Lecture

Public vs. Private

Public vs. Private

Restrict Access

Information-hiding

Public vs. Private

Public

- Can be accessed anywhere
- No access restriction

Private (non-public)

- Can’t (or shouldn’t) be accessed outside the class
- Two levels of access restriction

Public vs. Private

```
class Car:

    def __init__(self, model, year):
        self.model = model
        self.year = year
```

Public instance attributes

Public vs. Private

```
class Car:

    def __init__(self, model, year):
        self.model = model
        self.year = year
```

Public instance attributes

```
>>> my_car = Car("Escape", 2006)
>>> my_car.model
'Escape'
>>> my_car.year
2006
```

Public vs. Private

```
class Car:

    def __init__(self, model, year):
        self.model = model
        self.year = year
```

Public instance attributes

```
>>> my_car = Car("Escape", 2006)
>>> my_car.model
'Escape'
>>> my_car.year
2006
```

Public vs. Private

```
class Car:

    def __init__(self, model, year):
        self.model = model
        self.year = year
```

Public instance attributes

```
>>> my_car = Car("Escape", 2006)
>>> my_car.model
'Escape'
>>> my_car.year
2006
```

Public vs. Private

```
class Car:

    def __init__(self, model, year):
        self.model = model
        self.year = year
```

Public instance attributes

```
>>> my_car.year = 5600
>>> my_car.year
5600
```

Public vs. Private

```
class Car:

    def __init__(self, model, year):
        self.model = model
        self.year = year
```

Public instance attributes

```
>>> my_car.year = 5600
>>> my_car.year
5600
```

Public vs. Private

```
class Car:

    def __init__(self, model, year):
        self.model = model
        self.year = year
```

Public instance attributes

```
>>> my_car.year = 5600
>>> my_car.year
5600
```

Risk of Public Attributes

Public vs. Private

Non-Public

By Convention
(Protected) → `_<attribute>`

Functionally
(Private) → `__<attribute>`

Name Mangling

Public vs. Private

Non-Public

By Convention
(Protected) → **_<attribute>**

Functionally
(Private) → **__<attribute>**

Name Mangling

Only for special cases

Public vs. Private

Method Names and Instance Variables

Use the function naming rules: lowercase with words separated by underscores as necessary to improve readability.

Use one leading underscore only for non-public methods and instance variables.

To avoid name clashes with subclasses, use two leading underscores to invoke Python's name mangling rules.

Python mangles these names with the class name: if class Foo has an attribute named `__a`, it cannot be accessed by `Foo.__a`. (An insistent user could still gain access by calling `Foo._Foo__a`.) Generally, double leading underscores should be used only to avoid name conflicts with attributes in classes designed to be subclassed.

Note: there is some controversy about the use of __names (see below).

PEP 8

Public vs. Private

```
class Car:

    def __init__(self, model, year, id_num, engine_serial_num):
        self.model = model
        self.year = year
        self._id_num = id_num
        self.__engine_serial_num = engine_serial_num
```

Non-Public Attributes

Public vs. Private

```
class Car:

    def __init__(self, model, year, id_num, engine_serial_num):
        self.model = model
        self.year = year
        self._id_num = id_num
        self.__engine_serial_num = engine_serial_num
```

Non-Public Attributes

Public vs. Private

```
class Car:

    def __init__(self, model, year, id_num, engine_serial_num):
        self.model = model
        self.year = year
        self._id_num = id_num
        self.__engine_serial_num = engine_serial_num
```

Non-Public Attributes

Public vs. Private

```
>>> class Car:

    def __init__(self, model, year, id_num, engine_serial_num):
        self.model = model
        self.year = year
        self._id_num = id_num
        self.__engine_serial_num = engine_serial_num


>>> my_car = Car("Escape", 2006, "44542", "201109048934242")
>>> my_car.__engine_serial_num
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    my_car.__engine_serial_num
AttributeError: 'Car' object has no attribute '__engine_serial_num'
```

Public vs. Private

```
>>> class Car:

    def __init__(self, model, year, id_num, engine_serial_num):
        self.model = model
        self.year = year
        self._id_num = id_num
        self.__engine_serial_num = engine_serial_num


>>> my_car = Car("Escape", 2006, "44542", "201109048934242")
>>> my_car.__engine_serial_num
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    my_car.__engine_serial_num
AttributeError: 'Car' object has no attribute '__engine_serial_num'
```

Public vs. Private

```
>>> class Car:

    def __init__(self, model, year, id_num, engine_serial_num):
        self.model = model
        self.year = year
        self._id_num = id_num
        self.__engine_serial_num = engine_serial_num


>>> my_car = Car("Escape", 2006, "44542", "201109048934242")
>>> my_car.__engine_serial_num
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    my_car.__engine_serial_num
AttributeError: 'Car' object has no attribute '__engine_serial_num'
```

Non-Public?

Public vs. Private

We don't use the term "private" here, since no attribute is really private in Python

PEP 8

Public vs. Private

```
>>> class Car:

    def __init__(self, model, year, id_num, engine_serial_num):
        self.model = model
        self.year = year
        self._id_num = id_num
        self.__engine_serial_num = engine_serial_num


>>> my_car = Car("Escape", 2006, "44542", "201109048934242")
>>> my_car._id_num
'44542'
>>> my_car._Car__engine_serial_num
'201109048934242'
```

Public vs. Private

```
>>> class Car:

    def __init__(self, model, year, id_num, engine_serial_num):
        self.model = model
        self.year = year
        self._id_num = id_num
        self.__engine_serial_num = engine_serial_num


>>> my_car = Car("Escape", 2006, "44542", "201109048934242")
>>> my_car._id_num
'44542'
>>> my_car._Car__engine_serial_num
'201109048934242'
```

Public vs. Private

```
>>> class Car:

    def __init__(self, model, year, id_num, engine_serial_num):
        self.model = model
        self.year = year
        self._id_num = id_num
        self.__engine_serial_num = engine_serial_num


>>> my_car = Car("Escape", 2006, "44542", "201109048934242")
>>> my_car._id_num
'44542'
>>> my_car._Car__engine_serial_num
'201109048934242'
```

Public vs. Private

You can still
access them (technically),
but you shouldn't

Access Modifiers

```
public String model;
public int year;
protected String idNum;
private String EngineSerialNum;
```

Access Modifiers

```
public String model;
public int year;
protected String idNum;
private String EngineSerialNum;
```

Access Modifiers

```
public String model;
public int year;
protected String idNum;
private String EngineSerialNum;
```

Access Modifiers

```
public String model;
public int year;
protected String idNum;
private String EngineSerialNum;
```

No Access Modifiers

No Access Modifiers

Conventions

Public vs. Private

_<elem>

Not imported

from <module> import *

Public vs. Private

Encapsulation

- **Access**
  - **Public**
  - **Non-Public**
    - **By Convention** (Protected) → `_<attribute>`
    - **Functionally** (Private) → `__<attribute>`
      - Name Mangling

Now… An Example