# 面向对象与编程范式

Python Reference

## 类与对象

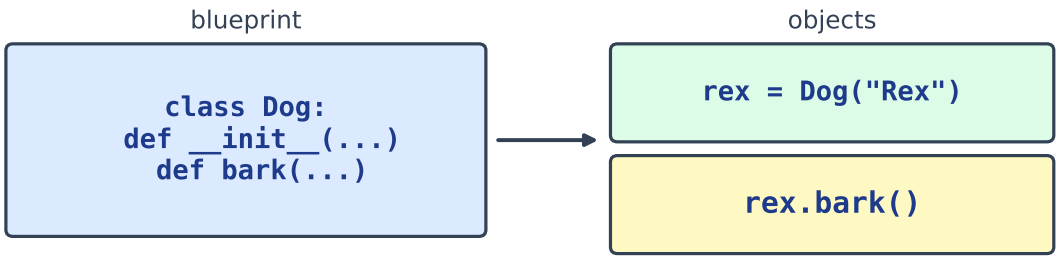
类 (class) 是一张蓝图。对象 (object) 是由它建造出来的一个具体东西 (一个实例 (instance))。`__init__` 是构造方法 (constructor), 用来设置每个对象;`self` 就是对象本身。

```
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` 是属性 (attribute)(对象上的数据);`speak` 是方法 (method)(一个动作)。

加上 `__str__`, 就能控制 `print(obj)` 显示什么:

```
class Dog:
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return f"Dog named {self.name}"

print(Dog("Rex"))    # Dog named Rex
```




Exam tip: class defines the type; calling ClassName(...) creates an object (instance)

*A class is a blueprint; calling it creates an object*

## 继承、封装与多态

继承 (inheritance) 让子类 (subclass) 复用父类 (superclass)。用 `super()` 调用父类; 重写 (override) 一个方法来改变它。

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

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

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

封装 (encapsulation) 把数据藏在方法后面; 名字前加一个下划线表示它是私有 (private) 的。

```
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
```

多态 (polymorphism) 的意思是同一个名字、多种行为——每个对象会运行各自正确的 `speak`。

```
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
```

## 编程范式

范式 (paradigm) 是写程序的风格。过程式 (procedural) 代码是一连串步骤和函数。面向对象 (object-oriented) 代码把数据和方法组织进对象。声明式 (declarative) 代码说的是你**要什么**, 而不是怎么做 (列表推导式或 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)
```

**常见错误**

- 每个方法的第一个参数都要是 `self`。
- `__init__` 用来初始化新对象, 创建对象时会自动运行。
- 同一个类的两个对象各有各的属性; 改一个不会影响另一个。