nsd1907_py01_day01

python虚拟环境

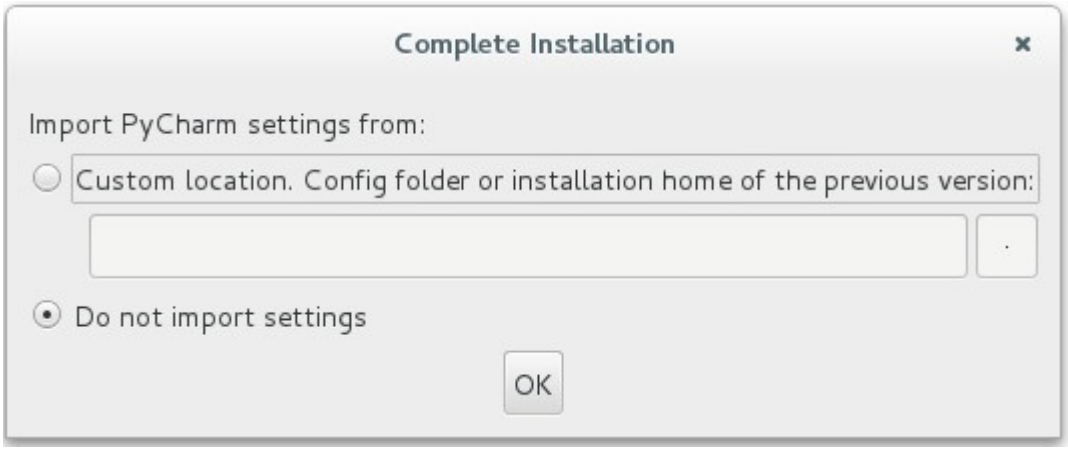
- 类似于虚拟机、容器
- 虚拟环境相当于是一个隔离的目录，安装软件包安装到虚拟环境的目录
- 项目完成后，只要把该目录删掉，就可以得到干净的环境了

```
# 创建虚拟环境
[root@room8pc16 nsd2019]# python3 -m venv ~/nsd1907
[root@room8pc16 nsd2019]# ls ~/nsd1907
bin  include  lib  lib64  pyvenv.cfg
# 激活虚拟环境
[root@room8pc16 nsd2019]# source ~/nsd1907/bin/activate
(nsd1907) [root@room8pc16 nsd2019]# python --version
Python 3.6.7
(nsd1907) [root@room8pc16 nsd2019]# which python
/root/nsd1907/bin/python
```

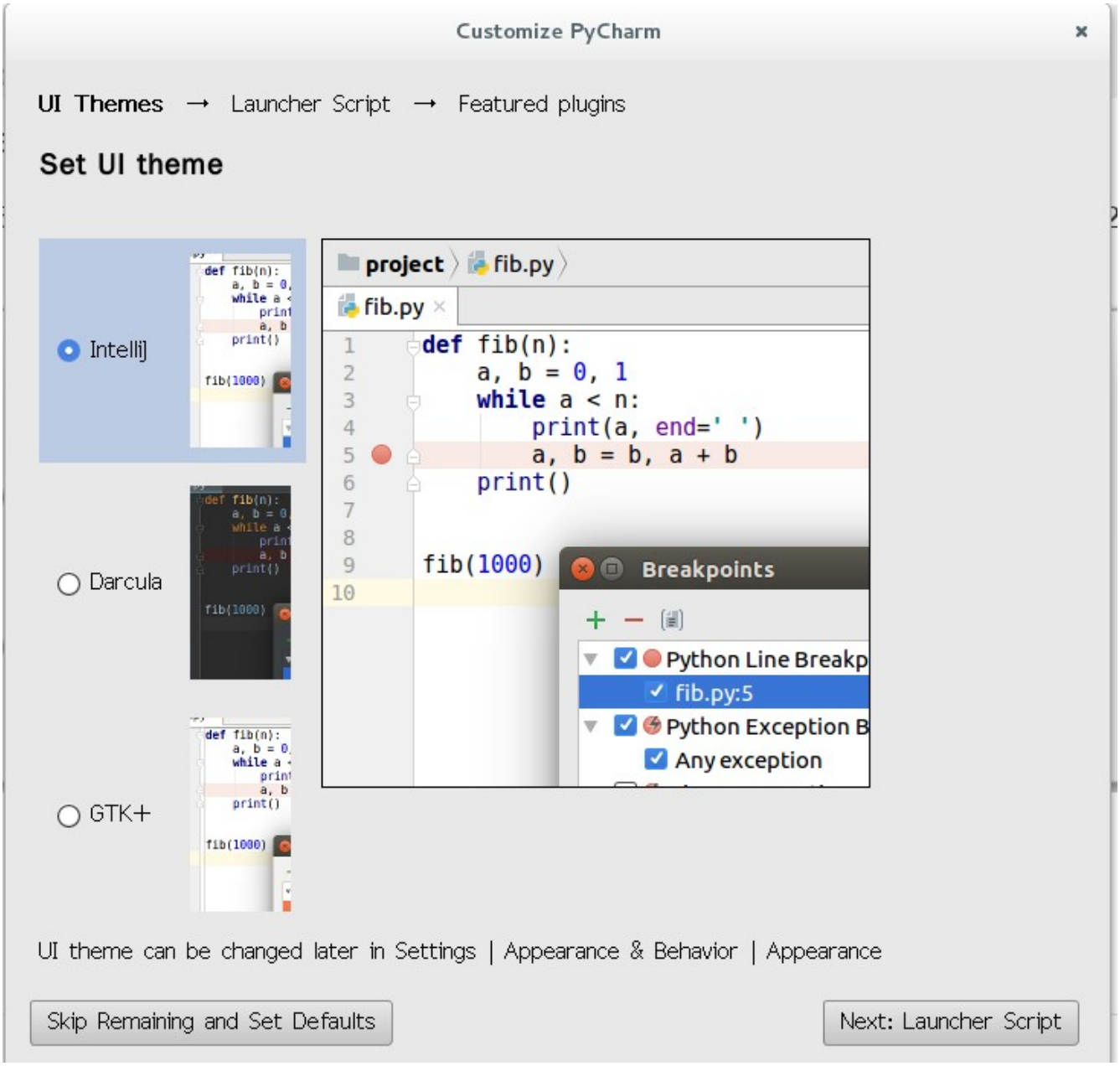
pycharm配置

- pycharm是专门用于python的IDE（集成开发环境）

初始化时，先选择不导入任何配置

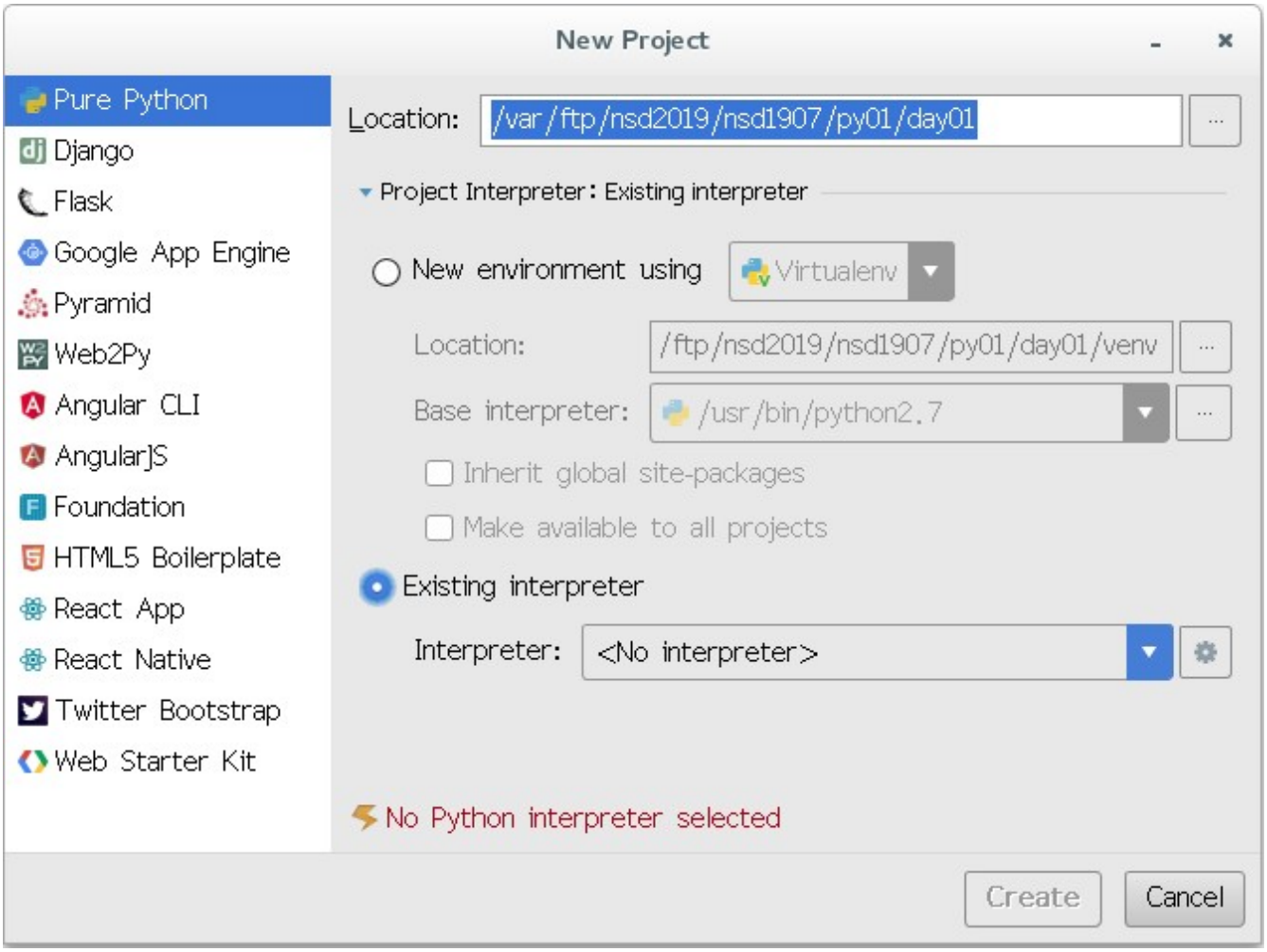


选择一种界面方案，点击Skip...

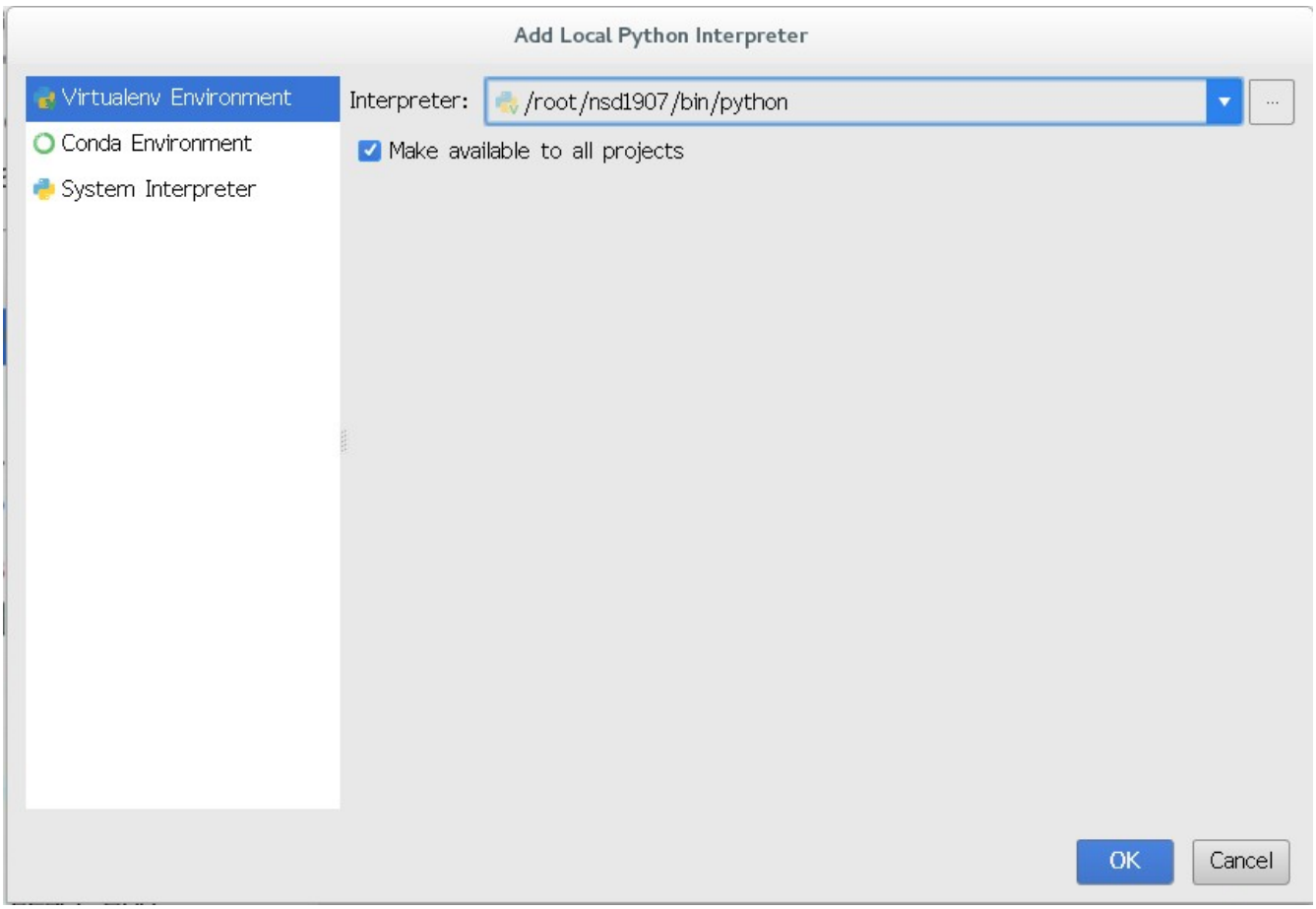

Customize PyCharm
UI Themes → Launcher Script → Featured plugins
Set UI theme
Intellij
Darcula
GTK+
project
fib.py
fib.py
def fib(n):
a, b = 0, 1
while a < n:
print(a, end=' ')
a, b = b, a + b
print()
fib(1000)
Breakpoints
Python Line Breakp
fib.py:5
Python Exception B
Any exception
UI theme can be changed later in Settings | Appearance & Behavior | Appearance
Skip Remaining and Set Defaults
Next: Launcher Script

新建项目Create...

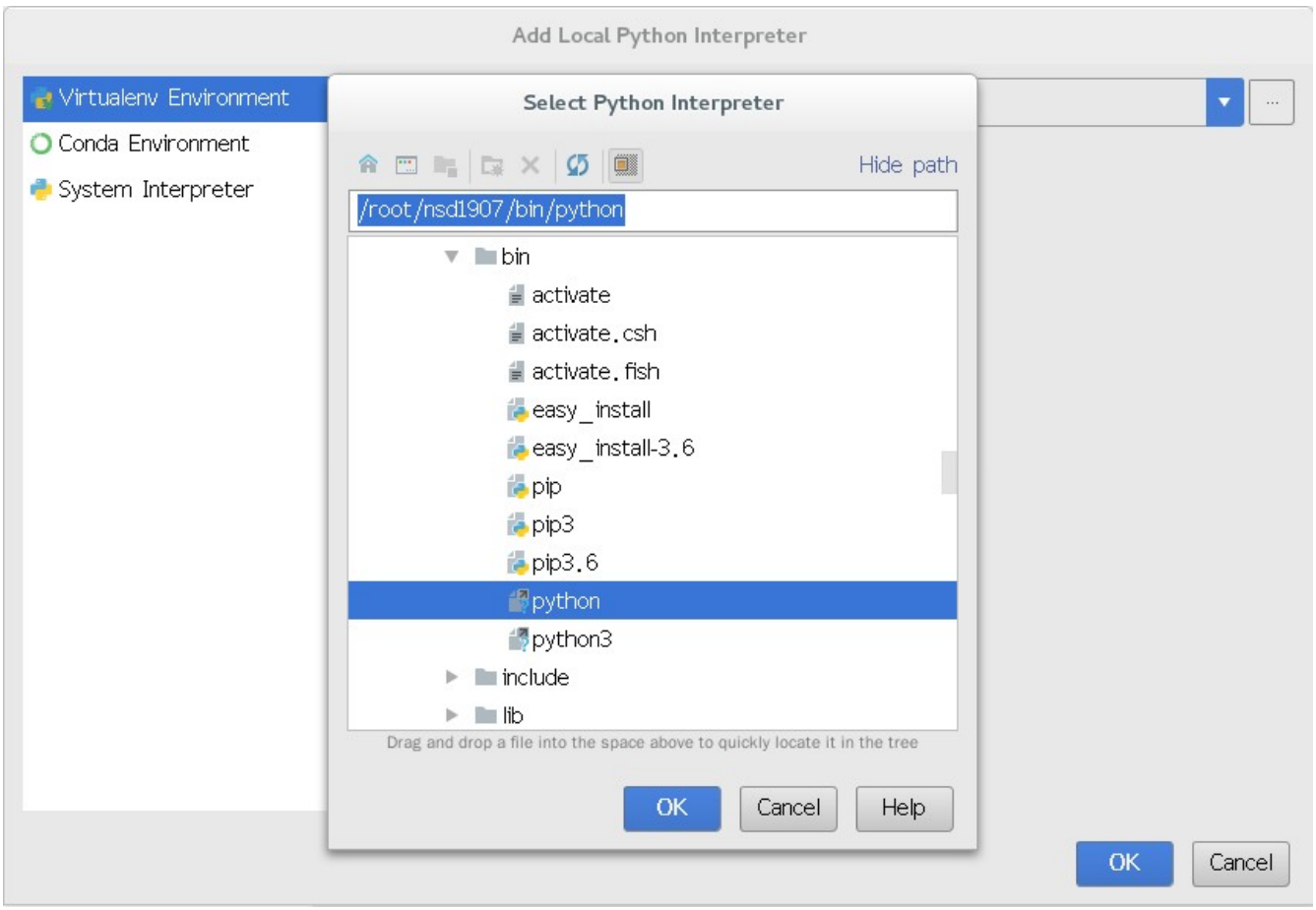
在Location处填写项目目录，点后选existing interpreter。再点击后面的齿轮图标，选add local



在弹出的窗口中，勾选Make available to all projects，然后点击右侧的三个点按钮



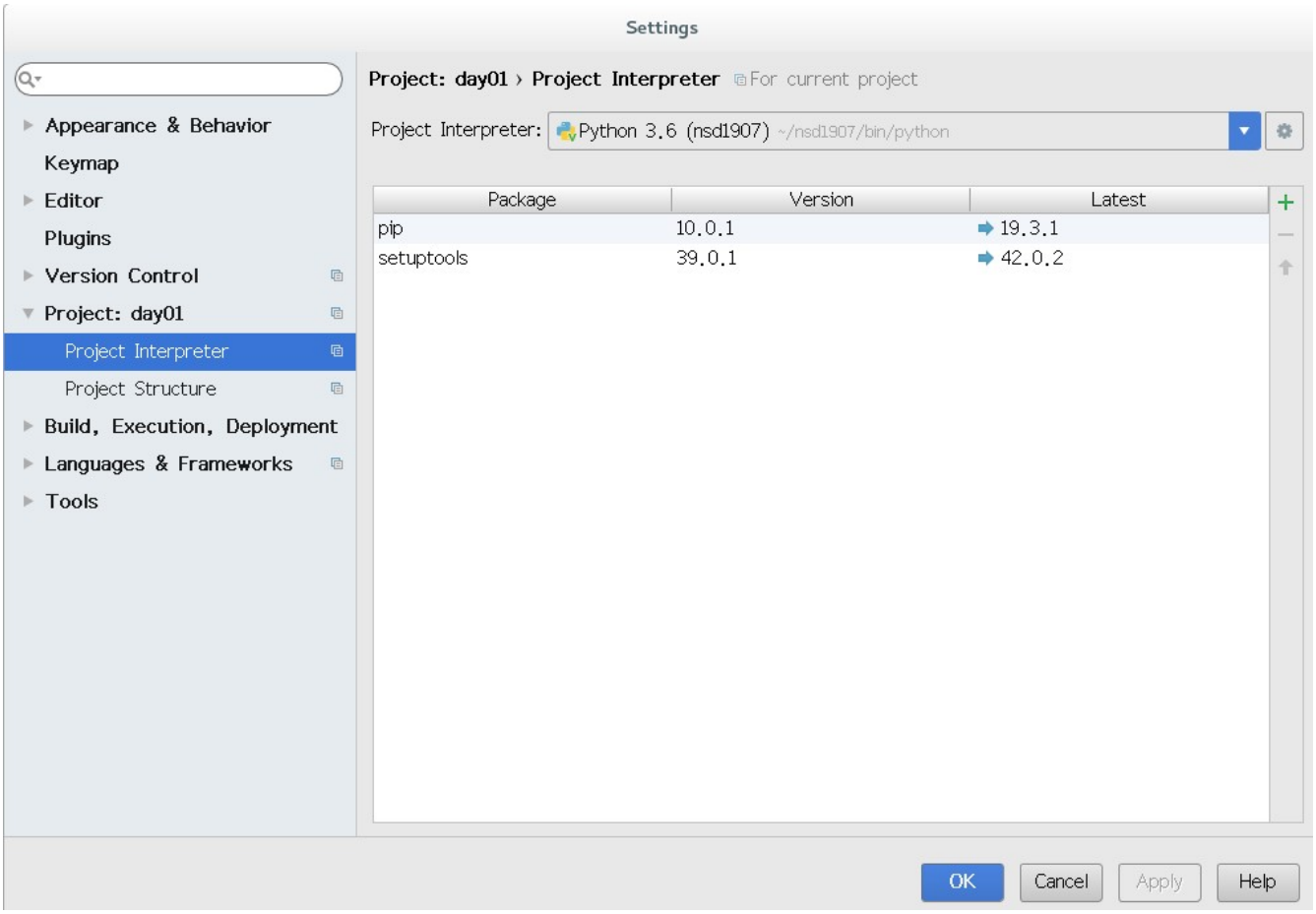
找到自己建立的虚拟环境



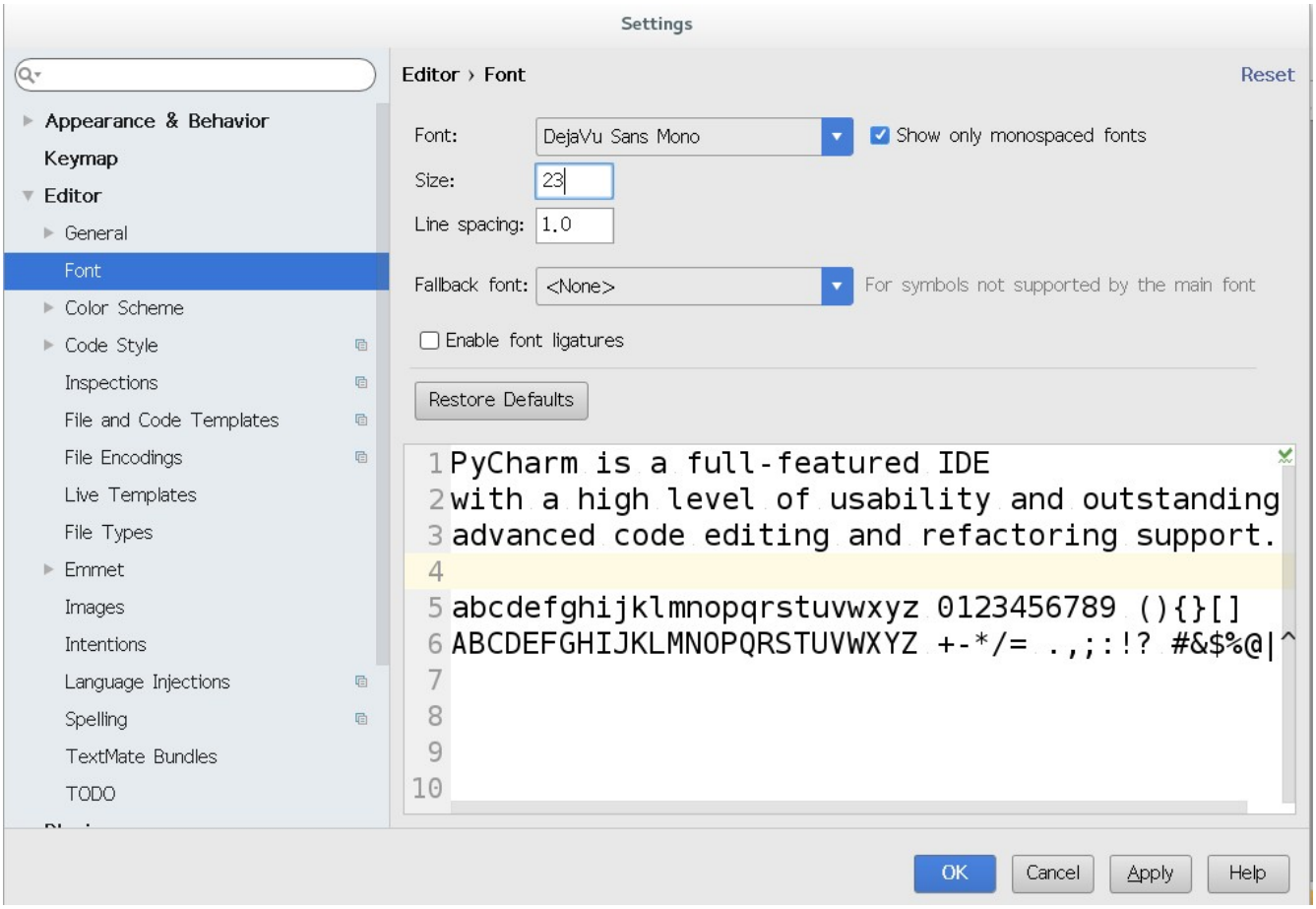
接下来，在各个页面点击OK，完成配置。

如果项目解释器配置错误，可以用以下方式修改：

点击pycharm软件的File -> settings ->Project



修改编辑器文字大小：File -> settings ->Editor -> Font -> Size



变量

- 会变化的量。与之对应的是“字面量”，也就是字面本身的含义
- 变量使用方便
- 变量可以使用有意义的名字

变量命名的约定

- 首字符必须是字母或下划线
- 后续字符可以是字母、数字或下划线
- 区分大小写
- 变量在使用之间，必须先初始化赋值，否则将出现名称错误

推荐的命名方式

- 变量名全部采用小写字母 pythonstring
- 简短、有意义 pystr
- 多个单词间用下划线分隔 py_str
- 变量名用名词，函数名用谓词(动词+名词) phone update_phone
- 类名采用驼峰形式 MyClass

变量赋值

- 变量赋值操作，是自右向左进行的。即，将=右边的表达式计算出结果后，赋值给左边的变量

```
>>> n = 5 + 5
>>> n
10
>>> n = n + 1   # 先将n + 1的结果计算出来，再赋值给变量n
>>> n
11
>>> n += 1   # 是n = n + 1的简写
>>> n
12
>>> import this   # python之禅：美胜丑、明胜暗、简胜繁
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
```

运算符

算术运算符

```
>>> 5 / 3
1.6666666666666667
>>> 5 // 3   # 只得到商
1
>>> 5 % 3    # 求余数，模运算
2
>>> divmod(5, 3)    # 同时得到商和余数
(1, 2)
>>> a, b = divmod(5, 3)
>>> a
1
>>> b
2
>>> 2 ** 3    # 2的3次方
8
```

比较运算符

- 得到的结果为真True或假False

```
>>> 3 == 3
True
>>> 3 != 4
True
>>> 10 < 15 < 20   # python支持连续比较
True
>>> 10 < 15 > 13   # 相当于10 < 15 and 15 > 13
True
```

逻辑运算符

```
# and两边的表达式全为True结果才为True
>>> 5 > 3 and 5 < 10
True
>>> 5 > 3 and 5 < 1
False
# or两边的表达式有一侧为True，最终结果就是True
>>> 5 > 3 or 5 < 1
True
>>> 5 > 30 or 5 < 1
False
# not是取反，将真变假、假变真
>>> 5 > 30
False
>>> not 5 > 30
True
>>> not 5 > 3
False
```

数据类型

数字

- 整体可以分为没有小数点的整数和有小数点的浮点数
- 布尔数，True的值是1，False的值是0

```
>>> True + 5
6
>>> False * 5
0
```

整数的表示方式

- 没有前缀的数字是10进制数
- 数字以0o或0O开头表示为8进制数
- 数字以0x或0X开头表示16进制数
- 数字以0b或0B开头表示2进制数

```
>>> 11
11
>>> 0o11
9   # 默认以10进制输出
>>> 0x11
17
>>> 0b11
3
# 2小时3分钟4秒是多少秒？ 2 x 60 x 60 + 3 x 60 + 4
# 0x234 => 2 x 16 x 16 + 3 x 16 + 4
# 0o234 => 2 x 8 x 8 + 3 x 8 + 4
>>> oct(20)   # 转8进制
'0o24'
```

```
>>> hex(20)   # 转16进制
'0x14'
>>> bin(20)   # 转2进制
'0b10100'
# 10000秒=>多少小时，多少分钟，多少秒
>>> divmod(10000, 60)
(166, 40)
>>> divmod(166, 60)
(2, 46)
# 最终是2小时46分40秒
```

字符串

- 字符串被定义为引号之间的字符集合
- 无论单引号,还是双引号,表示的意义相同
- python还支持三引号

```
>>> words = """hello
... ni hao
... welcome"""
>>> print(words)
hello
ni hao
welcome
>>> words   # 在python解释器中，直接写变量，将会输出内部存储的样式
'hello\nni hao\nwelcome'
>>> a = "aaa\nbbb\nccc\nddd"
>>> print(a)
aaa
bbb
ccc
ddd
```

字符串常用操作

```
>>> s = 'python'
>>> len(s)   # 求长度
6
>>> s[0]   # 下标从0开始
'p'
>>> 'python'[0]
'p'
>>> s[6]   # 错误，结束下标是5
>>> s[-1]   # 从右向左可以使用负数作为下标
'n'
>>> s[-6]
'p'
>>> s[2:4]   # 切片，起始下标包含，结束的不包含
'th'
>>> s[2:6000]   # 切片时，下标超过范围不报错
'thon'
>>> s[2:]   # 结束下标不指定，表示到结尾
```

```
'thon'
>>> s[:2]   # 超始下标不指定，表示从开头
'py'
>>> s[::2]   # 第2个冒号后面的数字是步长值
'pto'
>>> s[1::2]
'yhn'

>>> 't' in s
True
>>> 'th' in s
True
>>> 'to' in s   # to在字符串中吗？
False
>>> 'to' not in s    # to不字符串中吗？
True
>>> s + ' is good'   # 字符串拼接
'python is good'
>>> '*' * 30   # 将*号重复30遍
'******************************'
>>> s * 3
'pythonpythonpython'
```

列表

- 列表也是有顺序的
- 列表是容器类型，可以存储各种各样的数据

```
>>> alist = [10, 20, 'tom', 'jerry', [1, 2]]
>>> len(alist)
5
>>> 20 in alist
True
>>> alist[-1]
[1, 2]
>>> alist[2:4]
['tom', 'jerry']
>>> alist + [100]
>>> alist * 2
>>> alist[-1] = 50
>>> alist
[10, 20, 'tom', 'jerry', 50]
>>> alist.append(50)   # 向列表尾部追加一项
>>> alist
[10, 20, 'tom', 'jerry', 50, 50]
```

元组

- 元组与列表类似
- 可以认为元组是不可变的列表

```
>>> atuple = (10, 20, 'tom', 'jerry', 50, 50)
>>> atuple[0]
10
>>> atuple[2:4]
('tom', 'jerry')
>>> 20 in atuple
True
>>> atuple[0] = 100   # 报错，元组不可变
```

字典

```
>>> adict = {'name': 'tom', 'age': 20}
>>> len(adict)
2
>>> adict['name']   # 通过key取出value
'tom'
>>> 'name' in adict    # 'name'是字典的key吗？
True
>>> adict['age'] = 22  # 已有键，改值
>>> adict['email'] = 'tom@tedu.cn'   # 没有键，添加新值
>>> adict
{'name': 'tom', 'age': 22, 'email': 'tom@tedu.cn'}
```

数据类型分类

按存储模型分类

- 标量类型: 数值、字符串
- 容器类型: 列表、元组、字典

按更新模型分类

- 可变类型: 列表、字典
- 不可变类型: 数字、字符串、元组

按访问模型分类

- 直接访问: 数字
- 顺序访问: 字符串、列表、元组
- 映射访问: 字典