Object Oriented Programming in Python

Why OOP ?

- Any program can be implemented in BASIC or C (or assembler).

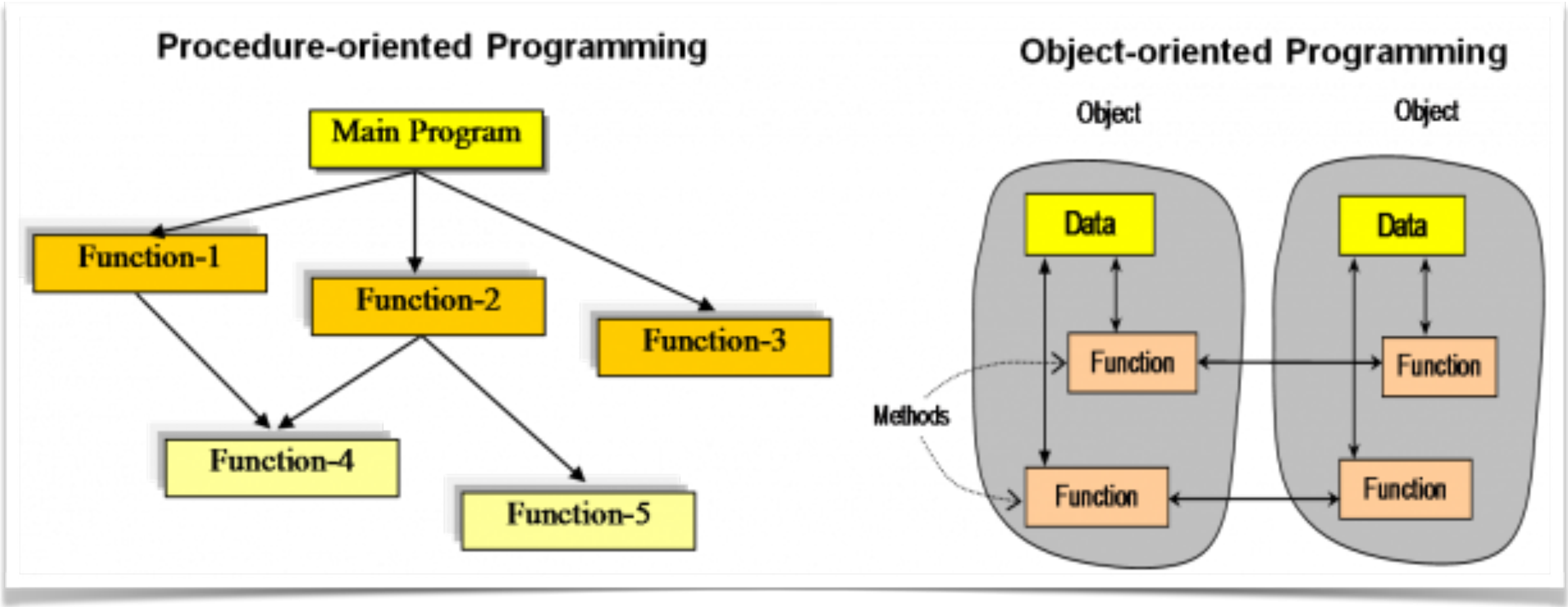
- However, when programs get large, we need to partition the code into short, modular pieces which can be tested and maintained in isolation.

- Code reuse: the “write once” rule. Avoiding duplication of code makes it easier to understand and fix.

- Functions provide a way to partition the code, however, they help little in organizing the data.

- Object oriented programing helps with that.

Procedure vs. Object oriented.



Real World Object

Dog Properties

Color
Eye Color
Height
Length
Weight

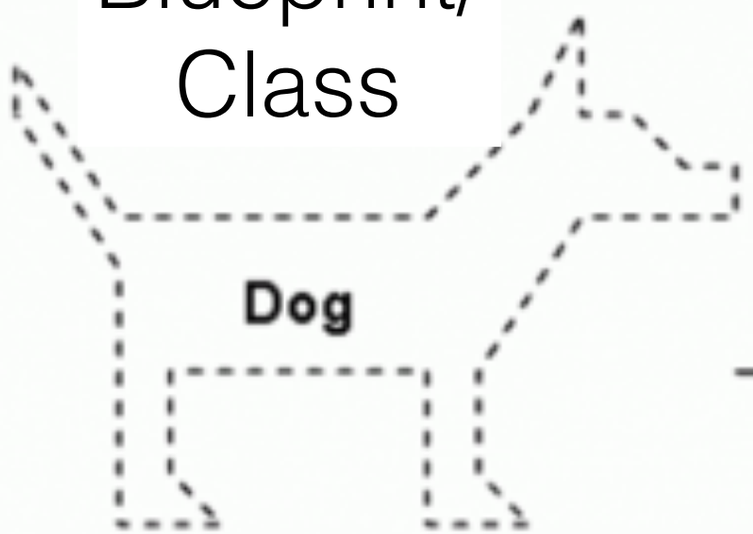
Dog Behavior

Bark
Sit
Lay Down
Shake
Come

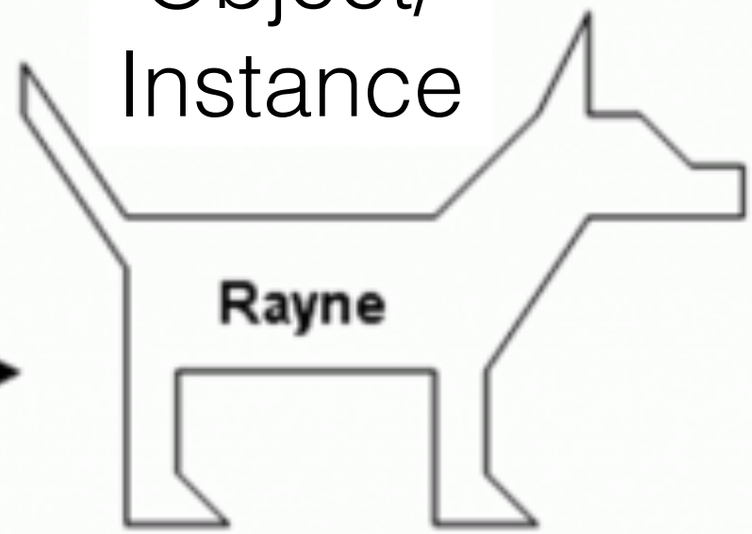
Software Object

Blueprint/
Class

Object/
Instance



Create
Instance



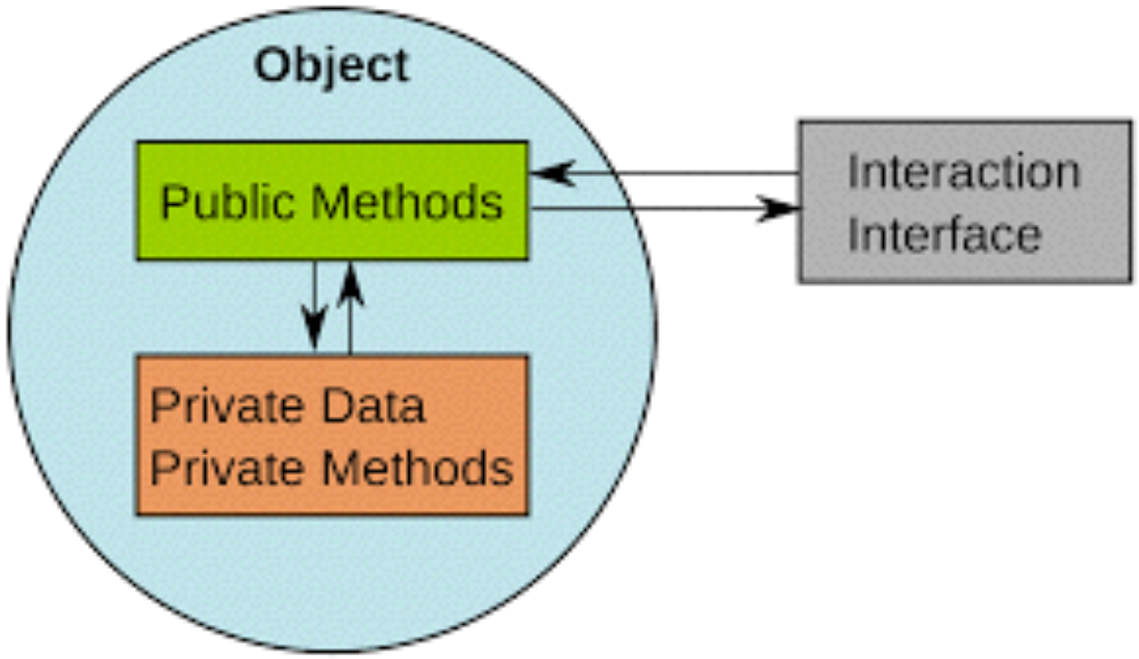
Properties	Methods	Property values	Methods
Color	Sit	Color: Gray, White, and Black	Sit
Eye Color	Lay Down	Eye Color: Blue and Brown	Lay Down
Height	Shake	Height: 18 Inches	Shake
Length	Come	Length: 36 Inches	Come
Weight		Weight: 30 Pounds	

```
class dog:
    """ This Class is a blueprint for digital dogs """
    def __init__(self,name,Color=['gray','white','black'],state='standing'):
        self._name=name;
        self._Color=Color;
        self._state=state;

    def sit(self):
        if self._state='sitting':
            print '%s is already sitting'%self._name;
        else:
            print '%s sits'%slef._name;
            self._state='sitting'

###  Main program that uses the class dog
if __name__ == '__main__':
    D=dog('stan',state='lying down');
    D.sit()
```

Public vs. Private and encapsulation



- **Public** methods (and properties) define the **interface** through which software can interact with the object.
- **Private** methods and properties define the **implementation** of the object.
- Keeping privates hidden ensures that the implementation can be changed without requiring changes in external program.
- For maximal encapsulation, properties (data) are all private and can only be accessed by “get” and “set” methods.
- When partitioning the work on a projects, it is best to start by defining the interfaces. This allows different pieces to be developed independently.

python private and public

- In python the default is that methods and properties are public.
- **By convention** methods and properties whose name starts with underscore “_” are considered private and should only be accessed by methods of the class.

Class hierarchy and inheritance

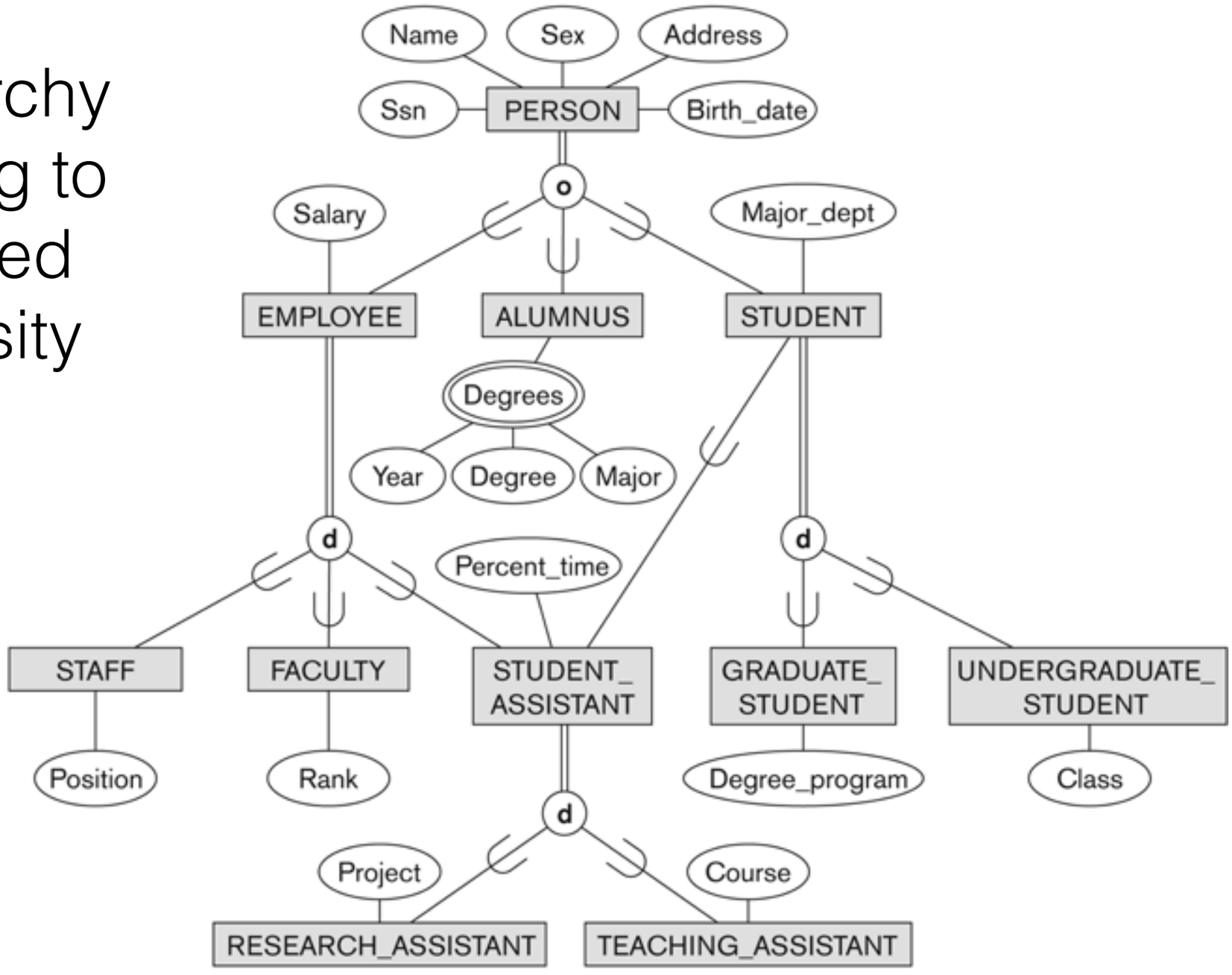
- Sometimes we want to define a class B that is a specialization of an existing class A.

- We say that B is “derived” from A, or that it inherits from A.

- This means that all properties and methods in A are members of B unless they are “over-ridden”.

Class Hierarchy and Inheritance

A class hierarchy corresponding to people affiliated with a University



d=disjoint o=overlapping
Both relevant for database schema not for class Hierarchy.

```
class PERSON:
    def __init__(self,name='john',ssn=0,sex='female',address='',birth_date=0):
        self._name=Name
        self._ssn=ssn
        self._sex=sex
        self._address=address
        self._birth_date=birth_date
    def get_name(self):
        return

class EMPLOYEE(PERSON):
    def __init__(self,salary=0,**kwargs):
        super().__init__(**kwargs)
        self._salary=salary
```

```
class STUDENT(PERSON):
    def __init__(self,major_dept='cse',**kwargs):
        super().__init__(**kwargs)
        self._major_dept=major_dept

class STUDENT_ASSISTANT(STUDENT,EMPLOYEE):
    def __init__(self,percent_time=10,**kwargs):
        super().__init__(**kwargs)
        self._percent_time=percent_time

###  Main program that uses the class Student_assistant
if __name__ == '__main__':
    A=STUDENT_ASSISTANT(name='beth',percent_time=30)
    print A.get_name()
```

Modules

- Files that contain python code (file.py) are called **Modules**. Modules can contain function definitions, class definitions and executable commands. Modules can be used in other modules through the "`import`" command.

- Python files can also be **scripts** - self-standing programs that are executed using
  `python script.py`

- Python files can double as both modules and scripts. The script code is written after the command:
  `if __name__=='__main__':`

- Common uses of scripting inside a module are:

  - To execute the main function of the script from the command line.

  - To test the code of the module.

Packages

- A package is a collection of modules and packages.
- A package corresponds to a directory. The directory contains module files and subdirectories.
- In order to be recognized as a package, the directory has to include a file with the name `__init__.py` (which is usually empty).
- Don’t confuse the Package Hierarchy with the Class Hierarchy. The first deals with the organization of the code into file, the second with functional relations between different classes.

Importing modules

- Suppose the class LipSync is stored in the file `sound/filters/karaoke.py`

  then it can be imported by the command
  `from sound.filters.karaoke import LipSync`

- The directory hierarchy is reflected in the package “dot” syntax.

- Python needs to know the absolute path to the root directory (sound). This can be achieved using:

```
import sys
sys.path.append('/user/yfreund/sound')
```

Example package Hierarchy

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversio
              __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
              ...
```

Summary

- A Class: the blueprint for a type of object
    - Class functions = Methods (get “self” as first parameter) - defines “behavior”.
    - Class Properties = Data structures that define the state of the object.
- Class Inheritance: a way to create new classes that are specializations of existing classes.
- Modules: files containing python code.
- Packages: a hierarchical organization of modules - corresponds to the directory tree containing the module files.