OOP & paradigms

Python Reference

Classes & objects

A class 类 is a blueprint. An object 对象 is one thing built from it (an instance 实例). `__init__` is the constructor 构造方法 that sets up each object; `self` is the object itself.

```
class Dog:
    def __init__(self, name):
        self.name = name            # an attribute
    def speak(self):
        return self.name + " says woof"

d = Dog("Rex")
print(d.speak())     # Rex says woof
```

- `name` is an attribute 属性 (data on the object); `speak` is a method 方法 (an action).

Inheritance, encapsulation & polymorphism

Inheritance 继承 lets a subclass 子类 reuse a superclass 父类. Use `super()` to call the parent; override 重写 a method to change it.

```
class Animal:
    def speak(self):
        return "some sound"

class Cat(Animal):
    def speak(self):                # override
        return "meow"

print(Cat().speak())   # meow
```

Encapsulation 封装 hides data behind methods; a leading underscore marks it private 私有.

```
class Account:
    def __init__(self):
        self._balance = 0           # private
    def deposit(self, n):
        self._balance += n
    def balance(self):
        return self._balance

a = Account()
a.deposit(50)
print(a.balance())     # 50
```

Polymorphism 多态 means one name, many behaviours —the right `speak` runs for each object.

```
class Cat:
    def speak(self):
        return "meow"

class Cow:
    def speak(self):
        return "moo"

for animal in [Cat(), Cow()]:
    print(animal.speak())   # meow, then moo
```

Programming paradigms

A paradigm 范式 is a style of writing programs. Procedural 过程式 code is a sequence of steps and functions. Object-oriented 面向对象 code groups data and methods into objects. Declarative 声明式 code says **what** you want, not how (a list comprehension or SQL).

```
def total(nums):            # procedural
    t = 0
    for n in nums:
        t += n
    return t
print(total([1, 2, 3]))     # 6

print(sum([1, 2, 3]))       # 6  (declarative: same result)
```