<https://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000>

## python简介

龟叔

优雅、简单、明确

可以用于：网络应用、脚本小工具

优点：代码量少

缺点：运行速度慢、代码不能加密

## 安装python

ubuntu：apt-get install python

命令行加强版：ipython、ipython3

ide：pycharm

解释器：cpython、ipython、pypy、jython、ironpython（.net版本）

## 运行python

区分命令行模式（python xx.py）和交互式。

交互式：python、python3、ipython、ipython3

命令行：python xx.py、python3 xx.py 、 ./xx.py

（第三种： 文件头部加：/usr/bin/env python3）

## 输入输出

print（）、input（）：str

## python基础

注释：#、””” ”””

缩进：tab=4个空格

数据类型：整型、浮点、字符串（”””可以有回车”””）、布尔（Flase，True）、空值（None）

运算符：+、-、\*、/、//、%

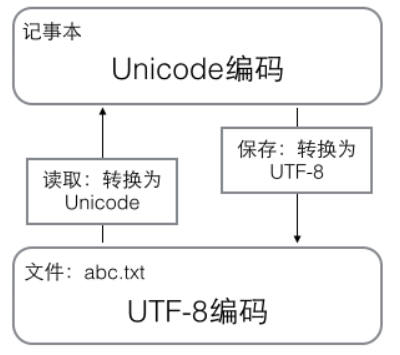
## 编码

ascii：1字节

Unicode：2字节

utf-8：英文1字节，常用汉字3字节，生僻汉字4-6字节

计算机工作的方式：内存中使用unicode，磁盘存储utf-8



ord(‘A’):65

chr(65):A

xxx.encode(“”)

xxx.decode(“”)

len()

应该在.py文件声明使用utf-8解释：

#！/usr/bin/env python3

# -\*- coding: utf-8 -\*-

格式化输出：

%06d、%.2f、%f、%s、%%、%x

format（）：  
a="{0} 分数高于{1:.2f}% 的人".format('小明',20.9999)

***print(a)***

## list和tuple

list：增（append，insert）、删（pop）、改（=）、使用[]赋值、元素类型可不同

tuple：不可改，使用（）赋值，当tuple只有一个元素时，例如：

a=(1),这时，a不是tuple，而是整数，a=(1,)，a是tuple

## 条件判断

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  a **=** int**(**input**(**"输入年龄："**))**  **if** a **>** 100**:**  **print(**"error"**)**  **elif** a **>** 50**:**  **print(**"老年"**)**  **elif** a **>** 30**:**  **print(**"中年"**)**  **elif** a **>** 20**:**  **print(**"青年"**)**  **else:**  **print(**"少年"**)** |

## 循环

for、while、break、continue：

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding:utf-8 -\*-  sum1 **=** 0  **for** i **in** range**(**102**): #记住冒号**  sum1 **=** sum1 **+** i # 记住先不要使用 +=  **if** i **==** 100**: # 记住冒号**  **break**  s1 **=** "sum1 = {0:.2f}"**.**format**(**sum1**)**  **print(**s1**) # format并不会输出，需要print**  n **=** 100  sum2 **=** 0  **while** n **>** 0**: # 冒号**  sum2 **=** sum2 **+** n  n **=** n **–** 1 #先不要使用 n-- 或者--n  **print(**"sum2 = %.2f" **%**sum2**)** |

## dict和set

dict和set都不能存入可变对象

dict：{}、增改（[xx]=xx）、删（pop）、查（get）

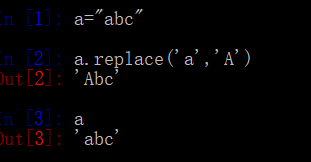
d = dict(name='Bob', age=20, score=88)

d = {‘name’=’Bob’,’age’:20,’score’:88}

set：set([])、增（add,update（添加多个元素））、删（remove,pop（不确定删除哪个，根hash函数相关））

不可变对象：

字符串和数字不可变，所以可用于hash函数，list可变不能用于hash。tuple中可能含有list，所以不能用于hash。



集合运算：

&（交） 、 |（并） 、-（差）

## 退出程序

import sys

sys.exit()

sys.exit(int)

或者

import os

os.\_exit()

区别：sys退出是异常，可以被捕获，然后继续执行

os退出解释器，finally都不会再被执行

一般使用sys退出，fork的子程序使用os退出

## 函数

内置函数：<https://docs.python.org/3/library/functions.html>

函数可以定义在函数中。

自定义函数：

def 函数名（参数）：

语句

|  |
| --- |
| #func\_main.py:  #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  **from** func **import** my\_abs  **print(**my\_abs**(-**10**))**  **print(**my\_abs**(**10**))**  #func.py:  #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  **def** my\_abs**(**x**):**  **if** x **<** 0**:**  **return** **-**x  **else:**  **return** x |

空函数：pass

参数检查：if not isinstance（x,(Type1,Type2,…)）

raise TypeError（““）

返回多个值：x,y=func()

## 参数

默认参数：默认参数必须指向不变对象。

|  |
| --- |
| **def** add\_end**(**L**=[]):**  L**.**append**(**'END'**)**  **return** L    **>>>** add\_end**()**  **[**'END'**,** 'END'**]**  **>>>** add\_end**()**  **[**'END'**,** 'END'**,** 'END'**]**  分析：  Python函数在定义的时候，默认参数L的值就被计算出来了，即**[]**，因为默认参数L也是一个变量，它指向对象**[]**，每次调用该函数，如果改变了L的内容，则下次调用时，默认参数的内容就变了，不再是函数定义时的**[]**了。  定义默认参数要牢记一点：默认参数必须指向不变对象！  修改：  **def** add\_end**(**L**=None):**  **if** L **is** **None:**  L **=** **[]**  L**.**append**(**'END'**)**  **return** L |

可变参数：def func(\*num),num自动组装为tuple，传入tuple时，\*tuple

关键字参数：def func(\*\*num),num组装为dict，传入dict时，\*\*dict

命名关键字参数：def func(a,b,\*,city,job)、def func(a,\*b,city,job),必须传入关键字名称

组合参数：必选，默认，可变，命名关键字参数，关键字参数

## 高级特性

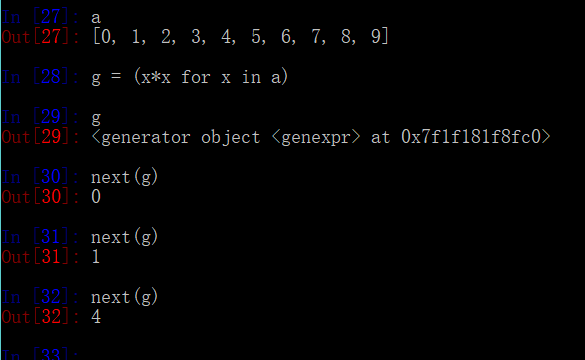
切片：a=list(range(100))、a[1:10:2](第二个到第10个元素，没两个取一个，1，3，5，7，9)

迭代：dict（for key in d，for value in d.values()，for key,value in d.items()）

list(for v in [‘a’,’b’,’c’]，for I,v in enumerate([‘a’,’b’,’c’]))

列表生成式：a = list(range(10)) L = [x\*x for x in a]

生成器：g = (x\*x for x in a)



要将一个函数变为生成器，yield，next遇到yeild即停止，下一个next从上一个yield下一行继续执行。

迭代器：区别Iterable和Iterator，from collections import Iterable

from collections import Iterator，isinstance（[],Iterable）//True

isinstance([],Iterator)//False。总结：可以使用for的都是Iterable；可以使用next的是Iterator；list，dict，str是Iterable不是Iterator，可以使用iter()获得Iterator；for本质使用next（），next使用Iterator

## 函数式编程

抽象程度较高的编程范式，纯函数式编程不允许有变量。python不是纯函数式编程，因为它允许函数中存在变量。

特点：允许把函数作为变量传入函数，并返回函数

高阶函数：函数的一个参数是一个函数，称为高阶函数

|  |  |
| --- | --- |
| map/reduce | map（fn，list）：返回iterator（惰性序列），fn的参数个数为1，对list的每一个元素应用fn  reduce（fn，list）：返回iterator，fn参数个数为2，第一个参数为上一个fn的结果，第二个参数使用list中的元素。需要from functools import reduce |
| filter（fn，list） | 接收一个函数（返回值为bool类型）和一个list，返回iterator惰性序列。 |
| sorted（list[，key][,reverse]） | 使用key函数应用于list的每一个元素，然后根据该函数排序，reverse=True指定反序排列 |

返回函数：

|  |
| --- |
| **def** lazy\_sum**(\***args**):**  **def** sum**():**  ax **=** 0  **for** n **in** args**:**  ax **=** ax **+** n  **return** ax  **return** sum  f = lazy\_sum([1,2,3])  调用lazy\_sum时sun不执行，只有f()时sum执行 |

闭包：

|  |
| --- |
| **def** count**():**  fs **=** **[]**  **for** i **in** range**(**1**,** 4**):**  **def** f**():**  **return** i**\***i  fs**.**append**(**f**)**  **return** fs  f1**,** f2**,** f3 **=** count**()**  f1()//9  f2()//9  f3()//9  因为f1，f2,f3引用了i，count返回时i变为了3，所以都是9  练习：  利用闭包返回一个计数器函数，每次调用它返回递增整数：  #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  **def** createCounter**():**  n **=** 0  **def** counter**():**  nonlocal n # 重要  n **=** n **+** 1  **return** n  **return** counter  # 测试:  counterA **=** createCounter**()**  **print(**counterA**(),** counterA**(),** counterA**(),** counterA**(),** counterA**())** # 1 2 3 4 5  counterB **=** createCounter**()**  **if** **[**counterB**(),** counterB**(),** counterB**(),** counterB**()]** **==** **[**1**,** 2**,** 3**,** 4**]:**  **print(**'测试通过!'**)**  **else:**  **print(**'测试失败!'**)** |

匿名函数：

|  |
| --- |
| 练习：  请用匿名函数改造下面的代码：  def is\_odd(n):  return n % 2 == 1  L = list(filter(is\_odd, range(1, 20)))  解：  #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  L **=** list**(**filter**(lambda** x**:**x**%**2**==**1**,**range**(**1**,**20**)))**  **print(**L**)** |

装饰器：

|  |
| --- |
| **import** functools  **def** log**(**func**):**  *@functools.wraps***(**func**) #没有这一句,下面f.\_\_name\_\_将会是wrapper**  **def** wrapper**(\***kw**,\*\***kkw**):**  **print(**"%s log"**%**func**.**\_\_name\_\_**)**  **return** func**(\***kw**,\*\***kkw**)**  **return** wrapper  *@log # 意为以下定义的函数将加上装饰器*  **def** now**():**  **print(**"2018-01-25"**)**  #以下对于now的调用都会装饰器  now()#加不加functools.wraps(func)都是now  f **=** now  **print(**f**.**\_\_name\_\_**) #因为加了functools.wraps(func),所以输出now**  # 如果装饰器本身有参数  **def** log2**(**text**):**  **def** decorator**(**func**):**  *@functools.wraps***(**func**)**  **def** wrapper**(\***kw**,\*\***kww**):**  **print(**"修饰函数 %s,%s" **%** **(**text**,**func**.**\_\_name\_\_**))**  func**(\***kw**,\*\***kww**) #加不加return感觉没区别**  **return** wrapper  **return** decorator  *@log2***(**'exe'**)**  **def** now2**():**  **print(**"2018/01/25"**)**  now2**()**  f2 **=** now2  **print(**f2**.**\_\_name\_\_**)** |
| 请设计一个decorator，它可作用于任何函数上，并打印该函数的执行时间：  **def** metric**(**fn**):**  *@functools.wraps***(**fn**)**  **def** wrapper**(\*args,\*\***kwargs**):**  time\_begin **=** time**.**time**()**  result **=** fn**(\*args,\*\***kwargs**)**  time\_end **=** time**.**time**()**  **print(**'%s executed in %s ms' **%** **(**fn**.**\_\_name\_\_**,** time\_end**-**time\_begin**))**  **return** result  **return** wrapper  # 测试  *@metric*  **def** fast**(**x**,** y**):**  time**.**sleep**(**0.0012**)**  **return** x **+** y**;**  *@metric*  **def** slow**(**x**,** y**,** z**):**  time**.**sleep**(**0.1234**)**  **return** x **\*** y **\*** z**;**  f **=** fast**(**11**,** 22**)**  s **=** slow**(**11**,** 22**,** 33**)**  **if** f **!=** 33**:**  **print(**'测试失败!'**)**  **elif** s **!=** 7986**:**  **print(**'测试失败!'**)** |

偏函数：

当函数的参数个数太多，需要简化时，使用functools.partial可以创建一个新的函数，这个新函数可以固定住原函数的部分参数，从而在调用时更简单。

|  |
| --- |
| int2 **=** functools**.**partial**(**int **,** base **=** 2**)**  int2**(**'1001'**)**  创建偏函数时，接收函数，**\***args，**\*\***kw这三个参数，int**(**函数**)**，base**=**2（**\*\***kw）  int2（'1001'）  相当于  kw **=** **{**'base'**:**2**}**  int**(**'1001'**,\*\***kw**)** |

## 模块

模块是一组代码的集合，可以引入包。模块名称不要和系统内置模块重名。

模块搜索路径：import sys

sys.path：['', '/usr/lib/python35.zip', '/usr/lib/python3.5', '/usr/lib/python3.5/plat-x86\_64-linux-gnu', '/usr/lib/python3.5/lib-dynload', '/usr/local/lib/python3.5/dist-packages', '/usr/lib/python3/dist-packages']

使用模块：

|  |
| --- |
| #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  'module test' # 第一行字符串作为注释  \_\_author\_\_ **=** 'Pengbo Zhu' # 作者  **import** sys  **def** test**():**  args **=** len**(**sys**.**argv**)**  argv **=** sys**.**argv  **print(**args**)**  **for** arg **in** argv**:**  **print(**arg**)**  **if** \_\_name\_\_ **==** '\_\_main\_\_'**: #只有当前文件是运行文件时返回True**  test**()** |

作用域：

\_\_xx\_\_作为特殊用途的变量

\_xx或者\_\_xx作为私有变量

xx共有变量

python并没有一种办法完全限制访问private变量，只是约定

安装模块：pip install xxx

添加搜索路径：sys.path.append(‘xxx’)临时作用

设置PYTHONPATH环境变量，该环境变量的内容会自动添加到搜索路径。

## 错误调试

try…expect …finally

raise:抛出错误

调试：print

assert：

|  |
| --- |
| #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  'debug test'  \_\_author\_\_ **=** 'Pengbo Zu'  n **=** 0  **assert** n **!=** 0**,**'error : n==0'  输出：  Traceback (most recent call last):  File "./debug.py", line 8, in <module>  assert n != 0,'error : n==0'  AssertionError: error : n==0 |

logging：

|  |
| --- |
| **import** logging  logging**.**basicConfig**(**level**=**logging**.**INFO**)**  n **=** 0  logging**.**info**(**'n=%d'**%**n**)**  10**/**n |

pdb：

|  |
| --- |
| **import** pdb  logging**.**basicConfig**(**level**=**logging**.**INFO**)**  n **=** 0  pdb**.**set\_trace**() #设置断点，运行到这里自动进入调试模式**  10**/**n |

## IO编程

文件读写：

open(path,’r/w/rb’)

f.close()

或者

with open(path,’r/w/rb’) as f:

f.read(size)/f.write()

|  |
| --- |
| #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  'io test'  \_\_author\_\_ **=** 'Pengbo Zhu'  **with** open**(**'./f1'**,**'w'**)** **as** f**:**  f**.**write**(**'hello world\nhello python3'**)**  **with** open**(**'./f1'**,**'r'**)** **as** f**:**  **print(**f**.**read**())** |

StringIO和BytesIO

|  |
| --- |
| **from** io **import** StringIO**,**BytesIO  str **=** StringIO**()**  str**.**write**(**'hello\nhi\npython3'**)**  **print(**str**.**getvalue**())**  bytes **=** BytesIO**()**  bytes**.**write**(**'中文'**.**encode**(**'utf-8'**))**  **print(**bytes**.**getvalue**())** |

操作文件和目录：

os.path.abspath(‘xx’):xx的绝对路径

os.path.join(‘xx’,’yy’):xx，yy拼凑为一个路径，不要直接使用xx/yy，因为linux系统是/分隔符，win是\分隔符

os.mkdir(‘xxx’):创建目录

os.rmdir(‘xxx’)：删除目录

os.path.split(‘a/b/c.txt’):分解为a/b和c.txt

os.path.splitext(‘a/b/c.txt’):分解为a/b/c和txt

拆分合并并不要求文件真实存在，他们只对字符串操作

os.rename(‘old’,’new’):重命名

os.remove(‘xx’):删除文件

复制文件的函数不在os模块中，因为复制文件并非由操作系统提供的系统调用。shutil模块的copyfile函数

序列化：把对象从内存中变为可存储或可传输的过程称之为序列化，python中称之为pickling。

|  |
| --- |
| In **[**9**]:** **import** pickle  In **[**10**]:** d **=** dict**(**name**=**'Bob'**,**age **=** 13**,** gender **=** 'female'**)**  In **[**11**]:** pickle**.**dumps**(**d**) # 把对象序列化成字节，区别于pickle.dump**  Out**[**11**]:** b'\x80\x03}q\x00(X\x06\x00\x00\x00genderq\x01X\x06\x00\x00\x00femaleq\x02X\x03\x00\x00\x00ageq\x03K\rX\x04\x00\x00\x00nameq\x04X\x03\x00\x00\x00Bobq\x05u.'  In **[**12**]:** **with** open**(**'pickle'**,**'wb'**)** **as** f**:**  **....:** pickle**.**dump**(**d**,**f**) #写入文件**  **....:**  In **[**19**]:** **with** open**(**'./pickle'**,**'rb'**)** **as** f**:**  **....:** d **=** pickle**.**load**(**f**)**  **....:**  In **[**20**]:** d  Out**[**20**]:** **{**'gender'**:** 'female'**,** 'age'**:** 13**,** 'name'**:** 'Bob'**}** |

json：

|  |  |
| --- | --- |
| dumps(dict):  返回str，内容为标准的json | In **[**32**]:** **import** json  In **[**33**]:** d **=** dict**(**name**=**'Bob'**,**age**=**18**,**gender**=**'female'**)**  In **[**34**]:** json**.**dumps**(**d**)**  Out**[**34**]:** '{"gender": "female", "age": 18, "name": "Bob"}' |
| dump(dict,f):str存储在文件f中 | In **[**37**]:** **with** open**(**'./json'**,**'w'**)** **as** f**:**  **....:** json**.**dump**(**d**,**f**)**  **....:**  In **[**38**]:** cat json  **{**"gender"**:** "female"**,** "age"**:** 18**,** "name"**:** "Bob"**}** |
| loads(str):将str转换为dict | In **[**47**]:** **with** open**(**'./json'**,**'r'**)** **as** f**:**  **print(**type**(**json**.**loads**(**f**.**read**())))**  **....:**  **<class** 'dict'**>** |
| load(f):将文件内容转化为dict | In **[**52**]:** **with** open**(**'./json'**,**'r'**)** **as** f**:**  **print(**type**(**json**.**load**(**f**)))**  **....:**  **<class** 'dict'**>** |

对于自定义类：json.dumps(s,default=lambda:obj:obj.\_\_dict\_\_)因为class一般有\_\_dict\_\_属性，就是一个dict，也有例外，比如类中定义了\_\_slots\_\_。同样反序列，首先转换为dict，然后由object\_hook函数转换为class：json.loads(json\_str,object\_hook=dict2Student)

|  |
| --- |
| 练习：对中文进行JSON序列化时，json.dumps()提供了一个ensure\_ascii参数，观察该参数对结果的影响：  In **[**53**]:** **import** json  In **[**54**]:** obj **=** dict**(**name**=**'小明'**,**age**=**20**)**  In **[**55**]:** s **=** json**.**dumps**(**obj**,**ensure\_ascii**=True)**  In **[**56**]:** s  Out**[**56**]:** '{"age": 20, "name": "\\u5c0f\\u660e"}'  In **[**57**]:** s **=** json**.**dumps**(**obj**,**ensure\_ascii**=False)**  In **[**58**]:** s  Out**[**58**]:** '{"age": 20, "name": "小明"}' |

总结：

序列化：class->dict->json

反序列化：json->dict->class

## 正则表达式

默认贪婪匹配，非贪婪匹配加？

使用re模块，import re，使用re.match(r‘正则表达式’,’字符串’)，匹配成功返回Match对象，否则返回None。

使用groups()，返回所有分组，Match对象.group(0/1/2)，0表示匹配的整体，1表示第一个分组，…。

|  |
| --- |
| 练习：  请尝试写一个验证Email地址的正则表达式。版本一应该可以验证出类似的Email：   * someone@gmail.com * bill.gates@microsoft.com   #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  '正则表达式测试'  \_\_author\_\_ **=** 'Pengbo Zhu'  **import** re  **def** is\_valid\_email**(**addr**):**  **if** re**.**match**(**r'^[a-zA-Z\.]+@[a-zA-Z]+\.com$'**,**addr**):**  **return** **True**  **return** **False**  # 测试:  **assert** is\_valid\_email**(**'someone@gmail.com'**)**  **assert** is\_valid\_email**(**'bill.gates@microsoft.com'**)**  **assert** **not** is\_valid\_email**(**'bob#example.com'**)**  **assert** **not** is\_valid\_email**(**'mr-bob@example.com'**)**  **print(**'ok'**)** |

## 进程和线程

多进程：

|  |
| --- |
| fork：  #fork  **import** os**,**time  pid **=** os**.**fork**()**  **if** pid **==** 0**:**  **print(**'child process'**)**  **else:**  time**.**sleep**(**1**)**  **print(**'father process'**)**  multiprocessing |
| multiprocessing：  **from** multiprocessing **import** Process  **def** run\_proc**(**name**):**  **print(**'child process running %s(%s)...'**%(**name**,**os**.**getpid**()))**  **print(**'Parent process %s'**%**os**.**getpid**())**  p **=** Process**(**target **=** run\_proc**,**args **=** **(**'test'**,))**  **print(**'child will start'**)**  p**.**start**()**  p**.**join**()**  **print(**'child process end'**)** |
| Pool：  #线程池 apply\_async，表示非阻塞，apply表示阻塞  **from** multiprocessing **import** Pool  **import** time**,**os**,**random  **def** run\_proc**(**name**):**  **print(**'run\_proc start(%s):%s '**%(**name**,**os**.**getpid**()))**  time**.**sleep**(**random**.**random**()\***3**)**  **print(**'run\_proc end(%s):%s '**%(**name**,**os**.**getpid**()))**  p **=** Pool**(**4**)**  **for** i **in** range**(**5**):**  p**.**apply\_async**(**run\_proc**,**args **=** **(**i**,))**  **print(**'mark...mark~'**)**  p**.**close**()**  p**.**join**()**  **print(**'all done'**)**  输出：  mark**...**mark**~ //因为使用了apply\_async方法非阻塞**  run\_proc start**(**0**):**11905  run\_proc start**(**1**):**11906  run\_proc start**(**2**):**11907  run\_proc start**(**3**):**11908  run\_proc end**(**1**):**11906 //因为进程池维持4个进程，只有等待一个进程结束，进程4才可以运行  run\_proc start**(**4**):**11906  run\_proc end**(**4**):**11906  run\_proc end**(**2**):**11907  run\_proc end**(**3**):**11908  run\_proc end**(**0**):**11905  all done  当apply\_async换成apply函数时，输出：  run\_proc start**(**0**):**11913  run\_proc end**(**0**):**11913  run\_proc start**(**1**):**11914  run\_proc end**(**1**):**11914  run\_proc start**(**2**):**11915  run\_proc end**(**2**):**11915  run\_proc start**(**3**):**11916  run\_proc end**(**3**):**11916  run\_proc start**(**4**):**11913  run\_proc end**(**4**):**11913  mark**...**mark**~**  all done |
| subprocess：  #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  **import** subprocess  r **=** subprocess**.**call**([**'ls'**,**'-l'**,**'./'**]) #是list**  **print(**'exit code:' **,**r**)** |
| 进程间通信：  **from** multiprocessing **import** Process**,**Queue  **import** os**,**time**,**random  **def** mread**(**q**):**  **print(**'p(%s) read from quene'**%**os**.**getpid**())**  **while** **True:**  value **=** q**.**get**(True)**  **print(**'get value %s'**%**value**)**  **def** mwrite**(**q**):**  **print(**'p(%s) write to queue'**%**os**.**getpid**())**  **for** i **in** **[**'a'**,**'b'**,**'c'**,**'d'**]:**  **print(**'put value %s'**%**i**)**  q**.**put**(**i**)**  time**.**sleep**(**random**.**random**())**    **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  q **=** Queue**()**  pw **=** Process**(**target **=** mread**,**args **=** **(**q**,))**  pr **=** Process**(**target **=** mwrite**,**args **=** **(**q**,))**  pw**.**start**()**  pr**.**start**()**  pw**.**join**()**  pr**.**close**()** |

多线程：

|  |
| --- |
| #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  'thread测试'  \_\_author\_\_ **=** 'Pengbo Zhu'  **import** threading**,**time  **def** loop**():**  **print(**'thread %s starat'**%**threading**.**current\_thread**().**name**)**  n **=** 0  **while** n **<** 5**:**  n **=** n **+** 1  **print(**'thread %s >>> %d'**%(**threading**.**current\_thread**().**name**,**n**))**  time**.**sleep**(**1**)**  **print(**'thread %s end'**%**threading**.**current\_thread**().**name**)**  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  **print(**'thread %s start'**%**threading**.**current\_thread**().**name**)**  t **=** threading**.**Thread**(**target **=** loop**,**name **=** 'LocalThread'**)**  t**.**start**()**  t**.**join**()**  **print(**'thread %s end'**%**threading**.**current\_thread**().**name**)** |
| lock：  #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  'threing lock测试'  \_\_author\_\_ **=** 'Pengbo Zhu'  **import** threading**,**time  balance **=** 0  lock **=** threading**.**Lock**()**  **def** run\_thread**():**  **global** balance  **for** i **in** range**(**100000**):**  lock**.**acquire**()**  **try:**  balance **=** balance **+** 1  **finally:**  lock**.**release**()**  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  t1 **=** threading**.**Thread**(**target **=** run\_thread**)**  t2 **=** threading**.**Thread**(**target **=** run\_thread**)**  t1**.**start**()**  t2**.**start**()**  t1**.**join**()**  t2**.**join**()**  **print(**balance**)** |

ThreadLocal:

同一线程局部变量之间的传递，使用ThreadLocal。ThreadLocal解决了参数在一个线程中各个函数之间互相传递的问题。

|  |
| --- |
| #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  'ThreadLocal测试'  \_\_author\_\_ **=** 'Pengbo Zhu'  **import** threading  local\_arg **=** threading**.**local**()**  **def** func0**():**  **print(**'thread %s: %s'**%(**threading**.**current\_thread**().**name**,**local\_arg**.**name**))**  **def** func1**(**name**):**  local\_arg**.**name **=** name  func0**()**  t1 **=** threading**.**Thread**(**target **=** func1**,**args **=** **(**'Bob'**,),**name **=** 'T1'**)**  t2 **=** threading**.**Thread**(**target **=** func1**,**args **=** **(**'Alice'**,),**name **=** 'T2'**)**  t1**.**start**()**  t2**.**start**()**  t1**.**join**()**  t2**.**join**()** |

多进程vs多线程：

多进程优点稳定性高，缺点代价大进程数有限

多线程优点快，缺点一个线程崩溃导致整个进程崩溃

计算密集型使用c语言，io密集型使用python

对于分布式应该使用多进程

## 网络编程

TCP

|  |
| --- |
| 服务端：  #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  'socket tcp服务端测试'  \_\_author\_\_ **=** '朱鹏博'  **import** socket**,**threading**,**time  **def** tcplink**(**sock**,**addr**):**  **print(**"tcplink"**)**  **while** **True:**  data **=** sock**.**recv**(**1024**)**  **if** **not** data **or** data**.**decode**(**'utf-8'**)** **==** 'exit'**:**  **break**  time**.**sleep**(**1**)**  sock**.**send**((**'hello %s' **%** data**.**decode**(**'utf-8'**)).**encode**(**'utf-8'**))**  sock**.**close**()**  **print(**'%s closed'**%** str**(**addr**))**  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  sockser **=** socket**.**socket**(**socket**.**AF\_INET**,**socket**.**SOCK\_STREAM**)**  sockser**.**bind**((**'127.0.0.1'**,**8888**))**  sockser**.**listen**(**5**)**  **while** **True:**  sockcli**,**addr **=** sockser**.**accept**()**  t **=** threading**.**Thread**(**target **=** tcplink**,**args **=** **(**sockcli**,**addr**))**  t**.**start**()** |
| 客户端：  #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  'sockt tcp client'  \_\_author\_\_ **=** 'Pengbo Zhu'  **import** socket  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  sock **=** socket**.**socket**(**socket**.**AF\_INET**,**socket**.**SOCK\_STREAM**)**  sock**.**connect**((**'127.0.0.1'**,**8888**))**  **for** data **in** **[**'锋锐'**,**'朱鹏博'**]:**  sock**.**send**(**data**.**encode**(**'utf-8'**))**  **print(**sock**.**recv**(**1024**).**decode**(**'utf-8'**))**  sock**.**close**()** |

UDP：

|  |
| --- |
| 服务端：  #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  'udp socket服务端测试'  \_\_author\_\_ **=** 'Pengbo Zhu'  **import** socket**,**time**,**threading  **def** udplink**(**sock**):**  **while** **True:**  data**,**addr **=** sock**.**recvfrom**(**1024**)**  **if** **not** data**:**  **break**  **print(**"recv from %s: %s" **%** **(**str**(**addr**),**data**.**decode**(**'utf-8'**)))**  sock**.**sendto**((**"hello %s"**%**data**.**decode**(**'utf-8'**)).**encode**(**'utf-8'**),**addr**)**  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  sockser **=** socket**.**socket**(**socket**.**AF\_INET**,**socket**.**SOCK\_DGRAM**)**  sockser**.**bind**((**'127.0.0.1'**,**9999**))**  t **=** threading**.**Thread**(**target **=** udplink**,**args **=** **(**sockser**,))**  t**.**start**()** |
| 客户端：  #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  'udp socket客户端测试'  \_\_author\_\_ **=** 'Pengbo Zhu'  **import** socket  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  sockcli **=** socket**.**socket**(**socket**.**AF\_INET**,**socket**.**SOCK\_DGRAM**)**  **for** data **in** **[**'锋锐'**,**'朱鹏博'**]:**  sockcli**.**sendto**(**data**.**encode**(**'utf-8'**),(**'127.0.0.1'**,**9999**))**  buf**,**addr **=** sockcli**.**recvfrom**(**1024**)**  **print(**'recv from %s: %s '**%(**str**(**addr**),**buf**.**decode**(**'utf-8'**)))**  sockcli**.**close**()** |

## Web开发

web经历的阶段：静态html、cgi、asp/jsp/php、框架

HTTP：

html、css、js

WSGI（web server gateway interface）：自定义application函数，python内置模块wsgiref。

flask框架：通过[装饰器](#_高级特性)把url和函数联系起来。

Django框架：全能型的web框架

web.py：小巧的web框架

bottle：flask类似的框架

tornado：facebook开源异步web框架

模板：实现了mvc的模板。jinja2、mako、cheetah、django

## Mysql数据库

驱动安装：pip3 install mysql-connector-python或者pip3 install PyMySQL。

第三方模块安装目录：/usr/local/lib/python3.5/dist-packages

使用pymysql.connect时需要设置charset=‘utf8’，否则中文乱码。

使用mysql.connector.connect不用设置charset

|  |
| --- |
| #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  'python 连接mysql测试'  \_\_author\_\_ **=** 'Pengbo Zhu'  **import** pymysql  host **=** '127.0.0.1'  user **=** 'root'  passwd **=** 'ubuntu'  dbname **=** 'test'  #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  'python 连接mysql测试'  \_\_author\_\_ **=** 'Pengbo Zhu'  **import** pymysql  host **=** '127.0.0.1'  user **=** 'root'  passwd **=** 'ubuntu'  dbname **=** 'test'  db **=** pymysql**.**connect**(**host**,**user**,**passwd**,**dbname**,**charset**=**'utf8'**)**  cursor **=** db**.**cursor**()**  #创建表  sql1 **=** """create table if not exists user4(id int primary key auto\_increment,  uname varchar(20) not null default 'sb')  """  cursor**.**execute**(**sql1**)**  #插入表  name **=** input**(**'输入姓名：'**)**  sql2 **=** "insert into user4(uname) values('%s')"**%**name  **print(**sql2**)**  **try:**  cursor**.**execute**(**sql2**)**  db**.**commit**()**  **except** Exception **as** e**:**  **print(**e**)**  db**.**rollback**()**  **print(**'error'**)**  #查询表  sql3 **=** 'select \* from user4'  **try:**  cursor**.**execute**(**sql3**)**  res **=** cursor**.**fetchall**()**  **for** row **in** res**:**  **print(**"id: %d,uname:%s"**%(**row**[**0**],**row**[**1**]))**  **except** Exception **as** e**:**  **print(**e**)**  db**.**close**()** |

## 异步IO

并发问题的解决：

1. 多线程和多进程。数量多的情况下，切换耗损严重，影响性能。
2. 异步IO。一个线程处理多个IO请求，没有线程切换损耗。

协程，又称为微线程，纤程。

子程序，即就是函数。一般的子程序是A调用B，B调用C，然后C返回，B返回，A返回。使用调用栈完成，其调用顺序明确。

协程看上去也是子程序，不过子程序内部可中断，然后转去另一个子程序中去执行，在适当的时候返回继续执行。注意，在一个子程序中中断，并非函数调用，而是类似于cpu中断。比如：

|  |
| --- |
| **def** A**():**  **print(**'1'**)**  **print(**'2'**)**  **print(**'3'**)**  **def** B**():**  **print(**'x'**)**  **print(**'y'**)**  **print(**'z'**)**  假设由协程执行，在执行A的过程中，可以随时中断，去执行B，B也可能在执行过程中中断再去执行A，结果可能是  1  2  x  y  3  z |

协程有些类似于多线程，优势：效率高，没有线程切换；不用锁机制。

使用generator实现协程：

|  |
| --- |
| #!/usr/bin/env python3  #-\*- coding: utf-8 -\*-  '协程测试'  \_\_author\_\_ **=** 'Pengbo Zhu'  **def** consumer**():**  r **=** 'begin'  **while** **True:**  n **=** **yield** r #2  **if** **not** n**:**  **return**  **print(**'consumer %s'**%**n**)**  r **=** '200 OK' **#4**  **def** producer**(**c**):**  **print(**'consumer return %s'**%**c**.**send**(None)) #1**  **for** i **in** range**(**1**,**6**):**  **print(**'producer %s'**%**i**)**  r **=** c**.**send**(**i**) #3**  **print(**'consumer return %s'**%**r**) #5**  c**.**close**()**  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  c **=** consumer**()**  producer**(**c**)**  总结：  #1处c.send(None),启动生成器，**必须步骤；**然后执行到#2步骤并返回r；接着执行#3c.send(i)，#2处接收到i，赋值给n，继续执行到#4,然后循环到#2，返回‘200 ok’；继续执行#5循环  结果：  consumer **return** begin  producer 1  consumer 1  consumer **return** 200 OK  producer 2  consumer 2  consumer **return** 200 OK  producer 3  consumer 3  consumer **return** 200 OK  producer 4  consumer 4  consumer **return** 200 OK  producer 5  consumer 5  consumer **return** 200 OK |

asyncio（python3.4引入）：

|  |
| --- |
| **import** asyncio  *@asyncio.coroutine*  **def** wget**(**host**):**  **print(**'wget %s'**%**host**)**  connect **=** asyncio**.**open\_connection**(**host**,**80**)**  reader**,**writer **=** **yield** **from** connect  header **=** 'GET / HTTP/1.0\r\nHost: %s\r\n\r\n'**%**host  writer**.**write**(**header**.**encode**(**'utf-8'**))**  **yield** **from** writer**.**drain**()**  **while** **True:**  line **=** **yield** **from** reader**.**readline**()**  **if** line **==** b'\r\n'**:**  **break**  **print(**'%s header > %s'**%(**host**,**line**.**decode**(**'utf-8'**).**rstrip**()))**  writer**.**close**()**  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  loop **=** asyncio**.**get\_event\_loop**()**  tasks **=** **[**wget**(**host**)** **for** host **in** **[**'www.baidu.com'**,**'www.sohu.com'**,**'www.163.com'**]]**  loop**.**run\_until\_complete**(asyncio.wait(tasks))**  loop**.**close**()** |

async/await（python3.5引入）：

|  |
| --- |
| 和3.4区别是：   1. 把@asyncio.coroutine替换为async； 2. 把yield from替换为await。   **import** asyncio  **async** **def** wget**(**host**):**  **print(**'wget %s'**%**host**)**  connect **=** asyncio**.**open\_connection**(**host**,**80**)**  reader**,**writer **=** **await** connect  header **=** 'GET / HTTP/1.0\r\nHost: %s\r\n\r\n'**%**host  writer**.**write**(**header**.**encode**(**'utf-8'**))**  **await** writer**.**drain**()**  **while** **True:**  line **=** **await** reader**.**readline**()**  **if** line **==** b'\r\n'**:**  **break**  **print(**'%s header > %s'**%(**host**,**line**.**decode**(**'utf-8'**).**rstrip**()))**  writer**.**close**()**  **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**  loop **=** asyncio**.**get\_event\_loop**()**  tasks **=** **[**wget**(**host**)** **for** host **in** **[**'www.baidu.com'**,**'www.sohu.com'**,**'www.163.com'**]]**  loop**.**run\_until\_complete**(**asyncio**.**wait**(**tasks**))**  loop**.**close**()** |

aiohttp：

asyncio实现了异步tcp、udp、ssl等协议，aiohttp实现了asyncio的http版本。