《人工智能与Python程序设计》— 面向对象编程（三）

人工智能与Python程序设计 教研组

人工智能与
Python程序设计

回顾

1. 类和实例
2. 继承和多态
3. 获取对象信息
4. 实例属性和类属性

类和实例

- 类和实例是OOP中最为重要的概念
  - 实例（对象）
    - 保存在计算机内存中的一个object
    - 对数据进行抽象和封装
    - 数据保存在**实例属性**里
  - 从求解问题的角度，类是实例的**抽象**
    - Michael，Kristen都是Student
      - 注意与Dog，Cat都是Animal区分
  - 从程序实现的角度，类是实例的**模板**
    - 通过__init__函数绑定实例共有的属性
    - 通过定义方法，定义实例共有的功能
      - 方法被保存在**类属性**中

```
In [15]: 1 animal

Out[15]: <__main__.Dog at 0x7fb6b106d9a0>
```

```python
class Animal(object):
    name = 'Animal'
    def __init__(self):
        print('This is an animal class.')

    def live(self):
        print("{}'s life:".format(self.name))
        self.eat()
        self.play()
        self.sleep()

    def play(self):
        print('Animal is playing')

    def eat(self):
        print('Animal is eating')

    def sleep(self):
        print('Animal is sleeping')
```

继承与多态

- 在OOP 中定义一个class 时，可以从某个现有的class继承，新的class 称为子类(subclass)，而被继承的class 称为基类、父类或超类(base class，super class)。
- 继承：一个派生类(derived class)继承基类的字段和方法，即子类获得了父类的全部功能。
- 举例：Dog类继承Animal类

[1]:
```
class Animal(object):
    def run(self):
        print('Animal is running')

class Dog(Animal):
    pass

class Cat(Animal):
    pass

animal = Dog()
animal.run()
```

```
Animal is running
```

继承与多态

- 多态：为不同数据类型的实体提供统一的**接口**。
  - 简单而言：相同的消息给予不同的对象会引发不同的动作

```
class Dog(Animal):
    def play(self):
        print('Dog like playing frisbee!')

class Husky(Dog):
    def play(self):
        print('Husky like playing sofa!')

animal1 = Husky()
animal1.play()
animal2 = Dog()
animal2.play()
```

```
This is an animal class.
Husky like playing sofa!
This is an animal class.
Dog like playing frisbee!
```

```
[4]: class Dog(Animal):

        def run(self):
            print('Dog is running...')

        def eat(self):
            print('Eating meat...')
```

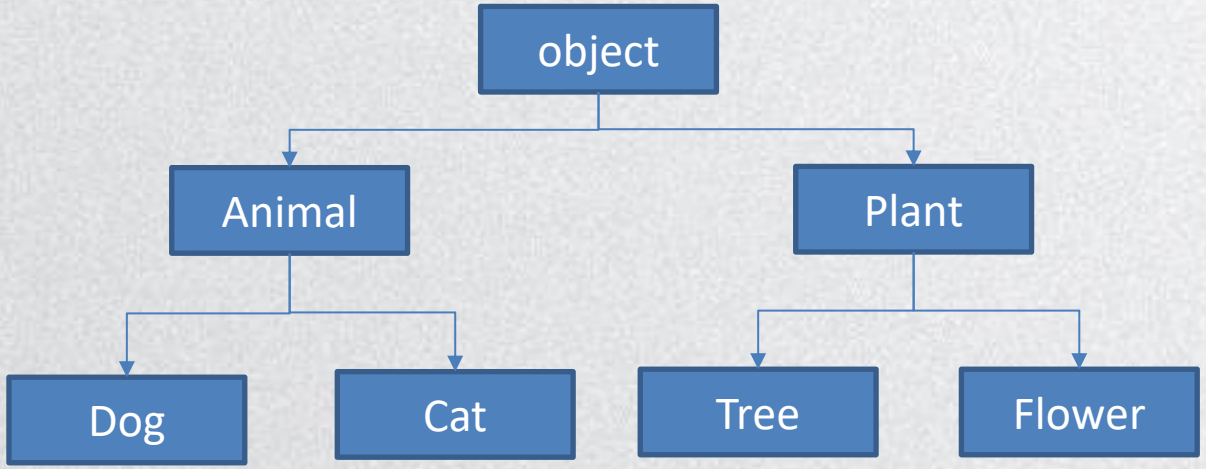
```
[5]: class Cat(Animal):

        def run(self):
            print('Cat is running...')

        def eat(self):
            print('Eating meat...')
```

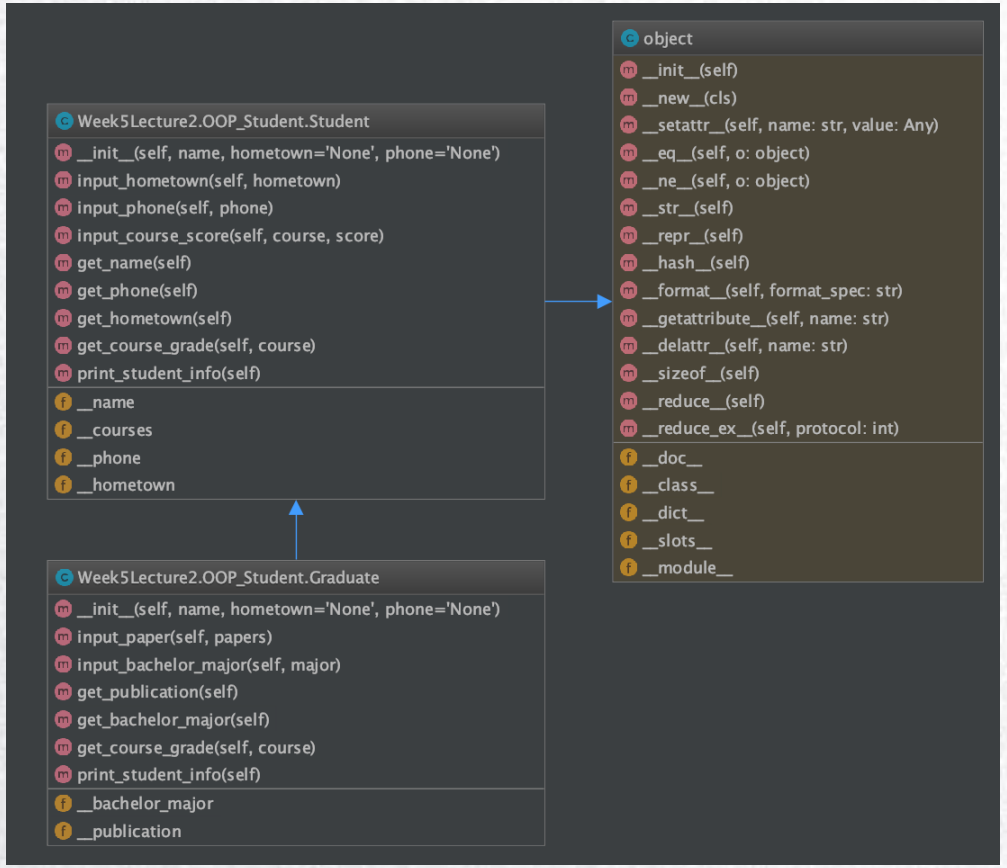
```
def run_animal(animal):
    animal.run()

run_animal(Animal())
run_animal(Dog())
run_animal(Cat())
```

```
Animal is running...
Dog is running...
Cat is running...
```

继承与多态

- 继承可以一级一级地继承下来，而任何类最终都可以追溯到根类object





获取对象信息

- 获取对象类型与方法
  - 判断对象类型，使用type()函数

```
print(type(123))
print(type('123'))
print(type([1,2,3]))

class Animal(object):
    def run(self):
        print('Animal is running ...')

a = Animal()
print(type(a))
```

```
<class 'int'>
<class 'str'>
<class 'list'>
<class '__main__.Animal'>
```

获取对象信息

- 类对应的对象的类型是什么?

```
[16]: # 类对应的对象的类型是什么?
      type(Animal)

[16]: type
```

- type函数的另外一个用途：创建类对象

```
[17]: type?

Init signature: type(self, /, *args, **kwargs)
Docstring:
type(object_or_name, bases, dict)
type(object) -> the object's type
type(name, bases, dict) -> a new type
Type:           type
Subclasses:     ABCMeta, EnumMeta, NamedTupleMeta, _TypedDictMeta, PyCStructType, UnionType, PyCPointerType, PyC
ArrayType, PyCSimpleType, PyCFuncPtrType, ...
```

```
[26]: Dog = type('Dog', (Animal,), {'run': lambda self: print('A dog is running')})

      d = Dog()
      d.run()

      A dog is running
```

获取对象信息

- 获取对象类型与方法
  - 判断对象类型，使用type()函数
  - 针对class的继承关系，使用isinstance()函数判断class类型。

例如继承关系：object -> Animal -> Dog -> Husky

```
a = Animal()
d = Dog()
h = Husky()
```

```
print(isinstance(h, Husky))
print(isinstance(h, Dog))
print(isinstance(h, Animal))
```

```
True
True
True
```

```
print(isinstance(d, Husky))
print(isinstance(h, Dog))
print(isinstance(d, Animal))
```

```
False
True
True
```

获取对象信息

- 获取对象类型与方法
  - 判断对象类型，使用type()函数
  - 针对class的继承关系，使用isinstance()函数判断class类型。

  例如继承关系：object -> Animal -> Dog -> Husky
  - 使用dir()来获得一个对象的所有属性和方法，并以一个字符串列表返回

```
a = Animal()
d = Dog()
h = Husky()
```

```
print(dir(d))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq
__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash_
_', '__init__', '__init_subclass__', '__le__', '__lt__', '__module_
_', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakr
ef__', 'eat', 'run']
```

```
print(d.__class__)
```

```
<class '__main__.Dog'>
```

获取对象信息

- 获取对象类型与方法
  - 判断对象类型，使用type()函数
  - 针对class的继承关系，使用isinstance()函数判断class类型。
  - 使用dir()来获得一个对象的所有属性和方法，并以一个字符串列表返回
  - getattr()、setattr()以及hasattr()，可直接操作一个对象的状态

```
class Animal(object):
    def __init__(self):
        self.able_to_move = True
    def run(self):
        print('Animal is running...')
```

```
a = Animal()
print(hasattr(a, 'able_to_move'))
setattr(a, 'able_to_move', False)
print(getattr(a, 'able_to_move'))
```

```
True
False
```

实例属性与类属性

- 由于Python是动态语言，根据类创建的实例可以任意绑定属性。
  - 给实例绑定属性的方法是通过实例变量，或者通过self变量
- 给类本身绑定属性，可以直接在class中定义属性，称为类属性，归类所有
  - 类属性本质上是类对象的实例属性

```
class Student(object):
    def __init__(self, name):
        self.name = name

s = Student('Bob')
s.score = 90
```

```
class Student(object):
    stu_num = 396

    def __init__(self, name):
        self.name = name
```

属性（和方法）查找过程:

- 首先，在实例属性中查找
  - 优先级最高！所以同名实例属性会覆盖类属性
- 若找不到则在实例所属的类的类属性中查找
  - 通过类属性，定义实例共有的方法!
- 若再找不到则在父类的类属性中查找
  - 实现继承
- 注意:
  - 若在类或父类属性中找到一个类型为**函数**的属性，则自动返回一个类型为**方法**的属性
  - 该查找过程是在程序运行时**动态完成**的
    - 父类方法有可能调用子类的属性和方法
    - 因此，Python的继承类似C++中的虚继承

属性（和方法）查找过程:

- 查找类属性的顺序在python里被称为Method Resolution Order

```python
# 继承
class A(object):
    def func(self):
        print("func of class A")

class B(A):
    def func(self):
        print("func of class B")

class C(B):
    def __init__(self):
        self.name = "c"
    def func(self):
        print("func of class C")

c = C()
```

```
__doc__: None
```

```python
# 查找顺序
C.mro()
```

```
[__main__.C, __main__.B, __main__.A, object]
```

鸭子类型

- 在程序设计中，鸭子类型(英语:duck typing)是动态类型(Dynamic Typing)的一种风格。
- 在这种风格中， 一个对象有效的语义，不是由继承自特定的类或实现特定的接口，而是由”当前方法和属性的集合”决定。
- 来源于由 James Whitcomb Riley 提出的鸭子测试:
  - 当看到一只鸟走起来像鸭子、游泳起来像鸭子、叫起来也像鸭子，那么这只鸟就可以被称为鸭子。
- 相较于继承来说更加灵活
  - 可以通过重载__xxx__等特殊方法来改变自定义类的行为

实现二维列表类

- 二维列表
  - 二维列表的初始化
  - 二维列表的访问，索引和获取
  - 二维列表的元素增加与删除

```
2   class my2DList(object):
3
4       def __init__(self, size=None, defaultValues=0):...
44
45      def is2DList(self, data):...
66
67      def getMy2DList(self):...
72
73      def myLen(self):...
78
79      def myIndex(self, rowIndex=None, colIndex=None):...
115
116     def myAppendRow(self, rowData):...
126
127     def myAppendCol(self, colData):...
138
139     def myDeleteRow(self, index):...
153
154     def myDeleteCol(self, index):...
```

实现二维列表类

- 二维列表
  - 二维列表的初始化（构造函数）
    - 从给定二维列表初始化
    - 指定列表大小，所有元素赋值相同
    - 从给定的一维列表初始化二维列表，并根据参数定义列表大小

实现二维列表类

- 二维列表
  - 二维列表的初始化（构造函数）

```
def __init__(self, size=None, defaultValues = 0):
    '''
    二维列表类的构造函数
    参数self：创建的实例本身
    参数size：指定二维列表大小
    参数defaultValues：输入指定列表
    '''

    if (size is None) and (not self.is2DList(defaultValues)):
        # 未指定二维列表大小，且指定列表不是二维列表
        print('Please input proper 2D list!')

    elif (size is None) and self.is2DList(defaultValues):
        # 未指定二维列表大小，但指定列表是二维列表
        self.__items = defaultValues
        self.__size = [len(defaultValues), len(defaultValues[0])]

    else:
        # 根据指定二维列表大小，创建二维列表
        self.__size = list(size)
        if(isinstance(defaultValues, int) or isinstance(defaultValues, float)):
           # 利用defaultValues作为二维列表元素
            self.__items = list()
            for i in range(self.__size[0]):
                subItems = list()
                for j in range(self.__size[1]):
                    subItems.append(defaultValues)
                self.__items.append(subItems)

        # 利用defaultValues指定的一维列表初始化二维列表
        elif(isinstance(defaultValues, list) and len(defaultValues) == self.__size[0]*self.__size[1]):
            self.__items = list()
            for i in range(self.__size[0]):
                subItems = list()
                for j in range(self.__size[1]):
                    subItems.append(defaultValues[i*self.__size[1]+j])
                self.__items.append(subItems)

        else:
            print('Incorrect defaultValues')
```

实现二维列表类

- 二维列表
  - 二维列表的初始化（构造函数）
    - 输入参数判定是否合法

```
def __init__(self, size=None, defaultValues = 0):
    '''
    二维列表类的构造函数
    参数self：创建的实例本身
    参数size：指定二维列表大小
    参数defaultValues：输入指定列表
    '''

    if (size is None) and (not self.is2DList(defaultValues)):
        # 未指定二维列表大小，且指定列表不是二维列表
        print('Please input proper 2D list!')

    elif (size is None) and self.is2DList(defaultValues):
        # 未指定二维列表大小，但指定列表是二维列表
        self.__items = defaultValues
        self.__size = [len(defaultValues), len(defaultValues[0])]

    else:
        # 根据指定二维列表大小，创建二维列表
        self.__size = list(size)
        if(isinstance(defaultValues, int) or isinstance(defaultValues, float)):
            # 利用defaultValues作为二维列表元素
            self.__items = list()
            for i in range(self.__size[0]):
                subItems = list()
                for j in range(self.__size[1]):
                    subItems.append(defaultValues)
                self.__items.append(subItems)

        # 利用defaultValues指定的一维列表初始化二维列表
        elif(isinstance(defaultValues, list) and len(defaultValues) == self.__size[0]*self.__size[1]):
            self.__items = list()
            for i in range(self.__size[0]):
                subItems = list()
                for j in range(self.__size[1]):
                    subItems.append(defaultValues[i*self.__size[1]+j])
                self.__items.append(subItems)

        else:
            print('Incorrect defaultValues')
```

实现二维列表类

- 二维列表
  - 二维列表的初始化（构造函数）
    - 输入参数判定是否合法

```python
def __init__(self, size=None, defaultValues = 0):
    '''
    二维列表类的构造函数
    参数self：创建的实例本身
    参数size：指定二维列表大小
    参数defaultValues：输入指定列表
    '''

    if (size is None) and (not self.is2DList(defaultValues)):
        # 未指定二维列表大小，且指定列表不是二维列表
        print('Please input proper 2D list!')

    elif (size is None) and self.is2DList(defaultValues):
        # 未指定二维列表大小，但指定列表是二维列表
        self.__items = defaultValues
        self.__size = [len(defaultValues), len(defaultValues[0])]

    else:
        # 根据指定二维列表大小，创建二维列表
        self.__size = list(size)
        if(isinstance(defaultValues, int) or isinstance(defaultValues, float)):
            # 利用defaultValues作为二维列表元素
            self.__items = list()
            for i in range(self.__size[0]):
                subItems = list()
                for j in range(self.__size[1]):
                    subItems.append(defaultValues)
                self.__items.append(subItems)

        # 利用defaultValues指定的一维列表初始化二维列表
        elif(isinstance(defaultValues, list) and len(defaultValues) == self.__size[0]*self.__size[1]):
            self.__items = list()
            for i in range(self.__size[0]):
                subItems = list()
                for j in range(self.__size[1]):
                    subItems.append(defaultValues[i*self.__size[1]+j])
                self.__items.append(subItems)

        else:
            print('Incorrect defaultValues')
```

```python
def is2DList(self, data):
    '''
    类的方法：判断data是否为二维列表
    '''

    if not isinstance(data, list):
        # 判断data的第一维是否为list数据类型
        return False
    else:
        colLen = len(data[0])
        for row in data:
            if (not isinstance(row, list)) or (len(row) != colLen):
                # 判断data的第二维是否为list数据类型，且元素数目是否相等
                return False
            else:
                for i in range(colLen):
                    if ((not isinstance(row[i], int)) and (not isinstance(row[j], float))):
                        # 判断data的第二维内元素是否为int类型或者float类型
                        return False

        return True # 以上均符合要求，返回True
```

实现二维列表类

- 二维列表
  - 二维列表的初始化（构造函数）
    - 从给定二维列表初始化

```
def __init__(self, size=None, defaultValues = 0):
    '''
    二维列表类的构造函数
    参数self：创建的实例本身
    参数size：指定二维列表大小
    参数defaultValues：输入指定列表
    '''

    if (size is None) and (not self.is2DList(defaultValues)):
        # 未指定二维列表大小，且指定列表不是二维列表
        print('Please input proper 2D list!')

    elif (size is None) and self.is2DList(defaultValues):
        # 未指定二维列表大小，但指定列表是二维列表
        self.__items = defaultValues
        self.__size = [len(defaultValues), len(defaultValues[0])]

    else:
        # 根据指定二维列表大小，创建二维列表
        self.__size = list(size)
        if(isinstance(defaultValues, int) or isinstance(defaultValues, float)):
            # 利用defaultValues作为二维列表元素
            self.__items = list()
            for i in range(self.__size[0]):
                subItems = list()
                for j in range(self.__size[1]):
                    subItems.append(defaultValues)
                self.__items.append(subItems)

        # 利用defaultValues指定的一维列表初始化二维列表
        elif(isinstance(defaultValues, list) and len(defaultValues) == self.__size[0]*self.__size[1]):
            self.__items = list()
            for i in range(self.__size[0]):
                subItems = list()
                for j in range(self.__size[1]):
                    subItems.append(defaultValues[i*self.__size[1]+j])
                self.__items.append(subItems)

        else:
            print('Incorrect defaultValues')
```

实现二维列表类

- 二维列表
  - 二维列表的初始化（构造函数）
    - 指定列表大小，元素赋值相同

```
def __init__(self, size=None, defaultValues = 0):
    '''
    二维列表类的构造函数
    参数self：创建的实例本身
    参数size：指定二维列表大小
    参数defaultValues：输入指定列表
    '''

    if (size is None) and (not self.is2DList(defaultValues)):
        # 未指定二维列表大小，且指定列表不是二维列表
        print('Please input proper 2D list!')

    elif (size is None) and self.is2DList(defaultValues):
        # 未指定二维列表大小，但指定列表是二维列表
        self.__items = defaultValues
        self.__size = [len(defaultValues), len(defaultValues[0])]

    else:
        # 根据指定二维列表大小，创建二维列表
        self.__size = list(size)
        if(isinstance(defaultValues, int) or isinstance(defaultValues, float)):
            # 利用defaultValues作为二维列表元素
            self.__items = list()
            for i in range(self.__size[0]):
                subItems = list()
                for j in range(self.__size[1]):
                    subItems.append(defaultValues)
                self.__items.append(subItems)

        # 利用defaultValues指定的一维列表初始化二维列表
        elif(isinstance(defaultValues, list) and len(defaultValues) == self.__size[0]*self.__size[1]):
            self.__items = list()
            for i in range(self.__size[0]):
                subItems = list()
                for j in range(self.__size[1]):
                    subItems.append(defaultValues[i*self.__size[1]+j])
                self.__items.append(subItems)

        else:
            print('Incorrect defaultValues')
```

实现二维列表类

- 二维列表
  - 二维列表的初始化（构造函数）
    - 从一维列表初始化二维列表

```
def __init__(self, size=None, defaultValues = 0):
    '''
    二维列表类的构造函数
    参数self：创建的实例本身
    参数size：指定二维列表大小
    参数defaultValues：输入指定列表
    '''

    if (size is None) and (not self.is2DList(defaultValues)):
        # 未指定二维列表大小，且指定列表不是二维列表
        print('Please input proper 2D list!')

    elif (size is None) and self.is2DList(defaultValues):
        # 未指定二维列表大小，但指定列表是二维列表
        self.__items = defaultValues
        self.__size = [len(defaultValues), len(defaultValues[0])]

    else:
        # 根据指定二维列表大小，创建二维列表
        self.__size = list(size)
        if(isinstance(defaultValues, int) or isinstance(defaultValues, float)):
            # 利用defaultValues作为二维列表元素
            self.__items = list()
            for i in range(self.__size[0]):
                subItems = list()
                for j in range(self.__size[1]):
                    subItems.append(defaultValues)
                self.__items.append(subItems)

        # 利用defaultValues指定的一维列表初始化二维列表
        elif(isinstance(defaultValues, list) and len(defaultValues) == self.__size[0]*self.__size[1]):
            self.__items = list()
            for i in range(self.__size[0]):
                subItems = list()
                for j in range(self.__size[1]):
                    subItems.append(defaultValues[i*self.__size[1]+j])
                self.__items.append(subItems)

        else:
            print('Incorrect defaultValues')
```

实现二维列表类

- 二维列表
  - 二维列表的访问，索引和获取
    - 二维列表的访问（私有变量访问）

```
def getMy2DList(self):
    '''
    类的方法：返回二维列表
    '''
    return self.__items
```

实现二维列表类

- 二维列表
  - 二维列表的访问，索引和获取
    - 二维列表的访问（私有变量访问）

```
def mySize(self):
    '''
    类的方法：返回二维列表的大小
    '''
    return self.__size[0], self.__size[1]
```

实现二维列表类

- 二维列表
  - 二维列表的访问，索引和获取
    - 二维列表的索引和获取

```
def myIndex(self, rowIndex=None, colIndex=None):
    '''
    类的方法：对二维列表索引
    参数rowIndex：索引的行坐标
    参数colIndex：索引的列坐标
    '''
    if (rowIndex is None) and (colIndex is None):
        # rowIndex和colIndex均未给定，输出错误
        print('Please input index number!')

    elif (isinstance(rowIndex, int) and (colIndex is None)):
        # 仅给定行索引值，则仅对行进行索引
        if (rowIndex >= 0) and (rowIndex < self.__size[0]):
            return self.__items[rowIndex]
        else:
            print('Row index out of range!')

    elif (rowIndex is None) and isinstance(colIndex, int):
        # 仅给定列索引值，则仅对列进行索引
        if (colIndex >= 0) and (colIndex < self.__size[1]):
            colItems = list()
            for row in self.__items:
                colItems.append(row[colIndex])
            return colItems
        else:
            print('Column index out of range!')

    elif (isinstance(colIndex, int) and isinstance(colIndex, int)):
        # 给定行和列索引值，则对指定元素进行索引
        if (rowIndex>=0) and (rowIndex < self.__size[0]) and (colIndex>=0) and (colIndex < self.__size[1]):
            return self.__items[rowIndex][colIndex]
        else:
            print('Index out of range!')

    else:
        print('Please input int-type Index!')
```

实现二维列表类

- 二维列表
  - 二维列表的访问，索引和获取
    - 二维列表的索引和获取
      - 索引值判定是否合法

```
def myIndex(self, rowIndex=None, colIndex=None):
    '''
    类的方法：对二维列表索引
    参数rowIndex：索引的行坐标
    参数colIndex：索引的列坐标
    '''
    if (rowIndex is None) and (colIndex is None):
        # rowIndex和colIndex均未给定，输出错误
        print('Please input index number!')

    elif (isinstance(rowIndex, int) and (colIndex is None)):
        # 仅给定行索引值，则仅对行进行索引
        if (rowIndex >= 0) and (rowIndex < self.__size[0]):
            return self.__items[rowIndex]
        else:
            print('Row index out of range!')

    elif (rowIndex is None) and isinstance(colIndex, int):
        # 仅给定列索引值，则仅对列进行索引
        if (colIndex >= 0) and (colIndex < self.__size[1]):
            colItems = list()
            for row in self.__items:
                colItems.append(row[colIndex])
            return colItems
        else:
            print('Column index out of range!')

    elif (isinstance(colIndex, int) and isinstance(colIndex, int)):
        # 给定行和列索引值，则对指定元素进行索引
        if (rowIndex>=0) and (rowIndex < self.__size[0]) and (colIndex>=0) and (colIndex < self.__size[1]):
            return self.__items[rowIndex][colIndex]
        else:
            print('Index out of range!')

    else:
        print('Please input int-type Index!')
```

实现二维列表类

- 二维列表
  - 二维列表的访问，索引和获取
    - 二维列表的索引和获取
      - 仅给定行索引值

```
def myIndex(self, rowIndex=None, colIndex=None):
    '''
    类的方法：对二维列表索引
    参数rowIndex：索引的行坐标
    参数colIndex：索引的列坐标
    '''
    if (rowIndex is None) and (colIndex is None):
        # rowIndex和colIndex均未给定，输出错误
        print('Please input index number!')

    elif (isinstance(rowIndex, int) and (colIndex is None)):
        # 仅给定行索引值，则仅对行进行索引
        if (rowIndex >= 0) and (rowIndex < self.__size[0]):
            return self.__items[rowIndex]
        else:
            print('Row index out of range!')

    elif (rowIndex is None) and isinstance(colIndex, int):
        # 仅给定列索引值，则仅对列进行索引
        if (colIndex >= 0) and (colIndex < self.__size[1]):
            colItems = list()
            for row in self.__items:
                colItems.append(row[colIndex])
            return colItems
        else:
            print('Column index out of range!')

    elif (isinstance(colIndex, int) and isinstance(colIndex, int)):
        # 给定行和列索引值，则对指定元素进行索引
        if (rowIndex>=0) and (rowIndex < self.__size[0]) and (colIndex>=0) and (colIndex < self.__size[1]):
            return self.__items[rowIndex][colIndex]
        else:
            print('Index out of range!')

    else:
        print('Please input int-type Index!')
```

实现二维列表类

- 二维列表
  - 二维列表的访问，索引和获取
    - 二维列表的索引和获取
      - 仅给定列索引值

```
def myIndex(self, rowIndex=None, colIndex=None):
    '''
    类的方法：对二维列表索引
    参数rowIndex：索引的行坐标
    参数colIndex：索引的列坐标
    '''
    if (rowIndex is None) and (colIndex is None):
        # rowIndex和colIndex均未给定，输出错误
        print('Please input index number!')

    elif (isinstance(rowIndex, int) and (colIndex is None)):
        # 仅给定行索引值，则仅对行进行索引
        if (rowIndex >= 0) and (rowIndex < self.__size[0]):
            return self.__items[rowIndex]
        else:
            print('Row index out of range!')

    elif (rowIndex is None) and isinstance(colIndex, int):
        # 仅给定列索引值，则仅对列进行索引
        if (colIndex >= 0) and (colIndex < self.__size[1]):
            colItems = list()
            for row in self.__items:
                colItems.append(row[colIndex])
            return colItems
        else:
            print('Column index out of range!')

    elif (isinstance(colIndex, int) and isinstance(colIndex, int)):
        # 给定行和列索引值，则对指定元素进行索引
        if (rowIndex>=0) and (rowIndex < self.__size[0]) and (colIndex>=0) and (colIndex < self.__size[1]):
            return self.__items[rowIndex][colIndex]
        else:
            print('Index out of range!')

    else:
        print('Please input int-type Index!')
```

实现二维列表类

- 二维列表
  - 二维列表的访问，索引和获取
    - 二维列表的索引和获取
      - 行、列索引均给定

```
def myIndex(self, rowIndex=None, colIndex=None):
    '''
    类的方法：对二维列表索引
    参数rowIndex：索引的行坐标
    参数colIndex：索引的列坐标
    '''
    if (rowIndex is None) and (colIndex is None):
        # rowIndex和colIndex均未给定，输出错误
        print('Please input index number!')

    elif (isinstance(rowIndex, int) and (colIndex is None)):
        # 仅给定行索引值，则仅对行进行索引
        if (rowIndex >= 0) and (rowIndex < self.__size[0]):
            return self.__items[rowIndex]
        else:
            print('Row index out of range!')

    elif (rowIndex is None) and isinstance(colIndex, int):
        # 仅给定列索引值，则仅对列进行索引
        if (colIndex >= 0) and (colIndex < self.__size[1]):
            colItems = list()
            for row in self.__items:
                colItems.append(row[colIndex])
            return colItems
        else:
            print('Column index out of range!')

    elif (isinstance(colIndex, int) and isinstance(colIndex, int)):
        # 给定行和列索引值，则对指定元素进行索引
        if (rowIndex>=0) and (rowIndex < self.__size[0]) and (colIndex>=0) and (colIndex < self.__size[1]):
            return self.__items[rowIndex][colIndex]
        else:
            print('Index out of range!')

    else:
        print('Please input int-type Index!')
```

实现二维列表类

- 二维列表
  - 二维列表的元素**增加**与删除

```
def myAppendRow(self, rowData):
    '''
    类的方法：向二维列表行末添加新的一行
    参数rowData：输入的添加行
    '''
    if self.__size[1] == len(rowData):
        self.__items.append(rowData)
        self.__size[0] += 1
    else:
        print('Appended data not fit the List row!')

def myAppendCol(self, colData):
    '''
    类的方法：向二维列表列末添加新的一列
    参数colData：输入的添加列
    '''
    if self.__size[0] == len(colData):
        for i in range(self.__size[0]):
            self.__items[i].append(colData[i])
        self.__size[1] += 1
    else:
        print('Appended data not fit the List column!')
```

实现二维列表类

- 二维列表
  - 二维列表的元素增加与**删除**

```
def myDeleteRow(self, index):
    '''
    类的方法：删除二维列表的中指定行
    参数index：需删除的行索引
    '''
    if index >= 0 and index < self.__size[0]:
        self.newItems = list()
        for i in range(self.__size[0]):
            if i != index:
                self.newItems.append(self.__items[i])
        self.__items = self.newItems
        self.__size[0] -= 1
    else:
        print('Index out of range!!')

def myDeleteCol(self, index):
    '''
    类的方法：删除二维列表的中指定列
    参数index：需删除的列索引
    '''
    if index >= 0 and index < self.__size[1]:
        self.newItems = list()
        for i in range(self.__size[0]):
            subItems = list()
            for j in range(self.__size[1]):
                if j != index:
                    subItems.append(self.__items[i][j])
            self.newItems.append(subItems)
        self.__items = self.newItems
        self.__size[1] -= 1
    else:
        print('Index out of range!!')
```

实现二维列表类

- 二维列表
  - 二维列表的初始化（构造函数）
  - 二维列表的访问，索引和获取
  - 二维列表的元素增加与删除

```
2   class my2DList(object):
3
4       def __init__(self, size=None, defaultValues=0):...
44
45      def is2DList(self, data):...
66
67      def getMy2DList(self):...
72
73      def myLen(self):...
78
79      def myIndex(self, rowIndex=None, colIndex=None):...
115
116     def myAppendRow(self, rowData):...
126
127     def myAppendCol(self, colData):...
138
139     def myDeleteRow(self, index):...
153
154     def myDeleteCol(self, index):...
```

实现二维列表类

- 二维列表
  - 二维列表的初始化
    - 指定列表大小，所有元素赋值相同
    - 从给定的一维列表初始化二维列表，并根据参数定义列表大小
    - 从给定二维列表初始化

```
In [70]: customList0 = my2DList()
         Please input proper 2D list!

In [71]: customList1 = my2DList(size=(2,3), defaultValues=1)

In [72]: customList1.getMy2DList()
Out[72]: [[1, 1, 1], [1, 1, 1]]

In [73]: customList2 = my2DList(size=(2,3), defaultValues=[1,2,3,4,5,6])

In [74]: customList2.getMy2DList()
Out[74]: [[1, 2, 3], [4, 5, 6]]

In [75]: customList3 = my2DList(size=None, defaultValues=[[1,3,6],[4,6,6]])

In [76]: customList3.getMy2DList()
Out[76]: [[1, 3, 6], [4, 6, 6]]
```

实现二维列表类

- 二维列表
  - 二维列表的访问，索引和获取

```
In [81]: customList = my2DList(size=(2,3), defaultValues=[1,2,3,4,5,6]) # 初始化
Out[81]: (2, 3)

In [82]: customList.getMy2DList()  # 2D list 获取
Out[82]: [[1, 2, 3], [4, 5, 6]]

In [83]: customList.mySize()  # 2D list size 获取
Out[83]: (2, 3)

In [84]: customList.myIndex(0,1) # # 2D list 索引
Out[84]: 2
```

实现二维列表类

- 二维列表
  - 二维列表的访问，索引和获取

```
In [85]: customList.myIndex(0,1) # 2D list 元素索引
Out[85]: 2

In [86]: customList.myIndex(0, None) # 2D list 行索引
Out[86]: [1, 2, 3]

In [87]: customList.myIndex(None, 1) # 2D list 列索引
Out[87]: [2, 5]
```

实现二维列表类

- 二维列表
  - 二维列表的元素增加与删除

```
In [99]: customList.myAppendCol([7,8]) # 2D list 列添加

In [100]: customList.getMy2DList()
Out[100]: [[1, 2, 3, 7], [4, 5, 6, 8]]

In [101]: customList.mySize()
Out[101]: (2, 4)

In [102]: customList.myDeleteCol(2) # 2D list 列删除

In [103]: customList.getMy2DList()
Out[103]: [[1, 2, 7], [4, 5, 8]]
```

实现二维列表类

- 通过重载__xxx__等特殊方法来改变自定义类的行为
  - 目的：使其行为更像python自带的list对象
  - 思考这样有什么优点和缺点
- 重载：
  - __repr__： 打印实例时输出字符串
  - __len__： 内置函数len()的返回值
  - __getitem__: []操作符
  - __setitem__: 赋值语句中的[]操作符
  - __iter__ and __next__： for循环中使用

人工智能与
Python程序设计

回顾

1. 类和实例
2. 数据封装
3. 访问限制
4. 继承和多态
5. 获取对象信息
6. 实例属性和类属性
7. 查找属性的过程
8. 鸭子类型和magic function

谢谢！