08-Python面向对象编程

创建和使用类

- 使用class关键字创建类
- 类名使用驼峰命名法，每个单词的首字母大写，例如：MyClass、BattleShip
- 但文件名使用小写: dog.py, model.py

In [ ]:
```
class Dog:
    """Emulate a dog"""

    def __init__(self, name, age):
        """Initialize name and age attributes"""
        self.name = name
        self.age = age

    def sit(self):
        """Simulate a dog sitting in response to a command"""
        print(f"{self.name} is now sitting.")

    def roll_over(self):
        """Simulate a dog rolling over in response to a command"""
        print(f"{self.name} rolled over!")

my_dog = Dog('Willie', 4)

# Python的属性和方法都不是私有的
print(my_dog.name, my_dog.age)
my_dog.sit()
my_dog.roll_over()
```

类的定义

- 类中的定义函数被称为方法(method)，类中定义的变量被称为属性（attribute）
- init这样以下划线开始和结束的方法或者属性都是特殊方法或者属性，特殊方法和属性通常被隐式地使用-
- 这里的方法的第一个参数都是self，self表示自己这个对象，通过self可以使用该对象的属性和方法

Car类的定义

- 属性的默认值
- 汽车里程应该只能增加，不能减少

In [ ]:
```
class Car:
    """A simple attempt to represent a car."""

    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        """
        Set the odometer reading to the given value.
        Reject the change if it attempts to roll the odometer back.
        """
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles

my_used_car = Car('subaru', 'outback', 2015)
print(my_used_car.get_descriptive_name())

my_used_car.update_odometer(23_500)
my_used_car.read_odometer()

my_used_car.increment_odometer(100)
my_used_car.read_odometer()
```

In [ ]:
```
# 无法阻止直接修改属性值
my_used_car.odometer_reading = 0
my_used_car.read_odometer()
```

继承

子类将自动获得另一个类（父类）的所有属性和方法，子类还可以定义自己的属性和方法。

In [ ]:
```python
class ElectricCar(Car):
    """Represent aspects of a car, specific to electric vehicles."""

    def __init__(self, make, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """

        # 调用父类的构造函数
        super().__init__(make, model, year)
        self.battery_size = 75

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

my_tesla = ElectricCar('tesla', 'model s', 2019)
print(my_tesla.get_descriptive_name())
my_tesla.describe_battery()
```

组合类，类的属性可以是自定义类的对象

In [ ]:
```python
class Battery:
    """A simple attempt to model a battery for an electric car."""

    def __init__(self, battery_size=75):
        """Initialize the battery's attributes."""
        self.battery_size = battery_size

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 75:
            range = 260
        elif self.battery_size == 100:
            range = 315

        print(f"This car can go about {range} miles on a full charge.")


class ElectricCar(Car):
    """Represent aspects of a car, specific to electric vehicles."""

    def __init__(self, make, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(make, model, year)
        self.battery = Battery()

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

my_tesla = ElectricCar('tesla', 'model s', 2019)
print(my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
my_tesla.battery.get_range()
```

重写（override）父类的方法

子类和父类完全一样的方法（相同的函数名，相同的参数列表），但是会表现出不同的行为，这就是重写（override）。

In [ ]:
```python
class Car:
    """A simple attempt to represent a car."""

    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def fill_gas_tank(self):
        """Fill gas tank"""
        print("Fill gas tank")

class ElectricCar(Car):
    """Represent aspects of a car, specific to electric vehicles."""

    def __init__(self, make, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(make, model, year)
        self.battery = Battery()

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

    def fill_gas_tank(self):
        """Electric car doesn't need a gas tank"""
        print("This car doesn't need a gas tank!")

my_tesla = ElectricCar('tesla', 'model s', 2019)
my_tesla.fill_gas_tank()
```

Random类

- Random类产生的伪随机数，当种子确定时，产生的随机数时确定的。
- randint方法: 产生随机整数
- choice:随机选择一个
- sample:随机选择若干个样本

In [ ]:
```python
import random

# random.seed(0)
print(random.randint(1, 6))
print(random.choice(['apple', 'pear', 'banana']))
print(random.sample(range(100), 10))
```

Python类的一些特殊方法：

- __str__
- __repr__
- __eq__
- __iter__
- __len__
- __getitem__
- __abs__

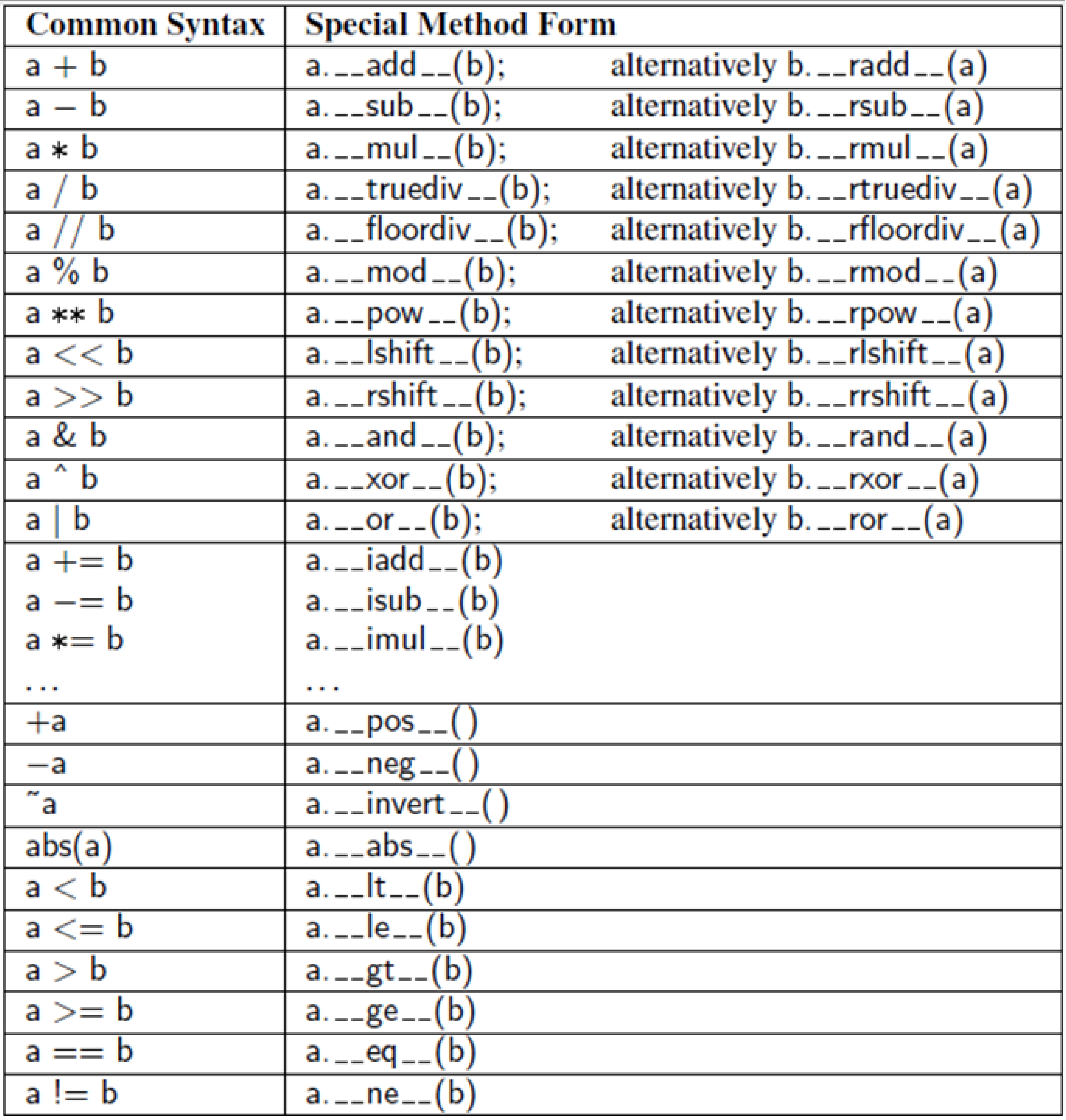
In [67]:
```
class Vector:

    # _components是特殊属性，存放表示向量的列表
    def __init__(self, components):
        self._components = tuple(components)

    def __iter__(self):
        return iter(self._components)

    def __repr__(self):
        return f'Vector({self._components})'

    def __str__(self):
        return str(tuple(self))

    def __eq__(self, other):
        return tuple(self) == tuple(other)

    def __hash__(self):
        return hash(self._components)

    def __len__(self):
        return len(self._components)

    def __getitem__(self, position):
        return self._components[position]

    def __abs__(self):
        return (sum(x**2 for x in self))**0.5

    def __add__(self, other):
        try:
            if len(self) != len(other):
                raise TypeError()
            pairs = zip(self, other)
            return Vector(a + b for a, b in pairs)
        except TypeError:
            print("Dimensions must agree")
            return NotImplemented


    def __radd__(self, other):
        return self + other
```

In [51]:
```
v1 = Vector([1, 2, 3])
print(v1)  #  __str__ called
```

```
(1, 2, 3)
```

In [6]:
```
v1 # __repr__ called
```
Out[6]:
```
Vector([1, 2, 3])
```
In [7]:
```
v2 = Vector([1, 2, 3])
v1 == v2 # __eq__ called
```
Out[7]:
```
True
```
In [60]:
```
print(hash(v1)) # __hash__ called
print({v1})
```
```
529344067295497451
{Vector((1, 2, 3))}
```
In [8]:
```
for value in v1:  # __iter__ called
    print(value)
```
```
1
2
3
```
In [9]:
```
len(v1)  # __len__ called
```
Out[9]:
```
3
```
In [69]:
```
print(v1[0]) # __getitem__ called
print(v1[1])
print(v1[2])
print(v1[1:]) # slice is supported
```
```
1
2
3
(2, 3)
```
In [ ]:
```
abs(v1) # __abs__ called
```

操作符重载



Common Syntax	Special Method Form	
a + b	a.__add__(b);	alternatively b.__radd__(a)
a − b	a.__sub__(b);	alternatively b.__rsub__(a)
a ∗ b	a.__mul__(b);	alternatively b.__rmul__(a)
a / b	a.__truediv__(b);	alternatively b.__rtruediv__(a)
a // b	a.__floordiv__(b);	alternatively b.__rfloordiv__(a)
a % b	a.__mod__(b);	alternatively b.__rmod__(a)
a ∗∗ b	a.__pow__(b);	alternatively b.__rpow__(a)
a << b	a.__lshift__(b);	alternatively b.__rlshift__(a)
a >> b	a.__rshift__(b);	alternatively b.__rrshift__(a)
a & b	a.__and__(b);	alternatively b.__rand__(a)
a ^ b	a.__xor__(b);	alternatively b.__rxor__(a)
a | b	a.__or__(b);	alternatively b.__ror__(a)

a += b
a −= b
a ∗= b
. . .
	a.__iadd__(b)
	a.__isub__(b)
	a.__imul__(b)
	. . .	

+a	a.__pos__()	
−a	a.__neg__()	
˜a	a.__invert__()	
abs(a)	a.__abs__()	
a < b	a.__lt__(b)	
a <= b	a.__le__(b)	
a > b	a.__gt__(b)	
a >= b	a.__ge__(b)	
a == b	a.__eq__(b)	
a != b	a.__ne__(b)	

In [10]:
```
v1 + v2 # __add__ called
```

Out[10]:
```
Vector([2, 4, 6])
```

In [11]:
```
v1 += v2 # __add__ called
v1
```

Out[11]:
```
Vector([2, 4, 6])
```

In [26]:
```
Vector([1, 2]) + Vector([1, 2, 3]) # TypeError: dimensions must agree
```

```
Dimensions must agree

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
c:\Users\zhouj\workspace\python_course\src\08-classes\08-classes.ipynb Cell 28 line
1
----> <a href='vscode-notebook-cell:/c%3A/Users/zhouj/workspace/python_course/src/0
8-classes/08-classes.ipynb#X46sZmlsZQ%3D%3D?line=0'>1</a> Vector([1, 2]) + Vector([1,
2, 3]) # TypeError: dimensions must agree

TypeError: unsupported operand type(s) for +: 'Vector' and 'Vector'
```

namedtuple

- 类似于元组（tuple）对象，也是不可变的
- 但是它的数据域是由像字典一样有名字的。

In [45]:
```
from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])
p1 = Point(1, 2)
print(p1)
print(p1.x, p1.y)
print(p1[0], p1[1])
print(p1 == Point(1, 2))
print({p1, p1})
```

```
Point(x=1, y=2)
1 2
1 2
True
{Point(x=1, y=2)}
```

In [31]:
```
# namedtupe是tuple的子类
print(isinstance(p1, tuple))
```

```
True
```

In [32]:
```
p1.x = 100 # AttributeError: can't set attribute
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
c:\Users\zhouj\workspace\python_course\src\08-classes\08-classes.ipynb Cell 32 line
1
----> <a href='vscode-notebook-cell:/c%3A/Users/zhouj/workspace/python_course/src/0
8-classes/08-classes.ipynb#X53sZmlsZQ%3D%3D?line=0'>1</a> p1.x = 100 # AttributeError:
can't set attribute

AttributeError: can't set attribute
```

FrenchDeck

In [63]:
```
import collections

Card = collections.namedtuple('Card', ['rank', 'suit'])

class FrenchDeck:
    ranks = [str(n) for n in range(2, 11)] + list('JQKA')
    suits = 'spades diamonds clubs hearts'.split()

    def __init__(self):
        self._cards = [Card(rank, suit) for suit in self.suits
                                        for rank in self.ranks]

    def __len__(self):
        return len(self._cards)

    def __getitem__(self, position):
        return self._cards[position]
```

In [64]:
```
beer_card = Card('7', 'diamonds')
beer_card
```

Out[64]:
```
Card(rank='7', suit='diamonds')
```

In [66]:
```
deck = FrenchDeck()
len(deck)
```

Out[66]:
```
52
```

In [72]:
```
print(deck[0])
print(deck[:3])
```

```
Card(rank='2', suit='spades')
[Card(rank='2', suit='spades'), Card(rank='3', suit='spades'), Card(rank='4', suit='spades')]
```

In [ ]:
```
for card in deck:
    print(card)
```

DataClass

- Python 3.7 新增的特性
- 在类的定义前面使用 `@dataclass`
- 自动具备了 `__str__`, `__repr__`, `__eq__`, `__hash__` 等方法
- 相比 `namedtuple`, `dataclass` 的数据域可以写，可以添加新的方法

In [85]:
```python
from dataclasses import dataclass
from math import asin, cos, radians, sin, sqrt

@dataclass
class Position:
    name: str
    lon: float
    lat: float

    def distance_to(self, other):
        r = 6371  # Earth radius in kilometers
        lam_1, lam_2 = radians(self.lon), radians(other.lon)
        phi_1, phi_2 = radians(self.lat), radians(other.lat)
        h = (sin((phi_2 - phi_1) / 2)**2
             + cos(phi_1) * cos(phi_2) * sin((lam_2 - lam_1) / 2)**2)
        return 2 * r * asin(sqrt(h))
```

In [86]:
```python
pos = Position('Oslo', 10.8, 59.9)
print(pos)
vancouver = Position('Vancouver', -123.1, 49.3)
print(pos.distance_to(vancouver))
```

```
Position(name='Oslo', lon=10.8, lat=59.9)
7181.784122942117
```

In [78]:
```python
print(f'{pos.name} is at {pos.lat}° N, {pos.lon}° E')
```

```
Oslo is at 59.9° N, 10.8° E
```

In [80]:
```python
print(dir(pos))
```

```
['__annotations__', '__class__', '__dataclass_fields__', '__dataclass_params__', '__delattr__',
'__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt_
_', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__match_args__', '__modul
e__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof_
_', '__str__', '__subclasshook__', '__weakref__', 'lat', 'lon', 'name']
```

练习

experiment7 (../../Experiments/experiment7.md)