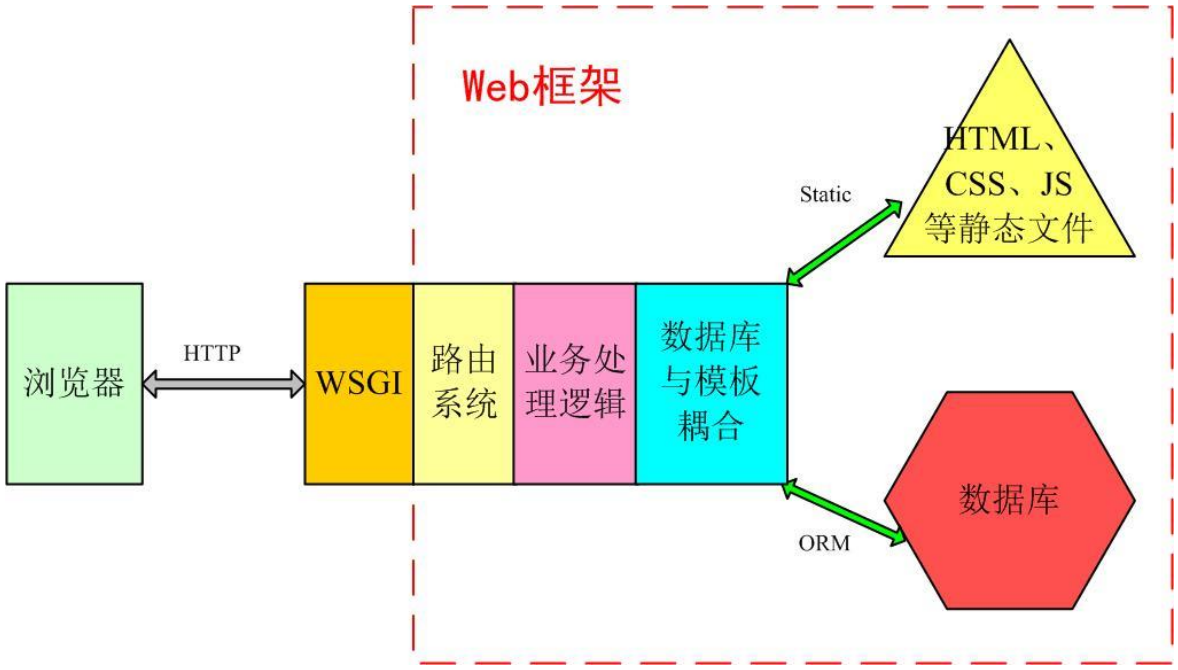
目录

一、 阅读说明

*绿色斜体*代表个人的思考理解，*黄色斜体*代表阅读理解过程中的疑问，红色正体代表关键重要信息，下划线代表次关键重要信息

二、 基本概念

1. Django:开放源代码的 Web 应用框架：



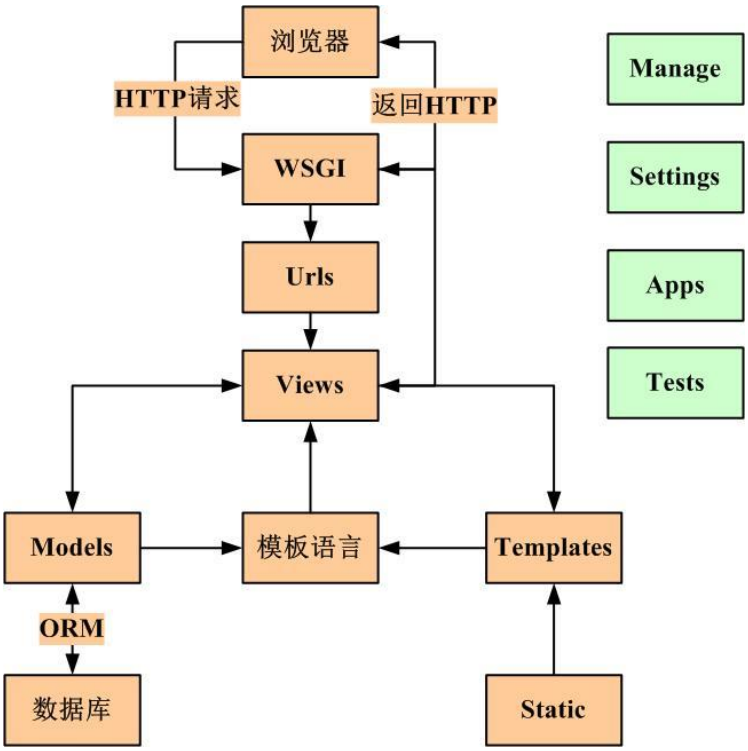
2. MVC/MTV

全名 Model View Controller，是模型(model)－视图(view)－控制器(controller)的缩写，一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。通俗解释：一种文件的组织和管理形式！这其实就是把不同类型的文件放到不同的目录下的一种方法，然后取了个高大上的名字。当然，它带来的好处有很多，比如前后端分离，松耦合等等

模型(model)：定义数据库相关的内容，一般放在 models.py 文件中。
视图(view)：定义 HTML 等静态网页文件相关，也就是那些 html、css、js 等前端的东西。
控制器(controller)：定义业务逻辑相关，就是你的主要代码

MTV: view 不再是 HTML 相关，而是主业务逻辑了，相当于控制器。html 被放在 Templates 中，称作模板，于是 MVC 就变成了 MTV。和 MVC 本质上是一样的，换汤不换药。

3. Django 的 MTV 模型组织



三、 Django 基本结构

urls.py：网址入口，关联到对应的 views.py 中的一个函数（或者 generic 类），访问网址就对应一个函数，网址是写在 urls.py 文件中，用正则表达式对应 views.py 中的一个函数(或者 generic 类);

views.py：处理用户发出的请求，从 urls.py 中对应过来，通过渲染 templates 中的网页可以将显示内容，比如登陆后的用户名，用户请求的数据，输出到网页;

models.py:与数据库操作相关，存入或读取数据时用到这个，用不到数据库的时候可以不使用;

forms.py:表单，用户在浏览器上输入数据提交，对数据的验证工作以及输入框的生成等工作，当然你也可以不使用;

templates 文件夹: views.py 中的函数渲染 templates 中的 Html 模板，得到动态内容的网页，当然可以用缓存来提高速度;

admin.py:后台，可以用很少量的代码就拥有一个强大的后台。

settings.py: Django 的设置，配置文件，比如 DEBUG 的开关，静态文件的位置等。

四、 Django 基本工作流程

简化过程：程序根据 urls.py 中网址调用 views.py 中函数，views.py 中操作函数会使用编写的模板文件。

五、 urls.py 中的 name 参数

假设我们在模板文件中采用如下方式书写网址(死网址)：

```
1. <a href="/add/4/5/">计算 4+5</a>
```

urls.py 中按照如下方式调用：

```
1. url('add/4/5/', calc_views.add2, name='add2'),
```

正常状态下 views.py 调用模板文件，在页面上显示了超链接计算 4+5，点击该链接跳转至：http://xxx.xxx.xxx/add/4/5，该链接在 urls.py 中对应 calc.views.add2 函数，从而调用 views.py 中的 add2 函数得出结果。即渲染过程为：点击超链接→跳转到模板文件中链接对应的 path→调用该 path 对应的函数。但是如果需求变更，用户希望输入 http://xxx.xxx.xxx/add_new/4/5 来计算结果，那么我们就需要更改模板中的 url 表达方式，同时还要改 urls.py 中的 url 表达方式，任何采用”/add/4/5”这种形式的文件都要改。但是如果我们把模板文件采取如下方式书写(灵活网址)：

```
1. <a href="{% url 'add2' 4 5 %}">link</a>
```

不直接采用 urls.py 中的 path，而是采用 urls.py 中的 name 参数，则渲染过程变为：点击超链接→寻找 name 为 add2→跳转到 name=add2 对应的 path→调用该 path 对应的函数。所以此时我们只需要把 urls.py 中的调用语句修改即可，模板文件无需改动：

```
1. url('add_new/4/5/', calc_views.add2, name='add2'),
```

六、 Django 模板查找机制

Django 查找模板的过程是在每个 app 的 templates 文件夹中找（而不只是当前 app 中的代码只在当前的 app 的 templates 文件夹中找）。各个 app 的 templates 形成一个文件夹列表，Django 遍历这个列表，一个个文件夹进行查找，当在某一个文件夹找到的时候就停止，所有的都遍历完了还找不到指定的模板的时候就是 Template Not Found。假设我们每个 app 的 templates 中都有一个 index.html，这样则会导致我们查找到不匹配的 index.html，解决方案是：每个 app 中的 templates 文件夹中再建一个 app 名称的文件夹，仅和该 app 相关的模板放在 app/templates/app/ 目录下。

七、 模板的常用规则(即 views.py 中函数如何使用模板文件)

1. 字符串(变量)使用

views.py 中：

```
1. # -*- coding: utf-8 -*-
2. from django.shortcuts import render
3.
4. def home(request):
5.     string = "我在学习 Django，用它来建网站"
6.     return render(request, 'home.html', {'string': string})
```

home.html 中：

```
1. {{ string }}
```

视图中我们传递了一个字符串名称是 string 到模板 home.html，home.html 按照上述方式(一般的变量之类的用 {{ }}（变量）)使用，显示一个基本的字符串在网页上。

2. for 循环 和 List 内容的显示

views.py 中：

```
1. # -*- coding: utf-8 -*-
2. from django.shortcuts import render
3.
4. def home(request):
5.     TutorialList = ["HTML", "CSS", "jQuery", "Python", "Django"]
6.     return render(request, 'home.html', {'TutorialList':
   TutorialList})
```

home.html 中：

```
1. {% for i in TutorialList %}
2. {{ i }}
```

```
3. {% endfor %}
```

循环，条件判断是用 {% %}（标签）

3. 显示字典的内容

views.py 中：

```
1. # -*- coding: utf-8 -*-
2. from django.shortcuts import render
3.
4. def home(request):
5.     info_dict = {'site': '自强学堂', 'content': '各种 IT 技术教程'}
6.     return render(request, 'home.html', {'info_dict': info_dict})
```

home.html 中：

```
1. 站点: {{ info_dict.site }} 内容: {{ info_dict.content }}
2. #  遍历字典
3. {% for key, value in info_dict.items %}
4.     {{ key }}: {{ value }}
5. {% endfor %}
```

4. 条件判断和 for 循环的详细操作

views.py 中：

```
1. # -*- coding: utf-8 -*-
2. from django.shortcuts import render
3.
4. def home(request):
5.     List = map(str, range(100))# 一个长度为 100 的 List
6.     return render(request, 'home.html', {'List': List})
```

home.html 中：

```
1. {% for item in List %}
2.     {{ item }}{% if not forloop.last %},{% endif %}
3. {% endfor %}
```

for 循环中的变量含义：

变量	描述

forloop.counter	索引从 1 开始算
forloop.counter0	索引从 0 开始算
forloop.revcounter	索引从最大长度到 1
forloop.revcounter0	索引从最大长度到 0
forloop.first	当遍历的元素为第一项时为真
forloop.last	当遍历的元素为最后一项时为真
forloop.parentloop	用在嵌套的 for 循环中，获取上一层 for 循环的 forloop

列表中可能为空值时用 for empty：

```
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% empty %}
    <li>抱歉，列表为空</li>
{% endfor %}
```

5. 模板中的逻辑操作

- ==, !=, >=, <=, >, < 这些比较都可以在模板中使用(比较符号前后必须有至少一个空格)

```
{% if var >= 90 %}
成绩优秀
{% elif var >= 80 %}
成绩良好
{% elif var >= 70 %}
成绩一般
{% elif var >= 60 %}
需要努力
{% else %}
  不及格
  {% endif %}
```

- and, or, not, in, not in 也可以在模板中使用

```
{% if num <= 100 and num >= 0 %}
num 在 0 到 100 之间
{% else %}
```

```
4. 数值不在范围之内!
5. {% endif %}
6. {% if 'ziqiangxuetang' in List %}
7. 在 list 中
8. {% endif %}
```

6. 模板中获取当前网址，当前用户等

- 获取当前用户

```
1. {{ request.user }}
```

- 获取当前网址

```
1. {{ request.path }}
```

- 获取当前 GET 参数

```
1. {{ request.GET.urlencode }}
```

八、 Django 模型(数据库)

1. 建立模型

```
1. from django.db import models
2.
3. class Person(models.Model):
4.     first_name = models.CharField(max_length=30)
5.     last_name = models.CharField(max_length=30)
```

first_name and last_name are fields of the model. Each field is specified as a class attribute, and each attribute maps to a database column

2. 使用模型

- changing the INSTALLED_APPS setting to add the name of the module that contains your models.py
- When you add new apps to INSTALLED_APPS, be sure to run manage.py migrate(Synchronizes the database state with the current set of models and migrations.), optionally making migrations for them first with manage.py makemigrations(Creates new migrations based on the changes detected to your models. Migrations, their relationship with apps and more are covered in depth in the migrations documentation.).

```
python manage.py makemigrations
python manage.py migrate
```

上述操作即创建数据表

九、 QuerySet API(对象创建(表的内容)、查询及其他接口)

1. QuerySet 创建对象的方法（创建表的内容）

```
# 方法 1
Author.objects.create(name="WeizhongTu", email="tuweizhong@163.com")

# 方法 2
twz = Author(name="WeizhongTu", email="tuweizhong@163.com")
twz.save()

# 方法 4，首先尝试获取，不存在就创建，可以防止重复
Author.objects.get_or_create(name="WeizhongTu",
email="tuweizhong@163.com")
    # 返回值(object, True/False)
```

2. 获取对象的方法

```
Person.objects.all() # 查询所有
Person.objects.all()[:10] 切片操作，获取 10 个人，不支持负索引，切片可以节约内存，不支持负索引，后面有相应解决办法，第 7 条
Person.objects.get(name="WeizhongTu") # 名称为 WeizhongTu 的一条，多条会报错
Person.objects.filter(name="abc") # 过滤，等于
Person.objects.filter(name__exact="abc") 名称严格等于 "abc" 的人
```

3. 删除符合条件的结果

```
Person.objects.filter(name__contains="abc").delete() # 删除 名称中包含 "abc"的人
```

4. 更新某个内容

- 批量更新，适用于 .all() .filter() .exclude() 等后面

```
1. Person.objects.filter(name__contains="abc").update(name='xxx') # 名称中包含 "abc"的人 都改成 xxx
2. Person.objects.all().delete() # 删除所有 Person 记录
```

- 单个 object 更新，适合于 .get(), get_or_create(), update_or_create() 等得到的 obj，和新建很类似

```
1. twz = Author.objects.get(name="WeizhongTu")
2. twz.name="WeizhongTu"
3. twz.email="tuweizhong@163.com"
4. twz.save()  # 最后不要忘了保存！！！
```

十、 QuerySet 进阶

十一、 数据表更改

```
1. python manage.py makemigrations
2. python manage.py migrate
```

十二、 Django 后台

每个 app 中的 admin.py 文件与后台相关

1. 创建管理员

```
1. python manage.py createsuperuser
```

1. __str__与__repr__方法

__repr__：repr() 函数将对象转化为供解释器读取的形式，返回一个对象的 string 格式

__str__：str() 函数将对象转化为适于人阅读的形式，返回一个对象的 string 格式，

两个方法均用于返回对象供人阅读，__str__()用于显示给用户，而__repr__()用于显示给开发人员。print 调用的是__str__方法，直接输出实例调用的是__repr__方法

```
>>> class Student(object):
... def __init__(self, name):
... self.name = name
...
>>> print(Student('Michael'))
<__main__.Student object at 0x109afb190>
>>> s = Student('Michael')
>>> s
  <__main__.Student object at 0x109afb310>
```

2. 查找模块下的方法(函数)、属性

均在 python 解释器下查询，详见附录 2.查询模块的帮助文档

3. Import 与 from…import…的区别

4. Tuple 与()的使用区别

tuple(seq)，seq -- 要转换为元组的序列。用法如下：

```
aList = [123, 'xyz', 'zara', 'abc'];
aTuple = tuple(aList)
print "Tuple elements : ", aTuple

>>>Tuple elements :  (123, 'xyz', 'zara', 'abc')
```

()则直接使用：

```
aTuple = (123, 'xyz', 'zara', 'abc')
```

5. Python 时间格式化输出

strftime()函数接收以时间元组(struct_time 对象)，并返回以可读字符串表示的当地时间，格式由参数 format 决定

```
t = time.time()//获得以秒为单位的时间
```

```
2. print(time.strftime("%b %d %Y %H:%M:%S", time.gmtime(t)))//gmtime 获得 struct_time 对象
```

格式参数 format，详见附录 3.python 中时间日期格式化符号

6. 格式化输出

格式输出详见参考文档 《python 的格式化输出》

7. Windows 环境下文件路径表示

Python 代码里面，反斜杠“\”是转义符，例如“\n”表示回车，采用以下三种方式表示路径：

- 斜杠 “/”，如“c:/test.txt”
- 两个反斜杠 “\\”，如“c:\\test.txt”
- 字符串前面加上字母 r，表示后面是一个原始字符串 raw string，如“r“c:\\test.txt””

8. Python 接收命令行参数

导入 argv，结果即为参数列表

```
1. from sys import argv
2. print(argv)
3. >>>python xx.py xxx
4. >>>['xx.py','xxx']
```

9. '\u'前缀字符串

\u4f60 十六进制代表对应汉字的 utf-16 编码

10.定义 1 个元素的 tuple

定义 1 个元素的 tuple: t = (1,)，加上一个逗号，避免成为数学意义上的括号

11.创建生成器(generator)的两种方式

方式 1: g = (x * x for x in range(10))，列表生成式的[]更改为()

方式 2: 如果一个函数定义中包含 yield 关键字，那么函数就不再是一个普通函数，而是一个 generator，函数是顺序执行，遇到 return 语句或者最后一行函数语句返回。而 generator 在每次调用 next()的时候执行，遇到 yield 语句返回，再次执行时从上次返回的 yield 语句处继续执行

```
1. def fib(max):
2.     n, a, b = 0, 0, 1
3.     while n < max:
```

```
4.          yield b
5.          a, b = b, a + b
6.          n = n + 1
7.      return 'done'
```

12.python 代码规范

请参考《python 代码规范》

13.迭代器总结

- 凡是可作用于 for 循环的对象都是 Iterable 类型；
- 凡是可作用于 next()函数的对象都是 Iterator 类型，它们表示一个惰性计算的序列；
- 集合数据类型如 list、dict、str 等是 Iterable 但不是 Iterator，不过可以通过 iter()函数获得一个 Iterator 对象。
- Python 的 for 循环本质上就是通过不断调用 next()函数实现的

14.Map 函数总结

map()函数接收两个参数，一个是函数，一个是 Iterable，map 将传入的函数依次作用到序列的每个元素，返回新的 Iterator。

```
map(function, iterable1,iterable2 ...)
```

例如三个列表相乘：

```
1. list1 = [1,2,3,4,5,6,7,8,9]
2. list2 = [1,2,3,4,5,6,7,8,9]
3. list3 = [9,8,7,6,5,4,3,2,1]
4. def foo(l1,l2,l3):
5.     return l1*l2*l3
6.
7. print(list(map(foo,list1,list2,list3)))
8. >>>[9,32,63,96,125,144,147,128,81]
```

15.Reduce 函数总结

reduce 把一个函数作用在一个序列[x1, x2, x3, ...]上，这个函数必须接收两个参数，reduce 把结果继续和序列的下一个元素做累积计算。

例如序列求和：

```
>>> from functools import reduce
>>> def add(x, y):
...     return x + y
...
>>> reduce(add, [1, 3, 5, 7, 9])
25
```

16.匿名函数(lambda)总结

lambda parameters: expression

parameters：可选，如果提供，通常是逗号分隔的变量表达式形式，即位置参数。

expression：**不能包含分支或循环（但允许条件表达式）**，也不能包含 return（或 yield）函数。如果为元组，则应用圆括号将其包含起来。调用 lambda 函数，**返回的结果是对表达式计算产生的结果**

```
#根据参数是否为 1 决定 s 为 yes 还是 no
>>> s = lambda x:"yes" if x==1 else "no"
```

17.全局变量的使用

使用到的全局变量只是作为引用，不在函数中修改它的值的话，不需要加 global 关键字

```
a = 1

def func():
    if a == 1:
        print("a: %d" %a)
```

使用到的全局变量，需要在函数中修改的话，就涉及到歧义问题，因此，需要修改全局变量 a，可以在"a = 2"之前加入 global a 声明

```
a = 1

def func():
    global a
    a = 2
    print("in func a:", a)
```

18.python 定义类时，内部方法的互相调用

每次调用内部的方法时，方法前面加 self

```
class MyClass:
    def __init__(self):
        pass
    def func1(self):
        print('a')
        self.common_func()
     def func2(self):
        self.common_func()

     def common_func(self):
          pass
```

19.filter 函数总结

filter()也接收一个函数和一个序列。filter()把传入的函数依次作用于每个元素，然后根据返回值是 True 还是 False 决定保留还是丢弃该元素

20.sorted 函数总结

sorted()函数也是一个高阶函数，它还可以接收一个 key 函数来实现自定义的排序，key 指定的函数将作用于 list 的每一个元素上，并根据 key 函数返回的结果进行排序

```
>>> sorted([36, 5, -12, 9, -21], key=abs)
[5, 9, -12, -21, 36]
```

21.返回函数(闭包)

22.

23.

24.装饰器

这种在代码运行期间动态增加功能的方式，称之为“装饰器”（Decorator）。本质上，decorator 就是一个返回函数的高阶函数。

```
def log(func):
    def wrapper(*args, **kw):
        print('call %s():' % func.__name__)
        return func(*args, **kw)
    return wrapper
```

如果将上述的 log 函数作为装饰器，则在其装饰的函数前添加@log

```
@log
def now():
    print('2015-3-25')
>>> now()
call now():
2015-3-25
```

装饰器原理及引入机制较为复杂，详情可参考《python 装饰器解释》

25.函数参数

- 必选参数(位置参数):
- 默认参数：如 def power(x, n=2)，x 即为位置参数，n 为默认参数，必选参数在前，默认参数在后
- 可变参数：传入的参数个数是可变的，不必自行将所有参数组装成一个 list 或 tuple，如 def calc(*numbers)。定义可变参数和定义一个 list 或 tuple 参数相比，仅仅在参数前面加了一个*号。在函数内部，参数 numbers 接收到的是一个 tuple

```
def calc(*numbers):
    sum = 0
    for n in numbers:
        sum = sum + n * n
    return sum
```

- 关键字参数:可变参数在函数调用时自动组装为一个 tuple。而关键字参数允许你传入 0 个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个 dict:

```
def person(name, age, **kw):
    print('name:', name, 'age:', age, 'other:', kw)

>>> person('Michael', 30)
name: Michael age: 30 other: {}
>>> person('Bob', 35, city='Beijing')
name: Bob age: 35 other: {'city': 'Beijing'}
```

26.字符串里面的引号

单引号''定义字符串的时候，它就会认为你字符串里面的双引号" "是普通字符，从而不需要转义。反之当你用双引号定义字符串的时候，就会认为你字符串里面的单引号是普通字符无需转义

```
Str1 = "We all know that 'A' and 'B' are two capital letters."
```

```
2. Str2 = 'The teacher said: "Practice makes perfect" is a very famous
   proverb.'
```

27.偏函数

functools.partial 的作用就是，把一个函数的某些参数给固定住（也就是设置默认值），返回一个新的函数，调用这个新函数会更简单

```
1. >>> import functools
2. >>> int2 = functools.partial(int, base=2)
3. >>> int2('1000000')
4. 64
```

28.算法速度的定义

- 大 O：T(N) = O(f(N))，T 增长率小于等于 f
- Ω：T(N) = Ω (f(N))，T 增长率大于 f
- Θ：T(N) = Θ (f(N))，T 增长率等于 f
- 小 o：T(N) =o (f(N))，T 增长率小于 f

29.计算运行时间的一般法则

- 法则 1：for 循环：一个 for 循环的运行时间至多是该循环内语句的运行时间乘以迭代次数
- 法则 2：嵌套 for 循环：嵌套循环内部的一条语句总运行时间为该语句运行时间乘以所有 for 循环的大小，如下程序片段运行时间为 $O(N^2)$

```
1. for i in range(N):
2.     for j in range(N):
3.         j += 1
```

- 法则 3：顺序语句：各个语句运行时间求和
- 法则 4：if/else 语句：运行时间至多是判断时间+S1 或 S2 中运行时间长者

```
1. if (condition):
2.     S1
3. else:
4.     S2
```

30.导入自定义模块

31.子类继承父类的属性问题

更加细致的继承中的属性和方法问题，请参考《python 类的继承、属性总结和方法总结》

如果子类自己定义了__init__方法，那么父类的属性是不能调用的，如下：

```
1. class Animal:
2.     def __init__(self):
3.         self.a = 'aaa'
4.
5. class Cat(Animal):
6.     def __init__(self):
7.         pass
8.
9. cat = Cat()
10.   print(cat.a)
11.
12.   >>>AttributeError: 'Cat' object has no attribute 'a'
```

可以在子类的 __init__中调用一下父类的 __init__ 方法,这样就可以调用父类的属性

```
1. class Animal:
2.     def __init__(self):
3.         self.a = 'aaa'
4.
5. class Cat(Animal):
6.     def __init__(self):
7.         super().__init__()
8.
9. cat = Cat()
10.   print(cat.a)
11.
12.   >>>aaa
```

32.子类扩展父类属性

33.附录

1. 方法解析顺序（Method Resolution Order, MRO）列表

MRO 列表的顺序遵循以下三条原则：

- 子类永远在父类前面
- 如果有多个父类，会根据它们在列表中的顺序被检查
- 如果对下一个类存在两个合法的选择，选择第一个父类

2. 查询模块的帮助文档

- 先导入模块，再查询普通模块的使用方法：help(module_name)，例：help(math)
- 先导入 sys，再查询系统内置模块的使用方法：sys.bultin_modulenames
- 查看模块下所有函数：dir(module_name)，例：dir(math)
- 查看模块下特定函数：help(module_name.func_name)，例：help(math.sin)
- 查看函数信息的另一种方法：print(func_name.__doc__)，例：print(sin.__doc__)

3. python 中时间日期格式化符号

- %y 两位数的年份表示（00-99）
- %Y 四位数的年份表示（000-9999）
- %m 月份（01-12）
- %d 月内中的一天（0-31）
- %H 24 小时制小时数（0-23）
- %I 12 小时制小时数（01-12）

- %M 分钟数（00=59）
- %S 秒（00-59）
- %a 本地简化星期名称
- %A 本地完整星期名称
- %b 本地简化的月份名称
- %B 本地完整的月份名称
- %c 本地相应的日期表示和时间表示
- %j 年内的一天（001-366）
- %p 本地 A.M.或 P.M.的等价符
- %U 一年中的星期数（00-53）星期天为星期的开始
- %w 星期（0-6），星期天为星期的开始
- %W 一年中的星期数（00-53）星期一为星期的开始
- %x 本地相应的日期表示
- %X 本地相应的时间表示
- %Z 当前时区的名称
- %% %号本身