利用 Python 进行生信分析 | 基础 + 实战

——Python for Bioinformatics

赵华男

2022-09-11

自我介绍

赵华男

学习经历

- 2014 ~ 2019　西北农林科技大学 (学士)
- 2016 ~ 2018　中国农业大学 (交换生)
- 2019 ~ 至今　清华大学博士研究生 (PTN 项目)
- 2020 ~ 至今　北京大学 (联合培养)

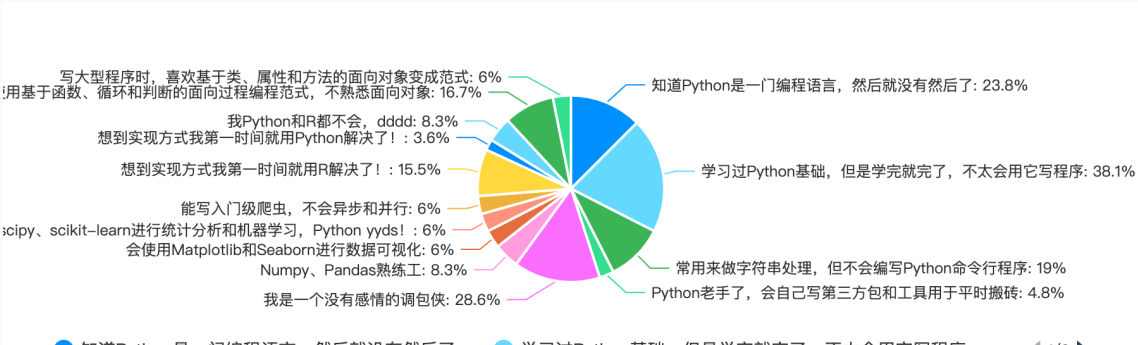
研究方向

生物信息学 | 基因编辑

第一节　课程介绍

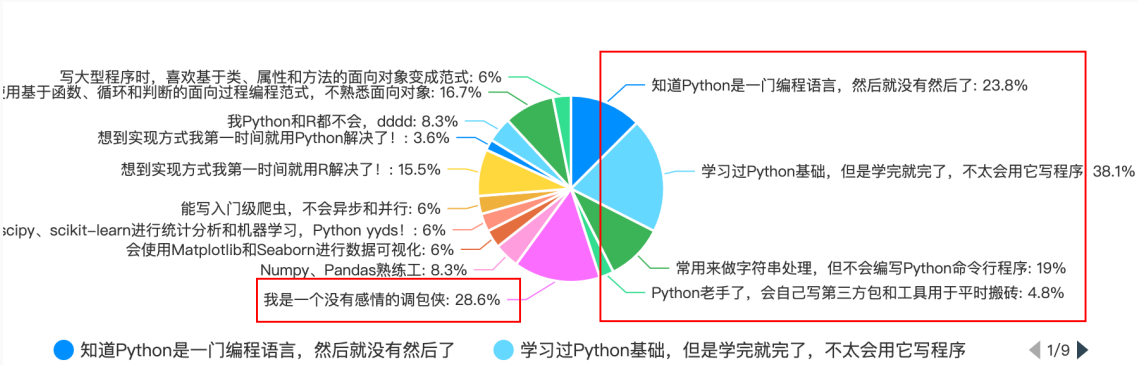
这门课的开设目的

教大家使用 Python 写程序



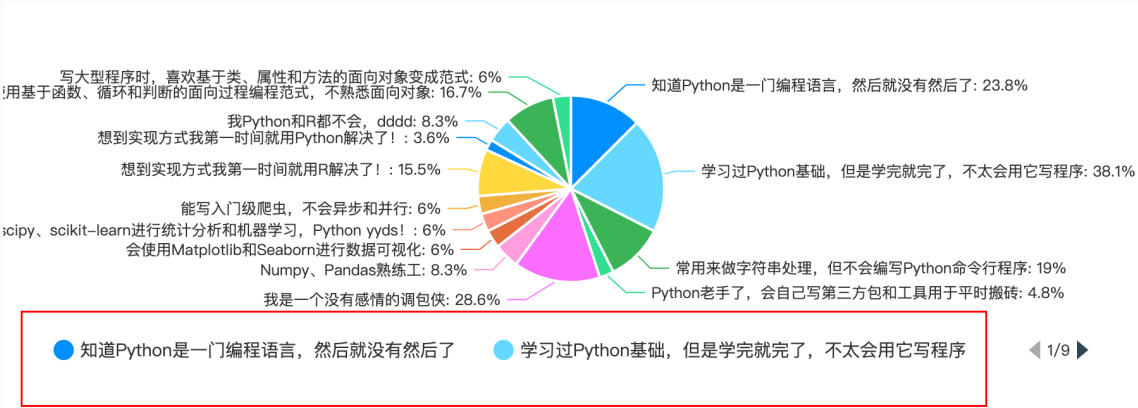
这门课的开设目的

教大家使用 Python 写程序



这门课的开设目的

教大家使用 Python 写程序 → 基础 + 有难度的几个实战项目



课程设计

- 操作系统和编程语言 (Shell, PATH)
- Python 基础知识 + Python 进阶知识
- 编程实战 (核心)
  - 序列文件的处理 (FASTA, FASTQ)
  - BED 文件的操作与处理 (BED)
  - 基因注释文件处理 (GFF, GTF)
  - BAM 文件的解析与操作 (SAM, BAM)
  - 支持多核计算程序的编写
  - 复杂命令行工具的编写与搭建

Thank you!

第二节　操作系统和编程语言

Python 语言的特点

- Guido van Rossum 在 1989 年创造了 Python, 在 1991 年将 Python 首次公开发行
- 跨平台、开源、解释型语言、高级语言
- 源代码可见
- 第三方包种类多, 可用底层语言 (如 C、Rust) 重写关键代码
- 开发效率高，执行效率低

Python 语言的特点

- Python 开发效率高，执行效率低 (科研, 数据分析)
- C 开发效率低, 执行效率高 (高频量化交易)

执行效率对比

- C 10000 行 0.01s
- Java 1000 行 0.05s
- Python 100 行 0.1s

Python 擅长的领域

- 爬虫
- Web 后端开发
- 自动化运维、自动化测试
- 数据科学
- 机器学习

Python 在生物信息学中的应用

- **数据获取 (爬取 UniProt)**
  - 爬虫
- **Web 前后端开发**
  - Django, Flask, 数据库...
- **工作流程**
  - Jupyterlab
  - Snakemake
- **字符串处理**
  - FASTA, FASTQ, BED-like, BAM/SAM
  - Biopython, Pysam
- **数据分析机器学习**
  - **数据清洗, 数据分析, 可视化**
  - 特征工程, 机器学习

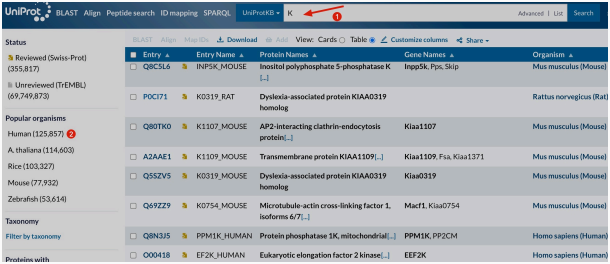


图 1: UniProt 蛋白数据库界面

Python 在生物信息学中的应用

- **数据获取**
  - 爬虫
- **Web 前后端开发**
  - Django, Flask, 数据库...
- **工作流程**
  - Jupyterlab
  - Snakemake
- **字符串处理**
  - FASTA, FASTQ, BED-like, BAM/SAM
  - Biopython, Pysam
- **数据分析机器学习 (AlphaFold2$_{TensorFlow}$)**
  - **数据清洗, 数据分析, 可视化**
  - 特征工程, 机器学习

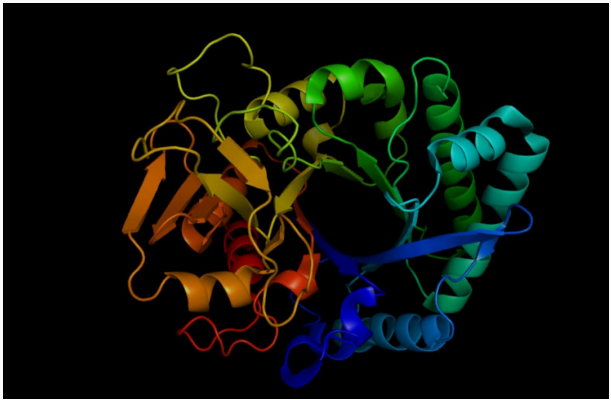


图 2: AlphaFold2 蛋白结构预测

Python 在生物信息学中的应用

- **数据获取**
  - 爬虫
- Web 前后端开发 (AnnoLnc2)
  - Django, Flask, 数据库...
- **工作流程**
  - Jupyterlab
  - Snakemake
- **字符串处理**
  - FASTA, FASTQ, BED-like, BAM/SAM
  - Biopython, Pysam
- **数据分析机器学习**
  - **数据清洗, 数据分析, 可视化**
  - 特征工程, 机器学习

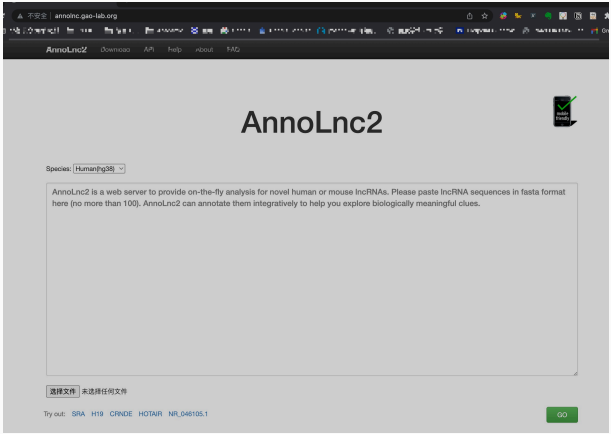


图 3: AnnoLnc2 lncRNA 网页端分析工具

Python 在生物信息学中的应用

- **数据获取**
  - 爬虫
- **Web 前后端开发**
  - Django, Flask, 数据库...
- **工作流程**
  - Jupyterlab
  - Snakemake
- **字符串处理 (强项)**
  - FASTA, FASTQ, BED-like, BAM/SAM
  - Biopython, Pysam
- **数据分析机器学习**
  - 数据清洗, 数据分析, 可视化
  - 特征工程, 机器学习

从 Windows 1.0 到 Windows 11

图 4: Windows 11

图 5: MS-DOS 命令行界面

现在在 Windows 系统中的 PowerShell 已经取代了 CMD 成为新一代壳程序 (Shell), 但是 CMD 仍然集成在操作系统中

https://zh.wikipedia.org/wiki/Microsoft_Windows
https://zh.wikipedia.org/wiki/File:MS-DOS_Deutsch.png
https://zh.wikipedia.org/wiki/Windows_11#/media/File:Windows_11_Desktop.png

Windows, Linux, MacOS

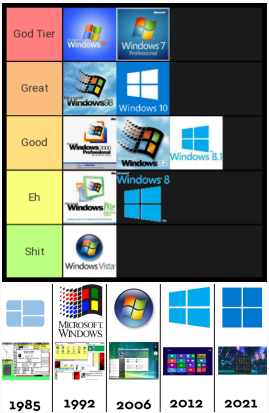


图 6: Windows

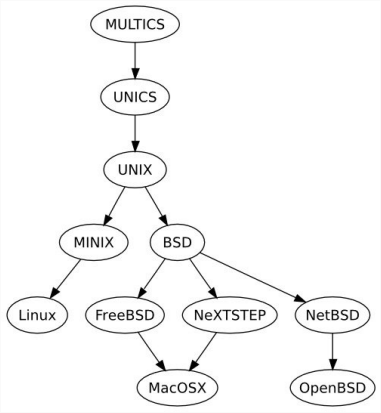


图 7: Linux, MacOS 起源于 UNIX

Windows, Linux, MacOS

Linux (Open source)

- 上百种不同的发行版...
- Ubuntu/Debian
- CentOS
- Android (Google Inc.)
- Shell: (sh, bash, zsh...)

MacOS (Apple Inc.)

- Monterey
- Shell: (sh, bash, zsh...)

图 8: Linux 发行版本

Windows 与 Unix-like 系统之间的区别

特点	Windows	Unix-like
参考版本	Windows10	Ubuntu20.04
是否开源	否	是
路径表示	反斜杠	斜杠
文件结构	分区管理	单一树状
Shell 命令	CMD 风格	Bash 风格

Windows:

C:\User\zhaohuanan\Document\a.txt

D:\somewhere\Document\b.txt

Ubuntu:

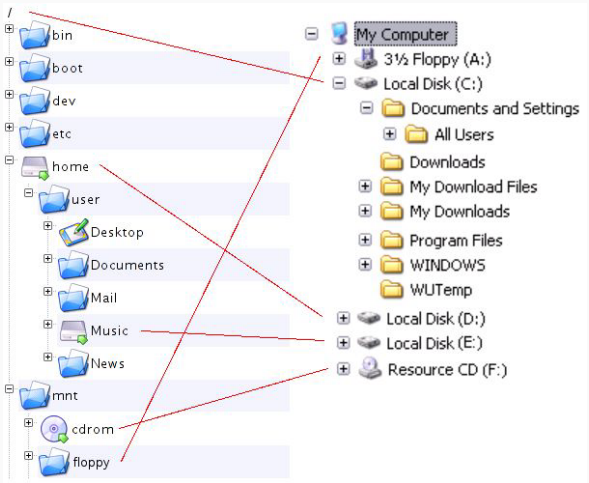
/home/zhaohuanan/Document/a.txt

/home/someone/Document/b.txt

Shell 命令举例

操作	Windows CMD 命令	Ubuntu Bash 命令	相似	不同
切换目录	cd	cd	√	
打印当前目录	cd	pwd		√
清除屏幕	cls	clear		√
列出文件与子目录	dir	ls ls -l		√
列出所有文件与子目录	dir /a	ls -a ls -la		√
创建目录	mkdir md	mkdir mkdir -p	√	
删除目录	rmdir rd	rmdir rm -rf	√	
新建文件	cd .> | notepad	touch		√
删除文件	del	rm		√
拷贝文件	copy	cp		√
在 Shell 中打印文件	type	cat		√
移动文件	move	mv		√
展示目录结构	tree	tree	√	

文件系统结构


/
bin
boot
dev
etc
home
user
Desktop
Documents
Mail
Music
News
mnt
cdrom
floppy
My Computer
3½ Floppy (A:)
Local Disk (C:)
Documents and Settings
All Users
Downloads
My Download Files
My Downloads
Program Files
WINDOWS
WUTemp
Local Disk (D:)
Local Disk (E:)
Resource CD (F:)

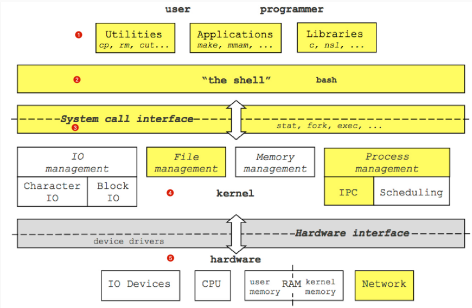
路径演示

- 演示 Windows, MacOS, Linux(以 Ubuntu 为例) 进入文件路径
- 打印当前目录, 切换目录
- 列出文件与子目录, 列出所有文件与子目录, 创建目录, 删除目录
- 新建文件, 拷贝文件, 移动文件, 删除文件
- 在 Shell 中打印文件, 清除屏幕
- 展示目录结构

演示结论

- 类 Unix 系统和 Windows 系统在具体的特点上区别很大
  - 路径分隔符不同
  - 文件系统的结构不同
  - Shell 命令不同
- 但是这些操作系统的抽象层面上的逻辑都是类似的!
  - Shell 命令不同, 但具有相同或相似的功能, 如 type 和 cat
  - 文件系统中的操作具有共性
    - 文件/文件夹的创建/拷贝/移动/删除
    - 目录切换
    - ...

Unix 系统层级



1. 用户使用 Shell 命令，利用 Shell 程序来对操作系统发出指令
2. Shell 程序是用户与操作系统交互的媒介
3. 操作系统负责调度文件，管理内存，管理进程等
4. 内核负责将系统层面的指令传递给硬件，如 CPU，内存条，网卡，硬盘等
5. 硬件，最终完成计算，读取和存储，网络传输等任务的元件

- **用户空间**
  - 定义用户可访问的应用程序、库和标准实用程序
  - Shell 也在 “用户空间” 中运行
- **内核空间**
  - 管理用户操作和硬件之间接口的操作系统的操作
  - 操作系统的核心部分，其主要工作是将用户应用程序与底层硬件配对，并允许多个程序共享单个硬件组件
- **硬件**
  - 计算机的底层物理组件
  - 输入/ 输出设备，如键盘和显示器、进行计算的 CPU、内存组件和网络接口

https://classes.adamaviv.com/ic221/s18/units/01/unit.html

Linux 的应用场景

- **桌面应用**
  - Ubuntu
  - Archlinux
- **嵌入式应用**
  - 电梯、汽车、玩具、灯具、冰箱、空调、电饭煲等，都属于嵌入式概念的应用
- **服务器应用**
  - 开源, 免费, 无需考虑商业软件授权问题
  - 高稳定性, 高可靠性
  - 硬件要求低

“the Shell”

概念: 在计算机科学中, Shell 俗称壳 (用来区别于核), 是指 “为使用者提供操作界面” 的软件。

- **图形界面 Shell**
  - Graphical User Interface Shell, GUI Shell
  - Windows
    - Windows Explorer
  - Linux
    - X-Window
    - GENOME
    - ...
- **命令行 Shell**
  - Command Line Interface Shell, CLI Shell
  - Windows
    - CMD
    - PowerShell
  - Linux
    - sh
    - bash
    - zsh
    - ...

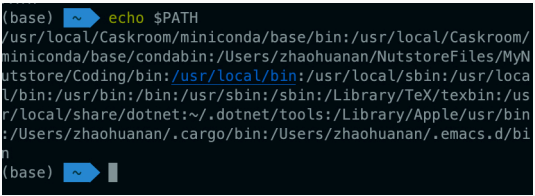
传统意义上的 Shell 指的是命令行式的 Shell

Shell 是一个操作界面, 里面的每一个命令是怎么来的?

命令与 PATH

PATH 是一个环境变量, PATH 中包含了可执行文件 的搜索路径

当要求系统运行一个程序而没有告诉它程序所在的完整路径时, 系统除了在当前目录下面寻找此程序外, 还应到 “PATH” 中指定的路径去找。用户通过设置环境变量, 来更好的运行进程。



PATH 的演示

- MacOS/Linux
  - $PATH
- Windows
  - %PATH%

演示结论

- 我们使用的命令, 其本质是可执行文件
  - 可执行文件具有完整的路径
  - 可以直接使用命令的名称, 是因为其所在文件目录在 PATH 中
- 通过 PATH 中添加路径, 可以省略可执行文件的完整路径
  - 当系统找不到用户输入的可执行文件时, 会在 PATH 中从前到后查找
  - 当在 PATH 中所有的路径下都找不到用户的可执行文件, 会报错没有此可执行文件
- 安装程序时, 可以将可执行文件目录加入 PATH, 以方便使用此程序

编译型语言

编译型语言要求使用编译器一次性将所有源代码编译为一个可执行程序，一次编译可重复执行。

- **编译型语言一般不能跨平台**
  - 编译出来的可执行程序不能跨平台：因为不同操作系统对可执行文件有着不同的要求，彼此之间不能兼容。
  - 源代码不能跨平台: 不同操作系统下的函数、变量、api 等可能会有不同。
- **代表语言：**
  - C、C++
  - Golang
  - Rust
  - Java?

解释型语言

解释型语言是使用解释器一边执行一边转换，用到些源代码就转换哪些，不会生成可执行程序。

- **解释型语言一般可以跨平台**
  - 跨平台是指源代码可以跨平台，解释器是不能跨平台的
  - 源代码在不同操作系统中运行的结果相同
  - 一个语言可以有不同的解释器用不同的语言实现这个解释器
- **代表语言：**
  - C#
  - JavaScript
  - PHP
  - Ruby
  - R
  - Python

编译器与解释器

- **编译器（编译型语言）**
  - 编译器在编译的过程中，读入源程序文件，输出一份等价的二进制可执行文件，就和笔译工作者一样，他们都会输出一份翻译后的文件。
- **解释器（解释型语言）**
  - 解释器在解释的过程中，读入源程序文件，输出的是执行的结果，就和口译工作者一样，他们输出的是已经完成翻译的结果。

输出的不同是这两者最大的区别，一个会输出用于执行的文件，另一个只会输出运行的结果。

https://zhuanlan.zhihu.com/p/389371438

Python 是一种解释型语言 (了解)

超纲内容, 入门同学可以先听不懂!

- 导入大型包的时候第一次导入很慢, 以后会很快的原因?
- Python 并非完全是解释性语言, 它是有编译的:
  - Python 在解释源程序时是分成两个步骤的:
    - 首先处理 py 中的源代码, 编译生成一个二进制 pyc字节码文件 (main + module)
    - 再对字节码进行处理, 才会生成 CPU 能够识别的机器码
    - 有了 module 的字节码文件之后, 下一次运行程序时, 如果在上次保存字节码之后没有修改过源代码, Python 将会加载.pyc 文件并跳过编译 module 的字节码这个步骤
    - 当 Python 重编译时, 它会自动检查 module 源文件和字节码文件的时间戳
    - 如果你又修改了 module 源代码, 下次程序运行时,module 的字节码 pyc 文件将自动重新创建
  - 相对于 py 文件来说, 编译成 pyc 本质上和 py 没有太大区别, 只是对于这个模块的加载速度提高了; 并没有提高代码的执行速度
  - 通常情况下不用主动去编译 pyc 文件, 除非需要隐藏源代码保持私密性
- ".pyc" 文件可以对 module 导入进行加速, Python CLI 程序设计原则:
  - 在 import 别的 py 文件 (module, 模块) 时, 那个 py 文件会被存一份 pyc 加速下次装载
  - 而主文件因为是直接执行而不是 import 导入所以不会保留 pyc
  - 这也是为什么写大程序原则是 CLI 入口代码尽量少的原因:
    - 将主要代码都构建在模块中以加速程序启动
    - 因为 CLI 入口是当前运行的主程序, 并不会生成 pyc!
- 演示

接下来的要求

- 听课要求 (零基础)
  - 手机 No!
  - iPad No!
  - 电脑 Yes!
- 养成记笔记的习惯
  - 使用 Markdown 来记笔记
  - 使用一个开箱即用的笔记工具
- 心态要求
  - 看别人码代码是很枯燥的
  - 动手自己跟着码代码, 获得成就感
  - 程序员/生信工作者 90% 的时间是在解决问题的路上以及学习, 只有 10% 的时间在流畅地写代码, 要从 Debug 的过程中得到进步和获得快乐!
- 不懂的概念和知识点
  - 读报错信息, 动脑思考, 如果还不行 ↓
  - 百度, 如果还不行 ↓
  - Google, 如果还不行 ↓
  - 群里问, 请教师兄师姐

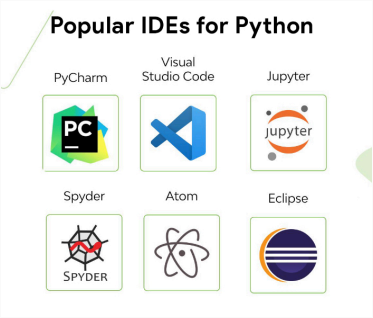
集成开发环境-IDE

集成开发环境 (Integrated Development Environment, IDE):

用于提供程序开发环境的应用程序，一般包括代码编辑器、编译器、调试器和图形用户界面等工具。集成了代码编写功能、分析功能、编译功能、调试功能等一体化的开发软件服务套件。所有具备这一特性的软件都可以叫集成开发环境。

流行的 Python IDE:

- PyCharm (√)
- VS Code
- Jupyter Lab (√)
- Spyder
- IDLE, Sublime, Vim, Emacs...



https://gknxt.com/python/whatisanide.php

Python 的 “解释器”

- Python 解释器 (Interpreter)
  - 解释执行 Python 源代码
  - 官方版本的 Python 解释器是由 C 语言开发的 (CPython)
  - 其他版本的 Python 解释器:
    - IPython
    - PyPy
    - Jython
    - IronPython
- 当我们说, “安装 Python”, 意思是安装 Python 的解释器
  - 默认情况我们认为是安装官方版本的 CPython 解释器
  - 默认我们使用的也是 CPython
  - IPython 以及 Jupyter Notebook/Lab 是基于 IPython 的
- IDE
  - PyCharm、VS Code、Spyder、Jupyter-lab
    - 编写 Python 代码的工具
    - 整合了 Python 解释器
    - 代码提示功能
    - 减轻工作量

安装官方的 Python 解释器

https://www.python.org/

- Windows
- MacOS
- Linux

安装官方的 Python 解释器–Windows

安装官方的 Python 解释器–Linux

安装官方的 Python 解释器–MacOS

Python 的包管理工具

- pip
  - 查询包
  - 安装包
  - 卸载包
  - 一定程度的自动配置环境依赖功能
- venv
  - 创建 Python 的虚拟环境
  - 其余功能类似 pip
- conda
  - 查询、安装、卸载 Python 包
  - 创建、切换、管理 Python 运行环境
  - 命令行工具安装（生信、数据科学必会工具）
  - 强大的自动配置环境依赖功能

Conda 简介

- Conda 是一款环境管理工具
  - 最流行的 Python 环境管理工具之一
  - 开源的**软件包管理系统**和**环境管理系统**,
    - 用于安装多个版本的软件包及其依赖关系, 并在不同环境间切换
    - Conda 是为 Python 程序创建的, 也可以打包和分发其他软件
  - Linux, MacOS 和 Windows 跨平台
- Conda
  - Anaconda
  - Miniconda

安装和配置 conda 环境

下载: tuna → conda → 清华大学 tuna 镜像站 → 搜索 conda → 选择 anaconda → 选择 miniconda → 选择 “latest” 版本!

- Windows
- MacOS
- Linux
- **用法**
  - conda init 初始化 Shell 环境
  - 创建一个指定 Python 版本的 Conda 环境
  - 切换不同 Python 版本的 Conda 环境
  - 对不同 Conda 环境安装不同版本的 Python 包
  - 对一个 Conda 环境安装其他软件

https://conda.io/projects/conda/en/latest/user-guide/install/index.html
https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/
https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/

安装和配置 conda 环境–Windows

安装和配置 conda 环境–MacOS

安装和配置 conda 环境–Linux

Conda 源–以 MacOS 为例

```
# 清除之前残留的conda channels
rm -rf ~/.condarc
# 按顺序依次添加channel, 尽可能使用官方源
# 其他源常常在同步库的时候发生
# md5 值校验错误装不上包的问题!!!!
conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda
conda config --add channels r
# 备用, 清华源
# https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/
# https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/msys2/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/
#设置搜索是显示通道地址
conda config --set show_channel_urls yes
#查看当前conda配置
conda config --show channels
conda config --get channels
```

```
cat ~/.condarc
# return
channels:
    - r
    - bioconda
    - conda-forge
    - microsoft
    - defaults
```

pip 源

```
# PyPI 镜像在每次同步成功后间隔 5 分钟同步一次。
# 临时使用，注意，simple 不能少，是 https 而不是 http
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple some-package

# 设为默认
# 升级 pip 到最新的版本 (>=10.0.0) 后进行配置：
python -m pip install --upgrade pip
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
# 如果您到 pip 默认源的网络连接较差，临时使用本镜像站来升级 pip:
python -m pip install -i https://pypi.tuna.tsinghua.edu.cn/simple --upgrade pip
```

https://mirrors.tuna.tsinghua.edu.cn/anaconda/

Conda, Pip 的扩展用例

```
# install mamba
conda install -c conda-forge mamba
mamba update -n base -c defaults conda
mamba install pandas
# conda 导出环境/导入环境
## 导出当前环境：
conda env export > requirements_conda.yml
## 导入环境：
conda env create -f requirements_conda.yml
## 备注：一些找不到的小组件直接删掉后再安装
# 滚回某个环境
conda list --revisions
conda install --revision=n
# 删除某环境
conda env remove -n learn

# pip 导出环境/导入环境
## 导出当前环境：
pip freeze > requirements.txt
## 导入环境：
pip install -r requirements.txt
```

安装和配置 JupyterLab 环境

- 使用 Conda 安装和配置 JupyterLab
  -
- JupyterLab 基本用法
  - ipynb
  - terminal

install JupyterLab–Windows

install JupyterLab

```
conda install jupyterlab nodejs # nodejs > 12.14
pip install jupyterlab_theme_hale # theme
jupyter-lab --no-browser --port 8888 # run
# 启用extension manager

# func: Settings Editor Form UI
# setting:
{
    "settingEditorType": "json"
}
# func: File Browser
# setting:
{
    "navigateToCurrentDirectory": true,
    "useFuzzyFilter": true,
    "showLastModifiedColumn": true,
    "showHiddenFiles": true,
}
```

```
# func: Text Editor
# setting:
{
    "editorConfig": {
        "autoClosingBrackets": true, // 补全括号
        "codeFolding": true, // 代码折叠
        "cursorBlinkRate": 530,
        "fontFamily": null,
        "fontSize": null,
        "insertSpaces": true,
        "lineHeight": null,
        "lineNumbers": true,
        "lineWrap": "on",
        "matchBrackets": true,
        "readOnly": false,
        "tabSize": 4,
        "rulers": [],
        "showTrailingSpace": false,
        "wordWrapColumn": 80
    }
}
```

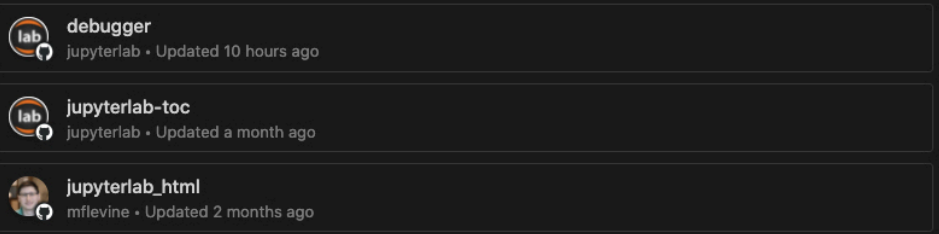
install JupyterLab–MacOS

已经整合的优秀插件

已经吸纳入Jupyterlab本体的优秀插件

（自带，不用安装）

老版本的 jupyter-lab v2.*可能需要安装



自动补全插件

```
pip install jupyterlab-lsp
pip install python-lsp-server pyright
conda install r-languageserver # 如果需要用 R


# func: Code Syntax
# 启用或者关闭 lsp 自动补全
# true 为关闭，false 为开启
# setting:
{
    "disable": false,
}
```

```
# func: Language Server
# setting:
{
    "language_servers": {
    "pyls": {
        "serverSettings": {
        "pyls.plugins.pydocstyle.enabled": true,
        "pyls.plugins.pyflakes.enabled": false,
        "pyls.plugins.flake8.enabled": true
        }
    },
    # 如果需要用 R
    "r-languageserver": {
        "serverSettings": {
        "r.lsp.debug": false,
        "r.lsp.diagnostics": false
        }
    }
    }
}
```

Q & A!

Thank you!

第三节 Python 基础知识

Hello World

- **提示: 输入法使用英文!**
- **两种运行方式**
  - python 的 shell 页面
    - “交互式”, 输入一行, 按回车, 返回一次结果, exit() 退出 (演示)
    - 应用” 在 python 中测试少量代码
  - python 源代码文件 “.py”
    - 编辑好每一行代码, 在命令行全部运行 (演示)
    - 本质: 使用安装在**指定路径下的** python **解释器** 运行指定目录下的 “.py” 源代码文件 (文本文件)(演示)
  - 源代码文件本质是 “文本文件”(可以使用 cat 或者 type 命令打印到命令行)
- **知识点**
  - “123” 字符串
  - 123 整数
  - # 注释
    - 注释一行
    - 行内注释
- **使用不同 IDE 进行演示并说明不同 IDE 运行逻辑**

变量与数据类型

- **变量: 变量是存放数据值的容器**
  - 与其他编程语言不同, Python 没有声明变量的命令
  - 首次为其赋值时，才会创建变量
  - 变量不需要使用任何特定类型声明，甚至可以在设置后更改其类型
  - 字符串变量可以使用单引号, 双引号, 三引号进行声明

Python 变量命名规则

- 只能包含字母数字字符和下划线 (A-z、0-9 和 _)
- 必须以字母或下划线字符开头, 不能以数字开头
- 称区分大小写 (age、Age 和 AGE 是三个不同的变量)
- 如果使用关键字作为变量名?
- Jupyterlab 演示

Python 变量赋值规则

- 常规赋值
- 向多个变量赋值（相同值）
- 向多个变量赋值（不同值），解包（了解）
- 问题：如何将两个变量的值互换?

Python 变量的打印

- 打印一个变量
- 打印多个变量
- 将变量连接到字符串后，进行打印
  - 有关于字符串和变量连接的内容，我们到字符串再讲

Python 内置数据类型 1

- **字符串**: str
  - 常规字符串, raw 字符串, 三引号 (单, 双三引号)
  - 常用方法: find, count, replace, startswith, endswith, upper, lower, split, join, strip
  - 练习: 将 RNA 序列整理为大写, 并替换为 DNA 序列?
  - 切片 (左闭右开): 常规, 步长, 反向
  - 格式化字符串: %, fstring, format 方法
- **二进制**: bytes
- **数值型**:
  - int
  - float
  - complex
- **序列**:
  - list: [] 新建列表,list 函数, 切片, 更改元素, 常用方法 (append, remove, pop)
  - tuple: (a,) 新建元组,tuple 函数, 切片, 元素不可更改
  - range 对象: 功能, 转 list, 转 tuple, 直接遍历, type
  - 字符串

Python 内置数据类型 2

- **集合**: set:
  - {} 新建集合, set 函数, 常用方法 (add, update, remove , discard)
- **字典**: dict:
  - key: value 新建字典, dict 函数, 访问键值对, 更改键值对中的值, 遍历键, 遍历值, 遍历键值对, 添加新的键值对, pop 方法弹出值, popitem 方法弹出键值对,del 关键字删除键值对
- **布尔型**: bool:
  - 定义,bool 函数
  - 大多数值都为 True
    - 如果有某种内容，则几乎所有值都将评估为 True
    - 除空字符串外，任何字符串均为 True
    - 除 0 外，任何数字均为 True
    - 除空列表, 空元组外，任何列表、元组、集合和字典均为 True
    - 对象为 True 或 False 的本质?(__*len*__) 方法返回 0 或 False, 则 bool 函数将其返回为 False

Q & A!

Thank you!