1+x Python程序开发（初级）

一、 证书相关知识点

1.1 考试级别

Python开发职业技能分为初、中、高三个等级

1.2 考核方式与题型

考核方式为闭卷考试，采用上机考试形式。考试包括理论考试和实操考试两部分。理论考试时长90分钟，试卷满分100分，共50道试题，其中单选题30道，多选题10道，判断题10道；实操考试时长150分钟，试卷满分100分，试卷含5道实践性试题，试题形式包括案例分析、软件代码编码或是网页效果呈现等。

1.3 合格标准

理论考试试卷满分为100分，权重40%;实操考试试卷满分为100分，权重60%。综合成绩等于理论和实操考试成绩的加权之和，综合成绩合格标准为大于等于60分，综合成绩合格的学员可以获得相应级别的职业技能等级证书。

1.4 考试计算机环境配置

1.4.1 硬件

CPU：四核以上，主频2. 0GHz以上

内存：4GB

硬盘：50G空闲

网卡：1000M

1.4.2 软件

操作系统：Windows10

开发工具：PyCharm2019 以上

输入法要求：安装英文、拼音、五笔输入。

浏览器要求：谷歌稳定版75.0以上版本

1.5 考试内容、题型及分值

（1）理论：（单选题（60分））、（多选题10道（20分））、（判断题（20分））

（2）实操：程序填空（5个项目）

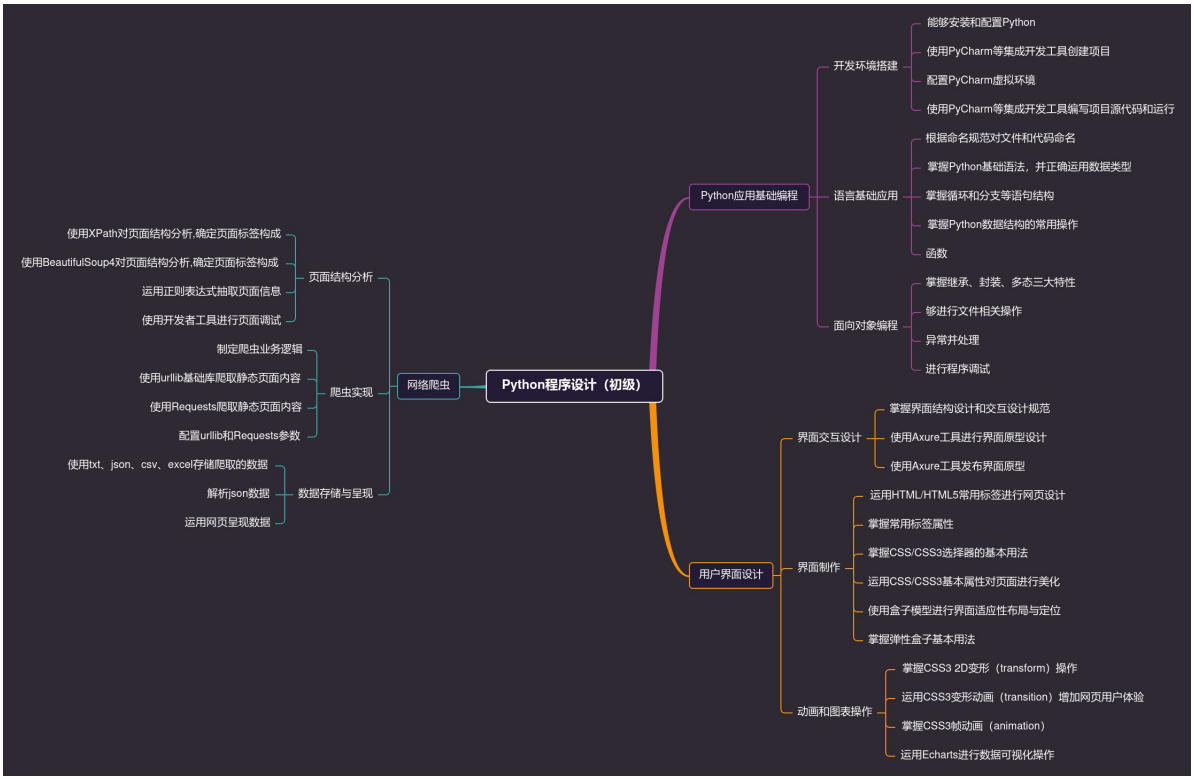
1.6 考试日期安排

初级：2022年12月17号 上午9:00~10:30， 下午14:00~16:30

中级：2022年12月17日 上午9:00~10:30， 下午14:00~16:30

高级：2022年12月17日 上午9:00~10:30， 下午14:00~16:30

1.7 初级证书涵盖的主要知识点



二、Python基础

2.1. Python的发展史

2.1.1 起源

1991年，第一个Python编译器诞生。它是用C语言实现的，并能够调用C语言的库文件。Python将许多机器层面上的细节隐藏，交给编译器处理，并凸显出逻辑层面的编程思考。Python程 序员可以花更多的时间用于思考程序的逻辑，而不是具体的实现细节。这一特征吸引了广大的程序员。Python开始流行。

Python崇尚优美、清晰、简单，是一个优秀并广泛使用的语言

2.1.2 Python常识

- Python的解释器如今有多个语言实现，我们常用的是CPython（官方版本的C语言实现），其他还有Jython（可以运行在Java平台）、IronPython（可以运行在.NET和Mono平台）、PyPy（Python实现的，支持JIT即时编译）
- Python目前有两个版本，Python2和Python3，最新版分别为2.7.18和3.10.0，Python2在2020年官方已停止维护，现阶段大部分公司用的是Python3，无需再学习Python2
- Life is shot, you need Python. 人生苦短，我用Python
- 2021年8月份，编程语言流行排行榜

Nov 2021	Nov 2020	Change	Programming Language	Ratings	Change
1	2	↑	Python	11.77%	-0.35%
2	1	↓	C	10.72%	-5.49%
3	3		Java	10.72%	-0.96%
4	4		C++	8.28%	+0.69%
5	5		C#	6.06%	+1.39%
6	6		Visual Basic	5.72%	+1.72%
7	7		JavaScript	2.66%	+0.63%
8	16	⇈	Assembly language	2.52%	+1.35%
9	10	↑	SQL	2.11%	+0.58%
10	8	↓	PHP	1.81%	+0.02%

2.2 第一个Python程序

2.2.1 hello world

1. 打开终端，进入Python交互环境，输入以下代码

```
print("hello word")
```

2. 使用编辑器将代码保存为hello.py，然后使用以下命令运行

```
python hello.py
```

2.3 注释

2.3.1 单行注释

```
# 我是注释，可以用来写代码的功能描述
print("hello world")
```

2.3.2 多行注释

```
"""
    我是多行注释，可以在这里写很多行的代码说明
"""
print("hello world")
```

2.4 变量及类型

2.4.1 变量的定义

```
a = 100
x, y = 10, 20
s = "hello world"
```

说明：变量的名字只能由数字、字母、下划线组成，并且数字不能在开头（中文可以，但不要使用）变量名推荐使用单词全小写，单词和单词之间使用下划线进行连接

可以使用 `id` 函数查看变量的内存地址

```
x = 100
print(id(x))
```

2.4.2 变量的类型

可以使用Type函数来查看变量的类型

2.5 输入和输出

2.5.1 输入

```
name = input("请输入你的姓名：")
```

2.5.2 输出

```
name = "lisa"
print(name)
```

2.6 运算符

2.6.1算数运算符

运算符	含义	描述
+	加	两个对象相加
-	减	得到负数或是一个数减去另一个数
*	乘	两个数相乘或是返回一个被重复若干次的字符串
/	除	两数相除
//	取整除	返回商的整数部分。如 9//2 输出结果 4 , 9.0//2.0 输出结果 4.0
%	取余	返回除法的余数
**	幂	幂运算。x**y 返回x的y次幂

运算符的优先级

```
>>> 3 + 5 * 2 ** 3 - 6 / 3 + 7 % 2
>>> 42.0
```

2.6.2 赋值运算符

运算符	描述	实例
=	赋值运算符	把=号右边的结果给左边的变量 num=1+2*3 结果num的值为7

2.6.3 复合赋值运算符

运算符	描述	实例
+=	加法赋值运算符	c += a 等效于 c = c + a
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
%=	取模赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	c **= a 等效于 c = c ** a
//=	取整除赋值运算符	c //= a 等效于 c = c // a

2.6.4 比较运算符

运算符	描述	示例
==	检查两个操作数的值是否相等，如果是则条件变为真。	如a=3,b=3则（a == b) 为 true.
!=	检查两个操作数的值是否相等，如果值不相等，则条件变为真。	如a=1,b=3则(a != b) 为 true.
>	检查左操作数的值是否大于右操作数的值，如果是，则条件成立。	如a=7,b=3则(a > b) 为 true.
<	检查左操作数的值是否小于右操作数的值，如果是，则条件成立。	如a=7,b=3则(a < b) 为 false.
>=	检查左操作数的值是否大于或等于右操作数的值，如果是，则条件成立。	如a=3,b=3则(a >= b) 为 true.
<=	检查左操作数的值是否小于或等于右操作数的值，如果是，则条件成立。	如a=3,b=3则(a <= b) 为 true.

2.6.5 逻辑运算符

运算符	逻辑表达式	描述	实例
and	x and y	布尔"与" - 如果 x 为 False，x and y 返回 False，否则它返回 y 的计算值。	(a and b) 返回 20。
or	x or y	布尔"或" - 如果 x 是 True，它返回 True，否则它返回 y 的计算值。	(a or b) 返回 10。
not	not x	布尔"非" - 如果 x 为 True，返回 False 。如果 x 为 False，它返回 True。	not(a and b) 返回 False

2.7 数据类型转换

2.7.1 常用的数据类型转换函数

函数	说明
int(x [,base])	将x转换为一个整数
float(x)	将x转换到一个浮点数
complex(real [,imag])	创建一个复数
str(x)	将对象 x 转换为字符串
repr(x)	将对象 x 转换为表达式字符串
eval(str)	用来计算在字符串中的有效Python表达式,并返回一个对象
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
chr(x)	将一个整数转换为一个字符
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

2.8 条件判断语句

2.8.1 条件语句

```
if 条件语句1:
    条件1成立执行的代码块
elif 条件语句2:
    条件2成立执行的代码块
else:
    条件不成立执行的代码块
```

条件表达式可以是一个返回布尔值的表达式，也可以是任何可以转换为bool值的数据类型

```
if 20:
    print("hello world")

>>> hello world
```

注：条件语句可以嵌套

2.9 循环语句

2.9.1 while循环

```
while True:
    pass
else:
    print("while循环正常结束执行的代码")
```

示例：求1~100的整数和

```
i = 1
sum = 0
while i<=100:
    sum = sum + i
    i += 1

print("1~100的累积和为:%d"%sum)
```

自己动手：计算0~100之间的偶数和

注：while循环可以嵌套使用

2.9.2 for循环

```
for i in range(100):
    pass
else:
    print("for循环正常结束执行的代码")
```

在Python中 for循环可以遍历任何序列，如一个列表或者一个字符串等。

```
for i in range(100):
    print(1)

for s in "lisa":
    print(s)

for lang in ["Python", "Ruby", "Go", "Rust"]:
    print(lang)
```

2.9.3 break和continue

1. break 结束整个循环
2. continue：结束本次循环，开始下一次循环

注意：

1. break和continue只能用在循环结构中，不能单独使用
2. break/continue在嵌套循环中，只对最近的一层循环起作用

三、Python数据类型

3.1 字符串

3.1.1 字符串的定义

双引号或者单引号中的数据，就是字符串

```
str1 = 'lisa'
str2 = "My name is 'Lisa'"
str3 = '''I'm Lucy'''
str4 = "中国"
```

3.1.2 字符串的输出

```
print("Python程序设计")
```

3.1.3 字符串的输入

```
ret = input("请输入：")
print(type(ret))
```

3.1.4 下标和切片

1. 下标

   所谓“下标”，就是编号，我们可以通过下标来获取字符串中的某个字符

   ```
   name = 'abcdef'

   print(name[0])
   print(name[1])
   print(name[2])
   ```

2. 切片

   切片是指对操作的对象截取其中一部分的操作。**字符串、列表、元组**都支持切片操作。

   切片的语法：**[起始:结束:步长]**

   注意：选取的区间属于左闭右开型，即从"起始"位开始，到"结束"位的前一位结束（不包含结束位本身）。

   ```
   name = 'abcdef'

   print(name[0:3]) # 取下标0~2的字符

   print(name[3:5]) # 取下标为3、4的字符

   print(name[2:]) # 取下标为2开始到最后的字符

   print(name[1:-1]) # 取下标为1开始到最后第2个之间的字符
   ```

   自己动手：给一个字符串string，请反转字符串

3.1.5 字符串常用方法

1. find

   检测 str 是否包含在 mystr中，如果是返回开始的索引值，否则返回-1

   ```
   mystr.find(str, start=0, end=len(mystr))
   ```

2. index

   跟find()方法一样，只不过如果str不在 mystr中会报一个异常

   ```
   mystr.index(str, start=0, end=len(mystr))
   ```

3. count

   返回 str在start和end之间 在 mystr里面出现的次数

   ```
   mystr.count(str, start=0, end=len(mystr))
   ```

4. replace

   把 mystr 中的 str1 替换成 str2,如果 count 指定，则替换不超过 count 次

   ```
   mystr.replace(str1, str2,  mystr.count(str1))
   ```

5. split

   以 str 为分隔符切片 mystr，如果 maxsplit有指定值，则仅分隔 maxsplit 次

   ```
   mystr.split(str=" ", 2)
   ```

6. lower

   转换 mystr 中所有大写字符为小写

   ```
   mystr.lower()
   ```

7. upper

   转换 mystr 中的小写字母为大写

   ```
   mystr.upper()
   ```

8. center

   返回一个原字符串居中,并使用空格填充至长度 width 的新字符串

   ```
   mystr.center(width)
   ```

9. strip

   去除字符左右两边的空白部分

   ```
   mystr.strip()
   ```

10. isalpha

    如果 mystr 所有字符都是字母 则返回 True,否则返回 False

```
mystr.isalpha()
```

11. isdigit

    如果 mystr 只包含数字则返回 True 否则返回 False

```
mystr.isdigit()
```

12. isalnum

    如果 mystr 所有字符都是字母或数字则返回 True,否则返回 False

```
mystr.isalnum()
```

13. join

    mystr 中每个字符后面插入 `|` ,构造出一个新的字符串

```
"|".join(mystr)
```

3.1.6 字符串格式化

1. %格式化

转换说明符	解释
%d、%i	转换为带符号的十进制整数
%o	转换为带符号的八进制整数
%x、%X	转换为带符号的十六进制整数
%e	转化为科学计数法表示的浮点数（e 小写）
%E	转化为科学计数法表示的浮点数（E 大写）
%f、%F	转化为十进制浮点数
%g	智能选择使用 %f 或 %e 格式
%G	智能选择使用 %F 或 %E 格式
%c	格式化字符及其 ASCII 码
%r	使用 repr() 函数将表达式转换为字符串
%s	使用 str() 函数将表达式转换为字符串

（1）整数的输出

%o：oct 八进制

%d：dec 十进制

%x：hex 十六进制

```
>>>print('%d' % 20)
```

（2）浮点数输出

%f：保留小数点后面六位有效数字。

如：%.3f，保留3位小数位

%e：保留小数点后面六位有效数字，指数形式输出

如：%.3e，保留3位小数位，使用科学计数法。

%g：在保证六位有效数字的前提下，使用小数方式，否则使用科学计数法

如：%.3g，保留3位有效数字，使用小数或科学计数法。

```
In [3]: '%.2f' % 314.9256
Out[3]: '314.93'
```

（3）字符串输出

%10s：右对齐，占位符10位

%-10s：左对齐，占位符10位

%.2s：截取2位字符串

%10.2s：10位占位符，截取两位字符串

```
>>> print('%s' % 'hello world')  # 字符串输出
>>> hello world

>>> print('%20s' % 'hello world')  # 右对齐，取20位，不够则补位
>>> hello world

>>> print('%-20s' % 'hello world')  # 左对齐，取20位，不够则补位
>>> hello world

>>> print('%.2s' % 'hello world')  # 取2位
>>> he
```

2.format 格式化

相对基本格式化输出采用‘%’的方法，format()功能更强大，该函数把字符串当成一个模板，通过传入的参数进行格式化，并且使用大括号‘{}’作为特殊字符代替‘%’，在开发过程中推荐使用此方式进行字符串格式化。

```
>>> print('{} {}'.format('hello','world'))  # 不带字段

>>> print('{0} {1}'.format('hello','world'))  # 带数字编号

>>>print('{0} {1} {0}'.format('hello','world'))  # 打乱顺序

>>>print('{a} {tom} {a}'.format(tom = 'hello',a = 'world'))  # 带关键字
```

3.f-Strings 格式化

f-strings是Python3.6开始加入标准库的格式化输出新的写法，这个格式化输出比之前的%s或者format效率高并且更加简化，更加好用。

```
name = '小王'
age = 18
sex = '男'
msg = F'姓名：{name},性别：{age}，年龄：{sex}'  # F或f都可以
print(msg)
```

3.2 列表

3.2.1 定义列表和遍历列表

1. 列表的定义

   ```
   # 定义含有元素的列表
   lang_list = ["Python", "Ruby", "Go", "Rust"]
   # 定义空列表
   my_list = []
   # 列表只作为元素
   my_list2 = [3.5, [1, 2], 'a']
   # 元素可以是不同数据类型
   my_list3 = [1, 2, 'red']
   # 使用list函数创建，注意这种方式参数必须是可迭代类型
   my_list = list("abcd")
   ```

   可以使用 `range` 函数快速生成列表

   ```
   list(range(1, 10, 3))
   ```

2. 使用for循环遍历列表

   ```
   for lang in lang_list:
       print(lang)
   ```

3. 使用while循环遍历

   ```
   i = 0
   while i < len(lang_list):
       print(lang_list[i])
       i += 1
   ```

3.2.2 列表的常用方法

1. `append`

   可以向列表中添加元素

   ```
   lang_list.append("Kotlin")
   ```

2. `extend`

   通过列表扩展列表

```
lang_list.extend(["Scala", "Java"])
```

3. `insert`

   指定位置插入元素

```
lang_list.insert(1, "C#")
```

4. 修改元素

   修改元素的时候，要通过下标来确定要修改的是哪个元素，然后才能进行修改

```
lang_list[-2] = "JavaScript"
```

5. 查找元素

   <1> in和not in：查看元素是否存在列表中

```
"Ruby" in lang_list
"Go" not in lang_list
```

   <2> index：查找元素在列表中的位置

```
lang_list.index("Ruby")
```

   <3> count：统计元素在列表中出现的次数

```
[1, 2, 3, 2, 4].count(2)
```

   我们可以使用 `in` 来测试一个对象是否是可迭代的对象

6. 删除元素

   <1> del：根据下标进行删除

```
del lang_list[-2]
```

   <2> pop：弹出最后一个元素

```
lang_list.pop()
```

   <3>remove: 根据元素的值删除

```
lang_list.remove("C#")
```

7. 元素排序（sort、reverse方法和reversed函数）

   `sort` 方法是将list按特定顺序重新排列，默认为由小到大，参数reverse=True可改为倒序，由大到小。

   `reverse` 方法是将list逆置。

   `reversed` 函数用来排序

```
>>> a = [1, 4, 2, 3]
>>> a
[1, 4, 2, 3]
```

```
>>> a.reverse()
>>> a
[3, 2, 4, 1]

>>> a.sort()
>>> a
[1, 2, 3, 4]

>>> a.sort(reverse=True)
>>> a
[4, 3, 2, 1]

>>> list(reversed([2, 4, 6]))
>>> [6, 4, 2]
```

8. `count`：统计元素在列表中出现的次数

```
>>> [1, 2, 3].count(4)

>>> 0
```

注：列表可进行嵌套使用

3.3 元组

3.3.1 元组的定义

Python的元组与列表类似，不同之处在于**元组的元素不能修改**。元组通常使用小括号，列表使用方括号。

```
atuple = ('et',77,99.9)

btuple = 1, 2, 3

ctuple = (1,)
```

注意： a = (1)， 此时a并不是一个元组

3.3.2 元组常用操作

1. 元素访问

```
t = 1, 2, 3
t[0]
```

2. count：统计元素出现的次数

```
t = (2, 1, 2, 3)
t.count(2)
```

3. index：查找元素

```
t = (2, 1, 2, 3)
t.index(2, 1, -1)
```

3.3.3 元组遍历

```
lang_list = ("Python", "Ruby", "Go", "Rust")
for lang in lang_list:
    print(lang)
```

3.4 字典

3.4.1 字典的定义

字典是使用键值对存储数据的一种数据类型

```
user = {"nickname": "@lisa", "age": 20, "gender": "female", "tel": 18888888888}
```

3.4.2 字典的常见操作

1. **访问元素值**

   根据键访问值，在我们不确定字典中是否存在某个键而又想获取其值时，可以使用`get`方法，还可以设置默认值

   ```
   user = {"nickname": "@lisa", "age": 20, "gender": "female", "tel":
   18888888888}
   print(user["age"])
   print(user.get("addr", "成都"))
   ```

2. **修改元素**

   字典的每个元素中的数据是可以修改的，只要通过key找到，即可修改

   ```
   user = {"nickname": "@lisa", "age": 20, "gender": "female", "tel":
   18888888888}
   user["age"] = 18
   print(user)
   ```

3. **添加元素**

   如果在使用 **变量名['键'] = 数据** 时，这个“键”在字典中，不存在，那么就会新增这个元素

4. **删除元素**

   <1> del：删除键值对

   ```
   user = {"nickname": "@lisa", "age": 20, "gender": "female", "tel":
   18888888888}
   del user["tel"]
   print(user)
   ```

   <2> clear：清空元素

```
user = {"nickname": "@lisa", "age": 20, "gender": "female", "tel":
18888888888}
user.clear()
print(user)
```

3.4.3 字典常用函数和方法

1. `len` 函数：测量字典中，键值对的个数

```
user = {"nickname": "@lisa", "age": 20, "gender": "female", "tel":
18888888888}
print(len(user))
```

2. `keys` 方法：获取字典中所有的键

```
user = {"nickname": "@lisa", "age": 20, "gender": "female", "tel":
18888888888}
print(user.keys())
```

3. `values` 方法：获取字典中所有的值

```
user = {"nickname": "@lisa", "age": 20, "gender": "female", "tel":
18888888888}
print(user.values())
```

4. `items` 方法：获取字典中所有的键值对

```
user = {"nickname": "@lisa", "age": 20, "gender": "female", "tel":
18888888888}
print(user.items())
```

3.4.4 字典的遍历

1. 字典和字符串、列表、元组一样都可以进行遍历

```
user = {"nickname": "@lisa", "age": 20, "gender": "female", "tel":
18888888888}
for key, value in user.items():
    print(key, value)
```

3.5 集合

3.5.1 集合的创建

集合是一种无序不重复的序列

```
lang_set = {"Python", "Ruby", "Go", "Rust", "Rust"}
print(lang_set)
{"Python", "Ruby", "Go", "Rust"}
```

3.5.2 集合常用方法

1. add：给集合添加元素

```
my_set = {1, 2}
my_set.add(3)
```

2. update：把集合 y 中的项目插入集合 x：

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.update(y)

print(x)
```

3. remove：移除元素

```
>>> thisset = set(("Google", "Runoob", "Taobao"))
>>> thisset.remove("Taobao")
>>> print(thisset)
{'Google', 'Runoob'}
```

4. 差集 `-`

```
>>> s1={1, 2, 3, 4}
>>> s2 = {2, 3, 4, 5}
>>> s1 - s2
>>> {1}
```

5. 对称差集 `^`

```
>>> s1={1, 2, 3, 4}
>>> s2 = {2, 3, 4, 5}
>>> s1 ^ s2
>>> {1, 5}
```

6. 交集 `&`

```
>>> s1={1, 2, 3, 4}
>>> s2 = {2, 3, 4, 5}
>>> s1 & s2
>>> {2, 3, 4}
```

7. 并集 `|`

```
>>> s1={1, 2, 3, 4}
>>> s2 = {2, 3, 4, 5}
>>> s1 | s2
>>> {1, 2, 3, 4, 5}
```

注：集合也可以使用循环进行遍历

3.6 公共方法

3.6.1 运算符

运算符	Python 表达式	结果	描述	支持的数据类型
+	[1, 2] + [3, 4]	[1, 2, 3, 4]	合并	字符串、列表、元组
*	'Hi!' * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	复制	字符串、列表、元组
in	3 in (1, 2, 3)	True	元素是否存在	字符串、列表、元组、字典
not in	4 not in (1, 2, 3)	True	元素是否不存在	字符串、列表、元组、字典

3.6.2 内置函数

序号	方法	描述
2	len(item)	计算容器中元素个数
3	max(item)	返回容器中元素最大值
4	min(item)	返回容器中元素最小值
5	del(item)	删除变量

四、函数

4.1 函数的定义和调用

如果在开发程序时，需要某块代码多次，但是为了提高编写的效率以及代码的重用，所以把具有独立功能的代码块组织为一个小模块，这就是函数

4.1.1 函数的定义

定义函数的格式如下：

```
def 函数名(参数列表):
        代码
```

示例:

```python
# 定义一个函数，能够完成打印信息的功能
def show():
    print '-------------------------------------'
    print '          人生苦短，我用Python'
    print '-------------------------------------'
```

4.1.2 函数的调用

定义了函数之后，就相当于有了一个具有某些功能的代码，想要让这些代码能够执行，需要调用它

调用函数很简单的，通过 **函数名()** 即可完成调用

```python
def fx(x, y=2, *args, **kwargs):
    return x, y, args, kwargs


result = fx(1, 10, 3, 4, 5, name="lisa", age=18)
print(result)
```

注： 函数参数是可选的，函数可以没有参数，函数可以没有返回值，`return`除了用来返回结果，还可以用来结束函数

4.2 变量的作用域

1. 局部变量

   局部变量，就是在函数内部定义的变量

   不同的函数，可以定义相同的名字的局部变量，但是各用个的不会产生影响

   局部变量的作用，为了临时保存数据需要在函数中定义变量来进行存储，这就是它的作用

2. 全局变量

   在函数外边定义的变量叫做`全局变量`

   全局变量能够在所有的函数中进行访问

   如果在函数中修改全局变量，那么就需要使用`global`进行声明，否则出错

   如果全局变量的名字和局部变量的名字相同，那么使用的是局部变量

4.2 递归函数

函数自己调用自己就是递归函数

```python
def fact(x):
    if x == 1:
        return 1
    return x * fact(x - 1)


ret = fact(5)
print(ret)
```

4.3 匿名函数

用lambda关键词能创建小型匿名函数。这种函数得名于省略了用def声明函数的标准步骤。

lambda函数的语法只包含一个语句，如下：

```
ret = (lambda x, y: x+y)(2, 3)
print(ret)
```

五、 Python面向对象

5.1 面向对象的概念

什么是对象?

对象是拥有具体属性值(名词)和行为（动词）的实体，比如在现实生活中的实际存在的一辆车，一个人，一个苹果等。对象，又称实例，在Python中，一切皆对象，比如我们之前使用的整数、字符串、列表、字典等。

什么是面向对象?

面向对象就是在程序中，把一切都看做对象，基于这些对象来编程。当你要做事情的时候，你不会亲自去做，而是调用些对象去做。

什么是类?

类可以理解为对象的模板，其中定义了同类对象应该具有的属性和方法，类可以用来创建一个或多个同类对象。类是一个抽象的概念，而对象是类具象化的结果。类比于现实世界中的事务，比如汽车就相当于是一个类，提到它我们知道它应具有行驶的功能，而车型、重量、最大行驶速度、车龄等等是汽车应该具有的属性。而一辆具体的汽车是一个对象，在这个对象中，车型、重量等属性有了具体的值。

类的创建语法：

```
class 类名(继承列表):
    语句块
```

5.2 实例方法

语法:

```
class 类名(继承列表):

    def 实例方法名(self, 参数1, 参数2, ...):

        '''实例方法的文档字符串'''

        语句块
```

作用:

用于描述一个对象的行为,让此类型的全部对象都拥有相同的行为

说明:

实例方法实质是函数,是定义在类内的函数 。实例方法至少有一个形参,第一个形参代表调用这个方法的实例,一般命名为'self'

调用:

实例.实例方法名(调用传参)

示例:

创建一个Car类，定义实例方法run

5.3 实例属性

每个实例都可以有自己的变量,此变量称为实例变量(也叫实例属性)

实例属性的使用

实例.实例属性名

实例属性的赋值

可以通过"="为实例属性赋值

实例.实例属性名 = 值

5.4 初始化方法

作用：对新创建的对象进行初始化。

语法:

```
class 类名:

    def __init__(self [,形参列表]):
        语句块
```

示例:

为刚才创建的Car类添加初始化方法，添加品牌和颜色属性

5.5 保护对象

如果有一个对象，当需要对其属性进行修改属性时，可以通过以下方法来实现：

（1）对象名.属性名 = 数据 ---->直接修改

（2）对象名.方法名() ---->间接修改

作用

为了更好的保存属性安全，即不能随意修改。

为了更好的保存属性安全，即不能随意修改，一般的处理方式为

（1）将属性定义为私有属性

（2）添加一个可以调用的方法，供调用

示例

```
class Person:
    def __init__(self, name=None, age=0):
        self.name = name
        self.__age = age

    def set_age(self, new_age):
        if 0 < new_age < 100:
            self.__age = new_age
        else:
            self.__age = 0

    def get_age(self):
        return self.__age


# 实例化对象
p = Person()
# 进行保护性赋值
p.set_age(18)
print(p.get_age())
```

5.6 析构方法

作用：当一个对象被Python解释器进行垃圾回收时会调用此方法

语法：

```
class 类名:

    def __del__(self [,形参列表]):
        语句块
```

示例：

```
class Animal(object):
```

```
    # 初始化方法
    # 创建完对象后会自动被调用
    def __init__(self, name):
        print('__init__方法被调用')
        self.name = name

    # 析构方法
    # 当对象被删除时，会自动被调用
    def __del__(self):
        print("__del__方法被调用")
        print(f"{self.name}对象被删除了...")


# 创建对象
dog = Animal("金毛")
# 删除对象
del dog
```

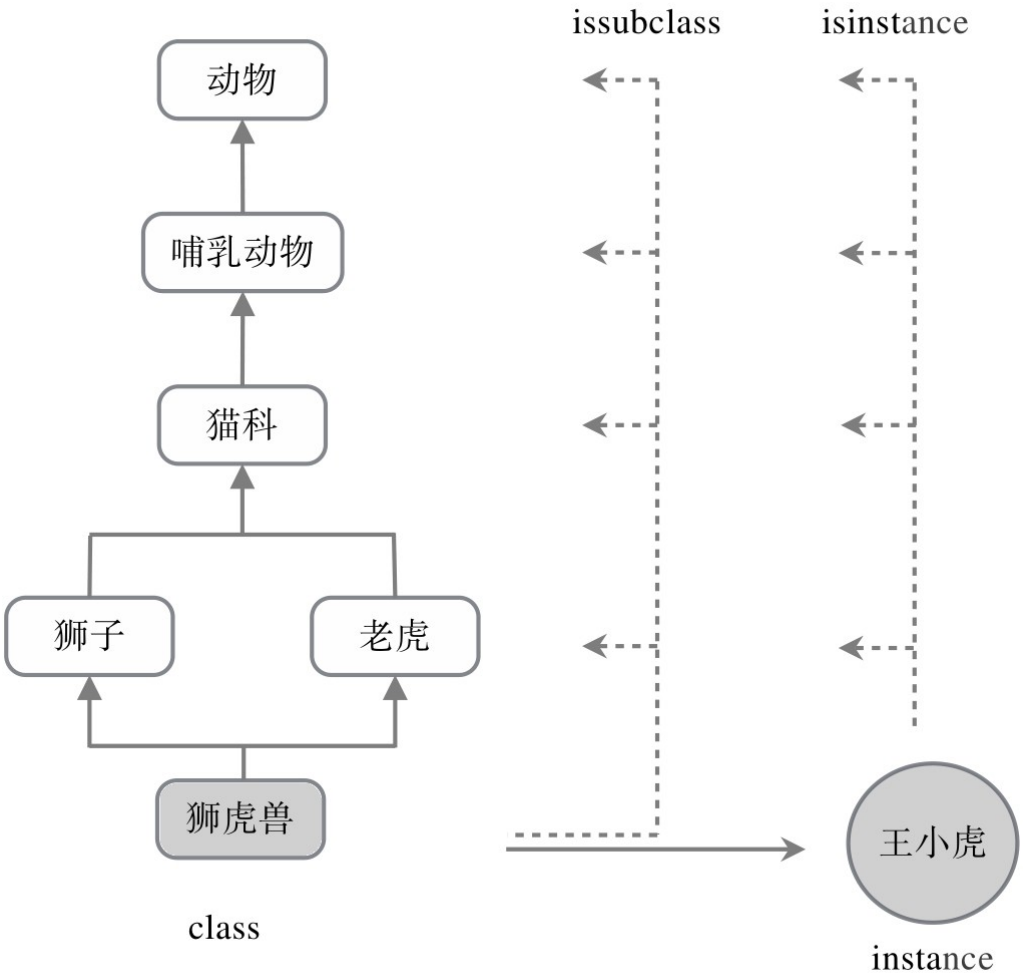
说明：

当对象的引用计数为0是对象会被回收，可以通过sys.getrefcount()来获取对象的引用计数。

5.7 继承

类型间可构成继承关系。就像老虎继承自猫科，而猫科又继承自哺乳动物，往上还有更顶层的类型。继承关系让类型拥有其所有祖先类型的特征。因历史原因，Python允许多继承，也就是说可有多个父类型，好似人类同时拥有父族、母族遗传特征。

继承关系图：



语法:

```
class 类名(继承列表):
    pass
```

示例:

```
class Animal:
    def __init__(self, name, age):
        self.name = name
        self.__age = age

    def introduce(self):
        print(f"大家好,我是{self.name}")


class Dog(Animal):
    pass


wc = Dog("旺财", 5)
wc.introduce()
```

注意:

- 私有的属性，不能在类的外面访问
- 私有的方法，不能在类的外面访问
- 私有的属性、方法，不会被子类继承，也不能被访问
- 一般情况下，私有的属性、方法都是不对外公布的，往往用来做内部的事情，起到安全的作用
- python中是可以多继承的
- 父类中的方法、属性，子类会继承
- 如果父类中有同名的方法，可以通过对象.__mro__来查看方法搜索的继承顺序

5.8 方法重写

作用：如果父类中的方法不满足自身需求可对父类方法进行重写

示例：

```
class Animal:
    def __init__(self, name, age):
        self.name = name
        self.__age = age

    def introduce(self):
        print(f"大家好,我是{self.name}")


class Dog(Animal):
    def introduce(self):
        print(f"大家好,我是世界上最帅的{self.name}")


wc = Dog("旺财", 5)
wc.introduce()
```

5.9 调用父类方法

作用：在某种情况下，我们需要在对象的方法内部调用基类的方法。

示例：

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

    def introduce(self):
        print(f"大家好,我是{self.name}")


class Dog(Animal):
    def __init__(self, name, age):
        self.__age = age
        super(Dog, self).__init__(name)

    def introduce(self):
```

```
        print(f"大家好,我是世界上最帅的{self.name}, 我今年{self.__age}岁了")


wc = Dog("旺财", 5)
wc.introduce()
```

5.10 多态

多态（英语：polymorphism）指为不同**数据类型**的实体提供统一的**接口**。调用这个接口时根据不同的实体有不同的表现形式。

示例：

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

    def introduce(self):
        print(f"animal:{self.name}")


class Dog(Animal):
    def introduce(self):
        print(f"dog:{self.name}")


class Cat(Animal):
    def introduce(self):
        print(f"cat:{self.name}")


def fx(obj):
    obj.introduce()


dog = Dog("旺财")
cat = Cat("波斯猫")
fx(dog)
fx(cat)
```

5.11 类属性和实例属性

类属性：用来定义这个类型的所有对象所公有的特征属性

实例属性：用来定义对象所特有的特征属性

示例：

```python
class Chinese:
    # 类属性
    lang = "zh"

    def __init__(self, name, age):
        # 实例属性
        self.name = name
        self.age = age
```

5.12 静态方法和类方法

静态方法：如果一个方法和类对象以及实例对象都没有关联关系，只是为了方便把函数放在类中进行代码封装，这个方法就是一个静态方法。

类方法：用来定义这个类型的所有对象所公有的行为。

示例：

```python
"""
    @time: 2021/9/8 11:02 上午
    @author: Mr.Chow
"""


class Chinese:
    # 类属性
    lang = "中文"

    def __init__(self, name, age):
        # 实例属性
        self.name = name
        self.age = age

    # 实例方法
    def introduce(self):
        print(f"I am {self.name}")

    # 类方法
    @classmethod
    def speak(cls):
        print(f"中国人说{cls.lang}")

    # 静态方法
    @staticmethod
    def who():
        print("我是谁,我在哪...")
```

六、Python高级编程

6.1 异常

当Python检测到一个错误时，解释器就无法继续执行，出现了一些错误提示，这就是"异常"。我们可以用异常处理来解决这个问题。

6.1.1异常处理

示例：

```
try:
    # 可能发生异常的代码块
    1 / 0
except Exception as e:
    # 捕捉到异常后执行的代码块
    print(e)
else:
    print("没有发生异常...")
finally:
    print("不管是否发生异常都要执行...")
```

说明：可以使用raise抛出自定义异常。

6.2模块和包

6.2.1 模块

1.模块：每一个.py文件就是一个模块。

2.模块的导入方法:

（1）import

（2）from ... import

（3）from ... import *

（4）as

3.模块导入的限制：

如果一个文件中有__all__变量，那么也就意味着这个变量中的元素，不会被from ... import *时导入。

4.模块中__name__的使用

说明：当导入一个模块时，Python将会去执行被导入模块中的代码。

6.2.2 包

将有联系的模块组织在一起，即放到同一个文件夹下，并且在这个文件夹创建一个名字为 `__init__.py` 文件，那么这个文件夹就称之为 `包` 。

作用：有效避免模块名称冲突问题，让应用组织结构更加清晰。

说明： `__init__.py` 控制着包的导入行为，在 `__init__.py` 文件中，定义一个 `__all__` 变量，它控制着 from 包名 import *时允许导入的模块。

6.3 文件操作

6.3.1 文件读写

文件是用来存储数据的，并且可以实现持久化存储。在Python里面是通过文件对象来操作文件。

- **文件打开函数open()**

语法格式

f = open(file, mode='r'，encoding=None)

file：文件路径

mode:读取模式，默认为r

encoding:编码方式，windows下默认为gbk

模式	说明
r	读模式，默认模式
w	写模式
a	追加模式

- **文件操作方法**

方法	说明
f.readline()	读取一行数据
f.readlines()	返回每行字符串的列表
f.write()	将字符串写入文件
f.close()	关闭文件

- **with语句**

可以通过with语句处理文件关闭

```
with open(file) as f:`
    content = f.read()
    print(content)
```

- **文件的迭代读取**

open函数得到的文件对象是一个可迭代对象，可以用for循环进行遍历

```
with open(file) as f:
    for line in f:
        print(line)
```

6.3.2 os模块

实际开发中，有时需要用程序的方式对文件夹进行一定的操作，比如创建、删除等。

创建文件夹：

```
import os

os.mkdir("文件夹名")
```

递归创建文件夹：

```
import os

os.makedirs("1/2/3")
```

获取当前目录：

```
import os

os.getcwd()
```

获取目录列表：

```
import os

os.listdir("./")
```

删除文件夹：

```
import os

os.rmdir("1/2/3")
```

删除文件：

```
import os

os.remove("1/2/3/test.txt")
```

递归删除文件夹：

```
import os

os.removedirs("1/2/3")
```

修改文件名：

```
import os

os.rename("1/2/3/test.txt", "1/2/3/test1.txt")
```

递归修改文件夹和文件名：

```
import os

os.renames("1/2/3/test1.txt", "4/5/6/test2.txt")
```

七、前端开发

7.1 常用标签

`<div>` div 标签它是可用于组合其他HTML元素的容器。

可用于对大的内容块设置样式属性。

文档布局。它取代了使用表格定义布局的老式方法。

hx 是HTML的标题 标签只用于标题。不要仅仅是为了生成粗体或大号的文本而使用标题 html提供的标题有六种分别是h1 h2 h3 h4 h5 h6。

`<h1>` 定义字号最大的标题,代表大标题,一般一个页面只用一次。

`<h6>` 定义字号最小的标题。

`<p>` p 元素定义段落,会自动在其前后创建一些空白。浏览器会自动添加这些空间。

`<br>` br 元素会在浏览器插入一个简单的换行符。

`<hr>` hr 标签定义 HTML 页面中的主题变化（比如话题的转移），并显示为一条水平线。

`<a>` a标签签用来设置超文本链接。 超链接可以是一个字，一个词，或者一组词，也可以是一幅图像，您可以点击这些内容来跳转到新的文档或者当前文档中的某个部分。

`href` 属性：描述了链接的目标URL。

`target` 属性：设置链接跳转方式

`<img>` img 标签用来申明图像的插入。

`src` 属性：规定显示图像的 URL。URL为图片的相对路径或者绝对路径均可。

`alt` 属性：规定图像的替代文本。

`title` 属性：定义图片的标题，鼠标移动到图片出现。

`<span>` span 用来组合文档中的行内元素,可用作文本的容器。 span 元素没有固定的格式表现，当对它应用样式时，它才会产生视觉上的变化。

`<ul>` ul 标签作为无序列表，它是一个项目的列表，此列项目使用粗体圆点（典型的小黑圆圈）进行标记，无序列表始于 `<ul>` 标签。每个列表项始于 `<li>` 标签

`<ol>` 有序列表也是一列项目，列表项目使用数字进行标记。
有序列表始于 `<ol>` 标签。每个列表项始于 `<li>` 标签

`<!-- 注释 -->` 注释标签用于在源代码中插入注释。注释不会显示在浏览器中。可使用注释对代码进行解释，这样做有助于在以后的时间对代码的修改，当编写了大量代码时尤其有用

7.2 CSS

7.2.1 css选择器

通配符选择器：“*”符号是通配符选择器，匹配html中所有元素。

```
* {color: red; }
```

标签选择器：标签选择器为HTML元素指定特定的样式。

```
p { color: red; }
```

类选择器：类选择器可以为标有特定class的HTML元素指定特定的样式。类选择器以“.”来定义。

```
.red { color: red; }
```

id选择器：id选择器可以为标有特定id的HTML元素指定特定的样式。id选择器以“#”来定义。

派生选择器：派生选择器允许你根据文档的上下文关系来确定某个标签的样式。

```
/* 指定p标签下的所有span标签颜色为红色 */
p span { color: red; }
/* 指定p标签下的子元素span标签颜色为红色 */
p > span { color: red; }
```

选择器分组：对选择器进行分组，这样，被分组的选择器就可以分享相同的声明。用逗号将需要分组的选择器分开。

```
h1, h2, h3, h4, h5, h6 { font-size: 12px; }
```

7.2.2 css常用属性

（1）CSS字体

font-size：设置文本大小。

- 属性值：
- - {number+px}：固定值尺寸像素。
  - {number+%}：其百分比取值是基于父对象中字体的尺寸大小。
- 示例：

```
p { font-size: 20px; }
p { font-size: 100%; }
```

color: 设置文本字体的颜色。

- 属性值：
- - name：颜色名称指定 color。
  - rgb：指定颜色为RGB值。
  - {颜色16进制}：指定颜色为16进制。
- 示例：

```
p { color: red; }
p { color: rgb(100,14,200); }
p { color: #345678; }
```

text-align：设置文本字体的对齐方式。

- 属性值：
- - left：默认值，左对齐。
  - center：居中对齐。
  - Right：右对齐。
- 示例：

```
p { text-align: left; }
p { text-align: center; }
p { text-align: right; }
```

（2）CSS 背景

background-color: 设置对象的背景颜色。

- 属性值：
- - transparent：默认值(背景色透明)。
  - {color}：指定颜色。
- 示例：

```
div { background-color: #666666; }
div { background-color: red; }
```

background-image: 设置对象的背景图像。

- 属性值：
- - none：默认值(无背景图)。
  - url({url})：使用绝对或相对 url 地址指定背景图像。
- 示例：

```
div { background-image: none; }
div { background-image: url('../images/pic.png') }
```

background-repeat: 设置对象的背景图像铺排方式。

- 属性值：
- - repeat：默认值(背景图像在纵向和横向平铺)。
  - no-repeat：背景图像不平铺。
  - repeat-x：背景图像仅在横向平铺。
  - repeat-y：背景图像仅在纵向平铺。
- 示例：

```
div {background-image: url('../images/pic.png'); background-repeat: repeat-y;}
```

7.2.3 盒子模型

（1）外边距（margin）

外边距（margin）：就是围绕在元素边框的空白区域，设置外边距会在元素外创建额外的“空白”设置外边距的最简单的方法就是使用 margin 属性，这个属性接受任何长度单位、百分数值甚至负值

- 属性：
- - margin-top：设置上方外边距。
  - margin-left：设置左方外边距。
  - margin-right：设置右方外边距。
  - margin-bottom：设置下方外边距。
- margin外边距简写：
- - {a}：只有一个值的时候，即上下左右外边距都为a值。
  - {a b}：只有两个值的时候，即上下外边距为a值，左右外边距为b值。
  - {a b c}：只有三个值的时候，即上外边距为a值，左右外边距为b值，下外边距为c值。
  - {a b c d}：只有四个值时候，即上外边距为a值，右外边距为b值，下外边距为c值，左外边距为d值（口诀：顺时针，上右下左）。
- 示例

```
.wrapper { margin-top: 10px; margin-bottom: 20px; margin-left: 30px; margin-right: 40px; }
/* 等同于 */
.wrapper { margin: 10px 40px 20px 30px; }
```

（2）内边距（padding）

内边距（padding）：就是在边框和内容区之间设置内边距的最简单的方法就是使用 padding 属性，这个属性接受任何长度单位、百分数值。

- 属性：
- - padding-top：设置上方内边距。
  - padding-left：设置左方内边距。
  - padding-right：设置右方内边距。
  - padding-bottom：设置下方内边距。
- padding内边距简写：
- - {a}：只有一个值的时候，即上下左右内边距都为a值。
  - {a b}：只有两个值的时候，即上下内边距为a值，左右内边距为b值。
  - {a b c}：只有三个值的时候，即上内边距为a值，左右内边距为b值，下内边距为c值。
  - {a b c d}：只有四个值时候，即上内边距为a值，右内边距为b值，下内边距为c值，左内边距为d值（口诀：顺时针，上右下左）。
- 示例：

```
.wrapper { padding-top: 10px; padding-bottom: 20px; padding-left: 30px; padding-right: 40px; }
/* 等同于 */
.wrapper { padding: 10px 40px 20px 30px; }
```

（3）边框（border）

边框（border）：就是围绕元素内容和内边距的一条或多条线，设置边框的最简单的方法就是使用 border 属性，允许规定元素边框的样式、宽度和颜色。请参照图10-9。

- 属性
- - border-width：设置边框的宽度。
  - border-style：设置边框的样式。
  - - none：默认值，无边框。
    - solid：定义实线边框。
    - double：定义双实线边框。
    - dotted：定义点状线边框。
    - dashed：定义虚线边框。
  - border-color：设置边框的颜色。
- border 边框的简写：
- - {width style color}：定义宽度为width，样式为style，颜色为color的边框。
- 示例：

```
.wrapper { border-width: 1px; border-style: solid; border-color: red; }
/* 等同于 */
.wrapper { border: 1px solid red; }
```

（4）display属性

display属性：设置元素如何显示。

- 属性值
- - inline：默认值。此元素会被显示为内联元素，元素前后没有换行符，内联元素所占具的空间就是他的标签所定义的大小（不能设置width和height）。
  - inline-block：设置元素为行内块状元素，所有的块级元素开始于新的一行，延展到其容器的宽度（能设置width和height）。
  - none：设置元素不显示不占空间，元素与其子元素从普通文档流中移除。这时文档的渲染就像元素从来没有存在过一样，也就是说它所占据的空间被折叠了。
  - block：设置元素为块状元素（能设置width和height）。
  - table：设置元素为块状表格元素。
  - inline-table：设置元素为内联表格元素。
- 示例：

```
.hide { display: none; }
.show { display: block; }
```

八、网络爬虫

8.1 robots协议

网站通过该协议告诉爬虫哪些页面可以抓取,哪些页面不能抓取

8.2 urllib库

Urllib是Python内置的URL处理模块，提供了一系列用于操作URL的功能。Urllib的request模块可以非常方便地抓取URL内容，也就是发送一个GET请求到指定的页面，然后返回HTTP的响应

```
from urllib import request
with request.urlopen('http: //www.baidu.com/') as f:
    data = f.read()
    print('Status: ', f.status, f.reason)
    for k, v in f.getheaders():
        print('%s: %s' % (k, v))
    print('Data: ', data.decode('utf-8'))
```

8.3 requests库

使用requests发送请求获取响应

```
In [1]: import requests

In [2]: r = requests.get("http://www.baid.com")  # 发送请求

In [3]: r.encoding  # 获取响应编码
Out[3]: 'UTF-8'

In [5]: r.status_code  # 获取响应状态码
Out[5]: 200

In [8]: r.text  # 获取响应字符串

In [9]: r.content  # 获取响应字节串
```

为了隐藏自己的ip，避免被封，我们可以在爬取数据的时候使用代理

```
import requests
proxies = {'http': 'http: //144.217.254.175: 3128', 'https':
'https://144.217.254.175: 3128'}
r = requests.get('https://www.baidu.com/', proxies = proxies)
print('Status: ', r.status_code, r.reason)
r.encoding = 'utf-8'
print('Headers: ', r.headers)
print('Data: ', r.text)
```

目前各大网站基本有自己的ca证书，但是不排除有的网站为了节约网站建设开销并没有购买ca证书。又因为requests模块在发送网络请求的时候，默认会验证ca证书。如果当前网站没有ca证书，那么就会抛出 SSLError 异常那么我们可以用verify关键字参数，在请求的时候不验证网站的ca证书

```
import requests
r = requests.get('url', verify=False)
```

8.3 xpath

XPath使用路径表达式在XML文档中选取节点。节点是通过路径或者步来选取的

表达式	描述
nodename	选取此节点名的所节点。
/	从根节点选取。
//	从匹配选择的当前节点是选择文档中的节点，而不考虑它们的位置。
.	选取当前节点。
..	选取当前节点的父节点。
@	选取属性。
text()	选取文字。

谓语是被嵌在方括号中的部分。在下面的表格所示，我们列出了带有谓语的一些路径表达式，以及表达式的结果。

路径表达式	结果
//bookstore/book[1]	选取属于bookstore子元素的第一个book元素。
//bookstore/book[last()]	选取属于bookstore子元素的最后一个book元素。
//bookstore/book[last()-1]	选取属于bookstore子元素的倒数第二个book元素。
//bookstore/book[position()<3]	选取最前面的两个属于bookstore元素的子元素的book元素。
//title[@lang]	选取所有拥有名为lang的属性的title元素。
//title[@lang = 'en']	选取所有title元素，且这些元素拥有值为en的lang属性。
//bookstore/book[price>50.00]	选取bookstore元素的所有book元素，且其中的price元素的值须大于50.00。
//bookstore/book[price>505.00]/title	选取bookstore元素中的book元素的所有title元素，且其中的price元素的值须大于50.00。

8.4 Beautiful Soup 4

Beautiful Soup是一个可以从HTML或XML文件中提取数据的Python库

```
from bs4 import BeautifulSoup


soup = BeautifulSoup(open('index.html'))

# 获取title标签的所有内容
print(soup.title)
# 获取title标签的名称
print(soup.title.name)
# 获取title标签的文本内容
print(soup.title.string)
# 获取head标签的所有内容
print(soup.head)
# 获取第一个p标签中的所有内容
print(soup.p)
# 获取第一个p标签的class的值
print(soup.p["class"])
# 获取第一个a标签中的所有内容
print(soup.a)
# 获取所有的a标签中的所有内容
print(soup.find_all("a"))
# 获取id = "link1"
print(soup.find(id = "link1"))
# 获取所有的a标签，并遍历打印a标签中的href的值
for item in soup.find_all("a"):
    print(item.get("href"))
# 获取所有的a标签，并遍历打印a标签的文本值
for item in soup.find_all("a"):
    print(item.get_text())
```