

第二章 Python 面向对象相关基础



九章算法-Justin

United States

林平之 老师



Scan the QR code to add me on WeChat

对课程有疑问？购买时遇到问题？获取更多优惠信息？
扫一扫，在线咨询

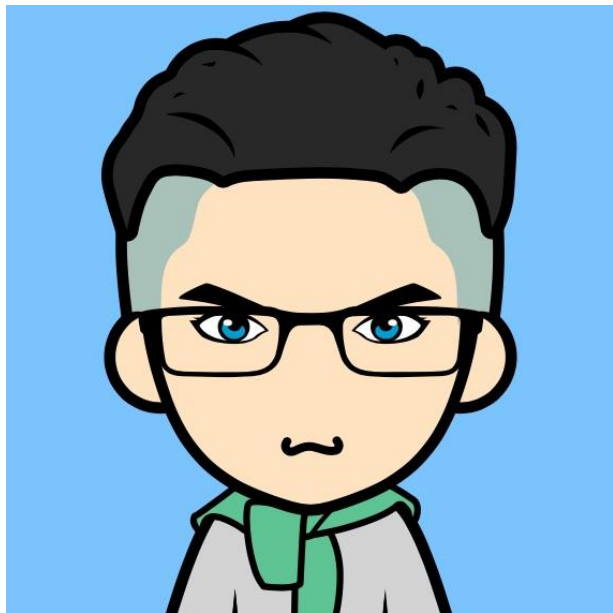
微信: [ninechapter](#)

知乎专栏: <http://zhuanlan.zhihu.com/jiuzhang>

微博: <http://www.weibo.com/ninechapter>

官网: www.jiuzhang.com

九章课程不提供视频，也严禁录制视频的侵权行为
否则将追求法律责任和经济赔偿
请不要缺课



林老师

全国算法竞赛一等奖
国内TOP2名校毕业
参加国家信息学竞赛NOI
前FLAG工程师
拥有丰富的面试经验

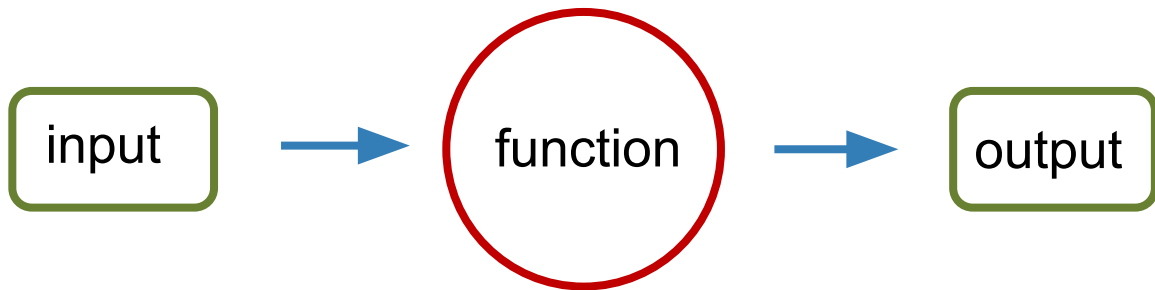
- 函数与类 Function & Class
- 类的继承 Inheritance
- 异常的捕捉 Exception
- Json数据类型

函数 Function

- 函数: 具有特定功能的代码段
- Why?
 - 增加代码复用 (code reuse)
 - 增强程序可读性 (readability)

Python有很多内建的函数, 如 `print()`, 用户也可以自己定义函数, 称之为用户自定义函数。

- 函数
 - 广义来讲，有输入有输出就是函数



def关键字

函数名

参数

函数体

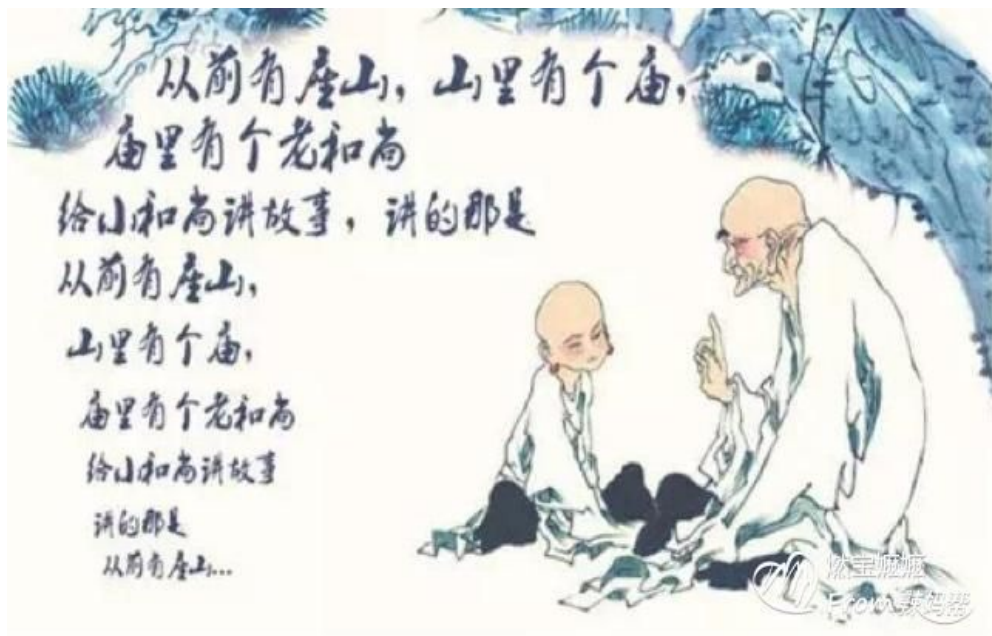
```
1 def find_number(nums, target):  
2     for num in nums:  
3         if num == target:  
4             return True  
5  
6     return False
```

返回值

- 函数的调用
 - 给定输入, 返回输出
 - 程序执行流程(演示)
 - 把函数的输出当做一个值来使用
- 函数的参数传递

什么是递归 (Recursion)?

- 函数自己调用自己



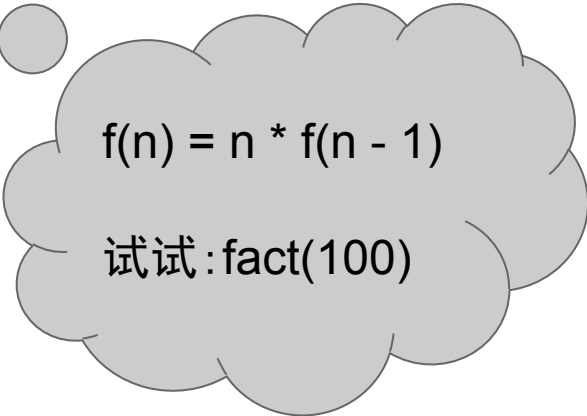
没错，这就是递归。。。

- 递归的定义
 - 首先这个问题或者数据结构得是递归定义的
- 递归的出口
 - 什么时候递归终止
- 递归的拆解
 - 递归不终止的时候, 如何分解问题

- 求n! 值 ($n! = 1 * 2 * 3 * \dots * n$)
- ```
def fact(n):

 if n == 0 or n == 1:
 return 1
 return n * fact(n - 1)

print(fact(10))
```


$$f(n) = n * f(n - 1)$$

试试: fact(100)

=fact(4)

=4 \* fact(3)

=4 \* (3 \* fact(2))

=4 \* (3 \* 2 \* fact(1))

=4 \* 3 \* 2 \* 1

=24

试试: fact(10000)

**RecursionError:** maximum  
recursion depth exceeded  
in comparison



# 类和对象 Class & Object

- 什么是面向对象(Object - Oriented) ?
  - 面向对象是一种世界观: 世间万物皆为对象
  - 面向对象是一种程序设计方式

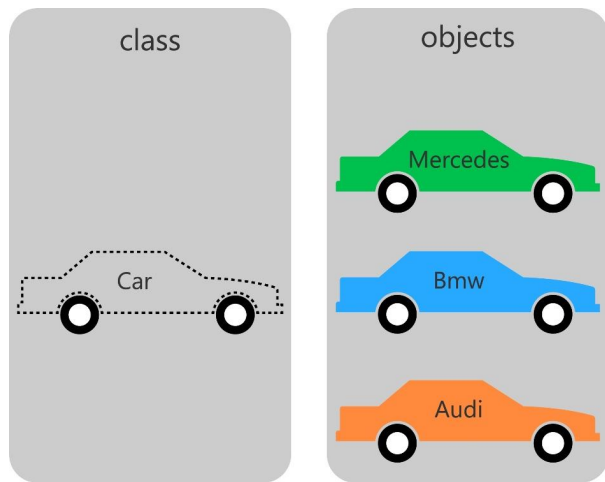


- 什么是对象(Object)？
  - 在面向对象的世界观中: 世间万物皆为对象
  - 属性 & 行为
  - 小狗
  - 汽车
  - 电脑



- 对象的属性
  - 小狗: 四条腿
  - 汽车: 四个轮子, 方向盘
  - 电脑: 屏幕, 键盘
- 对象的行为
  - 小狗: 汪汪叫
  - 汽车: 加减速, 转弯
  - 电脑: 运行程序, 播放视频

- 什么是类(class)？
  - 类是对象的蓝图



- 在Python中，类是对现实事物的抽象

class关键字      类名 (upper camel case)

方法(行为)

```
1 class Student():
2
3 def __init__(self, name, score):
4 self.name = name
5 self.score = score
6
7 def speak(self):
8 print(self.name, self.score)
```

变量(属性)

- `__init__()` 是构造函数，当创建这个类的对象时自动调用
- `self` 指的是对象本身，`self` 在定义类的方法时是必须有的，但是调用时看不到这个参数

```
1 class Student():
2
3 def __init__(self, name, score):
4 self.name = name
5 self.score = score
```

- 创建对象

```
class Student:

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

jack = Student('Jack', 80)
jack.score = 98
print(jack.name, jack.score)
```

- 什么是实例or对象 (Instance or Object) ?
  - 实例就是对象

- 成员变量(member variable)
  - 又叫域(field)
  - 表示对象的属性
  - 命名规则和变量名相同

- 成员函数(member function)
  - 又叫方法(method)
  - 表示对象的行为
  - 命名规则和函数名相同

- 什么是OOP？
  - 用对象构建程序
- 面向对象的三大特征
  - 封装(Encapsulation)
  - 继承(Inheritance)
  - 多态(Polymorphism)



- 封装性
  - 将属性和行为**封装**成一个类, 并尽可能**隐蔽**类(对象)的内部细节, 对外形成一个边界, 只保留有限的**对外接口**使之与外部发生联系
- 代码演示



- 封装性
  - 改变程序的组织方式
  - 增加代码的复用率
  - 提高程序开发效率

- 构造函数 (constructor)
- field和method的访问权限

## 类的属性 - 单、双下划线、头尾双下划线的情况：

- **xxx**: 普通的方法
- **\_\_xxx\_\_**: 这是特殊方法, 类似 `__init__()`
- **\_xxx**: 单下划线开头表示是 **protected** 类型, 保护类型只允许其本身与子类进行访问
- **\_\_xxx**: 双下划线表示私有类型(private)类型, 只能是允许这个类中进行访问。
- (对应Java或者Cpp的 **public,protected, private**)

print(Dog)

print(dog)

提问: 以上两者有什么不同?

```
class Dog:
```

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

```
 def speak(self):
 print(self.name + ': Wow')
```

```
dog = Dog('Wang cai')
dog.speak()
```

其他的访问类的属性的常用方法：

- `hasattr` 检查是否存在一个属性
- `getattr` 访问对象的属性
- `setattr` 设置一个属性, 如果属性不存在呢？

创建一个Vector类:

- “重写”\_\_str\_\_ 方法, 支持展示一个向量如[1,2]
- “重写”\_\_add\_\_ 方法, 支持向量的加和如[1,2] + [3,4]

```
class Vector:
```

```
 def __init__(self, x, y):
 self.x = x
 self.y = y
```

```
 def __str__(self):
 return "[%s, %s]" % (self.x, self.y)
```

```
 def __add__(self, other):
 return Vector(self.x + other.x, self.y + other.y)
```



# 使用@property



避免属性被直接修改, 如下:

```
jack = Student('Jack', 91)
jack.score = 9999
```

对于对象的属性直接对修改成了9999, 超出了正常值的范围, 我们需要做限制

```
class Student:

 def get_score(self):
 return self._score

 def set_score(self, value):
 assert isinstance(value, int)
 assert value >= 0 and value <= 100
 self._score = value

jack = Student()
jack.set_score(9999)
print(jack.get_score())
```

设置score为\_score; get\_score获取分数; set\_score来设置分数; 中途做检查！ - Python中有更加简单的方法

```
class Student:
```

```
 @property
```

```
 def score(self):
 return self._score
```

```
 @score.setter
```

```
 def score(self, value):
 assert isinstance(value, int)
 assert value >= 0 and value <= 100
 self._score = value
```

```
jack = Student()
jack.score = 99
print(jack.score)
```

Property做的事情：

1、把getter方法变成属性

2、创建了另一个装饰器

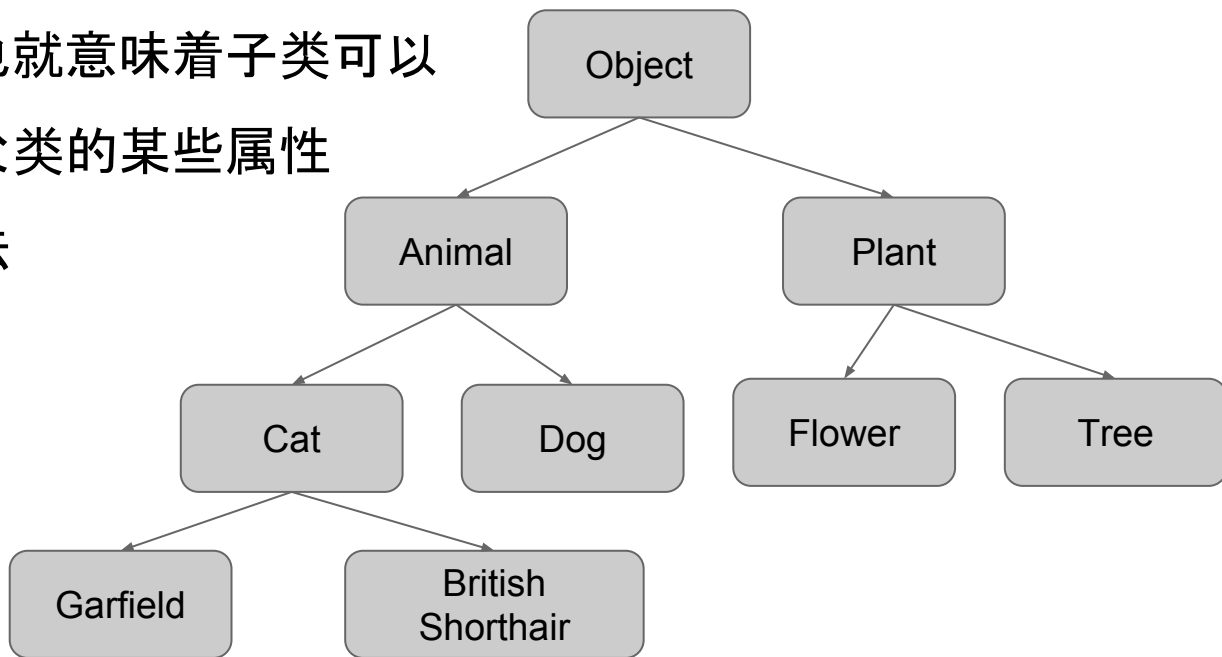
score.setter, 把一个setter方法  
变成属性赋值



# 类的继承Inheritance

子类继承父类：

继承也就意味着子类可以  
继承父类的某些属性  
和方法



- 创建父类Animal
- 创建子类Cat & Dog, 继承与Animal
  - 子类继承父类的属性和方法
  - 子类使用父类的protected的属性
  - 子类重载父类的方法
  - 子类添加父类没有的属性和方法

提问:Animal这个父类中, 哪些属性和方法是可以被子类继承和使用的?

name ?

\_\_color ?

get\_name ?

\_\_get\_color ?

\_\_get\_color ?

speak() ?

```
class Animal:

 def __init__(self, name):
 self.name = name
 self.__color = 'red'

 def get_name(self):
 return self.name

 def get_color(self):
 return self.__color

 def __get_color(self):
 return self.__color

 def __get_color(self):
 return self.__color

 def speak(self):
 return 'LoL'
```

Dog从Animal继承:

```
class Dog(Animal):

 def __init__(self, name):
 Animal.__init__(self, name)

 def speak(self):
 return self.name + ': Wow'

dog = Dog("Wang CaiCai")

print(dog.get_color())
print(dog.speak())
```



Isinstance(obj, type) 可以用来判断对象obj是否是type这种类型

如右的继承关系，四个选项，哪些是True：

```
class Animal:

```

```
class Dog(Animal):

```

A isinstance(dog, Dog)

B isinstance(dog, Animal)

C isinstance(animal, Animal)

D isinstance(animal, Dog)



给实例绑定属性的方法是通过实例变量，或者通过self变量：

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

s = Student('Alice')
```

如果Student类本身需要绑定一个属性呢？

可以直接在class中定义属性，这种属性是类属性，归Student类所有，所有该类创建的对象都共享这个属性

```
class Student:
 type = 'Student'

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

s1 = Student('Alice')
s2 = Student('Bob')

print(s1.type)
print(s2.type)
```

提问: 如何给Student类增加一个类属性, 对于每创建一个对象, 这个类属性自动增加, 最后可以统计出有多少学生被创建了

继承是面向对象编程的一个重要的方式，因为通过继承，子类就可以扩展父类的功能。

根据Animal类层次的设计，假设我们要实现以下4种动物：

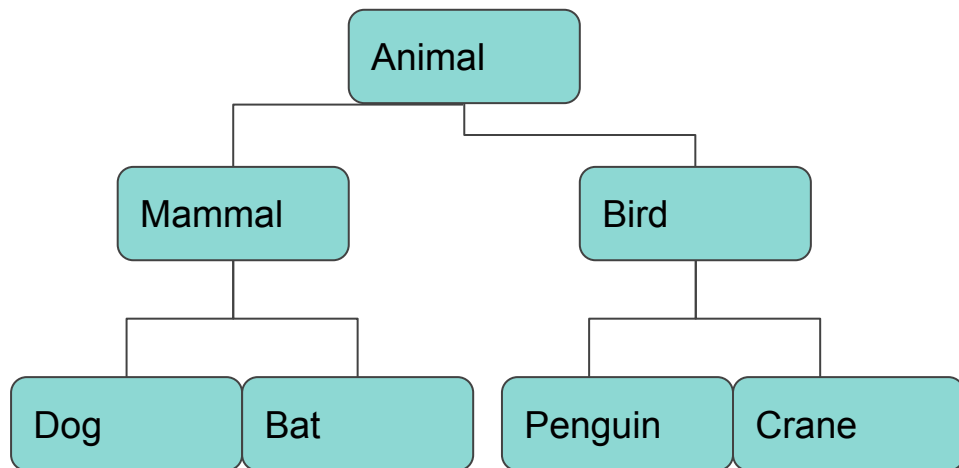
Dog 狗

Bat 蝙蝠

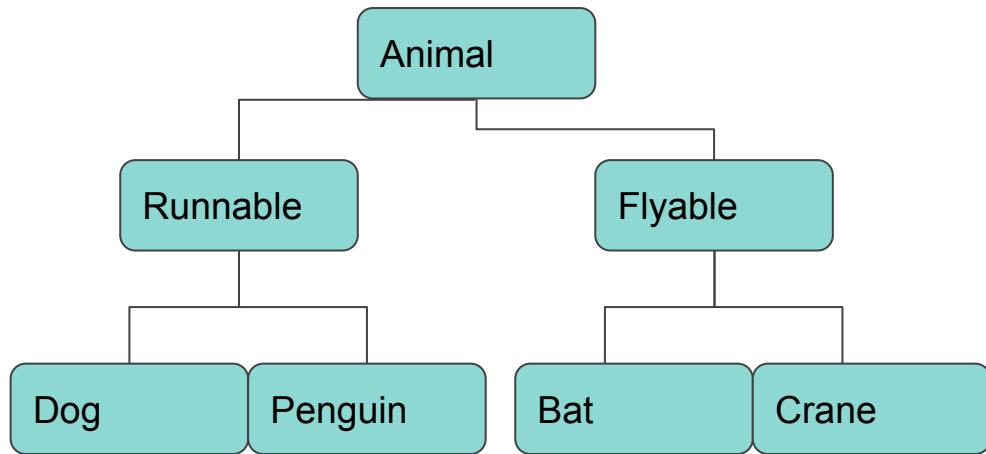
penguin 企鹅

Crane 仙鹤

## 根据哺乳类和鸟类分类



根据能否或者不能飞



我们可以设计Mammal, Bird, Runnable, Flyable

四个类, 使得动物有选择的继承:

例如 Dog继承自Mammal和Runnable

```
class Dog(Animal, Runnable):

 def __init__(self):
 super(Dog, self).__init__()

dog = Dog()
dog.run()
```

```
class Animal(object):
 pass

class Mammal(Animal):
 pass

class Bird(Animal):
 pass

class Runnable(object):
 def run(self):
 print('Running...')

class Flyable(object):
 def fly(self):
 print('Flying...')
```



在设计类的继承关系时，通常，主线都是单一继承下来的，例如，Crane继承自Bird。但是，如果需要“混入”额外的功能，通过多重继承就可以实现，比如，让Crane除了继承自Bird外，再同时继承Runnable。这种设计通常称之为MixIn。

为了更好地看出继承关系，我们把Runnable和Flyable改为RunnableMixIn和FlyableMixIn



# 文件读写File

- open 函数
- file object = open(file\_name [, access\_mode])
- file\_name: 是一个包含了要访问的文件路径和名称的字符串值
- access\_mode: 打开文件的模式, 如读, 写, 追加等

```
file = open('a.txt', 'w')
print(file.closed)
print(file.name)
```

- closed: true如果文件已被关闭, 否则返回false
- name: 返回被打开文件的名字

- read() 文件读取内容
- write() 向文件写入内容
- close() 关闭文件

- 重命名文件

```
import os
```

```
os.rename(src, dst)
```

- 判断文件是否存在

```
os.path.exists(path)
```



# 异常Exception

- 异常是什么？
  - 异常相当于一个事件，该事件会在程序执行过程中发生，影响了程序的正常执行。
  - 程序语句无法正常处理时就会发生异常
  - 异常是对象，表示一个错误。

提问：如果我们不捕捉这个异常，程序将会发生什么？





# try/except语句



```
1 try:
2 ...
3 except:
4 ...
5 else:
6 ...
7
```

提问环节:

代码块1和代码块2的  
输出结果分别是什么？

代码块1:

```
a, b = None, None
```

```
try:
```

```
 a = 2
```

```
except:
```

```
 b = 1
```

```
else:
```

```
 b = 200
```

```
print(a, b)
```

代码块2:

```
a, b = None, None
```

```
try:
```

```
 a[1] = 2
```

```
except:
```

```
 b = 1
```

```
else:
```

```
 b = 200
```

```
print(a, b)
```

可以在except后面指定一些异常如except IOError

提供了一下标准异常如：

- ValueError
- IOError
- RuntimeError
- TypeError
- NameError
- 更多Error: <https://docs.python.org/2/tutorial/errors.html>

- 无论是否发生异常都将执行finally最后的代码
- 做一些无论处理结果如何都必须要做的一些收尾工作
  - 比如最终都需要Close文件

```
1 try:
2 ...
3 finally:
4 ...
5
6
```



# Json数据格式

- json 模块提供了很简单的方式来编码和解码JSON数据
- 是轻量级的数据交换格式
- 两个主要的函数是
  - json.dumps()
  - json.loads()

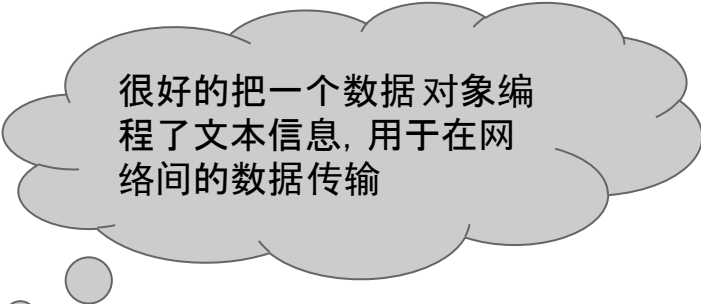
# Python Json数据 dumps



九章算法

```
import json
```

```
data = {
 'linpz': 6,
 'linghuc': 5,
 'zhongy': 9,
 'score': [1, 4, 5, 6]
}
response = json.dumps(data)
print(type(response))
print(response)
```

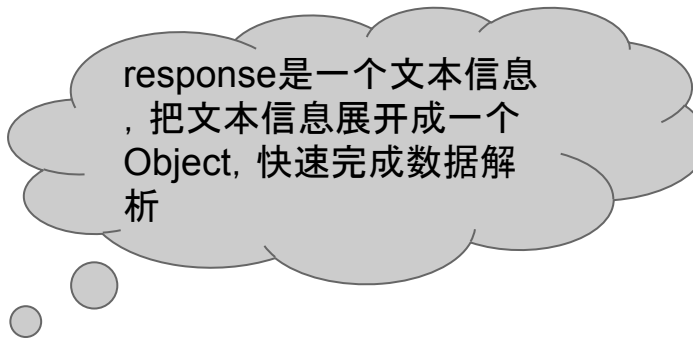


很好的把一个数据对象编程了文本信息, 用于在网络间的数据传输

# Python Json数据 loads

```
import json
```

```
data = {
 'linpz': 6,
 'linghuc': 5,
 'zhongy': 9,
 'score': [1, 4, 5, 6]
}
response = json.dumps(data)
print(type(response))
print(json.loads(response))
```



response是一个文本信息  
，把文本信息展开成一个  
Object，快速完成数据解  
析

1. 递归实现  $1 + 2 + 3 + \dots + n$
2. 实现Animal类, 从这个类继承, 创建更多的子类
3. 练习文件的读写, 内容任意
4. 使用Json完成字符串和数据之间的转换





谢谢大家