Object Oriented Programming (in Python)

Jennifer Thompson

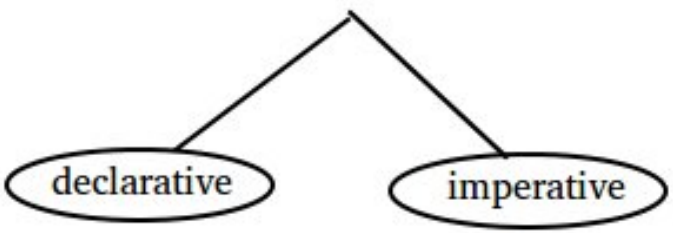
II. Physikalisches Institut, Universität Göttingen

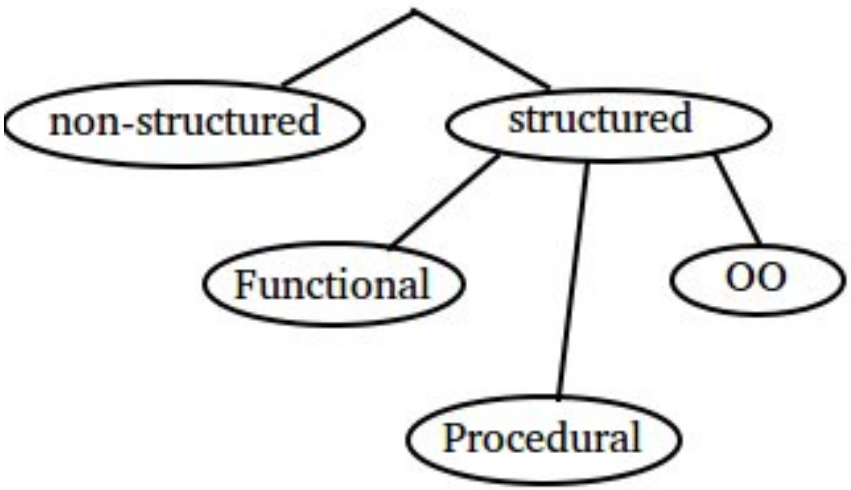
jennifer.thompson@physik.uni-goettingen.de

May 17, 2016

Overview

Styles

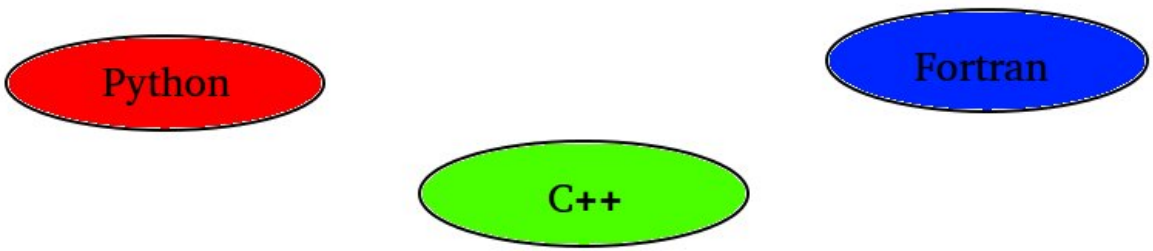

declarative
imperative


non-structured
structured
Functional
OO
Procedural

Languages

Styles

Programming styles (mostly) independent of language



Key Points

Object Oriented programming has some key features:

- Encapsulation/information hiding
- Inheritance
- Polymorphism

Encapsulation/Information hiding

- Not globally modifiable
- State kept as 'real world' variables
- State is controlled by object methods
- **Object state kept consistent**

Inheritance

- Can extract common behaviour
- Useful for an is-a relationship

Polymorphism

- Allows a single interface to related types
- Client code does not know exact type until runtime
- Objects are responsible for their own behaviour

Namespaces

- Python makes very good use of namespaces
- Enables code reuse
- Prerequisite for clean module system

In Python

`import` imports a module

`.` marks `thing` from `module` like `module.thing`

Modules

```
# helpers.py

def spam(x):
    return '{0}, {0}, {0}, {1} and {0}.'.format('spam',x)

N_A = 6.02214e+23
```

```
# work1.py

import helpers

print helpers.N_A
print helpers.spam('eggs')
```

```
# work2.py

import helpers as h

print h.N_A
print h.spam('eggs')
```

```
# work3.py

from helpers import *

print N_A
print spam('eggs')
```

```
# work4.py

from helpers import N_A as L, spam as foo

print L
print foo('eggs')
```

Modules

```
# helpers.py

def spam(x):
    return '{0}, {0}, {0}, {1} and {0}.'.format('spam',x)

N_A = 6.02214e+23
```

```
# work1.py

import helpers

print helpers.N_A
print helpers.spam('e
```

```
>>> import helpers
>>> dir(helpers)
[...,'N_A','spam']
```

```
s as h

print h.spam('eggs')
```

```
# work3.py

from helpers import *

print N_A
print spam('eggs')
```

```
# work4.py

from helpers import N_A as L, spam as foo

print L
print foo('eggs')
```

Modules

```
try:
    from fastlib import xyz as foo
except ImportError:
    from slowlib import abc as foo

foo('something',3,4)
```

different func names,
same argument order

```
try:
    from fastlib import xyz as foo
except ImportError:
    from slowlib import abc as _abc
    def foo(x,y,z): return _abc(z,x,y)

foo('something',3,4)
```

different func names,
different arg order

Packages

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

Packages

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Su
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

```
import sound.effects as se

from sound.effects import echo

from sound.effects.echo import echofilter
```

Classes

```
class TVseries(object):

    def __init__(self, name, eps):
        self.name = name
        self.eps_per_s = eps

    def status(self):
        text = '{} has {} episodes per season.'
        return text.format(self.name, self.eps_per_s)
```

```
bbt = TVseries('Big Bang Theory', 24)
gf  = TVseries('Gravity Falls',   20)

print bbt.name
print bbt.status()
print
print gf.name
print gf.status()

print dir(bbt)
```

Classes

```
class TVseries(object):

    def __init__(self, name, eps):
        self.name = name
        self.eps_per_s = eps

    def status(self):
        text = '{} has {} episodes per season.'
        return text.format(self.name, self.eps_per_s)
```

initialization (constructor)

member variables (attributes)

member function (method)

```
bbt = TVseries('Big Bang Theory', 24)
gf  = TVseries('Gravity Falls',   20)

print bbt.name
print bbt.status()
print
print gf.name
print gf.status()

print dir(bbt)
```

Classes

```
class TVseries(object):

    def __init__(self, name, eps):
        self.name = name
        self.eps_per_s = eps

    def status(self):
        text = '{} has {} episodes per season.'
        return text.format(self.name, self.eps_per_s)
```

initialization (constructor)

member variables (attributes)

member function (method)

```
bbt = TVseries('Big Bang Theory', 24)
gf  = TVseries('Gravity Falls',   20)

print bbt.name
print bbt.status()
print
print gf.name
print gf.status()

print dir(bbt)
```

parallel to module usage!

Classes

```
class TVseries(object):

    def __init__(self, name, eps):
        self.name = name
        self.eps_per_s = eps

    def status(self):
        te
        re
```

initialization (constructor)

member variables (attributes)

member function (method)

```
Big Bang Theory
Big Bang Theory has 24 episodes per season.

Gravity Falls
Gravity Falls has 20 episodes per season.
[...,'eps_per_s','name','status']
```

```
bbt = TVse
gf  = TVse

print bbt.
print bbt.status()
print
print gf.name
print gf.status()

print dir(bbt)
```

Methods

```
class TVseries(object):

    def __init__(self, name, eps):
        self.name = name
        self.eps_per_s = eps
        self.num_watched = 0

    def seen(self, num=1):
        self.num_watched += num

    def status(self):
        text = '{} has {} episodes per season. I saw {} of them.'
        return text.format(self.name, self.eps_per_s, self.num_watched)
```

```
bbt = TVseries('Big Bang Theory', 24)
gf  = TVseries('Gravity Falls',   20)

print bbt.name
bbt.seen(4)
print bbt.status()
print
print gf.name
gf.seen()
print gf.status()

print dir(bbt)
```

Methods

```
class TVseries(object):

    def __init__(self, name, eps):
        self.name = name
        self.eps_per_s = eps
        self.num_watched = 0

    def seen(self, num=1):
        self.num_watched += num

    def status(self):
```

```
Big Bang Theory
Big Bang Theory has 24 episodes per season. I saw 4 of them.

Gravity Falls
Gravity Falls has 20 episodes per season. I saw 1 of them.
[...,'eps_per_s','name','num_watched','seen','status']
```

```
print bbt.status()
print
print gf.name
gf.seen()
print gf.status()

print dir(bbt)
```

Built-in Methods

```
class TVseries(object):

    def __init__(self, name, eps):
        self.name = name
        self.eps_per_s = eps
        self.num_watched = 0

    def seen(self, num=1):
        self.num_watched += num

    def __str__(self):
        text = '{} has {} episodes per season. I saw {} of them.'
        return text.format(self.name, self.eps_per_s, self.num_watched)
```

```
bbt = TVseries('Big Bang Theory', 24)
gf  = TVseries('Gravity Falls',   20)

print bbt.name
bbt.seen(4)
print bbt
print
print gf.name
got.seen()
print gf

print dir(bbt)
```

Built-in Methods

```
class TVseries(object):

    def __init__(self, name, eps):
        self.name = name
        self.eps_per_s = eps
        self.num_watched = 0

    def seen(self, num=1):
        self.num_watched += num

    def __str__(self):
```

```
Big Bang Theory
Big Bang Theory has 24 episodes per season. I saw 4 of them.

Gravity Falls
Gravity Falls has 20 episodes per season. I saw 1 of them.
[...,'eps_per_s','name','num_watched','seen']
```

```
print bbt
print
print gf.name
got.seen()
print gf

print dir(bbt)
```

Inheritance

```
class Foo(object):
    def hello(self):
        print "Hello! Foo here."

    def bye(self):
        print "Bye bye from Foo!"

class Bar(Foo):
    def hello(self):
        print "Hello! Bar here."
```

```
>>> f = Foo()
>>> f.hello()
Hello! Foo here.
>>> f.bye()
Bye bye from Foo!
>>>
>>> b = Bar()
>>> b.hello()
Hello! Bar here.
>>> b.bye()
Bye bye from Foo!
```

Accessor methods

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
```

```
>>> p = Point(2,2)
>>> p.x, p.y
(2, 2)
>>> p.x = 5
>>> p.x, p.y
(5, 2)
```

Accessor methods

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
```

```
>>> p = Point(2,2)
>>> p.x, p.y
(2, 2)
>>> p.x = 5
>>> p.x, p.y
(5, 2)
```

How about polar coordinates, too?

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
        self.r = sqrt(x**2 + y**2)
        self.phi = atan2(y,x)
```

```
>>> p = Point(3,4)
>>> p.x, p.y
(3, 4)
>>> p.r, p.phi
(5.0, 0.9272952)
```

Accessor methods

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
```

```
>>> p = Point(2,2)
>>> p.x, p.y
(2, 2)
>>> p.x = 5
>>> p.x, p.y
(5, 2)
```

How about polar coordinates, too?

```
from math import sqrt, atan2
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
        self.r = sqrt(x**2 + y**2)
        self.phi = atan2(y,x)
```

```
>>> p = Point(3,4)
>>> p.x, p.y
(3, 4)
>>> p.r, p.phi
(5.0, 0.9272952)
```

Accessor methods

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
```

```
>>> p = Point(2,2)
>>> p.x, p.y
(2, 2)
>>> p.x = 5
>>> p.x, p.y
(5, 2)
```

How about polar coordinates, too?

```
from math import sqrt, atan2
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
        self.r = sqrt(x**2 + y**2)
        self.phi = atan2(y,x)
```

```
>>> p = Point(3,4)
>>> p.x, p.y
(3, 4)
>>> p.r, p.phi
(5.0, 0.9272952)
```

```
>>> p.r = 10 # Noooo!
```

Accessor methods

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
```

```
>>> p = Point(2,2)
>>> p.x, p.y
(2, 2)
>>> p.x = 5
>>> p.x, p.y
(5, 2)
```

try again:

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def r(self):
        return sqrt(self.x**2 + self.y**2)

    def phi(self):
        return atan2(self.y,self.x)
```

Accessor methods

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
```

```
>>> p = Point(2,2)
>>> p.x, p.y
(2, 2)
>>> p.x = 5
>>> p.x, p.y
(5, 2)
```

try again:

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def r(self):
        return sqrt(self.x**2 + self.y**2)

    def phi(self):
        return atan2(self.y,self.x)
```

```
>>> p = Point(3,4)
>>> p.x, p.y
(3, 4)
>>> p.r(), p.phi()
(5.0, 0.9272952)
```

Safe, but asymmetric:

Accessor methods

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    @property
    def r(self):
        return sqrt(self.x**2 + self.y**2)

    @property
    def phi(self):
        return atan2(self.y,self.x)
```

```
>>> p = Point(3,4)
>>> p.x, p.y
(3, 4)
>>> p.r, p.phi
(5.0, 0.9272952)
```

Accessor methods

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    @property
    def r(self):
        return sqrt(self.x**2 + self.y**2)

    @property
    def phi(self):
        return atan2(self.y,self.x)
```

```
>>> p = Point(3,4)
>>> p.x, p.y
(3, 4)
>>> p.r, p.phi
(5.0, 0.9272952)
```

still not quite symmetric...

```
>>> p.r = 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
```

Accessor methods

```
class Point(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    @property
    def r(self):
        return sqrt(self.x**2 + self.y**2)

    @r.setter
    def r(self,r_new):
        r_old = self.r
        scale = r_new / r_old
        self.x *= scale
        self.y *= scale

    @property
    def phi(self):
        return atan2(self.y,self.x)
```

```
>>> p = Point(3,4)
>>> p.x,p.y
(3, 4)
>>> p.r,p.phi
(5.0, 0.9272952)
>>> p.r = 10
>>> p.r,p.phi
(10.0, 0.9272952)
>>> p.x,p.y
(6.0, 8.0)
```

copying

```
class Test(object):
    def __init__(self):
        self.val  = 5       # immutable
        self.list = [5,6,7] # mutable
```

```
>>> a = Test()
>>> b = a

>>> a.val, b.val
(5, 5)

>>> a.val = 7
>>> a.val, b.val
(7, 7)

>>> a.list, b.list
([5, 6, 7], [5, 6, 7])

>>> a.list.append(999)
>>> a.list, b.list
([5, 6, 7, 999], [5, 6, 7, 999])

>>> a.list = 'Hello'
>>> a.list, b.list
('Hello', 'Hello')
```

copying

```
>>> from copy import copy, deepcopy

>>> a = Test()
>>> b = a
>>> c = copy(a)
>>> d = deepcopy(a)

>>> a.val, b.val, c.val, d.val
(5,              5,              5,              5)

>>> a.val = 7
>>> a.val, b.val, c.val, d.val
(7,              7,              5,              5)

>>> a.list, b.list, c.list, d.list
([5, 6, 7],      [5, 6, 7],      [5, 6, 7],      [5, 6, 7])

>>> a.list.append(999)
>>> a.list[0] = 0
>>> a.list, b.list, c.list, d.list
([0, 6, 7, 999], [0, 6, 7, 999], [0, 6, 7, 999], [5, 6, 7])

>>> a.list = 'Hello'
>>> a.list, b.list, c.list, d.list
('Hello',        'Hello',        [0, 6, 7, 999], [5, 6, 7])
```

Conclusions

- Object Oriented programming is very useful
- Type of structured programming
- Allows real world objects to be the main component
- Possible to implement in Python