第12~13课 类与对象

一、课程结构导图

风变科技

注：▲为重点知识点。

二、知识点讲解

2.1 类

概念：类是定义新类型的程序结构，里面有数据属性，以及用于操作数据属性的方法[1]。

2.1.1 类的创建

用法1：使用class 关键字创建类，语法：**class 类名:** 。

示例1：

```
# 定义My_class类
class My_class:
```

```
    my_name = '千寻'        # 定义类属性

    def my_function1(self):  # 定义实例方法
        print('我可以打印我自己的名字哟')
```

注意：属性和方法是面向对象的叫法，在类对象中，变量可称为属性，函数可称为方法。

2.1.2 类及其属性、方法的调用

用法1：直接使用**实例名.属性**和**实例名.方法名** 即可对类属性和类方法进行调用。

示例1：

```
# 定义My_class类
class My_class:
    my_name = '千寻'                # 定义类属性

    def my_function1(self):         # 定义类方法（也叫实例方法）
        print('我可以打印我自己的名字哟')

example = My_class()        # 创建实例对象
print(example.my_name)      # 调用类属性
example.my_function1()      # 调用类方法
```

注意：第8代码中，实例对象example是属于My_class类，当实例example被创建出来，就可以调用类中的属性和方法。

2.1.3 特殊参数：self

使用场景：特殊参数self的作用在于，self会接收实例化过程中传入的数据，当实例对象创建后，实例便会代替 self，在代码中运行。

用法：使用**self.属性名** 声明为类的实例属性和使用**def 方法名(self)** 定义实例方法。

示例：

```
# 定义My_class类
class My_class:
    def my_function1(self):  # 创建实例方法
        self.my_skin = 'yellow' # 创建实例属性
        print(self.my_skin)    # 打印实例属性
example = My_class()        # 创建实例对象
example.my_function1()    # 调用实例方法
```

注意：

1. self 是类的实例，使用self 的属性和方法都是属于实例所有。self 是一个Python编程的习惯用语，使用其他的名词也可以，但是不建议。
2. self 放在方法后面的括号中，作为首个参数。

2.1.4 特殊方法：初始化方法

使用场景：初始化方法的作用在于，当每个实例对象创建时，该方法内的代码无须调用就会自动运行。

用法：定义初始化方法的格式是def __init__(self)，是由init加左右两边的【双】下划线组成。

示例：

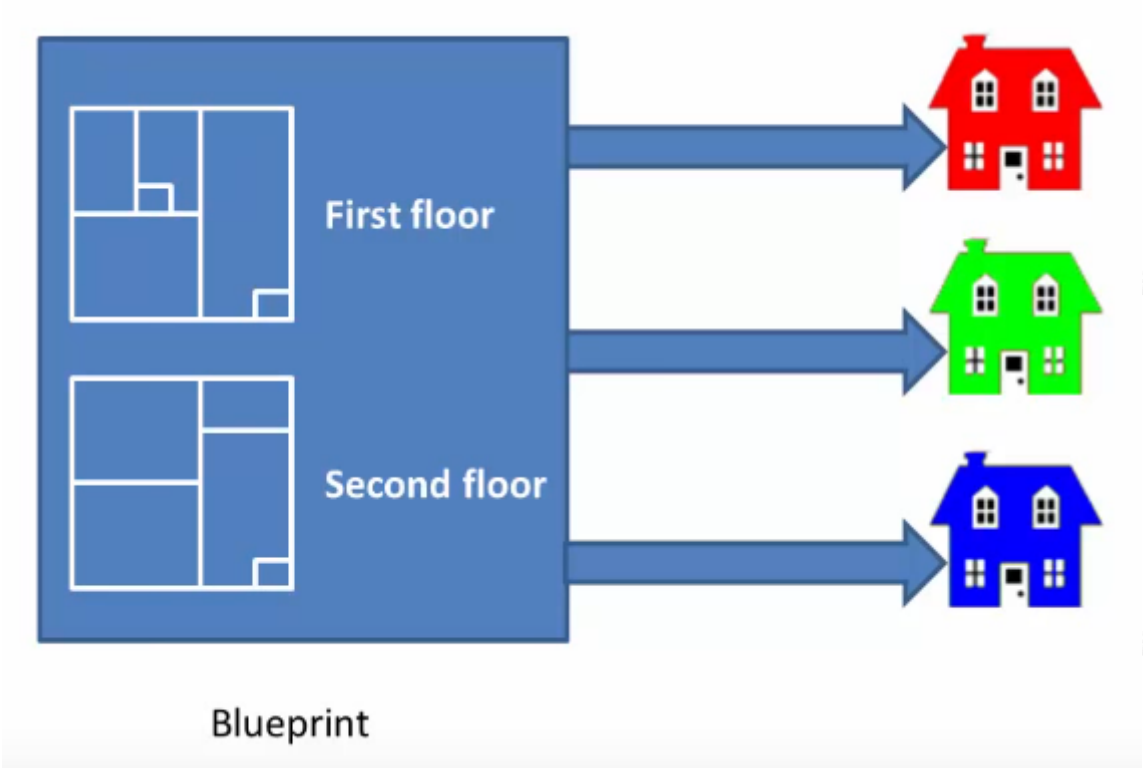
```
# 定义My_class类
class My_class:
    def __init__(self):      # 构造初始化函数
        self.my_nose = 'high' # 创建实例属性
        print(self.my_nose)     # 打印实例属性my_nose
example = My_class()    # 创建实例对象并调用初始化函数
```

注意：实例调用类时，__init__() 方法便会执行，而一般的方法需要再调用才会执行。另外，该方法只返回None ，不能返回其他类型。

2.1.5 类与对象的关系

下图，左边是一个“施工图纸”，右边是通过该图纸造出来的房子。同一个“施工图纸”，可以造出不同颜色的房子。类就好比“施工图纸”，而实例对象是造出来的房子。

类是对象的模板，描述不同对象的共同特征或行为，而对象是类的具体实例，这些实例都拥有类的所有特征和行为（即属性和方法）。



2.2 继承与重写

2.2.1 继承

概念：继承是让子类具有父类的属性和方法。类的继承能方便类的复用，不用将父类中的属性方法重新写一遍。

用法1：定义子类时，在子类名字后加上括号，并在括号中写入父类的名字。继承之后，便可以在子类中调用父类的类方法、类属性和实例方法、实例属性。

示例1：

```
# 定义My_class类
class My_class:
    my_name = '千寻'

    def __init__(self):
        self.my_nose = 'high'

    def my_function1(self):
        self.my_skin = 'yellow'
        print(self.my_skin)


# 继承My_class类
class Son_class(My_class):  # Son_class继承My_class类，获得父类的所有属性和方法
    pass

print(Son_class.my_name)    # 打印千寻
son_class = Son_class()     # 创建Son_class类的实例对象
son_class.my_function1()    # 打印yellow
print(son_class.my_nose)    # 打印high
```

注意：调用my_function1() 方法中的实例属性my_skin 之前，必须先调用该方法才可以！

用法2：多层继承，即一个类继承另一个类，另一个类继承其他类。

示例2：

```
# 定义My_class类
class My_class:
    my_name = '千寻'
    my_skin = 'white'

class S1_class(My_class):
    my_skin = 'yellow'      # 重写my_skin 属性

class S2_class(S1_class):
    pass
```

```
11
12 # 调用父类属性
13 print(S2_class.my_name) # 打印千寻
14 print(S2_class.my_skin) # 打印yellow
```

风变科技 风变科技

用法3：多重继承，即一个类同时继承多个类。

示例3：

```
1 # 定义My_class类
2 class My_class:
3     my_name = '千寻'
4
5 class S1_class(My_class):
6     my_skin = 'yellow'
7
8 class Her_class():
9     my_name = '酱酱'
10
11 class S2_class(S1_class, Her_class):    # 就近原则，优先在S1_class 类中查找
12     pass
13
14 # 调用父类属性
15 print(S2_class.my_name) # 打印千寻
16 print(S2_class.my_skin) # 打印yellow
```

风变科技 风变科技

用法4：继承并新增属性、方法。

示例4：

```
1 # 定义My_class类
2 class My_class:
3
4     def my_function1(self):
5         self.my_skin = 'yellow'
6         print(self.my_skin)
```

```
7
8  # 继承My_class类并新增实例方法
9  class Son_class(My_class):
10
11     # 新增实例方法
12     def my_function2(self):              # 新增方法
13         self.my_hair = 'golden'          # 新增属性
14         print(self.my_hair)
15
16 son_class = My_class()
17 son_class.my_function2()    # 打印golden
18 son_class.my_function1()    # 打印my_skin
```

注释：第12~14行是新增的实例方法，格式同一开始创建类的实例方法。

2.2.2 重写

概念：重写，顾名思义，就是重新写，是在继承父类的基础上对继承的属性或方法进行改写。。

用法1：全部重写一遍继承的属性、方法。

示例1：

```
1  # 定义My_class类
2  class My_class:
3      my_name = '千寻'
4
5      def __init__(self):
6          self.my_nose = 'high'
7          print(self.my_nose)
8
9      def my_function1(self):
10         self.my_skin = 'yellow'
11         print(self.my_skin)
12
13 # 继承My_class类并重写
14 class Son_class(My_class):
15     my_name = '酱酱'
```

```
16
17      def __init__(self):
18          self.my_nose = 'cute'
19
20      def my_function1(self):
21          self.my_skin = 'white'
22          print(self.my_skin)
23
24  # 调用Son_class类
25  print(Son_class.my_name)    # 打印酱酱
26
27  son_class = Son_class()     # 创建Son_class类的实例对象
28  print(son_class.my_nose)    # 打印cute
29  son_class.my_function1()    # 打印white
```

代码解析：在子类Son_class中，对父类的属性和方法都进行重写，重写之后再调用类的属性和方法，此时会优先在子类中查找相关的属性和方法，如果子类没有则在父类中查找。
（注：该例子把父类的所有属性和方法都改写了，此时父类的属性方法没有作用，一般情况下不会做如此大的改写，这只是为了说明可以进行此改写方法。）

用法2：修改部分继承的属性、方法。

示例2：

```
1  # 定义My_class类
2  class My_class:
3
4      def my_function1(self):
5          self.my_skin = 'yellow'
6          self.my_nose = 'high'
7
8  # 继承My_class类并重写
9  class Son_class(My_class):
10
11     def my_function1(self):
12         My_class.my_function1(self)      # 调用实例方法my_function1
13         self.my_skin = 'white'           # 修改实例属性my_skin的值
14         print(self.my_skin,self.my_nose)# 新增打印代码
15
16 son_class = Son_class()
```

```
17 son_class.my_function1()    # 打印white high
```

代码解析：第13行修改my_skin变量的值，然后第14行打印的my_skin变量会优先在子类的实例中进行搜索，故调用之后，打印修改后的值为white ，而my_nose变量 由于子类的实例和子类没有该属性，故打印父类中第6行的high。

疑难点：重写方法之后，子类的方法会覆盖父类的方法，在子类调用该方法时，只是调用到子类的方法，而不会调用父类对应的方法。第17行调用父类的my_function1() 方法，执行之后my_skin 和my_nose 两个实例属性才可以在子类的my_function1() 方法使用。

用法3：通过参数来修改属性的值。

示例3：

```
# 定义My_class类
class My_class:
    def my_function1(self, my_skin, my_nose):
        self.my_skin = my_skin
        self.my_nose = my_nose

# 继承My_class类并重写

# 代码一：没有默认值
class Son1_class(My_class):
    def my_function1(self, my_skin, my_nose):
        My_class.my_function1(self, my_skin, my_nose)
        print(self.my_skin,self.my_nose)

# 代码二：子类定义方法添加默认值
class Son2_class(My_class):
    def my_function1(self, my_skin='black', my_nose='high'):
        My_class.my_function1(self, my_skin, my_nose)
        print(self.my_skin,self.my_nose)

# 代码三：调用父类方法时添加默认值
class Son3_class(My_class):
    def my_function1(self, my_skin, my_nose):
        My_class.my_function1(self, my_skin='black', my_nose='high')
        print(self.my_skin,self.my_nose)
```

```
26
27 # 实例化调用
28 son1_class = Son1_class()
29 son1_class.my_function1('white', 'cute')    # 打印white cute
30 son2_class = Son2_class()
31 son2_class.my_function1('white', 'cute')    # 打印white cute
32 son3_class = Son3_class()
33 son3_class.my_function1('white', 'cute')    # 打印black high
```

代码解析：给父类参数加上两个参数。
代码一：定义子类Son1_class ， my_function1() 方法的参数没有默认值，调用父类的方法也没有默认值，最终打印的值都是实例调用时传入的值。

代码二：定义子类Son2_class ， my_function1() 方法的参数添加默认值，调用父类的方法也没有默认值，最终打印的值都是实例调用时传入的值；如果调用时没有传入值，则打印的值是添加的默认值。

代码三：定义子类Son3_class ， my_function1() 方法的参数没有默认值，**调用父类的方法添加默认值，最终打印的值都是添加的默认值，不管调用时传入什么值。**这是因为我们调用子类my_function1() 方法时，方法内部的代码是一行行往下执行，不管my_skin 和my_nose 前面赋什么值，都会在调用父类的my_function1() 方法中的两个默认值覆盖，所以最终的结果都是这两个默认值。

既然这两个没起作用，那能不能把调用时的传入值删去呢？因为在子类定义中的my_function1() 方法中有这两个参数，所以调用方法时，不能删除传入值。如果把方法中的这两个参数去掉，此时创建实例的时候就可以不用传入参数，代码如下：

```
1 # 代码三修改如下
2 class Son3_class(My_class):
3     def my_function1(self):
4         My_class.my_function1(self, my_skin='black', my_nose='high')
5         print(self.my_skin,self.my_nose)
6
7 son3_class = Son3_class()
8 son3_class.my_function1()   # 打印black high
```

注意：原代码三的代码中，添加了默认值之后， my_function1(self, my_skin, my_nose) 和 My_class.my_function1(self, my_skin='black', my_nose='high') 中的my_skin 和my_nose 没有任何关系。

我们可以简化为以下代码来辅助理解，调用函数是使用的是第8行括号中给两个参数赋的值，而跟外面的my_skin 和my_nose 变量无关。当不给参数赋值时，调用函数则会收到第6~7行的影响，如下代码中的第9行。

```
1 def my_function1(my_skin, my_nose):
2     my_skin = my_skin
3     my_nose = my_nose
4     print(my_skin,my_nose)
5 
6 my_skin = 'white'
7 my_nose = 'cute'
8 my_function1(my_skin='black', my_nose='high')   # 打印black high
9 my_function1(my_skin, my_nose)                  # 打印white cute
```

三、巩固练习

1. （单项选择）运行以下代码，结果是（ ）？

```
1 class Dog:
2     def character(self):
3         print('这个是一个狗类。')
4 
5 class Son_dog(Dog):
6     pass
7 
8 Son_dog()
```

A、pass　B、这个是一个狗类。　C、报错　D、无运行结果

2. （单项选择）运行以下代码，结果是（ ）？

```
1 class Dog:
2     def character(self):
3         action = '跑'
```

```
4
5 class Son_dog(Dog):
6     pass
7
8 son_dog = Son_dog()
9 son_dog.character()
10 son_dog.action
```

A、pass　B、跑　C、报错　D、无运行结果

3. （单项选择）运行以下代码，结果是（　）？

```
1 class Toy:
2     def describe(self, size,color):
3         self.size = size
4         self.color = color
5
6 class My_toy(Toy):
7     def describe(self,color,price,size='小号'):
8         Toy.describe(self, size,color='白色')
9         self.price = price
10        print('我花了%s元买了一个%s的%s玩具'% (self.price,self.color,self.size))
11
12 my_toy = My_toy()
13 my_toy.describe('米色', '10', '中号')
```

A、我花了10元买了一个白色的小号玩具
B、我花了10元买了一个白色的中号玩具
C、我花了10元买了一个米色的小号玩具
D、我花了10元买了一个米色的中号玩具

参考文献：

1. Luciano Ramalho著，安道 吴珂译，《Fluent Python》（流畅的Python），第1093页。