polynomials.

Representing $2x^8 - 3x^2 + 7$:

1. as coefficient vector $c = [7, 0, -3, 0, 0, 0, 0, 0, 2]$,
 $c[i]$ is coefficient of x^i ;
2. as list of tuples $L = [(2, 8), (-3, 2), (7, 0)]$
 $(c, i) \in L$ represents cx^i .

Both representations have advantages and disadvantages.

Solution offered by *encapsulation*:

1. make representation of the object polynomial private;
2. access to the data only via specific methods.
3. resolve the problem of normalization, i.e.: $p == 0$?

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

The Class Poly

file classpoly

```
class Poly:
    "defines a polynomial in one variable"

    def __init__(self,c=0,p=0):
        "monomial with coefficient c and power p"
        if c==0:
            self.__L = []
        else:
            self.__L = [(c,p)]
```

Notice:

1. We hide the coefficient list via `__`.
2. We do not store monomials with zero coefficients.

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

The Class Poly

file classpoly

MCS 260 L-26

26 October 2007

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

```
class Poly:
    "defines a polynomial in one variable"

    def __init__(self,c=0,p=0):
        "monomial with coefficient c and power p"
        if c==0:
            self.__L = []
        else:
            self.__L = [(c,p)]
```

Notice:

1. We hide the coefficient list via `__`.
2. We do not store monomials with zero coefficients.

String Representations

overloading `__str__`

```
>>> from classpoly import *
>>> p = Poly(2,3)
>>> str(p)
'+2*x^3'
```

```
def __str__(self):
    "returns a polynomial as a string"
    s = ''
    for m in self.__L:
        s += '%+2.f*x^%d' % (m[0],m[1])
    if s == '':
        return '0'
    else:
        return s
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

String Representations

overloading `__str__`

```
>>> from classpoly import *
>>> p = Poly(2,3)
>>> str(p)
'+2*x^3'
```

```
def __str__(self):
    "returns a polynomial as a string"
    s = ''
    for m in self.__L:
        s += '%+2.f*x^%d' % (m[0],m[1])
    if s == '':
        return '0'
    else:
        return s
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Adding Two Polynomials

relies on adding one monomial

```
def __add__(self, other):
    "adds two polynomials"
    p = Poly()
    for m in self.__L:
        p.addmon(m[0], m[1])
    for m in other.__L:
        p.addmon(m[0], m[1])
    return p
```

```
>>> p = Poly(2,4)
>>> q = Poly(3,4)
>>> s = p+q
>>> str(s)
'+5*x^4'
```

Avoid storing zero coefficients!

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Adding Two Polynomials

relies on adding one monomial

```
def __add__(self, other):
    "adds two polynomials"
    p = Poly()
    for m in self.__L:
        p.addmon(m[0], m[1])
    for m in other.__L:
        p.addmon(m[0], m[1])
    return p
```

```
>>> p = Poly(2,4)
>>> q = Poly(3,4)
>>> s = p+q
>>> str(s)
'+5*x^4'
```

Avoid storing zero coefficients!

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Adding a Monomial

avoiding to store zero

Adding a monomial to a polynomial:

```
def addmon(self,c,p):
    "adds a monomial"
    if c != 0:
        done = False
        for i in range(0,len(self.__L)):
            m = self.__L[i]
            if m[1] == p:
                nc = c + m[0]
                del(self.__L[i])
                if nc != 0:
                    self.__L.append((nc,p))
                done = True
                break
        if not done:
            self.__L.append((c,p))
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Adding a Monomial

avoiding to store zero

Adding a monomial to a polynomial:

```
def addmon(self,c,p):
    "adds a monomial"
    if c != 0:
        done = False
        for i in range(0,len(self.__L)):
            m = self.__L[i]
            if m[1] == p:
                nc = c + m[0]
                del(self.__L[i])
                if nc != 0:
                    self.__L.append((nc,p))
                done = True
                break
        if not done:
            self.__L.append((c,p))
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Adding a Monomial

avoiding to store zero

Adding a monomial to a polynomial:

```
def addmon(self,c,p):
    "adds a monomial"
    if c != 0:
        done = False
        for i in range(0,len(self.__L)):
            m = self.__L[i]
            if m[1] == p:
                nc = c + m[0]
                del(self.__L[i])
                if nc != 0:
                    self.__L.append((nc,p))
                done = True
                break
        if not done:
            self.__L.append((c,p))
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Adding a Monomial

avoiding to store zero

Adding a monomial to a polynomial:

```
def addmon(self,c,p):
    "adds a monomial"
    if c != 0:
        done = False
        for i in range(0,len(self.__L)):
            m = self.__L[i]
            if m[1] == p:
                nc = c + m[0]
                del(self.__L[i])
                if nc != 0:
                    self.__L.append((nc,p))
                done = True
                break
    if not done:
        self.__L.append((c,p))
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Adding a Monomial

avoiding to store zero

Adding a monomial to a polynomial:

```
def addmon(self,c,p):
    "adds a monomial"
    if c != 0:
        done = False
        for i in range(0,len(self.__L)):
            m = self.__L[i]
            if m[1] == p:
                nc = c + m[0]
                del(self.__L[i])
                if nc != 0:
                    self.__L.append((nc,p))
                done = True
                break
        if not done:
            self.__L.append((c,p))
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Outline

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Inheritance

base classes and derived classes

We can create new classes from existing classes.
These new classes are *derived* from *base* classes.

The derived class *inherits* the attributes of the base class and usually contains additional attributes.

Inheritance is a powerful mechanism to reuse software.

We distinguish between single and multiple inheritance:

single a derived class inherits from only *one* class;

multiple derivation from *multiple* different classes.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Inheritance

base classes and derived classes

We can create new classes from existing classes.
These new classes are *derived* from *base* classes.

The derived class *inherits* the attributes of the base class and usually contains additional attributes.
Inheritance is a powerful mechanism to reuse software.

We distinguish between single and multiple inheritance:

- single** a derived class inherits from only *one* class;
- multiple** derivation from *multiple* different classes.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Inheritance

base classes and derived classes

We can create new classes from existing classes.
These new classes are *derived* from *base* classes.

The derived class *inherits* the attributes of the base class and usually contains additional attributes.
Inheritance is a powerful mechanism to reuse software.

We distinguish between single and multiple inheritance:

- single** a derived class inherits from only *one* class;
- multiple** derivation from *multiple* different classes.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Outline

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Points and Circles

a first example of inheritance

We represent a point in the plane by the values for its coordinates, usually called x and y .

The class `Point` has attributes x and y and a string representation.

A circle is determined by a center and radius.

The class `Circle` will inherit from the class `Point` to represent its center. The radius of the circle is an additional object data attribute. The function `area` is an additional functional attribute.

The class `Circle` will use the string representation of `Point` for its center and extend the `__str()` function for its radius.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Points and Circles

a first example of inheritance

We represent a point in the plane by the values for its coordinates, usually called x and y .

The class `Point` has attributes x and y and a string representation.

A circle is determined by a center and radius.

The class `Circle` will inherit from the class `Point` to represent its center. The radius of the circle is an additional object data attribute. The function `area` is an additional functional attribute.

The class `Circle` will use the string representation of `Point` for its center and extend the `__str()` function for its radius.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Points and Circles

a first example of inheritance

We represent a point in the plane by the values for its coordinates, usually called x and y .

The class `Point` has attributes x and y and a string representation.

A circle is determined by a center and radius.

The class `Circle` will inherit from the class `Point` to represent its center. The radius of the circle is an additional object data attribute. The function `area` is an additional functional attribute.

The class `Circle` will use the string representation of `Point` for its center and extend the `__str()` function for its radius.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Points and Circles

a first example of inheritance

We represent a point in the plane by the values for its coordinates, usually called x and y .

The class `Point` has attributes x and y and a string representation.

A circle is determined by a center and radius.

The class `Circle` will inherit from the class `Point` to represent its center. The radius of the circle is an additional object data attribute. The function `area` is an additional functional attribute.

The class `Circle` will use the string representation of `Point` for its center and extend the `__str()` function for its radius.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Points and Circles

a first example of inheritance

We represent a point in the plane by the values for its coordinates, usually called x and y .

The class `Point` has attributes x and y and a string representation.

A circle is determined by a center and radius.

The class `Circle` will inherit from the class `Point` to represent its center. The radius of the circle is an additional object data attribute. The function `area` is an additional functional attribute.

The class `Circle` will use the string representation of `Point` for its center and extend the `__str()` function for its radius.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Points and Circles

a first example of inheritance

We represent a point in the plane by the values for its coordinates, usually called x and y .

The class `Point` has attributes x and y and a string representation.

A circle is determined by a center and radius.

The class `Circle` will inherit from the class `Point` to represent its center. The radius of the circle is an additional object data attribute. The function `area` is an additional functional attribute.

The class `Circle` will use the string representation of `Point` for its center and extend the `__str()` function for its radius.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Points and Circles

a first example of inheritance

We represent a point in the plane by the values for its coordinates, usually called x and y .

The class `Point` has attributes x and y and a string representation.

A circle is determined by a center and radius.

The class `Circle` will inherit from the class `Point` to represent its center. The radius of the circle is an additional object data attribute. The function `area` is an additional functional attribute.

The class `Circle` will use the string representation of `Point` for its center and extend the `__str()` function for its radius.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

The Class Point

in the file `classpoint.py`

The definition of the class `Point` is placed in a separate file `classpoint.py`, available to import as a module.

```
class Point:
    "defines a point in the plane"

    def __init__(self,a=0,b=0):
        "constructs a point in the plane"
        self.x = a
        self.y = b

    def __str__(self):
        "returns string representation of a point"
        return '( %.4e, %.4e )' % (self.x,self.y)
```

The string representation uses scientific notation for the coordinates, with 4 digits after the decimal point.

Encapsulation

- data hiding
- polynomials in one variable

Inheritance

- base classes and derived classes
- points and circles

Polymorphism

- builtin functions

Wrapping and Delegation

Summary + Assignments

The Class Point

in the file `classpoint.py`

The definition of the class `Point` is placed in a separate file `classpoint.py`, available to import as a module.

```
class Point:
    "defines a point in the plane"

    def __init__(self,a=0,b=0):
        "constructs a point in the plane"
        self.x = a
        self.y = b

    def __str__(self):
        "returns string representation of a point"
        return '( %.4e, %.4e )' % (self.x,self.y)
```

The string representation uses scientific notation for the coordinates, with 4 digits after the decimal point.

Encapsulation

- data hiding
- polynomials in one variable

Inheritance

- base classes and derived classes
- points and circles

Polymorphism

- builtin functions

Wrapping and Delegation

Summary + Assignments

The Class Point

in the file `classpoint.py`

The definition of the class `Point` is placed in a separate file `classpoint.py`, available to import as a module.

```
class Point:
    "defines a point in the plane"

    def __init__(self, a=0, b=0):
        "constructs a point in the plane"
        self.x = a
        self.y = b

    def __str__(self):
        "returns string representation of a point"
        return '( %.4e, %.4e )' % (self.x, self.y)
```

The string representation uses scientific notation for the coordinates, with 4 digits after the decimal point.

Encapsulation

- data hiding
- polynomials in one variable

Inheritance

- base classes and derived classes
- points and circles

Polymorphism

- builtin functions

Wrapping and Delegation

Summary + Assignments

The Class Circle

in the file `classcircle.py`

```
from classpoint import *
class Circle(Point):
    "defines a circle derived from point"

    def __init__(self, a=0, b=0, r=0):
        "the center is (a,b), radius = r"
        Point.__init__(self, a, b)
        self.r = r

    def area(self):
        "returns the area of the circle"
        from math import pi
        return pi*self.r**2

    def __str__(self):
        "returns string representation of a circle"
        s = 'center : ' + Point.__str__(self) + '\n'
        s += 'radius : ' + '%.4e' % self.r
        return s
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

The Class Circle

in the file `classcircle.py`

```

from classpoint import *
class Circle(Point):
    "defines a circle derived from point"

    def __init__(self,a=0,b=0,r=0):
        "the center is (a,b), radius = r"
        Point.__init__(self,a,b)
        self.r = r

    def area(self):
        "returns the area of the circle"
        from math import pi
        return pi*self.r**2

    def __str__(self):
        "returns string representation of a circle"
        s = 'center : ' + Point.__str__(self) + '\n'
        s += 'radius : ' + '%.4e' % self.r
        return s

```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

The Class Circle

in the file `classcircle.py`

```

from classpoint import *
class Circle(Point):
    "defines a circle derived from point"

    def __init__(self,a=0,b=0,r=0):
        "the center is (a,b), radius = r"
        Point.__init__(self,a,b)
        self.r = r

    def area(self):
        "returns the area of the circle"
        from math import pi
        return pi*self.r**2

    def __str__(self):
        "returns string representation of a circle"
        s = 'center : ' + Point.__str__(self) + '\n'
        s += 'radius : ' + '%.4e' % self.r
        return s

```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

The Class Circle

in the file `classcircle.py`

```

from classpoint import *
class Circle(Point):
    "defines a circle derived from point"

    def __init__(self,a=0,b=0,r=0):
        "the center is (a,b), radius = r"
        Point.__init__(self,a,b)
        self.r = r

    def area(self):
        "returns the area of the circle"
        from math import pi
        return pi*self.r**2

    def __str__(self):
        "returns string representation of a circle"
        s = 'center : ' + Point.__str__(self) + '\n'
        s += 'radius : ' + '%.4e' % self.r
        return s

```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Using the Classes

file pointcircle.py

```
# L-26 MCS 260 Fri 26 Oct 2007 : pointcircle
#
# This Python program uses the functionality
# of the classes circle and point.
# Note that Point is imported indirectly
# via the import of the class Circle.

from classcircle import *
print 'using classes point and circle'
x = input('give x : ')
y = input('give y : ')
p = Point(x,y)
print 'the point ' + str(p)
print 'has coordinates ', p.x, p.y
r = input('give r : ')
c = Circle(x,y,r)
print 'the circle :\n' + str(c)
print 'has area ', c.area()
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Using the Classes

file pointcircle.py

```
# L-26 MCS 260 Fri 26 Oct 2007 : pointcircle
#
# This Python program uses the functionality
# of the classes circle and point.
# Note that Point is imported indirectly
# via the import of the class Circle.

from classcircle import *
print 'using classes point and circle'
x = input('give x : ')
y = input('give y : ')
p = Point(x,y)
print 'the point ' + str(p)
print 'has coordinates ', p.x, p.y
r = input('give r : ')
c = Circle(x,y,r)
print 'the circle :\n' + str(c)
print 'has area ', c.area()
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Using the Classes

file pointcircle.py

```
# L-26 MCS 260 Fri 26 Oct 2007 : pointcircle
#
# This Python program uses the functionality
# of the classes circle and point.
# Note that Point is imported indirectly
# via the import of the class Circle.

from classcircle import *
print 'using classes point and circle'
x = input('give x : ')
y = input('give y : ')
p = Point(x,y)
print 'the point ' + str(p)
print 'has coordinates ', p.x, p.y
r = input('give r : ')
c = Circle(x,y,r)
print 'the circle :\n' + str(c)
print 'has area ', c.area()
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

the program pointcircle.py

Executing at the command prompt \$:

```
$ python pointcircle.py
using classes point and circle
give x : 2.3
give y : -0.2
the point ( 2.3000e+00, -2.0000e-01 )
has coordinates 2.3 -0.2
give r : 0.762
the circle :
center : ( 2.3000e+00, -2.0000e-01 )
radius : 7.6200e-01
has area 1.82414692475
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

the program pointcircle.py

Executing at the command prompt \$:

```
$ python pointcircle.py
using classes point and circle
give x : 2.3
give y : -0.2
the point ( 2.3000e+00, -2.0000e-01 )
has coordinates 2.3 -0.2
give r : 0.762
the circle :
center : ( 2.3000e+00, -2.0000e-01 )
radius : 7.6200e-01
has area 1.82414692475
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Polymorphism

method overriding

Recall Python dynamic typing: during run time Python determines the type of a variable to know which methods may be applied.

Calling `str()` on `x` gives different strings, depending on whether `x` is an instance of the class `Point` or `Circle`.

In `x.method()`, the meaning of the `method` depends on the type (or class) of `x`.

Polymorphism allows objects of different classes related by inheritance to respond differently to the same method.

Giving new definitions for methods in the derived class that have the same name as in the base class is called *method overriding*.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Polymorphism

method overriding

Recall Python dynamic typing: during run time Python determines the type of a variable to know which methods may be applied.

Calling `str()` on `x` gives different strings, depending on whether `x` is an instance of the class `Point` or `Circle`.

In `x.method()`, the meaning of the `method` depends on the type (or class) of `x`.

Polymorphism allows objects of different classes related by inheritance to respond differently to the same method.

Giving new definitions for methods in the derived class that have the same name as in the base class is called *method overriding*.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Polymorphism

method overriding

Recall Python dynamic typing: during run time Python determines the type of a variable to know which methods may be applied.

Calling `str()` on `x` gives different strings, depending on whether `x` is an instance of the class `Point` or `Circle`.

In `x.method()`, the meaning of the `method` depends on the type (or class) of `x`.

Polymorphism allows objects of different classes related by inheritance to respond differently to the same method.

Giving new definitions for methods in the derived class that have the same name as in the base class is called *method overriding*.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Polymorphism

method overriding

Recall Python dynamic typing: during run time Python determines the type of a variable to know which methods may be applied.

Calling `str()` on `x` gives different strings, depending on whether `x` is an instance of the class `Point` or `Circle`.

In `x.method()`, the meaning of the `method` depends on the type (or class) of `x`.

Polymorphism allows objects of different classes related by inheritance to respond differently to the same method.

Giving new definitions for methods in the derived class that have the same name as in the base class is called *method overriding*.

Encapsulation

- data hiding
- polynomials in one variable

Inheritance

- base classes and derived classes
- points and circles

Polymorphism

- builtin functions

Wrapping and Delegation

Summary + Assignments

Polymorphism

method overriding

Recall Python dynamic typing: during run time Python determines the type of a variable to know which methods may be applied.

Calling `str()` on `x` gives different strings, depending on whether `x` is an instance of the class `Point` or `Circle`.

In `x.method()`, the meaning of the `method` depends on the type (or class) of `x`.

Polymorphism allows objects of different classes related by inheritance to respond differently to the same method.

Giving new definitions for methods in the derived class that have the same name as in the base class is called *method overriding*.

Encapsulation

- data hiding
- polynomials in one variable

Inheritance

- base classes and derived classes
- points and circles

Polymorphism

- builtin functions

Wrapping and Delegation

Summary + Assignments

Outline

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Encapsulation

data hiding

polynomials in one variable

Inheritance

base classes and derived classes

points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Builtin Functions

to check types and relationships

We can check whether a variable is of a certain type:

```
>>> isinstance('z',int)
False
>>> isinstance(3,int)
True
```

In the OOP parlance, we check whether an object is an instance of another object or class.

`issubclass()` verifies relationships between classes:

```
>>> from classcircle import *
>>> issubclass(Point,Circle)
False
>>> issubclass(Circle,Point)
True
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes
points and circles

Polymorphism

builtin functions

Wrapping and
Delegation

Summary +
Assignments

Builtin Functions

to check types and relationships

We can check whether a variable is of a certain type:

```
>>> isinstance('z',int)
False
>>> isinstance(3,int)
True
```

In the OOP parlance, we check whether an object is an instance of another object or class.

`issubclass()` verifies relationships between classes:

```
>>> from classcircle import *
>>> issubclass(Point,Circle)
False
>>> issubclass(Circle,Point)
True
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes
points and circles

Polymorphism

builtin functions

Wrapping and
Delegation

Summary +
Assignments

Builtin Functions on Attributes

With `hasattr` we ask whether an attribute exists:

```
>>> from classpoint import *
>>> p = Point(2,3)
>>> hasattr(p, 'x')
True
>>> hasattr(p, 'z')
False
```

With `getattr` we get the value of an attribute:

```
>>> getattr(p, 'x')
2
```

With `setattr` we set the value of an attribute"

```
>>> setattr(p, 'x', 10)
>>> str(p)
'( 1.0000e+01, 3.0000e+00 )'
```

To delete an attribute:

```
>>> delattr(p, 'x')
>>> hasattr(p, 'x')
False
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Builtin Functions on Attributes

With `hasattr` we ask whether an attribute exists:

```
>>> from classpoint import *
>>> p = Point(2,3)
>>> hasattr(p, 'x')
True
>>> hasattr(p, 'z')
False
```

With `getattr` we get the value of an attribute:

```
>>> getattr(p, 'x')
2
```

With `setattr` we set the value of an attribute"

```
>>> setattr(p, 'x', 10)
>>> str(p)
'( 1.0000e+01, 3.0000e+00 )'
```

To delete an attribute:

```
>>> delattr(p, 'x')
>>> hasattr(p, 'x')
False
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Builtin Functions on Attributes

With `hasattr` we ask whether an attribute exists:

```
>>> from classpoint import *
>>> p = Point(2,3)
>>> hasattr(p, 'x')
True
>>> hasattr(p, 'z')
False
```

With `getattr` we get the value of an attribute:

```
>>> getattr(p, 'x')
2
```

With `setattr` we set the value of an attribute"

```
>>> setattr(p, 'x', 10)
>>> str(p)
'( 1.0000e+01, 3.0000e+00 )'
```

To delete an attribute:

```
>>> delattr(p, 'x')
>>> hasattr(p, 'x')
False
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Builtin Functions on Attributes

With `hasattr` we ask whether an attribute exists:

```
>>> from classpoint import *
>>> p = Point(2,3)
>>> hasattr(p, 'x')
True
>>> hasattr(p, 'z')
False
```

With `getattr` we get the value of an attribute:

```
>>> getattr(p, 'x')
2
```

With `setattr` we set the value of an attribute"

```
>>> setattr(p, 'x', 10)
>>> str(p)
'( 1.0000e+01, 3.0000e+00 )'
```

To delete an attribute:

```
>>> delattr(p, 'x')
>>> hasattr(p, 'x')
False
```

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Wrapping and Delegation

MCS 260 L-26

26 October 2007

Often we want customized interfaces.

Wrapping consists in

1. making a derived class of an existing class;
2. add, modify, or remove functionality.

Delegation is a characteristic of wrapping.

When a method is applied to an object, Python

1. first looks for a definition in the class of the object;
2. followed by a search in the parents of the class.

By overriding the `__getattr__()` method, we control this delegation.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Wrapping and Delegation

MCS 260 L-26

26 October 2007

Often we want customized interfaces.

Wrapping consists in

1. making a derived class of an existing class;
2. add, modify, or remove functionality.

Delegation is a characteristic of wrapping.

When a method is applied to an object, Python

1. first looks for a definition in the class of the object;
2. followed by a search in the parents of the class.

By overriding the `__getattr__()` method, we control this delegation.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Often we want customized interfaces.

Wrapping consists in

1. making a derived class of an existing class;
2. add, modify, or remove functionality.

Delegation is a characteristic of wrapping.

When a method is applied to an object, Python

1. first looks for a definition in the class of the object;
2. followed by a search in the parents of the class.

By overriding the `__getattr__()` method, we control this delegation.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Wrapping and Delegation

Often we want customized interfaces.

Wrapping consists in

1. making a derived class of an existing class;
2. add, modify, or remove functionality.

Delegation is a characteristic of wrapping.

When a method is applied to an object, Python

1. first looks for a definition in the class of the object;
2. followed by a search in the parents of the class.

By overriding the `__getattr__()` method,
we control this delegation.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Wrapping and Delegation

Often we want customized interfaces.

Wrapping consists in

1. making a derived class of an existing class;
2. add, modify, or remove functionality.

Delegation is a characteristic of wrapping.

When a method is applied to an object, Python

1. first looks for a definition in the class of the object;
2. followed by a search in the parents of the class.

By overriding the `__getattr__()` method,
we control this delegation.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived
classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Wrapping and Delegation

Often we want customized interfaces.

Wrapping consists in

1. making a derived class of an existing class;
2. add, modify, or remove functionality.

Delegation is a characteristic of wrapping.

When a method is applied to an object, Python

1. first looks for a definition in the class of the object;
2. followed by a search in the parents of the class.

By overriding the `__getattr__()` method, we control this delegation.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments

Summary + Assignments

We ended chapter 8 in *Making Use of Python*;
read Chapter 25 in *The Art & Craft of Computing*.

Assignments:

1. An additional invariant to the representation of a polynomial in one variable is that its monomials are ordered along descending degree. Describe how you would implement this in the class `Poly`.
2. Derive a class `Triangle` from the class `Point`. The constructor should take three points as argument. Write an `area` function.
3. Write Python code to define a class `Queue` (FIFO). It should export `enqueue` and `dequeue`.

Homework collected on Monday 28 October:

exercise 2 of Lecture 23; exercises 1 and 3 of Lecture 24;
and exercises 3 and 4 of Lecture 25.

Encapsulation

data hiding
polynomials in one variable

Inheritance

base classes and derived classes
points and circles

Polymorphism

builtin functions

Wrapping and Delegation

Summary + Assignments