nsd1903_python1_day01

http://www.jianshu.com -> python百例

https://www.jianshu.com/p/29e7847f7298 -> vim变成IDE

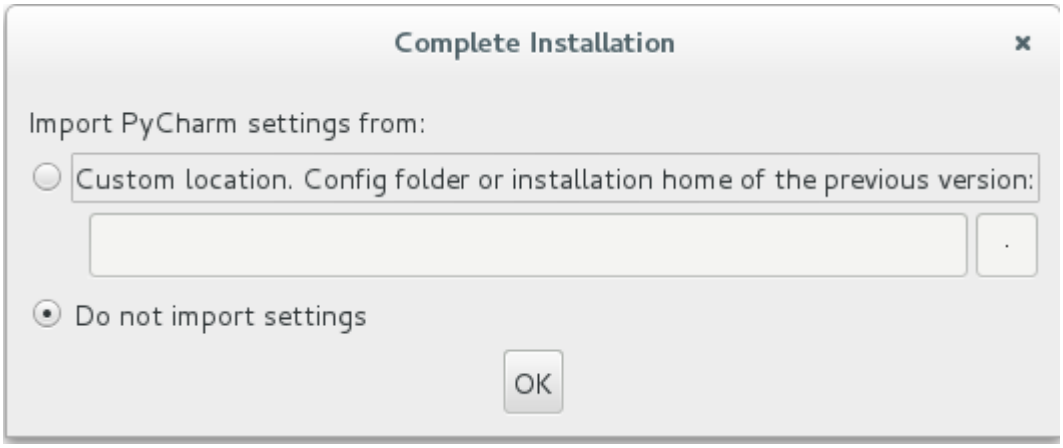
环境准备

官方站点：http://www.python.org

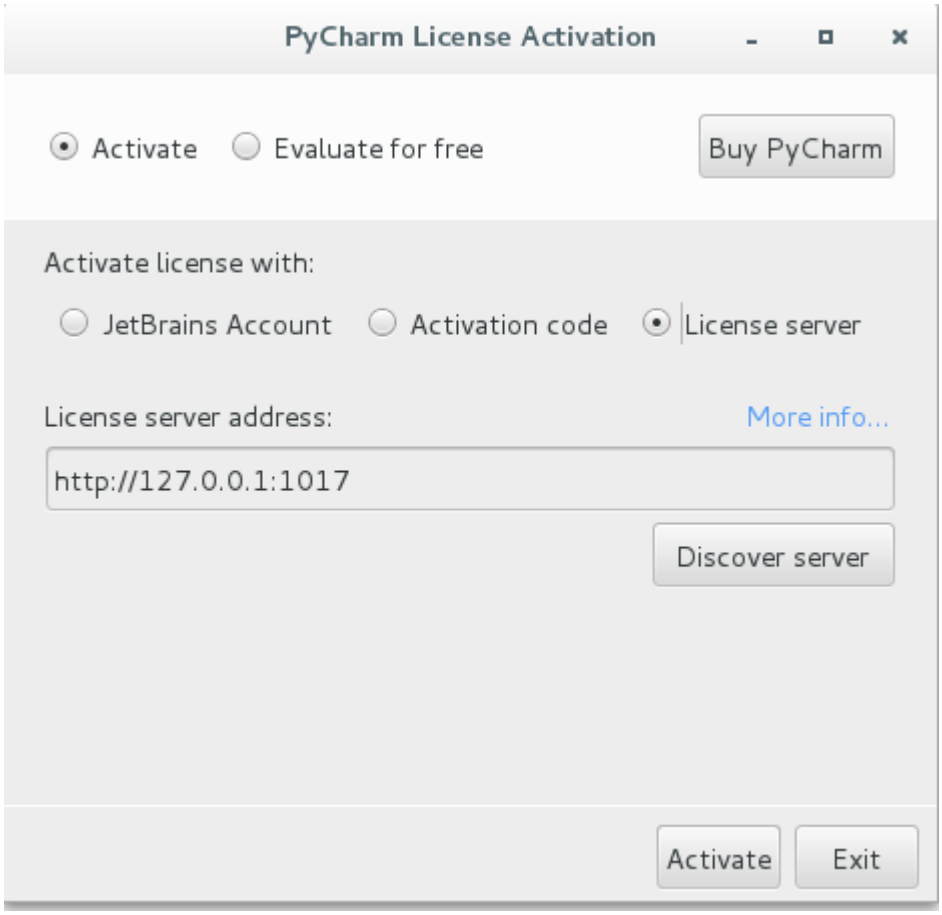
IDE：集成开发环境

pycharm的配置

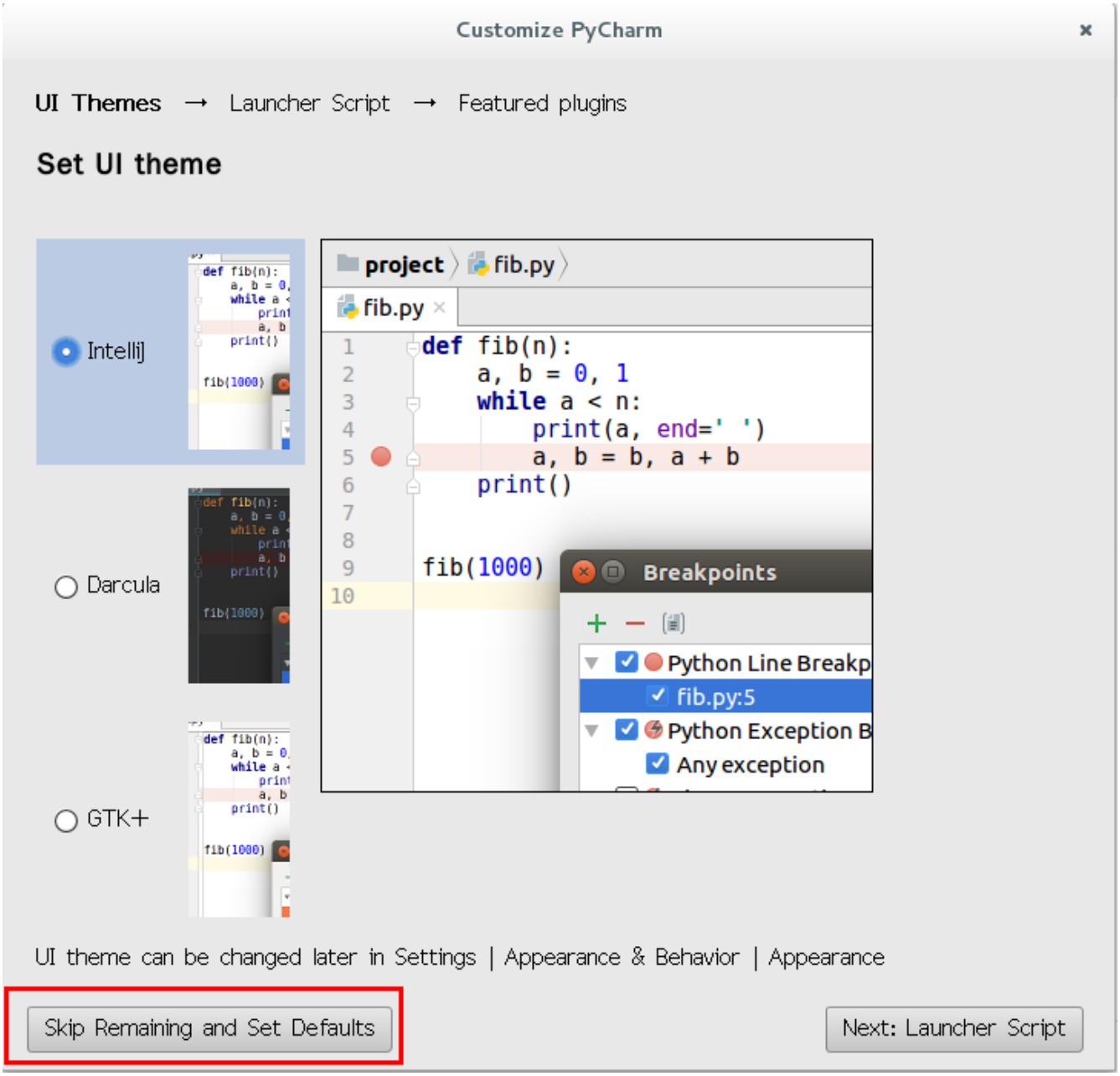
1. 如果是第一次使用，打开后，出现以下内容，选择下面的“不导入配置”



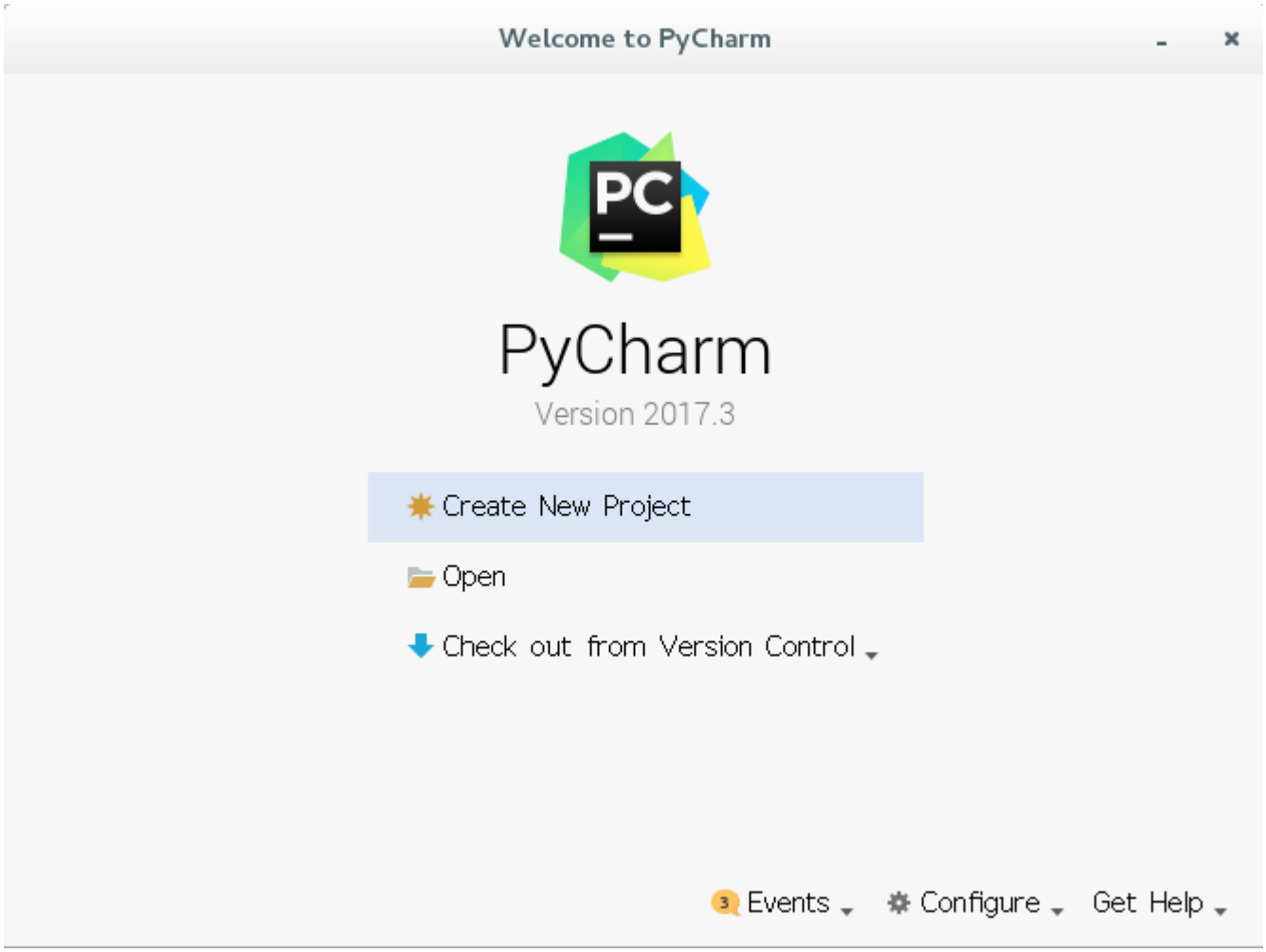
2. 如果是专业版，需要购买


PyCharm License Activation
Activate
Evaluate for free
Buy PyCharm
Activate license with:
JetBrains Account
Activation code
License server
License server address:
More info...
http://127.0.0.1:1017
Discover server
Activate
Exit

3. 选择界面风格



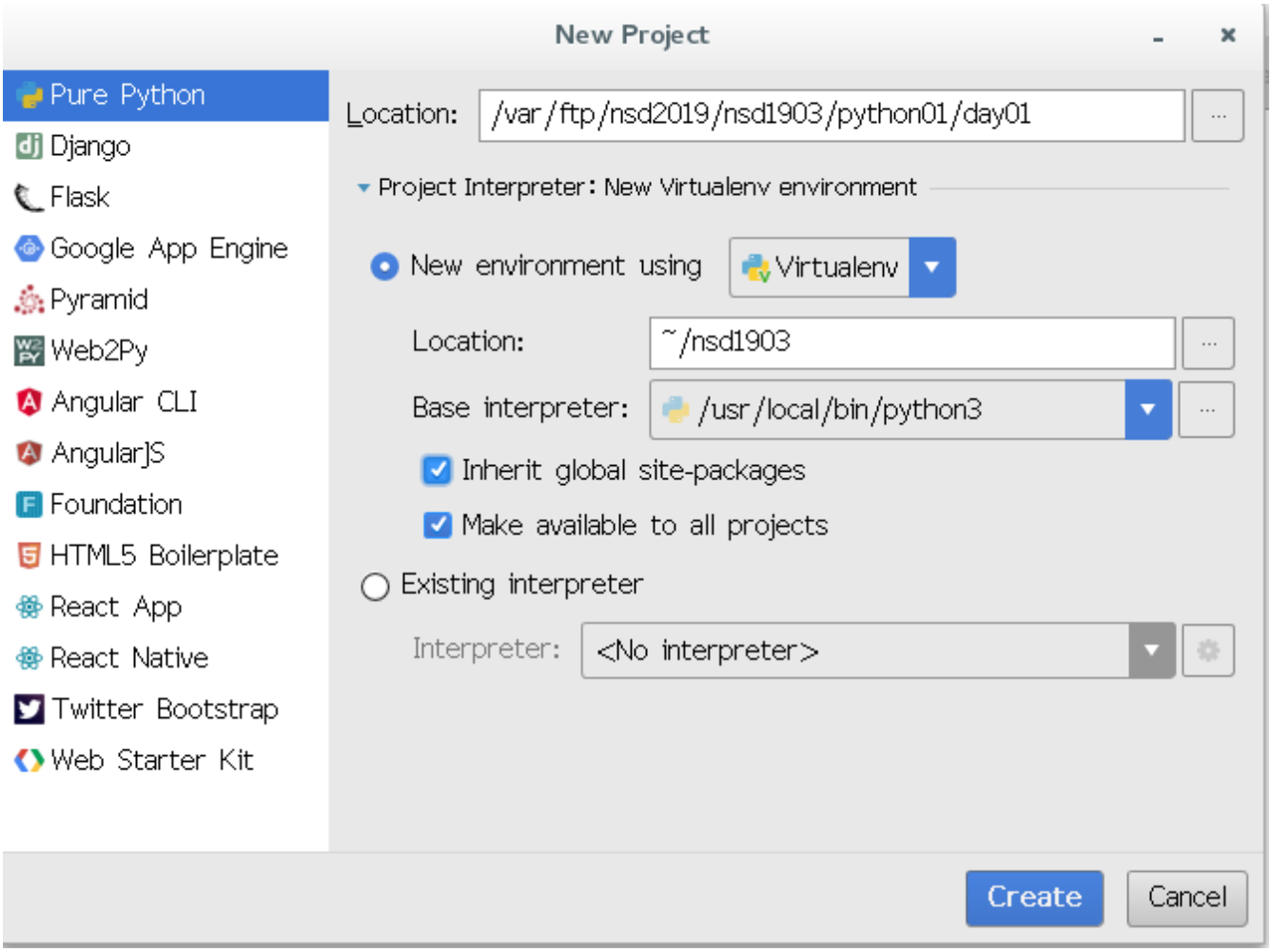
4. 创建新项目，一个软件工程就是一个项目，对应一个文件夹



5. 创建虚拟环境

Location: 项目目录，即代码文件存放的路径

Project Interpreter：解释器采用虚拟环境，相当于是把系统的python程序拷贝到了一个文件夹中，以后安装python软件都安装到这个文件夹。将来项目完成了，不需要这个环境了，只要把虚拟环境目录删除即可。

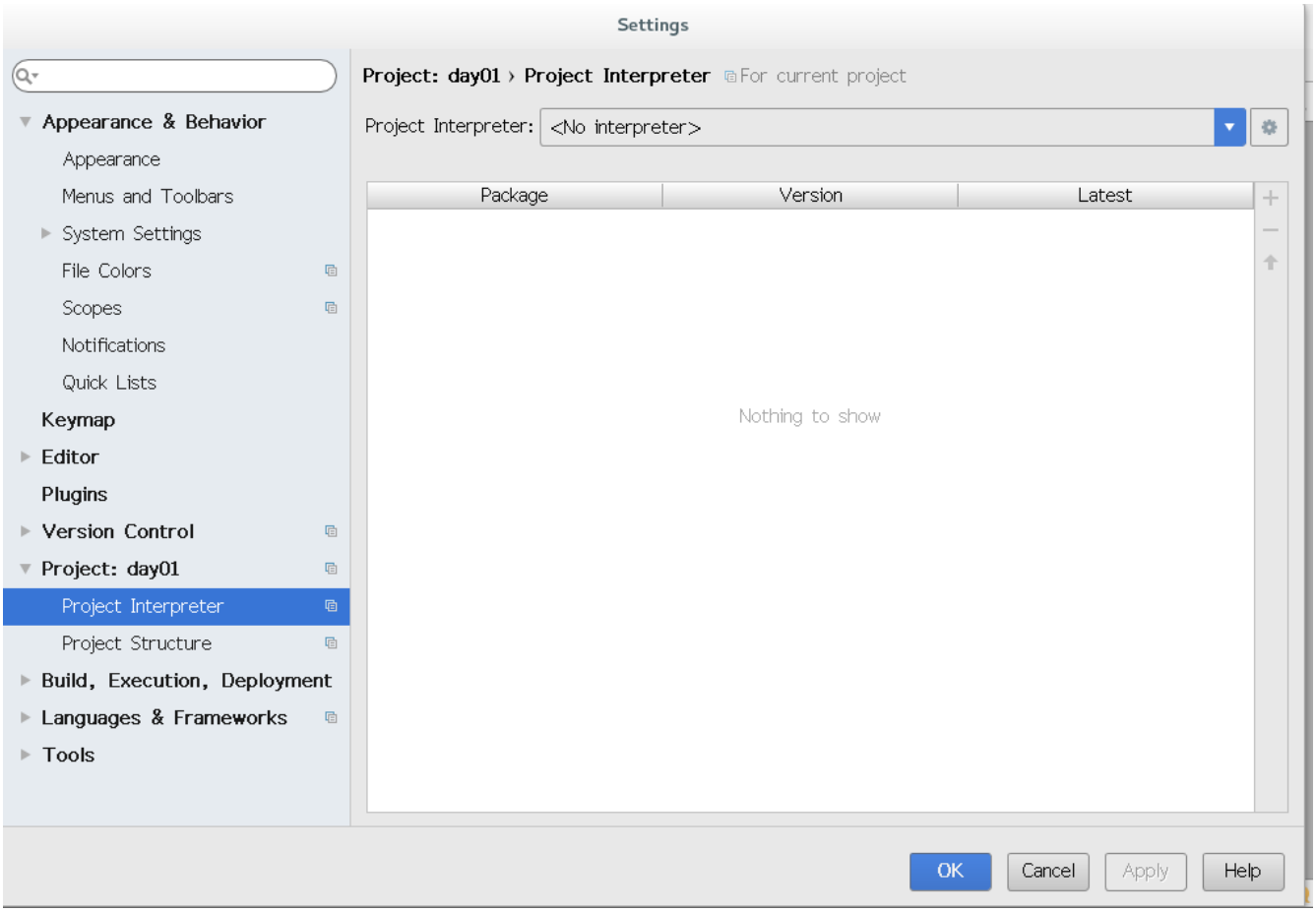


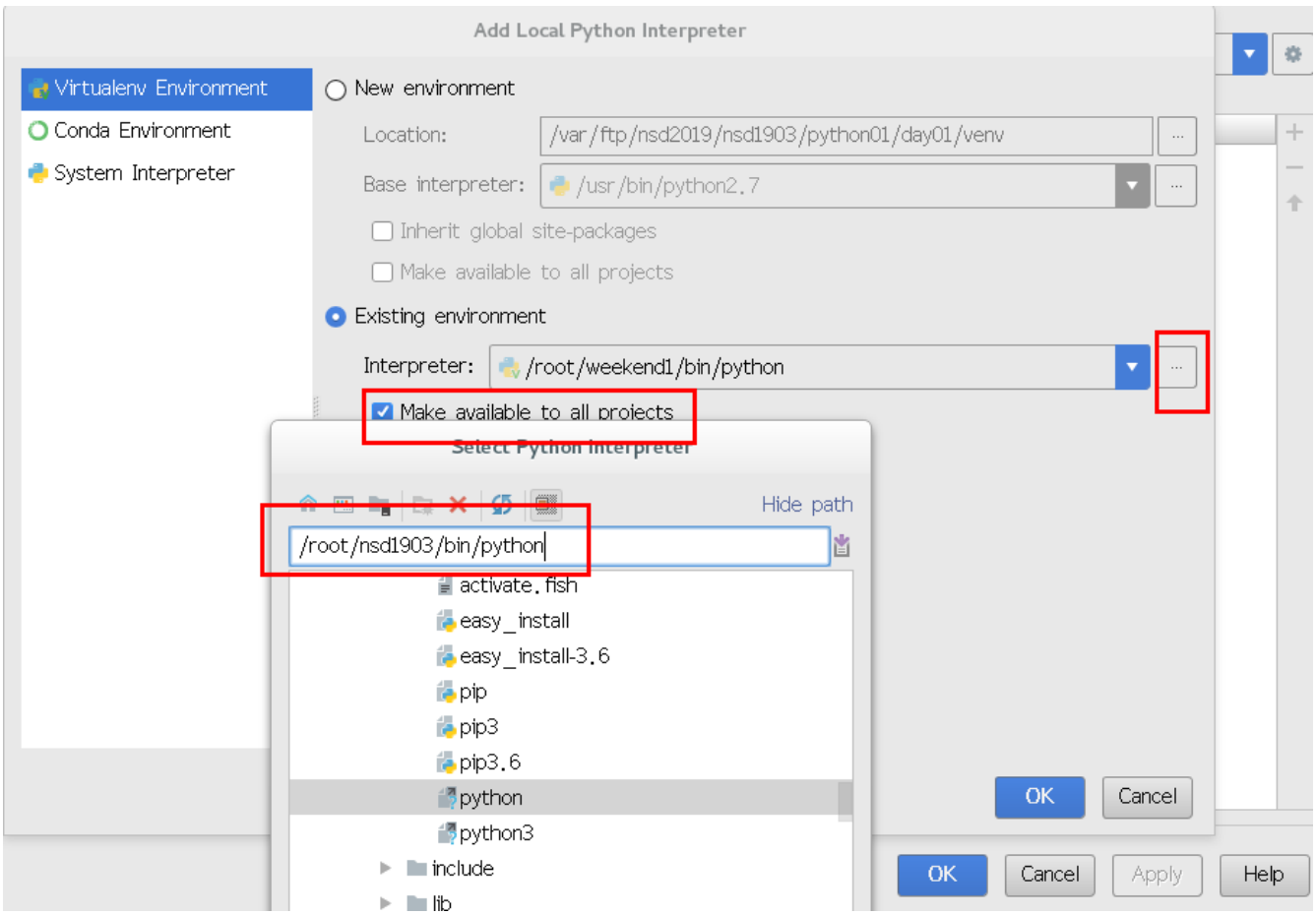
6. 如果上一步，自动创建虚拟环境出现故障，可以手工创建虚拟环境

```
# 创建虚拟环境
[root@room8pc16 nsd1903]# python3 -m venv ~/nsd1903
# 使用虚环境时，需要激活它
[root@room8pc16 nsd1903]# source ~/nsd1903/bin/activate
(nsd1903) [root@room8pc16 nsd1903]# python --version
Python 3.6.7
```

7. 修改pycharm项目的虚拟环境

File -> Settings -> Project: Day01 -> Project Interpreter -> 点右上角的齿轮 -> Add Local -> Existing Enviroment -> 点击... -> 输入/root/nsd1903/bin/python


Settings
Project: day01 › Project Interpreter For current project
Project Interpreter: <No interpreter>
Package
Version
Latest
Nothing to show
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
File Colors
Scopes
Notifications
Quick Lists
Keymap
Editor
Plugins
Version Control
Project: day01
Project Interpreter
Project Structure
Build, Execution, Deployment
Languages & Frameworks
Tools
OK
Cancel
Apply
Help


Add Local Python Interpreter
Virtualenv Environment
Conda Environment
System Interpreter
New environment
Location: /var/ftp/nsd2019/nsd1903/python01/day01/venv
Base interpreter: /usr/bin/python2.7
Inherit global site-packages
Make available to all projects
Existing environment
Interpreter: /root/weekend1/bin/python
Make available to all projects
Select Python Interpreter
Hide path
/root/nsd1903/bin/python
activate.fish
easy_install
easy_install-3.6
pip
pip3
pip3.6
python
python3
include
lib
OK
Cancel
OK
Cancel
Apply
Help

8. 修改编辑文字大小

File -> Settings -> Editor -> font -> size 修改大小

python运行方式

- 交互解释器

```
[root@room8pc16 devops0101]# source ~/nsd1903/bin/activate
(nsd1903) [root@room8pc16 devops0101]# python
>>> print("hello world!")
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
```

- 文件形式 在pycharm项目名上右击选第四项，可以拷贝到项目的绝对路径

```
# vim hi.py
print("Hello World!")
(nsd1903) [root@room8pc16 day01]# python hi.py
Hello World!
```

python语法

- python使用缩进表达代码逻辑，推荐缩进4个空格
- 有子语句的代码，后面都有冒号
- 注释使用#号，在pycharm中可以按ctrl + /进行注释或取消注释
- 多个语句在同一行，需要使用分号分隔，但是仍然不推荐。

输入输出语句

```
# 字符串必须写在引号中，单双引号没有区别
>>> print('hello world!')
# 一个print语句中，可以打印多项，123没有引号，表示数字。默认各项间用空格分隔
>>> print('hao', 123, 'world')
hao 123 world
# 输出时，也可以指定各项之间的分隔符
>>> print('hao', 123, 'world', sep='_')
hao_123_world
>>> print('hao', 123, 'world', sep='***')
hao***123***world
# 字符串可以使用+拼接
>>> print('hello' + 'world')
helloworld

# 通过input获取键盘输入，input括号中的字符串是屏幕提示符，把用户输入的结果保存在变量num中，num是变量，使用时不用像shell那样加$前缀。
>>> num = input("number: ")
number: 100
>>> num
'100'
# 只要通过input读取，都是字符类型，字符不能和数字进行四则运算
```

```
>>> num + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int

# 可以通过int函数把num转换成整数，再和数字进行运算
>>> int(num) + 10
110
```

变量

会变化的量。常量不会变，表示字面本身含义的量不会变。100，'abc'这样的量是字面量，表示看到的字面本身含义。

为什么要用变量：方便，变量可以用有意义的名字。

变量的命名约定：

- 首字符只能是字母或下划线
- 其他字符可以是字母或下划线或数字
- 区分大小写

```
>>> 2a = 10    # Error
>>> _a = 5     # OK, 不常用
>>> abc-1 = 10   # Error
# 以下的两个变量是不一样的
>>> a = 10
>>> A = 100
```

推荐的命名方法：

- 变量和函数名都用小写字母，如pythonstring
- 简短，如pystr
- 有意义
- 多个单词间用下划线分隔，如py_str
- 变量用名词，函数用谓词(动词＋名词)，如phone表示变量，用update_phone表示函数
- 类名采用驼峰形式，如MyClass

使用变量

```
# 变量使用之前必须先赋值
>>> print(a)    # a没有定义，所以报名字错误
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined

>>> counter = 0
# 赋值运算自右向左运算。以下代码含义是将counter的值取出并加1，然后再赋值给变量counter
>>> counter = counter + 1

# 上面代码可以简写为
>>> counter += 1
```

```
# python之禅
>>> import this
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
……
美胜丑，明胜暗，简胜繁
```

运算符

算术运算：数学四则运算

```
>>> 5 / 3
1.6666666666666667
>>> 5 // 3   # 只保留商
1
>>> 5 % 3    # 只保留余数
2
>>> divmod(5, 3)    # divmod函数可以同时返回商和余数
(1, 2)
>>> a, b = divmod(5, 3)   # 商和余数分别保存在a和b中
>>> a
1
>>> b
2
>>> 2 ** 3   # 2的3次方
8
```

比较运算

```
# 比较运算的结果是True或False
# 字符串比较时，逐个字符进行比较，一旦出现结果就不再继续了。字母按ASCII值比较。
>>> 'x' > 'abc'
True
# 在python中，支持连续比较
>>> 20 > 15 > 10
True
>>> 20 > 10 < 15   # 相当于20 > 10 and 10 < 15
True
```

逻辑运算符

```
# and两边的表达式都为真，结果才为真
>>> 10 > 5 and 10 < 20
True

# or两边的表达式只要一边为真，结果就是真
>>> 10 > 5 or 10 < 3
True
```

```
# not颠倒真假
>>> 10 > 5
True
>>> not 10 > 5
False
>>> 10 < 5
False
>>> not 10 < 5
True
```

数据类型

数字

- int整数，没小数点
- bool，布尔数，True的值是1，False值为0
- 浮点数，有小数点
- 复数。为了解决谁的平方是-1，数学家发明了复数

整数的表示的方式

```
# python默认以10进制输出，没有任何前缀的数字都认为是10进制数
>>> 23   # 10进制
23
>>> 0o23   # 0o开头表示8进制数
19
>>> 0O23
19
>>> 0x23   # 0x开头表示16进制
35
>>> 0X23
35
>>> 0b11   # 0b开头表示2进制
3
>>> 0B11
3
>>> 0x234
564
>>> 2 * 16 * 16 + 3 * 16 + 4
564

# 应用
>>> import os
>>> os.chmod('login.py', 755)
(nsd1903) [root@room8pc16 day01]# ll login.py
--wxrw--wt. 1 root root 87 7月  31 11:45 login.py
>>> os.chmod('login.py', 0o755)  # linux系统权限是8进制数，不是10进制
(nsd1903) [root@room8pc16 day01]# ll login.py
-rwxr-xr-x. 1 root root 87 7月  31 11:45 login.py
```

字符串

- 字符串是用引号引起来的一组字符，单双引号没有任何区别。

```
# s1和s2分别用了单又引号，它们表示了完全相同的含义
>>> s1 = 'hello world'
>>> s2 = "hello world"

# 将字符串tom赋值给变量name
>>> name = 'tom'
# s3中的name就是字面本身含义name
>>> s3 = "hello name"
>>> s3
'hello name'

# s4中的%s用后面变量name的值替换
>>> s4 = "hello %s" % name
>>> s4
'hello tom'

>>> "%s is %s years old" % ('tom', 22)
'tom is 22 years old'
```

- python允许使用三引号(三个连续的单引号或双引号)保存字符串的样式

```
>>> words = "hello\nwelcome\ngreet"
>>> print(words)   # 打印时，\n转义为回车
hello
welcome
greet
>>> words   # 内部存储时，回车保存为\n
'hello\nwelcome\ngreet'

# 三引号只是在输入时可以保留回车等输入时的样式，但是内部存储没有什么特殊
>>> wordlist = """hello
... welcome
... greet"""
>>> print(wordlist)
hello
welcome
greet
>>> wordlist
'hello\nwelcome\ngreet'
```

- 字符串相关操作

```
>>> py_str = 'python'
>>> len(py_str)   # 计算字符串的长度
6
>>> py_str[0]   # 取出第一个字符，下标为0
'p'
>>> py_str[5]
'n'
>>> py_str[6]    # 如果下标超出范围，将会出现错误
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range

# 下标为负数，表示从右向左取
>>> py_str[-1]
'n'
>>> py_str[-6]
'p'

# 切片，切取片段
>>> py_str[2:4]   # 包含起始下标对应的字符，不包含结束下标对应的字符
'th'
>>> py_str[2:6]   # 取切片时，下标超出范围不会报错
'thon'
>>> py_str[2:6000]
'thon'
>>> py_str[2:]   # 结束下标不写，表示取到结尾
'thon'
>>> py_str[:2]    # 起始下标不写，表示从开头取
'py'
>>> py_str[0:2]
'py'

>>> py_str[:]    # 从开头取到结尾
'python'
>>> py_str[::2]   # 步长值为2
'pto'
>>> py_str[1::2]
'yhn'
>>> py_str[::-1]   # 步长为负表示自右向左取
'nohtyp'

# 成员关系着断
>>> 't' in py_str    # t在字符串中吗？
True
>>> 'th' in py_str
True
>>> 'to' in py_str   # to虽然在字符串中，但不是连续的，返回False
False
>>> 'to' not in py_str   # to不在字符串中吗？
True

# 字符串通过+实现拼接
>>> 'hao' + '123'
'hao123'
>>> str(123)   # 内建函数str可以把数字转成字符串
'123'
>>> 'hao' + str(123)
'hao123'

# 字符串重复
>>> '*' * 30
```

```
'******************************'
>>> '#' * 30
'##############################'
>>> 'ab' * 30
'abababababababababababababababababababababababababababababababab'
```

列表

使用[]表示列表。列表是容器类型的，它里面可以存储各种数据。

```
>>> alist = [10, 20, 30, 'tom', 'jerry']
>>> len(alist)
5
>>> alist[0]
10
>>> alist[-1]
'jerry'
>>> 30 in alist
True
>>> alist[3:]
['tom', 'jerry']

# 列表支持修改内容
>>> alist[2] = 300
>>> alist
[10, 20, 300, 'tom', 'jerry']

# 通过append方法，可以向列表追加数据
>>> alist.append('bob')
>>> alist
[10, 20, 300, 'tom', 'jerry', 'bob']

>>> alist * 2
[10, 20, 300, 'tom', 'jerry', 'bob', 10, 20, 300, 'tom', 'jerry', 'bob']

>>> alist + 100   # 报错，列表不能和数字拼接
>>> alist + [100]   # 列表拼接
```

元组

元组可以认为是不可变的列表，其余完全一样。

```
>>> atuple = (10, 20, 300, 'tom', 'jerry', 'bob')
>>> len(atuple)
6
>>> atuple[0]
10
>>> atuple[2:4]
(300, 'tom')
>>> atuple[0] = 100   # 报错，不能修改
```

```
# 单元素元组，必须有逗号，否则不是元组。
>>> a = (10)
>>> a   # a不是元组，而是数字
10
>>> b = (10,)
>>> b
(10,)
>>> len(b)
1
```

字典

字典是无序的，所以不能像字符串那样取下标和切片。

```
>>> adict = {'name': 'tom', 'age': 22}
>>> len(adict)
2

>>> 'tom' in adict   # 'tom'是字典的key吗？
False

>>> 'name' in adict
True
>>> adict['name']    # 字典通过key取出value
'tom'

# 字典的key不能重复，赋值时，key存在则修改val，key不存在则新增
>>> adict['age'] = 25
>>> adict
{'name': 'tom', 'age': 25}
>>> adict['email'] = 'tom@tedu.cn'
>>> adict
{'name': 'tom', 'age': 25, 'email': 'tom@tedu.cn'}
```

数据类型分类

- 按存储模型
  - 标量：字符串、数字
  - 容器：列表、元组、字典
- 按更新模型
  - 可变：列表、字典
  - 不可变：字符串、元组、数字
- 按访问模型：
  - 直接：数字
  - 顺序：字符串、列表、元组
  - 映射：字典