

# 安装Python

## 下载

<https://www.python.org/downloads/>



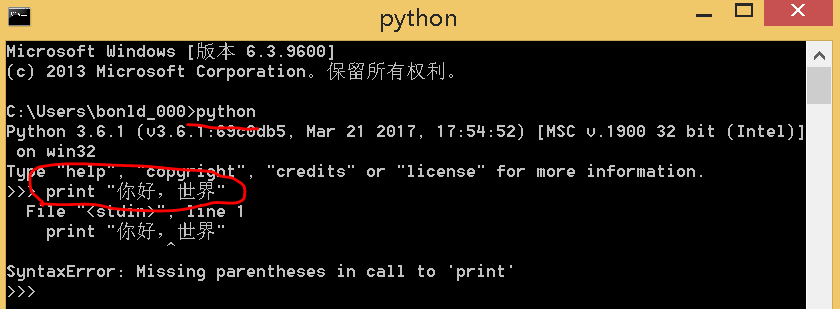
|  |  |
| --- | --- |
| **安装文件** | **python-3.6.1.exe** |

## 环境变量



## 测试

CMD -> python -> print “你好，世界”



## 安装Eclipse+pydev

### 遇到的坑

|  |
| --- |
| Err：使用eclipse3.7Indigo\_Free\_Inst安装pydev插件，无用。 |
| 下载aptana  http://aptana.com/products/studio3/download  Aptana\_Studio\_3\_Setup\_3.6.1.exe是在线安装，需要在线安装Node.js，不装了 |
|  |

### 下载pydev（Eclipse插件）

<http://www.pydev.org/download.html>

实际要转到：<https://sourceforge.net/projects/pydev/files/pydev/>

|  |
| --- |
| **PyDev 5.8.0.zip** |

### 装jdk1.8 32位

这是个32位的python，最好都用32位，不惹事。

|  |
| --- |
| **jdk\_1.8.0\_32bit.exe** |

### 下载eclipse

eclipse-jee-4.6-neon-3-win32-x86\_64（据说可兼容32位）打不开，说java已起动code=13.

**eclipse-jee-4.6-neon-3-win32**可用（打开即用）

### 装pydev插件

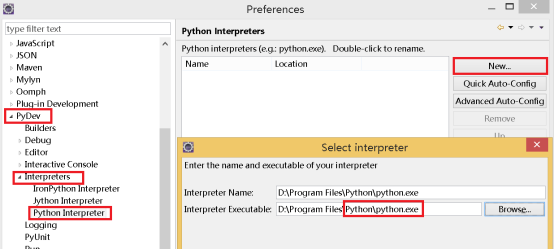
E:\03.SoftWare\002.Java\eclipse-jee-4.6-neon-3-win32\eclipse\**dropins**

中防止解压的pydev文件夹《**PyDev 5.8.0**》

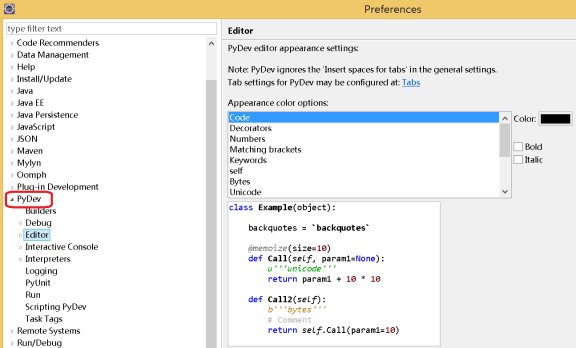
### 配置pydev

重启eclipse。

Window→Preferences→PyDev→Interpreters→Python Interpreter→New



颜色设置



### 测试

新建pyproject，新建Empty文件，写代码（自动生成的都删了），运行

|  |
| --- |
| print ("Hello, Python!"); |

\*python 3.x中，print必须加括号，之前不用

# 输入输出

|  |  |  |
| --- | --- | --- |
| s = input(*"请输入："*);  print (*"你输入的内容是: "*, s) |  |  |
| print(*"%s %s"* % (*'ABC'*, *'def'*))  print(*"A"* + *"B"* + str(3) + repr(4))  print(*"A"*, *"B"*, str(3), str(4)) | ABC def  AB34  A B 3 4 |  |

# 变量类型

## 数字

定义数字，无需类型；还能删除

|  |
| --- |
| n=100;  del n; |
| n=100.1;  print (n); |

## 字符串

不分单引号/双引号，没有字符

|  |
| --- |
| s1=*"ABCDE"*;  print (s1);  s2=*'abcde'*;  print (s2); |

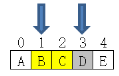
### 字符串对齐

|  |  |
| --- | --- |
| s = *'PMP'*  print (s.ljust(5))  print (s.center(5))  print (s.rjust(5)) | PMP  PMP  PMP |

### 子字符串

|  |
| --- |
| s1=*"ABCDE"*;  print (s1[1:3]); |
| BC |

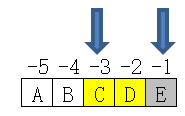
图示：s[1:3]



反向取字符串：

|  |
| --- |
| s1=*"ABCDE"*;  print (s1[-3:-1]); |
| CD |

图示：



## [列表]

|  |  |
| --- | --- |
| list1 = [ *"Andy"*, 99 , *"Bob"*, 97 ]  print (list1[1]);  print (list1[1:]);  print (list1[1:2]);  print (list1[1:2]\*2);#输出两次 | 99  [99, 'Bob', 97]  [99]  [99, 99] |
| lst = [*"孙悟空"*]  print (lst);  lst.append(*"猪悟能"*); print (lst);  lst.extend([*"沙悟净"*, *"小白龙"*]); print (lst);  lst.insert(0, *"六耳猕猴"*); print (lst);  lst.remove(*"六耳猕猴"*); print (lst);  lst.pop(); print (lst); # 删除最后一个元素  lst.sort(); print (lst);  lst.reverse(); print (lst);  print (lst.index(*"孙悟空"*,)); # 逗号要不要都可以，自动生成的  print (lst.count(*"孙悟空"*));  print (len(lst)); # 求长度 | ['孙悟空']  ['孙悟空', '猪悟能']  ['孙悟空', '猪悟能', '沙悟净', '小白龙']  ['六耳猕猴', '孙悟空', '猪悟能', '沙悟净', '小白龙']  ['孙悟空', '猪悟能', '沙悟净', '小白龙']  ['孙悟空', '猪悟能', '沙悟净']  ['孙悟空', '沙悟净', '猪悟能']  ['猪悟能', '沙悟净', '孙悟空']  2  1  3 |
| list1 = set() | 空列表 |

## (元祖)（tuple，相当于只读列表）

tuple本身有元祖、数组的意思，和C语言比较，元祖可以理解为数组类型不同的数组，有点像结构体。

|  |  |
| --- | --- |
| tp = ( *"Andy"*, 99 , *"Bob"*, 97 )  # tp[1] = 100;  print (tp[1]);  print (tp[1:]);  print (tp[1:2]);  print (tp[1:2]\*2);#输出两 | 99  (99, 'Bob', 97)  (99,)  (99, 99) |

元祖转列表

|  |  |
| --- | --- |
| tp = ( *"Andy"*, 99 , *"Bob"*, 97 )  **lst = list(tp)**  print (tp);  print (lst); | ('Andy', 99, 'Bob', 97)  ['Andy', 99, 'Bob', 97] |

## {字典}（dictionary）

赋值、取值：

|  |  |
| --- | --- |
| my\_data = {}  my\_data[*"支付宝"*] = 1000  my\_data[*"余额包"*] = 200  print(my\_data.get(*"支付宝"*))  print(my\_data[*"余额包"*]) | 1000  2000 |

取全部值、遍历

|  |
| --- |
| tinydict = {*'A001'*: *'孙悟空'*, *'A002'*:*'猪悟能'*, *'A003'*: *'沙悟净'*}  print (tinydict) # 输出完整的字典  print (tinydict.keys()) # 输出所有键  print (tinydict.values()) # 输出所有值  # 遍历  for key in tinydict:  print(key + tinydict.get(key)) |
| {'A001': '孙悟空', 'A002': '猪悟能', 'A003': '沙悟净'}  dict\_keys(['A001', 'A002', 'A003'])  dict\_values(['孙悟空', '猪悟能', '沙悟净'])  A001孙悟空  A002猪悟能  A003沙悟净 |

## 类型转换

|  |
| --- |
| n = int(*"100"*);  print (n)  f=float(*"100.50"*);  print (f) |

#没有double(…)

|  |
| --- |
| ord：将字符转为ASCII码：只能有一个字符，当字符串用  chr：ASCII码转为字符串 |
| n = ord(*"A"*)  print (n)  c = chr(n+1)  print (c) |
| 65  B |

## 日期和时间

|  |  |
| --- | --- |
| import time; # 引入time模块  ticks = **time.time**()  print (*"当前时间:"*, ticks) | 一个数字，带小数 |
| import time; # 引入time模块  localtime = **time.localtime**()  print (*"当前时间:"*, localtime) | struct\_time元组  tm\_year  tm\_mon  tm\_mday  tm\_hour  tm\_min  tm\_sec #0 到 61 (60或61 是闰秒)  tm\_wday #0到6 (0是周一)  tm\_yday  tm\_isdst |
| import time;  localtime = time.localtime()  time3 = **time.asctime**( localtime )  print (*"当前时间:"*, time3) | 能看的时间 |
| import time;  t = **time.strftime**(*"%Y-%m-%d %H:%M:%S"*, time.localtime())  print (*"当前时间:"*, t) | 格式化时间 |

asctime() VS ctime()

|  |
| --- |
| import time  print(**time.time()**);  print(**time.localtime()**);  print(time.ctime(**time.time()**));  print(time.asctime(**time.localtime()**)); |
| 1497148470.599267  time.struct\_time(tm\_year=2017, tm\_mon=6, tm\_mday=11, tm\_hour=10, tm\_min=34, tm\_sec=30, tm\_wday=6, tm\_yday=162, tm\_isdst=0)  Sun Jun 11 10:34:30 2017  Sun Jun 11 10:34:30 2017 |

### 月历

|  |  |
| --- | --- |
| import calendar  cal = calendar.month(2017, 1)  print (cal); | January 2017  Mo Tu We Th Fr Sa Su  1  2 3 4 5 6 7 8  9 10 11 12 13 14 15  16 17 18 19 20 21 22  23 24 25 26 27 28 29  30 31 |

## 查看类型

函数：type(…)

|  |  |
| --- | --- |
| n = 100.1;  print (type(n));  n = 1  print (type(n))  n = *'ABC'*  print (type(n))  n = *"DEF"*  print (type(n).\_\_name\_\_)  n = None  print (type(n)) | <class 'float'>  <class 'int'>  <class 'str'>  str  <class 'NoneType'> |

## 集合·set()

集合可以放元祖，不能放列表、字典

|  |  |
| --- | --- |
| s = set()  s.add(*"A"*)  s.add(1)  # Err s.add([1,2,3])  s.add((1, 2, 3))  # Err s.add({'name':'andy'})  print(s)  print(len(s))  print(type(s)) | {1, 'A', (1, 2, 3)}  3  <class 'set'> |

两个集合的操作

|  |
| --- |
| s1 = set();  s1.add(1);s1.add(2);s1.add(3);    s2 = set();  s2.add(3);s2.add(4);s2.add(5);    s3 = s1 & s2  print(*"交集"*, s3)    s3 = s1 | s2  print(*"并集"*, s3)    s3 = s1 - s2  print(*"差集"*, s3)    # 对称差集：只属于其中一个集合，而不属于另一个集合的元素组成的集合  s3 = s1 ^ s2  print(*"对称差集"*, s3) |
| 交集 {3}  并集 {1, 2, 3, 4, 5}  差集 {1, 2}  对称差集 {1, 2, 4, 5} |

集合元素的删除：使用discard(…)或clear()

|  |  |
| --- | --- |
| s1 = set();  s1.add(1);s1.add(2);s1.add(3);  # s1.remove(4) 元素不存在会出错  s1.discard(4)  print(s1)  s1.discard(3)  print(s1)  s1.clear()  print(s1) | {1, 2, 3}  {1, 2}  set() |

# 运算符

## 算数运算符

|  |  |
| --- | --- |
| a = 5  b = 2  print (a / b)  print (a % b) # 仅用于整数除法  print (a // b) # 商的整数  print (a \*\* b) | 2.5  1  2  25 |

## 成员运算符

in、not in

|  |  |
| --- | --- |
| lst = [1, 2, 3, 4, 5 ];  print (1 in lst)  print (99 in lst) | True  False |

## 身份运算符

is：判断两个标识符是不是引用自一个对象

取地址函数：id(…)

|  |
| --- |
| n=100  addr = **id(n)**  print (addr) |

|  |  |
| --- | --- |
| n1 = 100  n2 =100  print (n1 == n2)  print (n1 is n2) | True  True |

值相等但引用不同的例子：（列表可以，字符串、元祖不行）

|  |  |
| --- | --- |
| a = [1,2,3]  print (a[0:])  print (a)  print (a[0:] == a)  print (a[0:] is a) | [1, 2, 3]  [1, 2, 3]  True  False |

# 结构化编程

## 顺序结构

只有if，没有switch。

|  |  |
| --- | --- |
| n=0  if n>=4:  print(*"热干面"*)  else:  print(*"饿着"*) | n=2  if n>=4:  print(*"热干面"*)  elif n>=1.5:  print(*"包子"*)  else:  print(*"饿着"*) |

配合逻辑运算符（and、or、not）功能更强大。

|  |  |  |
| --- | --- | --- |
| a = 17  m = 30  if a >= 18 **and** m >= 10:  print(*"开机"*)  else:  print(*"Get out！"*) | z = 100  m = 0  if z >= 4 **or** m >= 4:  print(*"热干面"*)  else:  print(*"饿着"*) | r = False  if **not** r:  print(*"Get out"*)  else:  print(*"传经"*) |

短路逻辑运算：以and为例，前面不满足条件，右面不算

|  |  |
| --- | --- |
| a = 0  b = 1  if (a != 0) **and** (b / a > 2):  print (b / a)  else:  print (*"No"*) | No |
| a = 0  b = 1  if (a != 0) **or** (b / a > 2):  print (b / a)  else:  print (*"No"*) | if (a != 0) or (b / a > 2):  ZeroDivisionError: division by zero |

## 循环结构

**while循环**

|  |  |
| --- | --- |
|  | **可以带else** |
| a = 1  while a <= 81:  print (str(a) + *"继续整"*)  a += 1 | a = 1  while a <= 81:  print (str(a) + *"继续整"*)  a += 1  **else:**  **print (*'功德圆满'*)**  print (*'不整了'*) |

无限循环

|  |
| --- |
| while True: |
| while 1: |

**for循环（类似于C#里的foreach循环）**

|  |  |
| --- | --- |
| 列表 | n = [*'三打白骨精'*,*'三调芭蕉扇'*,*'西梁国留婚'*]  for s in n:  print (s) |
| 字典 | tinydict = {*'A001'*: *'孙悟空'*, *'A002'*:*'猪悟能'*, *'A003'*: *'沙悟净'*}  for n in tinydict:  print (n, **tinydict[n]**) |

使用range()函数遍历数字

|  |
| --- |
| for i in (1, 10): # 1~10  print (i); |
| for i in (5, 6, 9): # 遍历集合  print (i); |

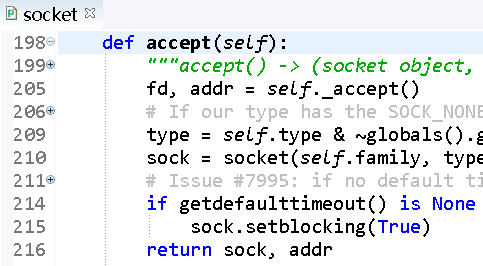
### 退出循环

|  |  |
| --- | --- |
| break | continue |
| i = 1  while i < 10:  print (i)    if i == 5:  break;  else:  i += 1; | i = 1  while i < 10:  print (i)  i += 1;  if i == 5:  print (*"i="*, 5)  continue; |
| 1  2  3  4  5 | 1  2  3  4  i= 5  5  6  7  8  9 |

# 函数

|  |
| --- |
| def **printStr**(s):  *"打印消息"*  print (*"Info:"*, s)  *"返回"*  return  printStr(*"万历十五年"*) |

* def开头
* 直接用”字符串”注释
* 可以不要return
* 可以返回多个值，其实是个元祖tuple(…,…)



## 传参

python中一切皆为对象，严格地说都是引用传递，没有值传递，但是可以有“传不可变对象”和“传可变对象”

|  |  |
| --- | --- |
| **不可变对象（mutable）** | **可变对象（immutable）** |
| 整数、字符串、元组 | 列表，字典 |
| 函数内修改的只是参数的副本，不影响参数本身的内容 |  |

例：

|  |  |
| --- | --- |
| **传不可变对象** | **传可变对象** |
| def **swap**(a, b):  temp = a  a = b;  b = a;  return  a = 1;b = 2;  swap(a, b)  print (a)  print (b) | def **wuzhishan**(mylist):  *"修改传入的列表"*  mylist.append(*"孙悟空"*);  return    mylist = [*'唐玄奘'*];  wuzhishan(mylist);  print (mylist) |

## 参数的写法

|  |  |
| --- | --- |
| 关键字参数 | def **printStr**(sInfo, sNote):  print (*"Info:"*, sInfo)  print (*"note:"*, sNote)  return  printStr(sNote=*"关键字参数必须和形参一致"*, sInfo=*"可以不顾顺序"*) |
| 缺省参数 | def **printStr**(sInfo=*"无信息"*, sNote=*"无备注"*):  print (*"Info:"*, sInfo)  print (*"note:"*, sNote)  return  printStr(sNote=*"缺省参数"*) |
| 变长参数（\*） | def **printStr**(\*paras):  for p in paras:  print (p);  return  printStr(1, 2, 3) |

## 匿名函数

|  |
| --- |
| n = lambda arg1, arg2: arg1 \*\* arg2;  print (n(2, 3)) |

## 全局变量·局部变量

|  |  |
| --- | --- |
| v1 = 1;  v2 = 1;  def **myFunc**():  global v1  v1 = 2;  v2 = 2;    myFunc()  print (v1)  print (v2) | 函数内使用全局变量，需要加global  否则是个同名的局部变量 |
| 2  1 |  |

嵌套函数改外层函数的局部变量

|  |  |
| --- | --- |
| def **outer**():  num = 10  def **inner**():  nonlocal num # nonlocal关键字声明(Eclipse游客报错，不影响运行)  num = 100  print(*"inner"*, num)  inner()  print(*"outer"*, num)  outer() | inner 100  outer 100 |

dir() 函数返回模块里定义过的名字

|  |  |
| --- | --- |
| import math  content = dir(math)  print (content); | ['\_\_doc\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc'] |

# IO

## 写文件

|  |
| --- |
| **f = open(*"book.txt"*, *"w"*)**  num = **f.write**(*"设使天下无有孤,不知当几人称帝,几人称王!\n"*)  print(num)  **f.close()** |
| def **writeFile**(fileName, s):  **f = open(fileName, *"w"*)**  **f.write(s)**  **f.close()**    writeFile(*"book.txt"*, *"提兵百万西湖上,立马吴山第一峰"*); |

## 读文件

|  |
| --- |
| def **writeFile**(fileName):  f = open(fileName, *"r"*)    s = f.read()#一行一行都读出来  print(s)    f.close()    writeFile(*"book.txt"*); |
| 读一行：  s = f.readline() |
| 读所有行[列表]：  s = f.readlines() |

使用with，不必close

|  |
| --- |
| with open(*'book.txt'*, *'w'*) as f:  w\_count = f.write(*"设使天下无有孤,不知当几人称帝,几人称王!\n"*)  print(w\_count)  with open(*'book.txt'*, *'r'*) as f:  read\_data = f.read()  print(read\_data) |

## 序列化·反序列化

pickle模块：序列化：将对象保存到文件

pickle模块：反序列化：从文件创建之前保存的对象

|  |
| --- |
| import pprint, pickle  mydata = [1, 2, 3]  # wb以二进制格式打开一个文件只用于写入(覆盖or新建)  def **saveData**():  output = open(*'data.txt'*, *'wb'*)  **pickle.dump**(mydata, output)  output.close()  return  def **readData**():  output = open(*'data.txt'*, *'rb'*)  \_d = **pickle.load**(output)  output.close()    # pprint:用于打印 Python 数据结构  **pprint.pprint**(\_d)    return  saveData();  readData(); |

# 面向对象

## 封装

|  |
| --- |
| class **Cat**:  name = *"咪咪"*  def **speak**(*self*): # 类的方法必须有“第一个参数”,按照惯例, 名字是self  return *'喵……'*  # ---------------------------  c = Cat()  # ---------------------------  print(c.name)  print(c.speak()) |

#类的方法必须有“第一个参数”,按照惯例, 名字是self，代表类的实例

#类的构造方法是\_\_init\_\_ (self);私有成员用\_\_开头

|  |
| --- |
| class **Cat**:  name = *''*;  \_\_age = 0;  def **\_\_init\_\_**(*self*, name, age):  print(*"构造方法"*);  *self*.name = name  *self*.\_\_age = age;    def **speak**(*self*):  print(*self*.\_\_age);  return *'喵……'*  # ---------------------------  c = Cat(*"咪咪"*, 2)  # ---------------------------  print(c.name)  # AttributeError: 'Cat' object has no attribute '\_\_age'  # print(c.\_\_age)  print(c.speak()) |

## 继承

|  |  |
| --- | --- |
| class **Cat**:  def **eat**(*self*):  print(*"食肉动物"*)  def **speak**(*self*):  return *'喵……'*  # ---------------------------  class **Tiger**(**Cat**):  def **speak**(*self*):  *self*.eat()  return *'嗷'*  # ---------------------------  c = Tiger()  # ---------------------------  print(c.speak()) | 食肉动物  嗷 |
| **多重继承** |  |
| class **GuoXiaoTian**:  def **character**(*self*):  print(*"忠义"*)  # ---------------------------  class **HongQiGong**:  def **skill**(*self*):  print(*"降龙十八掌"*)  # ---------------------------  class **GuoJing**(GuoXiaoTian, HongQiGong):  def **\_\_init\_\_**(*self*):  print(*"郭靖"*)  # ---------------------------  gj = GuoJing()  gj.skill()  gj.character() | 郭靖  降龙十八掌  忠义 |

### 方法重载和运算符重载

|  |  |
| --- | --- |
| class **Vector**:  def **\_\_init\_\_**(*self*, a):  *self*.a = a    def **\_\_str\_\_**(*self*):  return *'Vector (%d)'* % (*self*.a)    def **\_\_sub\_\_**(*self*, other):  return Vector(*self*.a - other.a)  def **\_\_add\_\_**(*self*, other):  return Vector(*self*.a + other.a)  v1 = Vector(100)  v2 = Vector(1)  print (v1)  print (v1 + v2)  print (v1 - v2) | Vector (100)  Vector (101)  Vector (99) |

## 多态

多态：方法调用将作用在参数x的实际类型上。调用某继承方法，总是先查找它自身的定义，如果没有定义，则沿着继承链查找，直到找到为止。

在实现的时候，定义某个继承方法，并且传入相应的类型就行

|  |
| --- |
| class **SunWuKong**:  def **speak**(*self*):  print(*"灵台方寸山，斜月三星洞"*, *self*.\_\_class\_\_)    class **QiTianDaSheng**(SunWuKong):  def **speak**(*self*):  print(*"皇帝轮流做，明年到我家"*, *self*.\_\_class\_\_)    class **SunXingZhe**(SunWuKong):  def **speak**(*self*):  print(*"师祖，老孙有礼了"*, *self*.\_\_class\_\_)    swk = SunWuKong()  qtds = QiTianDaSheng()  sxz = SunXingZhe()  **def speak2(x):**  **x.speak()**  speak2(qtds)  speak2(swk)  speak2(sxz) |
| 皇帝轮流做，明年到我家 <class '\_\_main\_\_.QiTianDaSheng'>  灵台方寸山，斜月三星洞 <class '\_\_main\_\_.SunWuKong'>  师祖，老孙有礼了 <class '\_\_main\_\_.SunXingZhe'> |

# 异常处理

|  |  |
| --- | --- |
| 所有异常   * 可以没有except或 finally | b = input(*"输入除数："*)  try:  a = 5 / int(b);  print(a)  except:  print(*"除数不得为0"*);  finally:  print(*"End"*); |
| 指定异常   * 可以有else | import sys  try:  f = open(*'book.txt'*)  s = f.readline()  i = int(s.strip())  except OSError as err:  print(*"OS error: {0}"*.format(err))  except ValueError:  print(*"Could not convert data to an integer."*)  else:  print(*"Unexpected error:"*, sys.exc\_info()[0]) |
| 抛出异常 | try:  raise NameError(*'抛出异常'*)  except NameError:  print(*'处理异常'*)  raise # 再度抛出 |

Python3中打印原生异常

|  |  |
| --- | --- |
| b = *"0"*  try:  a = 5 / int(b);  print(a)  except Exception as e:  print(*"除数不得为0"*);  print(e) | 早期版本是：except Exception,e: |

# 测试

使用unittest模块测试

|  |
| --- |
| def **max**(a, b):  if a > b:  return a  else:  return b  import unittest  class **TestFunctions**(unittest.TestCase):  def **test\_max**(*self*):  # 测试正确结果  *self*.assertEqual(max(10, 5), 10)    # 测试异常：异常类型，函数名，参数列表  *self*.assertRaises(TypeError, max, 10)    unittest.main() |

doctest模块测试：根据程序中内嵌的文档字符串测试

|  |  |
| --- | --- |
| def **average**(values):  *"""*  *>>> print(average([20, 30, 70]))*  *40.0*  *"""*  return sum(values) / len(values)  import doctest  doctest.testmod() # 自动验证嵌入测试 | def **average**(values):  *"""*  *>>> print(average([20, 30, 70]))*  *20.0*  *"""*  return sum(values) / len(values)  import doctest  doctest.testmod() # 自动验证嵌入测试 |
| 无输出  （Run as Python unit-test能看到OK） | Failed example:  print(average([20, 30, 70]))  Expected:  20.0  Got:  40.0 |

# 网络编程

|  |
| --- |
| 抓百度首页  （python 3.x中urllib库和urilib2库合并成了urllib库） |
| from urllib.request import urlopen  for line in urlopen(*'https://www.baidu.com'*):  line = line.decode(*'utf-8'*)  print(line) |

## Socket

注意：包名不能叫socket，否则和import socket冲突

|  |  |
| --- | --- |
| **Server** | **Client** |
| import socket  # 创建 socket对象  # AF\_INET：套接字家族(family)  # 套接字类型：SOCK\_STREAM（面向连接）/SOCK\_DGRAM（非连接）  skt\_s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  host = socket.gethostname()  port = 9999  # 绑定  skt\_s.bind((host, port))  # 设置最大连接数，超过后排队  skt\_s.listen(5)  while True:  # 建立客户端连接  (skt\_c, addr) = skt\_s.accept()  print(*"连接地址: %s"* % str(addr))    msg = skt\_c.recv(1024)  print(*"客户端说："* + msg.decode(*'utf-8'*))    msg = *'いらっしゃいませ'* + *"\r\n"*  skt\_c.send(msg.encode(*'utf-8'*))  skt\_c.close() | import socket  skt\_s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  host = socket.gethostname()  port = 9999  skt\_s.connect((host, port))  msg = *'こんにちわ'* + *"\r\n"*  skt\_s.send(msg.encode(*'utf-8'*))  # 接收小于 1024 字节的数据  msg = skt\_s.recv(1024)  skt\_s.close()  print (*"服务器说："*, msg.decode(*'utf-8'*)) |
| 连接地址: ('169.254.82.150', 45004)  客户端说：こんにちわ | 服务器说： いらっしゃいませ |

## 发送邮件

### 163邮箱



|  |
| --- |
| MY\_PWD = *"abc123"*  MY\_EMAIL = *"huxiajin\_work3@163.com"* |

MIME(Multipurpose Internet Mail Extensions)多用途互联网邮件扩展类型

|  |  |
| --- | --- |
| **发邮件套路** | |
| MIMEText |  |
| 'from' | 显示的发件人（不写就是“我”） |
| 'to' | 显示的收件人（不写就没有） |
| 'subject' | 标题（不写就是“无主题”） |
| smtp = smtplib.SMTP()  smtp.connect(邮件服务器, '25')  smtp.login(发送方用户名, 密码)  smtp.sendmail(发件邮箱, 收件邮箱, str(邮件内容)) | |

|  |  |
| --- | --- |
| import smtplib  from email.mime.text import MIMEText  mail\_msg = *"只需消费3000元即可领奖"*  print (mail\_msg)  msg = MIMEText(mail\_msg, *'plain'*, *'utf-8'*)    msg[*'from'*] = *"666@q.com"*  msg[*'to'*] = MY\_EMAIL  msg[*'subject'*] = *'您被大奖砸到了'*  try:  smtp = smtplib.SMTP()  smtp.connect(*'smtp.163.com'*, *'25'*)  smtp.login(MY\_EMAIL, MY\_PWD)  smtp.sendmail(MY\_EMAIL, MY\_EMAIL, str(msg))  print (*"邮件发送成功"*)  smtp.quit()  except smtplib.SMTPException as e:  print (*"Error: 无法发送邮件"*)  print(e) | import smtplib  from email.mime.text import MIMEText  mail\_msg = *"""*  *<p>您真幸运</p>*  *<p><a href="http://www.taobao.com">*  *您被大奖砸中了*  *</a></p>*  *"""*  print (mail\_msg)  msg = MIMEText(mail\_msg, *'html'*, *'utf-8'*)    msg[*'from'*] = *"666@q.com"*  msg[*'to'*] = MY\_EMAIL  msg[*'subject'*] = *'您被大奖砸到了'*  try:  smtp = smtplib.SMTP()  smtp.connect(*'smtp.163.com'*, *'25'*)  smtp.login(MY\_EMAIL, MY\_PWD)  smtp.sendmail(MY\_EMAIL, MY\_EMAIL, str(msg))  print (*"邮件发送成功"*)  smtp.quit()  except smtplib.SMTPException as e:  print (*"Error: 无法发送邮件"*)  print(e) |
|  | |

发送附件（黄底色部分为在之前代码基础上修改的内容）

|  |
| --- |
| import smtplib  from email.mime.text import MIMEText  from email.mime.multipart import MIMEMultipart  mail\_msg = *"只需消费3000元即可领奖"*  print (mail\_msg)  # --↓之前的MIMEText改为：MIMEMultipart  msg = MIMEMultipart()  #--↑---------------------------------  msg[*'from'*] = *"666@q.com"*  msg[*'to'*] = MY\_EMAIL  msg[*'subject'*] = *'您被大奖砸到了'*  #--↓MIMEText内容变为attach上---------------------------------  msg.attach(MIMEText(*'恭喜您又中奖了'*, *'plain'*, *'utf-8'*))  #--↑---------------------------------  #--↓附件---------------------------------  ATT1\_PATH = *'1.JPG'*  # # base64编码可用于在HTTP环境下传递较长的标识信息  att1 = MIMEText(open(ATT1\_PATH, *'rb'*).read(), *'base64'*, *'utf-8'*)  att1[*"Content-Type"*] = *'application/octet-stream'*  att1[*"Content-Disposition"*] = *'attachment; filename="%s"'* % ATT1\_PATH  msg.attach(att1)  #--↑---------------------------------  try:  smtp = smtplib.SMTP()  smtp.connect(*'smtp.163.com'*, *'25'*)  smtp.login(MY\_EMAIL, MY\_PWD)  smtp.sendmail(MY\_EMAIL, MY\_EMAIL, str(msg))  print (*"邮件发送成功"*)  smtp.quit()  except smtplib.SMTPException as e:  print (*"Error: 无法发送邮件"*)  print(e) |

显示图片（也是一种附件）

|  |
| --- |
| import smtplib  from email.mime.image import MIMEImage  from email.mime.text import MIMEText  from email.mime.multipart import MIMEMultipart  mail\_msg = *"""*  *<p>您真幸运</p>*  *<p><a href="http://www.taobao.com"><img src="cid:image1"/>*  *您被大奖砸中了*  *</a></p>*  *"""*  print (mail\_msg)  #--即使显示的图片，也是附件----------------------------  msg = MIMEMultipart()  msg.attach(MIMEText(mail\_msg, *'html'*, *'utf-8'*))    msg[*'from'*] = *"666@q.com"*  msg[*'to'*] = MY\_EMAIL  msg[*'subject'*] = *'您被大奖砸到了'*  #--读入图片----------------------------  fp = open(*'1.jpg'*, *'rb'*)  msgImage = MIMEImage(fp.read())  fp.close()  #--即使显示的图片，也是附件----------------------------  msgImage.add\_header(*'Content-ID'*, *'<image1>'*)  msg.attach(msgImage)  try:  smtp = smtplib.SMTP()  smtp.connect(*'smtp.163.com'*, *'25'*)  smtp.login(MY\_EMAIL, MY\_PWD)  smtp.sendmail(MY\_EMAIL, MY\_EMAIL, str(msg))  print (*"邮件发送成功"*)  smtp.quit()  except smtplib.SMTPException as e:  print (*"Error: 无法发送邮件"*)  print(e) |

设置昵称

|  |
| --- |
| import smtplib  from email.mime.text import MIMEText  from email.utils import formataddr  mail\_msg = *"只需消费3000元即可领奖"*  msg = MIMEText(mail\_msg, *'plain'*, *'utf-8'*)    msg[*'from'*] = formataddr([*"马云"*, *"666@q.com"*])  msg[*'to'*] = formataddr([*"幸运的你"*, MY\_EMAIL])  msg[*'subject'*] = *'您被大奖砸到了'*  try:  smtp = smtplib.SMTP()  smtp.connect(*'smtp.163.com'*, *'25'*)  smtp.login(MY\_EMAIL, MY\_PWD)  smtp.sendmail(MY\_EMAIL, MY\_EMAIL, str(msg))  print (*"邮件发送成功"*)  smtp.quit()  except smtplib.SMTPException as e:  print (*"Error: 无法发送邮件"*)  print(e) |
|  |

### QQ邮箱





QQ邮箱需要使用SMTP\_SSL协议，更安全，更麻烦，还经常发送失败。

|  |
| --- |
| MY\_PWD = *"bmzyixexacozdfea"*  MY\_EMAIL = *"2826528017@qq.com"*  #-------------------------------------  import smtplib  from email.mime.text import MIMEText  mail\_msg = *"只需消费3000元即可领奖"*  print (mail\_msg)  msg = MIMEText(mail\_msg, *'plain'*, *'utf-8'*)    msg[*'from'*] = MY\_EMAIL  msg[*'to'*] = MY\_EMAIL  msg[*'subject'*] = *'您被大奖砸到了'*  try:  smtp = smtplib.SMTP\_SSL(*'smtp.qq.com'*, 465)  smtp.login(MY\_EMAIL, MY\_PWD)  smtp.sendmail(MY\_EMAIL, MY\_EMAIL, str(msg))  print (*"邮件发送成功"*)  smtp.quit()  except smtplib.SMTPException as e:  print (*"Error: 无法发送邮件"*)  print(e) |

# 多线程



Python3线程中常用的两个模块为：

* \_thread
* threading(推荐使用)

threading模块包含了\_thread 模块中的所有方法

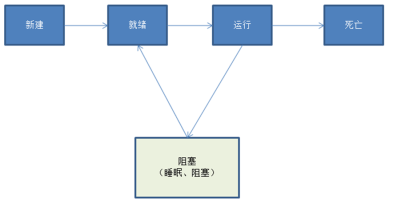
|  |  |
| --- | --- |
| import \_thread  # 为线程定义一个函数  def **print\_time**(threadName):  while 1:  print(threadName)  # 创建两个线程  try:  # 第二个参数必须是个元祖，所以只有一个参数的时候，加个逗号  \_thread.start\_new\_thread(print\_time, (*"Thread-1"*,))  \_thread.start\_new\_thread(print\_time, (*"Thread-2"*,))  except:  print (*"Error: 无法启动线程"*)  while 1:  pass # 主线程啥都不做 | Thread-2  Thread-1  Thread-1  Thread-2  Thread-1  Thread-1 |
| import threading  class **myThread** (threading.Thread):  def **run**(*self*):  while 1:  print(*self*.name)  # 创建新线程  thread1 = myThread()  thread2 = myThread()  # 开启新线程  thread1.start()  thread2.start() | Thread-1  Thread-1  Thread-1  Thread-1  Thread-1  Thread-2  Thread-1  Thread-1  Thread-1  Thread-1  Thread-2 |

复杂一些的线程

构造函数，传入线程编号和名称

主线程等待子线程结束（join()）

|  |
| --- |
| import threading  import time  C\_TIMES = 3  C\_SLEEP\_TIME = 1  class **myThread** (threading.Thread):  def **\_\_init\_\_**(*self*, threadID, name):  threading.Thread.\_\_init\_\_(*self*)  *self*.threadID = threadID  *self*.name = name    def **run**(*self*):  print (*"开始线程："* + str(*self*.threadID) + *":"* + *self*.name)  show(*self*.name)  print (*"退出线程："* + *self*.name)  def **show**(threadName):  counter = C\_TIMES  while counter:  time.sleep(C\_SLEEP\_TIME)  print (*"%s is working"* % threadName)  counter -= 1  # 创建新线程  thread1 = myThread(100, *"青龙营"*)  thread2 = myThread(101, *"白虎营"*)  # 开启新线程  thread1.start()  thread2.start()  thread1.join() # 等待至线程中止，否则主线程直接结束  thread2.join()  print (*"退出主线程"*) |
| 开始线程：100:青龙营  开始线程：101:白虎营  白虎营 is working  青龙营 is working  青龙营 is working  白虎营 is working  青龙营 is working  白虎营 is working  退出线程：白虎营  退出线程：青龙营  退出主线程 |



## 线程同步

如果多个线程都要对同一数据进行修改，可能出现冲突，结果无法预料，为了保证数据的正确性，需要对多个线程进行同步

假设“支付宝”“微信”都绑定银行卡消费，，不加同步，可能“支付宝”付款的过程中“微信”也跑去扣钱.加sleep为了模拟扣钱过程的时间消耗。

|  |  |
| --- | --- |
| **不加锁** | **加锁** |
| import threading  import time  my\_data = {*"银行卡"*:105}  class **myThread** (threading.Thread):  def **\_\_init\_\_**(*self*, name, money):  threading.Thread.\_\_init\_\_(*self*)  *self*.name = name  *self*.money = money  def **run**(*self*):  # 获取锁，用于线程同步  # threadLock.acquire()  while 1:  if(my\_data[*"银行卡"*] - *self*.money < 0):    print(*"余额不足"*)  break;  else:  time.sleep(1)  my\_data[*"银行卡"*] -= *self*.money;  print(*"%s%s"* % (*self*.getName(), my\_data[*"银行卡"*]))  # 释放锁，开启下一个线程  # threadLock.release()  # threadLock = threading.Lock()  threads = []  # 创建新线程  thread1 = myThread(*"支付宝"*, 100)  thread2 = myThread(*"微信"*, 10)  # # 开启新线程  thread1.start()  thread2.start()    # 添加线程到线程列表  threads.append(thread1)  threads.append(thread2)  # 主线程等待所有线程完成  for t in threads:  t.join()  print (*"退出主线程"*) | import threading  import time  my\_data = {*"银行卡"*:105}  class **myThread** (threading.Thread):  def **\_\_init\_\_**(*self*, name, money):  threading.Thread.\_\_init\_\_(*self*)  *self*.name = name  *self*.money = money  def **run**(*self*):  # 获取锁，用于线程同步  threadLock.acquire()  while 1:  if(my\_data[*"银行卡"*] - *self*.money < 0):    print(*"余额不足"*)  break;  else:  time.sleep(1)  my\_data[*"银行卡"*] -= *self*.money;  print(*"%s%s"* % (*self*.getName(), my\_data[*"银行卡"*]))  # 释放锁，开启下一个线程  threadLock.release()  threadLock = threading.Lock()  threads = []  # 创建新线程  thread1 = myThread(*"支付宝"*, 100)  thread2 = myThread(*"微信"*, 10)  # # 开启新线程  thread1.start()  thread2.start()    # 添加线程到线程列表  threads.append(thread1)  threads.append(thread2)  # 主线程等待所有线程完成  for t in threads:  t.join()  print (*"退出主线程"*) |
| 支付宝5  余额不足  微信-5  余额不足  退出主线程 | 支付宝5  余额不足  余额不足  退出主线程 |

消息队列：

FIFO（先入先出)队列Queue；LIFO（后入先出）队列LifoQueue：这些队列实现了锁原语，能够在多线程中直接使用，实现线程间的同步。

* empty()：判空
* put()：加入
* get()：取出

例：3个线程争夺6个资源

|  |
| --- |
| import queue  import threading  import time  class **myThread** (threading.Thread):  def **\_\_init\_\_**(*self*, name, q):  threading.Thread.\_\_init\_\_(*self*)  *self*.name = name  *self*.q = q    def **run**(*self*):  print (*"开启线程："* + *self*.name)  process\_data(*self*.name, *self*.q)  print (*"退出线程："* + *self*.name)  def **process\_data**(threadName, q):  # 有资源就去取，取的时候锁住  while not \_resource\_Queue.empty():  \_resource\_Lock.acquire()  if not \_resource\_Queue.empty():  data = q.get()  print (*" %s get %s"* % (threadName, data))  \_resource\_Lock.release()  time.sleep(0.1)  #--↓准备资源（填充队列）----------------------  \_resource\_Queue = queue.Queue(10)  \_resource\_Lock = threading.Lock()  \_resource\_Lock.acquire()  \_resource\_List = [*"资源1"*, *"资源2"*, *"资源3"*, *"资源4"*, *"资源5"*, *"资源6"*]  for \_rsc in \_resource\_List:  \_resource\_Queue.put(\_rsc)  print(*"资源集结完毕"*)  \_resource\_Lock.release()  #--↑----------------------  #--↓创建新线程---------------------------  \_thread\_Name\_List = [*"Thread-1"*, *"Thread-2"*, *"Thread-3"*]  \_threads = []  for \_tName in \_thread\_Name\_List:  thread = myThread(\_tName, \_resource\_Queue)  thread.start()  \_threads.append(thread)  # 等待所有线程完成  for t in \_threads:  t.join()  #--↑----------------------  print (*"退出主线程"*) |
| 资源集结完毕  开启线程：Thread-1  Thread-1 get 资源1  开启线程：Thread-2  Thread-2 get 资源2  开启线程：Thread-3  Thread-3 get 资源3  Thread-1 get 资源4  Thread-2 get 资源5  Thread-3 get 资源6  退出线程：Thread-2  退出线程：Thread-1  退出线程：Thread-3  退出主线程 |

# 数据解析

## XML解析

文件对象模型（Document Object Model， DOM），是W3C组织推荐的处理可扩展置标语言的标准编程接口。

使用DOM 的解析器解析XML文档时，读取整个文档，把文档中所有元素保存在一个树结构里，然后利用DOM 提供的函数进行操作。

|  |
| --- |
| movies.xml |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <collection shelf=*"经典影视"*>  <movie title=*"人民的名义"*>  <type>犯罪, 主旋律</type>  <format>DVD</format>  <year>2017</year>  <rating>PG</rating>  <stars>9</stars>  <description>最高检：感不感动？广电：不敢动</description>  </movie>  <movie title=*"Transformers5"*>  <type>Science Fiction</type>  <format>DVD</format>  <year>2017</year>  <rating>R</rating>  <stars>8</stars>  <description>最后的骑士</description>  </movie>  <movie title=*"大话西游记"*>  <type>喜剧</type>  <format>DVD</format>  <episodes>2</episodes>  <rating>PG</rating>  <stars>10</stars>  <description>无厘头</description>  </movie>  <movie title=*"中华国执法湄公河"*>  <type>Comedy, Action</type>  <format>Blu-ray</format>  <rating>NC-17</rating>  <stars>2</stars>  <description>依法治外国</description>  </movie>  </collection> |
| import xml.dom.minidom  # 使用minidom解析器打开 XML 文档  DOMTree = xml.dom.minidom.parse(*"movies.xml"*)  collection = DOMTree.documentElement  if collection.hasAttribute(*"shelf"*):  print (collection.getAttribute(*"shelf"*))    # 获取所有节点数据  \_movies = collection.getElementsByTagName(*"movie"*)    # 详细信息  for movie in \_movies:  print (*"\*\*\*\*\*Movie\*\*\*\*\*"*)  if movie.hasAttribute(*"title"*):  print (*"名: %s"* % movie.getAttribute(*"title"*))  \_type = movie.getElementsByTagName(*'type'*)[0]  print (*"类型: %s"* % \_type.childNodes[0].data)  \_format = movie.getElementsByTagName(*'format'*)[0]  print (*"格式: %s"* % \_format.childNodes[0].data)  \_rating = movie.getElementsByTagName(*'rating'*)[0]  print (*"评分: %s"* % \_rating.childNodes[0].data)  description = movie.getElementsByTagName(*'description'*)[0]  print (*"简介: %s"* % description.childNodes[0].data) |
| 经典影视  \*\*\*\*\*Movie\*\*\*\*\*  名: 人民的名义  类型: 犯罪, 主旋律  格式: DVD  评分: PG  简介: 最高检：感不感动？广电：不敢动  \*\*\*\*\*Movie\*\*\*\*\*  名: Transformers5  类型: Science Fiction  格式: DVD  评分: R  简介: 最后的骑士  \*\*\*\*\*Movie\*\*\*\*\*  名: 大话西游记  类型: 喜剧  格式: DVD  评分: PG  简介: 无厘头  \*\*\*\*\*Movie\*\*\*\*\*  名: 中华国执法湄公河  类型: Comedy, Action  格式: Blu-ray  评分: NC-17  简介: 依法治外国 |

## Json解析

json.dumps()：编码

json.loads()：解码

|  |
| --- |
| import json  # Python 字典类型  \_data = {*'id'* : 1, *'name'* : *'唐僧'*}  print (*"Python 原始数据：%s：%s"* % (type(\_data).\_\_name\_\_, \_data))  \_sJson = json.dumps(\_data)  print (*"JSON 对象：%s：%s"* % (type(\_sJson).\_\_name\_\_, \_sJson))  # 将 JSON 对象转换为 Python 字典  \_data2 = json.loads(\_sJson)  print (*"Python 解析出的数据：%s：%s"* % (type(\_data2).\_\_name\_\_, \_data2))  print (*"\tdata2['id']: "*, \_data2[*'id'*])  print (*"\tdata2['name']: "*, \_data2[*'name'*]) |
| Python 原始数据：dict：{'id': 1, 'name': '唐僧'}  JSON 对象：str：{"id": 1, "name": "\u5510\u50e7"}  Python 解析出的数据：dict：{'id': 1, 'name': '唐僧'}  data2['id']: 1  data2['name']: 唐僧 |

json是文件的场合：

|  |
| --- |
| import json  # Python 字典类型  \_data = {*"name"*:*"关羽"*, *"power"*:[100, 30], *"hs1"*:{*"type"*:*"赤兔"*}, *"hs2"*:[{*"type"*:*"乌骓"*}, {*"type"*:*"黄标"*}]}  # 写入 JSON 数据  with open(*'data.json'*, *'w'*) as f:  json.dump(\_data, f)  # 读取数据  with open(*'data.json'*, *'r'*) as f:  \_data2 = json.load(f)    print (*"Python 解析出的数据：%s：%s"* % (type(\_data2).\_\_name\_\_, \_data2)) |
| Python 解析出的数据：dict：{'name': '关羽', 'power': [100, 30], 'hs1': {'type': '赤兔'}, 'hs2': [{'type': '乌骓'}, {'type': '黄标'}]} |

# Python爬虫

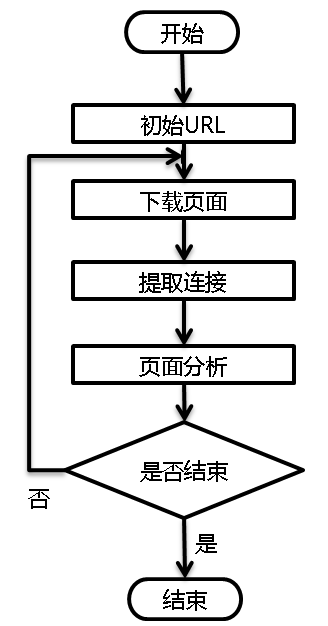
抓取单页面

例：抓取百度搜索页面，并写入到一个html文件中

|  |
| --- |
| import urllib.request  C\_URL\_PREFIX = *"http://www.baidu.com/s?"*  # 关键字字典  \_dic\_Word = {*'word'*:*'狮驼岭'*}  \_word = urllib.parse.urlencode(\_dic\_Word)  print(\_word)  \_url = C\_URL\_PREFIX + \_word  print(\_url)    \_html = urllib.request.urlopen(\_url).read()  \_html = \_html.decode(*'UTF-8'*)  with open(*"get.html"*, *"w"*, encoding=*'utf-8'*) as f:  f.write(\_html);  print(*"End"*) |
| word=%E7%8B%AE%E9%A9%BC%E5%B2%AD  http://www.baidu.com/s?word=%E7%8B%AE%E9%A9%BC%E5%B2%AD  End |

## 网络爬虫

网络爬虫（Crawler）又叫网络蜘蛛（Web Spider）或网络机器人，是一个自动提取网页的程序，Crawler会从一个或多个初始网页的URL开始，抓取网页，并且沿着页面上的超链接在网络上爬行,从而获取大量的网页，在抓取网页后会对得到的网页内容进行解析、过滤、存储。工作流程如图所示：



常用的开源爬虫有基于C++开发的Larbin，基于Java开发的Nutch、Heritrix、crawler4j等。Larbin的设计原则是可配置、效率高，但是需要用户亲自完成解析的任务；Nutch是一款支持分布式的爬虫，专为搜索引擎设计，依赖Hadoop运行，在大集群的场景下能通过Hadoop保证性能，但如果集群机器数量较少，因为Hadoop本身的资源消耗，爬取速度难以发挥出效果，效率反而不如单机爬虫；Heritrix爬虫功能丰富，有完整的参考文档，可扩展性型很强，但是操作起来比较复杂，只在Linux系统上通过了测试，跨平台的稳定性并没有得到验证，系统出现故障时不容易恢复；crawler4j是一款轻量级Java爬虫，只提供爬虫的核心功能，上手非常简单，但文档较少，不支持动态网页抓取（如网页的Ajax内容）。

例：一个简单的爬虫，爬取百度新闻页

|  |
| --- |
| import re  import urllib.request  **# from collections import deque # 双端队列，效率比较高**  import queue **# 适用于多线程**  **# 存放URL的数据结构，全局变量**  \_q **=** None;  *'''根据URL读取HTML内容'''*  def **FuncGetHtml(**inUrl**)**:  urlOpen **=** urllib.request.urlopen**(**inUrl**)**  if *'html'* not in urlOpen.getheader**(***'Content-Type'***)**:  return **-**1;  try:  data **=** urlOpen.read**()**.decode**(***'utf-8'***)**  return data;  except:  return **-**1;  *'''存放URL的数据结构：初始化'''*  def **FuncUrlsInit()**:  global \_q; **# 此处\_q是赋值操作，会被误解为新的局部变量，所以要加global**  **# \_q = deque();**  \_q **=** queue.Queue**()**  return;  *'''存放URL的数据结构:加入'''*  def **FuncUrlsInput(**sUrl**)**:  **# \_q.append(sUrl);**  \_q.put**(**sUrl**)**  return;  *'''存放URL的数据结构：取出'''*  def **FuncUrlsOutput()**:  **# return \_q.popleft();**  return \_q.get**()**;  \_visited **=** set**()**  url **=** *'http://news.baidu.com/'*  FuncUrlsInit**()**;  FuncUrlsInput**(**url**)** **# 初始URL**  cnt **=** 0    while \_q:  url **=** FuncUrlsOutput**()**  \_visited.add**(**url**)**    print**(***'已经抓取: '* **+** str**(**cnt**)** **+** *' 正在抓取 <--- '* **+** url**)**  cnt **+=** 1  sHtml **=** FuncGetHtml**(**url**)**;  if **(**sHtml **==** **-**1**)**:  continue;  else:  pass;    **# 正则表达式提取页面中所有队列, 并判断是否已经访问过, 然后加入待爬队列**  \_regExp **=** re.compile**(***'href=\"(.+?)\"'***)**  for x in \_regExp.findall**(**sHtml**)**:  if **(***'http'* in x**)** and **(**x not in \_visited**)**:  FuncUrlsInput**(**x**)**;  print**(***'加入队列 ---> '* **+** x**)** |
| 已经抓取: 0 正在抓取 <--- http://news.baidu.com/  加入队列 ---> https://gss0.（略）.css  （略）  加入队列 ---> http://sh.news.baidu.com/  已经抓取: 1 正在抓取 <--- https://gss0.（略）.css  已经抓取: 2 正在抓取 <--- https://gss0.（略）.css  已经抓取: 3 正在抓取 <--- https://www.baidu.com/ |