# **1. Python中主流的集成开发环境简介**

​Python是一种非常流行的开源编程语言。得益于无尽的模块选项，Python今天广泛用于脚本语言、Web开发、移动和桌面在许多领域。随着人工智能的复兴，数据科学的崛起，Python更是红火起来。

## PyCharm



链接：[https://www.jetbrains.com/pycharm/](https://www.jetbrains.com/pycharm)

由向来生产IDE利器的JetBrains出品，当之无愧的最最好的Python IDE（没有之一）

* 闭源
* 支持Windows，MacOS和Linux
* 专为Python，Javascript，Coffeescript，Typescript，HTML / CSS，AngularJS，Node.js等而设计
* 集成了单元测试，代码检查，集成版本控制，代码重构工具，各种工具用于项目导航，突出显示和自动完成等等特性
* 支持多个第三方Web开发框架，如Django，Pyramid，web2py，google应用程序引擎和Flask，使其成为一个完整的Rapid应用程序开发IDE。

​

## PyDev



链接：<http://www.pydev.org/>

* Eclipse的Python插件。
* 支持所有已知操作系统
* 开源并可在Eclipse的公共许可证下使用。
* 处理代码完成，集成Python调试，添加令牌浏览器，重构工具等等功能。
* 允许创建新的Django项目，通过热键执行Django操作，并为Django使用单独的运行配置。

## Wing IDE



链接：[http://wingware.com](http://wingware.com/)

* 由Wingware公司创建的一个坚实的IDE
* 支持Windows，Linux和MacOS
* 开源
* 具有强大的调试器，允许设置断点，逐步通过代码，检查数据，远程调试和调试Django模板
* 支持matplotlib，图绘图自动更新
* 还提供代码完成，语法突出显示，源代码浏览，图形调试器和版本控制系统的支持

​

## Komodo IDE



链接：<http://www.activestate.com/komodo-ide>

* 用于动态语言（如Python，PHP和Ruby）的IDE
* 支持Windows，MacOS和Linux
* 闭源 - 拥有专有许可证
* 对Django有限支持，例如模板的语法高亮和代码完成。 （虽然我不能让它正常工作）
* 包括代码重构，自动完成，调用提示，括号匹配，代码浏览器，定义，图形调试，多进程调试，多线程调试，断点配置，代码分析，集成单元测试和与第三方库（如pyWin32）集成的基本功能
* 还提供如包管理器集成，跟踪更改，Markdown查看器，Shell范围，快速书签，去任何东西（Commando）和更多功能

## Eric



链接：<http://eric-ide.python-projects.org/>

* 由Detlev Offenbach开发的Python编辑器和IDE。
* 闭源
* 仅支持Windows操作系统
* 具有Python和Ruby调试器，代码覆盖，自动代码检查，Python和Ruby shell，类浏览器等功能。它具有协作编辑的功能
* 还提供集成任务管理，集成单元测试支持，绳索重构，核心插件Regex和Qt对话框，从编辑器中启动第三方应用程序的选项，应用程序图，项目管理功能，交互式python shell，多语言用户界面，包括英语，德语，俄语，法语，捷克语，西班牙语，意大利语，土耳其语和中文，Subversion版本控制，使用额外插件支持Mercurial和Git等等

## Eclipse



链接：​[https://www.eclipse.org](https://www.eclipse.org/)

* 通用平台
* 支持所有操作系统
* 开源IDE并支持一个完整的语言生态系统
* 应有尽有的插件
* 开发java和Android主要用就是eclipse

## Spyder



链接：<https://github.com/spyder-ide/spyder>

* Python的开源IDE
* 可在MIT许可下免费使用
* Spyder python的一些基本特性是多语言编辑器，交互式控制台，文档查看器，变量浏览器，在文件中查找，文件浏览器等
* 虽然Spyder是一个独立的IDE，支持包括Windows，Linux，MacOS，MacOS X等多个平台，但它也可以作为一个PyQt扩展库嵌入到PyQt应用程序

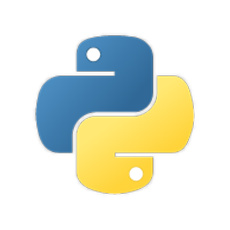
## PyScripter：



链接：<https://sourceforge.net/projects/pyscripter>

* 由Kiriakos Vlahos开发的免费和开源的Python IDE
* 仅支持Windows
* 支持项目，编辑文件和调试
* 代码完成和其他编辑功能
* Python源代码实用程序

## Python-IDLE



链接：<https://wiki.python.org/moin/IDLE>

* 作为Python标准库一部分的内置IDE
* 可在上面尝试小的Python片段和实验不同的功能

## Geany



链接：[http://www.geany.org](http://www.geany.org/)

* 可在GPL许可下使用。
* 跨平台（Linux，Windows和OSX），并提供括号匹配，集成调试器，自动完成和多语言支持。
* 小而快
* 支持插件

那么什么是最好的IDE - 这取决于你问谁，还是群众说了算！

​

# **2.** PyCharm的安装

## **2.1.** PyCharm的自我介绍

### 2.1.1. 智能Python辅助

PyCharm提供智能代码补全、代码检查、实时错误高亮显示和快速修复，还有自动化代码重构和丰富的导航功能

### 2.1.2. Web开发框架

PyCharm为现代web开发框架（如：Django、Flask、Google App Engine、Pyramid和web2py）提供丰富的框架针对性支持。

### 2.1.3. 科学的工具

PyCharm与IPython Notebook集成，提供交互式Python控制台， 并且支持Anaconda和多种科学化的包（例如matplotlib和NumPy）。

### 2.1.4. 跨技术开发

除了Python外，PyCharm还支持JavaScript、CoffeeScript、TypeScript、Cython、SQL、HTML/CSS、模板语言、AngularJS、Node.js等等

### 2.1.5. 远程开发能力

通过远程解释器、集成的ssh终端、Docker和Vagrant集成，在远程主机或虚拟机上运行、调试、测试和部署应用程序。

内建开发者工具

A内建提供海量工具集：集成调试器和测试器、Python分析器、内置终端，以及与主流VCS集成，还有内建的数据库工具。

## **2.2. 下载**

### 2.2.1. 下载地址

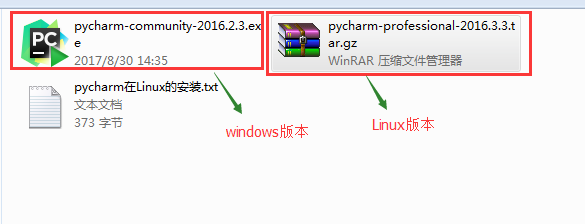
<https://www.jetbrains.com/pycharm/>

进入该页面选择下载linux的下pycharm-professional-2016.3.3.tar.gz

<https://www.jetbrains.com/pycharm/download/previous.html>



下载好后pycharm-professional-2016.3.3.tar.gz 为压缩文件



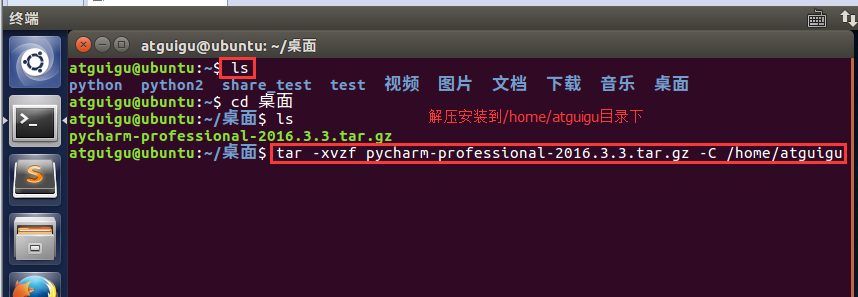
## **2.3. 安装**

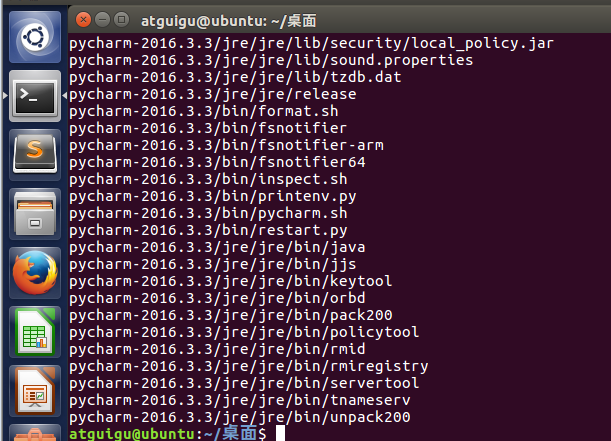
### 2.3.1. 把pycharm-professional-2016.3.3.tar.gz 拷贝到Linux下



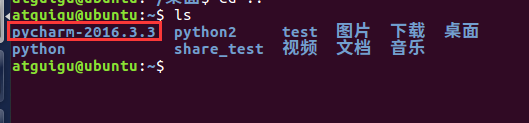
### 2.3.2. 找到pycharm-professional-2016.3.3.tar.gz所在目录，并打算安装到/home/atguigu下

命令：tar -xvzf pycharm-professional-2016.3.3.tar.gz -C /home/atguigu





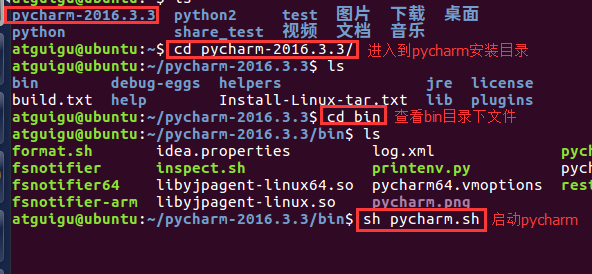
安装好后，会在/home/atguigu下有个名字叫pycharm-2016.3.3目录

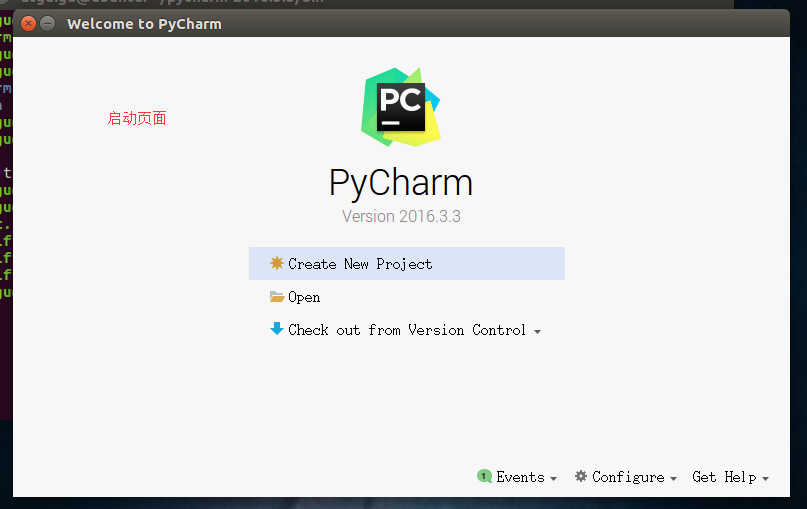


### 2.3.3. 启动PyCharm

进到pycharm-2016.3.3/bin目录

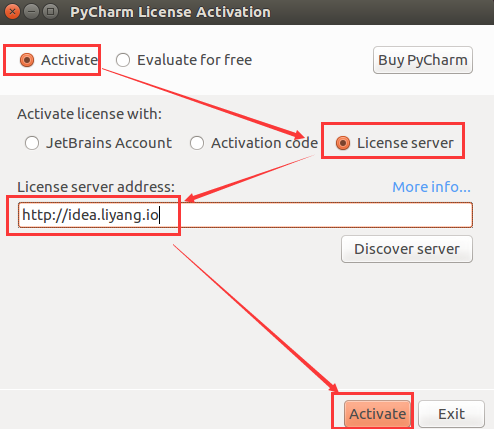
命令：sh pycharm.sh





### 2.3.4. Pycharm 激活码

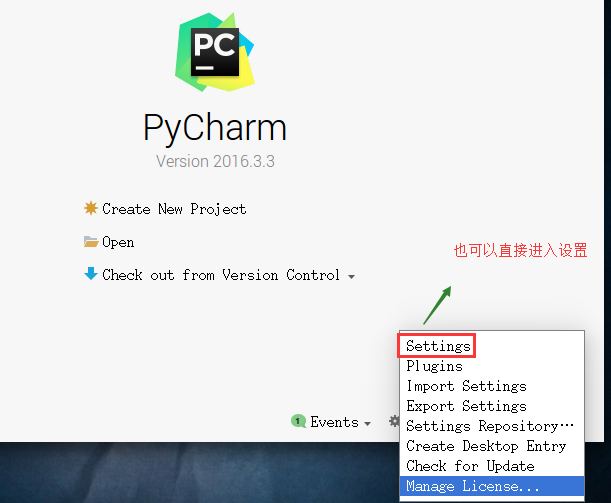
server选项里边输入:  <http://idea.liyang.io>



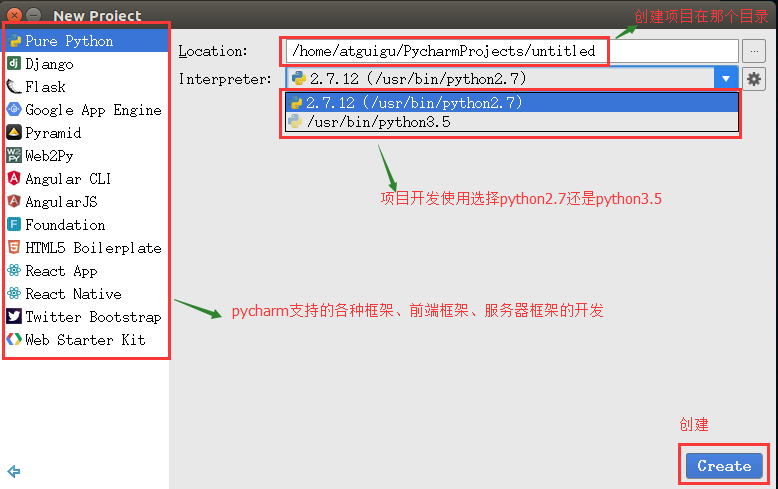
# **3. PyCharm创建Python项目**

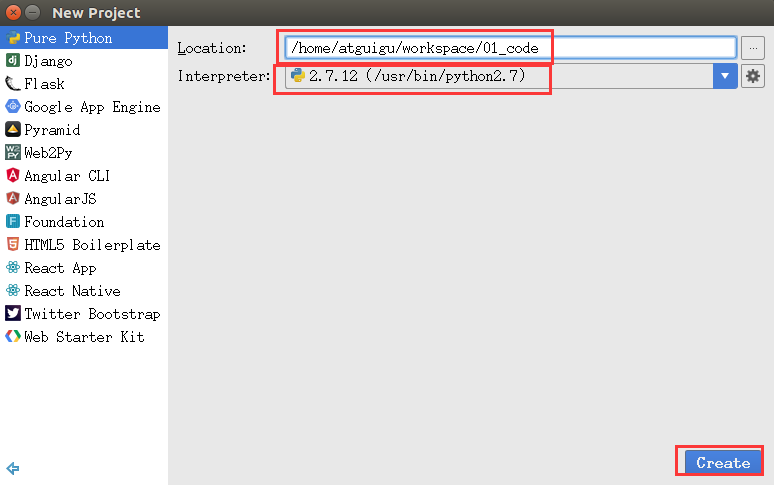
## 3.1. 创建新项目





## 3.2. 选择代码路径和Python解释器版本





django[ˈdʒæŋɡo]

Python下有许多款不同的 Web 框架。Django是重量级选手中最有代表性的一位。许多成功的网站和APP都基于Django。

Django是一个开放源代码的Web应用框架，由Python写成。

Django采用了MVT的软件设计模式，即模型M，视图V和模板T，从MVC发展而来的。

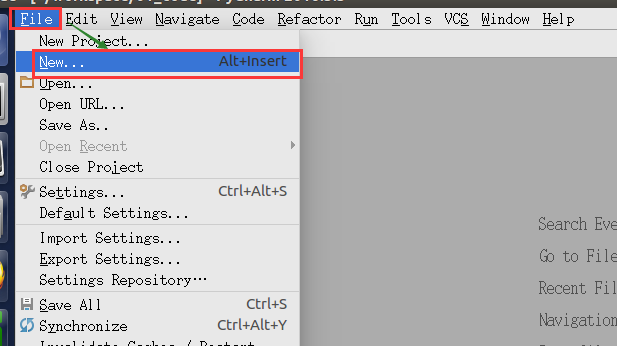
flask[flɑːsk]

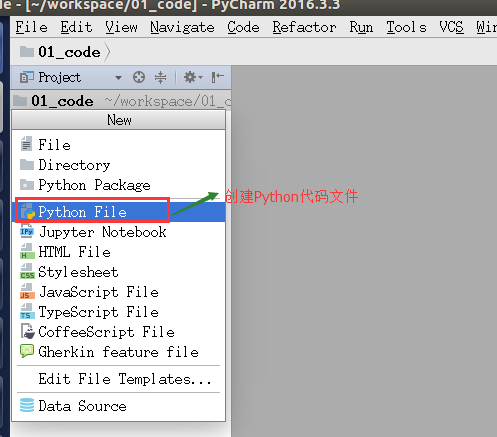
Flask是一个使用 [Python](https://baike.baidu.com/item/Python" \t "https://baike.baidu.com/item/Flask/_blank) 编写的轻量级 Web 应用框架。其 [WSGI](https://baike.baidu.com/item/WSGI" \t "https://baike.baidu.com/item/Flask/_blank) 工具箱采用 Werkzeug ，[模板引擎](https://baike.baidu.com/item/%E6%A8%A1%E6%9D%BF%E5%BC%95%E6%93%8E" \t "https://baike.baidu.com/item/Flask/_blank)则使用 Jinja2 。Flask使用 BSD 授权。

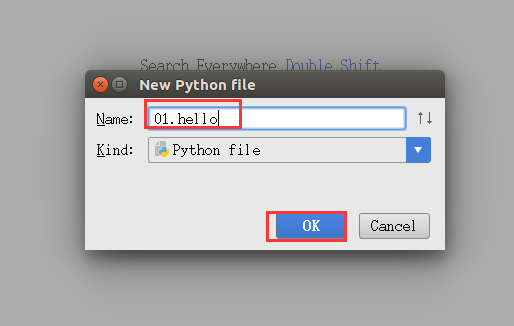
Flask也被称为 “microframework” ，因为它使用简单的核心，用 extension 增加其他功能。Flask没有默认使用的数据库、窗体验证工具。

Interpreter，解释器

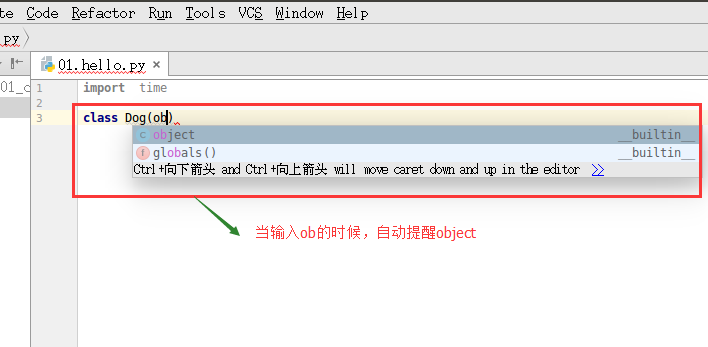
## 3.3. 创建Pyhton源文件



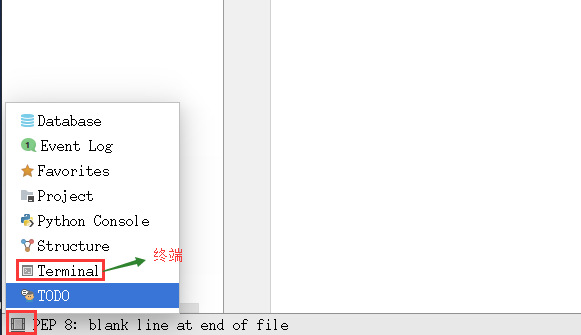




## 3.4. 可用自动提醒代码



## 3.5. 左下方按钮说明



Pycharm按照pep8的python规范检查我们写的代码，如果不对就会报红叉。强制我们遵循pep8规范。

## 3.6. 使用Pycharm完成下面代码

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

from socket import \*

num = 100

#创建套接字

server = socket(AF\_INET, SOCK\_STREAM)

#设置重新运行的时候端口不被占用

server.setsockopt(SOL\_SOCKET, SO\_REUSEADDR, 1)

server.bind(('', 7788))

#绑定端口

server.listen(10)

while True:

newSocket,address = server.accept()

while True:

recvData = newSocket.recv(1024)

if len(recvData) > 0:

print("来自[%s]的数据：%s"%(str(address),recvData))

else:

#

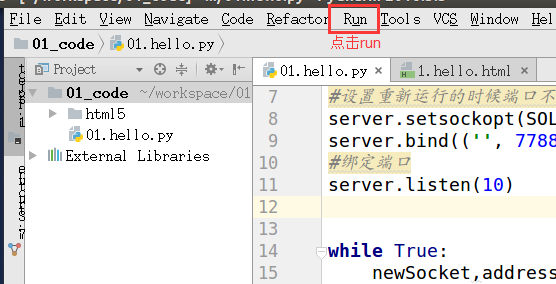
print("%s客户端关闭"%str(address))

newSocket.close()

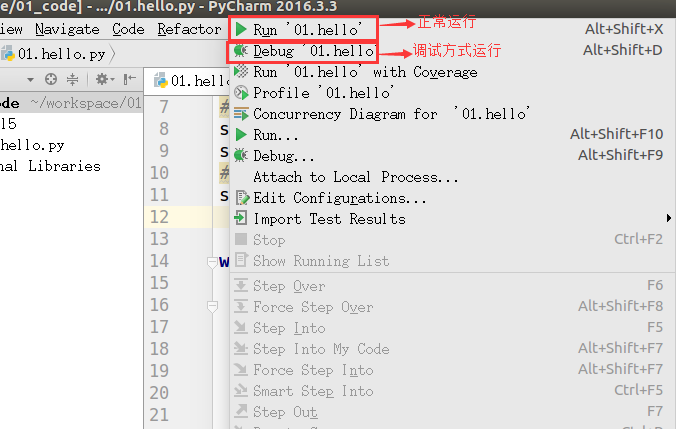
break

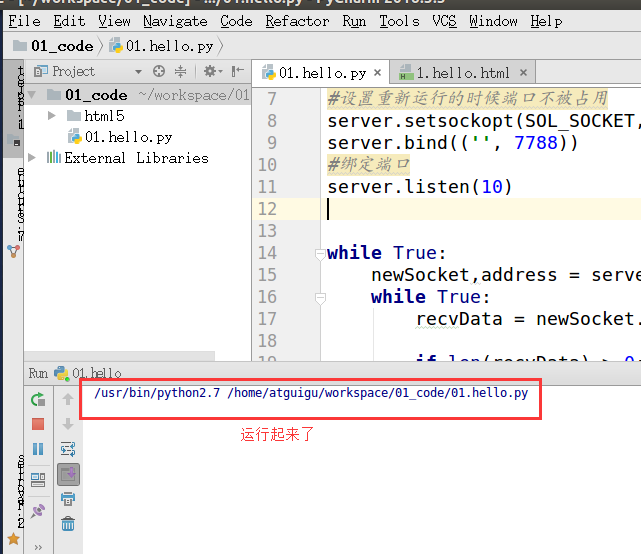
server.close()

## 3.7. 运行程序

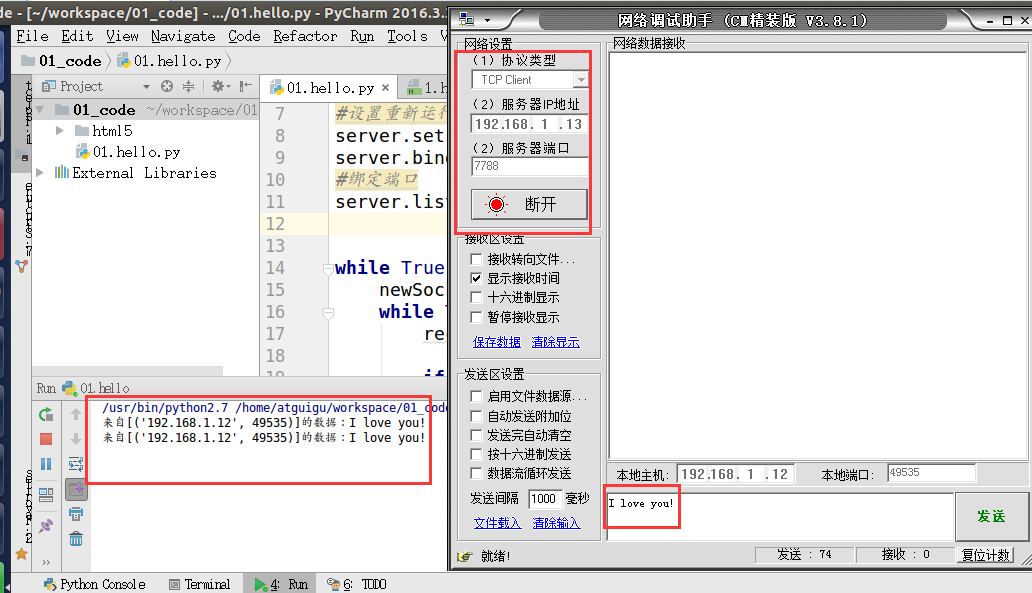


运行

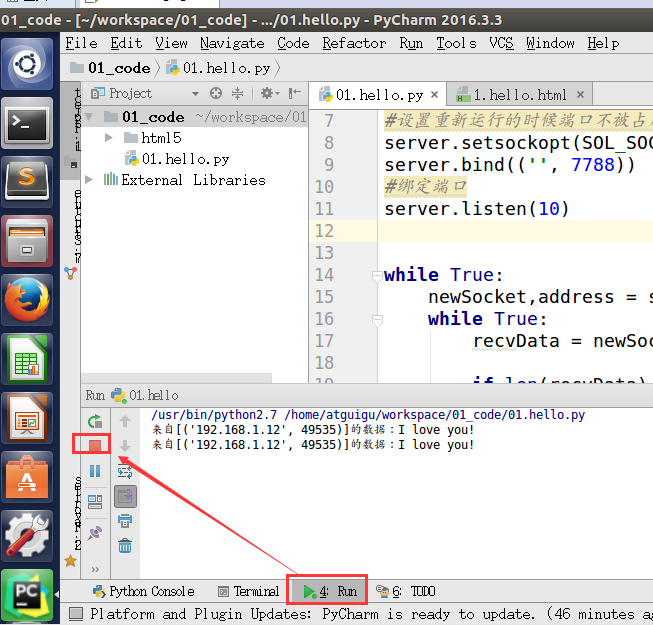




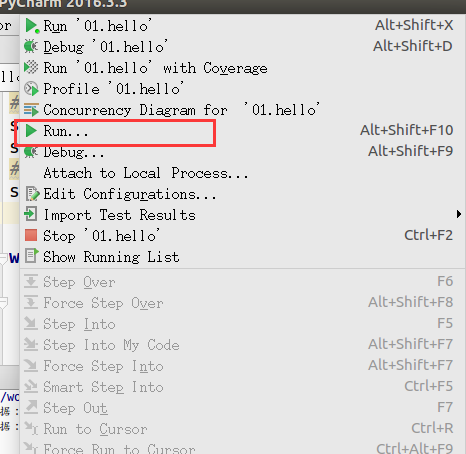
网络调试助手连接上

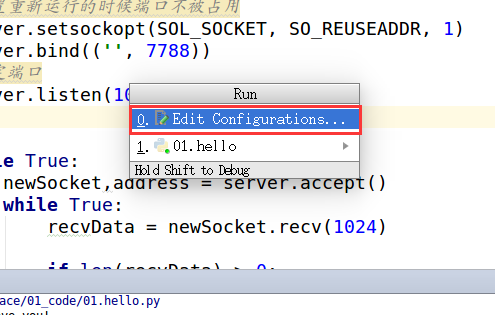


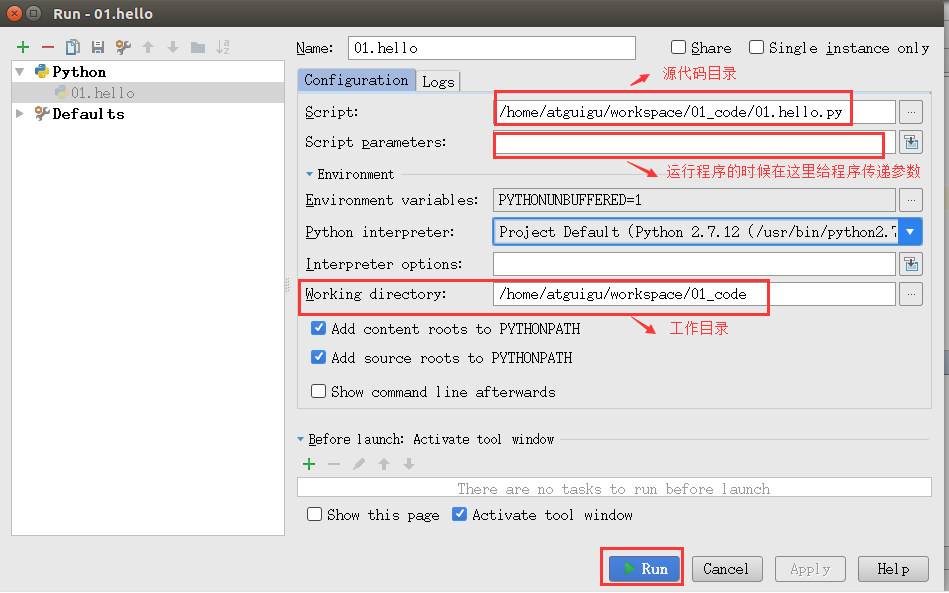
## 3.8. 停止程序



## 3.9. 可以配置方式运行

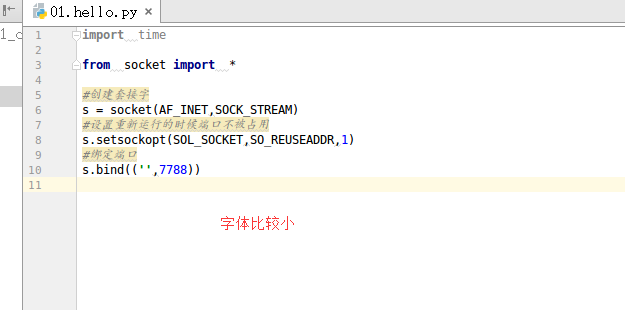




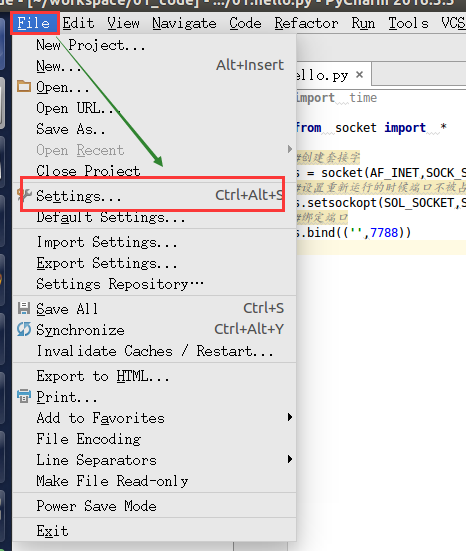


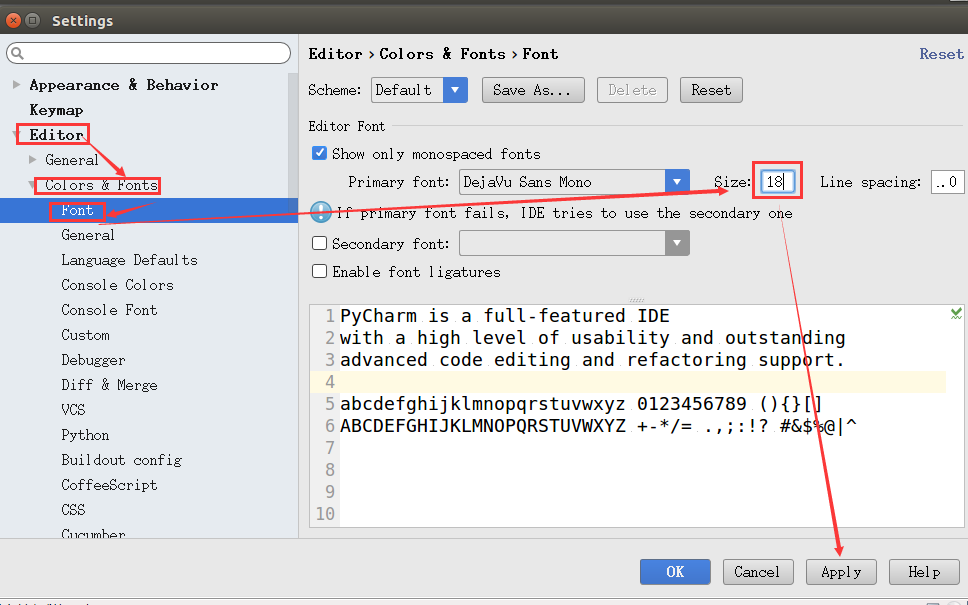
# **4. 设置字体**

## 4.1. 设置前字体

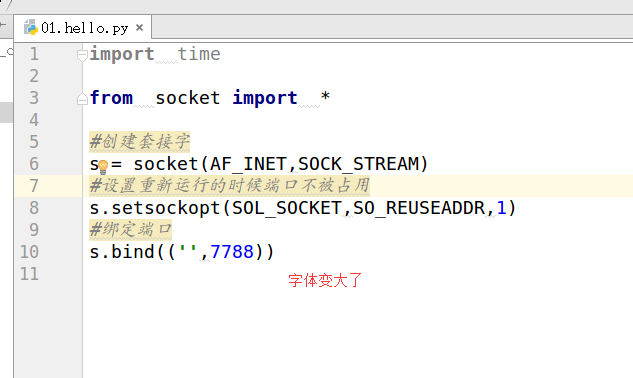


## 4.2. 设置步骤



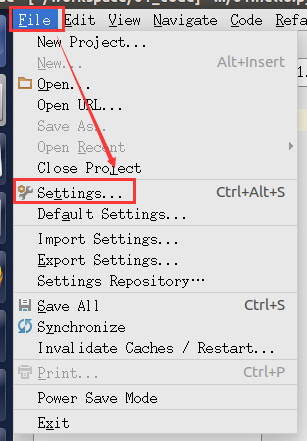


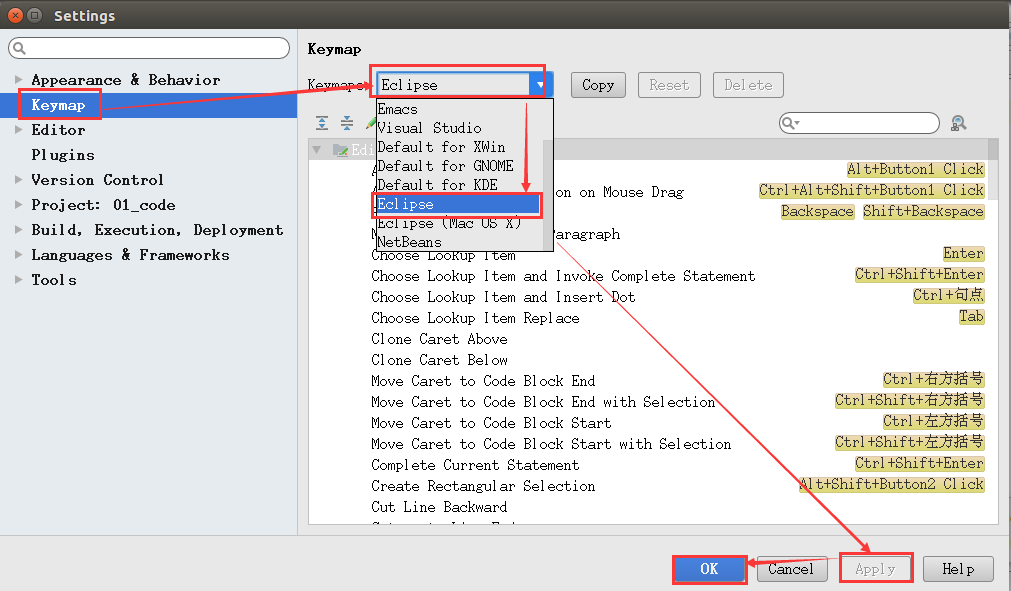
## 4.3. 设置后字体



# **5. 设置快捷键**

## 5.1. 设置步骤





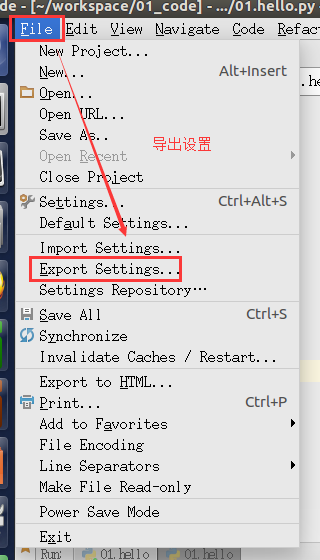
## 5.2. 设置PyCharm为Eclipse快捷键

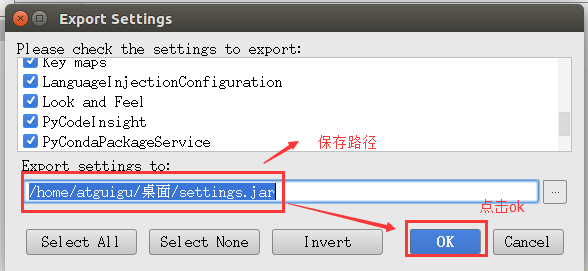
|  |
| --- |
| **Ctrl + O  根据name模糊查找当前文件中类、方法**  Alt + (向左箭头或者向右箭头) ，回退or前进到到之前查看或者编辑处  Alt + (向上箭头或者向下箭头) ，将当前方法整体往下或者往上移动  Ctrl+Shift+R 根据name模糊查找某个资源文件  Ctrl+Shift+T 根据name模糊查找当前project中某个类  **Ctrl+/  注释，取消注释代码**  **Ctrl+D 删除当前行代码**  **Ctrl+ H 整个项目中查找**  Ctrl+ S（没作用，pycharm是自动保存）  F3与Ctrl+鼠标左键效果一样：跳转到方法定义处  Shift+Enter 在行中编辑自动跳转到下一行开头处  Ctrl + Alt + F (Code Formatting)代码格式化(Ctrl + Alt + L是ubuntu的锁屏)  Ctrl + Shift + ]/[ 选定代码块结束、开始 Alt + Enter 快速修正-->Alt + /  Ctrl+Shilf+F12 最大最小化当前编辑窗口（相当于Eclipse ctrl+M）  Ctrl + F6 窗口间切换  Ctrl +（-/+） 折叠放开代码  **Alt+Shift+R 将当前文件重命名**（Eclipse中用F2、当时pycharm被冲突了）  单步调试按钮（F5/F6/F7/F8），自己多用几次就熟悉了，此处不讲。 |

# PyCharm常用设置

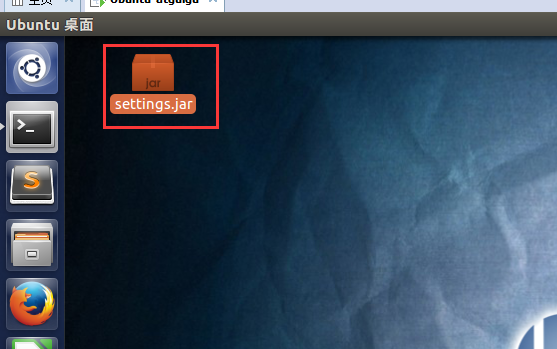
## 6.1. 保存设置

pycharm中的设置是可以导入和导出的，file>export settings可以保存当前pycharm中的设置为jar文件



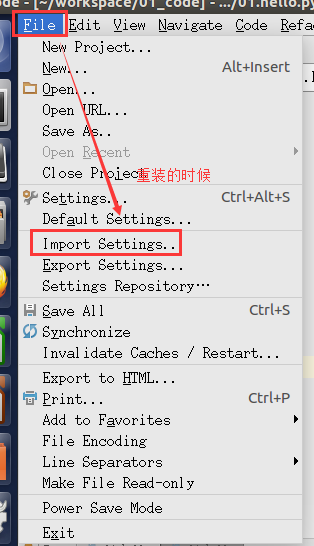


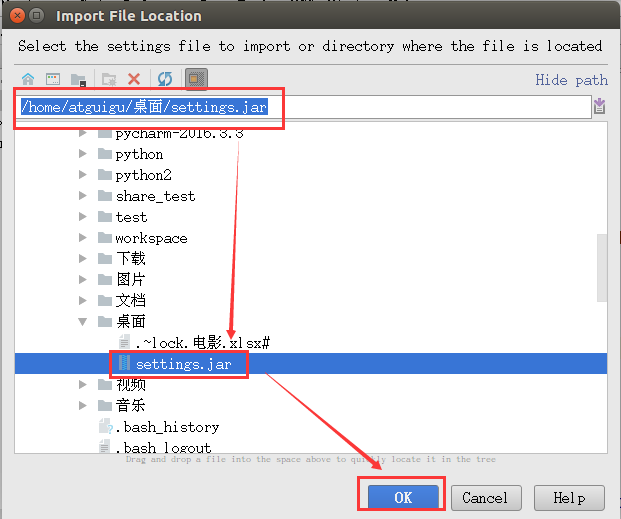
保存在桌面上



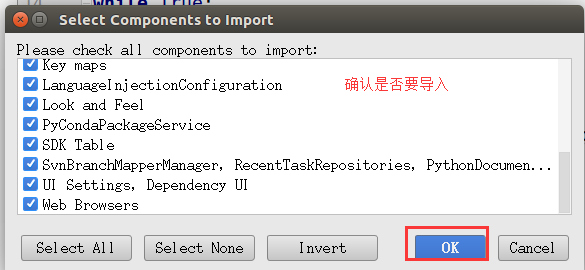
## 6.2. 导入设置

重装时可以直接import settings>jar文件，就不用重复配置了

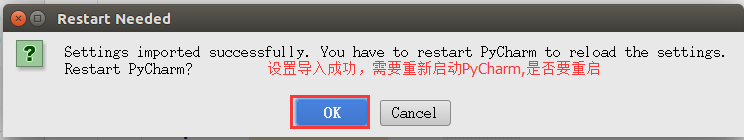




确认是否要导入



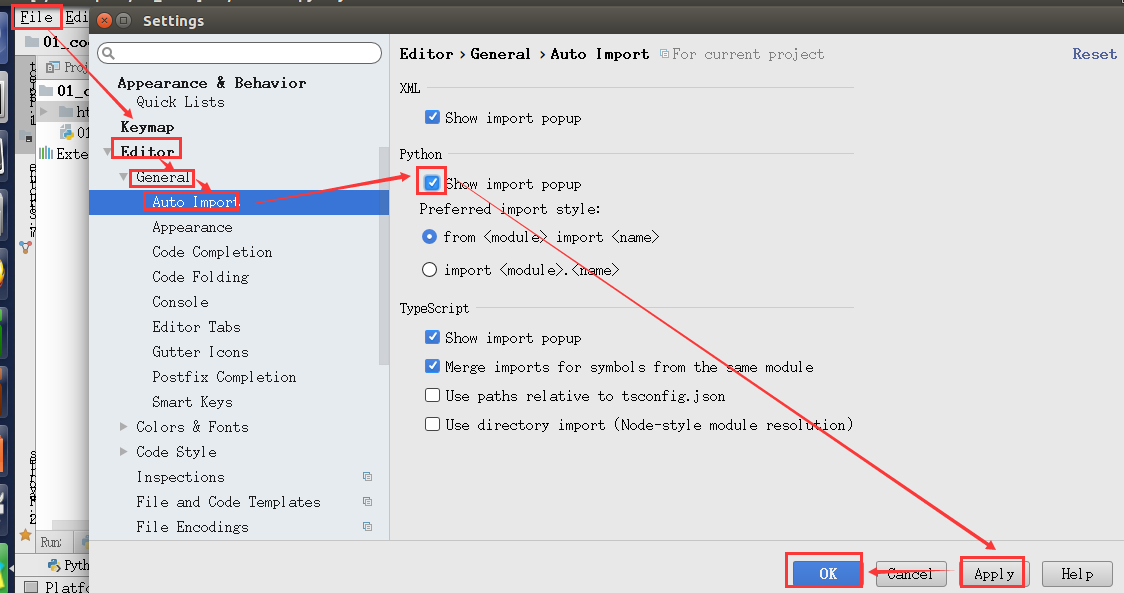
点击确认重新启动



## 6.3. 设置Python自动引入包

设置Python自动引入包，要先在 >general > autoimport -> python :show popup

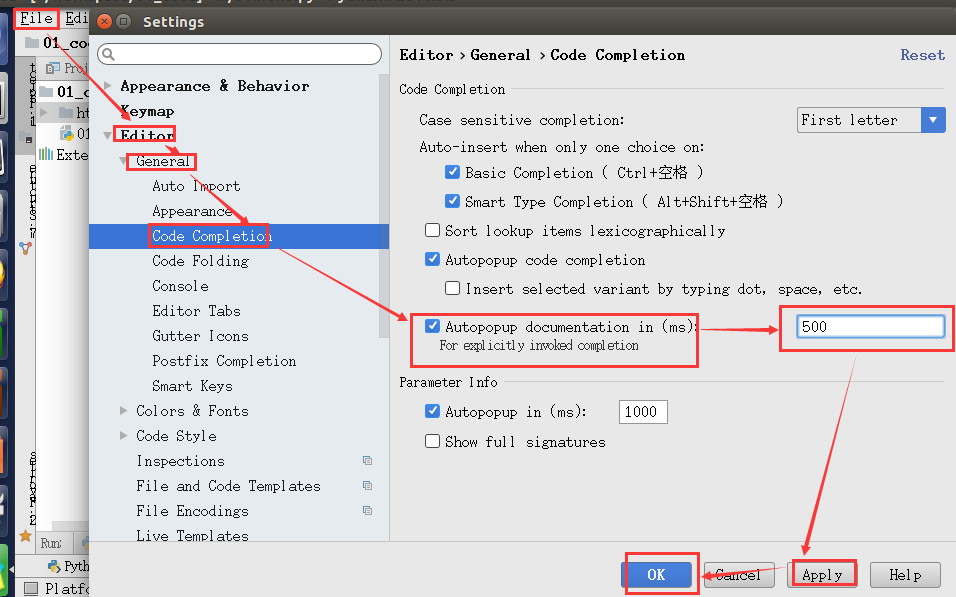
     快捷键：Alt + Enter: 自动添加包



## 6.4. 代码自动完成”时间延时设置

“代码自动完成”时间延时设置”

  > Code Completion   -> Auto code completion in (ms):0  -> Autopopup in (ms):500



## 6.5. 设置缩进符为制表符“Tab”

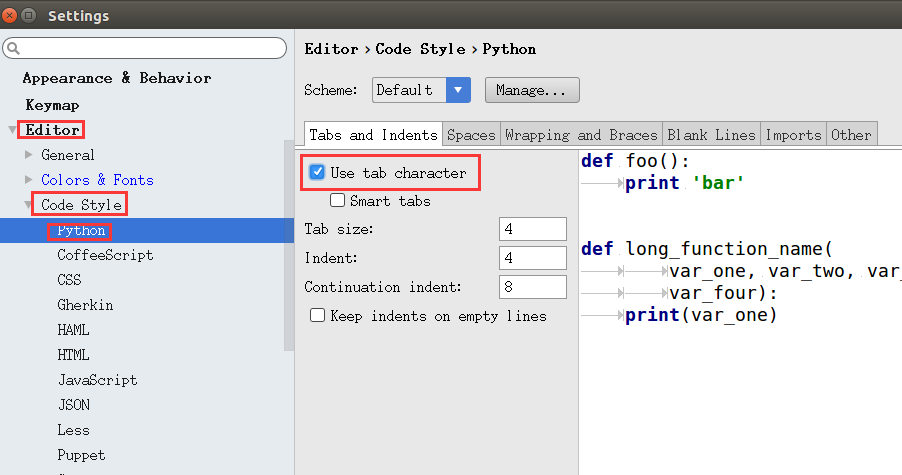
设置缩进符为制表符“Tab”

  File -> Default Settings -> Code Style

  -> General -> 勾选“Use tab character”

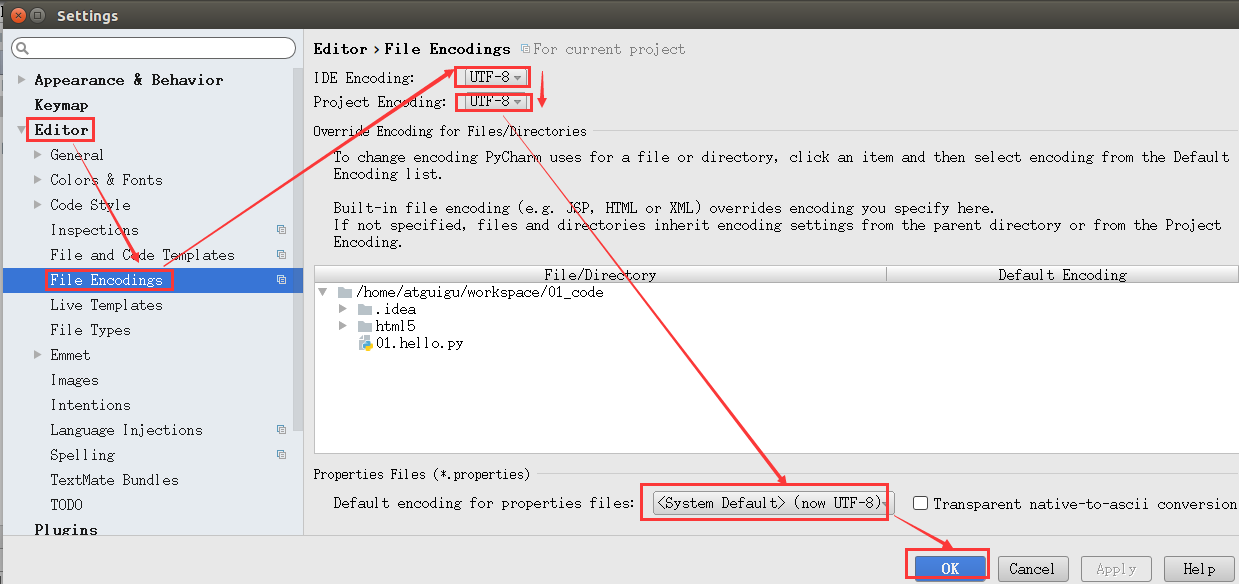
  -> Python -> 勾选“Use tab character”

  -> 其他的语言代码同理设置



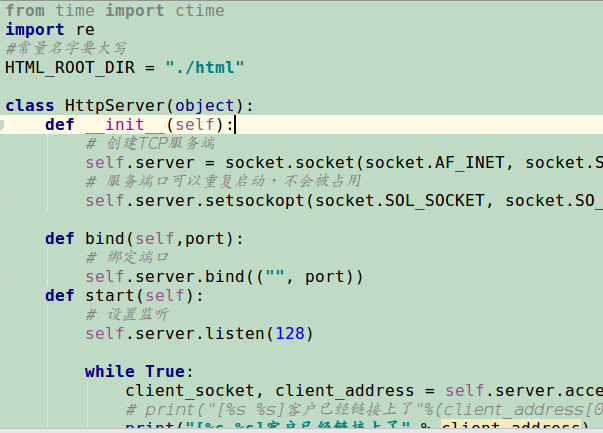
## 6.6. Python文件默认编码

File Encodings> IDE Encoding: UTF-8;Project Encoding: UTF-8;



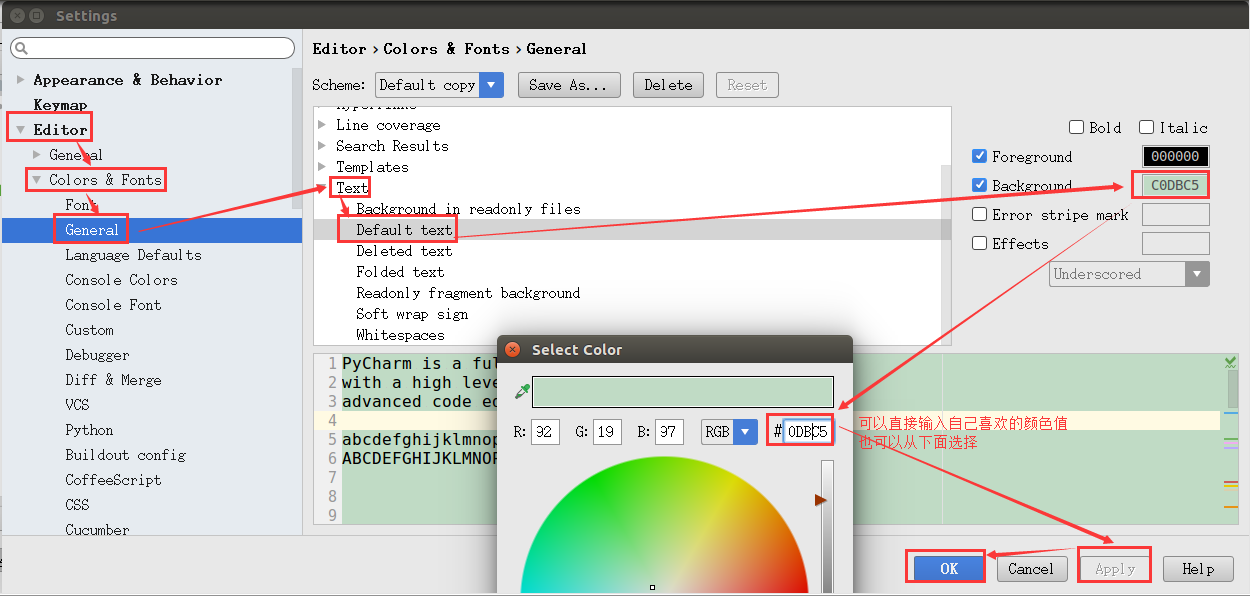
## 6.7. 设置代码背景

### 6.7.1. 要设置成的效果



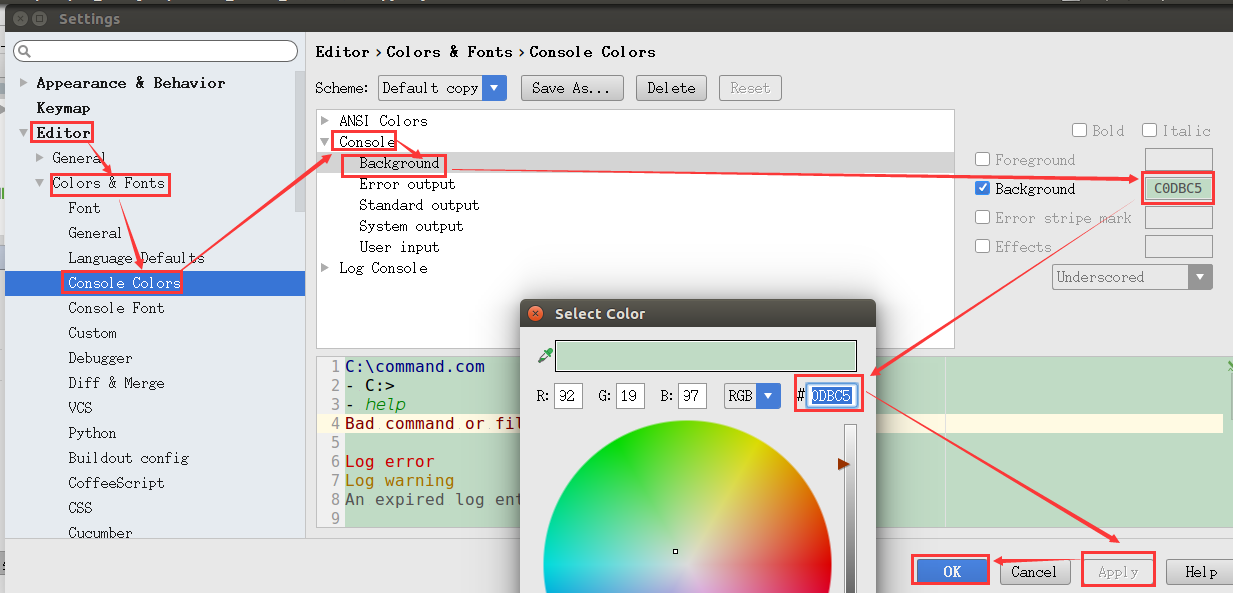
### 6.7.2. 设置步骤如图

设置代码编辑区背景



颜色值为：#C0DBC5

设置控制台背景颜色



# **调试**

## 7.1. PyCharm提供的debug功能

PyCharm提供了单步调试代码的功能。

优点：

提供图形化界面，很直观；功能强大；不需要修改代码

缺点：没有

### 7.1.1. 测试源代码

# 定义人类：老王和老宋都属于人

class Person(object):

def \_\_init\_\_(self, name):

self.name = name # 人名

self.gun = None

self.hp = 100 # 血量，为0的时候，人已挂

# 安装子弹，把子弹装入弹夹里面

def anzhuang\_zidan(self, dan\_jia, zi\_dan):

# 弹夹.保存子弹(子弹)

dan\_jia.bao\_cun\_zidan(zi\_dan)

# 安装弹夹-把弹夹装入枪里面

def anzhuang\_danjia(self, gun, dan\_jia):

# 枪.保存弹夹(弹夹)

gun.bao\_cun\_danjia(dan\_jia)

# 在实例对象中保存枪这个对象

def naqiang(self, gun):

# 保存到对应的对象中了

self.gun = gun

# 开枪

def kaiqiang(self, diren):

# 老王手中已经有枪

# 枪.开火(敌人)

self.gun.fire(diren)

# 人出血

def diao\_xue(self, sha\_shang\_li):

self.hp -= sha\_shang\_li

def \_\_str\_\_(self):

if self.gun:

# 有枪

if self.hp > 0:

msg = "%s有枪，当前血量%s" % (self.name, self.hp)

else:

msg = "%s已经挂" % (self.name)

return msg

else:

# 没有枪

if self.hp > 0:

msg = "%s没有枪，当前血量%s" % (self.name, self.hp)

else:

msg = "%s已经挂" % (self.name)

return msg

# 定义一个枪

class Gun(object):

def \_\_init\_\_(self, name):

self.name = name # 枪名

self.dan\_jia = None

# 保存弹夹-安装弹夹

def bao\_cun\_danjia(self, dan\_jia):

# 枪的这个对象就有弹夹这个对象了

self.dan\_jia = dan\_jia

def \_\_str\_\_(self):

msg = "当前%s信息如下：%s" % (self.name, self.dan\_jia)

return msg

# 枪开火

def fire(self, diren):

# 得到一颗子弹

# 子弹 = 弹夹.弹出子弹()#弹夹会少子弹

zi\_dan = self.dan\_jia.tanchu\_zidan()

# 子弹.让敌人掉血(敌人)

zi\_dan.diao\_xue(diren)

# 弹夹类

class DanJia(object):

def \_\_init\_\_(self, max\_num):

self.max\_num = max\_num # 最多能装多少颗子弹

self.zi\_dan\_list = [] # 装子弹的

# 保存子弹，以后子弹就在弹夹里面了，而且子弹不只一颗

def bao\_cun\_zidan(self, zi\_dan):

# 把子弹装到弹夹的列表中

self.zi\_dan\_list.append(zi\_dan)

def \_\_str\_\_(self):

msg = "最多支持子弹：%s颗,当前子弹：%s颗" % (self.max\_num, len(self.zi\_dan\_list))

return msg

# 弹出子弹

def tanchu\_zidan(self):

# 判断弹夹有没有子弹

if len(self.zi\_dan\_list) > 0:

# 取最后一颗子弹

zi\_dan = self.zi\_dan\_list.pop()

return zi\_dan

else:

print("弹夹里面没有子弹了")

return None

# 子弹类

class ZiDan(object):

def \_\_init\_\_(self, sha\_shang\_li):

self.sha\_shang\_li = sha\_shang\_li # 子弹的杀伤力

# 让敌人掉血

def diao\_xue(self, diren):

# 敌人.掉血(子弹的杀伤力)

diren.diao\_xue(self.sha\_shang\_li)

class Killings(object):

# y游戏的入口

def start\_game(self):

# 1.创建一个老王对象

laowang = Person("老王")

# 2.创建一个枪的对象

gun = Gun("AK47")

# 3.创建一个弹夹对象

dan\_jia = DanJia(20) # 最多支持20颗

# 4.安装子弹，把子弹装入(保存)弹夹

for i in range(15):

# 创建了15颗子弹

zi\_dan = ZiDan(10)

# 老王.安装子弹(弹夹，子弹)

laowang.anzhuang\_zidan(dan\_jia, zi\_dan)

# 打印弹夹的信息

print(dan\_jia)

# 5.安装弹夹，把弹夹装入枪里面

# 老王.安装弹夹(枪,弹夹)

laowang.anzhuang\_danjia(gun, dan\_jia)

# 打印枪里面的信息

print(gun)

# 6.打印信息

# 7.老拿枪

laowang.naqiang(gun)

# 8.创建隔壁老宋对象

gebi\_laosong = Person("隔壁老宋")

# 9.持续的对老宋开枪

# 老王.开枪(隔壁老宋)

for i in range(15):

laowang.kaiqiang(gebi\_laosong)

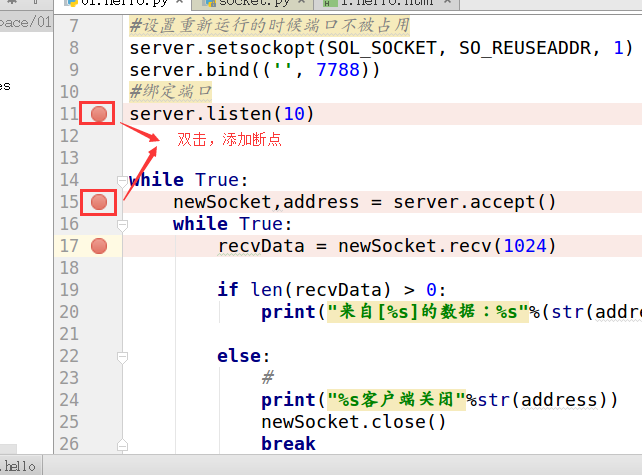
print(gebi\_laosong)

print(laowang)

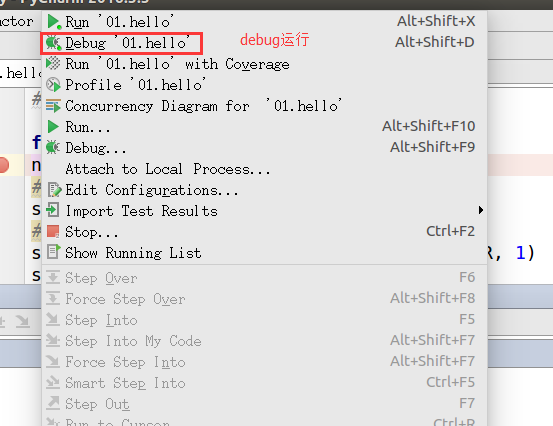
k = Killings()

k.start\_game()

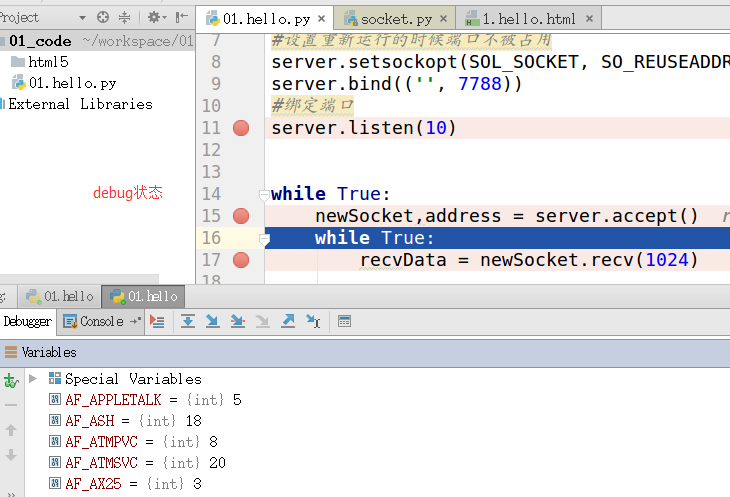
### 7.1.2. 添加断点



### 7.1.3. debug运行



调试中的debug



## 7.2. print

### 7.2.1. 用print调试代码是最简单、最常用的方式

用print调试代码是**最简单的一种方法，也很常**用，大部分人都掌握这种技巧。

在代码中合适的地方插入语句，可以是**输出**提示语句或者**输出某些变**量。

**优点：**

比较直观，使用简单

**缺点：**

需要入侵代码，也就是要修改代码

### 7.2.1. 找出错误

def test1():

print("----test1-1----")

print(num)

print("----test1-2----")

def test2():

print("----test2-1----")

test1()

print("----test2-2----")

def test3():

print("----test3-1----")

test1()

print("----test3-2----")

test3()

print("------华丽的分割线-----")

## **7.3.** 利用l**ogging模块日志调试**

### 7.3.1. l**ogging模块简介**

就是利用l**ogging模块**，在代码合适的地方插入输出log语句，把合适的信息输出到**log文件中**，通过查看log文件分析代码的执行情况。

优点：

logging模块可以指定输出格式和内容（可以输出时间，代码的行号，当前运行程序的名称、当前运行的函数名称、进程ID等等），因此能获得更精确的调试信息，应用场景更广泛，可以应用于**生产环境**。

**服务器不需要关闭，直接查看日志分析错误。**

**发现问题后解决，再用热修复技术把错误修补上。**

缺点：

需要入侵代码，也就是要修改代码；使用更复杂。

<http://www.aihami.com/a/dangjian/zugong/216808_2.html>

### 7.3.2. logging将日志打印到屏幕

默认情况下，logging将日志打印到屏幕，日志级别为WARNING；  
日志级别大小关系为：CRITICAL > ERROR > WARNING > INFO > DEBUG > NOTSET，当然也可以自己定义日志级别。

打印到终端

import logging

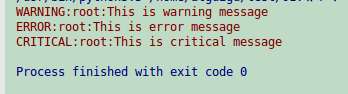
logging.debug('This is debug message')

logging.info('This is info message')

logging.warning('This is warning message')

logging.error("This is error message")

logging.critical("This is critical message")

运行结果  


### **7.3.3. 通过logging.basicConfig函数对日志的输出格式及方式做相关配置**

logging.basicConfig函数各参数:  
filename: 指定日志文件名  
filemode: 和file函数意义相同，指定日志文件的打开模式，'w'或'a'  
format: 指定输出的格式和内容，format可以输出很多有用信息，如上例所示:  
 %(levelno)s: 打印日志级别的数值  
 %(levelname)s: 打印日志级别名称  
 %(pathname)s: 打印当前执行程序的路径，其实就是sys.argv[0]  
 %(filename)s: 打印当前执行程序名  
 %(funcName)s: 打印日志的当前函数  
 %(lineno)d: 打印日志的当前行号  
 %(asctime)s: 打印日志的时间  
 %(thread)d: 打印线程ID  
 %(threadName)s: 打印线程名称  
 %(process)d: 打印进程ID  
 %(message)s: 打印日志信息  
datefmt: 指定时间格式，同time.strftime()  
level: 设置日志级别，默认为logging.WARNING  
stream: 指定将日志的输出流，可以指定输出到sys.stderr,sys.stdout或者文件，默认输出到sys.stderr，当stream和filename同时指定时，stream被忽略

import logging

logging.basicConfig(level=logging.DEBUG,

format='%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s',

datefmt='%a, %d %b %Y %H:%M:%S',

filename='myapp.log',

filemode='w')

logging.debug('This is debug message')

logging.info('This is info message')

logging.warning('This is warning message')

**./myapp.log文件中内容为**

Sun, 10 Dec 2017 22:37:36 test2.py[line:11] DEBUG This is debug message

Sun, 10 Dec 2017 22:37:36 test2.py[line:12] INFO This is info message

Sun, 10 Dec 2017 22:37:36 test2.py[line:13] WARNING This is warning message

### **7.3.4. 将日志同时输出到文件和屏幕**

import logging

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s',

datefmt='%a, %d %b %Y %H:%M:%S',

filename='myapp.log',

filemode='w')

#################################################################################################

#定义一个StreamHandler，将INFO级别或更高的日志信息打印到标准错误，并将其添加到当前的日志处理对象#

console = logging.StreamHandler()

console.setLevel(logging.INFO)

formatter = logging.Formatter('%(name)-12s: %(levelname)-8s %(message)s')

console.setFormatter(formatter)

logging.getLogger('').addHandler(console)

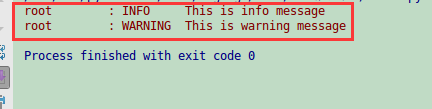
#################################################################################################

logging.debug('This is debug message')

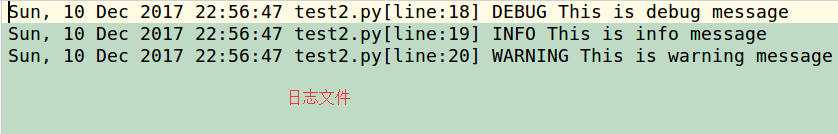
logging.info('This is info message')

logging.warning('This is warning message')

运行效果



**日志文件myapp.log内容如下**



## **7.4. pdb**

pdb 是 python 自带的一个包，为 python 程序提供了一种交互的源代码调试功能，主要特性包括设置断点、单步调试、进入函数调试、查看当前代码、查看栈片段、动态改变变量的值等。

优点：

功能强大，使用简单

缺点：

需要入侵代码，也就是要修改代码；使用更复杂。

pdb是基于命令行的调试工具，非常类似gnu的gdb（调试c/c++）。

| **命令** | **简写命令** | **作用** |
| --- | --- | --- |
| break | b | 设置断点 |
| **continue** | **c** | **继续执行程序** |
| **list** | **l** | 查看当前行的代码段 |
| **step** | **s** | 进入函数 |
| return | r | 执行代码直到从当前函数返回 |
| quit | q | 中止并退出 |
| **next** | **n** | 执行下一行 |
| print | p | 打印变量的值 |
| help | h | 帮助 |
| args | a | 查看传入参数 |
|  | 回车 | 重复上一条命令 |
| break | b | 显示所有断点 |
| break lineno | b lineno | 在指定行设置断点 |
| break file:lineno | b file:lineno | 在指定文件的行设置断点 |
| clear num |  | 删除指定断点 |
| bt |  | 查看函数调用栈帧 |

### 7.4.1. 常用操作

**l---->list 显示当前的代码**

**n---->next 向下执行一行代码**

**c---->continue 继续执行代码**

**b---->break 添加断点**

**clear -->删除断点**

**p---->print 打印一个变量的值**

**s---->step 进入到一个函数**

**a---->args 打印所有的形参数据**

**q---->quit退出调试**

**r---->return 快速执行到函数的最后一行**

### **7.4.2. 执行时调试**

test.py

def get\_num(a,b):

result = a + b

print("result==%d"%result)

return result

a = 100

b = 200

res = get\_num(a,b)

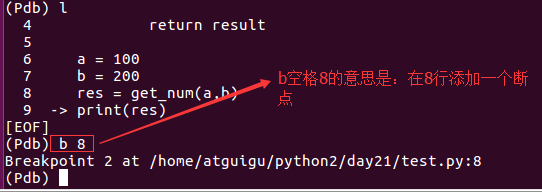
print(res)

程序启动，停止在第一行等待单步调试。

python -m pdb test.py

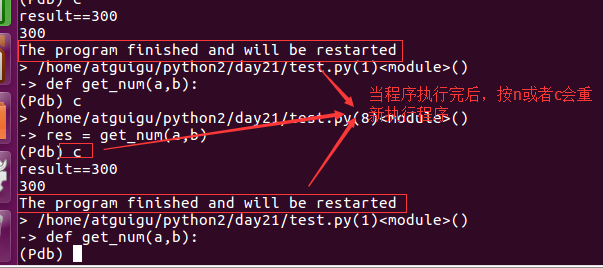
#### 添加和删除断点

#### 添加断点

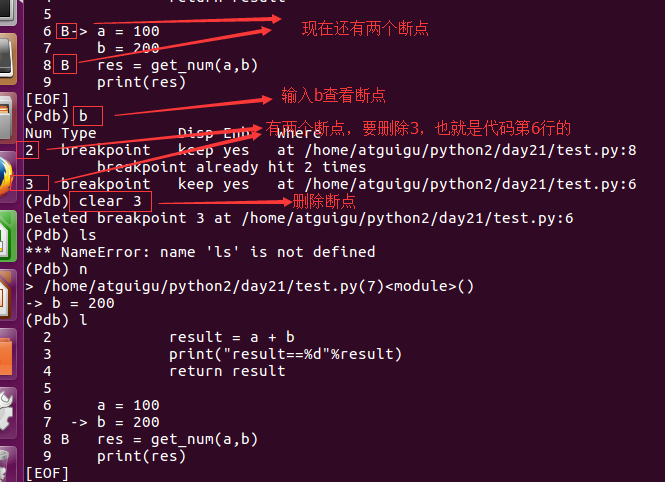


添加断点后，代码可以继续执行，当知道到这里的时候会停留，特别在代码很多的情况下很有用，可以用来跳过不关心的代码，关心的代码部分可以设置断点。

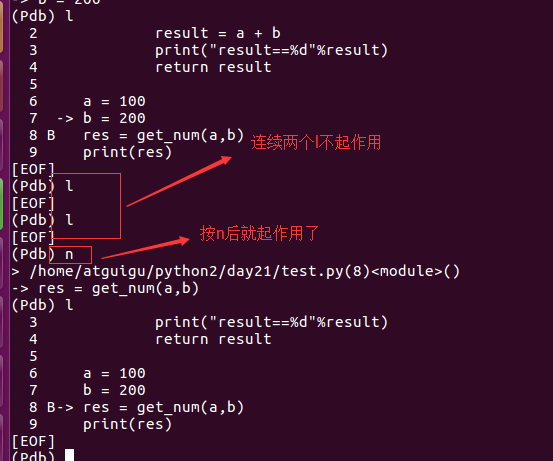
#### 当程序快执行完了的时候，按n或者c会重新执行程序



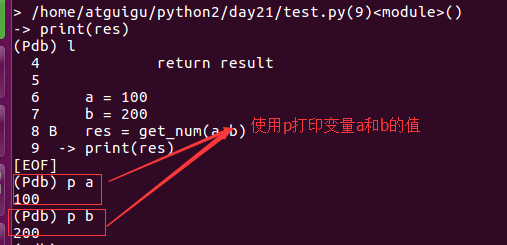
#### 删除断点



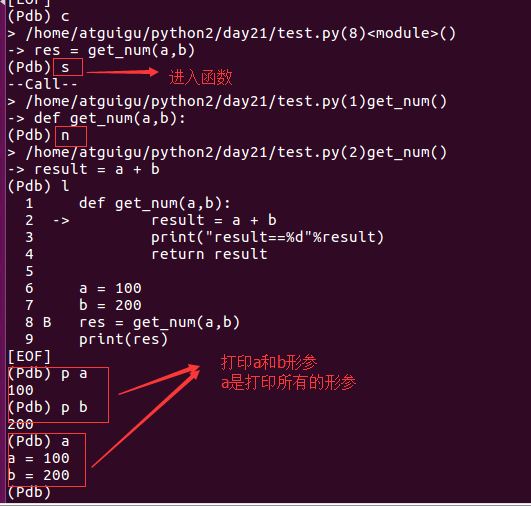
#### 如果连续执行两次l不起作用，如果按n执行一次，就可以按l了



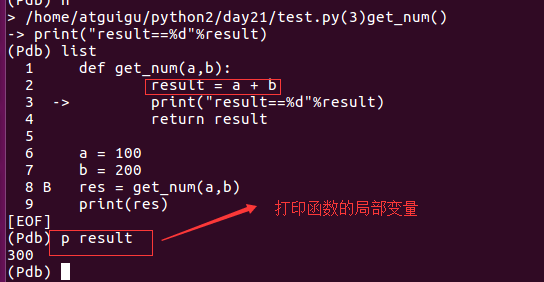
#### 使用p打印变量的值或者函数的形参



#### 使用a打印函数所有形参

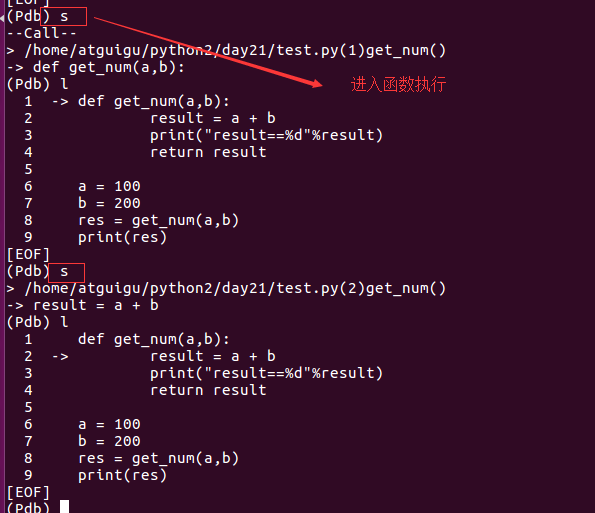


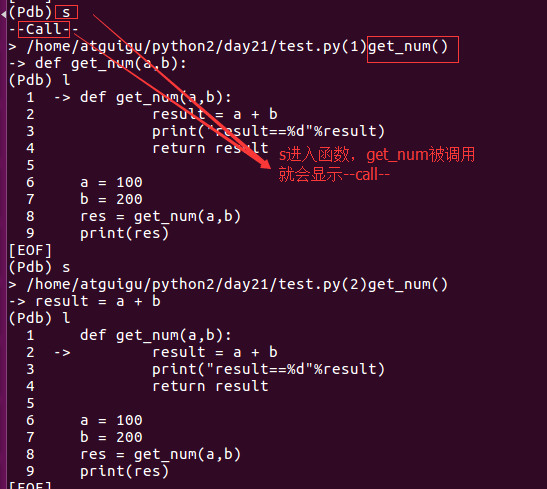




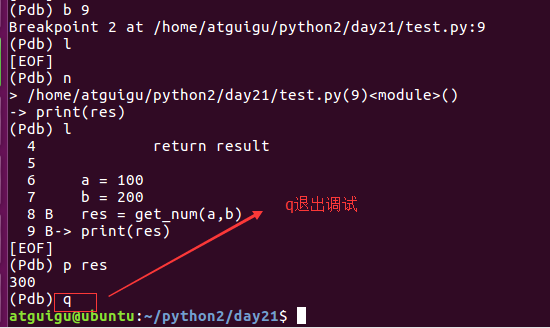
注意：如果要打印所有的参数，需要进入函数后才能使用a打印全部形参

#### 使用s进入函数执行





#### 使用q退出



### **7.4.3. 交互调试**

进入python或ipython解释器

import pdb

pdb.run('testfun(args)') #此时会打开pdb调试，注意：先使用s跳转到这个testfun函数中，然后就可以使用l看到代码了



### **7.4.4. 程序里埋点**

当程序执行到pdb.set\_trace() 位置时停下来调试

代码上下文

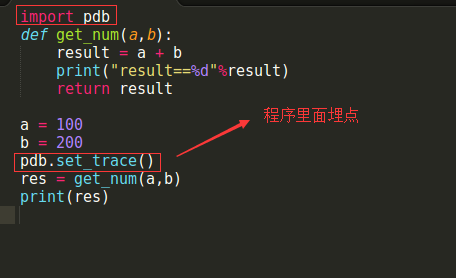
...

import pdb

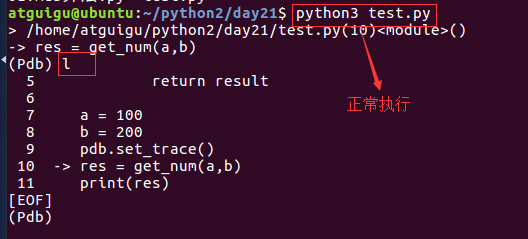
pdb.set\_trace()

...

例如：



执行



### 7.4.5. pdb调试案例

使用pdb调试的demo

#### **练一练:请使用所学的pdb调试技巧对其进行调试出bug**

#coding=utf-8

import pdb

def add3Nums(a1,a2,a3):

result = a1+a2+a3

return result

def get3NumsAvarage(s1,s2):

s3 = s1 + s2 + s1

result = 0

result = add3Nums(s1,s2,s3)/3

if \_\_name\_\_ == '\_\_main\_\_':

a = 11

# pdb.set\_trace()

b = 12

final = get3NumsAvarage(a,b)

print(final)

pdb 调试有个明显的缺陷就是对于多线程，远程调试等支持得不够好，同时没有较为直观的界面显示，不太适合大型的 python 项目。而在较大的 python 项目中，这些调试需求比较常见，因此需要使用更为高级的调试工具。