

class

#类的名字应该用大写字母

```
class Employee:
    #给一些定义
    """ define a class for all employee
    instances in a company """
    #没啥 内容就 写pass
    pass
#创建一个实体
variable_name = Employee()
```

Attributes

instance attributes 是每个实体都有自己的

```
# Provide attributes and assign values to the
instances
empl.firs tName = " Mar ia"
empl.l astName = " Ren a"
empl.b asi cSalary = 12000
empl.a llo wance = 5000
```

class attributes belong to the class itself, and are thus shared among all instances of the class

如果先声明i nstant attrib ute ,再声 明class attrib ute 。那 不会覆盖 原来的 i nstant attribute。但如果原来在 类里声 明了一 个类a ttr ibu te, 在后面 再修改 ,那原 来的值 确实会被覆盖

obj.__ dic t__ 返回【 实体】 和它的所有方法

class.__ d ict __ 返回【类】的所 有方法和属性

Methods

instant method是自定义的

```
class Employee:
    bon usP ercent = 0.2
    # Define an instance method that takes no
    arguments
    def calcul ate Bon us( self):
        return self.s alary * Employ ee.b on usP -
        ercent
```

如何更改bo nus Per cen t的默 认值? 用类的调用来改

```
Employ ee.b on usP erc ent=0.3
```

Constr uctor & Destructo r Method 是内置的建立 和销毁函数

Methods (cont)

> __init__, __del__, __str__

使用默认参数初始化实例

#这段代码有条件才执行, 如果不执行的话这个实例四个attribute 都不会有

```
def __init__ (self, first = None, last = None,
salary = None, allowances = None):
    if first!= None and last!= None and salary!=
None and allowa nces!= None:
        sel f.f irs tName = first
        sel f.l astName = last
        sel f.s alary = salary
        sel f.a llo wances = allowances
        pri nt( " Object initia lized with
supplied values ")
```

写入和加载obj

```
import _pickle as cpickle
class stock:
    def __init __ ( self, name, lots):
        sel f.name = name
        sel f.lots = lots
    def __str__ (s elf):
        return f'{sel f.n ame }=> {se lf.l ots}'
```

#写入

```
s1 = stock( " App le", 1500)
s2 = stock( " Goo gle ", 2000)
with open("s 1.o bj", " wb") as f:
    cpickl e.d ump (s1 / [s 1,s 2], f)
```

#加载

```
with open("s 1.o bj", " rb") as f:
    s1 = cpickl e.l oad(f)
    pri nt(s1)
with open("s s.o bj", " rb") as f:
    data = cpickl e.l oad(f)
    for each in data:
        pri nt( each)
```



self.variable和class.variable的区别

```
class employee:
    count=5
    def __init__(self, x):
        self.x=x
        self.count=self.count+1
        print("this method is executed")
        print(self.count)
        print(employee.count)

empl=employee("John")
empl.count+=1
print(empl.count)
print(employee.count)
```

结果是6, 5, 7, 5。self.variable 显示的是这个obj 自己当前的值, class.variable 显示的是他们共用的初始值

JSON data (json , dict和obj可以互相转化)

```
#把obj放进json再取出来
import json
from json import JSONEncoder
class stock(object):
    def __init__(self, name, lots):
        self.name = name
        self.lots = lots
    def __str__(self):
        return f'{self.name}=>{self.lots}'
class StockEncoder(JSONEncoder):
    def default(self, o):
        return o.__dict__
s1 = stock("Apple", 1500)
# encode Object it
s1Json = json.dumps(s1, cls=StockEncoder,
indent=4)
'''
json.loads() method can be used to parse a valid
JSON string and convert it into a Python Dictionary.
'''
```

JSON data (json , dict和obj可以互相转化) (cont)

```
> resultDict = json.loads(s1Json)
s1Obj = stock(**resultDict)
#json data写入/存储
with open("stockPrettyPrint.json", "w") as write_file:
    json.dump(s1, write_file, cls=StockEncoder, indent=4)
with open("stockPrettyPrint.json", "r") as read_file:
    s = json.load(read_file)
    PrettyJson = json.dumps(s, indent=4, separators=(',', ':'), sort_keys=True)
#Json, Dictionary, object
import json
from json import JSONEncoder
class stock(object):
    def __init__(self, name, lots):
        self.name = name
        self.lots = lots
    def __str__(self):
        return f'{self.name}=>{self.lots}'
class StockEncoder(JSONEncoder):
    def default(self, o):
        return o.__dict__
    def StockDecoder(obj):
        if '__type__' in obj and obj['__type__'] == 'stock':
            return stock(obj['name'], obj['lots'])
        return obj
s2Obj = json.loads('{"__type__": "stock", "lots":2000, "name": "Google"}', object_hook=StockDecoder)
with open("stockPrettyPrint.json", "w") as write_file:
    json.dump([s2Obj], write_file, cls=StockEncoder, indent=4)
with open("stockPrettyPrint.json", "r") as read_file:
    print("Read JSON file")
    s = json.load(read_file)
    for each in s:
        each = stock(**each)
        print(each)
```

JSON data (json , dict和obj是可以互转)

```
#把obj放进json再取出来
import json
from json import JSONEncoder
class stock( object):
    def __init__( self, name, lots):
        self.name = name
        self.lots = lots
    def __str__(self):
        return f'{self.name}=> {self.lots}'
class StockEncoder( JSONEncoder):
    def default(self, o):
        return o.__dict__
s1 = stock( " Apple", 1500)
# encode Object it
s1Json = json.dumps(s1, cls=StockEncoder,
indent=4)
'''
json.loads() method can be used to parse a valid
JSON string and convert it into a Python Dictionary.
'''
resultDict = json.loads(s1Json)
s1Obj = stock( **resultDict)
#json data写入/存储
with open("stockPrettyPrint.json", "w") as
write_file:
    json.dump(s1, write_file, cls=StockEncoder,
indent=4)
with open("stockPrettyPrint.json", "r") as
read_file:
    s = json.load(read_file)
    PrettyJson = json.dumps(s, indent=4, separator=(',', ': '), sort_keys=True)
#Json, Dictionary, object
import json
from json import JSONEncoder
class stock( object):
    def __init__( self, name, lots):
        self.name = name
        self.lots = lots
```

JSON data (json , dict和obj是可以互转) (cont)

```
> def __str__(self):
    return f'{self.name}=>{self.lots}'
class StockEncoder(JSONEncoder):
    def default(self, o):
        return o.__dict__
    def StockDecoder(obj):
        if '__type__' in obj and obj['__type__'] == 'stock':
            return stock(obj['name'], obj['lots'])
        return obj
s2Obj = json.loads('{"__type__": "stock", "lots":2000, "name": "Google"}', object_hook=StockDecoder)
with open("stockPrettyPrint.json", "w") as write_file:
    json.dump([s2Obj], write_file, cls=StockEncoder, indent=4)
with open("stockPrettyPrint.json", "r") as read_file:
    print("Read JSON file")
    s = json.load(read_file)
    for each in s:
        each = stock(**each)
    print(each)
```

