Argelander-
Institut
für
Astronomie

Python for Astronomers

Classes and instances

Some modules from the
Python Standard Library

Class definition

```
>>> class Hello:
...     """A simple, but friendly class"""
...     def sayIt(self):
...       """Prints a friendly message"""
...       print "Hi guys!"

>>> # Defines an object of type classobj,
>>> # which can be explored in the usual ways:

>>> Hello.<TAB>
>>> # or
>>> help(Hello)
```

Instances

```
>>> h = Hello()    # Creates an object, which is
>>>                # an instance of class Hello

>>> h.<TAB>

>>> # Do not get confused by:
>>> print type(h)   # type is 'instance'
>>> help(h)         # Help for class 'instance'

>>> # Use the object:
>>> h.sayIt()       # Definition was 'def sayIt(self):'
>>>                 # Object h takes place
>>>                 # of argument 'self'
```

Instances

```
>>> h1 = Hello()
>>> h2 = Hello()

>>> h1.sayIt()
>>> h2.sayIt()

>>> h1 is h2        # h1 and h2 have sperate identity
```

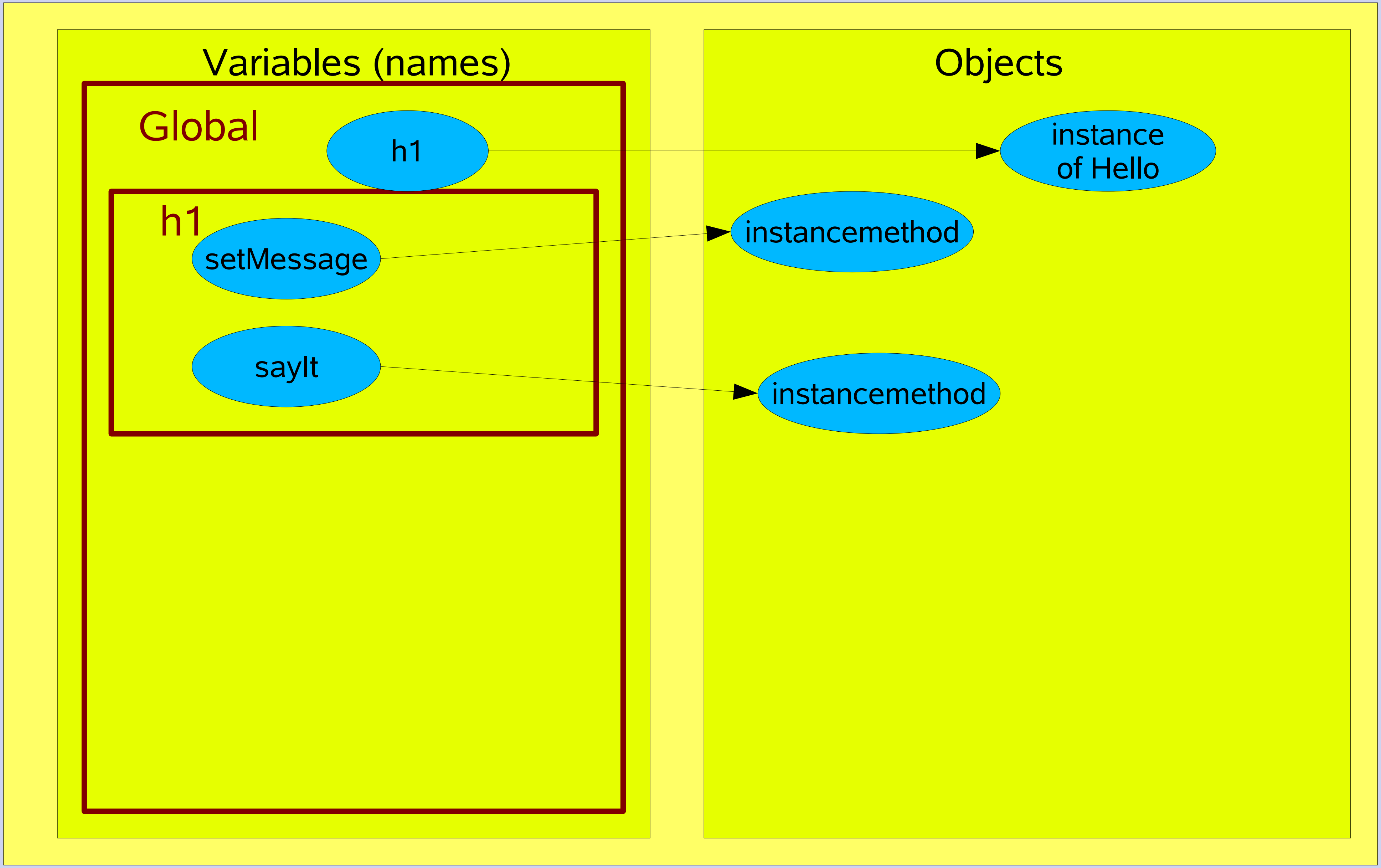
Data members

```
>>> class Hello:
...   def setMessage(self, message):
...     self.msg = message         # Create data member
...   def sayIt(self):
...     print self.msg

>>> h1 = Hello()
>>> h1.setMessage("Hi there")      # h1.msg created
>>> h1.sayIt()                     # h1.msg used

>>> print h1.msg                   # h1.msg accessible
>>> h1.msg = "Message changed"     # from the outside
>>> h1.sayIt()
```

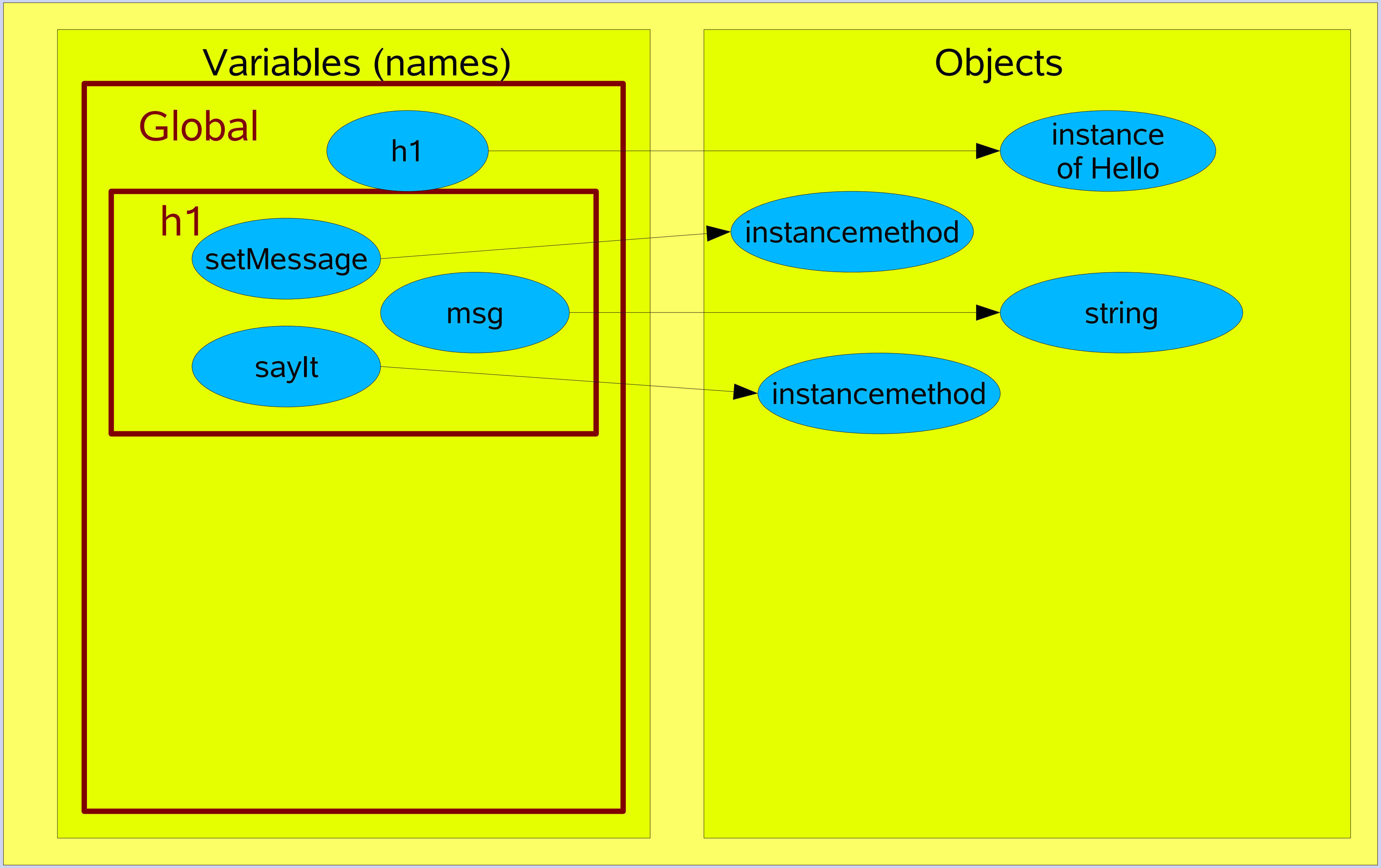
Data members

```
>>> # Data members of different instances can be set
>>> # independently:
>>> h1 = Hello()
>>> h1.setMessage("Message 1")

>>> h2 = Hello()
>>> h2.setMessage("Message 2")

>>> h1.sayIt()
>>> h2.sayIt()
```

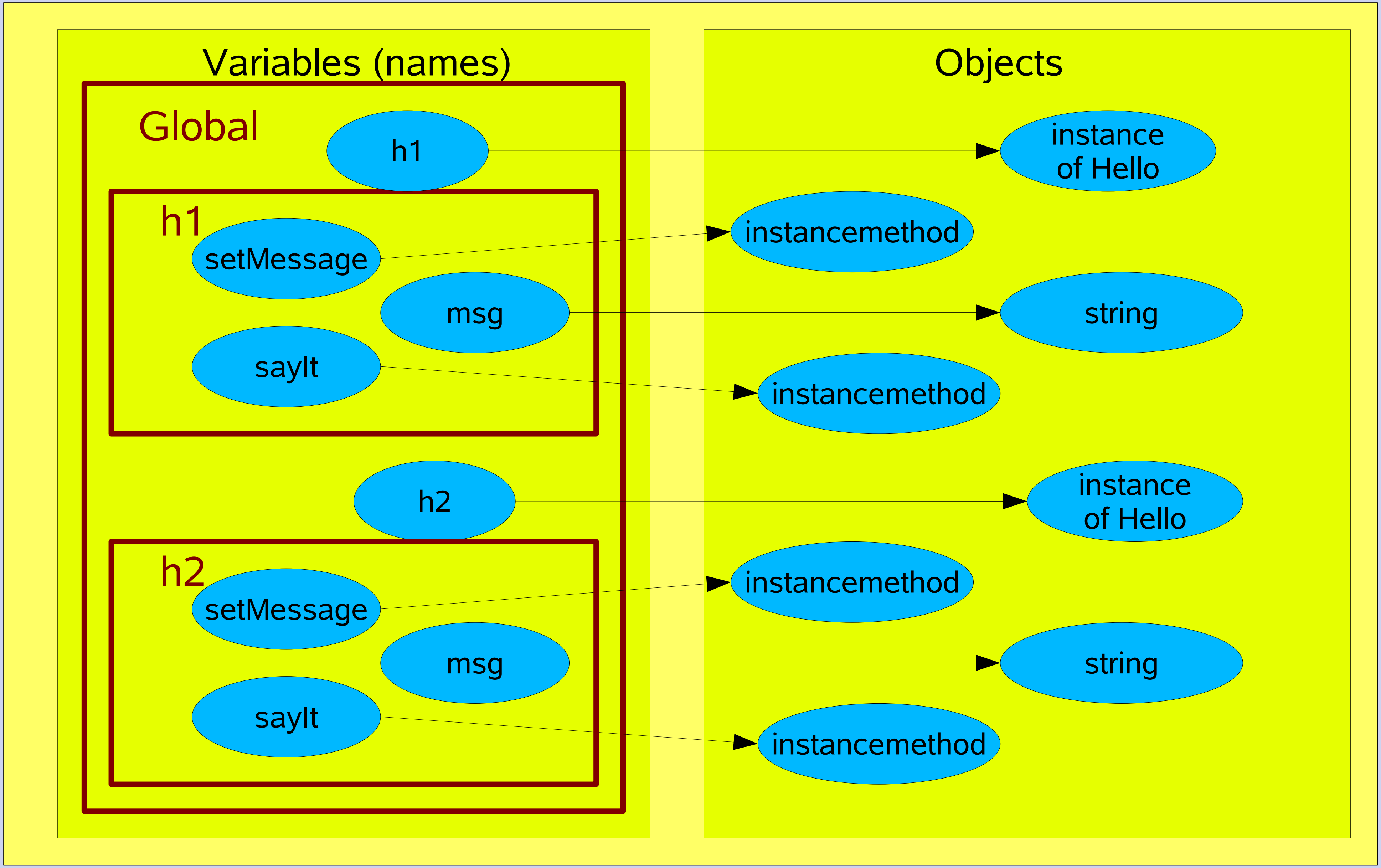
Instance namespaces



Instance namespaces



Instance namespaces



Namespace issues

```
>>> # Similar connection between namespaces as for
>>> # functions:

>>> m = "Message"        # Immutable
>>> h = Hello()
>>> h.setMessage(m)
>>> m = "Changed"
>>> h.sayIt()            # Changed?   No

>>> m = [1, 2, 3]        # Mutable
>>> h.setMessage(m)
>>> m[0] = 42
>>> h.sayIt()            # Changed?   Yes!
```

Constructors

```
>>> # Data members must be created before used:
>>> h = Hello()
>>> h.sayIt()       # AttributeError

>>> class Hello:
...   def __init__(self, message="Hello world"):
...     self.msg = message       # Create data member
...   def sayIt(self):
...     print self.msg

>>> h = Hello("Hi there") # Calls constructor __init__
>>> h.sayIt()
```

Data encapsulation – Information hiding

- Data encapsulation / Information hiding is one of the fundamental principles of Object Oriented Programming (OOP)
- The related syntactical mechanisms are classes and instances (Data encapsulation) and private data members (Information hiding)
- Python does not support private data members syntactically, but only by convention

Data encapsulation – Information hiding

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

>>> p = Point(1, 2)
>>> print "Coordinates: %f, %f" % (p.x, p.y)
```

Data encapsulation – Information hiding

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

>>> p = Point(1, 2)
>>> print "Coordinates: %f, %f" % (p.x, p.y)

>>> class Point:
...     def __init__(self, x, y):
...         self.xy = [x, y] # Implementation detail changed

>>> # Client code needs to be changed too!
>>> p = Point(1, 2)
>>> print "Coordinates: %f, %f" % (p.xy[0], p.xy[1])
```

Data encapsulation – Information hiding

```
>>> class Point:
...   def __init__(self, x, y):
...     self._coords = [x, y] # Important Convention:
...                           # Do not access from
...                           # outside the class!
...   def getCoords(self):
...     return tuple(self._coords)

>>> p = Point(1, 2)
>>> print "Coordinates: %f, %f" \
...   % (p.getCoords()[0], p.getCoords()[1])
```

Data encapsulation – Information hiding

```
>>> class Point:
...   def __init__(self, x, y):
...     self._x = x     # Changed internal representation
...     self._y = y
...   def getCoords(self):
...     # Have to adjust to new representation
...     return (self._x, self._y)

>>> # Client code need not to be changed!
>>> p = Point(1, 2)
>>> print "Coordinates: %f, %f" \
...   % (p.getCoords()[0], p.getCoords()[1])
```

A worked out example: Prisoner's dilemma

A classical problem from game theory:

> *Two suspects are arrested by the police. The police have insufficient evidence for a conviction, and, having separated both prisoners, visit each of them to offer the same deal. If one testifies ("defects") for the prosecution against the other and the other remains silent, the betrayer goes free and the silent accomplice receives the full 10-year sentence. If both remain silent, both prisoners are sentenced to only six months in jail for a minor charge. If each betrays the other, each receives a five-year sentence. Each prisoner must choose to betray the other or to remain silent. Each one is assured that the other would not know about the betrayal before the end of the investigation. How should the prisoners act?*

Write a program that runs a simulation of the Prisoner's dilemma. Prisoners with different probablities to testify are to be considered. Select random pairs of these prisoners, let them be visited by the police, and find out, which prisoner receives the shortest average sentence.

A worked out example: Prisoner's dilemma

A classical problem from game theory:

> *Two suspects are arrested by the police. The police have insufficient evidence for a conviction, and, having separated both prisoners, visit each of them to offer the same deal. If one testifies ("defects") for the prosecution against the other and the other remains silent, the betrayer goes free and the silent accomplice receives the full 10-year sentence. If both remain silent, both prisoners are sentenced to only six months in jail for a minor charge. If each betrays the other, each receives a five-year sentence. Each prisoner must choose to betray the other or to remain silent. Each one is assured that the other would not know about the betrayal before the end of the investigation. How should the prisoners act?*

Write a program that runs a simulation of the Prisoner's dilemma. Prisoners with different probablities to testify are to be considered. Select random pairs of these prisoners, let them be visited by the police, and find out, which prisoner receives the shortest average sentence.

A worked out example: Prisoner's dilemma

Example should demonstrate:

- Object oriented programs need not be large and complex
- Classes can often have a close connection to real-world entities
- This makes object oriented programs easy to understand
- Object oriented programming is fun!

Exercise should also demonstrate:

- Object oriented programs can be changed easily

Exercise: Iterated Prisoner's dilemma

In the iterated Prisoner's dilemma, the police vist the prisoners repeatedly. The prisoners know the previous testimonies of both themselves and the other prisoner, and adjust their reaction to that.

Modify the existing classes in the following form: The police visits each pair of prisoners nRepeat times. When visiting a prisoner, the police let him know what the other prisoner did when visited last time. The visited prisoner adjusts his probability to testify in the following way: When the other prisoner testified, his (the visited prisoner's) probability is increased by 5%; when the other prisoner did not testify, the probability is decreased by 5%. The new probability is remembered and increased or decreased during the next visit.

Of course, for a new pair of prisoners the initial probabilities must be at their original value.

Keep the modifications of the code at a minimum!

The Python Standard Library

- Distributed with Python (Python comes with batteries included)
- contains ~300 modules, implemented both in Python and C
- access to system services
- solutions for many everyday problems
- support for platform-independent programs
- documented at http://docs.python.org/library/

Many other free Python modules for all sorts of problems exist; at http://pypi.python.org/pypi > 5.000 free packages are listed!

math, cmath, and random

- math: elementary mathematical functions (trigonometric, hyperbolic, log, exp, etc.)
- cmath: the same for complex numbers
- random: pseudo-random number generators for various distributions

```
>>> import random
>>> mu, sigma = 5., 0.5
>>> x = mu
>>> while -3*sigma < x-mu < 3*sigma:
...     x = random.gauss(mu, sigma)
...     print x
```

time – Time access and conversion

Provides various time-related functions.

```
>>> import time
>>> t0 = time.time()     # Time since
>>>                      # 1970-01-01:00:00:00 UTC
>>> for i in range(10):
...     time.sleep(0.3)  # Sleep 0.3 s
>>> t1 = time.time()
>>> print "Sleeping + typing: %f s" % (t1 - t0)
```

time – Time access and conversion

```
>>> now = time.time()     # Time since
>>> print time.localtime(now)
>>> # (year, month, day, hour, min, sec,
>>> #  day of week, day of year, is DST)
>>> print time.gmtime(now)

>>> # No support for switch seconds:
>>> t2008 = time.mktime((2009,1,1,0,59,59,3,1,0))
>>> t2009 = time.mktime((2009,1,1,1,0,0,3,1,0))
>>> print t2009 - t2008


>>> Related module: calender, datetime
```

sys - System-specific parameters and functions

Provides access to variables used or maintained by the interpreter.

```
>>> import sys
>>> print sys.version       # Python version
>>> print sys.version_info # as tuple
>>> print sys.path          # Module search path
>>> print sys.maxint        # Largest regular int
>>> print sys.argv          # Command line args
>>>                         # See also optparse
>>> sys.exit()              # Exit from Python
```

os – Operation system interfaces

Provides a portable way of using operating system dependent functionality.

```
>>> import os
>>> os.getlogin()          # Login name
>>> os.getenv("HOME")      # Environment variable
>>> os.getloadavg()        # Load average (tuple)
>>> os.system("firefox")# Execute shell command
```

os – Operation system interfaces

Manipulation of files and directories:

```
>>> os.mkdir("osTest")
>>> os.chdir("osTest")
>>> print os.getcwd()

>>> for i in range(100):
...     f = open("file%d.dat" % i, "w")
...     f.close()

>>> os.listdir('.')
>>> for oldname in os.listdir('.'):
...     newname = oldname.split('.')[0] + '.txt'
...     os.rename(oldname, newname)
```

os – Operation system interfaces

Make sure you are still in the same directory!

```
>>> # Remove all files:
>>> import glob # Unix style pathname
>>>             # pattern expansion
>>> txtfiles = glob.glob("*.txt")
>>> for tf in txtfiles:
...    os.remove(tf)
```

os.path – Pathname maniulations

Mostly platform-independent.

```
>>> import os   # or os.path

>>> os.path.abspath('.')

>>> os.path.dirname("/home/rschaaf/mobydick.txt")

>>> os.path.basename("/home/rschaaf/mobydick.txt")


>>> os.path.exists("/home/rschaaf/mobydick.txt")

>>> os.path.isfile("/home/rschaaf/mobydick.dat")

>>> os.path.isdir("/home/rschaaf/mobydick.txt")


>>> time.localtime(os.path.getmtime("/tmp"))
```