3. ipython 解释器

3.1 Python解释器

当我们编写Python代码时，我们得到的是一个包含Python代码的以.py为扩展名的文本文件。要运行代码，就需要Python解释器去执行.py文件。

CPython

CPython是使用最广的Python解释器。

当我们从Python官方网站下载并安装好Python 2.7或Python 3.6后，我们就直接获得了一个官方版本的解释器：CPython。这个解释器是用C语言开发的，所以叫CPython。在命令行下运行python就是启动CPython解释器。

IPython

IPython是基于CPython之上的一个交互式解释器，也就是说，IPython只是在交互方式上有所增强，但是执行Python代码的功能和CPython是完全一样的。好比很多国产浏览器虽然外观不同，但内核其实都是调用了IE。

CPython用>>>作为提示符，而IPython用In [序号]:作为提示符。

PyPy

PyPy是另一个Python解释器，它的目标是执行速度。PyPy采用JIT技术，对Python代码进行动态编译（注意不是解释），所以可以显著提高Python代码的执行速度。

绝大部分Python代码都可以在PyPy下运行，但是PyPy和CPython有一些是不同的，这就导致相同的Python代码在两种解释器下执行可能会有不同的结果。如果你的代码要放到PyPy下执行，就需要了解PyPy和CPython的不同点。

Jython

Jython是运行在Java平台上的Python解释器，可以直接把Python代码编译成Java字节码执行。

IronPython

IronPython和Jython类似，只不过IronPython是运行在微软.Net平台上的Python解释器，可以直接把Python代码编译成.Net的字节码。

关于编译器与解释器的区别

3.2 进入ipython

通常我们并不使用**Python**自带的解释器，而是使用另一个比较方便的解释器——**ipython**解释器，命令行下输入：

```
ipython
```

即可进入**ipython**解释器。

所有在**python**解释器下可以运行的代码都可以在**ipython**解释器下运行：

```
print("hello, world")
```

```
hello, world
```

可以进行简单赋值操作：

```
a = 1
```

直接在解释器中输入变量名，会显示变量的值（不需要加`print`）：

```
a
```

```
1
```

```
b = [1, 2, 3]
```

3.3 ipython magic命令

ipython解释器提供了很多以百分号`%`开头的`magic`命令，这些命令很像linux系统下的命令行命令（事实上有些是一样的）。

查看所有的`magic`命令：

```
%lsmagic
```

```
Available line magics:
%alias  %alias_magic  %autocall  %automagic  %autosave  %bookmark  %cd  %clear  %cls
  %colors  %config  %connect_info  %copy  %ddir  %debug  %dhist  %dirs  %doctest_mod
e  %echo  %ed  %edit  %env  %gui  %hist  %history  %killbgscripts  %ldir  %less  %lo
ad  %load_ext  %loadpy  %logoff  %logon  %logstart  %logstate  %logstop  %ls  %lsmag
ic  %macro  %magic  %matplotlib  %mkdir  %more  %notebook  %page  %pastebin  %pdb  %
pdef  %pdoc  %pfile  %pinfo  %pinfo2  %popd  %pprint  %precision  %profile  %prun  %
psearch  %psource  %pushd  %pwd  %pycat  %pylab  %qtconsole  %quickref  %recall  %re
hashx  %reload_ext  %ren  %rep  %rerun  %reset  %reset_selective  %rmdir  %run  %sav
e  %sc  %set_env  %store  %sx  %system  %tb  %time  %timeit  %unalias  %unload_ext
%who  %who_ls  %whos  %xdel  %xmode

Available cell magics:
%%!  %%HTML  %%SVG  %%bash  %%capture  %%cmd  %%debug  %%file  %%html  %%javascript
 %%js  %%latex  %%markdown  %%perl  %%prun  %%pypy  %%python  %%python2  %%python3
%%ruby  %%script  %%sh  %%svg  %%sx  %%system  %%time  %%timeit  %%writefile

Automagic is ON, % prefix IS NOT needed for line magics.
```

`line magic` 以一个百分号开头，作用与一行；

`cell magic` 以两个百分号开头，作用于整个cell。

最后一行`Automagic is ON, % prefix IS NOT needed for line magics.`说明在此时即使不加上`%`也可以使用这些命令。

使用 `whos` 查看当前的变量空间：

```
%whos
```

```
Variable   Type    Data/Info
----------------------------
a          int     1
b          list    n=3
```

使用 `reset` 重置当前变量空间：

```
%reset -f
```

再查看当前变量空间：

```
%whos
```

```
Interactive namespace is empty.
```

使用 `pwd` 查看当前工作文件夹：

```
%pwd
```

```
'C:\\Code\\python\\notes-python-master\\01-python-tools'
```

使用 `mkdir` 产生新文件夹：

```
%mkdir demo_test
```

使用 `cd` 改变工作文件夹：

```
%cd demo_test/
```

```
C:\Code\python\notes-python-master\01-python-tools\demo_test
```

使用 `writefile` 将cell中的内容写入文件：

```
%%writefile hello_world.py
print ("hello world")
```

```
Writing hello_world.py
```

使用 `ls` 查看当前工作文件夹的文件：

```
%ls
```

```
 驱动器 C 中的卷是 Windows
 卷的序列号是 1CEB-7ABE

 C:\Code\python\notes-python-master\01-python-tools\demo_test 的目录

2018/03/14  13:48    <DIR>          .
2018/03/14  13:48    <DIR>          ..
2018/03/14  13:48                21 hello_world.py
               1 个文件             21 字节
               2 个目录 125,753,786,368 可用字节
```

使用 `run` 命令来运行这个代码：

```
%run hello_world.py
```

```
hello world
```

删除这个文件：

```
import os
os.remove('hello_world.py')
```

查看当前文件夹，`hello_world.py` 已被删除：

```
%ls
```

```
 驱动器 C 中的卷是 Windows
 卷的序列号是 1CEB-7ABE

 C:\Code\python\notes-python-master\01-python-tools\demo_test 的目录

2018/03/14  13:48    <DIR>          .
2018/03/14  13:48    <DIR>          ..
               0 个文件              0 字节
               2 个目录 125,753,438,208 可用字节
```

返回上一层文件夹：

```
%cd ..
```

```
C:\Code\python\notes-python-master\01-python-tools
```

使用 `rmdir` 删除文件夹：

```
%rmdir demo_test
```

使用 `hist` 查看历史命令：

```
%hist
```

```
[关于编译器与解释器的区别](http://blog.csdn.net/touzani/article/details/1625760)
```

```
print ("hello, world")
a = 1
a
b = [1, 2, 3]
%lsmagic
%whos
%reset -f
%whos
%pwd
%mkdir demo_test
%cd demo_test/
%%writefile hello_world.py
print ("hello world")
%ls
%run hello_world.py
import os
os.remove('hello_world.py')
%ls
%cd ..
%rmdir demo_test
%hist
```

3.4 ipython 帮助

使用 `?` 查看函数的帮助：

```
sum?
```

使用 `??` 查看函数帮助和函数源代码（如果是用**python**实现的）：

```
# 导入numpy和matplotlib两个包
%pylab
# 查看其中sort函数的帮助
sort??
```

```
Using matplotlib backend: Qt5Agg
Populating the interactive namespace from numpy and matplotlib
```

ipython 支持使用 `<tab>` 键自动补全命令。

使用 `_` 使用上个cell的输出结果：

```
a = 12
a
```

```
12
```

```
_ + 13
```

```
25
```

可以使用 `!` 来执行一些系统命令。

```
!ping baidu.com
```

```
正在 Ping baidu.com [111.13.101.208] 具有 32 字节的数据:
来自 111.13.101.208 的回复: 字节=32 时间=30ms TTL=54
来自 111.13.101.208 的回复: 字节=32 时间=30ms TTL=54
来自 111.13.101.208 的回复: 字节=32 时间=31ms TTL=54
来自 111.13.101.208 的回复: 字节=32 时间=30ms TTL=54

111.13.101.208 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 30ms, 最长 = 31ms, 平均 = 30ms
```

当输入出现错误时，**ipython**会指出出错的位置和原因：

```
1 + "hello"
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-25-d37bedb9732a> in <module>()
----> 1 1 + "hello"


TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

记录输入输出：

- %logstart #记录整个会话，包括之前的

- %logstop
- %logon
- %logoff

```
%logstart
```

```
Activating auto-logging. Current session state plus future input saved.
Filename       : ipython_log.py
Mode           : rotate
Output logging : False
Raw input log  : False
Timestamping   : False
State          : active
```