Programming Styles and Objects

Fermilab - TARGET 2017
Week 3

Programming styles

Imperative programming

- Procedural programming
- Object oriented programming

Declarative programming

- Database languages (Structured Query Language)
- Functional programming
- Logical programming

Structured Query Language (SQL)

Create and manipulate tables of Data

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SELECT CustomerName, City FROM Customers WHERE Country='Mexico';

UPDATE Supplier SET City = 'Sydney' WHERE Name = 'Pavlova, Ltd.'

DELETE FROM Product WHERE UnitPrice > 50

Complex command example

```
INSERT INTO Customer (FirstName, LastName, City, Country, Phone)
SELECT LEFT(ContactName, CHARINDEX(' ',ContactName) - 1),
      SUBSTRING(ContactName, CHARINDEX(' ',ContactName) + 1, 100),
      City, Country, Phone
 FROM Supplier
WHERE CompanyName = 'Bigfoot Breweries'
```

Functional programming

Evaluates mathematical functions and avoids changing-state and mutable data

Functions always return the same value

Variables don't change value

Is programming without assignment statements

```
from random import random

def move_cars(car_positions):
    return map(lambda x: x + 1 if random() > 0.3 else x,
               car_positions)

def output_car(car_position):
    return '-' * car_position

def run_step_of_race(state):
    return {'time': state['time'] - 1,
            'car_positions': move_cars(state['car_positions'])}

def draw(state):
    print ''
    print '\n'.join(map(output_car, state['car_positions']))

def race(state):
    draw(state)
    if state['time']:
        race(run_step_of_race(state))

race({'time': 5,
      'car_positions': [1, 1, 1]})
```

Logic programming

Programming based on the notion of logical deduction in symbolic logic

Facts

```
child(Pebbles,Fred)
child(Pebbles,Wilma)
child(Wilma,Freds-mother-in-law) (what's her name?)
child(Bam-bam,Barney)
child(Bam-bam,Betty)
```

Rules

```
descendent(A,B) := child(A,B)
descendent(A,B) := exists(x : child(A,x) && descendent(x,B))
```

Queries

```
child? (Pebbles,Fred) -> True
child? (Pebbles,Barney) -> False (at least Fred hopes not!)
descendent? (Pebbles,Fred) -> True
descendent? (Pebbles,Freds-mother-in-law)? True
descendent? (Pebbles,Barney) -> False
```

Elementary, my dear Watson!

Imperative programming

Give ordered commands to the computer, statements

Maintain the status in variables that can change value

Procedural language because the code is organized in procedures:

- blocks
- functions
- modules
- packages

```
#!/usr/bin/python
import os
import sys
import string
import optparse
...
from glideinwms.lib import condorExe
# Main function
def main(argv):
  feconfig=frontenvparse.FEConfig()  # FE configuration holder
  … # parse arguments
  feconfig.config_optparse(argparser)
  (options, other_args) = argparser.parse_args(argv[1:])

  if len(other_args)<1:
    raise ValueError, "Missing glidein_name"
  glidein_name = other_args[0]
  if len(other_args)>=2:
    log_type=other_args[1]
  else:
    log_type="STARTD"
  ...
  return 0
# S T A R T U P
if __name__ == '__main__':
  try:
    sys.exit(main(sys.argv))
  except Exception, e:
    sys.stderr.write("ERROR: Exception msg %s\n"%str(e))
    sys.exit(9)
```

Objects (Object Oriented Programming)

Object: Data and methods to manipulate it together as one unit

Class: blueprint to create an object (mold)

Some important properties:

- Abstraction
- Encapsulation
- Polymorphism
- Composition
- Inheritance
- Delegation

OOP concepts

1 .Polymorphism : The process of representing one form in multiple forms is known as Polymorphism.

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person present in different-different behaviors.

2. Abstraction : Abstraction is the concept of exposing only the required essential characteristics and behavior with respect to a context.

Abstraction shows only important things to the user and hides the internal details, for example, when we ride a bike, we only know about how to ride bikes but can not know about how it work? And also we do not know the internal functionality of a bike.

Abstraction is ATM Machine; All are performing operations on the ATM machine like cash withdrawal, money transfer, retrieve mini-statement...etc. but we can't know internal details about ATM.

OOP concepts (cont)

3. Encapsulation = Data Hiding + Abstraction.

Looking at the example of a power steering mechanism of a car. Power steering of a car is a complex system, which internally have lots of components tightly coupled together, they work synchronously to turn the car in the desired direction. It even controls the power delivered by the engine to the steering wheel. But to the external world there is only one interface is available and rest of the complexity is hidden. Moreover, the steering unit in itself is complete and independent. It does not affect the functioning of any other mechanism.

4. Inheritance - Something received from the previous holder

Father gives his property to child , father got that properties from child's grandfather , so child is the taker and father is giver , hence father works as a base class and child as derived class

Object Oriented Programming - Classes definition in shapes.py

Canvas class

Defines a frame buffer

```
class Canvas:
  def __init__(self, width, height):
    self.width = width
    self.height = height
    self.data = [[' '] * width for i in range(height)]

  def setpixel(self, row, col):
    self.data[row][col] = '*'

  def getpixel(self, row, col):
    return self.data[row][col]

  def display(self):
    print ("\n".join(["".join(row) for row in self.data]))
```

Shape class

Abstract base class for shapes. All Shapes can paint themselves on a canvas

```
class Shape:
  def paint(self, canvas): pass
```

Child

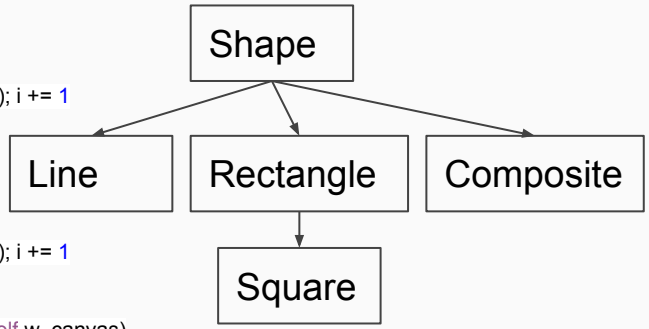
Inherits from Shape, implements paint() and the constructor __init__()

```
class Line(Shape):
  def __init__(self, x1, y1, x2, y2):
    self.x = x1
    self.y = y1
    self.w = x2 - x1
    self.h = y2 - y1

  def paint(self, canvas):
    ratio = self.w / self.h
    if self.w >= self.h:
      for i in range(self.h):
        canvas.setpixel(self.x+int(i*ratio), self.y+i)
    else:
      for i in range(self.w):
        canvas.setpixel(self.x+i, self.y+int(i/ratio))

# Continues on next column ...
```

```
# … Continues from previous column
class Rectangle(Shape):
  def __init__(self, x, y, w, h):
    self.x = x; self.y = y; self.w = w; self.h = h

  @staticmethod
  def hline(x, y, w, canvas):
    i = 0
    while i <= w:
      canvas.setpixel(x+i, y); i += 1

  @staticmethod
  def vline(x, y, h, canvas):
    i = 0
    while i <= h:
      canvas.setpixel(x, y+i); i += 1

  def paint(self, canvas):
    self.hline(self.x, self.y, self.w, canvas)
    self.hline(self.x, self.y + self.h, self.w, canvas)
    self.vline(self.x, self.y, self.h, canvas)
    self.vline(self.x + self.w, self.y, self.h, canvas)


class Square(Rectangle):
  def __init__(self, x, y, size):
    Rectangle.__init__(self, x, y, size, size)


class CompoundShape(Shape):
  def __init__(self, shapes):
    self.shapes = shapes

  def paint(self, canvas):
    for s in self.shapes:
      s.paint(canvas)
```

Inheritance

Tree of shapes



Inheritance (2)

Square is a Rectangle with height = width
Gets paint() from Rectangle

Composition

Uses shapes and that all Shapes can paint

Object Oriented Programming - Using classes (defined in shapes.py)

Import shapes

All classes are defined in the module shapes.py

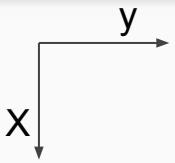
Instantiate the shapes

Creating objects from all the shapes
Using the CompoundShape to paint all at once on mycanvas
NOTE how you use the objects (casts). The classes (molds) are used only to instantiate new objects

```
# Import all the shapes
from shapes import *

# Create a canvas
mycanvas = Canvas(20, 20)

# Draw a rectangle, a square and a line on the canvas
r1 = Rectangle(2, 3, 4, 5)
s1 = Square(4,4,6)
l1 = Line(5, 2, 15, 15)
shapes = [r1, s1, l1]
c1 = CompoundShape(shapes)
c1.paint(mycanvas)

# Show the result
mycanvas.display()
```

Instantiate a Canvas

Creating an object of class (type) Canvas, a frame buffer



Display the result

The Canvas mycanvas is printed on the screen

```
>>> mycanvas.display()
```

```
******
*    *
********
***   * *
****** *
 *     *
 **    *
 *  *  *
 *******
      *
       *
        *
         *
```