自动化测试环境搭建

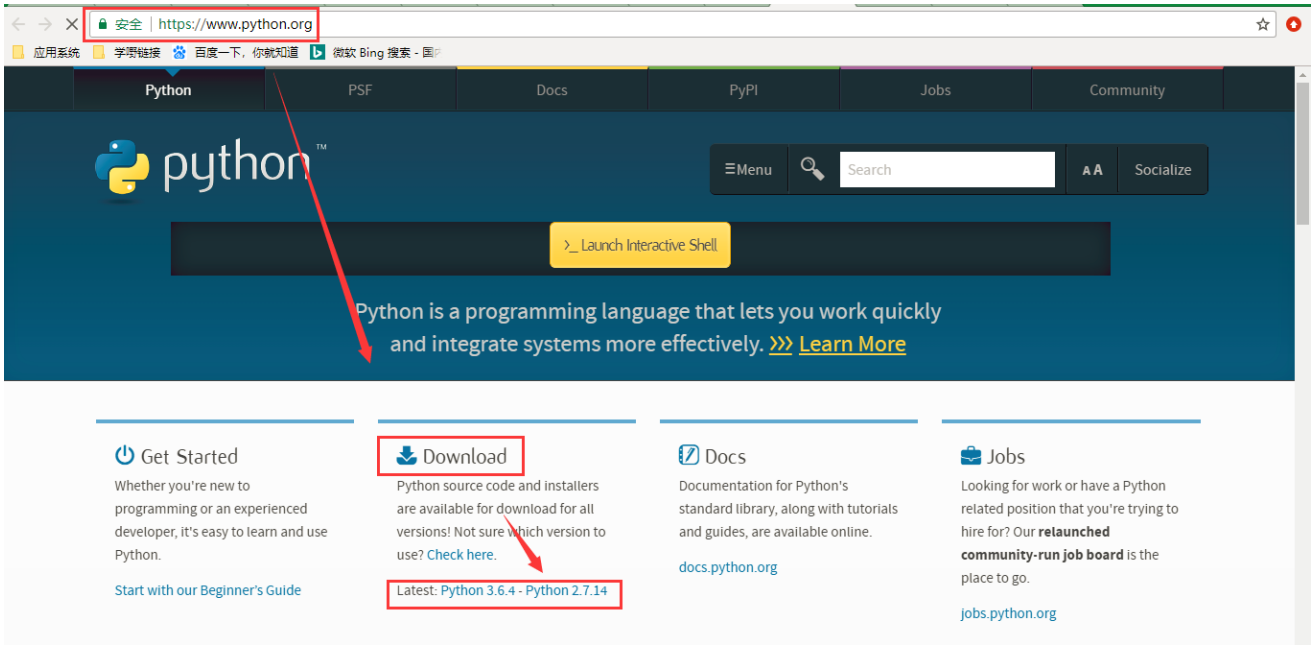
1.Python安装

1.1Python选择

如果你是第一次接触Python，一定会迷惑Python为什么会提供Python2.x 和 Python3.x两个版本？那么，直接使用Python3.x的最新版本就好了。因为Python2.x预计到2020年不在维护。注：Python3以后版本不再向Python2进行兼容。

1.2Python下载

打开 Python官网，找到“Download”, 在其内容下找到【Latest】直接点击链接：Python 3.6.4 进入下载界面； 根据列表中选择自己的平台（Windows/Mac）,一般的Linux平台已经自带的Python，所以不需要安装，通过打开“终端”，输入“python”命令来验证是否已经安装。本文以Windows平台安装为例。



如果你是Windows平台用户，会遇到一个版本为什么会提供多种个下载链接。例如：

Python 3.6.4 - 2017-12-19

- Download Windows x86 web-based installer
- Download Windows x86 executable installer
- Download Windows x86 embeddable zip file
- Download Windows x86-64 web-based installer
- **Download Windows x86-64 executable installer**
- Download Windows x86-64 embeddable zip file
- Download Windows help file

  相应解析：

  Windows x86 只支持32位的系统；

  Windows x86-64 支持64位的系统；

  web-based 格式文件在安装的过程中需要联网；

  executable 格式文件可执行文件(.exe)方式安装；

  embeddable zip file 嵌入式版本，可以集成到其它应用中。

Files

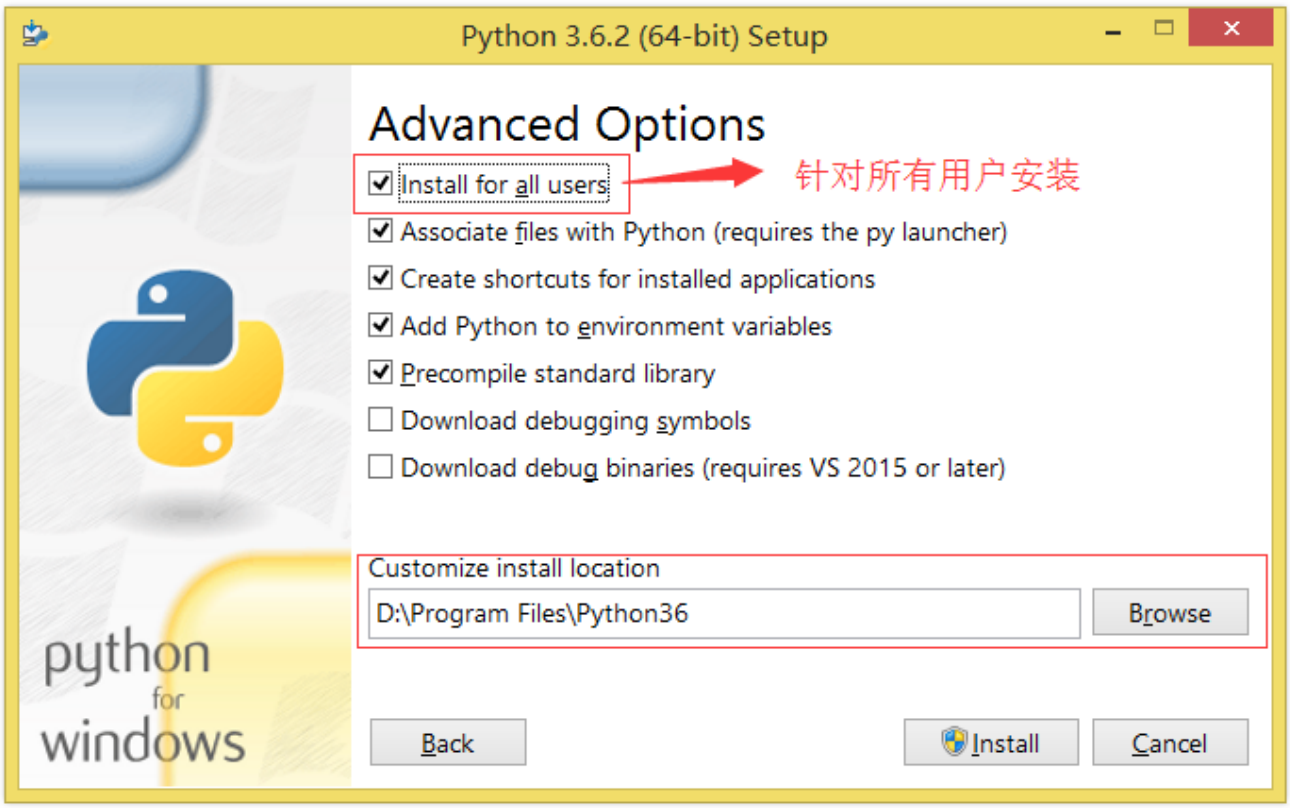
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		9de6494314ea199e3633211696735f65	22710891	SIG
XZ compressed source tarball	Source release		1325134dd525b4a2c3272a1a0214dd54	16992824	SIG
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	9fba50521dffa9238ce85ad640abaa92	27778156	SIG
Windows help file	Windows		17cc49512c3a2b876f2ed8022e0afe92	8041937	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64, not Itanium processors	d2fb546fd4b189146dbefeba85e7266b	7162335	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64, not Itanium processors	bee5746dc6ece6ab49573a9f54b5d0a1	31684744	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64, not Itanium processors	21525b3d132ce15cae6ba96d74961b5a	1320128	SIG
Windows x86 embeddable zip file	Windows		15802be75a6246070d85b87b3f43f83f	6400788	SIG
Windows x86 executable installer	Windows		67e1a9bb336a5eca0efcd481c9f262a4	30653888	SIG
Windows x86 web-based installer	Windows		6c8ff748c554559a385c986453df28ef	1294088	SIG

1.3Python安装

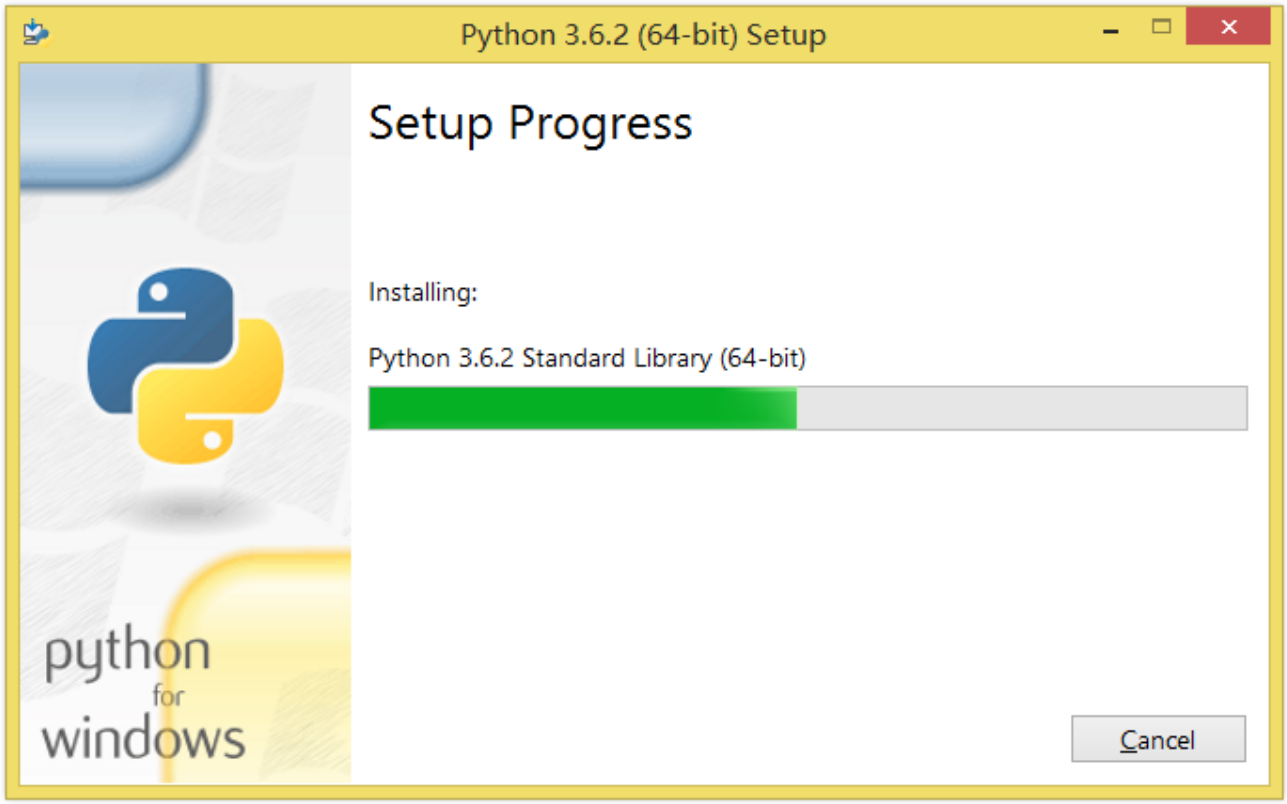
下载到本地双击安装，这里以【自定义安装】为例。安装python的时候需要勾选“Add Python 3.x to PATH”：

如果没有特殊需求，就全选上。**注意：pip要勾选**，这样下面安装selenium的时候就不用再单独安装；

若选中【Install for all users】安装目录会改变，请根据自己的需求修改安装路径再点击 【Install 】进行下一步：



安装中...



安装完成!

1.4环境变量配置

注意：在安装的过程中需要勾选："Add Python 3.x to PATH"，如果没有勾选，需要在安装完成之后，将Python的安装目录（如：C:\Python36）添加到环境变量PATH下面。可以通过以下方式设置：

- 右键点击"计算机"，然后点击"属性"
- 然后点击"高级系统设置"
- 点击“环境变量”
- 选择"系统变量"窗口下面的"Path",双击即可！
- 然后在"Path"行，添加python安装路径即可(我的D:\Python35)，所以在后面，添加该路径即可。

安装完成后，打开Windows命令提示符（cmd）/ Linux终端输入：

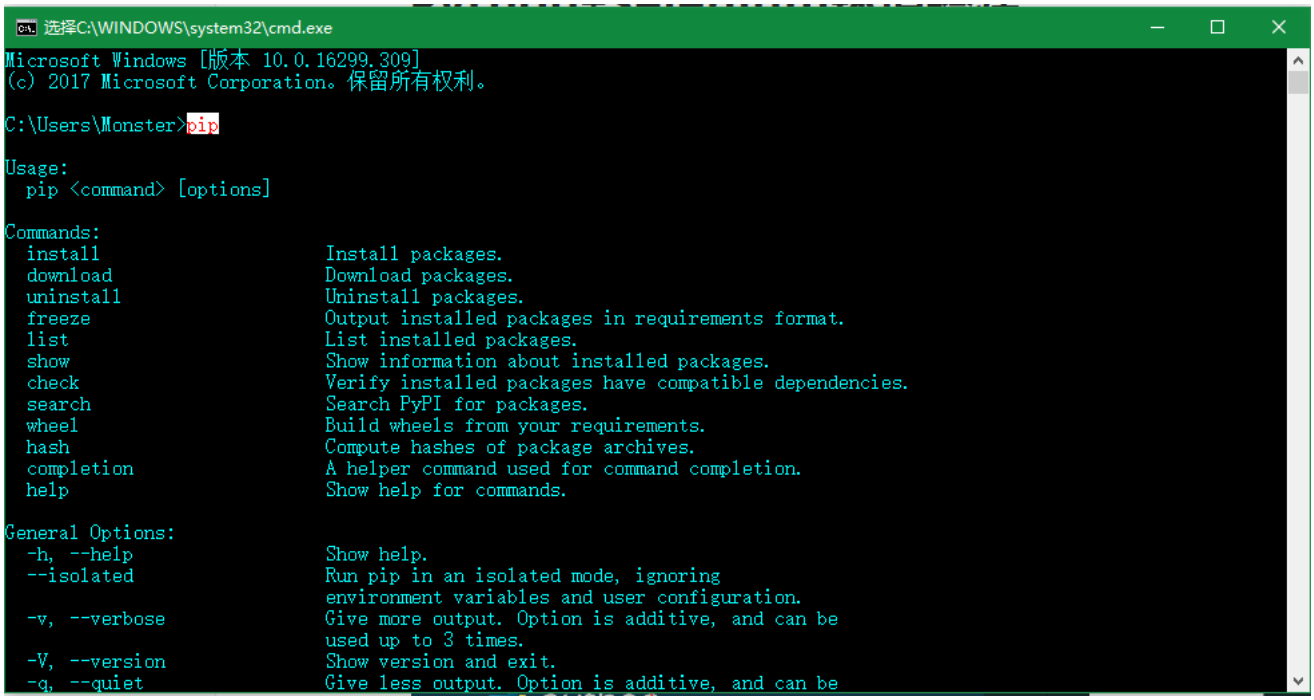
输入命令“python”以后。如上图控制符显示“>>>”则表示python安装成功；

2.Selenium安装

2.1通过pip安装

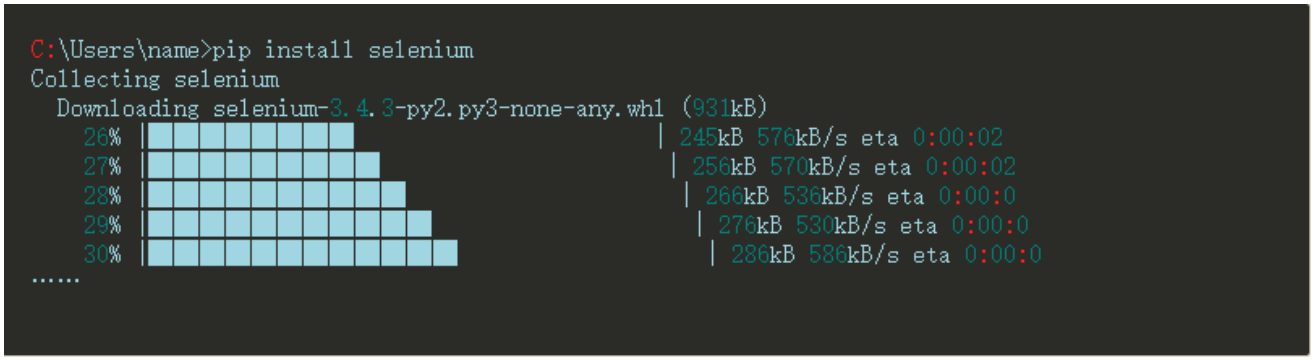
安装selenium前需要确保python安装成功，并且已经安装了pip。安装 pip 的好处是可以使用 pip 命令方便地安装 Python 第三方库。在通过 pip 安装 Python 第三方库时，如果只输入包名，则默认安装当前库中最新的版本，如果我们不想安装最新版本的包，则可以在包名后面加版本号。通过以下方法确认pip是否已经安装成功：

在Windows命令提示符（cmd）终端输入：



确保pip命令可用，如果提示“pip不是内部或外部命令”，需要将将pip的安装目录（如：C:\Python36\Scripts）添加到环境变量PATH下面。

接下来通过pip命令安装Selenium:**pip install selenium**



安装显示100%则表示安装完成。

3.浏览器驱动安装

3.1浏览器驱动下载

当官方selenium升级到3.0之后，对不同的浏览器驱动进行了规范。如果想使用selenium驱动不同的浏览器，必须单独下载并设置不同的浏览器驱动。

- Firefox浏览器驱动：geckodriver
- Chrome浏览器驱动：chromedriver
- IE浏览器驱动：IEDriverServer
- Edge浏览器驱动：MicrosoftWebDriver
- Opera浏览器驱动：operadriver
- PhantomJS浏览器驱动：phantomjs

注：部分浏览器驱动地址需要科学上网。

3.1设置浏览器驱动

设置浏览器的地址非常简单。 我们可以手动创建一个存放浏览器驱动的目录，如： C:\driver , 将下载的浏览器驱动文件（例如：chromedriver、geckodriver）丢到该目录下。

我的电脑->属性->系统设置->高级->环境变量->系统变量->Path，将“C:\driver”目录添加到Path的值中。

- Path
- ;C:\driver

3.2浏览器验证

验证不同的浏览器驱动是否正常使用。（**需要安装脚本编辑器，该部分可以在本文末端一起执行**）

```
from selenium import webdriver

driver = webdriver.Firefox()    # Firefox浏览器
driver = webdriver.Chrome()     # Chrome浏览器
driver = webdriver.Ie()         # Internet Explorer浏览器
driver = webdriver.Edge()       # Edge浏览器
driver = webdriver.Opera()      # Opera浏览器
driver = webdriver.PhantomJS()    # PhantomJS
.....
```

4.脚本编辑器安装

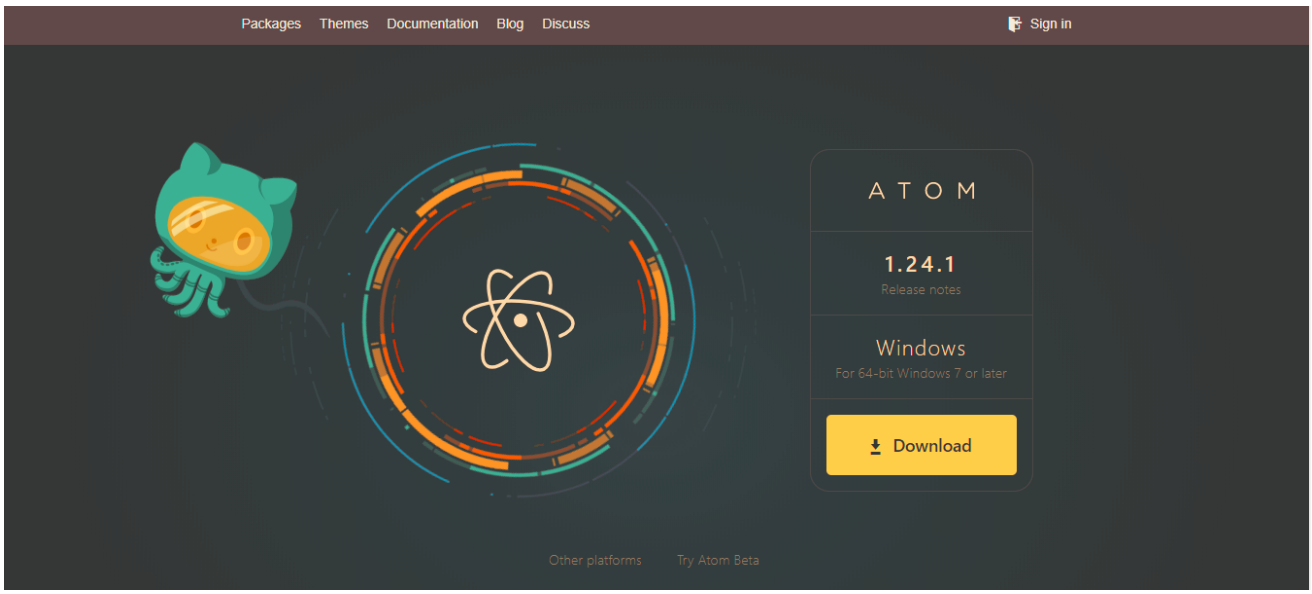
4.1Atom简介

Atom是专门为程序员推出的一个跨平台文本编辑器。具有简洁和直观的图形用户界面，并有很多有趣的特点：支持CSS，HTML，JavaScript等网页编程语言。它支持宏，自动完成分屏功能，集成了文件管理器。

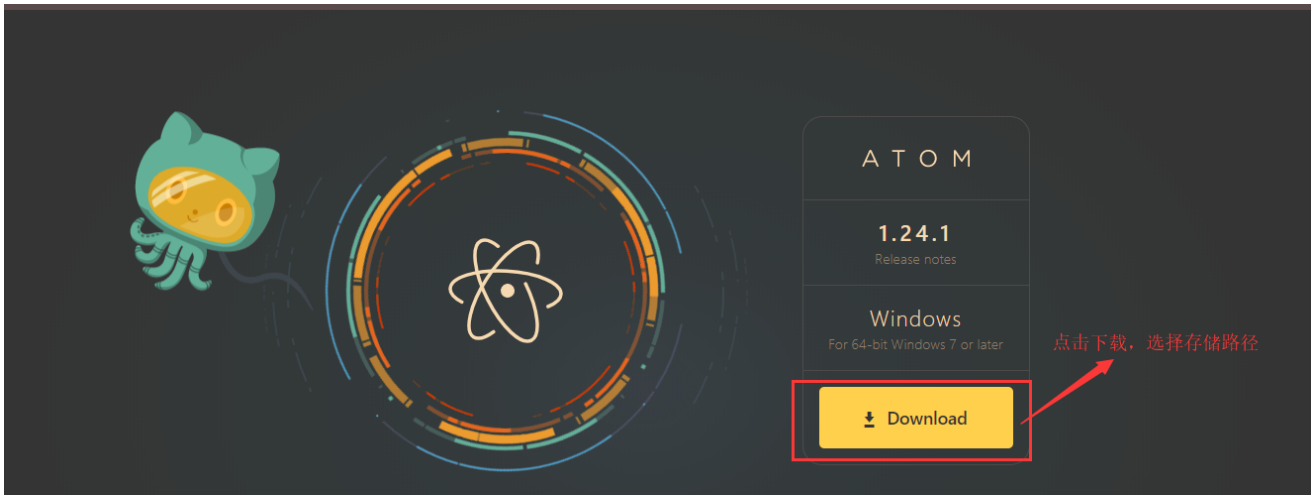
Atom编辑器上手简单，零门槛，开源免费，资源占用不高，自身支持的功能就挺多，配置起来也很方便，还有很多插件可以选择，支持中文。

4.2Atom下载

进入atom的官网；

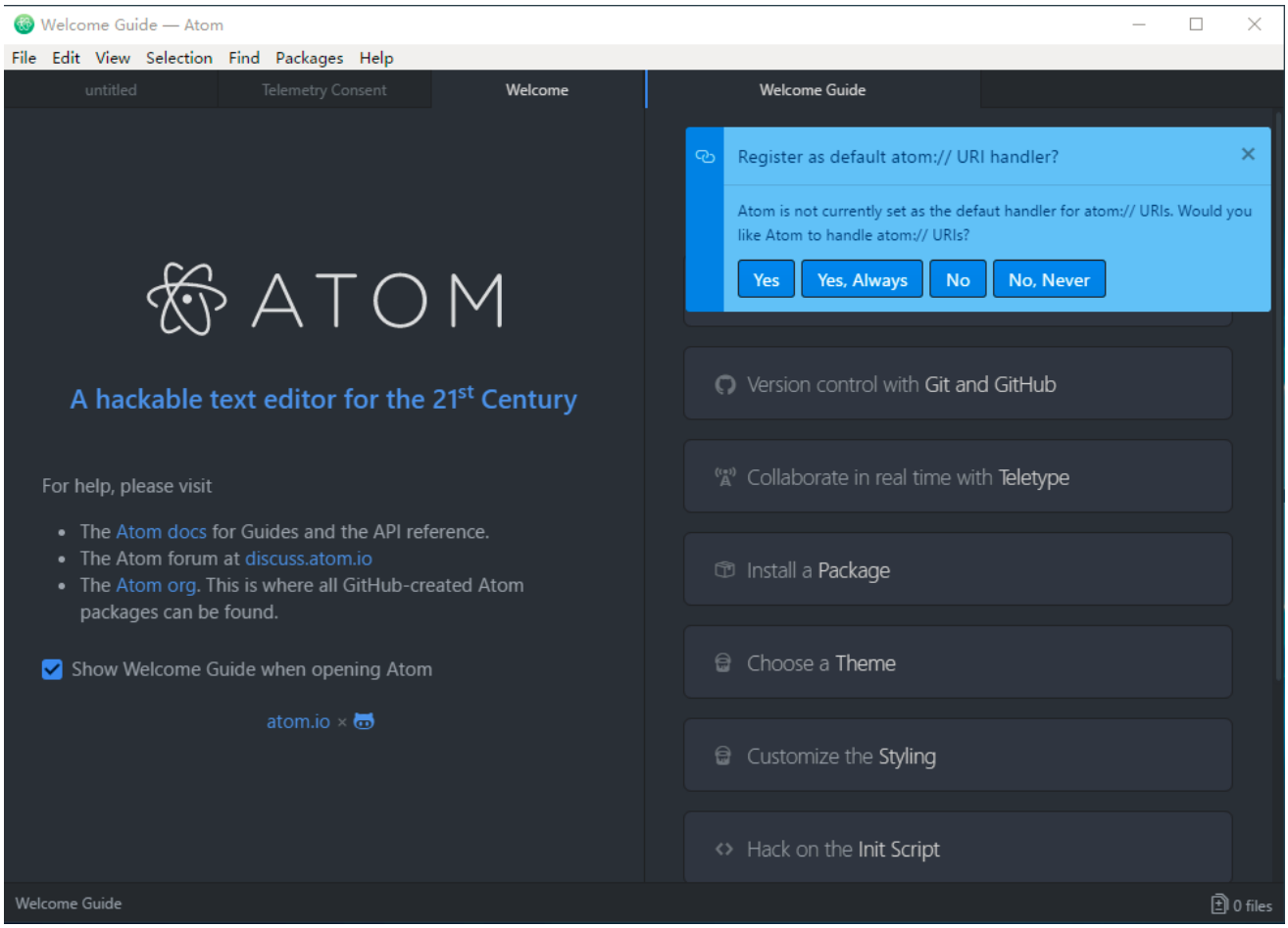


根据系统类型选择安装包包，本文针对widows平台安装，点击下载，根据个人需求选择存储路径；



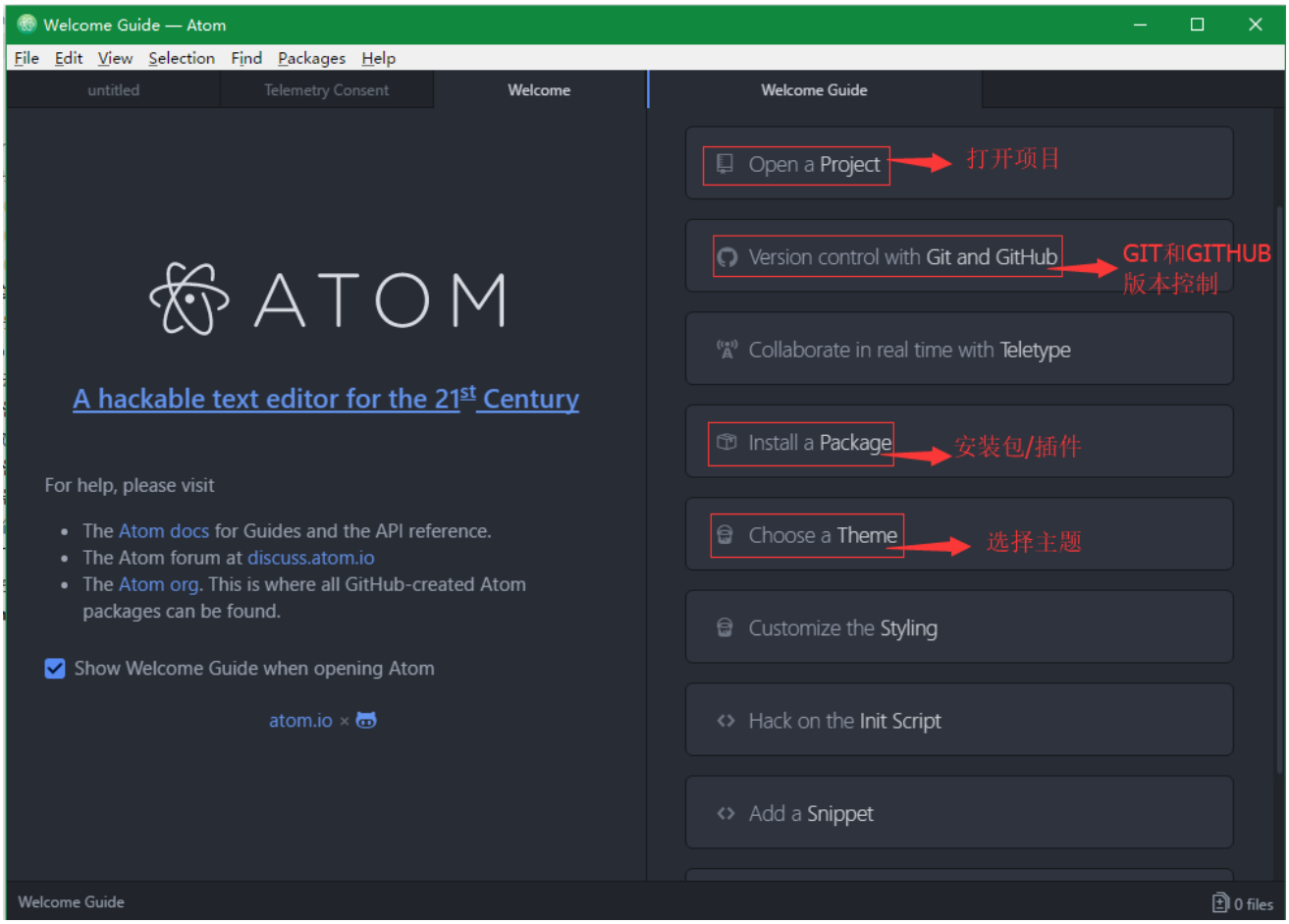
4.2Atom安装

双击AtomSetup-x64.exe直接运行默认安装，显示以下界面表示安装完成：

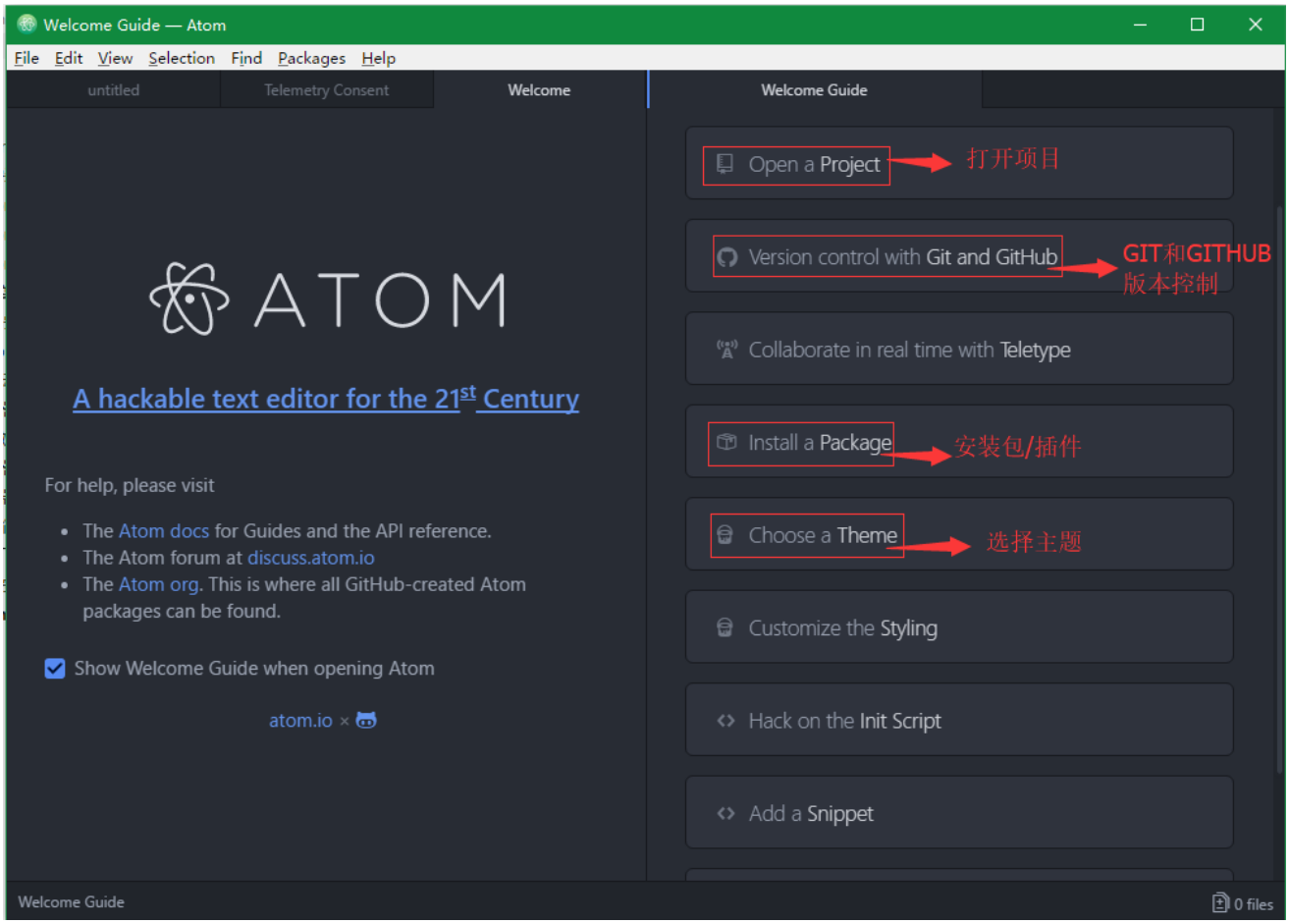


4.3Atom配置及插件安装

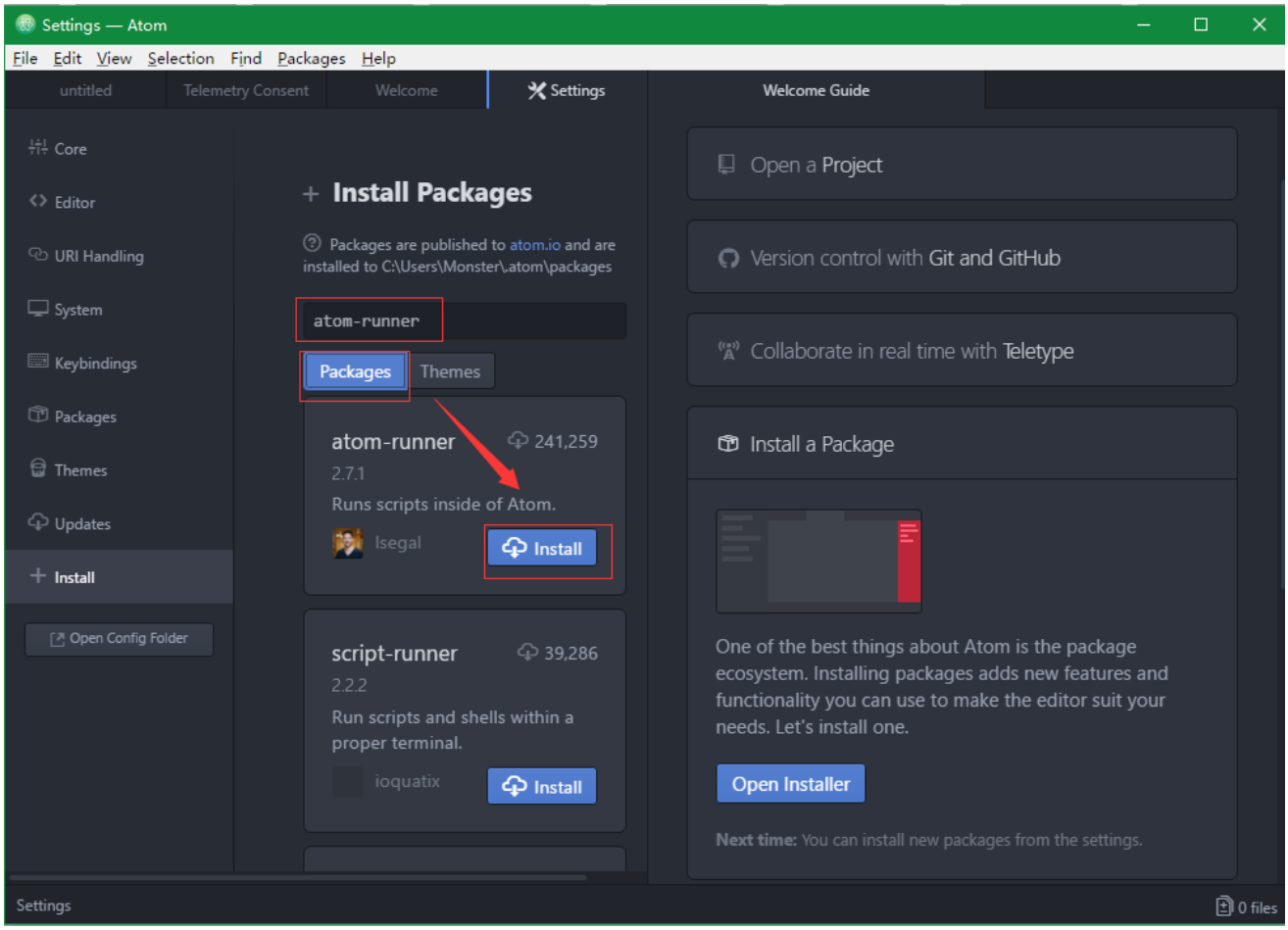
安装完成Atom以后，我们需要集成python开发环境，所以需要配置一下工具，并且安装一些日常使用可以提高工作效率的插件；



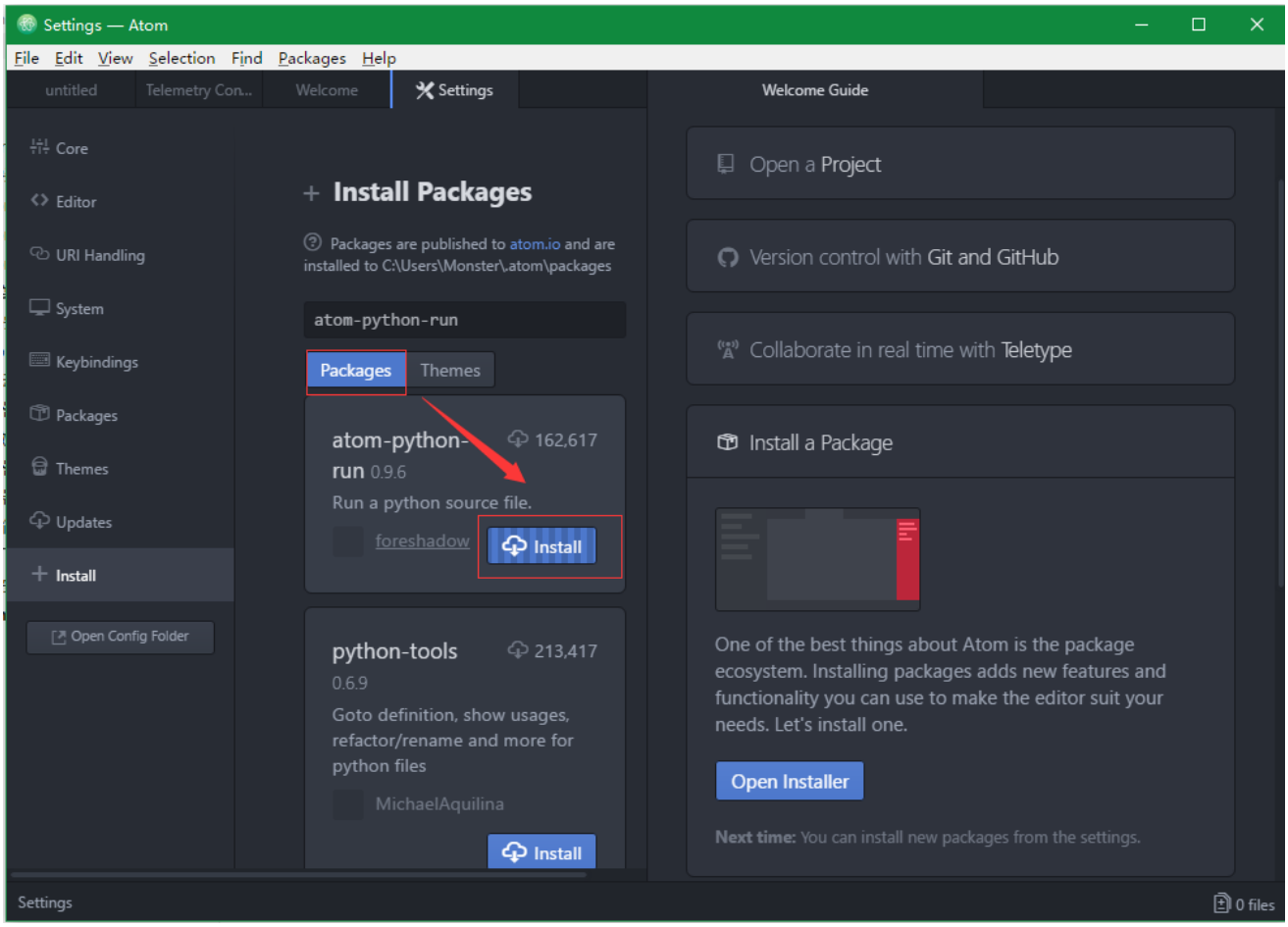
安装【atom-runner】包，点击【Install a Package--Open Install】,进入搜索界面；



输入“atom-runner”搜索，然后点击安装即可。如下图所示：



安装【atom-python-run】（针对windows系统运行python），输入“atom-python-run”搜索，然后点击安装即可。如下图所示：



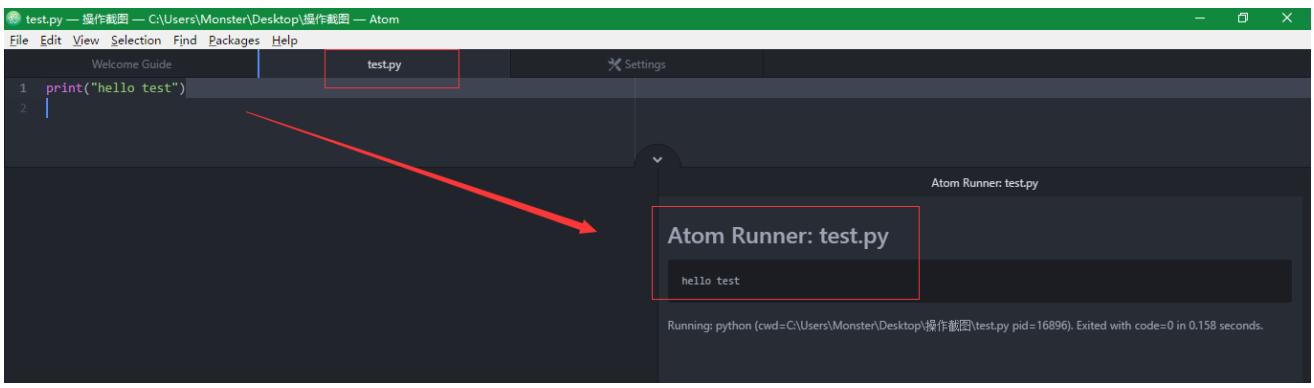
以下为选装插件包，可以根据个人喜好安装：

- 自动补全：autocomplete-python
- 语法检查：linter-flake8
- 定义跳转：python-tools
- 调试工具：python-debugger

4.4Atom开发Python程序

安装好了以上包之后，就可以使用Atom开发程序了，注意，这里Python必须确保已经安装成功。

打开Atom创建一个以.py结尾的文件。输入一些简单的Python代码，使用快捷键“Alt+r”就运行程序，例如：



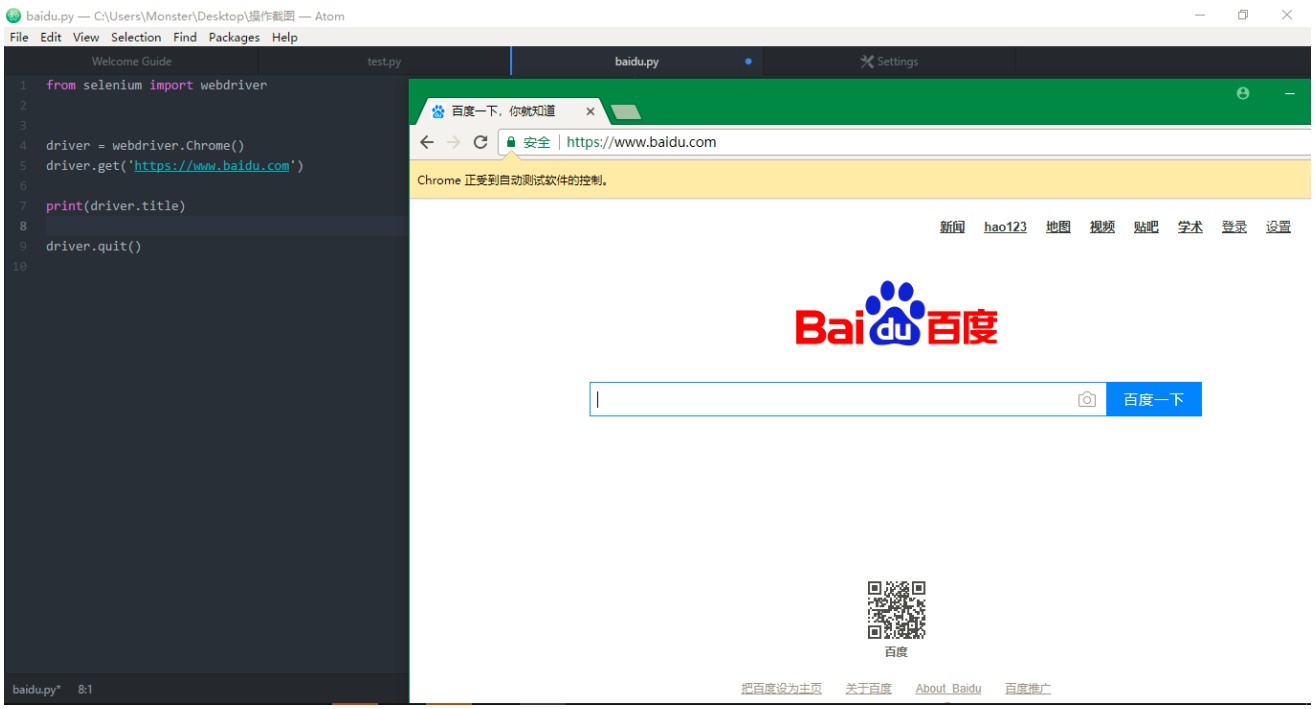
5.开发UI自动化脚本

按照以上步骤执行，就可以配置好Python+Selenium环境，接下来我们来开发一个自动化测试脚本；

打开Atom编辑器，新建baidu.py文件,编辑脚本：

```
from selenium import webdriver

driver = webdriver.Chrome()

driver.get('https://www.baidu.com')

print(driver.title)

driver.quit()
```

使用快捷键“Alt+r”就运行程序，就会自动打开Chrome浏览器，进入百度首页，然后退出浏览器。



在操作浏览器的时候请确保浏览器驱动已经下载并且放置到制定的路径；详情请见章节：**3.1浏览器驱动下载。**

6.开发API自动化脚本

接口测试使用python自带的unittest框架。

unittest是python自带的单元测试框架，尽管其主要是单元测试所用，但我们也可以用它来做接口的自动化测试。

unittest框架为我们编写用例提供了如下的能力

- 定义用例的能力。unittest框架有一套固有套路，可以让我们定义测试用例时更加简单和统一
- 断言的能力。unittest框架提供了一系列的断言

- 各种执行策略。通过test suit或者扩展的方式，我们可以自定义用例执行的策略

打开Atom编辑器，新建API_test.py文件,编辑脚本：

```
import unittest

class StringTestCase(unittest.TestCase):
    def setUp(self):
        # Arrange
        self.test_string = "This is a string"

    def testUpper(self):
        # Act and Assert
        self.assertEqual("THIS IS A STRING", self.test_string.upper())

if __name__ == '__main__':
    unittest.main()
```

6.1代码分析：

```
import unittest
```

导入unittest库，不导入就没办法使用，好比手机如果要使用某个app就必须先安装该app一样，记住就好。

```
class StringTestCase(unittest.TestCase):
```

定义测试类，初学者看到这一行就害怕，其实大可不必。这还是套路，测试类的名字你可以随意取，当然了首字母最好大写，这样更符合规范一些。所有的测试类都必须直接或间接的继承自`unittest.TestCase`类。

```
def setUp(self):
    # Arrange
    self.test_string = "This is a string"
```

`setUp(self)`方法是一个钩子方法，在每个测试用例执行之前都会执行一次，是做数据初始化的好地方。

在上面的例子里，我们为每一个测试方法都定义了被测对象，`self.test_string`

```
def testUpper(self):
    # Act and Assert
    self.assertEqual("THIS IS A STRING", self.test_string.upper())
```

这里定义了一个名为`testUpper`的测试方法，这个方法就是一个测试用例。

注意，只有方法名以test开头的方法才是测试用例

`self.assertEqual`是一个断言方法，作用是如果第一个参数跟第二个参数相等，那么用例通过，否则用例失败，并在测试报告中打印出错误原因。上面的例子里，我们判断`self.test_string.upper()`方法会将`"This is a string"`字符串转换成`"THIS IS A STRING"`

```
if __name__ == '__main__':
    unittest.main()
```

上面的代码表示，如果直接执行该python文件的话，就运行所有的测试类里的测试用例，也就是运行所有的以**test**开头的方法。

6.2总结

使用unittest的话需要记住下面的几点

- 导入unittest
- 定义继承自`unittest.TestCase`的测试类
- 定义以**test**开头的测试方法，这个方法就是测试用例，你可以在一个类里定义n个测试用例
- 断言
- `unittest.main()`是执行测试用例最简单的方式