

nsd1906_py01_day01

我的简书《python百例》：<https://www.jianshu.com/c/00c61372c46a>

配置python虚拟环境

- python虚拟环境只是一个隔离的空间，对应一个文件夹
- 配置好虚拟环境后，安装软件包，可以安装到虚拟环境
- 如果项目完成了，虚拟环境目录可以直接删除

创建虚拟环境

```
[root@room8pc16 ~]# python3 -m venv ~/nsd1906
[root@room8pc16 ~]# ls ~/nsd1906/
bin  include  lib  lib64  pyvenv.cfg
```

激活虚拟环境

```
[root@room8pc16 ~]# source ~/nsd1906/bin/activate
```

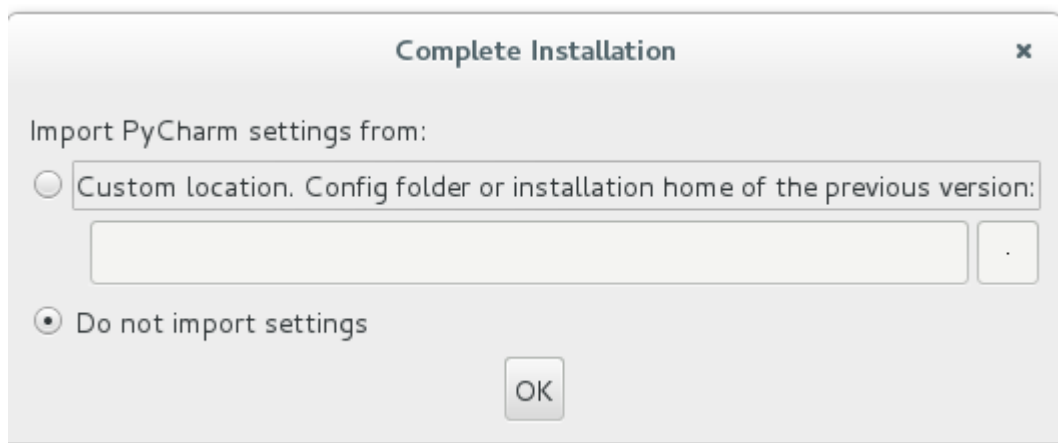
在虚拟环境下使用python，就是python3

```
(nsd1906) [root@room8pc16 ~]# python --version
Python 3.6.7
(nsd1906) [root@room8pc16 ~]# which python
/root/nsd1906/bin/python
```

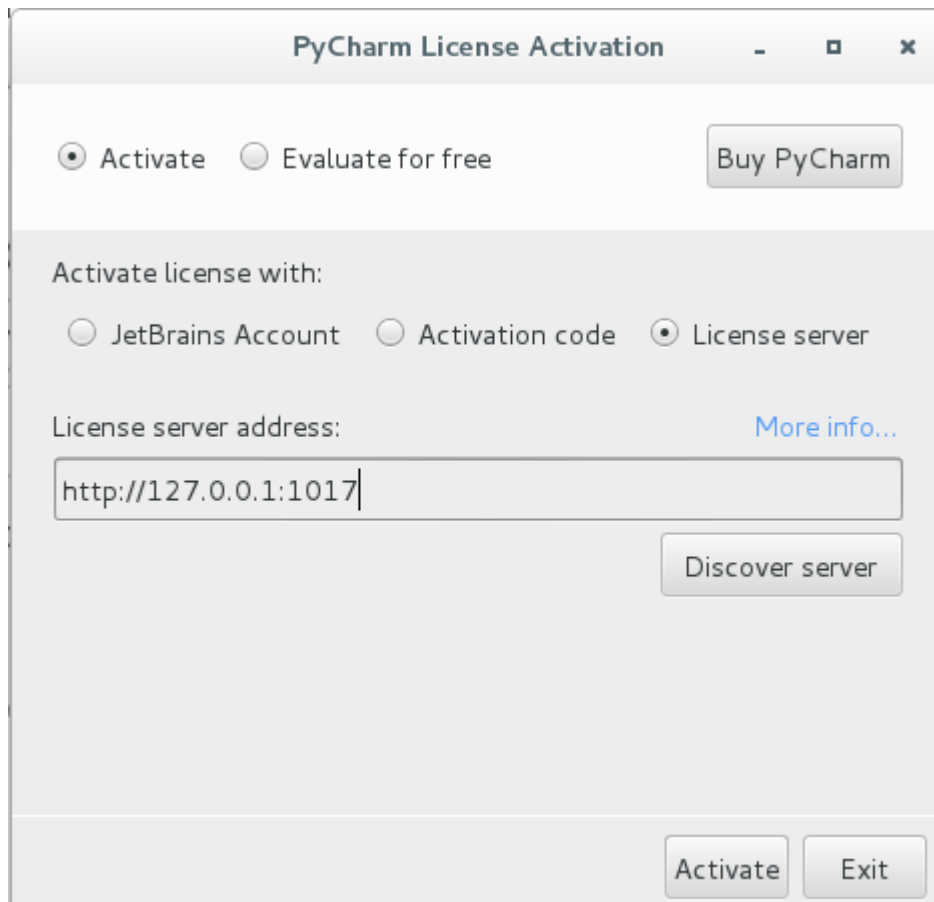
pycharm配置

- pycharm是专门编写python程序的IDE（集成开发环境）

第一次运行pycharm，需要进行相关的配置。首先询问是否要导入以前的配置，选不导入



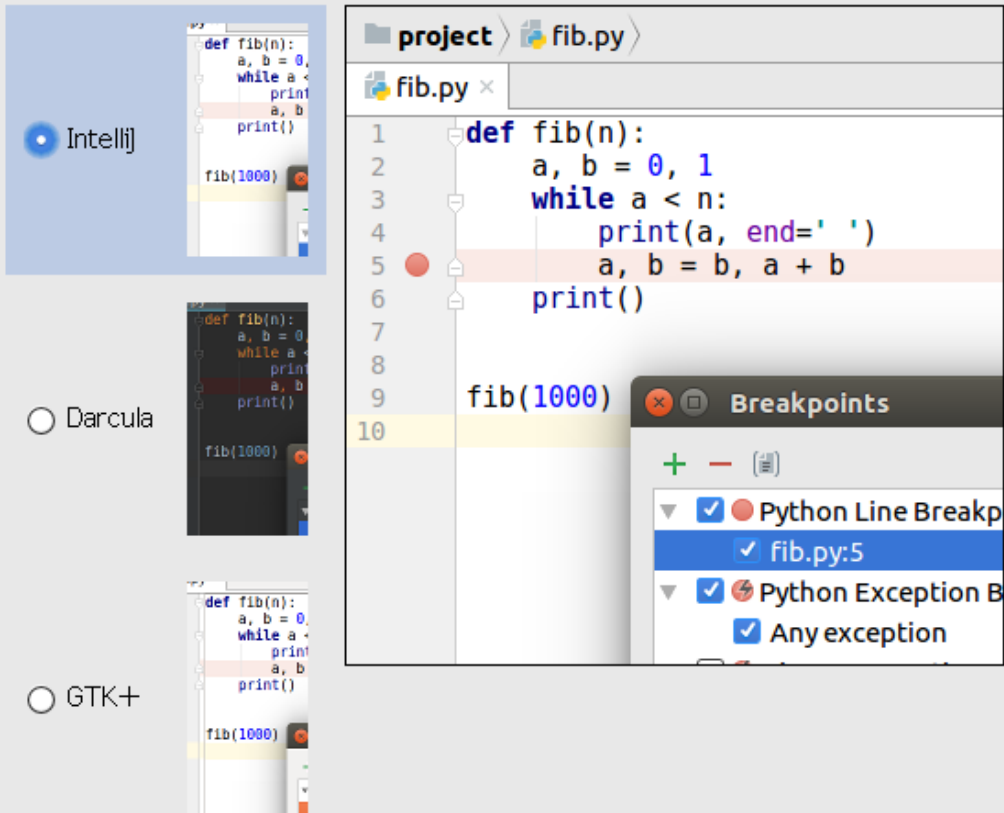
接下来，是授权页，需要购买获得注册码



在选择主题页面，选好主题后，点击Skip Remaining

UI Themes → Launcher Script → Featured plugins

Set UI theme



The screenshot shows the 'Customize PyCharm' dialog with the 'Set UI theme' step. On the left, three theme options are listed: 'IntelliJ' (selected with a blue dot), 'Darcula', and 'GTK+'. Each option has a corresponding preview of the code editor. The 'IntelliJ' preview shows a Python Fibonacci function and a call to 'fib(1000)'. The 'Darcula' preview shows the same code with a dark theme. The 'GTK+' preview shows the same code with a light theme. On the right, a larger preview of the code editor is shown with the 'IntelliJ' theme. The code is as follows:

```
def fib(n):  
    a, b = 0, 1  
    while a < n:  
        print(a, end=' ')  
        a, b = b, a + b  
    print()  
  
fib(1000)
```

A 'Breakpoints' window is also visible, showing a Python Line Breakpoint at line 5 of 'fib.py'.

UI theme can be changed later in Settings | Appearance & Behavior | Appearance

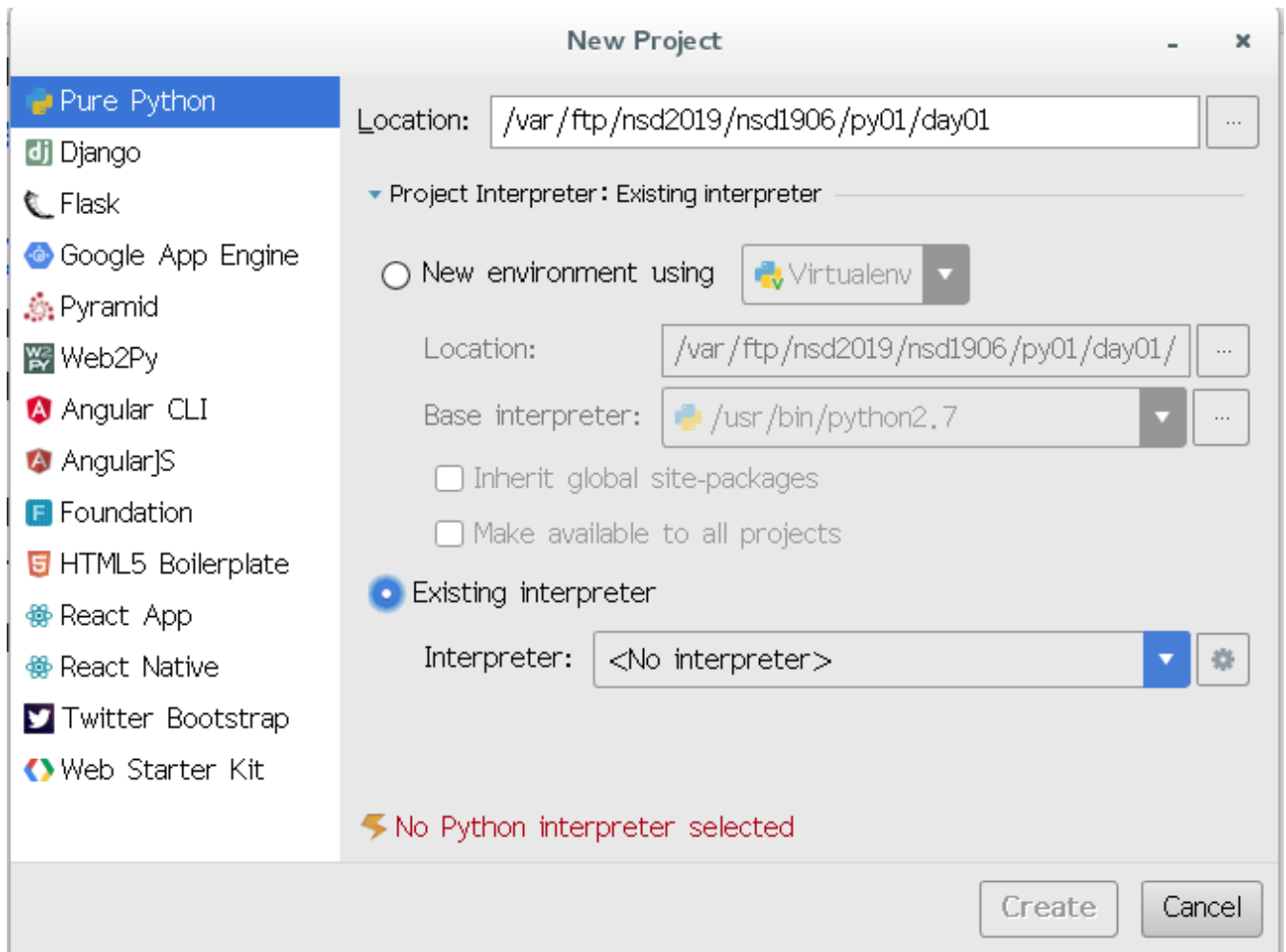
Skip Remaining and Set Defaults

Next: Launcher Script

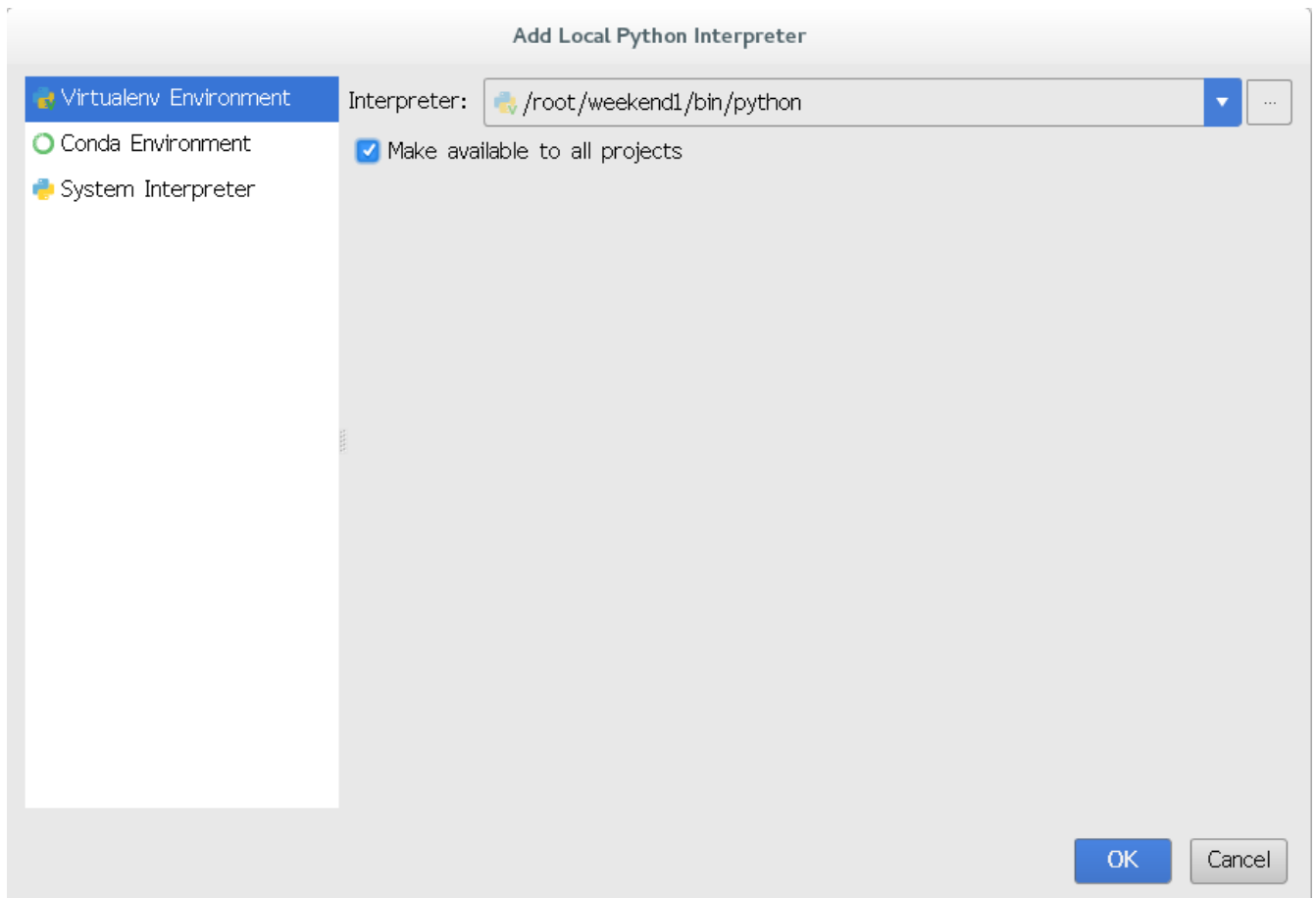
创建一个新项目



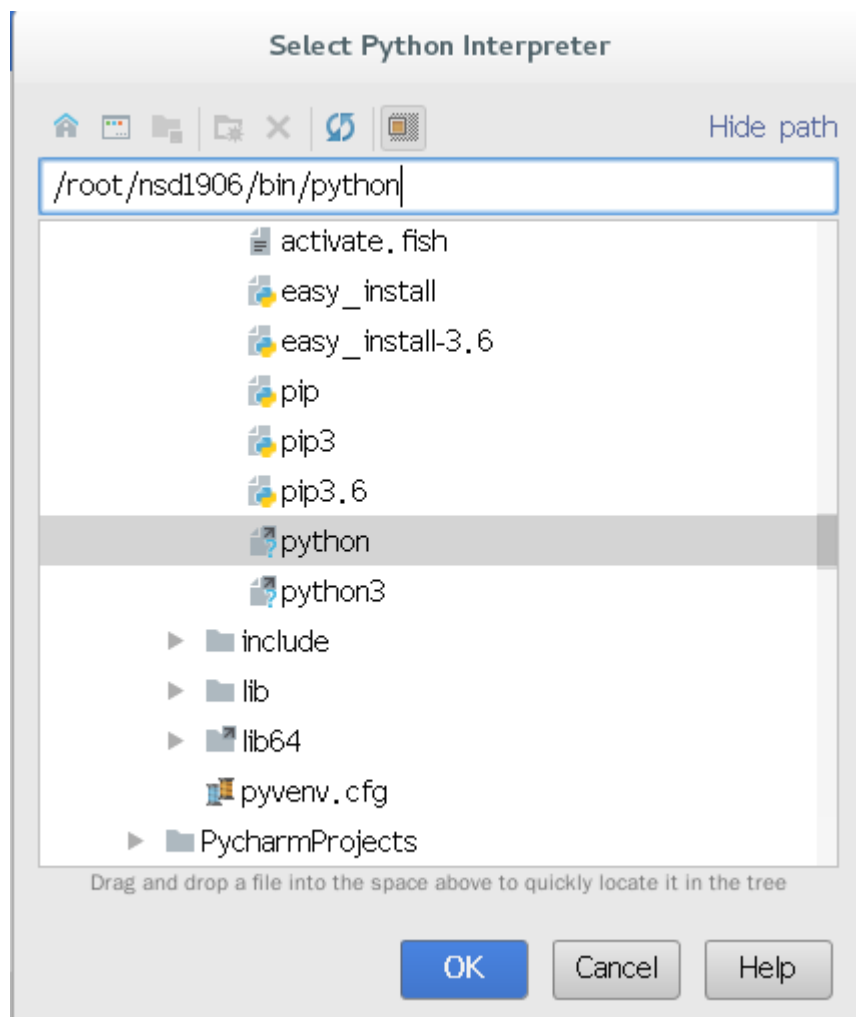
在创建项目的页面中，上面Location，填写的是你项目的目录。在Project Interpreter，选择Existing interpreter，再点击右下角的齿轮图标，选add local



在弹出的页面中，勾选Make available to all projects，点击右边的三个点按钮

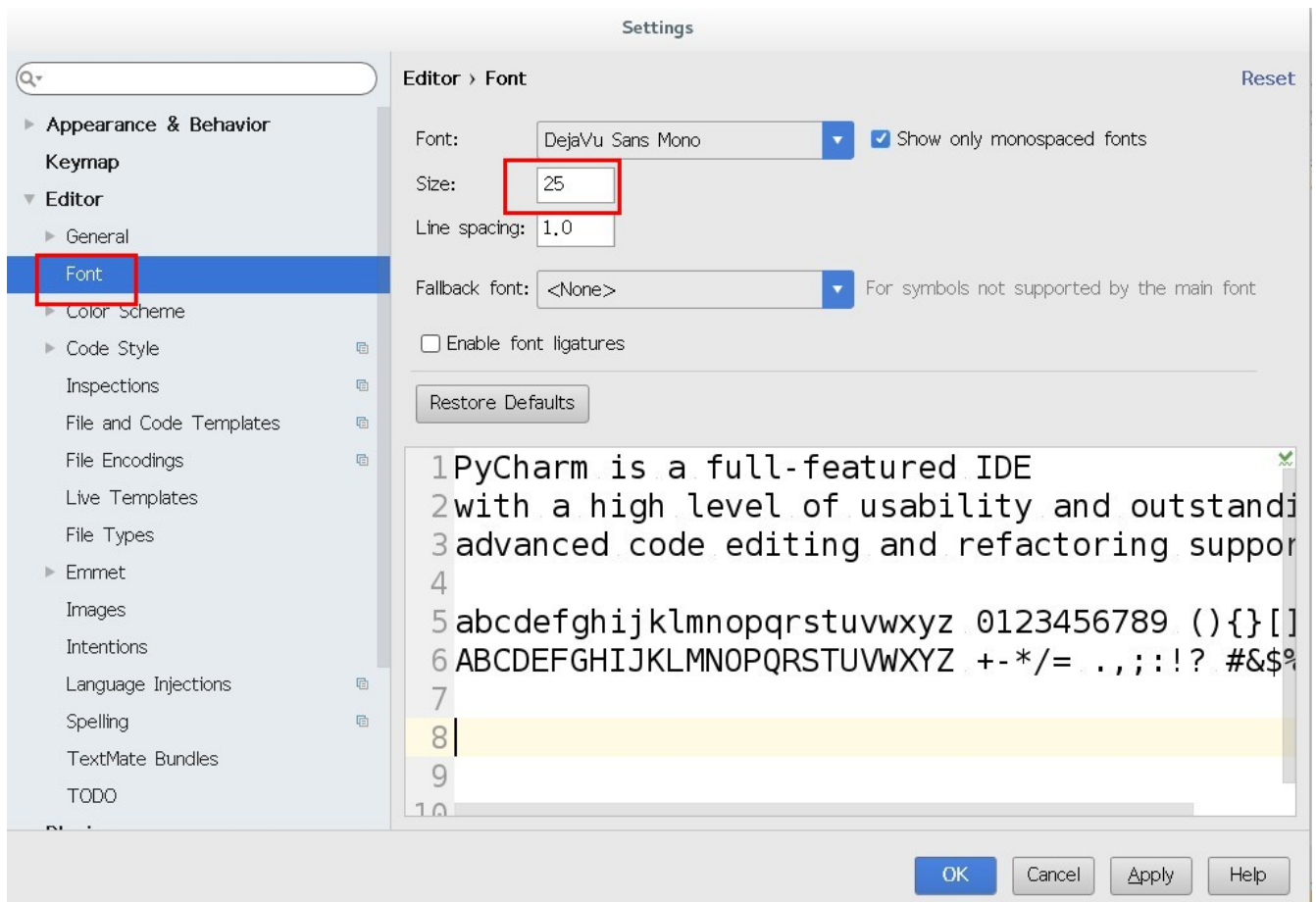


在弹出的对话框中，找到前面创建的python虚拟环境



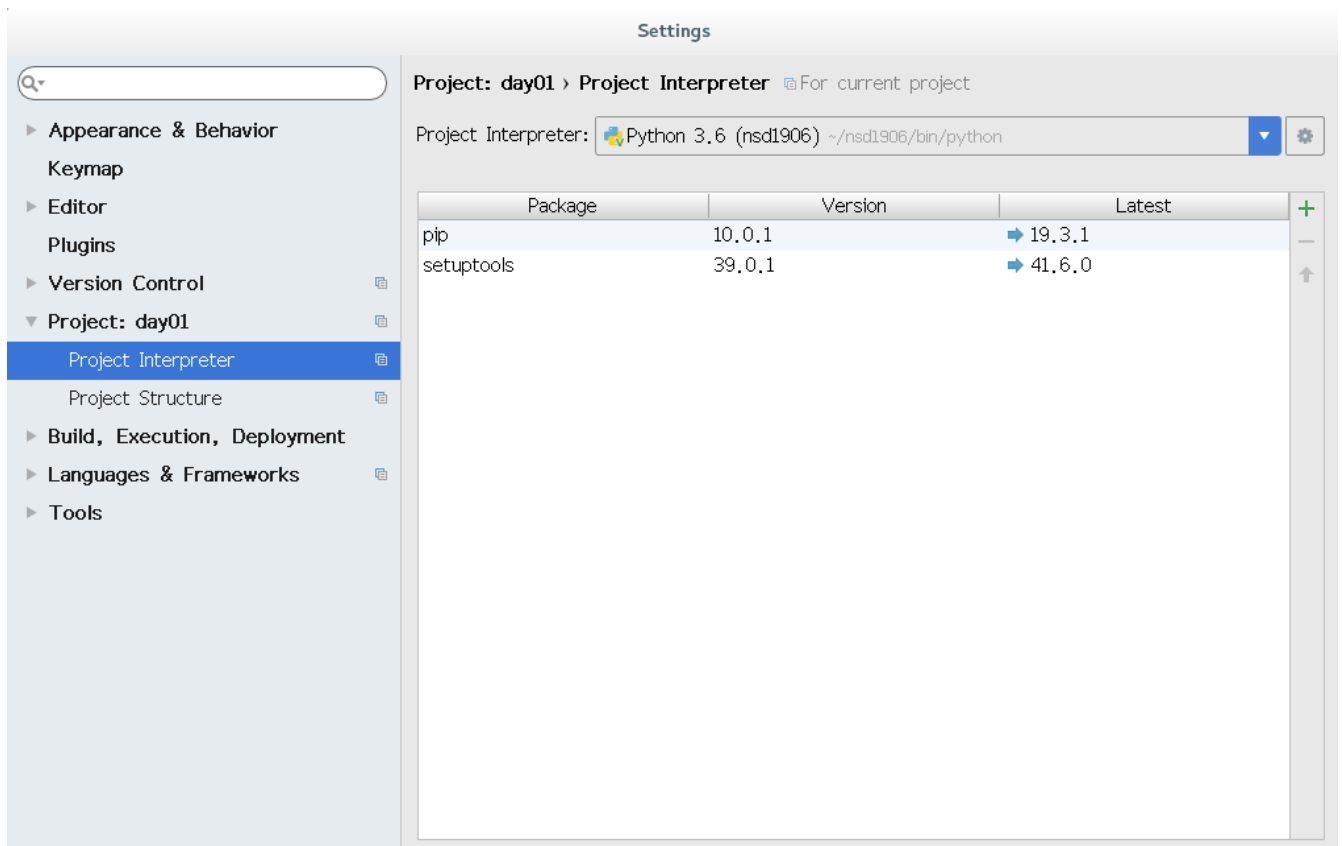
将pycharm的文字调大

File -> Settings



调整解释器

File -> Settings



python的语法结构

- python完全靠缩进表达代码逻辑，缩进4个空格
- 注释采用#，续行采用\，与shell一致
- 同一行如果书写多条语句，可以用;分隔，与shell一致。但是，可读性差，不推荐

输入输出

- print输出语句

```
>>> print('hello world!') # 字符串必须用引号引起来，单双引号无区别
hello world!
>>> print('hao', 123) # 打印多项，用逗号分隔，输出时各项间默认是空格
hao 123
>>> print('hao', 123, sep='***') # 设置输出的各项间以***作为分隔符
hao***123
>>> print('hao' + 123) # 字符串和数字不能拼接，报错
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> print('hao' + '123') # 字符串拼接用+
hao123
```

- input输入语句

```
>>> user = input('username: ')
username: tom
>>> print(user)
tom
>>> user
'tom'
>>> n = input('number: ') # input读入的一定是字符串
number: 10
>>> print(n)
10
>>> n + 5 # 字符串10不能和数字5做运算
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> int(n) + 5 # int将字符串形式的数字转成真正的数字
15
>>> n + str(5) # str将对象转成字符串
'105'
```

变量

- 可以变化的量是变量，如a = 10，以后还可以改变它，a = 100
- 与变量相反的是字面量，如字符串'hello'，数字100，都是字面量

- 写程序时，如果总是用字面量，就是硬编码。
- 合法变量名的要求：
 - 首字符必须是字母或下划线
 - 其他字符可以是数字、字母、下划线
 - 区分大小写
- 推荐的名称的写法
 - 变量名全部用小写，尽量有意义，pythonstring
 - 简短，pystr
 - 多个单词间用下划线分隔，py_str
 - 变量用名词，如phone；函数名用谓词(动词+名词)，update_phone
 - 类名采用驼峰的形式，MyClass
- 变量赋值自右向左进行，将 = 右边表达式的计算结果赋值给左边的变量
- 变量在使用之前必须赋值

```
>>> a = 10
>>> a = a + 5
>>> a
15
>>> a += 5
>>> a
20
>>> a -= 10
>>> a
10
>>> a *= 2
>>> a
20
>>> b + 5    # 变量b没有赋初值，将会出现名称错误
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
```

python之禅

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.      # 美胜丑
Explicit is better than implicit.  # 明胜暗
Simple is better than complex.     # 简胜繁
```

运算符

- 算术运算符

```
>>> 5 / 3
1.6666666666666667
>>> 5 // 3    # 只保留商
```

```

1
>>> 5 % 3    # 求余，模运算
2
>>> divmod(5, 3)    # 5除以3，返回商和余数
(1, 2)
>>> a, b = divmod(5, 3)    # 将商和余数分别赋值给a和b
>>> a
1
>>> b
2
>>> 2 ** 3    # 2的三次方，幂运算
8
>>> 3 ** 4
81

```

- 比较运算符，得到结果是True或False

```

>>> 3 = 3    # 这是赋值，不是比较
File "<stdin>", line 1
SyntaxError: can't assign to literal
>>> 3 == 3    # 比较必须使用==表示判断是否相等
True
>>> 10 > 5 > 1    # python支持连续比较
True
>>> 20 > 10 < 30    # 相当于是20 > 10 and 10 < 30
True

```

- 逻辑运算符，最终结果为True或False

```

>>> 10 > 50 and 2 < 5
False
>>> 10 > 5 and 2 < 5    # and 两边全为True，最终才是True
True
>>> 10 > 50 or 2 < 5    # or两边只要一边为True就是True
True
>>> 2 * 3 ** 2
18
>>> 2 * (3 ** 2)
18
>>> not 10 > 50 or 2 < 5    # 涉及到可读性差的代码，应该加()
True
>>> (not 10 > 50) or 2 < 5
True
>>> 10 > 3
True
>>> not 10 > 3    # not是取反，真变假，假变真
False

```

数据类型概述

- 数字

```
# 有小数点为浮点数
>>> type(1.3)
<class 'float'>
# 没有小数点为整数
>>> type(10)
<class 'int'>
# True和False值分别是1和0
>>> True + 1
2
>>> False * 5
0
# python默认使用10进制数表示数字
# 如果以0o或0O开头表示8进制数
>>> 0o11
9
>>> oct(10)    # 将10进制数转为8进制数
'0o12'
# 16进制数以0x开头
>>> 0x11
17
>>> hex(20)    # 10进制转16进制
'0x14'
# 2进制数以0b开头
>>> 0b11
3
>>> bin(7)
'0b111'
```

- 字符串：引号引起来的部分。单双引号没有任何区别

```
# 三引号就是三个连续的单引号或双引号
>>> words = "hello    # 希望每个单词占一行，但是不能直接回车，写到下一行
File "<stdin>", line 1
    words = "hello
            ^
SyntaxError: EOL while scanning string literal
>>> words = 'hello
File "<stdin>", line 1
    words = 'hello
            ^
SyntaxError: EOL while scanning string literal
>>> words = "hello\nwelcome\ngreet"
>>> print(words)    # print将\n转义成回车
hello
welcome
greet
# 三引号可以保存用户的输入格式
>>> wds = """hello
... welcome
... greet"""
```

```

>>> print(wds)
hello
welcome
greet
>>> wds
'hello\nwelcome\ngreet'

>>> py_str = 'python'
>>> len(py_str)    # 计算长度
6
>>> py_str[0]      # 取出下标为0的字符
'p'
>>> py_str[6]      # 下标超出范围将会报错
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> py_str[-1]     # 下标为负数，表示自右向左
'n'
>>> py_str[-2]
'o'
>>> py_str[2:4]    # 取切片，起始下标包含，结束下标不包含
'th'
>>> py_str[2:6]    # 切片，不会出现下标越界的错误
'thon'
>>> py_str[2:60]
'thon'
>>> py_str[2:]     # 结束不写，表示取到结尾
'thon'
>>> py_str[:2]     # 开头不写，表示从开头取
'py'
>>> py_str[:]      # 从开头取到结尾
'python'
>>> py_str[:2]     # 切片默认步长为1，改为2
'pto'
>>> py_str[1::2]
'yhn'
>>> py_str[::-1]   # 步长值为负，表示从右向左取
'nohtyp'
>>> 'abc' + '123'  # 字符串拼接
'abc123'
>>> py_str + ' good'
'python good'
>>> '*' * 30       # *号重复30遍
'*****'
>>> '#' * 30
'#####'
>>> py_str * 5
'pythonpythonpythonpythonpython'
>>> 't' in py_str   # t在字符串中吗？
True
>>> 'th' in py_str  # th在字符串中吗？
True
>>> 'to' in py_str  # to在字符串中吗？不连续的字符为False

```

```
False
>>> 'to' not in py_str
True
```

- 列表，与字符串类似，都是序列对象

```
>>> alist = [1, 2, 3, 'tom', 'jerry']
>>> len(alist)
5
>>> alist[0]
1
>>> alist[3:]
['tom', 'jerry']
>>> 3 in alist
True
>>> 'o' in alist
False
>>> alist + [10, 20]
[1, 2, 3, 'tom', 'jerry', 10, 20]
>>> alist * 2
[1, 2, 3, 'tom', 'jerry', 1, 2, 3, 'tom', 'jerry']
>>> alist.append('bob') # 向列表尾部追加一个字符串
>>> alist
[1, 2, 3, 'tom', 'jerry', 'bob']
```

- 元组：可以认为它是不可变的列表

```
>>> atop = (1, 2, 3, 'bob', 'tom')
>>> len(atop)
5
>>> atop[0]
1
>>> atop[3:]
('bob', 'tom')
>>> alist
[1, 2, 3, 'tom', 'jerry', 'bob']
>>> alist[0] = 10 # 列表可变，可以把元素重新赋值
>>> alist
[10, 2, 3, 'tom', 'jerry', 'bob']
>>> atop[0] = 10 # 元组不可变，所以不能把它的元素重新赋值
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

- 字典：采用的是key:val的形式

```
>>> adict = {'name': 'tom', 'age': 20}
>>> len(adict)
2
>>> adict[0]    # 字典是无序的，所以没有下标
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
>>> 'tom' in adict    # tom是字典的key吗？
False
>>> 'name' in adict
True
>>> adict['name']    # 通过key来找到val
'tom'
```

数据类型的特点

- 按存储模型分类
 - 标量：数据中不能包含其他类型数据。数字、字符串
 - 容器：可以包含其他类型数据。列表、元组、字典
- 按更新模型分类
 - 不可变：数字、字符串、元组
 - 可变：列表、字典
- 按访问模型分类
 - 直接：数字
 - 顺序：字符串、列表、元组
 - 映射：字典