

第 12-13 关 类与对象-我有一个“机器人”

课程目标

了解何为对象
区分变量与类属性，函数与类方法
掌握类的实例化及初始化函数
熟练继承与子类定制

课程难点

初始化函数的优雅改写
多层继承与多重继承的优先级

课程重要内容重现

一、类

1、类的创建

- 1) 类名首字母要大写
- 2) 为区别普通函数，在类中赋值的变量叫做类属性(attribute)，类中定义的函数叫做类方法(method)
- 3) 实例方法是指类中参数带 **self** 的函数，是类方法的一种形式，也是最常见的用法。
- 4) 类的创建：**class** 语句
- 5) 类属性的创建：赋值语句
- 6) 实例方法的创建：**def method1(self):**

2、实例化

刚被创造出来的实例与类一模一样。

格式：**实例名=类名()**

3、调用类的属性和方法

- 1) 当实例一被创建出来，就可以调用类中的属性和方法。参数 **self** 在定义时不能丢，在调用时要忽略。

格式：**实例名.属性** 和 **实例名.方法**

```
class Computer:
```

```
screen = True
```

```
def start(self):
```

```
print('电脑正在开机中.....')
```

```
my_computer = Computer()实例化
```

```
print(my_computer.screen)调用类属性
```

```
my_computer.start()调用类方法
```

2) 类的内部调用类属性

实例名 **person** 会像参数一样传给 **self**，替换掉 **self**，第六行的 **self.name** 等价于 **person.name**，然后 **person.name** 相当于调用了第 2 行的类属性 **name**。也就是说，**self** 起到提示传参位置的作用，当在类的方法内部想调用类属性或其他方法时，就要采用 **self.属性名** 或 **self.方法名** 的格式。（相当于改变了作用域）

```

class Chinese:
    name = '吴枫'  类属性 name
    def say(self):
        print(self.name + '是中国人')

person = Chinese()  创建 Chinese 的实例 person
person.say()  调用实例方法

```

二、初始化函数

在初始化方法内部完成类属性的创建，为类属性设置初始值，这样类中的其他方法就能直接、随时调用。不需要再调用__init__()。

```

class chinese:
    def __init__(self, name, birth, region)
        self.name = name    self.name = '吴枫'
        self.birth = birth   self.birth = '广东'
        self.region = region self.region = '深圳'
    def born(self):
        print(self.name + '出生在' + self.birth)
    def live(self):
        print(self.name + '居住在' + self.region)

person = Chinese('吴枫','广东','深圳')  传入初始化方法的参数
person.born()
person.live()

```

三、继承

子类也可以在继承的基础上进行个性化的定制，包括：（1）创建新属性、新方法；（2）修改继承到的属性或方法。

格式：class A(B):，表示 A 继承了类 B。

子类继承的属性和方法，也会传递给子类创建的实例。很多类在创建时也不带括号，如 class A: 实际上，class A:在运行时相当于 class A(object):。而 object，是所有类的父类，我们将其称为根类。

父类创建的实例不属于子类。

子类创建的实例同时也属于父类。

类创建的实例都属于根类。

1) 多层继承

```

class B(A):
class C(B):

```

多层继承（纵向）时，子类创建的实例可调用所有层级父类的属性和方法。

2) 多重继承

```

class A(B,C,D):

```

多重继承（横向）时，根据与子类的相关顺序从左往右排，A 是 B,C,D 三个父类的子类，但是在调用类方法和类属性时，优先从 B 中找，找不到再去 C（就近原则）。

```

class C0:

```

```
name = 'C0'
class C2(C0):
    num = 2
class C1:
    num = 1
class C3:
    name = 'C3'
class C4(C1,C2,C3):
    pass
ins = C4()
print(ins.name) 一直 打印出 C0，即会沿着 C2 父类一直往其父类里找，直到完全找不到再去 C3 找。
print(ins.num) 打印出 1
```

四、定制

新增：在子类下新建属性或方法。

修改：在子类中对继承的父类代码的修改（并不会改变父类内部代码）

五、其他

isinstance(实例，类)可判断该实例是否属于某个类。

如果方法名形式是左右带双下划线的，那么就属于特殊方法（如__init__，初始化方法），有着特殊的功能，如实例化时会直接调用。（定义了初始化函数之后还是可以定义多个这样的特殊方法的）

向上取整函数 ceil()，向下取整 int()，四舍五入 round()。

第 12 关课后习题讲解

课程要求是：新建一个方法，让实例只要调用一个方法，就能打印出两个信息。

这里我们会用到初始化函数，还有在函数内调用类属性

我们先来看直接调用类的实现方式

```
class Chinese:

    def __init__(self,hometown,region):
        self.hometown = hometown
        self.region = region
        print('程序持续更新中.....')

    def born(self):
        print('我生在%s。'%(self.hometown))

    def live(self):
        print('我在%s。'%(self.region))

Chinese('beijing','shanghai')
```

这里我们定义了一个类，类下包含一个初始化函数，两个普通方法

在初始化函数中，我们会给它传递两个参数，并将这两个参数赋值给 `self.hometown` 和 `self.region`

这里有同学的疑问是，为什么要把参数重新赋值呢？为什么不可以直接使用？

因为 `hometown` 和 `region` 作为参数的时候，是只在该方法下有效的，而其他方法不能直接调用这个参数

如果想调用，就要用到重新赋值给一个变量，且这个变量需带有 `self.` 的前缀，这样才能实现在不同的类方法下调用该变量（也叫属性）

```
class Chinese:

    def __init__(self,hometown,region):
        self.hometown = hometown
        self.region = region
        print('程序持续更新中.....')

    def born(self):
        print('我生在%s。'%(self.hometown))

    def live(self):
        print('我在%s。'%(self.region))
Chinese('beijing','shanghai')
```

所以我们看到在后面的两个方法中，分别对两个属性进行了调用

看上图的代码，最后的调用中，只调用了 `Chinese()` 这个类，这样的运行结果，其实没有实现题目的要求

因为单调用 `Chinese` 类的时候，只会执行初始化函数，而不会再进行其他的操作

既然题目说了，要新建一个方法，且只调用一个方法，就能打印出两个信息。那么我们就需要用 `main()` 来把打印两个信息的方法再封装起来

完整代码如下

```
# 新建一个方法，让实例只要调用一个方法，就能打印出两个信息。
# 代码完成后，请运行一下，验证是否成功。
class Chinese:

    def __init__(self,hometown,region):
        self.hometown = hometown
        self.region = region
        print('程序持续更新中.....')

    def born(self):
        print('我生在%s。'%(self.hometown))

    def live(self):
        print('我在%s。'%(self.region))

    def main(self):
        self.born()
        self.live()

zhouyou=Chinese('广东','深圳')
zhouyou.main()
```

在这里的 `main()` 方法下，按顺序调用 `self.born()` 和 `self.live()` 两个方法

所以当我们最后调用的时候，只需要调用 `main()` 就可以同时执行另外的两个方法了，运行结果就是根据调用时手动传递的参数打印出两个语句

第二个练习，简单讲一下，其实跟第一个练习相似，只不过在某个方法中还加入了一个循环，但是逻辑上理解是一样的

```
class Robot:
    def __init__(self):

        self.name = input('我现在刚诞生，还没有名字，帮我起一个吧。')
        self.master = input('对了，我要怎么称呼你呢? ')

        print('你好%s，我叫%s。很开心，遇见你~'%(self.master,self.name))

    def say_wish(self):

        wish = input('告诉一个你的愿望吧: ')

        print(self.master+'的愿望是: ')
        # 这里也可以用字符串的格式化，不过，用循环语句的话，之后改复述次数会方便些。

        for i in range(3):
            print(wish)

robot1 = Robot()
robot1.say_wish()
```

首先执行 `robot1 = Robot()` 进行实例化，实例化的同时也执行了函数的初始化函数

```
73
74 class Robot:
75     def __init__(self):
76
77         self.name = input('我现在刚诞生，还没有名字，帮我起一个吧。')
78         self.master = input('对了，我要怎么称呼你呢? ')
79
80         print('你好%s，我叫%s。很开心，遇见你~'%(self.master,self.name))
81
82     def say_wish(self):
83
84         wish = input('告诉一个你的愿望吧: ')
85
86         print(self.master+'的愿望是: ')
87         # 这里也可以用字符串的格式化，不过，用循环语句的话，之后改复述次数会方便些。
88
89         for i in range(3):
90             print(wish)
91
92 robot1 = Robot()
93 # robot1.say_wish()
```

问题 2 输出 调试控制台 终端

Windows PowerShell
版权所有 (c) Microsoft Corporation。保留所有权利。

PS D:\PythonClass> & C:/Users/42949/AppData/Local/Programs/Python/Python37/python.exe d:\Py
我现在刚诞生，还没有名字，帮我起一个吧。xixi
对了，我要怎么称呼你呢? haha
你好haha，我叫xixi。很开心，遇见你~
PS D:\PythonClass>

看我把最后一句注释掉，只执行 `robot1 = Robot()` 的时候，只执行初始化函数的部分

当我把最后一句加上时，实例 robot1 调用方法 say_wish()

```
73
74 class Robot:
75     def __init__(self):
76
77         self.name = input('我现在刚诞生，还没有名字，帮我起一个吧。')
78         self.master = input('对了，我要怎么称呼你呢? ')
79
80         print('你好%s，我叫%s。很开心，遇见你~'%(self.master,self.name))
81
82     def say_wish(self):
83
84         wish = input('告诉一个你的愿望吧: ')
85
86         print(self.master+'的愿望是: ')
87         # 这里也可以用字符串的格式化，不过，用循环语句的话，之后改复述次数会方便些。
88
89         for i in range(3):
90             print(wish)
91
92 robot1 = Robot()
93 robot1.say_wish()
```

问题 2 输出 调试控制台 终端

Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

PS D:\PythonClass> & C:/Users/42949/AppData/Local/Programs/Python/Python37/python.exe d:/Py
我现在刚诞生，还没有名字，帮我起一个吧。xixi
对了，我要怎么称呼你呢? haha
你好haha，我叫xixi。很开心，遇见你~
告诉一个你的愿望吧：我要学会py
haha的愿望是：
我要学会py
我要学会py
我要学会py
PS D:\PythonClass>

在 say_wish() 下，先用 input 输入并赋值给 wish

这里有同学问，为什么这里的 wish 不用加 self. 哇？

是因为 wish 这个变量，它只在该方法下进行使用，你看它不需要在其他类方法被调用，对吧。所以是可以不加 self. 的
所以最后通过一个循环，打印三遍，完成题目要求

第 13 关课后习题讲解

13 关我们学到的主要知识就是 【类的继承】

考虑到同学们对继承逻辑的理解会稍微比较绕，所以 13 关的习题也设置得比较简单，目的也是让同学们以最直观的方式进行继承的操作

而稍微复杂的操作，我们会在 14 关的实操课中去学习

来看习题，第一个目标和要求



练习介绍

练习目标：

在这个作业，我们会通过对类属性这个切入点，温习类的继承和定制。

练习要求：

每个人都有好几个不同的身份，且不同身份都附带一些特定的特征（属性）和行为（方法）。

例如，有这样一群人：在学校时被归在老师，脸是严肃的；亲子关系中（parenthood）则被归到父亲，脸是甜蜜的。

下面，我们就以这群人为例，探索类属性在类的继承和定制中的传递和改变。

这个题目在练习中已经给出了代码，但是同学们也要理解

```
'''版本1.0'''
# 直接运行代码即可。
class Teacher:
    face = 'serious'
    job = 'teacher'

class Father:
    face = 'sweet'
    parenthood = 'dad'

time1 = Teacher() # 在time1这个时刻，那个男人角色是老师。
time2 = Father()  # 在time2这个时刻，那个男人角色是父亲。
print(time1.face) # 时刻不同，角色不同，脸也不同。
print(time2.face)
```


首先这里 Teacher 类和 Father 类个字有两个属性，其中一个同名属性 face

我们用 time1 和 time2 分别对两个类进行实例化，再调用各自的 face 属性

```
time1 = Teacher()
time2 = Father()
print(time1.face)
print(time2.face)
```

记得在创建实例的时候，格式是 实例名 = 类（）

括号不能丢

然后用 实例名.属性 的方式调用了类的属性，并用 print 进行打印

第二部分，课程要求是

✓ 子类的继承和定制

请你创建两个子类，同时继承已有的两个类（注：多重继承）；

然后，在其中选个子类进行定制：将 face 属性的值改变

为'gentle'；

再者，创建实例 time3、time4，以调用子类的 face 属性。

代码如下

（属性）和行为（方法）。

例如，有这样一群人：在学校时被归入老师，脸是严肃的；亲子关系中（parenthood）则被归入父亲，脸是甜蜜的。

下面，我们就以这群人为例，探索类属性在类的继承和定制中的传递和改变。

✓ 创建两个类：老师和父亲

首先，我们需要创建两个类，并为它们添加属性。

✓ 子类的继承和定制

请你创建两个子类，同时继承已有的两个类（注：多重继承）；

然后，在其中选个子类进行定制：将 face 属性的值改变

为'gentle'；

再者，创建实例 time3、time4，以调用子类的 face 属性。

✓ 参考代码

```
1 class Teacher:
2     face = 'serious'
3     job = 'teacher'
4
5
6 class Father:
7     face = 'sweet'
8     parenthood = 'dad'
9
10
11 class TeacherMore(Teacher, Father):
12     pass
13
14 class FatherMore(Father, Teacher):
15     face = 'gentle'
16
17 time3 = TeacherMore()
18 time4 = FatherMore()
19 print(time3.face)
20 print(time4.face)
21
```

我们先定义了

```
class TeacherMore(Teacher, Father):  
    pass
```

这里按照多重继承的逻辑，会优先继承 **Teacher** 这个父类，在这个类下，**pass** 的意思是完全继承，不做修改

再者，我们定义了

```
class FatherMore(Father, Teacher):  
    face = 'gentle'
```

这里按照多重继承的逻辑，会优先继承 **Father** 这个父类，在这个类下，对属性 **face** 进行了重新赋值，修改为'gentle'

我们用 **time3** 和 **time4** 分别对两个子类进行实例化，再调用各自的 **face** 属性

time3 是子类 **TeacherMore()** 的实例，因为没有重新赋值 **face**，所以优先继承了 **Teacher** 这个父类的 **face** 属性，打印为 **serious**

time4 是子类 **FatherMore()** 的实例，因为重新赋值了 **face**，所以此时打印重新赋值后的 **face** 属性，打印为 **gentle**

第二个进阶练习：

```
'''  
练习目标：  
这个练习，主要是训练你对“子类的继承”的理解和运用。  
  
练习要求：  
练习会先提供一个类，用以记录学生学习 Python 的投入时间和有效时间。  
需要你创建一个子类，为某一类学生提供定制化的记录方案。'''  
  
# 请先读懂代码，再运行。  
class Student:  
    # 初始化函数，为每个实例创建4个参数（其中后3个参数有默认值）  
    def __init__(self, name, job=None, time=0.00, time_effective=0.00):  
        self.name = name  
        self.job = job  
        self.time = time  
        self.time_effective = time_effective  
  
    def count_time(self, hour, rate):  
        self.time += hour  
        self.time_effective += hour * rate # 有效时间=投入时间×学习效率  
  
student1 = Student('韩梅梅')  
print(student1.job)  
  
student1.count_time(10, 0.8) # 学习效率为0.8  
print(student1.time_effective)
```

先看初级版本，在类下两个方法，一个初始化方法，一个普通方法

初始化的作用是一般会用来执行一些默认的操作，比如把参数赋值，这样当调用类的时候，则默认执行初始化函数，不用手动进行调用

而普通方法则是实现特定功能的调用，这里的 `count_time()` 则用来计算效率

同学会问，为什么在 `init` 里，要重新对参数赋值呢，`self.` 的作用是什么

这里对参数重新赋值到一个变量，目的是为了在其他的类方法进行调用，只有加了 `self.` 前缀的才能进行调用哦

如果没有加 `self.`，则只能在该方法下使用

```
# 请先读懂代码，再运行。
class Student:
    # 初始化函数，为每个实例创建4个参数（其中后3个参数有默认值）
    def __init__(self, name, job=None, time=0.00, time_effective=0.00):
        self.name = name
        self.job = job
        self.time = time
        self.time_effective = time_effective

    def count_time(self, hour, rate):
        self.time += hour
        self.time_effective += hour * rate # 有效时间=投入时间×学习效率

student1 = Student('韩梅梅')
print(student1.job)
```

看第一个实例，调用了类 `Student`，并传入了一个参数给 `name`

在 `init` 中，因为其他的参数都是默认参数，所以调用的时候只传参给 `name` 即可

```

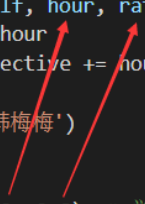
# 请先读懂代码，再运行。
class Student:
    # 初始化函数，为每个实例创建4个参数（其中后3个参数有默认值）
    def __init__(self, name, job=None, time=0.00, time_effective=0.00):
        self.name = name
        self.job = job
        self.time = time
        self.time_effective = time_effective

    def count_time(self, hour, rate):
        self.time += hour
        self.time_effective += hour * rate # 有效时间=投入时间x学习效率

student1 = Student('韩梅梅')
print(student1.job)

student1.count_time(10, 0.8) # 学习效率为0.8
print(student1.time_effective)

```



第二步，实例 student1 调用类方法 count_time()并传入两个参数，并打印属性 self.time_effective，具体逻辑如上图

```

def count_time(self, hour, rate):
    self.time += hour
    self.time_effective += hour * rate # 有效时间=投入时间x学习效率

student1 = Student('韩梅梅')
print(student1.job)

student1.count_time(10, 0.8) # 学习效率为0.8
print(student1.time_effective)

```

在实例调用某个属性时候，格式是 实例名.属性名，此时不用加 self.

接下来是升级版

```

class Student:
    def __init__(self, name, job=None, time=0.00, time_effective=0.00):
        self.name = name
        self.job = job
        self.time = time
        self.time_effective = time_effective

    def count_time(self, hour, rate):
        self.time += hour
        self.time_effective += hour * rate

class Programmer(Student):
    def __init__(self, name, job='programmer', time=0.00, time_effective=0.00):
        Student.__init__(self, name, job, time, time_effective)

    def count_time(self, hour, rate=1):
        Student.count_time(self, hour, rate)

student1 = Student('韩梅梅')
student2 = Programmer('李雷')

print(student1.job)
print(student2.job)

student1.count_time(10, 0.8)
student2.count_time(10)

print(student1.time_effective)
print(student2.time_effective)

```

子类 Programmer()继承了父类 Student(), 并进行了类方法的重写(定制)

```

class Student:
    def __init__(self, name, job=None, time=0.00, time_effective=0.00):
        self.name = name
        self.job = job
        self.time = time
        self.time_effective = time_effective

    def count_time(self, hour, rate):
        self.time += hour
        self.time_effective += hour * rate

class Programmer(Student):
    def __init__(self, name, job='programmer', time=0.00, time_effective=0.00):
        Student.__init__(self, name, job, time, time_effective)

    def count_time(self, hour, rate=1):
        Student.count_time(self, hour, rate)

```

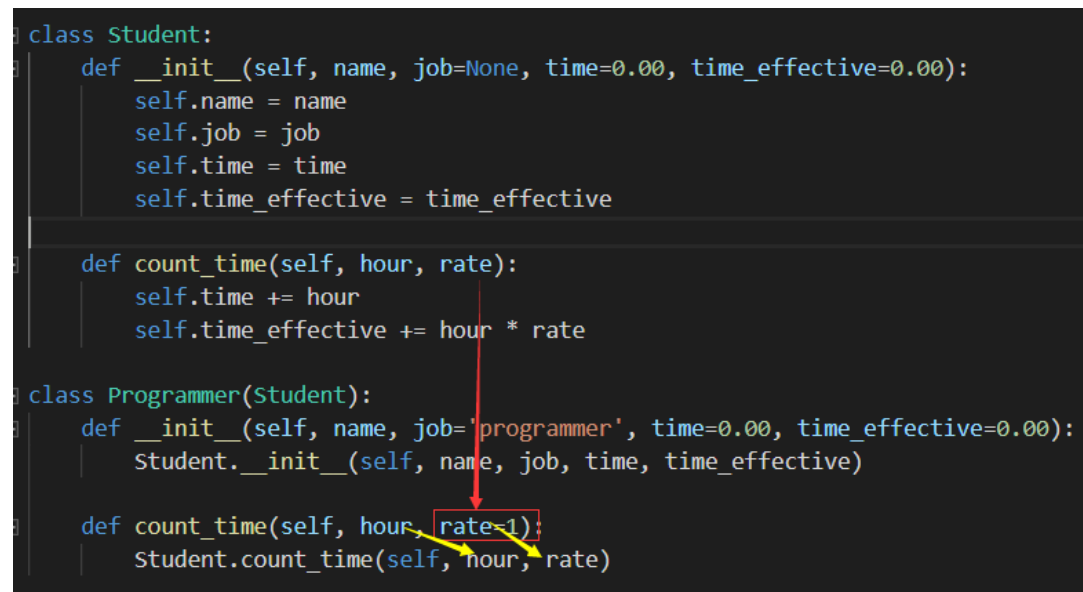
子类重写初始化函数，修改了原参数中 job 的默认值，并在在函数下调用了父类的初始化函数，参数传递如上图

```
class Student:
    def __init__(self, name, job=None, time=0.00, time_effective=0.00):
        self.name = name
        self.job = job
        self.time = time
        self.time_effective = time_effective

    def count_time(self, hour, rate):
        self.time += hour
        self.time_effective += hour * rate

class Programmer(Student):
    def __init__(self, name, job='programmer', time=0.00, time_effective=0.00):
        Student.__init__(self, name, job, time, time_effective)

    def count_time(self, hour, rate=1):
        Student.count_time(self, hour, rate)
```

A diagram illustrating parameter passing in Python. A red arrow points from the 'job' parameter in the Programmer.__init__ method call to the 'job' parameter in the Student.__init__ method definition. A yellow arrow points from the 'rate' parameter in the Programmer.count_time method call to the 'rate' parameter in the Student.count_time method definition.

子类重写 count_time 方法，修改了原参数中 rate 的默认值，参数传递如上图

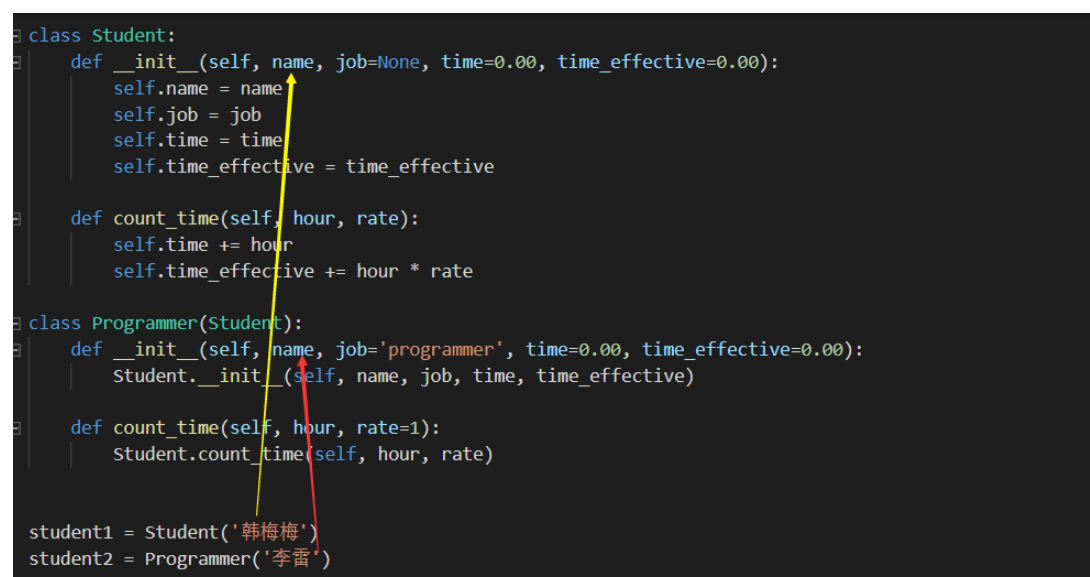
```
class Student:
    def __init__(self, name, job=None, time=0.00, time_effective=0.00):
        self.name = name
        self.job = job
        self.time = time
        self.time_effective = time_effective

    def count_time(self, hour, rate):
        self.time += hour
        self.time_effective += hour * rate

class Programmer(Student):
    def __init__(self, name, job='programmer', time=0.00, time_effective=0.00):
        Student.__init__(self, name, job, time, time_effective)

    def count_time(self, hour, rate=1):
        Student.count_time(self, hour, rate)

student1 = Student('韩梅梅')
student2 = Programmer('李雷')
```

A diagram illustrating parameter passing in Python. A yellow arrow points from the 'name' parameter in the Student.__init__ method definition to the 'name' parameter in the Programmer.__init__ method definition. A red arrow points from the 'rate' parameter in the Programmer.count_time method definition to the 'rate' parameter in the Student.count_time method definition.

创建实例 student1 和 student2，参数传递如上图

```
class Student:
    def __init__(self, name, job=None, time=0.00, time_effective=0.00):
        self.name = name
        self.job = job
        self.time = time
        self.time_effective = time_effective

    def count_time(self, hour, rate):
        self.time += hour
        self.time_effective += hour * rate

class Programmer(Student):
    def __init__(self, name, job='programmer', time=0.00, time_effective=0.00):
        Student.__init__(self, name, job, time, time_effective)

    def count_time(self, hour, rate=1):
        Student.count_time(self, hour, rate)

student1 = Student('韩梅梅')
student2 = Programmer('李雷')

# print(student1.job)
# print(student2.job)

student1.count_time(10, 0.8)
student2.count_time(10)

print(student1.time_effective)
print(student2.time_effective)
```

The diagram illustrates the parameter passing for the `count_time()` method calls. Red arrows show the call chain from `student1.count_time(10, 0.8)` to `Student.count_time`, and from `student2.count_time(10)` to `Programmer.count_time`, which then calls `Student.count_time`. A yellow arrow shows the call chain from `student2.count_time(10)` to `Student.count_time`.

实例 student1 和 student2 分别调用 count_time()方法，参数传递如上图