**Anaconda**

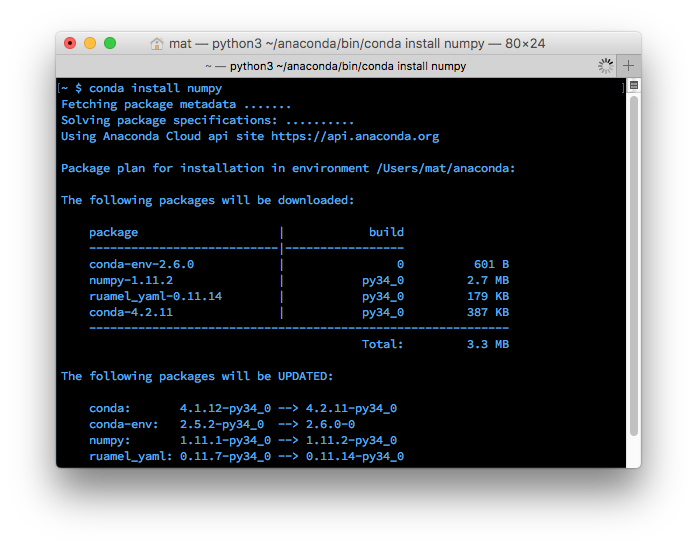
欢迎学习本课程，如何使用 [**Anaconda**](https://www.continuum.io/why-anaconda) 来管理 Python 所用的包和环境。Anaconda 能让你在数据科学的工作中轻松安装经常使用的程序包。你还将使用它创建虚拟环境，以便更轻松地处理多个项目。Anaconda 简化了工作流程，并且解决了多个包和 Python 版本之间遇到的大量问题。

Anaconda 实际上是一个软件发行版，它附带了 conda、Python 和 150 多个科学包及其依赖项。应用程序 conda 是包和环境管理器。Anaconda 的下载文件比较大（约 500 MB），因为它附带了 Python 中最常用的数据科学包。如果只需要某些包，或者需要节省带宽或存储空间，也可以使用 **Miniconda** 这个较小的发行版（仅包含 conda 和 Python）。但你仍可以使用 conda 来安装任何可用的包，它只是自身没有附带这些包而已。

conda 只能通过命令行来使用。因此，如果你觉得它很难用，可以参考**[面向 Windows 的命令提示符教程](https://www.lynda.com/-tutorials/Windows-command-line-basics/497312/513424-4.html" \t "_blank)**，或者学习面向 OSX/Linux 用户的 [**Linux 命令行基础知识**](https://www.udacity.com/course/linux-command-line-basics--ud595)课程。

你可能已经安装了 Python，并且想知道为何还需要 Anaconda。首先， Anaconda 附带了一大批常用数据科学包，因此你可以立即开始处理数据。其次，使用 conda 来管理包和环境能减少将来在处理数据过程中使用到的各种库与版本时遇到的问题。

**管理包**

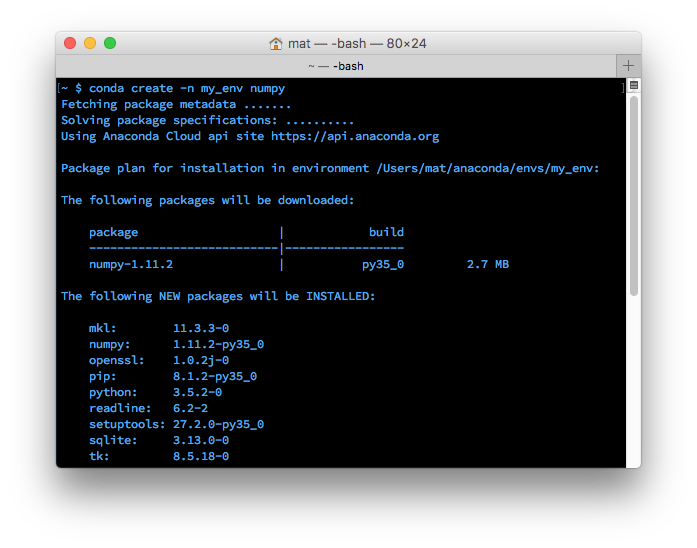


使用 conda 安装 numpy

包管理器用于在计算机上安装库和其他软件。你可能已经熟悉 pip，它是 Python 库的默认包管理器。conda 与 pip 相似，不同之处是可用的包以数据科学包为主，而 pip 适合一般用途。与此同时，conda *并非* 像 pip 那样专门适用于 Python，它也可以安装非 Python 的包。它是支持 *任何* 软件的包管理器。也就是说，虽然并非所有的 Python 库都能通过 Anaconda 发行版和 conda 获得，但同时它也支持非 Python 库的获得。在使用 conda 的同时，你仍可以使用 pip 来安装包。

Conda 安装了预编译的包。例如，Anaconda 发行版附带了使用 [**MKL 库**](https://docs.continuum.io/mkl-optimizations/)编译的 Numpy、Scipy 和 Scikit-learn，从而加快了各种数学运算的速度。这些包由发行版的贡献者维护，这意味着它们通常滞后于最新版本。但是，由于有人需要利用这些包来进行系统构建，因此它们往往更为稳定，而且也更便于你使用。

**环境**



使用 conda 创建环境

除了管理包之外，conda 还是虚拟环境管理器。它类似于另外两个很流行的环境管理器，即 [**virtualenv**](https://virtualenv.pypa.io/en/stable/) 和 [**pyenv**](https://github.com/yyuu/pyenv)。

环境能让你分隔用于不同项目的包。你常常要使用依赖于某个库的不同版本的代码。例如，你的代码可能使用了 Numpy 中的新功能，或者使用了已删除的旧功能。实际上，不可能同时安装两个 Numpy 版本。你要做的应该是，为每个 Numpy 版本创建一个环境，然后项目的对应环境中工作。

在应对 Python 2 和 Python 3 时，此问题也会常常发生。你可能会使用在 Python 3 中不能运行的旧代码，以及在 Python 2 中不能运行的新代码。同时安装两个版本可能会造成许多混乱和错误，而创建独立的环境会好很多。

你也可以将环境中的包列表导出为文件，然后将该文件与代码包括在一起。这能让其他人轻松加载代码的所有依赖项。pip 提供了类似的功能，即 pip freeze > requirements.txt。

**接下来介绍的内容**

接下来，我会详细介绍 Anaconda 的用法。首先，我会介绍它的安装过程，然后介绍如何使用包管理器，最后介绍如何创建和管理环境。

**安装 Anaconda**

Anaconda 可用于 Windows、Mac OS X 和 Linux。可以在 [**https://www.continuum.io/downloads**](https://www.continuum.io/downloads) 上找到安装程序和安装说明。

如果计算机上已经安装了 Python，这不会有任何影响。实际上，脚本和程序使用的默认 Python 是 Anaconda 附带的 Python。

选择 Python 3.6 版本（你可以在以后安装 Python 2 版本。不过，如果你是机器学习、数据分析纳米学位的学员，请选择 Python 2 版本）。此外，如果是 64 位操作系统，则选择 64 位安装程序，否则选择 32 位安装程序。继续并选择合适的版本，然后安装它。之后，继续进行！

完成安装后，会自动进入默认的 conda 环境，而且所有包均已安装完毕，如下面所示。可以在终端或命令提示符中键入 conda list，以查看你安装的内容。

在 Windows 上，会随 Anaconda 一起安装一批应用程序：

* Anaconda Navigator，它是用于管理环境和包的 GUI
* Anaconda Prompt 终端，它可让你使用命令行界面来管理环境和包
* Spyder，它是面向科学开发的 IDE

为了避免报错，我推荐在默认环境下更新所有的包。打开 Anaconda Prompt （或者 Mac 下的终端），键入：

**conda** upgrade --all

并在提示是否更新的时候输入 y（Yes）以便让更新继续。初次安装下的软件包版本一般都比较老旧，因此提前更新可以避免未来不必要的问题。

在本课的余下部分，我会要求你在终端中使用命令。我强烈建议你以这种方式开始使用 Anaconda，之后再根据需要使用 GUI。

**管理包**

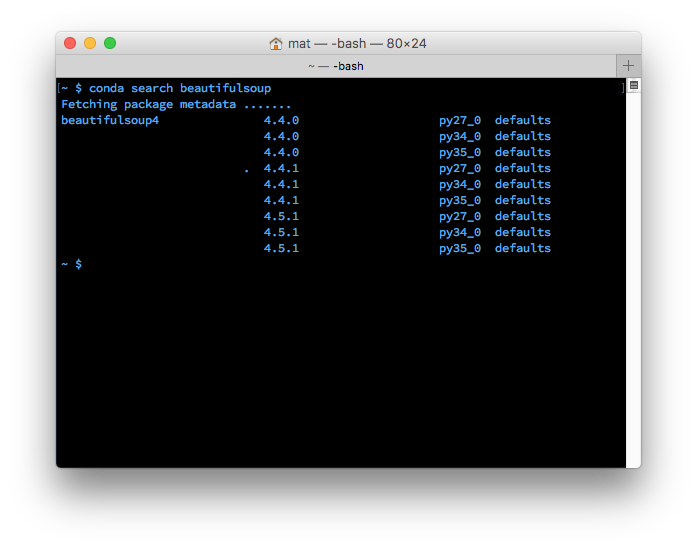
安装了 Anaconda 之后，管理包是相当简单的。要安装包，请在终端中键入 conda install package\_name。例如，要安装 numpy，请键入 conda install numpy。

你还可以同时安装多个包。类似 conda install numpy scipy pandas 的命令会同时安装所有这些包。还可以通过添加版本号（例如 conda install numpy=1.10）来指定所需的包版本。

Conda 还会自动为你安装依赖项。例如，scipy 依赖于 numpy，因为它使用并需要 numpy。如果你只安装 scipy (conda install scipy)，则 conda 还会安装 numpy（如果尚未安装的话）。

大多数命令都是很直观的。要卸载包，请使用 conda remove package\_name。要更新包，请使用 conda update package\_name。如果想更新环境中的所有包（这样做常常很有用），请使用 conda update --all。最后，要列出已安装的包，请使用前面提过的 conda list。

如果不知道要找的包的确切名称，可以尝试使用 conda search search\_term 进行搜索。例如，我知道我想安装 [**Beautiful Soup**](https://www.crummy.com/software/BeautifulSoup/)，但我不清楚确切的包名称。因此，我尝试执行 conda search beautifulsoup。

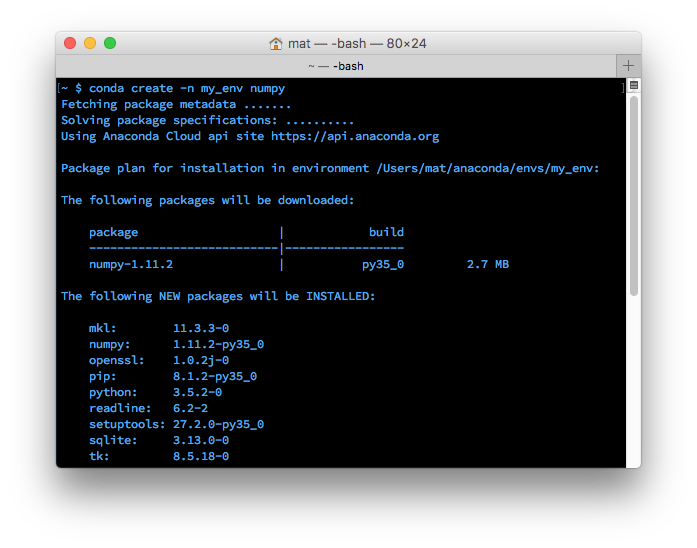


搜索 beautifulsoup

它返回可用的 Beautiful Soup 包的列表，并列出了相应的包名称 beautifulsoup4。

**管理环境**

如前所述，你可以使用 conda 创建环境以隔离项目。要创建环境，请在终端中使用 conda create -n env\_name list of packages。在这里，-n env\_name 设置环境的名称（-n 是指名称），而 list of packages 是要安装在环境中的包的列表。例如，要创建名为 my\_env 的环境并在其中安装 numpy，请键入 conda create -n my\_env numpy。



创建环境时，可以指定要安装在环境中的 Python 版本。这在你同时使用 Python 2.x 和 Python 3.x 中的代码时很有用。要创建具有特定 Python 版本的环境，请键入类似于 conda create -n py3 python=3 或 conda create -n py2 python=2 的命令。实际上，我在我的个人计算机上创建了这两个环境。我将它们用作与任何特定项目均无关的通用环境，以处理普通的工作（可轻松使用每个 Python 版本）。这些命令将分别安装 Python 3 和 Python 2 的最新版本。要安装特定版本（例如 Python 3.3），请使用 conda create -n py python=3.3。

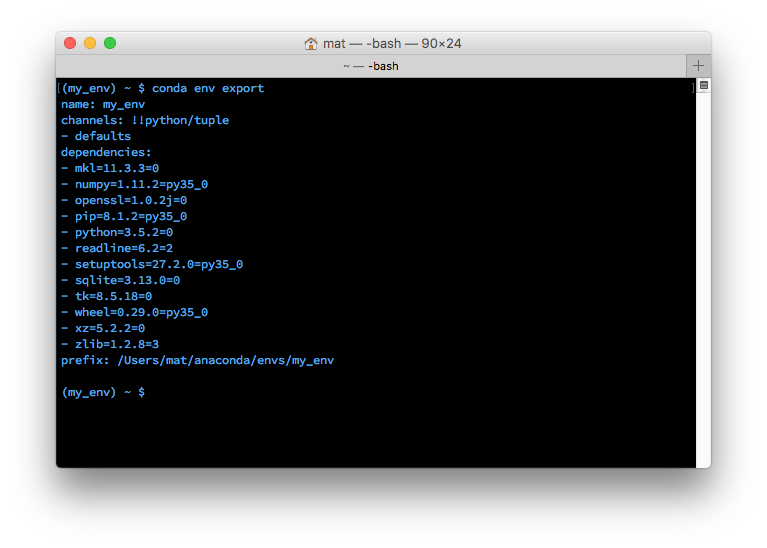
**进入环境**

创建了环境后，在 OSX/Linux 上使用 source activate my\_env 进入环境。在 Windows 上，请使用 activate my\_env。

进入环境后，你会在终端提示符中看到环境名称，它类似于 (my\_env) ~ $。环境中只安装了几个默认的包，以及你在创建它时安装的包。你可以使用 conda list 检查这一点。在环境中安装包的命令与前面一样：conda install package\_name。不过，这次你安装的特定包仅在你进入环境后才可用。要离开环境，请键入 source deactivate（在 OSX/Linux 上）。在 Windows 上，请使用 deactivate。

**保存和加载环境**

共享环境这项功能确实很有用，它能让其他人安装你的代码中使用的所有包，并确保这些包的版本正确。你可以使用 conda env export > environment.yaml 将包保存为 [**YAML**](http://www.yaml.org/)。命令的第一部分 conda env export 用于输出环境中的所有包的名称（包括 Python 版本）。



将导出的环境输出到终端中

上图中，你可以看到环境的名称和所有依赖项及其版本。导出命令的第二部分 > environment.yaml 将导出的文本写入到 YAML 文件 environment.yaml 中。现在可以共享此文件，而且其他人能够用于创建和你项目相同的环境。

要通过环境文件创建环境，请使用 conda env create -f environment.yaml。这会创建一个新环境，而且它具有同样的在 environment.yaml 中列出的库。

**列出环境**

如果忘记了环境的名称（我有时会这样），可以使用 conda env list 列出你创建的所有环境。你会看到环境的列表，而且你当前所在环境的旁边会有一个星号。默认的环境（即当你不在选定环境中时使用的环境）名为 root。

**删除环境**

如果你不再使用某些环境，可以使用 conda env remove -n env\_name 删除指定的环境（在这里名为 env\_name）。

# 最佳做法

## 使用环境

对我帮助很大的一点是，我的 Python 2 和 Python 3 具有独立的环境。我使用了 conda create -n py2 python=2 和 conda create -n py3 python=3 创建两个独立的环境，即 py2 和 py3。现在，我的每个 Python 版本都有一个通用环境。在所有这些环境中，我都安装了大多数标准的数据科学包（numpy、scipy、pandas 等）。

我还发现，为我从事的每个项目创建环境很有用。这对于与数据不相关的项目（例如使用 Flask 开发的 Web 应用）也很有用。例如，我为我的个人博客（使用 [**Pelican**](http://docs.getpelican.com/en/stable/)）创建了一个环境。

## 共享环境

在 GitHub 上共享代码时，最好同样创建环境文件并将其包括在代码库中。这能让其他人更轻松地安装你的代码的所有依赖项。对于不使用 conda 的用户，我通常还会使用 pip freeze（[**在此处了解详情**](https://pip.pypa.io/en/stable/reference/pip_freeze/)）将一个 pip requirements.txt 文件导出并包括在其中。

## 了解更多信息

要详细了解 conda 以及它如何融入到 Python 生态系统中，请查看这篇由 Jake Vanderplas 撰写的文章：[**Conda myths and misconceptions**](https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/)（有关 conda 的迷思和误解）。此外，如果你有空闲精力，也可以参考这篇 [**conda 文档**](http://conda.pydata.org/docs/using/index.html)。