

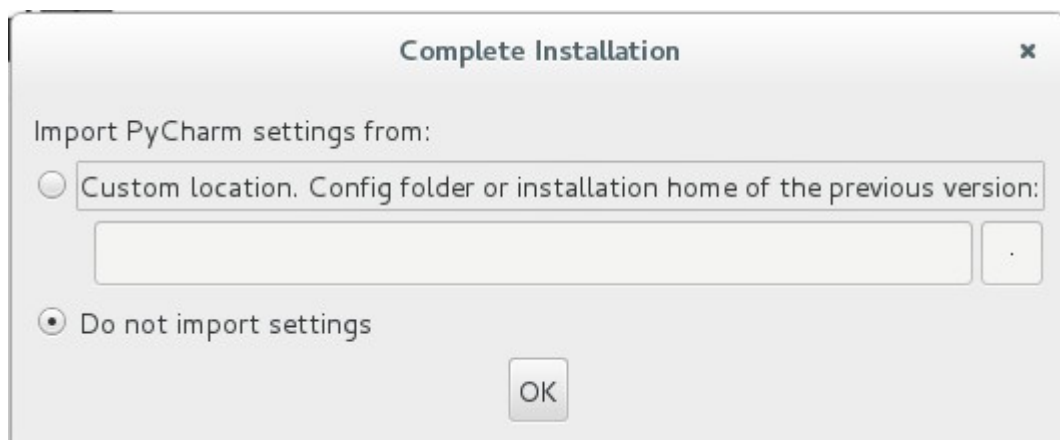
nsd1905_py01_day01

创建虚拟环境

python的虚拟环境，就是一个隔离的目录。将python放到这个隔离的目录，以后安装软件包都安装到这个虚拟环境。当虚拟环境不需要时，只要将其删除。

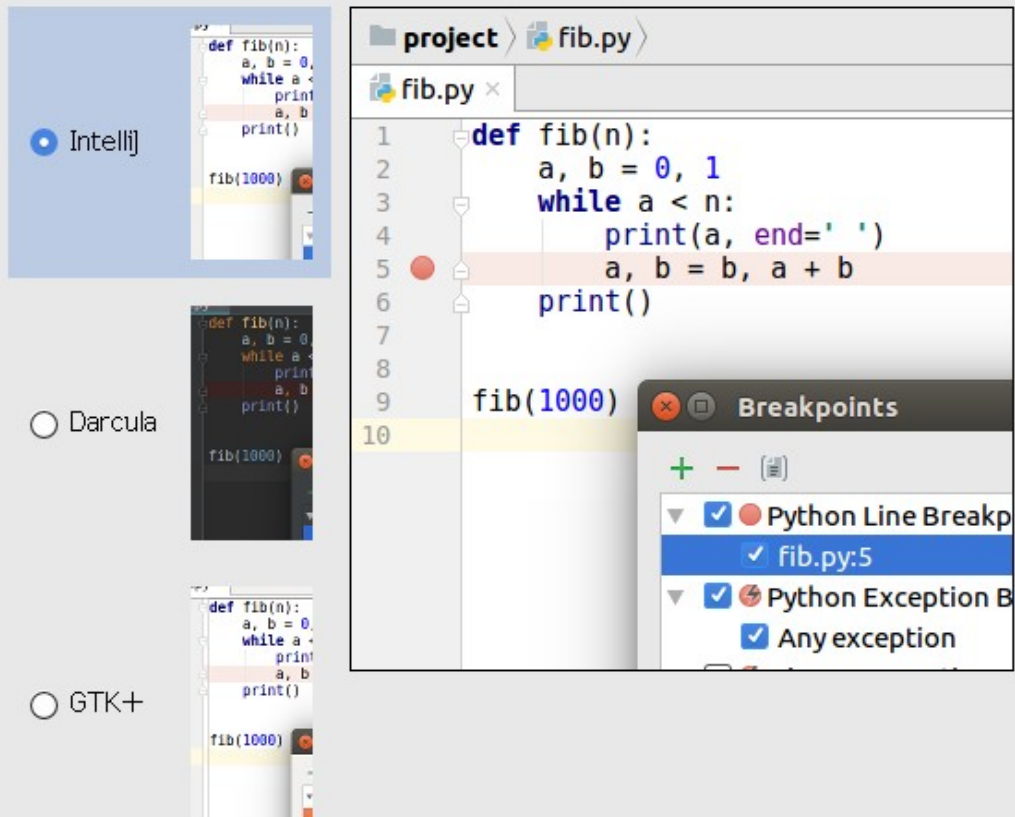
```
[root@room8pc16 nsd2019]# python3 -m venv ~/nsd1905
# 激活虚拟环境
[root@room8pc16 nsd2019]# source ~/nsd1905/bin/activate
(nsd1905) [root@room8pc16 nsd2019]# python --version
Python 3.6.7
(nsd1905) [root@room8pc16 nsd2019]# which python
/root/nsd1905/bin/python
```

pycharm配置



UI Themes → Launcher Script → Featured plugins

Set UI theme



The screenshot shows the 'Customize PyCharm' dialog with the 'Set UI theme' step. The 'IntelliJ' theme is selected. A preview of the code editor shows the 'IntelliJ' theme. A 'Breakpoints' window is also visible, showing a breakpoint set at line 5 of fib.py.

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

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

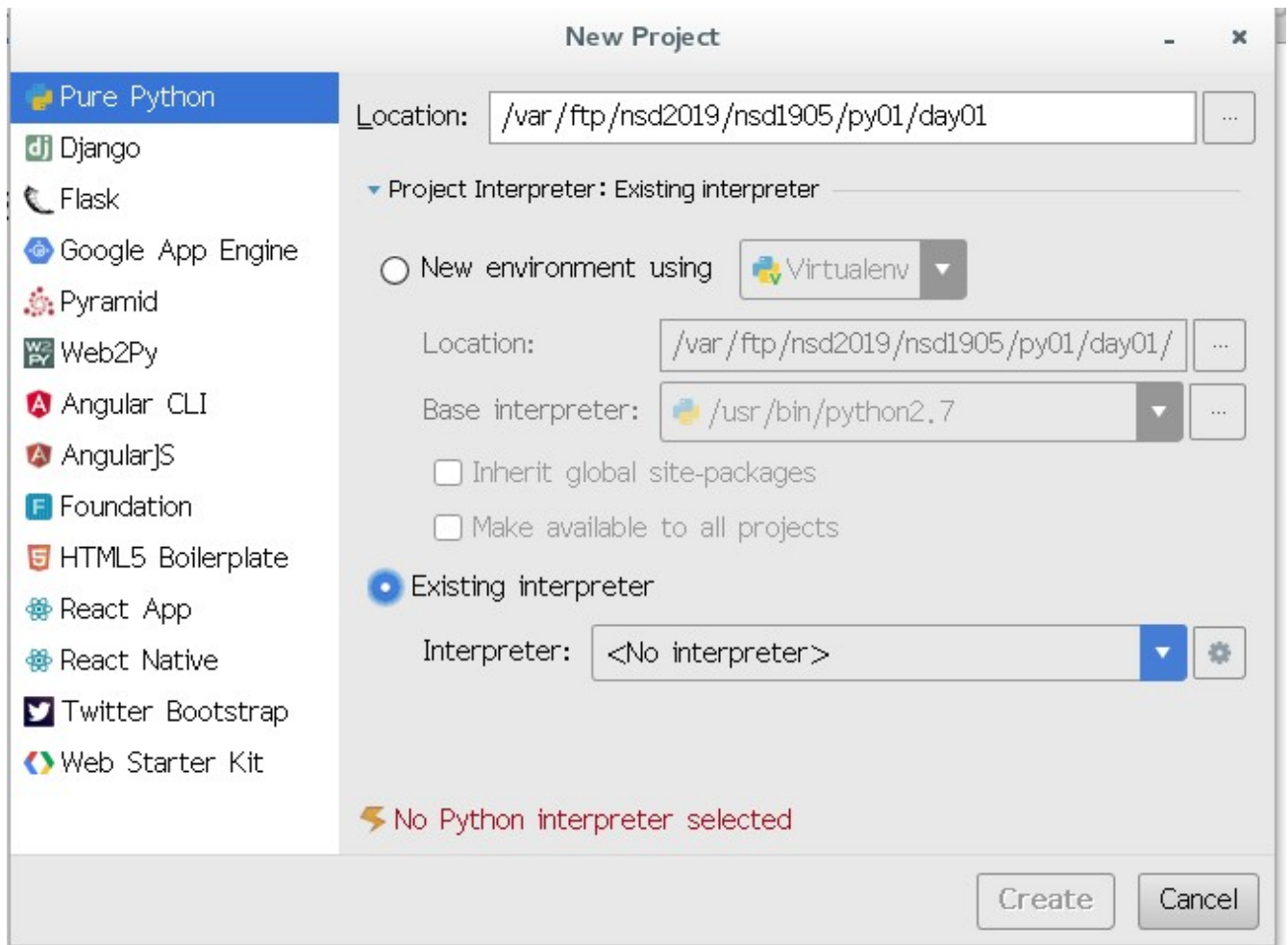
Skip Remaining and Set Defaults

Next: Launcher Script

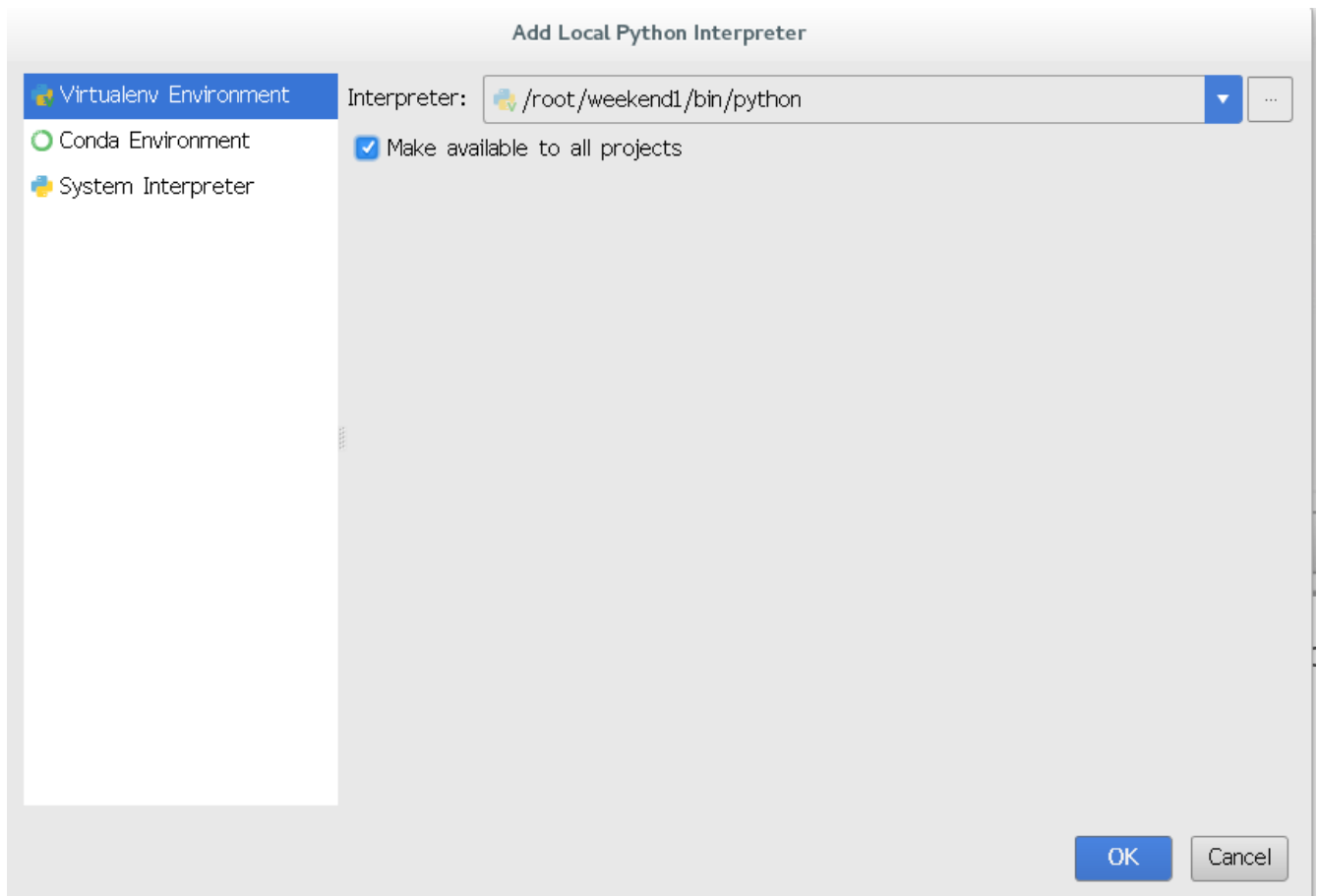
选Skip Remaining



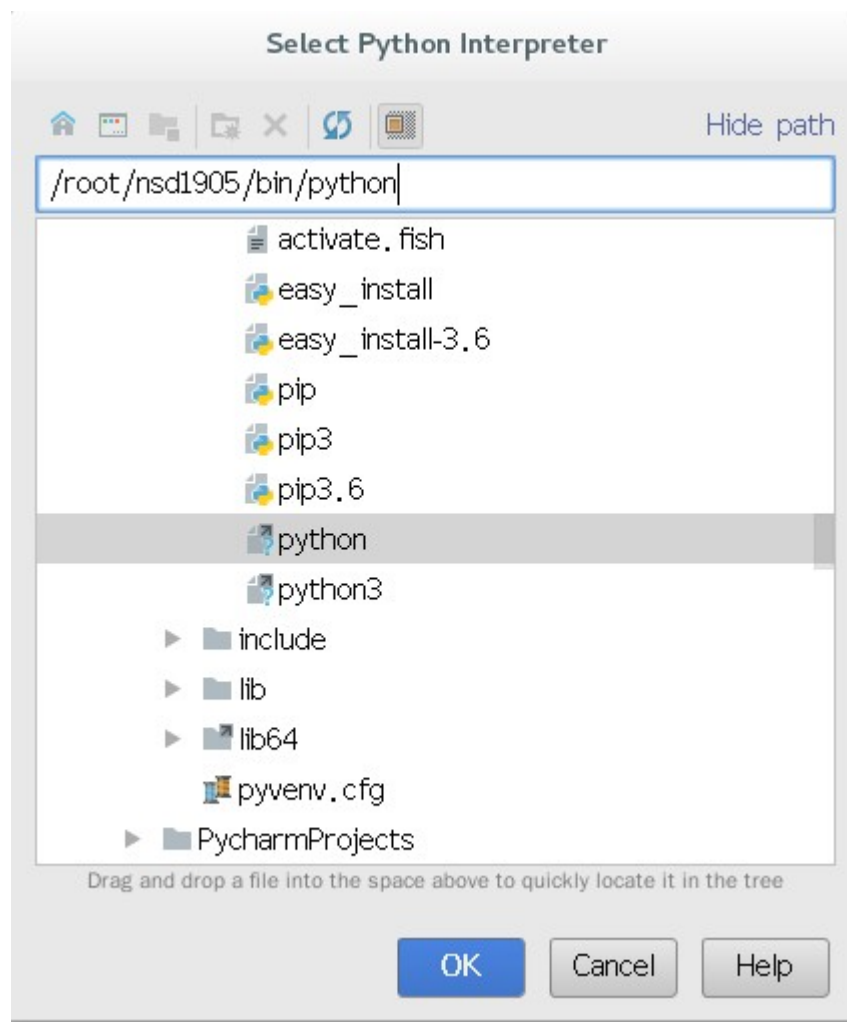
Create New Project



Location填写你的项目目录。再点击下面的Existing interpreter->右侧齿轮，点击add local

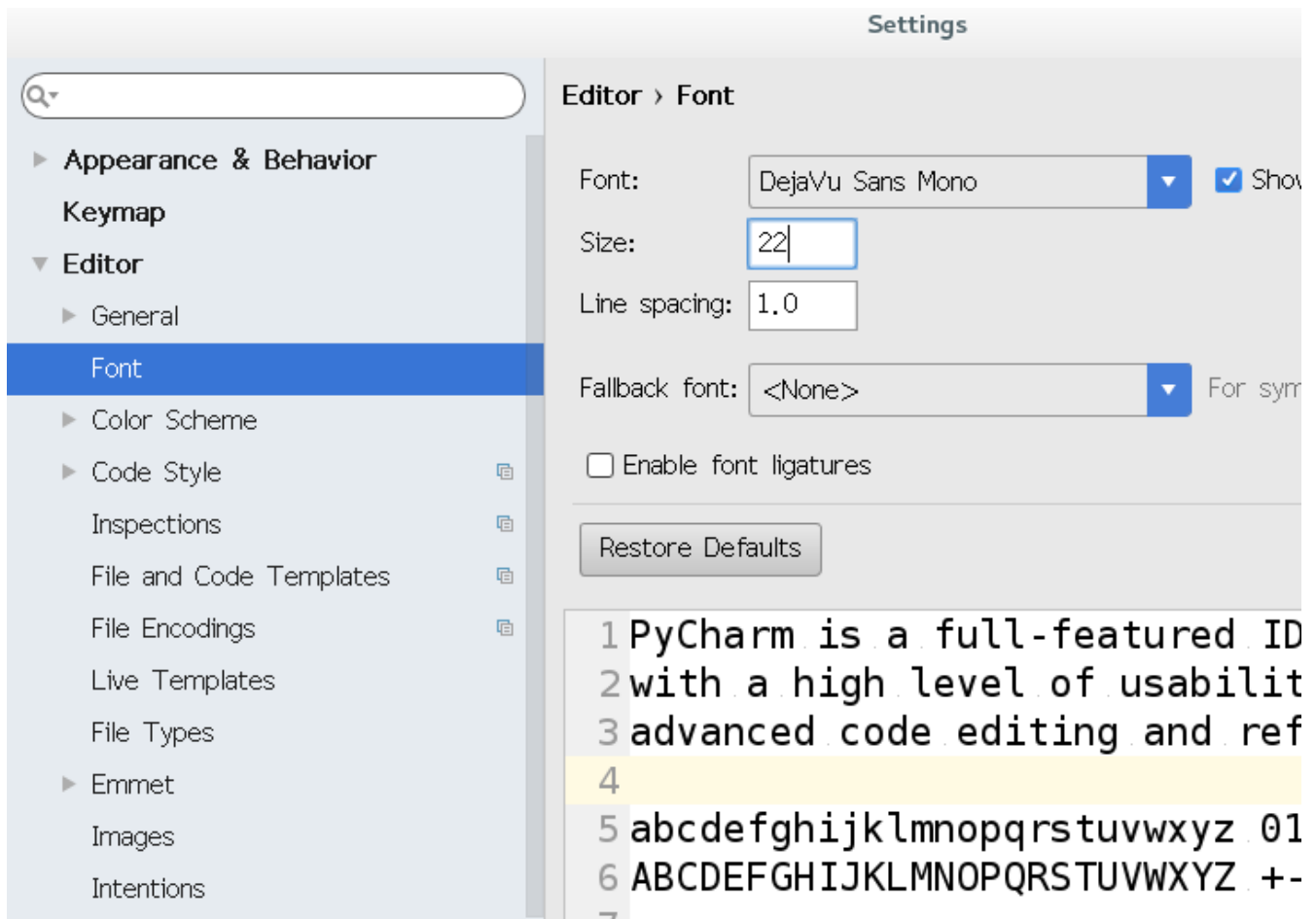


勾选Make available....，再点击三个点那个按钮



填写创建的虚拟环境的python。

调整编写代码文本的文字大小：File -> Settings



同步代码

1. 下载群共享里的压缩包并解压
2. 进入nsd2019目录

```
[root@room8pc16 nsd2019]# git pull # 如果失败则执行以下命令
# git pull https://github.com/MrZhangzhg/nsd2019.git
```

python的语法结构

- python完全靠缩进表达代码逻辑
- 顶层代码必须顶头写，不能有任何空格
- 某个代码的子代码，必须有缩进，缩进多少都行，建议4个空格

print语句

- print是函数，用于屏幕输出

input语句

- input是函数，用于获取用户的键盘输入
- input读入的数据都是字符类型的。相同类型的数据才能一起运算

```
>>> a = input('number: ')
number: 10
>>> a + 5    # 报错
>>> a + '5'   # 字符串拼接
'105'
>>> a + str(5)    # str用于将数据转成字符
'105'
>>> int(a) + 5    # int用于将字符类型的数字，转成整数
15
```

变量

- 会变化的量
- 字面量表示字面本身的含义，它不会改变。
- 命名约定
 - 首字符，必须是字母或下划线
 - 其他字符，可以是字母、数字、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
 - 有意义 pythonstring
 - 简短 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
- 变量使用之前必须赋值，进行初始化
- 变量赋值采用=，运算是自右向左进行

python之禅

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

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

- 数学运算符

```
>>> 10 / 2    # 真正的除法
5.0
>>> 5 / 3
1.6666666666666667
```



```

>>> 10 // 2    # 只保留商
5
>>> 5 // 3
1
>>> 5 % 3      # 模运算，返回余数
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     # 幂运算，2的3次方
8

```

- 比较运算符

```

>>> 5 > 3
True
>>> 5 == 5
True
>>> 5 != 5
False
>>> 10 < 20 < 30    # python支持连续比较
True
>>> 10 < 20 > 15    # 10 < 20 and 20 > 15
True

```

- 逻辑运算符

```

>>> 10 < 15 and 20 > 10    # and 两边的结果全为真，最终才为真
True
>>> 10 > 15 and 20 > 10
False
>>> 10 > 15 or 20 > 10
True
>>> 10 > 15 or 20 < 10    # or两边的结果全为假，最终才为假
False
>>> 20 > 15
True
>>> not 20 > 15    # not取反，将真变假，假变真
False

```

数据类型

数字

- 没有小数点的整数
 - 布尔数，也认为是整数

- True值为1，False值为0
- 有小数点的浮点数

```
>>> True + 1
2
>>> False * 10
0
```

- 整数有不同的进制表示方式
 - 没有任何前缀的数字为10进制
 - 0o或0O表示8进制
 - 0x或0X表示16进制
 - 0b或0B表示2进制

```
>>> 11
11
>>> 0o11
9
>>> 0x11
17
>>> 0b11
3
>>> oct(10)    # 转为8进制
'0o12'
>>> hex(10)    # 转为16进制
'0xa'
>>> bin(10)    # 转为2进制
'0b1010'
>>> oct(0x100) # 将16进制的100转为8进制
'0o400'
```

字符串

- 字符串必须使用引号引起来
- 单引号、双引号没有区别

```
>>> print('Hello World')
Hello World
>>> print("Hello World")
Hello World
>>> name = 'bob'
>>> print('hello name')
hello name
>>> print('hello %s' % name)
hello bob
>>> print('%s: %s' % (name, 20))
bob: 20
```

- python支持3引号。3引号是3个连续的单引号或双引号，它可以保留用户的输入样式

```

>>> words = '''hello
... world
... greet
... '''
>>> print(words)
hello
world
greet

>>> words
'hello\nworld\ngreet\n'
>>> danci = "hello\nhow\nare\nyou"
>>> print(danci)
hello
how
are
you

```

- 字符串切片

```

>>> py_str = 'python'
>>> len(py_str)
6
>>> py_str[0]
'p'
>>> py_str[5]
'n'
>>> py_str[6]    # 报错，IndexError
>>> py_str[-1]   # 负数，表示从右向左取
'n'
>>> py_str[-6]
'p'
>>> 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]   # 取切片，下标超出范围不会报错
'thon'
>>> py_str[2:60]
'thon'
>>> py_str[2:]    # 结束下标不写，表示取到结尾
'thon'
>>> py_str[0:2]
'py'
>>> py_str[:2]    # 开头不写，表示从开头取
'py'
>>> py_str[:]
'python'
>>> py_str[:2]    # 第2个冒号后面的数字是步长值
'pt'
>>> py_str[1:2]
'y'
>>> py_str[::-1]  # 步长值为负，表示自右向左取
'nohtyp'

```

- 字符串的拼接、重复、成员关系判断

```
>>> py_str + ' is cool.'
'python is cool.'
>>> '*' * 30
'*****'
>>> '#' * 30
'#####'
>>> py_str * 3
'pythonpythonpython'
>>> '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？
False
>>> 'to' not in py_str # to不在字符串中吗？
True
```

列表和元组

- 列表和元组也是序列对象，字符串常用的方法，在它们上面同样适用

```
>>> alist = [10, 20, 'bob', 'alice']
>>> len(alist)
4
>>> alist[0]
10
>>> alist[2:]
['bob', 'alice']
>>> alist * 2
[10, 20, 'bob', 'alice', 10, 20, 'bob', 'alice']
>>> 20 in alist
True
>>> 'bob' in alist
True

>>> atu = (10, 20, 'bob', 'alice')    # 元组用()定义
>>> atu[0]
10
>>> atu[2:]
('bob', 'alice')
>>> len(atu)
4
>>> 'alice' in atu
True

# 元组相当于静态的列表，一旦定义就不能再修改。
>>> alist[0] = 100
>>> alist
[100, 20, 'bob', 'alice']
>>> atu[0] = 100    # 报错
```

字典

- 字典是无序的
- 采用key: val对形式

```
>>> adict = {'name': 'bob', 'age': 22}
>>> len(adict)
2
>>> 'bob' in adict    # bob是字典的key吗?
False
>>> 'name' in adict   # name是字典的key吗?
True
>>> adict['name']     # 通过key取出value
'bob'
```

数据类型总结

按存储模型分类

- 标量：不能包含其他数据。数字、字符串
- 容器：可以包含其他数据。列表、元组、字典

按访问模型分类

- 直接访问：数字
- 顺序：字符串、列表、元组
- 映射：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