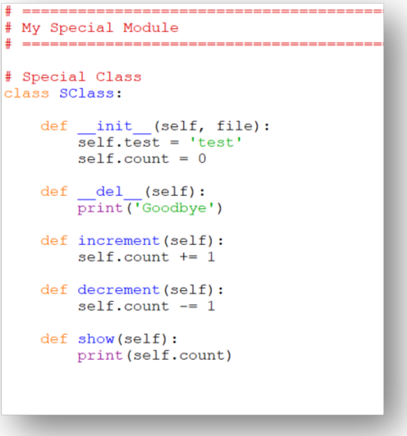
Interface Module

special.py

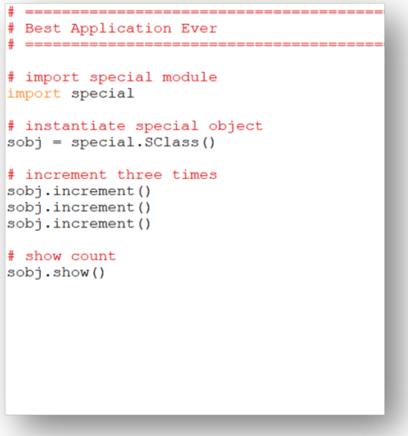


```
# =======================================
# My Special Module
# =======================================

# Special Class
class SClass:

    def __init__(self, file):
        self.test = 'test'
        self.count = 0

    def __del__(self):
        print('Goodbye')

    def increment(self):
        self.count += 1

    def decrement(self):
        self.count -= 1

    def show(self):
        print(self.count)
```

Python Module

import

best_app_ever.py



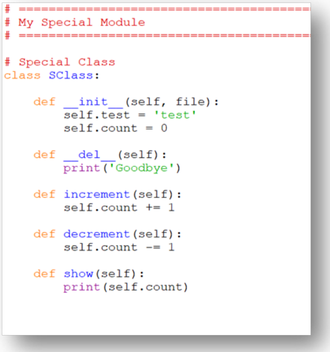
```
# =======================================
# Best Application Ever
# =======================================

# import special module
import special

# instantiate special object
sobj = special.SClass()

# increment three times
sobj.increment()
sobj.increment()
sobj.increment()

# show count
sobj.show()
```

Python Application

Content

- Python Class (refresher)
- Reusable Module
- Apply

special.py

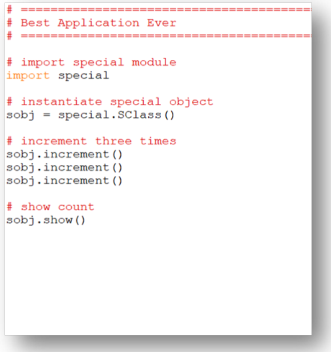


```
# ========================================
# My Special Module
# ========================================

# Special Class
class SClass:

    def __init__(self, file):
        self.test = 'test'
        self.count = 0

    def __del__(self):
        print('Goodbye')

    def increment(self):
        self.count += 1

    def decrement(self):
        self.count -= 1

    def show(self):
        print(self.count)
```

Python Module

import

best_app_ever.py

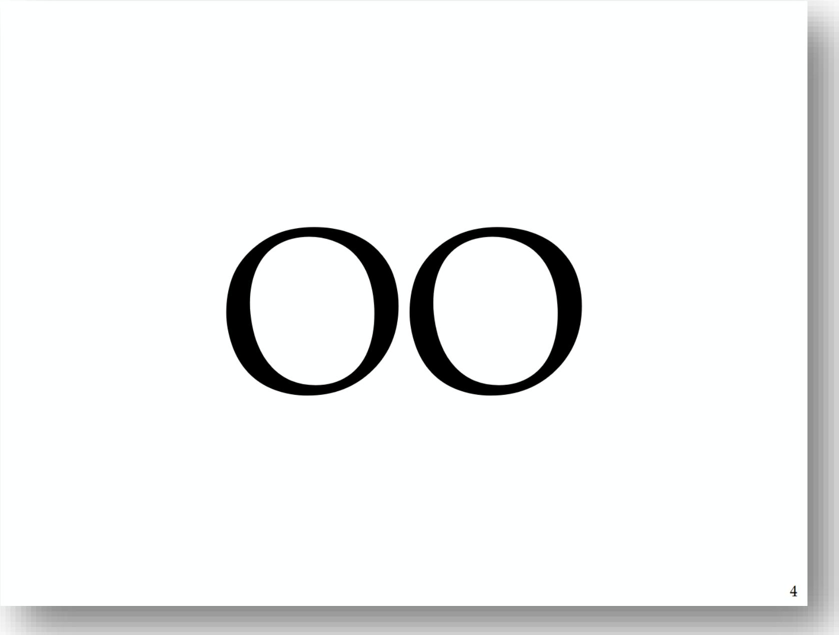


```
# ========================================
# Best Application Ever
# ========================================

# import special module
import special

# instantiate special object
sobj = special.SClass()

# increment three times
sobj.increment()
sobj.increment()
sobj.increment()

# show count
sobj.show()
```

Python Application

Content

- Python Class (refresher)
- Reusable Module
- Apply



OO

4

Python Class

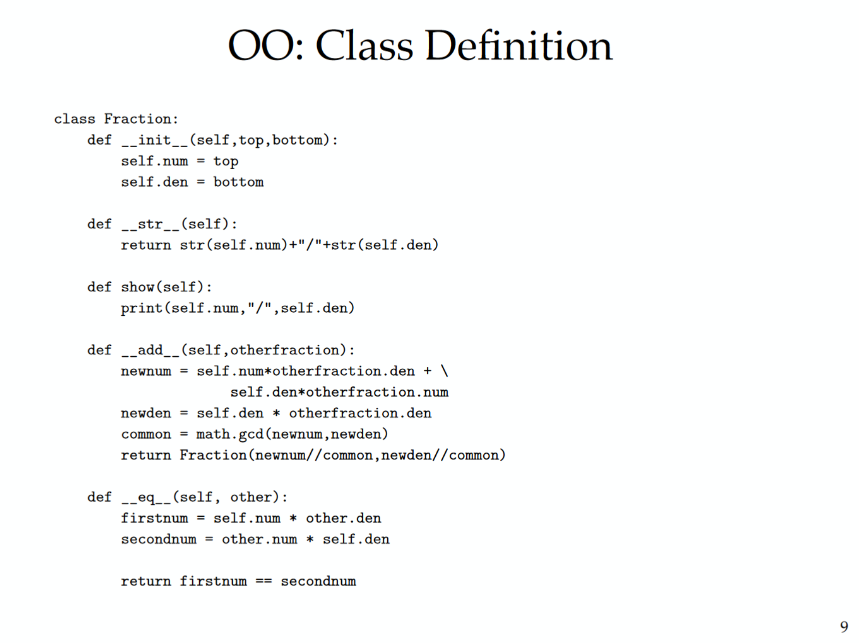
Programming, Algorithms and Data Structures

Definition

Instantiation

Methods

Variables



OO: Class Definition

```
class Fraction:
    def __init__(self,top,bottom):
        self.num = top
        self.den = bottom

    def __str__(self):
        return str(self.num)+"/"+str(self.den)

    def show(self):
        print(self.num,"/",self.den)

    def __add__(self,otherfraction):
        newnum = self.num*otherfraction.den + \
                     self.den*otherfraction.num
        newden = self.den * otherfraction.den
        common = math.gcd(newnum,newden)
        return Fraction(newnum//common,newden//common)

    def __eq__(self, other):
        firstnum = self.num * other.den
        secondnum = other.num * self.den

        return firstnum == secondnum
```

9

Python Class

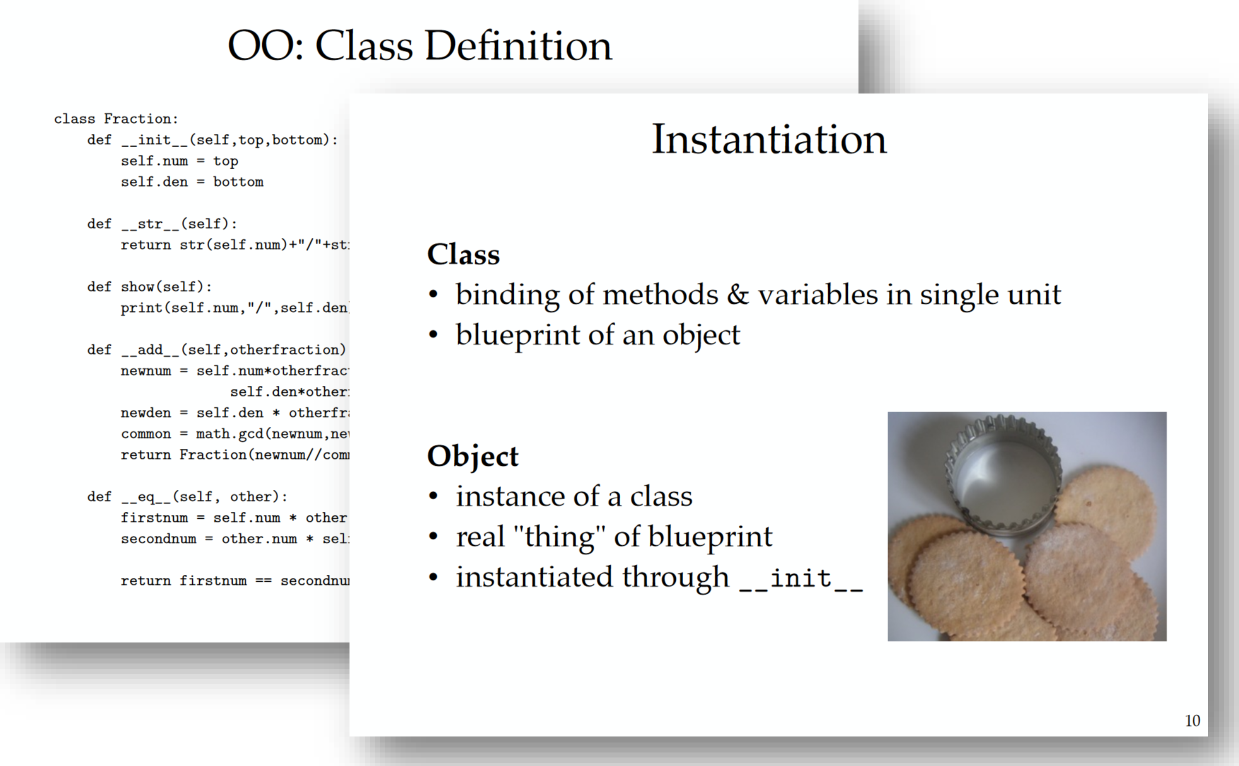
Programming, Algorithms and Data Structures

Definition

Instantiation

Methods

Variables



OO: Class Definition

```
class Fraction:
    def __init__(self,top,bottom):
        self.num = top
        self.den = bottom

    def __str__(self):
        return str(self.num)+"/"+st

    def show(self):
        print(self.num,"/",self.den

    def __add__(self,otherfraction)
        newnum = self.num*otherfrac
                     self.den*other
        newden = self.den * otherfr
        common = math.gcd(newnum,ne
        return Fraction(newnum//com

    def __eq__(self, other):
        firstnum = self.num * other
        secondnum = other.num * sel

        return firstnum == secondnu
```

Instantiation

Class

- binding of methods & variables in single unit
- blueprint of an object

Object

- instance of a class
- real "thing" of blueprint
- instantiated through `__init__`

10

Python Class

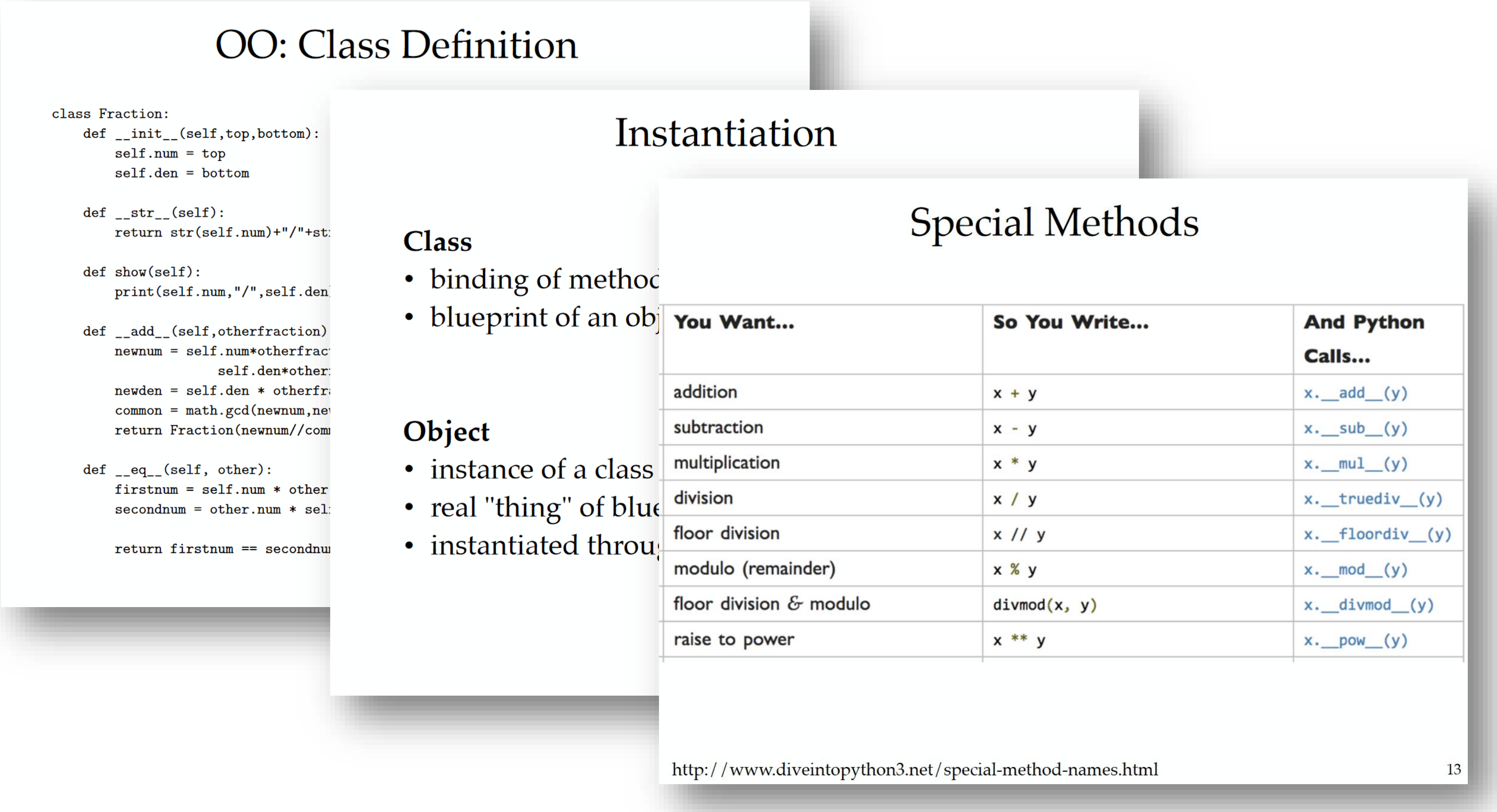
Programming, Algorithms and Data Structures

Definition

Instantiation

Methods

Variables



OO: Class Definition

```
class Fraction:
    def __init__(self,top,bottom):
        self.num = top
        self.den = bottom

    def __str__(self):
        return str(self.num)+"/"+st

    def show(self):
        print(self.num,"/",self.den

    def __add__(self,otherfraction)
        newnum = self.num*otherfrac
                     self.den*other
        newden = self.den * otherfra
        common = math.gcd(newnum,ne
        return Fraction(newnum//com

    def __eq__(self, other):
        firstnum = self.num * other
        secondnum = other.num * sel

        return firstnum == secondnu
```

Instantiation

Class

- binding of methoc
- blueprint of an obj

Object

- instance of a class
- real "thing" of blue
- instantiated throu

Special Methods

You Want...	So You Write...	And Python Calls...
addition	x + y	x.__add__(y)
subtraction	x - y	x.__sub__(y)
multiplication	x * y	x.__mul__(y)
division	x / y	x.__truediv__(y)
floor division	x // y	x.__floordiv__(y)
modulo (remainder)	x % y	x.__mod__(y)
floor division & modulo	divmod(x, y)	x.__divmod__(y)
raise to power	x ** y	x.__pow__(y)

http://www.diveintopython3.net/special-method-names.html 13

Python Class

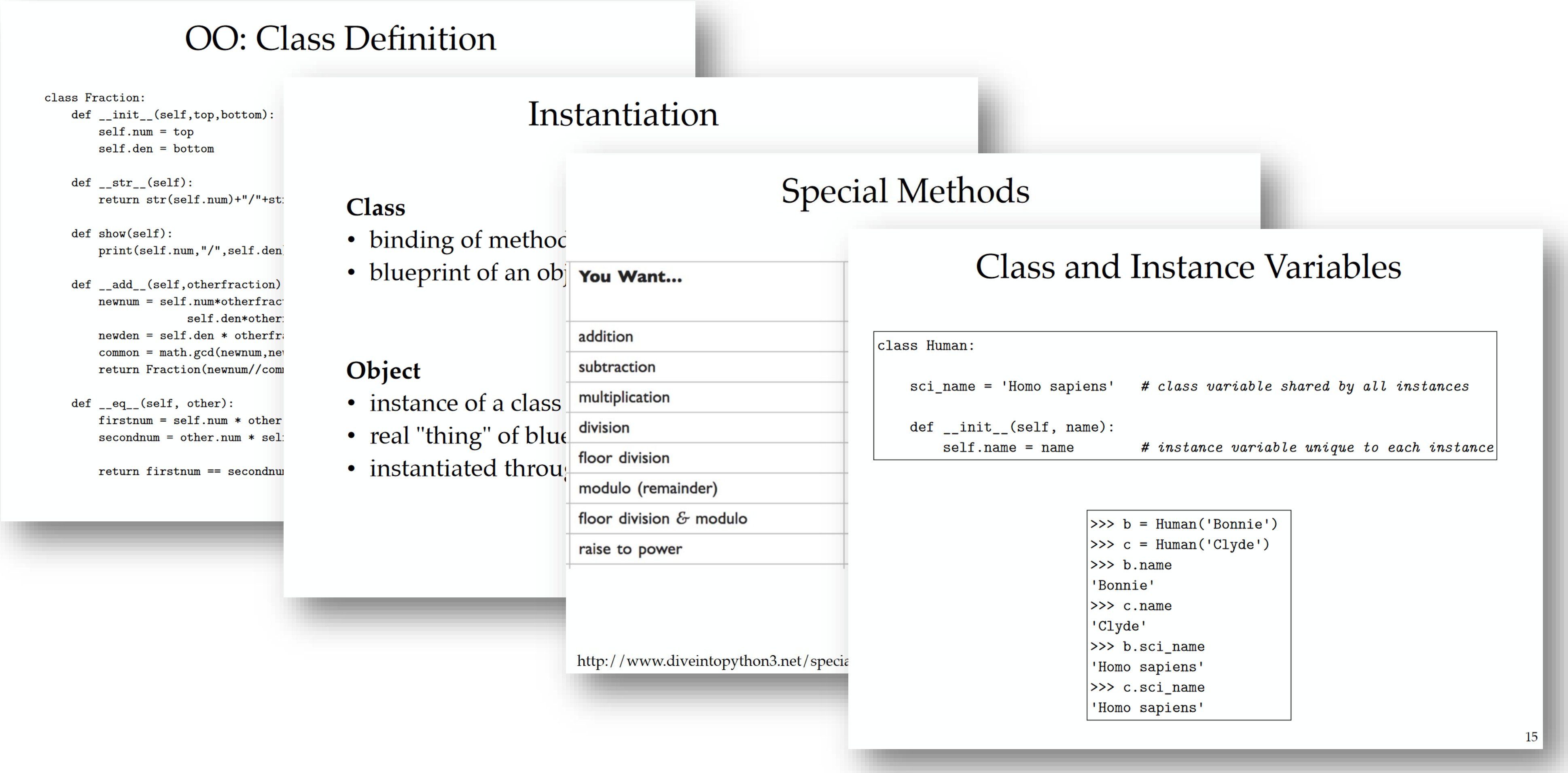
Programming, Algorithms and Data Structures

Definition

Instantiation

Methods

Variables



OO: Class Definition

```
class Fraction:
    def __init__(self,top,bottom):
        self.num = top
        self.den = bottom

    def __str__(self):
        return str(self.num)+"/"+st

    def show(self):
        print(self.num,"/",self.den

    def __add__(self,otherfraction)
        newnum = self.num*otherfrac
                     self.den*other
        newden = self.den * otherfr
        common = math.gcd(newnum,ne
        return Fraction(newnum//com

    def __eq__(self, other):
        firstnum = self.num * other
        secondnum = other.num * sel

        return firstnum == secondnu
```

Instantiation

Class

- binding of methoc
- blueprint of an obj

Object

- instance of a class
- real "thing" of blue
- instantiated throu

Special Methods

You Want...
addition
subtraction
multiplication
division
floor division
modulo (remainder)
floor division & modulo
raise to power

http://www.diveintopython3.net/specia

Class and Instance Variables

```
class Human:

    sci_name = 'Homo sapiens'   # class variable shared by all instances

    def __init__(self, name):
        self.name = name        # instance variable unique to each instance
```

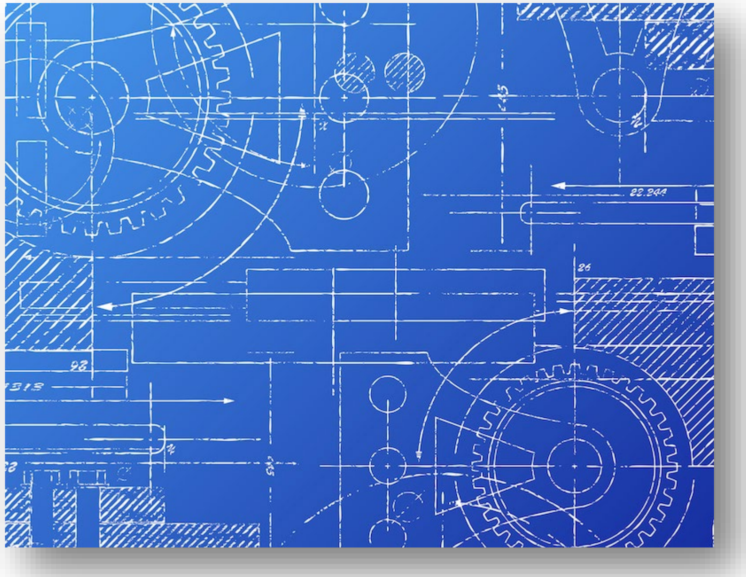
```
>>> b = Human('Bonnie')
>>> c = Human('Clyde')
>>> b.name
'Bonnie'
>>> c.name
'Clyde'
>>> b.sci_name
'Homo sapiens'
>>> c.sci_name
'Homo sapiens'
```

15

Python Class

Definition

```
# dog class
class Dog:

    type = 'carnivor'

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

    def __del__(self):
        print(self.name,'has left the building')

    def talk(self):
        print('wuff wuff')
```



Python Class

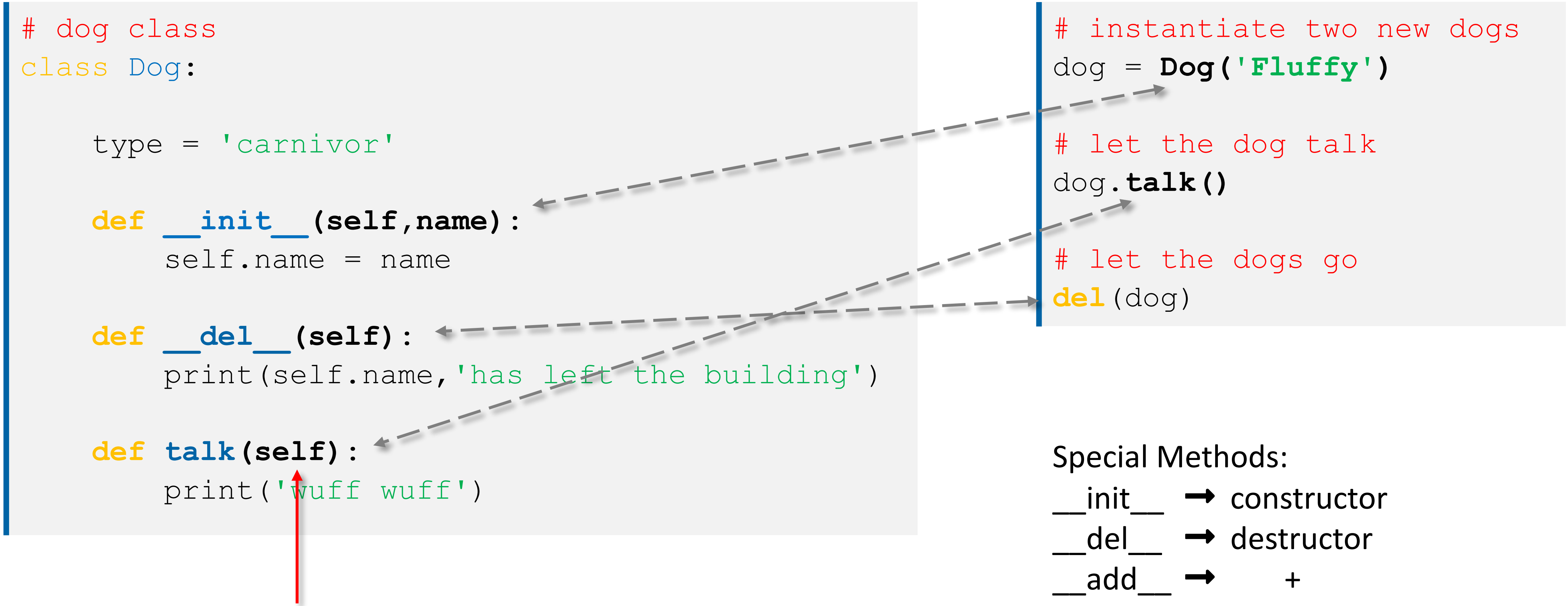
Instantiation and Use

```
# instantiate two new dogs
dog1 = Dog('Fluffy')
dog2 = Dog(‘Churchill')

# show type
print(dog1.type)
print(dog2.type)

# let the dogs talk
dog1.talk()
dog2.talk()

# let the dogs go
del(dog1)
del(dog2)
```

```
# dog class
class Dog:

    type = 'carnivor'

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

    def __del__(self):
        print(self.name,'has left the building')

    def talk(self):
        print('wuff wuff')
```

Python Class

Methods



```
# dog class
class Dog:

    type = 'carnivor'

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

    def __del__(self):
        print(self.name,'has left the building')

    def talk(self):
        print('wuff wuff')
```

```
# instantiate two new dogs
dog = Dog('Fluffy')

# let the dog talk
dog.talk()

# let the dogs go
del(dog)
```

in Python, the first argument is always
a reference to the object itself

Special Methods:
__init__ ➞ constructor
__del__ ➞ destructor
__add__ ➞ +

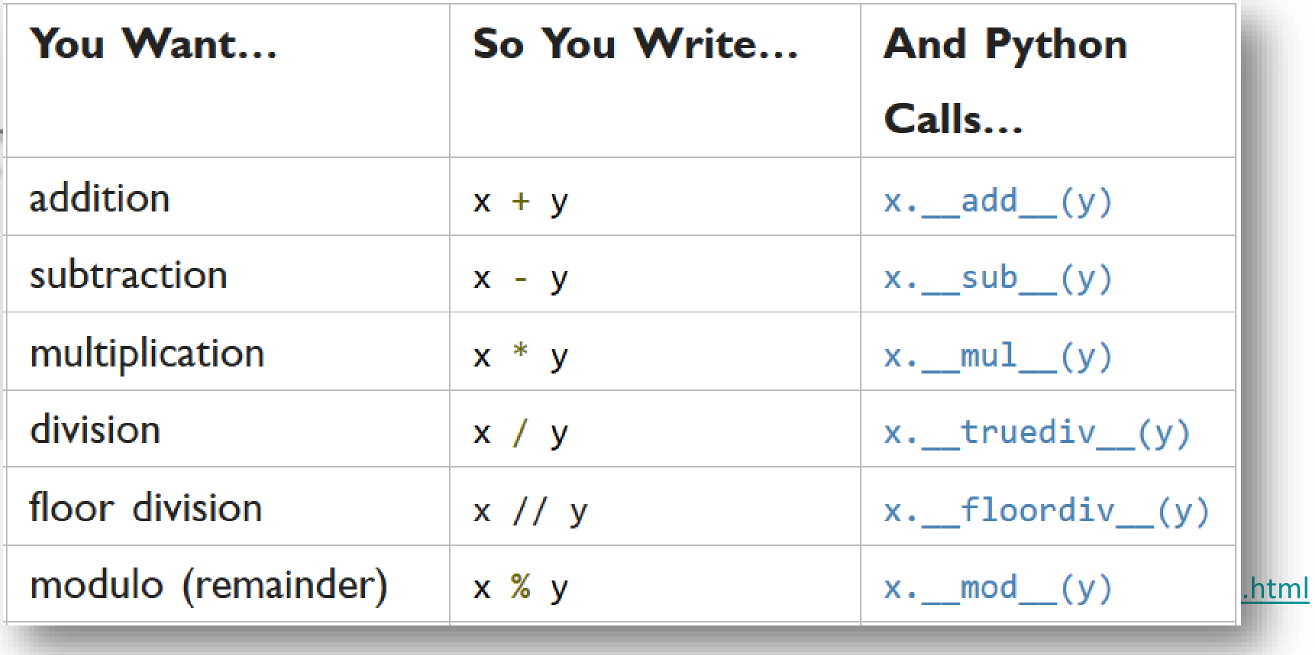
http://www.diveintopython3.net/special-method-names.html

Python Class

Methods

```
# dog class
class Dog:

    type = 'carnivor'

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

    def __del__(self):
        print(self.name,'has left the

    def talk(self):
        print('wuff wuff')
```

```
# instantiate two new dogs
dog = Dog('Fluffy')

# let the dog talk
dog.talk()
```



You Want...	So You Write...	And Python Calls...
addition	x + y	x.__add__(y)
subtraction	x - y	x.__sub__(y)
multiplication	x * y	x.__mul__(y)
division	x / y	x.__truediv__(y)
floor division	x // y	x.__floordiv__(y)
modulo (remainder)	x % y	x.__mod__(y)

.html

in Python, the first argument is always
a reference to the object itself

Python Class

Variables

class variable

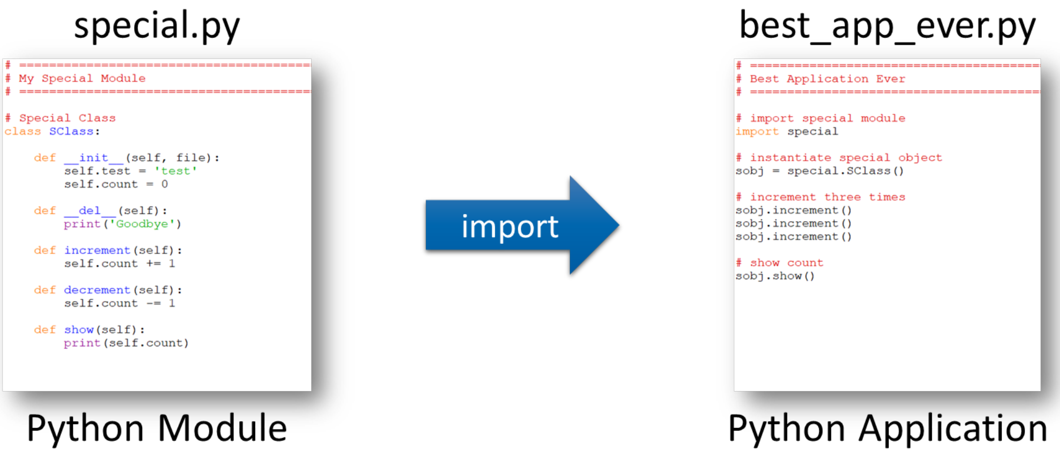
```
# dog class
class Dog:

    type = 'carnivor'

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

    def __del__(self):
        print(self.name,'has left the building')

    def talk(self):
        print('wuff wuff')
```

```
# instantiate a new dog
dog = Dog('Fluffy')

# get the type
print(dog.type)

# show the dog’s name
print(dog.name)
```

instance/object variable

Content

- Python Class (refresher)
- **Reusable Module**
- Apply



special.py

```python
# ========================================
# My Special Module
# ========================================

# Special Class
class SClass:

    def __init__(self, file):
        self.test = 'test'
        self.count = 0

    def __del__(self):
        print('Goodbye')

    def increment(self):
        self.count += 1

    def decrement(self):
        self.count -= 1

    def show(self):
        print(self.count)
```

Python Module

import

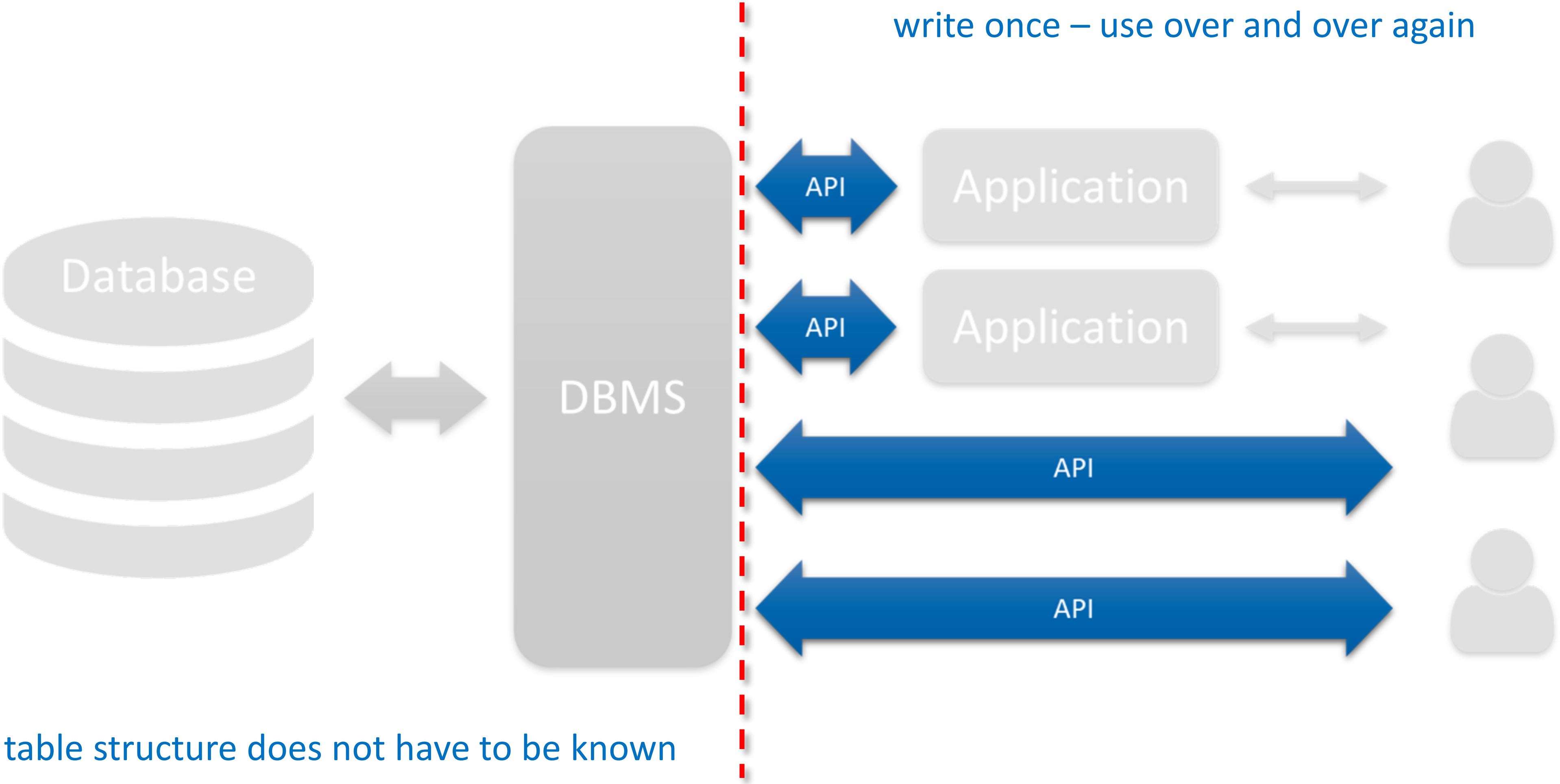
best_app_ever.py

```python
# ========================================
# Best Application Ever
# ========================================

# import special module
import special

# instantiate special object
sobj = special.SClass()

# increment three times
sobj.increment()
sobj.increment()
sobj.increment()

# show count
sobj.show()
```

Python Application

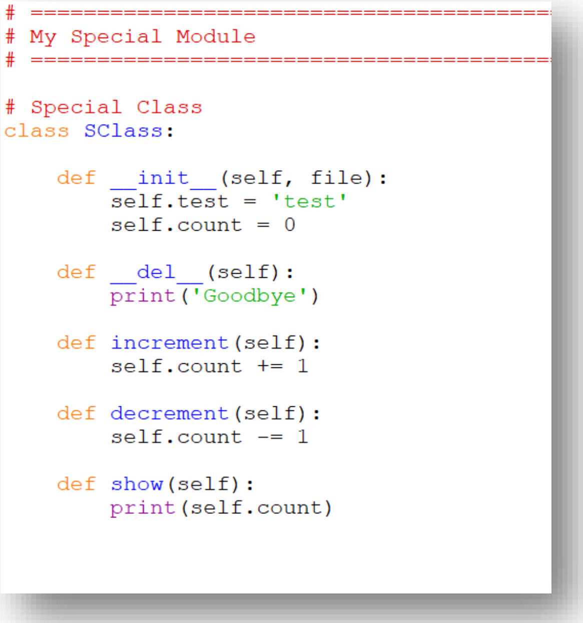
Reusable Module

Specific API


write once – use over and over again
Database
DBMS
API
Application
API
Application
API
API
table structure does not have to be known

Reusable Module

special.py

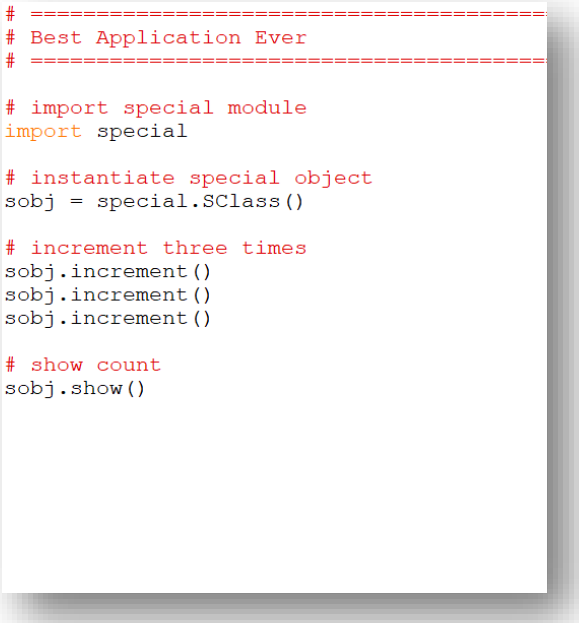


```
# ========================================
# My Special Module
# ========================================

# Special Class
class SClass:

    def __init__(self, file):
        self.test = 'test'
        self.count = 0

    def __del__(self):
        print('Goodbye')

    def increment(self):
        self.count += 1

    def decrement(self):
        self.count -= 1

    def show(self):
        print(self.count)
```

Python Module

import

best_app_ever.py

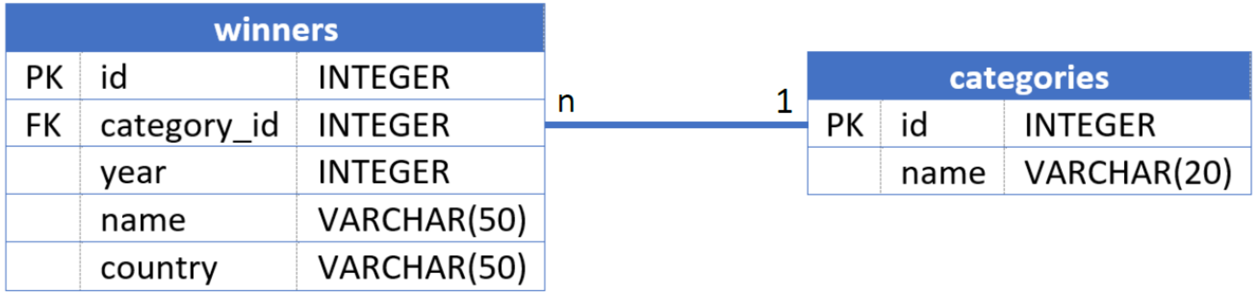


```
# ========================================
# Best Application Ever
# ========================================

# import special module
import special

# instantiate special object
sobj = special.SClass()

# increment three times
sobj.increment()
sobj.increment()
sobj.increment()

# show count
sobj.show()
```

Python Application

Content

- Python Class (refresher)
- Reusable Module
- Apply

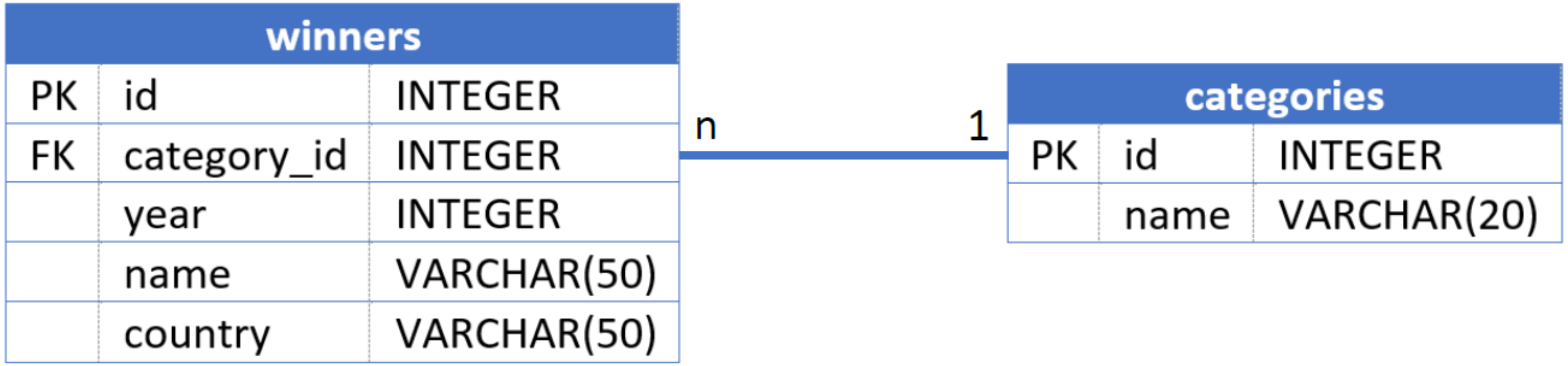


winners		
PK	id	INTEGER
FK	category_id	INTEGER
	year	INTEGER
	name	VARCHAR(50)
	country	VARCHAR(50)

categories		
PK	id	INTEGER
	name	VARCHAR(20)

Apply

Let’s write an interface module for the sqlite database ***nobel.sqlite*** created in the **exercise last week**



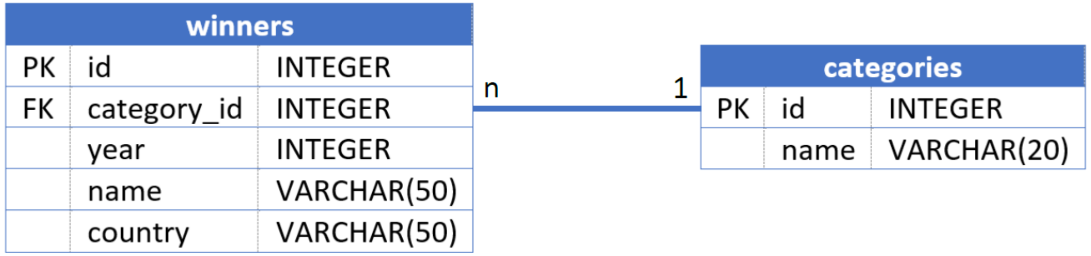
winners		
PK	id	INTEGER
FK	category_id	INTEGER
	year	INTEGER
	name	VARCHAR(50)
	country	VARCHAR(50)

categories		
PK	id	INTEGER
	name	VARCHAR(20)

ER-Model of the *nobel.sqlite* database

Apply

Interface Class



winners

PK	id	INTEGER
FK	category_id	INTEGER
	year	INTEGER
	name	VARCHAR(50)
	country	VARCHAR(50)

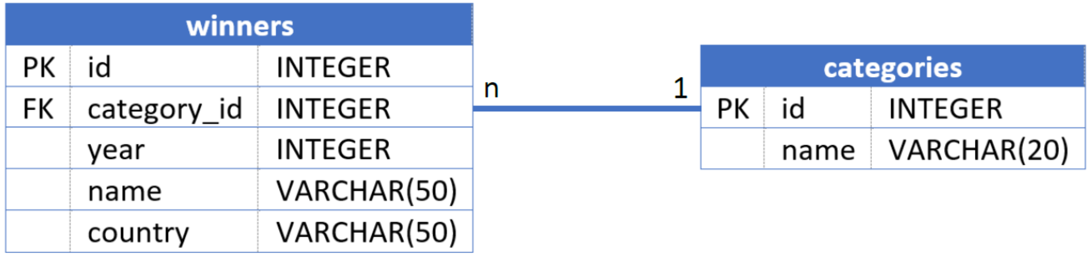
categories

PK	id	INTEGER
	name	VARCHAR(20)

```python
# import
import sqlite3

# define class
class NobelAPI:

    dbfile = 'nobel.sqlite'

    def __init__(self):
        self.connector = sqlite3.connect(self.dbfile)
        self.cursor = self.connector.cursor()

    def __del__(self):
        self.connector.close()
```

Apply

Insert



winners

PK	id	INTEGER
FK	category_id	INTEGER
	year	INTEGER
	name	VARCHAR(50)
	country	VARCHAR(50)

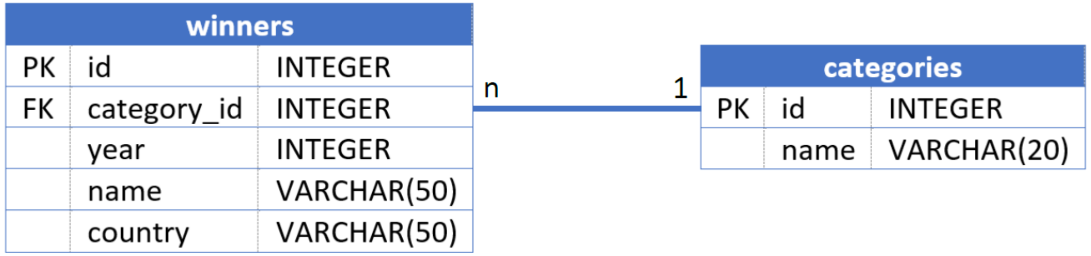
categories

PK	id	INTEGER
	name	VARCHAR(20)

```
def add_winner(self, attributes):
    query = '''INSERT INTO winners (category_id,year,name,country)
               VALUES (?,?,?,?)'''
    self.cursor.execute(query, attributes)
    self.connector.commit()
```

Apply

Retrieve



winners		
PK	id	INTEGER
FK	category_id	INTEGER
	year	INTEGER
	name	VARCHAR(50)
	country	VARCHAR(50)

categories		
PK	id	INTEGER
	name	VARCHAR(20)

```
def get_winners(self):
    query = '''SELECT * FROM winners'''
    self.cursor.execute(query)
    return self.cursor.fetchall()

def get_category_id(self,id):
    query = '''SELECT * FROM winners WHERE id=?'''
    self.cursor.execute(query,[id,])
    return self.cursor.fetchone()

def get_category_names(self):
    query = '''SELECT name FROM categories'''
    self.cursor.execute(query)
    return self.cursor.fetchall()
```