nsd_1908_py01_day01

虚拟环境

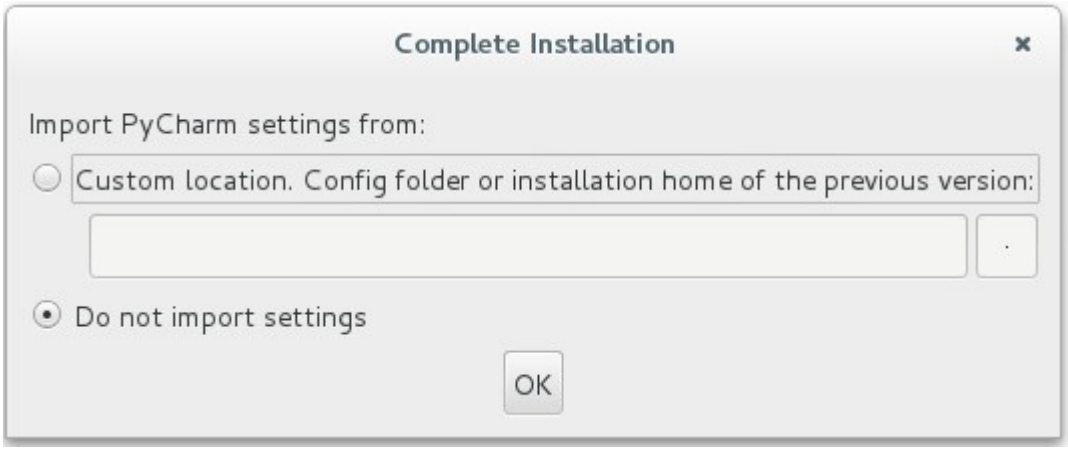
- 可以理解为一个虚拟环境就是一个隔离的工作目录
- 安装python模块，就是安装到了python的虚拟环境
- 删除虚拟环境目录，即可将环境清理干净

```
[root@room8pc16 ~]# python3 -m venv ~/nsd1908  # 创建虚拟环境
[root@room8pc16 ~]# ls ~/nsd1908
bin  include  lib  lib64  pyvenv.cfg
[root@room8pc16 ~]# source ~/nsd1908/bin/activate  # 激活虚拟环境
(nsd1908) [root@room8pc16 ~]# python --version
Python 3.6.7
(nsd1908) [root@room8pc16 ~]# which python
/root/nsd1908/bin/python
```

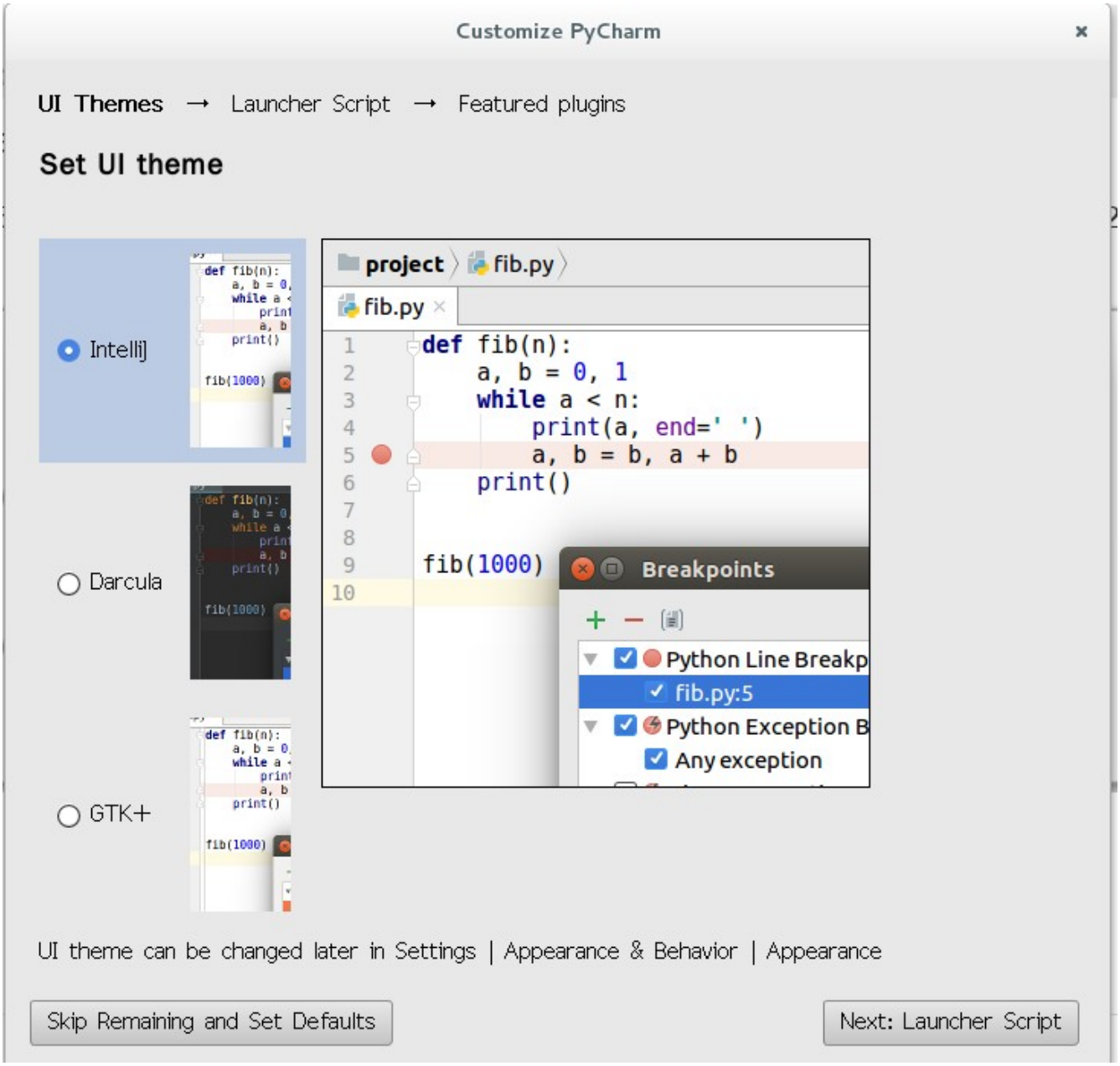
pycharm配置

- 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

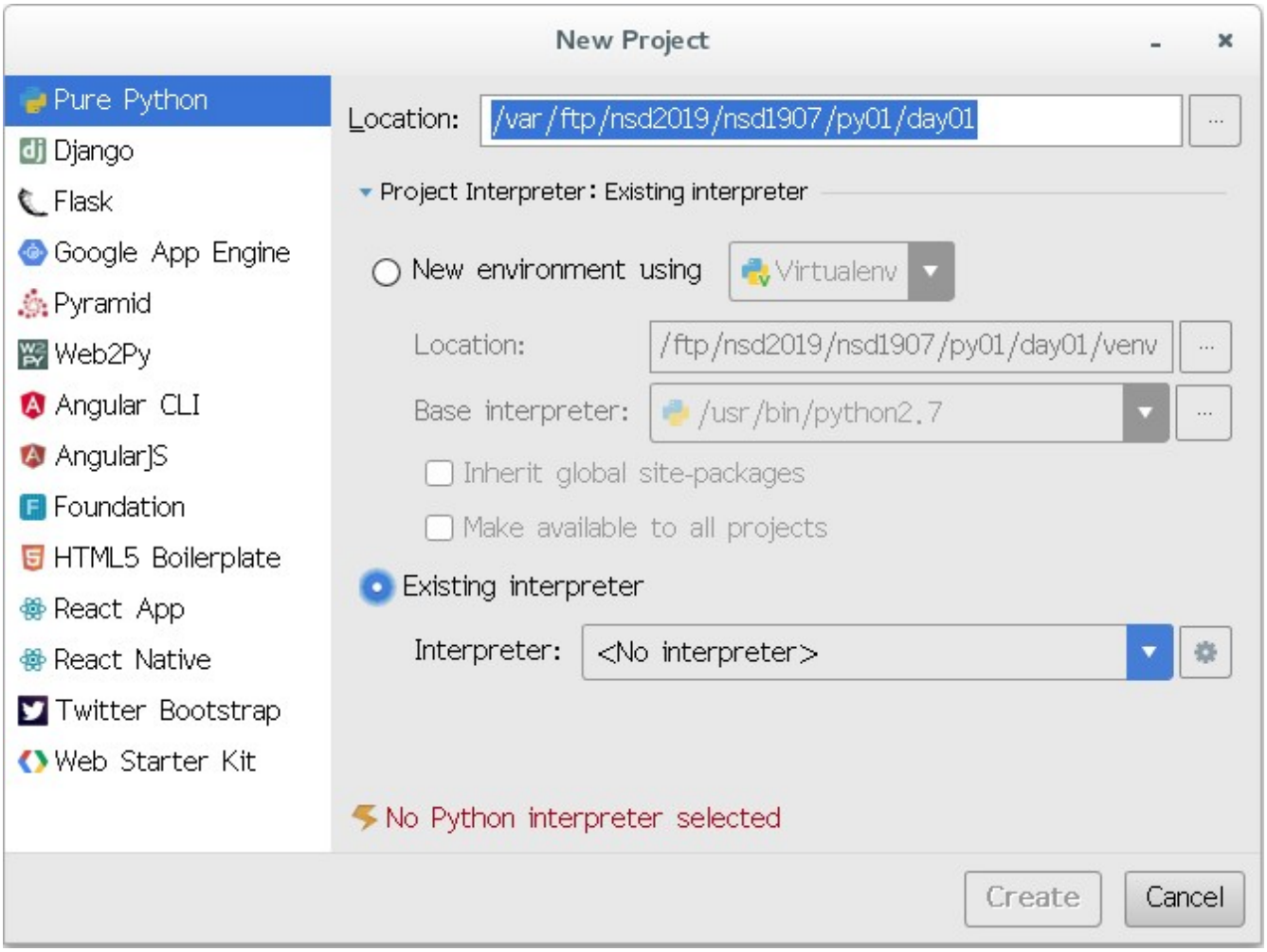


选择一种界面方案，点击Skip...

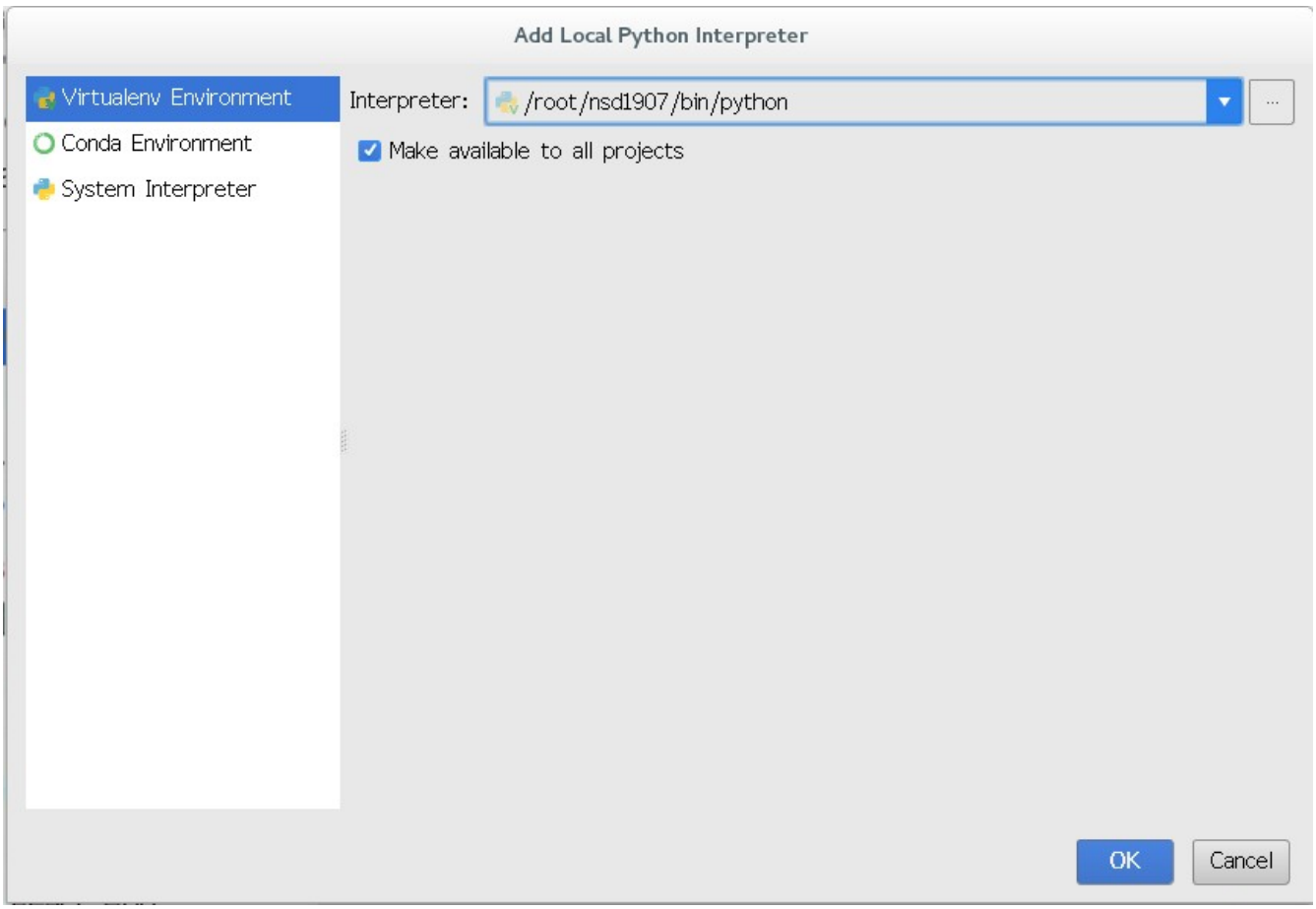

Customize PyCharm
UI Themes → Launcher Script → Featured plugins
Set UI theme
Intellij
Darcula
GTK+
project
fib.py
fib.py
def fib(n):
a, b = 0, 1
while a < n:
print(a, end=' ')
a, b = b, a + b
print()
fib(1000)
Breakpoints
Python Line Breakp
fib.py:5
Python Exception B
Any exception
UI theme can be changed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

新建项目Create...

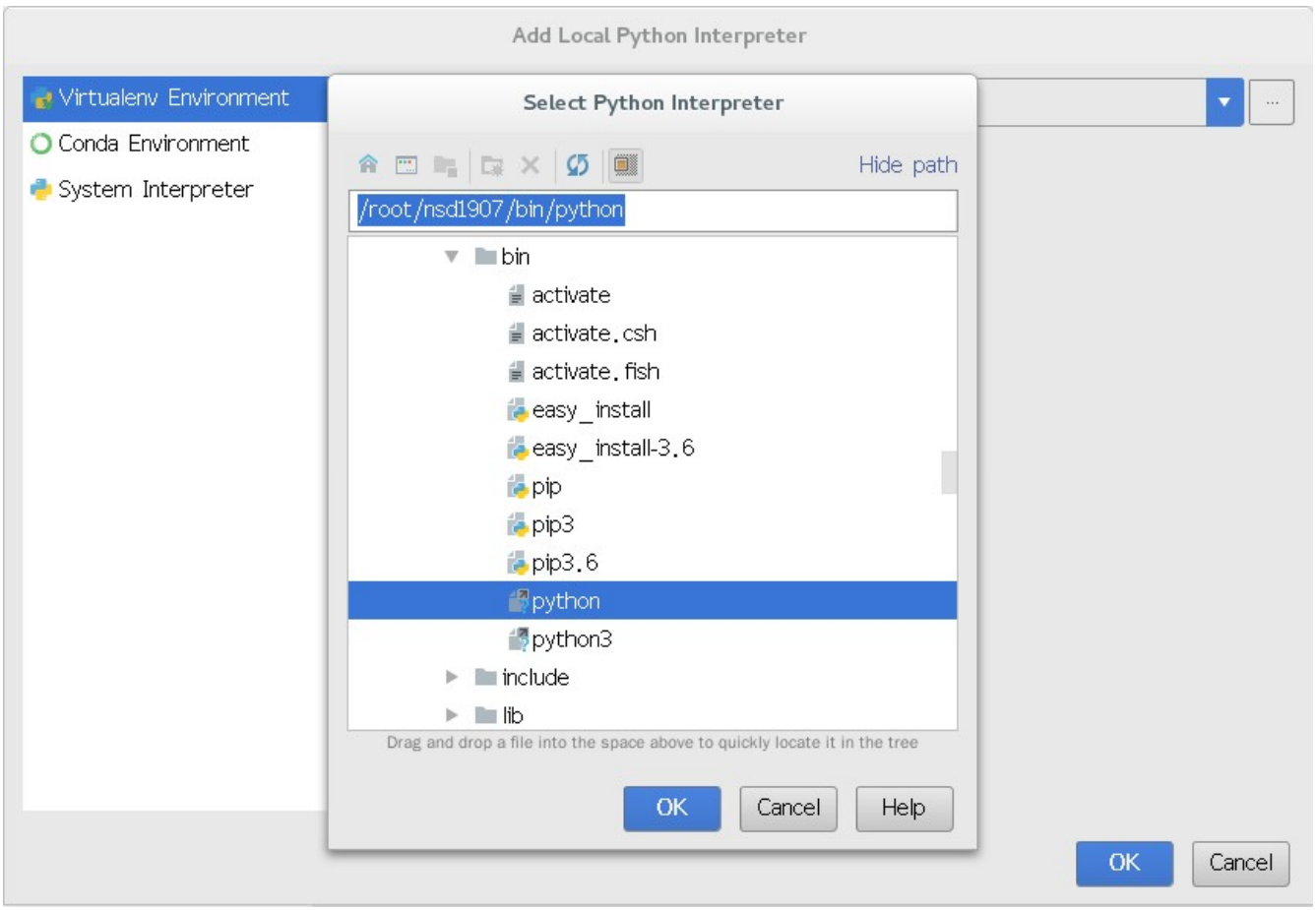
在Location处填写项目目录，点后选existing interpreter。再点击后面的齿轮图标，选add local



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



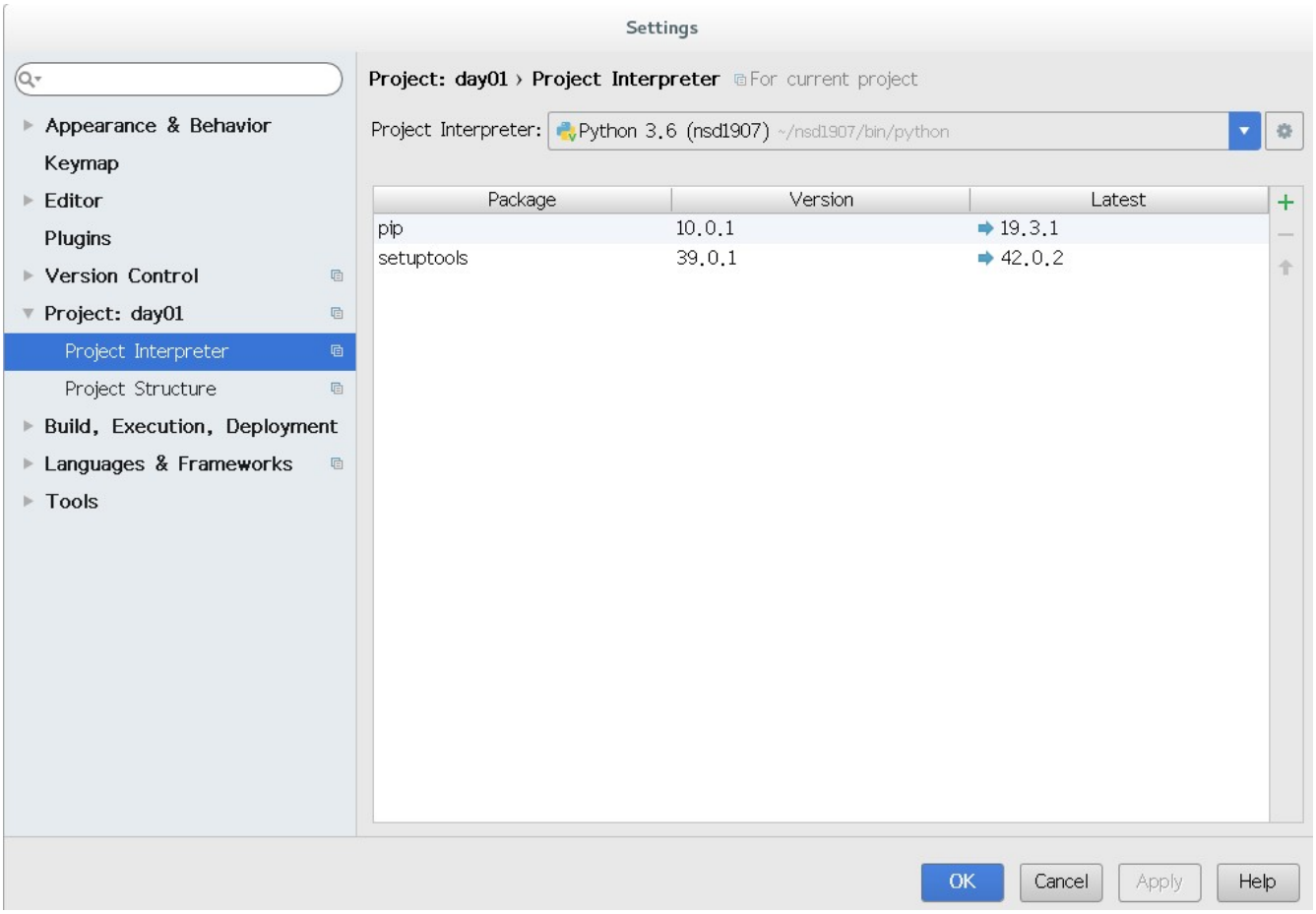
找到自己建立的虚拟环境



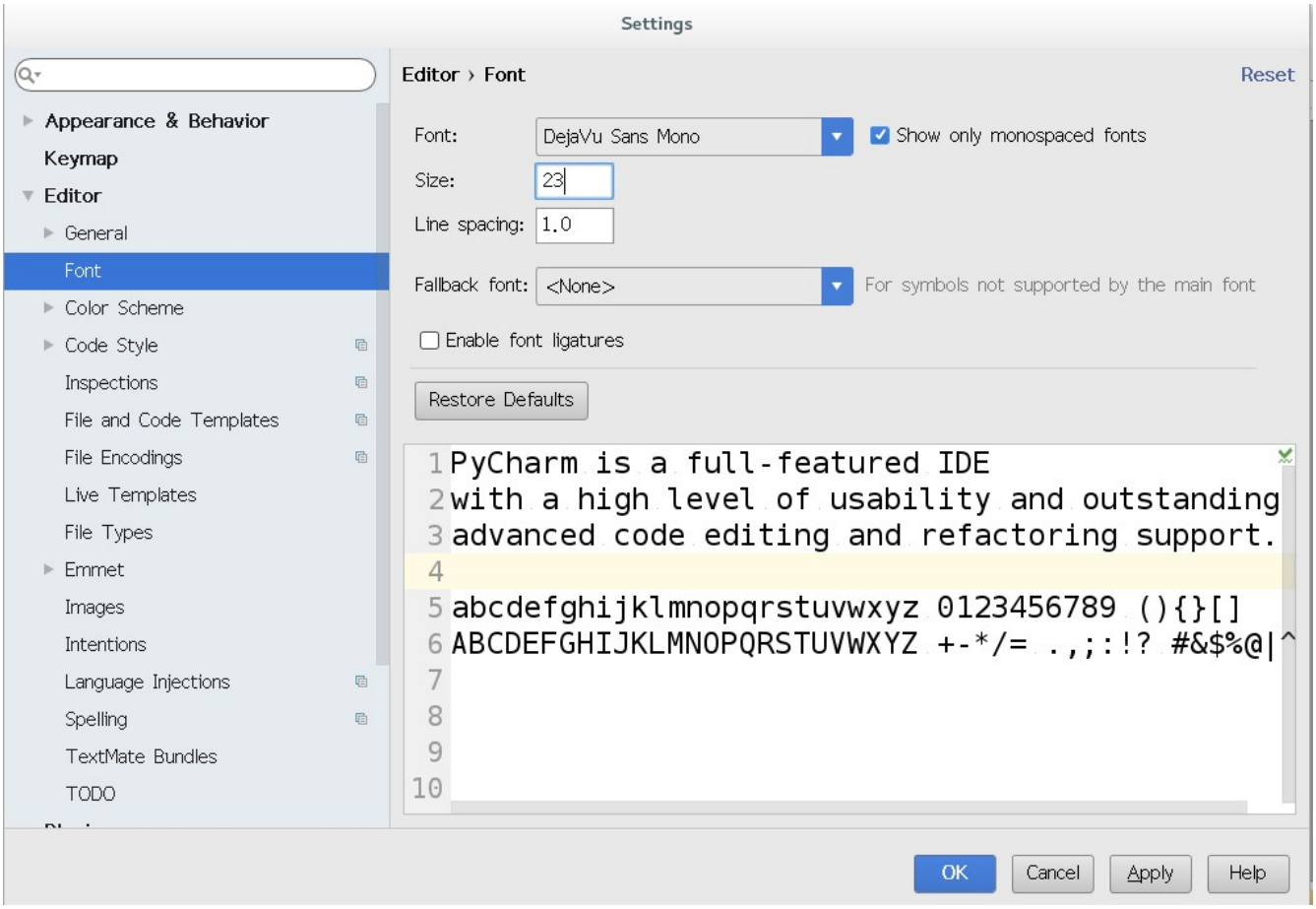
接下来，在各个页面点击OK，完成配置。

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



python语法结构

- python靠缩进表达代码逻辑

变量

- 会变化的量，如a
- 字面量literal，是不会变化的量，如100，'abc'
- 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、数字和下划线
  - 区分大小写
- 推荐的命名方法
  - 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
- 变量赋值是自右向左完成的
  - 将 = 号右边的表达式计算结果，赋值给左边的变量
  - 变量使用之前，必须先赋值进行初始化

```
>>> a = 10
>>> b = 10 + 5
>>> a = c + 5   # c未定义，报错
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'c' is not defined
>>> a = a + 5
>>> a
15
>>> a += 5   # 以上写法的简化方式
>>> a
20
>>> import this   # 查看《python之禅》
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)   # 同时得到商和余数
(1, 2)
>>> a, b = divmod(5, 3)  # 将商和余数分别赋值
>>> a
1
>>> b
2
>>> 2 ** 3   # 乘方、幂运算
8
>>> 3 ** 4
81
```

- 比较运算符，返回True或False

```
>>> 10 < 20 < 30   # py支持连续比较
True
>>> 10 < 20 > 15   # 相当于10 < 20 and 20 > 15，可读性不好，不建议
True
```

- 逻辑运算符

```
>>> 10 > 5 and 5 > 3   # and两侧全为True，最终结果才为True，否则为False
```

```
True
>>> 10 > 5 and 5 < 3
False
>>> 10 > 5 or 5 > 8    # or两侧全为False，最终才为Flase，否则为True
True
>>> 10 > 50 or 5 > 8
False
>>> 5 > 3
True
>>> not 5 > 3    # not取反，将True变False，将False变True
False
>>> not 5 > 30
True
```

数据类型

- 数字
  - 没有小数点的整数
  - 有小数点的浮点数

```
>>> True + 5    # True的值为1
6
>>> False * 5   # False的值为0
0
# python中，没有任何前缀的数字是10进制数
>>> 11
11
# 以0o或0O开头的，是8进制数
>>> 0o11
9
# 以0x或0X开头的，是16进制数
>>> 0x11
17
# 以0b或0B开头的，是2进制数
>>> 0b11
3
>>> hex(100)   # 转成16进制
'0x64'
>>> oct(100)   # 转成8进制
'0o144'
>>> bin(100)   # 转成2进制
'0b1100100'
```

- 字符串
  - 字符串被定义为引号之间的字符集合
  - 支持使用成对的单引号或双引号
  - 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
```

```
... welcome
... hao"""
>>> print(words)
hello
welcome
hao
>>> words
'hello\nwelcome\nhao'
>>> wds = "abc\nhello\nnihao"
>>> print(wds)
abc
hello
nihao

>>> s1 = 'python'
>>> len(s1)
6
>>> s1[6]   # 下标超出范围
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> 'python'[0]   # 取下标，与s1[0]完全一样
'p'
>>> s1[0]
'p'
>>> s1[-1]
'n'
>>> s1[2:4]    # 取切片，起始下标包含，结束下标不包含
'th'
>>> s1[2:6]    # 切片时，下标越界不报错
'thon'
>>> s1[2:]   # 结束下标不写，表示取到结尾
'thon'
>>> s1[:2]    # 起始下标不写，表示从开头取
'py'
>>> s1[:]   # 从开头取到结尾
'python'
>>> s1[::2]  # 第2个冒号后面的数字，是步长值
'pto'
>>> s1[1::2]
'yhn'
>>> s1[::-1]  # 步长为负，表示自右向左取
'nohtyp'
>>> s1[::-2]
'nhy'

>>> 't' in s1    # 成员关系判断，t在字符串中吗？
True
>>> 'th' in s1
True
>>> 'to' in s1   # t和o在字符串中不连续，返回False
False
>>> 'to' not in s1  # to不在字符串中吗？
```

```
True

>>> s1 + ' is cool'  # 字符串拼接
'python is cool'
>>> '*' * 30   # *重复30遍
'******************************'
>>> '#' * 30
'##############################'
>>> s1 * 5
'pythonpythonpythonpythonpython'
```

- 列表
  - 类似于shell中数组

```
>>> alist = [10, 20, 'tom', 'jerry', [1, 2]]
>>> len(alist)
5
>>> alist[0]
10
>>> alist[2:4]
['tom', 'jerry']
>>> alist[-1] = 100
>>> alist
[10, 20, 'tom', 'jerry', 100]
>>> 10 in alist
True
>>> 'o' in alist
False
>>> alist + [200]  # 列表拼接，返回新列表，原始列表不变
[10, 20, 'tom', 'jerry', 100, 200]
>>> alist * 3    # 列表重复3次，返回新列表，原始列表不变
```

- 元组
  - 元组相当于是不可变的列表

```
>>> atuple = (10, 20, 'tom', 'jerry')
>>> atuple[0]
10
>>> atuple[:2]
(10, 20)
>>> 'tom' in atuple
True
>>> atuple[0] = 100  # 元组不可变，不能改值
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

- 字典
  - 它是映射类型，没有顺序

```
>>> adict = {'name': 'tom', 'age': 20}
>>> len(adict)
2
>>> 'tom' in adict   # tom是字典的key吗？
False
>>> 'name' in adict
True
>>> adict['name']   # 通过key，找到对应的value
'tom'
```